Практическое занятие №1

Армбристер Никита Владиславович

Группа ПМИ-31

Вариант 1

Цель: Сформировать практические навыки интерполяции табличных функций и численного дифференцирования.

Задание:

1. Разработать подпрограмму генерации регулярных и адаптивных сеточных разбиений произвольного от резка [a,b] в зависимости от числа сегментов разбиения и величины коэффициента разрядки r (Регулярная равномерная сетка и Адаптивная сетка: каждый последующий шаг hi отличается от предыдущего в r раз)

Ссылка на репозиторий в GitHub:
https://github.com/Gribnik24/NumericalMethods-PracticalTasks/tree/main/Task%20%232.1.%20Cubic%20spline%20and%20finite%20difference

2. Разработать класс, реализующий интерфейс кубического интерполяционного сплайна.

3. Проведите исследование сплайна на вложенных сетках. Определите на отрезке [a, b] любую непрерывную неполиномиальную функцию f(x). Задайте шаг h и постройте равномерное сеточное разбиение отрезка [a,b]. Постройте табличную функцию по значениям f(x) в узлах сетки. Получите таблицу значений сплайна и его двух первых производных в точках, которые НЕ совпадают с узловыми (не менее 10). Повторить данные исследования на сетках с шагом h/2 и h/4. Оцените точность сплайн-аппроксимации функции f(x) и ее производных в зависимости от шага.

4. В центральной точке отрезка из вашего варианта за дания вычислите значение её первой производной при использовании конечных разностей. Предложите оптимальный вариант с заданной точностью ε.

5. С помощью AI-ассистента DeepSeek решите предыдущую задачу. Какие явные недостатки Вы можете обнаружить? Промт. Оптимальный метод численного дифференцирования для табличной функции | x = 0, 0.1, 0.2 | f = 0, 0.01, 0.04 с двумя верными десятичными знаками?

Данные варианта 1:

a	b	ε
0.08	0.32	0.001

Решение:

Для решения данной лабораторной работы был разработан проект на языке C++. Структура проекта представлена ниже

```
├── Заголовки
│   └── headers.h
├── Исходные
│   ├── grid_generator.cpp
│   ├── distribution_functions.cpp
│   ├── cubric_spline.cpp
│   ├── finite_difference.cpp
│   └── main.cpp
```

Рассмотрим каждую составляющую работы подробнее

- **headers.h**

  Заголовочный файл, в котором обозначены классы, отвечающие за реализацию генератора сеток, кубического сплайна и методов конечной разности.

  Генерация сетки может происходить по двум сценариям:

  Первый предполагает разбиение на равные по расстоянию сегменты, соответственно функция, которая реализует это разделение, принимает на вход границы a, b и необходимое количество сегментов.

  Второй помимо прошлых аргументов принимает на вход параметр r – коэффициент разрядки, который выполняет функцию знаменателя геометрической прогрессии

```
// Класс генератора сеток
class GridGenerator {
public:
    /**
     * a - начало отрезка
     * b - конец отрезка
     * nSegments - количество сегментов
     * r - коэффициент разрядки
     */

    // Генерация регулярной равномерной сетки на отрезке [a, b]. Возвращает вектор узлов сетки
    static std::vector<double> generateRegularGrid(double a, double b, int nSegments);

    // Генерация адаптивной сетки с геометрической прогрессией шагов. Возвращает вектор узлов адаптивной сетки
    static std::vector<double> generateAdaptiveGrid(double a, double b, int nSegments, double r);
};
```

  Структура кубического сплайна состоит из векторов x, y – узлов и значений функции в узлах соответственно, а также набора коэффициентов сплайна.

