**зМИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе № 7**

**по дисциплине «Вычислительная математика»**

**Тема: Решение СЛАУ методом Гаусса**

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

Санкт-Петербург

2025

**Задание**

В ходе выполнения работы студенты должны найти решение системы линейных уравнений с n неизвестными, заданной матрицей коэффициентов и вектором свободных членов, методом Гаусса. Выполнение работы состоит из следующих этапов:

1) с помощью преподавателя определить систему уравнений, которую нужно решить;

2) для решения системы уравнений разработать программу на языке C++, использующую подпрограмму-функцию. Данная функция имеет следующие параметры:

- a - матрица коэффициентов системы уравнений размера, тип,

- - вектор свободных членов размера, тип,

- - выходной вектор результата решения размера, тип,

- - размер системы (матрицы a и вектора свободных членов), тип.

В разрабатываемой программе должна быть описана константа nmax, равная максимальным размерам используемых матриц и векторов. Функция GAUSS в качестве значения типа возвращает:

а) 0 - в случае нормального завершения процесса вычисления;

б) 1 - в случае вырожденности матрицы;

в) 2 - если n < 2;

г) 3 - если n > n\_max.

Провести вычисления с использованием разработанной программы и исследовать обусловленность задачи с использованием пакета Matlab, при этом для определения числа обусловленности матрицы A рекомендуется использовать функцию cond(A) [14]. Кроме того, для проверки получаемых результатов можно провести вычисления с помощью пакетов Matlab и Derive.

## Теоретические положения

Метод Гаусса применяется для решения систем линейных алгебраических уравнений вида: AX = b, где A – матрица коэффициентов, X – вектор неизвестных, b – вектор свободных членов.

Метод включает в себя два этапа:

1. Прямой ход (преобразование к треугольному виду расширенной матрицы).

Пошаговое исполнение следующих действий:

1. Перестановка строк для выбора главного элемента.

При очередном переходе к очередному столбцу матрицы, мы хотим, чтобы на диагонали (где будет стоять ведущий элемент строки) был ненулевой и наибольший по модулю элемент.

1. Нормировка ведущей строки (деление на главный элемент).

Деление всей строки на главный элемент, чтобы тот элемент стал 1, так как в конечной форме нужно, чтобы на главной диагонали были единицы, что значительно упростит обратных ход

1. Обнуление элементов под главным элементом с помощью вычитания

После того как в строке есть ведущая 1, нужно обнулить все элементы ниже неё в этом столбце, чтобы получилась лестница из нулей (треугольный вид матрицы.

Результат – ступенчатая (треугольная) матрица.

2. Обратный ход (подстановка)

Пошаговое исполнение следующий действий:

1. Вычисление значений неизвестных, начиная с последнего уравнения идём снизу вверх.
2. Последовательно подставляются найденные значения в верхние строки.

Результатом будет вектор X.

**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\_extended\_matrix (n, precision, interval) – функция генерирует cлучайную расширенную матрицу A|b. n – размер матрицы, precision – точность округления генерируемых чисел с плавающей точкой, interval – максимальное число интервала для случайной генерации.

4. def generate\_hilbert\_matrix (n, precision, interval) - функция генерирует расширенную матрицу Гилберта A|b. n – размер матрицы, precision – точность округления генерируемых чисел с плавающей точкой, interval – максимальное число интервала для случайной генерации.

5. def generate\_diagonal\_dominant (n, precision, interval) - функция генерирует расширенную матрицу диагонально-доминантную A|b. n – размер матрицы, precision – точность округления генерируемых чисел с плавающей точкой, interval – максимальное число интервала для случайной генерации.

6. def straight\_elimination (n, matrix) – функция выполняет прямой ход метода Гаусса. n – размер матрицы, matrix – расширенная матрица.

7. def inverse\_step (n, matrix) – функция выполняет обратных ход метода Гаусса. n – размер матрицы, matrix – расширенная матрица.

8. 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||.

9. def compute\_conditional\_digit (A, x, b) – функция рассчитывает число обусловенности. A – матрица коэффициентов., x – решения полученный gauss, b – вектор свободных членов. Для нахождения обратной матрицы используем numpy.linalg.inv, и нормы матрицы, обратной матрицы рассчитывем с помощью numpy.linalg.norm.

10. def gauss(matrix, n, precision) – функция реализовывает метод Гаусса.

