**зМИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе № 8**

**по дисциплине «Вычислительная математика»**

**Тема: Решение СЛАУ методом Якоби**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Трофимов В.О. |
| Преподаватель |  | Пуеров Г.Ю. |

Санкт-Петербург

2025

**Задание**

В работе студенты должны найти решение системы линейных уравнений с n неизвестными, заданной матрицей коэффициентов и вектором свободных членов b, методом простых итераций. Выполнение работы включает следующие этапы:

1) с помощью преподавателя определить систему уравнений, которую нужно решить. Привести исходную систему к виду (2.3), пригодному для использования метода простых итераций;

2) задать необходимую точность получения результата (количество знаков мантиссы числа);

3) разработать программу решения задачи на языке С с использованием

подпрограммы-функции MITER из файла MITER.CPP. Функция MITER имеет следующие параметры:

- a - матрица коэффициентов преобразованной к виду (2.3) системы

уравнений размера n\*n , тип float;

- b - вектор свободных членов преобразованной системы размера , тип float;

- x - полученный в результате проведения итераций вектор решения размера , тип ;

- n - размер системы уравнений, тип ;

- ntch - количество знаков после запятой в мантиссе результата, остающихся после округления, тип int, 0 <= ntch <=6 ;

- it - выходной параметр, равный количеству произведенных итераций, тип int.

В качестве значения функции типа возвращается одно из следующих значений:

а) 0 - все нормально, получено решение ;

б) 1 - не выполняются условия сходимости итерационного процесса;

в) 2 - размер ;

г) 3 - значение .

Константа должна быть задана при разработке головной программы аналогично тому, как это делается при выполнении лабораторной работы № 11

## Теоретические положения

При решении системы линейных уравнений методом Якоби сначала проверяется размер подаваемой на вход матрицы, определить является ли она квадратной. После проверяется диагональное преобладание матрицы (одно из условий сходимости метода). Если диагональное преобладание отсутствует, можно попытаться переставить строки для его достижения, или может быть расхождение.

Система преобразуется к виду удобному для итераций диагональный элемент = оставшимся элементам с противоположным знаком и свободному члену

Система линейных уравнений приводится к форме, удобной для итерационного метода Якоби, где каждое уравнение выражается в виде:

, где k – номер итерации



Расшифровка формулы:

1. Все элементы строки, кроме диагонального, делятся на диагональный элемент с противоположным знаком.

2. Свободный член также делится на диагональный элемент.

3. Диагональные элементы матрицы замещаются нулями, так как они больше не участвуют в вычислениях напрямую.

Начальное приближение – вектор правой части. На каждой итерации вычисляем новые значения приближения на основе предыдущих. Итерации продолжаются до тех пор, пока норма разности между текущим и предыдущим приближениями не станет меньше заданной точности *epsilon*.

||xk + 1 -xk|| < *epsilon*

Если процесс сходится, то мы нашли приближения с заданной точностью, если после заданного максимального числа итераций, то сходимость метода не достигнута.

**Выполнение работы**

1. def parse\_args() – функция создаёт и обрабатывает аргументы командной строки, cli

2. def validate\_input(args) – функция проверяет корректность входных данных, полученных из аргументов командной строки.

3. def generate\_diagonal\_dominant (n, interval) - функция генерирует расширенную матрицу диагонально-доминантную A|b. n – размер матрицы, interval – максимальное число интервала для случайной генерации.

4. def prepare\_jacobi\_matrix(n, matrix, b, epsilon) – функция преобразует матрицу для метода Якоби(см. теоретические положения).

5. def jacobi\_solve(n, epsilon, matrix, b) – основная функция реализует метод Якоби

6. def compute\_residual\_norm (A, x, b) – функция рассчитывает невязку решения. A - матрица коэффициентов, x – решения полученный gauss, b – вектор свободных членов. Рассчитывается A\*x – b и после рассчитываем норму (норма вектора) по формуле ||Ax – b|| / ||b||.

7. def convert\_epsilon(digit\_epsilon) – функция конвертирует представление числовое в 10^(-digit\_epsilon).

8. def exploration(n, interval, digit\_epsilon) – функция вычисляет на разных точностях кол-во итераций и строит зависимость количество итераций от точности *epsilon*.