  Набор методов включает в себя: реализацию сплайна, а также вычисление значения сплайна в точке и его производных первого и второго порядка.

```cpp
// Класс кубического сплайна
class CubicSpline {
private:
    std::vector<double> x;          // Узлы сетки
    std::vector<double> y;          // Значения функции в узлах
    std::vector<double> a, b, c, d; // Коэффициенты сплайна

    // Подсчет коэффициентов сплайна
    void calculateCoefficients();

    // Поиск сегмента, в который попадает заданная точка
    int findSegment(double x_point) const;

public:
    CubicSpline() = default;

    /**
     * x_nodes - узлы сетки
     * y_nodes - значения функции в узлах
     * x_point - точка вычисления
     */

    // Инициализация сплайна по табличной функции
    void initialize(const std::vector<double>& x_nodes, const std::vector<double>& y_nodes);

    // Вычисление значения сплайна в точке
    double evaluate(double x_point) const;

    // Вычисление первой производной сплайна
    double derivative1(double x_point) const;

    // Вычисление второй производной сплайна
    double derivative2(double x_point) const;
};
```

В свою очередь в серии методов конечной разности реализованы двухшаговые левосторонний и правосторонний методы, а также трехшаговый центральный с двумя уровнями точности

```cpp
// Класс методов конечной разности
class FiniteDifference {
public:
    /**
     * f - функция
     * x - точка вычисления
     * h - шаг
     * epsilon - требуемая точность
     */

    // Левосторонняя разность
    static double forwardDifference(std::function<double(double)> f, double x, double h);

    // Правосторонняя разность
    static double backwardDifference(std::function<double(double)> f, double x, double h);

    // Центральная разность O(h^2)
    static double centralDifference(std::function<double(double)> f, double x, double h);

    // Центральная разность O(h^4)
    static double centralDifference4(std::function<double(double)> f, double x, double h);
};
```

- **grid_generator.cpp**

Генератор регулярной сетки разбивает область на равные промежутки. Сначала создается пустой вектор. Далее высчитывается величина шага. В конечном итоге вектор заполняется значениями по принципу: {начальная точка} + {номер элемента}*{шаг}

```
std::vector<double> GridGenerator::generateRegularGrid(double a, double b, int nSegments) {
    std::vector<double> grid;

    double h = (b - a) / nSegments;
    for (int i = 0; i <= nSegments; ++i) {
        grid.push_back(a + i * h);
    }

    return grid;
}
```

В свою очередь построение адаптивной сетки работает от идеи геометрической прогрессии. Сначала находится ее сумма для того, чтобы задать первый шаг в разбиении. Затем этот шаг умножается на значение r с каждой итерацией

```
std::vector<double> GridGenerator::generateAdaptiveGrid(double a, double b, int nSegments, double r) {
    std::vector<double> grid;
    grid.push_back(a);

    // Вычисляем сумму геометрической прогрессии
    double sum;
    if (std::abs(r - 1.0) > 1e-10) {
        sum = (1.0 - std::pow(r, nSegments)) / (1.0 - r);
    }
    else {
        sum = nSegments;
    }

    double totalLength = b - a;
    double firstStep = totalLength / sum;

    double current = a;
    for (int i = 0; i < nSegments; ++i) {
        double step = firstStep * std::pow(r, i);
        current += step;
        grid.push_back(current);
    }

    // Корректируем последнюю точку точно в b
    grid.back() = b;

    return grid;
}
```

- **cubric_spline.cpp**
  Функция initialize задает элементы структуры кубического сплайна – значения узлов и значения функции в этих узлах, а также вызывает функцию calculateCoefficients, которая отвечает за нахождение коэффициентов кубического сплайна

```
void CubicSpline::initialize(const std::vector<double>& x_nodes, const std::vector<double>& y_nodes) {
    x = x_nodes;
    y = y_nodes;
    calculateCoefficients();
}
```

Функция calculateCoefficients – это реализация алгоритма построения кубического сплайна методом прогонки. Сначала инициализируются коэффициенты

