Projekt 2 Metody Numeryczne

Układy równań liniowych

Klaudia Ratkowska

1. Układy równań mają postać: **Ax = b**, gdzie A jest macierzą systemową, pasmową, b jest wektorem pobudzenia, a x jest wektorem rozwiązań.
   Powyższy układ równań będzie rozwiązywany za pomocą metod iteracyjnych Jacobiego i Gaussa-Seidla oraz za pomocą bezpośredniej metody faktoryzacji LU. Rozwiązanie przybliżone oznacza się symbolem $\bar{\boldsymbol{x}}$**.** W celu oszacowania błędu w wynikach stosuje się tzw. wektor residuum: $\mathbf{r} = \mathbf{A}\,\bar{\boldsymbol{x}} - \mathbf{b}$**.** Wygodniej jednak przedstawić błąd w postaci skalara, dlatego wylicza się normę residuum. Badając normę wektora residuum, możemy w każdej iteracji algorytmu obliczyć jaki błąd wnosi wektor $\bar{\boldsymbol{x}}$. Przeważnie jako kryterium stopu przyjmuje się normę z residuum o wartości mniejszej niż 10^{-6} .

2. Metoda Jacobiego to metoda iteracyjna, podczas której w kolejnych iteracjach wyznaczamy kolejne przybliżenia rozwiązania układu równań, biorąc pod uwagę poprzednie przybliżenia i elementy macierzy układu. W ogólności:

$$x_i^{(k+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)})/a_{ii}$$

3. Metoda Gaussa-Seidla działa tak jak metoda Jacobiego, jednak w tym przypadku używamy aktualnego przybliżenia x_i. Tym sposobem otrzymujemy:

$$x_i^{(k+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)})/a_{ii}$$

4. Faktoryzacja LU to metoda bezpośrednia, podczas której przy rozwiązywaniu układów równań liniowych uzyskujemy wynik po określonej licznie operacji. L oznacza macierz trójkątną dolną, a U oznacza macierz trójkątną górną. Rozwiązanie liniowego układu równania Ax = b przebiega następująco:
   - tworzymy macierze L i U tak, że LUx = b
   - tworzymy wektor pomocnicy y = Ux
   - rozwiązujemy układ równań Ly = b
   - rozwiązujemy układ równań Ux = y

5. Zadanie A:
   Utworzony został układ równań dla kwadratowej macierzy pasmowej A, gdzie a_1 = 11 (e = 6), a_2 = a_3 = -1 i N = 939 (c = 3, d = 9) oraz dla wektora b, którego długość wynosi N, a jego n-ty element ma wartość sin(n*(8+1)), (f = 8).

```python
size = 939
matrix = Matrix(size)
findCourage = False

#taskA
A = matrix.generate_matrix(11, -1, -1)
b = matrix.generate_vector()
```

```python
def generate_matrix(self, a1, a2, a3):
    A = []
    for i in range(self.N):
        r = []
        for j in range(self.N):
            if i-2 == j or i+2 == j:
                r.append(a3)
            elif i-1 == j or i+1 == j:
                r.append(a2)
            elif i == j:
                r.append(a1)
            else:
                r.append(int(0))
        A.append(r)
    return A

def generate_vector(self):
    vectorB = []
    for i in range(self.N):
        vectorB.append(sin(i*9))
    return vectorB
```

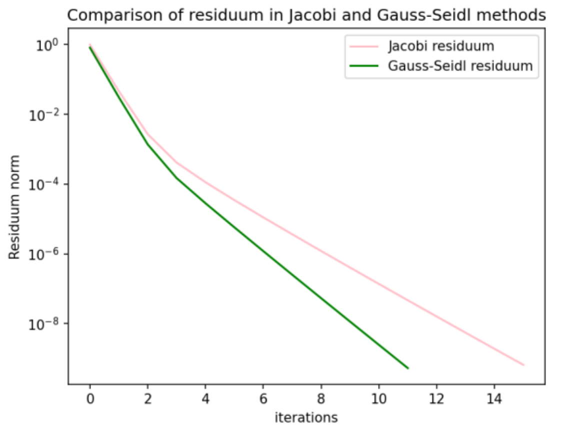
6. Zadanie B:
   Zaimplementowane zostały metody Jacobiego i Gaussa-Seidla. Pierwsza z metod potrzebowała 15 iteracji dla rozwiązania układu z punktu A, a druga metoda potrzebowała tylko 11 iteracji do wykonania tego samego zadania. Szybciej wykonała się metoda Gaussa-Seidla. Rozwiązanie dla obu metod jest dość dokładne, o czym świadczy bardzo mały błąd rezydualny. Na poniższym wykresie porównywany jest błąd rezydualny dla obu metod. Widać z niego, że mniejszy błąd zachodzi dla metody Jacobiego.

```python
def jacobi(A, b, findCourage):
    print("Jacobi method")
    iter = 0
    size = len(A)
    matrix = Matrix(size)
    tempA = matrix.temp_matrix(A)
    tempb = matrix.temp_vector(b)
    vec_x = matrix.vec_zero(len(tempA))
    tempx = matrix.temp_vector(vec_x)
    courage = []

    start = time.time()
    while True:
        for i in range(len(tempA)):
            value = tempb[i]
            for j in range(len(tempA)):
                if i != j:
                    value -= tempA[i][j] * vec_x[j]

            value /= tempA[i][i]
            tempx[i] = value
```

```python
def gauss_seidl(A, b, findCourage):
    print("Gauss-Seidl method")
    iter = 0
    size = len(A)
    matrix = Matrix(size)
    tempA = matrix.temp_matrix(A)
    tempb = matrix.temp_vector(b)
    vec_x = matrix.vec_zero(len(tempA))
    courage = []

    start = time.time()
    while True:
        for i in range(len(tempA)):
            value = tempb[i]
            for j in range(len(tempA)):
                if i != j:
                    value -= tempA[i][j] * vec_x[j]

            value /= tempA[i][i]
            vec_x[i] = value
```

```python
        vec_x = matrix.temp_vector(tempx)
        dot = matrix.dot_product(tempA, vec_x)
        res = matrix.subtract(tempb, dot)
        res_norm = norm(res)
        courage.append(res_norm)
        if iter >= 50 and findCourage :
            return courage

        if res_norm < residuum:
            break

        iter += 1

    finish = time.time() - start
    print("Number of iterations: ", iter)
    print("Time: ", finish)
    print("Residuum: ", norm(res))
    print(" ")
    return finish
```