Таблица 1 – Результаты тестирования при точности epsilon 10^-6

|  |  |  |  |
| --- | --- | --- | --- |
| n | Приближения | Точный результат | Данные |
| 10 | -0.14806666  0.03787248  -0.00411593  -0.04580303  -0.04342623  0.15113459  0.10927123  0.02366682  -0.05559853  -0.00774866 | 0.14806665  0.03787254  0.00411599  0.04580306  0.04342626  0.15113462  0.10927127  0.02366689  0.05559846  0.00774859 | Итерации: 14  Невязка:  7.647560e-07 |

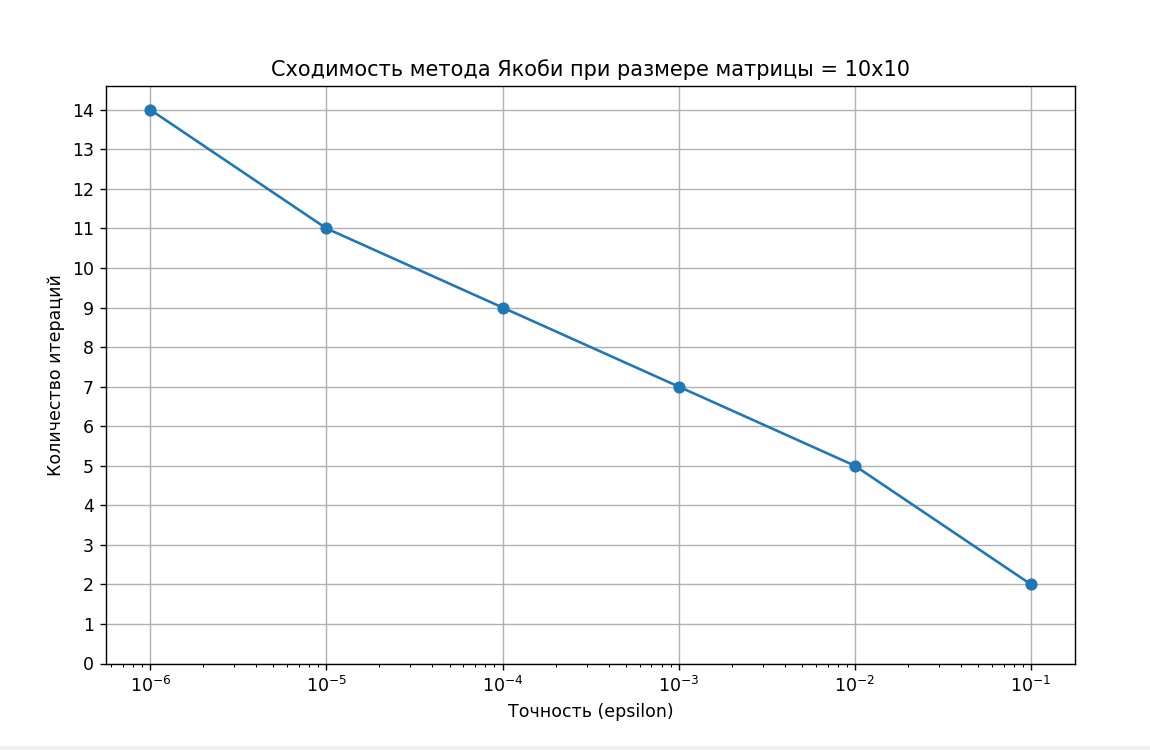


Рисунок № 1 – График зависимости количества итераций от точности

**Выводы**

В результате выполнения данной лабораторной работы был изучен алгоритм решения СЛАУ методом Якоби. Была исследована зависимость количества итераций от точности, сделана таблица в консоли с приближённым решением, точным решением.

# Приложение А Исходный код программы

Название файла: main.py

import random

import scipy

import argparse

import matplotlib.pyplot as plt

from matplotlib.ticker import MultipleLocator

def parse\_args():

parser = argparse.ArgumentParser(

description="Linear system solver and iterations analysis using method Jacobi",

formatter\_class=argparse.RawTextHelpFormatter,

epilog="Example usage \n"

" Research mode:\n"

" python main.py --n <size-matrix> --epsilon 6\n"

" python main.py --n <size-matrix> --interval <digit> --epsilon 6\n"

" Solution mode:\n"

" python main.py --n <size-matrix> --epsilon 6\n"

" python main.py --n <size-matrix> --interval <digit> --epsilon 6\n"

)

subparsers = parser.add\_subparsers(dest='command', required=True)

research\_parser = subparsers.add\_parser(

'research',

help="Compare matrix types and their conditioning"

)

research\_parser.add\_argument(

"--n",

type=int,

required=True,

help="Matrix dimension positive digit"

)

research\_parser.add\_argument(

"--interval",

type=int,

required=False,

default=100,

help="Max random digit (default 100)"

)

research\_parser.add\_argument(

"--epsilon",

type=int,

required=False,

default=6,

help="Last epsilon digit above 0 (default 6)"

)

jacobi\_parser = subparsers.add\_parser(

'solve',

help="Solve specific linear system"

)

jacobi\_parser.add\_argument(

"--n",

type=int,

required=True,

help="Matrix dimension positive digit"

)

jacobi\_parser.add\_argument(

"--interval",

type=int,

required=False,

default=10,

help="Max random digit (default 10)"

)

jacobi\_parser.add\_argument(

"--epsilon",

type=int,

required=True,

help="Epsilon digit above 0"