1) Затем в цикле находится длина для каждого сегмента

2) Задается граничное условие c0 = 0

```
void CubicSpline::calculateCoefficients() {
    int n = x.size() - 1;
    a.resize(n + 1);
    b.resize(n + 1);
    c.resize(n + 1);
    d.resize(n + 1);

    // 1. Вычисление длин сегментов h_k (индексация с 0)
    std::vector<double> h(n);
    for (int k = 0; k < n; ++k) {
        h[k] = x[k + 1] - x[k];
    }

    // 2. Граничные условия: c0 = 0
    c[0] = 0.0;
```

3) Строится система для c1, …, c_n. Затем она решается

```
    // 3. Построение системы для c1...c_{n-1}
    if (n >= 2) {
        int system_size = n - 1;
        std::vector<double> alpha(system_size), beta(system_size),
            gamma(system_size), phi(system_size);

        // Заполнение коэффициентов системы
        for (int k = 0; k < system_size; ++k) {
            beta[k] = 2.0 * (h[k] + h[k + 1]);
            gamma[k] = (k < system_size - 1) ? h[k + 1] : 0.0;
            alpha[k] = (k > 0) ? h[k] : 0.0;

            phi[k] = 3.0 * ((y[k + 2] - y[k + 1]) / h[k + 1] - (y[k + 1] - y[k]) / h[k]);
        }

        // Решение системы
        std::vector<double> mu(system_size), z(system_size);
        if (system_size > 0) {
            mu[0] = gamma[0] / beta[0];
            z[0] = phi[0] / beta[0];

            for (int k = 1; k < system_size; ++k) {
                double denominator = beta[k] - alpha[k] * mu[k - 1];
                mu[k] = gamma[k] / denominator;
                z[k] = (phi[k] - alpha[k] * z[k - 1]) / denominator;
            }

            c[system_size] = z[system_size - 1];
            for (int k = system_size - 1; k >= 0; --k) {
                c[k + 1] = z[k] - mu[k] * c[k + 2];
            }
        }
    }
```

4) По формулам, имея значения c и h, находим коэффициенты a, b, d

```
    // 4. Вычисление коэффициентов
    for (int k = 0; k < n; ++k) {
        a[k] = y[k];

        if (k < n - 1) {
            b[k] = (y[k + 1] - y[k]) / h[k] - (2.0 * c[k] + c[k + 1]) * h[k] / 3.0;
            d[k] = (c[k + 1] - c[k]) / (3.0 * h[k]);
        }
        else {
            b[k] = (y[k + 1] - y[k]) / h[k] - (2.0 * c[k]) * h[k] / 3.0;
            d[k] = -c[k] / (3.0 * h[k]);
        }
    }

    a[n] = y[n-1];
}
```

Функция findSegment отвечает за поиск сегмента, в котором находится точка x_point. std::upper_bound находит первый узел, который **больше** чем x_point, а std::distance вычисляет индекс этого узла. Функция min в выводе также предупреждает о выходе за границы

```
int CubicSpline::findSegment(double x_point) const {
    auto it = std::upper_bound(x.begin(), x.end(), x_point);
    return std::min(static_cast<int>(std::distance(x.begin(), it)) - 1, static_cast<int>(x.size()) - 2);
}
```

Функция evaluate, derivative1, derivative2 вычисляют значение сплайна и его производных в заданной точке

```
double CubicSpline::evaluate(double x_point) const {
    int i = findSegment(x_point);
    double dx = x_point - x[i];
    return a[i] + b[i] * dx + c[i] * std::pow(dx, 2) + d[i] * std::pow(dx, 3);
}

double CubicSpline::derivative1(double x_point) const {
    int i = findSegment(x_point);
    double dx = x_point - x[i];
    return b[i] + 2.0 * c[i] * dx + 3.0 * d[i] * dx * dx;
}

double CubicSpline::derivative2(double x_point) const {
    int i = findSegment(x_point);
    double dx = x_point - x[i];
    return 2.0 * c[i] + 6.0 * d[i] * dx;
}
```

