

**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	2
1.1	Постановка задачи	2
1.2	Код на Python	2
1.3	Результаты	5
1.3.1	Вывод	7
2	3.8.2	8
2.1	Постановка задачи	8
2.2	Код на Python	8
2.2.1	Метод Гаусса для чисел	8
2.2.2	Метод Гаусса для символов	9
2.3	Результаты	11

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 обеих задач аналогично я написал его в едином алгоритме.

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  import os
4  import pandas as pd
5
6  n = 6
7  N = 8
8  delta = 0.0005
9
10 with open('3.1.8-3.2.txt', "w") as f:
11     # заполним матрицу A
12     A = np.empty((n, n))
13     for i in range(n):
14         for j in range(n):
15             c = 0.1 * N * (i + 1) * (j + 1)
16             A[i][j] = 1 / np.sqrt(c ** 2 + 0.58 * c)
17     pd.DataFrame(A).to_csv('A.csv', float_format='%.5f')
18
19     # заполним вектор b
20     b = np.full(n, N, dtype=float)
21     pd.DataFrame(b).to_csv('b.csv')
22
23     # найдем лист из измененных векторов b
24     b_error_list = []

```

```

25     for i in range(n):
26         b_error = b.copy()
27         b_error[i] += delta
28         b_error_list.append(b_error)
29
30     # найдем лист из измененных матриц A
31     A_error_list = []
32     for i in range(n):
33         A_error_row = []
34         for j in range(n):
35             A_error = A.copy()
36             A_error[i][j] += delta
37             A_error_row.append(A_error)
38         A_error_list.append(A_error_row)
39
40     # найдем решение через встроенную функцию
41     x = np.linalg.solve(A, b)
42     # print(f"{'x = }", file=f)
43     pd.DataFrame(x).to_csv('x.csv', float_format='%.6f')
44
45     # найдем число обусловленности через встроенную функцию
46     A_cond = np.linalg.cond(A, np.inf)
47     print(f"A_cond = ", file=f)
48
49     # найдем вектор d для изменения b
50     d_for_b = np.empty(n)
51     for i in range(n):
52         b_error = b_error_list[i]
53         x_error = np.linalg.solve(A, b_error)
54         d_for_b[i] = np.linalg.norm(x - x_error, ord=np.inf) / np.linalg.norm(x, ord=np.inf)
55     # print(f"{'d_for_b = }", file=f)
56     pd.DataFrame(d_for_b).to_csv('d_for_b.csv', float_format='%.6f')
57
58     # найдем вектор d для изменения A
59     d_for_A = np.empty(A.shape)
60     for i in range(n):
61         for j in range(n):
62             A_error = A_error_list[i][j]
63             x_error = np.linalg.solve(A_error, b)
64             d_for_A[i][j] = np.linalg.norm(x - x_error, ord=np.inf) / np.linalg.norm(x, ord=np.inf)
65     # print(f"{'d_for_A = }", file=f)
66     pd.DataFrame(d_for_A).to_csv('d_for_A.csv', float_format='%.6f')
67
68     # найдем что оказывает наибольшее влияние на погрешность для b
69     d_for_b_max = d_for_b.max()
70     d_for_b_max_index = list(d_for_b).index(d_for_b_max)
71     print(f'Максимальный элемент d для b: {d_for_b_max} с индексом = {d_for_b_max_index}', file=f)
72
73
74     # найдем что оказывает наибольшее влияние на погрешность для A

```

```

75 d_for_A_max = d_for_A.max()
76 # d_for_b_max_index = d_for_A.argmax()
77 for i in range(n):
78     for j in range(n):
79         if d_for_A[i][j] == d_for_A_max:
80             d_for_b_max_index = [i, j]
81 print(f'Максимальный элемент d для A: {d_for_A_max} с индексом = {d_for_b_max_index}', file=f)
82
83 # построим гистограмму для b
84 plt.bar(range(n), d_for_b)
85 plt.savefig('3.1.8.png', dpi=300)
86 plt.show()
87
88 # построим гистограмму для A
89 data_2d = d_for_A
90 data_array = np.array(data_2d)
91 fig = plt.figure()
92 ax = fig.add_subplot(projection='3d')
93 x_data, y_data = np.meshgrid(np.arange(data_array.shape[1]), np.arange(data_array.shape[0]))
94 x_data = x_data.flatten()
95 y_data = y_data.flatten()
96 z_data = data_array.flatten()
97 ax.bar3d(x_data, y_data, np.zeros(len(z_data)),
98          0.5, 0.5, z_data)
99 fig.tight_layout()
100 plt.savefig('3.2.png', dpi=300)
101 fig.show()
102
103 # сравним погрешности для b
104 # print(f"Сравнение погрешностей для b:", file=f)
105 delta_rel_x_error_list = np.empty(n, dtype=float)
106 for i in range(n):
107     b_error = b_error_list[i]
108     # delta_rel_b_old = np.linalg.norm(b - b_error, ord=np.inf) / np.linalg.norm(b, ord=np.inf)
109     delta_rel_b = np.abs(delta) / np.linalg.norm(b, ord=np.inf)
110     delta_rel_x_error = A_cond * delta_rel_b
111     delta_rel_x_error_list[i] = delta_rel_x_error
112     # print(f"практически: {d_for_b[i]}, \tтеоретически: {delta_rel_x_error}", file=f)
113
114 print(f'Теоретическая оценка для для b: {A_cond * delta_rel_b}', file=f)
115 # pd.DataFrame(d_for_A).to_csv('d_for_A.csv', float_format='%.6f')
116 # df_for_b = pd.DataFrame({
117 #     'Индекс': list(range(n)),
118 #     'Практические': d_for_b,
119 #     "Теоретические": delta_rel_x_error_list
120 # })
121 # df_for_b.to_latex("3.1.8.tex", index=False)
122
123 # сравним погрешности для b
124 # print(f"Сравнение погрешностей для A:", file=f)

```

