|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_***ИУК «Информатика и Управление»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №1**

**«Методы численного дефференцирования»**

**ДИСЦИПЛИНА: «Моделирование»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-62Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Калашников А. С.)  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Никитенко У. В. )  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |
| Калуга, 2023 | | |

**Цель работы:** сформировать практические навыки анализа возможностей построения и выделения наиболее важных свойств объектов моделей для моделирования и использования специализированных программных пакетов и библиотек для стандартных вычислений и визуализации результатов численной аппроксимации производных и обоснования выбора алгоритма аппроксимации.

**Задача:** изучить методы аппроксимации производных, методы оценки точности аппроксимации, методы повышения точности аппроксимации, количественные характеристики методов, написать программы, указанные в вариантах.

**Вариант 2**

f(x) = e3x

**№1** Вычислить приближённо значения:

а) первой производной заданной функции f(x) с порядком погрешности

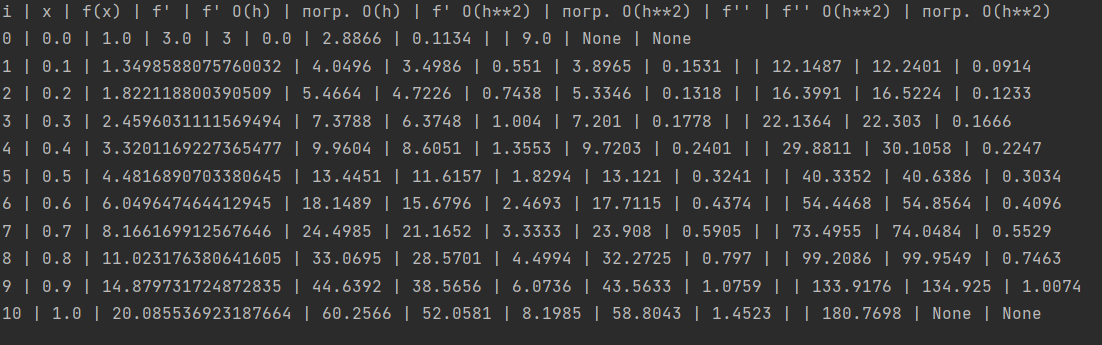
O(h) и O(h2);

б) второй производной заданной функции f(x) с порядком погрешности O(h2).

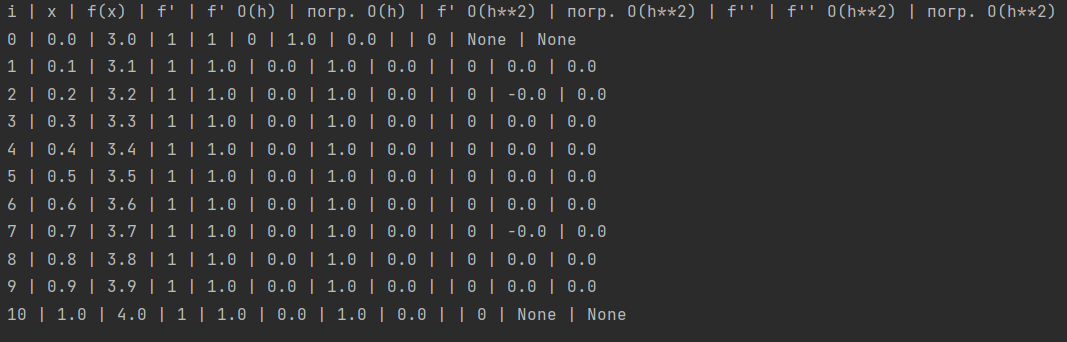
Напечатать таблицу значений узлов; значений производных, полученных аналитическим методом в узлах; приближённых значений производных в узлах и фактическую погрешность. Проверить результаты на многочленах соответствующих степеней.

Исходный код программы представлен в Приложении.

