МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 1

ОТЧЕТ

о выполненном задании

студента 202 учебной группы факультета ВМК МГУ

Савиных Юрия Сергеевича

гор. Москва

2020 год

Содержание

Цель работы

Изучить два вида решения систем линейных алгебраических уравнений: метод Гауса без выбора главного элемента и метод Гауса с выбором главного элемента

Постановка задачи

Дана система уравнений $Ax = f$, с невырожденной матрицей коэфицентов A размера $n \times n$. Написать программу, которая решает заданную систему линейных алгебраических уравнений методом Гауса и методом Гауса с выбором главного элемента. Предусмотреть возможность задания элементов матрицы коэфицентов и вектора-столбца правой части как во входном файле данных, так и путем задания специальных формул.

Задачи практической работы

1. Решить заданную СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента;

2. Вычислить $\det A$;

3. Вычислить A^{-1};

4. Определить число обусловленности $M_A = ||A|| \cdot ||A^{-1}||$;

5. Исследовать вопрос вычислительной устойчивости метода Гаусса (при больших значениях параметра n);

6. Правильность решения СЛАУ подтвердить системой тестов;

Алгоритм

Метод Гауса без выбора главного элемента

Метод Гаусса решения уравнения Ax = f с невырожденной матрицей A, удобно разделить на два этапа.

1. Прямой ход: система приводится к треугольному виду. Будем считать, что элемент $a_{11} \neq 0$, иначе поменяем местами первую строчку и i, такую, что $a_{i1} \neq 0$ (такой элемент найдется т.к. $\det A \neq 0$. Далее делим уравнение на a_{11}. После этого из каждого следующего уравнения вычтем первое, умноженное на коэфицент a_{i1}. Таким образом останется только одно уравнение с x_1. Выделяем из текущей системы, систему из $n-1$ уравнения с перменными $x_2, ..., x_n$. Повторяем вышеописанные действия для редуцированной системы. Спустя $n-1$ шаг получим систему $Cx = y$, где C - верхняя треугольная матрица с единицами на диагоналях.

2. Обратный ход: последовательное отыскание неизвестных $x_1, x_2, ..., x_n$ из этой треугольной системы.

Метод Гауса с выбором главного элемента

Проделываются практически все те же шаги, что и в Методе Гауса без выбора главного элемента, но теперь делим не на a_{ii} на i шаге, а на максимальный коэфицент в уравнении (строке) (для этого будем переставлять столбцы в матрице). В результате этого изменения коэфиценты треугольной матрцы будут удовлетворять неравенству:

$$c_{ij} \leq 1, \text{где } i, j = 1, 2, ..., n$$

В результате роль ошибок округления в процессе вычисления будет нивелироваться

Описание программы

Код программы и тесты можно найти на
https://github.com/YuriSavinykh/Numerical-Methods/tree/main/Task1/1

- main.py - для вычисления необходимой велечины нужно запустить эту программу интерпретатором python 3, с установленной библиотекой numpy.

- read.py - модуль в котором содержаться вспомогательные функции для считывания входных данных. В случае примеров данные считываются из файлов ./tests/i.txt, где i - номер теста ($i = 1, 2, 3$). При $i = 4$ данные читаются из файла ./tests/i.txt, в который пользователь может вписать свои входные данные. В текстовых файлах формат входных данных таков: сначала число n, следующее $n \times n$ чисел - матрица коэфицентов, далее идут n чисел - координаты вектора f.

- calc.py - модуль в котором содержаться все основные и вспомогательные фунции вычисления необходимых значений.

- output.py - модуль в котором реализованы форматный вывод матрицы и вектора.

При начале работы программы необходимо выбрать тип задания элементов матрицы и правой части. Далее необходимо ввести, что нужно найти:

1. Определитель
2. Обратную матрицу
3. Число обусловленности
4. Решение системы линейных алгебраических уравнений методом Гауса без выбора главного элемента
5. Решение системы линейных алгебраических уравнений методом Гауса с выбора главного элемента

Код программы

Код программы и полный вывод её работы для приложения 2 примера 2-2 можно найти на https://github.com/YuriSavinykh/Numerical-Methods/tree/main/Task1/1

```
import read
import output
import calc

if __name__ == '__main__':
    A, F = read.input_data_of_problem()

    mode = 0
    while mode < 1 or mode > 5:
        mode = int(input("What should i calculate? (1 - determinate, 2 - Invertible matrix, 3 - Conditional number, \n"
                         "4 - solve linear equation without choosing main element, 5 - solve linear equation with "
                         "choosing main element)\n"))
        if (mode == 1):
            print("Determinate(A) = " + str(calc.det(A) * read.norm ** A[0].size) + '\n')
            break
        elif (mode == 2):
            output.print_float_matrix(calc.inverse_matrix(A) / read.norm, "Inverse Matrix")
            break
        elif (mode == 3):
            print("Conditional number = " + str(calc.conditional_number(A * read.norm)) + '\n')
            break
        elif (mode == 4):
            output.print_float_vector(calc.solve(A, F, False), "Solution:")
            break
        elif (mode == 5):
            output.print_float_vector(calc.solve(A, F, True), "Solution with main element:")
            break
```

