Metody numeryczne

Projekt 2 - Układy równań liniowych

Autor: Jakub Kwiatkowski 184348

POLITECHNIKA GDAŃSKA

Wydział Elektroniki, Telekomunikacji i Informatyki Politechnika Gdańska
03.05.2022

Wstęp

Głównym celem tego projektu była implementacja metod iteracyjnych rozwiązywania układów równań liniowych takich jak metoda Jacobiego oraz metoda Gaussa-Seidla oraz metody bezpośredniej rozwiązywania układów równań liniowych o nazwie metoda faktoryzacji LU. Do wykonaniu projektu użyłem języka Python wraz z bibliotekami math, time oraz matplotlib. Projekt wykonałem za pomocą Jupyter Notebook.

[35]:
```
import math
import time
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
```

[45]:
```
#A
# index = 184348
a1=8
a2=-1
a3=-1
f=4
N=948
b=[math.sin(n*(f+1)) for n in range(N)]
def stworz_macierz_A(N,a1,a2,a3):
    macierz=[[0 for i in range(N)] for i in range(N)]
    for i in range(N):
        macierz[i][i]=a1
        if i<N-1:
            macierz[i][i+1]=a2
        if i>0:
            macierz [i][i-1]=a2
        if i>1:
            macierz [i][i-2]=a3
        if i<N-2:
            macierz [i][i+2]=a3
    return macierz
A=stworz_macierz_A(N,a1,a2,a3)
```

[6]:
```
def norma_euklidesowa_wektora_residuum(wektor):
    N=len(wektor)
    norma=0
    for i in range(N):
        norma = norma + wektor[i]**2
    return math.sqrt(norma)
```

[7]:
```
def wektor_residuum(A,x,b):
    N=len(x)
    res=[0 for i in range(N)]
    for i in range(N):
        for j in range(N):
```

```
            res[i]=res[i]+A[i][j]*x[j]
    res = [(res[i]-b[i]) for i in range(N)]
    return res
```

[27]:
```
#B - metoda Jacobiego
bariera=10**-9
def metoda_Jacobiego(A,b,bariera):
    start=time.time()
    N=len(A)
    licznik_iteracji=0
    x=[1 for i in range(N)]
    x_poprzednie = [1 for i in range(N)]
    res = [1 for i in range(N)]
    while norma_euklidesowa_wektora_residuum(res) > bariera:
        for i in range(N):
            sumaL=0
            sumaU=0
            for j in range(i):
                sumaL=sumaL+A[i][j]*x_poprzednie[j]
            for j in range(i+1,N):
                sumaU=sumaU+A[i][j]*x_poprzednie[j]
            x[i]=(b[i]-sumaL-sumaU)/A[i][i]
        x_poprzednie=[x[i] for i in range(N)]
        res=wektor_residuum(A,x,b)
        licznik_iteracji+=1
    koniec=time.time()
    czas_trwania=koniec-start
    return x,licznik_iteracji,czas_trwania
Jacobi_wyniki,liczba_iteracji_Jacobi,czas_trwania_Jacobi=metoda_Jacobiego(A,b,bariera)
print("Metoda Jacobiego")
print("Liczba iteracji:",liczba_iteracji_Jacobi)
print("Czas trwania:",czas_trwania_Jacobi)
```

```
Metoda Jacobiego
Liczba iteracji: 37
Czas trwania: 6.870602130889893
```

[28]:
```
#B - metoda Gaussa-Seidla
def metoda_Gaussa_Seidla(A,b,bariera):
    start=time.time()
    N=len(A)
    licznik_iteracji=0
    x=[1 for i in range(N)]
    x_poprzednie = [1 for i in range(N)]
    res = [1 for i in range(N)]
    while norma_euklidesowa_wektora_residuum(res) > bariera:
        for i in range(N):
            sumaL=0
```

```
            sumaU=0
            for j in range(i):
                sumaL=sumaL+A[i][j]*x[j]
            for j in range(i+1,N):
                sumaU=sumaU+A[i][j]*x_poprzednie[j]
            x[i]=(b[i]-sumaL-sumaU)/A[i][i]
        x_poprzednie=[x[i] for i in range(N)]
        res=wektor_residuum(A,x,b)
        licznik_iteracji+=1
    koniec=time.time()
    czas_trwania=koniec-start
    return x,licznik_iteracji,czas_trwania
Gauss_Seidl_wyniki,liczba_iteracji_Gauss_Seidl,czas_trwania_Gauss_Seidl=metoda_Gaussa_Seidla(A,
print("Metoda Gaussa-Seidla")
print("Liczba iteracji:",liczba_iteracji_Gauss_Seidl)
print("Czas trwania:",czas_trwania_Gauss_Seidl)
```

```
Metoda Gaussa-Seidla
Liczba iteracji: 24
Czas trwania: 4.348486423492432
```

Metoda Gaussa-Seidla jest około 1,5 razy szybsza niż metoda Jacobiego jeśli chodzi o czas trwania oraz potrzebuje około 1,5 raza mniej iteracji.