11. def exploration (n, precision, interval) – функция реализует исследование обусловленности используя метод гаусса для случайной, Гилберта, диагонально-доминантной матриц.

Таблица 1 – Результаты тестирования[0,100]

|  |  |  |  |
| --- | --- | --- | --- |
| Структура матрицы | n | Результат метода | Результат scipy |
| Случайно заполненная | 10 | -3.187673  3.102832  5.973351  2.364191  -2.545562  -1.836932  -3.30574  -2.946898  3.04243  3.014531 | -3.187673  3.102832  5.973351  2.364191  -2.545562  -1.836932  -3.30574  -2.946898  3.04243  3.014531 |
| Гилберта | 10 | 661657.899253  -17149519.731887  105278404.352221  -230389968.36512  149415279.911849  57342585.725192  -22240822.331564  -7186251.976397  -124132237.043717  88456560.615814 | 661657.899253  -17149519.731887  105278404.352221  -230389968.36512  149415279.911849  57342585.725192  -22240822.331564  -7186251.976397  -124132237.043717  88456560.615814 |
| Диагонально-Доминант | 10 | 0.153425  0.036785  0.037043  0.091631  0.075457  -0.041543  0.041208  0.161672  0.0325830  -7.6e-05 | 0.153425  0.036785  0.037043  0.091631  0.075457  -0.041543  0.041208  0.161672  0.0325830  -7.6e-05 |

**Исследование обусловленности**

Число обусловленности матрицы определяет чувствительность решения системы линейных уравнений к погрешностям исходных данных. Оно показывает насколько функция чувствительна к изменениям или ошибкам во входных, а значит и во выходных данных.

Число обусловленности определяется структурой матрицы, а не её размером.

Таблица 2 – Результаты влияния структуры матрица на обусловленность при интервале плавающих чисел [0,100]

|  |  |  |  |
| --- | --- | --- | --- |
| Структура матрицы | n | Невязка | Обусловленность |
| Случайно заполненная | 10 | 4.004118e-16 | 55.21394 |
| Гилберта | 10 | 2.710732e-11 | 15501390.02557 |
| Диагонально-Доминант | 10 | 1.004868e-16 | 3.205407 |
| Случайно заполненная | 100 | 3.681436e-15 | 786.83650 |
| Гилберта | 100 | 8.020396e-12 | 13984636.86177 |
| Диагонально-Доминант | 100 | 5.785133e-16 | 2.328148 |
| Случайно Заполненная | 1000 | 7.707717e-13 | 100676.50461 |
| Гилберта | 1000 | 7.694521e-12 | 23186029.63982 |
| Диагонально-Доминант | 1000 | 1.710987e-15 | 2.128605 |

Матрицы были взяты Гилберта (плохо обусловленная), диагонально-доминантная (хорошо обусловленная), случайная, по результатам видим, что размер матрицы не зависит на обусловленность и относительную невязку.

Экспериментальные данные подтверждают, что число обусловленности определяется структурой матрицы, а не её размером. Так, диагонально-доминантные матрицы сохраняют cond(A)~10^0 при увеличении n до 1000, в то время как матрицы Гильберта достигают cond(A)~10^7 уже при n=10. Это объясняет разницу в невязках на 5 порядков величины между разными типами матриц одного размера.

Из данных видно, что большая обусловленность порождает большую невязку, нельзя точно сказать, что если одна обусловленность больше другой, то невязка будет иметь подобное отношение, так как на это влияет множество других параметров, которые здесь не учитываются, но в часто такую ситуацию можно наблюдать на системах с маленькой обусловленностью - невязка будет маленькая, а на системах с большой обусловленностью – невязка будет более крупного масштаба.

**Выводы**

В ходе выполнения задания была написана программа на языке Python с CLI, выполняющая решение системы линейных уравнений через метод Гаусса. Программа получает через командную строку входные данные, по которым генерирует расширенную матрицу.

Метод был протестирован с помощью встроенного scipy.linalg.solve метода, который выдаёт точный результат решения системы. Реализованный и встроенный совпали.

Проведено исследование обусловленности, оценили насколько погрешность входных данных может влиять на решение системы линейных уравнений. Также была рассчитана относительная невязка. Реализован эксперимент с случайными, Гилберта и диагонально-доминантными матрицами.

Замечено было влияние числа обусловленности на невязку.