Листинг 1: main.py

```python
from math import sin
import numpy as np
import output

norm = 1
BIG_NUMBER = 1000000


def read_from_file():
    test_number = 0

    while test_number != 1 and test_number != 2 and test_number != 3 and test_number != 4:
        test_number = int(input("Which example should i test (4 - custom): "))

    if (test_number == 4):
        print("input your custom input in ./tests/4.txt in format: first number is n, next n \times n numbers - A, last n "
              "number - f")

    file = open("./tests/" + str(test_number) + ".txt", "r")

    numbers_pull = np.array(list(map(float, file.read().split())))

    n = int(numbers_pull.item(0))
    numbers_pull = np.delete(numbers_pull, 0, 0)

    A = np.empty((n, n))

    for i in range(n):
        for j in range(n):
            A[i][j] = numbers_pull.item(0)
            numbers_pull = np.delete(numbers_pull, 0, 0)

    F = np.empty((n))

    for i in range(n):
        F[i] = numbers_pull.item(0)
        numbers_pull = np.delete(numbers_pull, 0, 0)

    if (np.max(A) > BIG_NUMBER or np.max(F) > BIG_NUMBER):
        print("Matrix A or vector F has element greater then", BIG_NUMBER, "\n. They were normalized")

        global norm
        if (np.max(A) > BIG_NUMBER):
            norm = np.linalg.norm(A)
        else:
            norm = np.linalg.norm(A)

        A /= norm
        F /= norm

        mode = '0'
        while mode.lower() != 'y' and mode.lower() != 'n':
```

```
        mode = input("Do you want to see normalized A and F?(Y or N): ")

        if (mode.lower() == 'y'):
            print("norm:\n", norm, "\n")
            output.print_float_matrix(A, "normalized A:")
            output.print_float_vector(F, "normalized F:")

    return A, F


def read_as_formula():
    n = 40

    print("My formula: example 2-2")
    m = 2

    q = 1.001 - 2 * m * 10 ** (-3)

    A = np.empty((n, n))

    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if i != j:
                A[i - 1][j - 1] = q ** (i + j) + 0.1 * (j - i)
            else:
                A[i - 1][j - 1] = (q - 1) ** (i + j)

    x = float(input("Input x = "))

    F = np.empty(n)

    for i in range(1, n):
        F[i - 1] = abs(x - n / 10) * i * sin(x)

    return A, F


def input_data_of_problem():
    mode = int(input("Mode (1 - matrix from example or custom matrix, 2 - matrix formula) = "))

    while mode != 1 and mode != 2:
        print("Wrong mode")
        mode = int(input("Mode (1- matrix from example or custom matrix, 2 - matrix formula) = "))

    if (mode == 1):
        return read_from_file()
    else:
        return read_as_formula()
```

