Obliczenia naukowe
Lista 5

Arkadiusz Ziobrowski
229728

Wprowadzenie

Rozwiązania poniższych zadań uwzględniają specyficzną postać macierzy wejściowej $\mathbf{A}$:

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

Macierz $\mathbf{A}$ jest macierzą rzadką i blokową o powyższej strukturze, gdzie $v = n/l$, przy założeniu, że $n \geq 4$ jest podzielne przez rozmiar bloków $l \geq 2$. Ponadto bloki macierzy $\mathbf{A}$ mają dla $k = 2, ..., v$ następujące postaci:

- $\mathbf{A_k}$ jest kwadratową macierzą gęstą.
- $\mathbf{0}$ jest kwadratową macierzą zerową stopnia l.
- $\mathbf{B_k}$ jest kwadratową macierzą, w której tylko dwie ostatnie kolumny są niezerowe, to jest:

$$\mathbf{B_k} = \begin{pmatrix} 0 & \cdots & 0 & b^k_{1,l-1} & b^k_{1,l} \\ 0 & \cdots & 0 & b^k_{2,l-1} & b^k_{2,l} \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & b^k_{l,l-1} & b^k_{l,l} \end{pmatrix}$$

- $\mathbf{C_k}$ jest kwadratową macierzą diagonalną, to jest:

$$\mathbf{C_k} = \begin{pmatrix} c^k_1 & 0 & 0 & \cdots & 0 \\ 0 & c^k_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c^k_{l-1} & 0 \\ 0 & \cdots & 0 & 0 & c^k_l \end{pmatrix}$$

Dla przedstawionej powyżej postaci macierzy zostały zaadaptowane standardowe algorytmy, które dzięki uwzględnieniu w nich rzadkości i regularności występowania elementów zerowych i niezerowych, wynikających z blokowo-taśmowej postaci macierzy, mogły zostać zoptymalizowane dla konkretnych problemów związanych z zadaną macierzą.

1 Zadanie pierwsze

1.1 Opis problemu

Celem zadania była implementacja funkcji rozwiązującej układ $\mathbf{Ax} = \mathbf{b}$ metodą eliminacji Gaussa, uwzględniającą specyficzną postać macierzy $\mathbf{A}$ dla wariantów bez wyboru elementu głównego oraz z częściowym wyborem elementu głównego.

1.2 Rozwiązanie

1.2.1 Algorytm

Metoda eliminacji Gaussa polega na stopniowej eliminacji niewiadomych tak, aby układ równań liniowych $\mathbf{Ax} = \mathbf{b}$ zastąpić równoważnym mu układem $\mathbf{Ux} = \mathbf{c}$ z macierzą trójkątną górną $\mathbf{U}$. Nowy układ można łatwo rozwiązać, wyznaczając niewiadome od ostatniej do pierwszej. Wariant bez wyboru elementu głównego zakłada, że elementem głównym w k-tym kroku jest element a_{kk} macierzy $\mathbf{A}$, zaś wariant z częściowym wyborem elementu głównego zakłada, że jako element główny zostanie wybrana wartość największa co do skali rzędu w kolumnie k-tej. Różnice między tymi wariantami zostaną opisane dokładniej w dalszej części tej podsekcji.

W k-tym kroku eliminujemy zmienną x_k z równań od $(k+1)$-go do n-tego poprzez odjęcie pierwszego równania pomnożonego przez

$$l_{ik} = \frac{a_{ik}^{(1)}}{a_{kk}^{(1)}} \quad dla \quad i = k+1, ...n \tag{1}$$

od reszty. Uwzględniając blokowo-taśmową postać macierzy $\mathbf{A}$ nie będzie trzeba jednak wykonać $n-k$ mnożeń i odejmowań. Wykorzystując regularność występowania elementów niezerowych w k-tym kroku dla $k = 1, ..., n-2$ algorytmu wystarczy wykonać operacje dla $l-(k \mod l)$ równań, gdy $k \mod l \leq l-2$ lub dla $2l-(k \mod l)$ równań w przeciwnym przypadku. W ostatnich dwóch krokach trzeba wykonać $l-(k \mod l)$ operacji. Wynika to ze specyficznej postaci macierzy $\mathbf{A}$, która dla bloków o indeksach $i = 1, ..., v-1$ wymaga większej ilości operacji dla ostatnich dwóch kolumn bloku $\mathbf{A_i}$, gdyż znajdująca się pod blokiem macierz $\mathbf{B_{i+1}}$ ma niezerowe wyłącznie dwie ostatnie kolumny.

