МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«__»__________ 2025 г.

Проверено:

«__»__________2025 г.

Вариант 18

Отчет по лабораторной работе № 3

по дисциплине

«Вычислительная математика»

Разработал студент гр. ИВТб 2302-05-00 ________________ /Соловьев А.С./

(Подпись)

Проверил заведующий кафедры ЭВМ ________________ /Старостин П.А./

(Подпись)

Работа защищена «__»____________2025 г.

Киров 2025

1 Задание

1. По таблице с неравноотстоящими значениями аргумента выполнить интерполяцию, используя формулу Лангранджа. Точность $E <= 10^{-6}$ Для X=0,692

x	$f(x)$
0.62	0.537944
0.67	0.511709
0.74	0.477114
0.80	0.449329
0.87	0.418952
0.96	0.382893
0.99	0.371577

Таблица 1: Таблица значений функции

2 Теорическая часть.Описание методов решения уравнений

Суть метода Лагранжа

Метод интерполяции полиномом Лагранжа позволяет найти многочлен степени n, который проходит через заданные точки $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$.

Формула интерполяционного полинома

Интерполяционный полином Лагранжа имеет вид:

$$L_n(x) = \sum_{i=0}^{n} y_i \cdot l_i(x),$$

где базисные полиномы $l_i(x)$ определяются следующим образом:

$$l_i(x) = \prod_{j=0 j\neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

Каждый базисный полином $l_i(x)$ принимает значение 1 в точке x_i и 0 во всех остальных узлах x_j $(j \neq i)$.

Алгоритм вычисления

1. Заданы $n + 1$ узлов интерполяции (x_i, y_i). 2. Для каждого x_i вычисляются базисные полиномы $l_i(x)$. 3. Итоговый полином получается как сумма произведений значений функции y_i на соответствующие базисные полиномы. 4. При необходимости можно вычислить значение $L_n(x)$ в любой точке x.

2.1 Метод с формулами Ньютона

Интерполяционные формулы Ньютона используются для нахождения значения функции в произвольной точке на основе известных дискретных значений.

2.2 Первая интерполяционная формула Ньютона

Первая интерполяционная формула Ньютона применяется в случае равномерно распределенных узлов интерполяции и записывается в форме:

$$P_n(x) = f(x_0) + \sum_{k=1}^{n} \Delta^k f(x_0) \frac{(x - x_0)(x - x_1) \dots (x - x_{k-1})}{k! h^k}, \tag{1}$$

где $\Delta^k f(x_0)$ — разделенные разности, а h — шаг интерполяции.

2.3 Вторая интерполяционная формула Ньютона

Вторая интерполяционная формула Ньютона используется, если значение функции известно в точках, расположенных ближе к концу интервала. Она записывается в виде:

$$P_n(x) = f(x_n) + \sum_{k=1}^{n} \nabla^k f(x_n) \frac{(x - x_n)(x - x_{n-1}) \dots (x - x_{n-k+1})}{k! h^k}, \tag{2}$$

где $\nabla^k f(x_n)$ — конечные разности, вычисленные с конца интервала.

2.4 Разделенные разности

Разделенные разности определяются рекурсивно:

$$f[x_i] = f(x_i), \tag{3}$$

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}, \tag{4}$$

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}, \tag{5}$$

и так далее.

3 Оценка погрешности

Ошибка интерполяции определяется формулой:

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i), \quad \xi \in [x_0, x_n]. \tag{6}$$

Здесь $f^{(n+1)}(\xi)$ — производная порядка $n+1$ в некоторой точке ξ.

