Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра информатики

**Лабораторная работа №6**

**«**Интерполяционные многочлены. Интерполяция сплайнами**»**по учебной дисциплине «Методы численного анализа»

**Выполнили:**

студент гр. 153504 Климкович Н. В. студент гр. 153504 Тиханёнок И. А.  
студент гр. 153504 Тарасенко Ф. П.

**Проверила:**

ст. преподаватель   
кафедры информатики   
Стройникова Е. Д.

Минск 2022

**Цель работы:**

Изучить интерполяцию функций с помощью интерполяционных многочленов Лагранжа и Ньютона. Изучить и реализовать построение кубических интерполяционных сплайнов.

**Краткие теоретические сведения:**

# **Интерполяционный многочлен**

Пусть – функция, непрерывная на отрезке .

Выберем на этом отрезке точки, называемые *узлами интерполяции*:

.

Предположим, что известны значения функции в узлах интерполяции:

.

Ставится задача найти многочлен такой, что

Такой многочлен называется *интерполяционным многочленом*, а задача его нахождения – *задачей интерполяции*.

Можно показать, что задача всегда имеет решение, причем единственное.

Обозначим

.

Пусть . Тогда погрешность интерполяции оценивается по формуле:

# **Интерполяционный многочлен Лагранжа**

Пусть

т.е. .

Положим ,

т.е. .

Очевидно,

Построим многочлен .

Легко видеть, что , т.е. это интерполяционный многочлен. Его называют *интерполяционным многочленом Лагранжа*.

# **Интерполяционный многочлен Ньютона**

Пусть – набор узлов интерполирования, – значения функции в узлах.

Величину называют конечной разностью первого порядка в *k*-м узле.

Аналогично определяются конечные разности высших порядков.

Конечные разности обычно считают по схеме:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Разделенной разностью первого порядка называется выражение

Разделенной разностью второго порядка называется выражение

Пусть – любая точка отрезка, не совпадающая с узлами. Тогда

,

откуда

Далее ,

откуда .

Подставляя в (3), получаем

Далее ,

Откуда .

Подставляя в (4), имеем:

Продолжая процесс, получим:

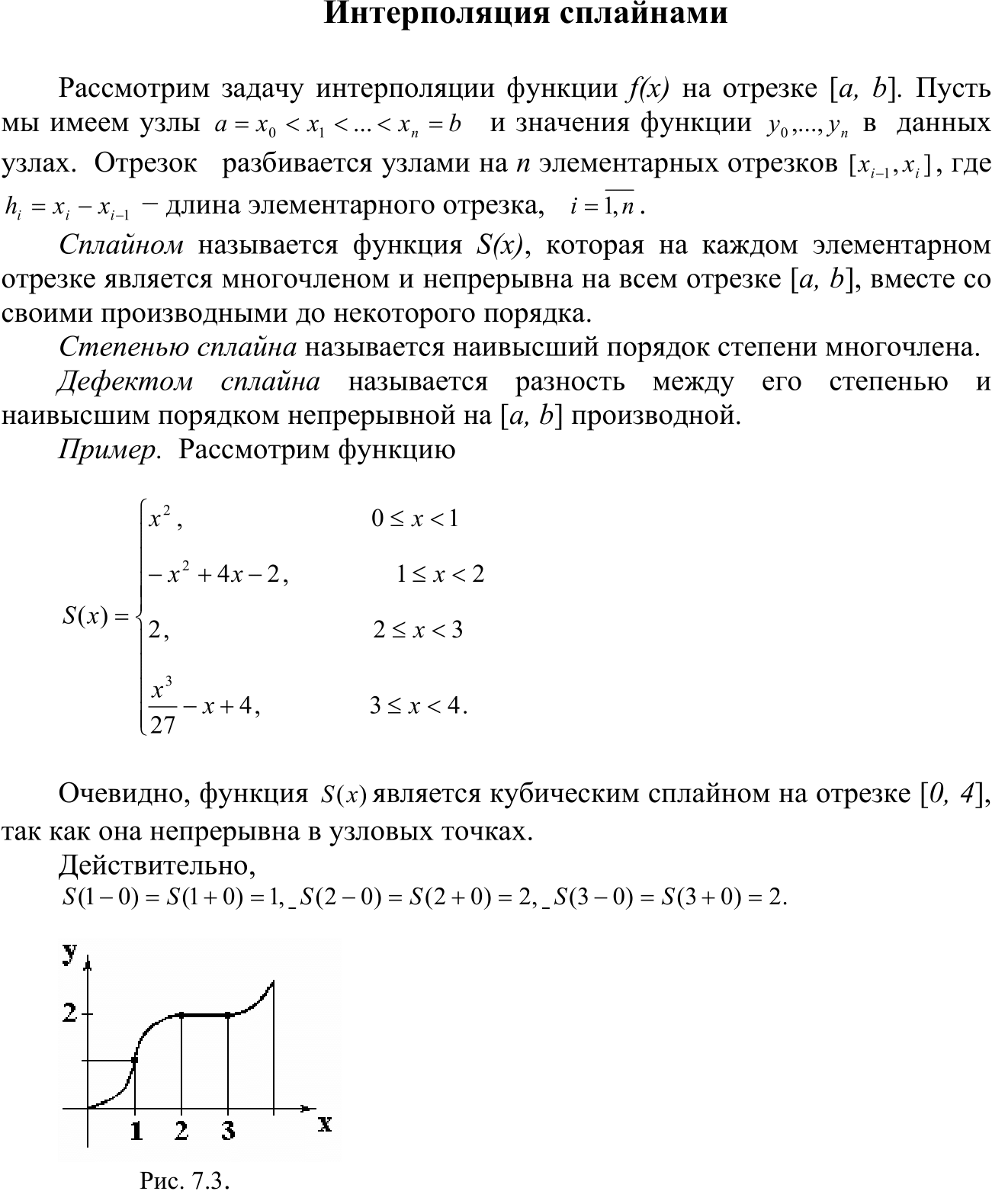
,

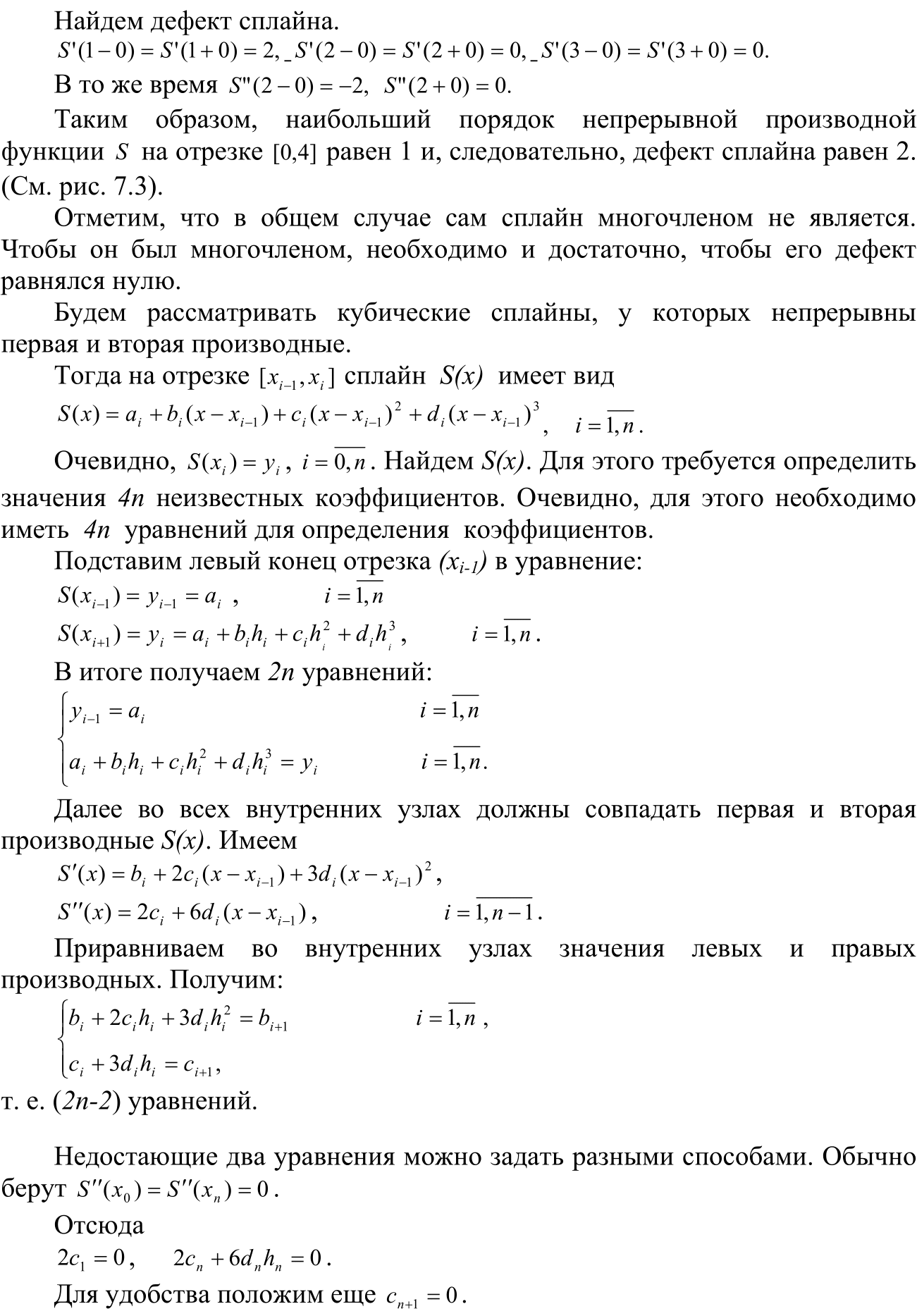
где .

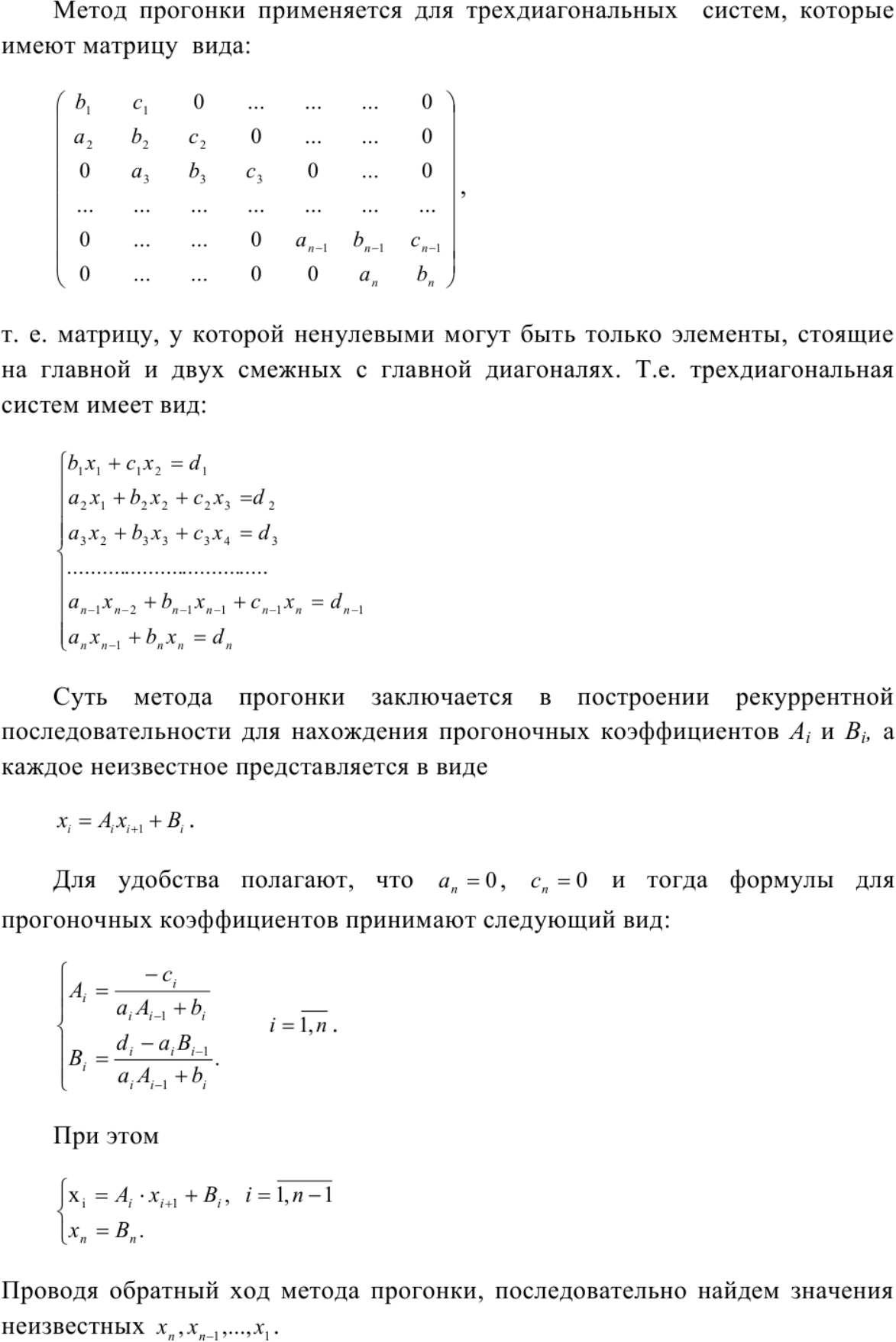
Очевидно, при , т.е. – интерполяционный многочлен. Его называют интерполяционным многочленом Ньютона.

Достоинство интерполяционного многочлена Ньютона: он удобен при расширении интерполяции и добавлении узлов.

Недостаток: в какой-то степени он сложнее в подсчете конечных разностей по сравнению с многочленом Лагранжа.







