

МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное  
образовательное учреждение высшего образования  
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической физики и вычислительной математики

ОТЧЁТ  
ПО ПРАКТИЧЕСКОЙ ПОДГОТОВКЕ  
по дисциплине «Методы вычислений»

студента 3 курса 351 группы

направления 09.03.04 — Программная инженерия

факультета компьютерных наук и информационных технологий

Карасева Вадима Дмитриевича

Проверено:

\_\_\_\_\_

\_\_\_\_\_

## 1 Построение интерполяционного многочлена в общем виде

Необходимо найти интерполяционный многочлен в общем виде.

$x$	0	1	2	3
$f(x)$	2	3	10	29

### Код

```
1 import numpy
2 # входные данные
3 x_data = [0, 1, 2, 3]
4 f_data = [2, 3, 10, 29]
5 x_np_data = numpy.array(x_data)
6 f_np_data = numpy.array(f_data)
7 print("Исходные данные:")
8 for i in range(0, 4):
9     print('X' + str(i) + ': ' + str(x_data[i]) + '\tF' + str(i) + ': ' + str(f_data[i]))
10 # строим матрицу
11 matrix = numpy.vander(x_np_data, len(x_data), increasing=True)
12 matrix_flipped = numpy.flip(matrix, axis=1)
13 print("\nСистема построена:")
14 for i in range(0, 4):
15     s = ""
16     for j in range(0, 4):
17         s += str(matrix_flipped[i][j]) + ' * a' + str(4 - j - 1) + ' + '
18     print(s[:-2] + ' = ' + str(f_data[i]))
19 print()
20 # решение системы уравнений для нахождения коэффициентов многочлена
21 coefficients = numpy.linalg.solve(matrix, f_np_data)
22 print("\nКоэффициенты найдены:")
23 for i in range(0, 4):
24     print('a' + str(i) + ': ' + str(coefficients[i]))
25 print("\nP3 (x): ")
26 for i in range(0, 7):
27     print('x = ' + str(i / 2) + ': ' + str(
28         coefficients[3] * (i / 2) * (i / 2) * (i / 2)
29         + coefficients[2] * (i / 2) * (i / 2)
30         + coefficients[1] * (i / 2)
31         + coefficients[0]
32     ))
```

### Результат

```
1 Исходные данные:
2 X0: 0   F0: 2
3 X1: 1   F1: 3
4 X2: 2   F2: 10
5 X3: 3   F3: 29
6 Система построена:
7 0 * a3 + 0 * a2 + 0 * a1 + 1 * a0 = 2
8 1 * a3 + 1 * a2 + 1 * a1 + 1 * a0 = 3
9 8 * a3 + 4 * a2 + 2 * a1 + 1 * a0 = 10
10 27 * a3 + 9 * a2 + 3 * a1 + 1 * a0 = 29
11 Коэффициенты найдены:
12 a0: 2.0
13 a1: 0.0
14 a2: -0.0
15 a3: 1.0
```

```

1 P3 (x):
2 x = 0.0: 2.0
3 x = 0.5: 2.125
4 x = 1.0: 3.0
5 x = 1.5: 5.375
6 x = 2.0: 10.0
7 x = 2.5: 17.625
8 x = 3.0: 29.0

```

## 2 Интерполяционный многочлен в форме Лагранжа

По данным интерполяции из предыдущего задания построить интерполяционный многочлен в форме Лагранжа.

$x$	0	1	2	3
$f(x)$	2	3	10	29

### Код

```

1 import numpy
2 def lagrange_fundamental(k, x_nodes, z):
3     Lk = 1.0
4     for i, xi in enumerate(x_nodes):
5         if i != k:
6             Lk *= (z - xi) / (x_nodes[k] - xi)
7     return Lk
8
9 def lagrange_interpolation(x_nodes, y_nodes, z):
10    P = 0.0
11    for k in range(len(x_nodes)):
12        P += y_nodes[k] * lagrange_fundamental(k, x_nodes, z)
13    return P
14 x_data = [0, 1, 2, 3]
15 f_data = [2, 3, 10, 29]
16 x_np_data = numpy.array(x_data)
17 f_np_data = numpy.array(f_data)
18 print("Входные данные:")
19 for i in range(0, 4):
20     print('X' + str(i) + ': ' + str(x_data[i]) + '\tF' + str(i) + ': ' + str(f_data[i]))
21 print("\nL3 (x): ")
22 for i in range(0, 7):
23     print('x = ' + str(i / 2) + ': ' + str(
24         lagrange_interpolation(x_data, f_data, i / 2)
25     ))

```

### Результат

```

1 Входные данные:
2 X0: 0    F0: 2
3 X1: 1    F1: 3
4 X2: 2    F2: 10
5 X3: 3    F3: 29
6 L3 (x):
7 x = 0.0: 2.0
8 x = 0.5: 2.125
9 x = 1.0: 3.0
10 x = 1.5: 5.375
11 x = 2.0: 10.0

```

```
1 x = 2.5: 17.625
2 x = 3.0: 29.0
```

### 3 Интерполяционный многочлен в форме Ньютона

По данным интерполяции из предыдущего задания построить интерполяционный многочлен в форме Ньютона.