Obserwacją, która pomaga zaadaptować algorytm pod specyficzne dane wejściowe i zmniejszyć złożoność metody eliminacji Gaussa jest również ilość współczynników w rzędzie macierzy $\mathbf{A}$, które musimy przemnożyć i odjąć. Dla podstawowej wersji algorytmu Gaussa należy w k-tym kroku przemnożyć $n-k$ współczynników przez l_{ik} dla $i = k+1, ..., n$, a następnie odjąć je od $n-k$ współczynników w rzędach poniżej k-tego. Postać macierzy $\mathbf{A}$ pozwala jednak na działanie na co najwyżej $l+1$ współczynnikach dla wariantu bez wyboru elementu głównego oraz na co najwyżej $2*l+1$ współczynnikach dla wariantu z częściowym wyborem elementu głównego, co wynika z regularności w blokach $\mathbf{A_i}$ oraz postaci diagonalnej bloku $\mathbf{C_i}$. W wariancie z częściowym wyborem elementu głównego większy zakres wynika z permutacji rzędów, a co za tym idzie możliwego przesunięcia najdalszego co do indeksu kolumny niezerowego elementu o maksymalnie l kolumn.

Po pierwszym kroku macierz $\mathbf{A}$ będzie wyglądać następująco:

$$\mathbf{A^{(2)}} = \begin{pmatrix} a_{11} & a_{12} & \cdots & 0 & 0 & 0 \\ 0 & a'_{22} & \cdots & 0 & 0 & 0 \\ 0 & a'_{32} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 & 0 \\ 0 & a'_{l2} & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn-1} & a_{nn} \end{pmatrix}$$

Pierwsza kolumna została wyzerowana poniżej pierwszego rzędu, co odpowiada wyeliminowaniu zmiennej x_1 z równań od drugiego do n-tego. Pierwszy rząd nie zmienił się, zaś wartości w kolumnach w pozostałych l pierwszych rzędach zostały pomniejszone o $l_{i1} * a_{1j}$. Po n krokach algorytmu macierz $\mathbf{A}$ zostanie sprowadzona do postaci macierzy trójkątnej górnej.

W czasie wykonywania algorytmu ważne jest również aktualizowanie wektora prawych stron $\mathbf{b}$ w celu zachowania równoważności układu równań. Przy odejmowaniu rzędów w macierzy $\mathbf{A}$ należy odjąć odpowiadające im rzędy wektora $\mathbf{b}$, przy czym odjemnik musi być pomnożony przez wartość l_{ik}.

Po sprowadzeniu układu $\mathbf{Ax} = \mathbf{b}$ do równoważnego mu układu $\mathbf{Ux} = \mathbf{c}$ z macierzą górnotrójkątną $\mathbf{U}$ jesteśmy w stanie łatwo wyznaczyć niewiadome od ostatniej do pierwszej:

$$\begin{aligned} x_n &= \frac{b_n}{\mathbf{U_{nn}}} \\ x_k &= \frac{b_k - \sum_{j=k+1}^{\min(n,k+l)} \mathbf{U_{kj}} * x_j}{\mathbf{U_{kk}}} \quad dla \quad k = n-1, ..., 1 \end{aligned} \tag{2}$$

Wyznaczenie niewiadomych możemy wykonać *in situ*, czyli bez alokacji dodatkowej pamięci na rozwiązanie układu. Rozwiązanie x_k możemy przechowywać na miejscu b_k w wektorze prawych stron $\mathbf{b}$. Nie będzie to wpływać na rozwiązani układu, gdyż wyznaczone wcześniej rozwiązania x_k nie są modyfikowane przy wyliczaniu x_{k-1}.

