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. К. Сайфуллин
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: 19

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

а)

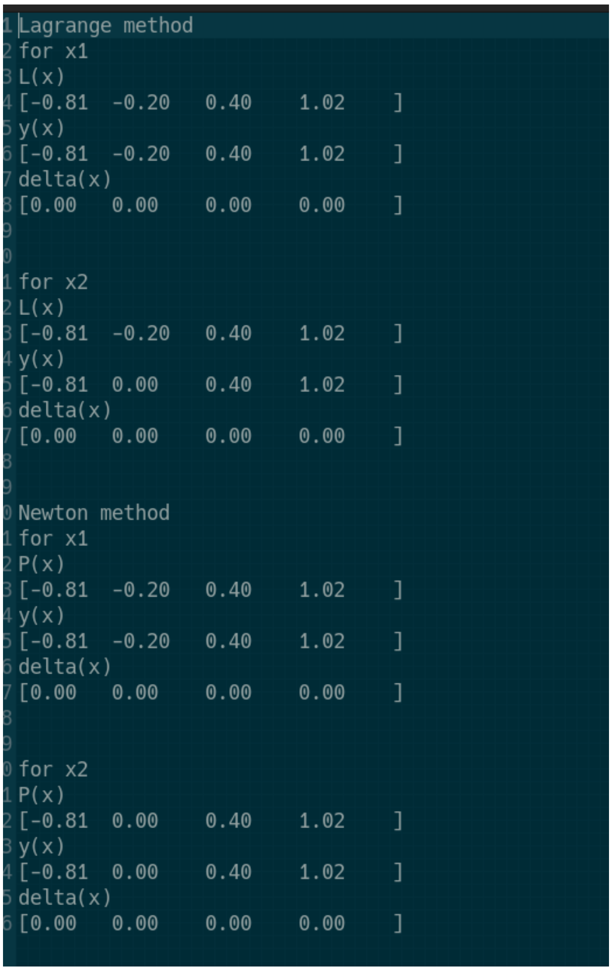
$$X_i = -0.4, -0.1, 0.2, 0.5 \tag{2}$$

б)

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

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

2 Результаты работы



```
Lagrange method
for x1
L(x)
[-0.81  -0.20   0.40    1.02    ]
y(x)
[-0.81  -0.20   0.40    1.02    ]
delta(x)
[0.00   0.00    0.00    0.00    ]


for x2
L(x)
[-0.81  -0.20   0.40    1.02    ]
y(x)
[-0.81  0.00    0.40    1.02    ]
delta(x)
[0.00   0.00    0.00    0.00    ]


Newton method
for x1
P(x)
[-0.81  -0.20   0.40    1.02    ]
y(x)
[-0.81  -0.20   0.40    1.02    ]
delta(x)
[0.00   0.00    0.00    0.00    ]


for x2
P(x)
[-0.81  0.00    0.40    1.02    ]
y(x)
[-0.81  0.00    0.40    1.02    ]
delta(x)
[0.00   0.00    0.00    0.00    ]
```

Рис. 1: Вывод в консоли

3 Исходный код

Lab3.1.cpp

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

using namespace std;

```

```
double func(double x1, double x2){
    return ((asin(x1) + x1) - (asin(x2) + x2)) / (x1 - x2);
}

double func1(double x1, double x2, double x3){
    return (func(x1,x2) - func(x2, x3)) / (x1 - x3);
}

double func2(double x1, double x2, double x3, double x4){
    return (func1(x1, x2, x3) - func1(x2, x3, x4)) / (x1 - x4);
}

vector <double> omega_values(vector <double> x){
    vector <double> omega(4,0);
    omega[0] = (x[0] - x[1]) * (x[0] - x[2]) * (x[0] - x[3]);
    omega[1] = (x[1] - x[0]) * (x[1] - x[2]) * (x[1] - x[3]);
    omega[2] = (x[2] - x[0]) * (x[2] - x[1]) * (x[2] - x[3]);
    omega[3] = (x[3] - x[0]) * (x[3] - x[1]) * (x[3] - x[2]);
    return omega;
}

