OBLICZENIA NAUKOWE – SPRAWOZDANIE 5

autor: Jan Sieradzki
nr indeksu: 236441

1. Opis problemu

1. Napisać funkcję rozwiązującą układ $Ax=b$ metodą eliminacji Gaussa uwzględniającą specyficzną postać (1) macierzy A dla dwóch wariantów:
(a) bez wyboru elementu głównego,
(b) z częściowym wyborem elementu głównego.

2. Napisać funkcję wyznaczającą rozkład LU macierzy A metodą eliminacji Gaussa uwzględniającą specyficzną postać (1) macierzy A dla dwóch wariantów:
(a) bez wyboru elementu głównego,
(b) z częściowym wyborem elementu głównego.

3. Napisać funkcję rozwiązującą układ równań $Ax=b$ (uwzględniającą specyficzną postać (1) macierzy
(a) jeśli wcześniej został już wyznaczony rozkład LU przez funkcję z punktu 2.

(1) Postać macierzy $A \in R^{n\times n}$ dla powyższych problemów :

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

$v=\frac{n}{l}$ przy założeniu, że n jest podzielne przez l, gdzie l jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych (bloków): A_k, B_k, C_k. Bloki : $A_k \in R^{l\times l}$, $k=1,\ldots,v$ jest macierzą gęstą, 0 jest kwadratową macierzą zerową stopnia l, $C_k \in R^{l\times l}$, $k=1,\ldots,v-1$ jest macierzą diagonalną,a macierz $B_k \in R^{l\times l}$, $k=2,\ldots,v$ jest następującej postaci:

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

2. Opis algorytmów

Na wstępie warto pomyśleć o efektywnym sposobie przechowywania macierzy, ponieważ jest rzadka, nie opłaca jej się trzymać w tablicy dwuwymiarowej. Skorzystałem z biblioteki SparseMatrixCSC, która „kompresuje” liczne zera macierzy, oszczędzając tym samym pamięć.

I) Rozwiązywanie układu $Ax=b$ zmodyfikowaną metodą eliminacji Gaussa

Metoda eliminacji Gaussa służy do wyliczania wektora x w równaniu Ax = b, tzn służy do rozwiązywania układu równań liniowych. Celem metody jest sprowadzenie macierzy A do tzn, macierzy trójkątnej górnej, gdzie wszystkie elementy poniżej głównej przekątnej są zerami, a sama przekątna jest niezerowa :

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \cdots & a_{2,n-1} & a_{2,n} \\ 0 & 0 & a_{3,3} & \cdots & a_{3,n-1} & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & a_{n,n} \end{pmatrix}$$

Aby to osiągnąć, potrzebujemy n iteracji algorytmu. W każdym kroku k bierzemy na celownik kolejny główny element $a_{k,k}$ i naszym celem jest wyzerowanie wszystkich elementów pod tym elementem, tzn wszystkich $a_{r,k}$ gdzie r to numer rzędu , k numer kolumny i r > k. W tym celu do kolejnych elementów wiersza *r*-tego (r = k+1,k+2,…,n) dodajemy kolejne elementy wiersza k-tego przemnożone przez $-\frac{a_{r,k}}{a_{k,k}}$, tzn od r=k+1 do r=n mamy $a_{r,i}=a_{r,i}-\frac{a_{r,k}}{a_{k,k}}*a_{k,i}$ gdzie i = k,k+1 ,…,n.

Analogicznie postępujemy dla wektora b, który w każdym korku iteracji (tak samo jak dla elementów a jest wyliczany : $b_r=b_r-\frac{a_{r,k}}{a_{k,k}}*b_k$, czyli tak naprawdę dołączamy wektor b do prawej strony macierzy a i traktujemy go tym samym algorytmem.

