

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

**Институт №8 «Информационные технологии и прикладная
математика»**

Кафедра 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Численные методы»

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

Москва, 2024

1 Методы приближения функций. Численное дифференцирование и интегрирование

1 Постановка задачи

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

Вариант: 20

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

а)

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

б)

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

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

2 Результаты работы

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

Рис. 1: Вывод в консоли

3 Исходный код

matrix.h

```
1  #pragma once  
2  #include <iostream>  
3  #include <vector>  
4  #include <ccomplex>  
5  #include <fstream>  
6  
7  using namespace std;  
8  
9  using cmd = complex <double>;  
10 const double pi = acos(-1);  
11  
12 struct matrix  
13 {  
14     int rows = 0, cols = 0;  
15     vector <vector <double>> v;  
16  
17     matrix() {}  
18     matrix(int _rows, int _cols)  
19     {  
20         rows = _rows;  
21         cols = _cols;  
22         v = vector <vector <double>>(rows, vector <double>(cols));  
23     }  
24  
25     vector <double>& operator[] (int row)  
26     {  
27         return v[row];  
28     }  
29  
30     operator double()
```

```

31     {
32         return v[0][0];
33     }
34 };
35
36 matrix operator*(matrix lhs, matrix rhs)
37 {
38     if (lhs.cols != rhs.rows)
39         return matrix(0, 0);
40     matrix res(lhs.rows, rhs.cols);
41     for (int i = 0; i < res.rows; i++)
42     {
43         for (int j = 0; j < res.cols; j++)
44         {
45             res[i][j] = 0;
46             for (int k = 0; k < lhs.cols; k++)
47                 res[i][j] += lhs[i][k] * rhs[k][j];
48         }
49     }
50     return res;
51 }
52
53 matrix operator*(double lhs, matrix rhs)
54 {
55     for (int i = 0; i < rhs.rows; i++)
56     {
57         for (int j = 0; j < rhs.cols; j++)
58             rhs[i][j] *= lhs;
59     }
60     return rhs;
61 }
62
63 matrix operator+(matrix lhs, matrix rhs)
64 {
65     if (lhs.rows != rhs.rows || rhs.cols != lhs.cols)
66         return matrix(0, 0);
67     matrix res(lhs.rows, lhs.cols);
68     for (int i = 0; i < rhs.rows; i++)
69     {
70         for (int j = 0; j < res.cols; j++)
71             res[i][j] = lhs[i][j] + rhs[i][j];
72     }
73     return res;
74 }
75
76 matrix operator-(matrix lhs, matrix rhs)
77 {
78     if (lhs.rows != rhs.rows || rhs.cols != lhs.cols)
79         return matrix(0, 0);

```

```

80     matrix res(lhs.rows, lhs.cols);
81     for (int i = 0; i < rhs.rows; i++)
82     {
83         for (int j = 0; j < res.cols; j++)
84             res[i][j] = lhs[i][j] - rhs[i][j];
85     }
86     return res;
87 }
88
89 ostream& operator<<(ostream& stream, matrix a)
90 {
91     for (int i = 0; i < a.rows; i++)
92     {
93         for (int j = 0; j < a.cols; j++)
94             stream << a[i][j] << ' ';
95         stream << '\n';
96     }
97     return stream;
98 }
99
100 istream& operator>>(istream& stream, matrix& a)
101 {
102     for (int i = 0; i < a.rows; i++)
103     {
104         for (int j = 0; j < a.cols; j++)
105             stream >> a[i][j];
106     }
107     return stream;
108 }
109
110 matrix transposition(matrix a)
111 {
112     matrix res(a.cols, a.rows);
113     for (int i = 0; i < a.rows; i++)
114     {
115         for (int j = 0; j < a.cols; j++)
116             res[j][i] = a[i][j];
117     }
118     return res;
119 }
120
121 vector<int> swp;
122
123 pair<matrix, matrix> lu_decomposition(matrix a)
124 {
125     int n = a.rows;
126     matrix l(n, n);
127     swp = vector<int>(0);
128     for (int k = 0; k < n; k++)