$x$	0	1	2	3
$f(x)$	2	3	10	29

Код

```
1 import numpy as np
2 import pandas as pd
3 # Входные данные
4 x_data = np.array([0, 1, 2, 3])
5 f_data = np.array([2, 3, 10, 29])
6 print("Исходные данные:")
7 print(pd.DataFrame({'X': x_data, 'F': f_data}))
8 # Построение таблицы разделённых разностей
9 def divided_difference_table(x, y):
10     n = len(x)
11     table = np.zeros((n, n))
12     table[:,0] = y
13     for j in range(1, n):
14         for i in range(n - j):
15             table[i,j] = (table[i+1,j-1] - table[i,j-1]) / (x[i+j] - x[i])
16     return table
17 dd_table = divided_difference_table(x_data, f_data)
18 # Вывод таблицы разделённых разностей
19 dd_df = pd.DataFrame(dd_table, columns=[f"f[x0..x{j}]" for j in range(len(x_data))])
20 print("\nТаблица разделённых разностей (матрица Ньютона):")
21 print(dd_df)
22 # Коэффициенты Ньютона (верхняя строка таблицы)
23 newton_coeffs = dd_table[0,:]
24 print("\nКоэффициенты многочлена Ньютона:")
25 print(newton_coeffs)
26
27 # Функция для вычисления значения многочлена Ньютона
28 def newton_polynomial(x, coeffs, x_data):
29     n = len(coeffs) - 1
30     p = coeffs[n]
31     for k in range(1, n + 1):
32         p = coeffs[n-k] + (x - x_data[n-k]) * p
33     return p
34 # Вычисление значений на промежуточных точках
35 x_half = [(x_data[i] + x_data[i+1])/2 for i in range(len(x_data)-1)]
36 f_half_newton = [newton_polynomial(x, newton_coeffs, x_data) for x in x_half]
37 # Объединяем узлы и промежуточные точки
38 x_combined = np.concatenate([x_data, x_half])
39 y_newton = np.concatenate([f_data, f_half_newton])
40 # Сортируем по X
41 sort_idx = np.argsort(x_combined)
42 x_combined = x_combined[sort_idx]
43 y_newton = y_newton[sort_idx]
```

```

1 # Создаем итоговую таблицу
2 newton_df = pd.DataFrame({
3     'X': x_combined,
4     'F (Ньютон)': y_newton
5 })
6 print("\nЗначения многочлена Ньютона:")
7 print(newton_df.to_string(index=False))

```

## Результат

```

1 Исходные данные:
2   X   F
3  0   2
4  1   3
5  2  10
6  3  29
7 Таблица разделённых разностей (матрица Ньютона):
8   f[x0..x0]  f[x0..x1]  f[x0..x2]  f[x0..x3]
9  0         2.0        1.0        3.0        1.0
10 1         3.0        7.0        6.0        0.0
11 2        10.0       19.0        0.0        0.0
12 3        29.0        0.0        0.0        0.0
13 Коэффициенты многочлена Ньютона:
14 [2.  1.  3.  1.]
15 Значения многочлена Ньютона:
16   X   F (Ньютон)
17 0.0    2.000
18 0.5    2.125
19 1.0    3.000
20 1.5    5.375
21 2.0   10.000
22 2.5   17.625
23 3.0   29.000

```

## 4 Интерполяция кубическими сплайнами

Необходимо построить интерполяционный многочлен с помощью кубических сплайнов (алгебраических многочленов третьей степени, где сплайн — фрагмент, отрезок чего-либо).

$x$	0	1	2	3
$f(x)$	2	3	10	29

## Код

```

1 import numpy as np
2 import pandas as pd
3 def create_spline_matrix(x, f):
4     n = len(x) - 1 # количество интервалов
5     # инициализируем матрицу 12x12 и вектор правой части
6     a = np.zeros((4*n, 4*n))
7     b = np.zeros(4*n)
8     # уравнение 1: интерполяция в левых концах
9     # s_i(x_i) = f_i
10    for i in range(n):
11        row = i
12        a[row, 4*i] = 1 # a_i
13        b[row] = f[i]

```

```

1 # уравнение 2: интерполяция в правых концах
2 # s_i(x_{i+1}) = f_{i+1}
3 for i in range(n):
4     row = n + i
5     h = x[i+1] - x[i]
6     a[row, 4*i] = 1 # a_i
7     a[row, 4*i + 1] = h # b_i
8     a[row, 4*i + 2] = h**2 # c_i
9     a[row, 4*i + 3] = h**3 # d_i
10    b[row] = f[i+1]
11 # уравнение 3: непрерывность первых производных
12 # s'_i(x_{i+1}) = s'_{i+1}(x_{i+1})
13 for i in range(n-1):
14     row = 2*n + i
15     h_i = x[i+1] - x[i]
16     h_ip1 = x[i+2] - x[i+1]
17     a[row, 4*i + 1] = 1 # b_i
18     a[row, 4*i + 2] = 2*h_i # 2*c_i*h_i
19     a[row, 4*i + 3] = 3*h_i**2 # 3*d_i*h_i^2
20     a[row, 4*(i+1) + 1] = -1 # -b_{i+1}
21     b[row] = 0
22 # уравнение 4: непрерывность вторых производных
23 # s''_i(x_{i+1}) = s''_{i+1}(x_{i+1})
24 for i in range(n-1):
25     row = 3*n - 1 + i
26     h_i = x[i+1] - x[i]
27     a[row, 4*i + 2] = 2 # 2*c_i
28     a[row, 4*i + 3] = 6*h_i # 6*d_i*h_i
29     a[row, 4*(i+1) + 2] = -2 # -2*c_{i+1}
30     b[row] = 0
31 # граничные условия (естественный сплайн)
32 # s''_0(x_0) = 0 и s''_{n-1}(x_n) = 0
33 row1 = 4*n - 2
34 a[row1, 2] = 2 # 2*c_0 = 0
35 row2 = 4*n - 1
36 h_last = x[n] - x[n-1]
37 a[row2, 4*(n-1) + 2] = 2 # 2*c_{n-1}
38 a[row2, 4*(n-1) + 3] = 6*h_last # 6*d_{n-1}*h_{n-1}
39 return a, b
40 def solve_spline_coefficients(x, f):
41     a, b = create_spline_matrix(x, f)
42     coefficients = np.linalg.solve(a, b)
43     return coefficients.reshape(-1, 4) # преобразуем в матрицу [n x 4]
44 # данные из условия
45 x = np.array([0, 1, 2, 3])
46 f = np.array([2, 3, 10, 29])
47 # создаем матрицу и решаем систему
48 a, b = create_spline_matrix(x, f)
49 coefficients = solve_spline_coefficients(x, f)
50 # создаем dataframe для красивого отображения матрицы
51 matrix_df = pd.DataFrame(a,
52                           columns=[f'coef_{i}' for i in range(12)],
53                           index=[f'eq_{i}' for i in range(12)])
54 # выводим результаты
55 print("Матрица системы 12x12:")
56 print(matrix_df)
57 print("\nВектор правой части:")
58 print(pd.Series(b, index=[f'eq_{i}' for i in range(12)]))
59 print("\nКоэффициенты сплайнов (по строкам: [a_i, b_i, c_i, d_i] для каждого интервала):")

