Sprawozdanie Obliczenia Naukowe
lista 5

Łukasz Bratos

styczeń 2020

1 Opis problemu

Naszym problemem jest rozwiązanie układu równań liniowych

$$Ax = b$$

dla macierzy $A \in R^{n\times n}$ i wektora prawych stron $b \in R^n$, $n \geqslant 4$. A jest macierzą rzadką i blokową o następującej strukturze:

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

gdzie $v = m/\ell$, zakładając, że n jest podzielne przez ℓ, gdzie $\ell \geqslant 2$ jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych (bloków): A_k, B_k i C_k, gdzie $A_k \in R^{\ell\times\ell}$, $k = 1, \ldots, v$ są macierzami gęstymi, $B_k \in R^{\ell\times\ell}$, $k = 2, \ldots, v$ są postaci:

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

a $C_k \in R^{\ell\times\ell}$, $k = 1, \ldots, v-1$ są macierzami diagonalnymi:

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

Naszym zadaniem jest zaimplementować metodę eliminacji Gaussa (w dwóch wariantach: bez wyboru elementu głównego oraz z częściowym wyborem elementu głównego) oraz funkcję rozwiązującą układ równań $Ax = b$ przy pomocy tej metody; funkcję wyznaczającą rozkład LU macierzy A metodą eliminacji Gaussa (również w dwóch powyższych wariantach) oraz funkcję rozwiązującą układ równań $Ax = b$ na podstawie rozkładu LU. Oczywiście na uwadze mamy mieć specyficzną strukturę macierzy i zoptymalizować nasze funkcje pod tym kątem. Aby efektywnie przechowywać macierz rzadką skorzystamy z biblioteki podstawowej języku `Julia` *Sparse Arrays*. Pamięta ona tylko niezerowe elementy naszej macierzy. Możemy założyć, że dostęp do elementu macierzy mamy wtedy w czasie stałym (powszechnie wiadomo jednak, że tak nie jest).

2 Opis algorytmów

2.1 Eliminacja Gaussa

2.1.1 Bez wyboru elementu głównego

Metoda eliminacji Gaussa to metoda sprowadzenia układu równań do równoważnego układu z macierzą trójkątną górną z której łatwo już wyznaczyć rozwiązanie metodą podstawiania wstecz. Przebieg algorytmu polega na zerowaniu elementów leżących pod przekątną. Przykładowo, aby wyzerować element a_{i1} to od i-tego wiersza odejmujemy pierwszy wiersz pomnożony przez *mnożnik* w postaci a_{i1}/a_{11}. Tym sposobem w pierwszym kroku zostają wyzerowane wszystkie elementy poniżej pierwszego wiersza w pierwszej kolumnie. Następnie jest to powtarzane na drugiej kolumnie itd. Należy pamiętać, aby podczas modyfikowania i-tego wiersza macierzy równolegle modyfikować też i-ty element wektora prawych stron, aby układy pozostały tożsame.

Metoda podstawiania wstecz polega na zastosowania poniższego wzoru:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}}$$

na kolejnych i-tych wierszach, rozpoczynając od ostatniego.

Metoda ta w swojej klasycznej wersji ma złożoność $\mathcal{O}(n^3)$, natomiast podstawianie wstecz - $\mathcal{O}(n^2)$. Stąd też wniosek, że aby rozwiązać układ równań tą metodą należy wykonać $\mathcal{O}(n^3)$ operacji.

Minusem tej metody jest to, że nie działa ona gdy na przekątnej znajduje się zerowy element (obliczając nasz mnożnik dzielilibyśmy przez zero) lub element bliski zeru (może wprowadzać błędy numeryczne). Rozwiązaniem tego problemu jest rozszerzenie algorytmu o częściowy wybór elementu głównego.

2.1.2 Z częściowym wyborem elementu głównego

Ten wariant metody eliminacji Gaussa polega na odpowiednim spermutowaniu kolejność wierszy macierzy, tak, aby na przekątnej znajdowały się niezerowe elementy oraz najlepiej stosunkowo duże. W tym wariancie wybór odpowiedniego wiersza polega na znalezieniu największej wartości w kolumnie spośród wierszy poniżej przekątnej.

