

**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 Ньютона

## 1 Постановка задачи

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

**Вариант: 15**

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

## 2 Результаты работы

```
answer.txt X
Lab3 > lab3_1 > answer.txt
1 A:
2 Lagrange polynom:
3 5.31371x^0 + -8.83218x^1 + 6.9454x^2 + -1.80775x^3
4 f(x*) = 2.15155
5 error = 0.0658942
6
7 Newton polynom
8 5.31371x^0 + -8.83218x^1 + 6.9454x^2 + -1.80775x^3
9 f(x*) = 2.15155
10 error = 0.0658942
11
12 B:
13 Lagrange polynom:
14 5.31371x^0 + -8.83218x^1 + 6.9454x^2 + -1.80775x^3
15 f(x*) = 2.15155
16 error = 0.0658942
17
18 Newton polynom:
19 4.99983x^0 + -7.56663x^1 + 5.58847x^2 + -1.37583x^3
20 f(x*) = 2.20059
21 error = 0.114938
22
```

Рис. 1: Вывод программы

### 3 Исходный код

```
1  #include <cmath>
2  #include <iostream>
3  #include <vector>
4  #include <fstream>
5
6  using namespace std;
7
8
9  class Polynom {
10 private:
11     vector<double> _poly;
12 public:
13     Polynom(int n) {
14         _poly.resize(n);
15     }
16     Polynom() {}
17     Polynom(const vector<double> &v){
18         _poly.resize(v.size());
19         for (int i = 0; i < v.size(); ++i){
20             _poly[i] = v[i];
21         }
22     }
23
24     int size() {
25         return _poly.size();
26     }
27
28     double & operator[](int i) {
29         return _poly[i];
30     }
31
32     void push_back(double k){
33         _poly.push_back(k);
34     }
35
36     friend Polynom operator+(Polynom &lhs, Polynom &rhs) {
37         int n_max = max(lhs.size(), rhs.size());
38         int n_min = min(lhs.size(), rhs.size());
39         Polynom res(n_max);
40         for (int i = 0; i < n_min; ++i) {
41             res[i] = lhs[i] + rhs[i];
42         }
43         for (int i = n_min; i < n_max; ++i) {
44             if (lhs.size() > rhs.size()) {
45                 res[i] = lhs[i];
46             } else {
47                 res[i] = rhs[i];
48             }
49         }
50         return res;
51     }
52
53     friend Polynom operator*(Polynom &lhs, Polynom &rhs) {
54         Polynom res(lhs.size() + rhs.size() - 1);
55         for (int i = 0; i < rhs.size(); ++i) {
56             for (int j = 0; j < lhs.size(); ++j) {
57                 res[i + j] += lhs[j] * rhs[i];
58             }
59         }
60         return res;
```

```

61     }
62
63     friend Polynom operator*(Polynom &lhs, double rhs) {
64         Polynom res(lhs.size());
65         for (int i = 0; i < lhs.size(); ++i) {
66             res[i] = lhs[i] * rhs;
67         }
68         return res;
69     }
70 };
71
72 ostream& operator<<(ostream& stream, Polynom poly)
73 {
74     for (int i = 0; i < poly.size(); ++i) {
75         if (i != poly.size() - 1)
76             stream << poly[i] << "x^" << i << " + ";
77         else
78             stream << poly[i] << "x^" << i << "\n";
79     }
80     return stream;
81 }
82
83 Polynom lagrange_polynom(vector<double> &x, vector<double> &y) {
84     int n = x.size();
85     Polynom L(n);
86     for (int i = 0; i < n; ++i) {
87         Polynom li;
88         li.push_back(1);
89         double c = 1;
90         for (int j = 0; j < n; ++j) {
91             if (i != j) {
92                 Polynom d({(-1) * x[j], 1});
93                 li = li*d;
94                 c *= (x[i] - x[j]);
95             }
96         }
97         L = L + li* (y[i]/c);
98     }
99     return L;
100 }
101
102 Polynom newton_polynom(vector<double> &x, vector<double> &y) {
103     int n = x.size();
104     vector<vector<double>> table(n, vector<double>(n));
105     for (int i = 0; i < n; ++i) {
106         table[i][0] = y[i];
107     }
108     for (int j = 1; j < n; ++j) {
109         for (int i = 0; i < n - j; ++i) {
110             table[i][j] = (table[i][j - 1] - table[i + 1][j - 1]) / (x[i] - x[i + j]);
111         }
112     }
113     Polynom P(n);
114     Polynom k;
115     for (int i = 0; i < n; ++i) {
116         if (i == 0) {
117             k.push_back(1);
118         } else {
119             Polynom d({(-1) * x[i - 1], 1});
120             k = k * d;
121         }
122         P = (P + k * table[0][i]);