4 Выбор вида эмпирической зависимости

Перед применением МНК необходимо выбрать тип функции $y = f(x)$, которая наилучшим образом описывает экспериментальные данные. Для этого можно использовать следующие подходы:

4.1 Анализ средних значений

Вычисляем средние арифметические, геометрические и гармонические для крайних точек данных:

$$x = \frac{x_1 + x_n}{2}, \quad y = \frac{y_1 + y_n}{2},$$

$$x = \sqrt{x_1 \cdot x_n}, \quad y = \sqrt{y_1 \cdot y_n},$$

$$x = \frac{2x_1 x_n}{x_1 + x_n}, \quad y = \frac{2y_1 y_n}{y_1 + y_n}.$$

4.2 Определение минимальной ошибки

Для каждой из средних точек (x, y), (x, y), (x, y) находим ближайшую экспериментальную точку и вычисляем ошибки:

$$= \sqrt{(x_i - x)^2 + (y_i - y)^2}.$$

Минимальная ошибка указывает на наиболее подходящий тип зависимости (линейная, степенная, логарифмическая и т.д.).

5 Метод наименьших квадратов для линейной зависимости

Если анализ показывает, что линейная зависимость $y = ax + b$ наиболее адекватна, параметры a и b находятся из условия минимизации суммы квадратов отклонений:

$$S(a, b) = \sum_{i=1}^{n} (y_i - (ax_i + b))^2 \to \min .$$

5.1 Нормальные уравнения

Частные производные $\frac{\partial S}{\partial a}$ и $\frac{\partial S}{\partial b}$ приводят к системе:

$$\{ \, a \sum x_i^2 + b \sum x_i = \sum x_i y_i, a \sum x_i + bn = \sum y_i.$$

5.2 Решение системы

Коэффициенты вычисляются по формулам:

$$a = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}, \quad b = \frac{\sum y_i - a\sum x_i}{n}.$$

6 Оценка точности аппроксимации

Среднеквадратичное отклонение:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - (ax_i + b))^2}.$$

7 Практическая часть

Исходные данные

Заданы узлы интерполяции:

x	$f(x)$
0.62	0.537944
0.67	0.511709
0.74	0.477114
0.80	0.449329
0.87	0.418952
0.96	0.382893
0.99	0.371577

Требуется найти значение функции в точке $x = 0.692$ с использованием полинома Лагранжа.

Алгоритм решения

Метод Лагранжа заключается в представлении интерполяционного многочлена в виде:

$$L_n(x) = \sum_{i=0}^{n} y_i \cdot l_i(x),$$

где базисные полиномы $l_i(x)$ определяются по формуле:

$$l_i(x) = \prod_{j=0 j\neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

Линейная интерполяция (L_1)

Используется два узла (x_0, y_0) и (x_1, y_1):

$$L_1(x) = y_0 \cdot \frac{x - x_1}{x_0 - x_1} + y_1 \cdot \frac{x - x_0}{x_1 - x_0}$$

Квадратичная интерполяция (L_2)

Добавляется третий узел (x_2, y_2):

$$L_2(x) = L_1(x) + y_2 \cdot \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Для полинома L_n составляется аналогичный полином как для L_1 и L_2 через рекурсивный вызов.

Алгоритм решения:

1. Заданы массивы значений x и y.
2. Для каждого узла вычисляется базисный полином $l_i(x)$.
3. Итоговое значение функции в точке $x = 0.692$ находится по формуле Лагранжа.
4. Строится график интерполяционного многочлена и отмечаются исходные точки.

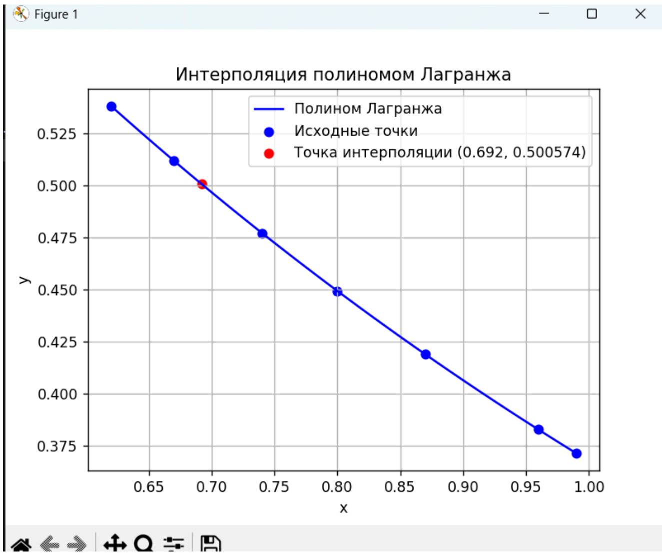


Рис. 1: Метод Лангранжа

Таким образом $L(x) = 0.500574 \pm 0.00001$.