```

125 delta_rel_x_error_list = np.empty(A.shape, dtype=float)
126 for i in range(n):
127     for j in range(n):
128         A_error = A_error_list[i][j]
129         delta_rel_A = np.abs(delta) / np.linalg.norm(A, ord=np.inf)
130         delta_rel_x_error = A_cond * delta_rel_A
131         delta_rel_x_error_list[i][j] = delta_rel_x_error
132         # print(f"[{i}][{j}] практически: {d_for_A[i][j]}, \t теоретически: {delta_rel_x_error}", file=f)
133
134 # ids = np.empty((n, n), dtype=tuple)
135 # for i in range(n):
136 #     for j in range(n):
137 #         ids[i][j] = (i, j)
138 # df_for_A = pd.DataFrame({
139 #     'i': ids.reshape(-1),
140 #     'Практические': d_for_A.reshape(-1),
141 #     "Теоретические": delta_rel_x_error_list.reshape(-1)
142 # })
143 # df_for_A.to_latex("3.2.tex", index=False)
144 print(f'Теоретическая оценка для для A: {A_cond * delta_rel_A}', file=f)
145

```

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:

```

1 A_cond = 1678868976234.5142
2 Максимальный элемент d для b: 0.010764672729249327 с индексом = 4
3 Максимальный элемент d для A: 2.2591034667228396 с индексом = [0, 0]
4 Теоретическая оценка для для b: 104929311.01465714
5 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 случаях.

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

Теоретическая оценка для 3.2: $\text{cond}(A) \cdot \delta(A^*)$

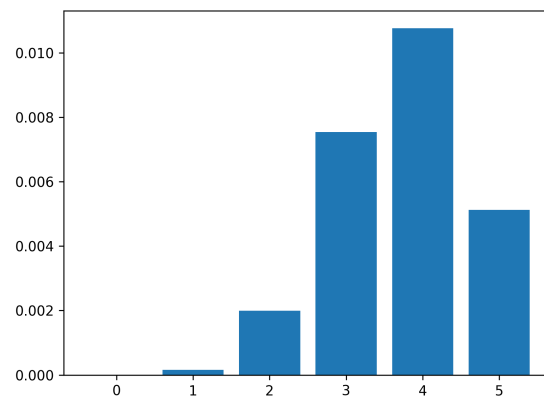


Рис. 1: 3.1.8

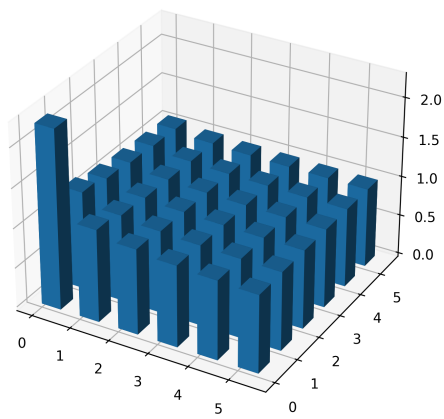


Рис. 2: 3.2

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 Код на Python

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

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

2.2.1 Метод Гаусса для чисел

```

1  import numpy as np
2
3
4  def gauss_elimination_number(A: np.ndarray, b: np.array) -> np.array:
5      n = A.shape[0]
6      x = np.empty(n, dtype=float)
7
8      # прямой ход
9      for i in range(n):
10
11         # ищем максимальный коэффициент среди уравнений с неизвестными x и берем его индекс
12         A_slice = A[i:][:]
13         max_element_row_index = np.unravel_index(np.argmax(A_slice), A_slice.shape)[0] + i
14
15         # меняем местами i-ое и max_element_row_index-ое уравнения
16         if max_element_row_index != i:
17             A[[i, max_element_row_index]] = A[[max_element_row_index, i]]
18             b[[i, max_element_row_index]] = b[[max_element_row_index, i]]
19
20         # убираем коэффициенты при x
21         for j in range(i + 1, n):
22             mu = A[j][i] / A[i][i]
23             A[j] -= mu * A[i]
24             b[j] -= mu * b[i]
25
26         # обратный ход
27         for m in range(n - 1, -1, -1):
28             numerator = b[m] # числитель
29             for l in range(m + 1, n):
30                 numerator -= A[m][l] * x[l]
31             denominator = A[m][m] # знаменатель
32             x[m] = numerator / denominator
33
34     return x

