Projekt 2

Michał Piechota

December 2023

1 Wstęp

Celem projektu było rozwiązywanie układu równań liniowych $Ax = b$, gdzie $A \in \mathbb{R}^{n \times n}$ jest macierzą pieciodiagonalną, za pomocą metody eliminacji Gaussa. Dodatkowo obliczymy wyznacznik A i jej odwrotności. W pamięci komputera przechowywane będą jedynie niezerowe przekątne macierzy A, więc algorytm eliminacji Gaussa zostanie odpowiednio zmodyfikowany.

2 Opis matematyczny metody

Aby rozwiąznie układu istniało i było jednoznaczne będziemy zakładać, że $\det(A) \neq 0$. Metoda eliminacji Gaussa opiera się na sprowadzeniu wyjściowego układu równań do układu równoważnego, ale o prostszej strukturze. W podstawowym wariancie metody eliminacji Gaussa, tym którego będziemy używać, wykorzystujemy tylko operacje $r_i = r_i - lr_j$, gdzie $i \neq j$, $l \in \mathbb{R}$. Czyli od i-tego wiersza odejmujemy j-ty przemnożony przez stałą. Eliminacje rozpoczynamy od pierwszej kolumny. Zerujemy wszystkie elementy oprócz pierwszego, potem w drugiej kolumnie wszystkie oprócz drugiego itd. Zerowanie w n-tej kolumnie odbywa się przez odejmowanie od wierszy $(n+1)$-szego, $(n+2)$-giego, ... odpowiednio przemnożonego wiersza n-tego. Mnożnikiem jest oczywiście iloraz $\frac{a_{in}}{a_{nn}}$, gdzie $i = n+1, n+2, \ldots$

3 Implentacja

Nasza macierz jest pięciodiagonalna, czyli jest następującej postaci.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & \ldots \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & \ldots \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & 0 & \ldots \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & \ldots \\ 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{pmatrix}$$

Nasz program będzie przechowywał jedynie niezerowe przekątne powyższej macierzy. Zostaną one wpisane jako wiersze w macierz. Otrzymamy zatem macierz rozmiarów $5 \times n$, gdzie i-tym wierszem będzie i-ta przekątna macierzy A, licząc od góry. Oczywistym jest, że przekątne są różnych rozmiarów, zatem puste miejsca zostaną wypełnione zerami. Otrzymamy zatem macierz B postaci.

$$\begin{pmatrix} 0 & 0 & a_{13} & a_{24} & a_{35} & \dots & a_{n-3n-1} & a_{n-2n} \\ 0 & a_{12} & a_{23} & a_{34} & a_{45} & \dots & a_{n-2n-1} & a_{n-1n} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & \dots & a_{n-1n-1} & a_{nn} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & \dots & a_{nn-1} & 0 \\ a_{31} & a_{42} & a_{53} & a_{64} & a_{75} & \dots & 0 & 0 \end{pmatrix}$$

Wiersze naszej oryginalnej macierzy są teraz ułożone ukośnie od lewej z dołu do prawej w górę. Dla przykładu na niebiesko zaznaczone są wyrazy pierwszego wiersza macierzy A. Indeksując odpowiednimi wektorami macierz B otrzymamy kolejne wiersze macierzy A.

3.1 Indeksowanie wierszy

Zacznijmy od przykładu. Wiersz pierwszy, ten którego wyrazy zaznaczone są na niebiesko, to 3, 7, 11 wyraz macierzy B licząc kolumnowo. Zatem wektorem będzie [3, 7, 11]. Łatwo wyznaczyć wzór na wektor n-tego wiersza:

$$[3 + 5(i-1), 7 + 5(i-1), 11 + 5(i-1)].$$

Przy czym zawsze wystarczy wybrac tylko trzy elementy, bo od elementu czwartego i piątego odejmujemy zera. Wynika to z tego, że A jest pięciodiagonalna.

