Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №6

Чисельні методи

Розв’язування системи лінійних рівнянь

Варіант 17

Виконала
студентка групи ІТ-91:

Луцай К. А.

Перевірила:

ас. Тимофєєва Ю. С.

Київ 2021

Мета: навчитися розв'язувати систему лінійних рівнянь за допомогою мови програмування Python, використати такі методи: Гауса, прогону, простих ітерацій, Зейделя.

Теоретичні відомості:

Серед чисельних методів алгебри існують прямі методи, у яких розв'язок обчислюється за скінченне фіксоване число операцій та ітераційні методи, у яких результат досягається в процесі послідовних наближень

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

Метод Гауса

У методі Гауса матриця СЛАР за допомогою рівносильних перетворень під час прямого ходу перетворюється у верхню трикутну матрицю.

$$\begin{array}{cccccc} x_1 & x_2 & x_3 & \cdots & x_n & b \\ \hline \end{array}$$
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 & \\ 0 & a_{22}^1 & a_{23}^1 & \cdots & a_{2n}^1 & b_2^1 & \\ 0 & 0 & a_{33}^2 & \cdots & a_{3n}^2 & b_3^2 & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{n-1} & b_n^{n-1} & \end{bmatrix}$$

Під час зворотного ходу визначаються невідомі. З останнього рівняння відразу визначається x n , з передостаннього – x n−1 тощо. З першого рівняння визначається x1.

Метод прогону

Метод прогону є одним з ефективних методів розв'язування СЛАР із трьохдіагональними матрицями, що виникають при кінцево-різницевій апроксимації задач для звичайних диференціальних рівнянь (ЗДР) і рівнянь у частинних похідних другого порядку і є окремим випадком методу Гауса.

$$a_1 = 0 \begin{cases} b_1x_1 + c_1x_2 = d_1 \\ a_2x_1 + b_2x_2 + c_2x_3 = d_2 \\ \qquad a_3x_2 + b_3x_3 + c_3x_4 = d_3 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \qquad\qquad a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n = d_{n-1} \\ \qquad\qquad\qquad a_nx_{n-1} + b_nx_n = d_n, \qquad c_n = 0, \end{cases}$$

Прогоночні коефіцієнти обчислюються за наступними формулами та визначаються при прямому ході:

$$P_i = \frac{-c_i}{b_i + a_i P_{i-1}}, \quad Q_i = \frac{d_i - a_i Q_{i-1}}{b_i + a_i P_{i-1}}, \quad i = \overline{2, n-1};$$

$$P_1 = \frac{-c_1}{b_1}, \quad Q_1 = \frac{d_1}{b_1}, \text{ тому що } a_1 = 0, \quad i = 1;$$

$$P_n = 0, \text{ через } c_n = 0, \; Q_n = \frac{d_n - a_n Q_{n-1}}{b_n + a_n P_{n-1}}, \quad i = n.$$

Зворотній хід методу прогону:

$$\begin{cases} x_n = P_n x_{n+1} + Q_n = 0 \cdot x_{n+1} + Q_n = Q_n \\ x_{n-1} = P_{n-1} x_n + Q_{n-1} \\ x_{n-1} = P_{n-2} x_{n-1} + Q_{n-2} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_1 = P_1 x_2 + Q_1. \end{cases}$$

Метод простих ітерацій

Методи послідовних наближень, у яких при обчисленні наступного наближення розв’язку використовуються попередні, вже відомі, наближені розв’язки, називаються ітераційними.

$$x = \beta + \alpha x.$$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \; \beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \; \alpha = \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & \cdots & \vdots \\ \alpha_{n1} & \ldots & \alpha_{nn} \end{pmatrix}.$$

Як нульове наближення x вектора невідомих приймемо вектор правих частин x = β

$$\beta_i = \frac{b_i}{a_{ii}}; \; \alpha_{ij} = -\frac{a_{ij}}{a_{ii}}$$

$$\begin{cases} x^{(0)} = \beta \\ x^{(1)} = \beta + \alpha x^{(0)} \\ x^{(2)} = \beta + \alpha x^{(1)} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ x^{(k)} = \beta + \alpha x^{(k-1)}. \end{cases}$$

Метод Зейделя

Метод простих ітерацій досить повільно збігається. Для того, щоб прискорити обчислення, можна використовувати метод Зейделя, що полягає в тому, що обчислюючи компонент k +1 x вектора невідомих на (k+1)-й ітерації використовуються вже обчислені на (k+1)-й ітерації.

