Sprawozdanie

Obliczenia naukowe 2019/2020 - lista 5

Tomasz Janik

1 Eliminacja Gaussa

1.1 Opis problemu

Należało tutaj napisać funckję rozwiązującą metodą Gaussa układ równań w postaci:

$$\mathbf{Ax} = \mathbf{b}$$

z lub bez wyboru elementu głównego, dla danej macierzy współczynników $\mathbf{A} \in \mathbb{R}^{n \times n}$ i wektora prawych stron $\mathbf{b} \in \mathbb{R}^n$, n $\geqslant$ 4. Macierz $\mathbf{A}$ jest macierzą rzadką i blokową o następującej strukturze:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{C}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{B}_2 & \mathbf{A}_2 & \mathbf{C}_2 & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_3 & \mathbf{A}_3 & \mathbf{C}_3 & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{B}_{v-2} & \mathbf{A}_{v-2} & \mathbf{C}_{v-2} & \mathbf{0} \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{B}_{v-1} & \mathbf{A}_{v-1} & \mathbf{C}_{v-1} \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_v & \mathbf{A}_v \end{pmatrix},$$

gdzie $v = \frac{n}{l}$, zakładając, że n jest podzielne przez l, gdzie $l \geqslant 2$ jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych (bloków): $\mathbf{A}_k, \mathbf{B}_k, \mathbf{C}_k$. Mianowicie, $\mathbf{A}_k \in \mathbb{R}^{l \times l}$, $k = 1, \dots, v$ jest macierzą gęstą, $\mathbf{0}$ jest kwadratową macierzą zerową stopnia l, macierz $\mathbf{B}_k \in \mathbb{R}^{l \times l}$, $k = 2, \dots, v$ jest następującej postaci:

$$\mathbf{B}_k = \begin{pmatrix} 0 & \dots & 0 & b^k_{1l-1} & b^k_{1l} \\ 0 & \dots & 0 & b^k_{2l-1} & b^k_{2l} \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & b^k_{ll-1} & b^k_{ll} \end{pmatrix},$$

$\mathbf{B}_k$ ma tylko dwie ostatnie kolumny niezerowe. Natomiast macierz $\mathbf{C}_k \in \mathbb{R}^{l \times l}$, $k = 1, \dots, v-1$ jest macierzą diagonalną:

$$\mathbf{C}_k = \begin{pmatrix} c^k_1 & 0 & 0 & \dots & 0 \\ 0 & c^k_2 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c^k_{l-1} & 0 \\ 0 & \dots & 0 & 0 & c^k_l \end{pmatrix}.$$

1.2 Metoda eliminacji Gaussa

Eliminacja Gaussa polega na sprowadzeniu wejściowego układu równań do macierzy schodkowej górnej. Uzyskuje się ją za pomocą operacji na wierszach - dodawania, odejmowania oraz mnożenia prez liczbę różną od 0. Możemny też w razie potrzeby dowolnie zamieniać miejscami wiersze i kolumny miejscami.

Jeśli mamy układ n równań, to w pierwszym kroku musimy wyeliminować zmienną x_1 z $n-1$ równań. W tym celu musimy od i-tego wiersza, $i = 2, 3, ..., n$, odjąć odpowiednią krotność wiersza

pierwszego. Odpowiednie mnożniki l_i wyliczamy dzieląc współczynnik stojący przy x_1 w i-tym wierszu przez współczynik stojący przy x_1 w rozważanym obecnie, pierwszym wierszu. W ten sposób po odjęciu od i-tego wiersza, wiersza pierwszego przemnożonego przez mnożnik l_i wyzerujemy w każdym z nich współcznynnik stojący przy x_1. Jednocześnie musimy też modyfikować wartości w wektorze prawych stron w analogiczny sposób.

W następnych krokach eliminujemy współcznynik sotojący przy x_2 z $n-2$ równań, x_3 z $n-3$ równań, i tak dalej. Warto zauważyć, że aby wyliczyć mnożniki dzielimy przez wartości leżące na przekątnej macierzy. Z uwagi na to, kiedy napotkamy 0, powyższy sposób nie zadziała. Po $n-1$ krokach w ostatnim wierzu otrzymamy równanie jednej zmiennej, na podstawie którego będziemy mogli obliczyć pozostałe.

