Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Численные методы»

Студент:	И. С. Своеволин
Преподаватель:	Д. Е. Пивоваров
Группа:	М8О-303Б-21
Дата:	
Оценка:	
Подпись:	

Москва, 2024

1 Методы приближения функций. Численное дифференцирование и интегрирование

1 Постановка задачи

3.1. Используя таблицу значений Y_i функции $y = f(x)$, вычисленных в точках $X_i, i = 0, \cdots, 3$ построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки X_i, Y_i. Вычислить значение погрешности интерполяции в точке X^*.

Вариант: 20

$$y = arccos(x) + x \tag{1}$$

а)

$$X_i = -0.4, -0.1, 0.2, 0.5 \tag{2}$$

б)

$$X_i = -0.4, 0, 0.2, 0.5 \tag{3}$$

$$X^* = 0.1 \tag{4}$$

2 Результаты работы

```
Значения функции в точках:
1.58231 1.57096 1.56944 1.54720
Многочлен Лагранжа: -0.18852*x^3-0.00198*x^2+0.00077*x^1+1.57087
Многочлен Ньютона: -0.18852*x^3-0.00198*x^2+0.00077*x^1+1.57087
Значения многочленов в точках:
1.58231 1.57096 1.56944 1.54720
```

Рис. 1: Вывод в консоли

3 Исходный код

matrix.h

```
#pragma once
#include <iostream>
#include <vector>
#include <ccomplex>
#include <fstream>

using namespace std;

using cmd = complex <double>;
const double pi = acos(-1);

struct matrix
{
    int rows = 0, cols = 0;
    vector <vector <double>> v;

    matrix() {}
    matrix(int _rows, int _cols)
    {
        rows = _rows;
        cols = _cols;
        v = vector <vector <double>>(rows, vector <double>(cols));
    }

    vector <double>& operator[](int row)
    {
        return v[row];
    }

    operator double()
```

```
    {
        return v[0][0];
    }
};

matrix operator*(matrix lhs, matrix rhs)
{
    if (lhs.cols != rhs.rows)
        return matrix(0, 0);
    matrix res(lhs.rows, rhs.cols);
    for (int i = 0; i < res.rows; i++)
    {
        for (int j = 0; j < res.cols; j++)
        {
            res[i][j] = 0;
            for (int k = 0; k < lhs.cols; k++)
                res[i][j] += lhs[i][k] * rhs[k][j];
        }
    }
    return res;
}

matrix operator*(double lhs, matrix rhs)
{
    for (int i = 0; i < rhs.rows; i++)
    {
        for (int j = 0; j < rhs.cols; j++)
            rhs[i][j] *= lhs;
    }
    return rhs;
}

matrix operator+(matrix lhs, matrix rhs)
{
    if (lhs.rows != rhs.rows || rhs.cols != lhs.cols)
        return matrix(0, 0);
    matrix res(lhs.rows, lhs.cols);
    for (int i = 0; i < rhs.rows; i++)
    {
        for (int j = 0; j < res.cols; j++)
            res[i][j] = lhs[i][j] + rhs[i][j];
    }
    return res;
}

matrix operator-(matrix lhs, matrix rhs)
{
    if (lhs.rows != rhs.rows || rhs.cols != lhs.cols)
        return matrix(0, 0);
```

```
    matrix res(lhs.rows, lhs.cols);
    for (int i = 0; i < rhs.rows; i++)
    {
        for (int j = 0; j < res.cols; j++)
            res[i][j] = lhs[i][j] - rhs[i][j];
    }
    return res;
}

ostream& operator<<(ostream& stream, matrix a)
{
    for (int i = 0; i < a.rows; i++)
    {
        for (int j = 0; j < a.cols; j++)
            stream << a[i][j] << ' ';
        stream << '\n';
    }
    return stream;
}

istream& operator>>(istream& stream, matrix& a)
{
    for (int i = 0; i < a.rows; i++)
    {
        for (int j = 0; j < a.cols; j++)
            stream >> a[i][j];
    }
    return stream;
}

matrix transposition(matrix a)
{
    matrix res(a.cols, a.rows);
    for (int i = 0; i < a.rows; i++)
    {
        for (int j = 0; j < a.cols; j++)
            res[j][i] = a[i][j];
    }
    return res;
}

vector <int> swp;

pair <matrix, matrix> lu_decomposition(matrix a)
{
    int n = a.rows;
    matrix l(n, n);
    swp = vector <int>(0);
