Sprawozdanie z zadania numerycznego 4

1. Instrukcja uruchomienia

- Aby uruchomić program należy użyć *python NUM4.py*

2. Cel ćwiczenia

Efektywne rozwiązanie równania $Ay = b$, gdzie A jest macierzą mającą liczby 10 na głównej diagonali i liczby 8 na diagonali znajdującej się nad główną.

3. Opis ćwiczenia

1) Przekształcenie polecenia i wyprowadzenie wzorów
2) Rozbicie problemu na mniejsze
3) Podstawianie wstecz
4) Złożenie wyniku

4. Wstęp teoretyczny

Zacznijmy od omówienia problemu przedstawionego w zadaniu. Mamy macierz która nie jest macierzą gęstą, obliczenie jej standardowymi metodami osiągnęło by durzą złożoność. My natomiast szukamy rozwiązania, które będzie działać najlepiej w czasie liniowym.

Jeśli sprowadzilibyśmy macierz A do macierzy rzadkiej, a jeszcze lepiej jakby ta macierz miała tylko niezerowe elementy na diagonalach to obliczenia znacznie by się skróciły. Naszą macierz można tak przekształcić ponieważ składa się praktycznie z samych jedynek nie licząc oczywiście diagonali. Wystarczy więc że obniżymy wartość każdego elementu macierzy o jeden i dostaniemy wtedy macierz rzadką z dwoma diagonalami.

$$A = \begin{pmatrix} 10 & 8 & 1 & & & & \\ 1 & 10 & 8 & \cdots & & 1 & \\ 1 & 1 & 10 & & & & \\ & \vdots & & \ddots & & \vdots & \\ & & & & 10 & 8 & 1 \\ & 1 & & \cdots & 1 & 10 & 8 \\ & & & & 1 & 1 & 10 \end{pmatrix} = \begin{pmatrix} 9 & 7 & 0 & & & & \\ 0 & 9 & 7 & \cdots & & 0 & \\ 0 & 0 & 9 & & & & \\ & \vdots & & \ddots & & \vdots & \\ & & & & 9 & 7 & 0 \\ & 0 & & \cdots & 0 & 9 & 7 \\ & & & & 0 & 0 & 9 \end{pmatrix} + \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix}$$

Oczywisty faktem jest że każda z tych macierzy to macierz N na N.

Widać już że do przechowywania naszej macierzy rzadkiej możemy użyć podobnie jak w zadaniu 3 macierzy wstęgowej.

$$\begin{pmatrix} 9 & 7 \\ 9 & 7 \\ \vdots & \vdots \\ 9 & 7 \\ 9 & 0 \end{pmatrix}$$

Pozostaje nam jeszcze drugi składnik sumy, czyli macierz zbudowana z samych jedynek. Możemy ją zamienić na mnożenie wektora zbudowanego z samych jedynek razy wektor transponowany też zbudowany z samych jedynek.

Jeśli oznaczymy sobie naszą macierz rzadką poprzez B a nasze wektory poprzez u i v, otrzymamy: $A = B + uv^T$. Pozwala nam to zapisać nasze równanie w postaci:

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

Celowo tutaj zapisujemy to jako macierz odwrotną aby móc skorzystać ze wzoru który znacznie uprości i przyspieszy nam rachunki. Wzór ten jest wzorem Shermana-Morrisona i wygląda on następująco:

$$(B + uv^T)^{-1} = B^{-1} - \frac{B^{-1}uv^TB^{-1}}{1 + v^TB^{-1}u}$$

Czyli nasze równanie będzie miało postać:

$$Ay = b \Leftrightarrow y = B^{-1}b - \frac{B^{-1}uv^TB^{-1}b}{1 + v^TB^{-1}u} \Leftrightarrow z - \frac{z'v^Tz}{1+ v^Tz'}$$

Gdzie $z = B^{-1}b$ oraz $z' = B^{-1}u$. Aby w łatwy sposób rozwiązać nasze zadanie wystarczy rozwiązać dwa układy równań:

$$\begin{cases} z = B^{-1}b \Leftrightarrow Bz = b \\ z' = B^{-1}u \Leftrightarrow Bz' = u \end{cases}$$

Macierz B jest macierzą trójkątną górną, więc nie musimy już przeprowadzać rozkładu. Wystarczy zastosować metodę backward substitution. Gdy uwzględnimy budowę macierzy B i to że elementy wektora u są jedynkami algorytm ten będzie działał w czasie O(n) i otrzymamy następujące wzory:

- Dla pierwszego układu

$$z_{49} = \frac{b_{49}}{B_{0_{49}}}$$

Dla n < 49: $z_n = \frac{b_n - B_{1_n}z_{n+1}}{B_{0_n}}$

- Dla drugiego układu

$$z'_{49} = \frac{1}{B_{0_{49}}}$$

Dla n < 49: $$z'_n = \frac{1 - B_{1_n} z'_{n+1}}{B_{0_n}}$$

B_0 - główna diagonala

B_1 - górna diagonala

Zajmijmy się jeszcze ostatnimi działaniami:

$$z - \frac{z' v^T z}{1 + v^T z'} = z - z' \frac{v^T z}{1 + v^T z'} = \frac{\text{sum}(z)}{1 + \text{sum}(z')}$$

sum(w) – suma elementów wektora w

Wektor transponowany razy wektor daje nam liczbę, a gdy v zawiera tylko jedynki to wynikiem takiego mnożenia będzie suma elementów drugiego wektora.