```

```

129 {
130     matrix prev = a;
131     int idx = k;
132     for (int i = k + 1; i < n; i++)
133     {
134         if (abs(prev[idx][k]) < abs(prev[i][k]))
135             idx = i;
136     }
137     swap(prev[k], prev[idx]);
138     swap(a[k], a[idx]);
139     swap(l[k], l[idx]);
140     swp.push_back(idx);
141     for (int i = k + 1; i < n; i++)
142     {
143         double h = prev[i][k] / prev[k][k];
144         l[i][k] = h;
145         for (int j = k; j < n; j++)
146             a[i][j] = prev[i][j] - h * prev[k][j];
147     }
148 }
149 }
150 for (int i = 0; i < n; i++)
151     l[i][i] = 1;
152 return { l, a };
153 }
154
155 matrix solve_triag(matrix a, matrix b, bool up)
156 {
157     int n = a.rows;
158     matrix res(n, 1);
159     int d = up ? -1 : 1;
160     int first = up ? n - 1 : 0;
161     for (int i = first; i < n && i >= 0; i += d)
162     {
163         res[i][0] = b[i][0];
164         for (int j = 0; j < n; j++)
165         {
166             if (i != j)
167                 res[i][0] -= a[i][j] * res[j][0];
168         }
169         res[i][0] = res[i][0] / a[i][i];
170     }
171     return res;
172 }
173
174 matrix solve_gauss(pair <matrix, matrix> lu, matrix b)
175 {
176     for (int i = 0; i < swp.size(); i++)
177         swap(b[i], b[swp[i]]);

```

```

178     matrix z = solve_triag(lu.first, b, false);
179     matrix x = solve_triag(lu.second, z, true);
180     //for (int i = 0; i < swp.size(); i++)
181         //swap(x[i], x[swp[i]]);
182     return x;
183 }
184
185 matrix inverse(matrix a)
186 {
187     int n = a.rows;
188     matrix b(n, 1);
189     pair <matrix, matrix> lu = lu_decomposition(a);
190     matrix res(n, n);
191     for (int i = 0; i < n; i++)
192     {
193         b[max(i - 1, 0)][0] = 0;
194         b[i][0] = 1;
195         matrix col = solve_gauss(lu, b);
196         for (int j = 0; j < n; j++)
197             res[j][i] = col[j][0];
198     }
199     return res;
200 }
201
202 double determinant(matrix a)
203 {
204     int n = a.rows;
205     pair <matrix, matrix> lu = lu_decomposition(a);
206     double det = 1;
207     for (int i = 0; i < n; i++)
208         det *= lu.second[i][i];
209     return det;
210 }
211
212 matrix solve_tridiagonal(matrix& a, matrix& b)
213 {
214     int n = a.rows;
215     vector <double> p(n), q(n);
216     p[0] = -a[0][1] / a[0][0];
217     q[0] = b[0][0] / a[0][0];
218     for (int i = 1; i < n; i++)
219     {
220         if (i != n - 1)
221             p[i] = -a[i][i + 1] / (a[i][i] + a[i][i - 1] * p[i - 1]);
222         else
223             p[i] = 0;
224         q[i] = (b[i][0] - a[i][i - 1] * q[i - 1]) / (a[i][i] + a[i][i - 1] * p[i - 1]);
225     }
226     matrix res(n, 1);