    for (int k = 0; k < n; k++)
```

```
    {
        matrix prev = a;
        int idx = k;
        for (int i = k + 1; i < n; i++)
        {
            if (abs(prev[idx][k]) < abs(prev[i][k]))
                idx = i;
        }
        swap(prev[k], prev[idx]);
        swap(a[k], a[idx]);
        swap(l[k], l[idx]);
        swp.push_back(idx);
        for (int i = k + 1; i < n; i++)
        {
            double h = prev[i][k] / prev[k][k];
            l[i][k] = h;
            for (int j = k; j < n; j++)
                a[i][j] = prev[i][j] - h * prev[k][j];

        }
    }
    for (int i = 0; i < n; i++)
        l[i][i] = 1;
    return { l, a };
}

matrix solve_triag(matrix a, matrix b, bool up)
{
    int n = a.rows;
    matrix res(n, 1);
    int d = up ? -1 : 1;
    int first = up ? n - 1 : 0;
    for (int i = first; i < n && i >= 0; i += d)
    {
        res[i][0] = b[i][0];
        for (int j = 0; j < n; j++)
        {
            if (i != j)
                res[i][0] -= a[i][j] * res[j][0];
        }
        res[i][0] = res[i][0] / a[i][i];
    }
    return res;
}

matrix solve_gauss(pair <matrix, matrix> lu, matrix b)
{
    for (int i = 0; i < swp.size(); i++)
        swap(b[i], b[swp[i]]);
```

```
    matrix z = solve_triag(lu.first, b, false);
    matrix x = solve_triag(lu.second, z, true);
    //for (int i = 0; i < swp.size(); i++)
        //swap(x[i], x[swp[i]]);
    return x;
}

matrix inverse(matrix a)
{
    int n = a.rows;
    matrix b(n, 1);
    pair <matrix, matrix> lu = lu_decomposition(a);
    matrix res(n, n);
    for (int i = 0; i < n; i++)
    {
        b[max(i - 1, 0)][0] = 0;
        b[i][0] = 1;
        matrix col = solve_gauss(lu, b);
        for (int j = 0; j < n; j++)
            res[j][i] = col[j][0];
    }
    return res;
}

double determinant(matrix a)
{
    int n = a.rows;
    pair <matrix, matrix> lu = lu_decomposition(a);
    double det = 1;
    for (int i = 0; i < n; i++)
        det *= lu.second[i][i];
    return det;
}

matrix solve_tridiagonal(matrix& a, matrix& b)
{
    int n = a.rows;
    vector <double> p(n), q(n);
    p[0] = -a[0][1] / a[0][0];
    q[0] = b[0][0] / a[0][0];
    for (int i = 1; i < n; i++)
    {
        if (i != n - 1)
            p[i] = -a[i][i + 1] / (a[i][i] + a[i][i - 1] * p[i - 1]);
        else
            p[i] = 0;
        q[i] = (b[i][0] - a[i][i - 1] * q[i - 1]) / (a[i][i] + a[i][i - 1] * p[i - 1]);
    }
    matrix res(n, 1);
```

```
    res[n - 1][0] = q[n - 1];
    for (int i = n - 2; i >= 0; i--)
        res[i][0] = p[i] * res[i + 1][0] + q[i];
    return res;
}

double abs(matrix a)
{
    double mx = 0;
    for (int i = 0; i < a.rows; i++)
    {
        double s = 0;
        for (int j = 0; j < a.cols; j++)
            s += abs(a[i][j]);
        mx = max(mx, s);
    }
    return mx;
}

matrix solve_iteration(matrix a, matrix b, double eps)
{
    int n = a.rows;
    matrix alpha(n, n), beta(n, 1);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            alpha[i][j] = -a[i][j] / a[i][i];
        alpha[i][i] = 0;
    }
    for (int i = 0; i < n; i++)
        beta[i][0] = b[i][0] / a[i][i];
    matrix x = beta;
    double m = abs(a);
    double epsk = 2 * eps;
    while (epsk > eps)
    {
        matrix prev = x;
        x = beta + alpha * x;
        if (m < 1)
            epsk = m / (1 - m) * abs(x - prev);
        else
            epsk = abs(x - prev);
    }
    return x;
}

matrix solve_seidel(matrix a, matrix b, double eps)
{
    int n = a.rows;
```

```
    matrix alpha(n, n), beta(n, 1);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            alpha[i][j] = -a[i][j] / a[i][i];
