МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Вятский государственный университет»
(ФГБОУ ВО «ВятГУ»)
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

МЕТОДЫ ПРИБЛИЖЕНИЯ ФУНКЦИЙ

Отчет по лабораторной работе №3 дисциплины
«Вычислительная математика»

Выполнил студент группы ИВТ-21 _____/Рзаев А.Э./
Проверил преподаватель _____/Архангельский В.В./

2016 г.

1 Постановка задачи

1. По таблице с неравноотстоящими значениями аргумента выполнить интерполяцию, используя формулу Лагранжа. Точность E <= 0,000001.
Задание:

X=0,125

0,02	1,02316
0,08	1,09590
0,12	1,14725
0,17	2,21483
0,23	2,30120
0,30	2,40976

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=0,1074; X2=0,1485; X3=0,1006; X4=0,1560;

0,101	1,26153
0,106	1,27644
0,111	1,29122
0,116	1,30617
0,121	1,32130
0,126	1,33660
0,131	1,35207
0,136	1,36773
0,141	1,38357
0,146	1,39959
0,151	1,41579

3. По заданным экспериментальным точкам выбрать вид эмпирической зависимости и выполнить среднеквадратичное приближение функции, применив метод наименьших квадратов для оценки параметров выбранной зависимости.
Задание:

4,0	1,23
4,1	1,28
4,2	1,33
4,3	1,38
4,4	1,44
4,5	1,49

4. Проверить результаты с помощью системы Wolphram Alpha.

2 Ход выполнения

Формула Лагранжа

При глобальной интерполяции на всем интервале [a;b] строится единый многочлен. Одной из форм записи интерполяционного многочлена для глобальной интерполяции является многочлен Лагранжа (1.1):

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

где $l_i(x)$ – базисные многочлены степени n (1.2):

$$l_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)(x_i - x_1)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)}$$

То есть многочлен Лагранжа (1.3):

$$L_n(x) = \sum_{i=0}^{n} y_i \cdot \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Многочлен $l_i(x)$ удовлетворяет условию (1.4):

$$l_i(x_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}.$$

Это условие означает, что многочлен равен нулю при каждом xj кроме xi, то есть x0, x1, …, xi-1, xi+1, …, xn – корни этого многочлена. Таким образом, степень многочлена $L_n(x)$ равна n и при $x \neq x_i$ в сумме обращаются в нуль все слагаемые, кроме слагаемого с номером $i = j$, равного y_i.

Выражение (1.1) применимо как для равноотстоящих, так и для не равноотстоящих узлов. Погрешность интерполяции методом Лагранжа зависит от свойств функции f(x), от расположения узлов интерполяции и точки x. Полином Лагранжа имеет малую погрешность при небольших значениях n (n<20). При больших n погрешность начинает расти, что свидетельствует о том, что метод Лагранжа не сходится (т.е. его погрешность не убывает с ростом n).

Многочлен Лагранжа в явном виде содержит значения функций в узлах интерполяции, поэтому он удобен, когда значения функций меняются, а узлы интерполяции неизменны. Число арифметических операции, необходимых для построения многочлена Лагранжа, пропорционально n2 и является наименьшим для всех форм записи. К недостаткам этой формы записи можно отнести то, что с изменением числа узлов приходится все вычисление проводить заново.