**Программная реализация:**

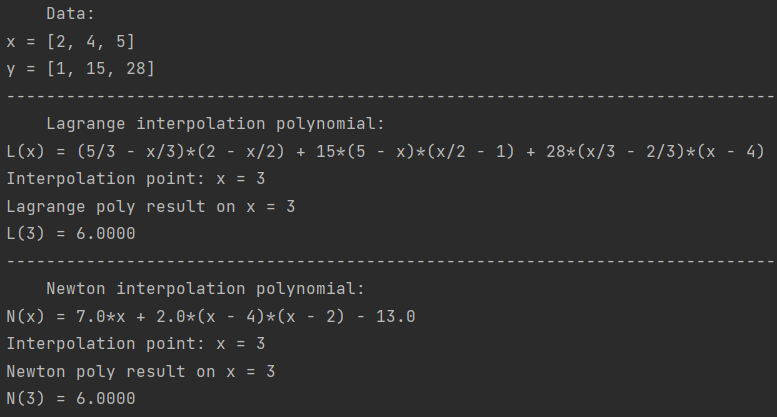
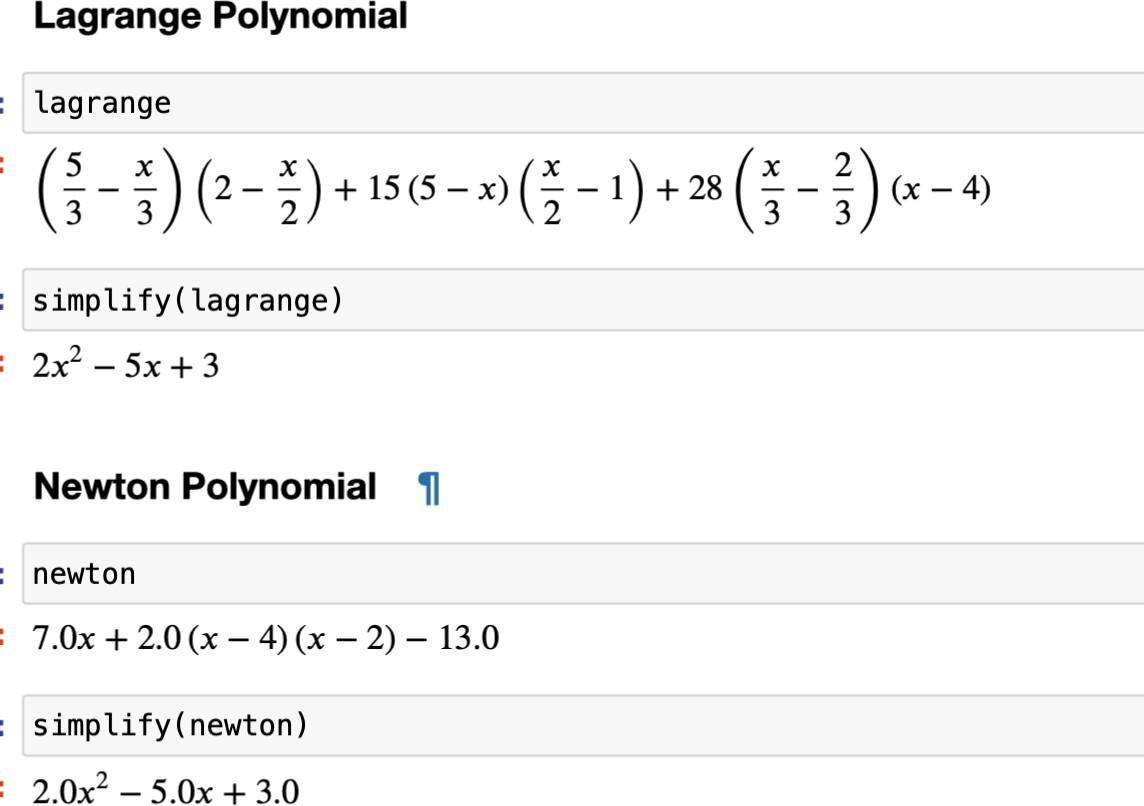
def calculate(nodes, spline: list, x: float) -> float:  
 for i in range(1, len(nodes)):  
 if nodes[i - 1][0] <= x <= nodes[i][0]:  
 value = nodes[i][1]  
 h = nodes[i][0] - nodes[i - 1][0]  
 b = (nodes[i][1] - nodes[i - 1][1]) / h \  
 + (2 \* (0.0 if i == len(nodes) - 1 else spline[i][1])) + spline[i - 1][1] \* h / 3  
 value += b \* (x - nodes[i][0])  
 value += (0.0 if i == len(nodes) - 1 else spline[i][1]) \* ((x - nodes[i][0]) \*\* 2)  
 d = ((0.0 if i == len(nodes) - 1 else spline[i][1]) - spline[i - 1][1]) / (3 \* h)  
 value += d \* ((x - nodes[i][0]) \*\* 3)  
 return value  
  
  
def main():  
 interval = (-1, 4)  
 n = 10  
  
 nodes = []  
 for i in range(n):  
 x = interval[0] + (interval[1] - interval[0]) / (n - 1) \* i  
 y = np.exp(x)  
 nodes.append((x, y))  
 nodes = np.array(nodes)  
  
 n = len(nodes) - 1  
 matrix = [[  
 0.0,  
 2 / 3 \* ((nodes[1][0] - nodes[0][0]) + (nodes[2][0] - nodes[1][0])),  
 (nodes[2][0] - nodes[1][0]) / 3  
 ]]  
  
 vector = [(nodes[2][1] - nodes[1][1]) / (nodes[2][0] - nodes[1][0])  
 - (nodes[1][1] - nodes[0][1]) / (nodes[1][0] - nodes[0][0])]  
  
 for i in range(2, n - 1):  
 h\_i = nodes[i][0] - nodes[i - 1][0]  
 h\_i\_1 = nodes[i + 1][0] - nodes[i][0]  
 a = h\_i / 3  
 b = 2 / 3 \* (h\_i + h\_i\_1)  
 c = h\_i\_1 / 3  
  
 matrix.append([a, b, c])  
 d = (nodes[i + 1][1] - nodes[i][1]) / h\_i\_1 - (nodes[i][1] - nodes[i - 1][1]) / h\_i  
 vector.append(d)  
  
 matrix.append([  
 (nodes[n - 1][0] - nodes[n - 2][0]) / 3,  
 2 / 3 \* ((nodes[n - 1][0] - nodes[n - 2][0]) + (nodes[n][0] - nodes[n - 1][0])),  
 0.0  
 ])  
  
 vector.append((nodes[n][1] - nodes[n - 1][1]) / (nodes[n][0] - nodes[n - 1][0])  
 - (nodes[n - 1][1] - nodes[n - 2][1]) / (nodes[n - 1][0] - nodes[n - 2][0]))  
 matrix = np.array(matrix)  
 vector = np.array(vector)  
  
 matrix = [[matrix[i][j] \* 3 for j in range(len(matrix[i]))] for i in range(len(matrix))]  
 vector = [vector[i] \* 3 for i in range(len(vector))]  
 a = [0.0]  
 b = [0.0]  
 for i in range(len(matrix)):  
 a.append(-matrix[i][2] / (matrix[i][0] \* a[i] + matrix[i][1]))  
 b.append((vector[i] - matrix[i][0] \* b[i]) / (matrix[i][0] \* a[i] + matrix[i][1]))  
 sol = [0.0 for \_ in range(len(matrix))]  
 sol[-1] = b[-1]  
  
 for i in range(len(matrix) - 2, -1, -1):  
 sol[i] = a[i + 1] \* sol[i + 1] + b[i + 1]  
  
 sys\_sol = np.array(sol)  
  
 c = [0.0] + list(sys\_sol) + [0.0]  
 spline\_coefficients = []  
 for i in range(len(c) - 1):  
 a = nodes[i][1]  
 h = nodes[i + 1][0] - nodes[i][0]  
 b = (nodes[i + 1][1] - nodes[i][1]) / h - (c[i + 1] + 2 \* c[i]) \* h / 3  
 d = (c[i + 1] - c[i]) / (3 \* h)  
 spline\_coefficients.append([a, b, c[i], d])  
  
 spline = [i[::-1] for i in spline\_coefficients]  
 for i in range(len(spline)):  
 print(f"({nodes[i][0]:.1f}, {nodes[i + 1][0]:.1f}):")  
 print(np.poly1d(spline[i]))  
 print()  
  
 x = (interval[1] - interval[0]) / 2  
 print(f"Значение в точке {x}: {0.784413:.6f}")  
 print(f"Разность значений в точке: {abs(np.arctan(x) - 0.784413)}")  
  
 points\_number = 100  
 for i in range(len(spline)):  
 points\_x = np.linspace(nodes[i][0], nodes[i + 1][0], points\_number)  
 points\_y = [calculate(nodes, spline, pnt) for pnt in points\_x]  
 if i == 0:  
 plt.plot(points\_x, points\_y, color='blue', label="spline estimation")  
 continue  
  
 plt.plot(points\_x, points\_y, color='blue')  
  
 points\_x = np.linspace(min(node[0] for node in nodes), max(node[0] for node in nodes), 1000)  
  
 points\_y = [np.arctan(pnt) for pnt in points\_x]  
 plt.plot(points\_x, points\_y, color='red', label="real function")  
 plt.plot([nodes[i][0] for i in range(len(nodes))], [nodes[i][1] for i in range(len(nodes))], 'go', label="nodes")  
 plt.legend()  
 plt.show()

