Obliczenia naukowe
Lista 5

Dominik Kaczmarek, nr albumu 261757

26 stycznia 2023

Spis treści

1 Opis problemu

Tło: *Jednostka badawcza dużej firmy działającej w branży chemicznej prowadzi intensywne badania. Wynikiem tych badań są modele pewnych zjawisk chemii kwantowej. Rozwiązanie tych modeli, w pewnym szczególnym przypadku, sprowadza się do rozwiązania układu równań liniowych*

$$\mathbf{A} \cdot \tilde{\mathbf{x}} = \tilde{\mathbf{b}}$$

Naszymi danymi są tutaj:

- Wektor prawych stron $\mathbf{b} \in \mathbb{R}^n$, gdzie $n \geqslant 4$,
- Macierz współczynników $\mathbf{A} \in \mathbb{R}^{n \times n}$.

Macierz $\mathbf{A}$ jest macierzą **rzadką** oraz **blokową** o następującej strukturze:

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

gdzie $v = n/l$ zakładając, że $l|n$. Macierze wewnętrzne (bloki) $\mathbf{0}$, $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{B}_k \in \mathbb{R}^{l \times l}$, $k = 1, \dots, v$. Macierze $\mathbf{0}$ są macierzami zerowymi, macierz natomiast pozostałe bloki mają następujące postacie:

$$B_k = \begin{bmatrix} b_{11}^k & \dots & b_{1,l-2}^k & b_{1,l-1}^k & b_{1,l}^k \\ 0 & \dots & 0 & 0 & b_{2,l}^k \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & b_{l,l}^k \end{bmatrix}$$

$$C_k = \begin{bmatrix} c_1^k & 0 & 0 & \dots & 0 \\ 0 & c_2^k & 0 & \dots & 0 \\ 0 & 0 & c_3^k & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c_{l-1}^k & 0 \\ 0 & \dots & 0 & \dots & c^k l \end{bmatrix}$$

Macierz $\mathbf{A}$ może być bardza duża (rzędu $\mathbb{R}^{500000\times500000}$), dlatego kluczowe jest opowiednie przechowanie jej w pamięci, co jest pierwszą trudnością zadania. Kolejną kwestią jest opytmalne wyliczenie zadanego układu równań liniowych tj. $\mathbf{A}\cdot\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$. Użycie tutaj klasycznego algorytmu eliminacji `Gaussa`, którego złożoność wynosi $O(n^3)$ w przpadku bardzo dużego n było bardzo czasochłonne. Musimy zatem zmodyfikować podstawowy algorytm Gaussa, mając na względzie specyficzną strukturę macierzy $\mathbf{A}$. Dodatkowo nasz algorytm powinien posiadać dwa warianty:

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

2 Opisy algorytmów

2.1 Trzymanie macierzy w pamięci

Pierwszym problemem, który musi rozwiązać jest sprytne przetrzymywanie macierzy $\mathbf{A}$ w pamięci komputera. Weźmy przykładową macierz $\mathbf{A} \in \mathbb{R}^{12\times12}$ spełniającą zadaną strukturę, gdzie $l = 3$:

	1	2	3	4	5	6	7	8	9	10	11	12
1	a	a	a	c	0	0	0	0	0	0	0	0
2	a	a	a	0	c	0	0	0	0	0	0	0
3	a	a	a	0	0	c	0	0	0	0	0	0
4	b	b	b	a	a	a	c	0	0	0	0	0
5	0	0	b	a	a	a	0	c	0	0	0	0
6	0	0	b	a	a	a	0	0	c	0	0	0
7	0	0	0	b	b	b	a	a	a	c	0	0
8	0	0	0	0	0	b	a	a	a	0	c	0
9	0	0	0	0	0	b	a	a	a	0	0	c
10	0	0	0	0	0	0	b	b	b	a	a	a
11	0	0	0	0	0	0	0	0	b	a	a	a
12	0	0	0	0	0	0	0	0	b	a	a	a

Widzimy, że znaczną część macierzy stanowią zera, więc trzymanie całej macierzy w pamięci byłoby zupełnie zbędne, a złożoność pamięciowa wyniosłaby wtedy $O(n^2)$. W przypadku $n = 12$ nie sytuacja nie wygląda źle, jednak dla bardzo dużych danych stanowiłoby to znaczący problem.

Zależy nam, żeby zapamiętywać tylko istotne dane, czyli komórki znajdujące się w otoczeniu przekątnej macierzy. W tym celu skorzystałem z `Sparse Arrays` zawartych w bibliotekach języka `Julia`, które umożliwiają zapisywanie wybranych komórek macierzy.

