

**Московский авиационный институт
(национальный исследовательский университет)**

Институт № 8 «Компьютерные науки и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа № 3 по курсу «Численные методы»

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

Москва, 2022

Лабораторная работа 3

Методы приближения функций. Численное дифференцирование и интегрирование

3.1

Используя таблицу значений 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'

Исходный код

```
1  #include <iostream>
2  #include <cmath>
3  #include <vector>
4
5  using namespace std;
6
7  class polynom {
8  private:
9      vector<double> data;
10     constexpr static double EPS = 0.000000001;
11     size_t n;
12 public:
13     polynom() : data(1), n(1) {}
14     polynom(int _n) : data(_n), n(_n) {}
15     polynom(const vector<double> & coef) : data(coef), n(data.size()) {}
16     size_t size() const {
17         return n;
18     }
19     double & operator [] (size_t id) {
20         return data[id];
21     }
22     const double & operator [] (size_t id) const {
23         return data[id];
24     }
25     friend polynom operator + (const polynom & lhs, const polynom & rhs) {
26         polynom res(max(lhs.size(), rhs.size()));
27         for (size_t i = 0; i < lhs.size(); ++i) {
28             res[i] += lhs[i];
29         }
30         for (size_t i = 0; i < rhs.size(); ++i) {
31             res[i] += rhs[i];
32         }
33         return res;
34     }
35     friend polynom operator - (const polynom & lhs, const polynom & rhs) {
36         polynom res(max(lhs.size(), rhs.size()));
37         for (size_t i = 0; i < lhs.size(); ++i) {
38             res[i] += lhs[i];
39         }
40         for (size_t i = 0; i < rhs.size(); ++i) {
41             res[i] -= rhs[i];
42         }
43         return res;
```

```

44     }
45     friend polynom operator * (double lambda, const polynom & p) {
46         polynom res(p);
47         for (size_t i = 0; i < res.size(); ++i) {
48             res[i] *= lambda;
49         }
50         return res;
51     }
52     friend polynom operator / (const polynom & p, double lambda) {
53         polynom res(p);
54         for (size_t i = 0; i < res.size(); ++i) {
55             res[i] /= lambda;
56         }
57         return res;
58     }
59     friend polynom operator * (const polynom & lhs, const polynom & rhs) {
60         polynom res(lhs.size() + rhs.size());
61         for (size_t i = 0; i < lhs.size(); ++i) {
62             for (size_t j = 0; j < rhs.size(); ++j) {
63                 res[i + j] += lhs[i] * rhs[j];
64             }
65         }
66         while (res.n > 1 and abs(res.data.back()) < EPS) {
67             res.data.pop_back();
68             --res.n;
69         }
70         return res;
71     }
72     polynom integrate() {
73         polynom res(n + 1);
74         for (size_t i = 1; i < n + 1; ++i) {
75             res.data[i] = data[i - 1] / (double)i;
76         }
77         return res;
78     }
79     double integrate(double l, double r) {
80         polynom F = integrate();
81         return F(r) - F(l);
82     }
83     polynom derivative() {
84         polynom res(n - 1);
85         for (size_t i = 1; i < n; ++i) {
86             res[i - 1] = data[i] * i;
87         }
88         return res;
89     }
90     double operator () (double x) {
91         double res = 0.0;
92         double xi = 1.0;
93         for (double elem : data) {
94             res += elem * xi;
95             xi *= x;
96         }
97         return res;
98     }
99     friend ostream & operator << (ostream & out, const polynom & poly) {
100         bool flag = false;
101         int deg = 0;
102         for (double elem : poly.data) {
103             if (!(abs(elem) < EPS)) {
104                 if (flag and deg) {
105                     out << (elem > EPS ? " + " : " - ");

```

```

106         }
107         out << abs(elem);
108         flag = true;
109         if (deg) {
110             out << " * x";
111             if (deg > 1) {
112                 out << " ^ " << deg;
113             }
114         }
115     }
116     ++deg;
117 }
118 if (!flag) {
119     out << 0;
120 }
121 return out;
122 }
123 ~polynom() = default;
124 };
125 class interLagrange {
126     vector<double> x;
127     vector<double> y;
128     size_t n;
129 public:
130     interLagrange(const vector<double> & _x, const vector<double> & _y) : x(_x), y(_y), n(x.size()) {};
131     polynom operator () () {
132         polynom res(vector<double>({0}));
133         for (size_t i = 0; i < n; ++i) {
134             polynom li(vector<double>({1}));
135             for (size_t j = 0; j < n; ++j) {
136                 if (i == j) {
137                     continue;
138                 }
139                 polynom xij(vector<double>({-x[j], 1}));
140                 li = li * xij / (x[i] - x[j]);
141             }
142             res = res + y[i] * li;
143         }
144         return res;
145     }
146 };
147 class interNewton {
148 private:
149     vector<double> x;
150     vector<double> y;
151     size_t n;
152     vector<vector<bool>> calc;
153     vector<vector<double>> memo;
154     double f(int l, int r) {
155         if (calc[l][r]) {
156             return memo[l][r];
157         }
158         calc[l][r] = true;
159         double res;
160         if (l + 1 == r) {
161             res = (y[l] - y[r]) / (x[l] - x[r]);
162         } else {
163             res = (f(l, r - 1) - f(l + 1, r)) / (x[l] - x[r]);
164         }
165         return memo[l][r] = res;
166     }
167 public:

```

```

168     interNewton(const vector<double> & _x, const vector<double> & _y) : x(_x), y(_y), n(x.size()) {
169         calc.resize(n, vector<bool>(n));
170         memo.resize(n, vector<double>(n));
171     };
172     polynom operator () () {
173         polynom li(vector<double>({-x[0], 1}));
174         polynom res(vector<double>({y[0]}));
175         int r = 0;
176         for (size_t i = 1; i < n; ++i) {
177             res = res + f(0, ++r) * li;
178             li = li * polynom(vector<double>({-x[i], 1}));
179         }
180         return res;
181     }
182 };
183 int main() {
184     int n;
185     cin >> n;
186     vector<double> x(n), y(n);
187     for (int i = 0; i < n; ++i) {
188         cin >> x[i];
189         y[i] = asin(x[i]) + x[i];
190     }
191     double x_error;
192     cin >> x_error;
193     interLagrange myLagrange(x, y);
194     polynom lagrange = myLagrange();
195     cout << " : \n" << lagrange << endl;
196     cout << " X' = " << abs(lagrange(x_error) - asin(x_error) - x_error) << " \n \n";
197     interNewton myNewton(x, y);
198     polynom newton = myNewton();
199     cout << " : \n" << newton << endl;
200     cout << " X' = " << abs(newton(x_error) - asin(x_error) - x_error) << " \n \n";
201 }

```

Входные данные

test1:

```

4
-0.4 -0.1 0.2 0.5
0.1

```

test2:

```

4
-0.4 0 0.2 0.5
0.1

```

Консоль

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

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-1$ ./a.out <test1
```

Интерполяционный многочлен Лагранжа:

$7.55373e-05 + 1.99923 * x + 0.00197657 * x^2 + 0.188516 * x^3$

Погрешность в точке $X' = 0.000111543$

Интерполяционный многочлен Ньютона:

$7.55373e-05 + 1.99923 * x + 0.00197657 * x^2 + 0.188516 * x^3$

Погрешность в точке $X' = 0.000111543$

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-1$ ./a.out <test2
```

Интерполяционный многочлен Лагранжа:

$1.99889 * x + 0.00141004 * x^2 + 0.190404 * x^3$

Погрешность в точке $X' = 7.37745e-05$

Интерполяционный многочлен Ньютона:

$1.99889 * x + 0.00141004 * x^2 + 0.190404 * x^3$

Погрешность в точке $X' = 7.37745e-05$

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

3.2

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

Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4
5  using namespace std;
6
7  template<class T>
8  class tridiag_t {
9  private:
10     const double EPS = 0.000001;
11     int n;
12     vector<T> a;
13     vector<T> b;
14     vector<T> c;
15 public:
16     tridiag_t(const int &_n) : n(_n), a(n), b(n), c(n) {}
17     tridiag_t(const vector<T> &_a, const vector<T> &_b, const vector<T> &_c) {
18         if (!(_a.size() == _b.size() and _a.size() == _c.size())) {
19             throw invalid_argument("Sizes of a, b, c are invalid");
20         }
21         n = _a.size();
22         a = _a;
23         b = _b;
24         c = _c;
25     }
26     vector<T> solve(const vector<T> &d) {
27         int m = d.size();
28         if (n != m) {
29             throw invalid_argument("Size of vector d is invalid");
30         }
31         vector<T> p(n);
32         p[0] = -c[0] / b[0];
33         vector<T> q(n);
34         q[0] = d[0] / b[0];
35         for (int i = 1; i < n; ++i) {
36             p[i] = -c[i] / (b[i] + a[i] * p[i - 1]);
37             q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]);
38         }
39         vector<T> x(n);
40         x.back() = q.back();
41         for (int i = n - 2; i >= 0; --i) {
42             x[i] = p[i] * x[i + 1] + q[i];
43         }
44         return x;
45     }
46     friend istream & operator >> (istream &in, tridiag_t<T> &tridiag) {
47         in >> tridiag.b[0] >> tridiag.c[0];
48         for (int i = 1; i < tridiag.n - 1; ++i) {
49             in >> tridiag.a[i] >> tridiag.b[i] >> tridiag.c[i];
50         }
51         in >> tridiag.a.back() >> tridiag.b.back();
52         return in;
53     }
```