Gdy już otrzymamy naszą macierz schodkową i odpowiadający jej nowy wektor b, możemy w prosty sposób wyliczyć rozwiązanie,tzn wektor x. Posłuży nam do tego rekurencyjny wzór :

$$x_n=\frac{b_n}{a_{n,n}}$$

$$x_r=\frac{b_r-a_{r,n}*x_n-\ldots-a_{r,r+1}*x_{r+1}}{a_{r,r}}$$

Jak wspomniałem wyżej, aby wykonać metodę, diagonala (główna przekątna) macierzy A musi być niezerowa. Nie zawsze tak jest. Odpowiedzią jest częściowy wybór elementu głównego (częściowy, bo wybieramy tylko rzędy, a nie rzędy i kolumny). Polega on na tym, iż w każdym kroku algorytmu wykonujemy dodatkową pętlę od i=k+1 do n, w której szukamy $max|a_{i,k}|$ i następnie zamieniamy rzędy, w której to maksimum występuje, z rzędem k, podmieniając w ten sposób element główny (element $a_{k,k}$) . Dzięki temu unikamy zer w diagonali oraz dzięki wyborowi bezwzględnie większych elementów, zwiększamy dokładność obliczeniową. W moim algorytmie w tym celu tworzę specjalny wektor permutacji, który zapamiętuje zamiany. W związku z operacją zamiany rzędów mam pewność, że algorytm się wykona, jeśli tylko macierz nie jest osobliwa.

Zważając, że nasze zadanie dotyczy macierzy rzadkiej, możemy w znaczący sposób usprawnić algorytm Gaussa, poprzez skrócenie granic poszczególnych pętli tak, aby pominąć zbędne kroki na zerowych elementach.

Dzięki niżej opisanym modyfikacjom, zamiast normalnej złożoności algorytmu $O(n^3)$ (mamy trzy pętle w algorytmie Gaussa) otrzymamy złożoność $O(n)$(ponieważ dwie z pętli będą pomijalne, gdyż znacznie ograniczymy ilość ich iteracji), co jest niewątpliwie ogromnym zyskiem. Modyfikacje algorytmu możemy przeprowadzić na dwa sposoby, ze względu na wcześniej opisany wybór elementu głównego.

a) bez wyboru elementu głównego

Modyfikacje, jak już wcześniej zostało zaznaczone, będą polegały na ustaleniu dwóch nowych granic pętli, wykorzystujących blokowość i rzadkość macierzy. Jako, że nie możemy skrócić pierwszej pętli, która przechodzi po wszystkich rzędach zerując kolejne kolumny, skupimy się na dwóch pozostałych.

Na początku pętla, która dla ustalonego już elementu głównego iteruje po kolejnych rzędach macierzy w celu usuwania elementów pod elementem głównym (dotychczas od r=k+1 do r=n). Można zauważyć, że pod każdym elementem głównym niezerowe elementy $a_{r,k}$ będą obecne tylko wewnątrz bloku A_k rozpatrywanego elementu głównego i ewentualnie (tylko dla jednej kolumny) w bloku niżej B_{k+1} , reszta elementu jest już i tak wyzerowana, dlatego nie trzeba po nich iterować. Z racji, że bloki są wielkości l x l, możemy wyznaczyć nową granicę pętli $limitRow = \lfloor \frac{k}{l} \rfloor * l + l$. Dzięki tak opisanej granicy pętla przejdzie do końca interesującego nas bloku A_k i jeśli nasz element główny znajduje się na kolumnie l w bloku A_k, to weźmie pod uwagę również elementy w bloku poniżej B_{k+1}. Przy skrajnym rzędzie n, za limitRow przyjmujemy n, aby granica nie przekroczyła wielkości macierzy. Przy założeniu, że l traktujemy jako stałą, mamy złożoność pętli zredukowaną z $O(n)$ do $O(1)$.

Zajmijmy się trzecią pętlą, która dla ustalonego elementu głównego i rzędu, który modyfikujemy algorytmem, ma przechodzić przez kolejne kolumny rzędu (dotychczas od i=k+1 do i=n) . Można zauważyć, że aby iteracje miały sens i faktycznie modyfikowały kolejne elementy, to elementy $a_{k,i}$ muszą być różne od 0. Korzystając z cech naszej macierzy, możemy zauważyć, że pętla ma sens dla granicy limitCol = k + l , ponieważ niezerowe elementy na prawo od elementu $a_{k,k}$ mogą wystąpić tylko wewnątrz bloku A_k i na diagonali C_k, która jest oddalona od $a_{k,k}$ zawsze o dokładnie wielkość bloku l. Stąd nasza nowa granica, która zmniejsza złożoność pętli z $O(n)$ do $O(1)$ przy zależeniu, że l traktujemy jako stałą . Jako, że w wyniku operacji eliminacji Gaussa nie powstają żadne nowe elementy niezerowe, analogiczne zmiany wykonywane są w części wyliczając rekurencyjnie wektor x. Zamiast odejmować elementy od r+1 do n, odejmujemy od r+1 do limitCol = i + l.