Na wstępie pobieram wielkości l, n na podstawie których dodam do `SparseArray` komórki, które na początku wypełnie zerami. Potrzebne komórki dodaje poniższym algorytmem:

Data: $n-$ rozmiar A, $l-$ rozmiary bloków
Result: A - macierz zawierająca potrzebne komórki
for $w \leftarrow 1$ **to** n **do**
  for $k \leftarrow max(1, w-l)$ **to** $min(n, w+l)$ **do**
    $A_{w,k} = 0$;
  end
end
return A;

Algorithm 1: `creatematrix(n,l)`

Złożoność czasowa tego algorytmu wynosi $O(n) * O(2l+1) = O(n) * O(l) = O(l)$. Jeśli traktujemy l jako stałą wtedy otrzymujemy złożoność czasową $O(n)$. Złożoność pamięciowa jest identyczna. Kolejnym krokiem jest pobranie reszty danych z pliku i napisanie odpowiednich komórek w macierzy **A**, co również robimy w czasie $O(n)$. Poniższa tabela przedstawia komórki, które od tej pory trzymamy w pamięci.

	1	2	3	4	5	6	7	8	9	10	11	12
1	a	a	a	c	0	0	0	0	0	0	0	0
2	a	a	a	0	c	0	0	0	0	0	0	0
3	a	a	a	0	0	c	0	0	0	0	0	0
4	b	b	b	a	a	a	c	0	0	0	0	0
5	0	0	b	a	a	a	0	c	0	0	0	0
6	0	0	b	a	a	a	0	0	c	0	0	0
7	0	0	0	b	b	b	a	a	a	c	0	0
8	0	0	0	0	0	b	a	a	a	0	c	0
9	0	0	0	0	0	b	a	a	a	0	0	c
10	0	0	0	0	0	0	b	b	b	a	a	a
11	0	0	0	0	0	0	0	0	b	a	a	a
12	0	0	0	0	0	0	0	0	b	a	a	a

2.2 Eliminacja Gaussa bez wyboru elementu głównego

Chcąc obliczyć $\mathbf{A} \cdot \tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ możemy skorzystać z algorytmu eliminacji Gaussa. Jego działanie polega na odpowiednim wymnażaniu wierszy macierzy oraz wektora b, tak aby wyzerować wszystkie komórki znajdujące się poniżej przekątnej macierzy. W rezultacie otrzymamy macierz trójkątną górną oraz przekształcony wektor b, z których wyliczymy szukany wektor x. Operując na macierzach o strukture nakreślonej w poleceniu nie będziemy zmieniać sposobu działania metody Gaussa - zmienimy tylko zakresy na których będziemy ją wykonywać, ponieważ dużo elementów macierzy jest już równa 0. Naszym celem jest zejście ze złożoności czasowej $O(n^3)$ do $O(n)$.
Rzeczy, której nie unikniemy jest przejście po całej przekątnej macierzy. To daje nam złożoność $O(n)$. Dla każdego elementu na przekątnej musimy wyzerować wszystkie komórki znajdujące się poniżej w kolumnie. Tutaj już nie musimy schodzić do n-tego wiersza. Wystarczy, że wyzerujmy $l-$ komórek poniżej elementu leżacego na przekątnej (O(l)). Ostatni pętla przechodzi po $w+l+1$ elementach w wierszu. Daje nam to w sumie złożoność $O(n) \cdot O(l) \cdot O(l) = O(n)$.

Data: A - macierz, b - wektor prawych ston, $n-$ rozmiar A i b, $l-$ rozmiary bloków
Result: x - szukany wektor
for $i \leftarrow 1$ **to** $n-1$ **do**
  $d \leftarrow A[i,i]$;
  for $w \leftarrow i+1$ **to** $min(n, l+i)$ **do**
    $m \leftarrow A[w,i]/d$ *//mnożnik zerujący A[w,i]*;
    for $k \leftarrow i$ **to** $min(n, i+l+1)$ **do**
      $A[w,k] \leftarrow A[w,k] - m * A[i,k]$*//przemnożenie wiersza przez mnożnik*;
    end
    $b[w] \leftarrow b[w] - m * b[i]$ *//przemnożenie wektora*;
  end
end
$x \leftarrow countx(A, b, n, l)$;
return x;

Algorithm 2: `gauss(A,b,n,l)`

Funkcją *count(A, b, n, l)* zajmiemy się za chwilę. Poniżej przedstawiłem graficznie kilka etapów działania algorytmu:

A	1	2	3	4	5	6	7	8	9	10	11	12		b
1	a	a	a	c	0	0	0	0	0	0	0	0		v
2	0	a*	a*	0	c*	0	0	0	0	0	0	0		v*
3	a	a	a	0	0	c	0	0	0	0	0	0		v
4	b	b	b	a	a	a	c	0	0	0	0	0		v
5	0	0	b	a	a	a	0	c	0	0	0	0		v
6	0	0	b	a	a	a	0	0	c	0	0	0		v
7	0	0	0	b	b	b	a	a	a	c	0	0		v
8	0	0	0	0	0	b	a	a	a	0	c	0		v
9	0	0	0	0	0	b	a	a	a	0	0	c		v
10	0	0	0	0	0	0	b	b	b	a	a	a		v
11	0	0	0	0	0	0	0	0	b	a	a	a		v
12	0	0	0	0	0	0	0	0	b	a	a	a		v

Po wyzerowaniu komórek pod przekątną macierzy A pozostało jeszcze wyzanczyć równania i wektor $\tilde{\mathbf{x}}$. Do tego przyda się funckja *count(A, b, n, l)*. Skorzytamy w niej z rekurencyjnego wzoru na wyznaczenie kolejnych elementów wektora:

$$x_n = \frac{b_n}{A[n,n]}$$

$$x_i = \frac{b_i - A[i,n]x_n - \cdots - A[1,i+1]x_{i+1}}{A[i,i]},\ i = n-1, n-2, \ldots, 1$$

Algorytm wygląda następująco:

A	1	2	3	4	5	6	7	8	9	10	11	12	b
1	a	a	a	c	0	0	0	0	0	0	0	0	v
2	0	a*	a*	0	c*	0	0	0	0	0	0	0	v*
3	0	0	a*	0	0	c*	0	0	0	0	0	0	v*
4	0	0	0	a*	a*	a*	c*	0	0	0	0	0	v*
5	0	0	0	0	a*	a*	0	c*	0	0	0	0	v*
6	0	0	0	0	a*	a*	0	0	c*	0	0	0	v*
7	0	0	0	b	b	b	a	a	a	c	0	0	v
8	0	0	0	0	0	b	a	a	a	0	c	0	v
9	0	0	0	0	0	b	a	a	a	0	0	c	v
10	0	0	0	0	0	0	b	b	b	a	a	a	v
11	0	0	0	0	0	0	0	0	b	a	a	a	v
12	0	0	0	0	0	0	0	0	b	a	a	a	v

Data: A - macierz, b - wektor prawych ston, $n-$ rozmiar A i b, $l-$ rozmiary bloków
Result: x - szukany wektor
$x[n] \leftarrow b[n]/A[n,n]$;
for $w \leftarrow n-1$ **to** 1 **do**
  $x[w] \leftarrow b[w]$;
  for $k \leftarrow w+1$ **to** $min(n, w+l)$ **do**
    $x[w] \leftarrow x[w] - A[w,k] * x[k]$;
  end
  $x[w] \leftarrow x[w]/A[w,w]$
end
return x;

Algorithm 3: `countx(A,b,n,l)`

Zewnętrzna pętla przechodzi po $n-1$ wierszach w macierzy, natomiast wewnętrzna pętla po l kolumnach. Daje nam to złożoność $O(n*l) = O(n)$. Sumując złożoność pierwszego i drugiego algorytmu otrzymujemy złożoność $O(n) + O(n) = O(n)$.

2.3 Eliminacja Gaussa z wyborem pivota

Ulepszeniem algorytmu eliminacji Gaussa jest metoda Crouta, która polega na tym, iż na początku eliminacji wyszukujemy w wierszu macierzy **A** element o największym module, po czym zamieniamy miejscami kolumnę ze znalezionym elementem z kolumną zawierającą element głównej przekątnej. W ten sposób dzielnik będzie posiadał największą na moduł wartość i pozbędziemy się sytuacji, gdy może on posiadać wartość bliską 0.

```
Data: A - macierz, b - wektor prawych ston, n− rozmiar A i b, l− rozmiary bloków
Result: x - szukany wektor
for i ← 1 to n − 1 do
    maxval ← −1;
    maxid ← −1;
    // szukanie największego modułu w kolumnie pod przekątną;
    for w ← i to min(n, l + i) do
        pom ← |A[w, i]|;
        if pom > maxval then
            maxval ← pom;
            maxid ← w;
        end
    end
    // zamiana wierszy w macierzy i wekotrze b;
    if maxid ≠ i then
        for w ← i to min(n, i + 2l + 1) do
            swap(A[i, w], A[maxid, w]);
        end
        swap(b[i], b[max_id]);
    end
    d ← A[i, i];
    // musimy zwiększyć zasięg w wewnętrznej pętli, ze względu na podmienianie wierszy;
    for w ← i + 1 to min(n, l + i) do
        m ← A[w, i]/d;
        for k ← i to min(n, i + 2l) do
            A[w, k] ← A[w, k] − m ∗ A[i, k];
        end
        b[w] ← b[w] − m ∗ b[i];
    end
end
x ← countpivotx(A, b, n, l);
return x;
```