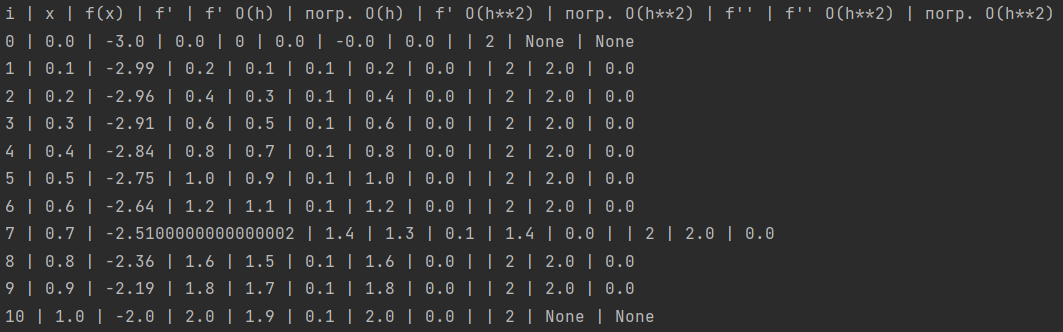
Результат выполнения программы для заданной функции f(x) = e3x и диапазона значений от 0,0 до 1,0 с шагом 0,1:

****

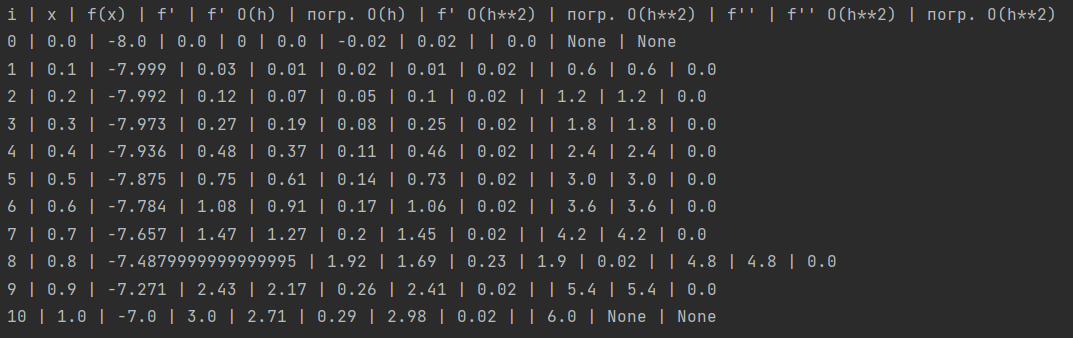
Используем программу с теми же параметрами, но с функцией f(x) = x + 3, у которой f’(x) = 1:

****

Далее возьмём многочлен второй степени f(x) = x2 – 3, у которого f’(x) = 2x, а f’’(x) = 2:

****

И, наконец, возьмём многочлен третьей степени f(x) = x3 - 8, у которого f’(x) = 3x2, а f’’(x) = 6x

****

Как видим погрешность вычислений появляется только тогда, когда производная является выпуклой функции, например, квадратичной. Это объясняется тем, что при аппроксимации мы принимаем функцию линейной на очень маленьком масштабе, соответственно чем меньше шаг, тем точнее полученное значение.

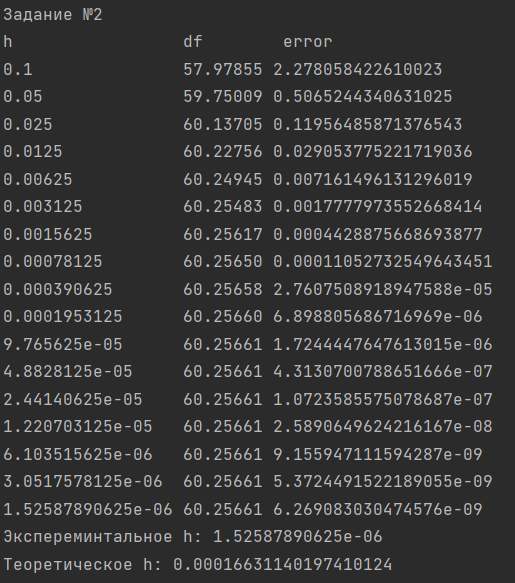
**№2** Пользуясь формулой:

D:\3-kurs-2-semestr\Моделирование\LB1\Картинки\6.png

в точке х0=1 вычислить вторую разностную производную второго порядка аппроксимации функции e3x, последовательно уменьшая шаг h до тех пор, пока фактическая погрешность не начнёт возрастать. Определить h оптимальное экспериментально и теоретически, объяснить полученные результаты.