        alpha[i][i] = 0;
    }
    for (int i = 0; i < n; i++)
        beta[i][0] = b[i][0] / a[i][i];
    matrix x = beta;
    double m = abs(alpha);
    double epsk = 2 * eps;
    while (epsk > eps)
    {
        matrix prev = x;
        for (int i = 0; i < n; i++)
        {
            double cur = beta[i][0];
            for (int j = 0; j < n; j++)
                cur += alpha[i][j] * x[j][0];
            x[i][0] = cur;
        }
        if (m < 1)
            epsk = m / (1 - m) * abs(x - prev);
        else
            epsk = abs(x - prev);
    }
    return x;
}

pair <matrix, matrix> method_jacobi(matrix a, double eps)
{
    int n = a.rows;
    double epsk = 2 * eps;
    matrix vec(n, n);
    for (int i = 0; i < n; i++)
        vec[i][i] = 1;
    while (epsk > eps)
    {
        int cur_i = 1, cur_j = 0;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < i; j++)
            {
                if (abs(a[cur_i][cur_j]) < abs(a[i][j]))
                {
                    cur_i = i;
                    cur_j = j;
                }
```

```
            }
        }
        matrix u(n, n);
        double phi = pi / 4;
        if (abs(a[cur_i][cur_i] - a[cur_j][cur_j]) > 1e-7)
            phi = 0.5 * atan((2 * a[cur_i][cur_j]) / (a[cur_i][cur_i] - a[cur_j][cur_j]));
        for (int i = 0; i < n; i++)
            u[i][i] = 1;
        u[cur_i][cur_j] = -sin(phi);
        u[cur_i][cur_i] = cos(phi);
        u[cur_j][cur_i] = sin(phi);
        u[cur_j][cur_j] = cos(phi);
        vec = vec * u;
        a = transposition(u) * a * u;
        epsk = 0;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < i; j++)
                epsk += a[i][j] * a[i][j];
        }
        epsk = sqrt(epsk);
    }
    matrix val(n, 1);
    for (int i = 0; i < n; i++)
        val[i][0] = a[i][i];
    return { val, vec };
}

double sign(double x)
{
    return x > 0 ? 1 : -1;
}

pair <matrix, matrix> qr_decomposition(matrix a)
{
    int n = a.rows;
    matrix e(n, n);
    for (int i = 0; i < n; i++)
        e[i][i] = 1;
    matrix q = e;
    for (int i = 0; i < n - 1; i++)
    {
        matrix v(n, 1);
        double s = 0;
        for (int j = i; j < n; j++)
            s += a[j][i] * a[j][i];
        v[i][0] = a[i][i] + sign(a[i][i]) * sqrt(s);
        for (int j = i + 1; j < n; j++)
```

```
            v[j][0] = a[j][i];
        matrix h = e - (2.0 / double(transposition(v) * v)) * (v * transposition(v));
        q = q * h;
        a = h * a;
    }
    return { q, a };
}

vector <cmd> qr_eigenvalues(matrix a, double eps)
{
    int n = a.rows;
    vector <cmd> prev(n);
    while (true)
    {
        pair <matrix, matrix> p = qr_decomposition(a);
        a = p.second * p.first;
        vector <cmd> cur;
        for (int i = 0; i < n; i++)
        {
            if (i < n - 1 && abs(a[i + 1][i]) > 1e-7)
            {
                double b = -(a[i][i] + a[i + 1][i + 1]);
                double c = a[i][i] * a[i + 1][i + 1] - a[i][i + 1] * a[i + 1][i];
                double d = b * b - 4 * c;
                cmd sgn = (d > 0) ? cmd(1, 0) : cmd(0, 1);
                d = sqrt(abs(d));
                cur.push_back(0.5 * (-b - sgn * d));
                cur.push_back(0.5 * (-b + sgn * d));
                i++;
            }
            else
                cur.push_back(a[i][i]);
        }
        bool ok = true;
        for (int i = 0; i < n; i++)
            ok = ok && abs(cur[i] - prev[i]) < eps;
        if (ok)
            break;
        prev = cur;
    }
    return prev;
}
```

