

# Obliczenia naukowe 5

Gabriel Wechta 250111

9 stycznia 2021

## 1 Wstęp

Lista 5 skupia się na zagadnieniu numerycznego rozwiązywania układów równań liniowych. Naturalną strukturą ułatwiającą opis rozwiązywania równań liniowych oraz umożliwiającym zapis w pamięci komputera są macierze. Wtedy zagadnienie sprowadza się do wyznaczenia  $x$  w równaniu:

$$Ax = b,$$

gdzie  $A \in R^{n \times n}$ ,  $x, b \in R^n$ .

### 1.1 Rzadka macierz blokowa

Macierz nazywamy rzadką gdy ma *dużo* elementów zerowych. Jest to istotny rodzaj macierzy z punktu widzenia obliczeń numerycznych ponieważ, ze względu na dużą liczbę  $0$ , nie biorących udziału w obliczeniach, taką macierz można efektywniej zapisywać w pamięci niż za pomocą tablicy tablic, więcej o tym poniżej.

Macierz nazywamy blokową gdy można wydzielić w niej bloki, podmacierze różnych rozmiarów zachowujące wspólne własności. Macierz przedstawiona w zadaniu jest postaci:

$$A = \begin{pmatrix} A_1 & C_1 & 0 & 0 & 0 & \dots & 0 \\ B_2 & A_2 & C_2 & 0 & 0 & \dots & 0 \\ 0 & B_3 & A_3 & C_3 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & B_{v-2} & A_{v-2} & C_{v-2} & 0 \\ 0 & \dots & 0 & 0 & B_{v-1} & A_{v-1} & C_{v-1} \\ 0 & \dots & 0 & 0 & 0 & B_v & A_v \end{pmatrix}$$

gdzie:

1.  $l$  - rozmiar bloków.
2.  $v = n/l$  - liczba bloków.
3.  $A_k \in R^{l \times l}$  - macierz gęsta.
4.  $B_k \in R^{l \times l}$  - macierz, której wyłącznie ostatnia kolumna jest niezerowa.
5.  $C_k \in R^{l \times l}$  - macierz diagonalna.
6.  $0$  - blok samych zer.

### 1.2 Cel zadania

Celem zadania jest rozwiązanie równania  $Ax = b$  korzystając z zoptymalizowanych algorytmów konkretnie pod wyżej opisaną macierz, tj. korzystając z wiedzy na temat struktury macierzy  $A$ . Należy podkreślić metodę eliminacji Gaussa z wyborem elementu głównego i bez, wyznaczanie rozkładu  $\mathbf{LU}$  macierzy  $A$  metodą eliminacji Gaussa z wyborem elementu głównego i bez oraz funkcję obliczającą  $x$  z wcześniej policzonego rozkładu  $\mathbf{LU}$ .

## 2 Algorytmy

Idealna sytuacja dla problemu szukania rozwiązania układu równań  $Ax = b$ , to taka gdy  $A$  jest macierzą trójkątną. Znalazienie rozwiązania sprowadza się wtedy do wyliczenia kolejno  $x_1, x_2, \dots, x_n$  lub  $x_n, x_{n-1}, \dots, x_1$ , korzystając z tego, że

