Sprawozdanie MNUM Projekt 01

Autor: **TOMASZ SACHANOWSKI**

Grupa: **czwartek 8-10**

Nr. Indexu: **276467**

Nr. Zadania**: 1.55**

Spis treści

Treść zadań 2
Zadanie 1 3
Cel: 3
Teoria: 3
Rozwiązanie: 3
Wynik: 3
Podsumowanie: 4
Zadanie 2 4
Cel: 4
Teoria: 4
Rozwiązanie: 7
Wynik: 8
Podsumowanie: 13
Zadanie 3 15
Cel: 15
Teoria: 15
Rozwiązanie: 16
Wynik: 17
Podsumowanie: 22
Kody programów 23
Dodatek do zadania 1 23
Dodatek do zadania 2 23
Dodatek do zadania 3 26

Treść zadań

MNUM–PROJEKT, zadanie 1.55

1. Proszę napisać program wyznaczający dokładność maszynową komputera i wyznaczyć ją na swoim komputerze.

2. Proszę napisać program rozwiązujący układ n równań liniowych $\boldsymbol{Ax=b}$ wykorzystując podaną metodę. Proszę zastosować program do rozwiązania podanych niżej układów równań dla rosnącej liczby równań $n = 10,20,40,80,160,\ldots$. Liczbę tych równań proszę zwiększać aż do momentu, gdy czas potrzebny na rozwiązanie układu staje się zbyt duży (lub metoda zawodzi).

   Metoda: eliminacja Gaussa z częściowym wyborem elementu podstawowego

   Dane: 1) $a_{ij} = \begin{cases} 8 & dla\ i = j \\ 4\ dla\ i = j-1 \text{ lub } i = j+1, & b_i = 4 + 0{,}3\ i; \\ 0 & dla\ pozostałych \end{cases}$

   2) $a_{ij} = 3(i-j) + 2;$ $a_{ii} = 1/3;$ $b_i = 4 + 0{,}4\ i;$

   3) $a_{ij} = 6/[7(i+j+1)]$ $b_i = 1/(3\ i)$, i – parzyste; $b_i = 0$, i – nieparzyste;

   Dla każdego rozwiązania proszę obliczyć błąd rozwiązania (liczony jako norma residuum) i dla każdego układu równań proszę wykonać rysunek zależności tego błędu od liczby równań n.

3. Proszę napisać program rozwiązujący układ n równań liniowych $\boldsymbol{Ax=b}$ wykorzystując metodę Jacobiego i użyć go do rozwiązania poniższego układu równań liniowych:

$$\begin{aligned} 17x_1 + 2x_2 + 11x_3 - 3x_4 &= 12 \\ 4x_1 - 12x_2 + 2x_3 - 3x_4 &= 7 \\ 2x_1 - 2x_2 + 8x_3 - x_4 &= 0 \\ 5x_1 - 2x_2 + x_3 - 9x_4 &= 12 \end{aligned}$$

   Proszę sprawdzić dokładność rozwiązania oraz spróbować zastosować zaprogramowaną metodę do rozwiązania układów równań z zadania 2.

Programy muszą być napisane w Matlabie, ale bez wykorzystania gotowych algorytmów (np. rozwiązywania układów równań).

Sprawozdanie powinno zawierać:
- krótki opis zastosowanych algorytmów,
- wydruki dobrze skomentowanych programów z implementacją użytych algorytmów,
- prezentację otrzymanych wyników,
- komentarz do otrzymanych wyników oraz wnioski z eksperymentów (ocena poprawności wyników, dokładności, efektywności algorytmów itd.).

Sprawozdanie powinno być wysłane na adres prowadzącego:
a.krzemienowski@elka.pw.edu.pl.

Zadanie 1

Cel:

Celem zadania jest wyznaczenie dokładności maszynowej komputera

Teoria:

Korzystając z definicji na dokładność maszynową.

$$eps \overset{\text{df}}{=} \min\{g \in M : fl(1+g) > 1\,,\, g > 0\}. \qquad (1.6)$$

Zgodnie z nią, dokładność maszynowa komputera to najmniejsza dodatnia liczba g taka, że $\boldsymbol{fl(1+g)>1}$, czyli najmniejsza liczba maszynowa (zmiennoprzecinkowa), która dodana do liczby 1 daje w wyniku więcej niż 1.

Rozwiązanie:

Algorytm opiera się na badaniu warunku **1+g > 1**, gdzie g jest zmieniane iteracyjnie. W każdym obiegu pętli wartość g jest dzielona przez 2, aż do momentu niespełnienia warunku. Zwracana jest przedostatnia wartość obiegu, która jest naszą szukaną wartością. Dzielenie przez 2 nie wprowadza dodatkowych błędów z uwagi na sposób komputerowej reprezentacji liczb zmiennoprzecinkowych, gdzie dzielenie przez 2 oznacza tylko zmniejszanie o 1 wykładnika liczby.

Wynik:

Funkcja zwraca wartość 2.2204e-16.

```
>> dokladnoscMaszynowa()

ans =

   2.2204e-16
```

Jest to wartość zgodna z wbudowaną funkcja MATLAB'a **eps**

```
>> eps

ans =

   2.2204e-16
```

Podsumowanie:

Wyznaczona dokładność maszynowa pokrywa się z dokładnością standardu **IEEE 754** dla liczb zmiennoprzecinkowych o podwójnej precyzji. W eksperymencie wyszła ona równa co do wartości stałej **eps** zapisanej w programie MATLAB. Ilość iteracji to 52, czyli tyle ile bitów jest przeznaczonych na mantysę.

Zadanie 2

Cel:

Celem zadania jest rozwiązanie układu **n** równań liniowych w postaci **Ax = b**, przy pomocy metody **ELIMINACJI GAUSSA Z CZĘŚCIOWYM WYBOREM ELEMENTU PODSTAWOWEGO** liczbę równań (**n**) należy zwiększać aż do momentu osiągnięcia zbyt dużego czasu potrzebnego na znalezienie rozwiązania.

Teoria:

Algorytm eliminacji Gaussa dzieli się na dwa etapy:

1. Eliminacja zmiennych – w wyniku przekształceń macierzy A i wektora b otrzymamy równoważny układ równań z macierzą trójkątną górną.
2. Postępowanie odwrotne (ang. ”back-substitution”) – stosujemy algorytm rozwiązania układu z macierzą trójkątną.

Klasyczny algorytmu eliminacji Gaussa nie może znaleźć rozwiązania wszystkich układów posiadających rozwiązanie. Gdyż w trakcie realizacji metody eliminacji Gaussa wymaga się, by wszystkie współczynniki na przekątnej były różne od zera, bo musimy przez nie dzielić kolejne wiersze. Wykonując przedstawiony algorytm eliminacji Gaussa, możemy spotkać blokującą obliczenia sytuację, gdy dla **k-tego** kroku $a_{kk}(k)$=0. Unikamy tego, stosując algorytm eliminacji Gaussa z wyborem elementu głównego. Wybór ten może być częściowy lub pełny.

Metoda z częściowym wyborem elementu głównego:
Na początku k-tego kroku jako element główny wybieramy $|a_{ik}(k)| = \max_j \{ |a_{jk}(k)|, j=k,k+1,\ldots,n\}$, a następnie zamieniamy wiersz i-ty z k-tym, dalej stosujemy algorytm standardowy k-tego kroku. Jest to stosowne w każdym k-tym kroku, nawet jak $a_{kk}(k)$ jest różne od zera.