b) z częściowym wyborem elementu głównego

W stosunku do wyżej opisanych modyfikacji, tutaj trzeba wziąć poprawkę na permutację rzędów. $limitRow = \lfloor \frac{k}{l} \rfloor * l + l$ się nie zmienia, zamiany nie mają na tę granicę wpływu, gdyż są i tak wykonywane wewnątrz tego limitRow. Inaczej to wygląda w trzeciej pętli, gdzie granica będzie dana wzorem $limitCol = \lfloor \frac{k-1}{l} \rfloor * l + 2 * l$. Wzór zmienił się w stosunku do wcześniejszego, ponieważ

jeżeli zamienimy rząd wewnątrz bloku A_k z końcowym rzędem tegoż bloku, to nie dojdziemy w k + l krokach do elementu w bloku C, dlatego zawieramy w granicy C cały blok A_k i C_k, mając pewność, że złapiemy wszystkie elementy niezerowe (przy założeniu o stałym l, złożoność dalej $O(1)$, choć nadal większa niż przy wcześniejszej granicy). Analogicznie postępujemy wyliczając rekurencyjnie wektor x, gdzie granicą będzie $limitCol = \lfloor \frac{k-1}{l} \rfloor * l + 2 * l$.

II) Wyznaczyć rozkład LU zmodyfikowaną metodą eliminacji Gaussa

Rozkład LU ma na celu wygenerowanie z macierzy A_{nxn} dwóch macierzy: trójkątnej dolnej $L_{n \times n}$ o elementach głównej przekątnej równych 1 i trójkątnej górnej $U_{n \times n}$, taką, że $A_{nxn} = L_{n \times n} * U_{n \times n}$. Aby ten cel osiągnąć, wystarczy lekko zmodyfikować metodę eliminacji Gaussa, która z definicji wytwarza macierz $U_{n \times n}$. Modyfikacja będzie polegała na tym, aby z mnożników c = $\frac{a_{r,k}}{a_{k,k}}$ utworzyć macierz trójkątną dolną $L_{n \times n}$. W celu oszczędności pamięciowej będziemy zapisywać macierze L i U w jednej macierzy, w miejscu macierzy wejściowej A. Wspomniane mnożniki $\frac{a_{r,k}}{a_{k,k}}$ w każdej iteracji, gdy usuwamy $a_{r,k,}$ będziemy wstawiać w miejsce właśnie tego elementu. W ten sposób otrzymamy macierz A przechowującą zarówno macierz L jak i U, tylko lekko modyfikując metodę eliminacji Gaussa. Złożoność obliczeniowa jest, tak samo jak w Gaussie, bez modyfikacji $O(n^3)$, z modyfikacjami $O(n)$.

a) bez wyboru elementu głównego

Modyfikacje analogiczne do modyfikacji w metodzie eliminacji Gaussa.

b) z częściowym wyborem elementu głównego

Modyfikacje analogiczne do modyfikacji w metodzie eliminacji Gaussa.

