Московский авиационный институт
(национальный исследовательский университет)

Институт № 8 «Компьютерные науки и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа № 3 по курсу «Численные методы»

Студент:	Н. О. Тимофеева
Преподаватель:	Д. Е. Пивоваров
Группа:	М8О-308Б-19
Вариант:	19
Дата:	
Оценка:	

Москва, 2022

Лабораторная работа 3

Методы приближения функций. Численное дифференцирование и интегрирование

3.1

Используя таблицу значений Yi функции y = f(x), вычисленных в точках Xi, i = 0,...,3 построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки [Xi, Yi]. Вычислить значение погрешности интерполяции в точке X'

Исходный код

```cpp
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

class polynom {
private:
    vector<double> data;
    constexpr static double EPS = 0.000000001;
    size_t n;
public:
    polynom() : data(1), n(1) {}
    polynom(int _n) : data(_n), n(_n) {}
    polynom(const vector<double> & coef) : data(coef), n(data.size()) {}
    size_t size() const {
        return n;
    }
    double & operator [] (size_t id) {
        return data[id];
    }
    const double & operator [] (size_t id) const {
        return data[id];
    }
    friend polynom operator + (const polynom & lhs, const polynom & rhs) {
        polynom res(max(lhs.size(), rhs.size()));
        for (size_t i = 0; i < lhs.size(); ++i) {
            res[i] += lhs[i];
        }
        for (size_t i = 0; i < rhs.size(); ++i) {
            res[i] += rhs[i];
        }
        return res;
    }
    friend polynom operator - (const polynom & lhs, const polynom & rhs) {
        polynom res(max(lhs.size(), rhs.size()));
        for (size_t i = 0; i < lhs.size(); ++i) {
            res[i] += lhs[i];
        }
        for (size_t i = 0; i < rhs.size(); ++i) {
            res[i] -= rhs[i];
        }
        return res;
    }
    friend polynom operator * (double lambda, const polynom & p) {
        polynom res(p);
        for (size_t i = 0; i < res.size(); ++i) {
            res[i] *= lambda;
        }
        return res;
    }
    friend polynom operator / (const polynom & p, double lambda) {
        polynom res(p);
        for (size_t i = 0; i < res.size(); ++i) {
            res[i] /= lambda;
        }
        return res;
    }
    friend polynom operator * (const polynom & lhs, const polynom & rhs) {
        polynom res(lhs.size() + rhs.size());
        for (size_t i = 0; i < lhs.size(); ++i) {
            for (size_t j = 0; j < rhs.size(); ++j) {
                res[i + j] += lhs[i] * rhs[j];
            }
        }
        while (res.n > 1 and abs(res.data.back()) < EPS) {
            res.data.pop_back();
            --res.n;
        }
        return res;
    }
    polynom integrate() {
        polynom res(n + 1);
        for (size_t i = 1; i < n + 1; ++i) {
            res.data[i] = data[i - 1] / (double)i;
        }
        return res;
    }
    double integrate(double l, double r) {
        polynom F = integrate();
        return F(r) - F(l);
    }
    polynom derivative() {
        polynom res(n - 1);
        for (size_t i = 1; i < n; ++i) {
            res[i - 1] = data[i] * i;
        }
        return res;
    }
    double operator () (double x) {
        double res = 0.0;
        double xi = 1.0;
        for (double elem : data) {
            res += elem * xi;
            xi *= x;
        }
        return res;
    }
    friend ostream & operator << (ostream & out, const polynom & poly) {
        bool flag = false;
        int deg = 0;
        for (double elem : poly.data) {
            if (!(abs(elem) < EPS)) {
                if (flag and deg) {
                    out << (elem > EPS ? " + " : " - ");
                }
                out << abs(elem);
                flag = true;
                if (deg) {
                    out << " * x";
                    if (deg > 1) {
                        out << " ^ " << deg;
                    }
                }
            }
            ++deg;
        }
        if (!flag) {
            out << 0;
        }
        return out;
    }
    ~polynom() = default;
};
class interLagrange {
    vector<double> x;
    vector<double> y;
    size_t n;
public:
    interLagrange(const vector<double> & _x, const vector<double> & _y) : x(_x), y(_y), n(x.size()) {};
    polynom operator () () {
        polynom res(vector<double>({0}));
        for (size_t i = 0; i < n; ++i) {
            polynom li(vector<double>({1}));
            for (size_t j = 0; j < n; ++j) {
                if (i == j) {
                    continue;
                }
                polynom xij(vector<double>({-x[j], 1}));
                li = li * xij / (x[i] - x[j]);
            }
            res = res + y[i] * li;
        }
        return res;
    }
};
class interNewton {
private:
    vector<double> x;
    vector<double> y;
    size_t n;
    vector<vector<bool>> calc;
    vector<vector<double>> memo;
    double f(int l, int r) {
        if (calc[l][r]) {
            return memo[l][r];
        }
        calc[l][r] = true;
        double res;
        if (l + 1 == r) {
            res = (y[l] - y[r]) / (x[l] - x[r]);
        } else {
            res = (f(l, r - 1) - f(l + 1, r)) / (x[l] - x[r]);
        }
        return memo[l][r] = res;
    }
public:
```

```cpp
    interNewton(const vector<double> & _x, const vector<double> & _y) : x(_x), y(_y), n(x.size()) {
        calc.resize(n, vector<bool>(n));
        memo.resize(n, vector<double>(n));
    };
    polynom operator () () {
        polynom li(vector<double>({-x[0], 1}));
        polynom res(vector<double>({y[0]}));
        int r = 0;
        for (size_t i = 1; i < n; ++i) {
            res = res + f(0, ++r) * li;
            li = li * polynom(vector<double>({-x[i], 1}));
        }
        return res;
    }
};
int main() {
    int n;
    cin >> n;
    vector<double> x(n), y(n);
    for (int i = 0; i < n; ++i) {
        cin >> x[i];
        y[i] = asin(x[i]) + x[i];
    }
    double x_error;
    cin >> x_error;
    interLagrange myLagrange(x, y);
    polynom lagrange = myLagrange();
    cout << "  :\n" << lagrange << endl;
    cout << "   X' = " << abs(lagrange(x_error) - asin(x_error) - x_error) << "\n\n";
    interNewton myNewton(x, y);
    polynom newton = myNewton();
    cout << "  :\n" << newton << endl;
    cout << "   X' = " << abs(newton(x_error) - asin(x_error) - x_error) << "\n\n";
}
```