Złożoność obliczeniowa przedstawionego powyżej algorytmu wynosi $\mathcal{O}(n^3)$.

1.3 Eliminacja Gaussa z częściowym wyborem elementu głównego

Jest to zmodyfikowany algorytm Gaussa, dzięki której poradzimy sobie z 0 występującymi na przekątnej macierzy. W realizacji numerycznej ważne jest, żeby te elementy nie tylko były różne od 0, ale też by nie były zbyt małe co do wartości bezwzględnej, ze względu na ograniczoną precyzję. W tym celu szukamy tak zwanego **elementu głównego** w kolumnie.

W k-tym krou eliminacji Gaussa szukamy, elemtu takiego, że

$$|a_{pk}^{(k)}| = max\{|a_{ik}^{(k)}| : i = k, \dots, n\}$$

i przestawiamy w macierzy $\mathbf{A^{(k)}}$ wiersz p-ty z k-tym oraz odpowiadajce im elementy w wektorze prawych stron $\mathbf{b^{(k)}}$.

1.4 Dostosowanie algorytmu do zadanej macierzy rzadkiej

Ze względu na to, że mamy do czynienia z macierzą, która posiada dużo elemntów zerowych oraz znamy jej strukturę, możemy dostosować do niej nasz algorytm, aby był bardziej efektywny. Po pierwsze, aby uniknąc pamiętania dużej ilości zer, posłużymy się w języku Julia strukturą $SparseMatrixSCS$ z pakietu $SparseArrays$, która pamięta jedynie indeksy oraz wartości komórek niezerowych. Wiemy, że macierz $\mathbf{A}$ składa się z podmacierzy rozmiaru $l \times l$ oraz $l \mid n$. Przyjrzyjmy się zatem przykładowej macierzy, kiedy $l = 5$:

$$\begin{pmatrix}
a_{11}^1 & a_{12}^1 & a_{13}^1 & a_{14}^1 & a_{15}^1 & c_{11}^1 & 0 & 0 & 0 & 0 & \dots \\
a_{21}^1 & a_{22}^1 & a_{23}^1 & a_{24}^1 & a_{25}^1 & 0 & c_{22}^1 & 0 & 0 & 0 & \dots \\
a_{31}^1 & a_{32}^1 & a_{33}^1 & a_{34}^1 & a_{35}^1 & 0 & 0 & c_{33}^1 & 0 & 0 & \dots \\
a_{41}^1 & a_{42}^1 & a_{43}^1 & a_{44}^1 & a_{45}^1 & 0 & 0 & 0 & c_{44}^1 & 0 & \dots \\
a_{51}^1 & a_{52}^1 & a_{53}^1 & a_{54}^1 & a_{55}^1 & 0 & 0 & 0 & 0 & c_{55}^1 & \dots \\
0 & 0 & 0 & b_{14}^2 & b_{15}^2 & a_{11}^2 & a_{12}^2 & a_{13}^2 & a_{14}^2 & a_{15}^2 & \dots \\
0 & 0 & 0 & b_{24}^2 & b_{25}^2 & a_{21}^2 & a_{22}^2 & a_{23}^2 & a_{24}^2 & a_{25}^2 & \dots \\
0 & 0 & 0 & b_{34}^2 & b_{35}^2 & a_{31}^2 & a_{32}^2 & a_{33}^2 & a_{34}^2 & a_{35}^2 & \dots \\
0 & 0 & 0 & b_{44}^2 & b_{45}^2 & a_{41}^2 & a_{42}^2 & a_{43}^2 & a_{34}^2 & a_{45}^2 & \dots \\
0 & 0 & 0 & b_{54}^2 & b_{55}^2 & a_{51}^2 & a_{52}^2 & a_{53}^2 & a_{54}^2 & a_{55}^2 & \dots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$