- **finite_difference.cpp**
  Набор функций Difference реализуют левосторонний, правосторонний и центральные шаблоны разности

```cpp
double FiniteDifference::forwardDifference(std::function<double(double)> f, double x, double h) {
    return (f(x + h) - f(x)) / h;
}

double FiniteDifference::backwardDifference(std::function<double(double)> f, double x, double h) {
    return (f(x) - f(x - h)) / h;
}

double FiniteDifference::centralDifference(std::function<double(double)> f, double x, double h) {
    return (f(x + h) - f(x - h)) / (2.0 * h);
}

double FiniteDifference::centralDifference4(std::function<double(double)> f, double x, double h) {
    return (-f(x + 2 * h) + 8 * f(x + h) - 8 * f(x - h) + f(x - 2 * h)) / (12.0 * h);
}
```

- **main.cpp**

  В качестве тестируемой функции была взята $y = e^x$. Это частично упрощает задачу анализа, поскольку ее производные равны ей самой.

```cpp
// Исследуемая функция f(x) = exp(x)
double f(double x) {
    return std::exp(x);
}

// Первая производная аналитически (e^x)
double f_prime(double x) {
    return std::exp(x);
}

// Вторая производная аналитически (e^x)
double f_double_prime(double x) {
    return std::exp(x);
}
```

  Помимо основного main, в файле есть функции printTableHeader, printTableRow и conductResearch. Их задача – вызов других функций, которые участвуют в тестировании сплайна, а также косметическая настройка вывода таблицы

```cpp
void printTableHeader() {
    std::cout << std::setw(12) << "x"
        << std::setw(15) << "Spline"
        << std::setw(15) << "f(x)"
        << std::setw(15) << "Error"
        << std::setw(15) << "Spline'"
        << std::setw(15) << "f'(x)"
        << std::setw(15) << "Error'"
        << std::setw(15) << "Spline''"
        << std::setw(15) << "f''(x)"
        << std::setw(15) << "Error''"
        << std::endl;
    std::cout << std::string(147, '-') << std::endl;
}

void printTableRow(double x, const CubicSpline& spline, double true_val,
    double true_deriv, double true_dderiv) {
    double spline_val = spline.evaluate(x);
    double spline_deriv = spline.derivative1(x);
    double spline_dderiv = spline.derivative2(x);

    std::cout << std::setw(12) << std::setprecision(4) << x
        << std::setw(15) << std::setprecision(6) << spline_val
        << std::setw(15) << true_val
        << std::setw(15) << std::abs(spline_val - true_val)
        << std::setw(15) << spline_deriv
        << std::setw(15) << true_deriv
        << std::setw(15) << std::abs(spline_deriv - true_deriv)
        << std::setw(15) << spline_dderiv
        << std::setw(15) << true_dderiv
        << std::setw(15) << std::abs(spline_dderiv - true_dderiv)
        << std::endl;
}
```

```
void conductResearch(double a, double b, int baseSegments, const std::string& label) {
    std::cout << "\n" << label << " (шаг h = " << (b - a) / baseSegments << "):" << std::endl;

    // Генерация сетки
    auto grid = GridGenerator::generateRegularGrid(a, b, baseSegments);

    // Вычисление значений функции в узлах
    std::vector<double> values;
    for (double node : grid) {
        values.push_back(f(node));
    }

    // Построение сплайна
    CubicSpline spline;
    spline.initialize(grid, values);

    // Точки для исследования
    std::vector<double> test_points;
    int num_points = 12;
    for (int i = 1; i <= num_points; ++i) {
        double point = a + (b - a) * i / (num_points + 1);
        test_points.push_back(point);
    }

    printTableHeader();
    for (double point : test_points) {
        printTableRow(point, spline, f(point), f_prime(point), f_double_prime(point));
    }
}
```

