МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительные системы и технологии

Решение нелинейных уравнений

Отчет

по лабораторной работе №2

по дисциплине

Решение системы линейных уравнений итерационным методом и методом Гаусса-Зейделя

РУКОВОДИТЕЛЬ:

_____ Суркова А. С.

СТУДЕНТ:

_____ Соляник Д. Р.

19-ИВТ-2

Работа защищена «___» _____

С оценкой _____

Нижний Новгород 2021

1. Тема лабораторной работы:

Решение системы линейных уравнений итерационным методом и методом Гаусса-Зейделя

Цель лабораторной работы

Закрепление знаний и умений по нахождению решений систем линейных уравнений различными способами.

2. Вариант задания на лабораторную работу

Вариант № 18

Решить систему линейных уравнений методом Гаусса, итерационным методом и методом Гаусса-Зейделя. При необходимости преобразовать систему к диагонально преобладающему виду. Сделать оценку количества итераций для итерационных методов, сравнить. ε=0.001

18. $\begin{cases} 0.63x_1 + 0.05x_2 + 0.15x_3 = 0.34 \\ 0.05x_1 + 0.34x_2 + 0.10x_3 = 0.32 \\ 0.15x_1 + 0.10x_2 + 0.71x_3 = 0.42 \end{cases}$

3. Теоретические сведения и описание лабораторной работы

- *Метод простых итерация (Якоби)*

Пусть система линейных уравнений каким-либо образом приведена к виду

X = Cx + d, где C – некоторая матрица, а d – вектор-столбец.

Выберем произвольно вектор начальных приближений $x^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)$ и построим итерационный процесс:

$$\begin{cases} x_1^{(k+1)} = c_{11} \cdot x_1^{(k)} + c_{12} \cdot x_2^{(k)} + \ldots + c_{1n} \cdot x_n^{(k)} + d_1 \\ x_2^{(k+1)} = c_{21} \cdot x_1^{(k)} + c_{22} \cdot x_2^{(k)} + \ldots + c_{2n} \cdot x_n^{(k)} + d_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n^{(k+1)} = c_{n1} \cdot x_1^{(k)} + c_{n2} \cdot x_2^{(k)} + \ldots + c_{nn} \cdot x_n^{(k)} + d_n \end{cases}$$

Производя итерации, получим последовательность векторов $x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(k)}, \ldots$

