МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Интерполирование функции многочленом Ньютона и многочленом Лагранжа

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к лабораторной работе №3

по дисциплине

Вычислительная Математика

РУКОВОДИТЕЛЬ:

Суркова Анна Сергеевна

_____

(подпись)

СТУДЕНТ:

Цветков Николай Максимович

_____

(подпись)

19-ИВТ-3

Работа защищена «___» _____

С оценкой _____

Нижний Новгород 2021

Оглавление

Цель

Закрепление знаний и умений по интерполированию функций с помощью многочленов Ньютона и Лагранжа

Постановка задачи

1)Вычислить значение функции при данных значениях аргумента, оценить погрешность:

а) используя первую или вторую интерполяционную формулу Ньютона, в зависимости от значения аргумента;

б) с помощью интерполяционного многочлена Лагранжа, используя формулу для равноотстоящих узлов.

x	y	№ варианта	Значение аргумента				
			x_1	x_2	x_3	x_4	x_5
0,15	4,4817	7	0,166	0,266	0,277	0,144	0,22
0,16	4,953						
0,17	5,4739						
0,18	6,0496						
0,19	6,6859						
0,2	7,3891						
0,21	8,1662						
0,22	9,025						
0,23	9,9742						
0,24	11,0232						
0,25	12,1825						
0,26	13,4637						
0,27	13,5123						

Найти приближенное значение функции при данных значениях аргумента с помощью интерполяционного многочлена Лагранжа, если функция задана в неравноотстоящих узлах таблицы, оценить погрешность

x	y	№ варианта	x_1	x_2
0.43	1.63597	7	0.512	0.441
0.48	1.73234			

0.55	1.87686			
0.62	2.03345			
0.70	2.22846			
0.75	2.35973			

Теоретические сведения

Многочлен Ньютона

Служит для построения многочлена n-й степени, который совпадает в (n+1) точке со значениями неизвестной искомой функции y = f(x).

Пусть в точках x0, x1,…, xn+1 значения функции y = f(x) равны соответственно y0 = f(x0), y1 = f(x1), …, yn+1 = f(xn+1).

Построим интерполяционный многочлен Ньютона с помощью метода неопределенных коэффициентов. Для этого запишем искомый многочлен в виде

Pn(x) = b0 + b1(x – x0) + b2(x – x0)(x – x1) + b3(x – x0)(x – x1)(x – x2) + … + bn(x – x0)…(x – xn). (1)

Последовательно подставляя в формулу (1) вместо x данные значения x0, x1, ..., xn+1, получим для нахождения неопределенных коэффициентов b0, b1, ..., bn «треугольную» систему уравнений

$$\begin{cases} y_0 = b_0, \\ y_1 = b_0 + b_1(x_1 - x_0), \\ y_2 = b_0 + b_1(x_2 - x_0) + b_2(x_2 - x_0)(x_2 - x_1), \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots, \\ y_{n+1} = b_0 + b_1(x_{n+1} - x_0) + \dots + b_n(x_{n+1} - x_0) \cdot \dots \cdot (x_{n+1} - x_n) \end{cases}$$

