Obliczenia naukowe, sprawozdanie z listy 5

Data Paweł 250105

January 9, 2021

1 Polecenie

Zadanie polega na rozwiązaniu równania

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

Gdzie:

- $\mathbf{b} \in \mathbb{R}^n$ jest wektorem prawych stron
- **A** jest macierzą rzadką postaci:

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

  v=n/l, l jest rozmiarem wewnętrznych macierzy

  - $A_k \in \mathbb{R}^{l \times l}$ jest macierzą gęstą
  - $B_k \in \mathbb{R}^{l \times l}$ jest macierzą z jedną, ostatnią, kolumną niezerową:

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

– $C_k \in \mathbb{R}^{l \times l}$ jest macierzą diagonalną:
$$\begin{pmatrix} c_1^k & 0 & 0 & \dots & 0 \\ 0 & c_2^k & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c_{l-1}^k & 0 \\ 0 & \dots & 0 & 0 & c_l^k \end{pmatrix}$$

2 Rozwiązanie

Klasycznym algorytem dla podanego problemu jest metoda eliminacji Gaussa oraz rozkład LU. Dla klasycznej macierzy algorytm ma złożoność obliczeniową równą $O(n^3)$. W tym zadaniu powinno się dostosować algorytm do danej macierzy, by działał w czasie $O(n)$.

2.1 Eliminacja Gaussa

Eliminacja Gaussa pozwala obliczyć x w równaniu $\mathbf{Ax} = \mathbf{b}$. Sprowadzamy macierz A. Używają podstawowych operacji na macierzach (dodawanie i odejmowanie kolumn/wierszy, zamianę kolumn/wierszy) doprowadzamy macierz A do macierzy górno-trójkątnej. Czyli wartości niezerowe są na przekątnej i na "górnym trójkącie", w "dolnym trójkącie" są same zera.

Iterując każdym elemencie na przekątnej, odejmujemy wszystkie wiersze poniżej w taki sposób, by pod daną wartością były same zera.

By zwiększyć dokladność obliczeń, przed odejmowaniem możemy poszukać największej liczby co do wartości bezwzględnej w danej kolumnie pod elementem na przekątnej i zamienić wiersze miejscami. Dzięki temu mamy lepsze wyniki.

Teraz, zamiast równań z taką samą ilością niewiadomych w każdej, mamy równania, w których jest kolejno 1, 2, ..., n niewiadomych. Zaczynając od równania z jedną niewiadomą, idąc do kolejnych, cały czas mamy jedno równanie, w którym jest tylko jedna niewiadoma (w k-tym kroku znamy $k-1$-niewiadomych, podstawiając je do k-tego równania zostaje nam jedna niewiadoma).

Złożoność obliczeniowa to $O(n^3)$, ponieważ dla pierwszej części przechodzimy po każdej kolumnie, następnie w pione po każdej wartości poniżej przekątnej, na koniec przechodzimy po danym wierszu, w którym zerujemy jedną wartość. Są 3 pętle zagnieżdżone.

2.2 Rozkład LU

Może zdarzyć się, że mamy jedną macierz i wiele prawych stron, do których mamy obliczyć rozwiązanie równania. Wtedy, zamiast liczyć za każdym razem eliminację Gaussa od nowa, możemy zapisać macież A w postaci:

$$\mathbf{A} = \mathbf{LU}$$

gdzie:

- U to górno-trójkątna macierz po piewrszym przekształceniu w eliminacji Gaussa,
- L to dolno-trójkątna macierz, w której na pozycji $L_{i,j}$ znajduje się dzielnik, który używaliśmy do wyzerowania wartości w j-tej kolumnie i-tego wiersza.

Możemy policzyć rozkład LU raz, a potem używać go do liczenia wyniku wielokrotnie. Algorytm wyglada bardzo podobnie, jak pierwsza część eliminacji Gaussa, jedynie trzeba zapamiętać w tablicy wyniki dzielenia.

Złożoność algorytmu to $O(n^3)$ (uzasadnienie takie, jak w eliminacji Gaussa, punkt 2.1). Gdy już mamy rozkład LU, to policzenie wyniku ma złożoność $O(n^2)$, czyli tyle, ile druga część eliminacji Gaussa, przejście po n równaniach, które mają odpowiednio 1, 2, ..., n elementów po lewej stronie równiania, po prawej zawsze jeden.

3 Dostosowanie algorytmów

Macierz **A** jest macierzą rzadką, zatem nie musimy przechodzić po wszystkich zerowych elementach. Poniżej jest fragrent macierz A dla $l = 4$.