Znając podejście do metody eliminacji Gaussa zaadaptowanej dla specyficznej postaci macierzy **A** można podać teraz dokładnie różnice między wariantami algorytmu bez wyboru elementu głównego oraz z częściowym wyborem elementu głównego. Wariant z częściowym wyborem elementu głównego chroni przed wyborem zera lub wartości małej w porównaniu z innymi elementami wiersza jako elementu głównego. Aby wybrać element główny najpierw wyznaczana jest skala rzędu, czyli:

$$s_i = \max_{1 \le j \le n} |a_{ij}| \quad dla \quad i = 1, ..., n \tag{3}$$

Skalowalny wybór elementu głównego polega na wyborze w k-tej kolumnie elementu takiego, że $|a_{ik}|/s_i$ jest największy. Gdy znajdziemy taki element wykonujemy przestawienie rzędu k-tego i i-tego. Do przechowywania przestawień wykorzystywany jest wektor permutacji. Wariant ten różni się od wariantu bez wyboru elementu głównego zmianami we wzorach (1) oraz (2). W (1) w mianowniku wykorzystujemy element $a_{p[k]k}$, który jest wybranym elementem głównym, zaś w (2) zmieniają się zakres sumowania oraz indeksy rzędów. Zakres sumowania należy rozszerzyć do $min(n, p[k] + 2 * l + 1)$, co wynika z przestawień rzędów i możliwości przesunięcia elementu niezerowego w rzędzie o l kolumn. Indeksy rzędów zmieniają się natomiast z i-tego na $p[i]$-ty, gdyż musimy brać pod uwagę przestawienia rzędów wynikajace z wyborów elementów głównych.

Algorytm eliminacji Gaussa ze skalowalnym wyborem wierszy głównych jest następujący:

```
1: function GaussElimination(A, b, n, l)
2:     compute s_i = max_{1≤j≤n} |a_ij| for i = 1, ..., n
3:     for k = 1 to n − 1 do
4:         choose j such that |a_{p[j]k}|/s_i is the largest in the column
5:         p_k ⟷ p_j
6:         if k mod l ≤ l − 2 ∨ k == n − 1 then
7:             rows = l − k mod l
8:         else
9:             rows = 2 ∗ l − k mod l
10:        end if
11:        for i = k + 1 to k + rows do
12:            l_{p[i]k} = a_{p[i]k}/a_{p[k]k}
13:            a_{p[i]k} = 0
14:            if k + 2l ≤ n then
15:                cols = 2l
16:            else
17:                cols = −k + n
18:            end if
19:            for j = k + 1 to k + cols do
20:                a_{p[i]j} = a_{p[i]j} − l_{p[i]k} ∗ a_{p[k]j}
21:            end for
22:            b_{p[i]} = b_{p[i]} − l_{p[i]k} ∗ b_{p[k]}
23:        end for
24:    end for
25:    b_{p[n]} = b_{p[n]}/a_{p[n]n}
26:    for i = n − 1 down to 1 do
27:        sum = b_{p[i]}
28:        offset = min(n, p[i] + 2l + 1)
29:        for j = offset down to i + 1 do
30:            sum = sum − a_{p[i]j} ∗ b_{p[j]}
31:        end for
32:        b_{p[i]} = sum/a_{p[i]i}
33:    end for
34:    return b
35: end function
```

W linijkach 2 - 24 odbywa się sprowadzanie macierzy $\mathbf{A}$ do postaci macierzy górnotrójkątnej $\mathbf{U}$, zaś w linijkach 25 - 33 jest wyliczane rozwiązanie układu $\mathbf{Ux} = \mathbf{c}$. Algorytm ten uogólnia się do wariantu bez wyboru elementu głównego dla $p[i] = i$ oraz odpowiednio zmienionych warunków pętli, podanych w powyższych akapitach.

1.2.2 Analiza złożoności

Złożoność obliczeniowa Metoda eliminacji Gaussa bez wyboru elementu głównego wykonuje n iteracji podczas sprowadzania macierzy $\mathbf{A}$ do postaci macierzy górnotrójkątnej $\mathbf{U}$. Dla każdej kolumny wykonywanych jest maksymalnie $l+1$ odejmowań rzędów, a w każdym rzędzie należy odjąć od siebie co najwyżej $l+1$ współczynników układu równań. Wyznaczanie macierzy $\mathbf{U}$ ma zatem złożoność $O(n * l^2)$. Rozwiązywanie układu równań $\mathbf{Ux} = \mathbf{c}$ wymaga n iteracji, w których każda odpowiada za wyznaczenie jednej zmiennej x_i. Do wyznaczenia zmiennej x_i potrzeba kolejnych co najwyżej $l+1$ iteracji po współczynnikach z macierzy $\mathbf{U}$. Złożoność metody eliminacji Gaussa bez wyboru elementu głównego to zatem $O(n * 2l + n * l)$, co przy założeniu, że l jest stałą daje $O(n)$.