```

53     }
54     ~tridiag_t() = default;
55 };
56 class cub_spline_t {
57     size_t n;
58     vector<double> x;
59     vector<double> y;
60     vector<double> a, b, c, d;
61     void buildSpline() {
62         vector<double> h(n + 1);
63         h[0] = NAN;
64         for (size_t i = 1; i <= n; ++i) {
65             h[i] = x[i] - x[i - 1];
66         }
67         vector<double> func1(n - 1);
68         vector<double> func2(n - 1);
69         vector<double> func3(n - 1);
70         vector<double> func4(n - 1);
71         for (size_t i = 2; i <= n; ++i) {
72             func1[i - 2] = h[i - 1];
73             func2[i - 2] = 2.0 * (h[i - 1] + h[i]);
74             func3[i - 2] = h[i];
75             func4[i - 2] = 3.0 * ((y[i] - y[i - 1]) / h[i] - (y[i - 1] - y[i - 2]) / h[i - 1]);
76         }
77         func1[0] = 0.0;
78         func3.back() = 0.0;
79         tridiag_t<double> systemOfFunc(func1, func2, func3);
80         vector<double> cSolved = systemOfFunc.solve(func4);
81         for (size_t i = 2; i <= n; ++i) {
82             c[i] = cSolved[i - 2];
83         }
84         for (size_t i = 1; i <= n; ++i) {
85             a[i] = y[i - 1];
86         }
87         for (size_t i = 1; i < n; ++i) {
88             b[i] = (y[i] - y[i - 1]) / h[i] - h[i] * (c[i + 1] + 2.0 * c[i]) / 3.0;
89             d[i] = (c[i + 1] - c[i]) / (3.0 * h[i]);
90         }
91         c[1] = 0.0;
92         b[n] = (y[n] - y[n - 1]) / h[n] - (2.0 / 3.0) * h[n] * c[n];
93         d[n] = -c[n] / (3.0 * h[n]);
94     }
95 public:
96     cub_spline_t(const vector<double> & _x, const vector<double> & _y) {
97         if (_x.size() != _y.size()) {
98             throw invalid_argument("Sizes does not match");
99         }
100         x = _x;
101         y = _y;
102         n = x.size() - 1;
103         a.resize(n + 1);
104         b.resize(n + 1);
105         c.resize(n + 1);
106         d.resize(n + 1);
107         buildSpline();
108     }
109
110     double operator () (double x0) {
111         for (size_t i = 1; i <= n; ++i) {
112             if (x[i - 1] <= x0 and x0 <= x[i]) {
113                 double x1 = x0 - x[i - 1];
114                 double x2 = x1 * x1;

```



```

115         double x3 = x2 * x1;
116         return a[i] + b[i] * x1 + c[i] * x2 + d[i] * x3;
117     }
118 }
119 return NAN;
120 }
121 friend ostream & operator << (ostream & out, const cub_spline_t & spline) {
122     for (size_t i = 1; i <= spline.n; ++i) {
123         out << "i = " << i << ", a = " << spline.a[i] << ", b = " << spline.b[i] << ", c = " << spline.c
            [i] << ", d = " << spline.d[i] << '\n';
124     }
125     return out;
126 }
127 };
128 int main() {
129     int n;
130     cin >> n;
131     vector<double> x(n), y(n);
132     for (int i = 0; i < n; ++i) {
133         cin >> x[i];
134     }
135     for (int i = 0; i < n; ++i) {
136         cin >> y[i];
137     }
138     double x0;
139     cin >> x0;
140     cout.precision(6);
141     cub_spline_t func(x, y);
142     cout << "C:\n" << func << endl;
143     cout << "f(X') = " << func(x0) << endl;
144 }

```

Входные данные

```

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

```

Консоль

```

natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-2$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab3/lab3-2$ ./a.out <test
Сплайн:

```

```

i = 1,a = -0.81152,b = 2.04668,c = 0,d = -0.0983333
i = 2,a = -0.20017,b = 2.02013,c = -0.0885,d = 0.127963
i = 3,a = 0.40136,b = 2.00158,c = 0.0266667,d = 0.717222
i = 4,a = 1.0236,b = 2.21123,c = 0.672167,d = -0.746852

```

$$f(X') = 0.20134$$

3.3

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

Исходный код

```
1  #include <iostream>
2  #include <cmath>
3  #include <functional>
4  #include <vector>
5
6  template<class T>
7  std::vector<T> operator + (const std::vector<T> & a, const std::vector<T> & b) {
8      size_t n = a.size();
9      std::vector<T> c(n);
10     for (size_t i = 0; i < n; ++i) {
11         c[i] = a[i] + b[i];
12     }
13     return c;
14 }
15
16 template<class T>
17 std::vector<T> operator - (const std::vector<T> & a, const std::vector<T> & b) {
18     size_t n = a.size();
19     std::vector<T> c(n);
20     for (size_t i = 0; i < n; ++i) {
21         c[i] = a[i] - b[i];
22     }
23     return c;
24 }
25
26 template<class T>
27 class matrix_t {
28 private:
29     size_t n, m;
30     std::vector<std::vector<T>> data;
31 public:
32     matrix_t() : n(1), m(1), data(1) {}
33     matrix_t(size_t _n) : n(_n), m(_n) {
34         data.resize(n, std::vector<T>(n));
35     }
36     matrix_t(size_t _n, size_t _m) : n(_n), m(_m) {
37         data.resize(n, std::vector<T>(m));
38     }
39     matrix_t(const matrix_t<T> & other) {
40         n = other.n;
41         m = other.m;
42         data = other.data;
43     }
44     matrix_t<T> & operator = (const matrix_t<T> & other) {
45         if (this == &other) {
46             return *this;
47         }
48         n = other.n;
49         m = other.m;
50         data = other.data;
51         return *this;
52     }
53 }
```

