Отчёт по лабораторной работе №3. Поповкин Артемий. Б9122-02.03.01 СЦТ

Вариант-15

Цель

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

Условия

Промежуток: $[0.1,\ 0.6]$

$f(x) = 2x - \cos(x)$

$\text{Количество точек} = 5$

$\text{Номер точки} = 5$

$\text{Степень производной} = 1$

Решение

Вычисление первой производной по Лагранжу

Вывод формулы

По условию, мне требуется вычислить лишь первую производную.

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

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

Произведём замену при условии равномерности сетки:

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

Реализация в коде

```
def lagrange(num_point: int, points: list[tuple[float, float]], step: float) -
> float:
    """
    Неньютоновская реализация производной от полинома Лагранжа первой степени

    :param points: Список точек
    :type points: list[tuple[float, float]]
    :param num_point: Номер точки
    :type num_point: int
    :param step: Шаг сетки
    :type step: float

    :return: Значение полинома Лагранжа в точке bp
    :rtype: float

    """
    count_points = len(points) - 1
    result = 0
    for i, point in enumerate(points):
        point_mult = point[1]

        def diff_mult_part(a: int, b: int) -> float:
            sub_mult = 1
            for j in range(a, b):
                sub_mult *= (i - j)
            return sub_mult

        diff_mult = diff_mult_part(0, i - 1 + 1) * diff_mult_part(i + 1, count
_points + 1)

        def grid_mult_part(a: int, b: int) -> float:
            alt_mult = 0
            for j in range(a, b):
                sub_mult = 1
                for j1 in range(0, min(i, j) - 1 + 1):
                    sub_mult *= (num_point - j1)
                for j1 in range(min(i, j) + 1, max(i, j) - 1 + 1):
                    sub_mult *= (num_point - j1)
                for j1 in range(max(i, j) + 1, count_points + 1):
```

```
                    sub_mult *= (num_point - j1)
                alt_mult += sub_mult
            return alt_mult

        grid_mult = grid_mult_part(0, i - 1 + 1) + grid_mult_part(i + 1, count
_points + 1)

        result += point_mult / diff_mult * grid_mult

    return result / step
```

Вычисление первой производной по функции ошибки

Вывод формулы

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x)$$

Где $\omega_{n+1}(x) = \prod_{j=0}^{n}(x - x_j)$

$$R'_n(x) = \frac{f^{(n+2)}(\xi)}{(n+2)!}\omega_{n+1}(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega'_{n+1}(x)$$

Произведём замену при условии равномерности сетки:

$$R'_n(x_m) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \cdot h^n \sum_{j=0}^{n} \prod_{j_1=0;\ j_1 \neq j}^{n} (m - j)$$

Реализация в коде

```
def teor_error(num_point: int, step: float,
               count_points: int, rng: tuple[float, float],
               function) -> tuple[float, float]:
    """
    Получение первой производной от теоретической ошибки.
    Теоретическая ошибка берётся от максимума и минимума функции.

    :param function: Функция
    :type function: function
    :param num_point: Номер точки
    :type num_point: int
    :param step: Шаг сетки
    :type step: float
    :param count_points: Количество точек
    :param rng: Кортеж границ
```

```
    :return: Теоретическая ошибка
    :rtype: float

    """
    pts = function(linspace(*rng, num=10 ** 3), count_points + 1)
    fct = factorial(count_points + 1)

    result: tuple[float, float] = min(pts), max(pts)

    def everything_else_done(value: float) -> float:
        sub_res = 0
        for j in range(count_points + 1):
            sub_mult = 1
            for j1 in range(count_points + 1):
                if j1 != j:
                    sub_mult *= (num_point - j1)
            sub_res += sub_mult
        return value * sub_res * step ** count_points / fct

    return map(everything_else_done, result)
```

Главный цикл программы

Для удобства берём первые 10 цифр после запятой.

```
koef_round = 10
k, count_pts, index_point = 1, 5, 5
range_graph = (0.1, 0.6)
step_grid = (range_graph[1] - range_graph[0]) / count_pts
mass_points = [(pt, func(pt)) for pt in linspace(*range_graph, count_pts + 1)]

res_lagrange = lagrange(index_point, mass_points, step_grid)
res_func = func(mass_points[index_point][0], k)

min_ter, max_ter = map(lambda num: round(num, koef_round),
                                             teor_error(index_point, step_grid, c

print(f"Лагранж:\t{round(res_lagrange, koef_round)}",
      f"Значение производной функции:\t{round(res_func, koef_round)}",
      f"Разница:\t{round(abs(res_lagrange - res_func), koef_round)}",
      sep='\n')
print()
print(f"Минимальная ошибка:\t{min_ter}",
      f"Максимальная ошибка:\t{max_ter}",
```

```
      sep='\n')
print()
print(f"Попадает ли ошибка в промежуток?:\t{min_ter <
                                                     abs(res_lagrange - res_func)
                                                      < max_ter}",
      sep='\n')
```

Результат

Лагранж	Значение производной функции	Разница	Минимальная ошибка	Максимальная ошибка	Попадает ли ошибка в промежуток
2.5646409323	2.5646424734	1.5411e-06	1.3756e-06	1.6583e-06	Да

Как видим, Лагранж дал достаточно точный результат, дающих ошибку, находящуюся в пределе минимума и максимума ошибки.

Ссылка на актуальную версию кода:

https://github.com/Jrol123/CalcMath/blob/main/Differenciate (Lagrange)/main.py