# Sprawozdanie

Kajetan Bilski 244942

6 stycznia 2020

## 1 Opis programu

Cały program jest w pliku blocksys.jl, który zawiera tylko moduł blocksys, w którym są wszystkie funkcje programu. Wszystkie liczby rzeczywiste są zapisywane jako Float64.

### 1.1 loadMatrix(filename)

Funkcja zwraca trójkę `(A,n,l)`, gdzie `A` to macierz załadowana z pliku o nazwie `filename` jako `SparseArray`.

### 1.2 loadVector(filename)

Funkcja zwraca wektor `b` z pliku o nazwie `filename`.

### 1.3 generateB(A,n,l)

Funkcja mnoży macierz `A` przez wektor jedynek o długości `n` i zwraca wynik `b`. Przy mnożeniu dla każdego wiersza funkcja bierze pod uwagę tylko pola, które według zadanej postaci macierzy $A$ są niezerowe. Dla stałego $l$ daje to złożoność $O(n)$.

### 1.4 elimination(A,b,n,l,withChoice)

Funkcja dokonuje eliminacji Gaussa w celu znalezienia rozwiązania $Ax = b$ i zwraca wektor `x`. Żeby nie zmieniać `A` ani `b` poza funkcją, najpierw robiona jest głęboka kopia obu zmiennych.
Jeżeli `n == l`, to znaczy, że macierz jest gęsta i funkcja wykonuje standardową wersję eliminacji Gaussa zmieniając przy tym postać `A` i `b`. Najpierw w `n` krokach macierz `A` zamieniana jest na macierz trójkątną. W każdym kroku $i = 1, 2, ..., n$ dla każdego wiersza $j > i$ liczone jest $l_{ij}$ = `A[i,j] / A[i,i]`, następnie dla każdego $x \geqslant i$, `A[x,j]` = `A[x,j]` $-$ `A[x,i]` $* \, l_{ij}$ i na końcu `b[j]` = `b[j]` $-$ `b[i]`. Otrzymujemy macierz $U$. Ta część funkcji składa się z 3 zagnieżdżonych pętli o złożoności $O(n^3)$. Po przekształceniu `A` i `b` funkcja

oblicza wektor `x[1]`, `x[2]`, ..., `x[n]`. W pętli dla $i$ malejącego od `n` do 1 obliczane jest `x[i]` $= \frac{b[i]-A[n,i]*x[n]-A[n-1,i]*x[n-1]-...-A[i+1,i]*x[i+1]}{A[i,i]}$. Ta część funkcji przechodzi raz po każdym niezerowym polu macierzy trójkątnej co daje złożoność $O(n^2)$. Wektor `x` jest rozwiązaniem `Ax=b`. Jeżeli `withChoice` jest prawdą to funkcja działa w trybie z częściowym wyborem elementu głównego. Wtedy przed każdym krokiem $i = 1, ..., n$ eliminacji szukany jest wiersz o numerze $y = i, i+1, ..., n$ z największym $i$-tym elementem, wybrany wiersz zostaje zamieniony miejscami z $i$-tym. Żeby sama zamiana zachowała złożoność $O(1)$ wartości w macierzy nie są zmieniane, zamieniane są tylko numery wierszy w osobnej tablicy referencji pamiętającej miejsce przechowywania każdego wiersza. Złożoność operacji wyboru elementu głównego ma złożoność $O(n)$, ale wykonuje się ona zawsze przed pętlą, której złożoność wynosi $O(n^2)$, więc wybór nie ma wpływu na ogólną złożoność. Złożoność całej funkcji w tym przypadku jest równa $O(n^3)$, ale `n` jest równe `l`, które jest stałą, więc $O(n^3) = O(l^3) = O(1)$.
Jeżeli `n` $>$ `l` to znaczy że macierz nie jest gęsta i funkcja stosuje usprawnienia dla pomijania działań na zerach. Macierz `A` podzielona jest na "bloki" gdzie w każdym bloku wszystkie wiersze mają swoje piersze (od lewej) niezerowe pola w tej samej kolumnie $x$ i same zerowe pola od kolumny $x + 2l + 2$. W każdym $i$-tym kroku zamiast mnożyć i odejmować cały $i$-ty wiersz od wszystkich o numerach $> i$, modyfikowane są tylko wiersze i kolumny do końca bloku. Jeśli $i$ jest większe lub równe indeksowi kolumny w której znajdują się pierwsze pola niezerowe następnego bloku, to modyfikowane się wiersze i kolumny do końca następnego bloku. Tak samo zmniejszony jest zakres wyboru elementu głównego. Zmniejsza to złożoność tej części funkcji do $O(n * l^2)$. W drugiej części do obliczania każdego `x[i]` też wykorzystywane jest to, że w każdym wierszu wszystkie pola niezerowe w $i$-tym wierszu mieszczą się w kolumnach od $i$ do $i + 2l + 1$, co zmniejsza złożoność do $O(n * l)$. W tym przypadku złożoność całej funkcji wynosi $O(n * l^2) = O(n)$.

