Kacper Kula

Zadanie numeryczne nr 5

Omówienie

Problem polega na rozwiązaniu układu macierzowego $Ay = b$, korzystając z metody Jacobiego i Gaussa-Seidela, wykorzystując przy tym charakterystyczną budowę macierzy, aby osiągnąć jak najlepszą złożoność naszych obliczeń i porównanie wykorzystanych metod.

Macierz A jest zdefiniowana następująco:

$$A = \begin{bmatrix} 3 & 1 & 0.15 & 0 & \dots & 0 & 0 & 0 & 0 \\ 1 & 3 & 1 & 0.15 & \dots & 0 & 0 & 0 & 0 \\ 0.15 & 1 & 3 & 1 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0.15 & 1 & 3 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 3 & 1 & 0.15 & 0 \\ 0 & 0 & 0 & 0 & \dots & 1 & 3 & 1 & 0.15 \\ 0 & 0 & 0 & 0 & \dots & 0.15 & 1 & 3 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0.15 & 1 & 3 \end{bmatrix}$$

Wektor b zdefiniowany jest następująco:
$b = (1, 2, 3, \dots, N-1, N)^T$

Wymiar macierzy i wektora ustalony jest na N = 124.

Narzędziem do napisania programu rozwiązującego zadanie jest język Python oraz biblioteka pythona służąca do tworzenia wykresów – matplotlib.
Do sprawdzenia poprawności programu wykorzystana została biblioteka z zakresu algebry liniowej - numpy.

Rozwiązanie układu macierzowego

Wstęp

Macierze, w których liczba miejsc niezerowych rośnie wolniej niż wymiar macierzy, nazywamy macierzami rzadkimi.
Wykorzystując odpowiednie algorytmy do operowania na macierzach rzadkich, możemy zaoszczędzić czas wykonania i pamięć potrzebną na obliczenia.

Metoda Jacobiego

Metoda Jacobiego jest metodą iteracyjną i pozwala nam obliczać układy macierzowe $Ay=b$. Wektorem startowym nazywamy wektor y_0, który wybieramy i od którego zaczynamy nasze obliczenia.

Kolejne przybliżenia wektora $y^{(k+1)}$ obliczane są z wzoru $y_i^{(k+1)}=\frac{1}{a_{ii}}*(b_i-\sum_{j\neq i}^{n} a_{ij}*y_j^{(k)})$

Metoda Gaussa-Seidela

Metoda Gaussa-Seidela jest kolejną metodą iteracyjną stosowaną do rozwiązywania układów macierzowych, których macierz główna jest macierzą przekątnie dominującą.

Kolejne przybliżenia wektora $y^{(k+1)}$ obliczane są z wzoru $y_i^{(k)}=\frac{1}{a_{ii}}*(b_i-\sum_{j=1}^{i-1} a_{ij}*y_j^{(k+1)}-\sum_{j=i+1}^{n} a_{ij}*y_j^{(k)})$

Porównanie i sprawdzenie wyników

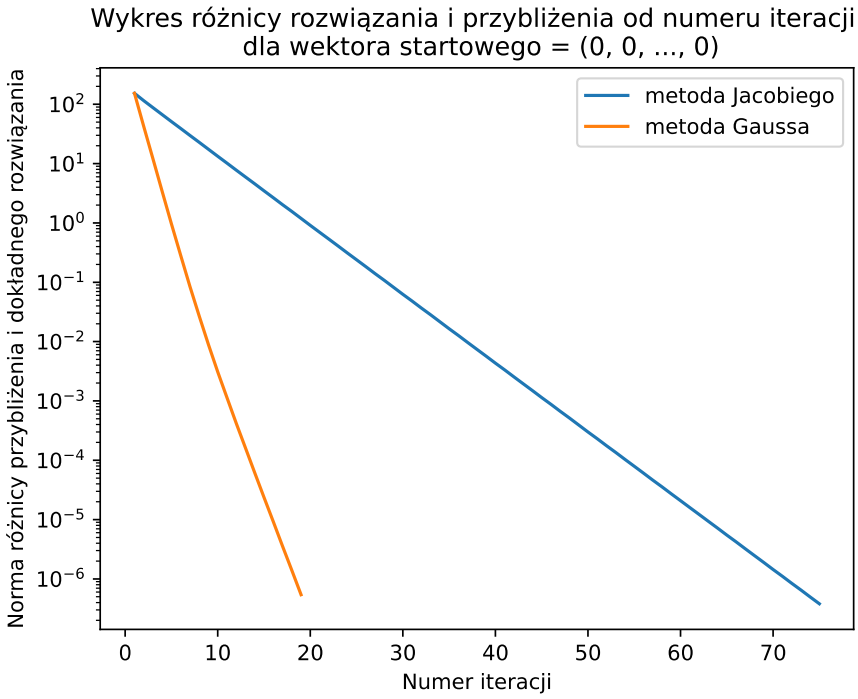
Przy pomocy funkcji linalg.solve() z biblioteki numpy obliczyłem układ macierzowy $Ay=b$, a następnie porównałem wynik z wektorami, które otrzymałem stosując opisane metody, z dokładnością do sześciu miejsc po przecinku.
Wektory okazały się równe.