Permutacje takie zapamiętujemy w dodatkowym wektorze permutacji, zamiast modyfikować macierz (co mogłoby być bardziej kosztowne dla macierzy o dużych rozmiarach). Zmiany w kodzie polegają na tym, że zamiast odnosić się bezpośrednio do wiersza, odnosimy się do odpowiedniej pozycji w wektorze permutacji.

2.1.3 Optymalizacje

Optymalizacja powyższych algorytmów uwzględniająca specjalną strukturę naszej macierzy polega na zawężeniu przedziałów iteracji pętli. Algorytm polega głównie na zerowaniu elementów poniżej przekątnej. Stąd też przy naszej macierzy nie ma sensu iterować przez całą kolumnę (zera nic nie zmieniają). Wystarczy więc, że w drugiej pętli będziemy iterowali od $k+1$ do $min(n, k+l+1)$. Podobnie w trzeciej pętli iterację możemy zawęzić do zakresu od $k+1$ do $min(n, k+l)$. Analogicznie przy wyznaczeniu wektora rozwiązań przedział można zawęzić w ten sam sposób.

Algorithm 1 Zoptymalizowny algorytm Gaussa z wyliczanim wektora wyników

1: **procedure** GAUSS(A, n, l, b)
2: **for** $k = 1$ *to* $n-1$ **do**
3: **for** $i = k+1$ *to* $\texttt{min}(n, k+l+1)$ **do**
4: $multiplier \leftarrow \frac{a_{ik}}{a_{kk}}$
5: $a_{ik} \leftarrow 0$
6: **for** $j = k+1$ *to* $\texttt{min}(n, k+l)$ **do**
7: $a_{ij} \leftarrow a_{i_j} - multiplier \cdot a_{kj}$
8: $b_i \leftarrow b_i - multiplier \cdot b_i$
9: **for** $i = n$ *downto* 1 **do**
10: $x_i \leftarrow \frac{b_i - \sum_{j=i+1}^{min(n,i+l)} a_{ij} x_j}{a_{ii}}$
11: **return** $\overline{x}$

Widzimy, że główna pętla wykonuje się $n-1$ razy, natomiast druga i trzecia najwyżej po l razy, więc jeśli l jest stałą to wyznaczenie macierzy wykonuje się w czasie liniowym. Wyliczenie wektora wyników to również pętla, która wykonuje się n razy, wraz z wewnętrzną pętlą, która wykonuje się l razy. Stąd też złożoność powyższego algorytmu to $\mathcal{O}(n)$.

Algorithm 2 Zoptymalizowany algorytm Gaussa z częściowym wyborem elementu głównego

```
1:  procedure GaussWithChoice(A, n, l, b)
2:      p ← [1..n]
3:      for k = 1 to n − 1 do
4:          wybierz j, t. że a_{p_j k} maksymalne poniżej przekątnej w kolumnie k
5:          p_k ↔ p_j
6:          for i = k + 1 to min(n, k + l + 1) do
7:              multiplier ← a_{p_i k} / a_{p_k k}
8:              a_{p_i k} ← 0
9:              for j = k + 1 to min(n, k + 2 · l) do
10:                 a_{p_i j} ← a_{p_i j} − multiplier · a_{p_k j}
11:             b_{p_i} ← b_{p_i} − multiplier · b_{p_i}
12:     return A
```

W wariancie z częściowym wyborem elementu głównego główna pętla wykonuje się $n-1$ razy, szukanie maksymalnego elementu to pętla wykonująca l iteracji, kolejna pętla również wykonuje się l razy, a ostatnia $2 \cdot l$ razy. Wyznaczanie rozwiązania jest analogiczne do wariantu bez wyboru elementu głównego, więc złożoność całego algorytmu to $\mathcal{O}(n)$.

