Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №5
по дисциплине «Вычислительная математика»
Вариант 17

Преподаватель:

Машина Е.А.

Выполнил:

Щетинин С.В.

P3208

Санкт-Петербург

2024 г.

Цель лабораторной работы

Решить задачу интерполяции, найти значения функции при заданных значениях аргумента, отличных от узловых точек.

Рабочие формулы методов

Многочлен Лагранжа

$$L_n(x) = \sum_{i=0}^{n} y_i\, l_i(x)$$

$$Ln(x) = \sum_{i=0}^{n} y_i \frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)}$$

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

Введем обозначение: $t = (x - x_0)/h$. Тогда получим формулу Ньютона, которая называется **первой интерполяционной формулой Ньютона для интерполирования вперед:**

$$N_n(x) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_0 + \cdots + \frac{t(t-1)\ldots(t-n+1)}{n!}\Delta^n y_0$$

Для правой половины отрезка разности вычисляют справа налево: $t = (x - x_n)/h$. Тогда получим формулу Ньютона, которая называется **второй интерполяционной формулой Ньютона для интерполирования назад:**

$$N_n(x) = y_n + t\Delta y_{n-1} + \frac{t(t+1)}{2!}\Delta^2 y_{n-2} + \cdots + \frac{t(t+1)\ldots(t+n-1)}{n!}\Delta^n y_0$$

Первая интерполяционная формула Гаусса $(x > a)$

$$\begin{aligned}P_n(x) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-1}\\ + \frac{(t+1)t(t-1)(t-2)}{4!}\Delta^4 y_{-2}\\ + \frac{(t+2)(t+1)t(t-1)(t-2)}{5!}\Delta^5 y_{-2}\ldots\\ + \frac{(t+n-1)\ldots(t-n+1)}{(2n-1)!}\Delta^{2n-1} y_{-(n-1)}\\ + \frac{(t+n-1)\ldots(t-n)}{(2n)!}\Delta^{2n} y_{-n}\end{aligned}$$

Вторая интерполяционная формула Гаусса $(x < a)$

$$\begin{aligned}P_n(x) = y_0 + t\Delta y_{-1} + \frac{t(t+1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-2}\\ + \frac{(t+2)(t+1)t(t-1)}{4!}\Delta^4 y_{-2} + \cdots\\ + \frac{(t+n-1)\ldots(t-n+1)}{(2n-1)!}\Delta^{2n-1} y_{-n}\\ + \frac{(t+n)(t+n-1)\ldots(t-n+1)}{(2n)!}\Delta^{2n} y_{-n}\end{aligned}$$

