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:	Т. В.Мохнач
Преподаватель:	Д. Е. Пивоваров
Группа:	М8О-303Б-21
Дата:	
Оценка:	
Подпись:	

Москва, 2024

1 Интерполяционные многочлены лагранжа и Ньютьона

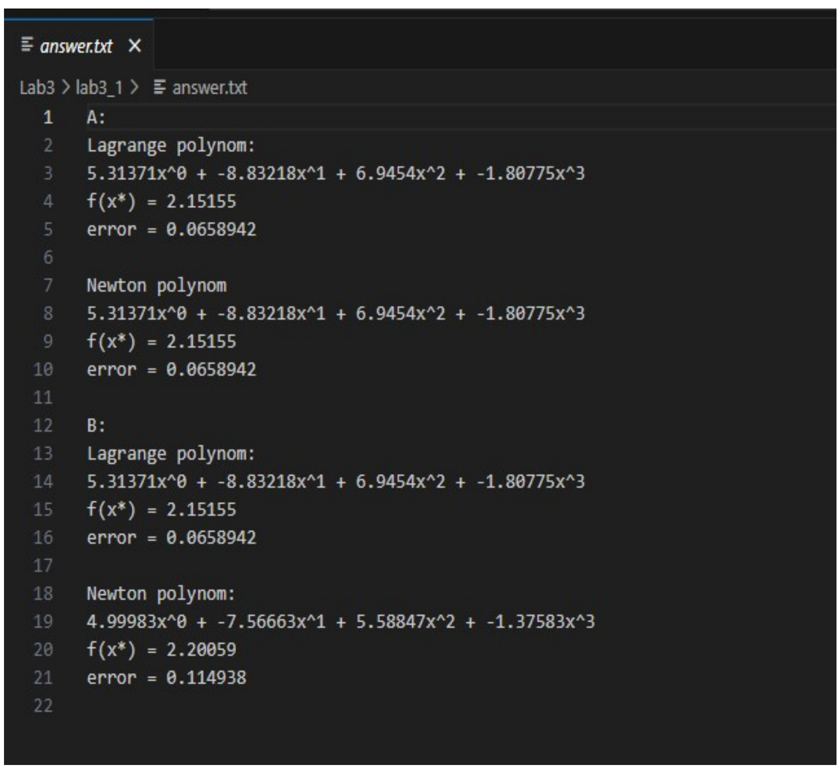
1 Постановка задачи

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

Вариант: 15

$$y = ctg(x) + x, a)X_i = \frac{\pi}{8}, \frac{2\pi}{8}, \frac{3\pi}{8}, \frac{4\pi}{8}\ b)X_i = \frac{\pi}{8}, \frac{\pi}{3}, \frac{3\pi}{8}, \frac{\pi}{2}\ X^* = \frac{3\pi}{16}$$

2 Результаты работы



answer.txt

Lab3 > lab3_1 > answer.txt

```
A:
Lagrange polynom:
5.31371x^0 + -8.83218x^1 + 6.9454x^2 + -1.80775x^3
f(x*) = 2.15155
error = 0.0658942

Newton polynom
5.31371x^0 + -8.83218x^1 + 6.9454x^2 + -1.80775x^3
f(x*) = 2.15155
error = 0.0658942

B:
Lagrange polynom:
5.31371x^0 + -8.83218x^1 + 6.9454x^2 + -1.80775x^3
f(x*) = 2.15155
error = 0.0658942

Newton polynom:
4.99983x^0 + -7.56663x^1 + 5.58847x^2 + -1.37583x^3
f(x*) = 2.20059
error = 0.114938
```

Рис. 1: Вывод программы

3 Исходный код

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

using namespace std;


class Polynom {
    private:
        vector<double> _poly;
    public:
    Polynom(int n) {
        _poly.resize(n);
    }
    Polynom() {}
    Polynom(const vector<double> &v){
        _poly.resize(v.size());
        for (int i = 0; i < v.size(); ++i){
            _poly[i] = v[i];
        }
    }