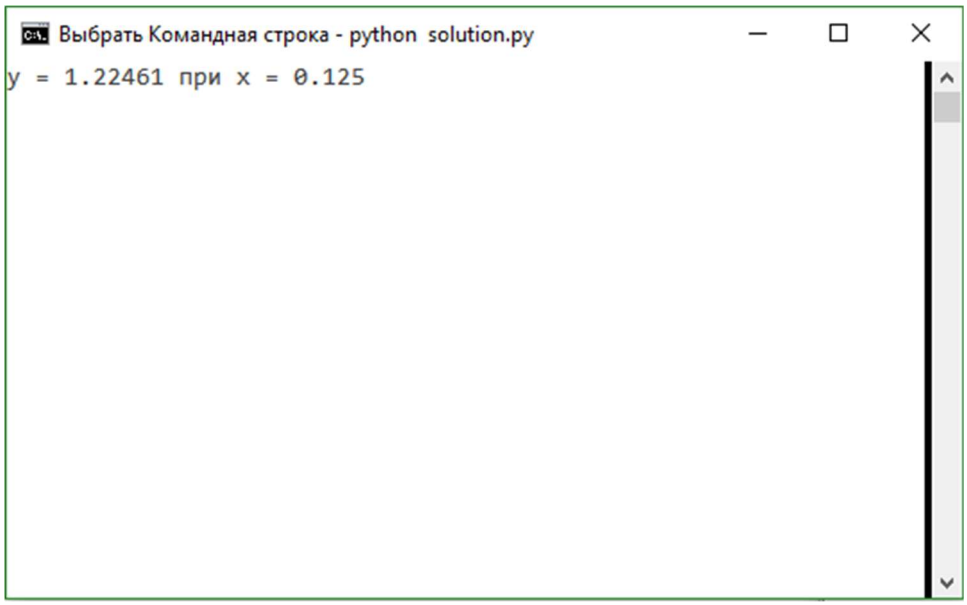

Рисунок 1 – Результат выполнения 1-го задания в программе

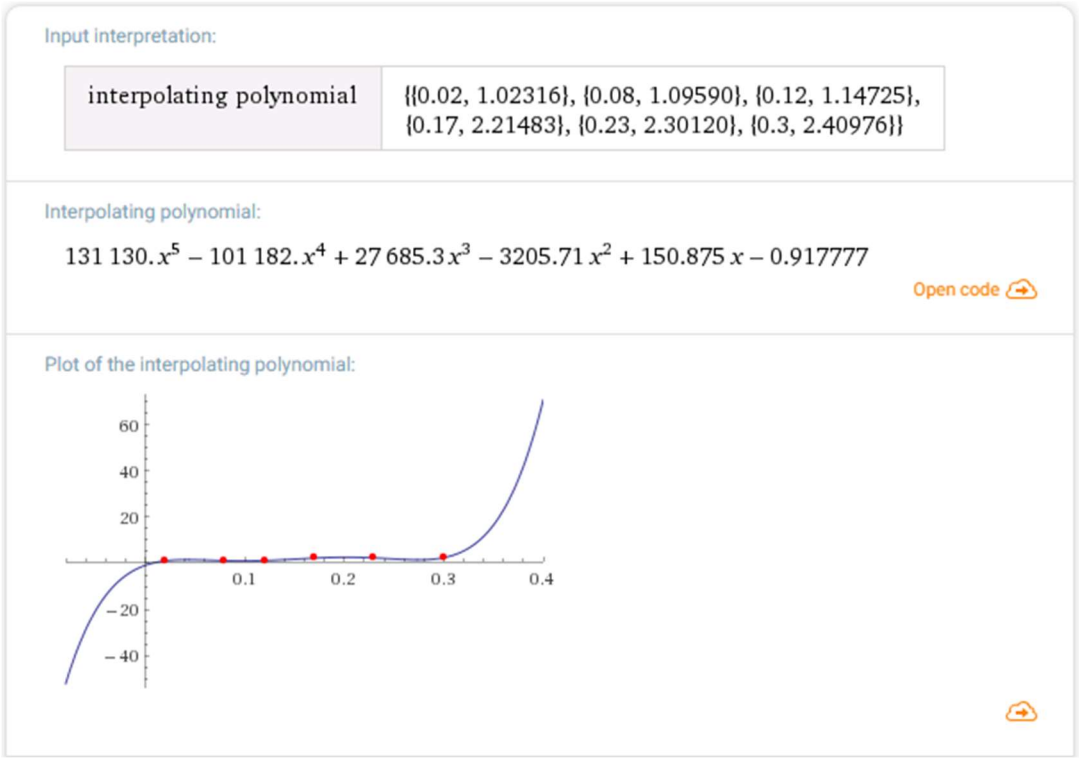

Рисунок 2 – Проверка в Wolphram Alpha

Формула Ньютона

Пусть функция f(x) задана с произвольным шагом и точки таблицы значений занумерованы в произвольном порядке.

