**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4**

**ЧИСЛЕННОЕ ИНТЕРПОЛИРОВАНИЕ И ДИФФЕРЕНЦИРОВАНИЕ**

**(Вариант 10)**

*Выполнил студент 3 курса МОиАИС*

*Соколов Арсений*

**Задание 1. Интерполирование**

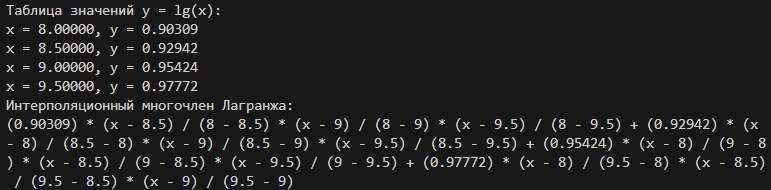
Известна функция . Заполнить таблицу значениями yk для указанных *xk*, с точностью 10-4. Составить по таблице интерполяционный многочлен Лагранжа. Привести его окончательный вид. Вычислить значение функции в заданной точке аналитически и с помощью многочлена Лагранжа. Оценить погрешность формулы Лагранжа и абсолютную погрешность вычислений.

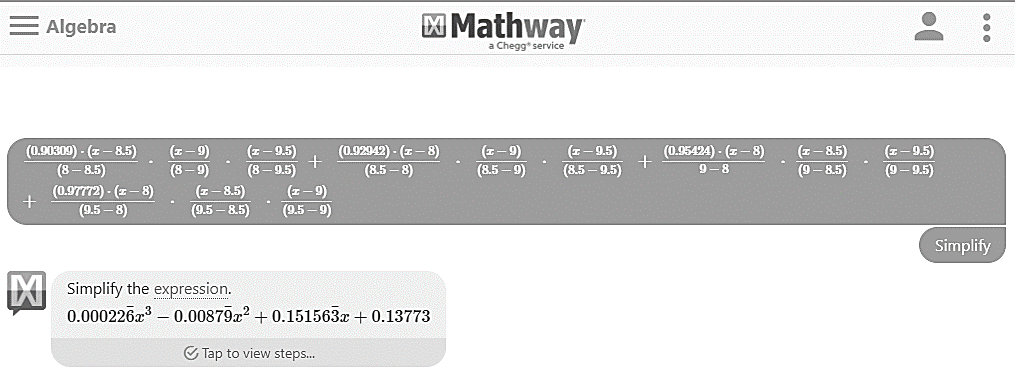
**Дано:**

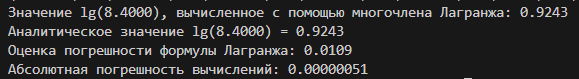
,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 8,0 | 8,5 | 9,0 | 9,5 |

**Решение:**

****



****

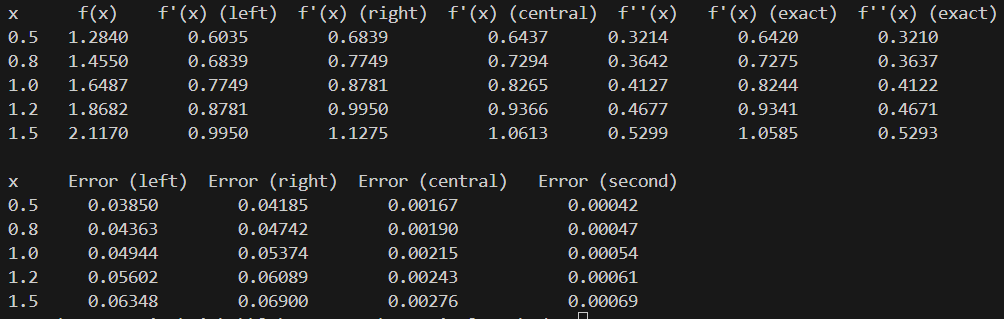
**Задание 2. Дифференцирование**

Вычислить таблицу на отрезке [a,b] на равномерной сетке (5 узлов), и в этих узлах найти значение первой производной функции по формулам 1-го (левая и правая) и 2-го порядка точности (центральная) и значение второй производной по формулам 2-го порядка точности, где это возможно Во всех точках найти точные значения производных. Оценить погрешность. Результаты свести в таблицу. Точность – 4 значащих цифры.