Wektor wynikowy zarówno dla metody Jacobiego i Gaussa-Seidela ma postać:
y = (0.17802, 0.38116, 0.56528, 0.75477, 0.94342, 1.13206, 1.32076, 1.50943, 1.69811, 1.88679, 2.07547, 2.26415, 2.45283, 2.64151, 2.83019, 3.01887, 3.20755, 3.39623, 3.58491, 3.77358, 3.96226, 4.15094, 4.33962, 4.5283 , 4.71698, 4.90566, 5.09434, 5.28302, 5.4717 , 5.66038, 5.84906, 6.03774, 6.22642, 6.41509, 6.60377, 6.79245, 6.98113, 7.16981, 7.35849, 7.54717, 7.73585, 7.92453, 8.11321, 8.30189, 8.49057, 8.67925, 8.86792, 9.0566 , 9.24528, 9.43396, 9.62264, 9.81132, 10.00000 , 10.18868, 10.37736, 10.56604, 10.75472, 10.9434 , 11.13208, 11.32075, 11.50943, 11.69811, 11.88679, 12.07547, 12.26415, 12.45283, 12.64151, 12.83019, 13.01887, 13.20755, 13.39623, 13.58491, 13.77358, 13.96226, 14.15094, 14.33962, 14.5283 , 14.71698, 14.90566, 15.09434, 15.28302, 15.4717 , 15.66038, 15.84906, 16.03774, 16.22642, 16.41509, 16.60377, 16.79245, 16.98113, 17.16981, 17.35849, 17.54717, 17.73585, 17.92453, 18.11321, 18.30189, 18.49057, 18.67925, 18.86792, 19.0566 , 19.24528, 19.43396, 19.62264, 19.81132, 20.00000 , 20.18868, 20.37736, 20.56604, 20.75472, 20.9434 , 21.13208, 21.32075, 21.50944, 21.69809, 21.88687, 22.07543, 22.26307, 22.46032, 22.61338, 22.87516, 23.23287, 21.06156, 33.15117)^T

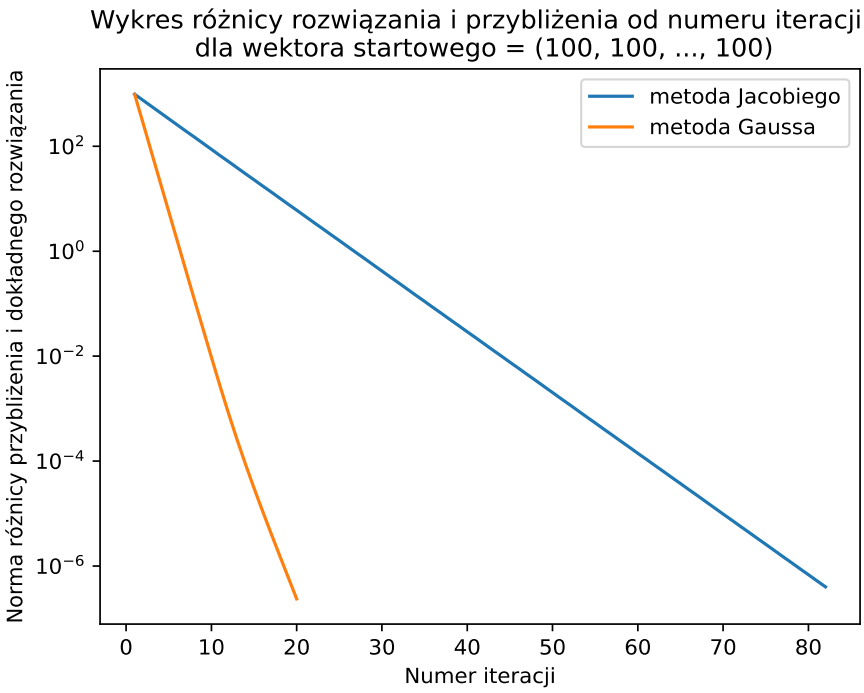
Liczba iteracji potrzebnych do osiągnięcia dokładnego rozwiązania różniła się jednak zależnie od wyboru wektora startowego v. Poniżej znajdują się wykresy dla różnych wektorów początkowych.