[10]:
```
#C
a1=3
a2=-1
a3=-1
N=948
A=stworz_macierz_A(N,a1,a2,a3)
Gauss_Seidl_wyniki,liczba_iteracji_Gauss_Seidl,czas_trwania_Gauss_Seidl=metoda_Gaussa_Seidla(A,
print("Metoda Gaussa-Seidla")
print("Liczba iteracji:",liczba_iteracji_Gauss_Seidl)
print("Czas trwania:",czas_trwania_Gauss_Seidl)
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
Input In [10], in <cell line: 7>()
      5 N=948
      6 A=stworz_macierz_A(N,a1,a2,a3)
----> 7␣
 ↪Gauss_Seidl_wyniki,liczba_iteracji_Gauss_Seidl,czas_trwania_Gauss_Seidl=metoda_Gaussa_Seidla
      8 print("Metoda Gaussa-Seidla")
      9 print("Liczba iteracji:",liczba_iteracji_Gauss_Seidl)

Input In [9], in metoda_Gaussa_Seidla(A, b, bariera)
      7 res = [1 for i in range(N)]
      8 start=time.time()
```

```
----> 9 while norma_euklidesowa_wektora_residuum(res) > bariera:
     10     for i in range(N):
     11         suma1=0

Input In [6], in norma_euklidesowa_wektora_residuum(wektor)
      3 norma=0
      4 for i in range(N):
----> 5     norma = norma + wektor[i]**2
      6 return math.sqrt(norma)

OverflowError: (34, 'Result too large')
```

[11]:
```
Jacobi_wyniki,liczba_iteracji_Jacobi,czas_trwania_Jacobi=metoda_Jacobiego(A,b,bariera)
print("Metoda Jacobiego")
print("Liczba iteracji:",liczba_iteracji_Jacobi)
print("Czas trwania:",czas_trwania_Jacobi)
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
Input In [11], in <cell line: 1>()
----> 1␣
 ↪Jacobi_wyniki,liczba_iteracji_Jacobi,czas_trwania_Jacobi=metoda_Jacobiego(A,b, ariera)
      2 print("Metoda Jacobiego")
      3 print("Liczba iteracji:",liczba_iteracji_Jacobi)

Input In [8], in metoda_Jacobiego(A, b, bariera)
      8 res = [1 for i in range(N)]
      9 start=time.time()
---> 10 while norma_euklidesowa_wektora_residuum(res) > bariera:
     11     for i in range(N):
     12         suma1=0

Input In [6], in norma_euklidesowa_wektora_residuum(wektor)
      3 norma=0
      4 for i in range(N):
----> 5     norma = norma + wektor[i]**2
      6 return math.sqrt(norma)

OverflowError: (34, 'Result too large')
```

Przy następujących parametrach podczas liczenia normy euklidesowej wektora residuum pojawia się błąd przy obu metodach. Wnioskiem z tego jest, że przy tych parametrach metody iteracyjne nie zbiegają się

[40]:
```
#D
a1=8
```

```
a2=-1
a3=-1
f=4
N=948
b = [math.sin(i * (f + 1)) for i in range(N)]
A=stworz_macierz_A(N,a1,a2,a3)
def tworzenie_macierzy_LU(A):
    N=len(A)
    U=[[0 for i in range(N)] for i in range(N)]
    for i in range (N):
        for j in range(N):
            U[i][j]=A[i][j]
    L=[[0 for i in range(N)] for i in range(N)]
    for i in range(N):
        L[i][i]=1
    for k in range(N-1):
        for j in range(k+1,N):
            L[j][k]=U[j][k]/U[k][k]
            for i in range(k,N):
                U[j][i]=U[j][i]-L[j][k]*U[k][i]
    return L,U
def faktoryzacja_LU(A,b):
    start = time.time()
    N=len(A)
    x = [0 for i in range(N)]
    y = [0 for i in range(N)]
    L, U = tworzenie_macierzy_LU(A)
    for i in range(N):
        sumaL = 0
        for j in range(i):
            sumaL = sumaL + L[i][j]*y[j]
        y[i] = (b[i] - sumaL)/L[i][i]
    for i in range(N-1, -1, -1):
        sumaU = 0
        for j in range(i+1, N):
            sumaU = sumaU + U[i][j] * x[j]
        x[i] = (y[i] - sumaU)/U[i][i]
    res = wektor_residuum(A, x, b)
    norma=norma_euklidesowa_wektora_residuum(res)
    koniec = time.time()
    czas_trwania=koniec-start
    return x, y, czas_trwania,norma
x,y,czas_trwania_faktoryzacji_LU,norma_residuum_faktoryzacji_LU=faktoryzacja_LU(A,b)
print("Faktoryzacja LU")
print("Norma residuum",norma_residuum_faktoryzacji_LU)
```