Сделаем проверку решения.

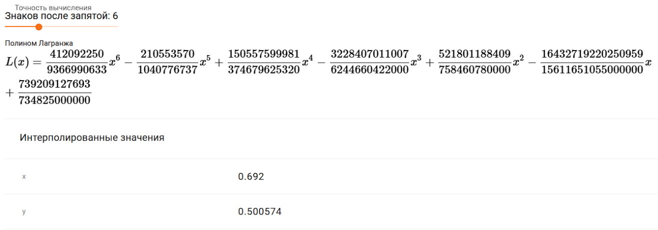


Рис. 2: Проверка решения

После проверки понятно, что программа работает верно и верный рассчеты.

Реализация на Python

```python
import numpy as np
import matplotlib.pyplot as plt

def Lagrange(x, x_arr, y_arr):
    polinom = 0
    for i in range(len(x_arr)):
        l = 1
        for j in range(len(x_arr)):
            if j != i:
                l *= (x - x_arr[j]) / (x_arr[i] - x_arr[j])
        polinom += l * y_arr[i]
    return polinom

x_arr = np.array([0.62, 0.67 , 0.74 , 0.80 , 0.87, 0.96, 0.99])
y_arr = np.array([0.537944, 0.511709, 0.477114, 0.449329, 0.418952, 0.382893,
    0.371577])
x = 0.692
lagr = round(Lagrange(x, x_arr, y_arr), 6)


x_smooth = np.linspace(min(x_arr), max(x_arr), 100)
y_smooth = [Lagrange(xi, x_arr, y_arr) for xi in x_smooth]


plt.plot(x_smooth, y_smooth, label="Полином Лагранжа", color="blue")  # Гладкий
    график полинома
```

```
plt.scatter(x_arr, y_arr, color="blue", label="Исходные точки")  # Узлы интерполяции
plt.scatter(x, lagr, color="red", label=f"Точка интерполяции ({x}, {round(lagr, 6)})")  # Интерполированная точка

plt.title("Интерполяция полиномом Лагранжа")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid()
plt.show()
```

Листинг 1: Python код программы

7.1 Метод Ньютона

Рассмотрим следующий набор данных:

x	$f(x)$
0.01	0.991824
0.06	0.951935
0.11	0.913650
0.16	0.876905
0.21	0.841638
0.26	0.807789
0.31	0.775301
0.36	0.744120
0.41	0.714198
0.46	0.685470
0.51	0.657902
0.56	0.631442

Таблица 2: Таблица значений функции

Произведем интерполяцию значений функции в точках $x_1 = 0.1243$, $x_2 = 0.492$, $x_3 = 0.0024$, $x_4 = 0.660$ с использованием первой и второй формулы Ньютона. Результаты вычислений представлены ниже:

x	Первая формула Ньютона	Вторая формула Ньютона	Оценка погрешности
0.1243	0.902986 ± 0.00001	0.271544 ± 0.00001	0.0 ± 0.00001
0.492	0.66769 ± 0.00001	0.036248 ± 0.00001	8×10^{-6}
0.0024	0.998053 ± 0.00001	0.366611 ± 0.00001	1.6×10^{-5}
0.660	0.55791 ± 0.00001	-0.073532 ± 0.00001	0.058344 ± 0.00001

