SPRAWOZDANIE
Z CZWARTEGO ZADANIA NUMERYCZNEGO

MACIEJ WÓJCIK

Spis treści:

1. Polecenie zadania

$$\mathbf{A} = \begin{pmatrix} 10 & 8 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \\ 1 & 10 & 8 & 1 & \dots & 1 & 1 & 1 & 1 \\ 1 & 1 & 10 & 8 & \dots & 1 & 1 & 1 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & 1 & \dots & 1 & 10 & 8 & 1 \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 10 & 8 \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 10 \end{pmatrix}$$

oraz wektor $\mathbf{b} \equiv (5, \dots, 5)^T$. Macierz $\mathbf{A}$ ma liczby 10 na diagonali, 8 na pierwszej pozycji nad diagonalą, a pozostałe elementy są równe 1. Wymiar macierzy ustalamy na $N = 50$.

- Rozwiąż numerycznie równanie $\mathbf{Ay} = \mathbf{b}$, dobierając odpowiedni algorytm. **Uwaga:** algorytm należy go zaimplementować samodzielnie (mile widziane jest jednak sprawdzenie wyniku przy użyciu procedur bibliotecznych lub pakietów algebry komputerowej).
- Potraktuj N jako zmienną i zmierz czas potrzebny do uzyskania rozwiązania w funkcji N. Wynik przedstaw na wykresie. Czy wykres jest zgodny z oczekiwaniami?

W zadaniu musimy efektywnie rozwiązać równanie **Ay = b.** Gdzie macierz A możemy przedstawić w innej formie dla usprawnienia obliczeń. Wektor b jest znany. Szukamy więc wektora y.

2. Instrukcja uruchomienia

Aby uruchomić program należy skorzystać z polecenia:

make run

Lub

python3 NUM4.py

4. Manipulacja macierzą

Wykonywanie obliczeń na zwykłej macierzy jest bardzo czasochłonne, dlatego żeby efektywnie wykonywać obliczenia na naszej macierzy A, musimy ją jakoś uprościć.

W naszym przypadku bardzo dobrze sprawdzi się rozłożenie macierzy A na sumę dwóch macierzy w następujący sposób:
(Dla uproszczenia przedstawione dla macierzy o rozmiarze 5 x 5)

$$\begin{bmatrix} 10 & 8 & 1 & 1 & 1 \\ 1 & 10 & 8 & 1 & 1 \\ 1 & 1 & 10 & 8 & 1 \\ 1 & 1 & 1 & 10 & 8 \\ 1 & 1 & 1 & 1 & 10 \end{bmatrix} = \begin{bmatrix} 9 & 7 & 0 & 0 & 0 \\ 0 & 9 & 7 & 0 & 0 \\ 0 & 0 & 9 & 7 & 0 \\ 0 & 0 & 0 & 9 & 7 \\ 0 & 0 & 0 & 0 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Przedstawiamy macierz A za pomocą sumy dwóch macierzy, z których jedna ma elementy o 1 mniejsze od macierzy A, a druga składa się z samych jedynek.

Przekształcenie to pozwoli nam zapisać macierz A jako macierz wstęgową z dwoma diagonalami, a drugą macierz jako iloczyn wektora złożonego z samych jedynek i transponowanego wektora złożonego z samych jedynek.

Wszystkie te przekształcenia zbliżają nas do rozwiązania naszego problemu przy użyciu wzoru Shermana-Morrisona.

Oznaczając macierz wstęgową jako B i wektory poprzez u i v, macierz A jest następująca:

$$A = B + ut^T$$

Korzystając z tych oznaczeń zapisujemy nasze równanie:

$$Ay = b <=> (B + uv^T)y = b$$

A następnie wyznaczamy y mnożąc lewostronnie przez A^{-1}:

$$y = (B + uv^T)^{-1}b$$

Teraz korzystając ze wzoru Shermana-Morrisona otrzymujemy:

$$y = B^{-1}b - \frac{B^{-1}uv^TB^{-1}}{1 + v^TB^{-1}u}b$$

Dla uproszczenia wyglądu tego wyniku możemy przyjąć: $B^{-1}b = z$
i $B^{-1}u = x$

$$y = z - \frac{xv^Tz}{1 + v^Tx}$$

Jeśli zwrócimy uwagę na to, że iloczyn $(wektor * wektor^T)$ daje nam liczbę i odniesiemy to do naszego przypadku to możemy wywnioskować, że skoro wektor v zawiera same jedynki, to wynik v^Tz będzie równy sumie wszystkich elementów wektora z.

To samo możemy zastosować do mnożenia v^Tx, które da nam sumę elementów wektora x.

Stosując to co wymieniłem wyżej otrzymujemy ostateczny wzór:

$$y = z - x\frac{suma\,.\,elementow(z)}{1 + suma\,.\,elementow(x)}$$

Do rozwiązania naszego wzoru potrzebujemy jeszcze obliczyć: z, x

Wzory $B^{-1}b = z$ i $B^{-1}u = x$ przekształcamy mnożąc lewostronnie przez B i dostajemy:

- $Bz = b$
- $Bx = u$

Macierz B jest trójkątnie górna, więc możemy od razu zastosować metodę backward substitution bez przeprowadzania rozkładu.