```

51 static matrix_t<T> identity(size_t n) {
52     matrix_t<T> res(n, n);
53     for (size_t i = 0; i < n; ++i) {
54         res[i][i] = T(1);
55     }
56     return res;
57 }
58 matrix_t<T> t() const {
59     matrix_t<T> res(m, n);
60     for (size_t i = 0; i < n; ++i) {
61         for (size_t j = 0; j < m; ++j) {
62             res[j][i] = data[i][j];
63         }
64     }
65     return res;
66 }
67 size_t rows() const {
68     return n;
69 }
70 size_t cols() const {
71     return m;
72 }
73 void swap_rows(size_t i, size_t j) {
74     if (i == j) {
75         return;
76     }
77     for (size_t k = 0; k < m; ++k) {
78         std::swap(data[i][k], data[j][k]);
79     }
80 }
81 void swap_cols(size_t i, size_t j) {
82     if (i == j) {
83         return;
84     }
85     for (size_t k = 0; k < n; ++k) {
86         std::swap(data[k][i], data[k][j]);
87     }
88 }
89 friend matrix_t<T> operator + (const matrix_t<T> & a, const matrix_t<T> & b) {
90     if (a.rows() != b.rows() or a.cols() != b.cols()) {
91         throw std::invalid_argument("Sizes does not match");
92     }
93     size_t n = a.rows();
94     size_t m = a.cols();
95     matrix_t<T> res(n, m);
96     for (size_t i = 0; i < n; ++i) {
97         for (size_t j = 0; j < m; ++j) {
98             res[i][j] = a[i][j] + b[i][j];
99         }
100     }
101     return res;
102 }
103 friend matrix_t<T> operator - (const matrix_t<T> & a, const matrix_t<T> & b) {
104     if (a.rows() != b.rows() or a.cols() != b.cols()) {
105         throw std::invalid_argument("Sizes does not match");
106     }
107     size_t n = a.rows();
108     size_t m = a.cols();
109     matrix_t<T> res(n, m);
110     for (size_t i = 0; i < n; ++i) {
111         for (size_t j = 0; j < m; ++j) {
112             res[i][j] = a[i][j] - b[i][j];

```

```

113     }
114 }
115 return res;
116 }
117 friend matrix_t<T> operator * (const matrix_t<T> & a, const matrix_t<T> & b) {
118     if (a.cols() != b.rows()) {
119         throw std::invalid_argument("Sizes does not match");
120     }
121     size_t n = a.rows();
122     size_t k = a.cols();
123     size_t m = b.cols();
124     matrix_t<T> res(n, m);
125     for (size_t i = 0; i < n; ++i) {
126         for (size_t j = 0; j < m; ++j) {
127             for (size_t ii = 0; ii < k; ++ii) {
128                 res[i][j] += a[i][ii] * b[ii][j];
129             }
130         }
131     }
132     return res;
133 }
134 friend std::vector<T> operator * (const matrix_t<T> & a, const std::vector<T> & b) {
135     if (a.cols() != b.size()) {
136         throw std::invalid_argument("Sizes does not match");
137     }
138     size_t n = a.rows();
139     size_t m = a.cols();
140     std::vector<T> c(n);
141     for (size_t i = 0; i < n; ++i) {
142         for (size_t j = 0; j < m; ++j) {
143             c[i] += a[i][j] * b[j];
144         }
145     }
146     return c;
147 }
148 friend matrix_t<T> operator * (T lambda, const matrix_t<T> & a) {
149     size_t n = a.rows();
150     size_t m = a.cols();
151     matrix_t<T> res(n, m);
152     for (size_t i = 0; i < n; ++i) {
153         for (size_t j = 0; j < m; ++j) {
154             res[i][j] = lambda * a[i][j];
155         }
156     }
157     return res;
158 }
159 std::vector<T> & operator [] (size_t i) {
160     return data[i];
161 }
162 const std::vector<T> & operator [] (size_t i) const {
163     return data[i];
164 }
165 friend std::istream & operator >> (std::istream & in, matrix_t<T> & matr) {
166     for (size_t i = 0; i < matr.rows(); ++i) {
167         for (size_t j = 0; j < matr.cols(); ++j) {
168             in >> matr[i][j];
169         }
170     }
171     return in;
172 }
173 friend std::ostream & operator << (std::ostream & out, const matrix_t<T> & matr) {
174     for (size_t i = 0; i < matr.rows(); ++i) {

```