## 1.5 decomposition(A,n,l,withChoice)

Funkcja dokonuje rozkładu LU macierzy `A` z opcjonalnym częściowym wyborem elementu głównego i zwraca trójkę `(LU,ref,nzCount)`, gdzie `ref` jest wektorem pamiętającym indeksy wierszy, a `nzCount` jest wektorem wektorów pamiętającym położenie pól niezerowych w macierzy $L$. Funkcja najpierw robi głęboką kopię `A`, żeby nie zmieniać jej poza sobą.
Działa ona podobnie jak pierwsza część funkcji `elimination` z tą różnicą że w `elimination` zapisywana była tylko macierz $U$ podczas gdy tutaj trzeba pamiętać też $L$ oraz nie używany jest wektor `b`. Macierz $LU$ zapisywana jest w sposób pokazany na wykładzie w miejscu macierzy `A`.
Po obliczeniu każdego $l_{ij} =$ `A[i,j]/A[i,i]$ podstawiane jest $\verb`$A[i,j]| = l_{ij}$. Oprócz tego do pamiętania niezerowych wartości używany jest wektor wektorów `nzCount`, gdzie `nzCount[i]` jest wektorem kolumn w których znajdują się wszystkie niezerowe pola $i$-tego wiersza macierzy $L$ (nie licząc pól na przekątnej). Do `nzCount` na bierząco w czasie $O(1)$ dopisywane są numery pól jak są wstawiane. W każdej kolumnie $L$ jest co najwyżej $l + 2$ pól niezerowych, czyli

$O(n*l)$, co będzie ważne dla funkcji `solveWithLU`. Złożoność jest taka sama jak `elimination`, czyli $O(n)$.

### 1.6 solveWithLU(LU,ref,nzCount,n,l,b)

Funkcja rozwiązuje równanie $LUx = b$ i zwraca `x`.
Funkcja rozwiązuje dwa trójkątne układy równań $Ly = b$ i $Ux = y$, w tej kolejności. Żeby obliczyć $y$, używany jest wzór dla $i = 1, 2, ..., n$, $y_i =$ `b[i]` $-$ `A[1,i]` $* y_1 -$ `A[2,i]` $* y_2 - ... -$ `A[i-1,i]` $* y_{i-1}$. Używamy tylko niezerowych wartości z `A`, które znamy z `nzCount` i wiedząc że jest ich $O(n*l)$, otrzymujemy taką samą złożoność. W drugiej części funkcja oblicza $Ux = y$ w taki sam sposób jak w `elimination` w czasie $O(n*l)$. Złożoność całej funkcji wynosi $O(n*l) = O(n)$.

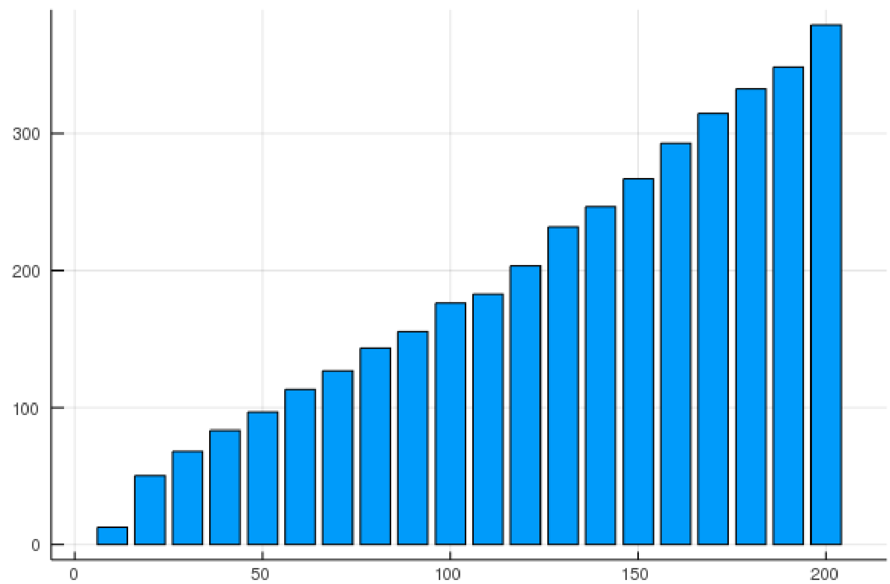
### 1.7 getError(x,n)

Funkcja zwraca błąd względny obliczany wzorem `norm(x - ones(n)) / norm(ones(n))`.