Warunkiem zakończenia liczenia kolejnych przybliżeń jest $\left\| y^{(k)} - y^{(k-1)} \right\| \leqslant 0.000001.$

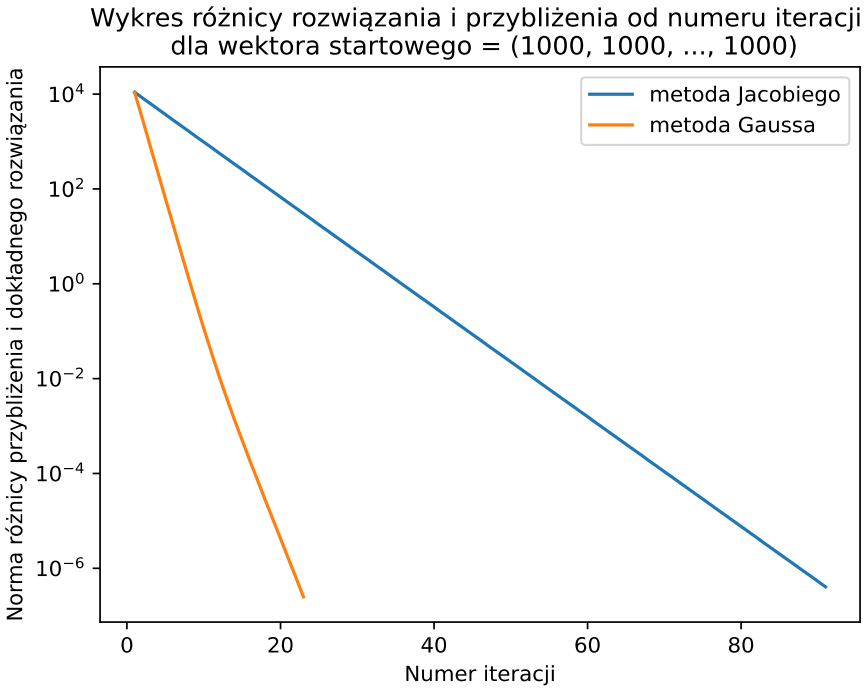
Wykres dla wektora startowego $v_0 = (0, 0, \ldots, 0)^T$



Wykres dla wektora startowego $v_0 = (100, 100, \ldots, 100)^T$



Wykres dla wektora startowego $v_0 = (1000, 1000, \ldots, 1000)^T$



Wnioski

Metoda Gaussa-Seidela okazała się efektywniejsza i osiągnięcie dokładnego rozwiązania następowało znacznie szybciej niż dla metody Jacobiego w wszystkich przypadkach.

Dla bardzo dużych wartości początkowych, błąd $\left\| y^{(k)} - y^{(*)} \right\|$, (gdzie $y^{(*)}$ jest dokładnym rozwiązaniem układu $Ay = b$, a $y^{(k)}$ jest aktualnym przybliżeniem) na początku był największy.

Wraz z wzrostem wartości początkowych, liczba potrzebnych iteracji do osiągnięcia dokładnego wyniku również rosła. Stosowana była norma euklidesowa.

Dla wektora startowego $v = (0, 0, 0, \ldots, 0, 0)^T$ liczba potrzebnych iteracji była mniejsza, niż dla innych wektorów startowych.

Czas działania programu

Czas mierzony był jedynie podczas korzystania z metody Jacobiego i Gaussa-Seidela dla wektora startowego postaci: $v = (1, 1, 1, \ldots, 1, 1)^T$ i dokładności $\left\| x_k - x_* \right\| \leqslant 0.000001$.

Średni czas wykonania programu dla N = 124 wyniósł: 8.5773 milisekundy. Przeprowadzone zostało 100 testów i wyciągnięta średnia czasu.

Program wykonałem również dla N = 10, 20, 30, …, 1000.
Dla otrzymanych rezultatów sporządziłem wykres zależności czasu działania programu od parametru N.