Сам же main по большей части тоже состоит из вывода различных сообщений, вызова вспомогательных функций.

```
int main() {
    setlocale(LC_ALL, "RU");
    const double a = 0.08;
    const double b = 0.32;
    const double epsilon = 0.001;

    // Исследование на вложенных сетках
    int base_segments = 10;
    conductResearch(a, b, base_segments, "Сетка h");
    conductResearch(a, b, base_segments * 2, "Сетка h/2");
    conductResearch(a, b, base_segments * 4, "Сетка h/4");

    // Вычисление производной в центральной точке конечными разностями
    double center = (b - a) / 2.0;
    std::cout << "\n\nВычисление производной в центральной точке x = " << center << std::endl;
    std::cout << "Аналитическое значение: " << f_prime(center) << std::endl;

    double h = (b - a) / 10.0;

    std::cout << "\nСравнение методов конечных разностей:" << std::endl;
    std::cout << std::setw(20) << "Метод"
        << std::setw(15) << "Значение"
        << std::setw(15) << "Ошибка"
        << std::setw(15) << "Шаг" << std::endl;
    std::cout << std::string(65, '-') << std::endl;

    auto printMethod = [&](const std::string& name, double (*method)(std::function<double(double)>, double, double), double h) {
        double result = method(f, center, h);
        double error = std::abs(result - f_prime(center));
        std::cout << std::setw(20) << name
            << std::setw(15) << std::setprecision(8) << result
            << std::setw(15) << error
            << std::setw(15) << h << std::endl;
        };

    printMethod("Левосторонняя O(h)", FiniteDifference::forwardDifference, h);
    printMethod("Правостороняя O(h)", FiniteDifference::backwardDifference, h);
    printMethod("Центральная O(h^2)", FiniteDifference::centralDifference, h);
    printMethod("Центральная O(h^4)", FiniteDifference::centralDifference4, h);

    std::cout << "\nОптимальный шаг для точности " << epsilon << ": h = " << h << std::endl;

    return 0;
```

- **Вывод и результаты задания 3:**

Сетка h (шаг h = 0.024):

x	Spline	f(x)	Error	Spline'	f'(x)	Error'	Spline''	f''(x)	Error''
0.09846	1.10349	1.10347	1.39679e-05	1.10073	1.10347	0.00274059	1.07678	1.10347	0.0266964
0.1169	1.12403	1.12403	7.26622e-06	1.12449	1.12403	0.000459617	1.21613	1.12403	0.0920958
0.1354	1.14498	1.14498	2.1311e-06	1.14505	1.14498	7.61008e-05	1.09758	1.14498	0.0473983
0.1538	1.16631	1.16631	2.30319e-07	1.1662	1.16631	0.000106712	1.18517	1.16631	0.0188615
0.1723	1.18804	1.18804	1.62149e-07	1.18809	1.18804	5.00876e-05	1.1861	1.18804	0.00194172
0.1908	1.21018	1.21018	6.60454e-08	1.21017	1.21018	1.16304e-05	1.20995	1.21018	0.00022819
0.2092	1.23273	1.23273	1.10248e-07	1.23274	1.23273	1.4758e-05	1.23196	1.23273	0.000768147
0.2277	1.2557	1.2557	2.10831e-07	1.25563	1.2557	6.45387e-05	1.25343	1.2557	0.00226635
0.2462	1.2791	1.2791	2.93715e-07	1.27923	1.2791	0.000136197	1.30304	1.2791	0.0239408
0.2646	1.30293	1.30293	2.71043e-06	1.30283	1.30293	9.6916e-05	1.24266	1.30293	0.0602658
0.2831	1.3272	1.32721	9.23759e-06	1.32662	1.32721	0.000584268	1.44429	1.32721	0.117082
0.3015	1.35195	1.35194	1.77569e-05	1.35542	1.35194	0.00348399	1.318	1.35194	0.0339407