Если элементы матрицы C удовлетворяют одному из условий

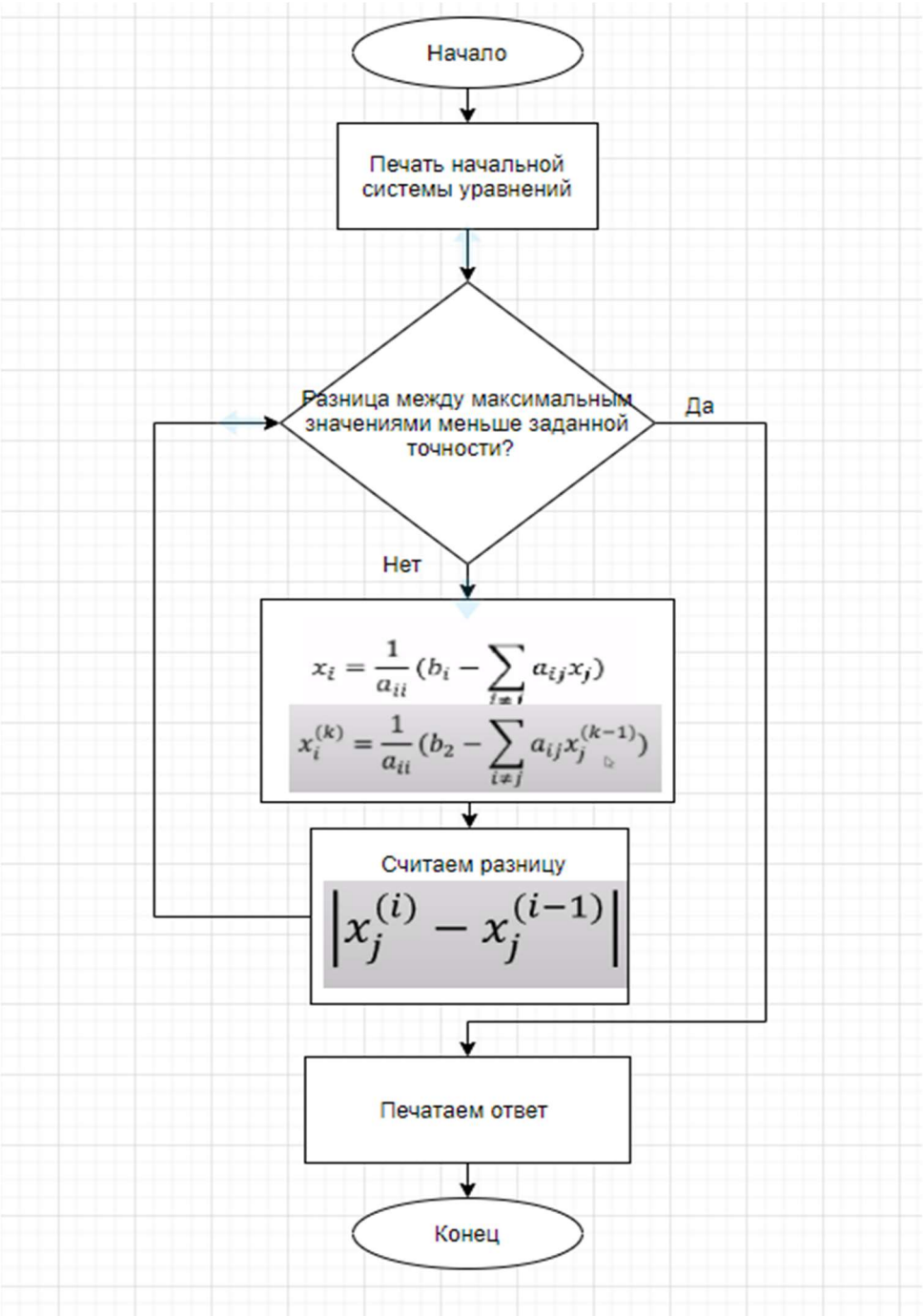
$$\sum_{j=1}^{n} |c_{ij}| \le \alpha < 1, \quad i = 1, 2, \ldots, n$$

$$\sum_{i=1}^{n} |c_{ij}| \le \beta < 1, \quad j = 1, 2, \ldots, n$$

то процесс итераций сходится к точному решению системы x при любом начальном векторе $x^{(0)}$

Точное решение системы получается лишь в результате бесконечного процесса и всякий вектор $x^{(k)}$ из полученной последовательности, является приближенным решением.

Процесс итераций заканчивают, когда достигнута заданная точность: $\left|x^{(k)} - x^{(k-1)}\right| < \varepsilon$.