3.2 Zerowanie kolumn

Zatem aby wyzerować pierwszą kolumnę odejmiemy od czerwonych i zielonych elementów odpowiednio przemnożone elementy niebieskie. Gdzie mnożniki to oczywiście ilorazy pierwszych wyrazów z wierszy.

$$\begin{pmatrix} 0 & 0 & a_{13} & a_{24} & a_{35} & \dots & a_{n-3n-1} & a_{n-2n} \\ 0 & a_{12} & a_{23} & a_{34} & a_{45} & \dots & a_{n-2n-1} & a_{n-1n} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & \dots & a_{n-1n-1} & a_{nn} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & \dots & a_{nn-1} & 0 \\ a_{31} & a_{42} & a_{53} & a_{64} & a_{75} & \dots & 0 & 0 \end{pmatrix}$$

Zerowanie dalszych kolumn przebiega analogicznie. Jedynie ostatnie zerowanie odbywa się delikatnie inaczej, bo używamy już tylko dwóch wyrazów z prostej przyczyny - macierz się kończy. Z tego powodu w funkcji przeprowadzającej algorytm ostatnie zerowanie odbywa się poza pętlą for.

4 Opis funkcji z programu

W tej sekcji podamy krótki opis każdej funkcji z programu.

4.1 Funkcja stworzMacierz

Przyjmuje od użykownika pięć wektorów, które reprezentują diagonale macierzy A. Funkcja tworzy macierz B, której wierszami są podane wektory. Brakujące[1] miejsca uzupełniane są zerami w sposób opisany w sekcji 3.

4.2 Funkcja wypiszMacierz

Motywacją do istnienia tej funkcji jest fakt, że sposób zapisania macierzy gdzie przekątne są przedstawione jako wiersze jest niezbyt czytelny. Funkcja przyjmuje macierz przekątnych i wektor wyrazów wolnych i wypisuje oraz zwraca macierz zapisaną w standardowy sposób.

4.3 Funkcja schodkowanie

Przeprowadza algorytm eliminacji Gaussa na macierzy przekątnych zgodnie ze schematem opisanym w sekcji 3. Wektor b jest przekształcany w oczywisty sposób. Funkcja zwraca rozszerzoną macierz[2] układu po zastosowaniu algorytmu Gaussa.

4.4 Funkcja rozwiazUklad

Funkcja przyjmuje już zeschodkowaną macierz, której wierszami są przekątne macierzy A oraz wektor układu b. Stosuje algorytm z wykładu odpowiednio zmieniając indeksy[3] oraz zwraca wektor rozwiązań.

$$
\begin{aligned}
&x_n := \frac{b_n}{a_{nn}} \\
&\text{for} \quad k = n-1, n-2, \ldots, 1 \\
&\qquad x_k := \frac{b_k - \sum_{j=k+1}^{n} a_{kj} x_j}{a_{kk}} \\
&\text{end}
\end{aligned}
$$

Rysunek 1: Algorytm obliczania rozwiązania

[1] przekątne są różnej długości
[2] wierszami macierzy nadal są przekątne
[3] pracujemy na macierzy, której wierszami są przekątne macierzy A

4.5 Funkcja obliczWyznacznik

Funkcja przyjmuje już zeschodkowaną macierz, której wierszami są przekątne macierzy A, której to wyznacznik mamy policzyć. Główną przekątną jest trzeci wiersz, więc funkcja zwraca iloczyn jego elementów oraz odwrotność tego iloczynu.

4.6 Funkcje pomocnicze w obliczaniu przykładów

W skrypcie *ciekawePrzyklady.m* testowany jest kod dla przykładów, o których więcej w sekcji 5. Wykorzystujemy tam funkcje *obliczPrzyklad*, która na podanej macierzy i wektorze wywołuje poprzednie funkcje. Tym samym rozwiązuje układ, oblicza wyznacznik i porównuje wynik z funkcją wbudowaną Matlaba. Jeśli chcemy wyświetlić błąd, to należy podać 1 jako ostatni argument funkcji. Funkcja *zmierzCzas* wywołuje te same funkcje co *obliczPrzyklad*, ale nie wyświetla wyników dla użytkownika tylko mierzy czas działania programu. Funkcja ta jest wykorzystywana do tworzenia tabeli i wykresu z czasem.