Листинг 2: read.py

```python
import numpy as np


def det(A):
    A = A.copy()

    if (A[:, 0].size != A[0].size) :
        raise Exception('Matrix is not square')
    n = A[:, 0].size
    sign = 0
    det = 1

    for i in range(n):
        shift = 0

        while i + shift < n and A[i + shift][i] == 0:
            shift += 1

        if i + shift >= n:
            return 0
        elif shift != 0:
            A[i], A[i + shift] = np.copy(A[i + shift]), np.copy(A[i])
            sign += 1

        switch_index = i
        for j in range(i + 1, n):
            if abs(A[i][j]) > abs(A[i][i]):
                switch_index = j

        if switch_index != i:
            sign += 1
            A[:, i], A[:, switch_index] = np.copy(A[:, switch_index]), np.copy(A[:, i])


        det *= A[i][i]
        A[i] /= A[i][i]

        for j in range(i + 1, n):
            A[j] -= A[j][i] * A[i]

    for i in range(n):
        det *= A[i][i]

    if (sign % 2 == 1):
        det *= -1


    return det


def inverse_matrix(A):
    A = A.copy()
    if (det(A) == 0):
        raise Exception("det(A) == 0")
```

```
    E = np.eye(A[0].size)
    A = np.hstack((A, E))
    ORDER = forward_elimination(A, True)
    back_substitution(A)
    A = A[:, A[0].size
    ORDER = ORDER[:, 0]
    ORDER.shape = (A[0].size, 1)
    ORDER.shape = (A[0].size, 1)
    B = np.hstack((A, ORDER))
    B = B[B[:, A[0].size].argsort()]
    A = B[:, :A[0].size]

    return A


def conditional_number(A):
    B = inverse_matrix(A)

    return np.linalg.norm(B) * np.linalg.norm(A)


def create_extended_matrix(A, F):
    F.shape = (A[0].size, 1)
    B = np.hstack((A, F))
    F.shape = (A[0].size)

    return B


def forward_elimination(A, with_main_elem):
    n = A[:, 0].size
    ORDER = np.empty((n, 2))

    for i in range(n):
        ORDER[i][0] = i

    for i in range(n):
        shift = 0

        while i + shift < n and A[i + shift][i] == 0:
            shift += 1

        if i + shift >= n:
            raise Exception('Det(A) = 0')
        elif shift != 0:
            A[i], A[i + shift] = np.copy(A[i + shift]), np.copy(A[i])

        if (with_main_elem):
            switch_index = i
            for j in range(i + 1, n):
                if abs(A[i][j]) > abs(A[i][i]):
                    switch_index = j

            if switch_index != i:
```

```
                A[:, i], A[:, switch_index] = np.copy(A[:, switch_index]), np.copy(A[:, i])
                ORDER[i], ORDER[switch_index] = np.copy(ORDER[switch_index]), np.copy(ORDER[i])

        A[i] /= A[i][i]

        for j in range(i + 1, n):
            A[j] -= A[j][i] * A[i]
    return ORDER


def back_substitution(A):
    n = A[:, 0].size

    for i in range(n - 1, -1, -1):
        for j in range(i):
            A[j] -= A[i] * A[j][i]

    return


def solve(A, F, with_main_element):
    if det(A) == 0:
        raise Exception("det(A) == 0")

    B = create_extended_matrix(A, F)
    X_WITH_ORDER = forward_elimination(B, with_main_element)
    back_substitution(B)

    for i in range(A[0].size):
        X_WITH_ORDER[i][1] = B[i][A[0].size]


    X_WITH_ORDER = X_WITH_ORDER[X_WITH_ORDER[:, 0].argsort()]
    X = X_WITH_ORDER[:, 1]

    return X
```

Листинг 3: calc.py

```python
def print_float_matrix(A, com=""):
    if com != '':
        print(com)

    for row in A:
        for val in row:
            print("{:13.10f}".format(val), end=' ')
        print()
    print()


def print_float_vector(A, com=""):
    if com != '':
        print(com)

    for val in A:
        print("{:13.10f}".format(val), end=' ')
    print('\n')
```

Листинг 4: output.py

Тестирование программы

Найденный вектор x, в каждом тесте был сравнён с решением, полученным с помощью функции numpy.linalg.solve.

Приложение 1. Вариант 9.

Первая система

$$A = \begin{pmatrix} 2 & -5 & 3 & 1 \\ 3 & -7 & 3 & -1 \\ 5 & -9 & 6 & 2 \\ 5 & -1 & 7 & 8 \end{pmatrix}, f = \begin{pmatrix} 5 \\ -1 \\ 7 \\ 8 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} -1 & 0 & 0.(3) & 0.(3) \\ -1 & 0 & 0.(6) & -0.(3) \\ -1 & 0.1(6) & 1.0(5) & -0.9(4) \\ 1 & -0.5 & -0.5 & 0.5 \end{pmatrix}$$

$$\det(A) = 18$$

Решение системы методом Гаусса без выбора главного элемента:

$$x = (0, -3, -5.(3), 6)^T$$

Решение системы методом Гаусса с выбором главного элемента:

$$x = (0, -3, -5.(3), 6)^T$$

Число обусловленности $=$ 48.80516821448774

Вторая система

$$A = \begin{pmatrix} 4 & 3 & -9 & 1 \\ 2 & 5 & -8 & -1 \\ 2 & 16 & -14 & 2 \\ 2 & 3 & -5 & -11 \end{pmatrix}, f = \begin{pmatrix} 9 \\ 8 \\ 24 \\ 7 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 0.6889400922 & -1.3640552995 & 0.2534562212 & 0.2327188940 \\ 0.1175115207 & -0.5069124424 & 0.1820276498 & 0.0898617512 \\ 0.2396313364 & -0.7788018433 & 0.1751152074 & 0.1244239631 \\ 0.0483870968 & -0.0322580645 & 0.0161290323 & -0.0806451613 \end{pmatrix}$$

$$\det(A) = -868$$

Решение системы методом Гаусса без выбора главного элемента:

$$x = (3, 2, 1, -0)^T$$

Решение системы методом Гаусса с выбором главного элемента:

$$x = (3, 2, 1, 0)^T$$

Число обусловленности $=$ 53.46443604446608

Третья система

$$A = \begin{pmatrix} 12 & 14 & -15 & 24 \\ 16 & 18 & -22 & 29 \\ 18 & 20 & -21 & 32 \\ 10 & 12 & -16 & 20 \end{pmatrix}, f = \begin{pmatrix} 5 \\ 8 \\ 9 \\ 14 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 1 & 2.(8) & -1.(6) & -2.7(2) \\ -2.5 & -4.(6) & 3.5 & 4.1(6) \\ -0 & -0.(4) & 0.(3) & 0.(1) \\ 1 & 1 & -1 & -1 \end{pmatrix}$$

$$\det(A) = 35.99999999999987$$

Решение системы методом Гаусса без выбора главного элемента:

$$x = (2.(2), -1.(6), -0.(1), -0)^T$$

Решение системы методом Гаусса с выбором главного элемента:

$$x = (2.(2), -1.(6), -0.(1), 0)^T$$

Число обусловленности = 707.0817701154748

Приложение 2. Вариант 2-2

$q_2 = 0.997, n = 40$

$$A_{ij} = \begin{cases} \mathrm{q}_2^{i+j} + 0.1 \cdot (j-i) & , i \neq j \\ (\mathrm{q}_2 - 1)^{i+j} & , i = j \end{cases}$$

$f = \begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_n \end{pmatrix}$, где $f_i = |x - \frac{n}{10}| \cdot i \cdot \sin x$

Приведу результаты теста для $x = 0$ и $x = 1$ (Для краткости не привожу обратную матрицу):

x = 0

$\det A = 19.517633865342454$

Число обусловленности $= 518.9447615060827$

Методом Гауса без выбора главного элемента:

$(-0.0000000000, -0.0000000000, -0.0000000000, -0.0000000000, -0.0000000000,$
$-0.0000000000, -0.0000000000, -0.0000000000, -0.0000000000, -0.0000000000,$
$-0.0000000000, -0.0000000000, -0.0000000000, -0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, -0.0000000000)^T$

Методом Гауса с выбором главного элемента:

$(-0.0000000000, -0.0000000000, -0.0000000000, -0.0000000000, -0.0000000000,$
$-0.0000000000, -0.0000000000, -0.0000000000, -0.0000000000, -0.0000000000,$
$-0.0000000000, -0.0000000000, -0.0000000000, -0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, -0.0000000000)^T$

x = 1

$\det A = 19.517633865342454$

Число обусловленности $= 518.9447615060827$

Методом Гауса без выбора главного элемента:

$(2.3595296535, 2.0942032358, 1.8254101132, 1.5530958876, 1.2772272715,$
$0.9977706640, 0.7146922395, 0.4279578659, 0.1375331246, -0.1566166739,$
$-0.4545265092, -0.7562316514, -1.0617676621, -1.3711703858, -1.6844759735,$
$-2.0017208778, -2.3229418285, -2.6481758829, -2.9774603824, -3.3108329753,$

$-\,3.6483316439, -3.9899946571, -4.3358605801, -4.6859683502, -5.0403571960,$
$-\,5.3990666246, -5.7621365190, -6.1296070921, -6.5015188517, -6.8779126097,$
$-\,7.2588295991, -7.6443113362, -8.0343996571, -8.4291367837, -8.8285652381,$
$-9.2327279059, -9.6416680357, -10.0554292199, -10.4740554066, 117.5150349211,)^T$

Методом Гауса с выбором главного элемента:
$(2.3595296368, 2.0942032356, 1.8254101158, 1.5530958909, 1.2772272640,$
$0.9977706630, 0.7146922387, 0.4279578621, 0.1375331225, -0.1566166747,$
$-\,0.4545265098, -0.7562316519, -1.0617676600, -1.3711703862, -1.6844759772,$
$-\,2.0017208770, -2.3229418291, -2.6481758787, -2.9774603750, -3.3108329735,$
$-\,3.6483316387, -3.9899946459, -4.3358605838, -4.6859683573, -5.0403571892,$
$-\,5.3990666232, -5.7621365259, -6.1296070898, -6.5015188351, -6.8779126131,$
$-\,7.2588296076, -7.6443113385, -8.0343996636, -8.4291367817, -8.8285652347,$
$-9.2327279108, -9.6416680463, -10.0554292293, -10.4740554015, 117.5150349309)^T$

Выводы

В ходе работы был рассмотрен метод Гаусса без выбора главного элемента и с выбором главного элемента. Реализован алгоритм нахождения определителя матрицы, обратной матрицы, числа обусловленности матрицы, решения системы линейных алгебраических уравнений.