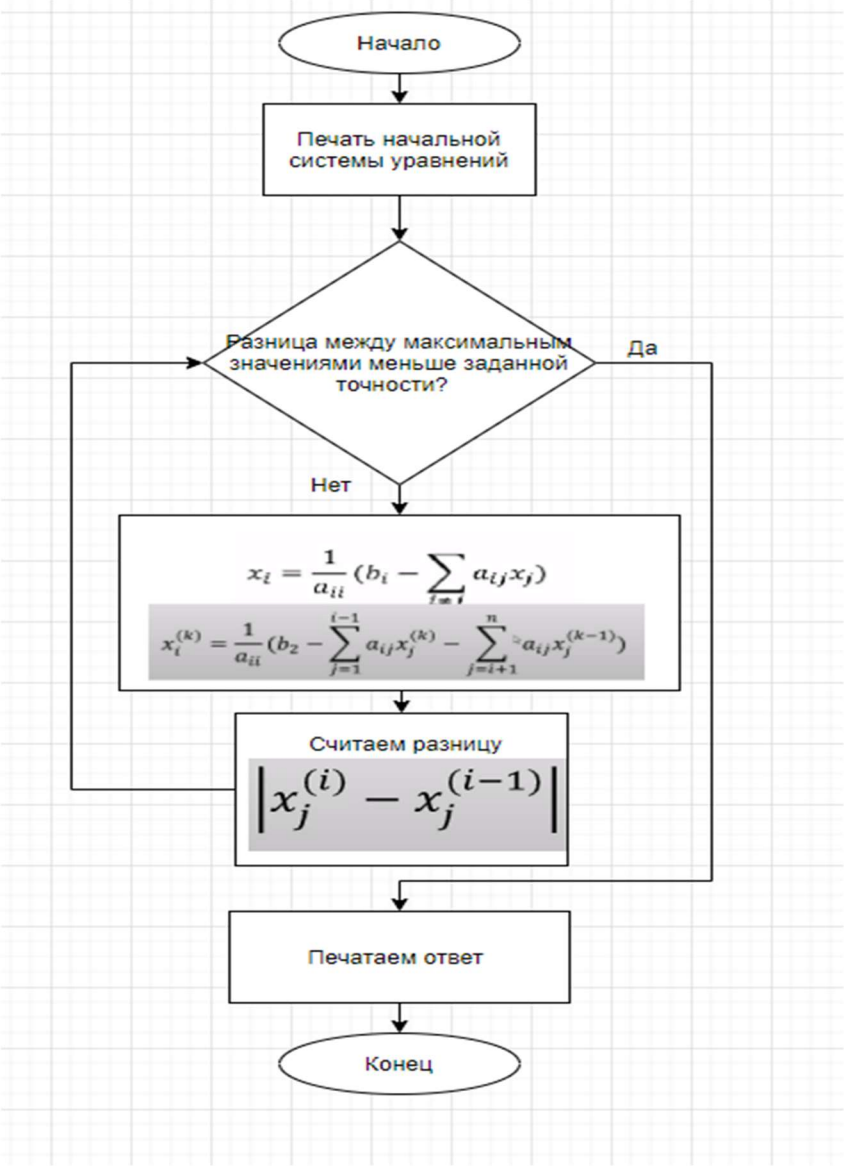


- *Метод Гаусса-Зейделя*

Метод Зейделя является модификацией метода простых итераций. Он заключается в том, что при вычислении (k+1)-ого приближения неизвестного xi (i>1) используется уже вычисленные ранее (k+1)-ое приближение неизвестных $x_1, x_2, \ldots, x_{i-1}$. Т.е. вычисление методу Зейделя ведутся по формулам:

$$\begin{cases} x_1^{(k+1)} = c_{11} \cdot x_1^{(k)} + c_{12} \cdot x_2^{(k)} + \ldots + c_{1n} \cdot x_n^{(k)} + d_1 \\ x_2^{(k+1)} = c_{21} \cdot x_1^{(k+1)} + c_{22} \cdot x_2^{(k)} + \ldots + c_{2n} \cdot x_n^{(k)} + d_2 \\ x_2^{(k+1)} = c_{21} \cdot x_1^{(k+1)} + c_{22} \cdot x_2^{(k+1)} + \ldots + c_{2n} \cdot x_n^{(k)} + d_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n^{(k+1)} = c_{n1} \cdot x_1^{(k+1)} + c_{n2} \cdot x_2^{(k+1)} + \ldots + c_{n-1,n-1} \cdot x_{n-1}^{(k+1)} + c_{nn} \cdot x_n^{(k)} + d_n \end{cases}$$

Условия сходимости метода Зейделя те же, что и для метода простых итераций. Однако, области сходимости методов лишь частично совпадают. В случаях, когда оба метода сходятся, метод Зейделя дает лучшую сходимость, чем метод простых итераций.