Algorithm 4: `gausspivot(A,b,n,l)`

Zbadajmy złożoność tego algorytmu. Głóna pętla programu iteruje po całej długości macierzy od i równego 1 do $n-1$, zatem jej złożoność wynosi $O(n)$. Pierwsza wewnętrzna pętla odpowiada szukania największego elementu w kolumnie. Ze względu na strukturę macierzy wystarczy że przeszukamy l elementów poniżej przekątnej oraz samą przekątną zatem otrzymujemy $O(l+1)$. Kolejna pętla podmienia wszyskie elementy dwóch wierszy których jest maksymalnie $2l+1$, czyli $O(2l+1)$. Pozostają ostatnie dwie zgnieżdzone w sobie pętle który pojawiły się róznież w pierwszej metodzie eliminacji Gaussa. Tym razem musieliśmy zwiększyć obszar działania w najbardziej zagnieżdżonej pętli *for*, ponieważ wiersz mógł zostać podmieniony przez co liczba ważnych dla nas danych mogła się zwiększyć w stosunku do pierwotnej wersji macierzy. Nie ma to jednak wpływu na złożoność tego bloku i jego złożoność czasowa podobnie jak w pierwszym Gaussie wynosi $O(2l * l) = O(l)$. Przed obliczeniem całościowego kosztu zajimijmy się jescze funkcją *countpivotx(A, b, n, l)*, która nieznacznie różni się od swojej poprzeniej wersji.

Data: A - macierz, b - wektor prawych ston, $n-$ rozmiar A i b, $l-$ rozmiary bloków
Result: x - szukany wektor
$x[n] \leftarrow b[n]/A[n,n]$;
for $w \leftarrow n-1$ **to** 1 **do**
  $x[w] \leftarrow b[w]$;
  // jedyna zmiana w stosunku do *countx* to zwiększony zakres ;
  // iteracji poniższej pętli, z tego samego powodu co *gausspivot* ;
  for $k \leftarrow w+1$ **to** $min(n, w+2*l)$ **do**
    $x[w] \leftarrow x[w] - A[w,k] * x[k]$;
  end
  $x[w] \leftarrow x[w]/A[w,w]$
end
return x;

Algorithm 5: `countxpivot(A,b,n,l)`

Koszt czasowy tego algorytmu jest identyczny jak funckji *countx*, ponieważ $O(n-1) \cdot O(2*l) = O(n)$.

Podsumowując dla *gausspivot* otrzymujemy złożoność:

$$O(n) \cdot (O(l+1) + O(2l+1) + O(2l^2)) + O(n) = O(n) + O(n) = O(n)$$

, ponieważ l jest stałą ($O(l) = O(1)$).

3 Wyniki

Metoda eliminacji Gaussa z wyborem elemetu głównego dużo lepiej radzi sobie z wyliczaniem wektora $\tilde{\mathbf{x}}$, ponieważ błąd względny jest w niektórych przypadkach o 3 rzędy wielkości.

n	Gauss **bez** pivota	Gauss **z** pivotem
16	1.4668500038754537e-15	3.3766115072321297e-16
10000	2.6979457399889902e-14	4.952490752895815e-16
50000	5.983696068447593e-14	5.378168315102863e-16
100000	7.42222985137446e-13	5.199630892522782e-16
300000	6.521306370065582e-14	4.554950254748486e-16
500000	8.329169633245344e-14	4.507270850385053e-16

Tabela 1: Wyniki metody Gaussa bez wyboru elementu głównego i z wyborem elementu głównego dla $n = 16, 10000, 50000, 100000, 300000, 500000$

4 Wnioski

Warto analizować dane przed rozpoczęciem pisania programów. W ten sposób możemy przygotować szybszy algorytm pod specyficzne dane.