3-1.cpp

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

```cpp
#include <functional>
#include <algorithm>
#include "matrix.h"

using namespace std;

struct polynomial
{
private:
    vector <double> v;

public:
    polynomial(vector <double> _v = {})
    {
        v = _v;
    }

    double size()
    {
        return v.size();
    }

    double operator[](int idx)
    {
        return v[idx];
    }

    double calculate(double x)
    {
        double res = 0;
        double cur = 1;
        for (int i = 0; i < v.size(); i++)
        {
            res += cur * v[i];
            cur = cur * x;
        }
        return res;
    }
};

ostream& operator<<(ostream& stream, polynomial a)
{
    for (int i = a.size() - 1; i > 0; i--)
        stream << ((i == a.size() - 1) ? a[i] : abs(a[i])) << "*" << "x^" << i << ((a[i - 1] > 0) ? '+' : '-');
    stream << abs(a[0]);
    return stream;
}

```

```
vector <double> open_brakets(vector <double> v)
{
    if (v.size() == 1)
        return { v[0], 1 };
    int n = v.size();
    double last = v.back();
    v.erase(--v.end());
    vector <double> res = open_brakets(v);
    vector <double> tmp = res;
    for (int i = 0; i < n; i++)
        res[i] = res[i] * last;
    res.push_back(0);
    for (int i = 1; i <= n; i++)
        res[i] += tmp[i - 1];
    return res;
}

polynomial interpolation_lagrange(vector <double> x, vector <double> y)
{
    int n = x.size();
    vector <double> res(n);
    for (int i = 0; i < n; i++)
    {
        vector <double> v;
        double k = y[i];
        for (int j = 0; j < n; j++)
        {
            if (i == j)
                continue;
            v.push_back(-x[j]);
            k = k / (x[i] - x[j]);
        }
        vector <double> tmp = open_brakets(v);
        for (int j = 0; j < n; j++)
            res[j] += k * tmp[j];
    }
    return polynomial(res);
}

polynomial interpolation_newton(vector <double> x, vector <double> y)
{
    int n = x.size();
    vector <double> res(n);
    res[0] = y[0];
    vector <vector <double>> diff(n - 1, vector <double>(n - 1));
    for (int i = 0; i < n - 1; i++)
        diff[0][i] = (y[i] - y[i + 1]) / (x[i] - x[i + 1]);
    for (int i = 1; i < n - 1; i++)
    {
```

```
        for (int j = 0; j < n - 1 - i; j++)
            diff[i][j] = (diff[i - 1][j] - diff[i - 1][j + 1]) / (x[j] - x[j + 1 + i]);
    }
    vector <double> cur;
    for (int i = 0; i < n - 1; i++)
    {
        cur.push_back(-x[i]);
        vector <double> tmp = open_brakets(cur);
        double k = diff[i][0];
        for (int j = 0; j < tmp.size(); j++)
            res[j] += k * tmp[j];
    }
    return polynomial(res);
}


double f1(double x)
{
    return acos(x) + x;
}

int main()
{
    setlocale(LC_ALL, "Rus");
    ofstream fout("answer3-1.txt");
    fout.precision(5);
    fout << fixed;

    vector <double> X = { -0.4, -0.1, 0.2, 0.5 };
    vector <double> Y;
    fout << "Значения функциивточках:\n";
    for (int i = 0; i < X.size(); i++)
    {
        fout << f1(X[i]) << ' ';
        Y.push_back(f1(X[i]));
    }
    polynomial p1 = interpolation_lagrange(X, Y);
    fout << "\Многочленn Лагранжа: " << p1 << '\n';
    polynomial p2 = interpolation_newton(X, Y);
    fout << "Многочлен Ньютона: " << p2 << '\n';
    fout << "Значения многочленоввточках:\n";
    for (int i = 0; i < X.size(); i++)
        fout << p1.calculate(X[i]) << ' ';

}
```

4 Постановка задачи

3.2. Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при $x = x_0$ и $x = x_4$. Вычислить значение функции в точке $x = X^*$.

Вариант: 20

$$X^* = 0.1 \tag{5}$$

i	0	1	2	3	4
x_i	-0.4	-0.1	0.2	0.5	0.8
f_i	1.5823	1.5710	1.5694	1.5472	1.4435

5 Результаты работы

```
[-0.40000;-0.10000] 1.00000*x^3+2.20000*x^2+2.28000*x^1+2.20630
[-0.10000;0.20000] 1.00000*x^3+1.30000*x^2+1.23000*x^1+1.68200
[0.20000;0.50000] 1.00000*x^3+0.40000*x^2+0.72000*x^1+1.40140
[0.50000;0.80000] 1.00000*x^3-0.50000*x^2+0.75000*x^1+1.17220
Значение сплайна в точке: 1.81900
```