$$\begin{pmatrix}
a^1_{11} & a^1_{12} & a^1_{13} & a^1_{14} & c^1_{11} & 0 & 0 & 0 & 0 & \dots \\
a^1_{21} & a^1_{22} & a^1_{23} & a^1_{24} & 0 & c^1_{22} & 0 & 0 & 0 & \dots \\
a^1_{31} & a^1_{32} & a^1_{33} & a^1_{34} & 0 & 0 & c^1_{33} & 0 & 0 & \dots \\
a^1_{41} & a^1_{42} & a^1_{43} & a^1_{44} & 0 & 0 & 0 & c^1_{44} & 0 & \dots \\
0 & 0 & 0 & b^2_{14} & a^2_{11} & a^2_{12} & a^2_{13} & a^2_{14} & c^2_{11} & \dots \\
0 & 0 & 0 & b^2_{24} & a^2_{21} & a^2_{22} & a^2_{23} & a^2_{24} & 0 & \dots \\
0 & 0 & 0 & b^2_{34} & a^2_{31} & a^2_{32} & a^2_{33} & a^2_{34} & 0 & \dots \\
0 & 0 & 0 & b^2_{44} & a^2_{41} & a^2_{42} & a^2_{43} & a^2_{44} & 0 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & b^3_{14} & a^3_{11} & \dots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$

Zewnętrzna pętla się nie zmieni, nadal musimy przejść po wszystkich kolumnach, czyli po całej przekątnej. Dla danej wartości na przekątnej mamy maksymalnie l wartości niezerowych poniżej, a dla każdej z nich maksymalnie l niezerowych elementów po prawej. Zatem złożoność całego algorytmu wynosi $O(n \cdot l^2)$.

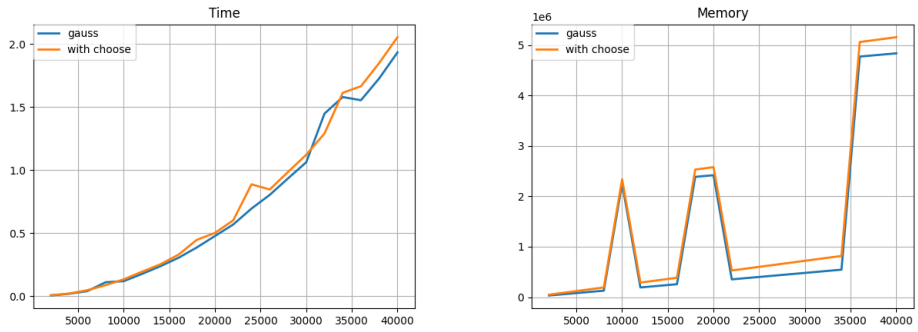
Złożoność eliminacji Gaussa z częściowym wyborem będzie nieznacznie większa, ponieważ musimy dodatkowo przejść po wszytkich elementach pod przekątną w danej kolumnie i znaleźć największą wartość co do wartości bezwzględnej. Zatem złożoność wynosi $O(n \cdot l^3)$.

Rozkład LU liczy się tyle, ile wykonuje się pierwsza część algorytmu eliminacji Gaussa, zatem na złożoność $O(n \cdot l^2)$.

Do zapamiętania macierzy wykorzystałem macierze rzadkie z biblioteki *SparseArrays*. W moich obliczeniach zakładam, że dostęp do wartości w macierzy jest stały (w praktyce jest logarytmiczny).

4 Wyniki testów

Przeprowadziłem testy dla algorytmów, sprawdzając złożoność czasową i pamięciową. Oto wyniki:



Wykres czasu pokazuje, że algorytmy rosną w tak, jak wskazuje obliczona złożoność algorytmów. Oczywiście eliminacja Gaussa z częściowym wyborem jest nieznacznie wolniejsza.

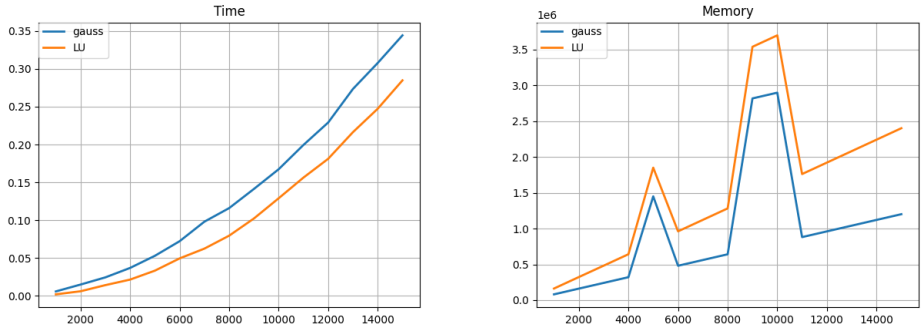
Podobnie to wygląda w kwestii pamięci.

Również sprawdziłem wydajność rozkładu LU. Rozważyłem dwa przypadki:

1. podstawowy algorytm Gaussa rozwiązujący $\mathbf{Ax} = \mathbf{b}$ 10 razy dla tych samych danych,

2. rozkład LU obliczony z macierzy A, a następnie 10-ktornie rozwiązałem $\mathbf{LUx} = \mathbf{b}$ dla tych samych danych,

Oto wyniki:



Rozkład LU zużywa więcej pamięci, co jest spowodowane używaniem sparseArrays (używamy więcej komórek w dolnym trójkącie macierzy), za to widać zdecydowaną przewagę czasową w porównaniu do zwykłego algorytmu.

5 Wnioski

Projekt pokazuje, że jeśli znamy dane wejściowe, to warto zmodyfikować algorytm w odpowiedni sposób. W tym przypadku omijając wiele komórek zerowych byliśmy w stanie zdecydowanie zmniejszyć złożoność obliczeniową algorytmu.