Metody Numeryczne Projekt 2

Piotr Sieński

May 2022

1 Konstrukcja macierzy

Numer indeksu - 184297, więc c = 9, d= 7,e = 2, f = 4,
$a_1 = 5 + e = 7$, $a_2 = a_3 = -1$, N = 997

Macierze A i C (używane w zadaniach A i C) oraz macierze użyte w zadaniu D wygenerowane zostały przy użyciu funkcji zwracającej macierz o rozmiarach N x N znajdującej się w pliku main.cpp. :

```
Matrix& constructA(double a1, double a2, double a3, int N);
```

A Macierz b przy użyciu znajdującej się w tym samym pliku funkcji zwracającej wektor kolumnowy o długości N :

```
Matrix& constructB(double f, int N);
```

$$A = \begin{bmatrix} 7 & -1 & -1 & 0 & 0 & 0 & \dots & 0 & 0 \\ -1 & 7 & -1 & -1 & 0 & 0 & \dots & 0 & 0 \\ -1 & -1 & 7 & -1 & -1 & 0 & \dots & 0 & 0 \\ 0 & -1 & -1 & 7 & -1 & -1 & \dots & 0 & 0 \\ 0 & 0 & -1 & -1 & 7 & -1 & \dots & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 7 & \dots & -1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 7 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 7 \end{bmatrix}$$

$$C = \begin{bmatrix} 3 & -1 & -1 & 0 & 0 & 0 & \dots & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & \dots & 0 & 0 \\ -1 & -1 & 3 & -1 & -1 & 0 & \dots & 0 & 0 \\ 0 & -1 & -1 & 3 & -1 & -1 & \dots & 0 & 0 \\ 0 & 0 & -1 & -1 & 3 & -1 & \dots & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 3 & \dots & -1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 3 \end{bmatrix}$$

$$\mathrm{b} = \begin{bmatrix} -0.958 \\ -0.544 \\ 0.650 \\ 0.912 \\ -0.132 \\ -0.988 \\ -0.428 \\ 0.745 \\ \vdots \end{bmatrix}$$

2 Algorytmy rozwiązywania układu równań liniowych

Wyszystkie opisywane ponizej algorytmy przyjmują na wejście macierze: A - o wymiarach N x N (bez zerowych elementów na diagonali) oraz b o wymiarach N x 1, algorytmy iteracyjne przyjmują również oczekiwaną normę z residuum. Implementacje opisywanych algorytmów znajdują się w pliku LinearEquations.cpp. Każda z funkcji implementujących te algorytmy zwraca macierz wynikową, liczbę wykonanych iteracji (LU zwraca 0), normę z residuum wyniku oraz dodatkową informację o przebiegu algorytmu - INVALID_SIZE w przypadku nieprawidłowych rozmiarów macierzy na wejściu, NORM_ACHIEVED dla algorytmów iteracyjnych i OK dla algorytmu LU w przypadku powodzenia, dodatkowo algorytmy iteracyjne mogą zwrócić ITERS_EXCEEDED jeśli algorytm zakończył działanie ze względu na przekroczenie maksymalnej liczby iteracji.

2.1 Algorytmy iteracyjne

Oba algorytmy realizowane są przez tę samą funkcję, gdyż różnią się jedynie postacią jednej z sum wyznaczanych dla każdego elementu wektora x (zależnie od wybranego algorytmu liczona jest inna suma). Po każdej iteracji liczona jest norma z wektora residuum i jeśli jest ona mniejsza od zadanej algorytm kończy dzialanie.

2.1.1 Jacobi

$i-$ty element wektora wynikowego wyznaczamy ze wzoru:

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} A_{ij} x_j^{(k)}}{A_{ii}}$$

2.1.2 Gauss - Seidel

$i-$ty element wektora wynikowego wyznaczamy ze wzoru:

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} A_{ij} x_j^{(k)}}{A_{ii}}$$

2.2 LU

Najpierw wykonywana jest faktoryzacja macierzy A na macierze trójkątne L i U. Macierz L jest inicjalizowana jako kopia macierzy A, a L jako macierz jednostkowa. Następnie obie macierze wypełniane są element po elemencie wedlug poniższego algorytmu :

```
for k = 1 to N-1 {
    for j = k to N {
        L[j][k] = U[j][k] / U[k][k];
        for i = k to N {
            U[j][i] = U[j][i] - L[j][k] * U[k][i];
        }
    }
}
```

Następnie, w celu wyznaczenia rozwiązania układu równań Ax=b (LUx=b) tworzymy wektor pomocniczy y = Ux i wyznaczamy go poprzez rozwiązanie Ly = b metodą podstawienia w przód.