```

```

1 print(pd.DataFrame(coefficients,
2                     columns=['a_i', 'b_i', 'c_i', 'd_i'],
3                     index=[f'interval {i}' for i in range(len(x)-1)]))
4 # проверка размерности
5 print(f"\nРазмер матрицы: {a.shape}")
6 print(f"Количество интервалов: {len(x)-1}")

```

## Результат

```

1 Матрица системы 12x12:
2   coef_0 coef_1 coef_2 coef_3 coef_4 coef_5 coef_6 coef_7 coef_8 coef_9 coef_10 coef_11
3 eq_0  1.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
4 eq_1  0.0   0.0   0.0   0.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
5 eq_2  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0   0.0   0.0   0.0
6 eq_3  1.0   1.0   1.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
7 eq_4  0.0   0.0   0.0   0.0   1.0   1.0   1.0   1.0   0.0   0.0   0.0   0.0
8 eq_5  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0   1.0   1.0   1.0
9 eq_6  0.0   1.0   2.0   3.0   0.0  -1.0   0.0   0.0   0.0   0.0   0.0   0.0
10 eq_7  0.0   0.0   0.0   0.0   0.0   1.0   2.0   3.0   0.0  -1.0   0.0   0.0
11 eq_8  0.0   0.0   2.0   6.0   0.0   0.0  -2.0   0.0   0.0   0.0   0.0   0.0
12 eq_9  0.0   0.0   0.0   0.0   0.0   0.0   2.0   6.0   0.0   0.0  -2.0   0.0
13 eq_10 0.0   0.0   2.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
14 eq_11 0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   6.0
15 Вектор правой части:
16 eq_0      2.0
17 eq_1      3.0
18 eq_2     10.0
19 eq_3      3.0
20 eq_4     10.0
21 eq_5     29.0
22 eq_6      0.0
23 eq_7      0.0
24 eq_8      0.0
25 eq_9      0.0
26 eq_10     0.0
27 eq_11     0.0

1 Коэффициенты сплайнов (по строкам: [a_i, b_i, c_i, d_i] для каждого интервала):
2   a_i  b_i  c_i  d_i
3 interval 0  2.0  0.2  0.0  0.8
4 interval 1  3.0  2.6  2.4  2.0
5 interval 2 10.0 13.4  8.4 -2.8
6 Размер матрицы: (12, 12)
7 Количество интервалов: 3

```

## 5 Метод Гаусса решения СЛАУ

Решить следующую СЛАУ методом Гаусса: метод Гаусса должен решать уравнения вида  $Ax = b$ , где

$$A = \begin{pmatrix} 5 & 0.05 & 0.05 & 0.05 & 0.05 \\ 0.06 & 6 & 0.06 & 0.06 & 0.06 \\ 0.07 & 0.07 & 7 & 0.07 & 0.07 \\ 0.08 & 0.08 & 0.08 & 8 & 0.08 \\ 0.09 & 0.09 & 0.09 & 0.09 & 9 \end{pmatrix} \quad b = \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{pmatrix}. \quad (1)$$

Код

```

1 import numpy as np
2 A = np.array([[5, 0.05, 0.05, 0.05, 0.05],
3               [0.06, 6, 0.06, 0.06, 0.06],
4               [0.07, 0.07, 7, 0.07, 0.07],
5               [0.08, 0.08, 0.08, 8, 0.08],
6               [0.09, 0.09, 0.09, 0.09, 9]])
7 print("1) Матрица A:\n", A)
8 print("Определитель матрицы A = ", np.linalg.det(A))
9 b = np.zeros(len(A))
10 print("\nКолонка b:")
11 for i in range(len(A)):
12     b[i] = A[i][i]
13     print("[", b[i], "]")
14 def forward_elimination(A, b):
15     n = len(b)
16     for k in range(n - 1):
17         for i in range(k + 1, n):
18             if A[i, k] != 0:
19                 factor = A[i, k] / A[k, k]
20                 A[i, k+1:n] = A[i, k+1:n] - factor * A[k, k+1:n]
21                 A[i, k] = 0 # Explicitly zero out for clarity
22                 b[i] = b[i] - factor * b[k]
23     return A, b
24 A_tmp, b_tmp = forward_elimination(A, b)
25 print("Матрица после прямого прохода:")
26 print(A_tmp)
27 def backward_substitution(U, b):
28     n = len(b)
29     x = np.zeros(n)
30     for i in range(n-1, -1, -1):
31         x[i] = b[i] - np.dot(U[i, i+1:], x[i+1:])
32         x[i] /= U[i, i]
33     return x
34 solution = backward_substitution(A_tmp, b_tmp)
35 print("Решение (A|b) методом Гаусса")
36 print("Вектор решения после обратного прохода: ")
37 for i in range(len(solution)):
38     print("x", str(i), "=", solution[i])
39 print(solution, "\n 4) x =")
40 for i in range(len(solution)):
41     print("[", solution[i], "]")
42 b = np.dot(A, b)

```

## Результат

```

1 1) Матрица A:
2 [[5.  0.05 0.05 0.05 0.05]
3  [0.06 6.  0.06 0.06 0.06]
4  [0.07 0.07 7.  0.07 0.07]
5  [0.08 0.08 0.08 8.  0.08]
6  [0.09 0.09 0.09 0.09 9.  ]]
7 Определитель матрицы A = 15105.180138048006
8 Колонка b:
9 [ 5.0 ]
10 [ 6.0 ]
11 [ 7.0 ]
12 [ 8.0 ]
13 [ 9.0 ]

