**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ  
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

по дисциплине: «Численные методы математической физики**»**

на тему: «Разработка программ по методам решения систем линейных алгебраических уравнений (СЛАУ)»

Выполнил: студент гр. ИТП-22

Расшивалов Н.И.  
 Принял: доцент

Стародубцев Е.Г.

Гомель 2021

**Цель работы:** научиться разрабатывать алгоритмы численных методов и программное обеспечение для решения СЛАУ.

**ЗАДАНИЕ**

**Вариант 30**

Разработать алгоритмы и написать программы, реализующие следующие методы решения СЛАУ:

1. метод Гаусса;

2. метод LU-разложения (в двух модификациях: 1) с единицами на главной диагонали матрицы U; 2) с единицами на главной диагонали матрицы L );

3. метод прогонки;

4. метод простой итерации;

5. метод Гаусса-Зейделя.

СЛАУ согласно варианту, представлена на рисунке 1.



Рисунок 1 – Уравнение согласно варианту

Решение уравнения с помощью калькулятора представлено на рисунке 2.

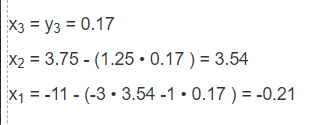


Рисунок 2 – Решение уравнения с помощью калькулятора

Решение методом Гаусса представлено на рисунке 3.



Рисунок 3 – Решение c с помощью метода Гаусса

Решение методом LU-разложения, первая модификация представлено на рисунке 4.

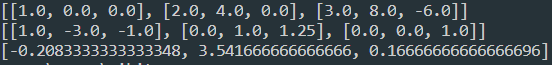


Рисунок 4 – Решение методом LU-разложения, первая модификация

Решение методом LU-разложения, вторая модификация представлено на рисунке 5.

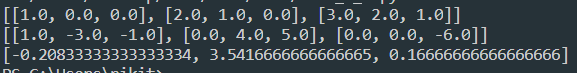


Рисунок 5 – Решение методом LU-разложения, вторая модификация

Решение СЛАУ методом прогонки представлено на рисунке 6.

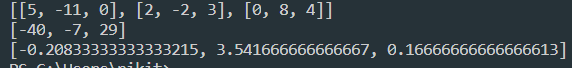


Рисунок 6 – Решение методом прогонки

Решение СЛАУ методом итераций представлено на рисунке 7.



Рисунок 7 – Решение СЛАУ методом итераций

Решение СЛАУ методом Гаусса-Зейделя представлено на рисунке 8.

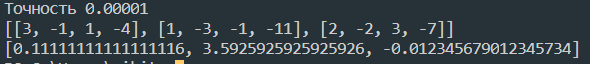


Рисунок 8 – Решение методом Гаусса-Зейделя

Подстановка полученных корней:

Метод Гаусса:

Корни: -0.2083,3.541,0.1666

-10.6 = -11

-7 = -7

-3.9999 = -4

Метод LU-разложения(1):

Корни: -0.2083,3.541,0.1666

-10.6 = -11

-7 = -7

-3.9999 = -4

Метод LU-разложения(2):

Корни: -0.2083,3.541,0.1666

-10.6 = -11

-7 = -7

-3.9999 = -4

Метод простых итераций:

Корни: 0.00000299, 3.99999, 5.00000165

Корни: -0.2083,3.541,0.1666

-10.6 = -11

-7 = -7

-3.9999 = -4

Метод прогонки:

Корни: -0.2083,3.541,0.1666

-10.6 = -11

-7 = -7

-3.9999 = -4

Метод Гаусса-Зейделя:

Корни: 0.111111, 3.5925, -0.01234

-10.6 = -11

-6.99999 = -7

-3.876 = -4

**Вывод**: изучена разработка алгоритмов решения СЛАУ различными методами.

Подставляя полученные решения в уравнения, можно сделать выводы, что методы Гаусса, LU-разложения и прогонки обладают самой высокой точностью, а метод Гаусса-Зейделя является самым неточным по сравнению с остальными.