Если вычислять обратную матрицу в Приложении 2 примере 2-2 и прямой ход выполнять без выбора главного элемента, то полученная матрица будет сильно отличаться от искомой из-за погрешностей округления. Также в этом же тесте решение с выбором главного элемента и без него немного отличаются (из-за погрешностей округления).

Из исследованных примеров становиться видно, что при увеличении числа n (числа уравнений) роль ошибок округления будет расти.

Содержание

Цель работы

Изучить итерационные методы решения систем линейных алгебраических уравнений, а именно методы Зейделя и верхней релаксации. Изучить скорость сходимости этих методов в зависимости от выбора итерационного параметра ω.

Постановка задачи

Дана система $Ax = f$, где A - невырожденная матрица коэфицентов размера $n \times n$, f - вектор-столбец правых значений. Необходимо написать программу, численно решающую систему линейный алгебраических уравнений с помощью метода верхней релаксации (а при параметре $\omega = 1$ метод Зейделя). Предусмотреть возможность задания элементов матрицы системы и ее правой части как во входном файле данных, так и путем задания специальных формул.

Задачи практической работы

1. Решить заданную СЛАУ итерационным методом Зейделя или методом верхней релаксации;

2. Разработать критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы СЛАУ с заданной точностью;

3. Изучить скорость сходимости к точному решению задачи (при использовании итерационного метода верхней релаксации провести эксперименты с различными значениями итерационного параметра)

4. Правильность решения СЛАУ подтвердить системой тестов.

Алгоритм

Рассмотрим произвольную квадратную матрицу:

$$A = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nn} \end{pmatrix}$$

Разложим её на сумму трёх матриц: $A = D + T_H + T_B$, где D - диагональная часть A, T_H - нижняя треугольная часть A, T_B - верхняя треугольная часть A, т.е. матрицы:

$$D = \begin{pmatrix} a_{11} & 0 & ... & 0 \\ 0 & a_{22} & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & a_{nn} \end{pmatrix}, T_H = \begin{pmatrix} 0 & 0 & ... & 0 \\ a_{21} & 0 & ... & 0 \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & 0 \end{pmatrix}, T_B = \begin{pmatrix} 0 & a_{12} & ... & a_{1n} \\ 0 & 0 & ... & a_{2n} \\ ... & ... & ... & ... \\ 0 & 0 & ... & 0 \end{pmatrix}$$

Введём параметр ω и запишем рекурентное соотношение:

$$(D + \omega T_H)\frac{x_{k+1} - x_k}{\omega} + Ax_k = f$$

Если перейти от векторной записи к покомпонентной записи, то можно получить уравнений для вычисления x_i^{k+1}:

$$x_i^{k+1} = x_i^k + \frac{\omega}{a_{ii}}(f_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i}^{n} a_{ij}x_j^k), \text{ где } i = 1, ..., n$$

Для оценки расстояние между найденым на k иттерации решением x_k и решением СЛАУ x используется невзяка: $\psi_k = Ax_k - f$. Обозначим $z_k = x_k - x$ Тогда:

$$\psi_k = Ax_k - f = A(z_k + x) - f = Az_k$$

Таким образом:

$$||x_k - x|| = ||z_k|| \leq ||A^{-1}|| \cdot ||\psi_k||$$

В качестве критерия остановки взято $||A^{-1}|| \cdot ||\psi_k|| < \varepsilon$, который позволяет найти решение с заранее заданной точностью ε.

Описание программы

Программа разделена на четыре модуля

- main.py - для вычисления решения СЛАУ необходимо запустить эту программу интерпретатором python 3, с установленной библиотекой numpy.

- read.py - модуль в котором содержаться вспомогательные функции для считывания входных данных. В случае примеров данные считываются из файлов ./tests/i.txt, где i - номер теста ($i = 1, 2, 3$). При $i = 4$ данные читаются из файла ./tests/i.txt, в который пользователь может вписать свои входные данные. В текстовых файлах формат входных данных таков: сначала число n, следующее $n \times n$ чисел - матрица коэфицентов, далее идут n чисел - координаты вектора f.

- calc.py - модуль в котором содержаться все основные и вспомогательные фунции вычисления необходимых значений.

- output.py - модуль в котором реализованы форматный вывод матрицы и вектора.

Глобальная константа eps = 1.0e-08 в модуле main.py задаёт точность решения. За начальный вектор всегда принят нулевой вектор.

В модуле calc.py определена функция `def get_converging_linear_system(A, F)` , которая возвращает $A^T A$ и $A^T F$. В результате этих операций будет полученая система с тем же решением, причём итерационный метод, применяемый к этой системе сойдётся при любом начальном векторе.