```



```

1 Матрица после прямого прохода:
2 [[5.          0.05      0.05      0.05      0.05      ]
3  [0.          5.9994    0.0594    0.0594    0.0594    ]
4  [0.          0.        6.99861386 0.06861386 0.06861386]
5  [0.          0.        0.        7.99764706 0.07764706]
6  [0.          0.        0.        0.        8.99650485]]
7 Решение (A|b) методом Гаусса
8 Вектор решения после обратного прохода:
9 x 0 = 0.9615384615384615
10 x 1 = 0.9615384615384616
11 x 2 = 0.9615384615384616
12 x 3 = 0.9615384615384616
13 x 4 = 0.9615384615384612
14 [0.96153846 0.96153846 0.96153846 0.96153846 0.96153846]
15 4) x =
16 [ 0.9615384615384615 ]
17 [ 0.9615384615384616 ]
18 [ 0.9615384615384616 ]
19 [ 0.9615384615384616 ]
20 [ 0.9615384615384612 ]

```

## 6 Метод прогонки решения СЛАУ (трехдиагональных)

В данном случае решается система линейных уравнений вида  $Ax = b$ , где

$$A = \begin{pmatrix} 5 & 0.05 & 0.05 & 0.05 & 0.05 \\ 0.06 & 6 & 0.06 & 0.06 & 0.06 \\ 0.07 & 0.07 & 7 & 0.07 & 0.07 \\ 0.08 & 0.08 & 0.08 & 8 & 0.08 \\ 0.09 & 0.09 & 0.09 & 0.09 & 9 \end{pmatrix} \quad b = \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{pmatrix}. \quad (2)$$

### Код

```

1 import numpy as np
2 A = np.array([[5, 0.05, 0, 0, 0],
3               [0.06, 6, 0.06, 0, 0.],
4               [0, 0.07, 7, 0.07, 0],
5               [0, 0, 0.08, 8, 0.08],
6               [0, 0, 0, 0.09, 9]])
7 # Вычисляем вектор b
8 b = np.zeros(len(A))
9 for i in range(len(A)):
10     b[i] = A[i][i]
11 b = np.dot(A, b.reshape(-1,1))
12 print("Матрица A:")
13 print(A)
14 print("Столбец b:")
15 print(b)
16 Q = np.zeros(len(A))
17 P = np.zeros(len(A) - 1)
18 P[0] = -A[0][1]/A[0][0]
19 Q[0] = b[0][0]/A[0][0]
20 print("Прямая прогонка")
21 print("Список Pi и Qi")
22 for i in range(1, len(P)):
23     P[i] = (A[i][i+1])/(-A[i][i] - A[i][i-1] * P[i-1])
24 for i in range(1, len(Q)):
25     Q[i] = (A[i][i-1] * Q[i-1] - b[i][0]) / (-A[i][i] - A[i][i-1] * P[i-1])
26 print(P, Q)

```

```

1 print("Обратная прогонка")
2 x = np.zeros(len(A))
3 x[len(A)-1] = Q[len(A)-1]
4 print("x 5 =",x[len(A)-1])
5 for i in range(len(x) - 2, -1, -1):
6     x[i] = P[i] * x[i + 1] + Q[i]
7     print("x",i + 1,"= ", x[i])
8 print(x.reshape(-1,1))

```

## Результат

```

1 Матрица A:
2 [[5.  0.05 0.  0.  0. ]
3  [0.06 6.  0.06 0.  0. ]
4  [0.  0.07 7.  0.07 0. ]
5  [0.  0.  0.08 8.  0.08]
6  [0.  0.  0.  0.09 9. ]]
7 Столбец b:
8 [[25.3 ]
9  [36.72]
10 [49.98]
11 [65.28]
12 [81.72]]
13 Прямая прогонка
14 Список Pi и Qi
15 [-0.01 -0.010001 -0.010001 -0.010001] [5.06 6.070007 7.080008 8.090009 9.]
16 Обратная прогонка
17 x 5 = 9.0
18 x 4 = 8.0000000000000002
19 x 3 = 6.999999999999999
20 x 2 = 6.0
21 x 1 = 5.0000000000000001
22 [[5.]
23 [6.]
24 [7.]
25 [8.]
26 [9.]]

```

## 7 Метод простой итерации

При решении СЛАУ вида  $Ax = b$ , где  $A$  — квадратная матрица, мы можем преобразовать ее к эквивалентному виду:

$$\begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \dots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \dots & 0 \end{pmatrix} x = \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{pmatrix}. \quad (3)$$

Таким образом исходная система допускает представление в виде:

$$\alpha x + \beta = x, \quad (4)$$

а критерий остановки вычислений:

$$\|x^k - x^{k-1}\| < e. \quad (5)$$

## Код

```
1 import numpy as np
2 A = np.array([[5, 0.05, 0.05, 0.05, 0.05],
3               [0.06, 6, 0.06, 0.06, 0.06],
4               [0.07, 0.07, 7, 0.07, 0.07],
5               [0.08, 0.08, 0.08, 8, 0.08],
6               [0.09, 0.09, 0.09, 0.09, 9]])
7 print("Матрица A:")
8 print(A)
9 print("Определитель матрицы A:", np.linalg.det(A))
10 # Вычисляем вектор b
11 b = np.zeros(len(A))
12 for i in range(len(A)):
13     b[i] = A[i][i]
14 b = np.dot(A, b.reshape(-1,1))
15 print("Столбец b:")
16 print(b)
17 alpha = A.copy()
18 beta = b.copy()
19 for i in range(len(A)):
20     for j in range(len(A)):
21         if i == j:
22             alpha[i][j] = 0
23         else:
24             alpha[i][j] = - A[i][j]/A[i][i]
25 print("Матрица alpha:")
26 print(alpha)
27 for i in range(len(A)):
28     beta[i][0] = b[i][0]/A[i][i]
29 print("Столбец beta:")
30 print(beta)
31 epsilon = 10**-9
32 print("Считаем до точности epsilon=", epsilon)
33 xk = np.zeros(len(A)).reshape(-1,1)
34 print("x^(0) = ", xk)
35 def normStop (xk, xkp1, epsilon):
36     return max(np.abs(np.add(xk, -xkp1))) < epsilon
37 for i in range(17):
38     xkp1 = np.add(np.dot(alpha, xk), beta)
39     print(f"x^{i+1}=", xkp1)
40     if normStop(xk, xkp1, epsilon):
41         break
42     xk = xkp1
```

