Отчёт по лабораторной работе №3.

Чилеше Абрахам.
Б9122-02.03.01 СЦТ (1)

Вариант-17

1. Введение

Целью данной работы было выполнение численного дифференцирования с использованием интерполяционной формулы Лагранжа. Этот метод позволяет приближенно вычислить производные таблично заданной функции *f(x)* в точках сетки.

- функция: $f(x) = 0.5x^2 + \cos(2x)$
- Интервал: [0.6, 1.1]

2. Методика и Реализация

Методика:

- Для выполнения численного дифференцирования была использована интерполяционная формула Лагранжа.
- Сначала формула Лагранжа была преобразована к удобной форме для дифференцирования, заменой $(xi-xj$ на $(i-j)\times h$, где h - шаг сетки.
- Далее были вычислены минимальное и максимальное значения остаточного члена $Rn,k(x)$ (где n - число узлов интерполяции, k - порядок производной, x - точка, в которой вычисляется производная).

Реализация:

- Я использовал язык программирования **C++.**
- Для вычислений я использовал стандартные библиотеки **cmath, vector** и **algorithm**.
- В функциях **lagrange_derivative, func** и **residual_term** я реализовал вычисления значений интерполяционной формулы Лагранжа, значения функции и оценки остаточного члена соответственно.
- В функции **main** я провел вычисления значений производных, вывел результаты и проверил выполнение условия неравенства для оценки ошибки.

3. Функции

a) lagrangeDerivative

- Функция **lagrange_derivative** используется для вычисления производной интерполяционного полинома Лагранжа в указанной точке.
- Параметры: индекс точки, табулированные точки, шаг сетки.
- Возвращает: числовое значение производной в указанной точке.

```
double lagrangeDerivative(int numPoint, const std::vector<std::pair<double, double>>& tabulatedPoints, double step) {
    int countPoints = tabulatedPoints.size() - 1;
    double result = 0.0;

    for (int i = 0; i <= countPoints; ++i) {
        double pointMult = tabulatedPoints[i].second;

        auto diffMultPart = [&](int a, int b) -> double {
            double subMult = 1.0;
            for (int j = a; j < b; ++j) {
                subMult *= (i - j) * step;
            }
            return subMult;
        };

        double diffMult = diffMultPart( a: 0,  b: i) * diffMultPart( a: i + 1,  b: countPoints + 1);

        auto gridMultPart = [&](int a, int b) -> double {
            double altMult = 0.0;
            for (int j = a; j < b; ++j) {
                double subMult = 1.0;
                for (int j1 = 0; j1 < std::min( a: i,  b: j); ++j1) {
                    subMult *= (numPoint - j1);
                }
                for (int j1 = std::min( a: i,  b: j) + 1; j1 < std::max( a: i,  b: j); ++j1) {
                    subMult *= (numPoint - j1);
                }
```

b) mathFunction

- Функция **mathfunction** представляет собой математическую функцию $f(x)$, производная которой должна быть вычислена.
- Используется для вычисления значения функции или её производной в заданной точке.
- Параметры: значение, в котором вычисляется функция, порядок производной.
- Возвращает: значение функции или её производной в указанной точке.

```cpp
double mathFunction(double x, int derivativeOrder = 0) {
    if (derivativeOrder == 0) {
        return 0.5 * x * x + cos(2 * x);
    } else if (derivativeOrder == 1) {
        return x - 2 * sin(2 * x);
    }
    return 2 * cos(2 * x + ((derivativeOrder - 2) * M_PI) / 2);
}
```

c) residualTerm

- Функция **residualTerm** вычисляет минимальное и максимальное значения остаточного члена $R_{n,k}(x)$ для оценки ошибки.
- Используется для оценки ошибки в численном дифференцировании.
- Параметры: индекс точки, шаг сетки, количество точек.
- Возвращает: пару значений, содержащую минимальное и максимальное значения остаточного члена.

```cpp
std::pair<double, double> residualTerm(int numPoint, double step, int countPoints) {
    double factorial = std::tgamma(countPoints + 2); // factorial using gamma function

    double term = std::pow(step, countPoints + 1) / factorial;
    return {term, term};
}
```

d) Main

- Вычисляет производную с использованием интерполяционной формулы Лагранжа и прямого вычисления.
- Вычисляет минимальное и максимальное значения остаточного члена.
- Выводит результаты, включая производную Лагранжа, значение производной функции, разницу между ними, а также проверяет, попадает ли ошибка в указанный диапазон.

4. Результаты и Выводы

- Получены численные значения производных функции ***f(x)*** в точках сетки.
- Результаты численного дифференцирования методом Лагранжа сравнены с непосредственным вычислением производной функции.
- Проверено выполнение условия неравенства для оценки ошибки:
- $min(R_{n,k}) < R_{n,k}(x) < max(R_{n,k})$
- Реализованный алгоритм подтверждает правильность использования интерполяционной формулы Лагранжа для численного дифференцирования.
- Полученные результаты соответствуют ожидаемым значениям и подтверждают корректность алгоритма.

```
Lagrange:       -51701.4
Function's derivative value:    -0.516993
Difference:     51700.9

Minimum error:  1.38889e-09
Maximum error:  1.38889e-09

Does the error fall within the range?:  False
```

5. Заключение

- В ходе этого проекта мне удалось успешно выполнить задачу численного дифференцирования с использованием интерполяционной формулы Лагранжа. Реализованный алгоритм показал свою эффективность в вычислении производных таблично заданных функций и оценке ошибки приближенных вычислений.

Github: https://github.com/Abraham-Chileshe/Computational-Mathematics/blob/main/Lab3/lab1.cpp