Разделенные разности нулевого порядка совпадают со значениями функции в узлах. Разделенные разности первого порядка определяются через разделенные разности нулевого порядка (2.1):

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

Разделенные разности второго порядка определяются через разделенные разности первого порядка (2.2):

$$f(x_i, x_{i+1}, x_{i+2}) = \frac{f(x_{i+1}, x_{i+2}) - f(x_i, x_{i+1})}{x_{i+2} - x_i}$$

Разделенные разности k-го порядка определяются через разделенную разность порядка k-1 (2.3):

$$f(x_i, x_{i+1}, \ldots, x_{i+k}) = \frac{f(x_{i+1}, \ldots, x_{i+k}) - f(x_i, \ldots, x_{i+k-1})}{x_{i+k} - x_i}$$

Используя понятие разделенной разности интерполяционный многочлен Ньютона можно записать в следующем виде (2.4):

$$P_n(x) = f(x_0) + f(x_0, x_1) \cdot (x - x_0) + f(x_0, x_1, x_2) \cdot (x - x_0) \cdot (x - x_1) + \ldots +$$
$$+ f(x_0, x_1, \ldots, x_n) \cdot (x - x_0) \cdot (x - x_1) \ldots \cdot (x - x_{n-1})$$

За точностью расчета можно следить по убыванию членов суммы (2.4). Если функция достаточно гладкая, то справедливо приближенное равенство $f(x) - P_n(x) \approx P_{n+1}(x) - P_n(x)$. Это приближенное равенство можно использовать для практической оценки погрешности интерполяции: $\varepsilon_n = |P_{n+1}(x) - P_n(x)|$.

Для повышения точности интерполяции в сумму могут быть добавлены новые члены, что требует подключения дополнительных узлов. При этом для формулы Ньютона безразлично, в каком порядке подключаются новые узлы, в то время как для формулы Лагранжа при добавлении новых узлов все расчеты надо производить заново.

Предположим, что необходимо увеличить степень многочлена на единицу, добавив в таблицу еще один узел x_{n+1}. Для вычисления $P_{n+1}(x)$ достаточно добавить к $P_n(x)$ лишь одно слагаемое $f(x_0, \ldots, x_n, x_n) \cdot (x - x_0) \cdot (x - x_1) \ldots \cdot (x - x_n)$.