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

lub odpowiednio

$$x_i = (b_i - \sum_{j=i+1}^n a_{ij}x_j)/a_{ii} \quad (2)$$

### 2.1 Eliminacja Gaussa

Celem eliminacji Gaussa jest doprowadzenie macierzy do właśnie takiej postaci, a konkretniej do macierzy trójkątnej górnej, poprzez przeprowadzanie operacji elementarnych na wierszach i kolumnach macierzy. Eliminacja polega na przeprowadzaniu podobnej operacji do tej, której uczy się w szkole - "dodawanie stronami", po uprzednim przemnożeniu wiersza o mnożnik wyznaczony w taki sposób aby wyzerował jeszcze niezerową kolumnę macierzy, oczywiście należy zmienić również wartości wektora  $b$ , ponieważ wracając do analogii ze szkoły, mnożenie wiersza przez mnożnik należy wykonać "stronami". Innymi słowy mnożnik dla wiersza  $k$  jest równy ułamkowi  $\frac{a_{ik}}{a_{kk}}$ . Należy zwrócić uwagę, że w tym miejscu istnieje niebezpieczeństwo dzielenia przez 0, będzie miało to miejsce gdy  $a_{kk} = 0$ . Standardowym obejściem tego problemu jest zamiana wierszy miejscami lub eliminacja Gaussa z wyborem elementu głównego, o której poniżej.

Przy tak zmodyfikowanej macierzy trójkątnej pozostaje jedynie wyliczyć  $x$  korzystając z (2).

```
function gauss(A,b,n,l)
    część przekształcająca macierz
    for k = 1 to n do
        for i = k + 1 to k + (l - k) mod l do
            m = A[i][k] / A[k][k]
            A[i][k] = 0
            for j = k + 1 to min{k + l, n} do
                A[i][j] = A[i][j] - A[k][j] * m
            end
            b[i] = b[i] - m * b[k]
        end
    end
    część obliczająca x
    x = [0.0, 0.0, ..., 0.0]
    for i = n downto 1 do
        s = 0
        for j = i + 1 to min{i + l, n} do
            s = s + A[i][j] * x[j]
        end
        x[i] = (b[i] - s) / A[i][i]
    end
    return x
```

Algorytm 1: Gauss.

### 2.2 Eliminacja Gaussa z wyborem elementu głównego

Wyżej wspomniane niebezpieczeństwo dzielenia przez 0 nie jest jedynym niebezpieczeństwem związanym z eliminacją Gaussa. Niebezpieczeństwo staje się jeszcze poważniejsze gdy przeanalizujemy co się dzieje, gdy dzielimy przez  $\epsilon$  bliski zero. Ze względu na to jak wykonywane jest dzielenie na liczbach zmiennoprzecinkowych, mogą być generowane olbrzymie błędy. Niebezpieczeństwo jest większe ponieważ zamiast otrzymać

spodziewaną informację o wyjątku dzielenie przez zero, otrzymamy wynik, który może być bardzo daleki od prawdy. Drugi przypadek nie zapewnia stabilności numerycznej algorytmowi eliminacji Gaussa. Aby rozwiązać ten problem algorytm rozszerza się o wybór elementu głównego.

Wybór elementu głównego oznacza wybranie w analizowanej kolumnie największego co do wartości bezwzględnej elementu i przestawieniu wierszy macierzy oraz wektora  $b$ , tak aby największy element znalazł się na diagonalii i dopiero wtedy przeprowadzenie eliminacji Gaussa. Przestawienie opisane powyżej jest realizowane komputerowo poprzez tablicę permutacji, w której zapisywana jest kolejność przestawień.

**function** gauss\_choosing\_main\_element( $A, b, n, l$ )

```

 $p = \{1, 2, \dots, n\}$ 
część przekształcająca macierz
for  $k = 1$  to  $n$  do
     $r = k$ 
     $e = \text{abs}(A[k][k])$ 
    for  $i = k + 1$  to  $k + (l - k) \bmod l$  do
        if  $\text{abs}(A[k][p[i]]) > e$  then
             $r = i$ 
             $e = \text{abs}(A[k][p[i]])$ 
        end
    end
     $p[k] \iff p[r]$ 
    for  $i = k + 1$  to  $k + (l - k) \bmod l$  do
         $m = \frac{A[p[i]][k]}{A[p[k]][k]}$ 
         $A[p[i]][k] = 0$ 
        for  $j = k + 1$  to  $\min\{k + 2l, n\}$  do
             $A[p[i]][j] = A[p[i]][j] - A[p[k]][j] \cdot m$ 
        end
         $b[p[i]] = b[p[i]] - m \cdot b[p[k]]$ 
    end
end
część obliczająca  $x$ 
for  $i = n$  downto  $1$  do
     $s = 0$ 
    for  $j = i + 1$  to  $\min\{i + l, n\}$  do
         $s = s + A[p[i]][j] \cdot x[j]$ 
    end
     $x[i] = \frac{b[p[i]] - s}{A[p[i]][i]}$ 
end
return  $x, p$ 