**Дано:**

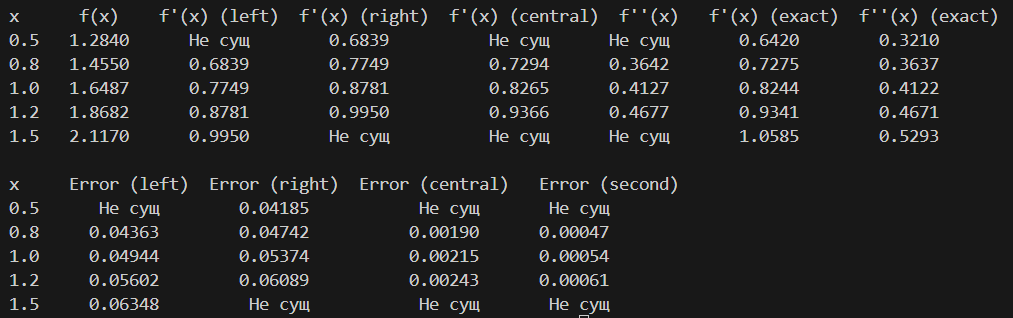
, *а=*0,5, *b=*1,5

**Решение:**

****

**Как вы находите левую, центральную и вторую производную в крайней левой точке? Аналогично для крайней правой.**

**Исправлено:**

****

(добавлена проверка на выход за пределы отрезка см код в Приложении 2)

**ПРИЛОЖЕНИЕ 1**

***Программа вычисления значений первого задания***

#include <iostream>

#include <iomanip>

#include <vector>

#include <cmath>

// Погрешность формулы Лагранжа

double errorEstimate = 0;

// Функция для вычисления значения lg(x)

double lg(double x) {

    return log10(x);

}

// Структура для хранения пар x и y = lg(x)

struct Point {

    double x;

    double y;

};

// Функция для вычисления и вывода интерполяционного многочлена Лагранжа

double lagrangeInterpolation(const std::vector<Point>& points, double x) {

    std::cout << std::resetiosflags(std::ios\_base::fmtflags(std::ios\_base::fixed | std::ios\_base::showpoint));

    std::cout << "Интерполяционный многочлен Лагранжа:" << std::endl;

    double result = 0;

    for (size\_t i = 0; i < points.size(); ++i) {

        if (i != 0) {

            std::cout << " + ";

        }

        double term = points[i].y;

        std::cout << "(" << points[i].y << ")";

        for (size\_t j = 0; j < points.size(); ++j) {

            if (j != i) {

                term \*= (x - points[j].x) / (points[i].x - points[j].x);

                std::cout << " \* (x - " << points[j].x << ") / (" << points[i].x << " - " << points[j].x << ")";

            }

        }

        errorEstimate += std::abs(x - points[i].x);

        result += term;

    }

    std::cout << std::endl;

    return result;

}

int main() {

    // Заданные значения xk

    std::vector<double> xk = {8.0, 8.5, 9.0, 9.5};

    // Заполнение таблицы значениями yk = lg(xk)

    std::vector<Point> table;

    for (double x : xk) {

        table.push\_back({x, lg(x)});

    }

    // Вывод таблицы

    std::cout << "Таблица значений y = lg(x):" << std::endl;

    for (const auto& point : table) {

        std::cout << std::fixed << std::setprecision(5) << std::showpoint;

        std::cout << "x = " << point.x << ", y = " << point.y << std::endl;

    }

    // Вычисление значения в заданной точке

    double x = 8.4;

    double analyticalValue = lg(x);

    double interpolatedValue = lagrangeInterpolation(table, x);

    // Вывод результатов

    std::cout << std::fixed << std::setprecision(4) << std::showpoint;

    std::cout << "Значение lg(" << x << "), вычисленное с помощью многочлена Лагранжа: " << interpolatedValue << std::endl;

    std::cout << "Аналитическое значение lg(" << x << ") = " << analyticalValue << std::endl;

    std::cout << "Оценка погрешности формулы Лагранжа: " << (1/x) \* errorEstimate/(2\*3\*4) << std::endl; // максимальное значение имеет (lnx(x))' = 1/x

    std::cout << std::fixed << std::setprecision(8) << std::showpoint;

    std::cout << "Абсолютная погрешность вычислений: " << std::abs(interpolatedValue - analyticalValue) << std::endl;

    return 0;