void Lagrange_method(vector <double> x, double X, vector <double> &L, vector <double>
    &y, vector <double> &delta){
    vector <vector <double>> table(5, vector<double>(4,0));
    vector <double> omega = omega_values(x);
    for (int i = 0; i < x.size(); i++){
        table[0][i] = x[i];
        table[1][i] = asin(x[i]) + x[i];
        table[2][i] = omega[i];
        table[3][i] = (asin(x[i]) + x[i]) / omega[i];
        table[4][i] = X - x[i];
    }

    for (int i = 0; i < x.size(); i++){
        L[i] = (table[3][0] * (x[i] - table[0][1]) * (x[i] - table[0][2]) * (x[i] -
            table[0][3]) \
         + table[3][1] * (x[i] - table[0][0]) * (x[i] - table[0][2]) * (x[i] - table
            [0][3]) \
         + table[3][2] * (x[i] - table[0][0]) * (x[i] - table[0][1]) * (x[i] - table
            [0][3]) \
         + table[3][3] * (x[i] - table[0][0]) * (x[i] - table[0][1]) * (x[i] - table
            [0][2]));
    }

    for (int i = 0; i < x.size(); i++){
        y[i] = asin(x[i]) + x[i];
    }

    for (int i = 0; i < x.size(); i++){
```

```
        delta[i] = fabs(y[i] - L[i]);
    }
}


void Newton_method(vector <double> x, double X, vector <double> &P, vector <double> &y
    , vector <double> &delta){
    vector <vector <double>> table(5, vector<double>(4,0));
    for (int i = 0; i < x.size(); i++){
        table[0][i] = x[i];
        table[1][i] = asin(x[i]) + x[i];
        if (i < 3){
            table[2][i] = func(x[i], x[i+1]);
        }
        if (i < 2){
            table[3][i] = func1(x[i], x[i+1], x[i+2]);
        }
    }
    table[4][0] = func2(x[0], x[1], x[2], x[3]);


    for (int i = 0; i < x.size(); i++){
        P[i] = (table[1][0] + table[2][0] * (x[i] - table[0][0]) + table[3][0] * (x[i]
            - table[0][0]) * (x[i] - table[0][1]) \
         + table[4][0] * (x[i] - table[0][0]) * (x[i] - table[0][1]) * (x[i] - table
             [0][2]));
    }

    for (int i = 0; i < x.size(); i++){
        y[i] = asin(x[i]) + x[i];
    }

    for (int i = 0; i < x.size(); i++){
        delta[i] = fabs(y[i] - P[i]);
    }
}

int main(){
    ofstream fout("answer1.txt");
    fout.precision(2);
    fout << fixed;
    int n = 4;
    vector <double> x1 = {-0.4, -0.1, 0.2, 0.5};
    vector <double> x2 = {-0.4, 0, 0.2, 0.5};
    double root = 0.1;
    vector <double> L(n,0), y(n,0), delta(n,0), P(n,0);
    fout << "Lagrange method" << endl;
    Lagrange_method(x1, root, L, y, delta);
    fout << "for x1" << endl << "L(x)\n" << "[";
```