```

**Algorytm 2:** Gauss z częściowym wyborem elementu głównego.

## 2.3 Rozkład LU

Rozkład LU jest kolejnym sposobem przedstawienia macierzy  $A$  w sposób przyjaźniejszy metodom numerycznym. Pomysł jest taki, aby  $A$  zapisać jako iloczyn macierzy  $L$  - trójkątnej dolnej i  $U$  - trójkątnej górnej, tj:

$$A = LU \quad (3)$$

wtedy rozwiązanie  $Ax = b$  dzieli się dwa etapy:

1. rozwiązanie  $Lz = b$ , względem  $z$ ,
2. rozwiązanie  $Ux = z$ , względem  $x$ .

Obliczenie każdego z etapów jest łatwe ze względu na to, że są to macierze trójkątne.

Wyznaczenie rozkładu  $A = LU$  tak aby był jednoznaczny wymaga zapisania diagonalii jednej z macierzy

1-kami.

Algorytm obliczenia  $LU$  jest taki sam jak wcześniejsza eliminacja Gaussa lub odpowiednio eliminacja Gaussa z wyborem elementu głównego z tą różnicą, że zamiast podstawiać 0 w miejsce eliminowanego elementu podstawiany jest mnożnik (w algorytmach  $m$ ). Poniżej znajduje się pseudokod algorytmu obliczającego  $x$  przy użyciu rozkładu  $LU$ .

```
function lu( $A, p, b, n, l$ )
     $y = [0.0, 0.0, \dots, 0.0]$ 
    for  $i = 1$  to  $n$  do
         $s = 0$ 
        for  $j = \max(l \cdot \lfloor \frac{i-1}{l} \rfloor, 1)$  to  $i - 1$  do
             $s = s + A[p[i]][j] \cdot y[j]$ 
        end
         $y[i] = b[p[i]] - s$ 
    end
     $x = [0.0, 0.0, \dots, 0.0]$ 
    for  $i = n$  downto 1 do
         $s = 0$ 
        for  $j = i + 1$  to  $\min\{i + l, n\}$  do
             $s = s + A[p[i]][j] \cdot x[j]$ 
        end
         $x[i] = \frac{y[i] - s}{A[p[i]][i]}$ 
    end
    return  $x$ 
```

**Algorytm 3:** Rozwiązanie równań przy użyciu rozkładu  $LU$ .

Aby otrzymać algorytm na wyliczenie  $x$  z  $LU$  bez wyboru elementu głównego, w powyższym pseudokodzie należy zamienić każde  $p[i]$  na  $i$  oraz nie przekazywać  $p$  do funkcji.

### 3 Optymalizacja

Analizując charakterystyczną postać macierzy  $A$  można dojść do wniosku, że w celu otrzymania macierzy górnotrójkątnej nie trzeba brać pod uwagę elementów znajdujących się pod przekątną w pewnej odległości zależnej od analizowanej kolumny, gdyż są tam już zera, czyli wartości jakie chcemy, żeby tam były. Liczba elementów do usunięcia w kolumnie  $k$ , w bloku rozmiaru  $l$  jest równa:

$$(l - k) \mod l \quad (4)$$

Tym samym nie trzeba iterować po wszystkich elementach kolumny, a wystarczy po kilku pierwszych poniżej elementu przekątniowego. Powyższa eliminacja została już użyta we wcześniejszych pseudokodach algorytmów.