## Результат

```
1 Матрица A:
2 [[5.  0.05 0.05 0.05 0.05]
3  [0.06 6.  0.06 0.06 0.06]
4  [0.07 0.07 7.  0.07 0.07]
5  [0.08 0.08 0.08 8.  0.08]
6  [0.09 0.09 0.09 0.09 9.  ]]
7 Определитель матрицы A: 15105.180138048006
8 Столбец b:
9 [[26.5 ]
10 [37.74]
11 [50.96]
12 [66.16]
13 [83.34]]
```

```

1 Матрица alpha:
2 [[ 0.   -0.01 -0.01 -0.01 -0.01]
3  [-0.01  0.   -0.01 -0.01 -0.01]
4  [-0.01 -0.01  0.   -0.01 -0.01]
5  [-0.01 -0.01 -0.01  0.   -0.01]
6  [-0.01 -0.01 -0.01 -0.01  0.   ]]
7 Столбец beta:
8 [[5.3 ]
9  [6.29]
10 [7.28]
11 [8.27]
12 [9.26]]
13 Считаю до точности epsilon= 1e-09
14 x^(0) = [[0.]
15 [0.]
16 [0.]
17 [0.]
18 [0.]]
19 x^(1)= [[5.3 ]
20 [6.29]
21 [7.28]
22 [8.27]
23 [9.26]]
24 x^(2)= [[4.989 ]
25 [5.9889]
26 [6.9888]
27 [7.9887]
28 [8.9886]]
29 x^(3)= [[5.00045 ]
30 [6.000449]
31 [7.000448]
32 [8.000447]
33 [9.000446]]
34 x^(4)= [[4.9999821 ]
35 [5.99998209]
36 [6.99998208]
37 [7.99998207]
38 [8.99998206]]
39 x^(5)= [[5.00000072]
40 [6.00000072]
41 [7.00000072]
42 [8.00000072]
43 [9.00000072]]
44 x^(6)= [[4.99999997]
45 [5.99999997]
46 [6.99999997]
47 [7.99999997]
48 [8.99999997]]
49 x^(7)= [[5.]
50 [6.]
51 [7.]
52 [8.]
53 [9.]]
54 x^(8)= [[5.]
55 [6.]
56 [7.]
57 [8.]
58 [9.]]
59 x^(9)= [[5.]
60 [6.]
61 [7.]
62 [8.]
63 [9.]]

```

## 8 Задача Коши методами Эйлера

Решить задачу Коши: а) методом Эйлера; б) усовершенствованным методом Эйлера:

$$\begin{cases} y'(x) = 2Vx + Vx^2 - y(x) \\ y(1) = V \end{cases} \quad (6)$$

где  $y_{\text{точн}}(x) = Vx^2$ ,  $V$  — номер варианта.

Код

```
1 import numpy as np
2 import pandas as pd
3 def f(x, y, V=5):
4     return 2 * V * x + V * x**2 - y
5 # Метод Эйлера
6 def euler_method(x0, y0, h, n, V):
7     x = np.zeros(n + 1)
8     y = np.zeros(n + 1)
9     x[0] = x0
10    y[0] = y0
11    for i in range(n):
12        x[i + 1] = x[i] + h
13        y[i + 1] = y[i] + h * f(x[i], y[i], V)
14    return x, y
15 # Усовершенствованный метод Эйлера
16 def improved_euler_method(x0, y0, h, n, V):
17     x = np.zeros(n + 1)
18     y = np.zeros(n + 1)
19     x[0] = x0
20     y[0] = y0
21     for i in range(n):
22         x[i + 1] = x[i] + h
23         y_half = y[i] + (h / 2) * f(x[i], y[i], V)
24         x_half = x[i] + h / 2
25         y[i + 1] = y[i] + h * f(x_half, y_half, V)
26     return x, y
27 def exact_solution(x, V=5):
28     return V * x**2
29 # Параметры
30 x0 = 1
31 V = y0 = 5
32 h = 0.001 # шаг
33 n = 10
34 # Вычисление решений
35 x_euler, y_euler = euler_method(x0, y0, h, n, V)
36 x_improved, y_improved = improved_euler_method(x0, y0, h, n, V)
37 y_exact = exact_solution(x_euler)
38 # Вычисление погрешностей
39 error_euler = np.abs(y_euler - y_exact)
40 error_improved = np.abs(y_improved - y_exact)
41 print("\nМетод Эйлера: ")
42 print("-" * 130)
43 print("x:      ", " ".join(f"{x:>10.7f}" for x in x_euler))
44 print("y_M:     ", " ".join(f"{y:>10.7f}" for y in y_euler))
45 print("y_T:      ", " ".join(f"{y:>10.7f}" for y in y_exact))
46 print("Погрешн:", " ".join(f"{e:>10.7f}" for e in error_euler))
47 print("-" * 130)
```

```

1 print("\nУсовершенствованный метод Эйлера: ")
2 print("-" * 130)
3 print("x:      ", " ".join("{x:>10.7f}" for x in x_improved))
4 print("y_M:     ", " ".join("{y:>10.7f}" for y in y_improved))
5 print("y_T:     ", " ".join("{y:>10.7f}" for y in y_exact))
6 print("Погрешн:", " ".join("{e:>10.7f}" for e in error_improved))
7 print("-" * 130)