Metoda eliminacji Gaussa z częściowym wyborem elementu głównego wprowadza dodatkowe wymagania obliczenia skali dla każdego z rzędów macierzy $\mathbf{A}$ i wyboru elementu głównego. Obliczenie skali dla całej macierzy $\mathbf{A}$ wykonuje się w czasie $O(n)$, zaś wybór elementu w czasie $O(l+1) = O(1)$, przy założeniu, że l jest stałą. Również zwiększenie ilości kolumn, które trzeba od siebie odjąć przy odejmowaniu rzędów zwiększa złożoność jedynie o stałą, dlatego metoda eliminacji Gaussa z częściowym wyborem elementu głównego również ma złożoność $O(n)$.

Złożoność pamięciowa Dzięki regularności występowania elementów niezerowych w macierzy $\mathbf{A}$ można określić złożoność pamięciową algorytmu eliminacji Gaussa. Blok $\mathbf{A_i}$ ma co najwyżej l^2 elementów niezerowych, blok $\mathbf{B_i}$ ma co najwyżej $2*l$ elementów niezerowych, a blok $\mathbf{C_i}$ l takich elementów. Mamy v bloków $\mathbf{A_i}$ oraz $v-1$ bloków $\mathbf{B_i}$ i $\mathbf{C_i}$. Przypomnijmy, że $v = n/l$. Reprezentacja macierzy wymaga zatem $O(v * (l^2 + 3l)) = O(n/l * (l^2 + 3l)) = O(nl)$. Wektor prawych stron $\mathbf{b}$ wymaga $O(n)$ pamięci. Obliczanie rozwiązania układu przeprowadzamy *in situ*, zatem łącznie potrzebujemy $O(nl)$ pamięci dla algorytmu eliminacji Gaussa bez wyboru elementu głównego. Przy częściowym wyborze elementu głównego potrzebujemy również $O(n)$ pamięci na przechowywanie permutacji i $O(n)$ pamięci na skalę każdego z wierszy. Asymptotycznie daje to nam również złożoność pamięciową $O(nl)$.

1.2.3 Kwestie implementacyjne

W języku programowania `Julia` zostały zaimplementowane funkcje `gaussian_elimination!` dla wariantu bez wyboru elementu głównego oraz funkcja `gaussian_elimination_pivoting!` dla wariantu z częściowym wyborem elementu głównego. Do przechowywania macierzy rzadkiej $\mathbf{A}$ została użyta struktura `SparseMatrixCSC` z języka `Julia`, która pozwala na efektywne przechowywanie elementów niezerowych macierzy w formacie CSC (Compressed Sparse Column) i efektywny dostęp do nich. Funkcje te zostały umieszczone w module `blocksys`, dodatkowo wyposażonym w funkcje użytkowe pozwalające między innymi na wczytywanie macierzy wejściowych i wektorów prawych stron z pliku i zapis rozwiązań do pliku.

1.3 Wyniki i interpretacja

Dla zaimplementowanych funkcji zostały stworzone programy testujące, które jako dane wejściowe przyjmowały dane wygenerowane przez funkcję `blockmat` z modułu `matrixgen`.

	Bez wyboru elementu głównego		Z wyborem elementu głównego	
n	czas (s)	δ (błąd względny)	czas (s)	δ (błąd względny)
16	0.084066	9.836601059614785e-16	0.132196	6.787359152847496e-16
10 000	0.157238	1.0083140309679606e-14	0.180174	5.401323443968982e-16
50 000	5.849069	1.9071532832302597e-13	6.274227	5.248824020867468e-16

Wyniki metody eliminacji Gaussa z częściowym wyborem elementu głównego charakteryzują się mniejszym błędem względnym. Mimo, że metoda ta wymaga nieznacznie większego nakładu czasowego to jest ona bezpieczna i bardziej dokładna.