Zauważmy, że ze względu na powyższą postać macierzy $\mathbf{A}$, jesteśmy w stanie policzyć ile wierszy będziemy musieli zmodyfikować w każdym kroku. Tutaj, dla $l = 5$ jest to kolejno $5-1$, $5-2$, $5-3$ współczynników macierzy $\mathbf{A_1}$, a następnie dochodzą nam współczynniki dwóch ostatnich kolumn macierzy $\mathbf{B_2}$, czyli w sumie mamy dalej $5-4+5$, $5-5+5$, a potem zaczynamy od nowa dla macierzy $\mathbf{A_2}$. Otrzymamy zatem ciąg ilości wierszy do modyfikacji w postaci $\{4, 3, 2, 6, 5, 4, 3, 2, 6, ...\}$. Kiedy zamiast 5 wstawimy tu l, otrzymamy ogólną postać ciąg - $\{l-1, l-2, ..., l-(l-2), l-(l-1)+l, l-l+l, l-1, ...\}$, czyli po uproszczeniu $\{l-1, l-2, ..., 2, l+1, l, l-1, ...\}$. k-ty wyraz tego ciągu możemy wyrazić za pomocą wzoru $a_k = l - ((k+1) \mod l) + 1$. Jednocześnie ze względu na to, że macierz C_K jest macierzą diagonalną, to wzdłuż każdego wiersza musimy zmodyfikować tylko l elementów. Teraz możemy zastosować tę wiedzę w naszym algorytmie.

1.4.1 Algorytm standardowej eliminacji Gaussa

```
1: function GAUSS(A, b, n, l)
2:     for k ← 1 to n − 1 do                                          ▷ 1
3:         for i ← k + 1 to k + l − ((k + 1) mod l) + 1 do             ▷ 2
4:             l_ik ← A[i,k] / A[k,k]
5:             A[i, k] ← 0
6:             for j ← k + 1 to MIN(k + l, n) do                      ▷ 3
7:                 A[i, j] ← A[i, j] − l_ik ∗ A[k, j]
8:             end for
9:             b[i] ← b[i] − l_ik · b[k]
10:        end for
11:    end for
12:    x[1..n] ← {0, . . . , 0}
13:    for i ← n downto 1 do                                          ▷ 4
14:        sum ← 0
15:        for j ← i + 1 to MIN(n, i + l) do                          ▷ 5
16:            sum ← sum + A[i, j] ∗ x[j]
17:        end for
18:        x[i] ← (b[i] − sum) / A[i,i]
19:    end for
20:    return x
21: end function
```

Parametry : A - macierz współczynników, b - wektor prawych stron, n - rozmiar macierzy A, l - rozmiar podmacierzy
Wynik : Wektor rozwiązań układu **x**.

W powyższym algorytmie pętla **1** wykona się (n-1) razy, pętla **2** - maksymalnie $l + 1$ razy pętla **3** - l razy, pętla **4** - n razy, a pętla **5** - l razy. Zatem złożoność całego algorytmu wynosi $(n - 1) * (l + 1) * l + n * l$, co przy stałym l, daje zlozoność liniową.

1.4.2 Algorytm eliminacji Gaussa z wyborem elementu głównego

Algorytm eliminacji Gaussa z częściowym wyborem elementu głównego jest bardzo podobny do standardowego, z tym że musimy jeszcze wyszukać maksymalny element oraz zapamiętać, które wiersze zamieniamy miejscami.

```
1: function GAUSS-WITH-CHOOSE(A, b, n, l)
2:     swaps ← {1, ..., n}
3:     for k ← 1 to n − 1 do                                          ▷ 1
4:         max ← |A[k, k]|
5:         maxid ← k
6:         for i ← k + 1 to k + l − ((k + 1) mod l) + 1 do             ▷ 2
7:             if |A[k, swaps[i]]| > max then
8:                 max ← |A[k, swaps[i]]|
9:                 maxid = i
10:            end if
11:        end for
12:        SWAP(swaps[k], swaps[maxid])
13:        for i ← k + 1 to k + l − ((k + 1) mod l) + 1 do             ▷ 3
14:            l_ik ← A[swaps[i],k] / A[swaps[k],k]
15:            A[perm[i], k] ← 0
16:            for j ← k + 1 to MIN(k + 2 ∗ l, n) do                  ▷ 4
17:                A[swaps[i], j] ← A[swaps[i], j] − l_ik ∗ A[swaps[k], j]
18:            end for
19:            b[swaps[i]] ← b[swaps[i]] − l_ik · b[swaps[k]]
20:        end for
21:    end for
22:    x[1..n] ← {0, ..., 0}
23:    for i ← n downto 1 do                                          ▷ 5
24:        sum ← 0
25:        for j ← i + 1 to MIN(n, i + 2 ∗ l + 1) do                  ▷ 6
26:            sum ← sum + A[swaps[i], j] ∗ x[j]
27:        end for
28:        x[i] ← (b[swaps[i]] − sum) / A[swaps[i],i]
29:    end for
30:    return x
31: end function
```