Etap eliminacji zmiennych:
Wyjściowy układ równań (górny indeks „(k)" – układ równań przed **k-tym** krokiem metody):

$$\begin{array}{ccccccccc} a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)}, \\ a_{21}^{(1)}x_1 & + & a_{22}^{(1)}x_2 & + & \cdots & + & a_{2n}^{(1)}x_n & = & b_2^{(1)}, \\ \cdot & & \cdot & & \cdot & & \cdot & & \cdot \\ a_{n1}^{(1)}x_1 & + & a_{n2}^{(1)}x_2 & + & \cdots & + & a_{nn}^{(1)}x_n & = & b_n^{(1)}. \end{array}$$

Krok 1
Znalezienie elementu głównego w pierwszej kolumnie (a_{i1}). Następnie zamiana wiersza 1 z wierszem i. Potem wyzerowanie elementów kolumny pierwszej, oprócz elementu w wierszu pierwszym, dzięki wyznaczeniu współczynników.

$$l_{i1} \stackrel{\text{df}}{=} \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \qquad i = 2, 3, \ldots, n.$$

Pierwszy wiersz w1 mnożymy przez l_{i1} i odejmujemy od wiersza i-tego w_i, kolejno dla i= 2,3,...,n:

$$\mathbf{w}_i = \mathbf{w}_i - l_{i1}\mathbf{w}_1 \quad \Longleftrightarrow \quad \begin{array}{ll} a_{ij}^{(2)} & = a_{ij}^{(1)} - l_{i1}a_{1j}^{(1)}, \; j = 1, 2, \ldots, n, \\ b_i^{(2)} & = b_i^{(1)} - l_{i1}b_1^{(1)}, \qquad i = 2, 3, \ldots, n. \end{array}$$

Otrzymujemy:

$$\begin{array}{ccccccccc} a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)}, \\ & & a_{22}^{(2)}x_2 & + & \cdots & + & a_{2n}^{(2)}x_n & = & b_2^{(2)}, \\ & & \cdot & & \cdot & & \cdot & & \cdot \\ & & a_{n2}^{(2)}x_2 & + & \cdots & + & a_{nn}^{(2)}x_n & = & b_n^{(2)}. \end{array}$$

Krok k-ty

Znalezienie elementu głównego w kolumnie k-tej (a_{ik}). Następnie zamiana wiersza k-tego z wierszem i. Potem wyzerowanie elementów kolumny k-tej, w wierszach poniżej k-tego, dzięki wyznaczeniu współczynników:

$$l_{ik} \stackrel{\text{df}}{=} \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \qquad i = k+1, k+2, ..., n.$$

Tak więc przekształcenia w k-tym kroku dane są zależnościami:

$$\mathbf{w}_i = \mathbf{w}_i - l_{ik}\mathbf{w}_k \iff \begin{array}{ll} a_{ij}^{(k+1)} & = a_{ij}^{(k)} - l_{ik}a_{kj}^{(k)}, \quad j = k, k+1, ..., n, \\ b_i^{(k+1)} & = b_i^{(k)} - l_{ik}b_k^{(k)}, \qquad i = k+1, k+2, ..., n. \end{array}$$

Ogólnie, po k−1 krokach otrzymujemy układ równań:

$$\begin{array}{ccccccccccccc} a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 + \cdots + & a_{1k}^{(1)}x_k & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)}, \\ & & a_{22}^{(2)}x_2 + \cdots + & a_{2k}^{(2)}x_k & + & \cdots & + & a_{2n}^{(2)}x_n & = & b_2^{(2)}, \\ & & & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & & & a_{kk}^{(k)}x_k & + & \cdots & + & a_{kn}^{(k)}x_n & = & b_k^{(k)}, \\ & & & a_{k+1,k}^{(k)}x_k & + & \cdots & + & a_{k+1,n}^{(k)}x_n & = & b_{k+1}^{(k)}, \\ & & & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & & & a_{nk}^{(k)}x_k & + & \cdots & + & a_{nn}^{(k)}x_n & = & b_n^{(k)}. \end{array}$$

W rezultacie, po n−1 krokach uzyskujemy układ równań:

$$A^{(n)}x = b^n,$$

gdzie A^n to macierz trójkątna górna.

Postępowanie odwrotne (ang. ”back-substitution”):

1. Iteracja po wierszach od ostatniego do pierwszego
2. Dla każdego wiersza korzystając ze wszystkich wierszy poniżej zerujemy wszystkie wartości na prawo od diagonali
3. Dzielimy wartość w wektorze wyników w danym wierszu przez wartość na diagonali.

Układ równań liniowych po podstawieniu otrzymanego algorytmem numerycznym rozwiązania (oznaczmy je przez $x(1)$) na ogół nie jest dokładnie spełniony, tzn:

$$r^{(1)} \stackrel{\text{def}}{=} A|x^{(1)} - b \neq 0,$$

gdzie błąd niespełnienia równań $r(1)$ nazywany jest resztą lub residuum. Błąd rozwiązania liczyłem jako normę residuum (norma euklidesowa):

$$\|r\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$

Rozwiązanie:

Program został podzielony na funkcje. Na początku korzystam z skryptów tworzących odpowiednie macierze A i wektory b. Mamy trzy funkcje które to realizują: ***CreatMatrix_A, CreatMatrix_B, CreatMatrix_C***. Każda działa dla odopowiedniego typu macierzy z zadania. Następnym krokiem jest metoda eliminacji gaussa z częściowym wyborem elementu głównego. Jest to realizowane przez funkcję ***gauss.***

Funkcja ta na samym początku sprawdza, czy wyznacznik macierzy jest różny od zera. Zgodnie z Twierdzeniem Cramera (dla układów, w których $liczba\ równań$=$liczba\ niewiadomych$) jeśli $detA \neq 0$, to układ ma nieskończenie wiele rozwiązań lub jest sprzeczny. Następnie liczony jest wskaźnik uwarunkowania rozwiązania układu równań liniowych. Potem przejście do algorytmu gaussa, w którym to iterujemy po kolumnach i dla każdego kroku szukamy elementu

głównego w danej kolumnie i odpowiednie zamienienie wierszy. Kolejnym etapem jest obliczenie współczynników do zerowania, dzięki którym możliwe będzie wyzerowanie elementów poniżej aktualnego elementu głównego jak przeliczenie pozostałych elementów w wierszu względem wyliczonego współczynnika. W wyniku tych operacji otrzymujemy macierz schodkowa(tójkątna górna). Ostatnim etapem funkcji jest wyliczenie wektora rozwiązań. Jest to realizowane od końca ze względu na macierz trójkątną. W wyniku tej matematycznej operacji przechodzimy do obliczenia normy residuum przy pomocy wzoru i funkcji **norm**:

$$r^{(1)} \stackrel{\text{def}}{=} A|x^{(1)} - b \neq 0,$$

Ostatnim krokiem jest badanie zmiany residuum i czasu dla coraz większej ilości równań. Operacja ta realizowana jest w funkcji ***wykres_residuum_time***.

Wynik:

Okazało się jednak, że w przypadku danych numer 3 wyznacznik macierzy zbiega do zera. Dlatego przy odpowiednich ilościach równań MATLAB wskazuje zero.

```
n = 3, detA = 9.396885e-24
n = 4, detA = 2.293971e-41
n = 5, detA = 7.829967e-64
n = 6, detA = 4.994858e-91
n = 7, detA = -9.797228e-120
n = 8, detA = -1.420007e-149
n = 9, detA = -1.058299e-179
n = 10, detA = 1.526331e-209
n = 11, detA = -1.662967e-241
n = 12, detA = 2.804591e-273
n = 13, detA = -2.680553e-303
n = 14, detA = 0
n = 15, detA = 0
n = 16, detA = 0
```

Dla każde z typów macierzy przy zwiększaniu ilości równań zwiększa się również wskaźnik uwarunkowań. Największe wartości osiąga przy typie C macierzy.

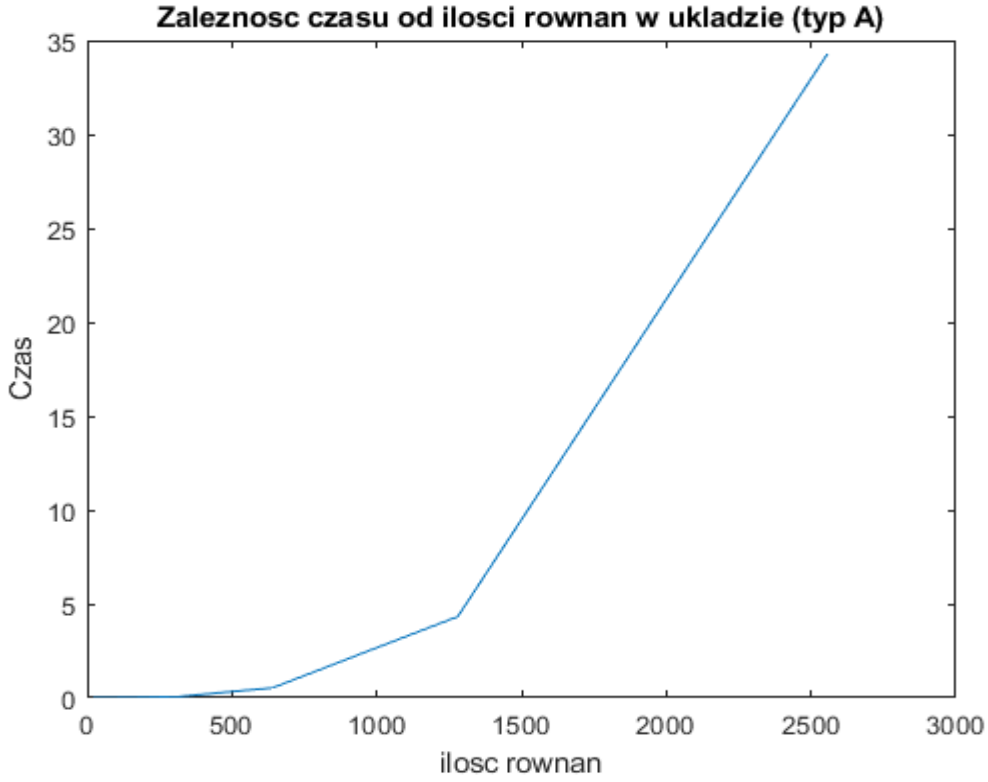
```
Wskanik uwarunkowania zadania 1.879546e+05
Wskanik uwarunkowania zadania 2.106830e+05
Wskanik uwarunkowania zadania 2.347082e+05
Wskanik uwarunkowania zadania 2.600304e+05
Wskanik uwarunkowania zadania 2.866495e+05
Wskanik uwarunkowania zadania 3.145655e+05
Wskanik uwarunkowania zadania 3.437785e+05
Wskanik uwarunkowania zadania 3.742883e+05
Wskanik uwarunkowania zadania 4.060950e+05
Wskanik uwarunkowania zadania 4.391987e+05
Wskanik uwarunkowania zadania 4.735993e+05
Wskanik uwarunkowania zadania 5.092967e+05
Wskanik uwarunkowania zadania 5.462911e+05
```

```
Wskanik uwarunkowania zadania 3.671446e+05
Wskanik uwarunkowania zadania 4.029181e+05
Wskanik uwarunkowania zadania 4.403544e+05
Wskanik uwarunkowania zadania 4.794535e+05
Wskanik uwarunkowania zadania 5.202153e+05
Wskanik uwarunkowania zadania 5.626399e+05
Wskanik uwarunkowania zadania 6.067273e+05
Wskanik uwarunkowania zadania 6.524774e+05
Wskanik uwarunkowania zadania 6.998904e+05
Wskanik uwarunkowania zadania 7.489660e+05
```

