Задание 1. По данным интерполяции построить интерполяционный многочлен в общем виде.

```
import numpy

# входные данные
x_data = [0, 1, 2, 3]
f_data = [2, 3, 10, 29]
x_np_data = numpy.array(x_data)
f_np_data = numpy.array(f_data)

print("Исходные данные:")
for i in range (0, 4):
    print('X' + str(i) + ': ' + str(x_data[i]) + '\tF' + str(i) + ': ' +
str(f_data[i]))

# сторим матрицу
matrix = numpy.vander(x_np_data, len(x_data), increasing=True)
matrix_flipped = numpy.flip(matrix, axis=1)
print("\nСистема построена:")
for i in range(0, 4):
    s = ""
    for j in range (0, 4):
        s += str(matrix_flipped[i][j]) + ' * a' + str(4 - j - 1) + ' + '
    print(s[:-2] + ' = ' + str(f_data[i]))

print()

# решение системы уравнений для нахождения коэффициентов многочлена
coefficients = numpy.linalg.solve(matrix, f_np_data)
print("\nКоэффициенты найдены:")
for i in range (0, 4):
    print('a' + str(i) + ': ' + str(coefficients[i]))

print("\nP3 (x): ")
for i in range (0, 7):
    print('x = ' + str(i / 2) + ': ' + str(
        coefficients[3] * (i / 2) * (i / 2) * (i / 2)
        + coefficients[2] * (i / 2) * (i / 2)
        + coefficients[1] * (i / 2)
        + coefficients[0]
        ))
```

```
Исходные данные:
X0: 0   F0: 2
X1: 1   F1: 3
X2: 2   F2: 10
X3: 3   F3: 29

Система построена:
0 * a3 + 0 * a2 + 0 * a1 + 1 * a0  = 2
1 * a3 + 1 * a2 + 1 * a1 + 1 * a0  = 3
8 * a3 + 4 * a2 + 2 * a1 + 1 * a0  = 10
27 * a3 + 9 * a2 + 3 * a1 + 1 * a0  = 29

Коэффициенты найдены:
a0: 2.0
```

```
a1: 0.0
a2: -0.0
a3: 1.0

P3 (x):
x = 0.0: 2.0
x = 0.5: 2.125
x = 1.0: 3.0
x = 1.5: 5.375
x = 2.0: 10.0
x = 2.5: 17.625
x = 3.0: 29.0
```

Задание 2. По данным интерполяции с предыдущего задания построить интерполяционный многочлен в форме Лагранжа.

```
import numpy

def lagrange_fundamental(k, x_nodes, z):
        Lk = 1.0
        for i, xi in enumerate(x_nodes):
            if i != k:
                Lk *= (z - xi) / (x_nodes[k] - xi)
        return Lk

def lagrange_interpolation(x_nodes, y_nodes, z):
        P = 0.0
        for k in range(len(x_nodes)):
            P += y_nodes[k] * lagrange_fundamental(k, x_nodes, z)
        return P

x_data = [0, 1, 2, 3]
f_data = [2, 3, 10, 29]
x_np_data = numpy.array(x_data)
f_np_data = numpy.array(f_data)

print("Входные данные:")
for i in range (0, 4):
    print('X' + str(i) + ': ' + str(x_data[i]) + '\tF' + str(i) + ': ' +
str(f_data[i]))

print("\nL3 (x): ")
for i in range (0, 7):
    print('x = ' + str(i / 2) + ': ' + str(
        lagrange_interpolation(x_data, f_data, i / 2)
        ))
```

```
Входные данные:
X0: 0   F0: 2
X1: 1   F1: 3
X2: 2   F2: 10
X3: 3   F3: 29

L3 (x):
x = 0.0: 2.0
x = 0.5: 2.125
x = 1.0: 3.0
x = 1.5: 5.375
```

```
x = 2.0: 10.0
x = 2.5: 17.625
x = 3.0: 29.0
```

