МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет» (ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ОТЧЁТ по лабораторной работе № 3

«Численное дифференцирование таблично заданной функции с помощью многочлена Лагранжа»

Вариант № 8

Выполнила: студент гр. Б9122-02.03.01сцт
Ф.И.О.
Ильяхова Алиса Алексеевна

Проверил: преподаватель
Ф.И.О.
Павленко Елизавета Робертовна

Владивосток

2024

Цель работы:

Привести интерполяционную формулу Лагранжа к удобному для дифференцирования виду, дополнительно заменив разности координат точек на разность индексов, умноженную на шаг сетки $(x_i - x_j) = (i - j) * h$. Сравнить $L_n^{(k)}(x_m) \sim f^{(k)}(x_m)$.
Получить и оценить минимальное и максимальное значение остаточного члена $R_{n,k}(x)$. Проверить, выполняется ли неравенство $\min(R_{n,k}) < R_{n,k}(x) < \max(R_{n,k})$.

Основное:

1.1. Данные и их инициализация в коде:

$y = x^2 - \sin(x)$ на промежутке [0.5, 1.0]

n = 4, k =1, m = 3

```
def func(x):
    return x ** 2 - sp.sin(x)
```

```
def main():
    x = sp.symbols('x')
    n = 4
    k = 1
    m = 3
    a = 0.5
    b = 1.0
    step = (b - a) / 3
```

1.2. Вывод первой производной по методу Лагранжа:

По условию лабораторной работы, вычисляем только одну производную.

$$L_n(x) = \sum_{i=0}^{n} f(x_i) \prod_{j=0;\, j\neq i}^{n} (x - x_j)$$

$$L_n'(x) = \sum_{i=0}^{n} f(x_i) \prod_{j=0;\, j\neq i}^{n} \frac{1}{x_i - x_j} \cdot \frac{d}{dx} \prod_{j=0;\, j\neq i}^{n} (x - x_j) = \sum_{i=0}^{n} f(x_i) \prod_{j=0;\, j\neq i}^{n} \frac{1}{x_i - x_j} \cdot \left(\sum_{j=0;\, j\neq i}^{n} \prod_{\substack{j_1=0 \\ j_1\neq j\neq i}}^{n} (x - x_{j_1}) \right)$$

$$L_n'(x_m) = \sum_{i=0}^{n} \frac{f(x_i)}{h} \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{1}{i - j} \cdot \left(\sum_{j=0}^{n} \prod_{\substack{j1=0 \\ j1\neq j\neq i}}^{n} (m - j) \right)$$

1.3. Создание таблицы значений функции:

x	f(x)
0.5	-0.229425538604203

0.6666666666666666	-0.173925358625293
0.8333333333333333	-0.0457324087515927
0.9999999999999999	0.158529015192103

Функция **values** создает таблицу значений функции на заданном интервале. Результатом является список, представляющий пары (x, f(x)), где x - значения аргумента, а f(x) - соответствующие значения функции на этом аргументе.

```
def values(a, b, step):
    table = []
    x = a
    while x <= b:
        table.append((x, func(x)))
        x += step
    return table
```

1.4. Вычисление многочлена Лагранжа:

Функция **lagrange_polynomial** вычисляет многочлен Лагранжа для заданных точек. Она принимает список точек (x_i, y_i) и аргумент x, для которого необходимо вычислить значение многочлена.

```
def lagrange_polynomial(points, x):
    l = 0
    for i, (x_i, y_i) in enumerate(points):
        l_i = 1
        for j, (x_j, _) in enumerate(points):
            if i != j:
                l_i *= (x - x_j) / (x_i - x_j)
        l += y_i * l_i
    return l
```

Вывод:

-0.229425538604203*(2.0 - 2.0*x)*(2.5 - 3.0*x)*(4.0 - 6.0*x) - 0.173925358625293*(3.0 - 3.0*x)*(5.0 - 6.0*x)*(6.0*x - 3.0) - 0.0457324087515927*(6.0 - 6.0*x)*(3.0*x - 1.5)*(6.0*x - 4.0) + 0.158529015192103*(2.0*x - 1.0)*(3.0*x - 2.0)*(6.0*x - 5.0)

1.5. Вычисление множителя omega:

Функция **omega** вычисляет множитель для разделенной разности в многочлене Лагранжа. Этот множитель используется при вычислении многочлена Лагранжа для определенных точек.

```
def omega(a, b, step, x):
    res = 1
    while round(a, 1) <= b:
```

```
        res *= (x - a)
        a += step
    return res
```