```

## Результат

Метод Эйлера:

```

-----
x:      1.0000000 1.0010000 1.0020000 1.0030000 1.0040000 1.0050000 1.0060000 1.0070000 1.0080000 1.0090000 1.0100000
y_M:     5.0000000 5.0100000 5.0200100 5.0300300 5.0400600 5.0501000 5.0601501 5.0702101 5.0802801 5.0903602 5.1004502
y_T:     5.0000000 5.0100050 5.0200200 5.0300450 5.0400800 5.0501250 5.0601800 5.0702450 5.0803200 5.0904050 5.1005000
Погрешн: 0.0000000 0.0000050 0.0000100 0.0000150 0.0000200 0.0000250 0.0000299 0.0000349 0.0000399 0.0000448 0.0000498
-----

```

Усовершенствованный метод Эйлера:

```

-----
x:      1.0000000 1.0010000 1.0020000 1.0030000 1.0040000 1.0050000 1.0060000 1.0070000 1.0080000 1.0090000 1.0100000
y_M:     5.0000000 5.0100050 5.0200200 5.0300450 5.0400800 5.0501250 5.0601800 5.0702450 5.0803200 5.0904050 5.1005000
y_T:     5.0000000 5.0100050 5.0200200 5.0300450 5.0400800 5.0501250 5.0601800 5.0702450 5.0803200 5.0904050 5.1005000
Погрешн: 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
-----

```

9 Решить краевую задачу разностным методом.

$$\begin{cases} y'' + x^2 y' + xy = 4Vx^4 - 3VTx^3 + 6Vx - 2VT \\ y'(0) = y(T) = 0 \\ V = T = 5 \end{cases} \quad (7)$$

```

1 V = 5 # номер варианта
2 def derivative(x):
3     return -(4 * V * x**4 - 3 * V**2 * x**3 + 6 * V * x - 2 * V**2)
4 def Y(x):
5     return V * x**2 * (x - V)
6 def p(x):
7     return -x**2
8 def q(x):
9     return -x
10 def main():
11     n = 10
12     x0 = 0
13     h = V / n
14     x = [x0 + i * h for i in range(n + 1)]
15     exact = [Y(xi) for xi in x]
16     f = [0.0] * (n + 1)
17     s = [0.0] * (n + 1)
18     t = [0.0] * (n + 1)
19     r = [0.0] * (n + 1)
20     f1 = [0.0] * (n + 1)
21     s1 = [0.0] * (n + 1)
22     y = [0.0] * (n + 1)
23     e = [0.0] * (n + 1)
24     for i in range(1, n):
25         f[i] = 0.5 * (1 + 0.5 * h * p(x[i]))
26         s[i] = 0.5 * (1 - 0.5 * h * p(x[i]))
27         t[i] = 1 + 0.5 * h**2 * q(x[i])
28         r[i] = 0.5 * h**2 * derivative(x[i])
29     f1[1] = 0.0
30     s1[1] = 0.0
31     for j in range(1, n):
32         denom = t[j] - f[j] * f1[j]
33         f1[j + 1] = s[j] / denom
34         s1[j + 1] = (r[j] + f[j] * s1[j]) / denom
35     # Граничное условие y[n] = 0 в оригинале неявно, принимаем y[n]=0
36     y[n] = 0.0

1     for j in range(n - 1, 0, -1):
2         y[j] = f1[j + 1] * y[j + 1] + s1[j + 1]
3     max_e = 0.0
4     max_e_index = 0
5     for i in range(n + 1):
6         e[i] = abs(y[i] - exact[i])
7         if (e[i] > max_e):
8             max_e = e[i]
9             max_e_index = i
10    print("x\t y\t exact\t e")
11    for i in range(n + 1):
12        print(f"{x[i]:.2f}\t {y[i]:.2f} \t {exact[i]:.2f}\t {e[i]:.8f}")
13    print("Максимальный e: ", max_e, "Номер максимального e: ", max_e_index)
14 if __name__ == "__main__":
15     main()

```

## Результат

1 x	y	exact	e
2 0.00	0.00	-0.00	0.00000000
3 0.50	-0.09	-5.62	5.53017248
4 1.00	-10.31	-20.00	9.68559849
5 1.50	-29.38	-39.38	9.99173440
6 2.00	-52.77	-60.00	7.22943620
7 2.50	-73.33	-78.12	4.79707715
8 3.00	-86.60	-90.00	3.39879569
9 3.50	-89.59	-91.88	2.28687417
10 4.00	-78.58	-80.00	1.41652296
11 4.50	-49.97	-50.62	0.65542909
12 5.00	0.00	0.00	0.00000000

13 Максимальный e: 9.991734404070492  
14 Номер максимального e: 3

## 10 Краевая задача методом неопределенных коэффициентов

$$\begin{cases} y'' + x^2 y' + xy = 4Vx^4 - 3VTx^3 + 6Vx - 2VT \\ y'(0) = y(T) = 0 \\ V = T = 5 \end{cases} \quad (8)$$

### Код

```

1 import numpy as np
2 V = 5 # номер варианта
3 h = 0.1 # шаг
4 n = int(V / h)
5 maxx = V
6 # y точный
7 def ytoch(x):
8     return V * x * x * (x - maxx)
9 def f(x):
10     return 4 * V * (x ** 4) - 3 * V * V * (x ** 3) + 6 * V * x - 2 * V * V
11 def p(x):
12     return x * x
13 def q(x):
14     return x
15 def phi_k(x, k):
16     return x ** k * (x - V)
17 def dphi_k(x, k):
18     return (k + 1) * x ** k - V * k * x ** (k - 1)
19 def ddphi_k(x, k):
20     return k * (k + 1) * x ** (k - 1) - V * k * (k - 1) * x ** (k - 2)
21 # Создаем сетку
22 xk = [x * h for x in range(n + 1)]
23 ykToch = [ytoch(x) for x in xk]
24 print("Проверка точного решения в ключевых точках:")
25 test_points = [0, 1, 2, 3, 4, 5]
26 for x in test_points:
27     print(f"x = {x}: y_toch = {ytoch(x):.2f}")
28 # Инициализация матрицы A и вектора b
29 A = np.zeros((n, n))
30 b = np.zeros(n)
31 # Заполнение матрицы A и вектора b для ВНУТРЕННИХ точек (i=1..n-1)
32 for i in range(1, n + 1):
33     b[i - 1] = f(xk[i])
34     for k in range(1, n + 1):
35         A[i - 1][k - 1] = ddphi_k(xk[i], k) + p(xk[i]) * dphi_k(xk[i], k) + q(xk[i]) *
            phi_k(xk[i], k)
36 print(f"\nРазмерность матрицы A: {A.shape}")