Сетка h/2 (шаг h = 0.012):

x	Spline	f(x)	Error	Spline'	f'(x)	Error'	Spline''	f''(x)	Error''
0.09846	1.10347	1.10347	1.8167e-06	1.1037	1.10347	0.000229791	1.19557	1.10347	0.0920946
0.1169	1.12403	1.12403	5.80535e-08	1.12398	1.12403	5.39027e-05	1.14283	1.12403	0.0187996
0.1354	1.14498	1.14498	3.0135e-08	1.14497	1.14498	5.79553e-06	1.14376	1.14498	0.00122119
0.1538	1.16631	1.16631	1.90779e-09	1.16631	1.16631	7.05569e-07	1.16599	1.16631	0.000325747
0.1723	1.18804	1.18804	5.14212e-10	1.18804	1.18804	1.42546e-07	1.18806	1.18804	1.69479e-05
0.1908	1.21018	1.21018	7.20217e-11	1.21018	1.21018	2.18579e-08	1.21019	1.21018	6.36973e-06
0.2092	1.23273	1.23273	8.67779e-11	1.23273	1.23273	2.50969e-08	1.23274	1.23273	7.81384e-06
0.2277	1.2557	1.2557	6.44851e-10	1.2557	1.2557	1.78246e-07	1.25572	1.2557	2.08163e-05
0.2462	1.2791	1.2791	2.42878e-09	1.2791	1.2791	8.99824e-07	1.27868	1.2791	0.000413641
0.2646	1.30293	1.30293	3.83161e-08	1.30294	1.30293	7.36884e-06	1.30138	1.30293	0.00155316
0.2831	1.32721	1.32721	7.37997e-08	1.32728	1.32721	6.85226e-05	1.35111	1.32721	0.0238996
0.3015	1.35193	1.35194	2.30947e-06	1.35164	1.35194	0.000292121	1.46901	1.35194	0.117075

Сетка h/4 (шаг h = 0.006):

x	Spline	f(x)	Error	Spline'	f'(x)	Error'	Spline''	f''(x)	Error''
0.09846	1.10347	1.10347	1.45126e-08	1.10345	1.10347	2.69479e-05	1.12228	1.10347	0.0188056
0.1169	1.12403	1.12403	4.80262e-10	1.12403	1.12403	3.586e-07	1.12371	1.12403	0.000323447
0.1354	1.14498	1.14498	1.37803e-11	1.14498	1.14498	6.21446e-09	1.14498	1.14498	5.68354e-06
0.1538	1.16631	1.16631	2.61013e-12	1.16631	1.16631	1.68332e-09	1.16631	1.16631	8.86015e-07
0.1723	1.18804	1.18804	3.60356e-12	1.18804	1.18804	1.16574e-09	1.18804	1.18804	1.50115e-06
0.1908	1.21018	1.21018	4.03788e-12	1.21018	1.21018	4.13015e-10	1.21018	1.21018	1.78385e-06
0.2092	1.23273	1.23273	4.10938e-12	1.23273	1.23273	4.27665e-10	1.23273	1.23273	1.81538e-06
0.2277	1.2557	1.2557	3.79985e-12	1.2557	1.2557	1.23686e-09	1.2557	1.2557	1.58201e-06
0.2462	1.2791	1.2791	2.80331e-12	1.2791	1.2791	1.84011e-09	1.2791	1.2791	9.48355e-07
0.2646	1.30293	1.30293	1.72413e-11	1.30293	1.30293	7.63178e-09	1.30294	1.30293	7.18591e-06
0.2831	1.32721	1.32721	6.1064e-10	1.32721	1.32721	4.56048e-07	1.3268	1.32721	0.000411124
0.3015	1.35194	1.35194	1.8449e-08	1.35197	1.35194	3.42574e-05	1.37584	1.35194	0.0239066

По данным фотографиям можно заметить, что наименьшую ошибку сплайн достигает примерно в середине предложенного отрезка. Самые худшие результаты показываются на последних значениях x: 0.2831 и 0.3015.