2.2 Rozkład LU

2.2.1 Bez wyboru elementu głównego

Rozkładem LU macierzy A nazywamy przedstawienie macierzy A w postaci iloczynu $A = LU$, gdzie L to macierz trójkątna dolna, a U to macierz trójkątna górna. Macierz U jest macierzą A przekształconą do postaci trójkątnej górnej (przy pomocy metody eliminacji Gaussa), a macierz L jest otrzymywana poprzez zapamiętywanie mnożników użytych do przekształceń (mnożnik służący do wyzerowania elementu a_{ij} zostanie zapisany w i-tym wierszu i j-tej kolumnie macierzy L). Dodatkowym wymogiem jest to, żeby macierz L na swojej przekątnej zawierała jedynki.

Zastosowane optymalizacje są analogiczne jak w przypadku metody eliminacji Gaussa. Złożoność obliczeniowa wyznaczania rozkładu LU również jest asymptotycznie taka sama jak metody eliminacji Gaussa, tj. $O(n)$.

Algorithm 3 Rozkład LU

```
1:  procedure LU(A, n, l)
2:      L_{ij} ← 0
3:      U_{ij} ← A_{ij}
4:      for k = 1 to n − 1 do
5:          L_{kk} ← 1
6:          for i = k + 1 to min(n, k + l + 1) do
7:              multiplier ← a_{ik} / a_{kk}
8:              L_{ik} ← multiplier
9:              U_{ik} ← 0
10:             for j = k + 1 to min(n, k + l) do
11:                 U_{ij} ← U_{i_j} − multiplier · U_{kj}
12:     L_{nn} ← 1
13:     return L, U
```

2.2.2 Z częściowym wyborem elementu głównego

Tak samo jak przy klasycznej metodzie eliminacji Gaussa, również przy wyznaczaniu rozkładu LU, możemy się wspomóc częściowym wyborem elementu głównego, aby uniknąć problemu z ewentualnymi

zerami na przekątnej. Dzieje się to analogicznie jak w metodzie eliminacji Gaussa z częściowym wyborem, tj. dokonujemy takich permutacji, aby na przekątnej znajdował się największy element z kolumny (poniżej przekątnej).

2.3 Rozwiązywanie układu równań przy wykorzystaniu rozkładu LU

Gdy znamy już rozkład LU macierzy A to rozwiązanie układu równań $Ax = b$ sprowadza się do rozwiązania dwóch prostych układów równań:

$$Lz = b$$

$$Ux = z$$

które rozwiązujemy stosując podstawienia w przód dla L i podstawienia wstecz dla U.

Algorytm z wariantem z częściowym wyborem elementu głównego wygląda podobnie. Różni się tylko dodaniem wektora permutacji i odnoszenia się do odpowiednich wierszy wskazywanych przez tenże wektor.

Obie pętle zewnętrzne wykonują się maksymalnie n razy, a wewnętrzne najwyżej l razy, stąd złożoność obliczeniowa tego algorytmu to $\mathcal{O}(n)$.

Algorithm 4 Wyliczanie rozwiązania na podstawie rozkładu LU

1: **procedure** SOLVELU(L, U, n, l, b)
2: **for** $i = 1$ *to* $n - 1$ **do**
3: $b_i \leftarrow b_i - \sum_{j=i+1}^{min(n,i+l+1)} L_{ij} \cdot b_j$
4: **for** $i = n$ *downto* 1 **do**
5: $x_i \leftarrow b_i - \sum_{j=i+1}^{min(n,i+l)} U_{ij} \cdot x_j$
6: **return** x

3 Testy

Sprawdzenie poprawności powyższych algorytmów można w prosty sposób sprawdzić. Generując macierz A i wyznaczając na jej podstawie wektor prawych stron b, tak aby zachodziło równanie $Ax = b$, gdzie x jest wektorem jednostkowym. Wtedy rozwiązując układ równań dla A i b wynikiem powinien być wektor jednostkowy.