Таблица 3: Результаты интерполяции

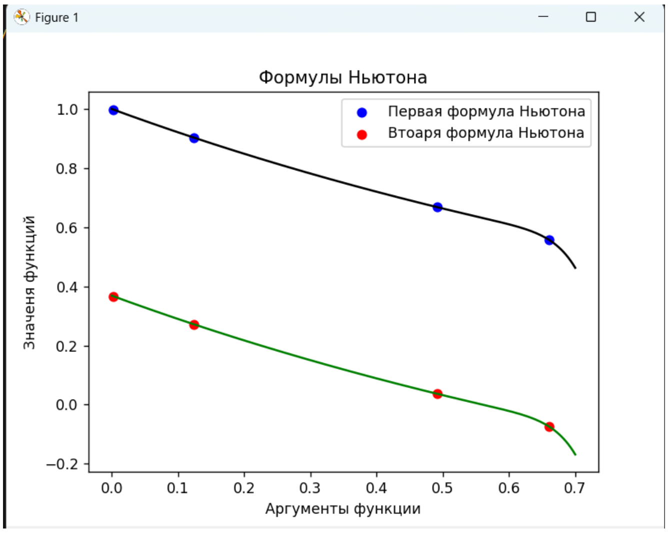
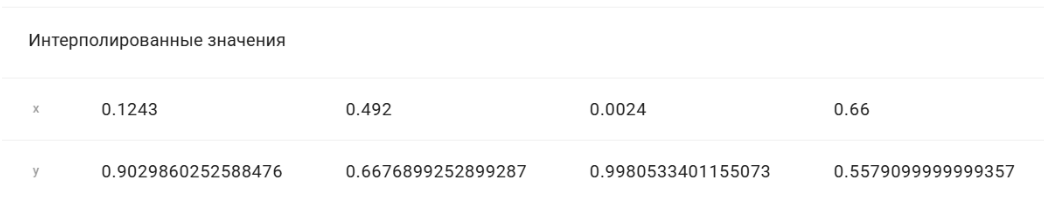


Рис. 3: Графическое представление интерполяции

Резулультаты работы формул для значений в точказ



Интерполированные значения

x	0.1243	0.492	0.0024	0.66
y	0.9029860252588476	0.6676899252899287	0.9980533401155073	0.5579099999999357

Рис. 4: Проверка значений метода Ньютона

Реализация на Python

```python
import numpy  as np
import matplotlib.pyplot as plt

'''
2. По таблице с равноотстоящими значениями аргумента вычислить значения функции
    для заданных значений аргументов,
```

```
    используя первую и вторую интерполяционные формулы Ньютона. Точность E
        <=0.000001.
Задание:

X1=0,1243; X2=0,492; X3=0,0024; X4=0,660;

0,01        0,991824
0,06        0,951935
0,11        0,913650
0,16        0,876905
0,21        0,841638
0,26        0,807789
0,31        0,775301
0,36        0,744120
0,41        0,714198
0,46        0,685470
0,51        0,657902
0,56        0,631442
'''

x_values = np.array([0.01, 0.06, 0.11, 0.16, 0.21, 0.26, 0.31, 0.36, 0.41,
    0.46, 0.51, 0.56])
f_values = np.array([0.991824, 0.951935, 0.913650, 0.876905, 0.841638,
    0.807789, 0.775301, 0.744120, 0.714198, 0.685470, 0.657902, 0.631442])

def newton_divided_diff(x_values, f_values):
    n = len(x_values)
    diff_table = np.zeros((n, n))
    diff_table[:, 0] = f_values

    for j in range(1, n):
        for i in range(n - j):
            diff_table[i, j] = (diff_table[i + 1, j - 1] - diff_table[i, j -
                1]) / (x_values[i + j] - x_values[i])

    return diff_table


def newton_polynomial_1(x, x_values, diff_table):
    n = len(x_values)
```

```
    result = diff_table[0, 0]
    product = 1
    for i in range(1, n):
        product *= (x - x_values[i - 1])
        result += diff_table[0, i] * product
    return result


def newton_polynomial_2(x, x_values, diff_table):
    n = len(x_values)
    result = diff_table[n - 1, n - 1]
    product = 1
    for i in range(n - 2, -1, -1):
        product *= (x - x_values[i + 1])
        result += diff_table[i, n - 1 - i] * product
    return result

def error_estimate(x, x_values, diff_table):
    n = len(x_values)
    product = 1
    for i in range(n - 1):
        product *= (x - x_values[i])
    return round(abs(diff_table[0, n - 1] * product),6)


diff_table = newton_divided_diff(x_values, f_values)


x_targets = [0.1243, 0.492, 0.0024, 0.660]
func_1,func_2=[],[]
x_helper = np.linspace(0,0.7,200)
y_helper_1,y_helper_2 = [],[]
for x in x_helper:
    y_helper_1.append(newton_polynomial_1(x, x_values, diff_table))
    y_helper_2.append(newton_polynomial_2(x, x_values, diff_table))

for x in x_targets:
    f1 =(newton_polynomial_1(x, x_values, diff_table))
    f2 = newton_polynomial_2(x, x_values, diff_table)
    func_1.append(f1)
```

```python
    func_2.append(f2)
    error = error_estimate(x, x_values, diff_table)

    print(f"Для x = {x}:")
    print(f"  Значение по первой формуле Ньютона: {round(f1,6)}")
    print(f"  Значение по второй формуле Ньютона: {round(f2,6)}")
    print(f"  Оценка погрешности: {error}")
    print("-" * 50)

plt.scatter(x_targets,func_1,color = "blue",label = ("Первая формула Ньютона"))
plt.scatter(x_targets,func_2,color = "red", label = ("Втоаря формула Ньютона"))
plt.legend()
plt.xlabel("Аргументы функции")
plt.ylabel("Значеня функций")
plt.title("Формулы Ньютона")
plt.plot(x_helper,y_helper_1,color = "black",label = ("Кривая по первой формуле
    Ньютона"))
plt.plot(x_helper,y_helper_2,color = "green",label = ("Кривая по второй формуле
    Ньютона"))
plt.show()
```

