MOWNIT - Zestaw 3 Układy równań

Opracował: Mateusz Woś

Wszystkie zadania zostały zaimplementowane w języku Python.
Korzystałem z bibliotek numpy i matplotlib.

Korzystając z przykładu napisz program, który:

1. *Jako parametr pobiera rozmiar układu równań n*
2. *Generuje macierz układu A(nxn) i wektor wyrazów wolnych b(n)*
3. *Rozwiazuje uklad (jest w przykladzie) i sprawdza poprawnosc rozwiązania tj., czy Ax=b (lub lepiej czy Ax-b=E, gdzie E jest macierza bledów) za pomoca wybranej funkcji z BLAS. Proszę też sprawdzić poprawność rozwiązania przy pomocy narzędzia Octave.*
4. *Mierzy czas dekompozycji macierzy - do mierzenia czasu można skorzystać z przykładowego programu dokonującego pomiaru czasu procesora spędzonego w danym fragmencie programu.*
5. *Mierzy czas rozwiązywania układu równań*

Import bibliotek:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as lin
```

Ad.1, Ad.2:

Korzystając z możliwości biblioteki numpy wygenerowałem macierze o określonym przeze mnie rozmiarze:

```
size = 5
matrix = np.random.rand(size,size)
matrix_r = np.around(matrix,decimals=4)
vector = np.random.rand(size)
vector_r = np.around(vector, decimals=4)
```

Ograniczyłem wygenerowane wartości do 4 miejsc po przecinku dla lepszej wizualizacji wyników.

```
matrix_r
```

```
array([[ 0.4032,  0.4649,  0.9018,  0.9638,  0.5316],
       [ 0.0914,  0.2021,  0.9832,  0.4556,  0.4794],
       [ 0.6768,  0.3884,  0.7019,  0.1141,  0.8732],
       [ 0.23  ,  0.9792,  0.4944,  0.706 ,  0.5501],
       [ 0.9964,  0.7448,  0.3252,  0.8514,  0.4434]])
```

```
vector_r
```

```
array([ 0.4444,  0.2099,  0.2109,  0.4217,  0.7091])
```

Ad.3:

Do rozwiązania układu równań Ax=b posłużyłem się podstawowym solverem z biblioteki numpy.

```
result = np.linalg.solve(matrix_r,vector_r)
result_r = np.around(result,decimals=4)
```

```
result_r
```

```
array([ 0.4997,  0.464 ,  0.3726,  0.0421, -0.6572])
```

Do sprawdzania poprawności posłużyłem się pythonem i jego built-in operatorem @ - mnożenie macierzy. Wynik mnożenia macierzy wynikowej i A powinien być równy B.

```
test_result = matrix_r @ result_r
test_result_r = np.round(test_result, 4)
```

```
np.array_equal(vector_r, test_result_r)
```

```
True
```

Macierz wynikowa i B są sobie równe.

Ad.4:

Do dekompozycji macierzy posłużyłem się jedną z wbudowanych do numpay'a funkcji. (np.linalg.qr). Czas mierzyłem funkcją clock().

```
from time import clock

start = clock()
L = np.linalg.qr(matrix)
print("Matrix decomposition time = {0}".format(clock() - start))
```

```
Matrix decomposition time = 0.005209635949540825
```

Ad.5:

Mierzenie czasu solvera wykonałem w identyczny sposób jak poprzednio.

```
start = clock()
result_t = np.linalg.solve(matrix_r,vector_r)
print("Solving time = {0}".format(clock() - start))
```

```
Solving time = 0.00014056614956103886
```

Zadanie domowe: Narysuj wykres zależności czasu dekompozycji i czasu rozwiązywania układu od rozmiaru układu równań. Wykonaj pomiary dla 10 wartości z przedziału od 10 do 1000.

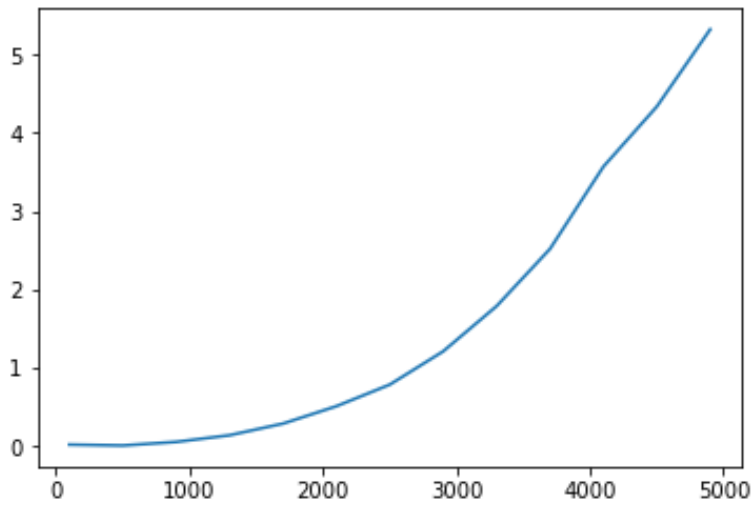
W powyższym zadaniu dla poprawności obliczeń zwiększyłem ilosc pomiarów do 13. CPU zbyt łatwo radziło sobie z macierzami o małych rozmiarach (<2000). Skoki wielkości macierzy po 400.

Obliczenia wykonałem na macierzach o rozmiarach od 100x100 do 5000x5000.

Dekompozycja:

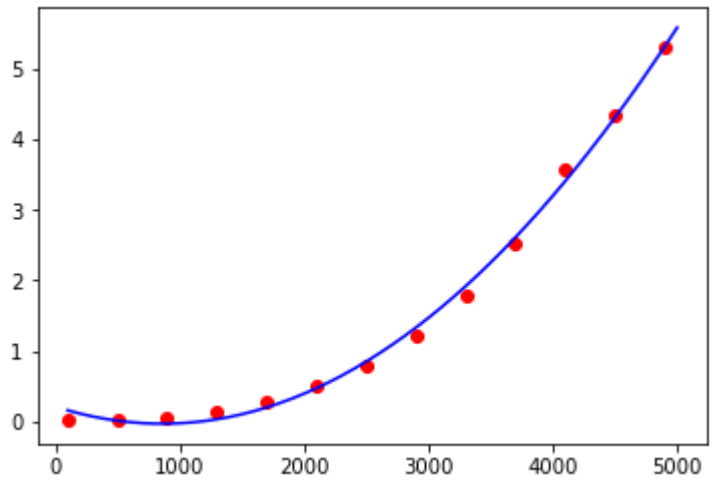
```
val_dict = {}
for i in range(100, 5001, 400):
    matrix_temp = np.random.rand(i,i)
    start = clock()
    L = np.linalg.qr(matrix_temp)
    end = clock() - start
    val_dict.update({i: end})