- *Метод Гаусса*

Алгоритм метода Гаусса подразделяется на два этапа.

На первом этапе осуществляется прямой ход, когда путём элементарных преобразований над строками систему приводят треугольной форме. Для этого среди элементов первого столбца матрицы выбирают ненулевой, перемещают содержащую его строку в крайнее верхнее положение, делая эту строку первой. Далее ненулевые элементы первого столбца всех нижележащих строк обнуляются путём вычитания из каждой строки первой строки, домноженной на отношение первого элемента этих строк к первому элементу первой строки. После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают, пока не останется матрица нулевого размера.

На втором этапе осуществляется обратный ход, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить фундаментальную систему решений, либо, если все переменные являются базисными, то выразить в численном виде единственное решение системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную и подставляют в предыдущие уравнения, и так далее, поднимаясь наверх. Каждой строчке соответствует ровно одна базисная переменная, поэтому на каждом шаге ситуация в точности повторяет случай последней строки.

4. Листинг разработанной программы

- *Метод простых итерация (Якоби)*

```
import numpy as np
# Задаем начальную систему линейных уравнений
A = np.array([[0.63, 0.05, 0.15],
              [0.05, 0.34, 0.10],
              [0.15, 0.10, 0.71]])

B = np.array([0.34,
              0.32,
              0.42])
# Задаем точность
epsilon = 0.001

# Печатаем матрицу
for i in range ( len(A) ):
    print("| ", end='')
    for j in range ( len(A[i]) ):
        if j < 3:
            print("%.2f" %A[i][j], "* x", j+1 , end='')
        else:
            print("%.2f" %A[i][j], end='')
        j += 1
        if j < 3:
            print(" + ", end='')
        if j == 3:
            print(" = ", B[i], end='')
    i += 1
    print(" |")
print()
```

```
# Задаем начальные нулевые значение x
x = np.zeros_like(B)

for j in range(100):
    if j != 0:
        print("Итерация", j, ":", x)
    # Создаем массив для хранений новых значений x
    x_new = np.zeros_like(x)

    # Алгоритм Якоби
    for i in range(len(A)):
        sum1 = A[i, :i] @ x[:i]
        sum2 = A[i, i + 1:] @ x[i + 1:]
        x_new[i] = (B[i] - sum1 - sum2) / A[i, i]

    # Проверка на достижение точности
    if np.allclose(x, x_new, atol=epsilon):
        break
    # Присваиваем x новые посчитанные значения
    x = x_new

print("\nОтвет:")
for i in range(len(B)):
    print("x", i+1, "=", "%.3f" % x[i])
```

- *Метод Гаусса-Зейделя*

```
import numpy as np
# Задаем начальную систему линейных уравнений
A = np.array([[0.63, 0.05, 0.15],
              [0.05, 0.34, 0.10],
              [0.15, 0.10, 0.71]])

B = np.array([0.34,
```

```
            0.32,
            0.42])
# Задаем точность
epsilon = 0.001

# Печатаем матрицу
for i in range ( len(A) ):
    print("| ", end='')
    for j in range ( len(A[i]) ):
        if j < 3:
            print("%.2f" %A[i][j], "* x", j+1 , end='')
        else:
            print("%.2f" %A[i][j], end='')
        j += 1
        if j < 3:
            print(" + ", end='')
        if j == 3:
            print(" = ", B[i], end='')
    i += 1
    print(" |")
print()

# Задаем начальные нулевые значение x
x = np.zeros_like(B)

for j in range(100):
            # Создаем массив для хранений новых значений x
    x_new = np.zeros_like(x)

    # Алгоритм Гаусса-Зейделя
    for i in range(len(A)):
        sum1 = sum(A[i][j] * x_new[j] for j in range(i))
        sum2 = sum(A[i][j] * x[j] for j in range(i + 1, len(A)))
```

```
        x_new[i] = (B[i] - sum1 - sum2) / A[i][i]


            # Проверка на достижение точности
    if np.allclose(x, x_new, atol=epsilon):
        break
    # Присваиваем x новые посчитанные значения
    x = x_new

    print("Итерация", j + 1, ":", x)

print("\nОтвет:")
for i in range(len(B)):
    print("x", i+1, "=", "%.3f" % x[i])
```