Задание 3. По данным интерполяции из предыдущих двух заданий построить интерполяционный многочлен в форме Ньютона.

```python
import numpy as np
import pandas as pd

# Входные данные
x_data = np.array([0, 1, 2, 3])
f_data = np.array([2, 3, 10, 29])

print("Исходные данные:")
print(pd.DataFrame({'X': x_data, 'F': f_data}))

# Построение таблицы разделённых разностей
def divided_difference_table(x, y):
    n = len(x)
    table = np.zeros((n, n))
    table[:,0] = y
    for j in range(1, n):
        for i in range(n - j):
            table[i,j] = (table[i+1,j-1] - table[i,j-1]) / (x[i+j] - x[i])
    return table

dd_table = divided_difference_table(x_data, f_data)

# Вывод таблицы разделённых разностей
dd_df = pd.DataFrame(dd_table, columns=[f"f[x0..x{j}]" for j in range(len(x_data))])
print("\nТаблица разделённых разностей (матрица Ньютона):")
print(dd_df)

# Коэффициенты Ньютона (верхняя строка таблицы)
newton_coeffs = dd_table[0,:]
print("\nКоэффициенты многочлена Ньютона:")
print(newton_coeffs)

# Функция для вычисления значения многочлена Ньютона
def newton_polynomial(x, coeffs, x_data):
    n = len(coeffs) - 1
    p = coeffs[n]
    for k in range(1, n + 1):
        p = coeffs[n-k] + (x - x_data[n-k]) * p
    return p

# Вычисление значений на промежуточных точках
x_half = [(x_data[i] + x_data[i+1])/2 for i in range(len(x_data)-1)]
f_half_newton = [newton_polynomial(x, newton_coeffs, x_data) for x in x_half]

# Объединяем узлы и промежуточные точки
x_combined = np.concatenate([x_data, x_half])
y_newton = np.concatenate([f_data, f_half_newton])

# Сортируем по X
sort_idx = np.argsort(x_combined)
```

```
x_combined = x_combined[sort_idx]
y_newton = y_newton[sort_idx]

# Создаем итоговую таблицу
newton_df = pd.DataFrame({
    'X': x_combined,
    'F (Ньютон)': y_newton
})

print("\nЗначения многочлена Ньютона:")
print(newton_df.to_string(index=False))
```

```
Исходные данные:
   X   F
0  0   2
1  1   3
2  2  10
3  3  29

Таблица разделённых разностей (матрица Ньютона):
   f[x0..x0]  f[x0..x1]  f[x0..x2]  f[x0..x3]
0        2.0        1.0        3.0        1.0
1        3.0        7.0        6.0        0.0
2       10.0       19.0        0.0        0.0
3       29.0        0.0        0.0        0.0

Коэффициенты многочлена Ньютона:
[2. 1. 3. 1.]

Значения многочлена Ньютона:
  X  F (Ньютон)
0.0       2.000
0.5       2.125
1.0       3.000
1.5       5.375
2.0      10.000
2.5      17.625
3.0      29.000
```

