Problematyka:

Program ma obliczyć numerycznie równanie $Ay = b$ dla danej macierzy A oraz wektora b, stosujac samodzielnie zaimplementowany algorytm. Nastepnie przedstawić na wykresie zależność czasu trwania programu od wartości zmiennej N.

Wyjaśnienie programu:

W programie wykorzystane sa funkcje z dodatkowej biblioteki (`matplotlib`) aby w prosty sposób stworzyć wykres zależności czasu od zmiennej N.

Funkcje stworzone przeze mnie dla potrzeb rozwiazania problemu:

- `numpy_library_time(n, b)`: funkcja oblicza numerycznie równanie dane w zadaniu przy pomocy funkcji z biblioteki `Numpy` i zwraca czas trwania obliczania.
- `sherman_algorithm(n, b, print_y=False)`: funkcja oblicza numerycznie równanie dane w zadaniu przy pomocy napisanej implementacji algorytmu Shermana-Morrisona i w zależności od argumentu `print_y` albo wypisuje obliczony wektor y albo zwraca czas trwania obliczania.

Wyjaśnienie problematyki:

Mamy do czynienia z macierza, która nie jest gesta, wiec zastosowanie standardowych metod obliczeniowych byłoby bardzo złożone. Naszym celem jest znalezienie rozwiazania, które bedzie działać optymalnie w czasie liniowym.

Możemy sprowadzić macierz A do postaci rzadkiej, ponieważ składa sie praktycznie tylko z samych jedynek, nie liczac diagonali. Odejmujac od każdego elementu wartość jedynki, otrzymamy macierz rzadka z wartościami jedynie na diagonali oraz w jednym wierszu nad nia.

$$A = \begin{bmatrix} 12 & 8 & 1 & \dots & & & \\ 1 & 12 & 8 & \dots & & & \\ 1 & 1 & 12 & \dots & & & \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ & & & \dots & 12 & 8 & 1 \\ & & & \dots & 1 & 12 & 8 \\ & & & \dots & 1 & 1 & 12 \end{bmatrix} = \begin{bmatrix} 11 & 7 & 0 & \dots & & & \\ 0 & 11 & 7 & \dots & & & \\ 0 & 0 & 11 & \dots & & & \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ & & & \dots & 11 & 7 & 0 \\ & & & \dots & 0 & 11 & 7 \\ & & & \dots & 0 & 0 & 11 \end{bmatrix} + \begin{bmatrix} 1 & 1 & \dots & & \\ 1 & 1 & \dots & & \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ & & \dots & 1 & 1 \\ & & \dots & 1 & 1 \end{bmatrix}$$

Widzimy już, że do przechowywania rzadkiej macierzy można wykorzystać macierz wstegowa.

Mateusz Kałwa

$$\begin{bmatrix} 11 & 7 \\ 11 & 7 \\ \vdots & \vdots \\ 11 & 7 \\ 11 & 7 \end{bmatrix}$$

Pozostaje nam drugi element sumy, czyli macierz składajaca sie tylko z jedności. Możemy zamienić ja na iloczyn wektora złożonego z samych jedności razy transponowany wektor również zbudowany z samych jedności.

Jeśli oznaczymy nasza rzadka macierz jako B, a wektory jako u i v, otrzymamy: $A = B + uv^t$. To pozwala nam zapisać nasze równanie jako:

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

Później, możemy wykorzystać wzór Shermana-Morrisona, który ma postać:

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

Po podstawieniu naszych wartości, równanie wyglada nastepujaco:

$$Ay = b \Rightarrow y = B^{-1}b - \frac{B^{-1}uv^T B^{-1}b}{1 + v^T B^{-1}u} \Rightarrow z - \frac{z'v^T z}{1 + v^T z'}$$

Zatem $z = B^{-1}b$ oraz $z' = B^{-1}u$. Do rozwiazania naszego zadania wystarczy rozwiazać dwa układy równań:

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

Macierz B jest macierza trójkatna górna, wiec nie potrzebujemy robić rozkładu. Wystarczy zastosować metode substitucji wstecznej (backward substitution). Biorac pod uwage strukture macierzy B oraz to, że elementy wektora u sa jedynkami, algorytm ten działa w czasie $O(n)$ i otrzymujemy nastepujace wzory:

Dla pierwszego równania:

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

Dla $n < 49$:

$$z_n = \frac{b_n - B_{1,n} * z_{z+1}}{B_{0,n}}$$

Dla drugiego równania:

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

dla $n < 49$:

$$z'_n = \frac{1 - B_{1,n} * z'_{n+1}}{B_{0,n}}$$

B_0 - diagonalna, B_1 - elementy nad diagonala

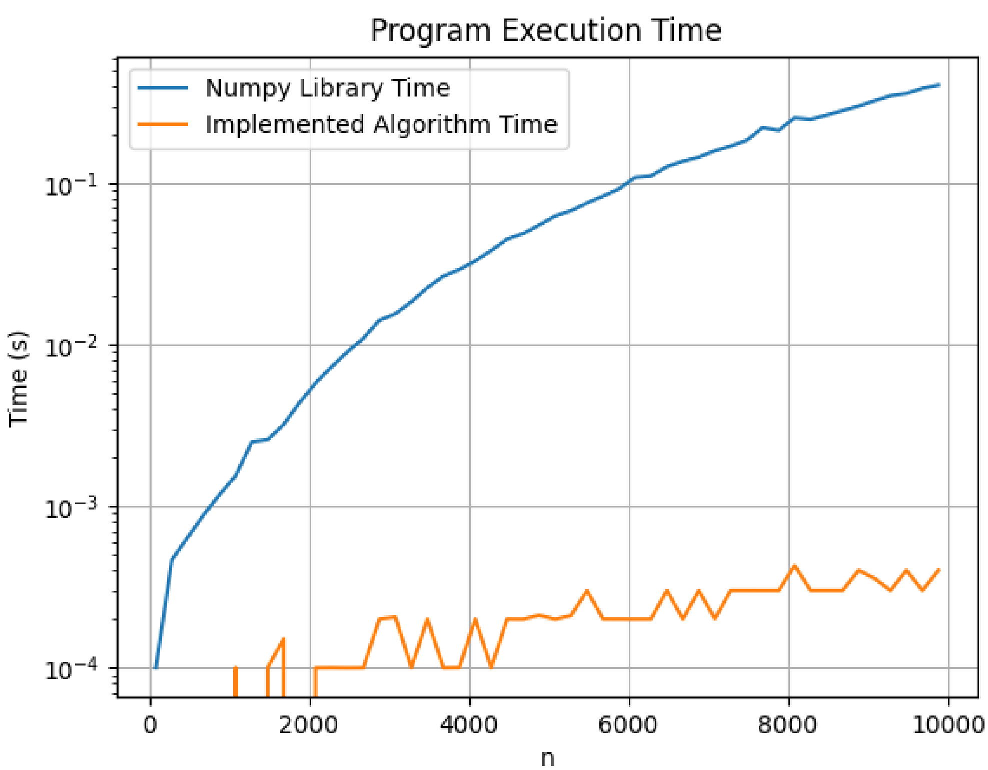
Pozostaje jeszcze ostatnia operacja:

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

Mnożenie transponowanego wektora przez wektor daje nam liczbe, a gdy v zawiera tylko jedynki, wynikiem takiego mnożenia bedzie suma elementów drugiego wektora.

Wyniki programu:

Wykres zależności czasu dla implementacji z biblioteki NumPy oraz własnej implementacji od wartości N, gdzie n przyjmuje wartości od 80 do 10000, przeprowadzajac każde obliczenie 10 razy:



Obliczony wektor y dla $N = 80$:

[0.05081874647092044, 0.050818746470920495, 0.05081874647092047, 0.05081874647092052, 0.05081874647092041, 0.05081874647092058, 0.050818746470920356, 0.05081874647092066, 0.050818746470920106, 0.05081874647092108, 0.050818746470919524, 0.05081874647092194, 0.05081874647091819, 0.050818746470924075, 0.050818746470914805, 0.050818746470929294, 0.050818746470906534, 0.05081874647094228, 0.05081874647088616, 0.05081874647097445, 0.050818746470835674, 0.05081874647105364, 0.050818746470711135, 0.05081874647124948, 0.050818746470403436, 0.050818746471732956, 0.05081874646964374, 0.05081874647292675, 0.050818746467767656, 0.05081874647587489, 0.05081874646313486, 0.05081874648315496, 0.05081874645169479, 0.050818746501132245, 0.050818746423444944, 0.0508187465455249, 0.05081874635368483, 0.05081874665514788, 0.05081874618142024, 0.05081874692584937, 0.050818745756032235, 0.05081874759431626, 0.05081874470558426, 0.05081874924502022, 0.05081874211162085, 0.050818753321248494, 0.05081873570611922, 0.05081876338703667, 0.05081871988845221, 0.05081878824337052, 0.05081868082849891, 0.05081884962329722, 0.05081858437432846, 0.050819001194136515, 0.05081834619158099, 0.0508193754813111, 0.05081775802602087, 0.050820299741476976, 0.05081630561718872, 0.05082258209821314, 0.05081271905660334, 0.05082821812199029, 0.05080386244781074, 0.05084213565009288, 0.05078199204650674, 0.05087650342357064, 0.050727985545327314, 0.05096137078256685, 0.050594622552619095, 0.05117094119967974, 0.050265297611441606, 0.05168845182153001, 0.04945206663424831, 0.05296638621426247, 0.047443884017097315, 0.05612210175549964, 0.04248490245229605, 0.06391478707161591, 0.03023925409839895, 0.08315794877059712]

Wnioski:

Wyniki programu prawie nie różnia sie od tych uzyskanych przy obliczaniu tego samego zadania za pomoca biblioteki NumPy. Dzieki zastosowaniu wzoru Shermana-Morrisona oraz backward substitution udało sie rozwiazać równanie macierzowe w czasie liniowym $O(n)$. Jest to znaczaco szybsze niż korzystanie z funkcji w bibliotece NumPy.

Uruchamianie programu:

`make run`: uruchamia program, który wypisuje obliczony wektor y oraz po dłuższym czasie wyświetla wykres zależności czasu trwania programu od wartości zmiennej N.

W razie problemów można zmienić w pliku `Makefile` wywołanie Pythona z "python" na "python3".

Mateusz Kałwa