lists = sorted(val_dict.items())
x, y = zip(*lists)

plt.plot(x, y)
plt.show()
```

Wykres:



Z wykresu można spodziewać się, iż złożoność funkcji wynosi O(n^2). Sprawdziłem to za pomocą funkcji polyfit z biblioteki numpy.

```
coef = np.polyfit(x,y,2)
plt.plot(x,y, 'ro')
t = np.linspace(100,5000)
plt.plot(t, coef[0]*t**2+coef[1]*t, 'b')
plt.show()
```

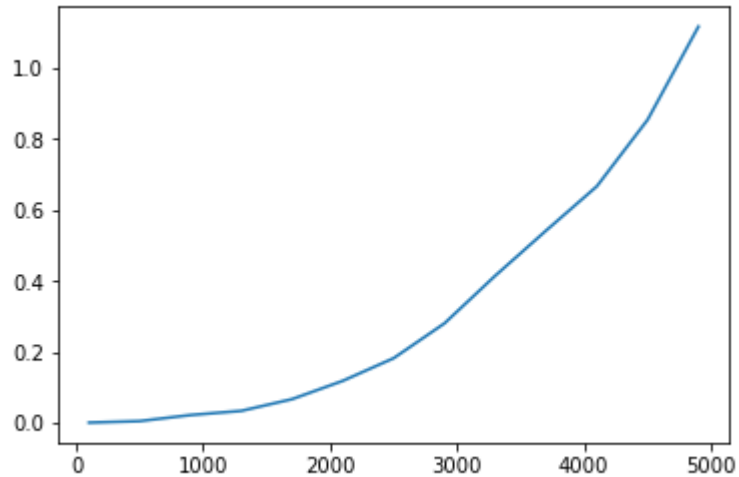


Widać tutaj dokładnie założoną przeze mnie hipotezę, funkcja ma złożoność n^2.

Solver:

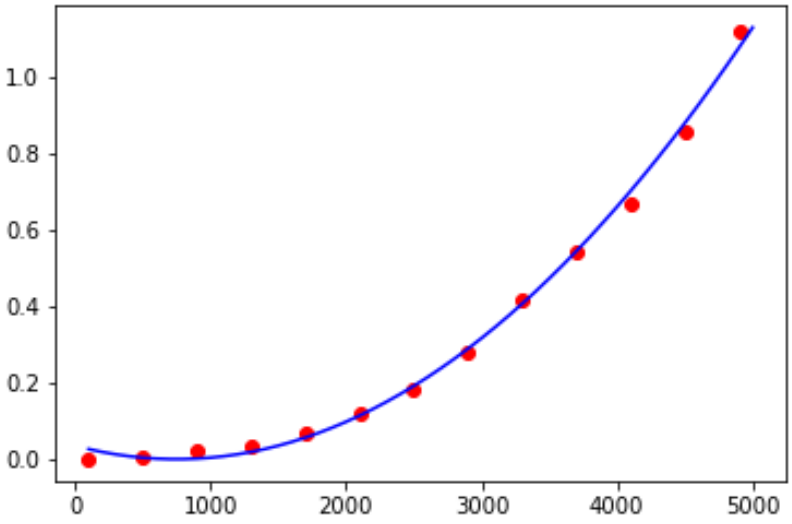
```
val_dict = {}
for i in range(100, 5001, 400):
    matrix_temp = np.random.rand(i,i)
    vector_temp = np.random.rand(i)
    start = clock()
    result = np.linalg.solve(matrix_temp,vector_temp)
    end = clock() - start
    val_dict.update({i: end})

lists = sorted(val_dict.items())
x2, y2 = zip(*lists)

plt.plot(x2,y2)
plt.show()
```

Wykres:



Z wykresu można spodziewać się znowu, iż złożoność funkcji wynosi O(n^2).

```
coef = np.polyfit(x2,y2,2)
plt.plot(x2,y2, 'ro')
t = np.linspace(100,5000)
plt.plot(t, coef[0]*t**2+coef[1]*t+coef[2], 'b')
plt.show()
```



Widać tutaj dokładnie założoną przeze mnie hipotezę, funkcja ma złożoność n^2.