W celu zaoszczędzenia miejsca i czasu wykonywania, w programach zamiast tablicy tablic wykorzystywana jest `SparseMatrixCSC` - struktura z pakietu `SparseArrays` w `Julii`. Taki sposób pamiętania macierzy zajmuje mniej miejsca i wymaga mniej czasu aby dostać się do elementu niż tablica tablic. Nie jest to jednak rozwiązanie idealne, ponieważ dostęp do elementu nie jest w czasie liniowym, niemniej w analizie złożoności założę, że tak jest.

## 4 Złożoność

### 4.1 Eliminacja Gaussa

Zauważmy, że pętla z iteratorem  $k$  wykona się  $n$  razy, a pozostałe dwie pętle wewnętrzne co najwyżej  $l$  razy. Daje to złożoność  $O(n \cdot l^2)$ . Zatem jest to złożoność liniowa.

## 4.2 Eliminacja Gaussa z częściowym wyborem elementu głównego

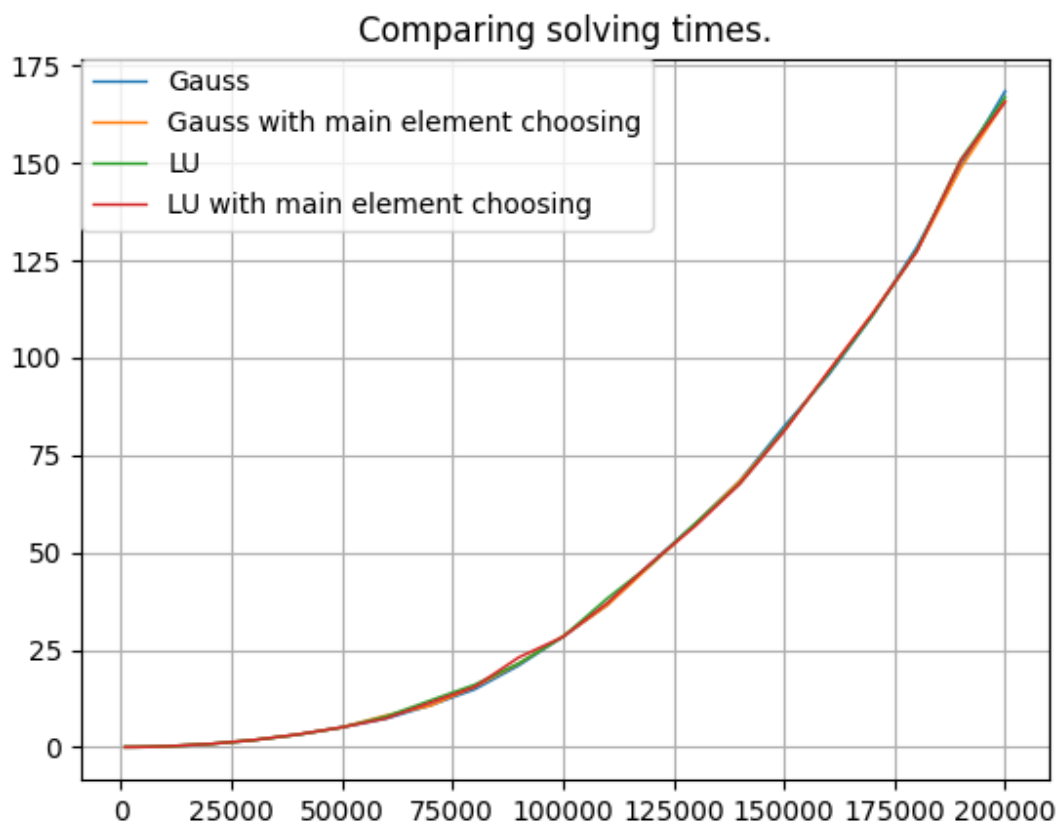
Podobnie jak algorytm eliminacji Gaussa, złożoność eliminacji Gaussa z częściowym wyborem elementu głównego również będzie  $O(n)$  z tą różnicą, że pierwsza, dodatkowa pętla, której iteratorem jest  $i$  wykona się co najwyżej  $l$  razy.