Код программы

Код программы и тесты можно найти на
https://github.com/YuriSavinykh/Numerical-Methods/tree/main/Task1/2

```python
import read
import calc
import output

eps = 1.0e-08

if __name__ == '__main__':
    A, F = read.input_data_of_problem()

    x, i = calc.solve(A, F, eps)

    output.print_float_vector(x, "Solution: ")
    print("Iterations = ", i)
```

Листинг 1: main.py

```python
import numpy as np

MAX_ITERATION = 1000000000


def get_next(A, F, w, prev):
    next = np.empty((prev.size))

    for i in range(prev.size):
        sum1 = 0.0

        for j in range(i):
            sum1 += A[i][j] / A[i][i] * next[j]

        sum2 = 0.0
        for j in range(i, prev.size):
            sum2 += A[i][j] / A[i][i] * prev[j]
        next[i] = prev[i] + w * (F[i] / A[i][i] - sum1 - sum2)
    return next


def criteria(B):
    tmp1, _ = np.linalg.eig(B)
    tmp1 = tmp1.astype(complex)
    tmp1 = np.abs(tmp1)
    tmp1 = np.max(tmp1)
    return tmp1 < 1


def get_converging_linear_system(A, F):
    A = np.copy(A)
    F = np.copy(F)

    B = A.transpose()

    F = np.dot(B, F)
```

```
    A = np.dot(B, A)

    return A, F


def solve(A, F, eps):
    if np.linalg.det(A) == 0:
        raise Exception("det(A) == 0")

    w = float(input("Input w: "))

    if w <= 0 or w >= 2:
        raise Exception("w must be: 0 < w < 2")

    A, F = get_converging_linear_system(A, F)

    x = np.zeros(F.size)

    norm_reversed_A = np.linalg.norm(np.linalg.inv(A))

    residual = np.dot(A, x) - F
    accuracy = norm_reversed_A * np.linalg.norm(residual)

    i = 0

    while accuracy >= eps:
        if i == MAX_ITERATION:
            print("Slow convergence i = ", i)
            break

        x = get_next(A, F, w, x)
        residual = np.dot(A, x) - F
        accuracy = norm_reversed_A * np.linalg.norm(residual)

        i += 1

    return x, i
```

Листинг 2: calc.py

```
from math import sin
import numpy as np


def read_from_file():
    test_number = 0

    while test_number != 1 and test_number != 2 and test_number != 3 and test_number != 4:
        test_number = int(input("Which example should i test (4 - custom): "))

    if (test_number == 4):
        print("input your custom input in ./tests/4.txt in format: first number is n, next n * n numbers - A, last n "
              "number - f")
```

```
    file = open("./tests/" + str(test_number) + ".txt", "r")

    numbers_pull = np.array(list(map(float, file.read().split())))

    n = int(numbers_pull.item(0))
    numbers_pull = np.delete(numbers_pull, 0, 0)

    A = np.empty((n, n))

    for i in range(n):
        for j in range(n):
            A[i][j] = numbers_pull.item(0)
            numbers_pull = np.delete(numbers_pull, 0, 0)


    F = np.empty((n))

    for i in range(n):
        F[i] = numbers_pull.item(0)
        numbers_pull = np.delete(numbers_pull, 0, 0)

    return A, F


def read_as_formula():
    n = 40

    print("My formula: example 2-2")
    m = 2

    q = 1.001 - 2 * m * 10 ** (-3)

    A = np.empty((n, n))

    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if i != j:
                A[i - 1][j - 1] = q ** (i + j) + 0.1 * (j - i)
            else:
                A[i - 1][j - 1] = (q - 1) ** (i + j)

    x = float(input("Input x = "))

    F = np.empty(n)

    for i in range(1, n):
        F[i - 1] = abs(x - n / 10) * i * sin(x)

    return A, F


def input_data_of_problem():
    mode = int(input("Mode (1 - matrix from example or custom
   matrix, 2 - matrix formula) = "))

```

```
    while mode != 1 and mode != 2:
        print("Wrong mode")
        mode = int(input("Mode (1- matrix from example or custom
   matrix, 2 - matrix formula) = "))
        print(mode)

    if (mode == 1):
        return read_from_file()
    else:
        return read_as_formula()
```

Листинг 3: read.py

```
def print_float_matrix(A, com=""):
    if com != '':
        print(com)

    for row in A:
        for val in row:
            print("{:13.10f}".format(val), end=' ')
        print()
    print()


def print_float_vector(A, com=""):
    if com != '':
        print(com)

    for val in A:
        print("{:13.10f}".format(val), end=' ')
    print()
    print()
```

