Politechnika Wrocławska
Wydział Podstawowych Problemów Techniki

# Obliczenia Naukowe
# Sprawozdanie z Laboratorium

## Lista 5

Autor:
Krzysztof Nowak
III Rok INF.
Nr Indeksu: 229807

16 stycznia 2018

# 1 Zadanie 1

## 1.1 Opis problemu

Naszym zadaniem było napisanie funkcji rozwiązującej układ $Ax = b$ metodą eliminacji Gaussa uwzględniając specyficzną postać macierzy A dla dwóch wariantów:
- bez wyboru elementu głównego
- z częściowym wyborem elementu głównego

Macierz $A \in \mathbb{R}^{n \times n}$ ma następującą strukturę:

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

Gdzie $v = n/l$ przy założeniu, że $n$ jest podzielne przez $l$, dla $l \geq 2$, które to jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych: $A_k, B_k, C_k \in \mathbb{R}^{l \times l}$ o następujących postaciach:

$$A_k = \begin{pmatrix} a^k_{1\,1} & a^k_{1\,2} & \cdots & a^k_{1\,l-1} & a^k_{1\,l} \\ a^k_{2\,1} & a^k_{2\,2} & \cdots & a^k_{2\,l-1} & a^k_{2\,l} \\ \vdots & & \vdots & \vdots & \vdots \\ a^k_{l\,1} & a^k_{l\,2} & \cdots & a^k_{l\,l-1} & a^k_{l\,l} \end{pmatrix}$$

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

$$C_k = \begin{pmatrix} c^k_{1\,1} & 0 & 0 & \cdots & 0 \\ 0 & c^k_{2\,2} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c^k_{l-1\,l-1} & 0 \\ 0 & \cdots & 0 & 0 & c^k_{l\,l} \end{pmatrix}$$

Natomiast $b \in \mathbb{R}^n$ jest wektorem prawych stron.

## 1.2 Opis rozwiązania

### 1.2.1 Przechowywanie macierzy oraz jej wczytywanie

Macierz A jest przechowywana w pamięci za pomocą SparseMatrixCSC - specjalnego typu stworzonego do tego typu problemów, dostępnego standardowo w języku programowania Julia. Macierz ta przechowuje wyłącznie elementy niezerowe, co efektywnie zmniejsza zapotrzebowanie na pamięć przy macierzach rzadkich. Dzięki wykorzystaniu tego typu spełniamy założenie zadania o efektywnym przechowywaniu macierzy A.

Macierz A jest wczytywana z pliku tekstowego, gdzie pierwszy wiersz zawiera kolejno:
n - rozmiar macierzy $A$ , l - rozmiar macierzy blokowych $A_k, B_k, C_L$ znajdujących się wewnątrz macierzy $A$
Kolejne wiersze przedstawiają elementy niezerowe macierzy i są one w tej postaci:
i - i-ty wiersz , j - j-ta kolumna, a- wartość znajdująca się na i-tym wierszu i j-tej kolumnie w macierzy A

### 1.2.2 Przechowywanie wektora oraz jego wczytywanie

Wektor b jest przechowywany w pamięci za pomocą Vector - typu dostępnego standardowo w języku programowania Julia. Wczytywany jest on z pliku tekstowego gdzie pierwszy wiersz to liczba elementów w wektorze, a następne wiersze to kolejne wartości wektora b.

### 1.2.3 Metoda eliminacji Gaussa

Załóżmy, że mamy następujący układ równań

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array}$$

możemy go przestawić jako iloraz Ax=b, gdzie A jest macierzą, x jest wektorem niewiadomych, a b jest wektorem prawej strony równania. Chcąc obliczyć to równanie możemy wykorzystać metodę eliminacji Gaussa, w której to z wyżej podanej postaci macierzy, przechodzimy do bardziej przystępnej jeżeli chodzi o liczenie niewiadomych macierzy schodkowej. Metoda ta polega na elimininowaniu zmienniej $x_i$ z równań od $i+1$-go do n-tego. Na początku mnożymy i-te równanie przez $I_{ji} = \frac{a_{ji}}{a_{ii}}$, gdzie $j = i+1, \ldots, n$ a następnie odejmujemy od pozostałych. Na przykład dla pierwszego kroku:
$I_{j1} = \frac{a_{j1}}{a_{11}}$ gdzie $j = 2, \ldots, n$ a po odjeciu otrzymamy

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ & & a'_{22}x_2 & + & \cdots & + & a'_{2n}x_n & = & b'_2 \\ & & \vdots & & & & \vdots & & \vdots \\ & & a'_{n2}x_2 & + & \cdots & + & a'_{nn}x_n & = & b'_n \end{array}$$

Powtarzając kolejno, po $n-1$ krokach otrzymamy układ z macierzą górno trójkątną

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ & & a'_{22}x_2 & + & \cdots & + & a'_{2n}x_n & = & b'_2 \\ & & & \ddots & & & \vdots & & \vdots \\ & & & & & & a^{(n)}_{nn}x_n & = & b^{(n)}_n \end{array}$$