```

```

123     }
124     return P;
125 }
126
127 double f(double x) {
128     return 1/tan(x) + x;
129 }
130
131 int main() {
132     double pi = 2*acos(0.0);
133     double X = 3*pi/16;
134     vector<double> x_a = {pi/8, 2*pi/8, 3*pi/8, 4*pi/8};
135     vector<double> y_a;
136     for (int i = 0; i < x_a.size(); ++i) {
137         y_a.push_back(f(x_a[i]));
138     }
139
140     ofstream fout("answer.txt");
141     fout << "A:\n";
142     fout << "Lagrange polynom:\n";
143     Polynomial polynom_l_a = lagrange_polynom(x_a, y_a);
144     fout << polynom_l_a;
145
146     double res_a = 0;
147     for (int i = 0; i < polynom_l_a.size(); ++i) {
148         res_a += polynom_l_a[i] * pow(X, i);
149     }
150     fout << "f(x*) = " << res_a << "\n";
151     fout << "error = " << abs(f(X) - res_a) << "\n";
152
153
154     fout << "\nNewton polynom\n";
155     Polynomial polynom_n_a = newton_polynom(x_a, y_a);
156     fout << polynom_n_a;
157     res_a = 0;
158     for (int i = 0; i < polynom_n_a.size(); ++i) {
159         res_a += polynom_n_a[i] * pow(X, i);
160     }
161     fout << "f(x*) = " << res_a << "\n";
162     fout << "error = " << abs(f(X) - res_a) << "\n\n";
163
164
165
166
167     fout << "B:\n";
168
169     vector<double> x_b = {pi/8, pi/3, 3*pi/8, pi/2};
170     vector<double> y_b;
171     for (int i = 0; i < x_b.size(); ++i) {
172         y_b.push_back(f(x_b[i]));
173     }
174
175     fout << "Lagrange polynom:\n";
176     Polynomial polynom_l_b = lagrange_polynom(x_b, y_b);
177     fout << polynom_l_b;
178
179     double res_b = 0;
180     for (int i = 0; i < polynom_l_b.size(); ++i) {
181         res_b += polynom_l_b[i] * pow(X, i);
182     }
183     fout << "f(x*) = " << res_b << "\n";
184     fout << "error = " << abs(f(X) - res_b) << "\n";

```

```

185
186
187     fout << "\nNewton polynom:\n";
188     Polynom polynom_n_b = newton_polynom(x_b, y_b);
189     fout << polynom_n_b;
190     res_b = 0;
191     for (int i = 0; i < polynom_n_b.size(); ++i) {
192         res_b += polynom_n_b[i] * pow(X, i);
193     }
194     fout << "f(x*) = " << res_b << "\n";
195     fout << "error = " << abs(f(X) - res_b) << "\n";
196     return 0;
197 }

```

## 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
1 Cubic spline:
2  $S(x) = 1.8279 + -0.662706 * (x - 1.9) + 0.708452 * (x - 1.9)^2 + -0.26915 * (x - 1.9)^3$ 
3  $f(x^*) = 1.61495$ 
```

Рис. 3: Вывод программы

### 3 Исходный код

