ONLista5

Paweł Kędzierski

January 2025

1 Co należało zrobić?

Należało zaimplementować funkcje rozwiązujące układ równań Ax = b. Macierz A jest macierzą rzadką, to jest taką, która ma dużo elementów zerowych, oraz blokową.

$$A = \begin{pmatrix} A_1 & C_1 & 0 & 0 & 0 & \cdots & 0 \\ B_2 & A_2 & C_2 & 0 & 0 & \cdots & 0 \\ 0 & B_3 & A_3 & C_3 & 0 & \cdots & 0 \\ 0 & 0 & B_4 & A_4 & C_4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & B_{v-2} & A_{v-2} & C_{v-2} & 0 \\ 0 & 0 & 0 & 0 & B_{v-1} & A_{v-1} & C_{v-1} \\ 0 & 0 & 0 & 0 & 0 & B_v & A_v \end{pmatrix}$$

A, B, i C są innej postaci. 0 jest kwadratową macierzą zerową, A jest zwykłą, gęstą macierzą, B ma tylko ostatnie dwie kolumny niezerowe, a C jest macierzą diagonalną (ma wartości tylko na jej przekątnej).

Większą część macierzy zajmują macierze zerowe, dlatego nie ma sensu pamiętać wartości wszystkich elementów. Przydatna będzie odpowiednia struktura, która będzie pamiętać tylko elementy niezerowe, no i odpowiednie zaadaptowanie algorytmów, które będą uwzględniać jej zmienioną postać.

2 Co zrobiłem?

Stworzyłem strukturę myMatrix, w której przechowuję informację o jej rozmiarze, rozmiarze bloku oraz samą macierz jako SparseMatrixCSC.

Funkcje: gauss, gaussPivoted, lu, luSolve znajdują się w module blocksys. Funkcje odpowiedzialne za odczyt z i zapis do pliku oraz obliczanie wektora prawych stron (readMatrix, readVector, computeRightSideVector, saveToFile) umieściłem w osobnym module IOMartix. Tam znajduje się też struktura myMatrix. Testy znajdują się w pliku tests.jl, gdzie przy pomocy paczki Test testuję zaimplementowane funkcje. Plik for_plots.jl służy do generowania danych, które poźniej będę wykorzystywać do narysowania wykresów.

3 Metoda eliminacji Gaussa

Aby przekształcić macierz, iteruję po liczbie wierszy. Dokonuję obliczenia tzw. *mnożnika eliminacyjnego* w każdym kroku k:

$$l_{ik} = \frac{A_{ik}^{(k)}}{A_{kk}^{(k)}}.$$

Następnie przekształcam macierz odpowiednio – wartości $A_{ij}^{(k)}$ ustawiam na 0 dla i należącego do $k+1, \ldots, n$ i $j = k$, bądź dla i, j należących do $k+1, \ldots, n$, odejmując od nich wartość $l_{ik} \cdot A_{kj}^{(k)}$. Reszta wartości pozostaje bez zmian. Analogicznie przekształcam wektor b (czyli zadany wektor prawych stron).

Mając tak przekształconą macierz metodą eliminacji Gaussa, mogę zacząć rozwiązywać zadany układ równań $Ax = b$. Obliczam po kolei ("od dołu") wartości wektora x, początkowo zainicjowanego na 0, używając wzoru:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{i,i}}.$$

Aby można było przeprowadzić eliminację, elementy znajdujące się na jej przekątnej powinny być daleko od zera. Spowodowane jest to tym, że w powyższym wzorze elementy główne są w mianowniku. Gdyby były bliskie zeru, mogłoby to być problematyczne z numerycznego punktu widzenia.

4 Metoda eliminacji Gaussa z wyborem

Ze względu na fakt, że w niezmodyfikowanej wersji metody Gaussa może się zdarzyć, że na przekątnej pojawią się elementy bliskie zeru, można zmodyfikować tę metodę o wybór elementu głównego. Przestawiam więc wiersze w macierzy tak, aby na jej przekądnej były duże elementy. Po modyfikacji macierzy, dalsza część algorytmu przebiega bez zmian.