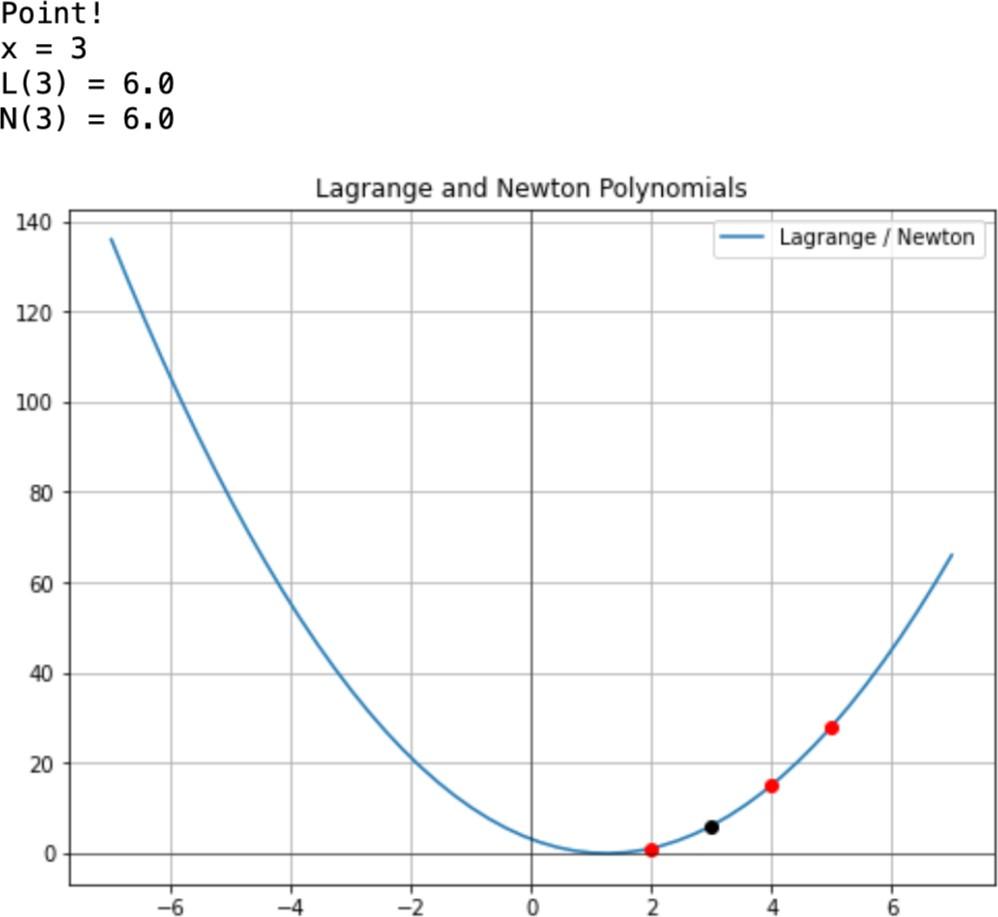
def lagrange\_l(x\_ls: list, x\_j):  
 x = symbols('x')  
 lg = 1  
 n = len(x\_ls)  
 for i in range(n):  
 if x\_j == x\_ls[i]:  
 continue  
 lg \*= (x - x\_ls[i]) / (x\_j - x\_ls[i])  
 return lg  
  
  
def lagrange\_poly(x, y):  
 if len(x) != len(y):  
 raise ValueError("x and y are not same length")  
 n = len(x)  
  
 L = 0  
 for i in range(n):  
 L += y[i] \* lagrange\_l(x, x[i])  
 return L

def get\_a(x: list, y: list):  
 a\_table = np.zeros(shape=(len(y) - 1, len(y) - 1))  
 n = a\_table.shape[0]  
  
 for i in range(n):  
 a\_table[0, i] = (y[i + 1] - y[i]) / (x[i+1] - x[i])  
  
 for i in range(1, n):  
 for j in range(n - i):  
 a\_table[i, j] = (a\_table[i - 1, j + 1] - a\_table[i - 1, j]) / (x[i+1+j] - x[j])  
  
 return a\_table  
  
  
def newton\_poly(x\_ls: list, y\_ls: list):  
 if len(x\_ls) != len(y\_ls):  
 raise TypeError("x and y must be same size")  
  
 n = len(x\_ls)  
  
 x = symbols('x')  
 N = 0  
 a\_table = get\_a(x\_ls, y\_ls)  
 for i in range(n):  
 if i == 0:  
 N += y\_ls[0]  
 continue  
 x\_poly = 1  
 for j in range(i):  
 x\_poly \*= (x-x\_ls[j])  
  
 N += a\_table[i-1, 0] \* x\_poly  
  
 return N

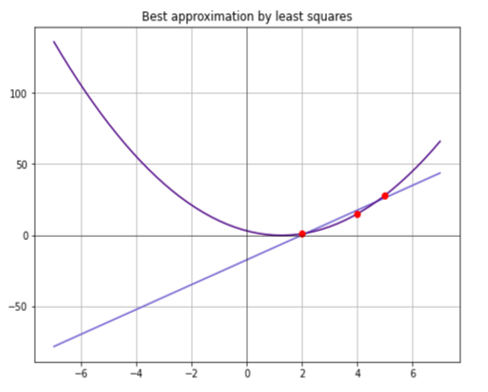
def sum\_of\_pow(x, pow):  
 n = x.shape[0]  
 sum = 0  
 for i in range(n):  
 sum += np.power(x[i], pow)  
 return sum  
  
  
def sum\_x\_y\_pows(x, y, pow):  
 n = x.shape[0]  
 sum = 0  
  
 for i in range(n):  
 sum += y[i] \* np.power(x[i], pow)  
 return sum  
  
  
def min\_square(x, y, m):  
 if y.shape[0] < m:  
 raise Exception("Low size")  
  
 if x.shape[0] != y.shape[0]:  
 raise Exception("size error")  
 n = x.shape[0]  
  
 A = np.zeros(shape=(m+1, m+1))  
 for i in range(m+1):  
 for j in range(m+1):  
 A[i, j] = sum\_of\_pow(x, i+j)  
  
 b = np.zeros(shape=m + 1)  
 for i in range(m+1):  
 b[i] = sum\_x\_y\_pows(x, y, i)  
  