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

    double & operator[](int i) {
        return _poly[i];
    }

    void push_back(double k){
        _poly.push_back(k);
    }

    friend Polynom operator+(Polynom &lhs, Polynom &rhs) {
        int n_max = max(lhs.size(), rhs.size());
        int n_min = min(lhs.size(), rhs.size());
        Polynom res(n_max);
        for (int i = 0; i < n_min; ++i) {
            res[i] = lhs[i] + rhs[i];
        }
        for (int i = n_min; i < n_max; ++i) {
            if (lhs.size() > rhs.size()) {
                res[i] = lhs[i];
            } else {
                res[i] = rhs[i];
            }
        }
        return res;
    }

    friend Polynom operator*(Polynom &lhs, Polynom &rhs) {
        Polynom res(lhs.size() + rhs.size() - 1);
        for (int i = 0; i < rhs.size(); ++i) {
            for (int j = 0; j < lhs.size(); ++j) {
                res[i + j] += lhs[j] * rhs[i];
            }
        }
        return res;
```

```
    }

    friend Polynom operator*(Polynom &lhs, double rhs) {
        Polynom res(lhs.size());
        for (int i = 0; i < lhs.size(); ++i) {
            res[i] = lhs[i] * rhs;
        }
        return res;
    }
};

ostream& operator<<(ostream& stream, Polynom poly)
{
    for (int i = 0; i < poly.size(); ++i) {
        if (i != poly.size() - 1)
            stream << poly[i] << "x^" << i << " + ";
        else
            stream << poly[i] << "x^" << i << "\n";
    }
    return stream;
}

Polynom lagrange_polynom(vector<double> &x, vector<double> &y) {
    int n = x.size();
    Polynom L(n);
    for (int i = 0; i < n; ++i) {
        Polynom li;
        li.push_back(1);
        double c = 1;
        for (int j = 0; j < n; ++j) {
            if (i != j) {
                Polynom d({(-1) * x[j], 1});
                li = li*d;
                c *= (x[i] - x[j]);
            }
        }
        L = L + li* (y[i]/c);
    }
    return L;
}

Polynom newton_polynom(vector<double> &x, vector<double> &y) {
    int n = x.size();
    vector<vector<double>> table(n, vector<double>(n));
    for (int i = 0; i < n; ++i) {
        table[i][0] = y[i];
    }
    for (int j = 1; j < n; ++j) {
        for (int i = 0; i < n - j; ++i) {
            table[i][j] = (table[i][j - 1] - table[i + 1][j - 1]) / (x[i] - x[i + j]);
        }
    }
    Polynom P(n);
    Polynom k;
    for (int i = 0; i < n; ++i) {
        if (i == 0) {
            k.push_back(1);
        } else {
            Polynom d({(-1) * x[i - 1], 1});
            k = k * d;
        }
        P = (P + k * table[0][i]);
```

```
    }
    return P;
}

double f(double x) {
    return 1/tan(x) + x;
}

int main() {
    double pi = 2*acos(0.0);
    double X = 3*pi/16;
    vector<double> x_a = {pi/8, 2*pi/8, 3*pi/8, 4*pi/8};
    vector<double> y_a;
    for (int i = 0; i < x_a.size(); ++i) {
        y_a.push_back(f(x_a[i]));
    }

    ofstream fout("answer.txt");
    fout << "A:\n";
    fout << "Lagrange polynom:\n";
    Polynom polynom_l_a = lagrange_polynom(x_a, y_a);
    fout << polynom_l_a;

    double res_a = 0;
    for (int i = 0; i < polynom_l_a.size(); ++i) {
        res_a += polynom_l_a[i] * pow(X, i);
    }
    fout << "f(x*) = " << res_a << "\n";
    fout << "error = " << abs(f(X) - res_a) << "\n";


    fout << "\nNewton polynom\n";
    Polynom polynom_n_a = newton_polynom(x_a, y_a);
    fout << polynom_n_a;
    res_a = 0;
    for (int i = 0; i < polynom_n_a.size(); ++i) {
        res_a += polynom_n_a[i] * pow(X, i);
    }
    fout << "f(x*) = " << res_a << "\n";
    fout << "error = " << abs(f(X) - res_a) << "\n\n";