1.4 Wnioski

Metoda eliminacji Gaussa pozwala na rozwiązanie układu równań w postaci $\mathbf{Ax} = \mathbf{b}$. Wariant z częściowym wyborem elementu głównego zabezpiecza przed wykorzystaniem elementu zerowego leżącego na przekątnej jako elementu głównego i chroni przed znaczną utratą dokładności w przypadku wybrania jako elementu głównego wartości małej co do modułu w porównaniu do pozostałych wartości w wierszu.

2 Zadanie drugie

2.1 Opis problemu

Celem zadania była implementacja funkcji wyznaczającej rozkład $\mathbf{LU}$ macierzy $\mathbf{A}$ metodą eliminacji Gaussa, uwzględniającą specyficzną postać macierzy $\mathbf{A}$ dla wariantów bez wyboru elementu głównego oraz z częściowym wyborem elementu głównego.

2.2 Rozwiązanie

2.2.1 Algorytm

Eliminacja Gaussa jest równoważna rozkładowi macierzy $\mathbf{A}$ na iloczyn $\mathbf{A} = \mathbf{LU}$, gdzie $\mathbf{L}$ jest macierzą trójkątną dolną mnożników, a $\mathbf{U}$ jest macierzą trójkątną górną, która jest macierzą wejściową przy wyliczaniu rozwiązania układu równań. Aby uzyskać rozkład $\mathbf{LU}$ należy wykonać algorytm analogiczny jak w zadaniu pierwszym, z wyłączeniem obliczania rozwiązania układu równań $\mathbf{Ux} = \mathbf{c}$. Rozkład $\mathbf{LU}$ możemy wyznaczyć *in situ*, zapamiętując mnożniki l_{ik} na miejscach wyzerowanych współczynników w k-tym kroku algorytmu. Dla wariantu z częściowym wyborem elementu głównego wyjściowym rozkładem jest rozkład $\mathbf{LU}$ taki, że $\mathbf{PA} = \mathbf{LU}$, gdzie $\mathbf{P}$ jest macierzą permutacji.

2.2.2 Analiza złożoności

Złożoność obliczeniowa Złożoność obliczeniowa wyznaczania rozkładu $\mathbf{LU}$ jest identyczna jak pierwszy fragment algorytmu z pierwszego zadania. Jest to zatem $O(n)$, przy założeniu, że l jest stałą.

Złożoność pamięciowa Pamięć potrzebna do wyznaczenia rozkładu $\mathbf{LU}$ jest identyczna jak pamięć potrzebna do eliminacji Gaussa na macierzy $\mathbf{A}$, jest to więc $O(nl)$, co zostało pokazane w analize złożoności w zadaniu pierwszym. Wyjściowy rozkład zajmuje dokładnie tyle pamięci, ile było potrzebne na przechowywanie macierzy rzadkiej $\mathbf{A}$.

2.2.3 Kwestie implementacyjne

W języku programowania `Julia` zostały zaimplementowane funkcje `lufactorization!` dla wariantu bez wyboru elementu głównego oraz funkcja `lufactorization_pivoting!` dla wariantu z częściowym wyborem elementu głównego. Do przechowywania macierzy rzadkiej $\mathbf{A}$ została użyta struktura `SparseMatrixCSC` z języka `Julia`, która pozwala na efektywne przechowywanie elementów niezerowych macierzy w formacie CSC (Compressed Sparse Column) i efektywny dostęp do nich. Funkcje te zostały umieszczone w module `blocksys`, dodatkowo wyposażonym w funkcje użytkowe pozwalające między innymi na wczytywanie macierzy wejściowych i wektorów prawych stron z pliku i zapis rozwiązań do pliku. Do efektywnego przechowywania rozkładu $\mathbf{LU}$ zostały stworzona macierz rzadka odpowiadające macierzy dolnotrójkątnej $\mathbf{L}$, zaś macierz $\mathbf{U}$ przechowywana jest na miejscu macierzy wejściowej $\mathbf{A}$. Są one zwracane jako wyniki funkcji. Aby pamięć zaalokowana, aby przechowywać rozkład $\mathbf{LU}$ była identyczna jak pamięć potrzebna do przechowywania macierzy $\mathbf{A}$, pod koniec algorytmu jest wywoływana metoda biblioteczna `dropzeros!`, które usuwa z macierzy rzadkiej elementy zerowe. Dzięki temu wyzerowane w macierzy $\mathbf{A} = \mathbf{U}$ elementy zostają usunięte. W module `blocksys` zostały ponadto dołączone funkcje użytkowe pozwalające na zweryfikowanie poprawności rozkładu $\mathbf{LU}$. Funkcje `permuted` pozwala

na nałożenie na pomnożenie macierzy przez macierz permutacji. Wywołanie `L * permuted(U, n, p)` da zatem macierz **A**.