Исходный код программы представлен в Приложении.

Результат выполнения приложения:

****

Выполним округление до 5-ого знака после запятой, то

.

Третья производная функции в точке 1 будет равно:

Подставляя данные значения и учитывая округление до 5-го знака, получаем:

Вычисленное экспериментально оптимальное значение шага:

.

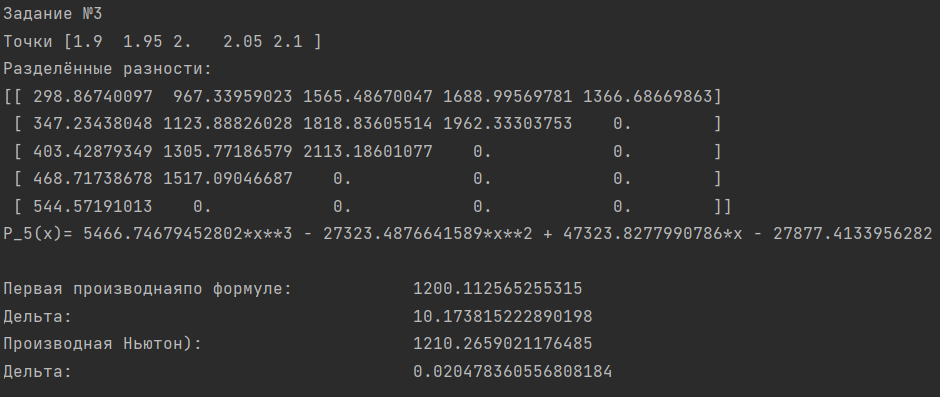
Таким образом, полученное аналитически значение шага примерно равно экспериментальному значению.

**№3** Предполагается заданной таблица значений функции в равноотстоящих узлах xi , i = 0, . . . , n. Требуется дифференцированием интерполяционного многочлена в форме Ньютона получить формулу численного дифференцирования для вычисления приближенного значения первой производной с третьим порядком аппроксимации в точке x = x1. Получить выражение для погрешности. Применить формулу для вычисления производной, сравнить с точным значением

Исходный код программы представлен в Приложении

Запишем интерполяционный полином Ньютона с использованием разделённых разностей:

Вычислим разделённые разности, воспользовавшись формулой:

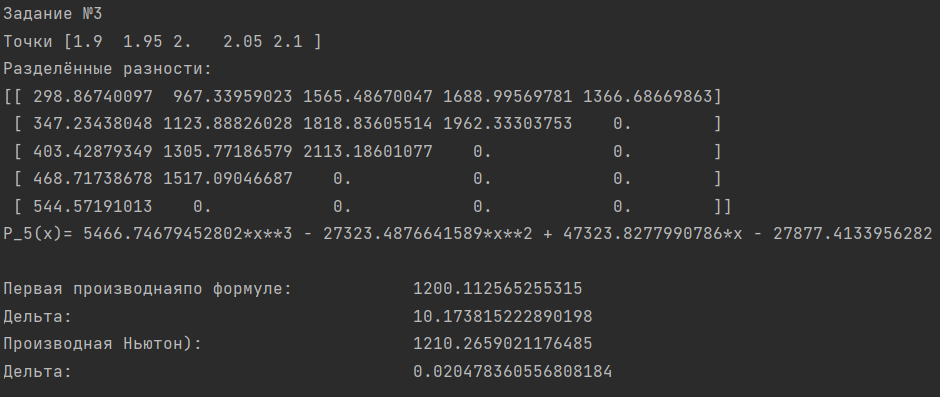
****

Для вычисления полинома воспользуемся формулой обратно разделенной разности Ньютона. Продифференцируем:

,

где .