- *Метод Гаусса*

```
import numpy as np
# Задаем начальную систему линейных уравнений
A = np.array([[0.63, 0.05, 0.15, 0.34],
              [0.05, 0.34, 0.10, 0.32],
              [0.15, 0.10, 0.71, 0.42]])

len1 = len(A[:, 0])  # Считаем количество строк матрицы
len2 = len(A[0, :])  # Счмтаем количсетво столбцов матрицы

def printArr(A, len1, len2):
    "=======Функция печати матрицы на экран==================="
    for i in range ( len(A) ):
        print("| ", end='')
        for j in range ( len(A[i]) ):
            if j < 3:
                print("%.2f" %A[i][j], "* x", j+1 , end='')
            else:
                print("%.2f" %A[i][j], end='')
```

```
            j += 1
            if j < 3:
                print(" + ", end='')
            if j == 3:
                print(" = ", end='')
        i += 1
        print(" |")
    print()


def forwardTrace(A, len1, len2):
    "========Функция расчета системы линейных уравнений========"
    for g in range(len1):
        print("=============================================")
        max = abs(A[g][g])        # Максимальное значение для x в столбце
        max_index = g             # Запоминает место нового максимального элемента
        t1 = g

        while t1 < len1: # Цикл поиска максимума
            if abs(A[t1][g]) > max:
                max = abs(A[t1][g])
                max_index = t1
            t1 += 1

        if max_index != g:        # Меняет местами нынешнюю строку со строкой максимума
            A[g][:], A[max_index][:] = A[max_index][:], A[g][:]

        amain = float(A[g][g])    # Значение коэфициента перед текущим x

        z = g
        while z < len2:           # Цикл изменяющий значения главной диагонали на 1, при помощи деления на amain
            A[g][z] = A[g][z] / amain
            z += 1
        printArr(A, len1, len2)
```

```
        j = g + 1
        while j < len1:                    #отнимаем строку, умноженную на коэффицент
            b = A[j][g]                    #  от следующей, в результате получаем столбец нулей.
            z = g                          #    Глобальный цикл выполняется до тех пор, пока
            while z < len2:                #не получатся нули в нижней треугольной матрице
                A[j][z] = A[j][z] - A[g][z] * b
                z += 1
            j += 1
        printArr(A, len1, len2)
    A = backTrace(A, len1)                 #обратный ход метода Гаусса
    return A



def backTrace(A, len1):
    "========Функция обратного хода============================="
    i = len1 - 1
    while i > 0:
        j = i - 1
        while j >= 0:
            A[j][len1] = A[j][len1] - A[j][i] * A[i][len1]
            j -= 1
        i -= 1
    print("================================================")
    return A[:, len1]

print(" Система уравнений:")
printArr(A, len1, len2)
b = forwardTrace(A, len1, len2)
print("Ответ:")
for i in range(len(b)):
    print("x", i+1, "=", "%.3f" % b[i])
```