Рис. 2: Вывод в консоли

6 Исходный код

3-2.cpp

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <functional>
#include <algorithm>
#include "matrix.h"

using namespace std;

struct polynomial
{
private:
    vector <double> v;

public:
    polynomial(vector <double> _v = {})
    {
        v = _v;
    }

    double size()
    {
        return v.size();
    }

    double operator[](int idx)
    {
        return v[idx];
    }
```

```
    double calculate(double x)
    {
        double res = 0;
        double cur = 1;
        for (int i = 0; i < v.size(); i++)
        {
            res += cur * v[i];
            cur = cur * x;
        }
        return res;
    }
};

ostream& operator<<(ostream& stream, polynomial a)
{
    for (int i = a.size() - 1; i > 0; i--)
        stream << ((i == a.size() - 1) ? a[i] : abs(a[i])) << "*" << "x^" << i << ((a[i - 1] > 0) ? '+' : '-');
    stream << abs(a[0]);
    return stream;
}

struct cubic_spline
{
private:
    vector <polynomial> p;
    vector <double> xn;

public:
    cubic_spline(vector <double> _xn = { 0 }, vector <polynomial> _p = {})
    {
        if (_p.size() == _xn.size() - 1)
        {
            p = _p;
            xn = _xn;
        }
        else
        {
            p = {};
            xn = { 0 };
        }
    }

    int size()
    {
        return p.size();
    }

```

```
    pair <pair <double, double>, polynomial> operator[](int idx)
    {
        return { {xn[idx], xn[idx + 1]}, p[idx] };
    }

    double calculate(double x)
    {
        int idx = upper_bound(xn.begin(), xn.end(), x) - xn.begin();
        idx = min(idx, (int)xn.size() - 1) - 1;
        idx = max(0, idx);
        return p[idx].calculate(x);
    }
};

ostream& operator<<(ostream& stream, cubic_spline a)
{
    for (int i = 0; i < a.size(); i++)
        stream << "[" << a[i].first.first << ";" << a[i].first.second << "] " << a[i].second << endl;
    return stream;
}

vector <double> open_brakets(vector <double> v)
{
    if (v.size() == 1)
        return { v[0], 1 };
    int n = v.size();
    double last = v.back();
    v.erase(--v.end());
    vector <double> res = open_brakets(v);
    vector <double> tmp = res;
    for (int i = 0; i < n; i++)
        res[i] = res[i] * last;
    res.push_back(0);
    for (int i = 1; i <= n; i++)
        res[i] += tmp[i - 1];
    return res;
}

cubic_spline make_spline(vector <double> x, vector <double> y)
{
    int n = x.size();
    vector <pair <double, double>> xy(n);
    for (int i = 0; i < n; i++)
        xy[i] = { x[i], y[i] };
    sort(xy.begin(), xy.end());
    for (int i = 0; i < n; i++)
    {
        x[i] = xy[i].first;
```

```
            y[i] = xy[i].second;
        }
        n--;
        vector <double> a(n), b(n), c(n), d(n);
        vector <double> h(n);
        for (int i = 0; i < n; i++)
            h[i] = x[i + 1] - x[i];
        matrix A(n - 1, n - 1), B(n - 1, 1);
        for (int i = 0; i < n - 1; i++)
        {
            if (i > 0)
                A[i][i - 1] = h[i];
            A[i][i] = 2 * (h[i] + h[i + 1]);
            if (i < n - 2)
                A[i][i + 1] = h[i + 1];
            B[i][0] = 3 * ((y[i + 2] - y[i + 1]) / h[i + 1] - (y[i + 1] - y[i]) / h[i]);
        }
        matrix s = solve_tridiagonal(A, B);
        for (int i = 1; i < n; i++)
            c[i] = s[i - 1][0];
        for (int i = 0; i < n; i++)
            a[i] = y[i];
        for (int i = 0; i < n - 1; i++)
            b[i] = (y[i + 1] - y[i]) / h[i] - 1. / 3. * (c[i + 1] + 2 * c[i]);
        b[n - 1] = (y[n] - y[n - 1]) / h[n - 1] - 2. / 3. * h[n - 1] * c[n - 1];
        for (int i = 0; i < n - 1; i++)
            d[i] = (c[i + 1] - c[i]) / (3 * h[i]);
        d[n - 1] = -c[n - 1] / (3 * h[n - 1]);
        vector <polynomial> vp(n);
        for (int i = 0; i < n; i++)
        {
            vector <double> res(4);
            res[0] = a[i];
            vector <double> tmp;
            for (int j = 1; j < 4; j++)
            {
                tmp.push_back(-x[i]);
                vector <double> v = open_brakets(tmp);
                for (int k = 0; k < v.size(); k++)
                    res[k] += v[k];
            }
            vp[i] = polynomial(res);
        }
        return cubic_spline(x, vp);
    }

    int main()
    {
        setlocale(LC_ALL, "Rus");
```

```
    ofstream fout("answer3-2.txt");
    fout.precision(5);
    fout << fixed;


    vector <double> X = { -0.4, -0.1, 0.2, 0.5, 0.8 };
    vector <double> Y = { 1.5823, 1.5710, 1.5694, 1.5472, 1.4435 };
    cubic_spline cs = make_spline(X, Y);
    fout << cs;
    fout << "Значение сплайнавточке: " << cs.calculate(0.1);