III) Napisać funkcję rozwiązującą układ równań $Ax = b$ wykorzystując rozkład LU

Rozwiązywanie układu równań Ax=b wykorzystując rozkład LU należy podzielić na dwa etapy. Korzystając z tego, że $A_{nxn} = L_{n \times n} * U_{n \times n,}$ mamy L * U * x= b, korzystając z łączności mnożenia, pierw znajdujemy takie Y, że L * y = b, a potem rozwiązujemy U * x = y, oba równania są proste dzięki trójkątnej budowie L i U. Sprawia to, że samo wyliczenie rozwiązania, mając już owy rozkład, staje się bardzo proste, co jest efektywne gdy macierz A jest niezmienna, a chcemy wyliczyć rozwiązanie x dla różnych b. Aby wyliczyć L * y = b, należy skorzystać z rekurencyjnego wzoru :

$$y_1 = b_1$$

$$y_r = b_r - l_{r,1} * y_1 - ... - l_{r,r-1} * y_{r-1}$$

Za jego pomocą otrzymujemy wektor y. Normalnie operacja ta ma złożoność $O(n^2)$, jednak podstawiając granicę startową $startCol = \lfloor \frac{k-1}{l} \rfloor * l$ redukujemy ją do $O(n)$, podobnie jak wcześniej, w ustalaniu granicy wykorzystujemy fakt, że macierz jest rzadka i blokowa, i zaczynamy odejmować kolejne składniki $l_{r,k} * y_k$ dopiero od momentu, kiedy $l_{r,k}$ jest niezerowa, a można zauważyć, że macierz L ma zera w tych samych miejscach co macierz A.

Przejdźmy do wyliczania U * x = y, które się niczym nie różni od wyliczania z którym mieliśmy do czynienia w metodzie eliminacji Gaussa. W celu zoptymalizowania, które przyniesie identyczną korzyść jak w równaniu wyżej, podstawiamy jak w Gaussie, za granicę limitCol = k + l ($limitCol = \lfloor \frac{k-1}{l} \rfloor * l + 2 * l$ dla LU z permutacjami). Wzory rekurencyjne dane są poniżej :

$$x_n = \frac{b_n}{u_{n,n}} \qquad\qquad x_r = \frac{b_r - u_{r,n} * x_n - ... - u_{r,r+1} * x_{r+1}}{u_{r,r}}$$

3. Wyniki eksperymentów

Przeprowadziłem testy stworzonych funkcji na trzech macierzach A o n=16, n=10000, n=50000 o rozmiarze bloku l=4. Macierze spełniają warnki zadania, tzn są blokowe i rzadkie. Oczekiwany wynik równań w zadaniu to wektor x = [1,1,1 … 1]. Porównałem czas, pamięć i popełniane błędy algorytmów. Metoda z podziałem LU jest sprawdzana dwuetapowo, tzn sam podział i rozwiązanie. Wszystkie algorytmy są w dwóch wersjach, z częściowym wyborem elementu głównego i bez niego. Wyniki są zaprezentowane w tabelach poniżej :

N	Gauss bez wyboru			Gauss z wyborem		
	czas [s]	**pamięć**	**błąd**	**czas[s]**	**pamięć**	**błąd**
16	0.000031	3.109 KiB	3.772410574826941e-15	0.000030	3.313 KiB	3.855927796350635e-16
10000	0.134816	1.907 MiB	8.174088292925233e-14	0.143965	1.984 MiB	3.7627717011537425e-16
50000	3.627491	4.959 MiB	3.360679143674386e-14	3.610516	5.341 MiB	4.1319303219604135e-16