Входные данные

```
test1:
4
-0.4 -0.1 0.2 0.5
0.1

test2:
4
-0.4 0 0.2 0.5
0.1
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-1$ g++ main.cpp
```

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-1$ ./a.out <test1
Интерполяционный многочлен Лагранжа:
7.55373e-05 + 1.99923 * x + 0.00197657 * x ^ 2 + 0.188516 * x ^ 3
Погрешность в точке X'= 0.000111543

Интерполяционный многочлен Ньютона:
7.55373e-05 + 1.99923 * x + 0.00197657 * x ^ 2 + 0.188516 * x ^ 3
Погрешность в точке X'= 0.000111543

natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-1$ ./a.out <test2
Интерполяционный многочлен Лагранжа:
1.99889 * x + 0.00141004 * x ^ 2 + 0.190404 * x ^ 3
Погрешность в точке X'= 7.37745e-05

Интерполяционный многочлен Ньютона:
1.99889 * x + 0.00141004 * x ^ 2 + 0.190404 * x ^ 3
Погрешность в точке X'= 7.37745e-05

natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-1$
```

3.2

Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при x = x0 и x = x4. Вычислить значение функции в точке x = X’.

Исходный код

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

using namespace std;

template<class T>
class tridiag_t {
private:
    const double EPS = 0.000001;
    int n;
    vector<T> a;
    vector<T> b;
    vector<T> c;
public:
    tridiag_t(const int & _n) : n(_n), a(n), b(n), c(n) {}
    tridiag_t(const vector<T> & _a, const vector<T> & _b, const vector<T> & _c) {
        if (!(_a.size() == _b.size() and _a.size() == _c.size())) {
            throw invalid_argument("Sizes of a, b, c are invalid");
        }
        n = _a.size();
        a = _a;
        b = _b;
        c = _c;
    }
    vector<T> solve(const vector<T> & d) {
        int m = d.size();
        if (n != m) {
            throw invalid_argument("Size of vector d is invalid");
        }
        vector<T> p(n);
        p[0] = -c[0] / b[0];
        vector<T> q(n);
        q[0] = d[0] / b[0];
        for (int i = 1; i < n; ++i) {
            p[i] = -c[i] / (b[i] + a[i] * p[i - 1]);
            q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]);
        }
        vector<T> x(n);
        x.back() = q.back();
        for (int i = n - 2; i >= 0; --i) {
            x[i] = p[i] * x[i + 1] + q[i];
        }
        return x;
    }
    friend istream & operator >> (istream & in, tridiag_t<T> & tridiag) {
        in >> tridiag.b[0] >> tridiag.c[0];
        for (int i = 1; i < tridiag.n - 1; ++i) {
            in >> tridiag.a[i] >> tridiag.b[i] >> tridiag.c[i];
        }
        in >> tridiag.a.back() >> tridiag.b.back();
        return in;
    }
    ~tridiag_t() = default;
};
class cub_spline_t {
    size_t n;
    vector<double> x;
    vector<double> y;
    vector<double> a, b, c, d;
    void buildSpline() {
        vector<double> h(n + 1);
        h[0] = NAN;
        for (size_t i = 1; i <= n; ++i) {
            h[i] = x[i] - x[i - 1];
        }
        vector<double> func1(n - 1);
        vector<double> func2(n - 1);
        vector<double> func3(n - 1);
        vector<double> func4(n - 1);
        for (size_t i = 2; i <= n; ++i) {
            func1[i - 2] = h[i - 1];
            func2[i - 2] = 2.0 * (h[i - 1] + h[i]);
            func3[i - 2] = h[i];
            func4[i - 2] = 3.0 * ((y[i] - y[i - 1]) / h[i] - (y[i - 1] - y[i - 2]) / h[i - 1]);
        }
        func1[0] = 0.0;
        func3.back() = 0.0;
        tridiag_t<double> systemOfFunc(func1, func2, func3);
        vector<double> cSolved = systemOfFunc.solve(func4);
        for (size_t i = 2; i <= n; ++i) {
            c[i] = cSolved[i - 2];
        }
        for (size_t i = 1; i <= n; ++i) {
            a[i] = y[i - 1];
        }
        for (size_t i = 1; i < n; ++i) {
            b[i] = (y[i] - y[i - 1]) / h[i] - h[i] * (c[i + 1] + 2.0 * c[i]) / 3.0;
            d[i] = (c[i + 1] - c[i]) / (3.0 * h[i]);
        }
        c[1] = 0.0;
        b[n] = (y[n] - y[n - 1]) / h[n] - (2.0 / 3.0) * h[n] * c[n];
        d[n] = -c[n] / (3.0 * h[n]);
    }
public:
    cub_spline_t(const vector<double> & _x, const vector<double> & _y) {
        if (_x.size() != _y.size()) {
            throw invalid_argument("Sizes does not match");
        }
        x = _x;
        y = _y;
        n = x.size() - 1;
        a.resize(n + 1);
        b.resize(n + 1);
        c.resize(n + 1);
        d.resize(n + 1);
        buildSpline();

    }
    double operator () (double x0) {
        for (size_t i = 1; i <= n; ++i) {
            if (x[i - 1] <= x0 and x0 <= x[i]) {
                double x1 = x0 - x[i - 1];
                double x2 = x1 * x1;
            double x3 = x2 * x1;
            return a[i] + b[i] * x1 + c[i] * x2 + d[i] * x3;
         }
      }
      return NAN;
   }
   friend ostream & operator << (ostream & out, const cub_spline_t & spline) {
      for (size_t i = 1; i <= spline.n; ++i) {
         out << "i = " << i << ", a = " << spline.a[i] << ", b = " << spline.b[i] << ", c = " << spline.c
             [i] << ", d = " << spline.d[i] << '\n';
      }
      return out;
   }
};
int main() {
   int n;
   cin >> n;
   vector<double> x(n), y(n);
   for (int i = 0; i < n; ++i) {
      cin >> x[i];
   }
   for (int i = 0; i < n; ++i) {
      cin >> y[i];
   }
   double x0;
   cin >> x0;
   cout.precision(6);
   cub_spline_t func(x, y);
   cout << "C:\n" << func << endl;
   cout << "f(X') = " << func(x0) << endl;
}
```

Входные данные

```
test:
5
-0.4 -0.1 0.2 0.5 0.8
-0.81152 -0.20017 0.40136 1.0236 1.7273
0.1
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-2$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-2$ ./a.out <test
Сплайн:
i = 1,a = -0.81152,b = 2.04668,c = 0,d = -0.0983333
i = 2,a = -0.20017,b = 2.02013,c = -0.0885,d = 0.127963
i = 3,a = 0.40136,b = 2.00158,c = 0.0266667,d = 0.717222
i = 4,a = 1.0236,b = 2.21123,c = 0.672167,d = -0.746852
```

```
f(X') = 0.20134
```