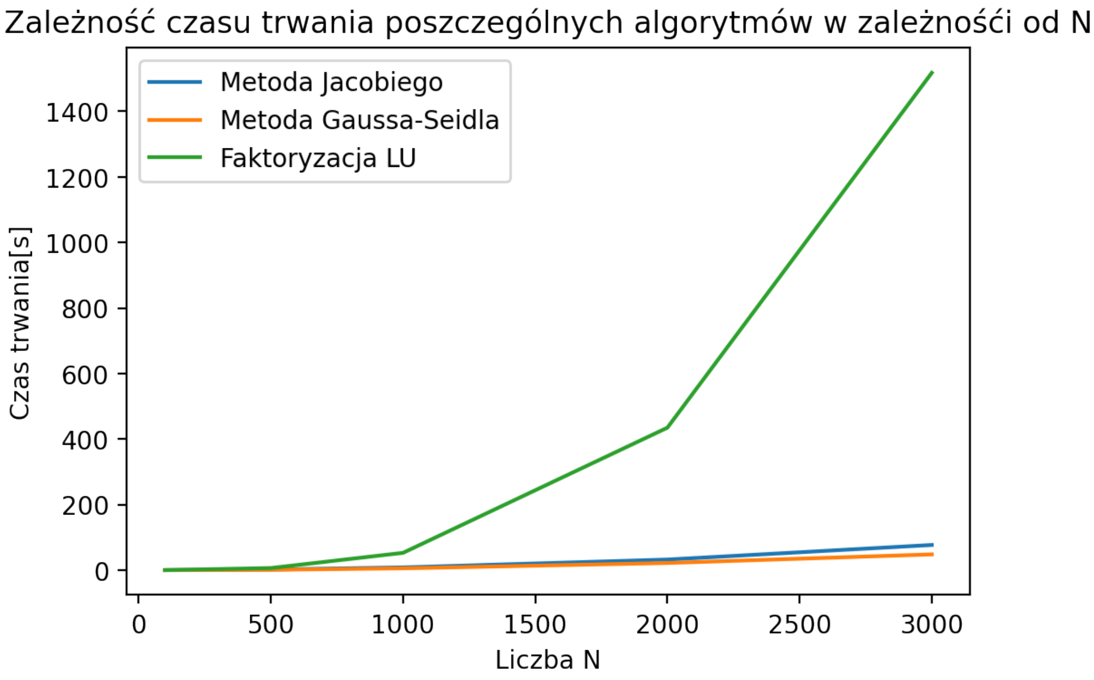
```
Faktoryzacja LU
Norma residuum 2.6623292784175393e-15
```

Norma residuum w przypadku metody bezpośredniej - metody faktoryzacji LU jest rzędu 10^(-15) czyli bardzo bliska zeru, co oznacza dużą dokładność obliczeń