}
```

7 Постановка задачи

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.

Вариант: 20

i	0	1	2	3	4	5
x_i	-0.7	-0.4	-0.1	0.2	0.5	0.8
y_i	1.6462	1.5823	1.571	1.5694	1.5472	1.4435

8 Результаты работы

```
Приближающий многочлен 1-ой степени: -0.10670*x^1+1.56527
Приближающий многочлен 2-ой степени: -0.04813*x^2-0.10189*x^1+1.57778
Сумма квадратов ошибок для 1-ой степени: 0.00394
Сумма квадратов ошибок для 2-ой степени: 0.00324
```

Рис. 3: Вывод в консоли

9 Исходный код

3-3.cpp

```
#include <iostream>
#include <vector>
#include <fstream>
#include <cmath>
#include "matrix.h"

using namespace std;

struct polynomial {
private:
    vector<double> v;

public:
    polynomial(vector<double> _v = {}) {
        v = _v;
    }

    double size() const {
        return v.size();
    }

    double operator[](int idx) const {
        return v[idx];
    }

    double& operator[](int idx) {
        return v[idx];
    }

    double calculate(double x) const {
        double res = 0;
        double cur = 1;
```

```
        for (int i = 0; i < v.size(); i++) {
            res += cur * v[i];
            cur = cur * x;
        }
        return res;
    }
};

ostream& operator<<(ostream& stream, polynomial a)
{
    for (int i = a.size() - 1; i > 0; i--)
        stream << ((i == a.size() - 1) ? a[i] : abs(a[i])) << "*" << "x^" << i << ((a[i
             - 1] > 0) ? '+' : '-');
    stream << abs(a[0]);
    return stream;
}

polynomial least_squares(const vector<double>& x, const vector<double>& y, int m,
    double& sum_sq_error) {
    int n = x.size();
    m++;

    matrix phi(n, m);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            phi[i][j] = pow(x[i], j);
        }
    }

    matrix G = transposition(phi) * phi;
    matrix Y(n, 1);
    for (int i = 0; i < n; i++)
        Y[i][0] = y[i];
    matrix Z = transposition(phi) * Y;
    matrix A = solve_gauss(lu_decomposition(G), Z);

    vector<double> a(m);
    for (int i = 0; i < m; i++)
        a[i] = A[i][0];

    sum_sq_error = 0.0;
    for (int i = 0; i < n; i++) {
        double approx_y = 0.0;
        for (int j = 0; j < m; j++) {
            approx_y += a[j] * pow(x[i], j);
        }
        sum_sq_error += pow(y[i] - approx_y, 2);
    }

```

```
    return polynomial(a);
}

int main() {
    vector<double> x = {-0.7, -0.4, -0.1, 0.2, 0.5, 0.8};
    vector<double> y = {1.6462, 1.5823, 1.571, 1.5694, 1.5472, 1.4435};

    ofstream fout("answer3-3.txt");
    fout.precision(5);
    fout << fixed;

    double error1;
    polynomial p1 = least_squares(x, y, 1, error1);

    double error2;
    polynomial p2 = least_squares(x, y, 2, error2);

    fout << "Приближающий многочленой1- степени: ";
    fout << p1 << endl;

    fout << "Приближающий многочленой2- степени: ";
    fout << p2 << endl;

    fout << "Сумма квадратовошибокдляой1- степени: " << error1 << endl;
    fout << "Сумма квадратовошибокдляой2- степени: " << error2 << endl;
}
```

10 Постановка задачи

3.4. Вычислить первую и вторую производную от таблично заданной функции $y_i = f(x_i), i = 0, 1, 2, 3, 4$ в точке $x = X^*$.

Вариант: 20

$$X^* = 0.1 \tag{6}$$

i	0	1	2	3	4
x_i	-1	0	1	2	3
f_i	1.3562	1.5708	1.7854	2.4636	3.3218

11 Результаты работы