# **ПРИЛОЖЕНИЕ А**

**Графические схемы алгоритмов**



Рисунок A.1 – Графическая схема алгоритма решения СЛАУ методом Гаусса

matrix – расширенная матрица чисел

n– количество решений

tmp – временная переменная для хранения элементов

i,j – счетчики циклов



Рисунок A.2 – Графическая схема алгоритма решения СЛАУ методом прогонки

matA – матрица

matB – свободные коэффиценты

y – едемент из главной диоганали

a – значения получаемые из формулы -с/y где c элемент побочной диоганали

B – значения получаемы из формулы d/y где d элемент свободный коэффицент

matRes – массив результатов



Рисунок A.5 – Графическая схема решения СЛАУ методом простых итераций

matA – матрица коэффицентов

matB – матрица свободных членов

eps – точность

n – количество корней

res – корни(предыдущая итерация)

current –текущая итерация

flag – проверка точности



Рисунок A.6 – Графическая схема алгоритма получения LU матрицы

A – исходная матрица

LU – матрица разложений



Рисунок A.7 – Графическая схема алгоритма получения решений СЛАУ методом LU-разложения, при единицах в главной диагонали U

L – матрица L

U – матрица U

b – свободные члены

x – решения

y – коэффициенты из формул



Рисунок A.8 – Графическая схема алгоритма получения решений СЛАУ методом LU-разложения, при единицах в главной диагонали L

L – матрица L

U – матрица U

b – свободные члены

x – решения

y – коэффициенты из формул



Рисунок A.3 – Графическая схема решения СЛАУ методом Гаусса-Зейделя (часть I)



Рисунок A.4 – Графическая схема решения СЛАУ методом Гаусса-Зейделя (часть II)

E – точность

matrix – расширенная матрица

n – количество решений

X – решения

C – свободные члены деленные на диагональные коэффициенты

z – количество точных корней

# **ПРИЛОЖЕНИЕ Б**

**Листинг программы**

**#Метод Гаусса**

import math

n=3 # ввод колчиества решений

matrix = [] #расширенная матрица

matrix =[[1, -3, -1,-11],[2,-2,3,-7],[3,-1,1,-4]]

x = list(range(n))

i = 0

while i<n: #прямой ход (приведение матрицы к треугольному виду)

    tmp = matrix[i][i]

    j = n

    while j >= i:

        matrix[i][j] /= tmp #расчет коэффицента для k-ой строки

        j -= 1

    j = i+1

    while j<n:

        tmp = matrix[j][i]

        k = n

        while k >= i:

            matrix[j][k] -= tmp\*matrix[i][k]

            k -= 1

        j += 1

    i += 1

x[n - 1] = matrix[n - 1][n] #последний корень уже есть,с помощью его по очереди находятся остальные

i = n-2 #начало с предпоследней строки

while i >= 0: #обратный ход

    x[i] = matrix[i][n]

    j = i+1

    while j<n:

        x[i] -= matrix[i][j] \* x[j]

        j += 1

    i -= 1

print(matrix)

print(x)

**#Метод LU-разложения 1 способ**

import numpy as np

import copy

def main():

    A = [[1.0, -3.0, -1.0], [2.0, -2.0, 3.0], [3.0, -1.0, 1.0]]

    b = [-11, -7, -4]

    get\_LU(A, 3, b)

def getX2(L,U,b,n):

    # получение y из формулы Ly=b

    y = [0 for i in range(n)]

    y[0] = b[0]

    for i in range(1,n):

        y[i] = b[i]

        for k in range(0,i):

            y[i] -= y[k]\*L[i][k]

    # получение x из формулы Ux=y

    x = [0 for i in range(n)]

    x[n-1] = y[n-1]/float(U[n-1][n-1])

    for i in range(n-2,-1,-1):

        x[i] = y[i]

        for k in range(i+1, n):

            x[i] -= x[k]\*U[i][k]

        x[i]/=U[i][i]

    print(x)

    return x

def get\_LU(A, n, b):

    L = [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]