$i-$ty element wektora y wyznaczamy ze wzoru:

$$y_i = b_i - \sum_{k=1}^{i-1}(L_{ik} \cdot y_k)$$

Finalnie wyznaczamy wektor x rozwiązując układ równań Ux = y metodą podstawiania wstecz.

$i-$ty element wektora x wyznaczamy ze wzoru:

$$x_i = \frac{y_i - \sum_{k=N}^{i+1}(U_{ik} \cdot x_k)}{U_{ii}}$$

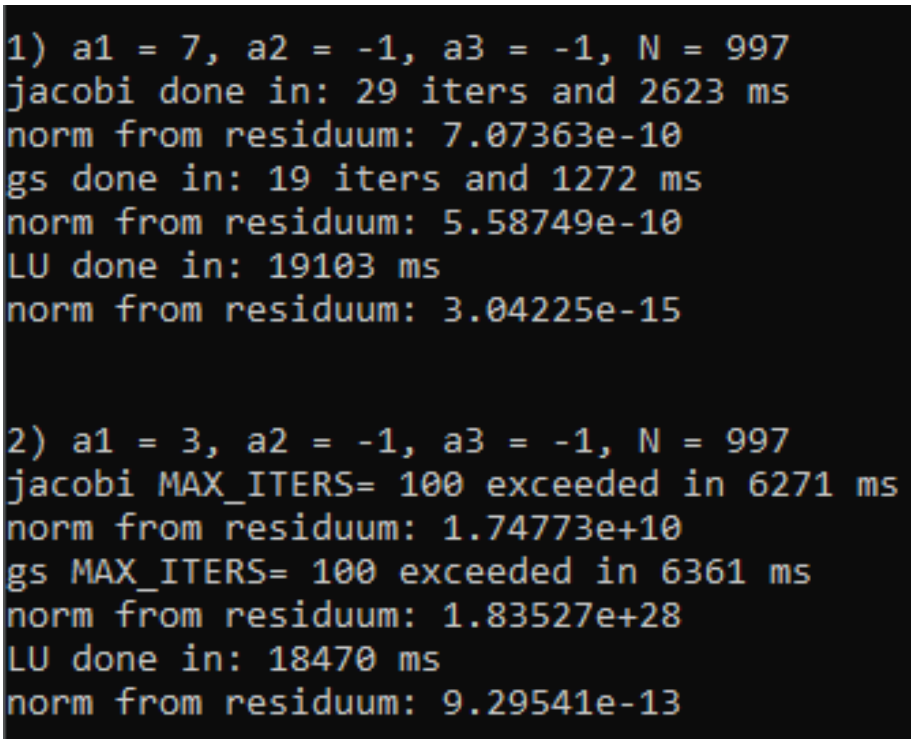
3 Zadania

3.1 A, B

Algorytm Jacobiego znajduje satysfakcjonujące rozwiązanie w 29 iteracji i w czasie 2623 milisekund, a algorytm Gaussa-Seidla w 19 iteracji i w czasie 1272 milisekund