Задание 4. По данным интерполяции с предыдущего задания построить кусочно-непрерывную склейку кубических сплайнов.

```
import numpy as np
import pandas as pd

def create_spline_matrix(x, f):
    n = len(x) - 1  # количество интервалов

    # инициализируем матрицу 12x12 и вектор правой части
    a = np.zeros((4*n, 4*n))
    b = np.zeros(4*n)

    # уравнение 1: интерполяция в левых концах
    # s_i(x_i) = f_i
    for i in range(n):
        row = i
        a[row, 4*i] = 1  # a_i
```

```
        b[row] = f[i]

    # уравнение 2: интерполяция в правых концах
    # s_i(x_{i+1}) = f_{i+1}
    for i in range(n):
        row = n + i
        h = x[i+1] - x[i]
        a[row, 4*i] = 1          # a_i
        a[row, 4*i + 1] = h      # b_i
        a[row, 4*i + 2] = h**2   # c_i
        a[row, 4*i + 3] = h**3   # d_i
        b[row] = f[i+1]

    # уравнение 3: непрерывность первых производных
    # s'_i(x_{i+1}) = s'_{i+1}(x_{i+1})
    for i in range(n-1):
        row = 2*n + i
        h_i = x[i+1] - x[i]
        h_ip1 = x[i+2] - x[i+1]

        a[row, 4*i + 1] = 1          # b_i
        a[row, 4*i + 2] = 2*h_i      # 2*c_i*h_i
        a[row, 4*i + 3] = 3*h_i**2   # 3*d_i*h_i^2
        a[row, 4*(i+1) + 1] = -1     # -b_{i+1}
        b[row] = 0

    # уравнение 4: непрерывность вторых производных
    # s''_i(x_{i+1}) = s''_{i+1}(x_{i+1})
    for i in range(n-1):
        row = 3*n - 1 + i
        h_i = x[i+1] - x[i]

        a[row, 4*i + 2] = 2          # 2*c_i
        a[row, 4*i + 3] = 6*h_i      # 6*d_i*h_i
        a[row, 4*(i+1) + 2] = -2     # -2*c_{i+1}
        b[row] = 0

    # граничные условия (естественный сплайн)
    # s''_0(x_0) = 0 и s''_{n-1}(x_n) = 0
    row1 = 4*n - 2
    a[row1, 2] = 2  # 2*c_0 = 0

    row2 = 4*n - 1
    h_last = x[n] - x[n-1]
    a[row2, 4*(n-1) + 2] = 2          # 2*c_{n-1}
    a[row2, 4*(n-1) + 3] = 6*h_last  # 6*d_{n-1}*h_{n-1}

    return a, b

def solve_spline_coefficients(x, f):
    a, b = create_spline_matrix(x, f)
    coefficients = np.linalg.solve(a, b)
    return coefficients.reshape(-1, 4)  # преобразуем в матрицу [n x 4]

# данные из условия
x = np.array([0, 1, 2, 3])
```

```python
f = np.array([2, 3, 10, 29])

# создаем матрицу и решаем систему
a, b = create_spline_matrix(x, f)
coefficients = solve_spline_coefficients(x, f)

# создаем dataframe для красивого отображения матрицы
matrix_df = pd.DataFrame(a,
                         columns=[f'coef_{i}' for i in range(12)],
                         index=[f'eq_{i}' for i in range(12)])

# выводим результаты
print("Матрица системы 12x12:")
print(matrix_df)
print("\nВектор правой части:")
print(pd.Series(b, index=[f'eq_{i}' for i in range(12)]))
print("\nКоэффициенты сплайнов (по строкам: [a_i, b_i, c_i, d_i] для каждого
интервала):")
print(pd.DataFrame(coefficients,
                   columns=['a_i', 'b_i', 'c_i', 'd_i'],
                   index=[f'interval {i}' for i in range(len(x)-1)]))

# проверка размерности
print(f"\nРазмер матрицы: {a.shape}")
print(f"Количество интервалов: {len(x)-1}")
```

```
Матрица системы 12x12:
       coef_0  coef_1  coef_2  coef_3  coef_4  coef_5  coef_6  coef_7  coef_8  coef_9
coef_10  coef_11
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
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
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
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
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
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
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
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
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
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
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
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

Вектор правой части:
eq_0      2.0
eq_1      3.0
```

```
eq_2     10.0
eq_3      3.0
eq_4     10.0
eq_5     29.0
eq_6      0.0
eq_7      0.0
eq_8      0.0
eq_9      0.0
eq_10     0.0
eq_11     0.0

Коэффициенты сплайнов (по строкам: [a_i, b_i, c_i, d_i] для каждого интервала):
             a_i   b_i  c_i  d_i
interval 0   2.0   0.2  0.0  0.8
interval 1   3.0   2.6  2.4  2.0
interval 2  10.0  13.4  8.4 -2.8

Размер матрицы: (12, 12)
Количество интервалов: 3
```

