**Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»**

Образовательная программа
«Прикладная математика»

ОТЧЕТ
По лабораторной работе № 2

По предмету

«Численные методы»

По теме

«Решение СЛАУ»

Выполнил
студент группы БПМ211
Кудряшов Максим Дмитриевич

Проверил
Брандышев Петр Евгеньевич

Москва - 2024

Содержание

1 3.1.8 и 3.2

1.1 Постановка задачи

Дана система уравнений $Ax = b$ порядка $n = 6$.

3.1.8. Исследовать зависимость погрешности решения x от погрешностей правой части системы b.

3.2. Исследовать зависимость погрешности решения системы от погрешностей коэффициентов матрицы A.

$$A_{ij} = \frac{1}{\sqrt{c_{ij}^2 + 0.58c_{ij}}}$$

$$b_i = N$$

$$c_{ij} = 0.1Nij$$

$$N = 8$$

1.2 Код на Python

Так как решение обоих задач аналогично я написал его в едином алгоритме.

```
import numpy as np
import matplotlib.pyplot as plt
import os
import pandas as pd

n = 6
N = 8
delta = 0.0005

with open('3.1.8-3.2.txt', "w") as f:
    # заполним матрицу A
    A = np.empty((n, n))
    for i in range(n):
        for j in range(n):
            c = 0.1 * N * (i + 1) * (j + 1)
            A[i][j] = 1 / np.sqrt(c ** 2 + 0.58 * c)
    pd.DataFrame(A).to_csv('A.csv', float_format='%.5f')

    # заполним вектор b
    b = np.full(n, N, dtype=float)
    pd.DataFrame(b).to_csv('b.csv')

    # найдем лист из измененных векторов b
    b_error_list = []
```

```
    for i in range(n):
        b_error = b.copy()
        b_error[i] += delta
        b_error_list.append(b_error)

    # найдем лист из измененных матриц A
    A_error_list = []
    for i in range(n):
        A_error_row = []
        for j in range(n):
            A_error = A.copy()
            A_error[i][j] += delta
            A_error_row.append(A_error)
        A_error_list.append(A_error_row)

    # найдем решение через встроенную функцию
    x = np.linalg.solve(A, b)
    # print(f"{x = }", file=f)
    pd.DataFrame(x).to_csv('x.csv', float_format='%.6f')

    # найдем число обусловленности через встроенную функцию
    A_cond = np.linalg.cond(A, np.inf)
    print(f"{A_cond = }", file=f)

    # найдем вектор d для изменения b
    d_for_b = np.empty(n)
    for i in range(n):
        b_error = b_error_list[i]
        x_error = np.linalg.solve(A, b_error)
        d_for_b[i] = np.linalg.norm(x - x_error, ord=np.inf) / np.linalg.norm(x, ord=np.inf)
    # print(f"{d_for_b = }", file=f)
    pd.DataFrame(d_for_b).to_csv('d_for_b.csv', float_format='%.6f')

    # найдем вектор d для изменения A
    d_for_A = np.empty(A.shape)
    for i in range(n):
        for j in range(n):
            A_error = A_error_list[i][j]
            x_error = np.linalg.solve(A_error, b)
            d_for_A[i][j] = np.linalg.norm(x - x_error, ord=np.inf) / np.linalg.norm(x, ord=np.inf)
    # print(f"{d_for_A = }", file=f)
    pd.DataFrame(d_for_A).to_csv('d_for_A.csv', float_format='%.6f')

    # найдем что оказывает наибольшее влияние на погрешность для b
    d_for_b_max = d_for_b.max()
    d_for_b_max_index = list(d_for_b).index(d_for_b_max)
    print(f'Максимальный элемент d для b: {d_for_b_max} с индексом = {d_for_b_max_index}', file=f)


    # найдем что оказывает наибольшее влияние на погрешность для A
```

```python
    d_for_A_max = d_for_A.max()
    # d_for_b_max_index = d_for_A.argmax()
    for i in range(n):
        for j in range(n):
            if d_for_A[i][j] == d_for_A_max:
                d_for_b_max_index = [i, j]
    print(f'Максимальный элемент d для A: {d_for_A_max} с индексом = {d_for_b_max_index}', file=f)

    # построим гистограмму для b
    plt.bar(range(n), d_for_b)
    plt.savefig('3.1.8.png', dpi=300)
    plt.show()

    # построим гистограмму для A
    data_2d = d_for_A
    data_array = np.array(data_2d)
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    x_data, y_data = np.meshgrid(np.arange(data_array.shape[1]), np.arange(data_array.shape[0]))
    x_data = x_data.flatten()
    y_data = y_data.flatten()
    z_data = data_array.flatten()
    ax.bar3d(x_data, y_data, np.zeros(len(z_data)),
             0.5, 0.5, z_data)
    fig.tight_layout()
    plt.savefig('3.2.png', dpi=300)
    fig.show()

    # сделаем теоретическую оценку для b
    delta_rel_b = np.abs(delta) / np.linalg.norm(b, ord=np.inf)  # тоже что и np.linalg.norm(b - b_error, ord=np.i
    delta_rel_x_error_for_b = A_cond * delta_rel_b
    print(f'Теоретическая оценка для для b: {delta_rel_x_error_for_b}', file=f)

    # сделаем теоретическую оценку для A
    delta_rel_A = np.abs(delta) / np.linalg.norm(A, ord=np.inf)
    delta_rel_x_error_for_A = A_cond * delta_rel_A
    print(f'Теоретическая оценка для для A: {delta_rel_x_error_for_A}', file=f)