```
1  #include <cmath>
2  #include <iostream>
3  #include <vector>
4  #include <fstream>
5
6  using namespace std;
7
8  class matrix
9  {
10     private:
11         vector <vector <double>> _obj;
12     public:
13         int cols = 0, rows = 0;
14
15         matrix() {}
16         matrix(int _rows, int _cols)
17         {
18             rows = _rows;
19             cols = _cols;
20             _obj = vector <vector <double>>(rows, vector <double>(cols));
21         }
22
23         vector <double> &operator[](int i)
24         {
25             return _obj[i];
26         }
27
28         operator double()
29         {
30             return _obj[0][0];
31         }
32     };
33
34
35     istream& operator>>(istream& stream, matrix& m)
36     {
37         for (int i = 0; i < m.rows; i++)
38         {
39             for (int j = 0; j < m.cols; j++)
40                 stream >> m[i][j];
41         }
42         return stream;
43     }
44
45     ostream& operator<<(ostream& stream, matrix m)
46     {
47         for (int i = 0; i < m.rows; i++)
48         {
49             for (int j = 0; j < m.cols; j++)
50                 stream << m[i][j] << ' ';
51             stream << '\n';
52         }
53         return stream;
54     }
55
56
57     matrix solve_SLE (matrix& A, matrix& b)
58     {
59         int n = A.cols;
60         vector <double> p(n), q(n);
```



```

61     matrix ans(n, 1);
62     p[0] = -A[0][1] / A[0][0];
63     q[0] = b[0][0] / A[0][0];
64     for (int i = 1; i < n; i++)
65     {
66         if (i != n - 1)
67             p[i] = -A[i][i + 1] / (A[i][i] + A[i][i - 1] * p[i - 1]);
68         else
69             p[i] = 0;
70         q[i] = (b[i][0] - A[i][i - 1] * q[i - 1]) / (A[i][i] + A[i][i - 1] * p[i - 1]);
71     }
72     ans[n - 1][0] = q[n - 1];
73     for (int i = n - 2; i >= 0; i--)
74         ans[i][0] = p[i] * ans[i + 1][0] + q[i];
75     return ans;
76 }
77
78 double spline(vector<double> &x, vector<double> &y, double dot) {
79     int n = x.size() - 1;
80     vector<double> a(n), b(n), c(n), d(n), h(n);
81     for (int i = 0; i < n; ++i) {
82         h[i] = x[i + 1] - x[i];
83     }
84     matrix A(n - 1, n - 1), B(n - 1, 1);
85     for (int i = 0; i < n - 1; ++i) {
86         A[i][i] = 2 * (h[i] + h[i + 1]);
87         if (i > 0)
88             A[i][i - 1] = h[i];
89         if (i < n - 2)
90             A[i][i + 1] = h[i + 1];
91         B[i][0] = 3 * ((y[i + 2] - y[i + 1]) / h[i + 1] - (y[i + 1] - y[i]) / h[i]);
92     }
93     matrix C = solve_SLE(A, B);
94     for (int i = 0; i < n; ++i) {
95         if (i == 0)
96             c[i] = 0;
97         else
98             c[i] = C[i - 1][0];
99     }
100    for (int i = 0; i < n; ++i) {
101        a[i] = y[i];
102        if (i < n - 1) {
103            b[i] = (y[i + 1] - y[i]) / h[i] - 1.0 / 3 * h[i] * (c[i + 1] + 2 * c[i]);
104            d[i] = (c[i + 1] - c[i]) / (3 * h[i]);
105        } else {
106            b[i] = (y[i + 1] - y[i]) / h[i] - 2.0 / 3 * h[i] * c[i];
107            d[i] = (-1) * c[i] / (3 * h[i]);
108        }
109    }
110    auto it = lower_bound(x.begin(), x.end(), dot);
111    int interval = it - x.begin() - 1;
112    if (interval == -1)
113        interval = 0;
114    ofstream fout("answer.txt");
115    fout<< "Cubic spline:\n";
116    fout << "S(x) = " << a[interval] << " + " << b[interval] << " * (x - " << x[interval] << ") + " << c[
        interval] << " * (x - " << x[interval] << ")^2 + " << d[interval] << " * (x - " << x[interval] <<
        ")^3\n";
117    double res = a[interval] + b[interval] * (dot - x[interval]) + c[interval] * pow((dot - x[interval]),
        2) + d[interval] * pow((dot - x[interval]), 3);
118    fout << "f(x*) = " << res;
119    return res;

```