 a\_vec = np.linalg.solve(A, b)  
 return a\_vec

**Тестовый пример 1.**

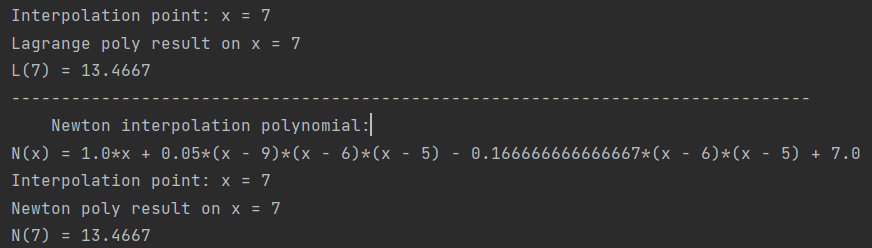
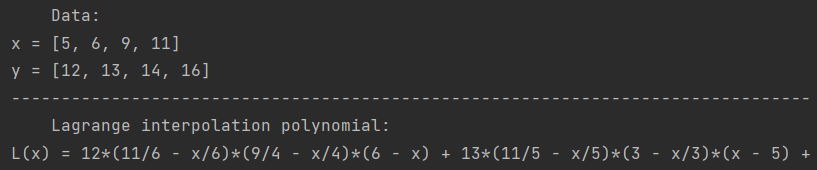
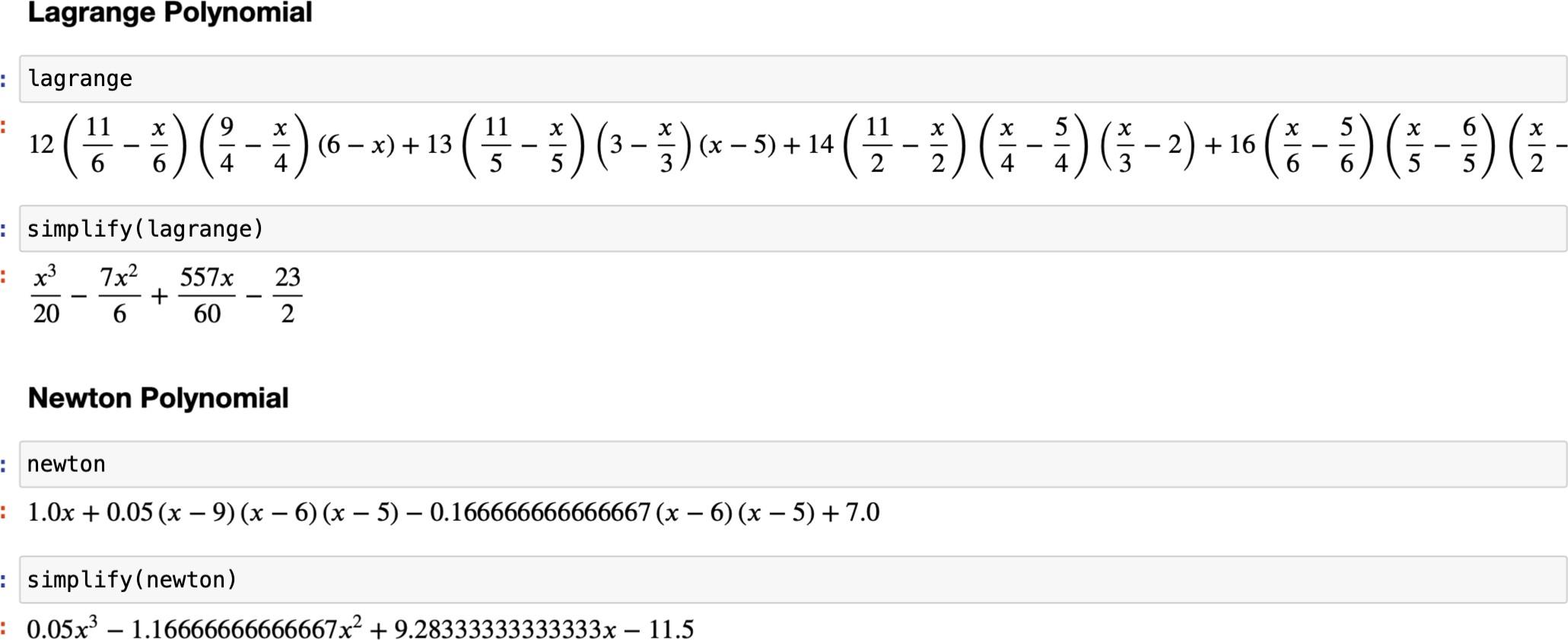


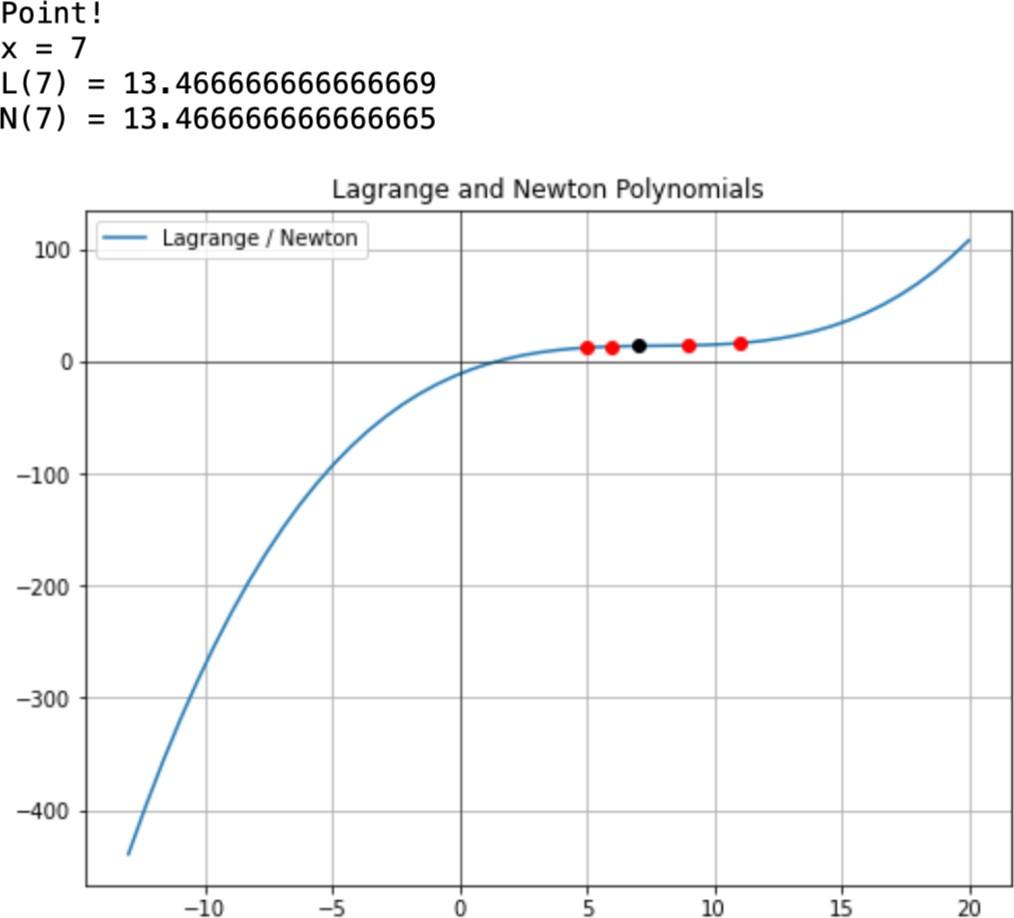
**Значение в точке**

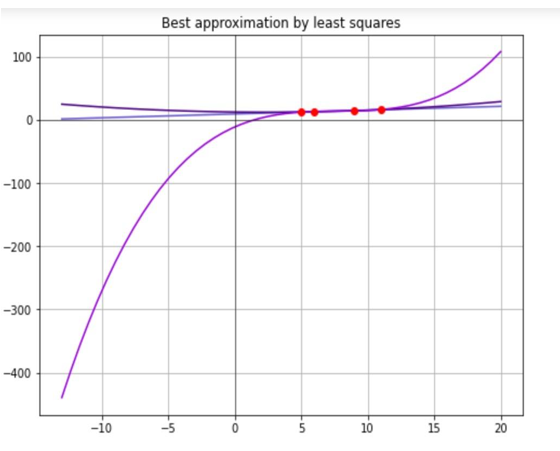
**Графики функций наилучшего приближения:**