```
Значение первой производной: 0.21460
Значение второй производной: 0.46360
```

Рис. 4: Вывод в консоли

12 Исходный код

3-4.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <functional>
#include <algorithm>
#include "matrix.h"

using namespace std;

struct polynomial
{
private:
    vector <double> v;

public:
    polynomial(vector <double> _v = {})
    {
        v = _v;
    }

    double size()
    {
        return v.size();
    }

    double operator[](int idx)
    {
        return v[idx];
    }

    double calculate(double x)
    {
        double res = 0;
```

```
        double cur = 1;
        for (int i = 0; i < v.size(); i++)
        {
            res += cur * v[i];
            cur = cur * x;
        }
        return res;
    }
};

ostream& operator<<(ostream& stream, polynomial a)
{
    for (int i = a.size() - 1; i > 0; i--)
        stream << ((i == a.size() - 1) ? a[i] : abs(a[i])) << "*" << "x^" << i << ((a[i - 1] > 0) ? '+' : '-');
    stream << abs(a[0]);
    return stream;
}

struct cubic_spline
{
private:
    vector <polynomial> p;
    vector <double> xn;

public:
    cubic_spline(vector <double> _xn = { 0 }, vector <polynomial> _p = {})
    {
        if (_p.size() == _xn.size() - 1)
        {
            p = _p;
            xn = _xn;
        }
        else
        {
            p = {};
            xn = { 0 };
        }
    }

    int size()
    {
        return p.size();
    }

    pair <pair <double, double>, polynomial> operator[](int idx)
    {
        return { {xn[idx], xn[idx + 1]}, p[idx] };
    }
```

```
    double calculate(double x)
    {
        int idx = upper_bound(xn.begin(), xn.end(), x) - xn.begin();
        idx = min(idx, (int)xn.size() - 1) - 1;
        idx = max(0, idx);
        return p[idx].calculate(x);
    }
};

vector <double> open_brakets(vector <double> v)
{
    if (v.size() == 1)
        return { v[0], 1 };
    int n = v.size();
    double last = v.back();
    v.erase(--v.end());
    vector <double> res = open_brakets(v);
    vector <double> tmp = res;
    for (int i = 0; i < n; i++)
        res[i] = res[i] * last;
    res.push_back(0);
    for (int i = 1; i <= n; i++)
        res[i] += tmp[i - 1];
    return res;
}

polynomial interpolation_lagrange(vector <double> x, vector <double> y)
{
    int n = x.size();
    vector <double> res(n);
    for (int i = 0; i < n; i++)
    {
        vector <double> v;
        double k = y[i];
        for (int j = 0; j < n; j++)
        {
            if (i == j)
                continue;
            v.push_back(-x[j]);
            k = k / (x[i] - x[j]);
        }
        vector <double> tmp = open_brakets(v);
        for (int j = 0; j < n; j++)
            res[j] += k * tmp[j];
    }
    return polynomial(res);
}
```

```
polynomial derivative(polynomial p)
{
    int n = p.size();
    vector <double> res;
    for (int i = 1; i < n; i++)
        res.push_back(i * p[i]);
    return polynomial(res);
}

function <double(double)> derivative(vector <double> x, vector <double> y, int m)
{
    int n = x.size();
    vector <pair <double, double>> xy(n);
    for (int i = 0; i < n; i++)
        xy[i] = { x[i], y[i] };
    sort(xy.begin(), xy.end());
    for (int i = 0; i < n; i++)
    {
        x[i] = xy[i].first;
        y[i] = xy[i].second;
    }
    vector <polynomial> p(n - m);
    for (int i = 0; i < n - m; i++)
    {
        vector <double> xm, ym;
        for (int j = 0; j < m + 1; j++)
        {
            xm.push_back(x[i + j]);
            ym.push_back(y[i + j]);
        }
        p[i] = interpolation_lagrange(xm, ym);
        for (int j = 0; j < m; j++)
            p[i] = derivative(p[i]);
    }
    auto res = [=](double _x) mutable
    {
        int idx = lower_bound(x.begin() + 1, x.end() - m, _x) - x.begin();
        //int idx = upper_bound(x.begin() + 1, x.end() - m, _x) - x.begin();
        idx = idx - 1;
        return p[idx].calculate(_x);
    };
    return res;
}

int main()
{
    setlocale(LC_ALL, "Rus");
    ofstream fout("answer3-4.txt");
    fout.precision(5);
```

```
    fout << fixed;
    function <double(double)> df = derivative({-1, 0, 1, 2, 3}, {1.3562, 1.5708, 1.7854, 2.4636, 3.3218}, 1);
    function <double(double)> ddf = derivative({-1, 0, 1, 2, 3}, {1.3562, 1.5708, 1.7854, 2.4636, 3.3218}, 2);
    fout << "Значение первойпроизводной: " << df(1);
    fout << "\Значениеn второйпроизводной: " << ddf(1);
}
```