    U = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]

    for j in range(0, n):

        U[0][j] = A[0][j]

    for i in range(1, n):

        L[i][0] = A[i][0]/U[0][0]

    for i in range(0, n):

        for j in range(0, n):

            if i>j:

                L[i][j] = A[i][j]

                for k in range(0, j):

                    L[i][j] -= L[i][k]\*U[k][j]

                L[i][j] /= U[j][j]

            elif i <= j:

                U[i][j] = A[i][j]

                for k in range(0, i):

                    U[i][j] -= L[i][k]\*U[k][j]

    print(L)

    print(U)

    getX2(L, U, b, 3)

main()

**#Метод LU-разложения 2 способ**

import numpy as np

import copy

def main():

    A = [[1.0, -3.0, -1.0], [2.0, -2.0, 3.0], [3.0, -1.0, 1.0]]

    b = [-11, -7, -4]

    get\_LU(A, 3, b)

def getX2(L,U,b,n):

    # получение y из формулы Ly=b

    y = [0 for i in range(n)]

    y[0] = b[0]

    for i in range(1,n):

        y[i] = b[i]

        for k in range(0,i):

            y[i] -= y[k]\*L[i][k]

    # получение x из формулы Ux=y

    x = [0 for i in range(n)]

    x[n-1] = y[n-1]/float(U[n-1][n-1])

    for i in range(n-2,-1,-1):

        x[i] = y[i]

        for k in range(i+1, n):

            x[i] -= x[k]\*U[i][k]

        x[i]/=U[i][i]

    print(x)

    return x

def get\_LU(A, n, b):

    L = [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]

    U = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]

    for j in range(0, n):

        U[0][j] = A[0][j]

    for i in range(1, n):

        L[i][0] = A[i][0]/U[0][0]

    for i in range(0, n):

        for j in range(0, n):

            if i>j:

                L[i][j] = A[i][j]

                for k in range(0, j):

                    L[i][j] -= L[i][k]\*U[k][j]

                L[i][j] /= U[j][j]

            elif i <= j:

                U[i][j] = A[i][j]

                for k in range(0, i):

                    U[i][j] -= L[i][k]\*U[k][j]

    print(L)

    print(U)

    getX2(L, U, b, 3)

main()

**#Метод прогонки**

import math

n=3 # ввод колчиества решений

m = n+1

matA = [] #матрица

matB = [] #свободные коэффиценты

matA = [[5, -11, 0],[2,-2,3],[0,8,4]]

matB = [-40,-7,29]

y = matA[0][0] #элемент из главной диоганали

a = [] #значения получаемые из формулы -с/y где c элемент побочной диоганали

B = [] #значения получаемы из формулы d/y где d элемент свободный коэффицент

matRes = [] # массив результатов

for i in range(0,n):

    a.append(0)

    B.append(0)

    matRes.append(0)

N1 = n - 1  #n-1, т.к. по-особому обрабатывается последняя строка матрицы

a[0] = -matA[0][1] / y

B[0] = matB[0] / y

i = 1

while i < N1:

    y = matA[i][i] + matA[i][i - 1] \* a[i - 1]

    a[i] = -matA[i][i + 1] / y

    B[i] = (matB[i] - matA[i][i - 1] \* B[i - 1]) / y

    i += 1

matRes[N1] = (matB[N1] - matA[N1][N1 - 1] \* B[N1 - 1]) / (matA[N1][N1] + matA[N1][N1 - 1] \* a[N1 - 1])

i = N1 -1

while i >= 0:

    matRes[i] = a[i] \* matRes[i + 1] + B[i]

    i -= 1

print(matA)

print(matB)

print((matRes))