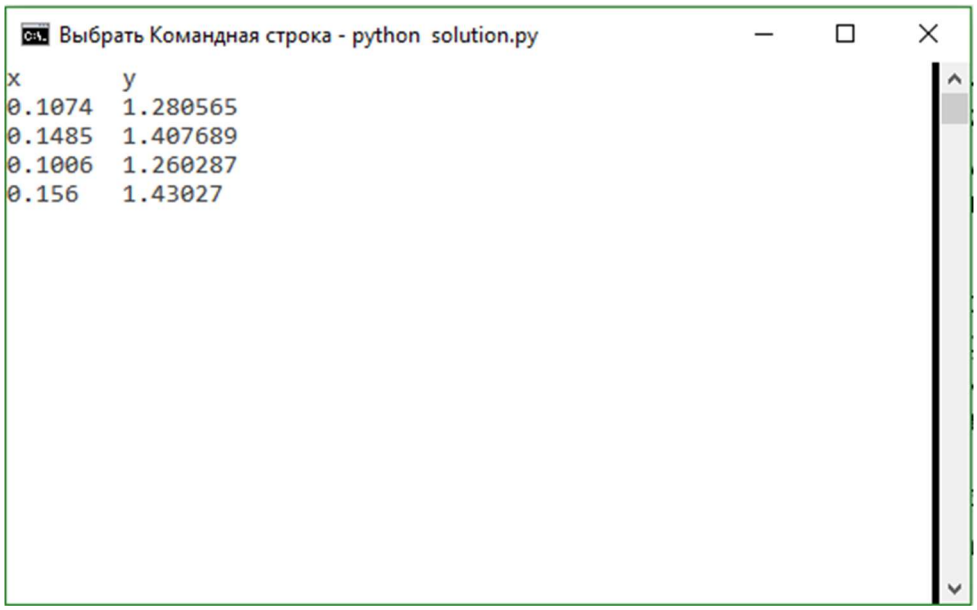

Рисунок 3 – Результат выполнения 2-го задания в программе

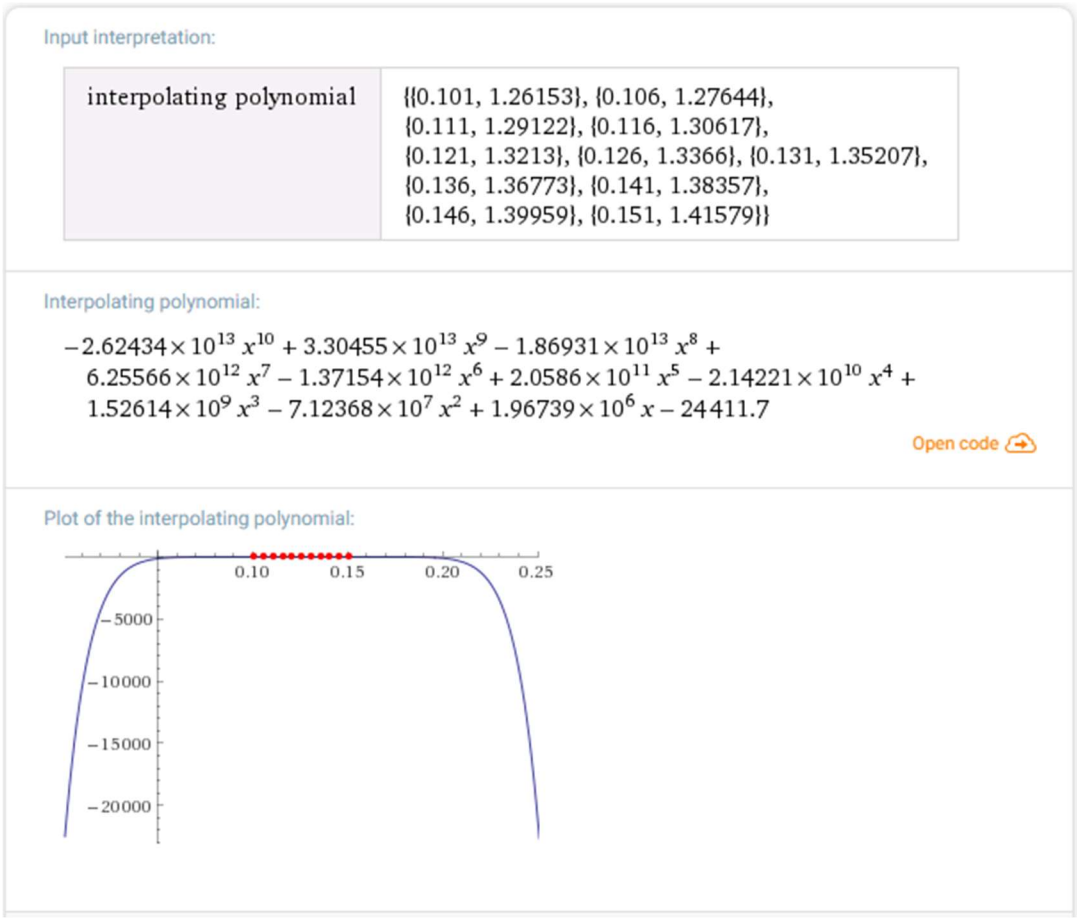

Рисунок 4 – Проверка в WolphramAlpha

Среднеквадратичное приближение

Постановка задачи приближения функции по методу наименьших квадратов.