5 Złożoność metody eliminacji Gaussa z wyborem

Cały algorytm obliczania rozwiązania układu równań ma złożoność $O(n^3)$. Jednakże korzystając z wiedzy na temat specyficznego wyglądu macierzy A, złożoność ta może zostać obniżona. Wyznaczam indeks kolumny w macierzy, w której znajduje się element różny od zera. Jest ich minimum z rzędu + rozmiaru bloku, lub rozmiaru macierzy. Dzieje się tak, ponieważ indeks nie może być wyższy od

rozmiaru macierzy, a opisana wczesniej jej struktura pokazuje, że "najbardziej na prawo"sa macierze diagonalne C. Wyznaczam także indeks rzędu tego ostatniego elementu, czyli minimum z wielkości macierzy i l + l * floor(c/l). Przyjmując więc założenie z zadania, że l jest stałą, widać, że złożoność obliczeniowa tej metody została zmniejszona do O(n). Jeżeli chodzi o wersję z wyborem elementu głównego, w celu ograniczenia liczby przestawień, używam wektora, który przechowuje przestawienia dokonane na poszczególnym etapie. Złożoność pamięciowa również będzie O(n) - struktura myMatrix ma rozmiar liniowy, wektor prawych stron również. W kodzie nie jest przechowywane nic, co potrzebuje większego rozmiaru pamięci.

Algorithm 1 Gauss

```
1:  function GAUSS(A, b)
2:      for i ← 1 to A.size do
3:          l_col ← min(i + A.block_size, n)
4:          l_row ← min(n, A.block_size + A.block_size · ⌊i/A.block_size⌋)
5:          for j ← i + 1 to l_row do
6:              z ← A[j, i]/A[i, i]
7:              A[j, i] ← 0
8:              for k ← i + 1 to l_col do
9:                  A[j, k] ← A[j, k] − z · A[i, k]
10:             end for
11:             b[j] ← b[j] − z · b[i]
12:         end for
13:     end for
14:     x ← zeros(n)
15:     for i ← n downto 1 do
16:         sum ← 0.0
17:         l_col ← min(n, i + A.block_size)
18:         for j ← i + 1 to l_col do
19:             sum ← sum +A[i, j] · x[j]
20:         end for
21:         x[i] ← (b[i] − sum)/A[i, i]
22:     end for
23:     return x
24: end function
```

6 Algorytm LU

Rozkład LU to inaczej podział macierzy A na dolnotrojkątną L i gornotrójkątną U w taki sposób, że ich iloczyn $LU = A$. Taki rozkład będzie możliwy tylko wtedy, gdy

$$\text{Rank}(A_{11}) + k \geq \text{Rank} \begin{bmatrix} A_{11} \\ A_{12} \end{bmatrix} + \text{Rank} \begin{bmatrix} A_{11} & A_{21} \end{bmatrix}, \quad \text{dla } k \in \{1, 2, \ldots, n-1\}. \tag{1}$$

W celu dokonania rozkładu LU używam wcześniej zaimplementowanej metody Gaussa, z drobną modyfikacją. Nie zeruje elementów pod przekątna, tylko wstawiam tam współczynniki ułatwiające to obliczenie.

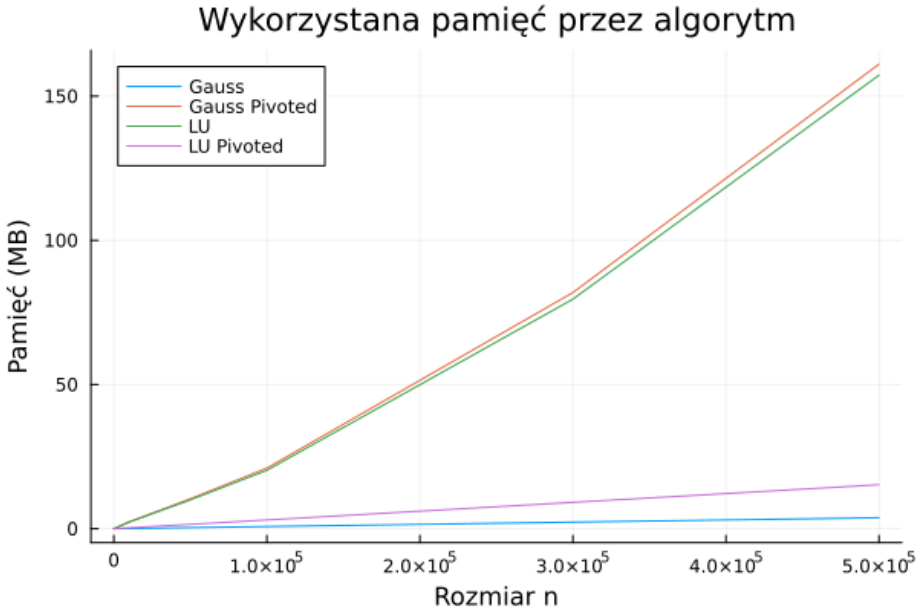
Jako że, po rozkładzie na LU, rozwiązanie równania $Ax = b$ sprowadza się tak naprawdę do rozwiązania równania $LUx = b$, czyli układu równań

$$\begin{cases} L \cdot a = b \\ U \cdot c = a \end{cases}$$

to złożoność obliczeniowa nie będzie różna od złożoności przy metodzie eliminacji Gaussa. Teoretycznie oczywiście ta metoda ma złożoność $O(n^3)$, jednakże ze względu na zastosowane optymalizacje przy implementacji metody Gaussa, wykorzystywanej przez rozkład LU, złożoność będzie wynosić $O(n)$. Wykonywanie algorytmu podstawienia wstecz oraz w przód nie wpłynie na nią – oba mają złożoność liniową. Tak samo będzie ze złożonością pamięciową.