```
120 | }  
121 |  
122 | int main() {  
123 |     vector<double> x = {1.0, 1.9, 2.8, 3.7, 4.6};  
124 |     vector<double> y = {2.8069, 1.8279, 1.6091, 1.5713, 1.5663};  
125 |     double X = 2.66666667;  
126 |     double res = spline(x, y, X);  
127 |     return 0;  
128 | }
```

### 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 X
Lab3 > lab3_3 > answer.txt
1 Least squares polynom [1st power]:
2 2.60728x^0 + 0.382267x^1
3 error = 0.637538
4
5 Least squares polynom [2nd power]:
6 3.20861x^0 + -0.0811603x^1 + 0.0715993x^2
7 error = 0.490258
8
9
```

Рис. 5: Вывод программы

### 3 Исходный код

ОСНОВНОЙ КОД:

```
1  #include <cmath>
2  #include <iostream>
3  #include <vector>
4  #include <fstream>
5  using namespace std;
6
7  class Matrix {
8  private:
9      int rows_, cols_;
10     vector<vector<double>>> matrix_;
11     vector<int> swp_;
12
13     void SwapMatrix(Matrix &other) {
14         swap(rows_, other.rows_);
15         swap(cols_, other.cols_);
16         swap(matrix_, other.matrix_);
17     }
18
19 public:
20     Matrix(int rows, int cols) {
21         rows_ = rows;
22         cols_ = cols;
23         matrix_.resize(rows_);
24         for (int i = 0; i < rows_; ++i) {
25             matrix_[i].resize(cols_);
26         }
27     }
28     Matrix(const Matrix &other) : Matrix(other.rows_, other.cols_) {
29         for (int i = 0; i < rows_; ++i) {
30             for (int j = 0; j < cols_; ++j) {
31                 matrix_[i][j] = other.matrix_[i][j];
32             }
33         }
34     }
35     Matrix() : Matrix(1, 1) {}
36     Matrix(Matrix &&other) {
37         this->SwapMatrix(other);
38         other.rows_ = 0;
39         other.cols_ = 0;
40     }
41
42     int GetRows() const { return rows_; }
43     int GetCols() const { return cols_; }
44     const vector<int> &GetSwp() const { return swp_; }
45
46     void MulNumber(const double num) {
47         for (int i = 0; i < rows_; ++i) {
48             for (int j = 0; j < cols_; ++j) {
49                 matrix_[i][j] *= num;
50             }
51         }
52     }
53
54     Matrix MulMatrixReturn(const double num) {
55         Matrix res = *this;
56         res.MulNumber(num);
57         return res;
58     }
59 }
```

```

60 void MulMatrix(const Matrix &other) {
61     Matrix tmp(rows_, other.cols_);
62     for (int i = 0; i < rows_; ++i) {
63         for (int j = 0; j < other.cols_; ++j) {
64             for (int k = 0; k < cols_; ++k)
65                 tmp.matrix_[i][j] += matrix_[i][k] * other.matrix_[k][j];
66         }
67     }
68     this->SwapMatrix(tmp);
69 }
70
71 Matrix MulMatrixReturn(const Matrix &other) {
72     Matrix res = *this;
73     res.MulMatrix(other);
74     return res;
75 }
76
77 Matrix Transpose() const {
78     Matrix result(cols_, rows_);
79     for (int i = 0; i < result.rows_; ++i) {
80         for (int j = 0; j < result.cols_; ++j) {
81             result.matrix_[i][j] = matrix_[j][i];
82         }
83     }
84     return result;
85 }
86 pair<Matrix, Matrix> LU(Matrix & L, Matrix & U) {
87     swp_.clear();
88     int n = this->GetRows();
89     for (int k = 0; k < n; ++k) {
90         int index = k;
91         for (int i = k + 1; i < n; ++i) {
92             if (abs(U(i, k)) > abs(U(index, k))) {
93                 index = i;
94             }
95         }
96         swap(U(k), U(index));
97         swap(L(k), L(index));
98         swp_.push_back(index);
99         for (int i = k + 1; i < n; ++i) {
100             double m = U(i, k) / U(k, k);
101             L(i, k) = m;
102             for (int j = k; j < n; ++j) {
103                 U(i, j) -= m * U(k, j);
104             }
105         }
106     }
107     for (int i = 0; i < n; ++i) {
108         L(i, i) = 1;
109     }
110     return {L, U};
111 }
112 Matrix Solve(Matrix &C) {
113     Matrix U(*this);
114     Matrix L(this->GetCols(), this->GetCols());
115     LU(L, U);
116     Matrix B(C);
117     vector<int> swp = this->GetSwp();
118     for (int i = 0; i < swp.size(); ++i) {
119         swap(B(i), B(swp[i]));
120     }
121     int n = this->GetRows();

