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

**Численное дифференцирование функций**

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к лабораторной работе №5

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 4](#_Toc69654469)

[Постановка задачи 5](#_Toc69654470)

[Теоретические сведения 6](#_Toc69654471)

[Расчетные данные 10](#_Toc69654472)

[Листинг разработанной программы 11](#_Toc69654473)

[Результаты работы программы 13](#_Toc69654474)

[Вывод 14](#_Toc69654475)

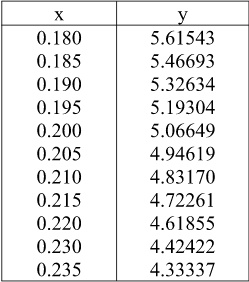
# Цель

Закрепление знаний и умений по численному дифференцированию функций с помощью интерполяционного многочлена Ньютона.

# Постановка задачи

Вычислить первую и вторую производные функции в точках х, заданные таблицей.

**Вариант №4**

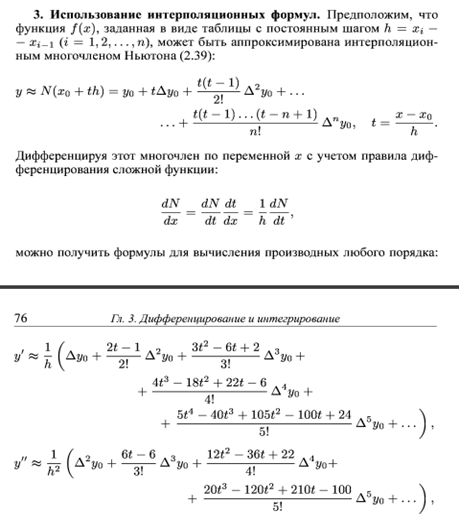


# Теоретические сведения

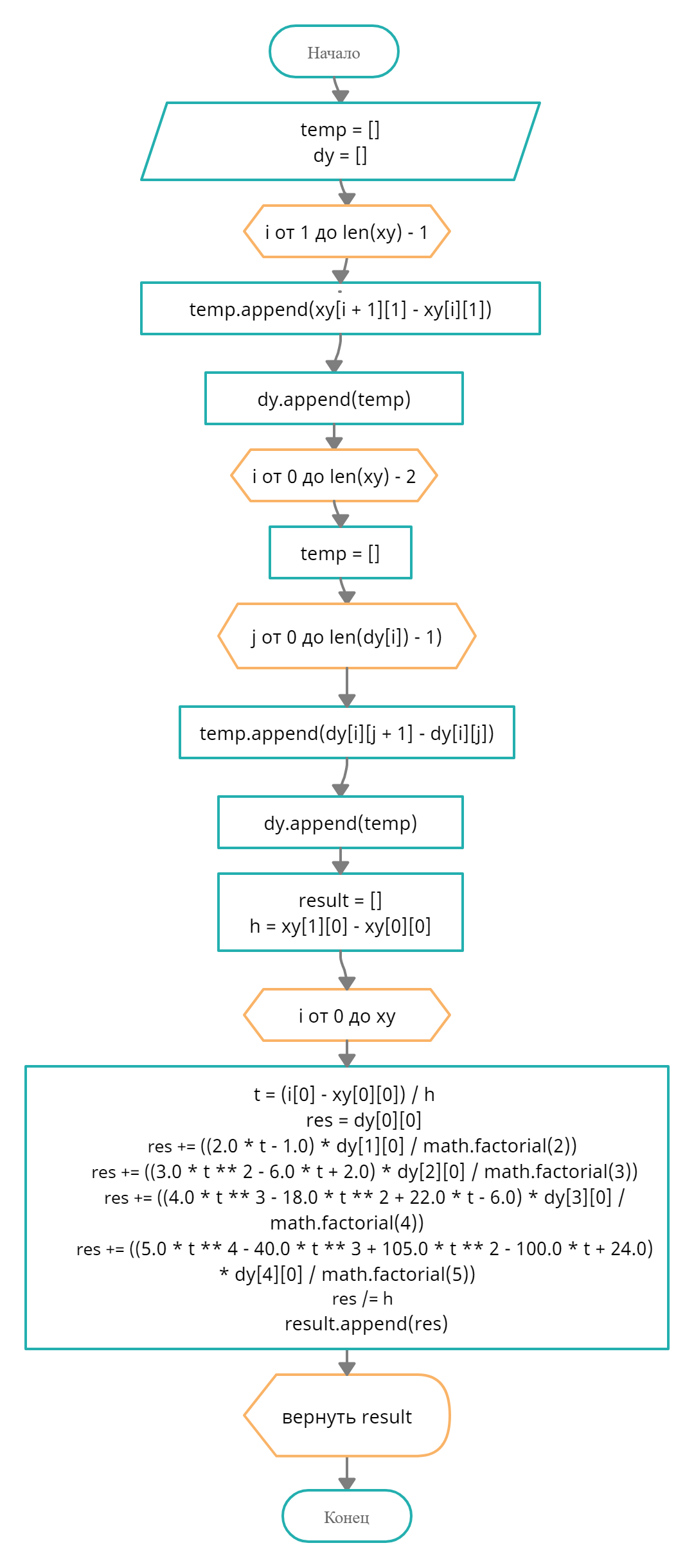
Предположим, что функция , заданная в виде таблицы с постоянным шагом может быть аппроксимированная интерполяционным многочленом Ньютона:

Дифференцируя этот многочлен по переменной x с учетом правила дифференцирования сложной функции:

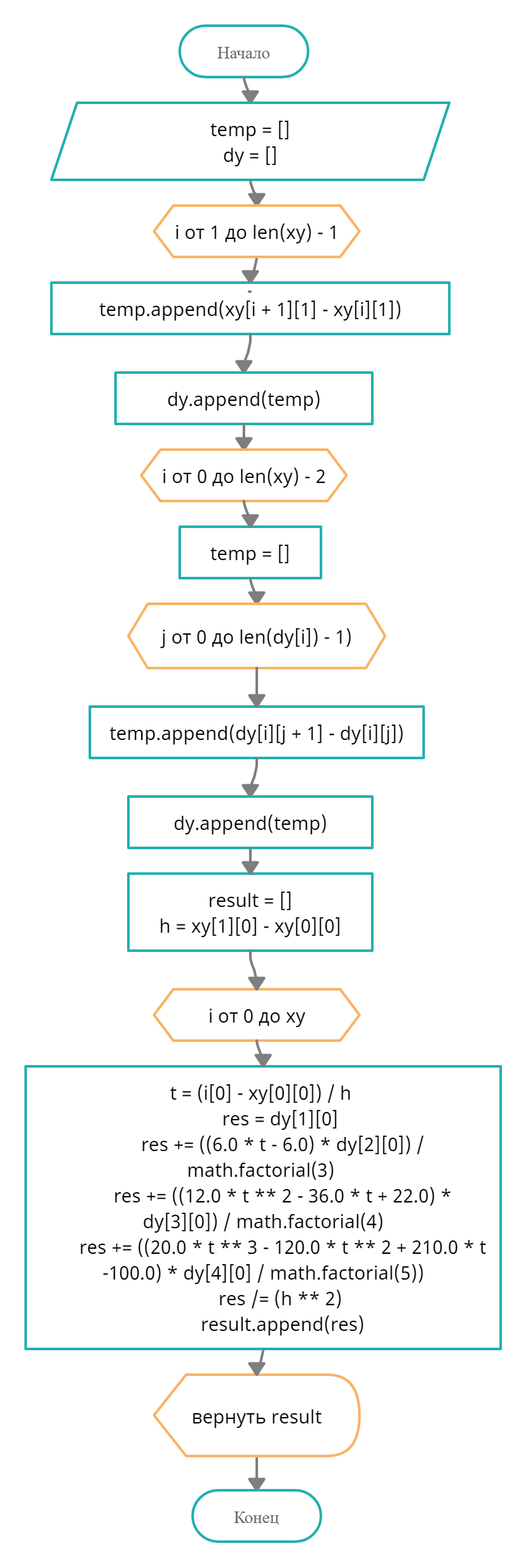
можно получить формулы для вычисления производных любого порядка:



**Нахождение первой производной**

****

**Нахождение второй производной**

****

# Расчетные данные

|  |  |  |  |
| --- | --- | --- | --- |
| X | Y | Y’ | Y” |
| 0.180 | 5.61543 | -30.54 | 345.466667 |
| 0.185 | 5.46693 | -28.89 | 316.000000 |
| 0.190 | 5.32634 | -27.37 | 291.333333 |
| 0.195 | 5.19304 | -25.97 | 269.866667 |
| 0.200 | 5.06649 | -24.67 | 250.000000 |
| 0.205 | 4.94619 | -23.47 | 230.133333 |
| 0.210 | 4.83170 | -22.37 | 208.666667 |
| 0.215 | 4.72261 | -21.39 | 184.000000 |
| 0.220 | 4.61855 | -20.54 | 154.533333 |
| 0.230 | 4.42422 | -19.36 | 74.8000000 |
| 0.235 | 4.33337 | -19.12 | 21.3333333 |

# Листинг разработанной программы

**Main.py**

from solutionMethods import \*

# Таблица значений исходной функции

xy =  [[0.180, 5.61543], [0.185, 5.46693], [0.190, 5.32634], [0.195, 5.19304], [0.200, 5.06649], [0.205, 4.94619], [0.210, 4.83170], [0.215, 4.72261], [0.220, 4.61855], [0.230, 4.42422], [0.235, 4.33337]]

firstDer = firstDerivative(xy)

secondDer = secondDerivative(xy)

print("\n  X       Y  ")

for i in range(len(xy)):

    print("%.2f" % (xy[i][0]), " %.6f" % (xy[i][1]))

print("\n  Y'      Y''")

for i in range(len(xy)):

    print("%.2f" % (firstDer[i]), " %.6f" % (secondDer[i]))