3.3

Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

Исходный код

```cpp
#include <iostream>
#include <cmath>
#include <functional>
#include <vector>

template<class T>
std::vector<T> operator + (const std::vector<T> & a, const std::vector<T> & b) {
    size_t n = a.size();
    std::vector<T> c(n);
    for (size_t i = 0; i < n; ++i) {
        c[i] = a[i] + b[i];
    }
    return c;
}
template<class T>
std::vector<T> operator - (const std::vector<T> & a, const std::vector<T> & b) {
    size_t n = a.size();
    std::vector<T> c(n);
    for (size_t i = 0; i < n; ++i) {
        c[i] = a[i] - b[i];
    }
    return c;
}
template<class T>
class matrix_t {
private:
    size_t n, m;
    std::vector<std::vector<T>> data;
public:
    matrix_t() : n(1), m(1), data(1) {}
    matrix_t(size_t _n) : n(_n), m(_n) {
        data.resize(n, std::vector<T>(n));
    }
    matrix_t(size_t _n, size_t _m) : n(_n), m(_m) {
        data.resize(n, std::vector<T>(m));
    }
    matrix_t(const matrix_t<T> & other) {
        n = other.n;
        m = other.m;
        data = other.data;
    }
    matrix_t<T> & operator = (const matrix_t<T> & other) {
        if (this == &other) {
            return *this;
        }
        n = other.n;
        m = other.m;
        data = other.data;
        return *this;
    }
    static matrix_t<T> identity(size_t n) {
        matrix_t<T> res(n, n);
        for (size_t i = 0; i < n; ++i) {
            res[i][i] = T(1);
        }
        return res;
    }
    matrix_t<T> t() const {
        matrix_t<T> res(m, n);
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < m; ++j) {
                res[j][i] = data[i][j];
            }
        }
        return res;
    }
    size_t rows() const {
        return n;
    }
    size_t cols() const {
        return m;
    }
    void swap_rows(size_t i, size_t j) {
        if (i == j) {
            return;
        }
        for (size_t k = 0; k < m; ++k) {
            std::swap(data[i][k], data[j][k]);
        }
    }
    void swap_cols(size_t i, size_t j) {
        if (i == j) {
            return;
        }
        for (size_t k = 0; k < n; ++k) {
            std::swap(data[k][i], data[k][j]);
        }
    }
    friend matrix_t<T> operator + (const matrix_t<T> & a, const matrix_t<T> & b) {
        if (a.rows() != b.rows() or a.cols() != b.cols()) {
            throw std::invalid_argument("Sizes does not match");
        }
        size_t n = a.rows();
        size_t m = a.cols();
        matrix_t<T> res(n, m);
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < m; ++j) {
                res[i][j] = a[i][j] + b[i][j];
            }
        }
        return res;
    }
    friend matrix_t<T> operator - (const matrix_t<T> & a, const matrix_t<T> & b) {
        if (a.rows() != b.rows() or a.cols() != b.cols()) {
            throw std::invalid_argument("Sizes does not match");
        }
        size_t n = a.rows();
        size_t m = a.cols();
        matrix_t<T> res(n, m);
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < m; ++j) {
                res[i][j] = a[i][j] - b[i][j];
            }
        }
        return res;
    }
    friend matrix_t<T> operator * (const matrix_t<T> & a, const matrix_t<T> & b) {
        if (a.cols() != b.rows()) {
            throw std::invalid_argument("Sizes does not match");
        }
        size_t n = a.rows();
        size_t k = a.cols();
        size_t m = b.cols();
        matrix_t<T> res(n, m);
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < m; ++j) {
                for (size_t ii = 0; ii < k; ++ii) {
                    res[i][j] += a[i][ii] * b[ii][j];
                }
            }
        }
        return res;
    }
    friend std::vector<T> operator * (const matrix_t<T> & a, const std::vector<T> & b) {
        if (a.cols() != b.size()) {
            throw std::invalid_argument("Sizes does not match");
        }
        size_t n = a.rows();
        size_t m = a.cols();
        std::vector<T> c(n);
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < m; ++j) {
                c[i] += a[i][j] * b[j];
            }
        }
        return c;
    }
    friend matrix_t<T> operator * (T lambda, const matrix_t<T> & a) {
        size_t n = a.rows();
        size_t m = a.cols();
        matrix_t<T> res(n, m);
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < m; ++j) {
                res[i][j] = lambda * a[i][j];
            }
        }
        return res;
    }
    std::vector<T> & operator [] (size_t i) {
        return data[i];
    }
    const std::vector<T> & operator [] (size_t i) const {
        return data[i];
    }
    friend std::istream & operator >> (std::istream & in, matrix_t<T> & matr) {
        for (size_t i = 0; i < matr.rows(); ++i) {