```
    for (int i = 0; i < L.size(); i++) fout << L[i] << "\t";
    fout << "]\n" << "y(x)\n" << "[";
    for (int i = 0; i < y.size(); i++) fout << y[i] << "\t";
    fout << "]\n" << "delta(x)\n" << "[";
    for (int i = 0; i < delta.size(); i++) fout << delta[i] << "\t";
    fout << "]\n";
    Lagrange_method(x2, root, P, y, delta);
    fout << "\n\nfor x2" << endl << "L(x)\n" << "[";
    for (int i = 0; i < L.size(); i++) fout << L[i] << "\t";
    fout << "]\n" << "y(x)\n" << "[";
    for (int i = 0; i < y.size(); i++) fout << y[i] << "\t";
    fout << "]\n" << "delta(x)\n" << "[";
    for (int i = 0; i < delta.size(); i++) fout << delta[i] << "\t";
    fout << "]\n";
    fout << "\n\nNewton method" << endl;
    Newton_method(x1, root, P, y, delta);
    fout << "for x1" << endl << "P(x)\n" << "[";
    for (int i = 0; i < P.size(); i++) fout << P[i] << "\t";
    fout << "]\n" << "y(x)\n" << "[";
    for (int i = 0; i < y.size(); i++) fout << y[i] << "\t";
    fout << "]\n" << "delta(x)\n" << "[";
    for (int i = 0; i < delta.size(); i++) fout << delta[i] << "\t";
    fout << "]\n";
    Newton_method(x2, root, P, y, delta);
    fout << "\n\nfor x2" << endl << "P(x)\n" << "[";
    for (int i = 0; i < P.size(); i++) fout << P[i] << "\t";
    fout << "]\n" << "y(x)\n" << "[";
    for (int i = 0; i < y.size(); i++) fout << y[i] << "\t";
    fout << "]\n" << "delta(x)\n" << "[";
    for (int i = 0; i < delta.size(); i++) fout << delta[i] << "\t";
    fout << "]\n";
}
```

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: 19

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

i	0	1	2	3	4
x_i	-0.4	-0.1	0.2	0.5	0.8
f_i	-0.81152	-0.20017	0.40136	1.0236	1.7273

5 Результаты работы

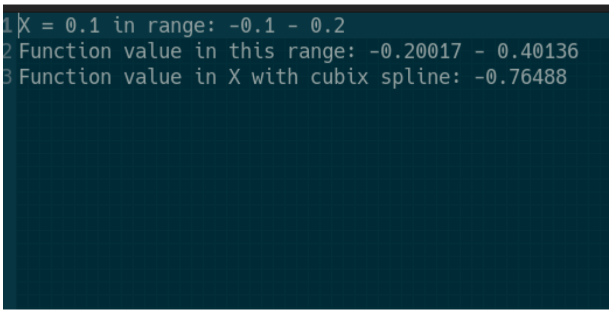


Рис. 2: Вывод в консоли

6 Исходный код

Lab3.2.cpp

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

using namespace std;

vector<double> solve_linear_system(vector<vector<double>>& A, vector<double>& b) {
    int n = A.size();


    for (int i = 0; i < n; i++) {
        int max_row = i;
        for (int k = i + 1; k < n; k++) {
            if (abs(A[k][i]) > abs(A[max_row][i])) {
                max_row = k;
            }
        }
        swap(A[i], A[max_row]);
        swap(b[i], b[max_row]);
        for (int k = i + 1; k < n; k++) {
            double factor = A[k][i] / A[i][i];
            for (int j = i; j < n; j++) {
                A[k][j] -= factor * A[i][j];
            }
            b[k] -= factor * b[i];
        }
```

```
    }

    vector<double> x(n);
    for (int i = n - 1; i >= 0; i--) {
        x[i] = b[i];
        for (int j = i + 1; j < n; j++) {
            x[i] -= A[i][j] * x[j];
        }
        x[i] /= A[i][i];
    }

    return x;
}


double Cubic_spline(vector<double>& x, vector<double>& y, double X) {
    vector<double> A(3, 0);
    vector<vector<double>> B(3, vector<double>(3, 0));
    vector<vector<double>> results(4, vector<double>(4, 0));
    vector<double> roots(3, 0);
    double f = 0;

    for (int i = 2; i < 5; i++) {
        A[i - 2] = (3 * ((y[i] - y[i - 1]) / (x[i] - x[i - 1]) +
                         (y[i - 1] - y[i - 2]) / (x[i - 1] - x[i - 2])));
        for (int j = 0; j < B.size(); j++) {
            for (int k = 0; k < B[0].size(); k++) {
                if (j == k) {
                    B[j][k] = 2 * ((x[i - 1] - x[i - 2]) + (x[i] - x[i - 1]));
                } else if (j < k && (k != B.size() - 1 || j != 0)) {
                    B[j][k] = (x[i] - x[i - 1]);
                } else if (j > k && (j != B.size() - 1 || k != 0)) {
                    B[j][k] = (x[i - 1] - x[i - 2]);
                }
            }
        }
    }

    roots = solve_linear_system(B,A);
    roots.insert(roots.begin(), 0);

    for (int i = 0; i < results.size(); i++) {
        for (int j = 0; j < results[0].size(); j++) {
            if (i != results.size() - 1) {
                if (j == 0) {
                    results[i][j] = y[i];
                }
                if (j == 1) {
                    results[i][j] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]) - (
```

```
                            (x[i + 1] - x[i]) * roots[i + 1] + 2 * roots[i]) / 3;
                }
                if (j == 2) {
                    if (i == 0) {
                        results[i][j] = 0;
                    } else {
                        results[i][j] = roots[i];
                    }
                }
                if (j == 3) {
                    results[i][j] = (roots[i + 1] - roots[i]) / (3 * (x[i + 1] - x[i]));
                }
            } else {
                if (j == 0) {
                    results[i][j] = y[i];
                }
                if (j == 1) {
                    results[i][j] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]) - (2 * (x[i +
                        1] - x[i]) * roots[i]) / 3;
                }
                if (j == 2) {
                    results[i][j] = roots[i];
                }
                if (j == 3) {
                    results[i][j] = -roots[i] / (3 * (x[i + 1] - x[i]));
                }
            }
        }
    }

    f = results[1][0] + results[1][1] * (X - x[1]) + results[1][2] * pow((X - x[2]), 2)
         + results[1][3] * pow((X - x[3]), 3);

    return f;
}


int main() {
    ofstream fout("answer2.txt");
    vector<double> x = {-0.4, -0.1, 0.2, 0.5, 0.8};
    vector<double> y = {-0.81152, -0.20017, 0.40136, 1.0236, 1.7273};
    double X = 0.1;
    fout << "X = " << X << " in range: " << x[1] << " - " << x[2] << endl;
    fout << "Function value in this range: " << y[1] << " - " << y[2] << endl;
    fout << "Function value in X with cubix spline: " << Cubic_spline(x, y, X) << endl;

    return 0;
}
```