```

```

227     res[n - 1][0] = q[n - 1];
228     for (int i = n - 2; i >= 0; i--)
229         res[i][0] = p[i] * res[i + 1][0] + q[i];
230     return res;
231 }
232
233 double abs(matrix a)
234 {
235     double mx = 0;
236     for (int i = 0; i < a.rows; i++)
237     {
238         double s = 0;
239         for (int j = 0; j < a.cols; j++)
240             s += abs(a[i][j]);
241         mx = max(mx, s);
242     }
243     return mx;
244 }
245
246 matrix solve_iteration(matrix a, matrix b, double eps)
247 {
248     int n = a.rows;
249     matrix alpha(n, n), beta(n, 1);
250     for (int i = 0; i < n; i++)
251     {
252         for (int j = 0; j < n; j++)
253             alpha[i][j] = -a[i][j] / a[i][i];
254         alpha[i][i] = 0;
255     }
256     for (int i = 0; i < n; i++)
257         beta[i][0] = b[i][0] / a[i][i];
258     matrix x = beta;
259     double m = abs(a);
260     double epsk = 2 * eps;
261     while (epsk > eps)
262     {
263         matrix prev = x;
264         x = beta + alpha * x;
265         if (m < 1)
266             epsk = m / (1 - m) * abs(x - prev);
267         else
268             epsk = abs(x - prev);
269     }
270     return x;
271 }
272
273 matrix solve_seidel(matrix a, matrix b, double eps)
274 {
275     int n = a.rows;

```



```

276 matrix alpha(n, n), beta(n, 1);
277 for (int i = 0; i < n; i++)
278 {
279     for (int j = 0; j < n; j++)
280         alpha[i][j] = -a[i][j] / a[i][i];
281     alpha[i][i] = 0;
282 }
283 for (int i = 0; i < n; i++)
284     beta[i][0] = b[i][0] / a[i][i];
285 matrix x = beta;
286 double m = abs(alpha);
287 double epsk = 2 * eps;
288 while (epsk > eps)
289 {
290     matrix prev = x;
291     for (int i = 0; i < n; i++)
292     {
293         double cur = beta[i][0];
294         for (int j = 0; j < n; j++)
295             cur += alpha[i][j] * x[j][0];
296         x[i][0] = cur;
297     }
298     if (m < 1)
299         epsk = m / (1 - m) * abs(x - prev);
300     else
301         epsk = abs(x - prev);
302 }
303 return x;
304 }
305
306 pair <matrix, matrix> method_jacobi(matrix a, double eps)
307 {
308     int n = a.rows;
309     double epsk = 2 * eps;
310     matrix vec(n, n);
311     for (int i = 0; i < n; i++)
312         vec[i][i] = 1;
313     while (epsk > eps)
314     {
315         int cur_i = 1, cur_j = 0;
316         for (int i = 0; i < n; i++)
317         {
318             for (int j = 0; j < i; j++)
319             {
320                 if (abs(a[cur_i][cur_j]) < abs(a[i][j]))
321                 {
322                     cur_i = i;
323                     cur_j = j;
324                 }

```

```

325     }
326 }
327 matrix u(n, n);
328 double phi = pi / 4;
329 if (abs(a[cur_i][cur_i] - a[cur_j][cur_j]) > 1e-7)
330     phi = 0.5 * atan((2 * a[cur_i][cur_j]) / (a[cur_i][cur_i] - a[cur_j][cur_j]
331         ));
332 for (int i = 0; i < n; i++)
333     u[i][i] = 1;
334 u[cur_i][cur_j] = -sin(phi);
335 u[cur_i][cur_i] = cos(phi);
336 u[cur_j][cur_i] = sin(phi);
337 u[cur_j][cur_j] = cos(phi);
338 vec = vec * u;
339 a = transposition(u) * a * u;
340 epsk = 0;
341 for (int i = 0; i < n; i++)
342 {
343     for (int j = 0; j < i; j++)
344         epsk += a[i][j] * a[i][j];
345 }
346 epsk = sqrt(epsk);
347 }
348 matrix val(n, 1);
349 for (int i = 0; i < n; i++)
350     val[i][0] = a[i][i];
351 return { val, vec };
352 }
353 double sign(double x)
354 {
355     return x > 0 ? 1 : -1;
356 }
357
358 pair <matrix, matrix> qr_decomposition(matrix a)
359 {
360     int n = a.rows;
361     matrix e(n, n);
362     for (int i = 0; i < n; i++)
363         e[i][i] = 1;
364     matrix q = e;
365     for (int i = 0; i < n - 1; i++)
366     {
367         matrix v(n, 1);
368         double s = 0;
369         for (int j = i; j < n; j++)
370             s += a[j][i] * a[j][i];
371         v[i][0] = a[i][i] + sign(a[i][i]) * sqrt(s);
372         for (int j = i + 1; j < n; j++)