Получаем следующий результат интерполяционного полинома для производной заданной функции:

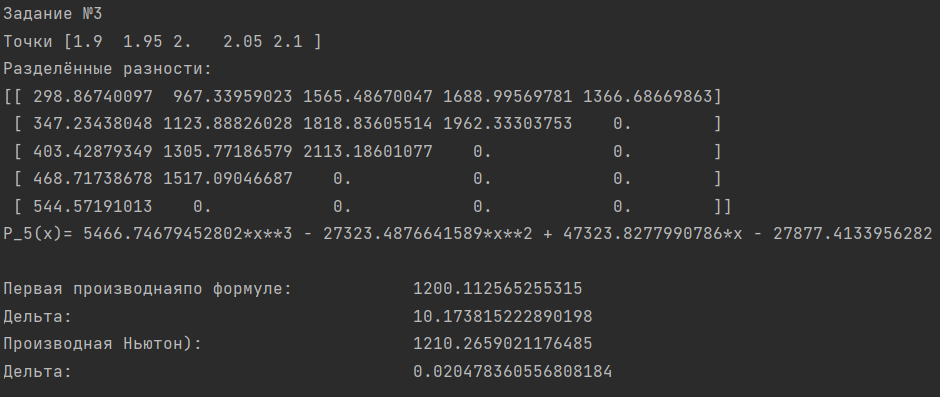
****

Найдем выражение для погрешности.

Пусть – случайная точка, отличная от точек . Если – полином интерполяции Ньютона, которые интерполирует функцию в точках и , тогда . Тогда

Продифференцируем и получим:

Воспользуемся формулой для вычисления первой производной:

****

Погрешность при вычислении с помощью полинома Ньютона, имеет погрешность более чем в 1000 раз меньше значения, полученного с помощью формулы численного дифференцирования.

**Вывод:** в ходе выполнения данной лабораторной работы были сформированы практические навыки анализа возможностей построения и выделения наиболее важных свойств объектов моделей для моделирования и использования специализированных программных пакетов и библиотек для стандартных вычислений и визуализации результатов численной аппроксимации производных и обоснования выбора алгоритма аппроксимации.

Приложение

import numpy  
import sympy  
from math import \*  
  
def analite\_first\_derivative(x):  
 return 3\*exp(3\*x)  
  
def analite\_second\_derivative(x):  
 return 9\*exp(3\*x)  
def function(x):  
 return exp(3\*x)  
  
  
def divided\_diff(x, y):  
 n = len(y)  
 coef = zeros([n, n])  
 coef[:, 0] = y  
  
 for j in range(1, n):  
 for i in range(n - j):  
 coef[i][j] = \  
 (coef[i + 1][j - 1] - coef[i][j - 1]) / (x[i + j] - x[i])  
  
 return coef  
  
  
def newton\_poly\_derivative(coef, points, x):  
 n = len(points) - 1  
 dp = 0  
 p = coef[n]  
  
 for k in range(1, n + 1):  
 dp \*= (x - points[n-k])  
 dp += p  
 p \*= (x - points[n-k])  
 p += coef[n-k]  
  
 return dp  
  
def ex3():  
  
 point = 2  
 start = 1.9  
 stop = 2.1  
 n = 5  
  
 points = linspace(start=start, stop=stop, num=n)  
 y = [function(x) for x in points]  
  
 newton\_divided\_diff = divided\_diff(points, y)  
 coef = newton\_divided\_diff[0, :]  
  
 newton\_first\_derivative = newton\_poly\_derivative(coef, points, 2)  
 delta = abs(analite\_first\_derivative(point) - newton\_first\_derivative)  
  
 print("Точки",points)  
 print(f"Разделённые разности:")  
 print(newton\_divided\_diff)  
  
 sym\_x = sympy.Symbol("x")  
 polynom = newton\_poly\_derivative(coef, points, sym\_x)  
 polynom = sympy.simplify(polynom)  
 print(f"P\_{n}(x)=", polynom, '\n')  
  
 print(f"{'Первая производнаяпо формуле:':40} {(4 \* function(2.05) - 3 \* function(2.0) - function(2.1)) / .05 / 2}")  
 print(  
 f"{'Дельта:':40} {abs(analite\_first\_derivative(point) - (4 \* function(2.05) - 3 \* function(2.0) - function(2.1)) / .05 / 2)}")  
 print(f"{'Производная Ньютон:':40} {newton\_first\_derivative}")  
 print(f"{'Дельта:':40} {abs(analite\_first\_derivative(point) - newton\_first\_derivative)}")

