Projekt 2 Metody Numeryczne

Jakub Sachajko 179976

Maj 2021

1 Zadanie A

Stworzyłem układ równań dla indexu 179976 - c=7, d=6, e=9, f=9, dla którego a1= 14 a2 = a3 = -1 i dla którego N = 976. Wektor b o długości N, w którym n-ty element wyliczamy ze wzoru $sin(n * (f + 1))$ przy czym uznałem że numerować zaczynamy od n = 0.

2 Zadanie B

Zaimplementowałem dwie metody iteracyjne rozwiązywania układów równań liniowych: Jacobiego i Gaussa-Seidla. Implementacje wyglądają następująco:
-Dla jacobiego

```
def jacobi(A,b,N,border):

  r=[]
  for i in range(N):
    r.append(1)
  rPrev=[]
  for i in range(N):
    rPrev.append(1)

  check = 1
  numberOfIterations=0
  timeNumbers=0
  zbieg=True

  tic=time.time()
  while check > 0:
    for i in range(N):

      withoutDiag=0
      for j in range(N):
        if j!=i:
          withoutDiag = withoutDiag + (A[i][j] * rPrev[j])
      r[i]=(b[i]-withoutDiag)/A[i][i]
    for i in range(N):
      rPrev[i]=r[i]
    numberOfIterations=numberOfIterations+1
    res = resid(A,r,b,N)
    if norm(res,N) <= border:
```

```
      toc=time.time()
      timeNumbers=toc-tic
      check = 0
    if math.isinf(norm(res,N)) == True:
      toc=time.time()
      timeNumbers=toc-tic
      zbieg=False
      check = -1
```

-Dla Gaussa-Seidla

```
def GaussSeidl(A,b,N,border):

  L=[]
  U=[]
  D=[]
  for i in range(N):
    listOfZeros1 = [0]*N
    L.append(listOfZeros1)
    listOfZeros2 = [0]*N
    U.append(listOfZeros2)
    listOfZeros3 = [0]*N
    D.append(listOfZeros3)

  for i in range(N):
    for j in range(N):
      L[i][j]=0
      U[i][j]=0
      D[i][j]=0
      if i<j:
        U[i][j]=A[i][j]
      if i==j:
        D[i][j]=A[i][j]
      if i>j:
        L[i][j]=A[i][j]

  r=[]
  for i in range(N):
    r.append(1)
  rPrev=[]
  for i in range(N):
    rPrev.append(1)

  check = 1
  numberOfIterations=0
  timeNumbers=0
  zbieg=True
  tic=time.time()
  while check > 0:
    for i in range(N):
      sum=0

      for j in range(i+1,N):
        sum = sum + (U[i][j])*rPrev[j]

      for j in range(i):
        sum = sum + (L[i][j])*r[j]

```

```
            r[i]=(b[i]-sum)/D[i][i]

        for i in range(N):
            rPrev[i]=r[i]
        numberOfIterations=numberOfIterations+1
        res = resid(A,r,b,N)
        if norm(res,N) <= border:
            toc=time.time()
            timeNumbers=toc-tic
            check = 0
        if math.isinf(norm(res,N)) == True:
            toc=time.time()
            timeNumbers=toc-tic
            zbieg=False
            check = -1

    return timeNumbers,numberOfIterations,zbieg
```

Dla układu równań z Podpunktu A Jacobi potrzebował 2 sekundy więcej przy czym miał także większą liczbę iteracji, aż 22. Porównując go do Gaussa-Seidla, którego czas działania wynosił niecałe 3 sekundy i liczba iteracji wynosiła zaledwie 15.

```
Zad B
Czas i iteracje (Jacobi)
czas:  4.701998233795166
Liczba iteracji:  22
-------------
Czas i iteracje (Gaussa-Seidla)
czas:  2.8929994106292725
Liczba iteracji:  15
-------------
```

3 Zadanie C

Następnie zamieniłem wartość a1 na równą 3. Po uruchomieniu programu można pomyśleć, iż program ten zaciął się jednak norma z wektora residuum rośnie, a funkje w tym programie dopiero przy stwierdzeniu, że wartość osiąga "inf" wtedy zwraca komunikat o rozbieżności. W tym przypadku metoda Gaussa-Seidla przy wykonaniu mniejszed ilości iteracji o ponad połowe, zwróci informację zwrotną szybciej o $3/5$ czasu działania dla metody Jacobiego. Wnioski z tego przykładu nasuwają się same. Ponieważ metoda Gaussa-Seidla jest modyfikacją metody jacobiego jest ona szybszą wersją potrzebującą mniejszej ilości iteracji.

```
Zad C
Czas i iteracje (Jacobi)
nie zbiega się
czas:  250.02150130271912
Liczba iteracji:  1222
-------------
Czas i iteracje (Gaussa-Seidla)
nie zbiega się
czas:  99.32700061798096
Liczba iteracji:  509
```