Листинг 4: output.py

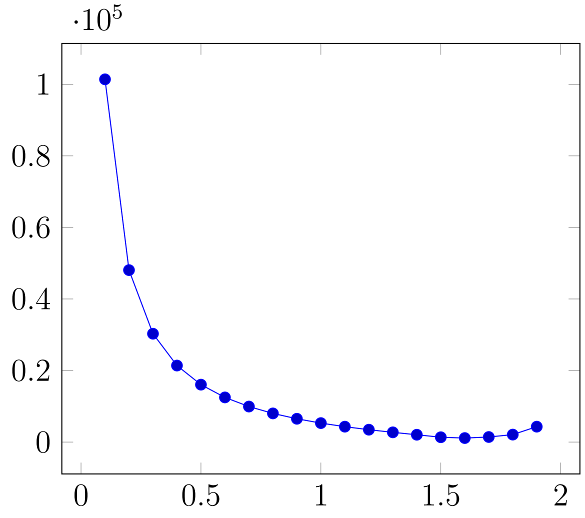
Тестирование программы

Изучение скорости сходимости от параметра ω происходит с шагом 0.1 при $\omega \in [0.1, 1.9]$. Найденный вектор x, в каждом тесте был сравнён с решением, полученным с помощью функции numpy.linalg.solve.

Приложение 1. Вариант 9.

Первая система

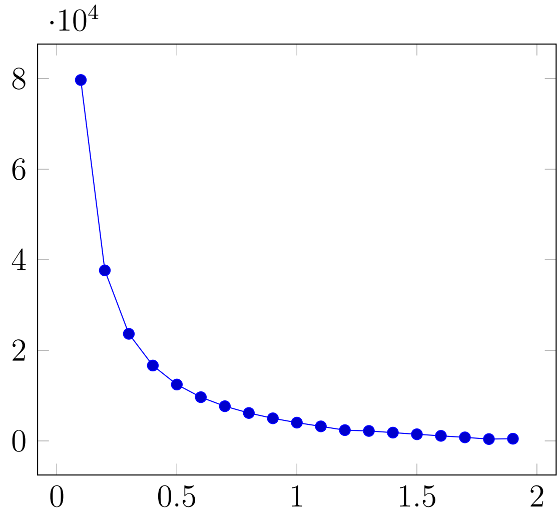
$$A = \begin{pmatrix} 2 & -5 & 3 & 1 \\ 3 & -7 & 3 & -1 \\ 5 & -9 & 6 & 2 \\ 5 & -1 & 7 & 8 \end{pmatrix}, f = \begin{pmatrix} 5 \\ -1 \\ 7 \\ 8 \end{pmatrix}$$



Наилучшая сходимость про $\omega = 1.6$: число итераций $= 1120$.
Решение при наилучшей сходимости:
$x = (0.0000000015, -2.9999999989, -5.3333333326, 5.9999999991)^T$

Вторая система

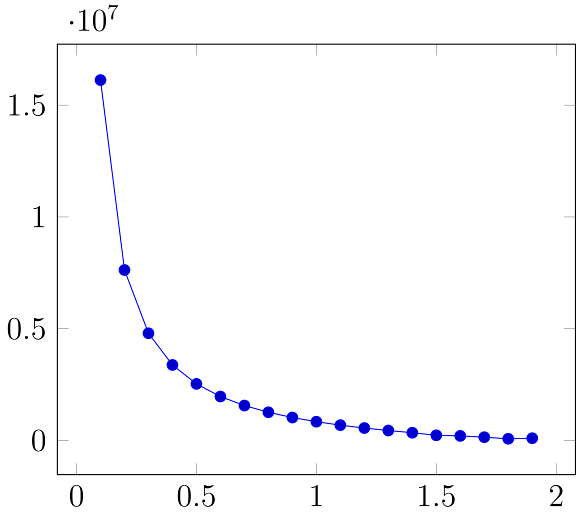
$$A = \begin{pmatrix} 4 & 3 & -9 & 1 \\ 2 & 5 & -8 & -1 \\ 2 & 16 & -14 & 2 \\ 2 & 3 & -5 & -11 \end{pmatrix}, f = \begin{pmatrix} 9 \\ 8 \\ 24 \\ 7 \end{pmatrix}$$



Наилучшая сходимость про $\omega = 1.8$: число итераций = 408.
Решение при наилучшей сходимости:
$x = (3.0000000003, 2.0000000001, 1.0000000002, 0.0000000000)^T$

Третья система

$$A = \begin{pmatrix} 12 & 14 & -15 & 24 \\ 16 & 18 & -22 & 29 \\ 18 & 20 & -21 & 32 \\ 10 & 12 & -16 & 20 \end{pmatrix}, f = \begin{pmatrix} 5 \\ 8 \\ 9 \\ 14 \end{pmatrix}$$