Листинг 2: Python код программы

8 Задание 3

3. По заданным эксперементальным точкам выбрать вид эмпиреической зависимости и выполнить среднеквадратичное приближение функции,

x_i	y_i
0.1	1.91
0.2	3.03
0.3	3.98
0.4	4.82
0.5	5.59
0.6	6.31
0.7	7.00
0.8	7.65
0.9	8.27
1.0	8.88

8.1 Анализ выбора эмпирической зависимости

Перед построением регрессии необходимо определить тип зависимости. В вашем коде использовался следующий подход:

8.2 Расчет характерных точек

- Среднее арифметическое:

$$x = x_1 + \frac{x_n}{2} = 0.1 + \frac{1.0}{2} = 0.6$$

$$y = y_1 + \frac{y_n}{2} = 1.91 + \frac{8.88}{2} \approx 5.35$$

- Среднее геометрическое:

$$x = \sqrt{x_1 \cdot x_n} = \sqrt{0.1 \cdot 1.0} \approx 0.316$$

$$y = \sqrt{y_1 \cdot y_n} = \sqrt{1.91 \cdot 8.88} \approx 4.12$$

- Среднее гармоническое:

$$x = \frac{2x_1 x_n}{x_1 + x_n} = \frac{2 \cdot 0.1 \cdot 1.0}{0.1 + 1.0} \approx 0.182$$

$$y = \frac{2y_1 y_n}{y_1 + y_n} = \frac{2 \cdot 1.91 \cdot 8.88}{1.91 + 8.88} \approx 3.14$$

8.3 Определение ближайших экспериментальных точек

Для каждой характерной точки находим ближайшую экспериментальную точку:

```
Ошибки для каждой модели:
[(0.040, 2.192, 2.533),
(2.370, 0.138, 0.203),
(2.370, 0.138, 0.203)]
```

Минимальные ошибки:

- Для арифметического среднего: 0.040
- Для геометрического среднего: 0.138
- Для гармонического среднего: 0.203

Наименьшая ошибка (0.040) соответствует **арифметическому среднему**, что указывает на линейную зависимость.

После определения типа зависимости строим линейную модель...