```

1.3 Результаты

Матрица A:

	0	1	2	3	4	5
0	0.95173	0.53544	0.37393	0.28753	0.23363	0.19678
1	0.53544	0.28753	0.19678	0.14962	0.12070	0.10116
2	0.37393	0.19678	0.13361	0.10116	0.08139	0.06809
3	0.28753	0.14962	0.10116	0.07641	0.06140	0.05131
4	0.23363	0.12070	0.08139	0.06140	0.04929	0.04117
5	0.19678	0.10116	0.06809	0.05131	0.04117	0.03438

Вектор b:

	0
0	8.0
1	8.0
2	8.0
3	8.0
4	8.0
5	8.0

Вектор x:

	0
0	2286259.627517
1	-197930832.742647
2	2395876438.190361
3	-8968288582.023243
4	12723307598.022757
5	-6027213311.850702

Вектор d для b:

	0
0	0.000002
1	0.000161
2	0.001993
3	0.007539
4	0.010765
5	0.005121

Матрица d для A:

	0	1	2	3	4	5
0	2.259103	1.165517	1.075877	1.028340	1.000045	0.980934
1	1.234877	1.090988	1.040624	1.015275	1.000000	0.989791
2	1.163512	1.062178	1.027787	1.010449	1.000000	0.993014
3	1.123843	1.047243	1.021115	1.007941	1.000000	0.994691
4	1.099919	1.038092	1.017028	1.006404	1.000000	0.995718
5	1.083539	1.031914	1.014266	1.005366	1.000000	0.996412

Полученные данные:

```
A_cond = 1678868976234.5142
Максимальный элемент d для b: 0.010764672729249327 с индексом = 4
Максимальный элемент d для A: 2.2591034667228396 с индексом = [0, 0]
Теоретическая оценка для для b: 104929311.01465714
Теоретическая оценка для для A: 325482471.3620952
```

Наибольший вклад в 3.1.8 вносит $m = 4$

Наибольший вклад в 3.2 вносит элемент с индексами $(0, 0)$

Видно, что теоретические оценки выполняются в обоих случиях.

Теоретическая оценка для 3.1.8: $cond(A) \cdot \delta(b^m)$

Теоретическая оценка для 3.2: $cond(A) \cdot \delta(A*)$

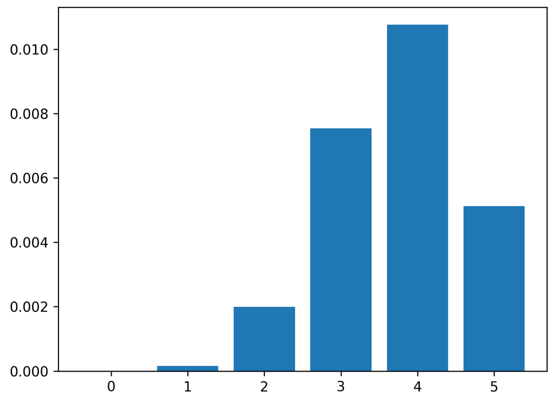

Рис. 1: 3.1.8

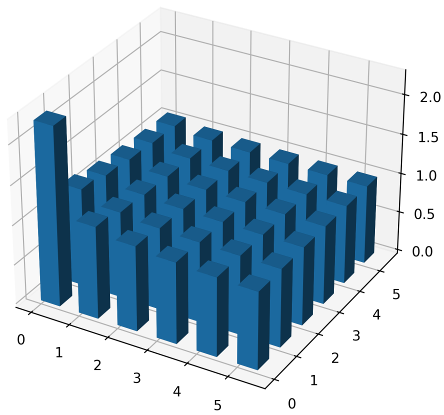

Рис. 2: 3.2

1.3.1 Вывод

Видно, что практически полученная погрешность сильно меньше теоретической оценки.

