Sprawozdanie NUM3

Szymon Tomaszewski

01 listopada 2024

1. Wstęp

1.1. Treść zadania

Wyznacz $y = A^{-1}x$ dla

$$\mathbf{A} = \begin{pmatrix} 1.01 & \frac{0.2}{1} & \frac{0.15}{1^3} & & & & & & \\ 0.3 & 1.01 & \frac{0.2}{2} & \frac{0.15}{2^3} & & & & & \\ & 0.3 & 1.01 & \frac{0.2}{3} & \frac{0.15}{3^3} & & & & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & & & 0.3 & 1.01 & \frac{0.2}{N-2} & \frac{0.15}{(N-2)^3} \\ & & & & & & 0.3 & 1.01 & \frac{0.2}{N-1} \\ & & & & & & & 0.3 & 1.01 \end{pmatrix}$$

oraz $x = (1, 2, \dots, N)^T$. Ustalamy N=300. Oblicz również wyznacznik macierzy A. Zadanie rozwiąż właściwą metodą (uzasadnij wybór) i wykorzystaj strukturę macierzy. Algorytm proszę zaprogramować samodzielnie; wyjątkowo nie należy stosować procedur bibliotecznych z zakresu algebry liniowej ani pakietów algebry komputerowej (chyba, że do sprawdzenia swojego rozwiązania, co zawsze jest mile widziane). Ponadto, potraktuj N jako zmienną i zmierz czas działania swojego programu w funkcji N. Wynik przedstaw na wykresie. Jakiej zależności się spodziewamy?

1.2. Wprowadzenie

Zadanie polega na wyznaczeniu macierzy odwrotnej bez użycia gotowych bibliotek. Proces ten jest operacją złożoną obliczeniowo i pamięciową ale jesteśmy w stanie to zoptymalizować.

1.2.1. Pamięć

Przechowywanie macierzy rzadkiej w postaci tablicy N*N jest mało wydajnym rozwiązaniem. Możemy tutaj zauważyć, że macierz zbudowana jest z jednej wstęgi pod diagonalą, diagonali oraz dwóch wstęg nad diagonalą, a pozostałe miejsca są wypełnione zerami. Taka budowa pozwala nam zapisać tą macierz w tablicy 4*N, gdzie 4 jest związane z ilością wstęg.

1.2.2. Obliczenia

Aby zoptymalizować program ze względu na obliczenia przekształcimy równanie $y = A^{-1}x$ mnożąc lewostronnie przez A. Otrzymamy wtedy równanie postaci $Ay = x$. Teraz przedstawimy macierz A w postaci rozkładu LU gdzie są to macierze odpowiednio trójkątne dolne i górne, pozwala nam to skorzystać z forward i backward substitution. Czyli innymi słowy rozbijamy równanie na dwa równania postaci Lz=x i Uy=z . Zauważamy także, że stosując algorytm Doolittle'a:

$$U_{IJ} = A_{IJ} - \sum_{k<I} L_{IK}U_{KJ} \qquad L_{IJ} = \frac{A_{IJ} - \sum_{k<I} L_{IK}U_{KJ}}{U_{JJ}}$$

wiedząc, że większość macierzy jest wypełniona zerami możemy uprościć równanie do postaci:

$$U_{i,i} = A_{i,i} - L_{i,i-1}U_{i-1,i} - L_{i,i-2}U_{i-2,i} - \ \ldots \ = A_{i,i} - L_{i,i-1}U_{i-1,i}$$

$$U_{i,i+1} = A_{i,i+1} - L_{i,i-1}U_{i-1,i+1} - L_{i,i-2}U_{i-2,i+1} - \ \ldots \ = A_{i,i+1} - L_{i,i-1}U_{i-1,i+1}$$

$$U_{i,i+2} = A_{i,i+2} - L_{i,i-1}U_{i-1,i+2} - \ \ldots \ = A_{i,i+1}$$

$$L_{i+1,i} = \frac{A_{i+1,i} - L_{i+1,i-1}U_{i-1,i} - \ldots}{U_{ii}} = \frac{A_{i+1,i}}{U_{ii}}$$

Spodziewamy się więc łącznej złożoności obliczeniowej równej O(N).

2. Kod

Program został napisany w języku Python3, dodatkowo w celu porównania wyników i czasu wykorzystana została biblioteka NumPy i Matplotlib.