Mając tą postać jesteśmy zdolni obliczyć w prosty sposób nasze niewiadome. Wyznaczany najpierw $x_n = \frac{b^{(n)}_n}{a^{(n)}_{nn}}$, a potem dalej dla $x_k$, gdzie $k = n-1, \ldots, 1$

$$x_k = \frac{b^{(k)}_k - \sum_{j=k+1}^{n} a^{(k)}_{kj} x_j}{a^{(k)}_{kk}}$$

W ramach tej metody możemy wybierać jeszcze w każdym k-tym kroku element główny to jest $|A_{pk}| = max|A_{ik}|$ dla $k \leq i \leq n$, a następnie przestawiamy p-ty wiersz z k-tym w macierzy A,

oraz p-ty element z k-tym wektora b. Ma to na celu umożliwienie wykonania tej metody na niektórych macierzach nieosobliwych, oraz zapewnia z numerycznego punktu widzenia mniejszy błąd przy obliczeniach, dla zadanej dokładności. Metoda ta ma złożoność obliczeniową na poziomie $O(n^3)$.
Pseudokod tej metody prezentuje się następująco:

**Algorithm 1:** gauss

**Data:** $A$,$b$,$n$, gdzie
$A$ - Macierz określająca współczynniki przy równaniach ,
$b$ - Wektor prawej strony
$n$ - Rozmiar macierzy i wektora
**Result:** $x$, gdzie
$x$ - Wektor rozwiązań równania

```
1  begin
2      for k = 1 to n − 1 do
3          index ⟵ p dla którego |A[p][k]| = max|A[i][k]| dla k ≤ i ≤ n /* Wybór
               elementu głównego                                          */
4
5          swap(A[k], A[index])
6          swap(b[k], b[index])
7          for i = k + 1 to n do
8              I[i][k] ⟵ A[i][k]/A[k][k])
9              for j = k + 1 to n do
10                 A[i][j] ⟵ A[i][j] − I[i][k] ∗ A[k][j]
11             end
12             b[i] ⟵ b[i] − I[i][k] ∗ b[k]
13         end
14     end
15     for i = n down to 1 do
16         x[i] ⟵ b[i]
17         for j = i + 1 to n do
18             x[i] ⟵ x[i] − A[i][j] ∗ x[j]
19         end
20         x[i] ⟵ x[i]/A[i][i]
21     end
22     return x
23 end
```

### 1.2.4 Uwzględnienie specyficznej postaci macierzy

Niestety w naszym przypadku standardowa metoda eliminacji Gaussa się nie sprawdzi, spowodowane jest to tym, że dla dużych rozmiarów macierzy, obliczenie rozwiązania staje się czasochłonne,a każdorazowa pomyłka w danych, zmusza nas do kolejnego długiego oczekiwania na wynik. Jednakże specyficzna postać naszej macierzy umożliwia nam dokonanie pewnych optymalizacji względem standardowej metody. Wiedząc, że każda macierz blokowa ma rozmiar $l \times l$ i w tylko ich obrębie są wartości niezerowe, możemy sprawdzać w 2 i 3 pętli maksymalnie $2 * l$ elementów, zamiast przechodzić każdorazowo $n$ razy, gdzie w znacznej większości przypadków były to mnożenia i odejmowania zer. co zmniejsza już nam złożoność z $O(n^3)$ do $O(4l^2n)$ co przy założeniu, że l jest stałe daje nam $O(n)$ a przy naszych dużych rozmiarach macierzy jest to diametralna optymalizacja i przyśpieszenie obliczeń.

### 1.3 Rozwiązania

Dla każdego zestawu danych tj. dla rozmiarów macierzy 16, 10000 oraz 50000 wynikiem był wektor jedynek o zadanym rozmiarze. Skupmy się natomiast na czasach wykonywania naszej implementacji metody eliminacji Gaussa, oraz dla standardowej metody

| Funkcja | Rozmiar danych | Wybór | Czas (s) |
|---|---|---|---|
| Gauss | 16 | Nie | $8.5283913e-5$ |
| Gauss z uwzględnieniem | 16 | Tak | 0.00010327101 |
| Gauss z uwzględnieniem | 16 | Nie | $4.2476961e-5$ |
| Gauss | 10000 | Nie | $ok.60000$ |
| Gauss z uwzględnieniem | 10000 | Tak | 0.52993301475 |
| Gauss z uwzględnieniem | 10000 | Nie | 0.358175245 |
| Gauss | 50000 | Nie | – |
| Gauss z uwzględnieniem | 50000 | Tak | 17.4584088357 |
| Gauss z uwzględnieniem | 10000 | Nie | 10.2199510277 |

Jak widać, dla większych danych zdecydowanie lepsze czasy uzyskuje Gauss z uwzględnieniem specyficznej postaci macierzy A. Niestety nie mogłem sprawdzić jak prezentują się porównania pod względem wykorzystanej pamięci z powodu braków zasobów, aczkolwiek można wywnioskować, ze normalne przechowywanie zerowych elementów w arytmetyce Float64 dla dużych rozmiarów może zająć całą pamięć RAM komputera.

### 1.4 Wnioski

Jeżeli jest to możliwe, oraz stosunkowo nieskomplikowane do implementacji, to warto wykonywać optymalizację funkcji pod nasze dane. Pomimo tego, że dokonał się wielki postęp technologiczny i nie musimy już tak bardzo dbać o optymalność rozwiązania, to jednak są przypadki, gdzie bez tego, możemy niepotrzebnie tracić mnóstwo czasu, oraz zasobów pamięci.