B – нижня трикутна матриця з діагональними елементами , рівними нулю, а C – верхня трикутна матриця з діагональними елементами, відмінними від нуля, α = B + C . Отже

$$(E-B)x^{k+1}=Cx^{k}+\beta,$$

Розв’язок задачі в аналітичній формі:

$$\begin{cases} 8\cdot x_1+8\cdot x_2-5\cdot x_3-8\cdot x_4=13\\ 8\cdot x_1-5\cdot x_2+9\cdot x_3-8\cdot x_4=38\\ 5\cdot x_1-4\cdot x_2-6\cdot x_3-2\cdot x_4=14\\ 8\cdot x_1+3\cdot x_2+6\cdot x_3+6\cdot x_4=-95 \end{cases}$$

X = [-4, -3, -1, -8]

$$\begin{cases} -6\cdot x_1+5\cdot x_2=51\\ -x_1+13\cdot x_2+6\cdot x_3=100\\ -9\cdot x_2-15\cdot x_3-4\cdot x_4=-12\\ -x_3-7\cdot x_4+x_5=47\\ 9\cdot x_4-18\cdot x_5=-90 \end{cases}$$

X = [-1, 9, -3, -6, 2]

$$\begin{cases} -19\cdot x_1+2\cdot x_2-x_3-8\cdot x_4=38\\ 2\cdot x_1+14\cdot x_2-4\cdot x_4=20\\ 6\cdot x_1-5\cdot x_2-20\cdot x_3-6\cdot x_4=52\\ -6\cdot x_1+4\cdot x_2-2\cdot x_3+15\cdot x_4=43 \end{cases}$$

X = [-2, 2, -4, 1]

Лістинги програм розв’язування задачі:

Початкові задані за умовою значення:

```
import math
import numpy as np

mat1 = [[8, 8, -5, -8],
        [8, -5, 9, -8],
        [5, -4, -6, -2],
        [8, 3, 6, 6]]
stolb1 = [13, 38, 14, -95]

mat2 = [[-6, 5, 0, 0, 0],
        [-1, 13, 6, 0, 0],
        [0, -9, -15, -4, 0],
        [0, 0, -1, -7, 1],
        [0, 0, 0, 9, -18]]
stolb2 = [51, 100, -12, 47, -90]

mat3 = np.array([[-19, 2, -1, -8],
        [2, 14, 0, -4],
        [6, -5, -20, -6],
        [-6, 4, -2, 15]], dtype=float)
stolb3 = np.array([38, 20, 52, 43], dtype=float)
```