Задание 5. Решить СЛАУ методом Гаусса.

```python
import numpy as np

A = np.array([[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]])
print("1) Матрица A:\n", A)
print("Определитель матрицы A = ", np.linalg.det(A))
b = np.zeros(len(A))
print("\nКолонка b:")
for i in range(len(A)):
    b[i] = A[i][i]
    print("[", b[i], "]")

def forward_elimination(A, b):
    n = len(b)
    for k in range(n - 1):
        for i in range(k + 1, n):
            if A[i, k] != 0:
                factor = A[i, k] / A[k, k]
                A[i, k+1:n] = A[i, k+1:n] - factor * A[k, k+1:n]
                A[i, k] = 0  # Explicitly zero out for clarity
                b[i] = b[i] - factor * b[k]

    return A, b

A_tmp, b_tmp = forward_elimination(A, b)
print("Матрица после прямого прохода:")
print(A_tmp)

def backward_substitution(U, b):
    n = len(b)
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
```

```
        x[i] = b[i] - np.dot(U[i, i+1:], x[i+1:])
        x[i] /= U[i, i]
    return x

solution = backward_substitution(A_tmp, b_tmp)

print("Решение (A|b) методом Гаусса")
print("Вектор решения после обратного прохода: ")

for i in range(len(solution)):
    print("x", str(i), "=", solution[i])

print(solution, "\n 4) x =")
for i in range(len(solution)):
    print("[", solution[i], "]")
b = np.dot(A, b)
```

```
1) Матрица A:
 [[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.  ]]
Определитель матрицы A =  15105.180138048006

Колонка b:
[ 5.0 ]
[ 6.0 ]
[ 7.0 ]
[ 8.0 ]
[ 9.0 ]
Матрица после прямого прохода:
[[5.         0.05       0.05       0.05       0.05      ]
 [0.         5.9994     0.0594     0.0594     0.0594    ]
 [0.         0.         6.99861386 0.06861386 0.06861386]
 [0.         0.         0.         7.99764706 0.07764706]
 [0.         0.         0.         0.         8.99650485]]
Решение (A|b) методом Гаусса
Вектор решения после обратного прохода:
x 0 = 0.9615384615384615
x 1 = 0.9615384615384616
x 2 = 0.9615384615384616
x 3 = 0.9615384615384616
x 4 = 0.9615384615384612
[0.96153846 0.96153846 0.96153846 0.96153846 0.96153846]
 4) x =
[ 0.9615384615384615 ]
[ 0.9615384615384616 ]
[ 0.9615384615384616 ]
[ 0.9615384615384616 ]
[ 0.9615384615384612 ]
```

Задание 6. Решить СЛАУ с помощью прогона.

```
import numpy as np

# Исходные данные
A = np.array([[5, 0.05, 0, 0, 0],
```

```
              [0.06, 6, 0.06, 0, 0.],
              [0, 0.07, 7, 0.07, 0],
              [0, 0, 0.08, 8, 0.08],
              [0, 0, 0, 0.09, 9]])

# Вычисляем вектор b
b = np.zeros(len(A))
for i in range(len(A)):
    b[i] = A[i][i]
b = np.dot(A, b.reshape(-1,1))

print("Матрица A:")
print(A)
print("Столбец b:")
print(b)

Q = np.zeros(len(A))
P = np.zeros(len(A) - 1)
P[0] = -A[0][1]/A[0][0]
Q[0] = b[0][0]/A[0][0]

print("Прямая прогонка")
print("Список Pi и Qi")
for i in range(1, len(P)):
    P[i] = (A[i][i+1])/(-A[i][i] - A[i][i-1] * P[i-1])
for i in range(1, len(Q)):
    Q[i] = (A[i][i-1] * Q[i-1] - b[i][0]) / (-A[i][i] - A[i][i-1] * P[i-1])
print(P, Q)

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

```
Матрица A:
[[5.   0.05 0.   0.   0.  ]
 [0.06 6.   0.06 0.   0.  ]
 [0.   0.07 7.   0.07 0.  ]
 [0.   0.   0.08 8.   0.08]
 [0.   0.   0.   0.09 9.  ]]
Столбец b:
[[25.3 ]
 [36.72]
 [49.98]
 [65.28]
 [81.72]]
Прямая прогонка
Список Pi и Qi
[-0.01 -0.010001 -0.010001 -0.010001] [5.06 6.070007 7.080008 8.090009 9.]
Обратная прогонка
x 5 = 9.0
x 4 =  8.000000000000002
x 3 =  6.999999999999999
```

```
x 2 =  6.0
x 1 =  5.000000000000001
[[5.]
 [6.]
 [7.]
 [8.]
 [9.]]
```

Задание 7. Решить СЛАУ методом простой итерации.