[32]:
```
#E
czas_trwania_faktoryzacja_LU_tab=[]
czas_trwania_Gauss_Seidl_tab=[]
czas_trwania_Jacobi_tab=[]
a1=8
a2=-1
a3=-1
f=4
N=100
b=[math.sin(i * (f + 1)) for i in range(N)]
A=stworz_macierz_A(N,a1,a2,a3)
x,y,czas_trwania_faktoryzacji_LU,norma_residuum_faktoryzacji_LU=faktoryzacja_LU(A,b)
czas_trwania_faktoryzacja_LU_tab.append(czas_trwania_faktoryzacji_LU)
Gauss_Seidl_wyniki,liczba_iteracji_Gauss_Seidl,czas_trwania_Gauss_Seidl=metoda_Gaussa_Seidla(A,
czas_trwania_Gauss_Seidl_tab.append(czas_trwania_Gauss_Seidl)
Jacobi_wyniki,liczba_iteracji_Jacobi,czas_trwania_Jacobi=metoda_Jacobiego(A,b,bariera)
czas_trwania_Jacobi_tab.append(czas_trwania_Jacobi)
N=500
b=[math.sin(i * (f + 1)) for i in range(N)]
A=stworz_macierz_A(N,a1,a2,a3)
x,y,czas_trwania_faktoryzacji_LU,norma_residuum_faktoryzacji_LU=faktoryzacja_LU(A,b)
czas_trwania_faktoryzacja_LU_tab.append(czas_trwania_faktoryzacji_LU)
Gauss_Seidl_wyniki,liczba_iteracji_Gauss_Seidl,czas_trwania_Gauss_Seidl=metoda_Gaussa_Seidla(A,
czas_trwania_Gauss_Seidl_tab.append(czas_trwania_Gauss_Seidl)
Jacobi_wyniki,liczba_iteracji_Jacobi,czas_trwania_Jacobi=metoda_Jacobiego(A,b,bariera)
czas_trwania_Jacobi_tab.append(czas_trwania_Jacobi)
N=1000
b=[math.sin(i * (f + 1)) for i in range(N)]
A=stworz_macierz_A(N,a1,a2,a3)
x,y,czas_trwania_faktoryzacji_LU,norma_residuum_faktoryzacji_LU=faktoryzacja_LU(A,b)
czas_trwania_faktoryzacja_LU_tab.append(czas_trwania_faktoryzacji_LU)
Gauss_Seidl_wyniki,liczba_iteracji_Gauss_Seidl,czas_trwania_Gauss_Seidl=metoda_Gaussa_Seidla(A,
czas_trwania_Gauss_Seidl_tab.append(czas_trwania_Gauss_Seidl)
Jacobi_wyniki,liczba_iteracji_Jacobi,czas_trwania_Jacobi=metoda_Jacobiego(A,b,bariera)
czas_trwania_Jacobi_tab.append(czas_trwania_Jacobi)
N=2000
b=[math.sin(i * (f + 1)) for i in range(N)]
A=stworz_macierz_A(N,a1,a2,a3)
x,y,czas_trwania_faktoryzacji_LU,norma_residuum_faktoryzacji_LU=faktoryzacja_LU(A,b)
czas_trwania_faktoryzacja_LU_tab.append(czas_trwania_faktoryzacji_LU)
Gauss_Seidl_wyniki,liczba_iteracji_Gauss_Seidl,czas_trwania_Gauss_Seidl=metoda_Gaussa_Seidla(A,
czas_trwania_Gauss_Seidl_tab.append(czas_trwania_Gauss_Seidl)
Jacobi_wyniki,liczba_iteracji_Jacobi,czas_trwania_Jacobi=metoda_Jacobiego(A,b,bariera)
czas_trwania_Jacobi_tab.append(czas_trwania_Jacobi)
```

```
N=3000
b=[math.sin(i * (f + 1)) for i in range(N)]
A=stworz_macierz_A(N,a1,a2,a3)
x,y,czas_trwania_faktoryzacji_LU,norma_residuum_faktoryzacji_LU=faktoryzacja_LU(A,b)
czas_trwania_faktoryzacja_LU_tab.append(czas_trwania_faktoryzacji_LU)
Gauss_Seidl_wyniki,liczba_iteracji_Gauss_Seidl,czas_trwania_Gauss_Seidl=metoda_Gaussa_Seidla(A,b
czas_trwania_Gauss_Seidl_tab.append(czas_trwania_Gauss_Seidl)
Jacobi_wyniki,liczba_iteracji_Jacobi,czas_trwania_Jacobi=metoda_Jacobiego(A,b,bariera)
czas_trwania_Jacobi_tab.append(czas_trwania_Jacobi)
```

[44]:
```
#E
argumenty=[100,500,1000,2000,3000]
plt.figure(dpi=200)
plt.plot(argumenty,czas_trwania_Jacobi_tab,label='Metoda Jacobiego')
plt.plot(argumenty,czas_trwania_Gauss_Seidl_tab,label='Metoda Gaussa-Seidla')
plt.plot(argumenty,czas_trwania_faktoryzacja_LU_tab,label='Faktoryzacja LU')
plt.xlabel('Liczba N')
plt.ylabel('Czas trwania[s]')
plt.title('Zależność czasu trwania poszczególnych algorytmów w zależnośći od N')
plt.legend()
plt.show()
```



Podpunkt F - obserwacje

Dla każdej metody czas wykonanywania algorytmu wzrasta wraz z liczbą niewiadomych, jednakże dla metod iteracyjnych takich jak metoda Jacobiego lub metoda Gaussa-Seidla wzrost ten odbywa się łagodnie w przeciwieństwie do metody faktoryzacji LU, która jest metodą bezpośredniego rozwiązywania układów równań liniowych i obserwowalny wzrost czasu działaniu w tej metodzie jest bardzo duży. Zaletą jednak metody faktoryzacji LU jest to, że za jej pomocą pomimo nawet bardzo długiego czasu działania damy radę rozwiązać każdy układ równań liniowych w przeciwieństwie do metod iteracyjnych, co stało się w podpunkcie C, gdzie mogliśmy zaobserwować przypadek gdzie metody te się nie zbiegają. Metody iteracyjne także nie są aż tak dokładne, lecz czas ich działania jest znacząco mniejszy dla dużych ilości argumentów. Wynika z tego, że każdy rodzaj metod ma swoje wady i zalety, więc powinniśmy dopasowywać używany przez nas algorytm do tego co od niego oczekujemy.