#Метод простых итераций

def main():

    n=3 # ввод количества решений

    eps=float(input("Точность ")) # ввод точности

    matA = [] #матрица

    matB = [] #свободные коэффиценты

    matA = [[7,-5,1],

            [4,-8,1],

            [7,-5,7]]

    matB = [-19,-29,-18]

    if check(matA):

        iteration(matA,matB,eps,n)

    else:

        print("Матрица не удовлетворяет условию сходимости")

def check(matA): #проверка выполняется ли условие сходимости

    isGood = True

    i = 0

    for line in matA:

        j = 0

        for elem in line:

            sum = 0

            if j!= i:

                sum += elem

            j += 1

        if abs(matA[i][i]) < abs(sum):

            isGood = False

        i += 1

    return isGood

def iteration(matA,matB,eps,n):

    res = [] #массив корней(предыдущая итерация)

    i = 0

    while i<n:

        res.append(matB[i]/matA[i][i])

        i += 1

    current = [] #текущая итерация

    for i in range(0,n):

        current.append(0)

    while(True):

        i = 0

        while i < n: #выполнение итерационных формул,для приближения

            current[i] = matB[i] / matA[i][i]

            j = 0

            while j < n:

                if i != j:

                    current[i] -= matA[i][j] / matA[i][i] \* res[j]

                j += 1

            i += 1

        flag = True #флаг,достигнута ли точность

        i = 0

        while i < n-1: #проверка точности

            if abs(current[i] - res[i]) > eps:

                flag = False

                break

            i += 1

        i = 0

        while i < n: #сохранение последней итерации

            res[i] = current[i]

            i += 1

        if flag == True:

            break

    print(res)

main()

**#Метод Зейделя**

import numpy as np

matrix = []

n = 0

Y = []

E = 0.0

def vvodSort():

    global n

    global Y

    global matrix #расширенная матрица

    global E

    E = float(input("Точность ")) # ввод точности

    n = 3 # ввод количества решений

    matrix =[[1, -3, -1,-11],[2,-2,3,-7],[3,-1,1,-4]]

    i = 0

    max = 0

    Y = list(range(n))

    while i<n:

        j = 0

        while j<n:

            if abs(matrix[j][i])>abs(matrix[i][i]):

                max = j

            j += 1

        k = 0

        while k<n:

            Y[k] = matrix[i][k]

            k += 1

        temp=matrix[i][n]

        matrix[i][n]=matrix[max][n]

        matrix[max][n]=temp

        k = 0

        while k < n:

            matrix[i][k]=matrix[max][k]

            matrix[max][k]=Y[k]

            k += 1

        i += 1

def check(): # функция проверки подходит ли метод

    global n

    global matrix

    z = 0

    i = 0

    while i<n:

        sum = 0

        j = 0

        while j<n:

            sum=sum+abs(matrix[i][j])

            j += 1

        if abs(matrix[i][i])<abs(sum-abs(matrix[i][i])):

            z = 1

            break

        i += 1

    if z == 0:

        z = 1

    return z

def zegeli(): #функция вычисления корней

    global matrix

    global n

    global E

    global Y

    C = []

    for i in range(0,n): #заполнение расширенной матрицы

        C.append([])

        for j in range(0,n+1):

            C[i].append(0)

    X = list(range(n)) # корни

    D = list(range(n)) # свободные члены деленные на диоганальные коэффиценты

    i = 0

    while i<n:

        Y[i]=X[i]=matrix[i][n]/matrix[i][i]

        D[i]=matrix[i][n]/matrix[i][i]

        i += 1

    i = 0

    while i<n:

        j = 0

        while j<n:

            if i==j:

                C[i][j]=0

            else:

                C[i][j]=matrix[i][j]/(-matrix[i][i])

            j += 1

        i += 1

    z = 0 #количество точных корней

    while z!=n:

        i = 0

        while i<n:#сохранение предыдущей итерации

            Y[i]=X[i]

            i += 1

        t=0

        i = 0

        while i<n:

            j = 0

            while j<n:

                t=t+C[i][j]\*X[j]

                X[i]=t+D[i]

                j += 1

            t=0

            i += 1

        z = 0

        i = 0

        while i<n: #проверка на точность вычисленных корней

            if X[i]-Y[i]<E:

                z += 1

            i += 1

    print(matrix)

    print(X)

vvodSort()

if check():

    zegeli()

else:

    print("Не подходит")