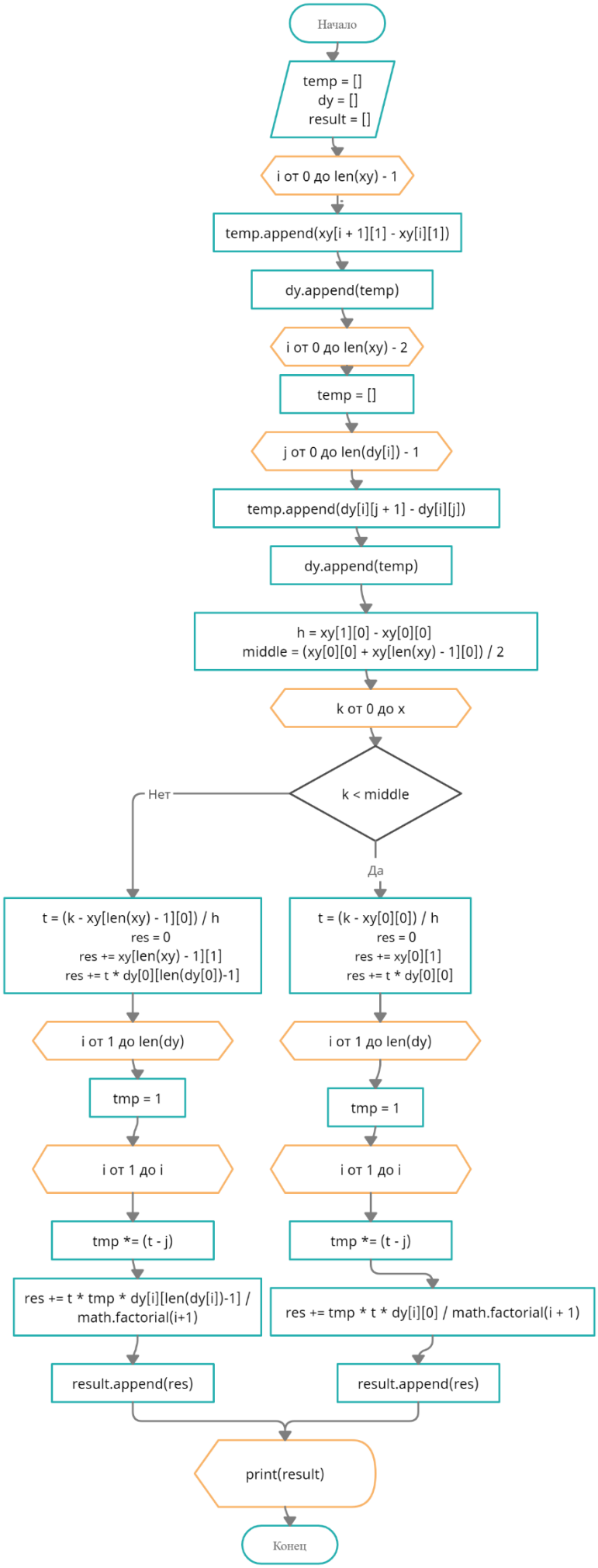
(при подстановке в равенство (1) вместо x числа x0 в правой части равенства обратились в нуль все слагаемые, кроме первого: там везде был множитель (x – x0), обратившийся в нуль; при подстановке x = x1 обратились в нуль все слагаемые, кроме первого и второго – они содержат множитель (x – x1) и т.д.).

Полученную систему удобно решать: из первого её уравнения находим свободный член искомого многочлена b0; подставив его во второе уравнение, находим коэффициент b1 при первой степени x в искомом многочлене:

$$b_1 = \frac{y_1 - b_0}{x_1 - x_0}$$

и т.д.

Для интерполяционного многочлена Ньютона можно выписать явные выражения коэффициентов через данные задачи, а также и оценки точности замены неизвестной функции f(x) этим многочленом.


Начало
temp = []
dy = []
result = []
i от 0 до len(xy) - 1
temp.append(xy[i + 1][1] - xy[i][1])
dy.append(temp)
i от 0 до len(xy) - 2
temp = []
j от 0 до len(dy[i]) - 1
temp.append(dy[i][j + 1] - dy[i][j])
dy.append(temp)
h = xy[1][0] - xy[0][0]
middle = (xy[0][0] + xy[len(xy) - 1][0]) / 2
k от 0 до x
k < middle
Нет
Да
t = (k - xy[len(xy) - 1][0]) / h
res = 0
res += xy[len(xy) - 1][1]
res += t * dy[0][len(dy[0])-1]
t = (k - xy[0][0]) / h
res = 0
res += xy[0][1]
res += t * dy[0][0]
i от 1 до len(dy)
tmp = 1
i от 1 до i
tmp *= (t - j)
res += t * tmp * dy[i][len(dy[i])-1] / math.factorial(i+1)
i от 1 до len(dy)
tmp = 1
i от 1 до i
tmp *= (t - j)
res += tmp * t * dy[i][0] / math.factorial(i + 1)
result.append(res)
result.append(res)
print(result)
Конец

Многочлен Лагранжа для неравноотстоящих узлов

Для решения задачи интерполяции функцию $y = f(x)$ заменяют многочленом $L_n(x)$ степени не выше n, который используют для приближенного вычисления значений функции. Многочлен полностью определяется требованием, чтобы его значения и значения $f(x)$ совпадали в узлах интерполяции:

$$f(x_0) = L_n(x_0),\ f(x_1) = L_n(x_1),\ \ldots,\ f(x_n) = L_n(x_n). \qquad (1)$$

Будем искать многочлен в виде

$$L_n(x) = \sum_{k=0}^{n} c_k \prod_{\substack{j=0 \\ j\neq k}}^{n} (x - x_j), \qquad (2)$$

где символ $\prod$ называется знаком произведения, а выражение $\prod_{j=0}^{n} a_j$ представляет собой сокращенную запись произведения $a_0 \cdot \ldots \cdot a_n$. Например,

$$\sum_{k=0}^{2} c_k \prod_{\substack{j=0 \\ j\neq k}}^{2} (x - x_j) =$$

$$= c_0(x - x_1)(x - x_2) + c_1(x - x_0)(x - x_2) + c_2(x - x_0)(x - x_1).$$

Для решения задачи интерполяции необходимо определить коэффициенты $c_0, c_1, \ldots, c_n$ многочлена (2).