3 Zadanie trzecie

3.1 Opis problemu

Celem zadania była implementacja funkcji rozwiązującej układ $\mathbf{Ax} = \mathbf{b}$, jeśli wcześniej został już wyznaczony rozkład **LU** przez funkcję z poprzedniego zadania.

3.2 Rozwiązanie

3.2.1 Algorytm

Aby rozwiązać układ $\mathbf{Ax} = \mathbf{b}$, jeśli wcześniej został już wyznaczony rozkład **LU** należy wykonać $\mathbf{Ly} = \mathbf{b}$, a następnie $\mathbf{Ux} = \mathbf{y}$. Wyliczenie $\mathbf{Ly} = \mathbf{b}$ odpowiada aktualizacjom wektora prawych stron z metody eliminacji Gaussa, przy odejmowaniu rzędów. Jest to widoczne w linijce `22` w pseudokodzie przedstawionym w zadaniu pierwszym. Obliczenie $\mathbf{Ux} = \mathbf{y}$ odpowiada natomiast fragmentowi algorytmu widocznego w linijkach `25 - 33` z pseudokodu z zadania pierwszego. Jest to więc podejście analogiczne jak w przypadku zadania pierwszego.

3.2.2 Analiza złożoności

Złożoność obliczeniowa Do wyznaczenia $\mathbf{y}$ z układu $\mathbf{Ly} = \mathbf{b}$ potrzebujemy n iteracji, z których w każdej wykonamy co najwyżej $l+1$ modyfikacji. Przy założeniu, że l jest stałą daje to $O(n)$. Rozwiązanie $\mathbf{Ux} = \mathbf{y}$ wykonuje się w czasie $O(n)$ co zostało pokazane w analizie złożoności w zadaniu pierwszym. Całościowo więc złożoność obliczeniowa rozwiązywana układu równań przy znanym rozkładzie **LU** to $O(n)$.

Złożoność pamięciowa Do rozwiązania układu równań $\mathbf{Ax} = \mathbf{b}$ przy znanym rozkładzie **LU** nie jest potrzebna dodatkowa pamięć. Wszystkie operacje na wektorze prawych stron wykonujemy `in situ`.

3.2.3 Kwestie implementacyjne

W języku programowania `Julia` zostały zaimplementowane funkcje `solve_linear_system` dla wariantu bez wyboru elementu głównego oraz dla wariantu z częściowym wyborem elementu głównego. Funkcje różnią się jedynie sygnaturą. Funkcja dla częściowego wyboru elementu głównego przyjmuje dodatkowo permutację. W module `blocksys` została umieszczona funkcja służąca do testowania dwuetapowego `solve_linear_system!`. Jest to funkcja użytkowa operująca na zaimplementowanych już funkcjach `lufactorization!` oraz `solve_linear_system`.

3.3 Wyniki i interpretacja

Dla zaimplementowanych funkcji zostały stworzone programy testujące, które jako dane wejściowe przyjmowały dane wygenerowane przez funkcję `blockmat` z modułu `matrixgen`.

	Bez wyboru elementu głównego		Z wyborem elementu głównego	
n	czas (s)	δ (błąd względny)	czas (s)	δ (błąd względny)
16	0.339209	9.836601059614785e-16	0.000153	6.787359152847496e-16
10 000	0.183928	1.0083140309679606e-14	0.209167	5.401323443968982e-16
50 000	6.252203	1.9071532832302597e-13	6.411412	5.248824020867468e-16

Wyniki działania funkcji `solve_linear_system!` są identyczne jak odpowiadające im wyniki metody eliminacji Gaussa z pierwszego zadania. Wynika to z tego, że są to algorytmy sobie równoważne.