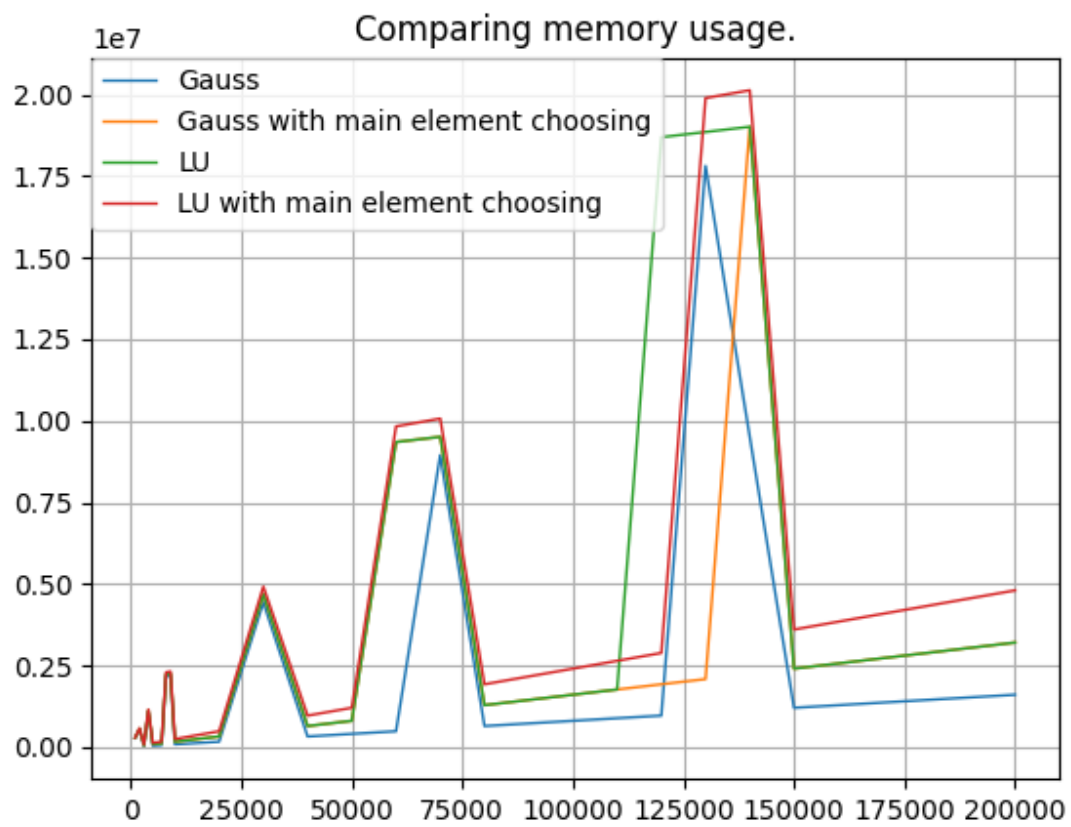
## 4.3 Rozkład LU

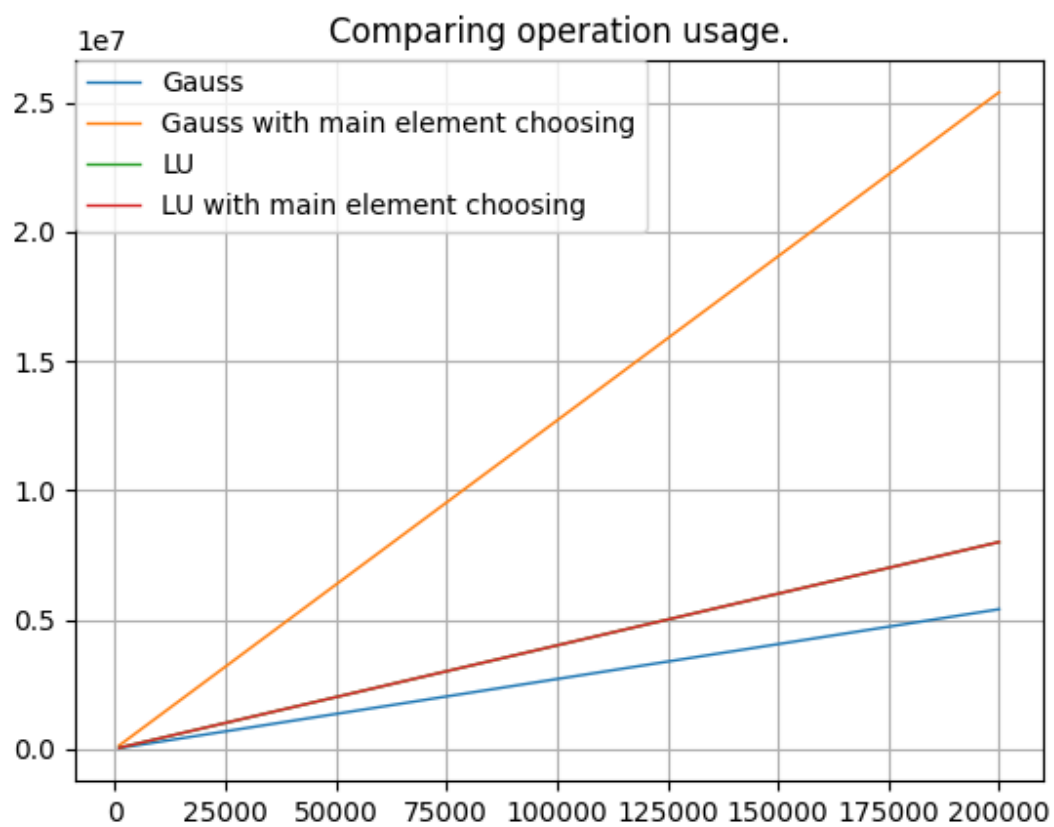
Główna pętla programu wykona się  $n$  razy, wewnętrzna zaś maksymalnie  $i - 1$  co również prowadzi do złożoności  $O(n)$ .