****

**Тестовый пример 2.**

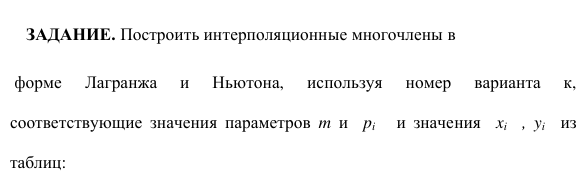


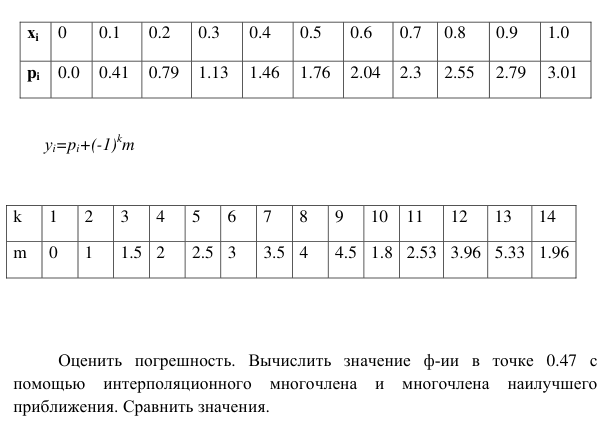
**Найдем значение в точке**

**Графики функций наилучшего приближения:**

**Решение задания №1**

**Вариант 5**

****



Data:

x = [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]

y = [-2.5 -2.09 -1.71 -1.37 -1.04 -0.74 -0.46 -0.2 0.05 0.29 0.51]

--------------------------------------------------------------------------------

Lagrange interpolation polynomial:

L(x) = -20.9\*x\*(1.11111111111111 - 1.11111111111111\*x)\*(1.125 - 1.25\*x)\*(1.14285714285714 - 1.42857142857143\*x)\*(1.16666666666667 - 1.66666666666667\*x)\*(1.2 - 2.0\*x)\*(1.25 - 2.5\*x)\*(1.33333333333333 - 3.33333333333333\*x)\*(1.5 - 5.0\*x)\*(2.0 - 10.0\*x) - 8.55\*x\*(1.25 - 1.25\*x)\*(1.28571428571429 - 1.42857142857143\*x)\*(1.33333333333333 - 1.66666666666667\*x)\*(1.4 - 2.0\*x)\*(1.5 - 2.5\*x)\*(1.66666666666667 - 3.33333333333333\*x)\*(2.0 - 5.0\*x)\*(3.0 - 10.0\*x)\*(10.0\*x - 1.0) - 4.56666666666667\*x\*(1.42857142857143 - 1.42857142857143\*x)\*(1.5 - 1.66666666666667\*x)\*(1.6 - 2.0\*x)\*(1.75 - 2.5\*x)\*(2.0 - 3.33333333333333\*x)\*(2.5 - 5.0\*x)\*(4.0 - 10.0\*x)\*(5.0\*x - 0.5)\*(10.0\*x - 2.0) - 2.6\*x\*(1.66666666666667 - 1.66666666666667\*x)\*(1.8 - 2.0\*x)\*(2.0 - 2.5\*x)\*(2.33333333333333 - 3.33333333333333\*x)\*(3.0 - 5.0\*x)\*(5.0 - 10.0\*x)\*(3.33333333333333\*x - 0.333333333333333)\*(5.0\*x - 1.0)\*(10.0\*x - 3.0) - 1.48\*x\*(2.0 - 2.0\*x)\*(2.25 - 2.5\*x)\*(2.66666666666667 - 3.33333333333333\*x)\*(3.5 - 5.0\*x)\*(6.0 - 10.0\*x)\*(2.5\*x - 0.25)\*(3.33333333333333\*x - 0.666666666666667)\*(5.0\*x - 1.5)\*(10.0\*x - 4.0) - 0.766666666666667\*x\*(2.5 - 2.5\*x)\*(3.0 - 3.33333333333333\*x)\*(4.0 - 5.0\*x)\*(7.0 - 10.0\*x)\*(2.0\*x - 0.2)\*(2.5\*x - 0.5)\*(3.33333333333333\*x - 1.0)\*(5.0\*x - 2.0)\*(10.0\*x - 5.0) - 0.285714285714286\*x\*(3.33333333333333 - 3.33333333333333\*x)\*(4.5 - 5.0\*x)\*(7.99999999999999 - 9.99999999999999\*x)\*(1.66666666666667\*x - 0.166666666666667)\*(2.0\*x - 0.4)\*(2.5\*x - 0.75)\*(3.33333333333333\*x - 1.33333333333333)\*(5.0\*x - 2.5)\*(10.0\*x - 6.0) + 0.0624999999999998\*x\*(5.0 - 5.0\*x)\*(9.0 - 10.0\*x)\*(1.42857142857143\*x - 0.142857142857143)\*(1.66666666666667\*x - 0.333333333333333)\*(2.0\*x - 0.6)\*(2.5\*x - 1.0)\*(3.33333333333333\*x - 1.66666666666667)\*(5.0\*x - 3.0)\*(9.99999999999999\*x - 6.99999999999999) + 0.322222222222222\*x\*(10.0 - 10.0\*x)\*(1.25\*x - 0.125)\*(1.42857142857143\*x - 0.285714285714286)\*(1.66666666666667\*x - 0.5)\*(2.0\*x - 0.8)\*(2.5\*x - 1.25)\*(3.33333333333333\*x - 2.0)\*(5.0\*x - 3.5)\*(10.0\*x - 8.0) + 0.51\*x\*(1.11111111111111\*x - 0.111111111111111)\*(1.25\*x - 0.25)\*(1.42857142857143\*x - 0.428571428571429)\*(1.66666666666667\*x - 0.666666666666667)\*(2.0\*x - 1.0)\*(2.5\*x - 1.5)\*(3.33333333333333\*x - 2.33333333333333)\*(5.0\*x - 4.0)\*(10.0\*x - 9.0) - 2.5\*(1.0 - 10.0\*x)\*(1.0 - 5.0\*x)\*(1.0 - 3.33333333333333\*x)\*(1.0 - 2.5\*x)\*(1.0 - 2.0\*x)\*(1.0 - 1.66666666666667\*x)\*(1.0 - 1.42857142857143\*x)\*(1.0 - 1.25\*x)\*(1.0 - 1.11111111111111\*x)\*(1.0 - 1.0\*x)