5. Wyniki

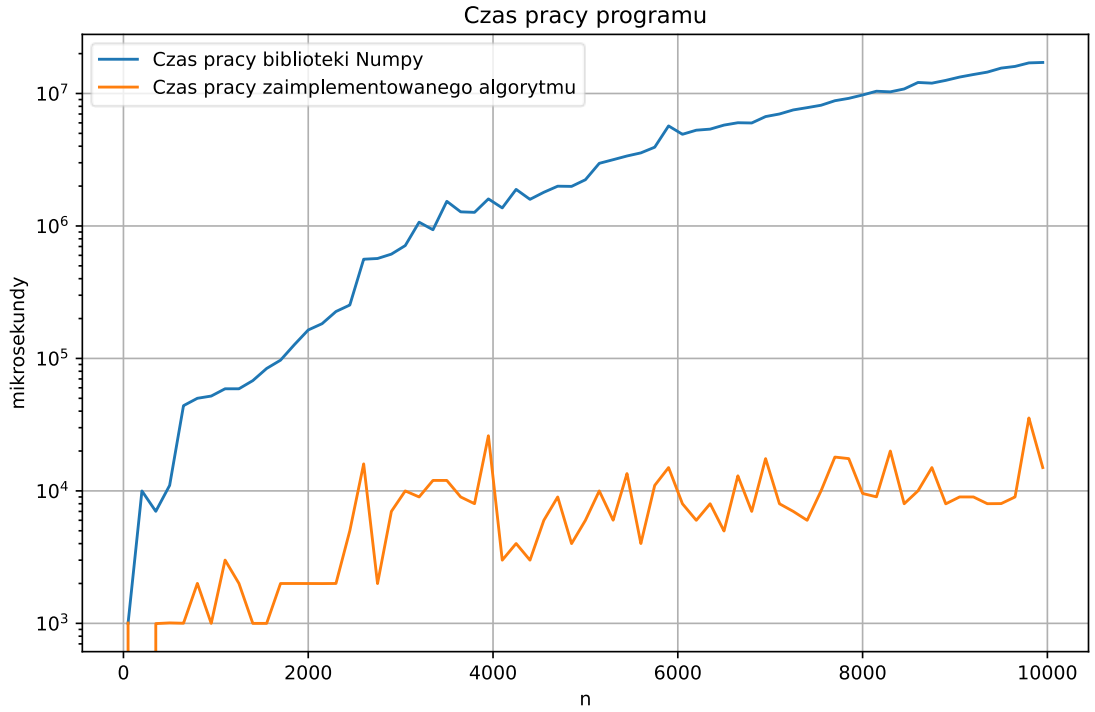
Rozwiązaniem równania Ay = b jest wektor:

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, 0.0752344969214272,
0.07528982628228972, 0.07521868853260927, 0.07531015135362709, 0.07519255629803279,
0.07534374994093965, 0.07514935811434514, 0.07539929046282379, 0.07507794887192268,
0.07549110234593842, 0.07495990502220382, 0.07564287300986267, 0.07476477131144413,
0.0758937592094108, 0.07444220334059656, 0.07630848945764337, 0.07390897873572605,
0.07699406394961972, 0.07302752581747077, 0.0781273605588052, 0.07157043017708939,
0.08000076923929544, 0.0691617618736019, 0.08309762848663654, 0.06518008569844908,
0.08821692642611872, 0.058598131204829124, 0.09667943934648726, 0.04771775745006959,
0.11066849131689238, 0.029731833488120224, 0.13379325069654147]^T

6. Wnioski

Wyniki programu są prawie identyczne w porównaniu do wyników które otrzymałem obliczając to samo zadanie z użyciem biblioteki Numpy. Różnicę w tych wynikach można zauważyć dopiero na 15 miejscu po przecinku co prawdopodobnie spowodowane jest błędem wynikającym z precyzji obliczeń. Równanie macierzowe udało się obliczyć również w czasie liniowym O(n) dzięki zastosowaniu wzoru

Shermana-Morrisona i backward substitution. Własnoręczna implementacja działa więc znacznie krócej w porównaniu do czasu pracy metod z biblioteki Numpy.



Widać że czas pracy zaimplementowanego algorytmu jest liniowy natomiast czas obliczania przy użyciu biblioteki jest znacznie większy.