Kacper Kula

Zadanie numeryczne nr 4

Omówienie

Problem polega na rozwiązaniu układu macierzowego $Ay=b$, wybierając odpowiednią, czyli najbardziej optymalną metodę, wykorzystując przy tym charakterystyczną budowę macierzy.

Macierz A jest zdefiniowana następująco:

$$A=\begin{bmatrix} 12 & 8 & 1 & 1 & ... & 1 & 1 & 1 & 1 \\ 1 & 12 & 8 & 1 & ... & 1 & 1 & 1 & 1 \\ 1 & 1 & 12 & 8 & ... & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 12 & ... & 1 & 1 & 1 & 1 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ 1 & 1 & 1 & 1 & ... & 12 & 8 & 1 & 1 \\ 1 & 1 & 1 & 1 & ... & 1 & 12 & 8 & 1 \\ 1 & 1 & 1 & 1 & ... & 1 & 1 & 12 & 8 \\ 1 & 1 & 1 & 1 & ... & 1 & 1 & 1 & 12 \end{bmatrix}$$

Wektor b zdefiniowany jest następująco:
b = $(5,5,5,...,5)^T$

Wymiar macierzy i wektora ustalony jest na N = 80.

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

Przekształcanie macierzy

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.

Macierz A nie posiada elementów zerowych, jednak możemy ją rozbić na dwie macierze, z czego jedna z powstałych będzie macierzą rzadką. Zróbmy to następująco:

$$A=\begin{bmatrix} 11 & 7 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 11 & 7 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 11 & 7 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 11 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 11 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 11 & 7 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 11 & 7 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 11 \end{bmatrix}+\begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \end{bmatrix}$$

Jednak macierz wypełnioną jedynkami możemy przedstawić jako następujący iloczyn:

$$u*v^t=\begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \end{bmatrix}*\begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

Macierz rzadką powstałą z macierzy A oznaczmy A'.
Macierz A' jest typem macierzy wstęgowej lub pasmowej, czyli taką, której wszystkie elementy są zerami oprócz jej diagonali i elementów wokół niej.
Podsumowując, macierz A możemy wyrazić wzorem $A'+u*v^T$.

Efektywne rozwiązanie

Wykorzystując wzór Shermana-Morrisona możemy z łatwością obliczać równania postaci:
$(A'+u*v^T)*x=b$

Pomnóżmy macierz przez jej odwrotność, doprowadzi nas do to postaci:
$x=(A'+u*v^T)^{-1}*b$

Wzór Shermana-Morrisona ma postać:

$$(A'+u*v^T)^{-1}=(A')^{-1}-\frac{(A')^{-1}*u*v^T*(A')^{-1}}{1+v^T*(A')^{-1}*u}$$

Podstawiając, mamy więc:

$$x=(A')^{-1}*b-\frac{(A')^{-1}*u*v^T*(A')^{-1}*b}{1+v^T*(A')^{-1}*u}$$

Wzór możemy uprościć jeszcze bardziej. Niech $z=(A')^{-1}*b$ i $y=(A')^{-1}*u$, mamy więc:

$$x=z-\frac{y*(v^T*z)}{1+v^T*y}$$

Wektory z i y możemy z łatwością wyliczyć, przekształcając podstawienia i rozwiązując układy macierzowe:

$$A'*z=b$$
$$A'*y=u$$

Zauważmy również, że $a=v^T*z$ i $b=v^T*y$ są liczbami, co dodatkowo upraszcza nasz wzór do postaci.

$$x=z-\frac{a}{1+b}*y$$

Dla ogólnego przypadku złożoność czasowa wyliczania układów macierzowych metodą „forward substitution” wynosi $O(n^2)$, jednak wykorzystując fakt, że macierz A' jest pasmowa, możemy osiągnąć złożoność $O(n)$.

Operacje na wektorach, które wykonujemy są również złożoności czasowej $O(n)$, co daje nam całkowitą złożoność czasową programu $O(n)$.

Dodatkowo, złożoność pamięciowa programu również będzie wynosić $O(n)$, ponieważ potrzebne dane będziemy przechowywać w skończonej liczbie tablic o długości N.

Porównanie i sprawdzenie wyników

Przy pomocy funkcji linalg.solve() z biblioteki numpy obliczyłem układ macierzowy $Ay = b$, a następnie porównałem wynik z wektorem, który otrzymałem stosując opisane metody, z dokładnością do siedmiu miejsc po przecinku.
<u>Wektory okazały się równe.</u>

Wektor wynikowy ma postać:
y = (0.0508187, 0.0508188, 0.0508187, 0.0508188, 0.0508187, 0.0508188, 0.0508187, 0.0508188, 0.0508186, 0.050819 , 0.0508183, 0.0508194, 0.0508178, 0.0508203, 0.0508163, 0.0508226, 0.0508127, 0.0508282, 0.0508039, 0.0508421, 0.050782 , 0.0508765, 0.050728 , 0.0509614, 0.0505946, 0.0511709, 0.0502653, 0.0516885, 0.0494521, 0.0529664, 0.0474439, 0.0561221, 0.0424849, 0.0639148, 0.0302393, 0.0831579)^T

Czas działania programu

Średni czas wykonania programu dla N = 80 wyniósł: 0.0426 milisekundy. Przeprowadzone zostało 100 testów i wyciągnięta średnia czasu.

Program wykonałem również dla N = 100, 200, 300, ..., 10000.
Dla otrzymanych rezultatów sporządziłem wykres zależności czasu działania programu od parametru N.

Spodziewaliśmy się zależności czasowej O(n) - tę zależność potwierdza nam widziana linia prosta na wykresie, po której rozkładają się wyniki poszczególnych testów.