```

```

373         v[j][0] = a[j][i];
374         matrix h = e - (2.0 / double(transposition(v) * v)) * (v * transposition(v));
375         q = q * h;
376         a = h * a;
377     }
378     return { q, a };
379 }
380
381 vector <cmd> qr_eigenvalues(matrix a, double eps)
382 {
383     int n = a.rows;
384     vector <cmd> prev(n);
385     while (true)
386     {
387         pair <matrix, matrix> p = qr_decomposition(a);
388         a = p.second * p.first;
389         vector <cmd> cur;
390         for (int i = 0; i < n; i++)
391         {
392             if (i < n - 1 && abs(a[i + 1][i]) > 1e-7)
393             {
394                 double b = -(a[i][i] + a[i + 1][i + 1]);
395                 double c = a[i][i] * a[i + 1][i + 1] - a[i][i + 1] * a[i + 1][i];
396                 double d = b * b - 4 * c;
397                 cmd sgn = (d > 0) ? cmd(1, 0) : cmd(0, 1);
398                 d = sqrt(abs(d));
399                 cur.push_back(0.5 * (-b - sgn * d));
400                 cur.push_back(0.5 * (-b + sgn * d));
401                 i++;
402             }
403             else
404                 cur.push_back(a[i][i]);
405         }
406         bool ok = true;
407         for (int i = 0; i < n; i++)
408             ok = ok && abs(cur[i] - prev[i]) < eps;
409         if (ok)
410             break;
411         prev = cur;
412     }
413     return prev;
414 }

```

3-1.cpp

```

1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <fstream>

```

```

5 | #include <functional>
6 | #include <algorithm>
7 | #include "matrix.h"
8 |
9 | using namespace std;
10 |
11 | struct polynomial
12 | {
13 | private:
14 |     vector <double> v;
15 |
16 | public:
17 |     polynomial(vector <double> _v = {})
18 |     {
19 |         v = _v;
20 |     }
21 |
22 |     double size()
23 |     {
24 |         return v.size();
25 |     }
26 |
27 |     double operator[](int idx)
28 |     {
29 |         return v[idx];
30 |     }
31 |
32 |     double calculate(double x)
33 |     {
34 |         double res = 0;
35 |         double cur = 1;
36 |         for (int i = 0; i < v.size(); i++)
37 |         {
38 |             res += cur * v[i];
39 |             cur = cur * x;
40 |         }
41 |         return res;
42 |     }
43 | };
44 |
45 | ostream& operator<<(ostream& stream, polynomial a)
46 | {
47 |     for (int i = a.size() - 1; i > 0; i--)
48 |         stream << ((i == a.size() - 1) ? a[i] : abs(a[i])) << "*" << "x^" << i << ((a[i]
49 |             - 1] > 0) ? '+' : '-');
50 |     stream << abs(a[0]);
51 |     return stream;
52 | }