)

return parser.parse\_args()

def validate\_input(args):

if args.n < 3: raise ValueError("Matrix size must be above 3")

if args.epsilon < 0: raise ValueError("Epsilon value must be above 0")

if (args.command != "solve") and (args.command != "research"): raise ValueError("Command not found try python main.py --help")

if args.interval < 0 or args.interval > 1000000: raise ValueError("Interval must be above 0 or below 1000000")

def generate\_diagonal\_dominant(n, interval):

matrix = [[random.uniform(-interval, interval + 1) for \_ in range(n)] for \_ in range(n)]

for i in range(n):

row\_sum = sum(abs(matrix[i][j]) for j in range(n) if i != j)

matrix[i][i] = row\_sum + random.uniform(1, interval)

b = [random.uniform(-interval, interval) for \_ in range(n)]

return matrix, b

def prepare\_jacobi\_matrix(n, matrix, b, epsilon):

jacobi\_matrix = []

for i in range(n):

if abs(matrix[i][i]) < epsilon:

raise ValueError(f"Матрица вырождена, нулевой элемент в строке {i}")

row = []

for j in range(n):

if i != j: row.append(-matrix[i][j] / matrix[i][i])

else: row.append(0.0)

jacobi\_matrix.append(row)

initial\_approach = [b[i] / matrix[i][i] for i in range(n)]

return jacobi\_matrix, initial\_approach

def jacobi\_solve(n, epsilon, matrix, b):

jacobi\_matrix, initial\_approach = prepare\_jacobi\_matrix(n, matrix, b, epsilon)

approaches = [initial\_approach]

prev\_approach = initial\_approach

iteration = 1

max\_iter = 1000

while iteration < max\_iter:

current\_approach = []

for i in range(n):

sigma = 0.0

for j in range(n):

sigma += jacobi\_matrix[i][j] \* prev\_approach[j]

current\_approach.append(initial\_approach[i] + sigma)

approaches.append(current\_approach)

iteration += 1

if max(abs(current - previous) for current, previous in zip(current\_approach, prev\_approach)) < epsilon:

return current\_approach, approaches, iteration

prev\_approach = current\_approach

print(f"Достигнут лимит итераций {max\_iter} точность не достигнута")

def compute\_residual\_norm(A, x, b):

# calculate A \* x

# residual = Ax - b

# vector norm

n = len(A)

residual = [0.0] \* n

for i in range(n):

Ax = sum(A[i][j] \* x[j] for j in range(n))

residual[i] = Ax - b[i]

residual\_norm = sum(r \*\* 2 for r in residual) \*\* 0.5

b\_norm = sum(b\_i \*\* 2 for b\_i in b) \*\* 0.5

return residual\_norm / b\_norm

def convert\_epsilon(digit\_epsilon):

return float(10 \*\* (-digit\_epsilon))

def exploration(n, interval, digit\_epsilon):

iterations= []

epsilons = []

matrix, b = generate\_diagonal\_dominant(n, interval)

residuals = []

for i in range(1, digit\_epsilon + 1):

epsilon = convert\_epsilon(i)

approach, approaches, iteration = jacobi\_solve(n, epsilon, matrix, b)

residuals.append(compute\_residual\_norm(matrix, approach, b))

epsilons.append(epsilon)

iterations.append(iteration)

for i in range(len(residuals)):

print(f"Residual {residuals[i]:<10.2e} with epsilon {epsilons[i]}")

plt.figure(figsize=(10, 6))

plt.plot(epsilons, iterations, marker='o')

plt.xscale("log")

plt.xlabel("Точность (epsilon)")

plt.ylabel("Количество итераций")

plt.title(f"Сходимость метода Якоби при размере матрицы = {n}x{n}")

plt.grid(True)

plt.gca().yaxis.set\_major\_locator(MultipleLocator(1))

plt.ylim(bottom=0)

plt.show()

def main():

args = parse\_args()

validate\_input(args)

if args.command == "research":

exploration(args.n, args.interval, args.epsilon)

elif args.command == "solve":

matrix, b = generate\_diagonal\_dominant(args.n, args.interval)

epsilon = convert\_epsilon(args.epsilon)

approach, approaches, iteration = jacobi\_solve(args.n, epsilon, matrix, b)

residual = compute\_residual\_norm(matrix, approach, b)

scipy\_solution = scipy.linalg.solve(matrix, b)

print(f"{'Approach:':<20}{'Scipy':<15}{'Diff':<15}")

max\_display = 20

format\_epsilon = f".{args.epsilon + 2}f"

for i in range(min(args.n, max\_display)):

diff = abs(approach[i] - scipy\_solution[i])

print("{:<15} {:<15} {:<15}".format(

f"{approach[i]:{format\_epsilon}}",

f"{scipy\_solution[i]:{format\_epsilon}}",

f"{diff:.2e}"

))

print()

print(f"Residual: {residual}")

print(f"Iterations: {iteration}")

if \_\_name\_\_ == "\_\_main\_\_":

main()