2 3.8.2

2.1 Постановка задачи

Дана система уравнений $Az(x) = b(x)$ порядка n. Построить график функции $y(x) = \sum_{i=1}^{n} z_i(x)$ на отрезке $[a, b]$. Для решения системы написать и использовать метод Гаусса со схемой полного перебора.

2.2 Теория

2.2.1 Метод Гаусса

Вычисления с помощью метода Гаусса состоят из двух основных этапов, называемых прямым ходом и обратныл ходом. Прямой ход метода Гаусса заключается в последовательном исключении неизвестных из системы для преобразования ее к эквивалентной системе с верхней треугольной матрицей. Вычисления значений неизвестных производят на этапе обратного хода.

Прямой ход. Состоит из $m-1$ шагов исключения. k-й шаг. В предположении, что ведуший элелент k-го шага $a_{kk}^{(k-1)}$ отличен от нуля, вычислим множители k-го шала

$$\mu_{ik} = a_{ik}^{(k-1)}/a_{kk}^{(k-1)} \quad (i = k+1, \ldots, m)$$

и вычтем последовательно из $(k+1)$-го, .,., m-го уравнений полученной на предыдущем шаге системы k-е уравнение, умноженное соответственно на $\mu_{k+1,k}, \mu_{k+2,k}, \ldots, \mu_{mk}$. После $(m-1)$-го шага исключения получим систему уравнений

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1m}x_m &= b_1 \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2m}^{(1)}x_m &= b_2^{(1)} \\ a_{33}^{(2)}x_3 + \ldots + a_{3m}^{(2)}x_m &= b_3^{(2)} \\ \ddots \ldots \cdots\cdots & \\ a_{mm}^{(m-1)}x_m &= b_m^{(m-1)} \end{aligned}$$

матрица $\boldsymbol{A}^{(m-1)}$ которой является верхней треугольной. На этом вычисления прямого хода заканчиваются.

Обратный ход. Из последнего уравнения системы находим x_m. Подставляя найденное значение x_m в предпоследнее уравнение, получим x_{m-1}. Осуществляя обратную подстановку, далее последовательно находим $x_{m-2}, x_{m-3}, \ldots, x_1$. Вычисления неизвестных здесь проводятся по формулам

$$\begin{aligned} & x_m = b_m^{(m-1)}/a_{mm}^{(m-1)}, \\ & x_k = \left(b_k^{(k-1)} - ak_{k,k+1}^{(k-1)}x_{k+1} - \ldots - a_{km}^{(k-1)}x_m\right)/a_{kk}^{(k-1)}, (k = m-1, \ldots, 1). \end{aligned}$$

2.2.2 Схема полного выбора

На 1 -м шаге метода среди элементов a_{ij} определяют максимальный по модулю элемент $a_{i_1 j_1}$. Первое уравнение системы и уравнение с номером i_1 меняют местами. Далее стандартным образом производят исключение неизвестного x_{j_1} из всех уравнений, кроме первого.

На k-м шаге метода среди коэффициентов $a_{ij}^{(k-1)}$ при неизвестных в уравнениях системы с номерами $i = k, \ldots, m$ выбирают максимальный по модулю коэффициент $a_{i}^{(k-1)}$. Затем k-е уравнение и уравнение, содержащее найденный коэффициент, меняют местами и исключают неизвестное x_{j_k} из уравнений с номерами $i = k+1, \ldots, m$. На этапе обратного хода неизвестные вычисляют в следующем порядке: $x_{j_m}, x_{j_{m-1}}, \ldots, x_{j_1}$

2.3 Код на Python

Я реализовал два метода Гаусса. Сначала метод Гаусса для численно заданных матрицы и вектора, а потом для численно заданой матрицы, но при этом символьго заданного вектора b.

Пояснение к тому, как работает метод Гаусса находятся в коде в комментариях.