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: 19

i	0	1	2	3	4	5
x_i	-0.7	-0.4	-0.1	0.2	0.5	0.8
y_i	-1.4754	-0.81152	-0.20017	0.40136	1.0236	1.7273

8 Результаты работы

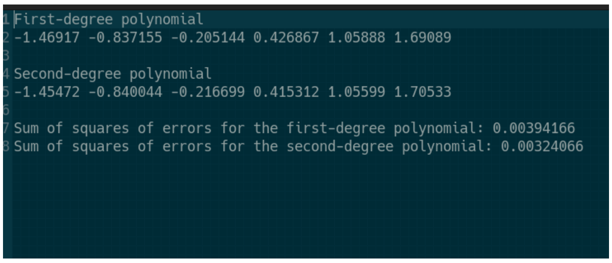


Рис. 3: Вывод в консоли

9 Исходный код

Lab3.3.cpp

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

using namespace std;

vector<double> solve_linear_system(vector<vector<double>>& A, vector<double>& b) {
    int n = A.size();


    for (int i = 0; i < n; i++) {
        int max_row = i;
        for (int k = i + 1; k < n; k++) {
            if (abs(A[k][i]) > abs(A[max_row][i])) {
                max_row = k;
            }
        }
        swap(A[i], A[max_row]);
        swap(b[i], b[max_row]);
        for (int k = i + 1; k < n; k++) {
            double factor = A[k][i] / A[i][i];
            for (int j = i; j < n; j++) {
                A[k][j] -= factor * A[i][j];
            }
            b[k] -= factor * b[i];
        }
    }

```

```
    vector<double> x(n);
    for (int i = n - 1; i >= 0; i--) {
        x[i] = b[i];
        for (int j = i + 1; j < n; j++) {
            x[i] -= A[i][j] * x[j];
        }
        x[i] /= A[i][i];
    }

    return x;
}

pair<vector<double>, vector<double>> method_least_squares(vector<double> x, vector<double> y) {
    vector <vector <double>> A(2, vector<double>(3,0));
    double sumx = 0, sumy = 0, sumx2 = 0, sumxy = 0;
    for (int i = 0; i < x.size(); ++i) {
        sumx += x[i];
        sumy += y[i];
        sumx2 += pow(x[i], 2);
        sumxy += x[i] * y[i];
    }
    A[0][0] = x.size();
    A[0][1] = sumx;
    A[0][2] = sumy;
    A[1][0] = sumx;
    A[1][1] = sumx2;
    A[1][2] = sumxy;

    vector <vector <double>> first_slice_A = {{A[0][0], A[0][1]}, {A[1][0], A[1][1]}};
    vector <double> second_slice_A = {A[0][2], A[1][2]};
    vector <double> roots1 = solve_linear_system(first_slice_A, second_slice_A);

    vector <double> y1 (x.size(), 0);
    for (int i = 0; i<y1.size(); i++){
        y1[i] = roots1[0] + x[i] * roots1[1];
    }

    vector <vector <double>> A2(3, vector<double>(4,0));
    sumx = 0, sumy = 0, sumx2 = 0, sumxy = 0;
    double sumx3 = 0, sumx4 = 0, sumx2y = 0;
    for (int i = 0; i < x.size(); ++i) {
        sumx += x[i];
        sumy += y[i];
        sumx2 += pow(x[i],2);
        sumx3 += pow(x[i],3);
        sumx4 += pow(x[i],4);
        sumxy += x[i] * y[i];
        sumx2y += pow(x[i],2) * y[i];
```