```

```

122
123     Matrix Z(n, 1);
124     for (int i = 0; i < n; ++i) {
125         Z(i, 0) = B(i, 0);
126         for (int j = 0; j < i; ++j) {
127             Z(i, 0) -= L(i, j) * Z(j, 0);
128         }
129     }
130
131     Matrix X(n, 1);
132     for (int i = n - 1; i >= 0; --i) {
133         X(i, 0) = Z(i, 0);
134         for (int j = i + 1; j < n; ++j) {
135             X(i, 0) -= U(i, j) * X(j, 0);
136         }
137         X(i, 0) = X(i, 0) / U(i, i);
138     }
139     return X;
140 }
141 double &operator()(int i, int j) {
142     return matrix_[i][j];
143 }
144 vector<double> &operator()(int row) { return matrix_[row]; }
145 };
146
147 istream& operator>>(istream& stream, Matrix& m)
148 {
149     for (int i = 0; i < m.GetRows(); i++)
150     {
151         for (int j = 0; j < m.GetCols(); j++)
152             stream >> m(i, j);
153     }
154     return stream;
155 }
156
157 ostream& operator<<(ostream& stream, Matrix m)
158 {
159     for (int i = 0; i < m.GetRows(); i++)
160     {
161         for (int j = 0; j < m.GetCols(); j++)
162             stream << m(i, j) << ' ';
163         stream << '\n';
164     }
165     return stream;
166 }
167
168 ostream& operator<<(ostream& stream, vector<double> v)
169 {
170     for (int i = 0; i < v.size(); ++i) {
171         stream << v[i] << " ";
172     }
173     return stream;
174 }
175
176 vector<double> least_squares_method(vector<double> &x, vector<double> &y, int m) {
177     int n = x.size();
178     m = m+1;
179     Matrix Y(n, 1), Phi(n, m);
180     for (int i = 0; i < n; ++i) {
181         Y(i, 0) = y[i];
182     }
183     for (int i = 0; i < n; ++i) {