Пусть функция y=f(x) задана таблицей своих значений: $y_i = f(x_i)$, i=0,1,-n. Требуется найти многочлен фиксированной степени m, для которого среднеквадратичное отклонение (СКО) $\sigma = \sqrt{\frac{1}{n+1}\sum_{i=0}^{n}\left(P_m(x_i) - y_i\right)^2}$ минимально.

Так как многочлен $P_m(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m$ определяется своими коэффициентами, то фактически нужно подобрать набор коэффициентов $a_0, a_1, \ldots a_m$, минимизирующих функцию:

$$\Phi(a_0, a_1, \ldots a_m) = \sum_{i=0}^{n}\left(P_m(x_i) - y_i\right)^2 = \sum_{i=0}^{n}\left(\sum_{j=0}^{m} a_j x_i^j - y_i\right)^2$$

Используя необходимое условие экстремума $\frac{\partial \Phi}{\partial a_k} = 0$ k=0,1,-m получаем так называемую нормальную систему метода наименьших квадратов:

$$\sum_{j=0}^{m}\left(\sum_{i=0}^{n} x_i^{j+k}\right) a_j = \sum_{i=0}^{n} y_i x_i^k$$, k=0,1,-m.

Полученная система есть система алгебраических уравнений относительно неизвестных $a_0, a_1, \ldots a_m$. Можно показать, что определитель этой системы отличен от нуля, то есть решение существует и единственно. Однако при высоких степенях m

система является плохо обусловленной. Поэтому метод наименьших квадратов применяют для нахождения многочленов, степень которых не выше 5. Решение нормальной системы можно найти, например, методом Гаусса.

Запишем нормальную систему наименьших квадратов для двух простых случаев: m = 0 и m = 2. При m = 0 многочлен примет вид: $P_0(x) = a_0$. Для нахождения неизвестного коэффициента a_0 имеем уравнение: $(n+1)a_0 = \sum_{i=0}^{n} y_i$. Получаем, что коэффициент a_0 есть среднее арифметическое значений функции в заданных точках.

Если же используется многочлен второй степени $P_2(x) = a_0 + a_1 x + a_2 x^2$, то нормальная система уравнений примет вид:

$$\begin{cases} (n+1)a_0 + \left(\sum_{i=0}^{n} x_i\right)a_1 + \left(\sum_{i=0}^{n} x_i^2\right)a_2 = \sum_{i=0}^{n} y_i \\ \left(\sum_{i=0}^{n} x_i\right)a_0 + \left(\sum_{i=0}^{n} x_i^2\right)a_1 + \left(\sum_{i=0}^{n} x_i^3\right)a_2 = \sum_{i=0}^{n} y_i x_i \\ \left(\sum_{i=0}^{n} x_i^2\right)a_0 + \left(\sum_{i=0}^{n} x_i^3\right)a_1 + \left(\sum_{i=0}^{n} x_i^4\right)a_2 = \sum_{i=0}^{n} y_i x_i^2 \end{cases}$$