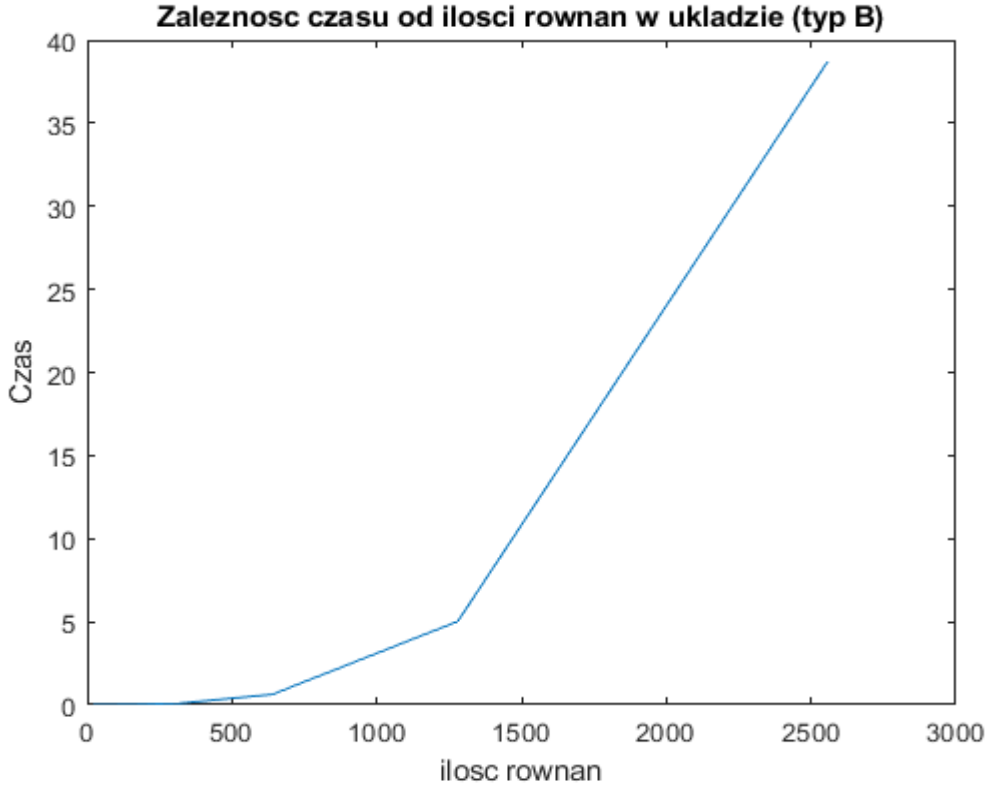
```
Wskanik uwarunkowania zadania 4.092688e+20
Wskanik uwarunkowania zadania 7.552991e+20
Wskanik uwarunkowania zadania 1.082570e+21
Wskanik uwarunkowania zadania 2.781372e+20
Wskanik uwarunkowania zadania 4.622334e+20
Wskanik uwarunkowania zadania 5.723954e+20
Wskanik uwarunkowania zadania 8.774046e+20
Wskanik uwarunkowania zadania 1.618613e+21
Wskanik uwarunkowania zadania 3.650850e+21
Wskanik uwarunkowania zadania 2.380075e+21
```

Wykresy przedstawiające zależność czasu od ilości równań:

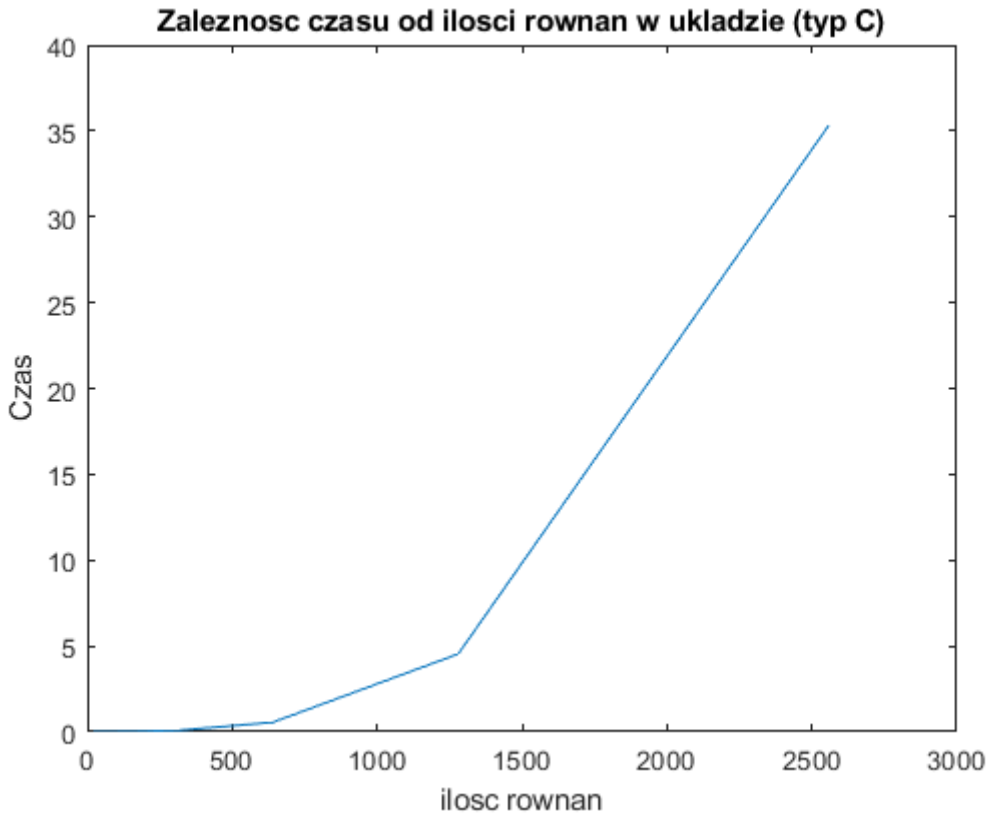
- Typ A



- Typ B

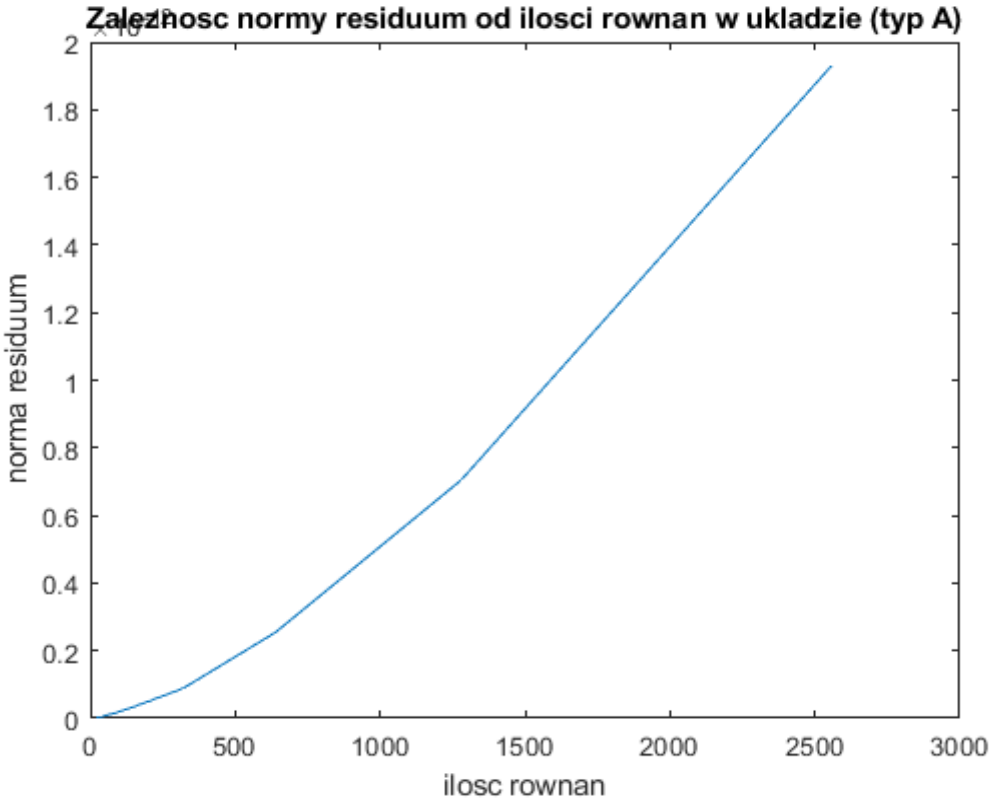


- Typ C

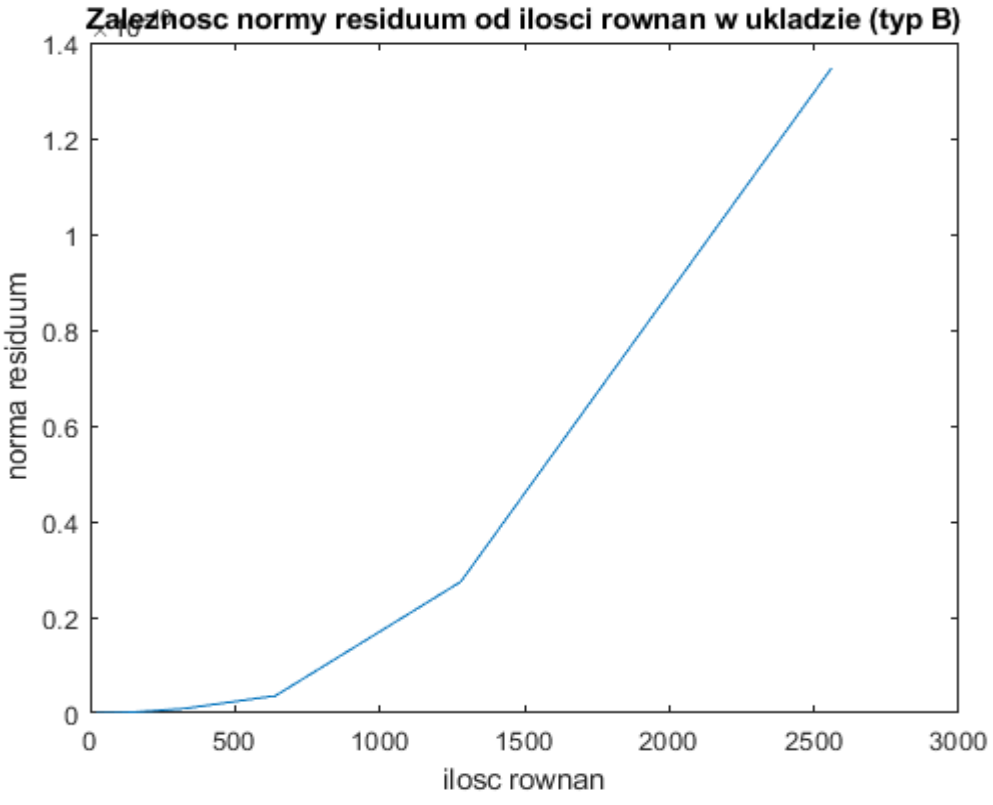

Zaleznosc czasu od ilosci rownan w ukladzie (typ C)
Czas
ilosc rownan
0
5
10
15
20
25
30
35
40
0
500
1000
1500
2000
2500
3000

Wykresy przedstawiające zależność normy residuum od ilości równań:

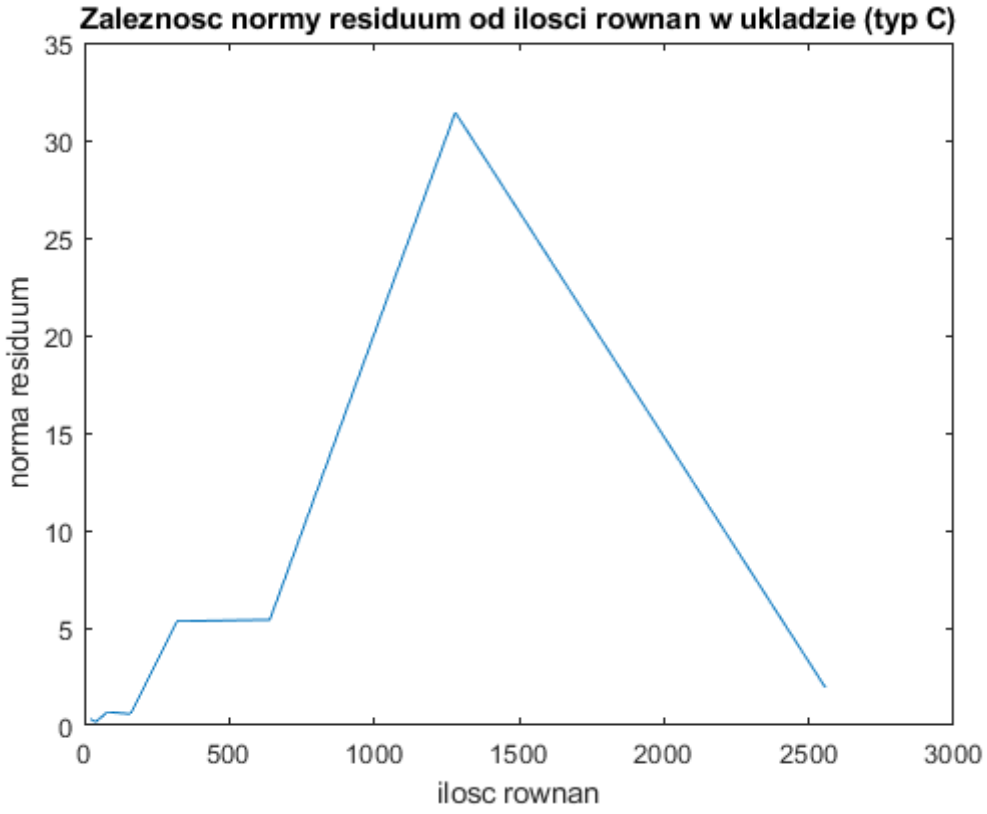
- Typ A



- Typ B



- Typ C



Podsumowanie:

Dla wszystkich rodzajów macierzy czas wzrasta wraz z zwiększaniem się ilości równań co jest spodziewanym efektem. Natomiast macierz typu B obliczała się najdłużej z drugiej strony macierz A najkrócej.

Zarówno dla macierzy typa A jak i dla typu B wartość błędu rozwiązania wzrasta nieliniowo wraz z liczbą równań, ale w obu przypadkach są to względnie nieduże wartości (rząd wielkości 10−13−10−12) , więc możemy uznać otrzymane wyniki za prawidłowe.

Problemy pojawiają się dla macierzy typu C. Tam problem był związany z wyznacznikiem macierzy detA. Przy zwiększającej się ilości równań zbiegał on do zera. Co mogło sugerować o nietypowym zachowaniu. Jako pierwszy rzuca się w oczy nietypowy kształt wykresu, co może oznaczać o kumulacji błędów przy obliczeniach numerycznych. Jednak największym problemem jest rząd wielkości

tych błędów 10^{-5}. Jest to spowodowane złym uwarunkowaniem zadania. Jak już wspominałem wskaźnik uwarunkowań dochodzi do 10^{20} co jest bardzo dużą wartością.

Pojęcie numerycznego uwarunkowania zadania określa wrażliwość wyniku na zaburzenia danych. Niski (względny) wskaźnik uwarunkowania oznacza zadanie dobrze uwarunkowane, wysoki (względny) wskaźnik uwarunkowania – źle uwarunkowane. Zadanie jest źle uwarunkowane, jeśli niewielkie (względne) zaburzenia danych powodują duże (względne) zmiany jego rozwiązania. Uwarunkowanie układu równań liniowych określa, jak zmieni się rozwiązanie układu, jeśli dane, tj. macierz A lub wektor b zaburzymy.