Interpolation point: x = 0.47

Lagrange poly result on x = 0.47

L(0.47) = -0.8273

--------------------------------------------------------------------------------

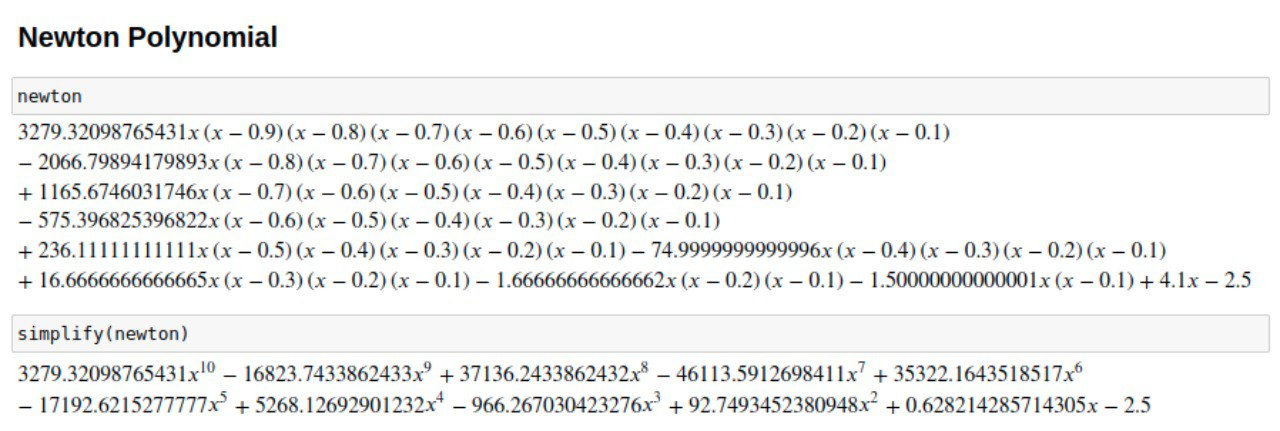
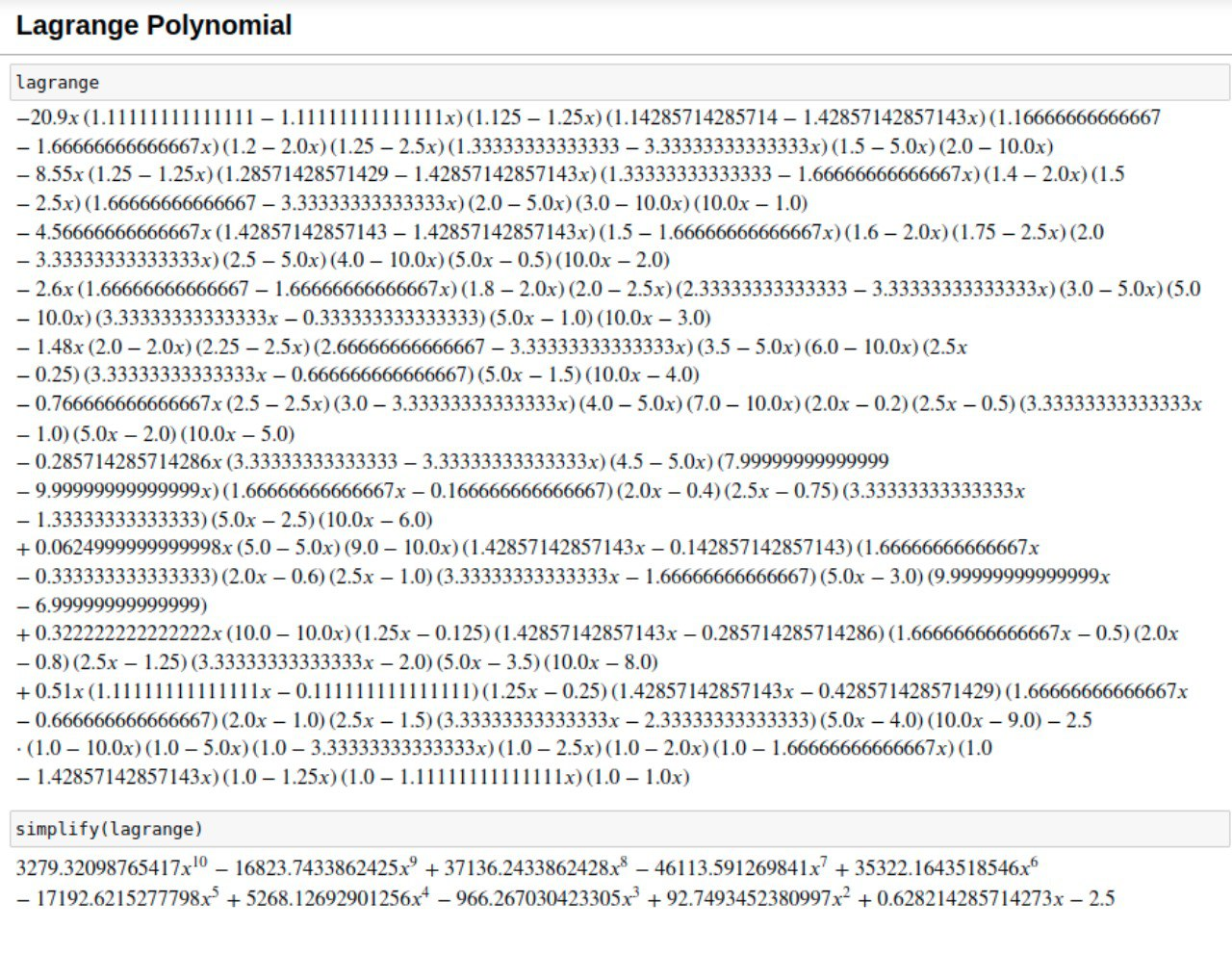
Newton interpolation polynomial:

N(x) = 3279.32098765431\*x\*(x - 0.9)\*(x - 0.8)\*(x - 0.7)\*(x - 0.6)\*(x - 0.5)\*(x - 0.4)\*(x - 0.3)\*(x - 0.2)\*(x - 0.1) - 2066.79894179893\*x\*(x - 0.8)\*(x - 0.7)\*(x - 0.6)\*(x - 0.5)\*(x - 0.4)\*(x - 0.3)\*(x - 0.2)\*(x - 0.1) + 1165.6746031746\*x\*(x - 0.7)\*(x - 0.6)\*(x - 0.5)\*(x - 0.4)\*(x - 0.3)\*(x - 0.2)\*(x - 0.1) - 575.396825396822\*x\*(x - 0.6)\*(x - 0.5)\*(x - 0.4)\*(x - 0.3)\*(x - 0.2)\*(x - 0.1) + 236.11111111111\*x\*(x - 0.5)\*(x - 0.4)\*(x - 0.3)\*(x - 0.2)\*(x - 0.1) - 74.9999999999996\*x\*(x - 0.4)\*(x - 0.3)\*(x - 0.2)\*(x - 0.1) + 16.6666666666665\*x\*(x - 0.3)\*(x - 0.2)\*(x - 0.1) - 1.66666666666662\*x\*(x - 0.2)\*(x - 0.1) - 1.50000000000001\*x\*(x - 0.1) + 4.1\*x - 2.5