5 "Ciekawe" przykłady

5.1 Macierz o losowych wyrazach

W tym przykładzie macierzą układu będzie macierz o wyrazach losowych, generowanych z rozkładu jednostajnego na $[0, 1]$ o rozmiarze 10000×10000. Wyniki naszego algorytmu zgadzają się z funkcją wbudowaną Matlaba. Błądy są rzędu 10^{-12}, więc możemy je uznać za pomijalne. Pamiętajmy, że odejmowanie bliskich sobie liczb zawsze obarczone jest błędem, więc nawet dla dokładnych wyników błąd będzie niezerowy. W pozostałych przykładach było podobnie, błędy były rzędu w okolicach $10^{-10\pm2}$ więc nie będziemy już się nimi zajmować. Za to skupimy się na czasie potrzebnym do rozwiązania układu i obliczenia wyznacznika.

```
Blad wzgledny:
   1.0e-12 *

  Columns 1 through 9

   -0.0207   -0.0883   -0.0471   -0.0460   -0.0395   -0.0491   -0.0044   -0.4234   -0.0480

  Columns 10 through 18

    0.0127   -0.1034   -0.0457   -0.0170   -0.0499   -0.0780   -0.0516   -0.0478   -0.0440

  Columns 19 through 27

   -0.0710   -0.1305   -0.0525    0.0313    0.0023    0.0087   -0.0675   -0.0410   -0.0017
```

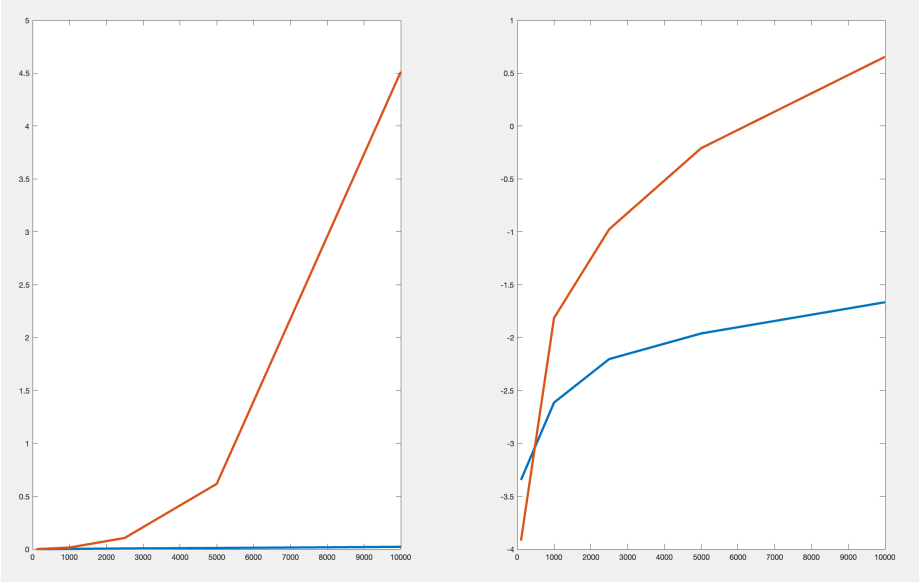
Rysunek 2: Błąd względny dla przykładu 1.

5.2 Macierz o losowych wyrazach - mierzenie czasu

Rozwiążemy układ i obliczymy dla macierzy o wymiarach $n \times n$ gdzie $n \in \{100, 1000, 2500, 5000, 10000\}$ i porównamy czas działania naszego programu i funkcji matlabowych. Poniżej znajduje się tabelka oraz wykresy porównujące czas działania. Przy czym nasz czas obliczania wyznacznika jest tak mały bo wykorzystuje obliczenia wykonane przy rozwiązywaniu układu[4].

	czasObliczaniaWyznacznika	czasObliczaniaWyznacznikaMatlab	czasRozwiazywaniaUkladu	czasRozwiazywaniaUkladuMatlab
n = 100	2.5e-06	0.00014017	0.00025942	0.00012154
n = 1000	5.4167e-06	0.026507	0.002235	0.017316
n = 2500	1.1708e-05	0.16595	0.0065968	0.10424
n = 5000	1.65e-05	1.1775	0.010933	0.62122
n = 10000	3.0583e-05	7.9308	0.021825	4.2385

Rysunek 3: Tabela czasu działa dla przykładu 2.