W przypadku tego algorytmu złożoność będzie podobna jak wcześniej, ponieważ mamy tylko jedną dodatkową pętle **2**, która wykona się maksymalnie $l + 1$ razy oraz pętla **4** wykona się $2l$ zamiast l razy, a pętle **5** oraz **6** tak samo jak w algorytmie poprzednim. Zatem tutaj złożoność wyniesie $(n-1) * 2 * (l+1) * 2l + n * l$, która jest większa niż poprzednio, ale dalej jest $\mathcal{O}(n)$.

2 Zadanie 2

2.1 Opis problemu

W tym zadaniu należało napisać funckję wyznaczającą rozkład **LU** macierzy **A** z uwzględnieniem jej specyficznej postaci, w wariancie bez wyboru elementu głównego oraz z jego wyborem.

2.2 Rozkład LU

Jest to rozkład na dwie macierze trójkątne - dolną i górną, które wyznaczane są podczas wykonywania eliminacji Gaussa. Rozkładu tego można użyć do rozwiązania układu równań kolejną metodą. Macierze dolna L oraz górna U, gdzie $A = LU$, mają nastepujące postacie:

$$\mathbf{L} = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix}$$

$$\mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{pmatrix}$$

Wtedy początkowy układ $Ax = LUx = b$, sprowadzamy, do rozwiązania dwóch układów trójkątnych:

$$\mathbf{Ly} = \mathbf{b}$$
$$\mathbf{Ux} = \mathbf{y}$$

Jak już było wspomniane rozkład ten wyznacza się przy użyciu metody eliminacji Gaussa. Macierz U jest macierzą uzyskiwaną podczas eliminacji Gaussa, a macierz L wypełniamy mnoznikami, które zostały użyte do wyzerowania danego elementu. Mnożnik dla A_{ij} zapisujemy w komórce L_{ij}.
Złożoność otrzmania tego rozkładu jest taka jak eliminacji Gauusa, czyli standardowo $\mathcal{O}(n^3)$, natomiast rozwiązanie układu z obliczonych L, U $\mathcal{O}(n^2)$.

2.3 Algorytmy i dostosowanie

Do wyznaczania rozkładu używamy zmodyfikowanych funkcji do eliminacji Gaussa z poprzedniego zadania. Zmiana polega, na tym, że w miejsce zerowanego elementu wstawiamy mnożnik l_{ik}, oraz zapominamy o wektorach x i b. Złożoność będzie więc taka sama, jak wprzypadku eliminacji Gaussa. Warto zaznaczyć, że kiedy mamy do rozwiązania wiele układów z tą samą macierzą współczynników rozkład policzymy tylko raz.

Algorytymy rozwiązujące ograniczają się do obliczenia układów trójkątnych, co wykonywalismy już w prypadku eliminacji Gaussa analogicznie dla wersji z oraz bez wyboru elementu głównego. Przypomnijmy, że samo obliczenie rozwązania po wykonaniu eliminacji miała złożoność $\mathcal{O}(nl)$, zatem te algorytmy również będą miały złożoność liniową.