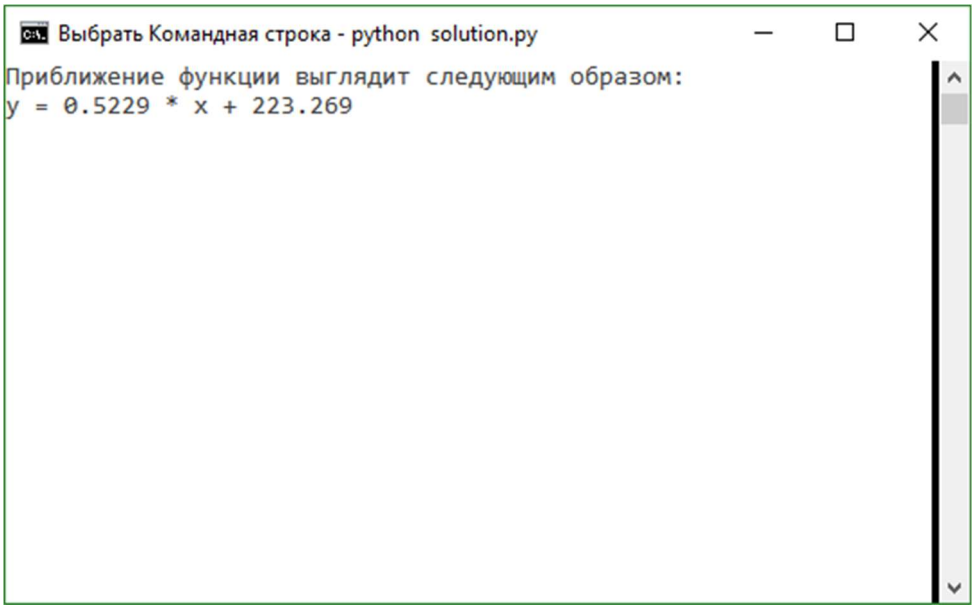

Рисунок 5 – Результат выполнения 3-го задания

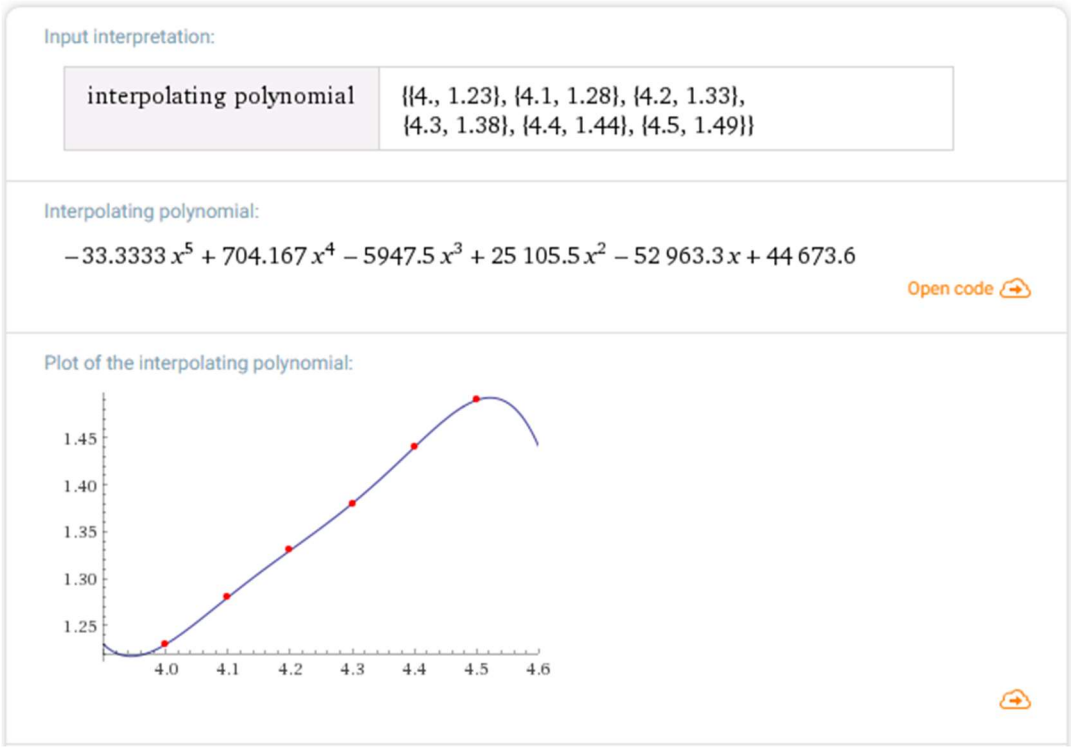

Рисунок 6 – Проверка в Wolphram Alpha

3 Листинг программы

Листинг программы представлен в приложении А.