Rysunek 4: Wykres zależności czasu działania programu od rozmiaru macierzy. Kolorem czerwonym zaznaczony Matlab, niebieskim nasz program. Prawy wykres to lewy po nałożeniu skali logarytmicznej.

Widzimy, że czas obliczeniowy Matlaba bardzo szybko rośnie, a nasz program radzi sobie o wiele lepiej. Z prawego wykresu możemy spodziewać się, że nasz program działa w czasie liniowym. Powodem tego jest dostosowanie sposobu rozwiązywania do faktu, że większość macierzy jest wypełniona zerami.

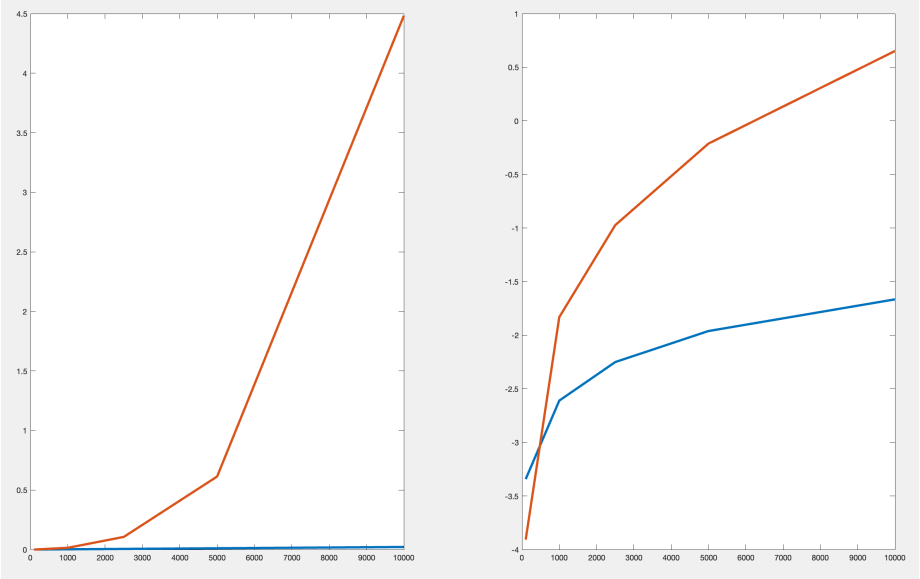
[4]Możemy jednak przyjąc ten czas gdy wiemy, że będziemy wykonywać te dwie operacje, rozwiązywanie układu a później obliczanie wyznacznika. W przypadku Matlaba rozwiązanie układu nie ułatwia obliczenia wyznacznika.

5.3 Macierz Hilberta

Macierz Hilberta to klasyczny przykład bardzo źle uwarunkowanej macierzy. Nasza macierz ma być pięciodiagonalna, więc z wygenerowanej macierzy Hilberta wybierzemy po prostu pięć przekątnych. Tutaj ponownie nasz program zgadza się z wbudowaną funkcją Matlaba, nawet dla bardzo dużych rozmiarów macierzy. Liczyłem, że chociaż czasowo będzie to wyglądało inaczej. Otrzymujemy jednak niemalże identyczne wykresy oraz tabelę. Taka zgodność z losowymi macierzami sugeruje nam, że czas działania będzie zależał jedynie od rozmiaru macierzy, a nie od jej rodzaju[5].

	czasObliczaniaWyznacznika	czasObliczaniaWyznacznikaMatlab	czasRozwiazywaniaUkladu	czasRozwiazywaniaUkladuMatlab
n = 100	4.1667e-05	0.00018204	0.00045433	0.00012375
n = 1000	4.6833e-05	0.027261	0.0024535	0.014746
n = 2500	4.7375e-05	0.16551	0.0056135	0.10634
n = 5000	0.00015483	1.0942	0.010935	0.6134
n = 10000	6.4583e-05	7.949	0.021602	4.4825

Rysunek 5: Tabela czasu działa dla przykładu 3.