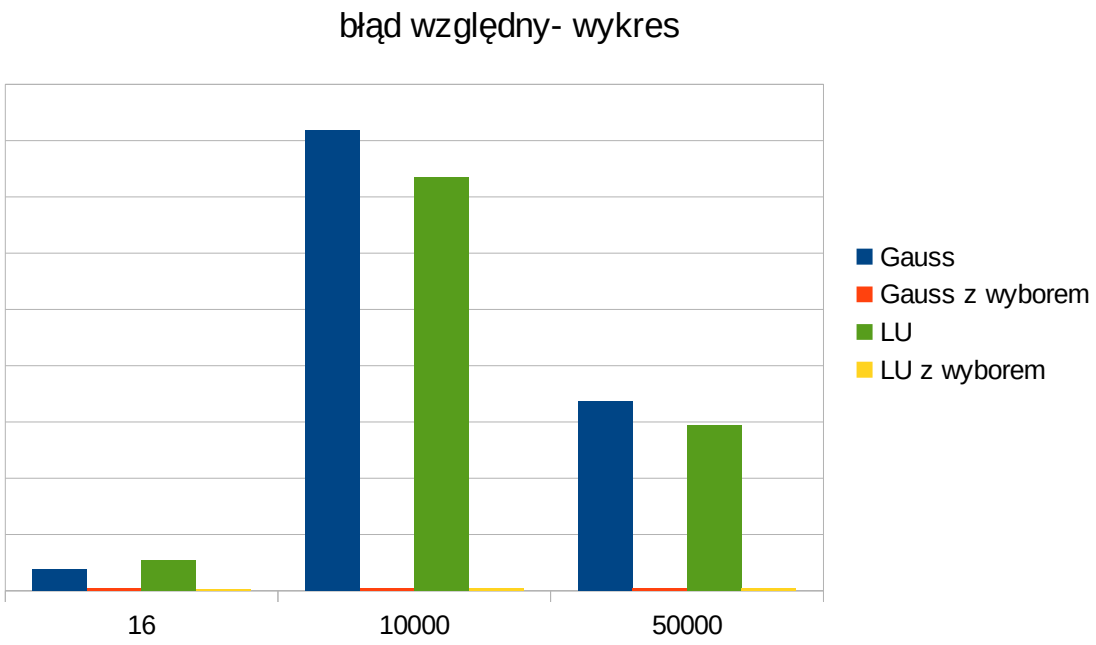
Porównanie rozwiązania układu równań metodą eliminacji Gaussa z wyborem i bez niego

N	Podział na LU bez wyboru			Rozwiązanie za pomocą LU bez wyboru		
	czas [s]	**pamięć**		**czas[s]**	**pamięć**	**błąd**
16	0.000015	2.906 KiB		0.000017	208 bytes	5.339180149127962e-15
10000	0.134769	1.831 MiB		0.001056	78.203 KiB	7.350041543996924e-14
50000	3.543605	4.578 MiB		0.005112	390.703 KiB	2.931976314263311e-14

Porównanie rozwiązania układu równań za pomocą LU i samego generowania LU, bez wyboru

N	Podział na LU z wyborem			Rozwiązanie za pomocą LU z wyborem		
	czas [s]	**pamięć**		**czas[s]**	**pamięć**	**błąd**
16	0.000027	3.109 KiB		0.000006	208 bytes	2.9634845277824204e-16
10000	0.142498	1.907 MiB		0.002914	78.219 KiB	3.7184019413221594e-16
50000	3.669222	4.959 MiB		0.010500	390.719 KiB	4.0961422339512155e-16

Porównanie rozwiązania układu równań za pomocą LU i samego generowania LU, z wyborem



Dane o błędzie względnym przedstawione na wykresie, algorytmy z wyborem wyraźnie mniejsze

4. Wnioski

Dzięki optymalizacji algorytmów pod konkretny rodzaj macierzy osiągnęliśmy bardzo duży zysk złożoności w porównaniu do klasycznych metod. Na podstawie powyższych wyników można wyciągnąć kilka wniosków. Przede wszystkim w każdej z metod pod względem dokładności obliczeniowej wyraźnie lepsze okazują się ich wersje z częściowym wyborem. Oprócz tego warto pamiętać, że częściowy wybór naprawia potencjalnie „zepsute” macierze dla eliminacji Gaussa z powodu zerowych elementów na diagonali. Jest to jednak okupione lekko zwiększonym zużyciem pamięci i czasu. W porównaniu LU, a czysta eliminacja Gaussa, nieznacznie lepiej wypada LU, wymagania czasowe i pamięciowe są bardzo podobne, co nie powinno dziwić, ponieważ podział LU jest zaimplementowany, niemal identycznie, jak eliminacja Gaussa. Warto zwrócić uwagę na fakt, iż samo rozwiązanie układu za pomocą LU zajmuje niewielki ułamek pamięci i czasu w porównaniu do generowania macierzy LU. Jest to bardzo pomocna własność, gdy np. operujemy na niezmiennej macierzy A, liczymy raz jej LU i możemy wyliczyć wektor x dla różnych wektorów b w bardzo efektywny sposób.