    fout << "B:\n";

    vector<double> x_b = {pi/8, pi/3, 3*pi/8, pi/2};
    vector<double> y_b;
    for (int i = 0; i < x_b.size(); ++i) {
        y_b.push_back(f(x_b[i]));
    }

    fout << "Lagrange polynom:\n";
    Polynom polynom_l_b = lagrange_polynom(x_a, y_a);
    fout << polynom_l_b;

    double res_b = 0;
    for (int i = 0; i < polynom_l_b.size(); ++i) {
        res_b += polynom_l_b[i] * pow(X, i);
    }
    fout << "f(x*) = " << res_b << "\n";
    fout << "error = " << abs(f(X) - res_b) << "\n";
```

```


    fout << "\nNewton polynom:\n";
    Polynom polynom_n_b = newton_polynom(x_b, y_b);
    fout << polynom_n_b;
    res_b = 0;
    for (int i = 0; i < polynom_n_b.size(); ++i) {
        res_b += polynom_n_b[i] * pow(X, i);
    }
    fout << "f(x*) = " << res_b << "\n";
    fout << "error = " << abs(f(X) - res_b) << "\n";
    return 0;
}
```

2 Кубический сплайн

1 Постановка задачи

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: 15

$$X^* = 2.66666667$$

i	0	1	2	3	4
x_i	1.0	1.9	2.8	3.7	4.6
f_i	2.8069	1.8279	1.6091	1.5713	1.5663

Рис. 2: Условие

2 Результаты работы

```
answer.txt X
Lab3 > lab3_2 > answer.txt
Cubic spline:
S(x) = 1.8279 + -0.662706 * (x - 1.9) + 0.708452 * (x - 1.9)^2 + -0.26915 * (x - 1.9)^3
f(x*) = 1.61495
```

Рис. 3: Вывод программы

3 Исходный код

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

using namespace std;

class matrix
{
    private:
        vector <vector <double>> _obj;
    public:
        int cols = 0, rows = 0;

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

        vector <double> &operator[](int i)
        {
            return _obj[i];
        }

        operator double()
        {
            return _obj[0][0];
        }

};

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

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


matrix solve_SLE (matrix& A, matrix& b)
{
    int n = A.cols;
    vector <double> p(n), q(n);
```

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

double spline(vector<double> &x, vector<double> &y, double dot) {
    int n = x.size() - 1;
    vector<double> a(n), b(n), c(n), d(n), 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) {
        A[i][i] = 2 * (h[i] + h[i + 1]);
        if (i > 0)
            A[i][i - 1] = h[i];
        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 C = solve_SLE(A,B);
    for (int i = 0; i < n; ++i) {
        if (i == 0)
            c[i] = 0;
        else
            c[i] = C[i - 1][0];
    }
    for (int i = 0; i < n; ++i) {
        a[i] = y[i];
        if (i < n - 1) {
            b[i] = (y[i + 1] - y[i]) / h[i] - 1.0 / 3 * h[i] * (c[i + 1] + 2 * c[i]);
            d[i] = (c[i + 1] - c[i]) / (3 * h[i]);
        } else {
            b[i] = (y[i + 1] - y[i]) / h[i] - 2.0 / 3 * h[i] * c[i];
            d[i] = (-1) * c[i] / (3 * h[i]);
        }
    }
    auto it = lower_bound(x.begin(), x.end(), dot);
    int interval = it - x.begin() - 1;
    if (interval == -1)
        interval = 0;
    ofstream fout("answer.txt");
    fout<< "Cubic spline:\n";
    fout << "S(x) = " << a[interval] << " + " << b[interval] << " * (x - " << x[interval] << ") + " << c[interval] << " * (x - " << x[interval] << ")^2 + " << d[interval] << " * (x - " << x[interval] << ")^3\n";
    double res = a[interval] + b[interval] * (dot - x[interval]) + c[interval] * pow((dot - x[interval]), 2) + d[interval] * pow((dot - x[interval]), 3);
    fout << "f(x*) = " << res;
    return res;
```