Положим в формуле (2) $x = x_i$ и используем условия (1). Получим

$$y_i = f(x_i) = L_n(x_i) = c_i \prod_{\substack{j=0 \\ j\neq i}}^{n} (x_i - x_j),$$

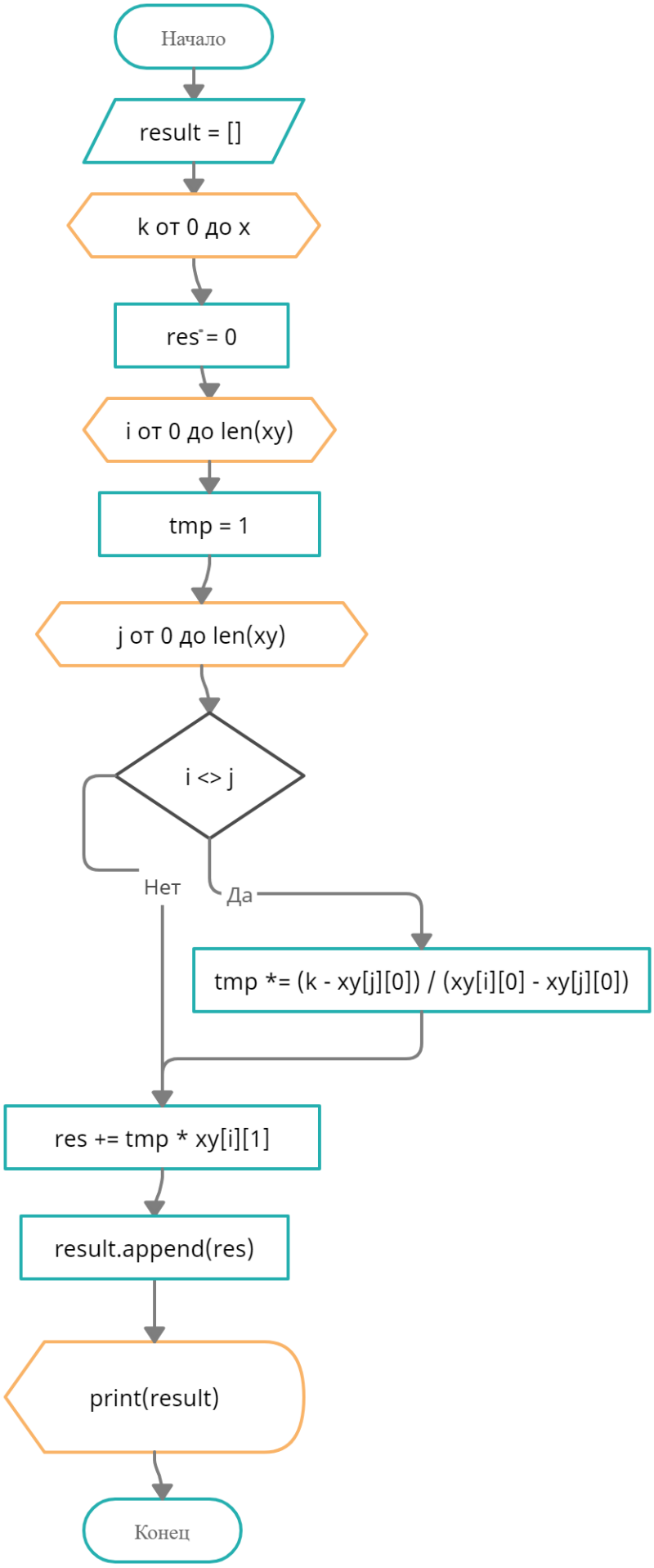
следовательно,

$$c_i = y_i / \prod_{\substack{j=0 \\ j\neq i}}^{n} (x_i - x_j).$$

Таким образом, искомый многочлен имеет вид

$$L_n(x) = \sum_{k=0}^{n} y_k \prod_{\substack{j=0 \\ j\neq k}}^{n} \frac{x - x_j}{x_k - x_j}. \qquad (3)$$

Он называется ***интерполяционным многочленом Лагранжа для неравноотстоящих узлов***. Это — единственный многочлен, решающий задачу интерполяции[1)].