```

```

53 | vector <double> open_brackets(vector <double> v)
54 | {
55 |     if (v.size() == 1)
56 |         return { v[0], 1 };
57 |     int n = v.size();
58 |     double last = v.back();
59 |     v.erase(--v.end());
60 |     vector <double> res = open_brackets(v);
61 |     vector <double> tmp = res;
62 |     for (int i = 0; i < n; i++)
63 |         res[i] = res[i] * last;
64 |     res.push_back(0);
65 |     for (int i = 1; i <= n; i++)
66 |         res[i] += tmp[i - 1];
67 |     return res;
68 | }
69 |
70 | polynomial interpolation_lagrange(vector <double> x, vector <double> y)
71 | {
72 |     int n = x.size();
73 |     vector <double> res(n);
74 |     for (int i = 0; i < n; i++)
75 |     {
76 |         vector <double> v;
77 |         double k = y[i];
78 |         for (int j = 0; j < n; j++)
79 |         {
80 |             if (i == j)
81 |                 continue;
82 |             v.push_back(-x[j]);
83 |             k = k / (x[i] - x[j]);
84 |         }
85 |         vector <double> tmp = open_brackets(v);
86 |         for (int j = 0; j < n; j++)
87 |             res[j] += k * tmp[j];
88 |     }
89 |     return polynomial(res);
90 | }
91 |
92 | polynomial interpolation_newton(vector <double> x, vector <double> y)
93 | {
94 |     int n = x.size();
95 |     vector <double> res(n);
96 |     res[0] = y[0];
97 |     vector <vector <double>> diff(n - 1, vector <double>(n - 1));
98 |     for (int i = 0; i < n - 1; i++)
99 |         diff[0][i] = (y[i] - y[i + 1]) / (x[i] - x[i + 1]);
100 |     for (int i = 1; i < n - 1; i++)
101 |     {

```

```

102     for (int j = 0; j < n - 1 - i; j++)
103         diff[i][j] = (diff[i - 1][j] - diff[i - 1][j + 1]) / (x[j] - x[j + 1 + i]);
104     }
105     vector <double> cur;
106     for (int i = 0; i < n - 1; i++)
107     {
108         cur.push_back(-x[i]);
109         vector <double> tmp = open_brackets(cur);
110         double k = diff[i][0];
111         for (int j = 0; j < tmp.size(); j++)
112             res[j] += k * tmp[j];
113     }
114     return polynomial(res);
115 }
116
117
118 double f1(double x)
119 {
120     return acos(x) + x;
121 }
122
123 int main()
124 {
125     setlocale(LC_ALL, "Rus");
126     ofstream fout("answer3-1.txt");
127     fout.precision(5);
128     fout << fixed;
129
130     vector <double> X = { -0.4, -0.1, 0.2, 0.5 };
131     vector <double> Y;
132     fout << "Значения функции в точках:\n";
133     for (int i = 0; i < X.size(); i++)
134     {
135         fout << f1(X[i]) << ' ';
136         Y.push_back(f1(X[i]));
137     }
138     polynomial p1 = interpolation_lagrange(X, Y);
139     fout << "\Многочленн Лагранжа: " << p1 << '\n';
140     polynomial p2 = interpolation_newton(X, Y);
141     fout << "Многочлен Ньютона: " << p2 << '\n';
142     fout << "Значения многочленов в точках:\n";
143     for (int i = 0; i < X.size(); i++)
144         fout << p1.calculate(X[i]) << ' ';
145
146 }

```

4 Постановка задачи

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

Вариант: 20

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

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

5 Результаты работы

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

Рис. 2: Вывод в консоли

6 Исходный код

3-2.cpp

```
1  #include <iostream>  
2  #include <vector>  
3  #include <cmath>  
4  #include <fstream>  
5  #include <functional>  
6  #include <algorithm>  
7  #include "matrix.h"  
8  
9  using namespace std;  
10  
11 struct polynomial  
12 {  
13 private:  
14     vector <double> v;  
15  
16 public:  
17     polynomial(vector <double> _v = {})  
18     {  
19         v = _v;  
20     }  
21  
22     double size()  
23     {  
24         return v.size();  
25     }  
26  
27     double operator[](int idx)  
28     {  
29         return v[idx];  
30     }
```



```

31
32     double calculate(double x)
33     {
34         double res = 0;
35         double cur = 1;
36         for (int i = 0; i < v.size(); i++)
37         {
38             res += cur * v[i];
39             cur = cur * x;
40         }
41         return res;
42     }
43 };
44
45 ostream& operator<<(ostream& stream, polynomial a)
46 {
47     for (int i = a.size() - 1; i > 0; i--)
48         stream << ((i == a.size() - 1) ? a[i] : abs(a[i])) << "*" << "x^" << i << ((a[i]
49             - 1] > 0) ? '+' : '-');
50     stream << abs(a[0]);
51     return stream;
52 }
53 struct cubic_spline
54 {
55 private:
56     vector <polynomial> p;
57     vector <double> xn;
58
59 public:
60     cubic_spline(vector <double> _xn = { 0 }, vector <polynomial> _p = {})
61     {
62         if (_p.size() == _xn.size() - 1)
63         {
64             p = _p;
65             xn = _xn;
66         }
67         else
68         {
69             p = {};
70             xn = { 0 };
71         }
72     }
73
74     int size()
75     {
76         return p.size();
77     }
78