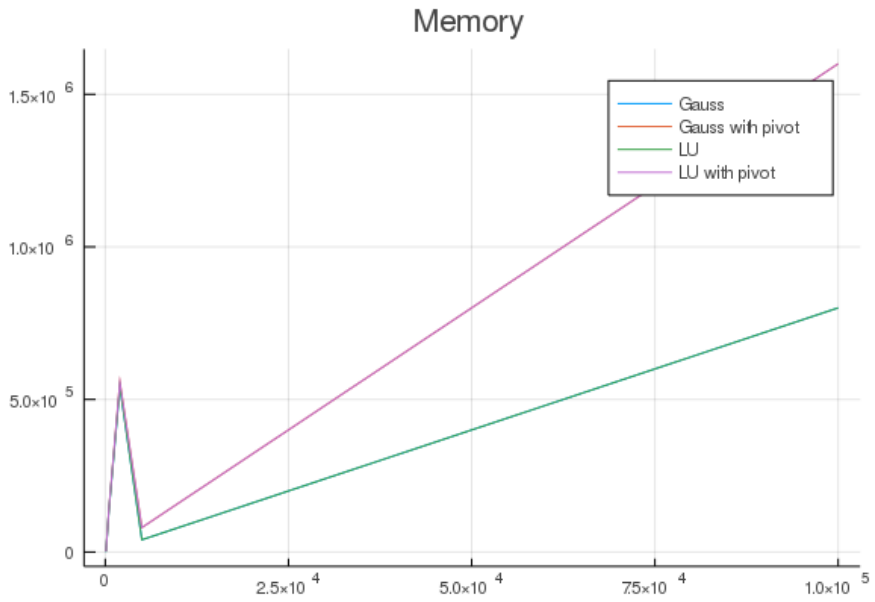
3 Testy

Aby porównać złożoność czasową i pamięciową algorytmów zsotały przeprowadzone testy dla wartości $l = 5$ oraz zwiększających się wartości n. Do zbadania tych rzeczy zostało wykorzystanie makro *@timed* w języku Julia.

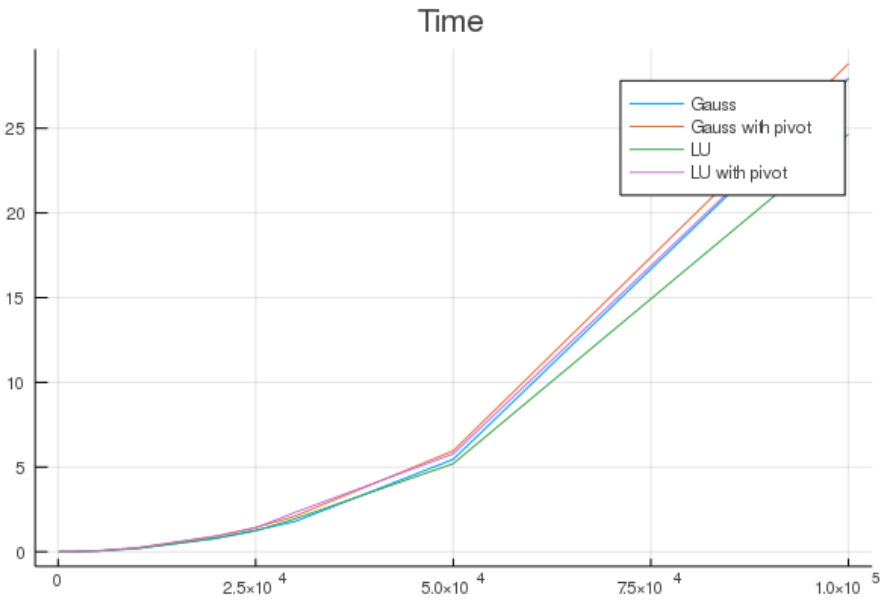
Po analizie wykresów dochodzimy do wniosku, że algorytmy wykorzystujące eliminację Gaussa, są wolniejsze od tych wykorzysujących rozkład LU. Warianty z częściowym wyborem elementu głównego są bardziej czaso-i pamięciochłonne. Widać też, że zużycie pamięci rośnie liniowo wraz ze wzrostem wartości, z małymi wahaniami przy mniejszych warotściach. Czas działania natomiast, mimo iż powinien być liniowy, przypomina wykres funkcji kwadratowej. Wynika to z faktu, że w rzeczywistości dostęp do elementów struktury $SparseMatrix$ nie jest stały.

4 Podsumowanie

Ta lista zdań pokazuje, że dostosowanie algorytmu pod konkretny przypadek ma bardzo duże znaczenie. Tutaj jedynie po przyjrzeniu się strukturze przetwarzanej macierzy, mogliśmy określić, których elementów nie musimy w ogóle przetwarzać, co doprowadziło do zmniejszenia złożoności algorytmu o z $\mathcal{O}(n^3)$ do $\mathcal{O}(n)$.



Rysunek 1: Wykres pamięci



Rysunek 2: Wykres czasu