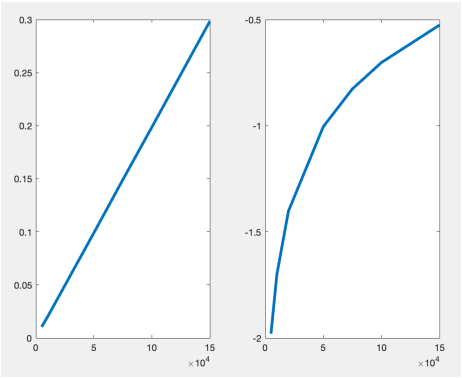
Rysunek 6: Wykres zależności czasu działania programu od rozmiaru macierzy. Kolorem czerwonym zaznaczony Matlab, niebieskim nasz program. Prawy wykres to lewy po nałożeniu skali logarytmicznej.

[5]Przy testach na najróżniejszych typach macierzy dostępnych w matlabie wykresy i tabela były niemalże identyczne

5.4 Testowanie limitów kodu

We wszystkich przypadkach macierzy wykresy z czasem wygladają niemalże identycznie. Zajmiemy się teraz sprawdzeniem jakie są ograniczenia naszego programu. Dla jakich rozmiarów macierzy policzy on wyznacznik i rozwiąże układ w zadawalającym czasie.

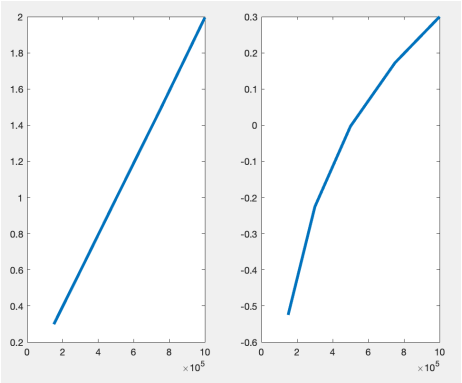
	czasRozwiazywaniaUkladu
n = 5000	0.010283
n = 10000	0.019772
n = 20000	0.039858
n = 50000	0.098764
n = 75000	0.14857
n = 100000	0.19879
n = 150000	0.29807



Rysunek 7: Wykres zależności czasu działania programu od rozmiaru macierzy. Prawy wykres to lewy po nałożeniu skali logarytmicznej.

Nawet dla $n = 150000$ czas obliczania jest bardzo szybki. Dlatego zaczniemy testować jeszcze większe liczby.

	czasRozwiazywaniaUkladu
n = 150000	0.29867
n = 300000	0.59486
n = 500000	0.99321
n = 750000	1.4883
n = 1000000	1.9952



Rysunek 8: Wykres zależności czasu działania programu od rozmiaru macierzy. Prawy wykres to lewy po nałożeniu skali logarytmicznej.

5.5 Układ bez rozwiązań

Sprawdźmy teraz jak nasz program zachowa się w sytuacji, gdy układ nie ma rozwiązań. Rozważmy następujący układ równań:

$$\begin{cases} x_1 + x_2 + x_3 = 1 \\ x_1 + x_2 + x_3 = 2 \\ 2x_1 + 1x_2 + 3x_3 = 7 \end{cases} \tag{1}$$

Układ ten nie ma rozwiązań, a jego macierzą jest oczywiście:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 2 & 1 & 3 \end{pmatrix}$$

Wbudowana funkcja matlaba poradzi sobie z policzeniem wyznacznika oraz ostrzeże nas, że podany układ nie ma rozwiązań. Wyniki naszego programu prezentują się natomiast następująco:

```
Podana macierz:
     1     1     1     1
     1     1     1     2
     2     1     3     7

Macierz po schodkowaniu:
     1     1     1     1
     0     0     0     1
     0   NaN   NaN   Inf

Wyznacznik macierzy:
   NaN

Wyznacznik macierzy odwrotnej:
   NaN

Rozwiazanie ukladu:
   NaN   NaN   NaN
```