5. Результаты работы программы

- *Метод Якоби*

```
| 0.63 * x 1 + 0.05 * x 2 + 0.15 * x 3 =  0.34 |
| 0.05 * x 1 + 0.34 * x 2 + 0.10 * x 3 =  0.32 |
| 0.15 * x 1 + 0.10 * x 2 + 0.71 * x 3 =  0.42 |

Итерация 1 : [0.53968254 0.94117647 0.5915493 ]
Итерация 2 : [0.32414092 0.6878263  0.34497179]
Итерация 3 : [0.40295701 0.7920464  0.42619187]
Итерация 4 : [0.37534746 0.75656754 0.3948617 ]
Итерация 5 : [0.38562281 0.76984252 0.40569173]
Итерация 6 : [0.38199066 0.76514614 0.40165116]
Итерация 7 : [0.38332543 0.76686868 0.40307998]

Ответ:
x 1 = 0.383
x 2 = 0.767
x 3 = 0.403
[Finished in 0.3s]
```

- *Метод Гаусса-Зейделя*

```
| 0.63 * x 1 + 0.05 * x 2 + 0.15 * x 3 =  0.34 |
| 0.05 * x 1 + 0.34 * x 2 + 0.10 * x 3 =  0.32 |
| 0.15 * x 1 + 0.10 * x 2 + 0.71 * x 3 =  0.42 |

Итерация 1 : [0.53968254 0.86181139 0.35614997]
Итерация 2 : [0.3864872  0.77959013 0.40009564]
Итерация 3 : [0.38254944 0.76724401 0.40266645]

Ответ:
x 1 = 0.383
x 2 = 0.767
x 3 = 0.403
[Finished in 0.3s]
```

- *Метод Гаусса*

```
 Система уравнений:
| 0.63 * x 1 + 0.05 * x 2 + 0.15 * x 3 = 0.34 |
| 0.05 * x 1 + 0.34 * x 2 + 0.10 * x 3 = 0.32 |
| 0.15 * x 1 + 0.10 * x 2 + 0.71 * x 3 = 0.42 |

=============================================
| 1.00 * x 1 + 0.08 * x 2 + 0.24 * x 3 = 0.54 |
| 0.05 * x 1 + 0.34 * x 2 + 0.10 * x 3 = 0.32 |
| 0.15 * x 1 + 0.10 * x 2 + 0.71 * x 3 = 0.42 |

| 1.00 * x 1 + 0.08 * x 2 + 0.24 * x 3 = 0.54 |
| 0.00 * x 1 + 0.34 * x 2 + 0.09 * x 3 = 0.29 |
| 0.00 * x 1 + 0.09 * x 2 + 0.67 * x 3 = 0.34 |

=============================================
| 1.00 * x 1 + 0.08 * x 2 + 0.24 * x 3 = 0.54 |
| 0.00 * x 1 + 1.00 * x 2 + 0.26 * x 3 = 0.87 |
| 0.00 * x 1 + 0.09 * x 2 + 0.67 * x 3 = 0.34 |

| 1.00 * x 1 + 0.08 * x 2 + 0.24 * x 3 = 0.54 |
| 0.00 * x 1 + 1.00 * x 2 + 0.26 * x 3 = 0.87 |
| 0.00 * x 1 + 0.00 * x 2 + 0.65 * x 3 = 0.26 |

=============================================
| 1.00 * x 1 + 0.08 * x 2 + 0.24 * x 3 = 0.54 |
| 0.00 * x 1 + 1.00 * x 2 + 0.26 * x 3 = 0.87 |
| 0.00 * x 1 + 0.00 * x 2 + 1.00 * x 3 = 0.40 |

| 1.00 * x 1 + 0.08 * x 2 + 0.24 * x 3 = 0.54 |
| 0.00 * x 1 + 1.00 * x 2 + 0.26 * x 3 = 0.87 |
| 0.00 * x 1 + 0.00 * x 2 + 1.00 * x 3 = 0.40 |

=============================================
Ответ:
x 1 = 0.383
x 2 = 0.766
x 3 = 0.403
[Finished in 0.3s]
```

6. Вывод

В ходе лабораторной работы были изучены и применены методы: Гаусса, итерационный и Гаусса-Зейделя при решении системы линейных уравнений.