```
    }
    A2[0][0] = x.size();
    A2[0][1] = sumx;
    A2[0][2] = sumx2;
    A2[0][3] = sumy;
    A2[1][0] = sumx;
    A2[1][1] = sumx2;
    A2[1][2] = sumx3;
    A2[1][3] = sumxy;
    A2[2][0] = sumx2;
    A2[2][1] = sumx3;
    A2[2][2] = sumx4;
    A2[2][3] = sumx2y;

    vector <vector <double>> first_slice_A2 = {{A2[0][0], A2[0][1], A2[0][2]}, {A2
        [1][0], A2[1][1], A2[1][2]}, {A2[2][0], A2[2][1], A2[2][2]}};
    vector <double> second_slice_A2 = {A2[0][3], A2[1][3], A2[2][3]};

    vector <double> roots2 = solve_linear_system(first_slice_A2, second_slice_A2);

    vector <double> y2(x.size(), 0);

    for (int i = 0; i < y2.size(); i++){
        y2[i] = roots2[0] + x[i] * roots2[1] + pow(x[i],2) * roots2[2];
    }

    return make_pair(y1, y2);
}

int main() {
    ofstream fout("answer3.txt");
    vector<double> x = {-0.7, -0.4, -0.1, 0.2, 0.5, 0.8};
    vector<double> y = {-1.4754, -0.81152, -0.20017, 0.40136, 1.0236, 1.7273};

    vector <double> y1 = method_least_squares(x,y).first;
    vector <double> y2 = method_least_squares(x,y).second;

    double F1 = 0.0, F2 = 0.0;
    for (int i = 0; i < y.size(); ++i) {
        F1 += pow((y[i] - y1[i]), 2);
        F2 += pow((y[i] - y2[i]), 2);
    }

    fout << "First-degree polynomial" << endl;
    for (int i = 0; i < y1.size(); i++){
        fout << y1[i] << " ";
    }

    fout << endl << endl;
```

```
    fout << "Second-degree polynomial" << endl;
    for (int i = 0; i < y2.size(); i++){
        fout << y2[i] << " ";
    }
    fout << endl << endl;
    fout << "Sum of squares of errors for the first-degree polynomial: " << F1 << endl;
    fout << "Sum of squares of errors for the second-degree polynomial: " << F2 << endl
        ;