4 Вывод

В ходе данной лабораторной работы были подробно изучены методы интерполирования, а именно: интерполяция по формулам Лагранжа и Ньютона. Также было изучено среднеквадратичное приближение функции с применением метода наименьших квадратов.

После изучения материала по данным темам, была написана программа, в которой реализовано решение заданий, предоставленных на лабораторную работу, при помощи соответствующих методов.

Полученные при решении ответы можно считать правильными, т.к. они совпадают с параметрами, которые получаются при решении соответствующих заданий при помощи сервиса Wolfram Alpha.

Приложение А
(обязательное)
Листинг кода программы

```
from math import *
from functools import reduce


def newton_interpolation(h: float, func: list, eps: float, points: list):
    xs, ys = func
    n = len(ys)
    diffs = [0] * n

    diffs[0] = ys[:]

    for i in range(1, n):
        diffs[i] = [d2 - d1 for d1, d2 in zip(diffs[i - 1][:-1], diffs[i -
1][1:])]

    res = [0] * len(points)

    factorials = [factorial(n) for n in range(n)]
    for i in range(len(res)):
        q = (points[i] - xs[0]) / h
        #  1, q, q - 1, ... , q - n + 1
        qs = [reduce(lambda a, b: a * b, (q - i for i in range(j)), 1) for j
in range(n)]

        res[i] = sum(qs[i] * diffs[i][0] / factorials[i] for i in range(n))

    return [round(y, int(-log10(eps))) for y in res]


func_newton = [
    (0.101, 0.106, 0.111, 0.116, 0.121, 0.126, 0.131, 0.136, 0.141, 0.146,
0.151),
    (1.26153, 1.27644, 1.29122, 1.30617, 1.32130, 1.33660, 1.35207, 1.36773,
1.38357, 1.39959, 1.41579)
]

h_newton = 0.005

x_newton = [0.1074, 0.1485, 0.1006, 0.1560]

eps_newton = 0.000001

y_newton = newton_interpolation(h_newton, func_newton, eps_newton, x_newton)

print('x\ty')
print('\n'.join('{}\t{}'.format(x, y) for x, y in zip(x_newton, y_newton)))


def lagrange_interpolation(x: float, func: list, eps: float):
    xs, ys = func
    n = len(xs)

    ls = [reduce(lambda a, b: a * b, ((x - xs[j]) / (xs[i] - xs[j]) for j in
range(n) if i != j)) for i in range(n)]
```

```
    res = sum(y * l for y, l in zip(ys, ls))

    return round(res, int(-log10(eps)))


func_lagrange = [
    (0.02, 0.08, 0.12, 0.17, 0.23, 0.3),
    (1.02316, 1.09590, 1.14725, 2.21483, 2.30120, 2.40976)
]

x_lagrange = 0.125

eps_lagrange = 0.000001

print('y = {} при x = {}'.format(lagrange_interpolation(x_lagrange,
func_lagrange, eps_lagrange), x_lagrange))


def ols_interpolation(func: list):
    sum_x = sum(func[0])
    sum_x2 = sum(x ** 2 for x in func[0])
    sum_y = sum(func[1])
    sum_xy = sum(x * y for x, y in zip(*func))
    n = len(func[0])

    d = sum_x2 * n - sum_x ** 2
    d_a = sum_xy * n - sum_y * sum_x
    d_b = sum_x2 * sum_y - sum_x ** 2

    a = d_a / d
    b = d_b / d

    return 'y = {} * x + {}'.format(round(a, 4), round(b, 4))

func_ols = [
    (4.0, 4.1, 4.2, 4.3, 4.4, 4.5),
    (1.23, 1.28, 1.33, 1.38, 1.44, 1.49)
]

print('Приближение функции выглядит следующим
образом:\n{}'.format(ols_interpolation(func_ols)))
```