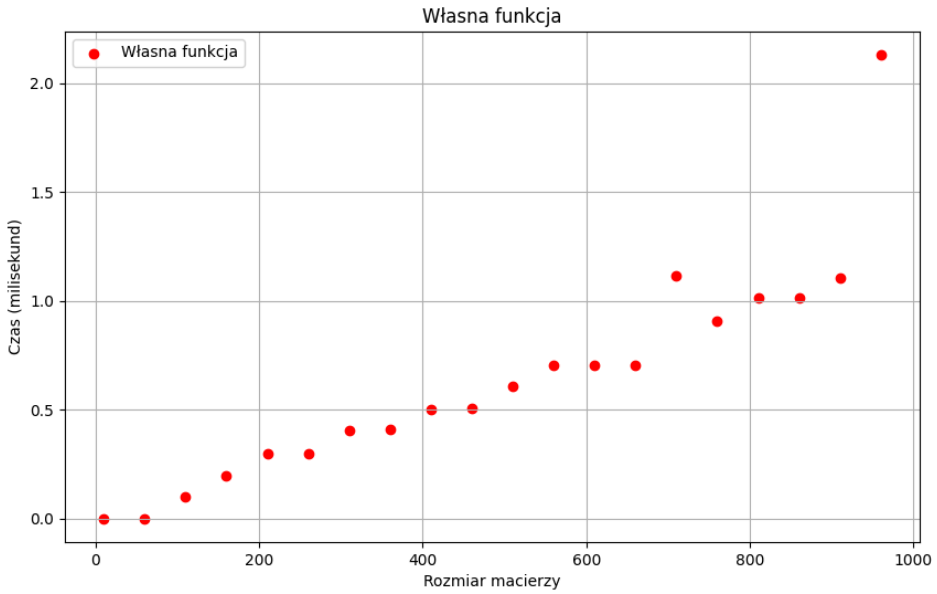
Aby otrzymać rzetelniejsze wyniki pomiaru czasu została wyciągnięta średnia z kilku pomiarów na daną macierz.

