SPRAWOZDANIE NUM3

JAKUB KRĘCISZ

Treść zadania

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

$$A = \begin{pmatrix} 1.2 & \frac{0.1}{1} & \frac{0.4}{1^2} & & & \\ 0.2 & 1.2 & \frac{0.1}{2} & & & \\ & 0.2 & 1.2 & & & \\ \cdots & \cdots & \cdots & & \cdots & \cdots & \cdots \\ & & & 1.2 & \frac{0.1}{N-1} & \frac{0.4}{(N-1)^2} \\ & & & 0.2 & 1.2 & \frac{0.1}{N-2} \\ & & & & 0.2 & 1.2 \end{pmatrix}$$

oraz $\mathbf{x} = (1,2,\ldots,N)^T$. Ustalamy $N = 100$. Oblicz również wyznacznik macierzy **A**. Zadanie rozwiąż właściwą metodą (uzasadnij wybór) i wykorzystaj strukturę macierzy (w przeciwnym wypadku zadanie nie będzie zaliczone). 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).

Dla ambitnych: potraktuj *N* jako zmienną i zmierz czas działania swojego programu w funkcji *N*. Wynik przedstaw na wykresie.

Omówienie zadania

Zacznijmy od omówienia problemu, z którym musimy się uporać. Mamy dane **A** i **x** oraz niewiadomy wektor **y**. Naszym zadaniem jest rozwiązanie równania: $\mathbf{y} = \mathbf{A}^{-1}\mathbf{x}$**.** Jak widzimy, w naszym równaniu występuje odwrotność macierzy A, obliczenie tego typu macierzy ma bardzo wielką złożoność obliczeniową, czego chcielibyśmy uniknąć. W takim razie najlepszym sposobem będzie pozbycie się tej macierzy mnożąc obustronnie przez A:

$$y = A^{-1}x \iff Ay = x$$

Jeżeli chodzi o macierz A, mamy do czynienia z macierzą, która jedyne niezerowe wartości ma na diagonali. Również możemy stwierdzić że nasza macierz A jest macierzą rzadką, ponieważ większość jej elementów to zera. Pomyślmy co moglibyśmy zrobić w takim wypadku.

Jak widać, nie jest konieczne przetrzymywać naszą macierz w całej postaci. Wystarczy gdy będziemy przechowywać jej diagonale, w których mamy jakieś wartości inne niż zero. Dzięki takiemu rozwiązaniu o wiele mniej pamięci użyjemy do przechowania danej macierzy. Dlatego też przechowajmy naszą macierz jako macierz wstęgową, będzie to macierz wymiaru N x ilość_diagonali:

$$\begin{pmatrix} 1.2 & \frac{0.1}{1} & \frac{0.4}{1^2} \\ 0.2 & 1.2 & \frac{0.1}{2} \\ & 0.2 & 1.2 \\ \cdots & \cdots & \cdots \end{pmatrix} \quad \begin{matrix} & \cdots & \cdots & \cdots \\ 1.2 & \frac{0.1}{N-1} & \frac{0.4}{(N-1)^2} & \\ 0.2 & 1.2 & \frac{0.1}{N-2} & \\ & 0.2 & 1.2 & \end{matrix} \rightarrow \begin{pmatrix} 0 & 1.2 & \frac{0.1}{1} & \frac{0.4}{1^2} \\ 0.2 & 1.2 & \vdots & \vdots \\ 0.2 & 1.2 & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ 0.2 & 1.2 & \vdots & 0 \\ 0.2 & 1.2 & 0 & 0 \end{pmatrix}$$

Teraz musimy pomyśleć jaki algorytm użyjemy. Sprowadzenie naszej macierzy do macierzy trójkątnej pozwoliłoby na zastosowanie forward i backward substitution żeby wyliczyć wynik. Tak więc, żeby to zrobić, zastosujemy rozkład macierzy A = LU. Rozkład ten nie jest liniowy w zasadzie, ale w naszym przypadku będzie ze względu na to jaką macierz my chcemy rozłożyć. Użyjemy w tym przypadku algorytmu Dollitle'a, którego wzory mają taką postać:

$$U_{ij} = A_{ij} - \sum_{k<i} L_{ik}U_{kj}$$

$$L_{ij} = \frac{A_{ij} - \sum_{k<j} L_{ik}U_{kj}}{U_{jj}}$$

W naszym przypadku potrzebne są nam tylko wzory na elementy z diagonali macierzy A. Oznaczając odpowiednio tak jak we wzorach nasze wzory diagonalne to będą:

$$U_{ii} = A_{ii} - \sum_{k<i} L_{ik}U_{ki} = A_{ii} - L_{i,i-1}U_{i-1,i} - L_{i,i-2}U_{i-2,i} - L_{i,i-3}U_{i-3,i} - \cdots = A_{ii} - L_{i,i-1}U_{i-1,i}$$

$$U_{i,i+1} = A_{i,i+1} - \sum_{k<i} L_{ik}U_{k,i+1} = A_{i,i+1} - L_{i,i-1}U_{i-1,i+1} - L_{i,i-2}U_{i-2,i+1} - L_{i,i-3}U_{i-3,i+1} - \cdots = A_{ii+1} - L_{i,i-1}U_{i-1,i+1}$$

$$U_{i,i+2} = A_{i,i+2} - \sum_{k<i} L_{ik}U_{k,i+2} = A_{i,i+2} - L_{i,i-1}U_{i-1,i+2} - L_{i,i-2}U_{i-2,i+2} - L_{i,i-3}U_{i-3,i+2} - \cdots = A_{ii+2} - L_{i,i-1}U_{i-1,i+2}$$

$$U_{i,i+1} = A_{i,i+1} - \sum_{k<i} L_{ik}U_{k,i+1} = A_{i,i+1} - L_{i,i-1}U_{i-1,i+1} - L_{i,i-2}U_{i-2,i+1} - L_{i,i-3}U_{i-3,i+1} - \cdots = A_{ii+1} - L_{i,i-1}U_{i-1,i+1}$$

$$L_{i+1,i} = \frac{A_{i+1,i} - \sum_{k<i} L_{i+1,k} U_{k,i}}{U_{i,i}} = \frac{A_{i+1,i} - L_{i+1,i-1} U_{i-1,i} \cdots}{U_{i,i}} = \frac{A_{i+1,i}}{U_{i,i}}$$

Fragmenty, które są na czerwono, upraszczają nam się ze względu na zera w naszej macierzy A.

Następnym krokiem jest podstawienie naszego rozkładu A=LU do naszego równania, które chcemy rozwiązać:

$$A = LU \ \wedge Ax = y \implies LUx = y$$

Teraz równanie LUx = y rozbijemy na dwa równania by rozwiązać je za pomocą odpowiednio forward i backward substitution:

$$Lz = x \ \text{ oraz } \ Uy = z$$

Gdzie:

Forward substitution

$$z_0 = x_0$$

$$z_n = x_n - L_n z_{n-1}$$

Backward substitution

$$y_{99} = \frac{z_{99}}{U_{0_{99}}}$$

$$y_n = \frac{z_n - U_{1_n} y_{n+1} - U_{2_n} y_{n+2}}{U_{0_n}} \qquad Dla: n < 98$$

*Jako U oznaczyłem diagonale macierzy U

Uruchomienie programu

Do uruchomienia programu wykorzystamy Makefile:

Aby uruchomić nasz program, wystarczy użyć polecenia:

```
make run
```

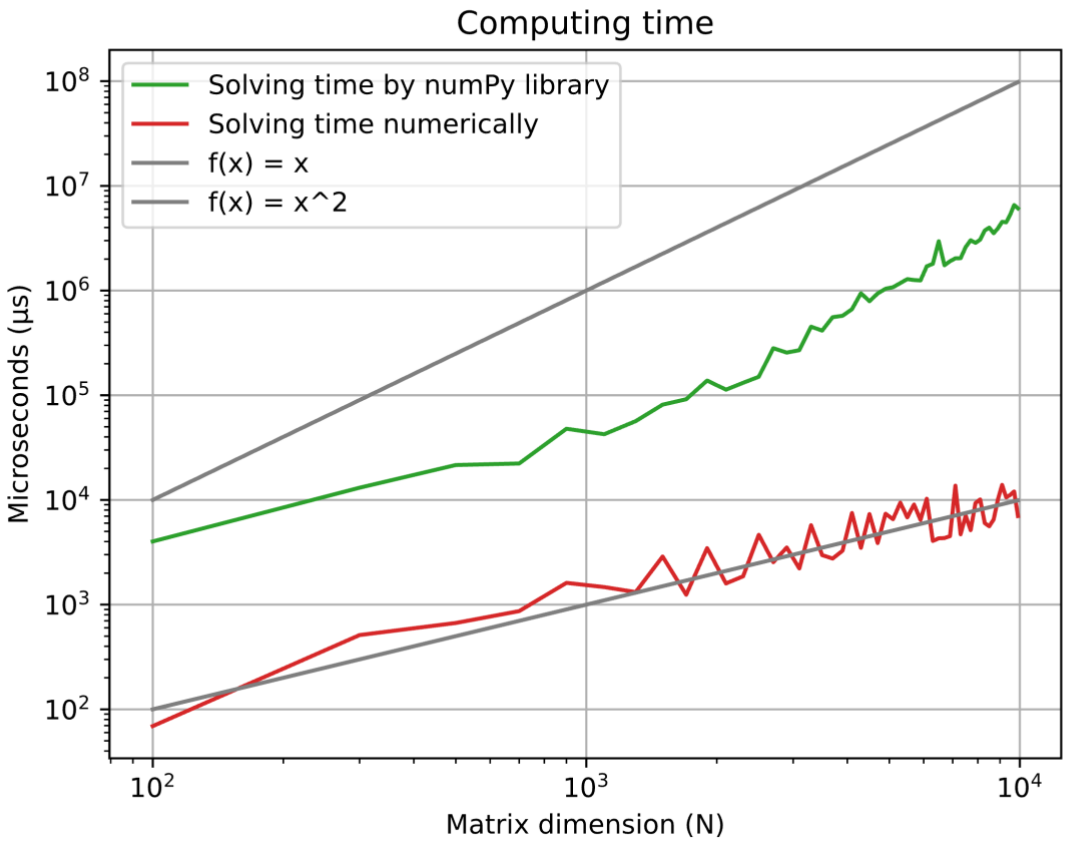
Wyniki

1. Pierwszy podpunktem było policzenie dla N=100, wynik to:

det(A) = 78240161.00959387

y = [0.03287133486041395, 1.3396227980963753, 2.066480295894664, 2.825543605175336, 3.557571715528883, 4.284492868897645, 5.00721018451999, 5.727664002754518, 6.446615582748809, 7.164554400995276, 7.881773878242026, 8.598465868371878, 9.314759799907844, 10.030746230199034, 10.74649032115277, 11.462040127963592, 12.177431844626687, 12.892693237901542, 13.60784595684208, 14.322907124390252, 15.03789045794619, 15.75280707355121, 16.467666073000725, 17.182474979167374, 17.897240063340146, 18.611966594532937, 19.32665903159678, 20.041321172855753, 20.75595627381683, 21.47056714061568, 22.185156204831525, 22.899725583859315, 23.61427712998635, 24.328812470561147, 25.043333041083297, 25.757840112626393, 26.472334814693667, 27.186818154368854, 27.901291032443737, 28.615754257064278, 29.33020855532933, 30.04465458319117, 30.75909293394065, 31.473524145507586, 32.18794870676451, 32.902367062989086, 33.61677962061327, 34.331186751365145, 35.04558879589254, 35.75998606694211, 36.47437885215638, 37.18876741654113, 37.90315200464761, 38.617532842507245, 39.331910139350974, 40.04628408914067, 40.7606548719361, 41.47502265511775, 42.189387594482916, 42.90374983523002, 43.6181095128443, 44.33246675389621, 45.04682167676243, 45.76117439227791, 46.47552500432681, 47.189873610378676, 47.904220301975755, 48.61856516517662, 49.33290828096055, 50.047249725596565, 50.761589570980924, 51.47592788494589, 52.19026473154275, 52.904600171301595, 53.61893426146981, 54.33326705623165, 55.04759860691019, 55.761928962153874, 56.47625816810818, 57.19058626857465, 57.90491330515779, 58.61923931740096, 59.33356434291259, 60.04788841748285, 60.76221157519233, 61.47653384851288, 62.1908552684013, 62.9051758643867, 63.61949566465193, 64.33381469610926, 65.04813298447127, 65.76245055431694, 66.47676742915336, 67.19108363147355, 67.9053991828134, 68.61971410401006, 69.33402833257784, 70.04833794418792, 70.7650588638003, 71.53915685603329]

2. Dodatkowym zadaniem było przyjęcie N jako zmienną, zmierzenie czasu i przedstawienie wyników na wykresie:



Wnioski

Jeżeli chodzi o wyniki, nasz zaimplementowany algorytm oblicza macierze z takim samym rezultatem jak wbudowane biblioteki do tego stworzone (w naszym przypadku numPy).

Jeżeli chodzi o czas wykonania. Nasz zaimplementowany algorytm okazał się być naprawdę zauważalnie szybszy od metody ogólnej, którą jest w naszym przypadku, użycie funkcji z biblioteki numPy. Jak widzimy na wykresie, algorytm sobie o wiele lepiej radzi, dzięki zastosowaniu forward i backward substitution. Wniosek z tego płynie taki, że zastosowanie odpowiednich algorytmów do danej sytuacji, pozwala na o wiele szybsze poradzenie sobie z problemem rozwiązania takiego równania.