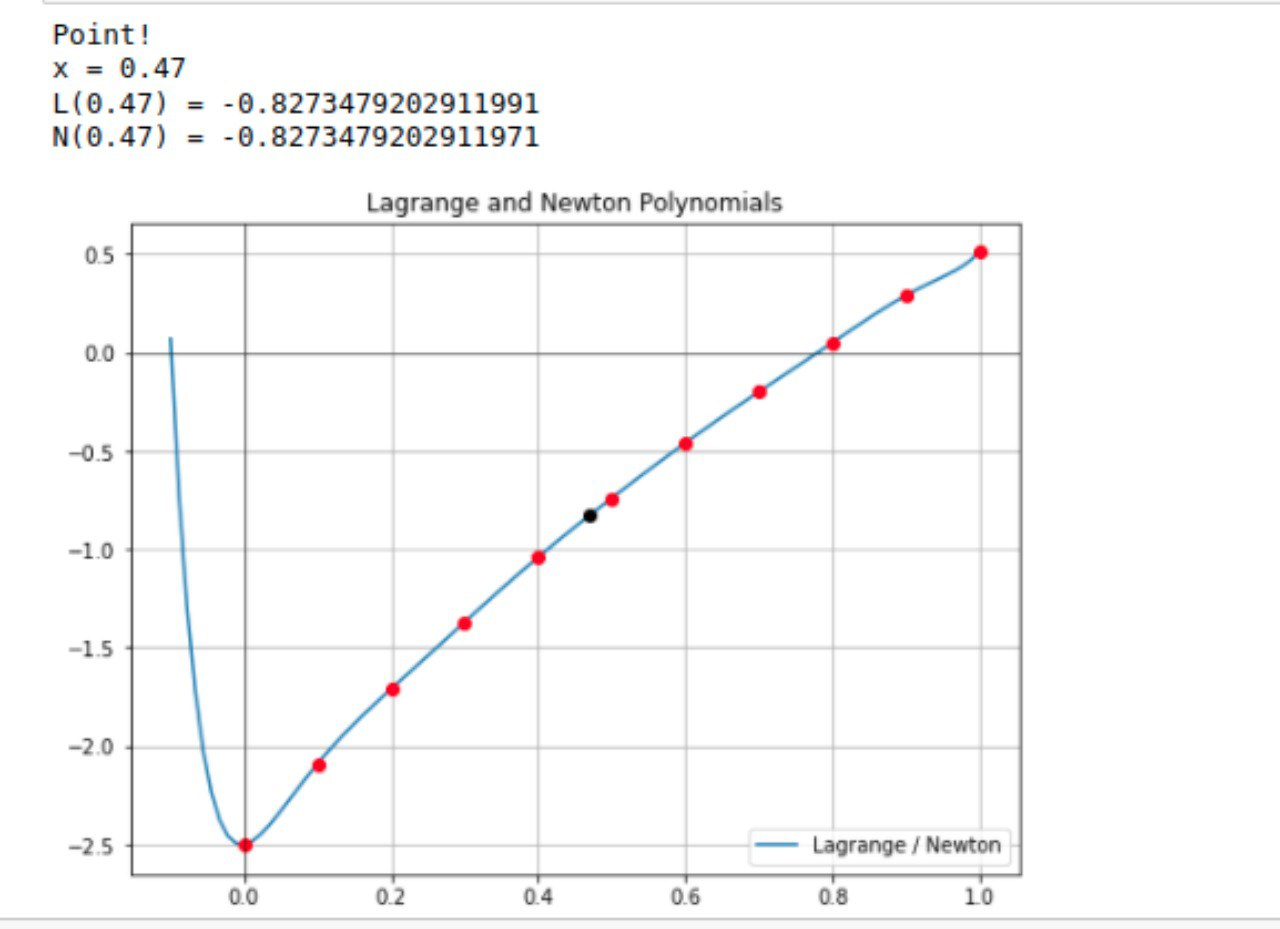
Interpolation point: x = 0.47

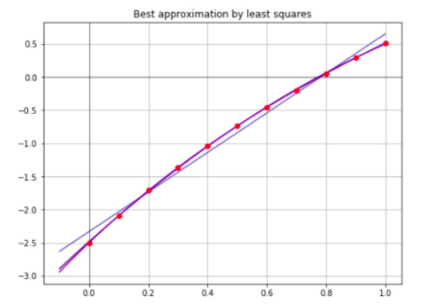
Newton poly result on x = 0.47

N(0.47) = -0.8273



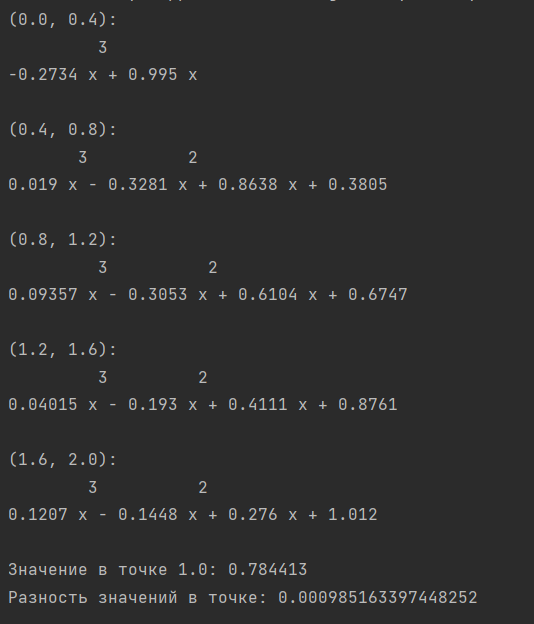
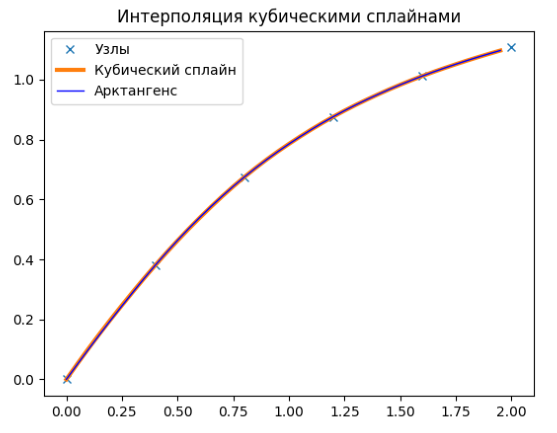
**Найдем приближение в точке**



**Многочлены меньшей степени наилучшего приближения**

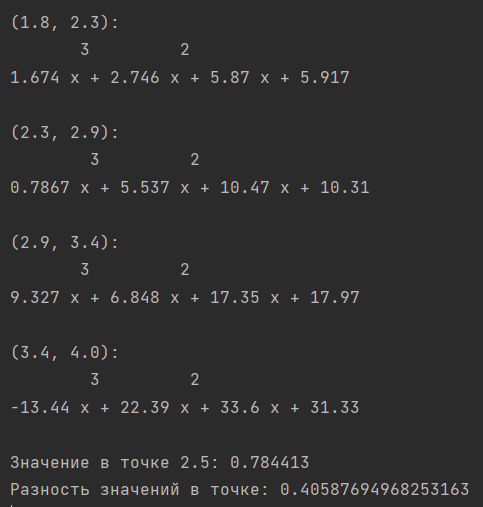
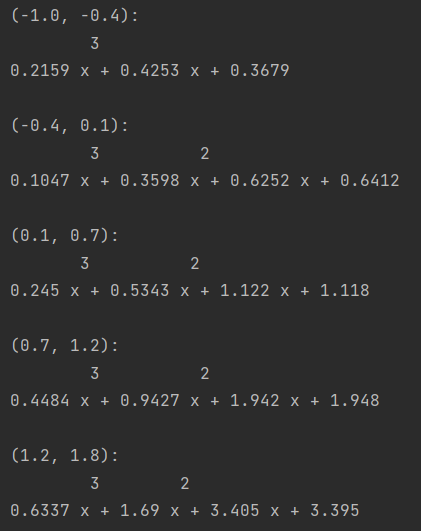
**Решение задания №2**

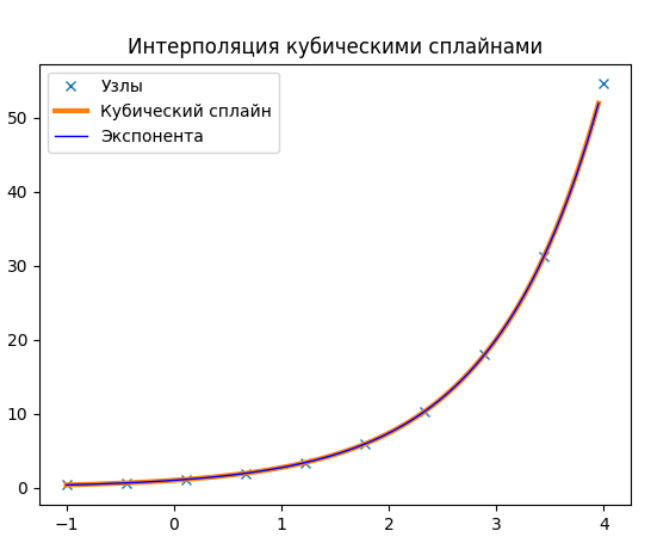
Функция: y = arctg(x)  
Интервал: [0, 2]  
Число узлов: 6

****

**Тестовый пример 3**

Функция: y = exp(x)  
Интервал: [-1, 4]  
Число узлов: 10



****

**Вывод:**

Таким образом, в ходе выполнения лабораторной работы была освоена интерполяция функций с помощью интерполяционных многочленов Лагранжа и Ньютона. Составлена компьютерная программа, на тестовых примерах проверена правильность её работы, построены интерполяционные многочлены в форме Лагранжа и Ньютона, построен многочлен наилучшего приближения по методу наименьших квадратов, вычислено значение функции в точке согласно заданному варианту. Также был изучен метод построения кубического интерполяционного сплайна, составлена программа для вычисления сплайна для заданной функции, вычислено приближенное значение функции в заданной точке, произведена оценка полученных результатов.