# Приложение А Исходный код программы

Название файла: main.py

import random

import scipy

import numpy as np

import argparse

def parse\_args():

parser = argparse.ArgumentParser(

description="Linear system solver and matrix condition analysis using method Gaussuian",

formatter\_class=argparse.RawTextHelpFormatter,

epilog="Example usage \n"

" Exploration mode:\n"

" python main.py --n <size-matrix> --precision <digit>\n"

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

" Solution mode:\n"

" python main.py --n <size-matrix> --precision <digit>\n"

" python main.py --n <size-matrix> --precision <digit> --type <matrix-type> \n"

" python main.py --n <size-matrix> --precision <digit> --type <matrix-type> --interval <digit>\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(

"--precision",

type=int,

required=True,

help="Computational precision (decimal places)"

)

research\_parser.add\_argument(

"--interval",

type=int,

required=False,

default=100,

help="Max random digit (default 100)"

)

gauss\_parser = subparsers.add\_parser(

'solve',

help = "Solve specific linear system"

)

gauss\_parser.add\_argument(

"--n",

type=int,

required=True,

help="Matrix dimension positive digit"

)

gauss\_parser.add\_argument(

"--precision",

type=int,

required=True,

help="Computational precision (decimal places)"

)

gauss\_parser.add\_argument(

"--type",

type=str,

required=False,

choices=["random", "hilbert", "diagonal-dominant"],

default="random",

help="Matrix type:\n"

" random - random matrix\n"

" hilbert - matrix Hilbert\n"

" diagonal-dominant - matrix diagonal-dominant\n"

" default (random)"

)

gauss\_parser.add\_argument(

"--interval",

type=int,

required=False,

default=100,

help="Max random digit (default 100)"

)

return parser.parse\_args()

def validate\_input(args):

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

if args.precision < 0: raise ValueError("Precision cannot be negative")

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

def generate\_extended\_matrix(n, precision, interval):

matrix = [[round(random.uniform(0, interval), precision) for \_ in range(n)] for \_ in range(n)]

vectorB = [round(random.uniform(0, interval), precision) for \_ in range(n)]

return [matrix[i] + [vectorB[i]] for i in range(n)]

def generate\_hilbert\_matrix(n, precision, interval):

# H\_ij = 1 / i + j - 1 индексация с 1

matrix = [[round(1 / (i + j - 1), precision) for j in range(1, n + 1)] for i in range(1, n + 1)]

b = [round(random.uniform(0, interval), precision) for \_ in range(n)]

return [matrix[i] + [b[i]] for i in range(n)]

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

# Генерируем обычную матрицу после делаем так чтобы условие a\_ii >= оставшихся чисел в строке

matrix = [[round(random.uniform(0, interval), precision) for \_ in range(n)] for \_ in range(n)]

for i in range(n):

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

matrix[i][i] += round(row\_sum, precision)

b = [round(random.uniform(0, interval), precision) for \_ in range(n)]

return [matrix[i] + [b[i]] for i in range(n)]

def straight\_elimination(n, matrix):

row\_count = n

column\_count = n + 1

for pivot\_column in range(n):

# search max row in current column

max\_row = pivot\_column

for row in range(pivot\_column + 1, row\_count):

if abs(matrix[row][pivot\_column]) > abs(matrix[max\_row][pivot\_column]):

max\_row = row

if abs(matrix[max\_row][pivot\_column]) == 0:

raise ValueError("Система вырождена - ведущий элемент нулевой")

# swap rows

matrix[pivot\_column], matrix[max\_row] = matrix[max\_row], matrix[pivot\_column]

# normalize lead row

pivot = matrix[pivot\_column][pivot\_column]

for i in range(pivot\_column, column\_count):

matrix[pivot\_column][i] /= pivot

for row in range(pivot\_column + 1, row\_count):

factor = matrix[row][pivot\_column]

for col in range(pivot\_column, column\_count):

matrix[row][col] -= factor \* matrix[pivot\_column][col]

def inverse\_step(n, matrix):

x = [0.0] \* n

for i in range(n - 1, -1, -1):

if abs(matrix[i][i]) == 0: continue

x[i] = matrix[i][n]

for c in range(i + 1, n):

x[i] -= matrix[i][c] \* x[c]

return x

def gauss(matrix, n, precision):