            for (size_t j = 0; j < matr.cols(); ++j) {
                in >> matr[i][j];
            }
        }
        return in;
    }
    friend std::ostream & operator << (std::ostream & out, const matrix_t<T> & matr) {
        for (size_t i = 0; i < matr.rows(); ++i) {
            for (size_t j = 0; j < matr.cols(); ++j) {
                if (j) {
                    out << ", ";
                }
                out << matr[i][j];
            }
            out << '\n';
        }
        return out;
    }
    ~matrix_t() = default;
};
template<class T>
class lu_t {
private:
    const T EPS = 0.000001;
    matrix_t<T> l;
    matrix_t<T> u;
    T det;
    std::vector<std::pair<size_t, size_t>> swaps;
    void do_swaps(std::vector<T> & x) {
        for (std::pair<size_t, size_t> elem : swaps) {
            std::swap(x[elem.first], x[elem.second]);
        }
    }
    void decompose() {
        size_t n = u.rows();
        for (size_t i = 0; i < n; ++i) {
            size_t max_el_ind = i;
            for (size_t j = i + 1; j < n; ++j) {
                if (abs(u[j][i]) > abs(u[max_el_ind][i])) {
                    max_el_ind = j;
                }
            }
            if (max_el_ind != i) {
                std::pair<size_t, size_t> perm = std::make_pair(i, max_el_ind);
                swaps.push_back(perm);
                u.swap_rows(i, max_el_ind);
                l.swap_rows(i, max_el_ind);
                l.swap_cols(i, max_el_ind);
            }
            for (size_t j = i + 1; j < n; ++j) {
                if (abs(u[i][i]) < EPS) {
                    continue;
                }
                T mu = u[j][i] / u[i][i];
                l[j][i] = mu;
                for (size_t k = 0; k < n; ++k) {
                    u[j][k] -= mu * u[i][k];
                }
            }
        }
        det = (swaps.size() & 1 ? -1 : 1);
        for (size_t i = 0; i < n; ++i) {
            det *= u[i][i];
        }
    }
public:
    lu_t(const matrix_t<T> & matr) {
        if (matr.rows() != matr.cols()) {
            throw std::invalid_argument("Matrix is not square");
        }
        l = matrix_t<T>::identity(matr.rows());
        u = matrix_t<T>(matr);
        decompose();
    }
    std::vector<T> solve(std::vector<T> b) {
        int n = b.size();
        do_swaps(b);
        std::vector<T> z(n);
        for (int i = 0; i < n; ++i) {
            T summary = b[i];
            for (int j = 0; j < i; ++j) {
                summary -= z[j] * l[i][j];
            }
            z[i] = summary;
        }
        std::vector<T> x(n);
        for (int i = n - 1; i >= 0; --i) {
            if (abs(u[i][i]) < EPS) {
                continue;
            }
            T summary = z[i];
            for (int j = n - 1; j > i; --j) {
                summary -= x[j] * u[i][j];
            }
            x[i] = summary / u[i][i];
        }
        return x;
    }
    T get_det() {
        return det;
    }
    matrix_t<T> inv_matrix() {
        size_t n = l.rows();
        matrix_t<T> res(n);
        for (size_t i = 0; i < n; ++i) {
            std::vector<T> b(n);
            b[i] = T(1);
            std::vector<T> x = solve(b);
            for (size_t j = 0; j < n; ++j) {
                res[j][i] = x[j];
            }
        }
        return res;
    }
    friend std::ostream & operator << (std::ostream & out, const lu_t<T> & lu) {
        out << "Matrix L:\n" << lu.l << "Matrix U:\n" << lu.u;
        return out;
    }
    ~lu_t() = default;
};
class polynom {
private:
    std::vector<double> data;
    constexpr static double EPS = 0.000000001;
    size_t n;
public:
    polynom() : data(1), n(1) {}
    polynom(int _n) : data(_n), n(_n) {}
    polynom(const std::vector<double> & coef) : data(coef), n(data.size()) {}
    size_t size() const {
        return n;
    }
    double & operator [] (size_t id) {
        return data[id];
    }
    const double & operator [] (size_t id) const {
        return data[id];
    }
    friend polynom operator + (const polynom & lhs, const polynom & rhs) {
        polynom res(std::max(lhs.size(), rhs.size()));
        for (size_t i = 0; i < lhs.size(); ++i) {
            res[i] += lhs[i];
        }
        for (size_t i = 0; i < rhs.size(); ++i) {
            res[i] += rhs[i];
        }
        return res;
    }
    friend polynom operator - (const polynom & lhs, const polynom & rhs) {
        polynom res(std::max(lhs.size(), rhs.size()));
        for (size_t i = 0; i < lhs.size(); ++i) {
            res[i] += lhs[i];
        }
        for (size_t i = 0; i < rhs.size(); ++i) {
            res[i] -= rhs[i];
        }
        return res;
    }
    friend polynom operator * (double lambda, const polynom & p) {