1.6. Основная работа функции main:

В функции **main** происходит организация последовательности вычислений и анализа результатов. Это включает вычисление многочлена Лагранжа, его производной, вычисление исходной функции и анализ погрешности.

```
points = values(a, b, step)
print(points)

L = lagrange_polynomial(points, x)

print(f"Многочлен Лагранжа: {L}")

L_diff = sp.diff(L, x)

f = x ** 2 - sp.sin(x)

d = take_diff(f, x, n)
df = take_diff(f, x, n)
r_1 = d.subs(x, 0.5) - L_diff.subs(x, 0.5)
r_min = (df.subs(x, a) / math.factorial(4)) * omega(a, b, step, x)

r_max = df.subs(x, b) / math.factorial(4) * omega(a, b, step, x)

print(f"Производная многочлена Лагранжа: {L_diff}")

print(L.subs(x, 0.5))

print(func(0.5).evalf())
print('-')
print(L_diff.subs(x, 0.5))
print(d.subs(x, 0.5))
print('-')
print(r_1)
print(r_min.subs(x, a))
print(r_max.subs(x, b))
```

1.7. Остаточный член:

Остаточный член = разность $\min(R_{n,k})$ и $\max(R_{n,k})$ =>

1.08127368274043

1.8. Проверка неравенства:

$\min(R_{n,k}) < R_{n,k}(x) < \max(R_{n,k})$

0 < -0.601099717143710 < -1.08127368274043 => условия не выполняются

Используемые библиотеки:

В ходе работы мне потребовалось использовать следующие библиотеки: sympy, math.

Библиотека **sympy** представляет собой мощный символьный математический пакет для Python. Она способна обрабатывать символьные выражения, уравнения, и действия, что делает её полезным инструментом в области научных вычислений, анализа данных и математического моделирования.
Библиотека **math** в Python предоставляет функции для выполнения математических операций над числами. Она включает в себя функции для работы с простыми и сложными математическими операциями, такими как тригонометрия, логарифмы, округления чисел и т.д.

Вывод:

1. Была реализована программа для дифференцирования таблично заданной функции при помощи многочлена Лагранжа.
2. Программа вычисляет многочлен Лагранжа, его производную, а также погрешности для заданных значений
3. В конечном итоге программа выводит многочлен Лагранжа, его производную, значения многочлена и производной в указанных точках, а также оценки погрешностей.
4. Полученные данные позволяют оценить аппроксимацию дифференцирования таблично заданной функции при помощи многочлена Лагранжа.

Полный код:

```
import sympy as sp
import math
def func(x):
    return x ** 2 - sp.sin(x)
def values(a, b, step):
    table = []
    x = a
    while x <= b:
        table.append((x, func(x)))
        x += step
    return table
```

```
def lagrange_polynomial(points, x):
    l = 0
    for i, (x_i, y_i) in enumerate(points):
        l_i = 1
        for j, (x_j, _) in enumerate(points):
            if i != j:
                l_i *= (x - x_j) / (x_i - x_j)
        l += y_i * l_i
    return l
def take_diff(func, x, n):
    new_func = func
    for _ in range(n):
    new_func = sp.diff(new_func, x)
    return new_func
def omega(a, b, step, x):
    res = 1
    while round(a, 1) <= b:
        res *= (x - a)
        a += step
    return res
def main():
    x = sp.symbols('x')
    n = 4
    k = 1
    m = 3
    a = 0.5
    b = 1.0
    step = (b - a) / 3

    points = values(a, b, step)
    print(points)

    L = lagrange_polynomial(points, x)

    print(f"Многочлен Лагранжа: {L}")

    L_diff = sp.diff(L, x)

    f = x ** 2 - sp.sin(x)

    d = take_diff(f, x, n)
```

```
    df = take_diff(f, x, n)
    r_1 = d.subs(x, 0.5) - L_diff.subs(x, 0.5)
    r_min = (df.subs(x, a) / math.factorial(4)) * omega(a, b, step,
x)

    r_max = df.subs(x, b) / math.factorial(4) * omega(a, b, step,
x)

    print(f"Производная многочлена Лагранжа: {L_diff}")
    print(L.subs(x, 0.5))
    print(func(0.5).evalf())
    print('-')
    print(L_diff.subs(x, 0.5))
    print(d.subs(x, 0.5))
    print('-')
    print(r_1)
    print(r_min.subs(x, a))
    print(r_max.subs(x, b))

if __name__ == '__main__':
    main()
```