Przeprowadziłem również testy czasowe i pamięciowe zaimplementowanych przeze mnie metod. Rezultaty umieściłem na poniższych wykresach. Dodatkowo porównałem czas działania i zużycie pamięci funkcji \ z pakietu LinearAlgebra ze zmodyfikowaną wersją metody eliminacji Gaussa.

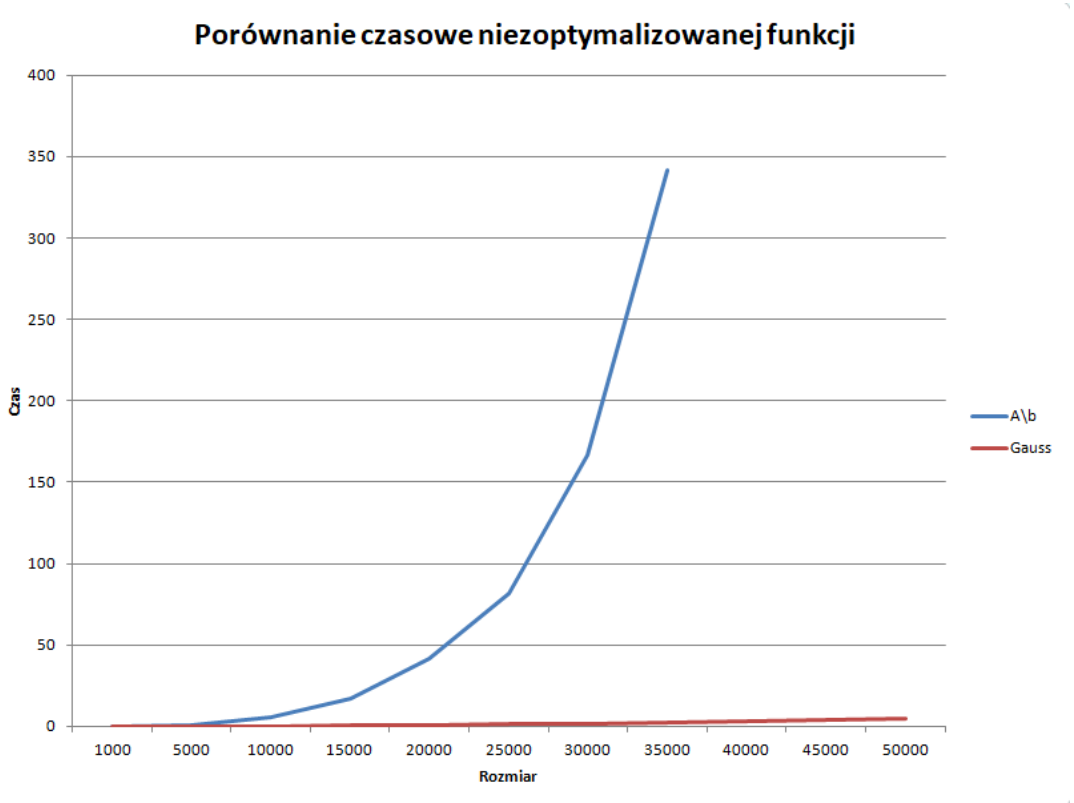
Widzimy jak bardzo różne są asymptotycznie obie funkcje. Zaimplementowane modyfikacje pozwoliły znacząco zmniejszyć zapotrzebowanie na pamięć i czas wykonania. Dodatkowo przy próbie rozwiązania układu równań dla $n = 40000$ przez funkcję \ program wyrzucił błąd i nie rozwiązał układu.

Widzimy też, że wariant z wyborem zabiera trochę więcej pamięci i czasu, od klasycznych wersji, lecz umożliwia rozwiązanie układu równań z zerami na przekątnej. Obliczanie rozkładu LU również wymaga więcej zasobów, ale wygenerowane macierze mogą potem zostać użyte ponownie do rozwiązania układów równań z innym wektorem prawych stron, bez konieczności obliczania ich od nowa.