def ex2():  
 x = 1  
 h = 0.1  
 acc = 3 \* exp(3 \* x)  
 y = df(x, h)  
 error = y - acc  
 print("h df error")  
 while (abs(y - acc) <= abs(error)):  
 error = y - acc  
 print("{0:<17} {1:3.5f} {2}".format(h, y, abs(error)))  
 h = h / 2;  
 y = df(x, h)  
 print("{0:<17} {1:3.5f} {2}".format(h, y, abs(y - acc)))  
 print('Экспереминтальное h: {0}'.format(h))  
 print('Теоретическое h: {0}'.format(sqrt(3 \* (5 / 1000000) / (27 \* exp(3 \* x)))))  
def derivative\_oh(func, x: list, n: int):  
 derivative = n \* [None]  
 derivative[0] = round((func(x[1]) - func(x[0])) / (x[1] - x[0]))  
 for i in range(1, n):  
 derivative[i] = round((func(x[i]) - func(x[i - 1])) /(x[i] - x[i - 1]), 4)  
 return derivative  
  
def derivative\_oh2(func, x: list, n: int):  
 derivative = n \* [None]  
 derivative[0] = round((4 \* func(x[1]) - 3 \* func(x[0]) - func(x[2])) / (x[1] - x[0]) / 2, 4)  
 derivative[1] = round((4 \* func(x[2]) - 3 \* func(x[1]) - func(x[3])) / (x[2] - x[1]) / 2, 4)  
 for i in range(2, n):  
 derivative[i] = round((3 \* func(x[i]) - 4 \* func(x[i - 1]) + func(x[i - 2])) / (x[i] - x[i - 1]) / 2, 4)  
 return derivative  
  
def second\_derivative\_oh2(func, x: list, n: int):  
 derivative = n \* [None]  
 for i in range(1, n - 1):  
 derivative[i] = round((func(x[i + 1]) - 2 \* func(x[i]) + func(x[i - 1])) / (x[i] - x[i - 1]) \*\* 2, 4)  
 return derivative  
  
def calculate\_inaccuraty(first: list, second: list):  
 if len(first) != len(second):  
 return [0]  
 inacuraty = len(first) \* [0];  
 for i in range(0, len(first)):  
 if first[i] is None or second[i] is None:  
 inacuraty[i] = None  
 else:  
 inacuraty[i] = round(abs(first[i] - second[i]), 4)  
 return inacuraty  
  
  
  
def analite\_first\_derivative(x):  
 return 3\*exp(3\*x)  
  
def analite\_second\_derivative(x):  
 return 9\*exp(3\*x)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 count = 11  
 x = [round(i, 3) for i in numpy.linspace(0, 1, count)]  
 first\_derivative = [round(analite\_first\_derivative(i), 4) for i in x]  
 second\_derivative = [round(analite\_second\_derivative(i), 4) for i in x]  
  
 a1 = derivative\_oh(function, x, len(x))  
 a2 = derivative\_oh2(function, x, len(x))  
 a3 = second\_derivative\_oh2(function, x, len(x))  
 inaccuraty = calculate\_inaccuraty(a1, first\_derivative)  
 inaccuraty2 = calculate\_inaccuraty(a2, first\_derivative)  
 inaccuraty3 = calculate\_inaccuraty(a3, second\_derivative)  
 print("i","|","x","|","f(x)","|","f'","|","f' O(h)","|","погр. O(h)","|"  
 ,"f' O(h\*\*2)","|","погр. O(h\*\*2)","|","f''","|"  
 ,"f'' O(h\*\*2)","|","погр. O(h\*\*2)")  
 for i in range(count):  
 print(str(i),"|",x[i],"|",function(x[i]),"|",first\_derivative[i],"|",a1[i],"|"  
 ,inaccuraty[i],"|",a2[i],"|",inaccuraty2[i],"|"  
 ,"|",second\_derivative[i],"|",a3[i],"|",inaccuraty3[i])  
 print("\n")  
 print("Задание №2")  
 ex2()

print("\n")  
 print("Задание №3")  
 ex3()