```



```

1 print(f"Размерность вектора b: {b.shape}")
2 # Метод Гаусса для решения СЛАУ  $A * c = b$ 
3 def gauss_elimination(A, b):
4     n = len(b)
5     # Прямой ход метода Гаусса
6     for i in range(n):
7         # Поиск максимального элемента в столбце для улучшения устойчивости
8         max_row = i
9         max_val = abs(A[i, i])
10        for j in range(i + 1, n):
11            if abs(A[j, i]) > max_val:
12                max_val = abs(A[j, i])
13                max_row = j
14        # Обмен строк, если необходимо
15        if max_row != i:
16            A[[i, max_row]] = A[[max_row, i]]
17            b[i], b[max_row] = b[max_row], b[i]
18        # Проверка на нулевой диагональный элемент
19        if abs(A[i, i]) < 1e-12:
20            print(f"Внимание: малый диагональный элемент A[{i},{i}] = {A[i, i]}")
21            A[i, i] = 1e-12
22        # Нормировка строки
23        pivot = A[i, i]
24        for j in range(i, n):
25            A[i, j] /= pivot
26        b[i] /= pivot
27        # Исключение переменной из нижележащих строк
28        for j in range(i + 1, n):
29            factor = A[j, i]
30            for k in range(i, n):
31                A[j, k] -= factor * A[i, k]
32            b[j] -= factor * b[i]
33        # Обратный ход метода Гаусса
34        c = np.zeros(n)
35        for i in range(n - 1, -1, -1):
36            c[i] = b[i]
37            for j in range(i + 1, n):
38                c[i] -= A[i, j] * c[j]
39        return c
40 # Решение системы
41 print("\nРешение СЛАУ методом Гаусса")
42 c = gauss_elimination(A.copy(), b.copy())
43 print("\nПервые 10 коэффициентов a_k:")
44 for i in range(min(10, len(c))):
45     print(f"a_{i+1} = {c[i]:.6e}")
46 # Построение приближенного решения
47 def y_approx(x, c):
48     result = 0
49     for k in range(1, len(c) + 1):
50         result += c[k - 1] * phi_k(x, k)
51     return result
52 # Сравнение точного и приближенного решений
53 print("\nСравнение решений в некоторых точках:")
54 print("x\t\tТочное y\t\tПриближенное y\t\tОтносительная погрешность")
55 for x in range(V + 1):
56     y_exact = ytoch(x)
57     y_appr = y_approx(x, c)
58     rel_error = abs((y_appr - y_exact) / y_exact) if abs(y_exact) > 1e-12
59     else abs(y_appr)
60     print(f"{x}\t\t{y_exact:.4f}\t\t{y_appr:.4f}\t\t{rel_error:.2e}")

```

## Результат

```

1 Проверка точного решения в ключевых точках:
2 x = 0: y_toch = 0.00
3 x = 1: y_toch = -20.00
4 x = 2: y_toch = -60.00
5 x = 3: y_toch = -90.00
6 x = 4: y_toch = -80.00
7 x = 5: y_toch = 0.00
8 Размерность матрицы A: (50, 50)
9 Размерность вектора b: (50,)
10 Решение СЛАУ методом Гаусса
11 Первые 10 коэффициентов a_k:
12 a_1 = 1.507017e-12
13 a_2 = 5.000000e+00
14 a_3 = 2.660881e-10
15 a_4 = -1.744293e-09
16 a_5 = 7.754601e-09
17 a_6 = -2.451899e-08
18 a_7 = 5.696992e-08
19 a_8 = -9.949613e-08
20 a_9 = 1.324989e-07
21 a_10 = -1.354025e-07
22 Сравнение решений в некоторых точках:
23 x          Точное y          Приближенное y          Относительная погрешность
24 0           0.0000           0.0000           0.00e+00
25 1          -20.0000          -20.0000          2.84e-14
26 2          -60.0000          -60.0000          3.67e-15
27 3          -90.0000          -90.0000          4.74e-16
28 4          -80.0000          -80.0000          1.78e-16
29 5           0.0000           0.0000          0.00e+00

```

## 11 Метод неопределенных коэффициентов

Решите следующее интегральное уравнение:

$$y(x) + 1 * \int_0^1 (xt + x^2 t^2 + x^3 t^3) y(t) dt = V \left( \frac{4}{3}x + \frac{1}{4}x^2 + \frac{1}{5}x^3 \right) \quad (9)$$

Код

```

1 import numpy as np
2 from scipy import integrate
3 def rhs_func(x, variant): # Правая часть уравнения
4     return variant * (4 / 3 * x + 1 / 4 * x**2 + 1 / 5 * x**3)
5 def build_alpha(size): # Матрица коэффициентов a_ij
6     alpha = np.zeros((size, size))
7     for i in range(size):
8         for j in range(size):
9             def integrand(t):
10                 if i == 0:
11                     ai = t
12                 elif i == 1:
13                     ai = t**2
14                 else:
15                     ai = t**3
16                 if j == 0:
17                     bj = t
18                 elif j == 1:
19                     bj = t**2
20                 else:
21                     bj = t**3