Это закономерно, ведь в середине отрезка сплайн "опирается" на узлы с обеих сторон, а на границах сплайн имеет информацию только с одной стороны, с другой его ограничивает формальное граничное условие. Ближе к середине влияние этого условия становится все слабее и слабее

- **Вывод и результаты задания 4:**

```
Вычисление производной в центральной точке x = 0.12
Аналитическое значение: 1.1275

Сравнение методов конечных разностей:
                Метод       Значение          Ошибка            Шаг
------------------------------------------------------------------
  Левосторонняя O(h)      1.1411357    0.013638854           0.024
  Правостороняя O(h)      1.1140745    0.013422369           0.024
  Центральная O(h^2)      1.1276051  0.00010824282           0.024
  Центральная O(h^4)      1.1274968  1.2470071e-08           0.024

Оптимальный шаг для точности 0.001: h = 0.024
```

По данной фотографии можно проследить, что центральные разности показывают лучший результат, чем двухшаговые, что тоже является закономерным результатом

- **Задание 5:**
  Для выполнения 5 задание занесем необходимы промт в AI-ассистент DeepSeek

Оптимальный метод численного дифференцирования для табличной функции | x = 0, 0.1, 0.2 | f = 0, 0.01, 0.04 с двумя верными десятичными знаками

Для табличной функции:

x	f(x)
0.0	0.00
0.1	0.01
0.2	0.04

Требуется найти производную с **двумя верными десятичными знаками**.

Анализ задачи

Имеем 3 точки с постоянным шагом h = 0.1. Для численного дифференцирования с такой точностью оптимально использовать методы с погрешностью $O(h^2)$.

Решение

1. Производная в точке x = 0.1 (центральная разность)

Формула центральной разности:

$$f'(x_1) \approx \frac{f(x_2) - f(x_0)}{2h}$$

$$f'(0.1) \approx \frac{0.04 - 0.00}{2 \cdot 0.1} = \frac{0.04}{0.2} = 0.20$$

Погрешность: $O(h^2) \approx 0.01^2 = 0.0001$

2. Производная в точке x = 0.0 (правая разность)

Формула с погрешностью O(h):

$$f'(x_0) \approx \frac{f(x_1) - f(x_0)}{h} = \frac{0.01 - 0.00}{0.1} = 0.10$$

Более точная формула (погрешность $O(h^2)$):

$$f'(x_0) \approx \frac{-3f(x_0) + 4f(x_1) - f(x_2)}{2h}$$

$$f'(0.0) \approx \frac{-3 \cdot 0.00 + 4 \cdot 0.01 - 0.04}{2 \cdot 0.1} = \frac{0.04 - 0.04}{0.2} = 0.00$$

3. Производная в точке x = 0.2 (левая разность)

Формула с погрешностью $O(h^2)$:

$$f'(x_2) \approx \frac{3f(x_2) - 4f(x_1) + f(x_0)}{2h}$$

$$f'(0.2) \approx \frac{3 \cdot 0.04 - 4 \cdot 0.01 + 0.00}{2 \cdot 0.1} = \frac{0.12 - 0.04}{0.2} = \frac{0.08}{0.2} = 0.40$$

Проверка точности

Для $f(x) = x^2$ (похоже на наши данные):

- Точная производная: f'(x) = 2x
- f'(0.0) = 0.00
- f'(0.1) = 0.20
- f'(0.2) = 0.40

Наши численные результаты совпадают с точными значениями.

Ответ

Оптимальный метод - использование формул с погрешностью $O(h^2)$:

- **f'(0.0) = 0.00**
- **f'(0.1) = 0.20**
- **f'(0.2) = 0.40**

Все значения имеют **два верных десятичных знака**.

В качестве недостатков можно выделить то, что перед решением задачи не было произведено начального анализа данных. Также не было приведено объяснения, почему требуется искать производную именно с двумя верными знаками. Для ответов на подобные вопросы требуется написание более подробного начального промта или последующих уточняющих вопросов