**solutionMethods.py**

import math

def firstDerivative(xy): # Метод получения первой производной при помощи интерполяции многочленом Ньютона

    temp = [] # Здесь мы храним временные значения конечных разностей

    dy = [] # Список списков для хранения значений конечных разностей

    for i in range(len(xy) - 1): # В промежуточный список заносим конечные разности 1-ого порядка

        temp.append(xy[i + 1][1] - xy[i][1]) # Считаем конечные разности

    dy.append(temp) # Заносим промежуточный список в список списков конечных разностей

    for i in range(len(xy) - 2): # На каждом i-ом шаге вычисляем значения конечных разностей нового порядка

                                 # и заносим в промежуточный список.

        temp = []  # Инициализация промежуточного списка пустым

        for j in range(len(dy[i]) - 1):

            temp.append(dy[i][j + 1] - dy[i][j]) # Считаем конечные разности

        dy.append(temp) # Промежуточный список заносим в список списков промежуточных разностей

    result = [] # В этом списке мы храним полученные значения

    h = xy[1][0] - xy[0][0] # Считаем шаг h

    for i in xy:

        t = (i[0] - xy[0][0]) / h # Вычисляем параметр t, который зависит от h

        res = dy[0][0] # К результату прибавляем Δy0

        res += ((2.0 \* t - 1.0) \* dy[1][0] / math.factorial(2)) # Прибавляем к результату последующие слагаемые до Δ^5y0

        res += ((3.0 \* t \*\* 2 - 6.0 \* t + 2.0) \* dy[2][0] / math.factorial(3))

        res += ((4.0 \* t \*\* 3 - 18.0 \* t \*\* 2 + 22.0 \* t - 6.0) \* dy[3][0] / math.factorial(4))

        res += ((5.0 \* t \*\* 4 - 40.0 \* t \*\* 3 + 105.0 \* t \*\* 2 - 100.0 \* t + 24.0) \* dy[4][0] / math.factorial(5))

        res /= h # Из-за того, что узлы равноотстоящие, нужно разделить результат на h

        result.append(res) # Заносим результат в список ответов

    return result

def secondDerivative(xy): # Метод получения второй производной при помощи интерполяции многочленом Ньютона

    temp = [] # Здесь мы храним временные значения конечных разностей

    dy = [] # Список списков для хранения значений конченых разностей

    for i in range(len(xy) - 1): # В промежуточный список заносим конечные разности 1-ого порядка

        temp.append(xy[i + 1][1] - xy[i][1]) # Cчитаем конечные разности

    dy.append(temp) # Промежуточный список заносим в список списков конечных разностей

    for i in range(len(xy) - 2):# На каждом i-ом шаге вычисляем значения конечных разностей нового порядка

                                # и заносим в промежуточный список.

        temp = [] # Инициализация промежуточного списка пустым списком

        for j in range(len(dy[i]) - 1):

            temp.append(dy[i][j + 1] - dy[i][j]) # Считаем конечные разности

        dy.append(temp) # Полученный промежуточный список заносим в список списков промежуточных разностей

    result = [] # В этом списке мы храним полученные значения

    h = xy[1][0] - xy[0][0] # Считаем шаг h

    for i in xy:

        t = (i[0] - xy[0][0]) / h # Считаем параметр t, зависящий от h

        res = dy[1][0] # К результату прибавляем Δ^2y0

        res += ((6.0 \* t - 6.0) \* dy[2][0]) / math.factorial(3) # Прибавляем к результату последующие слагаемые до Δ^5y0

        res += ((12.0 \* t \*\* 2 - 36.0 \* t + 22.0) \* dy[3][0]) / math.factorial(4)

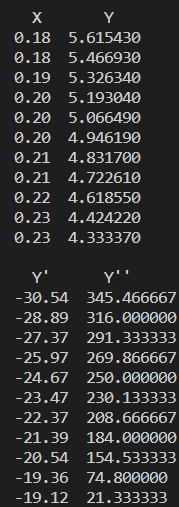
        res += ((20.0 \* t \*\* 3 - 120.0 \* t \*\* 2 + 210.0 \* t -100.0) \* dy[4][0] / math.factorial(5))

        res /= (h \*\* 2) # Так как узлы равноотстоящие, делим полученный результат на h

        result.append(res) # Заносим результат в список ответов

    return result

# Результаты работы программы



# Вывод

В ходе данной работы были закреплены знания и умения по вычислению производных первого и второго порядка при помощи интерполяционного многочлена Ньютона.