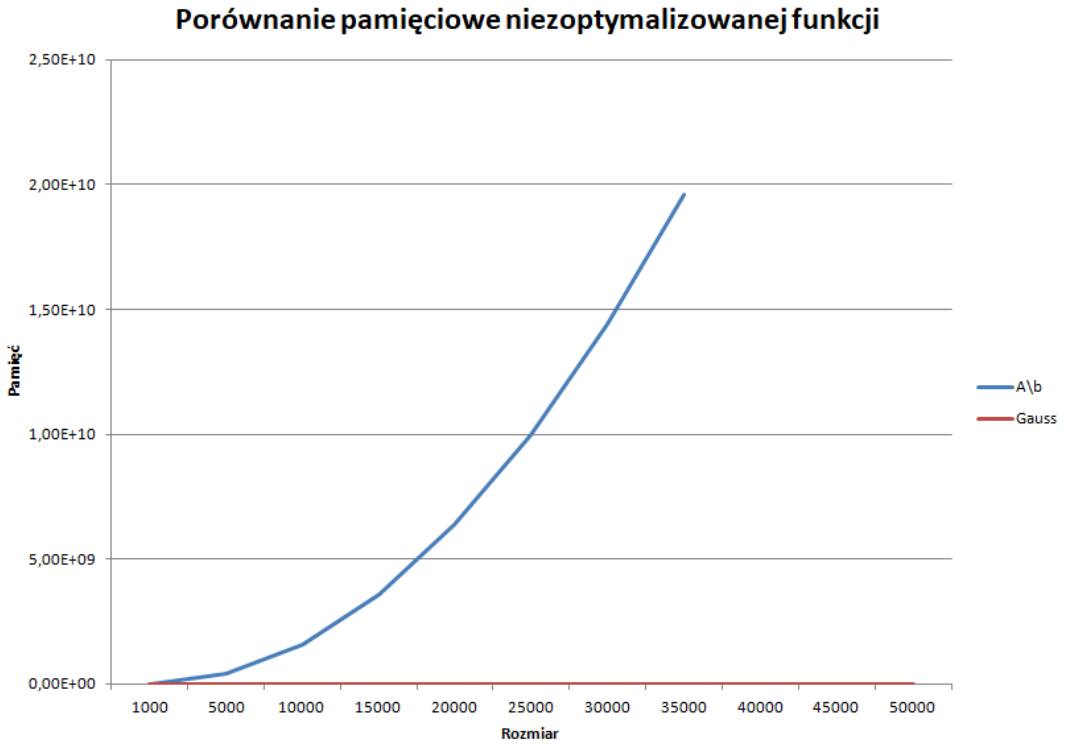
4 Wnioski

Widzimy, że dokładniejsza analiza problemu i modyfikacja znanych algorytmów pod nasz konkretny przykład potrafi przynieść znaczące korzyści jeśli chodzi o złożoność obliczeniową problemu. Udało nam się uzyskać złożoność $\mathcal{O}(n)$, zamiast $\mathcal{O}(n^3)$, co jest różnicą znaczącą. Dodatkowo przechowując tylko niezerowe elementy macierzy oszczędzamy pamięć.