```

```

1         return ai * bj
2         alpha[i, j], _ = integrate.quad(integrand, 0, 1)
3     return alpha
4 def build_gamma(size, variant): # Вектор gamma_i
5     gamma = np.zeros(size)
6     for i in range(size):
7         def integrand(t):
8             if i == 0:
9                 bi = t
10            elif i == 1:
11                bi = t**2
12            else:
13                bi = t**3
14            return rhs_func(t, variant) * bi
15        gamma[i], _ = integrate.quad(integrand, 0, 1)
16    return gamma
17 def gauss_method(A, b):# Метод Гаусса
18     A = A.astype(float)
19     b = b.astype(float)
20     n = len(b)
21     for i in range(n): # Прямой ход
22         max_row = i + np.argmax(abs(A[i:, i])) # Поиск максимального элемента для выбора
            главного элемента
23         if A[max_row, i] == 0:
24             raise ValueError("Система не имеет единственного решения (нулевой ведущий
            элемент).")
25         if max_row != i: # Перестановка строк
26             A[[i, max_row]] = A[[max_row, i]]
27             b[[i, max_row]] = b[[max_row, i]]
28         pivot = A[i, i] # Нормализация ведущей строки
29         A[i] = A[i] / pivot
30         b[i] = b[i] / pivot
31         for j in range(i + 1, n): # Обнуление элементов под ведущим
32             factor = A[j, i]
33             A[j] = A[j] - factor * A[i]
34             b[j] = b[j] - factor * b[i]
35     x = np.zeros(n) # Обратный ход
36     for i in range(n - 1, -1, -1):
37         x[i] = b[i] - np.dot(A[i, i + 1:], x[i + 1:])
38     return x
39 def fredholm_solver(variant, rank=3):
40     alpha = build_alpha(rank)
41     gamma = build_gamma(rank, variant)
42     system_matrix = np.eye(rank) + alpha
43     coeffs = gauss_method(system_matrix, gamma)
44     step = 0.1
45     x_vals = np.arange(0, 1 + step, step)
46     y_num = np.zeros_like(x_vals)
47     for i, x in enumerate(x_vals):
48         y_val = rhs_func(x, variant)
49         for j in range(rank):
50             if j == 0:
51                 aj = x
52             elif j == 1:
53                 aj = x**2
54             else:
55                 aj = x**3
56             y_val -= coeffs[j] * aj
57         y_num[i] = y_val
58     y_true = variant * x_vals
59     err = np.abs(y_num - y_true)
60     return x_vals, y_num, y_true, err

```

```

1 def main():
2     V = int(input("Введите номер варианта (v): "))
3     x, y_calc, y_exact, error = fredholm_solver(V)
4     print("\nРешение интегрального уравнения Фредгольма (вырожденное ядро)")
5     print("x:          ", " ".join(f"{float(xi):7.3f}" for xi in x))
6     print("y_мет:       ", " ".join(f"{float(yi):7.3f}" for yi in y_calc))
7     print("y_точн:       ", " ".join(f"{float(yi):7.3f}" for yi in y_exact))
8     print("погрешн: ", " ".join(f"{ei:7.3f}" for ei in error))
9 if __name__ == "__main__":
10     main()

```

## Результат

```

1 Введите номер варианта (v): 5
2 Решение интегрального уравнения Фредгольма (вырожденное ядро)
3 x:      0.000  0.100  0.200  0.300  0.400  0.500  0.600  0.700  0.800  0.900  1.000
4 y_мет:   0.000  0.500  1.000  1.500  2.000  2.500  3.000  3.500  4.000  4.500  5.000
5 y_точн:  0.000  0.500  1.000  1.500  2.000  2.500  3.000  3.500  4.000  4.500  5.000
6 погрешн: 0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000

```

## 12 Метод квадратур

Решите следующее интегральное уравнение:

$$y(x) + 1 * \int_0^1 (xt + x^2 t^2 + x^3 t^3) y(t) dt = V \left( \frac{4}{3}x + \frac{1}{4}x^2 + \frac{1}{5}x^3 \right) \quad (10)$$

## Код

```

1 import numpy as np
2 import pandas as pd
3 from scipy.integrate import quad
4 from scipy.linalg import solve
5 def solve_fredholm_degenerate_correct(V):
6     a, b = 0, 1
7     n = 3
8     lam = 1
9     def y_exact(x):
10         return V * x
11     def f(x):
12         return V * (4 / 3 * x + 1 / 4 * x**2 + 1 / 5 * x**3)
13     a_funcs = [lambda x: x, lambda x: x**2, lambda x: x**3]
14     b_funcs = [lambda t: t, lambda t: t**2, lambda t: t**3]
15     alpha = np.zeros((n, n))
16     gamma = np.zeros(n)
17     for i in range(n):
18         for k in range(n):
19             alpha[i, k], _ = quad(lambda x: a_funcs[i](x) * b_funcs[k](x), a, b)
20             gamma[i], _ = quad(lambda x: f(x) * b_funcs[i](x), a, b)
21     A = np.eye(n) + lam * alpha.T
22     q = solve(A, gamma)
23     def y_numerical(x):
24         result = f(x)
25         for i in range(n):
26             result -= lam * q[i] * a_funcs[i](x)
27     return result
28 x_test = np.linspace(a, b, 10)
29 y_num_vals = [y_numerical(x) for x in x_test]
30 y_ex_vals = [y_exact(x) for x in x_test]

```

```

1 errors = [abs(y_num_vals[i] - y_ex_vals[i]) for i in range(len(x_test))]
2 df = pd.DataFrame(
3     {"x": x_test, "y_метода": y_num_vals, "y_точн": y_ex_vals, "eps": errors}
4 )
5 print(df.to_string(index=False))
6 print("\n")
7 return y_numerical, y_exact
8 if __name__ == "__main__":
9     V = 5 # номер варианта
10    solve_fredholm_degenerate_correct(V)

```

## Результат

	x	y_метода	y_точн	eps
2	0.000000	0.000000	0.000000	0.000000e+00
3	0.111111	0.555556	0.555556	1.110223e-16
4	0.222222	1.111111	1.111111	0.000000e+00
5	0.333333	1.666667	1.666667	2.220446e-16
6	0.444444	2.222222	2.222222	0.000000e+00
7	0.555556	2.777778	2.777778	4.440892e-16
8	0.666667	3.333333	3.333333	1.332268e-15
9	0.777778	3.888889	3.888889	0.000000e+00
10	0.888889	4.444444	4.444444	0.000000e+00
11	1.000000	5.000000	5.000000	0.000000e+00