```
}

int main() {
    vector<double> x = {1.0, 1.9, 2.8, 3.7, 4.6};
    vector<double> y = {2.8069, 1.8279, 1.6091, 1.5713, 1.5663};
    double X = 2.66666667;
    double res = spline(x, y, X);
    return 0;
}
```

3 Приближающие многочлены

1 Постановка задачи

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: 15

i	0	1	2	3	4	5
x_i	1.0	1.9	2.8	3.7	4.6	5.5
y_i	3.4142	2.9818	3.3095	3.8184	4.3599	4.8318

Рис. 4: Условие

2 Результаты работы

```
answer.txt
Lab3 > lab3_3 > answer.txt
Least squares polynom [1st power]:
2.60728x^0 + 0.382267x^1
error = 0.637538

Least squares polynom [2nd power]:
3.20861x^0 + -0.0811603x^1 + 0.0715993x^2
error = 0.490258
```

Рис. 5: Вывод программы

3 Исходный код

Основной код:

```
#include <cmath>
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;

class Matrix {
private:
    int rows_, cols_;
    vector<vector<double>> matrix_;
    vector<int> swp_;

    void SwapMatrix(Matrix &other) {
        swap(rows_, other.rows_);
        swap(cols_, other.cols_);
        swap(matrix_, other.matrix_);
    }

public:
    Matrix(int rows, int cols) {
        rows_ = rows;
        cols_ = cols;
        matrix_.resize(rows_);
        for (int i = 0; i < rows_; ++i) {
            matrix_[i].resize(cols_);
        }
    }
    Matrix(const Matrix &other) : Matrix(other.rows_, other.cols_) {
        for (int i = 0; i < rows_; ++i) {
            for (int j = 0; j < cols_; ++j) {
                matrix_[i][j] = other.matrix_[i][j];
            }
        }
    }
    Matrix() : Matrix(1, 1) {}
    Matrix(Matrix &&other) {
        this->SwapMatrix(other);
        other.rows_ = 0;
        other.cols_ = 0;
    }

    int GetRows() const { return rows_; }
    int GetCols() const { return cols_; }
    const vector<int> &GetSwp() const { return swp_; }

    void MulNumber(const double num) {
        for (int i = 0; i < rows_; ++i) {
            for (int j = 0; j < cols_; ++j) {
                matrix_[i][j] *= num;
            }
        }
    }

    Matrix MulMatrixReturn(const double num) {
        Matrix res = *this;
        res.MulNumber(num);
        return res;
    }

```

```
    void MulMatrix(const Matrix &other) {
        Matrix tmp(rows_, other.cols_);
        for (int i = 0; i < rows_; ++i) {
            for (int j = 0; j < other.cols_; ++j) {
                for (int k = 0; k < cols_; ++k)
                    tmp.matrix_[i][j] += matrix_[i][k] * other.matrix_[k][j];
            }
        }
        this->SwapMatrix(tmp);
    }

    Matrix MulMatrixReturn(const Matrix &other) {
        Matrix res = *this;
        res.MulMatrix(other);
        return res;
    }

    Matrix Transpose() const {
        Matrix result(cols_, rows_);
        for (int i = 0; i < result.rows_; ++i) {
            for (int j = 0; j < result.cols_; ++j) {
                result.matrix_[i][j] = matrix_[j][i];
            }
        }
        return result;
    }
    pair<Matrix, Matrix> LU(Matrix & L, Matrix & U) {
        swp_.clear();
        int n = this->GetRows();
        for (int k = 0; k < n; ++k) {
            int index = k;
            for (int i = k + 1; i < n; ++i) {
                if (abs(U(i, k)) > abs(U(index, k))) {
                    index = i;
                }
            }
            swap(U(k), U(index));
            swap(L(k), L(index));
            swp_.push_back(index);
            for (int i = k + 1; i < n; ++i) {
                double m = U(i, k) / U(k, k);
                L(i, k) = m;
                for (int j = k; j < n; ++j) {
                    U(i, j) -= m * U(k, j);
                }
            }
        }
        for (int i = 0; i < n; ++i) {
            L(i, i) = 1;
        }
        return {L, U};
    }
    Matrix Solve(Matrix &C) {
        Matrix U(*this);
        Matrix L(this->GetCols(), this->GetCols());
        LU(L,U);
        Matrix B(C);
        vector<int> swp = this->GetSwp();
        for (int i = 0; i < swp.size(); ++i) {
            swap(B(i), B(swp[i]));
        }
        int n = this->GetRows();
```