Начало
result = []
k от 0 до x
res = 0
i от 0 до len(xy)
tmp = 1
j от 0 до len(xy)
i <> j
Нет
Да
tmp *= (k - xy[j][0]) / (xy[i][0] - xy[j][0])
res += tmp * xy[i][1]
result.append(res)
print(result)
Конец

2. Интерполяционный многочлен Лагранжа для равноотстоящих узлов. Рассмотрим случай, когда значения x_i являются равноотстоящими, т. е.

$$x_1 - x_0 = x_2 - x_1 = \ldots = x_n - x_{n-1} = h.$$

При этом, если ввести обозначение $\frac{x - x_0}{h} = t$, то получим:

$$\Phi_i(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)} =$$

$$= \frac{th\,(th-h)\ldots[th-(i-1)\,h]\,[th-(i+1)\,h]\ldots(th-nh)}{ih\,(i-1)\,h\ldots h\,(-h)\ldots[-(n-i)\,h]} =$$

$$= \frac{t(t-1)\ldots(t-n)}{t-i}\,\frac{(-1)^{n-i}}{i!\,(n-i)!} = (-1)^{n-i}\,C_n^i\,\frac{1}{t-i}\,\frac{t(t-1)\ldots(t-n)}{n!}.$$

Итак,

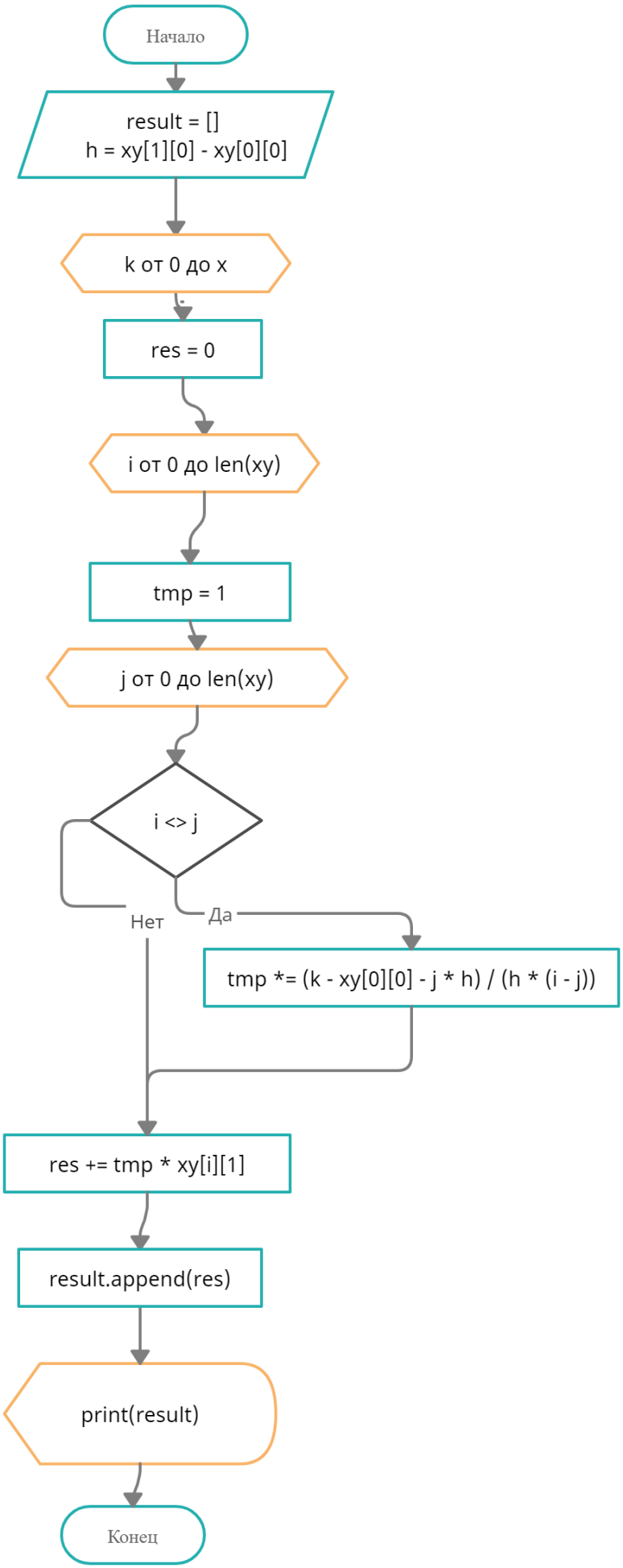
$$L_n(x) = L_n(x_0 + th) =$$

$$= (-1)^n\,\frac{t(t-1)\ldots(t-n)}{n!}\sum_{i=0}^{n}(-1)^i\,\frac{C_n^i y_i}{t-i}. \qquad (4)$$

Здесь и в дальнейшем для сокращения записей мы будем обозначать $f(x_i)$ через y_i. В последнем выражении коэффициенты, стоящие перед y_i:

$$(-1)^{n-i}\,C_n^i\,\frac{t(t-1)\ldots(t-n)}{(t-i)\,n!},$$

не зависят ни от функции $f(x)$, ни от h — шага таблицы. Их можно табулировать и использовать в самых различных случаях. Такие таблицы составлены и известны под названием таблиц *коэффициентов Лагранжа*.


Начало
result = []
h = xy[1][0] - xy[0][0]
k от 0 до x
res = 0
i от 0 до len(xy)
tmp = 1
j от 0 до len(xy)
i <> j
Нет
Да
tmp *= (k - xy[0][0] - j * h) / (h * (i - j))
res += tmp * xy[i][1]
result.append(res)
print(result)
Конец

Расчетные данные

Задание №1