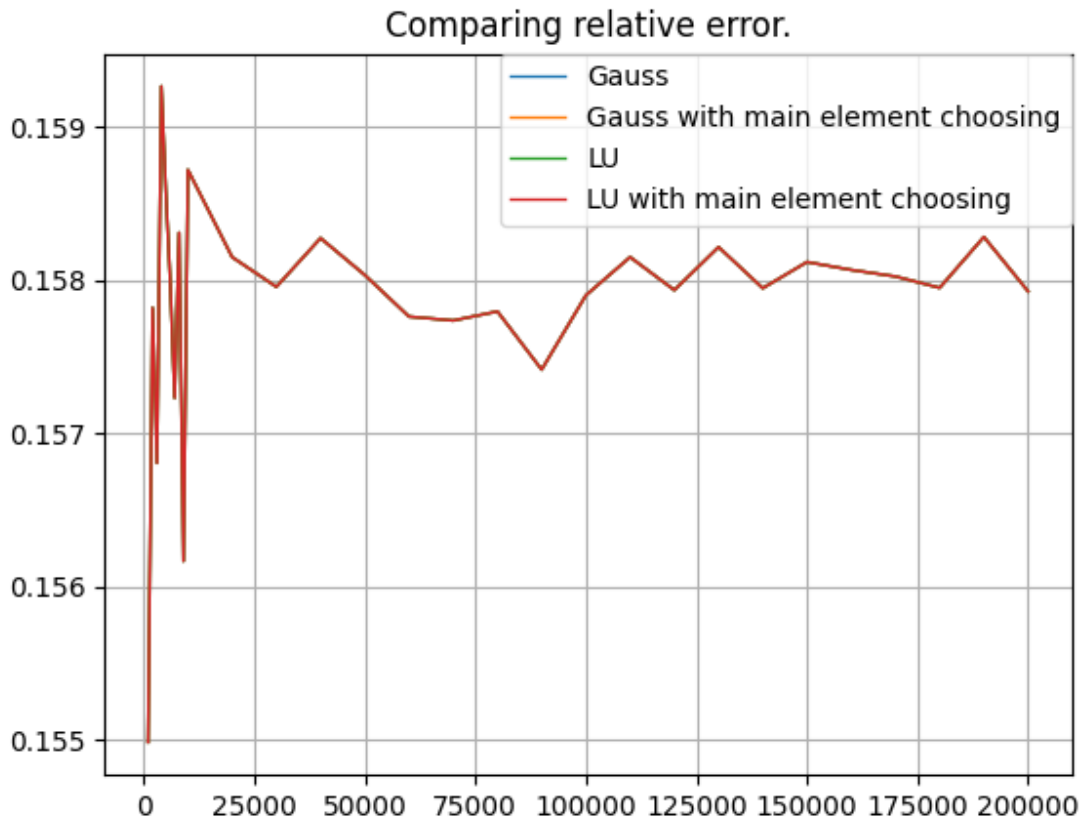
## 5 Wyniki

Poniższe wykresy pokazują wyniki przeprowadzone dla zaimplementowanych w pliku `blocksys.jl` programów. Do zebrania danych zostało użyte makro `Julii @timed`, które zwraca informacje o czasie i wykorzystanej pamięci. Testy zostały przeprowadzone na macierzach wygenerowanych przez funkcję `blockmat` z parametrami: `uwarunkowanie = 1.0`, `l = 5` i rozmiarach od 1000 do 200000 z gęstym próbkowaniem.









## 6 Obserwacje i wnioski

Rzeczywisty czas potrzebny na obliczenie  $x$  jest podobny dla wszystkich programów, ponadto jest kwadratowo zależny od rozmiaru macierzy. Taki stan rzeczy jest spowodowany tym, że dostęp do komórki pamięci przy użyciu `SparseMatrixCSC` nie jest tak naprawdę liniowy a bliższy logarytmowi.

Jeżeli chodzi o pamięć można zauważyć nietypowe zachowanie, mianowicie ogromne skoki w użyciu pamięci i potem nagłe spadki, jest to również spowodowane sposobem pamiętania w pamięci komputera obiektów `SparseMatrixCSC`. Należy zwrócić uwagę również na to, że czysty algorytm eliminacji Gaussa zużywa najmniej pamięci a wcale nie oddstaje czasowo od innych.

Złożoność obliczeniowa z kolei rośnie liniowo dla wszystkich metod. Należy zwrócić szczególną uwagę na algorytm eliminacji Gaussa z częściowym wyborem elementu głównego, który zwiększa liczbę operacji prawie pięciokrotnie w porównaniu do zwykłego algorytmu eliminacji Gaussa.

Pozostaje jeszcze kwestia błędu względnego. Liczę go w następujący sposób  $\delta = \frac{\|I-x\|}{\|x\|}$ , gdzie:

1.  $I$  - wektor 1.0 typu `Float64`.
2.  $\| \cdot \|$  - norma wektora z pakietu `LinearAlgebra`.

Na na tak szerokim wykresie przedstawiającym zależność pomiędzy błędem a metodą w zasadzie nie widać różnicy. Sprawdziłem ręcznie i różnice pomiędzy wartościami  $x$  dla różnych metod, są na ostatnich cyfrach reprezentacji. Sam błąd względny ma tendencję do ustakowania się w okolicach 0.158.

Lista 5 pokazała, że nieznaczna modyfikacja, dokręcenie algorytmu pod konkretny problem może doprowadzić do znacząco lepszych wyników oraz nauczyła wiele o tym jak zaprząć komputer do rozwiązywania ogromnych układów równań liniowych.