NUM4

Cel:

Obliczenie równania $Ay = b$ biorąc pod uwagę fakt, iż macierz A ma liczby 12 na diagonali, 8 na pierwszej pozycji nad diagonalą, a pozostałe elementy są równe 1.

Opis:

- Importowanie bibiliotek matplotlib oraz time
- Zdefiniowanie funkcji implementującej formułe Shermana Morrisona
- Zdefiniowanie funkcji, która będzie mierzyła czas wykonania funkcji Shermana Morrisona i zwracała listę średnich czasów wykonania dla każdego różnych N.
- Zdefiniowanie funkcji, która stworzy wykres do wizualizacji złożoności algorytmu Shermana Morrisona dla naszej macierzy A za pomocą wcześniej zaimplementowanej biblioteki matplotlib.

Teoria:

Na samym początku powinniśmy się zastanowić, jak możemy ułatwić sobie życie przekształcając podaną w zadaniu macierz, tak abyśmy mogli ją policzyć w czasie liniowym a nie O(n^3).

Dostajemy macierz, która ma liczby 12 na diagonali, 8 na pierwszej pozycji nad diagonalą, a pozostałe elementy są równe 1. W tym przypadku możemy obniżyć każdy element macierzy o jeden to znaczy, że otrzymamy macierz, która ma 11 na diagonali, 7 na pierwszej pozycji nad diagonalą, a pozostałe elementy są równe 0. Najważniejszy jest fakt, iż te pozostałe elementy są zerowe, ponieważ doprowadzi to do skrócenia wielu obliczeń tak jak w przypadku poprzedniego zadania numerycznego w którym użyłem algorytmu Doolittle'a do policzenia macierzy rzadkiej.

Dzięki odpowiedniemu rozłożeniu macierzy A, dwa równania rozwiązujemy, ze złożonością O(n) a nie O(n^3) co jest niesamowitą poprawą złożoności algorytmu.

Podobnie jak w zadaniu NUM3 do przechowywania macierzy rzadkiej, czyli tej, której większość elementów wynosi zero, użyjemy macierzy wstęgowej.

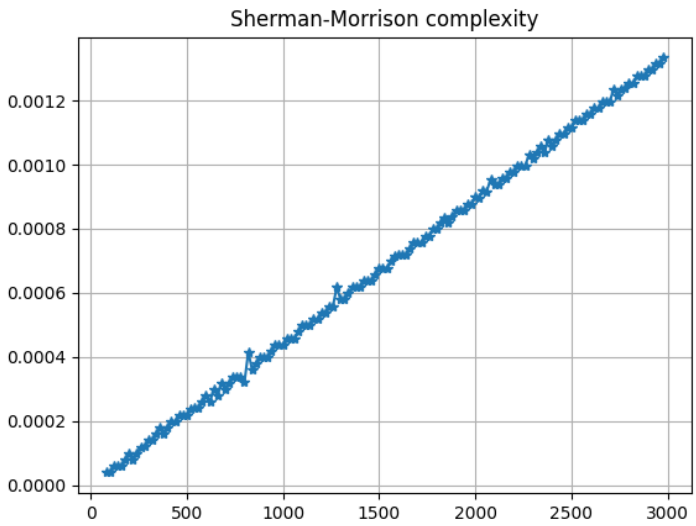
$$\begin{matrix} 11 & 7 \\ \vdots & \vdots \\ 11 & 7 \end{matrix}$$

Wyniki:

y = [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, 0.04744388401709731 5,
0.05612210175549964, 0.04248490245229605, 0.06391478707161591,
0.03023925409839895, 0.08315794877059712]

Wnioski:

Złożoność obliczeniowa tak jak widać na wykresie jest liniowa.



Wyniki dla mojej implementacji oraz tej policzonej przez wbudowaną bibliotekę NUMPY są niemal takie same (różnice pokroju 10e-15). Ponad to moja implementacja jest znacznie szybsza od tej zaproponowanej przez bibliotekę pythona.

Uruchomienie:

make run – uruchamia obliczenia numpy, mojej implementacji oraz wykres.