```
import numpy as np

A = np.array([[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]])

print("Матрица A:")
print(A)
print("Определитель матрицы A:", np.linalg.det(A))

# Вычисляем вектор b
b = np.zeros(len(A))
for i in range(len(A)):
    b[i] = A[i][i]
b = np.dot(A, b.reshape(-1,1))

print("Столбец b:")
print(b)

alpha = A.copy()
beta = b.copy()
for i in range(len(A)):
    for j in range(len(A)):
        if i == j:
            alpha[i][j] = 0
        else:
            alpha[i][j] = - A[i][j]/A[i][i]
print("Матрица alpha:")
print(alpha)

for i in range(len(A)):
    beta[i][0] = b[i][0]/A[i][i]
print("Столбец beta:")
print(beta)
epsilon = 10**-9
print("Считаем до точности epsilon=", epsilon)
xk = np.zeros(len(A)).reshape(-1,1)

print("x^(0) = ", xk)
def normStop (xk, xkp1, epsilon):
    return max(np.abs(np.add(xk, -xkp1))) < epsilon
for i in range(17):
    xkp1 = np.add(np.dot(alpha, xk), beta)
    print(f"x^({i+1})=", xkp1)
    if normStop(xk, xkp1, epsilon):
```

```
        break
    xk = xkp1
```

```
Матрица A:
[[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.  ]]
Определитель матрицы A: 15105.180138048006
Столбец b:
[[26.5 ]
 [37.74]
 [50.96]
 [66.16]
 [83.34]]
Матрица alpha:
[[ 0.   -0.01 -0.01 -0.01 -0.01]
 [-0.01  0.   -0.01 -0.01 -0.01]
 [-0.01 -0.01  0.   -0.01 -0.01]
 [-0.01 -0.01 -0.01  0.   -0.01]
 [-0.01 -0.01 -0.01 -0.01  0.  ]]
Столбец beta:
[[5.3 ]
 [6.29]
 [7.28]
 [8.27]
 [9.26]]
Считаем до точности epsilon= 1e-09
x^(0) =  [[0.]
 [0.]
 [0.]
 [0.]
 [0.]]
x^(1)= [[5.3 ]
 [6.29]
 [7.28]
 [8.27]
 [9.26]]
x^(2)= [[4.989 ]
 [5.9889]
 [6.9888]
 [7.9887]
 [8.9886]]
x^(3)= [[5.00045 ]
 [6.000449]
 [7.000448]
 [8.000447]
 [9.000446]]
x^(4)= [[4.9999821 ]
 [5.99998209]
 [6.99998208]
 [7.99998207]
 [8.99998206]]
x^(5)= [[5.00000072]
 [6.00000072]
 [7.00000072]
```

```
 [8.00000072]
 [9.00000072]]
x^(6)= [[4.99999997]
 [5.99999997]
 [6.99999997]
 [7.99999997]
 [8.99999997]]
x^(7)= [[5.]
 [6.]
 [7.]
 [8.]
 [9.]]
x^(8)= [[5.]
 [6.]
 [7.]
 [8.]
 [9.]]
x^(9)= [[5.]
 [6.]
 [7.]
 [8.]
 [9.]]
```

Задание 8. Решить задачу Коши: а) методом Эйлера; б) усовершенствованным методом Эйлера: $y' = 2 \cdot V \cdot x + V \cdot x^2 - y, y(x_0) = V \cdot x^2$.

```python
import numpy as np
import pandas as pd

def f(x, y, V=5):
    return 2 * V * x + V * x**2 - y

# Метод Эйлера
def euler_method(x0, y0, h, n, V):
    x = np.zeros(n + 1)
    y = np.zeros(n + 1)
    x[0] = x0
    y[0] = y0

    for i in range(n):
        x[i + 1] = x[i] + h
        y[i + 1] = y[i] + h * f(x[i], y[i], V)

    return x, y

# Усовершенствованный метод Эйлера
def improved_euler_method(x0, y0, h, n, V):
    x = np.zeros(n + 1)
    y = np.zeros(n + 1)
    x[0] = x0
    y[0] = y0

    for i in range(n):
        x[i + 1] = x[i] + h
        y_half = y[i] + (h / 2) * f(x[i], y[i], V)
        x_half = x[i] + h / 2
        y[i + 1] = y[i] + h * f(x_half, y_half, V)
```