    return 0;
}
```

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: 19

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

i	0	1	2	3	4
x_i	-1	0	1	2	3
f_i	-1.7854	0.0	1.7854	3.1071	4.249

11 Результаты работы

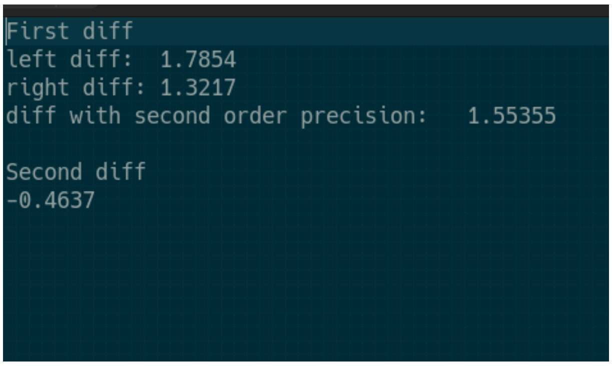


Рис. 4: Вывод в консоли

12 Исходный код

Lab3.4.cpp

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

using namespace std;

vector <double> first_diff(vector<double> x, vector <double> y, double root_index){
    int n = x.size();
    double left_diff, right_diff, diff;
    if (root_index != 1 && root_index != (n - 2)){
        left_diff = (y[root_index] - y[root_index - 1]) / (x[root_index] - x[root_index - 1]);
        right_diff = (y[root_index + 1] - y[root_index]) / (x[root_index + 1] - x[root_index]);

        diff = (y[root_index] - y[root_index - 1]) / (x[root_index] - x[root_index - 1]) \
        + ((y[root_index + 1] - y[root_index])/(x[root_index+1] - x[root_index]) - (y[root_index] - y[root_index - 1])/(x[root_index] - x[root_index-1]) ) \
        / (x[root_index + 1] - x[root_index - 1]) \
        * (2 * x[root_index] - x[root_index] - x[root_index - 1]);
        return vector <double> {left_diff, right_diff, diff};
    }
    cout << "Error!" << endl;
    return vector <double> (0,0);
```

```
}

double second_diff(vector <double> x, vector <double> y, double root_index){
    int n = x.size();
    if (root_index != 1 && root_index != (n - 2)){
        double dx2 = 2 * ((y[root_index + 1] - y[root_index])/(x[root_index+1] - x[root_index]) - (y[root_index] - y[root_index - 1])/(x[root_index] - x[root_index-1]) ) \
            / (x[root_index + 1] - x[root_index - 1]) ;
        return dx2;
    }
    cout << "Error!" << endl;
    return 0.0;

}


int main(){
    ofstream fout("answer4.txt");
    // fout.precision(2);
    // fout << fixed;
    int n = 5;
    vector <double> x = {-1, 0, 1, 2, 3};
    vector <double> y = {-1.7854, 0.0, 1.7854, 3.1071, 4.249};
    double X = 2; //  1-    x

    vector <double> first_diffs = first_diff(x,y,X);
    double dx2 = second_diff(x,y,X);
    fout << "First diff" << endl;
    fout << "left diff:\t" << first_diffs[0] << endl;
    fout << "right diff:\t" << first_diffs[1] << endl;
    fout << "diff with second order precision:\t" << first_diffs[2] << endl << endl;
    fout << "Second diff" << endl;
    fout << dx2 << endl;
}
```