```

```

184         for (int j = 0; j < m; ++j) {
185             Phi(i, j) = pow(x[i], j);
186         }
187     }
188     Matrix Phi_T = Phi.Transpose();
189     Matrix B = Phi_T.MulMatrixReturn(Y);
190     Matrix res = (Phi_T.MulMatrixReturn(Phi)).Solve(B);
191     vector<double> polynom;
192     for (int i = 0; i < res.GetRows(); ++i) {
193         for (int j = 0; j < res.GetCols(); ++j) {
194             polynom.push_back(res(i, j));
195         }
196     }
197     return polynom;
198 }
199
200 double get_error(vector<double> &x, vector<double> &y, vector<double> &p) {
201     double f_x = 0, error = 0;
202     for (int i = 0; i < x.size(); ++i) {
203         f_x = 0;
204         for (int j = 0; j < p.size(); ++j) {
205             f_x += p[j] * pow(x[i], j);
206         }
207         error += (f_x - y[i]) * (f_x - y[i]);
208     }
209     return error;
210 }
211
212
213 int main() {
214     vector<double> x = {1.0, 1.9, 2.8, 3.7, 3.6, 5.5};
215     vector<double> y = {3.4142, 2.9818, 3.3095, 3.8184, 4.3599, 4.8318};
216     ofstream fout("answer.txt");
217     ofstream fout_py_args("py_args.txt");
218     fout_py_args << x << "\n";
219     fout_py_args << y << "\n";
220
221     vector<double> polynom = least_squares_method(x, y, 1);
222     fout << "Least squares polynom [1st power]:\n";
223     for (int i = 0; i < polynom.size(); ++i) {
224         if (i != polynom.size() - 1)
225             fout << polynom[i] << "x^" << i << " + ";
226         else
227             fout << polynom[i] << "x^" << i << "\n";
228     }
229     double error = get_error(x, y, polynom);
230     fout << "error = " << error << "\n\n";
231     fout_py_args << polynom << "\n";
232
233
234     polynom.clear();
235     polynom = least_squares_method(x, y, 2);
236     fout << "Least squares polynom [2nd power]:\n";
237     for (int i = 0; i < polynom.size(); ++i) {
238         if (i != polynom.size() - 1)
239             fout << polynom[i] << "x^" << i << " + ";
240         else
241             fout << polynom[i] << "x^" << i << "\n";
242     }
243     error = get_error(x, y, polynom);
244     fout << "error = " << error << "\n\n";
245     fout_py_args << polynom << "\n";

```

```

246 |     fout_py_args.close();
247 |     system("python show_plot.py");
248 |     return 0;
249 | }

```

Вывод графика:

```

1 | import matplotlib.pyplot as plt
2 | import numpy as np
3 |
4 | file = open ("./py_args.txt", "r")
5 |
6 | x = [float(x) for x in file.readline().split()]
7 | y = [float(x) for x in file.readline().split()]
8 | poly_1 = [float(x) for x in file.readline().split()]
9 | poly_2 = [float(x) for x in file.readline().split()]
10 |
11 | x_1 = np.linspace(x[0], x[-1], 100)
12 | y_1 = poly_1[0] + poly_1[1]*x_1
13 |
14 | x_2 = np.linspace(x[0], x[-1], 100)
15 | y_2 = poly_2[0] + poly_2[1]*x_1 + poly_2[2]*x_1*x_1
16 | plt.plot(x, y, 'ro')
17 | plt.plot(x_1, y_1)
18 | plt.plot(x_2, y_2)
19 |
20 | plt.xlabel('x')
21 | plt.ylabel('y')
22 |
23 | plt.title('My graph')
24 |
25 | plt.show()
26 | 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

```
≡ answer.txt X
Lab3 > lab3_4 > ≡ answer.txt
1 Derivatives of f(x):
2  $y'(X^*) = 2.50175$ 
3  $y''(X^*) = 1.4975$ 
4
```

Рис. 7: Вывод программы

### 3 Исходный код

```
1 #include <cmath>
2 #include <iostream>
3 #include <vector>
4 #include <fstream>
5
6 using namespace std;
7
8 double df(vector<double> &x, vector<double> &y, double X) {
9     auto lb = lower_bound(x.begin(), x.end(), X);
10    int def = lb - x.begin() - 1;
11    if (def == -1) def = 0;
12    double a = (y[def + 1] - y[def]) / (x[def + 1] - x[def]),
13    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]);
14    return (a + b * (2 * X - x[def] - x[def + 1]));
15 }
16
17 double ddf(vector<double> &x, vector<double> &y, double X) {
18     auto lb = lower_bound(x.begin(), x.end(), X);
19     int def = lb - x.begin() - 1;
20     if (def == -1) def = 0;
21     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]);
22 }
23
24 int main() {
25     vector<double> x = {0.0, 0.2, 0.4, 0.6, 0.8};
26     vector<double> y = {1.0, 1.4214, 1.8918, 2.4221, 3.0255};
27     double X = 0.4;
28     ofstream fout("answer.txt");
29     fout << "Derivatives of f(x):\n";
30     fout << "y'(X*) = " << df(x, y, X) << "\n";
31     fout << "y''(X*) = " << ddf(x, y, X) << "\n";
32     return 0;
33 }
```

## 5 Вычисление интегралла

### 1 Постановка задачи

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

**Вариант: 15**

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

### 2 Результаты работы

