Отчет по лабораторной работе № 3
Вариант № 3

Винницкая Дина Сергеевна

Группа: Б9122-02-03-01сцт

Цель работы

1. Реализовать формулу дифференцирования с учётом равномерной сетки для порядка первой производной;
2. Получить значения $\min R$ и $\max R$ для остаточного члена R;
3. Проверить выполнение неравентсва $\min R < R(x_m) < \max(R)$ где x_m– заданный узел
4. Сделать вывод по проделанной работе.

Входные данные:

1. **Функция:** $y = x^2 + \ln(x) - 4$
2. **Отрезок** $[1.5, 2.0]$
3. $n = 3; \quad k = 1; \quad m = 2;$

Ход работы:

1 Вывод первой производной по методу Лагранжа

Из указанной в методической книжке таблицы, необходимо вычислислить первую производную:

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

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

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

2 Используемые библиотеки

Для реализации необходимой программы используются следующие библиотеки языка Python:

- **sympy** - библиотека **sympy** представляет собой мощный символьный математический пакет для **Python**. Она способна обрабатывать символьные выражения, уравнения, и действия, что делает ее полезным инструментом в области научных вычислений, анализа данных и математического моделирования.
  - **Функциональность:** библиотека **sympy** позволяет выполнять различные операции с символами, такие как дифференцирование, интегрирование, решение уравнений и многое другое, что делает ее важным ресурсом для работы с математическими вычислениями в **Python**.

- **math** - 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.
  - **Функциональность:** Библиотека math предоставляет широкий спектр математических функций, которые могут использоваться для решения различных задач, как в научных и инженерных вычислениях, так и в разработке программного обеспечения.

```
import sympy as sp
import math
```

3 Инициализация входных данных

Начиная реализацию алгоритма известна непосредственная функция$y = x^2 + \ln(x) - 4$ и отрезок $[1.5, 2.0]$

```
    x = sp.symbols('x')
    n = 3
    k = 1
    m = 2
    a = 1.5
    b = 2.0
    step = (b - a) / 3
    points = values(a, b, step)
    L = lagrange_polynomial(points, x)
```

```
def func(x):
    return x ** 2 + sp.log(x) - 4
```

4 Реализация основного алгоритма

Создание таблицы значений функции

```
def values(a, b, step):
    table = []
    x = a

    while x <= b:
        table.append((x, func(x)))
        x += step

    return table
```

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

Таблица значений функции

x	y(x)
1.5	-1.34453489189184
1.6666666666666667	-0.711396598456231
1.8333333333333335	-0.0327530853185727
2.0	0.693147180559945

Вычисление многочлена Лагранжа

```python
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
```

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

Вывод

$-1.34453489189184 \times (4.0-2.0x) \times (5.5-3.0x) \times (10.0-6.0x) - 0.711396598456231 \times (6.0-3.0x) \times (11.0-6.0x) \times (6.0x-9.0) - 0.0327530853185727 \times (12.0-6.00000000000001x) \times (3.0x-4.5) \times (6.0x-10.0) + 0.693147180559945 \times (2.0x-3.0) \times (3.0x-5.0) \times (6.00000000000001x-11.0)$

Взятие n-ой производной функции

```python
def take_diff(func, x, n):
    new_func = func
    for _ in range(n):
        new_func = sp.diff(new_func, x)
    return new_func
```

- Функция **take_diff(func, x, n)** вычисляет n-ую производную заданной функции по переменной x. Она использует библиотеку **SymPy** для символьных вычислений.

Вывод

$8.06720935135101 \times (4.0-2.0x) \times (5.5-3.0x) + 4.0336046756755 \times (4.0-2.0x) \times (10.0-6.0x) + 2.68906978378367 \times (5.5-3.0x) \times (10.0-6.0x) - 4.26837959073738 \times (6.0-3.0x) \times (11.0-6.0x) + 4.26837959073738 \times (6.0-3.0x) \times (6.0x-9.0) + 2.13418979536869 \times (11.0-6.0x) \times (6.0x-9.0) - 0.196518511911436 \times (12.0-6.00000000000001x) \times (3.0x-4.5) - 0.098259255955718 \times (12.0-6.00000000000001x) \times (6.0x-10.0) + 4.15888308335968 \times (2.0x-3.0) \times (3.0x-5.0) + 2.07944154167984 \times (2.0x-3.0) \times (6.00000000000001x-11.0) + 0.196518511911436 \times (3.0x-4.5) \times (6.0x-10.0) + 1.38629436111989 \times (3.0x-5.0) \times (6.00000000000001x-11.0)$

Вычисление множителя omega

```python
def omega(a, b, step, x):
    res = 1
    while round(a, 2) <= b:
        res *= (x - a)
        a += step
    return res
```

- Функция **omega(a, b, step, x)** вычисляет множитель для разделенной разности в многочлене Лагранжа. Этот множитель используется при вычислении многочлена Лагранжа для определенных точек.

Основная функция

- В основной функции **main()** происходит организация последовательности вычислений и анализа результатов. Это включает вычисление многочлена Лагранжа, его производной, вычисление исходной функции, вычисление n-ой производной и анализ погрешности.

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

    L = lagrange_polynomial(points, x)

    print(f"The Lagrange polynomial: {L}")

    L_diff = sp.diff(L, x)
    f = x ** 2 + sp.log(x) - 4

    d = take_diff(f, x, n)
    df = take_diff(f, x, n)
    r_1 = d.subs(x, 1.5) - L_diff.subs(x, 1.5)

    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"The derivative of the Lagrange polynomial: {L_diff}")
    print(L.subs(x, 1.5))

    print(func(1.5).evalf())
    print('=============================')

    print(L_diff.subs(x, 1.5))
    print(d.subs(x, 1.5))

    print('=============================')

    print(r_1)
    print(r_min.subs(x, a))
    print(r_max.subs(x, b))
```

5 Остаточный член

Заметим, что остаточный член по определению является разницей между точной функцией и её приближением
В данном случае он равен $= 0.0002414081$

```
Разница:    0.0002414081
```

6 Проверка неравенства

Необходимо проверить выполнение неравентсва $\min(R) < R(x_m) < \max(R)$ где x_m- заданный узел. Известно, что:

$$R = 0.0002414081$$

$$\min(R) = -0.0001446759$$

$$\max(R) = -0.0004572474$$

$$-0.0001446759 < 0.000241408 < -0.0004572474 \Rightarrow \text{не выполняется}$$

7 Вывод

По результатам выполнения лабораторной работы можно сделать следующие выводы:

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.

Так же следует сказать о том, что Лагранж выдает довольно точный результат

```
Лагранж:    4.211879804
Значение производной функции:   4.2121212121
R   0.0002414081

Min -0.0001446759
Max -0.0004572474
```