```
    return x, y

def exact_solution(x, V=5):
    return V * x**2

# Параметры
x0 = 1
V = y0 = 5
h = 0.001 # шаг
n = 10

# Вычисление решений
x_euler, y_euler = euler_method(x0, y0, h, n, V)
x_improved, y_improved = improved_euler_method(x0, y0, h, n, V)
y_exact = exact_solution(x_euler)

# Вычисление погрешностей
error_euler = np.abs(y_euler - y_exact)
error_improved = np.abs(y_improved - y_exact)

print("\nМетод Эйлера: ")
print("-" * 130)
print("x:      ", " ".join(f"{x:>10.7f}" for x in x_euler))
print("y_M:    ", " ".join(f"{y:>10.7f}" for y in y_euler))
print("y_T:    ", " ".join(f"{y:>10.7f}" for y in y_exact))
print("Погрешн:", " ".join(f"{e:>10.7f}" for e in error_euler))
print("-" * 130)

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

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
```

Задание 9. Решить краевую задачу разностным методом.

```
V = 5 # номер варанта

def derivative(x):
    return -(4 * V * x**4 - 3 * V**2 * x**3 + 6 * V * x - 2 * V**2)
def Y(x):
```

```
    return V * x**2 * (x - V)
def p(x):
    return -x**2
def q(x):
    return -x

def main():
    n = 10
    x0 = 0
    h = V / n
    x = [x0 + i * h for i in range(n + 1)]
    exact = [Y(xi) for xi in x]

    f = [0.0] * (n + 1)
    s = [0.0] * (n + 1)
    t = [0.0] * (n + 1)
    r = [0.0] * (n + 1)
    f1 = [0.0] * (n + 1)
    s1 = [0.0] * (n + 1)
    y = [0.0] * (n + 1)
    e = [0.0] * (n + 1)

    for i in range(1, n):
        f[i] = 0.5 * (1 + 0.5 * h * p(x[i]))
        s[i] = 0.5 * (1 - 0.5 * h * p(x[i]))
        t[i] = 1 + 0.5 * h**2 * q(x[i])
        r[i] = 0.5 * h**2 * derivative(x[i])

    f1[1] = 0.0
    s1[1] = 0.0

    for j in range(1, n):
        denom = t[j] - f[j] * f1[j]
        f1[j + 1] = s[j] / denom
        s1[j + 1] = (r[j] + f[j] * s1[j]) / denom

    # Граничное условие y[n] = 0 в оригинале неявно, принимаем y[n]=0
    y[n] = 0.0
    for j in range(n - 1, 0, -1):
        y[j] = f1[j + 1] * y[j + 1] + s1[j + 1]

    max_e = 0.0
    max_e_index = 0
    for i in range(n + 1):
        e[i] = abs(y[i] - exact[i])
        if (e[i] > max_e):
            max_e = e[i]
            max_e_index = i

    print("x\t y\t\t exact\t e")
    for i in range(n + 1):
        print(f"{x[i]:.2f}\t {y[i]:.2f} \t{exact[i]:.2f}\t {e[i]:.8f}")
    print("Максимальный e: ", max_e, "Номер максимального e: ", max_e_index)

if __name__ == "__main__":
    main()
```

```
x        y               exact   e
0.00     0.00   -0.00    0.00000000
0.50     -0.09  -5.62    5.53017248
1.00     -10.31         -20.00   9.68559849
1.50     -29.38         -39.38   9.99173440
2.00     -52.77         -60.00   7.22943620
2.50     -73.33         -78.12   4.79707715
3.00     -86.60         -90.00   3.39879569
3.50     -89.59         -91.88   2.28687417
4.00     -78.58         -80.00   1.41652296
4.50     -49.97         -50.62   0.65542909
5.00     0.00   0.00     0.00000000
Максимальный e:  9.991734404070492 Номер максимального e:  3
```