Obliczenia Naukowe,
Laboratorium, Lista 5

Wiktoria Byra

2021-01-12

1 Problem

Problem polegał na rozwiązaniu układu równań liniowych postaci

$$Ax = b$$

dla danej macierzy współczynników $A \in \mathbb{R}^{n \times n}$ i wektora prawych stron $b \in \mathbb{R}^n$. Macierz A jesr rzadką i blokowa o następującej strukturze:

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

$v = n/l$, zakładając, że n jest podzielne przez l, gdzie l jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych: A_k, B_k, C_k. Mianowicie, $A_k / in \mathbb{R}^{l \times l}, k = 1, \dots, v-1$ jest macierzą gęstą, 0 jest kwadratową macierzą zerową stopnia l, macierz $B_k \in \mathbb{R}^{l \times l}, k = 2, \dots, v$ jest następującej postaci:

$$B_k = \begin{pmatrix} 0 & \dots & 0 & b_1^k \\ 0 & \dots & 0 & b_2^k \\ \vdots & & \vdots & \vdots \\ 0 & \dots & 0 & b_l^k \end{pmatrix}$$

B_k ma tylko jedną, ostatnią kolumnę niezerową. Natomiast $C_k \in \mathbb{R}^{l \times l}, k = 1, \dots, v-1$ jest macierzą diagonalną:

$$C_k = \begin{pmatrix} c_1^k & 0 & 0 & \dots & 0 \\ 0 & c_2^k & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c_{l-1}^k & 0 \\ 0 & \dots & 0 & 0 & c_l^k \end{pmatrix}$$

Duży rozmiar macierzy oznacza duże wymagania czasowe i pamięciowe. Dlatego należało wykorzystać inny sposób reprezentacji macierzy niż tablica kwadratowa, zapamiętując tylko niepuste komórki oraz uwzględniając postać macierzy.

1.1

Napisać funkcję rozwiązującą układ $Ax = b$ metodą eliminacji Gaussa uwzględniającą specyficzną postać macierzy A dla dwóch wariantów:

1. bez wyboru elementu głównego
2. z częściowym wyborem elementu głównego

2 Rozwiązanie

2.1 Reprezentacja macierzy

Do reprezentacji macierzy można skorzystać z wbudowanego pakietu języka Julia - SparseArrays, który zawiera strukturę danych SparseMatrixSCS. Przetrzymuje ona trzy tablice tego samego rozmiaru. Pierwsza zawiera numery kolumn, druga wierszy, a trzecia wartości znajdujące się w komórkach macierzy o danych "współrzędnych".

2.2 Obliczanie prawego wektora

W przypadku kiedy wektor nie jest podany, należy go obliczyć. Obliczenie prawego wektora polega na zsumowaniu elementów w każdym wierszu. Ze względu na specyficzną strukturę macierzy nie trzeba iterować po wszytskich komórkach, wystarczy po niepustych. Wiemy, że w każdej wierszu niepuste są komórki od ostatniej kolumny bloku B do przekątnej bloku C. Pseudokod:

Data: M - macierz, n - rozmiar macierzy, l - rozmiar bloku
Result: vector - prawy wektor b macierzy
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow (max(i-l,1))$ **to** $(min(i+l,n))$ **do**
    vector[i] += M[i, j]
  end
end

2.3 Metoda eliminacji Gaussa

Metoda eliminacji Gaussa polega na sprowadzeniu macierzy do postaci macierzy trójkątnej, czyli wyzerowaniu wszystkich elementów pod jej przekątną. Aby to uzyskać możemy zamieniać wiersze między sobą, dodawać je do siebie, oraz mnożyć przez liczbę różną od zera.
W programie zerujemy kolejno wiersze pod przekątną w każdej kolumnie poprzez odjęcie od każdego elementu odpowiedniej krotności elementu znajdującego się na przekątnej w danej kolumnie, równocześnie analogicznie modyfikując prawy wektor b. Algorytm eliminacji Gaussa ma złożoność czasową $O(n^3)$.

Ponieważ operujemy na macierzy o specyficznej strukturze nie trzeba zerować wszystkich elementów pod przekątną, większość z nich już jest równa zero. Zauważmy, że w dowolnej kolumnie będziemy mieć do wyzerowania dolny macierz dolnotrójkątną bloku A ($\frac{l}{2}$ elementów) lub (jeżeli będziemy w kolumnie będącej krotnością rozmiaru bloku) wszystkie niepuste elementy bloku B (l elementów). Założenie, że rozmiar bloku l jest stały zmniejsza złożoność czasową algorytmu do $O(n)$. Pseudokod:

Data: M - macierz, b - prawy wektor, n - rozmiar macierzy, l - rozmiar bloku

Result: M - macierz górnotrójkątna, b - prawy wektor po eliminacji Gaussa

```
for k ← 1 to n − 1 do
    for i ← k + 1 to min(n, k + l) do
        z = M[i, k] / M[k, k]
        M[i, k] = 0.0
        for j ← k + 1 to min(n, k + l) do
            M[i, j] -= z * M[k, j]
        end
        b[i] -= z * b[k]
    end
end
```

2.4 Rozwiązanie układu równań po eliminacji Gaussa

W celu rozwiązania układu równań po przekształceniu go przez liminację Gaussa należy skorzystać z algorytmu podstawiania wstecz. Wektor wynikowy jest postaci $x_1, x_2, \ldots, x_n$ gdzie $X_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}}{a_{ii}}$. Zauważmy, ze znowu nie musimy brać pod uwagę wszystkich komórek, tylko bloki w obrębie przekątnej.

2.5 Metoda eliminacji Gaussa z częściowym wyborem elementu głównego

Na diagonali macierzy mogą wystąpić zera, wtedy zwykła metoda eliminacji Gaussa nie zadziała i trzeba zmodyfikować ją o częściowy wybór elementu głównego. Częściowy wybór polega na znalezieniu w kolumnie największego (z dokładnością do wartości bezwzględnej) elementu, przestawieniu wierszy macierzy tak, żeby znalazł się on na diagonali i zapisaniu wektora wykonanych permutacji. Znowu możemy ograniczyć rozpatrywane komórki ze względu na specyfikę macierzy, co również w tym przypadku sprowadza złożoność czasową algorytmu do $O(n)$. Pseudokod:

Data: M - macierz, b - prawy wektor, n - rozmiar macierzy, l - rozmiar bloku

Result: M - macierz górnotrójkątna, b - prawy wektor po eliminacji Gaussa, pivots - wektor permutacji wykonanych przy wyborze elementu głównego

pivots - na początku wektor dł. n uzupełniony kolejnymi liczbami naturalnymi, czyli numerami nieprzestawionych wierszy

```
for k ← 1 to n − 1 do
    lastColumn = 0
    lastRow = 0
    for i ← k to min(n, k + l) do
        if abs(M[pivots[i], k]) > lastColumn then
            lastColumn = abs(M[pivots[i], k])
            lastRow = i
        end
    end
    pivots[lastRow], pivots[k] = pivots[k], pivots[lastRow]
    for i ← k + 1 to min(n, k + l) do
        z = M[pivots[i], k] / M[pivots[k], k]
        M[pivots[i], k] = 0
        for j ← k + 1 to min(n, k + 2 ∗ l) do
            M[pivots[i], j] = M[pivots[i], j] - z * M[pivots[k], j]
        end
        b[pivots[i]] = b[pivots[i]] - z * b[pivots[k]]
    end
end
```