Algorithm 2 Algorytm LU

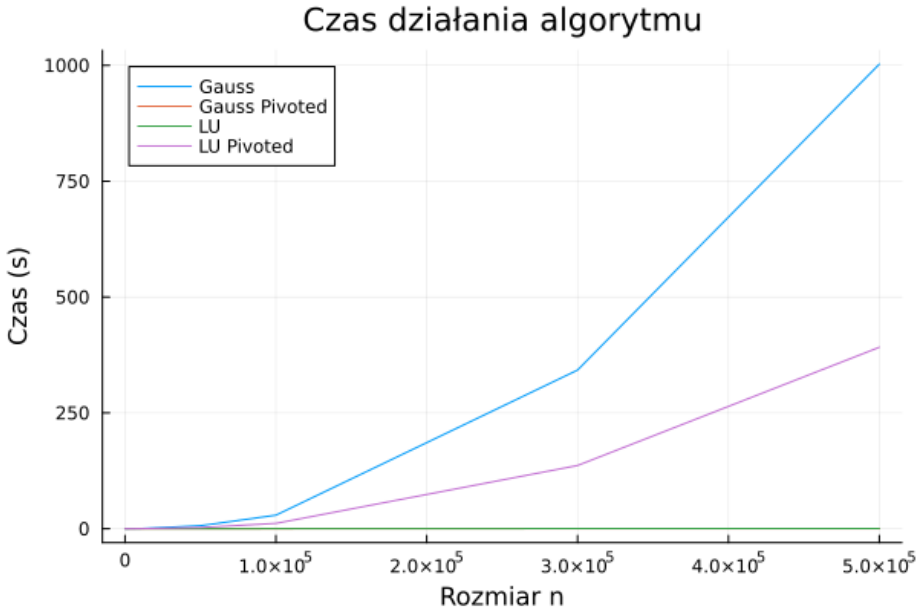
```
1: function LU(A)
2:     for i = 1 to A.size − 1 do
3:         l_col ← min(i + A.block_size, A.size)
4:         l_row ← min(A.size, A.block_size + (i/A.block_size) ∗ A.block_size)
5:         for j = i + 1 to l_row do
6:             A.matrix[j, i] ← A.matrix[j, i]/A.matrix[i, i]
7:             for k = i + 1 to l_col do
8:                 A.matrix[j, k] ← A.matrix[j, k] − A.matrix[j, i] ∗ A.matrix[i, k]
9:             end for
10:        end for
11:    end for
12: end function
```



7 Porównanie algorytmów

Przeprowadziłem porównanie czasu działania oraz ilości zajętej pamięci przez programy i na poniższych wykresach przedstawiłem średnie wyniki. Porównuję ze sobą: własne implementacje Gaussa, Gaussa z wyborem, LU oraz LU z wyborem. W poniższej tabeli umieściłem wartości błędów względnych, które powstały przy obliczeniach wektorów prawych stron na podstawie macierzy testowych. Pominąłem w nich wartości dla metody LU, ponieważ przez fakt, że wykorzystuje ona metodę Gaussa, wyniki będą identyczne. Własną implementację algorytmów porównuję z wbudowaną funkcją Julii A \ b.

n	Gauss	Gauss Pivoted	Julia
16	0.08270014887395218	0.03558622915986037	0.08270014887395191
10000	0.12491705971293866	5.716537824382048	0.12491705971293865
50000	0.12491307684222912	3.491332617673817	0.12491307684222921
100000	0.12474382817455507	3.6815710179782437	0.12474382817455479
300000	0.12502965649057068	9.974929083902435	0.12502965649057068
500000	0.12489549399094038	28.13724717372499	0.12489549399094105



8 Obserwacje i wnioski końcowe

Brak częściowego wyboru znacząco zmniejsza błąd względny. Jak można zauważyć, dostosowanie podejścia do problemu, w tym odpowiednie modyfikacje algorytmów oraz struktury przechowywania danych, ma istotny wpływ na szybkość obliczeń. Dlatego przed rozpoczęciem działań warto przeanalizować problem i rozważyć potencjalne usprawnienia, które mogą poprawić efektywność obliczeń.