}

**ПРИЛОЖЕНИЕ 2**

***Программа вычисления значений второго задания***

#include <iostream>

#include <iomanip>

#include <cmath>

// Функция f(x) = e^(x/2)

double f(double x) {

    return exp(x / 2);

}

// Первая производная функции f(x)

double firstDerivative(double x) {

    return 0.5 \* exp(x / 2);

}

// Вторая производная функции f(x)

double secondDerivative(double x) {

    return 0.25 \* exp(x / 2);

}

int main() {

    const int numNodes = 5;

    const double start = 0.5;

    const double end = 1.5;

    const double h = (end - start) / (numNodes - 1);

    // Вывод таблицы значений функции и производных;

    std::cout << "x      f(x)    f'(x) (left)  f'(x) (right)  f'(x) (central)  f''(x)   f'(x) (exact)  f''(x) (exact)" << std::endl;

    for (int i = 0; i < numNodes; ++i) {

        double x = start + i \* h;

        double exactFirstDerivative = firstDerivative(x);

        double exactSecondDerivative = secondDerivative(x);

        std::cout << x << std::fixed << std::setprecision(4) << std::showpoint << "   " << f(x) << "      ";

        // Вычисление производных

        if (x - h < start) {

            std::cout << "Не сущ        ";

        } else {

            double leftFirstDerivative = (f(x) - f(x - h)) / h;

            std::cout << leftFirstDerivative << "        ";

        }

        if (x + h > end) {

            std::cout << "Не сущ          ";

        } else {

            double rightFirstDerivative = (f(x + h) - f(x)) / h;

            std::cout <<  rightFirstDerivative << "          ";

        }

        if (x - h < start || x + h > end) {

            std::cout << "Не сущ      ";

        } else {

            double centralFirstDerivative = (f(x + h) - f(x - h)) / (2 \* h);

            std::cout << centralFirstDerivative << "      ";

        }

        if (x - h < start || x + h > end) {

            std::cout << "Не сущ       ";

        } else {

            double secondDerivative = (f(x - h) - 2 \* f(x) + f(x + h)) / (h \* h);

            std::cout << secondDerivative << "       ";

        }

        // Вывод значений

        std::cout << exactFirstDerivative << "        " << exactSecondDerivative << std::setprecision(1) << std::endl;

    }

    // Таблица погрешностей

    std::cout << std::endl;

    std::cout << "x     Error (left)  Error (right)  Error (central)   Error (second)" << std::endl;;

    for (int i = 0; i < numNodes; ++i) {

        double x = start + i \* h;

        double exactFirstDerivative = firstDerivative(x);

        double exactSecondDerivative = secondDerivative(x);

        std::cout << x << std::setprecision(5) << "     ";

        // Вычисление производных

        if (x - h < start) {

            std::cout << " Не сущ        ";

        }

        else {

            double leftFirstDerivative = (f(x) - f(x - h)) / h;

            std::cout << std::abs(exactFirstDerivative - leftFirstDerivative) << "        ";

        }

        if (x + h > end) {

            std::cout << " Не сущ          ";

        }

        else {

            double rightFirstDerivative = (f(x + h) - f(x)) / h;

            std::cout << std::abs(exactFirstDerivative - rightFirstDerivative) << "          ";

        }

        if (x - h < start || x + h > end) {

            std::cout << " Не сущ      ";

        }

        else {

            double centralFirstDerivative = (f(x + h) - f(x - h)) / (2 \* h);

            std::cout << std::abs(exactFirstDerivative - centralFirstDerivative) << "      ";

        }

        if (x - h < start || x + h > end) {

            std::cout << " Не сущ       ";

        }

        else {

            double secondDerivative = (f(x - h) - 2 \* f(x) + f(x + h)) / (h \* h);

            std::cout << std::abs(exactSecondDerivative - secondDerivative) << "       ";

        }

        // Вывод погрешностей

        std::cout << std::setprecision(1) << std::endl;

    }

    return 0;

}