Wskaźniki uwarunkowania dla macierzy typu C mają ogromne wartości, jest to zdecydowanie źle uwarunkowane zadanie. Tak więc powraca wcześniej wspomniany problem $detA$. Jednak jak widzimy na wykresie, czasami obliczenia układają się fortunnie i błędy się znoszą, a innym razem kumulują.

Zadanie 3

Cel:

Celem zadania jest rozwiązanie równania liniowego postaci ***Ax = b*** przy użyciu metody Jacobiego. Zaimplementowaną metodę przetestować na zadanym układzie równań i tym z poprzedniego zadania.

Teoria:

Metody rozwiązywania układów równań liniowych można podzielić na dwie podstawowe grupy:

- Metody skończone – wynik otrzymujemy po skończonej, określonej ilości przekształceń zależnej od rozmiaru zadania.
- Metody iteracyjne – startując z przybliżenia początkowego (znanego, założonego) w kolejnych iteracjach poprawiamy przybliżenie rozwiązania. Nie znana jest ilość iteracji potrzebna do osiągnięcia założonej dokładności.

Metoda Jacobiego jest metodą iteracyjną przeciwnie niż metoda Gaussa użyta poprzednim zadaniu. Oznacza to że aby uzyskać wynik należy wykonywać powtarzające się operacje na macierzy wyjściowej do momentu spełnienia założeń o wyniku działania.

Metoda Jacobiego

Aby rozwiązać układ równań A*x = b należy dokonać dekompozycji macierzy:

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U},$$

Gdzie L jest macierzą poddiagonalną, D diagonalną, a U naddiagonalną. Dekom-pozycja taka jest zawsze możliwa.

Układ równań A*x=b można teraz zapisać w postaci:

$$\mathbf{D}\mathbf{x} = -(\mathbf{L} + \mathbf{U})\mathbf{x} + \mathbf{b}.$$

Zakładając, że wartości na diagonali macierzy A są różne od zera (tzn. macierz D jest nieosobliwa), można (intuicyjnie, arbitralnie) zaproponować metodę iteracyjną:

$$\mathbf{D}\mathbf{x}^{(i+1)} = -(\mathbf{L}+\mathbf{U})\,\mathbf{x}^{(i)} + \mathbf{b}, \qquad i = 0, 1, 2, \ldots, \tag{2.57}$$

podawaną w postaci równoważnej:

$$\mathbf{x}^{(i+1)} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\,\mathbf{x}^{(i)} + \mathbf{D}^{-1}\mathbf{b}, \qquad i = 0, 1, 2, \ldots \tag{2.58}$$

Obliczanie należy wykonywać do momentu uzyskania jak najdokładniejszego wyniku, jeżeli jest to możliwe lub wykonać ustaloną liczbę iteracji. Warunkiem zbieżności metody jest silna dominacja diagonalna macierzy, wierszowa lub kolumnowa:

1. $|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad i = 1, 2, \ldots, n$ – dominacja wierszowa,
2. $|a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}|, \quad j = 1, 2, \ldots, n$ – dominacja kolumnowa.

Rozwiązanie:

Program jest podzielny na dwie funkcje. Pierwszą jest funkcja ***jacobi***. Dzieli ona wyjściową macierz na diagonalna, poddiagonalna+ naddiagonalną. Kolejnym krokiem funkcji jest sprawdzenie dominacji diagonalnej w wierszach i kolumnach. Po tym wszystkim przechodzimy do pętli która aktualizuje wektor x. Pętla kończy działanie gdy osiągniemy zadany błąd lub ilość iteracji. Błąd jest liczony według poniższego wzoru:

$$r = \sum_{i-1}^{n} r_i ,$$

gdzie:

$$r_i = \left| a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n - b_i \right| .$$

Ostatnią funkcją jest ***wykres_jacobi.*** Realizuje ona tworzenie wykresów, które prezentowane są niżej.

Wynik:

Ilość iteracji = 30.

Wyniki rozwiązania równania dla zdania 3:

ERROR = 8.753303e-04

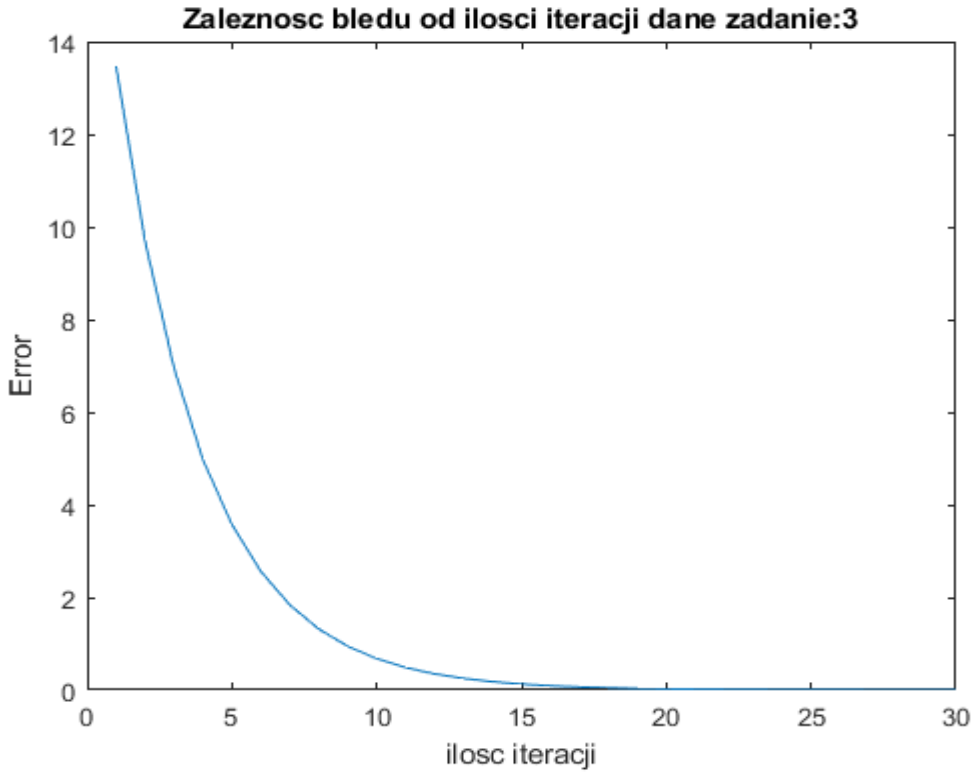
X1 = 0.7907

X2 = -0.1532

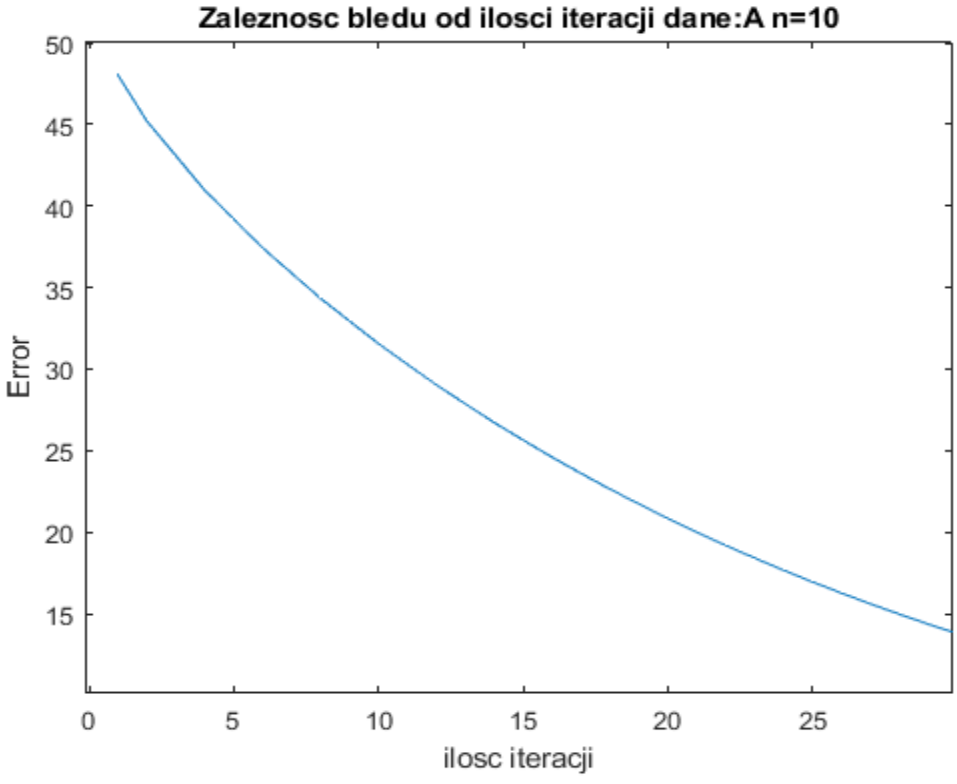
X3 = -0.3483

X4 = -0.8988

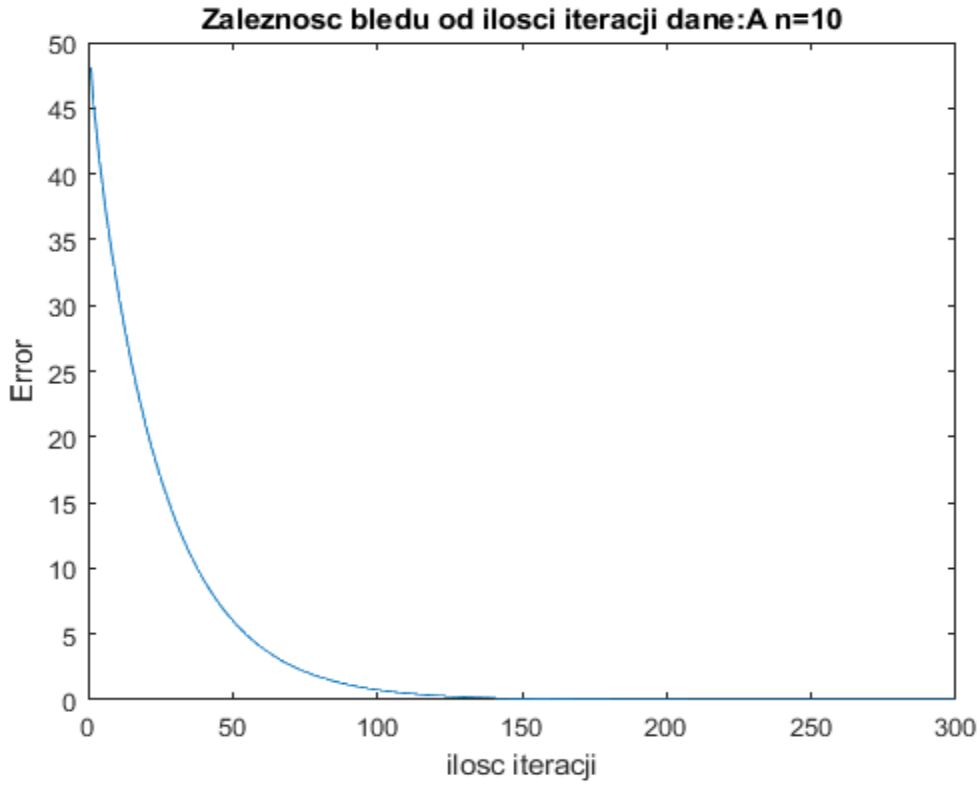
- Zestaw:Zadanie_3 n = 4 iter=30



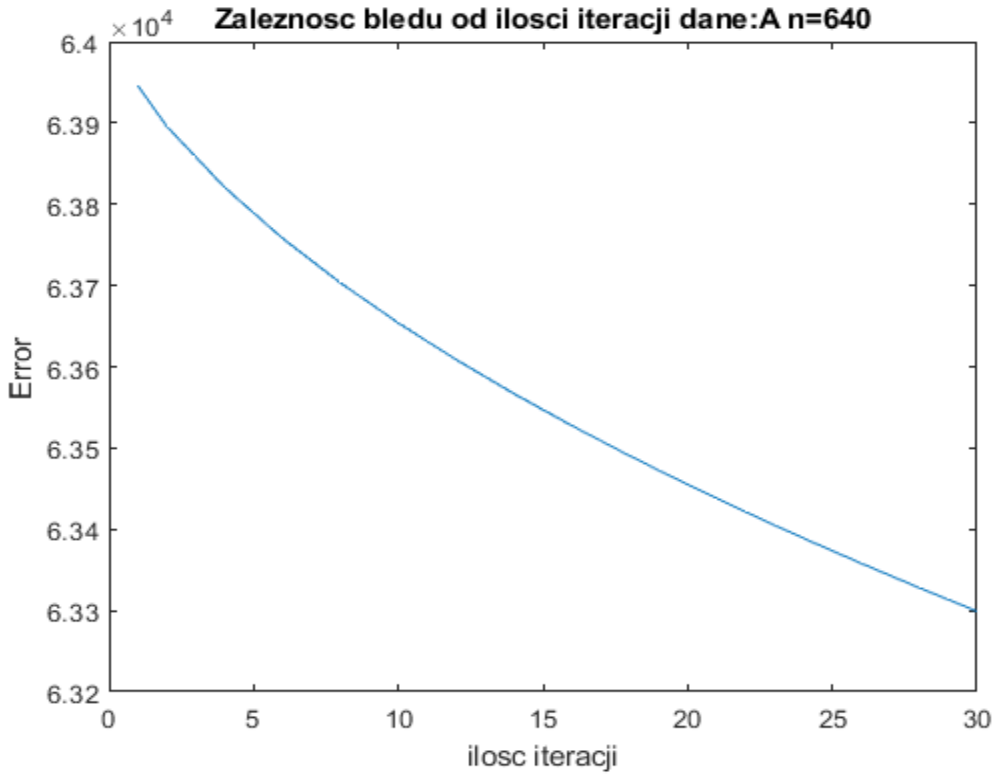
- Zestaw:A n = 10 iter=30



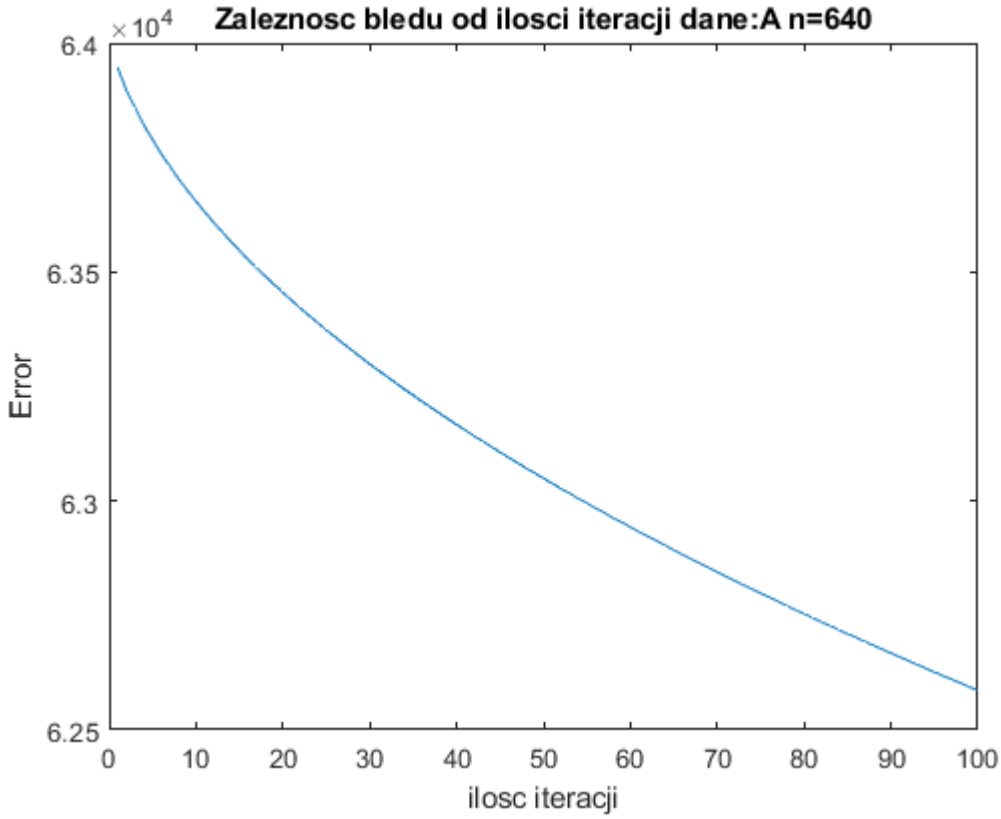
- Zestaw:A n = 10 iter=300



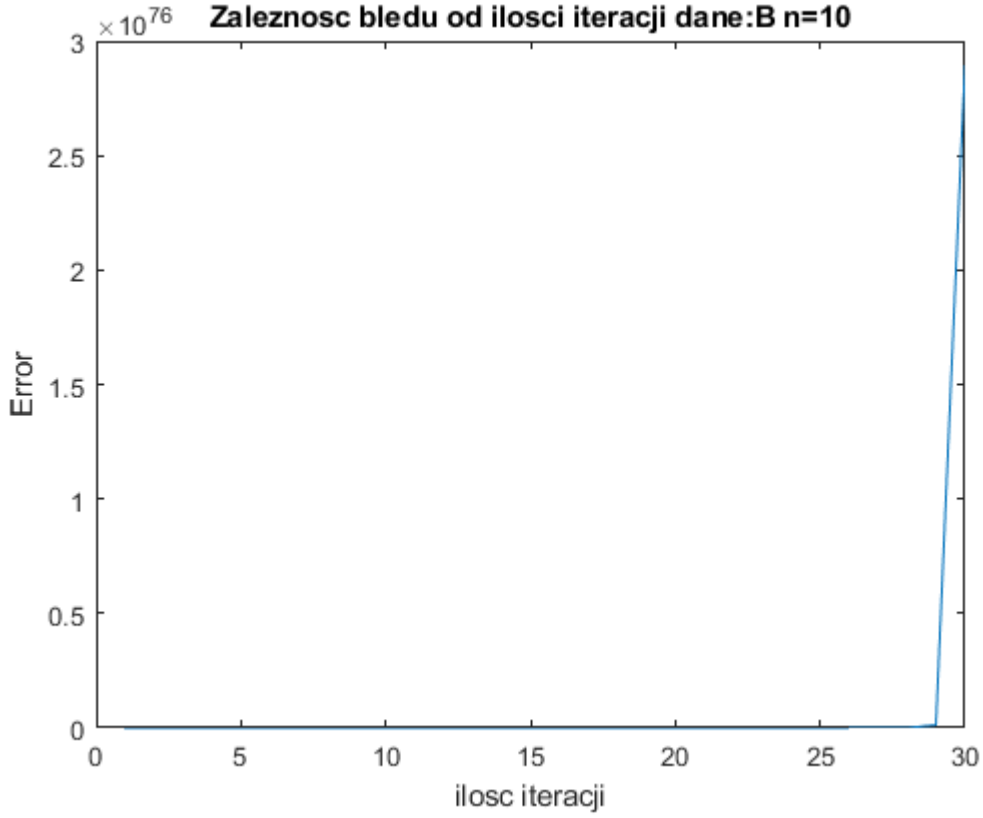
- Zestaw:A n = 640 iter=30



- Zestaw:A n = 640 iter=100

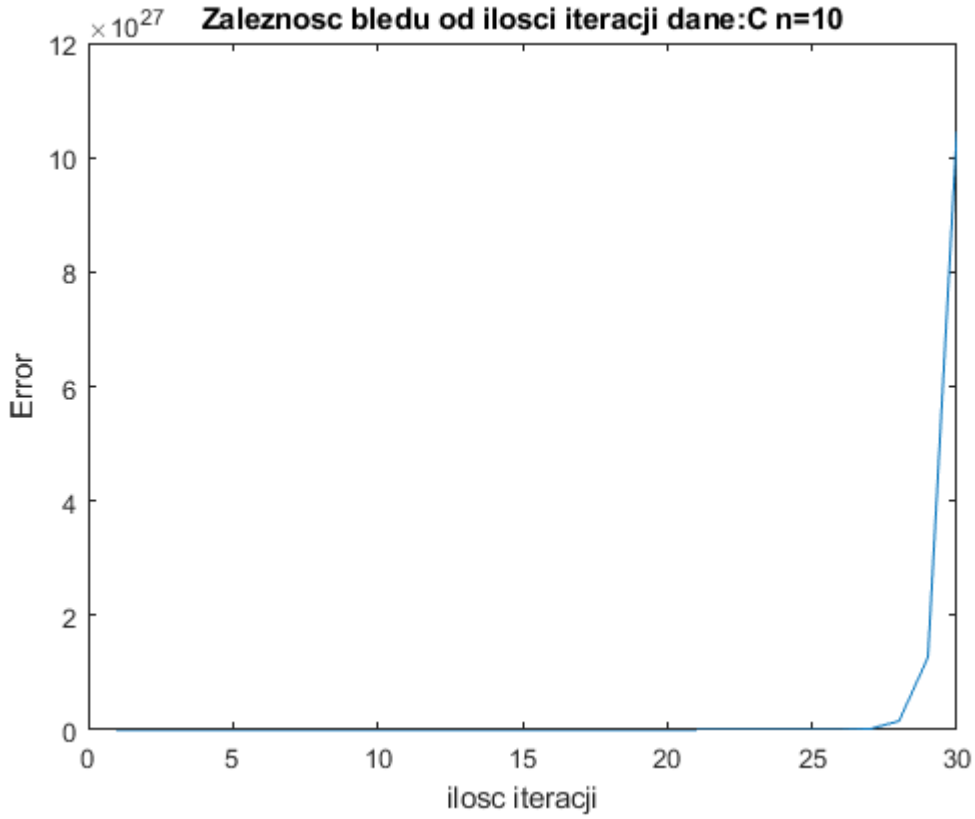


- Zestaw:B n = 10 iter=30



```
Iteracja 4
Macierz nie ma dominacji diagonalnej wierszowej ani kolumnowej
Koniec iteracji, error wynosi: 6.238964e+13
Iteracja 5
Macierz nie ma dominacji diagonalnej wierszowej ani kolumnowej
Koniec iteracji, error wynosi: 16130765032599080
Iteracja 6
Macierz nie ma dominacji diagonalnej wierszowej ani kolumnowej
Koniec iteracji, error wynosi: 4115610996552214016
Iteracja 7
Macierz nie ma dominacji diagonalnej wierszowej ani kolumnowej
Koniec iteracji, error wynosi: 9.607653e+20
Iteracja 8
Macierz nie ma dominacji diagonalnej wierszowej ani kolumnowej
Koniec iteracji, error wynosi: 2.695125e+23
```

- Zestaw:C n = 10 iter=30



```
Iteracja 4
Macierz nie ma dominacji diagonalnej wierszowej ani kolumnowej
Koniec iteracji, error wynosi: 1.440365e+04
Iteracja 5
Macierz nie ma dominacji diagonalnej wierszowej ani kolumnowej
Koniec iteracji, error wynosi: 1.191849e+05
Iteracja 6
Macierz nie ma dominacji diagonalnej wierszowej ani kolumnowej
Koniec iteracji, error wynosi: 9.862114e+05
Iteracja 7
Macierz nie ma dominacji diagonalnej wierszowej ani kolumnowej
Koniec iteracji, error wynosi: 8.160540e+06
Iteracja 8
Macierz nie ma dominacji diagonalnej wierszowej ani kolumnowej
Koniec iteracji, error wynosi: 6.752549e+07
```