Коэффициенты линейной регрессии:

$$a \approx 7.577, \quad b \approx 1.577, \quad \sigma \approx 0.2090.$$

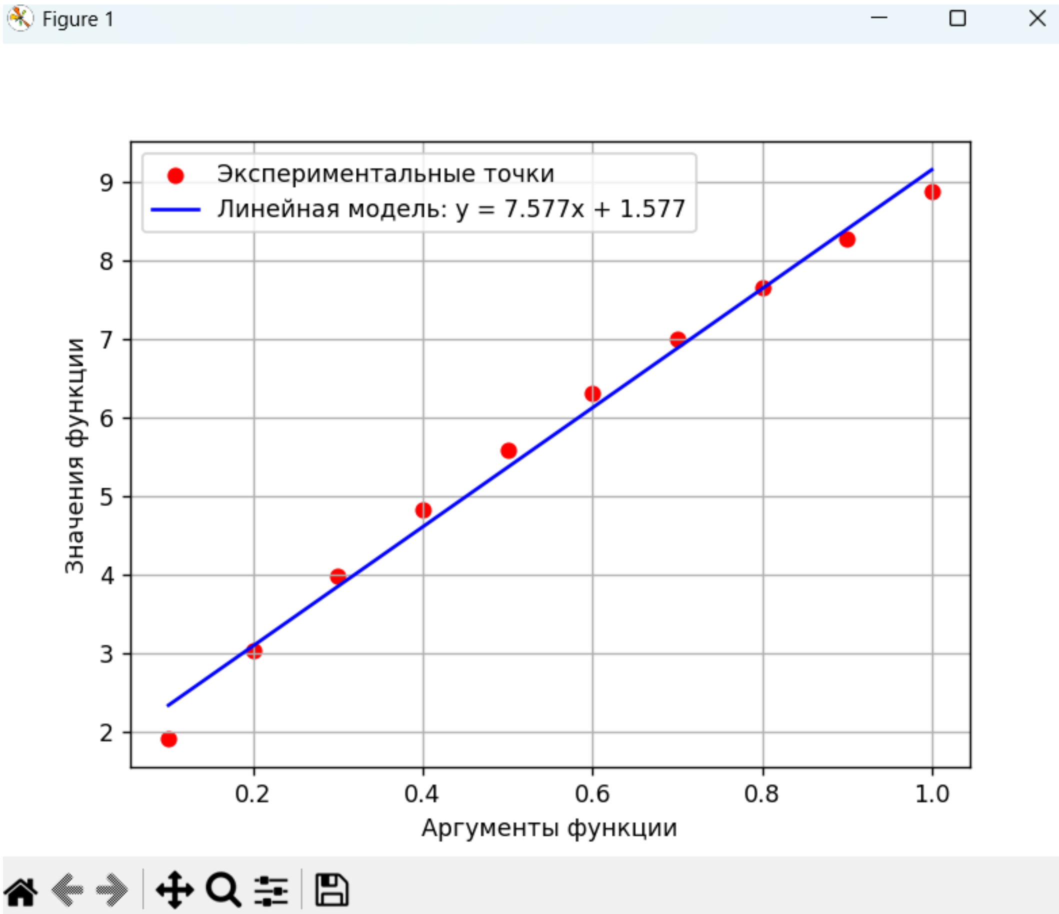


Рис. 5: График линейной зависимости

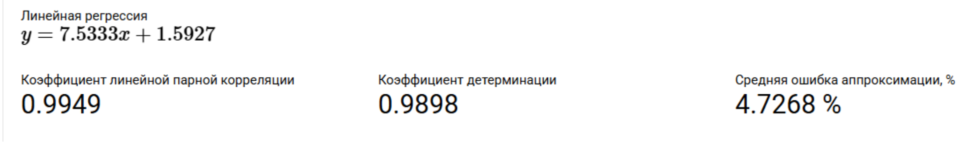


Рис. 6: Проверка метода МНК

8.4 проверка

$\sigma^2 = 0.4768$, значит $\sigma \approx 0.2090$. Поэтому наши вычисления сходятся с проверкой, как и сами коэффиценты.