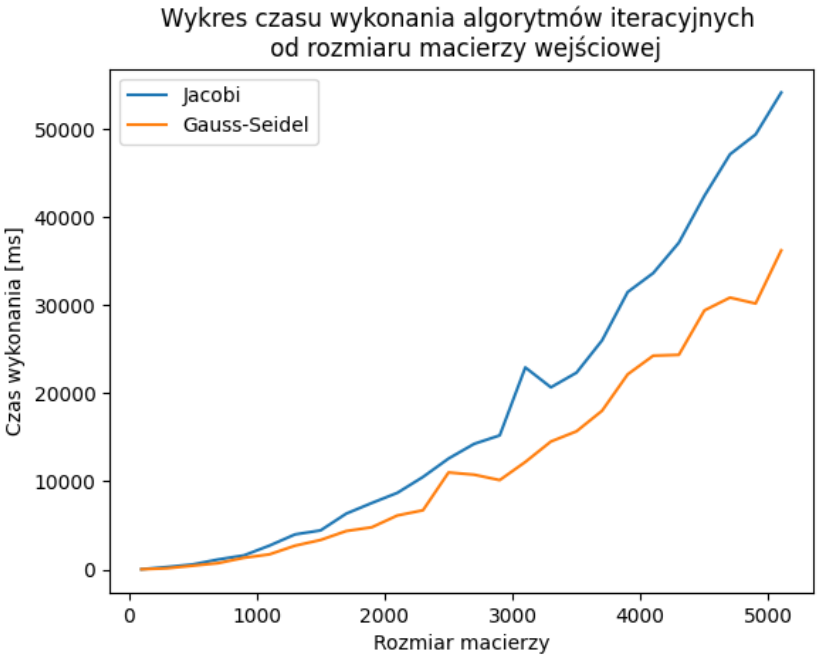
3.2 C, D

Oba algorytmy iteracyjne nie zbiegają się do rozwiązania i po 100 iteracjach normy z residuum osiągają wartości dalece odbiegające od 0 (rzędu 10^{10} i 10^{28}. Algorytm korzystający z faktoryzacji LU znajduje rozwiązanie w czasie 18470 milisekund i osiąga normę z residuum rzędu 10^{-13}.



Wyniki działania programu dla podpunktów A-D

3.3 E



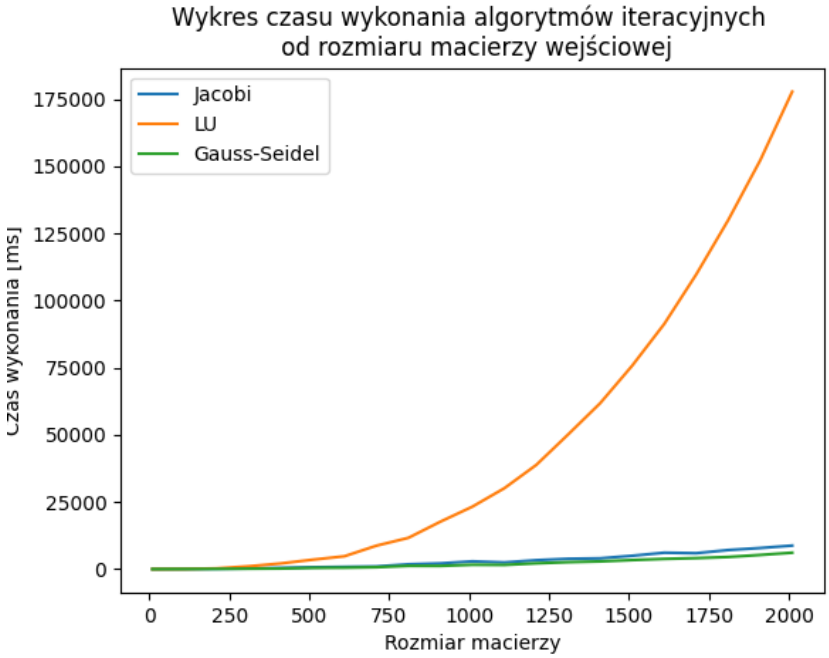
Średni stosunek czasu działania algorytmu Jacobiego do czasu działania algorytmu Gaussa-Seidla dla danych dla których wygenerowany został wykres wynosi około 1,4.

3.3.1 Sposób wygenerowania wykresu

Dane wygenerowane przez program w C++ zapisane zostały w formacie .csv, następnie wczytane przez skrypt w pythonie (main.py) i przy użyciu biblioteki matplotlib wygenerowany został wykres czasu trwania algorytmów Jacobiego i Gaussa-Seidla od liczby niewiadomych.

3.4 F

Omówione wyżej metody znacząco różnią się czasem działania rosnącym wraz z liczbą niewiadomych - dla metody korzystającej z faktoryzacji LU tempo wzrostu wynosi $O(n^3)$, zaś dla metod iteracyjnych $O(n^2)$, przy czym metoda Gaussa-Seidla jest ok. 1.4 raza szybsza od metody Jacobiego (dla rozpatrywanych wyżej danych). Wadą rozwiązań iteracyjnych jest brak pewności czy zbiegną one do rozwiązania, zaś zdecydowaną zaletą metody wykorzystującej faktoryzację LU jest (w odpowiednich warunkach) pewność co do znalezienia rozwiązania. Poniższy wykres przedstawia czas działania omawianych metod w zależności od liczby niewiadomych.



Można z niego wywnioskować , że dla większych układów równań metody iteracyjne mogą znaleźć rozwiązanie w sensownym czasie, w przeciwieństwie do metody z faktoryzacją LU, która dla dużych układów równań może działać tak długo, że jej przydatność staje pod znakiem zapytania.