В якості аргументів методи приймають матрицю коефіцієнтів та правий вектор рівнянь.

```
def gauss(A, B):
    def swapRows(A, B, i1, i2):
        A[i1], A[i2] = A[i2], A[i1]
        B[i1], B[i2] = B[i2], B[i1]

    def divRow(A, B, i, div):
        A[i] = [a/div for a in A[i]]
        B[i] = B[i]/div

    def combRows(A, B, i1, i2, weight):
        A[i1] = [(a + k*weight) for a, k in zip(A[i1], A[i2])]
        B[i1] += B[i2] * weight

    column = 0
    while (column < len(B)):
        row = None
        for r in range(column, len(A)):
            if row is None or math.fabs(A[r][column]) > math.fabs(A[row][column])
:
                row = r

        if row != column:
            swapRows(A, B, row, column)
        divRow(A, B, column, A[column][column])
        for r in range(column + 1, len(A)):
```

```python
            combRows(A, B, r, column, -A[r][column])

        column += 1

    X = [0 for b in B]
    for i in range(len(B)-1, -1, -1):
        X[i] = int(B[i] - sum(x * a for x, a in zip(X[(i+1):], A[i][(i+1):])))

    return np.array(X, dtype=float)

def progon(A, B):
    n = len(B)
    U, M, D = [], [], [0]
    for i in range(n):
        M.append(A[i][i])
        if i < n-1:
            U.append(A[i][i+1])
        if i > 0:
            D.append(A[i][i-1])

    a, b = [], []
    for i in range(n):
        if i == 0:
            a.append(-U[i] / M[i])
            b.append(B[i] / M[i])
        else:
            y = M[i] + D[i] * a[i-1]
            b.append((B[i] - D[i] * b[i-1]) / y)
            if i != n-1:
                a.append(-U[i] / y)

    X = [0] * n
    X[-1] = int(b[-1])
    for i in reversed(range(n-1)):
        X[i] = int((a[i] * X[i+1] + b[i]))
    return np.array(X, dtype=float)

def simpiter(A, B):
    n = B.shape[0]
    X = np.zeros_like(B)
    for it in range(100):
        x = np.zeros_like(X)
        for i in range(n):
            s = 0
            for j in range(n):
                if j!= i:
                    s += A[i, j]/A[i,i] * X[j]
            x[i] = B[i] / A[i,i] - s
        X = x
    return X
```

```python
def seidel(A, B):
    n = B.shape[0]
    X = np.zeros_like(B)
    while True:
        x = np.copy(X)
        for i in range(B.shape[0]):
            s1 = np.dot(A[i, :i], X[:i])
            s2 = np.dot(A[i, i+1:], X[i+1:])
            x[i] = (B[i] - s1 - s2) / A[i, i]
        if np.allclose(X, x, atol=1e-10, rtol=0.):
            break
        X = x
    return X
```

Допоміжний метод для друкування СЛАР:

```python
def printSLAR(A, B):
    for row in range(len(B)):
        for col in range(len(A[row])):
            if A[row][col] > 0:
                print(" + ", end='')
            if A[row][col] < 0:
                print(" - ", end='')
            if A[row][col] == 0:
                print("      ", end='')
                continue
            print(int(math.fabs(A[row][col])), end='')
            print("x", end='')
            print(col+1, end='')
        print(" =", B[row])
```

Виклик методів для порівняння з точними значеннями

```python
printSLAR(mat1, stolb1)
print("Точний розв'язок:", np.linalg.solve(mat1, stolb1))
print("Метод Гауса:", gauss(mat1, stolb1))

printSLAR(mat2, stolb2)
print("Точний розв'язок:", np.linalg.solve(mat2, stolb2))
print("Метод Прогонки:", progon(mat2, stolb2))

printSLAR(mat3, stolb3)
print("Точний розв'язок:", np.linalg.solve(mat3, stolb3))
print("Метод Простих ітерацій:", simpiter(mat3, stolb3))
print("Метод Зейделя:", seidel(mat3, stolb3))
```

Результати виконання програми:

```
 + 8x1 + 8x2 - 5x3 - 8x4 = 13
 + 8x1 - 5x2 + 9x3 - 8x4 = 38
 + 5x1 - 4x2 - 6x3 - 2x4 = 14
 + 8x1 + 3x2 + 6x3 + 6x4 = -95
Точний розв'язок: [-4. -3. -1. -8.]
Метод Гауса: [-4. -3. -1. -8.]
 - 6x1 + 5x2                  = 51
 - 1x1 + 13x2 + 6x3            = 100
       - 9x2 - 15x3 - 4x4      = -12
             - 1x3 - 7x4 + 1x5 = 47
                   + 9x4 - 18x5 = -90
Точний розв'язок: [-1.  9. -3. -6.  2.]
Метод Прогонки: [-1.  9. -3. -6.  2.]
 - 19x1 + 2x2 - 1x3 - 8x4 = 38.0
 + 2x1 + 14x2       - 4x4 = 20.0
 + 6x1 - 5x2 - 20x3 - 6x4 = 52.0
 - 6x1 + 4x2 - 2x3 + 15x4 = 43.0
Точний розв'язок: [-2.  2. -4.  1.]
Метод Простих ітерацій: [-2.  2. -4.  1.]
Метод Зейделя: [-2.  2. -4.  1.]
```

Висновки: було написано програму для розв’язання системи лінійних рівнянь на мові Python, було реалізовано такі методи: Гауса, прогону, простих ітерацій, Зейделя.