Mimo, że wyznacznik macierzy oczywiście istnieje nasz program nie poradził sobie z jego obliczeniem. Jest to spowodowane tym, że funkcja obliczająca wyznacznik korzysta ze zeschodkowanej postaci macierzy A. W przypadku gdy układ nie ma rozwiązań w trakcie schodkowania dochodzi do dzielenia przez zero. Stąd brak wyniku w przypadku działania naszego programu. Obliczanie wyznacznika w przypadku macierzy pięciodiagonalnej, które będzie odporne na ten błąd może wykorzystywać, np. rozwinięcie Laplace'a. Nadal wykorzysta ona fakt, że większość wyrazów w macierzy jest zerowa natomiast zadziała w przypadku gdy układ nie ma rozwiązań.

5.6 Układ z dokładnym rozwiązaniem, o źle uwarunkowanej macierzy

Rozważmy układ równań:

$$\begin{cases} 10x_1 + 7x_2 + x_3 = 18 \\ 0.9999x_1 + 0.7x_2 + 1x_3 = 2.6999 \\ x_1 + 0.9999x_2 + x_3 = 2.9999 \end{cases} \tag{2}$$

Układ ten ma oczywiście rozwiązanie $x_1 = x_2 = x_3 = 1$ Macierzą tego układu jest:

$$A = \begin{pmatrix} 10 & 7 & 1 \\ 0.9999 & 0.7 & 1 \\ 1 & 0.9999 & 1 \end{pmatrix}$$

Wskaźnik uwarunkowania macierz A wynosi $2.3306e + 05$. Zobaczmy jak nasz program poradzi sobie z tym układem. Mimo bardzo złego uwarunkowania macierzy nasz program wyznaczył dokładne rowiązanie.

```
Wyznacznik macierzy:
   6.0001e-04

Wyznacznik macierzy odwrotnej:
   1.6666e+03

Wyznacznik macierzy (matlab):
   6.0001e-04

Wyznacznik macierzy odwrotnej (matlab):
   1.6666e+03

Rozwiazanie ukladu:
    1.0000    1.0000    1.0000

Rozwiazanie matlaba:
    1.0000    1.0000    1.0000

Blad bezwgledny:
   1.0e-15 *

    0.2220   -0.2220    0.2220

Blad wzgledny:
   1.0e-15 *

    0.2220   -0.2220    0.2220
```

6 Analiza wyników

Nasza metoda w każdym przypadku jest o wiele szybsza niż wbudowane funkcje matlabowe. Różnica ta staje się coraz większa im większe macierze rozważamy. W praktycznych zastosowaniach będzie robiło to więc znaczącą różnice w wydajności, ponieważ macierze, na których pracujemy będą znaczących rozmiarów. W dodatku wyniki są zgodne w każdym przypadku więc prawie zawsze lepiej użyc naszego programu. Jedynym wyjątkiem jest układ bez rozwiązań. Wtedy nasz program nie radzi sobie z obliczeniem wyznacznika. Zwraca NaN. Zawsze możemy jednak przyjąć, że jak wyznacznik wyszedł NaN to znaczy, że tak naprawdę był zerem (bo układ nie miał rozwiązań).

7 Zastosowanie metody w praktycznym użyciu

Nasza metoda znacznie szybciej oblicza wyznacznik macierzy oraz rozwiązuje układy równań. W Data Science oraz Machine Learingu często zdarza się, że macierze, na których wykonujemy operacje mają dużo zer. Jeśli uda nam się ustalić, że te niezerowe wyrazy znajdują się jedynie w okolicy przekątnej to możemy stosować naszą metodę. Łatwo ją zmodyfikować na macierze $n-$przekątne. Robiąc krótki research dowiedziałem się, że w uczeniu maszynowym podczas tworzenia modeli liniowych macierze układów często mają niezerowe wyrazy jedynie wokół diagonali. Wtedy nasza metoda może bardzo przyspieszyć obliczenia i znacznie zmniejszyć użycie pamięci.