```

175         for (size_t j = 0; j < matr.cols(); ++j) {
176             if (j) {
177                 out << ", ";
178             }
179             out << matr[i][j];
180         }
181         out << '\n';
182     }
183     return out;
184 }
185 ~matrix_t() = default;
186 };
187 template<class T>
188 class lu_t {
189 private:
190     const T EPS = 0.000001;
191     matrix_t<T> l;
192     matrix_t<T> u;
193     T det;
194     std::vector<std::pair<size_t, size_t>> swaps;
195     void do_swaps(std::vector<T> & x) {
196         for (std::pair<size_t, size_t> elem : swaps) {
197             std::swap(x[elem.first], x[elem.second]);
198         }
199     }
200     void decompose() {
201         size_t n = u.rows();
202         for (size_t i = 0; i < n; ++i) {
203             size_t max_el_ind = i;
204             for (size_t j = i + 1; j < n; ++j) {
205                 if (abs(u[j][i]) > abs(u[max_el_ind][i])) {
206                     max_el_ind = j;
207                 }
208             }
209             if (max_el_ind != i) {
210                 std::pair<size_t, size_t> perm = std::make_pair(i, max_el_ind);
211                 swaps.push_back(perm);
212                 u.swap_rows(i, max_el_ind);
213                 l.swap_rows(i, max_el_ind);
214                 l.swap_cols(i, max_el_ind);
215             }
216             for (size_t j = i + 1; j < n; ++j) {
217                 if (abs(u[i][i]) < EPS) {
218                     continue;
219                 }
220                 T mu = u[j][i] / u[i][i];
221                 l[j][i] = mu;
222                 for (size_t k = 0; k < n; ++k) {
223                     u[j][k] -= mu * u[i][k];
224                 }
225             }
226         }
227         det = (swaps.size() & 1 ? -1 : 1);
228         for (size_t i = 0; i < n; ++i) {
229             det *= u[i][i];
230         }
231     }
232 public:
233     lu_t(const matrix_t<T> & matr) {
234         if (matr.rows() != matr.cols()) {
235             throw std::invalid_argument("Matrix is not square");
236         }

```

```

237     l = matrix_t<T>::identity(matr.rows());
238     u = matrix_t<T>(matr);
239     decompose();
240 }
241 std::vector<T> solve(std::vector<T> b) {
242     int n = b.size();
243     do_swaps(b);
244     std::vector<T> z(n);
245     for (int i = 0; i < n; ++i) {
246         T summary = b[i];
247         for (int j = 0; j < i; ++j) {
248             summary -= z[j] * l[i][j];
249         }
250         z[i] = summary;
251     }
252     std::vector<T> x(n);
253     for (int i = n - 1; i >= 0; --i) {
254         if (abs(u[i][i]) < EPS) {
255             continue;
256         }
257         T summary = z[i];
258         for (int j = n - 1; j > i; --j) {
259             summary -= x[j] * u[i][j];
260         }
261         x[i] = summary / u[i][i];
262     }
263     return x;
264 }
265 T get_det() {
266     return det;
267 }
268 matrix_t<T> inv_matrix() {
269     size_t n = l.rows();
270     matrix_t<T> res(n);
271     for (size_t i = 0; i < n; ++i) {
272         std::vector<T> b(n);
273         b[i] = T(1);
274         std::vector<T> x = solve(b);
275         for (size_t j = 0; j < n; ++j) {
276             res[j][i] = x[j];
277         }
278     }
279     return res;
280 }
281 friend std::ostream & operator << (std::ostream & out, const lu_t<T> & lu) {
282     out << "Matrix L:\n" << lu.l << "Matrix U:\n" << lu.u;
283     return out;
284 }
285 ~lu_t() = default;
286 };
287 class polynom {
288 private:
289     std::vector<double> data;
290     constexpr static double EPS = 0.000000001;
291     size_t n;
292 public:
293     polynom() : data(1), n(1) {}
294     polynom(int _n) : data(_n), n(_n) {}
295     polynom(const std::vector<double> & coef) : data(coef), n(data.size()) {}
296     size_t size() const {
297         return n;
298     }

```

```

299 double & operator [] (size_t id) {
300     return data[id];
301 }
302 const double & operator [] (size_t id) const {
303     return data[id];
304 }
305 friend polynom operator + (const polynom & lhs, const polynom & rhs) {
306     polynom res(std::max(lhs.size(), rhs.size()));
307     for (size_t i = 0; i < lhs.size(); ++i) {
308         res[i] += lhs[i];
309     }
310     for (size_t i = 0; i < rhs.size(); ++i) {
311         res[i] += rhs[i];
312     }
313     return res;
314 }
315 friend polynom operator - (const polynom & lhs, const polynom & rhs) {
316     polynom res(std::max(lhs.size(), rhs.size()));
317     for (size_t i = 0; i < lhs.size(); ++i) {
318         res[i] += lhs[i];
319     }
320     for (size_t i = 0; i < rhs.size(); ++i) {
321         res[i] -= rhs[i];
322     }
323     return res;
324 }
325 friend polynom operator * (double lambda, const polynom & p) {
326     polynom res(p);
327     for (size_t i = 0; i < res.size(); ++i) {
328         res[i] *= lambda;
329     }
330     return res;
331 }
332 friend polynom operator / (const polynom & p, double lambda) {
333     polynom res(p);
334     for (size_t i = 0; i < res.size(); ++i) {
335         res[i] /= lambda;
336     }
337     return res;
338 }
339 friend polynom operator * (const polynom & lhs, const polynom & rhs) {
340     polynom res(lhs.size() + rhs.size());
341     for (size_t i = 0; i < lhs.size(); ++i) {
342         for (size_t j = 0; j < rhs.size(); ++j) {
343             res[i + j] += lhs[i] * rhs[j];
344         }
345     }
346     while (res.n > 1 and abs(res.data.back()) < EPS) {
347         res.data.pop_back();
348         --res.n;
349     }
350     return res;
351 }
352 polynom integrate() {
353     polynom res(n + 1);
354     for (size_t i = 1; i < n + 1; ++i) {
355         res.data[i] = data[i - 1] / (double)i;
356     }
357     return res;
358 }
359 double integrate(double l, double r) {
360     polynom F = integrate();

```