Наилучшая сходимость про $\omega = 1.8$: число итераций = 77675.
Решение при наилучшей сходимости:
$x = (2.2222222226, -1.6666666673, -0.1111111111, 0.0000000001)^T$

Приложение 2. Вариант 2-2

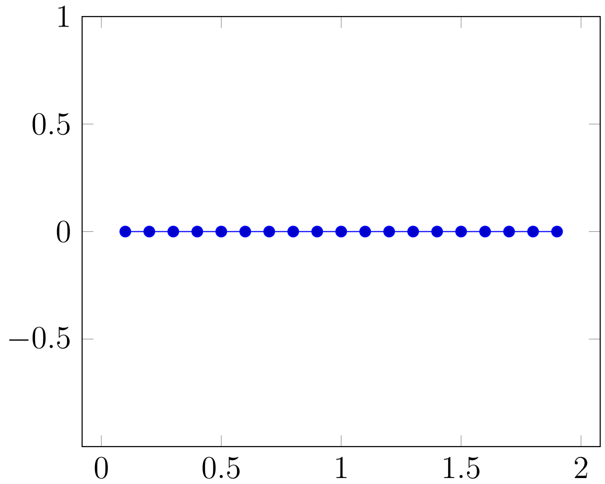
$q_2 = 0.997, n = 40$

$$A_{ij} = \begin{cases} \mathrm{q}_2^{i+j} + 0.1 \cdot (j-i) & , i \neq j \\ (\mathrm{q}_2 - 1)^{i+j} & , i = j \end{cases}$$

$f = \begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_n \end{pmatrix}$, где $f_i = |x - \frac{n}{10}| \cdot i \cdot \sin x$

Приведу результаты теста для $x = 0$ и $x = 1$:

$\mathbf{x = 0}$

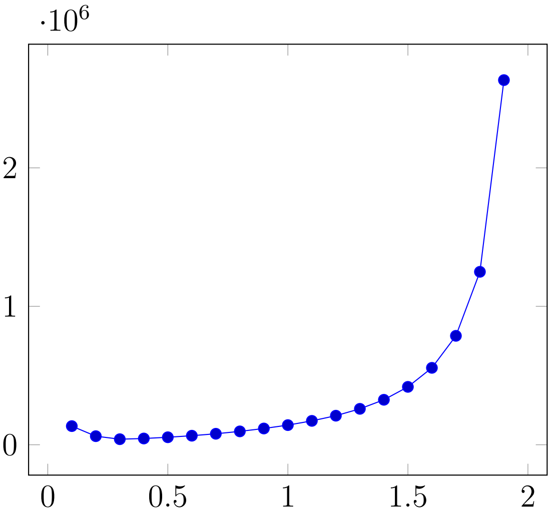


Так как начальный вектор приближения оказался решением, то число итераций для любого ω равно 0. Наилучшая сходимость $(\forall\omega)$: число итераций $=$ 0.

Решение при наилучшей сходимости $(\forall\omega)$:

$x = (0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,$
$0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000)^T$

x = 1



Наилучшая сходимость при $\omega = 0.3$: число итераций = 40437.

Решение при наилучшей сходимости:

$x = (2.3595296368, 2.0942032356, 1.8254101159, 1.5530958910, 1.2772272640, 0.9977706631, 0.7146922388, 0.4279578621, 0.1375331225, -0.1566166746, -0.4545265098, -0.7562316519, -1.0617676600, -1.3711703862, -1.6844759772, -2.0017208770, -2.3229418291, -2.6481758787, -2.9774603750, -3.3108329736, -3.6483316388, -3.9899946460, -4.3358605840, -4.6859683575, -5.0403571895, -5.3990666234, -5.7621365262, -6.1296070901, -6.5015188354, -6.8779126132, -7.2588296074, -7.6443113380, -8.0343996628, -8.4291367808, -8.8285652342, -9.2327279107, -9.6416680464, -10.0554292300, -10.4740554018, 117.5150349307)^T$

Выводы

В ходе работы:

1. Был изучен метод верхней релаксации и метод Зейделя. У метода есть достоинства (легкая реализация на ЭВМ) и недостатки (скорость сходимости метода для каждой системы сильно зависит от выбора итерационного параметра ω).

2. Был разработан критерий останки итерирования для получения решения с произвольной заранее заданной точностью

3. Была изучена зависимость количества итераций (скорости схождения метода) от иттерационного параметра. Скорость схождения метода сильно зависит от параметра ω. Например, в приложении 2 при x = 2 отношение колчиства итераций при наихудшей сходимости ($\omega = 1.9$) к количества итераций при наилучшей сходимости ($\omega = 0.3$) равна $\frac{2633869}{40437} \approx 65.135123773$.