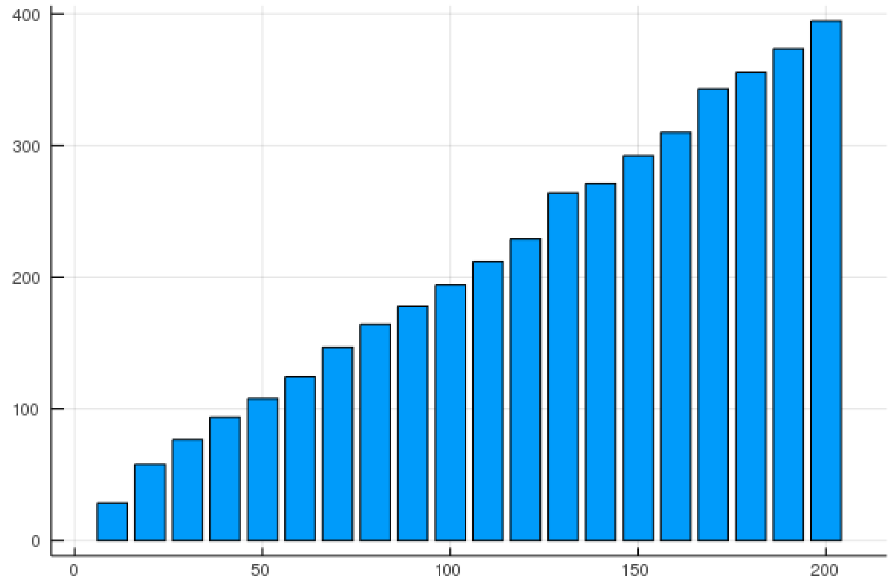
### 1.8 printX(filename,x,n,withError)

Funkcja zapisuje do pliku o `filename` wektor `x`. Jeżeli `withError` jest prawdą to w pierwszej linijce zapisuje błąd względny zwrócony przez `getError()`.

## 2 Eksperymenty

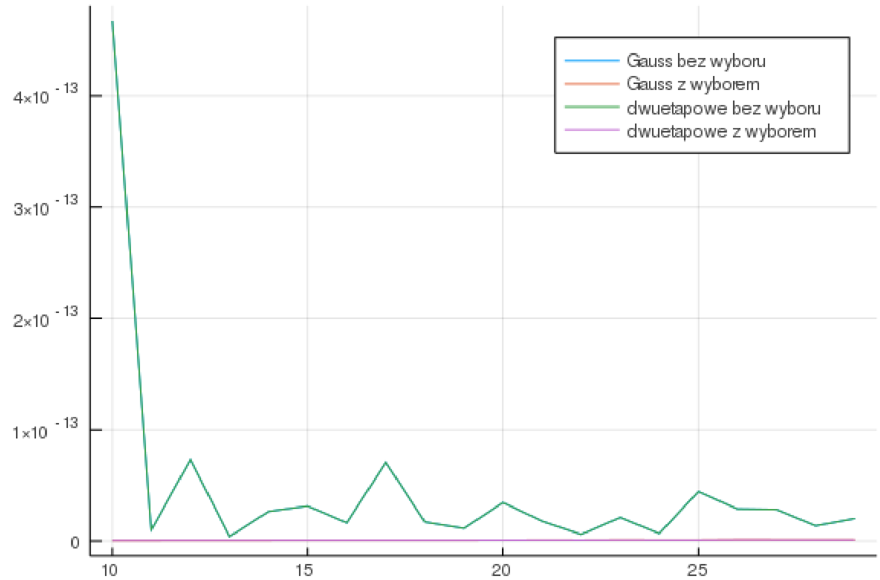



Rysunek 1: Czas rozwiązywania $Ax = b$ metodą eliminacji Gaussa w milisekundach w zależności od $n$.

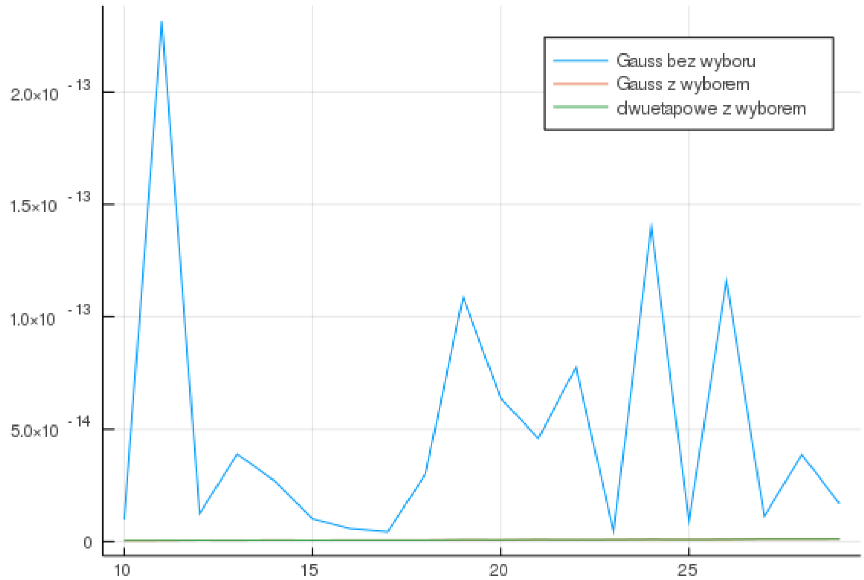



Rysunek 2: Czas rozwiązywania $Ax = b$ dwuetapowo z rozkładem LU w milisekundach w zależności od $n$.