```

361     return F(r) - F(1);
362 }
363 polynom derivative() {
364     polynom res(n - 1);
365     for (size_t i = 1; i < n; ++i) {
366         res[i - 1] = data[i] * i;
367     }
368     return res;
369 }
370 double operator () (double x) {
371     double res = 0.0;
372     double xi = 1.0;
373     for (double elem : data) {
374         res += elem * xi;
375         xi *= x;
376     }
377     return res;
378 }
379 friend std::ostream & operator << (std::ostream & out, const polynom & poly) {
380     bool flag = false;
381     int deg = 0;
382     for (double elem : poly.data) {
383         if (!(abs(elem) < EPS)) {
384             if (flag and deg) {
385                 out << (elem > EPS ? " + " : " - ");
386             }
387             out << abs(elem);
388             flag = true;
389             if (deg) {
390                 out << " * x";
391                 if (deg > 1) {
392                     out << " ^ " << deg;
393                 }
394             }
395             ++deg;
396         }
397     }
398     if (!flag) {
399         out << 0;
400     }
401     return out;
402 }
403 ~polynom() = default;
404 };
405 class minimal_square_t {
406     size_t n;
407     size_t m;
408     std::vector<double> x;
409     std::vector<double> y;
410     std::vector<double> a;
411     std::vector<std::function<double(double)>> phi;
412     double get(double x0) {
413         double res = 0.0;
414         for (size_t i = 0; i < m; ++i) {
415             res += a[i] * phi[i](x0);
416         }
417         return res;
418     }
419     void build() {
420         matrix_t<double> lhs(n, m);
421         for (size_t i = 0; i < n; ++i) {
422             for (size_t j = 0; j < m; ++j) {

```

```

423         lhs[i][j] = phi[j](x[i]);
424     }
425 }
426 matrix_t<double> lhs_t = lhs.t();
427 lu_t<double> lhs_lu(lhs_t * lhs);
428 std::vector<double> rhs = lhs_t * y;
429 a = lhs_lu.solve(rhs);
430 }
431 public:
432 minimal_square_t(const std::vector<double> & _x, const std::vector<double> & _y, const std::vector<std
    ::function<double(double)>> & _phi) {
433     if (_x.size() != _y.size()) {
434         throw std::invalid_argument("Sizes does not match");
435     }
436     x = _x;
437     y = _y;
438     n = _x.size();
439     m = _phi.size();
440     a.resize(m);
441     phi = _phi;
442     build();
443 }
444 double operator () (double x0) {
445     return get(x0);
446 }
447 double mmse() {
448     double res = 0;
449     for (size_t i = 0; i < n; ++i) {
450         res += pow(get(x[i]) - y[i], 2.0);
451     }
452     return res;
453 }
454 friend std::ostream & operator << (std::ostream & out, const minimal_square_t & item) {
455     for (size_t i = 0; i < item.m; ++i) {
456         if (i) {
457             out << ' ';
458         }
459         out << item.a[i];
460     }
461     return out;
462 }
463 };
464 double f0(double x0) {
465     return 1.0;
466 }
467 double f1(double x0) {
468     return x0;
469 }
470 double f2(double x0) {
471     return x0 * x0;
472 }
473 int main() {
474     int n;
475     std::cin >> n;
476     std::vector<double> x(n), y(n);
477     for (int i = 0; i < n; ++i) {
478         std::cin >> x[i];
479     }
480     for (int i = 0; i < n; ++i) {
481         std::cin >> y[i];
482     }
483     std::cout.precision(6);

```

```

484     std::cout << std::fixed;
485     std::vector<std::function<double(double)>> phi1 = {f0, f1};
486     minimal_square_t ms1(x, y, phi1);
487     std::cout << "    1-  a0, a1: " << std::endl;
488     std::cout << ms1 << std::endl;
489     std::cout << "    1-  = " << ms1.mmse() << "\n\n";
490     std::vector<std::function<double(double)>> phi2 = {f0, f1, f2};
491     minimal_square_t ms2(x, y, phi2);
492     std::cout << "    2-  a0, a1, a2: " << std::endl;
493     std::cout << ms2 << std::endl;
494     std::cout << "    2-  = " << ms2.mmse() << "\n\n";
495 }

```

Входные данные