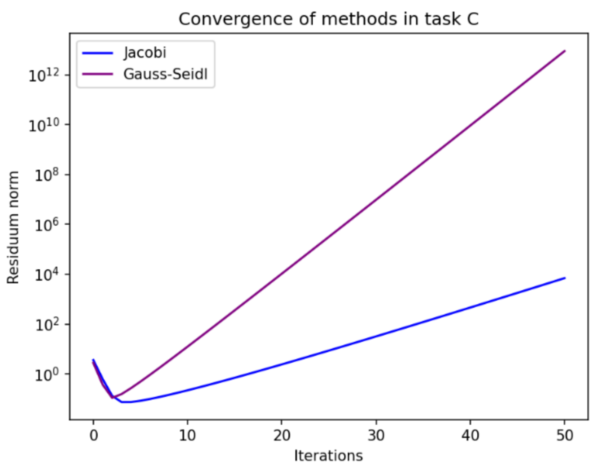
```python
        dot = matrix.dot_product(tempA, vec_x)
        res = matrix.subtract(tempb, dot)
        res_norm = norm(res)
        courage.append(res_norm)
        if iter >= 50 and findCourage :
            return courage

        if res_norm < residuum:
            break

        iter += 1

    finish = time.time() - start
    print("Number of iterations: ", iter)
    print("Time: ", finish)
    print("Residuum: ", norm(res))
    print(" ")
    return finish
```




```
Jacobi method
Number of iterations:  15
Time:  3.2481460571289062
Residuum:  6.617108228596583e-10

Gauss-Seidl method
Number of iterations:  11
Time:  2.874709367752075
Residuum:  5.356210250849924e-10
```

7. Zadanie C:

   Stworzony został układ równań dla macierzy A, gdzie $a_1 = 3$, $a_2 = a_3 = -1$ i $N = 939$. Wektor b zostaje bez zmian. Dla układu równań $Ax = b$ ponownie zostają zastosowane metody iteracyjne Jacobiego oraz Gaussa-Seidla, jednak w tym przypadku nie zbiegają się, co przedstawia poniższy wykres.



8. Zadanie D:
   Zaimplementowana została metoda faktoryzacji LU, zastosowana do przypadku z punktu C. Norma z residuum w tym przypadku wynosi ok. $1.69*10^{-12}$.

```python
def lu_decomposition(C, b):
    print("LU decomposition method")
    size = len(C)
    matrix = Matrix(size)
    tempA = matrix.temp_matrix(C)
    tempb = matrix.temp_vector(b)
    tempx = matrix.vector_one(len(C))
    tempy = matrix.vec_zero(len(C))
    matrixL = matrix.matrixL(matrix.vector_one(len(C)))
    matrixU = matrix.matrix_zero(len(C), len(C))

    start = time.time()
    for i in range(len(C)):
        for j in range(i + 1):
            matrixU[j][i] += tempA[i][j]
            for k in range(j):
                matrixU[j][i] -= matrixL[j][k] * matrixU[k][i]

        for j in range(i + 1, len(C)):
            for k in range(i):
                matrixL[j][i] -= matrixL[j][k] * matrixU[k][i]
            matrixL[j][i] += tempA[j][i]
            matrixL[j][i] /= matrixU[i][i]

    for i in range(len(C)):
        value = tempb[i]
        for j in range(i):
            if i != j:
                value -= matrixL[i][j] * tempy[j]
        tempy[i] = value / matrixL[i][i]

    for i in range(len(C) - 1, -1, -1):
        value = tempy[i]
        for j in range(i + 1, len(C)):
            if i != j:
                value -= matrixU[i][j] * tempx[j]
        tempx[i] = value / matrixU[i][i]

    dot = matrix.dot_product(tempA, tempx)
    res = matrix.subtract(dot, tempb)
    finish = time.time() - start
    print("Time: ", finish)
    print("Residuum: ", norm(res))
    print(" ")
    return finish
```

```
LU decomposition method
Time:  111.38758754730225
Residuum:  1.6796425382591011e-12
```

