Zadanie numeryczne NUM4 - zastosowanie wzoru Shermana-Morrisona

Aleksander Pugowski

12-11-2022

1 Wprowadzenie

Zadanie polega na rozwiązaniu szeregu równań liniowych danych równaniem $Ay = b$, gdzie A jest macierzą zawierającą na diagonali same 10, jeden rząd nad diagonalą same 8, a reszta wartości to jedynki. Wektor wyrazów wolnych zawierał same 5.

Zdecydowano się wykorzystać algorytm Shermana-Morrisona, co umożliwiło bardzo zwinne przejście do rzadkiej macierzy, o dwóch niezerowych diagonalach, której w zasadzie (ze względu na jedynie dwie różne wartości) nie trzeba było przechowywać w pamięci, a jedynie operować na tych dwóch wartościach. Wyniki zostały zweryfikowane przy użyciu biblioteki numerycznej NumPy. Dodatkowo, zmierzono również czas działania funkcji w zależności od rozmiaru macierzy oraz porównano go z wynikami funkcji bibliotecznej.

Skorzystano więc z faktu, że macierz A można zapisać jako $A_1 = A + uv^T$, gdzie u,v to wektory zawierające same jedynki. Korzystając ze wzoru Shermana-Morrisona, możemy rozwiązać początkowe równanie poprzez rozwiązanie

$$Az = b \tag{1}$$

$$Aq = u \tag{2}$$

i wtedy

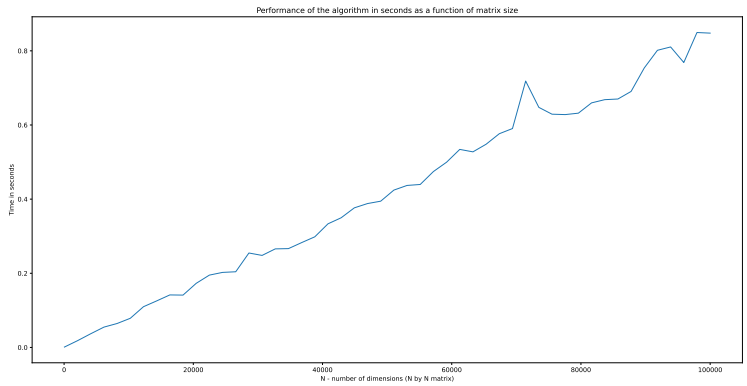
$$y = z - \frac{v^T z}{1 + v^T q} q \tag{3}$$

2 Wyniki i dyskusja

Dla macierzy A o wymiarach 50 na 50, wynikiem jest wektor o 50 współrzędnych, których wypisywanie tutaj mija się z celem. Dokładne dane można znaleźć w kodzie dołączonym do tego raportu, natomiast dwie pierwsze i dwie ostatnie współrzędne wektora y wynoszą:

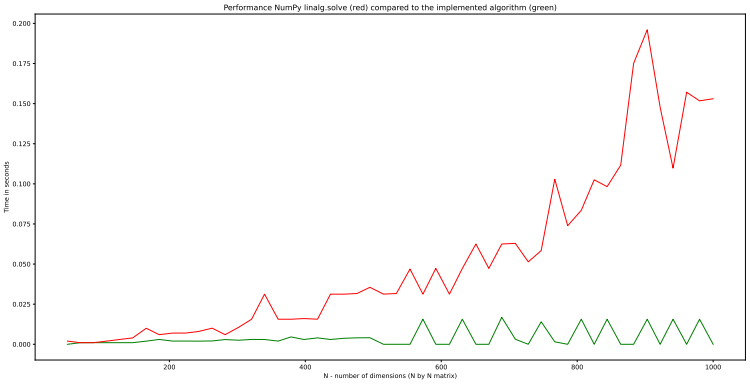
$$y = \begin{bmatrix} 0.07525844089350037 \\ 0.07525904117533852 \\ \vdots \\ 0.029731833488120224 \\ 0.13379325069654147 \end{bmatrix}$$

Wykres czasu wykonania algorytmu (równanie było rozwiązywane 10 razy - dla mniejszej ilości czas rozwiązania dla małych macierzy był rejestrowany jako 0) w zależności od rozmiaru macierzy znajduje się poniżej.



Wykres jest zgodny z oczekiwaniami (to znaczy można zaobserwować tutaj jasny trend liniowy), jednakże widać tu pewne anomalie. Program został napisany w języku Python, który ma zawiłe mechanizmy zarządzania pamięcią, co może wpływać na czas wykonania programu i niejednokrotnie prowadzi do tego typu anomalii. W związku z tym, nie jestem w stanie stwierdzić, czy anomalie te mają źródło numeryczne czy stricte implementacyjne (choć wydaje się, że chodzi jednak o implementację, o ile liczby nie zostały 'spreparowane' aby przy pewnych wartościach obliczenia się upraszczały).

Poniżej znajduje się wykres porównujący czas rozwiązywania danego układu równań zaimplementowanym algorytmem, a funkcją biblioteczną (metoda LU z częściowym pivotingiem).



Jasno widać, że wykonana implementacja jest o wiele efektywniejsza, szczególnie dla większych N. Na tym wykresie, anomalie w czasie wykonania algorytmu wydają się być o wiele bardziej systematyczne, co może sugerować że istnieje

inne (niż to trywialne) powiązanie między czasem wykonania algorytmu, a licznoscią elementów w macierzy. Ponownie, ze względu na język programowania jest to trudne w ocenie.

Po analizie tego zadania i wykresów, można zauważyć jak ważne jest odpowiednie dobranie algorytmu do problemu numerycznego. Nie tylko pozwala to zaoszczędzić czas, ale również miejsce w pamięci (co być może dzisiaj nie jest kluczowym zagadnieniem, ale dla naprawdę dużych macierzy również może mieć znaczenie). Dzięki doborze algorytmu, równanie udało się rozwiązać w czasie $O(n)$ zamiast $O(n^3)$ (np. klasyczną eliminacją Gaussa lub inną faktoryzacją). Przy niewielkich macierzach różnica nie jest duża (co widać na ostatnim wykresie), jednak (co też zresztą widać) zaczyna ona gwałtownie rosnąć dla większych N.