```

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

```

Консоль

```

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

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

```

3.4

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

Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <exception>
5
6  using namespace std;
7
8  const double EPS = 0.000000001;
9  bool border(double a, double b) {
10     return (a < b) or (abs(b - a) < EPS);
11 }
12 class derivative_t {
13     int n;
14     vector<double> x;
15     vector<double> y;
16 public:
17     derivative_t(const vector<double> & x1, const vector<double> & y1) {
18         if (x1.size() != y1.size()) {
19             throw invalid_argument(" ");
20         }
21         x = x1;
22         y = y1;
23         n = x.size();
24     }
25     double derivative1(double x0) {
26         for (int i = 0; i < n - 2; ++i) {
27             if (x[i] < x0 && border(x0, x[i + 1])) {
28                 double dydx1 = (y[i + 1] - y[i]) / (x[i + 1] - x[i]);
29                 double dydx2 = (y[i + 2] - y[i + 1]) / (x[i + 2] - x[i + 1]);
30                 double res = dydx1 + (dydx2 - dydx1) * (2.0 * x0 - x[i] - x[i + 1]) / (x[i + 2] - x[i]);
31                 return res;
32             }
33         }
34         return NAN;
35     }
36     double derivative2(double x0) {
37         for (int i = 0; i < n - 2; ++i) {
38             if (x[i] < x0 && border(x0, x[i + 1])) {
39                 double dydx1 = (y[i + 1] - y[i]) / (x[i + 1] - x[i]);
40                 double dydx2 = (y[i + 2] - y[i + 1]) / (x[i + 2] - x[i + 1]);
41                 double res = 2.0 * (dydx2 - dydx1) / (x[i + 2] - x[i]);
42                 return res;
43             }
44         }
45         return NAN;
46     }
47 };
48 int main() {
49     int n;
50     cin >> n;
51     vector<double> x(n), y(n);
52     for (int i = 0; i < n; ++i) {
```

```

53     cin >> x[i];
54 }
55 for (int i = 0; i < n; ++i) {
56     cin >> y[i];
57 }
58 double x0;
59 cin >> x0;
60 cout.precision(6);
61 derivative_t f(x, y);
62 cout << "f'(" << x0 << ") = " << f.derivative1(x0) << endl;
63 cout << "f''(" << x0 << ") = " << f.derivative2(x0) << endl;
64 }

```

Входные данные

```

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

```

Консоль

```

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

```

3.5

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

Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4
5  using namespace std;
6
7  class polynom {
8  private:
9      vector<double> data;
10     constexpr static double EPS = 0.000000001;
11     size_t n;
12 public:
13     polynom() : data(1), n(1) {}
14     polynom(int _n) : data(_n), n(_n) {}
15     polynom(const vector<double> & coef) : data(coef), n(data.size()) {}
16     size_t size() const {
17         return n;
18     }
19     double & operator [] (size_t id) {
20         return data[id];
21     }
22     const double & operator [] (size_t id) const {
23         return data[id];
24     }
25     friend polynom operator + (const polynom & lhs, const polynom & rhs) {
26         polynom res(max(lhs.size(), rhs.size()));
27         for (size_t i = 0; i < lhs.size(); ++i) {
28             res[i] += lhs[i];
29         }
30         for (size_t i = 0; i < rhs.size(); ++i) {
31             res[i] += rhs[i];
32         }
33         return res;
34     }
35     friend polynom operator - (const polynom & lhs, const polynom & rhs) {
36         polynom res(max(lhs.size(), rhs.size()));
37         for (size_t i = 0; i < lhs.size(); ++i) {
38             res[i] += lhs[i];
39         }
40         for (size_t i = 0; i < rhs.size(); ++i) {
41             res[i] -= rhs[i];
42         }
43         return res;
44     }
45     friend polynom operator * (double lambda, const polynom & p) {
46         polynom res(p);
47         for (size_t i = 0; i < res.size(); ++i) {
48             res[i] *= lambda;
49         }
50         return res;
51     }
52     friend polynom operator / (const polynom & p, double lambda) {
```

```

53     polynom res(p);
54     for (size_t i = 0; i < res.size(); ++i) {
55         res[i] /= lambda;
56     }
57     return res;
58 }
59 friend polynom operator * (const polynom & lhs, const polynom & rhs) {
60     polynom res(lhs.size() + rhs.size());
61     for (size_t i = 0; i < lhs.size(); ++i) {
62         for (size_t j = 0; j < rhs.size(); ++j) {
63             res[i + j] += lhs[i] * rhs[j];
64         }
65     }
66     while (res.n > 1 and abs(res.data.back()) < EPS) {
67         res.data.pop_back();
68         --res.n;
69     }
70     return res;
71 }
72 polynom integrate() {
73     polynom res(n + 1);
74     for (size_t i = 1; i < n + 1; ++i) {
75         res.data[i] = data[i - 1] / (double)i;
76     }
77     return res;
78 }
79 double integrate(double l, double r) {
80     polynom F = integrate();
81     return F(r) - F(l);
82 }
83 polynom derivative() {
84     polynom res(n - 1);
85     for (size_t i = 1; i < n; ++i) {
86         res[i - 1] = data[i] * i;
87     }
88     return res;
89 }
90 double operator () (double x) {
91     double res = 0.0;
92     double xi = 1.0;
93     for (double elem : data) {
94         res += elem * xi;
95         xi *= x;
96     }
97     return res;
98 }
99 friend ostream & operator << (ostream & out, const polynom & poly) {
100     bool flag = false;
101     int deg = 0;
102     for (double elem : poly.data) {
103         if (!(abs(elem) < EPS)) {
104             if (flag and deg) {
105                 out << (elem > EPS ? " + " : " - ");
106             }
107             out << abs(elem);
108             flag = true;
109             if (deg) {
110                 out << " * x";
111                 if (deg > 1) {
112                     out << " ^ " << deg;
113                 }
114             }

```