Dla $Bz = b$:

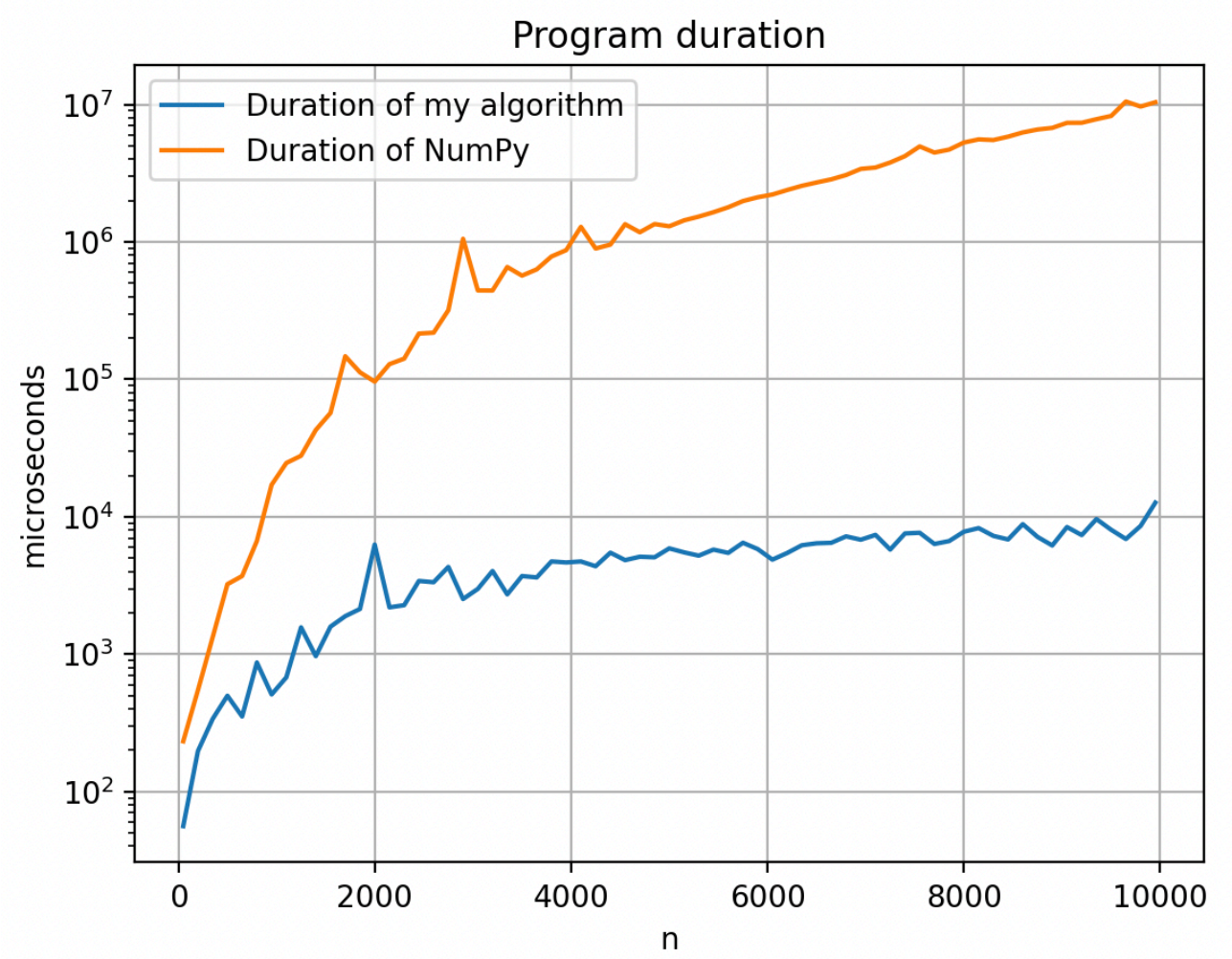
$$z_n = \frac{b_n - B_{1,n} z_{n+1}}{B_{0,n}}, \qquad \text{dla } n_{max} = \text{wymiar macierzy - 1}$$

Dla $Bx = u$:

$$x_n = \frac{1 - B_{1,n} x_{n+1}}{B_{0,n}}, \qquad \text{dla } n_{max} = \text{wymiar macierzy - 1}$$

5. Wyniki

y =
[0.07525844089350037, 0.07525904117533852, 0.07525826938440369,
0.07525926168703423, 0.07525798586936636, 0.07525962620636797,
0.07525751720165161, 0.07526022877914401, 0.07525674246522518,
0.07526122486883524, 0.07525546177847939, 0.07526287146607977,
0.07525334472487927, 0.07526559339213704, 0.07524984510566277,
0.07527009290255826, 0.07524406002083556, 0.07527753086876468,
0.07523449692142720, 0.07528982628228972, 0.07521868853260927,
0.07531015135362709, 0.07519255629803279, 0.07534374994093965,
0.07514935811434514, 0.07539929046282379, 0.07507794887192268,
0.07549110234593842, 0.07495990502220382, 0.07564287300986267,
0.07476477131144413, 0.07589375920941080, 0.07444220334059656,
0.07630848945764337, 0.07390897873572605, 0.07699406394961972,
0.07302752581747077, 0.07812736055880520, 0.07157043017708939,
0.08000076923929544, 0.06916176187360190, 0.08309762848663654,
0.065180085698449080, 0.088216926426118720 0.058598131204829124,
0.096679439346487260, 0.04771775745006959, 0.11066849131689238,
0.029731833488120224, 0.13379325069654147]



6. Wnioski

Wyniki policzone przy użyciu zaimplementowanego algorytmu wychodzą takie same jak przy zastosowaniu biblioteki NumPy.

Jeśli spojrzymy na wykres czasu trwania obydwóch algorytmów widzimy, że algorytm dobrany do struktury macierzy działa znacznie szybciej niż ogólny sposób rozwiązywania. Udowadnia to, że dobranie odpowiedniego algorytmu do macierzy ma ogromne znaczenie dla szybkości działania programu.