2.3.1 Метод Гаусса для чисел

```python
import numpy as np


def gauss_elimination_number(A: np.ndarray, b: np.array) -> np.array:
    n = A.shape[0]
    x = np.empty(n, dtype=float)

    # прямой ход
    for i in range(n):

        # ищем максимальный коэффициент среди уравнений с неизвестными x и берем его индекс
        A_slice = A[i:][:]
        max_element_row_index = np.unravel_index(np.argmax(A_slice), A_slice.shape)[0] + i

        # меняем местами i-ое и max_element_row_index-ое уравнения
        if max_element_row_index != i:
            A[[i, max_element_row_index]] = A[[max_element_row_index, i]]
            b[[i, max_element_row_index]] = b[[max_element_row_index, i]]

        # убираем коэффициенты при x
        for j in range(i + 1, n):
            mu = A[j][i] / A[i][i]
            A[j] -= mu * A[i]
            b[j] -= mu * b[i]

    # обратный ход
    for m in range(n - 1, -1, -1):
        numerator = b[m]  # числитель
        for l in range(m + 1, n):
            numerator -= A[m][l] * x[l]
        denominator = A[m][m]  # знаменатель
        x[m] = numerator / denominator

    return x


A = np.array([[2.0, 1.0, -1.0],
              [-3.0, -1.0, 2.0],
```

```
              [-2.0, 1.0, 2.0]])
b = np.array([8.0, -11.0, -3.0])

my_x = gauss_elimination_number(A, b)
np_x = np.linalg.solve(A, b)

print("Решение системы уравнений Ax=b методом Гаусса:", my_x)
print("Решение системы уравнений Ax=b методом Гаусса:", np_x)
print(f"Решения через np.solve и мое решения {'совпадают' if np.allclose(my_x, np_x) else 'не совпадают'}")
```

2.3.2 Метод Гаусса для символов

```
import numpy as np
import sympy as sp
import os
import pandas as pd

output_file = "3.8.2"


def gauss_elimination_sympy(A: np.array, b: sp.Matrix):
    n = A.shape[0]
    x = sp.Matrix([0] * n)

    # прямой ход
    for i in range(n):

        # ищем максимальный коэффициент среди уравнений с неизвестными x и берем его индекс
        A_slice = A[i:][:]
        max_element_row_index = np.unravel_index(np.argmax(A_slice), A_slice.shape)[0] + i

        # меняем местами i-ое и max_element_row_index-ое уравнения
        if max_element_row_index != i:
            A[[i, max_element_row_index]] = A[[max_element_row_index, i]]  # синтаксис numpy
            b.elementary_row_op('n<->m', row1=i, row2=max_element_row_index)  # синтаксис sympy

        # избавляемся от коэффициентов при x
        for j in range(i + 1, n):
            mu = A[j][i] / A[i][i]
            A[j] -= mu * A[i]
            b[j] -= mu * b[i]

    # обратный ход
    for m in range(n - 1, -1, -1):
        numerator = b[m]  # числитель
        for l in range(m + 1, n):
            numerator -= A[m][l] * x[l]
        denominator = A[m][m]  # знаменатель
```

```
        x[m] = numerator / denominator

    return sp.sympify(x)


# os.remove(output_file + '.txt')
with open(output_file + '.txt', "w") as f:
    # зададим константы
    n = 40
    M = 2
    q = 1.001 - 2 * M / 1000
    X = sp.Symbol('x')

    # заполним матрицу A и вектор b
    A = np.ndarray((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)
    b = sp.Matrix([sp.Abs(X - n / 10) * i * sp.sin(X) for i in range(1, n + 1)])
    pd.DataFrame(A).to_csv('A.csv', float_format='%.3f')

    # решим систему
    my_z = gauss_elimination_sympy(A, b)
    sp_z = sp.simplify(sp.Matrix(A.tolist()).solve(b))
    print("Решение системы посчитанное через мою функцию:\n", my_z, file=f)
    print("Решение системы посчитанное через sympy:\n", sp_z, file=f)
    print(file=f)

    # вычислим y
    my_y = 0
    sp_y = 0
    for i in range(len(my_z)):
        my_y += my_z[i]
        sp_y += sp_z[i]
    print("Значение 'y' посчитанное через мою функцию:", my_y, file=f)
    print("Значение 'y' посчитанное через sympy:", sp_y, file=f)

    # построим график
    graph = sp.plot(my_y, sp_y, (X, -5, 5), line_color='red', dpi=300)
    backend = graph.backend(graph)
    backend.process_series()
    backend.fig.savefig(output_file + '.png', dpi=300)
    backend.show()
```

2.4 Результаты

```
Решение системы посчитанное через мою функцию:
 Matrix([[-158335.979769201*sin(x)*Abs(x - 4.0)], [-166233.537343039*sin(x)*Abs(x - 4.0)], [-254109.815812915*sin(
Решение системы посчитанное через sympy:
 Matrix([[-158335.979459671*sin(x)*Abs(x - 4.0)], [-166233.537340764*sin(x)*Abs(x - 4.0)], [-254109.815813078*sin(
```

Видно, что результаты, полученные через написанную мной функцию и высичленные через готовую функцию из sympy совпадают.