13 Постановка задачи

3.5. Вычислить определенный интеграл

$$F = \int_{x_0}^{x_1} y dx$$

, методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга:

Вариант: 20

$$y = \frac{\sqrt{x}}{4+3x}; X_0 = 1, X_k = 5, h_1 = 1.0, h_2 = 0.5 \tag{7}$$

14 Результаты работы

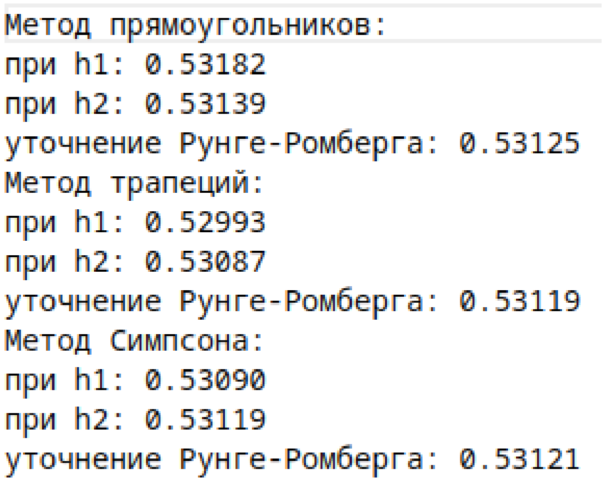

Рис. 5: Вывод в консоли

15 Исходный код

3-5.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <functional>
#include <algorithm>
#include "matrix.h"

using namespace std;


double integrate_rectangles(function <double(double)> f, double x0, double x1, double h)
{
    double res = 0;
    while (x0 < x1)
    {
        double x = x0 + h;
        res += f((x + x0) / 2) * h;
        x0 = x;
    }
```

```
    return res;
}

double integrate_trapezoids(function <double(double)> f, double x0, double x1, double
    h)
{
    double res = 0;
    while (x0 < x1)
    {
        double x = x0 + h;
        res += (f(x0) + f(x)) * h;
        x0 = x;
    }
    return res / 2;
}

double integrate_simpson(function <double(double)> f, double x0, double x1, double h)
{
    double res = 0;
    while (x0 < x1)
    {
        double x = x0 + 2 * h;
        double xm = x0 + h;
        res += (f(x0) + 4 * f(xm) + f(x)) * h;
        x0 = x;
    }
    return res / 3;
}

// rectangles -> p = 2
// trapezoids -> p = 2
// simpson -> p = 4
double method_runge(double i1, double i2, double h1, double h2, double p)
{
    double k = h2 / h1;
    return i1 + (i1 - i2) / (pow(k, p) - 1);
}

double f2(double x)
{
    return sqrt(x) / (4 + 3 * x);
}

int main()
{
    setlocale(LC_ALL, "Rus");
    ofstream fout("answer3-5.txt");
    fout.precision(5);
    fout << fixed;
```

```

    double h1 = 1, h2 = 0.5;
    fout << "Метод прямоугольников:\n";
    double i1 = integrate_rectangles(f2, 1, 5, h1);
    double i2 = integrate_rectangles(f2, 1, 5, h2);
    fout << "при h1: " << i1;
    fout << "\прин h2: " << i2;
    fout << "\уточнениеn РунгеРомберга-: " << method_runge(i1, i2, h1, h2, 2);
    fout << "\Методn трапеций:\n";
    i1 = integrate_trapezoids(f2, 1, 5, h1);
    i2 = integrate_trapezoids(f2, 1, 5, h2);
    fout << "при h1: " << i1;
    fout << "\прин h2: " << i2;
    fout << "\уточнениеn РунгеРомберга-: " << method_runge(i1, i2, h1, h2, 2);
    fout << "\Методn Симпсона:\n";
    i1 = integrate_simpson(f2, 1, 5, h1);
    i2 = integrate_simpson(f2, 1, 5, h2);
    fout << "при h1: " << i1;
    fout << "\прин h2: " << i2;
    fout << "\уточнениеn РунгеРомберга-: " << method_runge(i1, i2, h1, h2, 4);
}
```