        polynom res(p);
        for (size_t i = 0; i < res.size(); ++i) {
            res[i] *= lambda;
        }
        return res;
    }
    friend polynom operator / (const polynom & p, double lambda) {
        polynom res(p);
        for (size_t i = 0; i < res.size(); ++i) {
            res[i] /= lambda;
        }
        return res;
    }
    friend polynom operator * (const polynom & lhs, const polynom & rhs) {
        polynom res(lhs.size() + rhs.size());
        for (size_t i = 0; i < lhs.size(); ++i) {
            for (size_t j = 0; j < rhs.size(); ++j) {
                res[i + j] += lhs[i] * rhs[j];
            }
        }
        while (res.n > 1 and abs(res.data.back()) < EPS) {
            res.data.pop_back();
            --res.n;
        }
        return res;
    }
    polynom integrate() {
        polynom res(n + 1);
        for (size_t i = 1; i < n + 1; ++i) {
            res.data[i] = data[i - 1] / (double)i;
        }
        return res;
    }
    double integrate(double l, double r) {
        polynom F = integrate();
        return F(r) - F(l);
    }
    polynom derivative() {
        polynom res(n - 1);
        for (size_t i = 1; i < n; ++i) {
            res[i - 1] = data[i] * i;
        }
        return res;
    }
    double operator () (double x) {
        double res = 0.0;
        double xi = 1.0;
        for (double elem : data) {
            res += elem * xi;
            xi *= x;
        }
        return res;
    }
    friend std::ostream & operator << (std::ostream & out, const polynom & poly) {
        bool flag = false;
        int deg = 0;
        for (double elem : poly.data) {
            if (!(abs(elem) < EPS)) {
                if (flag and deg) {
                    out << (elem > EPS ? " + " : " - ");
                }
                out << abs(elem);
                flag = true;
                if (deg) {
                    out << " * x";
                    if (deg > 1) {
                        out << " ^ " << deg;
                    }
                }
            }
            ++deg;
        }
        if (!flag) {
            out << 0;
        }
        return out;
    }
    ~polynom() = default;
};
class minimal_square_t {
    size_t n;
    size_t m;
    std::vector<double> x;
    std::vector<double> y;
    std::vector<double> a;
    std::vector<std::function<double(double)>> phi;
    double get(double x0) {
        double res = 0.0;
        for (size_t i = 0; i < m; ++i) {
            res += a[i] * phi[i](x0);
        }
        return res;
    }
    void build() {
        matrix_t<double> lhs(n, m);
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < m; ++j) {
                lhs[i][j] = phi[j](x[i]);
            }
        }
        matrix_t<double> lhs_t = lhs.t();
        lu_t<double> lhs_lu(lhs_t * lhs);
        std::vector<double> rhs = lhs_t * y;
        a = lhs_lu.solve(rhs);
    }
public:
    minimal_square_t(const std::vector<double> & _x, const std::vector<double> & _y, const std::vector<std
         ::function<double(double)>> & _phi) {
        if (_x.size() != _y.size()) {
            throw std::invalid_argument("Sizes does not match");
        }
        x = _x;
        y = _y;
        n = _x.size();
        m = _phi.size();
        a.resize(m);
        phi = _phi;
        build();
    }
    double operator () (double x0) {
        return get(x0);
    }
    double mmse() {
        double res = 0;
        for (size_t i = 0; i < n; ++i) {
            res += pow(get(x[i]) - y[i], 2.0);
        }
        return res;
    }
    friend std::ostream & operator << (std::ostream & out, const minimal_square_t & item) {
        for (size_t i = 0; i < item.m; ++i) {
            if (i) {
                out << ' ';
            }
            out << item.a[i];
        }
        return out;
    }
};
double f0(double x0) {
    return 1.0;
}
double f1(double x0) {
    return x0;
}
double f2(double x0) {
    return x0 * x0;
}
int main() {
    int n;
    std::cin >> n;
    std::vector<double> x(n), y(n);
    for (int i = 0; i < n; ++i) {
        std::cin >> x[i];
    }
    for (int i = 0; i < n; ++i) {
        std::cin >> y[i];
    }
    std::cout.precision(6);
```

```cpp
    std::cout << std::fixed;
    std::vector<std::function<double(double)>> phi1 = {f0, f1};
    minimal_square_t ms1(x, y, phi1);
    std::cout << "   1-  a0, a1: " << std::endl;
    std::cout << ms1 << std::endl;
    std::cout << "    1-   = " << ms1.mmse() << "\n\n";
    std::vector<std::function<double(double)>> phi2 = {f0, f1, f2};
    minimal_square_t ms2(x, y, phi2);
    std::cout << "   2-  a0, a1, a2: " << std::endl;
    std::cout << ms2 << std::endl;
    std::cout << "    2-   = " << ms2.mmse() << "\n\n";
}
```