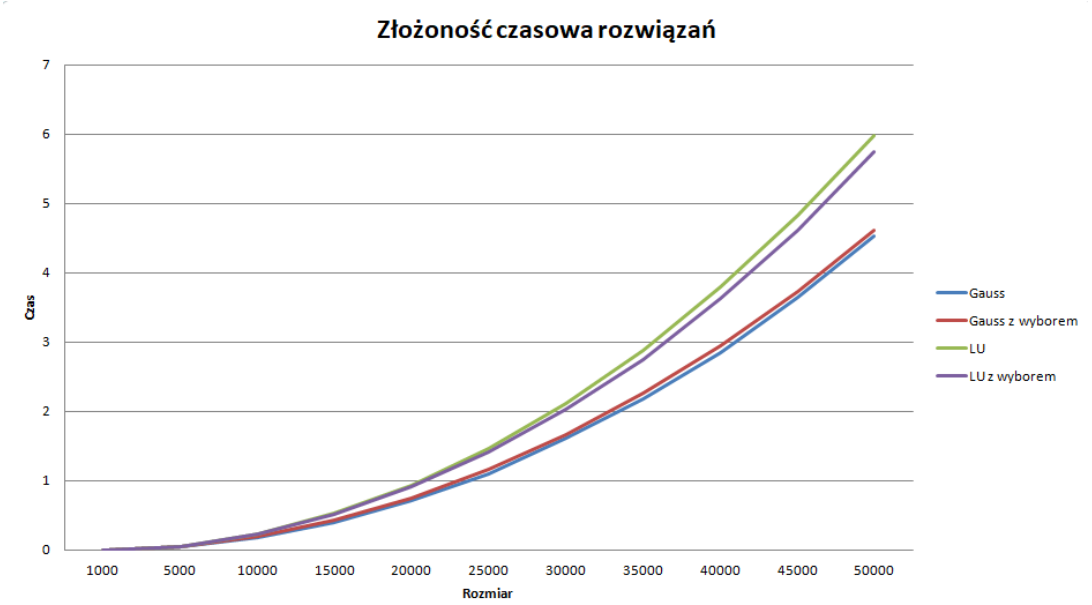
Testy pokazały, że zaimplementowane metody mogą posłużyć do efektywnego rozwiązywania dużych układów równań, gdzie klasyczne metody zawodzą lub wymagają bardzo dużej ilości zasobów.



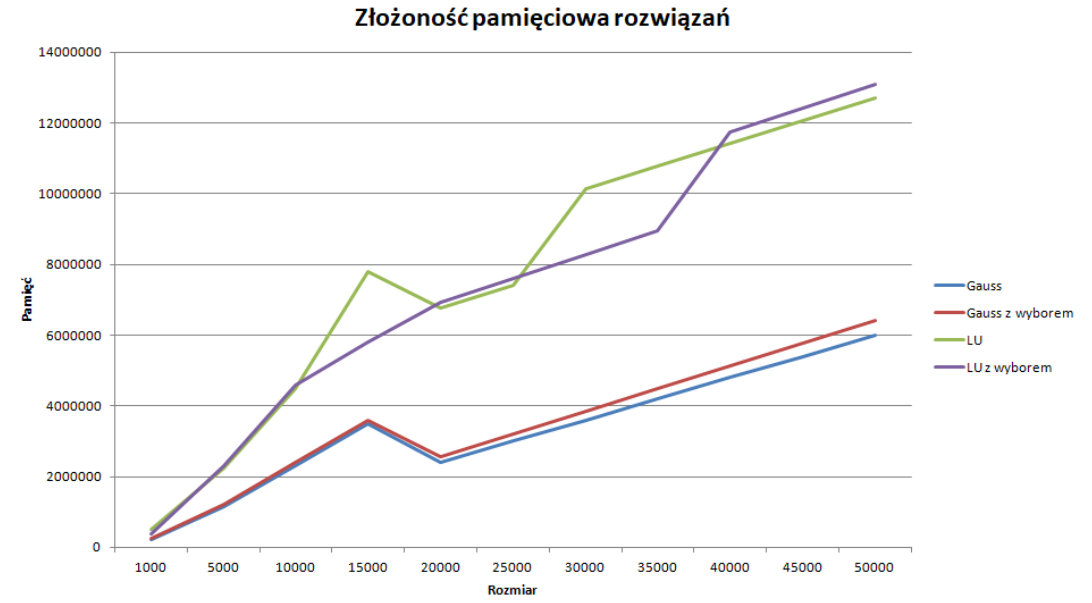
Rysunek 1: Porównanie czasu wykonania zaimplementowanej eliminacji Gaussa z funkcją z pakietu LinearAlgebra



Rysunek 2: Porównanie zużycia pamięci zaimplementowanej eliminacji Gaussa z funkcją z pakietu LinearAlgebra



Rysunek 3: Porównanie czasów wykonania zaimplementowanych algorytmów



Rysunek 4: Porównanie zużycia pamięci zaimplementowanych algorytmów