3 Testy i wyniki

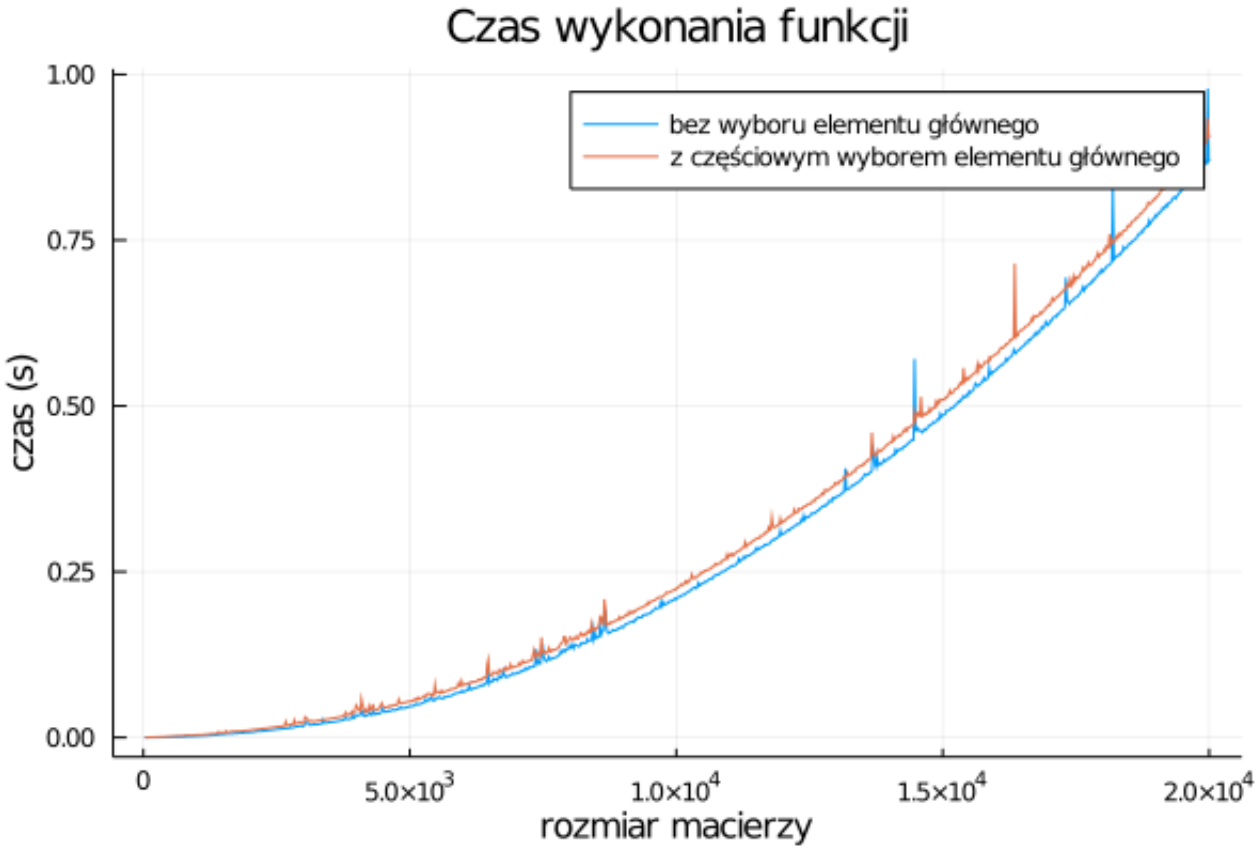
3.1 Błąd odchylenia wektora x

Rozmiar macierzy	Metoda eliminacji Gaussa (z wygenerowanym wektorem b)	Metoda eliminacji Gaussa (z wektorem b z pliku)	Metoda eliminacji Gaussa z częściowym wyborem elementu głównego (z wygenerowanym wektorem b)	Metoda eliminacji Gaussa z częściowym wyborem elementu głównego (z wektorem b z pliku)
16	5.102800490722269e-16	5.102800490722269e-16	2.8844440295753465e-16	2.8844440295753465e-16
10000	8.252489306370072e-15	8.252489306370072e-15	4.2158809864856956e-16	4.2158809864856956e-16
50000	1.8398915733251618e-13	1.8398915733251618e-13	3.9285991202114373e-16	3.9285991202114373e-16

Dla metody eliminacji Gaussa błąd odchylenia obliczonego wektora x wzrasta z rozmiarem macierzy, ale dla przetestowanych danych cały czas jest akceptowalny (rzędu $10^{-16} - 10^{-13}$), dla metody z częściowym wyborem elementu głównego błąd ma stały rząd 10^{-16}.

3.2 Czas wykonania

Rozmiar macierzy	Metoda eliminacji Gaussa (z wygenerowanym wektorem b)	Metoda eliminacji Gaussa (z wektorem b z pliku)	Metoda eliminacji Gaussa z częściowym wyborem elementu głównego (z wygenerowanym wektorem b)	Metoda eliminacji Gaussa z częściowym wyborem elementu głównego (z wektorem b z pliku)	$x = A/b$
16	0.000024 s	0.000024 s	0.000048 s	0.000043 s	0.000022 s
10000	0.212021 s	0.210651 s	0.226567 s	0.228981 s	13.706715 s
50000	6.230536 s	6.791345 s	6.180658 s	6.569049 s	OutOfMemoryError