3. Wyniki

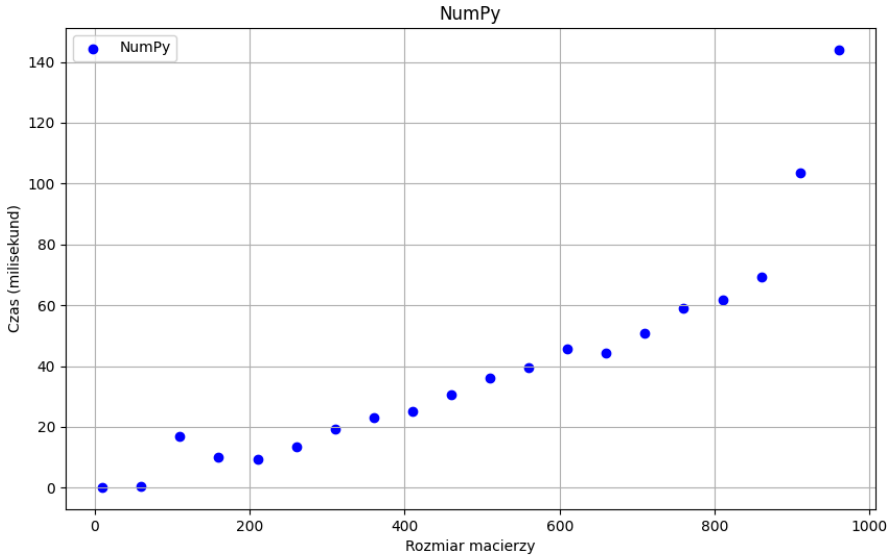
[0.9900990099009901, 1.6890209754236867, 2.6281571788073057, 3.2404355273428345, 4.041310987960568, 4.79920576485246,
5.563810740362443, 6.327205727858818, 7.091429481038214, 7.8557047198923255, 8.62017751719259, 9.384748061739252,
10.149407861972719, 10.914132497618905, 11.678909772369376, 12.443728942723103, 13.208582381424797, 13.973464063731573,
14.738369326030355, 15.503294459424898, 16.268236492980023, 17.033193015771214, 17.798162052839757, 18.563141970988408,
19.328131408343534, 20.093129220433923, 20.858134438630767, 21.623146237725187, 22.388163910406046, 23.15318684698417,
23.918214519151334, 24.683246466869555, 25.4482822877103, 26.21332162812696, 26.97836417626432, 27.74340965599881,
28.508457821971035, 29.27350845542345, 30.03856136069485, 30.8036163622545, 31.56867330218105, 32.33373203801081,
33.098792440893526, 33.86385439400585, 34.6289177911814, 35.39398253572371, 36.15904853937424, 36.9241157214124,
37.68918400786821, 38.45425333083158, 39.21932362784464, 39.98439484136567, 40.74946691829508, 41.51453980955507,
42.27961346971623, 43.04468785666484, 43.80976293130594, 44.574838657297846, 45.33991500081396, 46.10499193032905,
46.87006941642695, 47.63514743162705, 48.40022595022788, 49.16530494816536, 49.930384402884556, 50.69546429322326,
51.46054459930617, 52.22562530244847, 52.99070638506821, 53.755787830605975, 54.520869623451766, 55.28595174887786,
56.05103419297742, 56.816116942608055, 57.581199985340085, 58.3462833094088, 59.11136690367065, 59.87645075756285,
60.64153486106588, 61.40661920466916, 62.17170377933908, 62.93678857648953, 63.701873587954424, 64.46695880596243,
65.23204422311348, 65.99712983235659, 66.76221562696983, 67.52730160054118, 68.29238774695091, 69.05747406035513,
69.82256053517045, 70.58764716605954, 71.35273394791783, 72.11782087586084, 72.88290794521262, 73.64799515149457,
74.41308249041523, 75.17816995786059, 75.94325754988498, 76.70834526270262, 77.47343309267956, 78.23852103632609,
79.00360909028979, 79.76869725134868, 80.53378551640516, 81.29887388247974, 82.06396234670585, 82.8290509063242,
83.59413955867801, 84.35922830120828, 85.12431713144923, 85.88940604702418, 86.65449504564158, 87.41958412509108,
88.18467328324022, 88.94976251803075, 89.7148518274757, 90.47994120965605, 91.24503066271812, 92.01012018487057,
92.77520977438198, 93.54029942957825, 94.30538914884038, 95.07047893060204, 95.83556877334756, 96.60065867560996,
97.36574863596884, 98.13083865304868, 98.89592872551701, 99.66101885208285, 100.42610903149492, 101.1911992625403,
101.95628954404283, 102.72137987486195, 103.48647025389106, 104.25156068005644, 105.01665115231617, 105.78174166965856,
106.54683223110158, 107.31192283569128, 108.07701348250114, 108.84210417063102, 109.60719489920608, 110.37228566737603,
111.13737647431432, 111.90246731921718, 112.66755820130287, 113.43264911981109, 114.19774007400197, 114.96283106315566,
115.72792208657152, 116.49301314356747, 117.25810423347941, 118.0231953556607, 118.78828650948148, 119.55337769432816,
120.31846890960294, 121.0835601547233, 121.84865142912142, 122.61374273224389, 123.37883406355105, 124.14392542251682,
124.90901680862794, 125.67410822138396, 126.43919966029652, 127.20429112488918, 127.96938261469703, 128.7344741292662,
129.49956566815382, 130.26465723092736, 131.02974881716455, 131.79484042645302, 132.55993205839002, 133.32502371258207,
134.09011538864482, 134.85520708620268, 135.62029880488862, 136.38539054434392, 137.15048230421797, 137.91557408416787,
138.6806658838585, 139.44575770296217, 140.2108495411583, 140.97594139813336, 141.74103327358065, 142.50612516720014,
143.27121707869827, 144.03630900778776, 144.80140095418733, 145.5664929176219, 146.33158489782198, 147.09667689452385,
147.8617689074692, 148.62686093640522, 149.3919529810842, 150.15704504126347, 150.9221371167056, 151.68722920717758,
152.4523213124515, 153.21741343230374, 153.9825055665154, 154.74759771487177, 155.5126898771625, 156.27778205318123,
157.0428742427259, 157.8079664455982, 158.5730586616038, 159.33815089055201, 160.10324313225595, 160.86833538653224,
161.63342765320093, 162.3985199320856, 163.1636122230132, 163.92870452581383, 164.69379684032072, 165.45888916637043,
166.22398150380226, 166.98907385245877, 167.7541662121851, 168.51925858282968, 169.28435096424303, 170.0494433562791,
170.81453575879397, 171.57962817164648, 172.344720594698, 173.10981302781238, 173.87490547085585, 174.63999792369694,
175.4050903862067, 176.17018285825816, 176.93527533972673, 177.70036783049, 178.4654603304275, 179.23055283942108,
179.99564535735442, 180.76073788411318, 181.5258304195852, 182.29092296365982, 183.0560155162286, 183.82110807718473,
184.5862006464233, 185.35129322384097, 186.11638580933632, 186.88147840280936, 187.64657100416207, 188.4116636132976,
189.17675623012113, 189.94184885453905, 190.7069414864594, 191.47203412579174, 192.23712677244689, 193.0022194263373,
193.76731208737672, 194.53240475548023, 195.2974974305644, 196.06259011254681, 196.82768280134673, 197.59277549688434,
198.3578681990813, 199.1229609078603, 199.88805362314534, 200.65314634486148, 201.41823907293514, 202.18333180729354,
202.94842454786533, 203.71351729457996, 204.47861004736816, 205.24370280616154, 206.00879557089283, 206.77388834149576,
207.53898111790497, 208.30407390005612, 209.0691666878858, 209.83425948133151, 210.5993522803318, 211.3644450848259,
212.129537894754, 212.89463071005724, 213.6597235306776, 214.42481635655776, 215.18990918764132, 215.95500202387262,
216.72009486519696, 217.48518771156017, 218.25028056290898, 219.01537341919098, 219.78046628035418, 220.5455591463476,
221.3106520171209, 222.07574489262427, 222.8408377728089, 223.60593065762632, 224.37102354702898, 225.13611644096994,
225.9012093394026, 226.6663022422815, 227.4313951495614, 228.19648806119778, 228.9615809771468, 229.72667389736512]