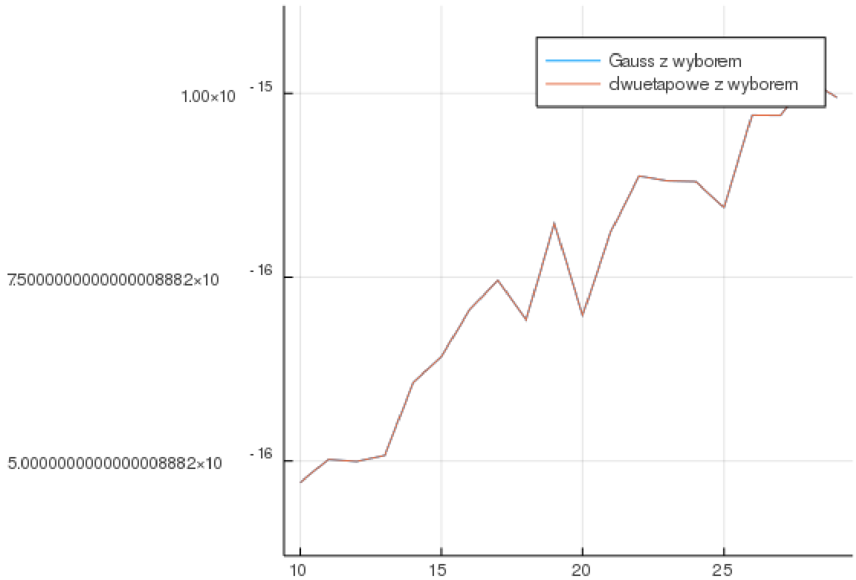
Jak widać z wykresów złożoność czasowa obu algorytmów w zależności od $n$ jest liniowa.




Rysunek 3: Błąd względny wyniku w zależności od współczynnika uwarunkowania macierzy $A$.

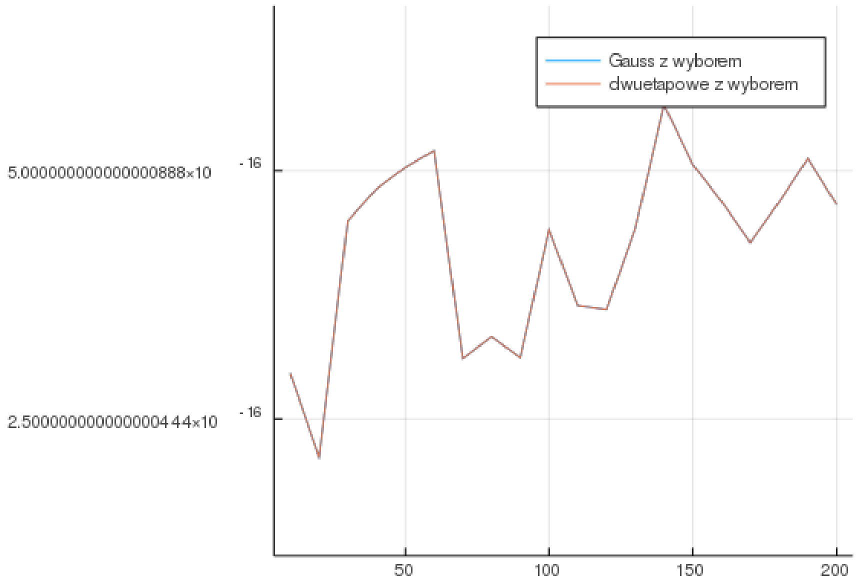



Rysunek 4: Błąd względny wyniku w zależności od współczynnika uwarunkowania macierzy $A$.




Rysunek 5: Błąd względny wyniku w zależności od współczynnika uwarunkowania macierzy $A$.

Jak widać największe błędy generuje metoda dwuetapowa bez wyboru elementu głównego, potem metoda eliminacji Gaussa bez wybory, a obie metody z częściowym wyborem dają takie same błędy względne.




Rysunek 6: Błąd względny wyniku w zależności od współczynnika uwarunkowania macierzy $A$.

Błąd względny jest rośnie wraz ze wzrostem $n$ lub współczynnika uwarunkowania macierzy.