Podsumowanie:

Metoda Jacobiego przy niewielkiej ilości iteracji daje duże błędy. Zwiększenie ilości iteracji pozwala dość szybko niwelować błąd. Co jest pokazane na wykresie „Zestaw:Zadanie_3 n = 4 iter=30”.

Pomimo tylko 4 równań błędy są duże, czego przy tak małej liczbie równań rozwiązywanych metoda Gaussa nie zauważaliśmy. Wraz ze wzrostem liczby iteracji błąd ten maleje. Już przy 20 iteracjach wartość jest bliska zeru i dla kolejnych iteracji utrzymuje się. Przy dostatecznej liczbie iteracji metoda jest efektywna, szybka i daje dokładne wyniki.

Metoda Jacobiego przy zestawie danych A przy n=10 równań i 30 iteracjach daje spory błąd, który osiąga wartość w granicy 15 (wykres „Zestaw:A n = 10 iter=30”). Natomiast zwiększenie iteracji do 300 przyczynia się do osiągnięcia znacznie mniejszego błędu. I przy tej kombinacji metoda daje pozytywny rezultat (wykres „Zestaw:A n = 10 iter=300”).

Problem pojawia się jak zwiększymy ilość równań do n =640. Tu metoda zawodzi gdyż nawet zwiększeni iteracji powoduje utrzymywanie się dużego błędu powyżej 6.25e4. Metoda zawodzi dla danych z zestawu „A” gdy mamy spora ilość równań (wykresy „Zestaw:A n = 640 iter=30”, „Zestaw:A n = 640 iter=100”).

Metoda nie spisuje się całkowicie dla danych B i C, W obu macierzach nie było spełnionego warunku dominacji diagonalnej. Zwiększanie ilości iteracji powodowało, że błąd rósł do bardzo dużych wartości (wykresy „Zestaw:B n = 10 iter=30”, „Zestaw:C n = 10 iter=30”).

Metoda Jacobiego ma największe powodzenie dla danych w których występuje silna dominacja diagonalna macierzy. Błąd rozwiązania przy odpowiednio dużej liczbie iteracji jest bliski zeru. Jednak metoda Gaussa jest bardziej uniwersalna i sprawdzi się ona w większości równań, mimo że w tej metodzie błędu również mogą być zauważalne.