Входные данные

```
test:
6
-0.7 -0.4 -0.1 0.2 0.5 0.8
-1.4754 -0.81152 -0.20017 0.40136 1.0236 1.7273
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-3$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-3$ ./a.out <test
Коэффициенты приближающего многочлена 1-ой степени a0,a1:
0.005526 2.106704
Сумма квадратов ошибок многочлена 1-ой степени = 0.003942

Коэффициенты приближающего многочлена 2-ой степени a0,a1,a2:
0.005526 2.106704 0.000000
Сумма квадратов ошибок многочлена 2-ой степени = 0.003942
```

3.4

Вычислить первую и вторую производную от таблично заданной функции yi = f(xi), i = 0, 1, 2, 3, 4 в точке x = X'.

Исходный код

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <exception>

using namespace std;

const double EPS = 0.000000001;
bool border(double a, double b) {
    return (a < b) or (abs(b - a) < EPS);
}
class derivative_t {
    int n;
    vector<double> x;
    vector<double> y;
public:
    derivative_t(const vector<double> & xl, const vector<double> & yl) {
        if (xl.size() != yl.size()) {
            throw invalid_argument(" ");
        }
        x = xl;
        y = yl;
        n = x.size();
    }
    double derivative1(double x0) {
        for (int i = 0; i < n - 2; ++i) {
            if (x[i] < x0 && border(x0, x[i + 1])) {
                double dydx1 = (y[i + 1] - y[i]) / (x[i + 1] - x[i]);
                double dydx2 = (y[i + 2] - y[i + 1]) / (x[i + 2] - x[i + 1]);
                double res = dydx1 + (dydx2 - dydx1) * (2.0 * x0 - x[i] - x[i + 1]) / (x[i + 2] - x[i]);
                return res;
            }
        }
        return NAN;
    }
    double derivative2(double x0) {
        for (int i = 0; i < n - 2; ++i) {
            if (x[i] < x0 && border(x0, x[i + 1])) {
                double dydx1 = (y[i + 1] - y[i]) / (x[i + 1] - x[i]);
                double dydx2 = (y[i + 2] - y[i + 1]) / (x[i + 2] - x[i + 1]);
                double res = 2.0 * (dydx2 - dydx1) / (x[i + 2] - x[i]);
                return res;
            }
        }
        return NAN;
    }
};
int main() {
    int n;
    cin >> n;
    vector<double> x(n), y(n);
    for (int i = 0; i < n; ++i) {
        cin >> x[i];
    }
    for (int i = 0; i < n; ++i) {
        cin >> y[i];
    }
    double x0;
    cin >> x0;
    cout.precision(6);
    derivative_t f(x, y);
    cout << "f'(" << x0 << ") = " << f.derivative1(x0) << endl;
    cout << "f''(" << x0 << ") = " << f.derivative2(x0) << endl;
}
```

Входные данные