```

```

35
36
37 A = np.array([[2.0, 1.0, -1.0],
38               [-3.0, -1.0, 2.0],
39               [-2.0, 1.0, 2.0]])
40 b = np.array([8.0, -11.0, -3.0])
41
42 my_x = gauss_elimination_number(A, b)
43 np_x = np.linalg.solve(A, b)
44
45 print("Решение системы уравнений Ax=b методом Гаусса:", my_x)
46 print("Решение системы уравнений Ax=b методом Гаусса:", np_x)
47 print(f"Решения через np.solve и мое решения {'совпадают' if np.allclose(my_x, np_x) else 'не совпадают'}")

```

2.2.2 Метод Гаусса для символов

```

1  import numpy as np
2  import sympy as sp
3  import os
4
5  output_file = "3.8.2"
6
7
8  def gauss_elimination_sympy(A: np.array, b: sp.Matrix):
9      n = A.shape[0]
10     x = sp.Matrix([0] * n)
11
12     # прямой ход
13     for i in range(n):
14
15         # ищем максимальный коэффициент среди уравнений с неизвестными x и берем его индекс
16         A_slice = A[i:][:]
17         max_element_row_index = np.unravel_index(np.argmax(A_slice), A_slice.shape)[0] + i
18
19         # меняем местами i-ое и max_element_row_index-ое уравнения
20         if max_element_row_index != i:
21             A[[i, max_element_row_index]] = A[[max_element_row_index, i]] # синтаксис numpy
22             b.elementary_row_op('n<->m', row1=i, row2=max_element_row_index) # синтаксис sympy
23
24         # избавляемся от коэффициентов при x
25         for j in range(i + 1, n):
26             mu = A[j][i] / A[i][i]
27             A[j] -= mu * A[i]
28             b[j] -= mu * b[i]
29
30     # обратный ход
31     for m in range(n - 1, -1, -1):
32         numerator = b[m] # числитель

```

```

33     for l in range(m + 1, n):
34         numerator -= A[m][l] * x[l]
35         denominator = A[m][m] # знаменатель
36         x[m] = numerator / denominator
37
38     return sp.simplify(x)
39
40
41 # os.remove(output_file + '.txt')
42 with open(output_file + '.txt', "w") as f:
43     # зададим константы
44     n = 40
45     M = 2
46     q = 1.001 - 2 * M / 1000
47     X = sp.Symbol('x')
48
49     # заполним матрицу A и вектор b
50     A = np.ndarray((n, n))
51     for i in range(1, n + 1):
52         for j in range(1, n + 1):
53             if i != j:
54                 A[i - 1][j - 1] = q ** (i + j) + 0.1 * (j - i)
55             else:
56                 A[i - 1][j - 1] = (q - 1) ** (i + j)
57     b = sp.Matrix([sp.Abs(X - n / 10) * i * sp.sin(X) for i in range(1, n + 1)])
58
59     # решим систему
60     my_z = gauss_elimination_sympy(A, b)
61     sp_z = sp.simplify(sp.Matrix(A.tolist()).solve(b))
62     print("Решение системы посчитанное через мою функцию:\n", my_z, file=f)
63     print("Решение системы посчитанное через sympy:\n", sp_z, file=f)
64     print(file=f)
65
66     # вычислим y
67     my_y = 0
68     sp_y = 0
69     for i in range(len(my_z)):
70         my_y += my_z[i]
71         sp_y += sp_z[i]
72     print("Значение 'y' посчитанное через мою функцию:", my_y, file=f)
73     print("Значение 'y' посчитанное через sympy:", sp_y, file=f)
74
75     # построим график
76     graph = sp.plot(my_y, sp_y, (X, -5, 5), line_color='red', dpi=300)
77     backend = graph.backend(graph)
78     backend.process_series()
79     backend.fig.savefig(output_file + '.png', dpi=300)
80     backend.show()

```

2.3 Результаты

```

1  Решение системы посчитанное через мою функцию:
2  Matrix([[-158335.979769201*sin(x)*Abs(x - 4.0)], [-166233.537343039*sin(x)*Abs(x - 4.0)], [-254109.815812915*sin(x)
3  Решение системы посчитанное через sympy:
4  Matrix([[-158335.979459671*sin(x)*Abs(x - 4.0)], [-166233.537340764*sin(x)*Abs(x - 4.0)], [-254109.815813078*sin(x)
5
6  Значение 'y' посчитанное через мою функцию: -77224.4853234382*sin(x)*Abs(x - 4.0)
7  Значение 'y' посчитанное через sympy: -77224.4850137634*sin(x)*Abs(x - 4.0)

```

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