x	y	№ варианта	Значение аргумента				
			x_1	x_2	x_3	x_4	x_5
0,15	4,4817	7	0,166	0,266	0,277	0,144	0,22
0,16	4,953						
0,17	5,4739						
0,18	6,0496						
0,19	6,6859						
0,2	7,3891						
0,21	8,1662						
0,22	9,025						
0,23	9,9742						
0,24	11,0232						
0,25	12,1825						
0,26	13,4637						
0,27	13,5123						

Значения, полученные при помощи многочлена Ньютона для равноотстоящих узлов:

X	У
0,166	5.276395
0,266	13.847712
0,277	12.058941
0,144	4.21309
0,22	14.111716

Значения, полученные при помощи многочлена Лагранжа для равноотстоящих узлов:

X	У
0,166	5.276395
0,266	13.847712
0,277	12.058941

0,144	4.21309
0,22	14.111716

Задание №2

x	y	№ варианта	x_1	x_2
0.43	1.63597	7	0.512	0.441
0.48	1.73234			
0.55	1.87686			
0.62	2.03345			
0.70	2.22846			
0.75	2.35973			

Значения, полученные при помощи многочлена Лагранжа для неравноотстоящих узлов:

X	У
0.512	1.7969695
0.441	1.656703

Листинг разработанной программы

Main.py

```
import solutionMethods as sm

# x и y из 1 задания
xy1 = [[0.15, 4.4817], [0.16, 4.953], [0.17, 5.4739], [0.18, 6.0496],[0.19, 6.685
9], [0.20, 7.3891], [0.21, 8.1662], [0.22, 9.025], [0.23, 9.9742], [0.24, 11.0232
], [0.25, 12.1825], [0.26, 13.4637], [0.27, 13.5123]]
xi = [0.166, 0.266, 0.277, 0.144, 0.22] # Точки, в которых нужно найти значения ф
ункции в первом задании

xy2 = [[0.43, 1.63597], [0.48, 1.73234], [0.55, 1.87686], [0.62, 2.03345], [0.70,
 2.22846], [0.75, 2.35973]] # x и y  из 2 задания
xj = [0.512, 0.441] # Точки, в которых нужно найти значения функции во втором зад
ании



sm.newtonPolynomial(xy1, xi)
sm.lagrangEquitable(xy1, xi)
sm.lagrangUnequitable(xy2, xj)
```