A = [row[:-1] for row in matrix]

b = [row[-1] for row in matrix]

straight\_elimination(n, matrix)

my\_solution = inverse\_step(n, matrix)

solution\_scipy = scipy.linalg.solve(A, b)

return {

"custom\_solution": [round(x, precision) for x in my\_solution],

"scipy\_solution": [round(x, precision) for x in solution\_scipy],

"residual\_norm": compute\_residual\_norm(A, my\_solution, b),

"condition\_number\_custom": round(compute\_conditional\_digit(A, my\_solution, b), precision),

"condition\_number\_scipy": round(np.linalg.cond(A), precision)

}

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 compute\_conditional\_digit(A, x, b):

# cond2(A) = norm(A) \* norm(inv\_A)

A\_inv = np.linalg.inv(A)

norm\_A = np.linalg.norm(A, ord=2)

norm\_A\_inv = np.linalg.norm(A\_inv, ord=2)

return norm\_A \* norm\_A\_inv

def exploration(n, precision, interval):

# Исследование обусловленности таблица матрицы Диагонально-преобладающей, Гилберта, Случайно сгенерированной

names = ["Random", "Hilbert", "Diagonal-dominant"]

generators = [

generate\_extended\_matrix,

generate\_hilbert\_matrix,

generate\_diagonal\_dominant

]

results = []

for name, generator in zip(names, generators):

matrix = generator(n, precision, interval)

gauss\_result = gauss(matrix, n , precision)

results.append({

"matrix\_type" : name,

"residual\_norm": gauss\_result["residual\_norm"],

"condition\_number\_custom": gauss\_result["condition\_number\_custom"],

"condition\_number\_scipy": gauss\_result["condition\_number\_scipy"]

})

print(f"{'Matrix Type':<25}{'Residual Norm':<30}{'Cond Num (Custom)':<25}{'Cond Num (Scipy)'}")

print("-" \* 128)

for result in results:

print(f"{result['matrix\_type']:<25}{result['residual\_norm']:<30}{result['condition\_number\_custom']:<25}{result['condition\_number\_scipy']:<25}")

def main():

args = parse\_args()

validate\_input(args)

if args.command == "research":

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

elif args.command == "solve":

matrix\_generator = {

"random": generate\_extended\_matrix,

"hilbert": generate\_hilbert\_matrix,

"diagonal-dominant": generate\_diagonal\_dominant

}[args.type]

system = matrix\_generator(args.n, args.precision,args.interval)

solution = gauss(system, args.n, args.precision)

custom = solution['custom\_solution']

scipy = solution['scipy\_solution']

print(f"{'Index':<6}{'Custom Solution':<20}{'Scipy Solution':<20}")

print("-" \* 46)

for i, (custom, scipy) in enumerate(zip(custom, scipy), start=1):

print(f"x{i:<5}{str(custom):<25}{str(scipy):<30}")

print("-" \* 46)

print(f"{'Residual norm:':<30}{solution['residual\_norm']:.5e}")

print(f"{'Custom condition number:':<30}{solution['condition\_number\_custom']:.5e}")

print(f"{'Scipy condition number:':<30}{solution['condition\_number\_scipy']:.5e}")

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

main()

iterations\_chord = explore\_iterations\_chord(left, right, eps\_values\_reversed)

iterations\_newton = explore\_iterations\_newton(x0, eps\_values\_reversed)

iterations\_simple\_iter = explore\_simple\_iter(x0, eps\_values\_reversed)

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

# График для метода Бисекции

plt.plot(eps\_values\_reversed, iterations\_bisection, marker="o", color="green", label="Метод Бисекции")

# График для метода Хорд

plt.plot(eps\_values\_reversed, iterations\_chord, marker='o', color='blue', label='Метод Хорд')

# График для метода Ньютона

plt.plot(eps\_values\_reversed, iterations\_newton, marker='o', color='red', label='Метод Ньютона')

# График для метода простых итераций

plt.plot(eps\_values\_reversed, iterations\_simple\_iter, marker='o', color='orange', label="Метод простых итераций")

# Настройки графика

plt.xscale('log')

plt.yscale('linear')

plt.xlabel('Eps (точность)')

plt.ylabel('Число итераций')

plt.title('Зависимость числа итераций от точности Eps')

plt.yticks(range(0, 21, 1))

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

def main():

epsilons = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6]

plot\_iterations(0, 1, 1, epsilons)

explore\_error(0, 1, epsilons)

main()