Порядок выполнения работы

Программная реализация задачи

```
#include <iostream>
#include <vector>
#include <fstream>
#include <algorithm>
#include <cmath>

struct Point {
    long double x, y;

    bool operator<(const Point& other) const
    {
        return (x < other.x)
               || ((x == other.x) && (y < other.y));
    }

};

std::vector <Point> input_points() {

    int n;
    std::cout << "Input count of points:\n";
    std::cin >> n;

    std::cout << "Input " << n << " x's:\n";

    std::vector <Point> res;

    for (int i = 0; i < n; ++i) {
        long double x;
        std::cin >> x;
        res.push_back(Point{x});
    }

    std::cout << "Input " << n << " y's:\n";

    for (int i = 0; i < n; ++i) {
        long double y;
        std::cin >> y;
        res[i].y = y;
    }

    return res;

}

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

    long double res = 0;
    size_t n = points.size();

    for (size_t i = 0; i < n; ++i) {
        long double xres = 1;
        for (size_t j = 0; j < n; ++j) {
            if (j == i) continue;

            xres *= (x - points[j].x) / (points[i].x - points[j].x);
```

```
        }
        res += points[i].y * xres;
    }

    return res;
}

template <typename Iterator>
auto slice(Iterator begin, Iterator end) {
    using ValueType = typename std::iterator_traits<Iterator>::value_type;
    std::vector<ValueType> res;

    for (auto it = begin; it != end; ++it) {
        res.push_back(*it);
    }

    return res;
}

long double f_newt(const std::vector <Point> &points,
                   const std::vector <size_t>& nums_x) {

    if (nums_x.size() == 1) {
        return points[nums_x[0]].y;
    }

    return (f_newt(points, slice(nums_x.begin() + 1, nums_x.end())) -
            f_newt(points, slice(nums_x.begin(), nums_x.end() - 1))) /
                    (points[nums_x.back()].x - points[nums_x[0]].x);

}

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

    long double res = f_newt(points, {0});
    size_t n = points.size();

    for (size_t k = 0; k < n; ++k) {
        long double cur_res = 1;
        std::vector <size_t> nums;
        for (size_t j = 0; j < k; ++j) {
            cur_res *= (x - points[j].x);
            nums.push_back(j);
        }
        nums.push_back(k);
        cur_res *= f_newt(points, nums);
        res += cur_res;
    }
    return res;
}


std::vector<std::vector<long double>> compute_finite_differences(const
std::vector<Point>& pt) {
    size_t n = pt.size();
    std::vector<std::vector<long double>> finite_differences(n,
std::vector<long double>(n));

    for (int i = 0; i < n; ++i) {
        finite_differences[i][0] = pt[i].y;
    }

    for (int j = 1; j < n; ++j) {
        for (int i = 0; i < n - j; ++i) {
```

```
            finite_differences[i][j] = finite_differences[i+1][j-1] -
finite_differences[i][j-1];
        }
    }

    return finite_differences;
}

long double gauss(const std::vector<Point>& pt, long double x) {
    size_t n = pt.size();
    std::vector<std::vector<long double>> finite_differences =
compute_finite_differences(pt);

    long double a = pt[n / 2].x; // Центральная точка
    long double h = pt[1].x - pt[0].x; // Шаг (предполагается равномерный)

    long double t = (x - a) / h;

    long double result = finite_differences[n / 2][0];
    long double term = 1.0;
    int sign = 1;

    for (int i = 1; i < n; ++i) {
        term *= (t - (i / 2.)) / i;
        if (i % 2 == 0) {
            sign = -sign;
        }
        result += sign * term * finite_differences[(n - i) / 2][i];
    }

    return result;
}

long double stirling(const std::vector<Point>& pt, long double x) {
    size_t n = pt.size();
    std::vector<std::vector<long double>> finite_differences =
compute_finite_differences(pt);

    size_t mid = n / 2;  // Центральный индекс
    long double a = pt[mid].x; // Центральная точка
    long double h = pt[1].x - pt[0].x; // Шаг (предполагается равномерный)

    long double t = (x - a) / h;
    long double t2 = t * t;

    long double result = finite_differences[mid][0];
    long double term = 1.0;

    for (size_t i = 1; i < n; ++i) {
        if (i % 2 == 0) {
            term *= (t2 - ((long double)i / 2.) * ((long double)i / 2.)) / (i
* (i - 1));
            result += term * (finite_differences[mid - i / 2][i] +
finite_differences[mid - i / 2 + 1][i]) / 2;
        } else {
            term *= t * (t2 - (((long double)i - 1.) / 2) * (((long double)i
- 1.) / 2)) / (i * (i - 1));
            result += term * finite_differences[mid - (i - 1) / 2][i];
        }
    }

    return result;
}
```

```cpp
long double bessel(const std::vector<Point>& pt, long double x) {
    size_t n = pt.size();
    std::vector<std::vector<long double>> finite_differences =
compute_finite_differences(pt);

    int mid = n / 2;  // Центральный индекс
    long double a = pt[mid].x; // Центральная точка
    long double h = pt[1].x - pt[0].x; // Шаг (предполагается равномерный)

    long double t = (x - a) / h;
    long double t2 = t * t;

    long double result = (finite_differences[mid][0] + finite_differences[mid
- 1][1]) / 2;
    long double term = t * (finite_differences[mid][1] +
finite_differences[mid - 1][2]) / 2;
    result += term;

    for (int i = 1; i < n / 2; ++i) {
        term *= (t2 - i * i) / (4 * i * i - 1);
        if (i % 2 == 0) {
            result += term * (finite_differences[mid - i][2 * i] +
finite_differences[mid - i - 1][2 * i + 1]) / 2;
        } else {
            result += term * finite_differences[mid - i][2 * i + 1];
        }
    }

    return result;
}


void write_points(const std::vector <Point> &v, std::ofstream & out) {
    for (auto &p : v) {
        out << p.x << " ";
        std::cout << p.x << " ";
    }
    std::cout << "\n";
    out << "\n";
    for (auto &p : v) {
        out << p.y << " ";
        if (!_isnan(p.y))
            std::cout << p.y << " ";
    }
    out << "\n";
    std::cout << "\n";
}

int main() {

    std::vector <Point> points = input_points();

    std::ofstream out("../script/df");

    std::sort(points.begin(), points.end());
    std::vector <Point> lag, nw, gs, strl, bes;

    for (long double x = points[0].x; x <= points.back().x; x += 0.5) {
        lag.push_back({x, lagrange(points, x)});
        nw.push_back({x, newton(points, x)});
        gs.push_back({x, gauss(points, x)});
        strl.push_back({x, stirling(points, x)});
        bes.push_back({x, bessel(points, x)});
    }
```

```
    std::cout << "Lagrange Interpolation: \n";
    write_points(lag, out);
    std::cout << "Newton's Interpolation:: \n";
    write_points(nw, out);
    std::cout << "Gauss's Interpolation: \n";
    write_points(gs, out);
    write_points(strl, out);
    write_points(strl, out);
    std::cout << "Starting points: \n";
    write_points(points, out);

    out.close();

    system(DRAW_GRAPH);

    return 0;
}
```

Вывод

В ходе выполнения лабораторной работы были изучены методы для решения задачи интерполяции такие как многочлен Лагранжа, многочлен Ньютона, многочлен Гаусса и найдены с помощью них значения функции при заданных значениях аргумента, отличных от узловых точек.