```cpp

        Matrix Z(n, 1);
        for (int i = 0; i < n; ++i) {
            Z(i, 0) = B(i, 0);
            for (int j = 0; j < i; ++j) {
                Z(i, 0) -= L(i, j) * Z(j, 0);
            }
        }

        Matrix X(n, 1);
        for (int i = n - 1; i >= 0; --i) {
            X(i, 0) = Z(i, 0);
            for (int j = i + 1; j < n; ++j) {
                X(i, 0) -= U(i, j) * X(j, 0);
            }
            X(i, 0) = X(i, 0) / U(i, i);
        }
        return X;
    }
    double &operator()(int i, int j) {
        return matrix_[i][j];
    }
    vector<double> &operator()(int row) { return matrix_[row]; }
};

istream& operator>>(istream& stream, Matrix& m)
{
    for (int i = 0; i < m.GetRows(); i++)
    {
        for (int j = 0; j < m.GetCols(); j++)
            stream >> m(i, j);
    }
    return stream;
}

ostream& operator<<(ostream& stream, Matrix m)
{
    for (int i = 0; i < m.GetRows(); i++)
    {
        for (int j = 0; j < m.GetCols(); j++)
            stream << m(i, j) << ' ';
        stream << '\n';
    }
    return stream;
}

ostream& operator<<(ostream& stream, vector<double> v)
{
    for (int i = 0; i < v.size(); ++i) {
        stream << v[i] << " ";
    }
    return stream;
}

vector<double> least_squares_method(vector<double> &x, vector<double> &y, int m) {
    int n = x.size();
    m = m+1;
    Matrix Y(n, 1), Phi(n, m);
    for (int i = 0; i < n; ++i) {
        Y(i, 0) = y[i];
    }
    for (int i = 0; i < n; ++i) {
```

```
        for (int j = 0; j < m; ++j) {
            Phi(i, j) = pow(x[i], j);
        }
    }
    Matrix Phi_T = Phi.Transpose();
    Matrix B = Phi_T.MulMatrixReturn(Y);
    Matrix res = (Phi_T.MulMatrixReturn(Phi)).Solve(B);
    vector<double> polynom;
    for (int i = 0; i < res.GetRows(); ++i) {
        for (int j = 0; j < res.GetCols(); ++j) {
            polynom.push_back(res(i, j));
        }
    }
    return polynom;
}

double get_error(vector<double> &x, vector<double> &y, vector<double> &p) {
    double f_x = 0, error = 0;
    for (int i = 0; i < x.size(); ++i) {
        f_x = 0;
        for (int j = 0; j < p.size(); ++j) {
            f_x += p[j] * pow(x[i], j);
        }
        error += (f_x - y[i]) * (f_x - y[i]);
    }
    return error;
}


int main() {
    vector<double> x = {1.0, 1.9, 2.8, 3.7, 3.6, 5.5};
    vector<double> y = {3.4142, 2.9818, 3.3095, 3.8184, 4.3599, 4.8318};
    ofstream fout("answer.txt");
    ofstream fout_py_args("py_args.txt");
    fout_py_args << x << "\n";
    fout_py_args << y << "\n";

    vector<double> polynom = least_squares_method(x, y, 1);
    fout << "Least squares polynom [1st power]:\n";
    for (int i = 0; i < polynom.size(); ++i) {
        if (i != polynom.size() - 1)
            fout << polynom[i] << "x^" << i << " + ";
        else
            fout << polynom[i] << "x^" << i << "\n";
    }
    double error = get_error(x, y, polynom);
    fout << "error = " << error << "\n\n";
    fout_py_args << polynom << "\n";


    polynom.clear();
    polynom = least_squares_method(x, y, 2);
    fout << "Least squares polynom [2nd power]:\n";
    for (int i = 0; i < polynom.size(); ++i) {
        if (i != polynom.size() - 1)
            fout << polynom[i] << "x^" << i << " + ";
        else
            fout << polynom[i] << "x^" << i << "\n";
    }
    error = get_error(x, y, polynom);
    fout << "error = " << error << "\n\n";
    fout_py_args << polynom << "\n";
```