4 Zadanie D

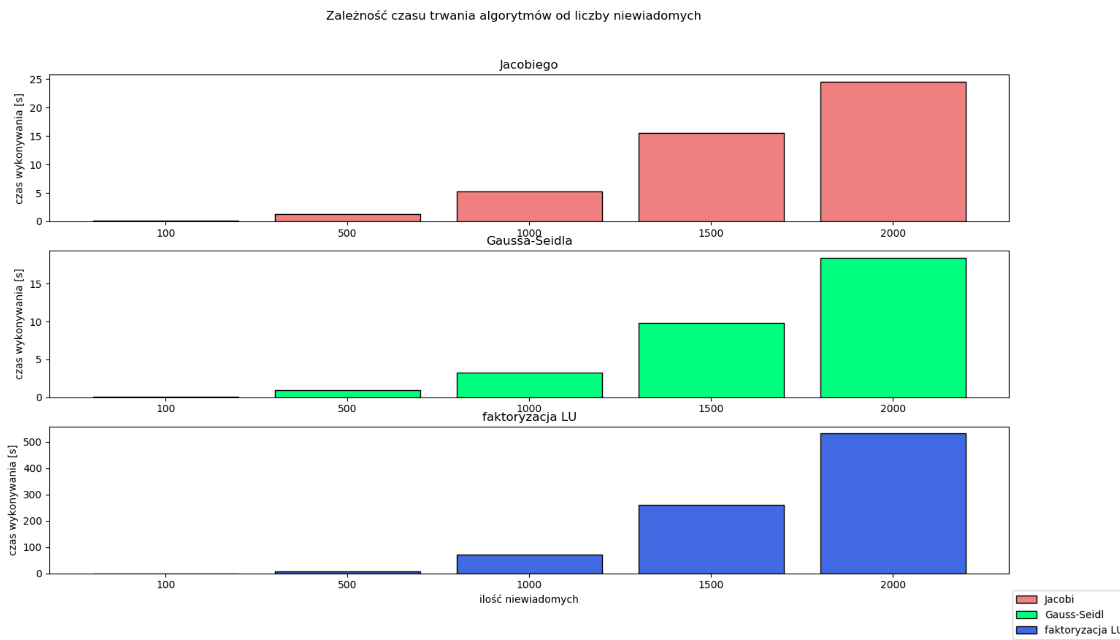
Po zaimplementowaniu metody bezpośredniego rozwiązywania równań układów liniowych, czyli faktoryzacji LU i zastosowaniu jej do przypadku z zadania C otrzymujemy wynik z dokładnością 10^{-13}.

```
Zad D
Norma z residuum (faktoryzacjaLU)
norma z residuum wynosi:  1.6071464184796135e-13
```

Po wykonaniu faktoryzacji LU zadałem sobie pytanie dlaczego jeżeli metody Jacobiego i Gaussa-Seidla nie rozwiązały tego przypadku używa sie ich zamiast faktoryzacji LU. Dlatego po chwilowej analizie zauważyłem, że czas wykonywania faktoryzacji LU zajmuje znacznie więcej czasu, ponieważ liczona jest dokładna wartośc rozwiązania.

5 Zadanie E

Jako ostatnie zadanie wykonane zostały wykresy obrazujące zależności czasu trwania algorytmów w zależności od liczby niewiadomych. Na wykresie zostało zmierzone 5 pomiaró dla ilości niewiadomych $N = 100, 500, 1000, 1500, 2000$ dla przypadku z podpunktu A. Do wykonania ostatniego zadania na potrzeby tworzenia wykresów użyłem biblioteki matplotlib.




```
[100, 500, 1000, 1500, 2000]
[0.06099987030029297, 1.2930002212524414, 5.253500699996948, 15.501498460769653, 24.553000926971436]
[0.02899932861328125, 0.9309988021850586, 3.2755014896392822, 9.832998037338257, 18.432499647140503]
[0.05549979209899902, 7.501000642776489, 69.85400152206421, 260.4800007343292, 531.0875008106232]
```

Jak możemy zauważyć dla $dlaN = 100$ różnice w czasie nie są bardzo znaczące jednak dla kolejnych pomiarów zaczyna wyróżniać się różnica pomiędzy Jacobim i Gaussem-Seidlem, a faktoryzacją LU zaczynając od 7 razy dłuższego czasu wykonywania dla N = 500 do ponad 22 razy dłuższego czasu wykonywania dla N = 2000. Możemy także zauważyć że czas pomiędzy Gaussem-Seidlem a Jacobim także zaczyna się wyróżniać jednak nie jest on tak znaczący. Z wykresów można wywnioskować, że najlepiej próbować rozwiązać układ równań N niewiadomych metodą Gaussa-Seidla ,a jeżeli nie otrzymamy wyniku należy spróbować metody faktoryzacji LU.