```
test:
5
-1.0 0.0 1.0 2.0 3.0
-1.7854 0.0 1.7854 3.1071 4.249
1.0
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-4$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-4$ ./a.out <test
f'(1) = 1.55355
f''(1) = -0.4637
```

3.5

Вычислить определенный интеграл, методами прямоугольников, трапеций, Симпсона с шагами h1, h2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга.

Исходный код

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

using namespace std;

class polynom {
private:
    vector<double> data;
    constexpr static double EPS = 0.000000001;
    size_t n;
public:
    polynom() : data(1), n(1) {}
    polynom(int _n) : data(_n), n(_n) {}
    polynom(const vector<double> & coef) : data(coef), n(data.size()) {}
    size_t size() const {
        return n;
    }
    double & operator [] (size_t id) {
        return data[id];
    }
    const double & operator [] (size_t id) const {
        return data[id];
    }
    friend polynom operator + (const polynom & lhs, const polynom & rhs) {
        polynom res(max(lhs.size(), rhs.size()));
        for (size_t i = 0; i < lhs.size(); ++i) {
            res[i] += lhs[i];
        }
        for (size_t i = 0; i < rhs.size(); ++i) {
            res[i] += rhs[i];
        }
        return res;
    }
    friend polynom operator - (const polynom & lhs, const polynom & rhs) {
        polynom res(max(lhs.size(), rhs.size()));
        for (size_t i = 0; i < lhs.size(); ++i) {
            res[i] += lhs[i];
        }
        for (size_t i = 0; i < rhs.size(); ++i) {
            res[i] -= rhs[i];
        }
        return res;
    }
    friend polynom operator * (double lambda, const polynom & p) {
        polynom res(p);
        for (size_t i = 0; i < res.size(); ++i) {
            res[i] *= lambda;
        }
        return res;
    }
    friend polynom operator / (const polynom & p, double lambda) {
        polynom res(p);
        for (size_t i = 0; i < res.size(); ++i) {
            res[i] /= lambda;
        }
        return res;
    }
    friend polynom operator * (const polynom & lhs, const polynom & rhs) {
        polynom res(lhs.size() + rhs.size());
        for (size_t i = 0; i < lhs.size(); ++i) {
            for (size_t j = 0; j < rhs.size(); ++j) {
                res[i + j] += lhs[i] * rhs[j];
            }
        }
        while (res.n > 1 and abs(res.data.back()) < EPS) {
            res.data.pop_back();
            --res.n;
        }
        return res;
    }
    polynom integrate() {
        polynom res(n + 1);
        for (size_t i = 1; i < n + 1; ++i) {
            res.data[i] = data[i - 1] / (double)i;
        }
        return res;
    }
    double integrate(double l, double r) {
        polynom F = integrate();
        return F(r) - F(l);
    }
    polynom derivative() {
        polynom res(n - 1);
        for (size_t i = 1; i < n; ++i) {
            res[i - 1] = data[i] * i;
        }
        return res;
    }
    double operator () (double x) {
        double res = 0.0;
        double xi = 1.0;
        for (double elem : data) {
            res += elem * xi;
            xi *= x;
        }
        return res;
    }
    friend ostream & operator << (ostream & out, const polynom & poly) {
        bool flag = false;
        int deg = 0;
        for (double elem : poly.data) {
            if (!(abs(elem) < EPS)) {
                if (flag and deg) {
                    out << (elem > EPS ? " + " : " - ");
                }
                out << abs(elem);
                flag = true;
                if (deg) {
                    out << " * x";
                    if (deg > 1) {