```
    fout_py_args.close();
    system("python show_plot.py");
    return 0;
}
```

Вывод графика:

```
import matplotlib.pyplot as plt
import numpy as np

file = open ("./py_args.txt", "r")

x = [float(x) for x in file.readline().split()]
y = [float(x) for x in file.readline().split()]
poly_1 = [float(x) for x in file.readline().split()]
poly_2 = [float(x) for x in file.readline().split()]

x_1 = np.linspace(x[0], x[-1], 100)
y_1 = poly_1[0] + poly_1[1]*x_1

x_2 = np.linspace(x[0], x[-1], 100)
y_2 = poly_2[0] + poly_2[1]*x_1 + poly_2[2]*x_1*x_1
plt.plot(x, y, 'ro')
plt.plot(x_1, y_1)
plt.plot(x_2, y_2)

plt.xlabel('x')
plt.ylabel('y')

plt.title('My graph')

plt.show()
file.close()
```

4 Вычисление производных

1 Постановка задачи

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

Вариант: 15

$X^* = 0.4$

i	0	1	2	3	4
x_i	0.0	0.2	0.4	0.6	0.8
y_i	1.0	1.4214	1.8918	2.4221	3.0255

Рис. 6: Условие

2 Результаты работы

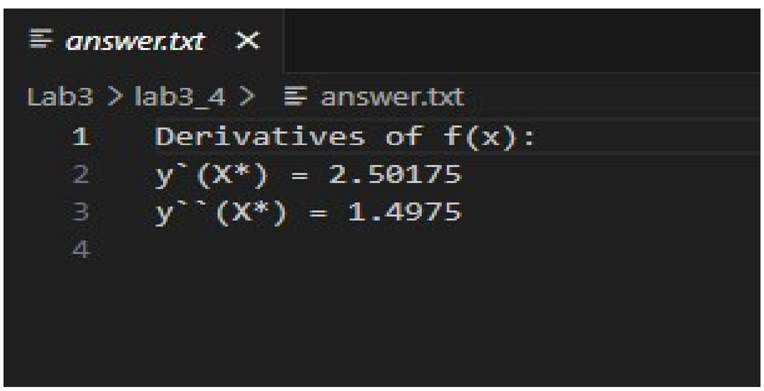


Рис. 7: Вывод программы

3 Исходный код

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

using namespace std;

double df(vector<double> &x, vector<double> &y, double X) {
    auto lb = lower_bound(x.begin(), x.end(), X);
    int def = lb - x.begin() - 1;
    if (def == -1) def = 0;
    double a = (y[def + 1] - y[def]) / (x[def + 1] - x[def]),
    b = ((y[def + 2] - y[def + 1]) / (x[def + 2] - x[def + 1]) - (y[def + 1] - y[def]) / (x[def + 1] - x[def])) / (x[def + 2] - x[def]);
    return (a + b * (2 * X - x[def] - x[def + 1]));
}

double ddf(vector<double> &x, vector<double> &y, double X) {
    auto lb = lower_bound(x.begin(), x.end(), X);
    int def = lb - x.begin() - 1;
    if (def == -1) def = 0;
    return (2 * (((y[def + 2] - y[def + 1]) / (x[def + 2] - x[def + 1])) - ((y[def + 1] - y[def])) / (x[def + 1] - x[def]))) / (x[def + 2] - x[def]);
}

int main() {
    vector<double> x = {0.0, 0.2, 0.4, 0.6, 0.8};
    vector<double> y = {1.0, 1.4214, 1.8918, 2.4221, 3.0255};
    double X = 0.4;
    ofstream fout("answer.txt");
    fout << "Derivatives of f(x):\n";
    fout << "y‘(X*) = " << df(x, y, X) << "\n";
    fout << "y‘‘(X*) = " << ddf(x, y, X) << "\n";
    return 0;
}
```