Wyznacznik macierzy: 13.629575988041536

3.1. Wykres czasu potrzebnego do wykonania obliczeń dla macierzy o zmiennym rozmiarze - własna implementacja



3.2. Wykres czasu potrzebnego do wykonania obliczeń dla macierzy o zmiennym rozmiarze - zastosowanie NumPy



4. Wnioski

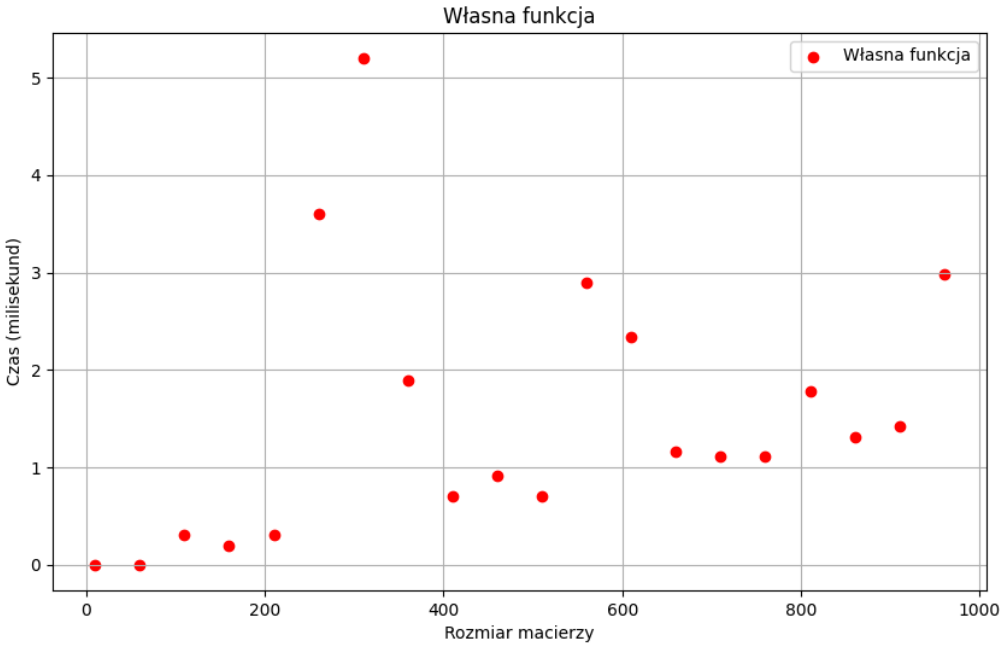
Tak jak przypuszczaliśmy czas obliczeń rośnie wprost proporcjonalnie do ilości danych(O(n)) a ponadto program jest wydajniejszy w porównaniu z wykorzystaniem gotowego rozwiązania z NumPy. Wyniki są poprawne a nieprecyzyjność spowodowana jest prawdopodobnie błędem wynikającym z precyzji obliczeń.