                        out << " ^ " << deg;
                    }
                }
            }
            ++deg;
        }
        if (!flag) {
            out << 0;
        }
        return out;
    }
    ~polynom() = default;
};
class interLagrange {
    vector<double> x;
    vector<double> y;
    size_t n;
public:
    interLagrange(const vector<double> & _x, const vector<double> & _y) : x(_x), y(_y), n(x.size()) {};
    polynom operator () () {
        polynom res(vector<double>({0}));
        for (size_t i = 0; i < n; ++i) {
            polynom li(vector<double>({1}));
            for (size_t j = 0; j < n; ++j) {
                if (i == j) {
                    continue;
                }
                polynom xij(vector<double>({-x[j], 1}));
                li = li * xij / (x[i] - x[j]);
            }
            res = res + y[i] * li;
        }
        return res;
    }
};
using func = double(double);
double integrRec(double l, double r, double h, func f) {
    double res = 0;
    double x0 = l;
    double x1 = l + h;
    while (x0 < r) {
        res += h * f((x0 + x1) / 2);
        x0 = x1;
        x1 += h;
    }
    return res;
}
double integrTrap(double l, double r, double h, func f) {
    double res = 0;
    double x0 = l;
    double x1 = l + h;
    while (x0 < r) {
        res += h * (f(x0) + f(x1));
        x0 = x1;
        x1 += h;
    }
    return res / 2;
}
double integrSimp(double l, double r, double h, func f) {
    double res = 0;
    double x0 = l;
    double x1 = l + h;
    while (x0 < r) {
        vector<double> x = {x0, (x0 + x1) * 0.5, x1};
        vector<double> y = {f(x[0]), f(x[1]), f(x[2])};
        interLagrange lagr(x, y);
        res += lagr().integrate(x0, x1);
        x0 = x1;
        x1 += h;
    }
    return res / 3;
}
double rungeRomberg(double fh, double fhk, double k, double d) {
    return (fh - fhk) / (pow(k, d) - 1.0);
}
double f(double x) {
    return (x * x) / (625.0 - pow(x, 4));
}
int main() {
    double l, r;
    cin >> l >> r;
    double h1, h2;
    cin >> h1 >> h2;
    double rec1 = integrRec(l, r, h1, f);
    double trap1 = integrTrap(l, r, h1, f);
    double simp1 = integrSimp(l, r, h1, f);
    cout.precision(5);
    cout << "      " << h1 << " = " << rec1 << endl;
    cout << "      " << h1 << " = " << trap1 << endl;
    cout << "      " << h1 << " = " << simp1 << "\n\n";
    double rec2 = integrRec(l, r, h2, f);
    double trap2 = integrTrap(l, r, h2, f);
    double simp2 = integrSimp(l, r, h2, f);
    cout << "      " << h2 << " = " << rec2 << endl;
    cout << "      " << h2 << " = " << trap2 << endl;
    cout << "      " << h2 << " = " << simp2 << "\n\n";
    double recError = abs(rungeRomberg(rec1, rec2, h2 / h1, 2));
    double trapError = abs(rungeRomberg(trap1, trap2, h2 / h1, 2));
    double simpError = abs(rungeRomberg(simp1, simp2, h2 / h1, 2));
    cout << "    = " << recError << endl;
    cout << "    = " << trapError << endl;
    cout << "    = " << simpError << endl;
}
```

Входные данные

```
test:
0 4
1.0 0.5
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-5$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-5$ ./a.out <test
Интеграл по методу прямоугольников с шагом 1 = 0.040489
Интеграл по методу трапеций с шагом 1 = 0.046395
```

```
Интеграл по методу Симпсона с шагом 1 = 0.014153

Интеграл по метод прямоугольников с шагом 0.5 = 0.041868
Интеграл по метод трапеций с шагом 0.5 = 0.043442
Интеграл по метод Симпсона с шагом 0.5 = 0.014131

Погрешность вычислений методом прямоугольников = 0.0018391
Погрешность вычислений методом трапеций = 0.0039374
Погрешность вычислений методом Симпсона = 2.88e-05
```