5 Вычисление интегралла

1 Постановка задачи

Вычислить определенный интеграл $\int\limits_{X_0}^{X_1} y dx$, методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга:

Вариант: 15

$$y = \frac{x}{x^4 + 81} X_0 = 0, X_k = 2, h_1 = 0.5, h_2 = 0.25$$

2 Результаты работы

```
lab3_5 > answer.txt
Rectangle method:
F = 0.0233265; h = 0.5; error = 2.31465e-11
F = 0.0232577; h = 0.25; error = 6.88126e-05

Trapeze method:
F = 0.0230508; h = 0.5; error = 0.000137966
F = 0.0231887; h = 0.25; error = 1.34732e-07

Simpson method:
F = 0.0232346; h = 0.5 error = 6.87099e-08
F = 0.0232347; h = 0.25 error = 6.70995e-11
```

Рис. 8: Вывод программы

3 Исходный код

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

using namespace std;

double f(double x) {
    return (x /(pow(x,4)+81));
}

double RungeRombergMethod(double F1, double h1, double F2, double h2, double k) {
    return (F1 + (F1 - F2) / (pow(h2 / h1, k) - 1));
}


double RectangleMethod(double a, double b, double step) {
    double res = 0;
    double x_prev = a;
    for (double cur = a + step; cur <= b; cur += step) {
        res += step * f((x_prev + cur) / 2);
        x_prev = cur;
    }
    return res;
}

double SimpsonMethod(double a, double b, double step) {
    double res = 0;
    step = step/2;
    for (double cur = a + 2 * step; cur <= b; cur += 2 * step) {
        res += step * (f(cur - 2 * step) + 4 * f(cur - step) + f(cur));
    }
    return (1.0 / 3 * res);
}

double TrapezeMethod(double a, double b, double step) {
    double res = 0;
    double x_prev = a;
    for (double cur = a + step; cur <= b; cur += step) {
        res += step * (f(cur) + f(x_prev));
        x_prev = cur;
    }
    return 0.5 * res;
}

int main() {
    double X_0 = 0;
    double X_k = 2;
    double h1 = 0.5, h2 = 0.25;
    ofstream fout("answer.txt");

    fout << "Rectangle method:\n";
    double F_h1 = RectangleMethod(X_0, X_k, h1);
    double F_h2 = RectangleMethod(X_0, X_k, h2);
    double F = RungeRombergMethod(F_h1, F_h2, h1, h2, 10);
    fout << "F = " << F_h1 << "; h = " << h1 << "; error = " << abs(F - F_h1) << "\n";
    fout << "F = " << F_h2 << "; h = " << h2 << "; error = " << abs(F - F_h2) << "\n";

    fout << "\nTrapeze method:\n";
    F_h1 = TrapezeMethod(X_0, X_k, h1);
```

```
    F_h2 = TrapezeMethod(X_0, X_k, h2);
    F = RungeRombergMethod(F_h1, h1, F_h2, h2, 10);
    fout << "F = " << F_h1 << "; h = " << h1 << "; error = " << abs(F - F_h1) << "\n";
    fout << "F = " << F_h2 << "; h = " << h2 << "; error = " << abs(F - F_h2) << "\n ";

    fout << "\nSimpson method:\n";
    F_h1 = SimpsonMethod(X_0, X_k, h1);
    F_h2 = SimpsonMethod(X_0, X_k, h2);
    F = RungeRombergMethod(F_h1, h1, F_h2, h2, 10);
    fout << "F = " << SimpsonMethod(X_0, X_k, h1) << "; h = " << h1 << " error = " << abs(F - F_h1) << "\n"
        ;
    fout << "F = " << SimpsonMethod(X_0, X_k, h2) << "; h = " << h2 << " error = " << abs(F - F_h2) << "\n
        ";

    return 0;
}
```