13 Постановка задачи

3.5. Вычислить определенный интеграл

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

, , ,

h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга:
Вариант: 19

$$y = x^2, 625 - x^4; X_0 = 0, X_k = 4, h_1 = 1.0, h_2 = 0.5 \tag{7}$$

14 Результаты работы

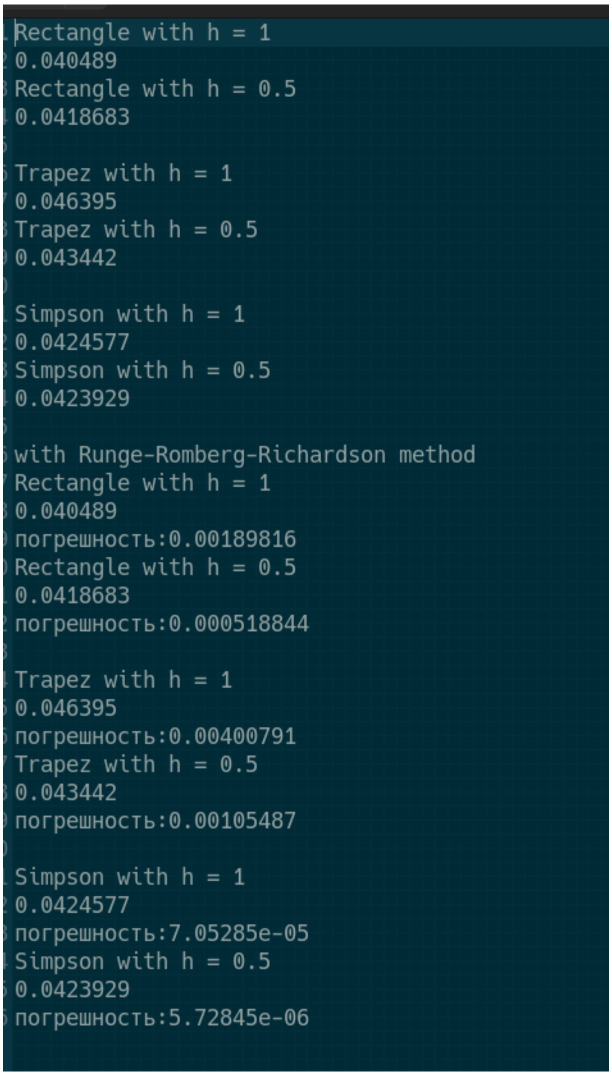


Рис. 5: Вывод в консоли

15 Исходный код

Lab3.5.cpp

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

```cpp
#include <cmath>

using namespace std;

double func(double x){
    return (x * x) / (625 - x * x * x * x);
}


double rectangle_method(double x0, double xk, double h){
    double F = 0;
    double n = (int) ((xk - x0) / h);
    n += 1;
    vector <double> x_values(n, 0);
    for (int i = 0; i < n; i++){
        x_values[i] = x0 + h*i;
    }
    for (int i = 1; i < n; i++){
        F += h * func((x_values[i] + x_values[i-1])/2);
    }
    return F;
}

double trapez_method(double x0, double xk, double h){
    double F = 0;
    double n = (int) ((xk - x0) / h);
    n += 1;
    vector <double> x_values(n, 0);
    for (int i = 0; i < n; i++){
        x_values[i] = x0 + h*i;
    }
    for (int i = 1; i < n; i++){
        F += (func(x_values[i]) + func(x_values[i-1])) / 2 * h;
    }
    return F;
}

double simps_method(double x0, double xk, double h){
    int n = (int)((xk - x0) / h);
    double F = 0;
    for (int i = 0; i < n; i++) {
        double x1 = x0 + i * h;
        double x2 = x0 + (i + 1) * h;
        double x3 = x0 + (i + 0.5) * h;
        F += (h / 6) * (func(x1) + 4 * func(x3) + func(x2));
    }
    return F;
}

```

```
vector <double> runge_romb_rich(double x0, double xk, double h){
    double F = 0;
    vector <double> results(3,0);
    results[0] = rectangle_method(x0, xk, h) + (rectangle_method(x0, xk, h/2) -
        rectangle_method(x0, xk, h/2))/(1-0.5*0.5);
    results[1] = trapez_method(x0, xk, h) + (trapez_method(x0, xk, h/2) - trapez_method
        (x0, xk, h/2))/(1-0.5*0.5);
    results[2] = simps_method(x0, xk, h) + (simps_method(x0, xk, h/2) - simps_method(x0
        , xk, h/2))/(1-0.5*0.5*0.5*0.5);
    return results;
}


int main(){
    ofstream fout("answer5.txt");
    // fout.precision(2);
    // fout << fixed;
    int n = 5;
    double x0 = 0, xk = 4, h1 = 1.0, h2 = 0.5;
    double integral = 0.042387134;
    fout << "Rectangle with h = 1\n";
    fout << rectangle_method(x0, xk, h1);
    fout << "\nRectangle with h = 0.5\n";
    fout << rectangle_method(x0, xk, h2);
    fout << "\n\nTrapez with h = 1\n";
    fout << trapez_method(x0,xk,h1);
    fout << "\nTrapez with h = 0.5\n";
    fout << trapez_method(x0,xk,h2);
    fout << "\n\nSimpson with h = 1\n";
    fout << simps_method(x0,xk,h1);
    fout << "\nSimpson with h = 0.5\n";
    fout << simps_method(x0,xk,h2) << endl << endl;
    vector <double> RRR = runge_romb_rich(x0, xk, h1);
    vector <double> RRR2 = runge_romb_rich(x0,xk,h2);
    fout << "with Runge-Romberg-Richardson method" << endl;
    fout << "Rectangle with h = 1\n";
    fout << RRR[0];
    fout << "\n:" << fabs(integral - RRR[0]);
    fout << "\nRectangle with h = 0.5\n";
    fout << RRR2[0];
    fout << "\n:" << fabs(integral - RRR2[0]);
    fout << "\n\nTrapez with h = 1\n";
    fout << RRR[1];
    fout << "\n:" << fabs(integral - RRR[1]);
    fout << "\nTrapez with h = 0.5\n";
    fout << RRR2[1];
    fout << "\n:" << fabs(integral - RRR2[1]);
    fout << "\n\nSimpson with h = 1\n";
    fout << RRR[2];
```

```
    fout << "\n:" << fabs(integral - RRR[2]);
    fout << "\nSimpson with h = 0.5\n";
    fout << RRR2[2];
    fout << "\n:" << fabs(integral - RRR2[2]);
}
```