Реализация на Python

```
# 0,1          1,91
# 0,2          3,03
# 0,3          3,98
# 0,4          4,82
# 0,5          5,59
# 0,6          6,31
# 0,7          7,00
# 0,8          7,65
# 0,9          8,27
# 1,0          8,88
# '''

import matplotlib.pyplot as plt
import numpy as np


# def abolute_error(value_x,value_y,predict_x,predict_y):
#     distance = [0]*len(value_x)
#     for i in range(len(value_x)):
#         distance[i] = np.sqrt((value_x[i]-predict_x)**2 + (value_y[i]-predict_y)**2 )
#     min_dist = min(distance)
#     return distance.index(min_dist)


```

```


# x_value = np.array([0.1,.02,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0],float)
# y_value = np.array([1.91,3.03,3.98,4.82,5.59,6.31,7.00,7.065,8.27,8.88],float
    )

# x_avg = x_value[0]+x_value[-1]/2
# y_avg = y_value[0]+y_value[-1]/2

# x_geom = np.sqrt(x_value[0]*x_value[-1])
# y_geom = np.sqrt(y_value[0]*y_value[-1])

# x_garm = (2*x_value[0]*x_value[-1])/(x_value[0]+x_value[-1])
# y_garm = (2*y_value[0]*y_value[-1])/(x_value[0]+y_value[-1])


# index_y1 = (abolute_error(x_value,y_value,x_avg,y_avg))
# index_y2 = (abolute_error(x_value,y_value,x_geom,y_geom))
# index_y3 = (abolute_error(x_value,y_value,x_garm,y_garm))

# y1 = y_value[index_y1]
# y2 = y_value[index_y2]
# y3 = y_value[index_y3]

# arr_y = [y1,y2,y3]

# y_predict = [y_avg,y_geom,y_garm]
# def calculcate_error(arr_y, y_predict):
#     e=[]
#     for y in arr_y:
#         e1,e2,e3 = [abs(y - y_pred) for y_pred in y_predict]
#         e.append((e1,e2,e3))
#     return e
# error = (calculcate_error(arr_y,y_predict))
# error_min = np.array([min(e) for e in error])
# index_min_e = np.argmin(error_min)
# #index_min_e = 0 => зависимость линейная



```

```python
# Данные
x_value = np.array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0], float)
y_value = np.array([1.91, 3.03, 3.98, 4.82, 5.59, 6.31, 7.00, 7.65, 8.27, 8.88], float)

# Вычисление коэффициентов линейной регрессии
n = len(x_value)
sum_x = np.sum(x_value)
sum_y = np.sum(y_value)
sum_xy = np.sum(x_value * y_value)
sum_x2 = np.sum(x_value ** 2)

a = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x ** 2)
b = (sum_y - a * sum_x) / n

def linear_function(x):
    return a * x + b

# Вычисление среднеквадратичного отклонения
errors = y_value - linear_function(x_value)
print(errors)
sigma = np.sqrt(np.sum(errors ** 2) / n)

# Построение графика
plt.scatter(x_value, y_value, color='red', label='Экспериментальные точки')
plt.plot(x_value, linear_function(x_value), color='blue', label=f'Линейная модель: y = {a:.3f}x + {b:.3f}')
plt.xlabel("Аргументы функции")
plt.ylabel("Значения функции")
plt.legend()
plt.grid()
plt.show()

# Вывод коэффициентов и среднеквадратичного отклонения
print(f"Коэффициенты линейной модели: a = {a:.3f}, b = {b:.3f}")
print(f"Среднеквадратичное отклонение:    = {sigma:.4f}")
```

Листинг 3: Python код программы

9 Вывод

В ходе данной лабораторной работы, я смог ознакомлся с 4 методами интерполяции полинома. А именно: полином Лагранжа, 1 и 2 форма Ньютона и метод наименьших квадратов(МНК). Эти методы были реализованы в программес использованием высокого языка программирования (Python). Также вычесленные корни были проверены, они сошлись с минимальной погрешностью.