```
lab3_5 > ≡ answer.txt
1  Rectangle method:
2  F = 0.0233265; h = 0.5; error = 2.31465e-11
3  F = 0.0232577; h = 0.25; error = 6.88126e-05
4
5  Trapeze method:
6  F = 0.0230508; h = 0.5; error = 0.000137966
7  F = 0.0231887; h = 0.25; error = 1.34732e-07
8
9  Simpson method:
10 F = 0.0232346; h = 0.5 error = 6.87099e-08
11 F = 0.0232347; h = 0.25 error = 6.70995e-11
12
```

Рис. 8: Вывод программы

### 3 Исходный код

```
1 #include <cmath>
2 #include <iostream>
3 #include <vector>
4 #include <fstream>
5
6 using namespace std;
7
8 double f(double x) {
9     return (x / (pow(x,4)+81));
10 }
11
12 double RungeRombergMethod(double F1, double h1, double F2, double h2, double k) {
13     return (F1 + (F1 - F2) / (pow(h2 / h1, k) - 1));
14 }
15
16 double RectangleMethod(double a, double b, double step) {
17     double res = 0;
18     double x_prev = a;
19     for (double cur = a + step; cur <= b; cur += step) {
20         res += step * f((x_prev + cur) / 2);
21         x_prev = cur;
22     }
23     return res;
24 }
25
26 double SimpsonMethod(double a, double b, double step) {
27     double res = 0;
28     step = step/2;
29     for (double cur = a + 2 * step; cur <= b; cur += 2 * step) {
30         res += step * (f(cur - 2 * step) + 4 * f(cur - step) + f(cur));
31     }
32     return (1.0 / 3 * res);
33 }
34
35 double TrapezeMethod(double a, double b, double step) {
36     double res = 0;
37     double x_prev = a;
38     for (double cur = a + step; cur <= b; cur += step) {
39         res += step * (f(cur) + f(x_prev));
40         x_prev = cur;
41     }
42     return 0.5 * res;
43 }
44
45 int main() {
46     double X_0 = 0;
47     double X_k = 2;
48     double h1 = 0.5, h2 = 0.25;
49     ofstream fout("answer.txt");
50
51     fout << "Rectangle method:\n";
52     double F_h1 = RectangleMethod(X_0, X_k, h1);
53     double F_h2 = RectangleMethod(X_0, X_k, h2);
54     double F = RungeRombergMethod(F_h1, F_h2, h1, h2, 10);
55     fout << "F = " << F_h1 << "; h = " << h1 << "; error = " << abs(F - F_h1) << "\n";
56     fout << "F = " << F_h2 << "; h = " << h2 << "; error = " << abs(F - F_h2) << "\n";
57
58     fout << "\nTrapeze method:\n";
59     F_h1 = TrapezeMethod(X_0, X_k, h1);
60 }
```

```

61 | F_h2 = TrapezeMethod(X_0, X_k, h2);
62 | F = RungeRombergMethod(F_h1, h1, F_h2, h2, 10);
63 | fout << "F = " << F_h1 << "; h = " << h1 << "; error = " << abs(F - F_h1) << "\n";
64 | fout << "F = " << F_h2 << "; h = " << h2 << "; error = " << abs(F - F_h2) << "\n ";
65 |
66 | fout << "\nSimpson method:\n";
67 | F_h1 = SimpsonMethod(X_0, X_k, h1);
68 | F_h2 = SimpsonMethod(X_0, X_k, h2);
69 | F = RungeRombergMethod(F_h1, h1, F_h2, h2, 10);
70 | fout << "F = " << SimpsonMethod(X_0, X_k, h1) << "; h = " << h1 << " error = " << abs(F - F_h1) << "\n"
    | ;
71 | fout << "F = " << SimpsonMethod(X_0, X_k, h2) << "; h = " << h2 << " error = " << abs(F - F_h2) << "\n"
    | ";
72 |
73 | return 0;
74 | }

```