```

115         }
116         ++deg;
117     }
118     if (!flag) {
119         out << 0;
120     }
121     return out;
122 }
123 ~polynom() = default;
124 };
125 class interLagrange {
126     vector<double> x;
127     vector<double> y;
128     size_t n;
129 public:
130     interLagrange(const vector<double> & _x, const vector<double> & _y) : x(_x), y(_y), n(x.size()) {};
131     polynom operator () () {
132         polynom res(vector<double>({0}));
133         for (size_t i = 0; i < n; ++i) {
134             polynom li(vector<double>({1}));
135             for (size_t j = 0; j < n; ++j) {
136                 if (i == j) {
137                     continue;
138                 }
139                 polynom xij(vector<double>({-x[j], 1}));
140                 li = li * xij / (x[i] - x[j]);
141             }
142             res = res + y[i] * li;
143         }
144         return res;
145     }
146 };
147 using func = double(double);
148 double integrRec(double l, double r, double h, func f) {
149     double res = 0;
150     double x0 = l;
151     double x1 = l + h;
152     while (x0 < r) {
153         res += h * f((x0 + x1) / 2);
154         x0 = x1;
155         x1 += h;
156     }
157     return res;
158 }
159 double integrTrap(double l, double r, double h, func f) {
160     double res = 0;
161     double x0 = l;
162     double x1 = l + h;
163     while (x0 < r) {
164         res += h * (f(x0) + f(x1));
165         x0 = x1;
166         x1 += h;
167     }
168     return res / 2;
169 }
170 double integrSimp(double l, double r, double h, func f) {
171     double res = 0;
172     double x0 = l;
173     double x1 = l + h;
174     while (x0 < r) {
175         vector<double> x = {x0, (x0 + x1) * 0.5, x1};
176         vector<double> y = {f(x[0]), f(x[1]), f(x[2])};

```



```

177         interLagrange lagr(x, y);
178         res += lagr().integrate(x0, x1);
179         x0 = x1;
180         x1 += h;
181     }
182     return res / 3;
183 }
184 double rungeRomberg(double fh, double fhk, double k, double d) {
185     return (fh - fhk) / (pow(k, d) - 1.0);
186 }
187 double f(double x) {
188     return (x * x) / (625.0 - pow(x, 4));
189 }
190 int main() {
191     double l, r;
192     cin >> l >> r;
193     double h1, h2;
194     cin >> h1 >> h2;
195     double rec1 = integrRec(l, r, h1, f);
196     double trap1 = integrTrap(l, r, h1, f);
197     double simp1 = integrSimp(l, r, h1, f);
198     cout.precision(5);
199     cout << "      " << h1 << " = " << rec1 << endl;
200     cout << "      " << h1 << " = " << trap1 << endl;
201     cout << "      " << h1 << " = " << simp1 << "\n\n";
202     double rec2 = integrRec(l, r, h2, f);
203     double trap2 = integrTrap(l, r, h2, f);
204     double simp2 = integrSimp(l, r, h2, f);
205     cout << "      " << h2 << " = " << rec2 << endl;
206     cout << "      " << h2 << " = " << trap2 << endl;
207     cout << "      " << h2 << " = " << simp2 << "\n\n";
208     double recError = abs(rungeRomberg(rec1, rec2, h2 / h1, 2));
209     double trapError = abs(rungeRomberg(trap1, trap2, h2 / h1, 2));
210     double simpError = abs(rungeRomberg(simp1, simp2, h2 / h1, 2));
211     cout << "      " << recError << endl;
212     cout << "      " << trapError << endl;
213     cout << "      " << simpError << endl;
214 }

```

Входные данные

```

test:
0 4
1.0 0.5

```

Консоль

```

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

```

Интеграл по методу Симпсона с шагом 1 = 0.014153

Интеграл по метод прямоугольников с шагом 0.5 = 0.041868

Интеграл по метод трапеций с шагом 0.5 = 0.043442

Интеграл по метод Симпсона с шагом 0.5 = 0.014131

Погрешность вычислений методом прямоугольников = 0.0018391

Погрешность вычислений методом трапеций = 0.0039374

Погрешность вычислений методом Симпсона = 2.88e-05