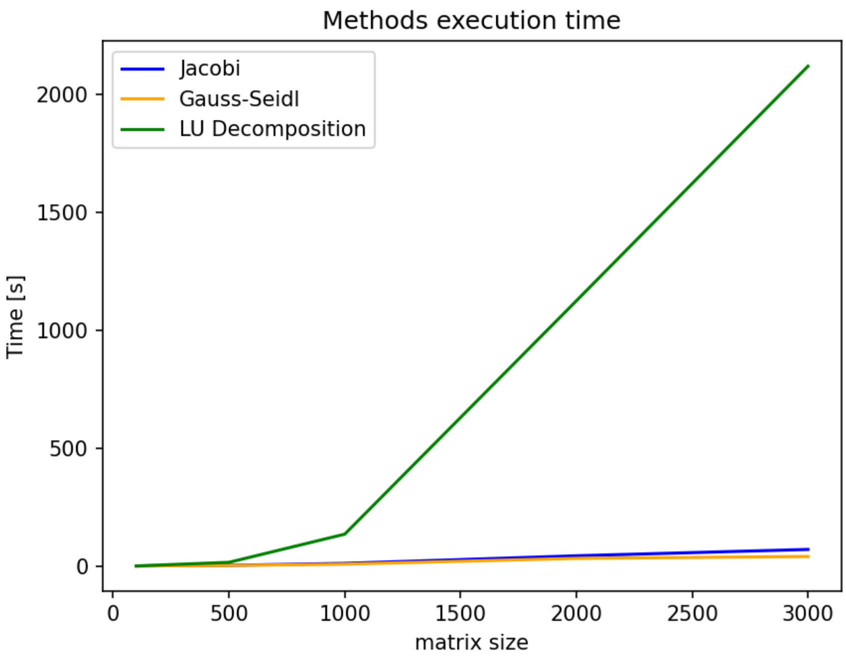
9. Zadanie E:
   Stworzony został wykres dla czasu trwania algorytmów Jacobiego, Gaussa-Seidla i faktoryzacji LU dla kolejnych wartości N = {100, 500, 1000, 2000, 3000} dla przypadku z punktu A.

```
Matrix size:  100
Jacobi method
Number of iterations:  16
Time:  0.12460541725158691
Residuum:  5.829815059437838e-10

Gauss-Seidl method
Number of iterations:  12
Time:  0.09973573684692383
Residuum:  3.4202269574623617e-10

LU decomposition method
Time:  0.1393423080444336
Residuum:  1.2068303035057478e-15
```

```
Matrix size:  500
Jacobi method
Number of iterations:  16
Time:  2.5892937183380127
Residuum:  5.89584718317371e-10

Gauss-Seidl method
Number of iterations:  12
Time:  2.035330295562744
Residuum:  3.4441028792900595e-10

LU decomposition method
Time:  15.365249872207642
Residuum:  2.246514884149351e-15
```

```
Matrix size:  1000
Jacobi method
Number of iterations:  15
Time:  10.938706874847412
Residuum:  7.476791958159261e-10

Gauss-Seidl method
Number of iterations:  11
Time:  7.803574323654175
Residuum:  6.788150500590597e-10

LU decomposition method
Time:  134.93015718460083
Residuum:  3.1204848836209056e-15
```

```
Matrix size:  2000
Jacobi method
Number of iterations:  16
Time:  43.157572984695435
Residuum:  5.252700372892865e-10

Gauss-Seidl method
Number of iterations:  12
Time:  32.06783747673035
Residuum:  3.088519442334586e-10

LU decomposition method
Time:  1124.980729341507
Residuum:  4.4144050960258986e-15
```



10. Zadanie F:
    Najefektywniejsza okazała się metoda Gaussa-Seidla, która wymaga najmniej iteracji oraz najkrótszego czasu do rozwiązania układu równań. Metoda Jacobiego okazała się niewiele wolniejsza. Różnica w czasie rozwiązania układu równań obiema metodami dla macierzy o małych rozmiarach jest niewielka, jednak zwiększa się wraz ze wzrostem rozmiaru macierzy. Metoda Gaussa-Seidla uzyskuje niewiele większy błąd rezydualny, niż metoda Jacobiego. Najwolniejsza okazała się metoda faktoryzacji LU, która dla macierzy o dużych rozmiarach wymaga poświęcenia dużej ilości czasu. Należy jednak pamiętać, że metoda faktoryzacji jest dużo dokładniejsza, niż metody Jacobiego i Gaussa-Seidla, ponieważ uzyskuje najmniejszy błąd rezydualny.