5. Droga do Sukcesu (I Ostre Zakręty) - Co Poszło Nie Tak?

Sekcję tą stworzyłem dodatkowo aby poruszyć, według mnie, ciekawe zagadnienia, które udało mi się zaobserwować podczas tworzenia tego programu i jak starałem się z nimi poradzić, czy też znaleźć wyjaśnienie ich zaistnienia. To swego rodzaju uzupełnienie pokazujące, że oprócz opracowania wydajnych algorytmów, warto również zwracać uwagę na kwestie związane ze sprzętem i architekturą systemu.

5.1. Problemy

Przy mierzeniu czasu tego typu zadań trzeba uwzględnić sporo czynników, poniekąd od nas niezależnych, które mogą doprowadzić do tego, że nasz wynik będzie bezużyteczny mimo poprawności stosowanych metod.



1. *Wykres liniowy - zniekształcony z powodu uruchomienia się trybu “oszczędzania energii” na testowanym urządzeniu*

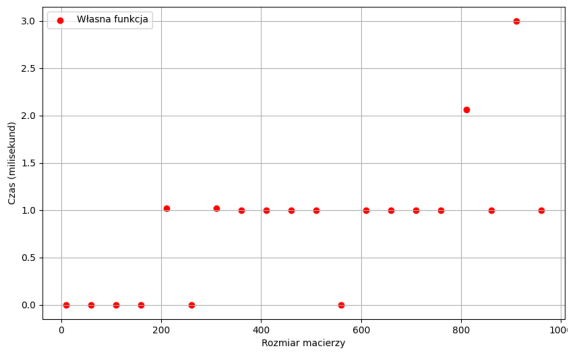
Program jest testowany na urządzeniu na którym działają inne procesy, które wpływają pośrednio na stabilność i dokładność pomiarów czasu. Ponadto Python automatycznie zarządza pamięcią, a cykl garbage collectora może się uruchomić podczas pomiarów, co prowadzi do nagłego, nieregularnego wzrostu czasu wykonania.

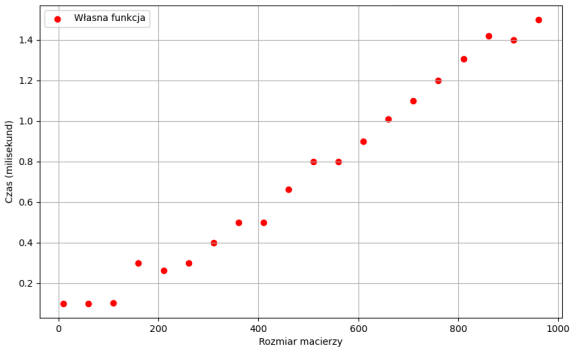
Sama architektura procesora, RAMu(przepustowość pamięci) czy też to na jakim systemie wykonywany jest program ma wiele do znaczenia.

5.2. Jak temu zapobiec?

Najprostszym sposobem jest uruchomienie kodu wiele razy dla danej wartości macierzy i obliczenie średniej.

```
def measure_time(func, N):
    total_time = 0
    for _ in range(10):
        start = time.time()
        func(N)
        stop = time.time()
        total_time += (stop - start)
    return total_time / 10
```





Przed wyciąganiem średniej z wyników*(wykres po lewej stronie)* wykres ten wyglądem jest bardziej zbliżony do piłokształtnego czy też prostokątnego niż do wykresu liniowego. Po zastosowaniu uśrednienia w postaci wykonania tych samych obliczeń dziesięciokrotnie nasz wykres zaczyna coraz bardziej przypominać liniowy.

Widzimy też, że czasy są średnio niższe, przypuszczam, że głównym powodem takiego stanu rzeczy jest uśrednieniem owych wyników chociaż można się doszukać też innych ciekawych rzeczy.

Pierwszym z nich jest optymalizacja po stronie Pythona, gdy kod jest wykonywany wielokrotnie, interpreter “uczy się", które operacje będą najczęściej używane i przechowuje ich wyniki w pamięci.

CPU cache warming czyli rozgrzanie pamięci procesora polega na wstępnym ładowaniu danych do pamięci podręcznej, zanim w ogóle będa one faktycznie potrzebne systemowi.

Jeśli chcielibyśmy uniknąć efektów cachingu czy rozgrzania CPU, można uruchamiać kod w oddzielnych procesach lub wyłączyć garbage collectora i użyć narzędzia do mierzenia czasu wyłącznie dla faktycznego działania procesora.