solutionMethods.py

```
import math
import os
import keyboard

def newtonPolynomial(xy, x):
    os.system('cls')
    print("Многочлен Ньютона:")
    temp = [] # Временные значения
    dy = [] # Конечные разности
    result = [] # Список результатов
    for i in range(len(xy) - 1): # Конечные разности первого порядка заносим во в
ременный список
        temp.append(xy[i + 1][1] - xy[i][1]) # Вычисляем сами разности
    dy.append(temp) # Временный список заносим в список конечных разностей

    for i in range(len(xy) - 2): # На каждом новом шаге вычисляем конечные разнос
ти следующих порядков
        temp = [] # Очищаем временные значения
        for j in range(len(dy[i]) - 1):
            temp.append(dy[i][j + 1] - dy[i][j]) # Вычисляем конечные разности
        dy.append(temp) # Временный список заносим в список конечных разностей

    h = xy[1][0] - xy[0][0] # Вычисляем шаг
    middle = (xy[0][0] + xy[len(xy) - 1][0]) / 2 # Считаем середину отрезка иксов

    for k in x: # Для каждой точки, в которой нужно найти значение, находим это з
начение
```

```
        if k < middle: # Если xi лежит в промежутке от x0 до (x0 + xn) / 2
            t = (k - xy[0][0]) / h # Вычисляем t по формуле (x - x0)/h
            res = 0  # Интерполяция вперед
            res += xy[0][1] # Прибавляем к результату  y0 + t * dy0
            res += t * dy[0][0]
            for i in range(1, len(dy)): # Считаем остальное:  (t * (t-
1) * ...  * (t - n + 1) * dny0) / n!
                tmp = 1
                for j in range(1, i): # Для удобства отдельно считаем (t - 1)..(t
 - n+1)
                    tmp *= (t - j)
                res += tmp * t * dy[i][0] / math.factorial(i + 1) # Добавляем  (t
 * (t-1) * ...  * (t - n + 1) * dny0) / n!
            result.append(res)
        else:
            t = (k - xy[len(xy) - 1][0]) / h # Вычисляем t по формуле (x - x0)/h
            res = 0 # Интерполяция назад
            res += xy[len(xy) - 1][1] # Прибавляем к результату  y0 + t * dy0
            res += t * dy[0][len(dy[0])-1]
            for i in range(1, len(dy)): # Считаем остальное:  (t * (t-
1) * ...  * (t - n + 1) * dny0) / n!
                tmp = 1
                for j in range(1, i): # Для удобства отдельно считаем (t - 1)..(t
 - n+1)
                    tmp *= (t + j)
                res += t * tmp * dy[i][len(dy[i])-
1] / math.factorial(i+1) # Добавляем  (t * (t-
1) * ...  * (t - n + 1) * dny0) / n!
            result.append(res)
    print(result)
    keyboard.wait('\n')
    os.system('cls')



def lagrangEquitable(xy, x):
    print("Лагранж равноотстоящий:")
    result = [] # Список результатов
    h = xy[1][0] - xy[0][0] # Считаем шаг
    for k in x: # Для каждой точки, в которой нужно найти значение, находим это з
начение
        res = 0
        for i in range(len(xy)):
            tmp = 1 # Временный буфер
            for j in range(len(xy)):
                if i != j:
                    tmp *= (k - xy[0][0] - j * h) / (h * (i - j)) # Считаем (X -
Xi - j * h) / (h(i - j))
            res += tmp * xy[i][1]
        result.append(res) # Заносим в список результатов
```

```
    print(result)
    keyboard.wait('\n')
    os.system('cls')



def lagrangUnequitable(xy, x):
    print("Лагранж неравноотстоящий:")
    result = [] # Список результатов
    for k in x: # Для каждой точки, в которой нужно найти значение, находим это з
начение
        res = 0
        for i in range(len(xy)):
            tmp = 1 # Временный буфер
            for j in range(len(xy)):
                if i != j:
                    tmp *= (k - xy[j][0]) / (xy[i][0] - xy[j][0]) # Считаем (X -
Xj) / (Xi - Xj)
            res += tmp * xy[i][1]
        result.append(res) # Заносим в список результатов
    print(result)
    keyboard.wait('\n')
    os.system('cls')
```

Результаты работы программы

```
Многочлен Ньютона:
[5.276395987956776, 13.847712786353782, 12.05894102887251, 4.213090444351655, 14.111716666666666]
```

```
Лагранж равноотстоящий:
[5.258189268299199, 13.959348404357364, 6.98422075758991, 3.892011395243887, 9.024999999999995]
```

```
Лагранж неравноотстоящий:
[1.7969695304244333, 1.6567032877950671]
```

Вывод

В ходе данной работы были закреплены знания и умения по интерполированию функции с помощью многочленов Ньютона и Лагранжа.