Kody programów

Dodatek do zadania 1

```
% g - najmniejsza dodatnia liczba maszynowa, ktĂłra speĹ‚nia warunkek
fl(1+g)>1
function [g] = dokladnoscMaszynowa()
% tmp - zmienna pomocnicza
tmp = 1;
while (tmp+1>1)
g = tmp;
tmp = tmp/2;
end
end
```

Dodatek do zadania 2

```
function [A,b] = CreatMatrix_A(size)
A = zeros(size);
b = zeros(size, 1);

for i = 1: size
    b(i,1) = 4 + 0.3*i;
    for j = 1: size
        if(i==j)
            A(i,j) = 8;
        elseif(i == j - 1 || i == j + 1)
            A(i,j) = 4;
        end
    end
end
end
```

```
function [A,b] = CreatMatrix_B(size)
A = zeros(size);
b = zeros(size, 1);

for i = 1: size
    b(i,1) = 4 + 0.4*i;
    for j = 1: size
        if(i==j)
            A(i,j) = 1/3;
        else
            A(i,j) = 3*(i-j) + 2;
        end
    end
end
end
```

```
function [A,b] = CreatMatrix_C(size)
A = zeros(size);
b = zeros(size, 1);

for i = 1: size
    b(i, 1) = 1/(3*i);
    if mod(i, 2)== 1
        b(i, 1) = 0;
    end
    for j = 1: size
        A(i,j) = 6/(7*(i+j+1));
    end
end
end
```

```
function [r, time] = gauss(M, b)
%b- wektor wynikow
%M- macierz wspolczynikow
%r- norma residuum
    tic;
%Sprawdzenie czy uklad rownan ma rozw. (Tw. CRAMERA)
    if (det(M) == 0)
        text = "Wyznacznik macierzy jest rowny zero(bliski) ";
        text = text + "Nie ma pojedynczego rozw";
        fprintf(text);
        return;
    end
    fprintf('Wskanik uwarunkowania zadania %d\n',cond(M));
    %laczenie M i b
    A = [M,b];
    % ilosc rownan
    size_n = max(size(M));
    % iteracja po kolumnach, -1 bo jest dolaczony wektor odpowiedzi b.
    % Trzeba skonczyc operacje na przedostatniej kolumnie macierzy M a
    % po dolaczeniu A jest dluzszy o jedna kolumne
    for c=1:size_n - 1
        % ustawienie pomocniczej wartosci dla elementu
        % glownego w danym wierszu.
        tmp_max_elemnet = A(c, c);
        index_max = c;
        %szukanie elementu glownego w danym wierszu
        for r=c:size_n
            if (tmp_max_elemnet < A(r,c))
                tmp_max_elemnet = A(r,c);
                index_max = r;
```

```
            end
        end
        % zamiana wierszy aby ten z elementem glownym byl
        % uzyty do wyliczenia wspolczynikow
        if (index_max ~= c)
            tmp_row = A(index_max, 1:(size_n+1));
            A(index_max,1:(size_n+1)) = A(c,1:(size_n+1));
            A(c,1:(size_n+1)) = tmp_row;
        end
        %wyznaczenie wspolczynnikow.
        for r=(c+1):size_n
            A(r,c) = A(r,c)/A(c,c);
        end
        %zmiana wartoci macierzy
        % kozystajac z znanego wspolczynika dla danego wiersza.
        for c2=(c+1):(size_n+1)
            for r=(c+1):size_n
                A(r,c2)=A(r,c2)- A(r,c)*A(c,c2);
            end
        end
    end

    %macierz trojkatna i b(odpowiedzi dla macierzy trojkatnej)
    % sa wyliczone teraz operacja odwrotna
    %results_x- wektor ktory przechowyje rozwiazanie
    results_x = zeros(size_n,1); % utworzenie na wartosc zerowa
    % obliczenie ostatneij pozycji bo jest wolna.
    results_x(size_n) = A(size_n,size_n+1)/A(size_n,size_n);
    % zaczynamy wyliczać od przedodstatniego wiersza
    % ostatni wiersz zotal wyzej wyliczony
    r = size_n-1;
    % petla po wierszach od konca
    while(r > 0)
        % sum- zmienna pomocnicza do wyliczneia wartosic
        % zmiennych x znanych w danym wierszu.
        sum = 0;
        for c=r+1:size_n
           % w danym wierszu r znam wartosci x(r+1) i wiedzac jakie
           % maja wspolczyniki z macierzy obliczam ich wplyw
           sum = sum + results_x(c)*A(r,c);
        end
        % majac juz znane sum i kozystajac z wartoci dla wszystkch
        % wspolczynikiow moge obliczyc wartosc nieznaego x.
        results_x(r) = ( A(r,size_n+1)-sum)/A(r,r);
        r= r-1;
    end

    %obliczenie bledu rozw (% norma residuum)
    r = M*results_x - b;
    r = norm(r);
    time =toc;
    disp(results_x);
end
```

```
function [results_time] = wykres_residuum_time()
    value = 10;
    iteration = 8;
    results_r = zeros(iteration,1);
    results_time = zeros(iteration,1);
    count_equations = zeros(iteration,1);

    for i=1:iteration
        fprintf("Iteracja %d\n",i);
        size = value*2^i;
        [A, b] = CreatMatrix_A(size);
        [r, time] = gauss(A,b);
        results_r(i) = r;
        results_time(i) = time;
        count_equations(i) = size;
    end
    figure()
    plot(count_equations, results_time);
    title('Zaleznosc czasu od ilosci rownan w ukladzie (typ A)')
    xlabel('ilosc rownan')
    ylabel('Czas')
    figure()
    plot(count_equations, results_r);
    title('Zaleznosc normy residuum od ilosci rownan w ukladzie (typ A)')
    xlabel('ilosc rownan')
    ylabel('norma residuum')
end
```

Dodatek do zadania 3

```
function [x, err] = jacobi(M,b, max_iteration, max_error)
% A - macierz układu równań
% b - wektor wyników
% max_iteration - ilosc iteracji
% max_errror - blad jaki jest juz akceptowalny

% x - wektor argumentów
% err - blad otrzymanego wyniku
    D_diagonal = zeros(size(M));
    x = zeros(size(M,1),1);
    L_U = M;

    % utworzenie macierzy diagonalnej poddiagonalnej+naddiagonalnej
    for i=1:size(M,1)
        D_diagonal(i, i) = M(i, i);
        L_U(i, i) = 0;
    end
    %ustawienie flag dominacji
    dom_row = true;
    dom_col = true;
    % sprawdzenie czy dominacja wystepuje
    for i=1:size(M, 1)
        tmp_row = abs(M(i,:));
        tmp_col = abs(M(:,i));
        sum_row = sum(tmp_row) - abs(M(i,i));
        sum_col = sum(tmp_col) - abs(M(i,i));

        if sum_row > abs(D_diagonal(i,i))
            dom_row = false;
        end
        if sum_col > abs(D_diagonal(i,i))
            dom_col = false;
        end
    end
    %zakończenie jeśli metoda się nie powiedzie
```

```
    if ~(dom_col || dom_row)
        fprintf("Macierz nie ma dominacji diagonalnej wierszowej ani
kolumnowej\n");
        %return;
    end

    err = inf;
    iter = 0;
    %temp_x tymczasowy wektor argumentow
    temp_x = zeros(size(x));

    while err > max_error
           iter = iter + 1;
           % kolejna wartosc argumentow x(i+1)
           temp_x(:,1) = (-inv(D_diagonal)*L_U)*x(:,1) + inv(D_diagonal)*b;
           x(:,1) = temp_x(:,1);
           % Obliczenie bledu
           err = sum(abs(M*x - b));
           if iter > max_iteration
              fprintf("Koniec iteracji, error wynosi: %d\n",err);
              return;
           end
    end
    fprintf("Zakonczono po: %d Error=%d\n", iter, err);
end
```

```
function [] = wykres_jacobi()
    iteration = 30;
    max_err = 0.1e-10;
    results_err = zeros(iteration,1);
    iteration_number = zeros(iteration,1);
    %A = [17 2 11 -3; 4 -12 2 -3; 2 -2 8 -1; 5 -2 1 -9];
    %b = [12; 7; 0; 12];
    size = 10;
    [A, b] = CreatMatrix_A(size);
    for i=1:iteration
        fprintf("Iteracja %d\n",i);
        [x,err] = jacobi(A,b,i,max_err);
        results_err(i) = err;
        iteration_number(i) = i;
    end
    disp(x);
    figure()
    plot(iteration_number, results_err);
    title('Zaleznosc bledu od ilosci iteracji dane:A n=10')
    xlabel('ilosc iteracji')
    ylabel('Error')
end
```