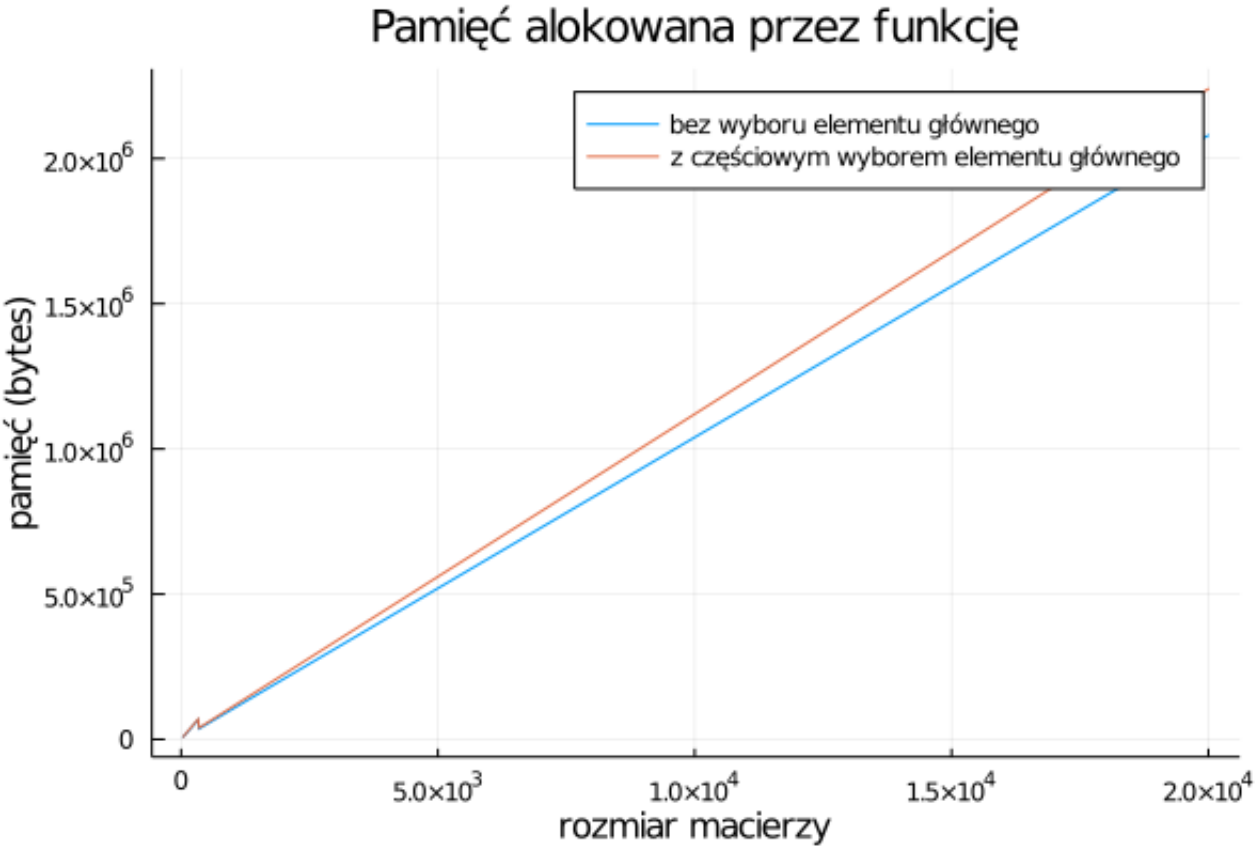


Ryc. 1: wykres wykonany dla danych wygenerowanych przez funkcję blockmat

Przy małym rozmiarze macierzy czas wykonania zoptymalizowanego algorytmu nie wypada lepiej niż standardowe podejście $x = A/b$. Jednak wraz ze wzrostem rozmiaru macierzy pojawia się bardzo duża różnica w czasie, zoptypalizowany algorytm wypada znacznie lepiej.
Dane, dla których został stworzony wykres zostały wygenerowane dla stałego rozmiaru bloku $l = 4$. Pomimo to czas wykonania nie rośnie liniowo. Wynika to nie ze złożoności samego algorytmu (która jest liniowa), ale z czasu dostępu do poszczególnych wartości w *SparseArrays*.

3.3 Zajmowana pamięć

Rozmiar macierzy	Metoda eliminacji Gaussa (z wygenerowanym wektorem b)	Metoda eliminacji Gaussa (z wektorem b z pliku)	Metoda eliminacji Gaussa z częściowym wyborem elementu głównego (z wygenerowanym wektorem b)	Metoda eliminacji Gaussa z częściowym wyborem elementu głównego (z wektorem b z pliku)	$x = A/b$
16	3.109 KiB	3.109 KiB	3.312 KiB	3.312 KiB	2.531 KiB
10000	1015.578 KiB	1015.578 KiB	1.068 MiB	1.068 MiB	763.092 MiB
50000	4.959 MiB	4.959 MiB	5.341 MiB	5.341 MiB	OutOfMemoryError



Ryc. 1: wykres wykonany dla danych wygenerowanych przez funkcję blockmat

Pamięć alokowana przez funkcje wzrasta podobnie do czasu - przy małych rozmiarach macierzy zoptymalizowany algorytm wypada nawet nieznacznie gorzej niż standardowe podejście, jednak ze wzrostem rozmiaru macierzy pojawia się duża różnica i zoptymalizowany algorytm zużywa znacznie mniej pamięci.
Pamięć alokowana przez funkcję wedlug wykresu, zgodnie z oczekiwaniami rośnie liniowo.

4 Wnioski

Porównując wyniki testów zoptymalizowanych funkcji ze standartowym podejściem $x = A/b$ widzimy, że algorytm działa w krótszym czasie i zużywa mniej pamięci, co jest szczególnie ważne przy dużej ilości danych. Możemy z tego wywnioskować, że bardzo ważne jest przeanalizowanie danych przez wyborem sposobu rozwiązania i dostosowanie odpowiedniej metody do danych warunków.