```

```

79     pair <pair <double, double>, polynomial> operator[](int idx)
80     {
81         return { {xn[idx], xn[idx + 1]}, p[idx] };
82     }
83
84     double calculate(double x)
85     {
86         int idx = upper_bound(xn.begin(), xn.end(), x) - xn.begin();
87         idx = min(idx, (int)xn.size() - 1) - 1;
88         idx = max(0, idx);
89         return p[idx].calculate(x);
90     }
91 };
92
93 ostream& operator<<(ostream& stream, cubic_spline a)
94 {
95     for (int i = 0; i < a.size(); i++)
96         stream << "[" << a[i].first.first << "," << a[i].first.second << "]" << a[i].
97             second << endl;
98     return stream;
99 }
100 vector <double> open_brackets(vector <double> v)
101 {
102     if (v.size() == 1)
103         return { v[0], 1 };
104     int n = v.size();
105     double last = v.back();
106     v.erase(--v.end());
107     vector <double> res = open_brackets(v);
108     vector <double> tmp = res;
109     for (int i = 0; i < n; i++)
110         res[i] = res[i] * last;
111     res.push_back(0);
112     for (int i = 1; i <= n; i++)
113         res[i] += tmp[i - 1];
114     return res;
115 }
116
117 cubic_spline make_spline(vector <double> x, vector <double> y)
118 {
119     int n = x.size();
120     vector <pair <double, double>> xy(n);
121     for (int i = 0; i < n; i++)
122         xy[i] = { x[i], y[i] };
123     sort(xy.begin(), xy.end());
124     for (int i = 0; i < n; i++)
125     {
126         x[i] = xy[i].first;

```

```

127     y[i] = xy[i].second;
128 }
129 n--;
130 vector <double> a(n), b(n), c(n), d(n);
131 vector <double> h(n);
132 for (int i = 0; i < n; i++)
133     h[i] = x[i + 1] - x[i];
134 matrix A(n - 1, n - 1), B(n - 1, 1);
135 for (int i = 0; i < n - 1; i++)
136 {
137     if (i > 0)
138         A[i][i - 1] = h[i];
139     A[i][i] = 2 * (h[i] + h[i + 1]);
140     if (i < n - 2)
141         A[i][i + 1] = h[i + 1];
142     B[i][0] = 3 * ((y[i + 2] - y[i + 1]) / h[i + 1] - (y[i + 1] - y[i]) / h[i]);
143 }
144 matrix s = solve_tridiagonal(A, B);
145 for (int i = 1; i < n; i++)
146     c[i] = s[i - 1][0];
147 for (int i = 0; i < n; i++)
148     a[i] = y[i];
149 for (int i = 0; i < n - 1; i++)
150     b[i] = (y[i + 1] - y[i]) / h[i] - 1. / 3. * (c[i + 1] + 2 * c[i]);
151 b[n - 1] = (y[n] - y[n - 1]) / h[n - 1] - 2. / 3. * h[n - 1] * c[n - 1];
152 for (int i = 0; i < n - 1; i++)
153     d[i] = (c[i + 1] - c[i]) / (3 * h[i]);
154 d[n - 1] = -c[n - 1] / (3 * h[n - 1]);
155 vector <polynomial> vp(n);
156 for (int i = 0; i < n; i++)
157 {
158     vector <double> res(4);
159     res[0] = a[i];
160     vector <double> tmp;
161     for (int j = 1; j < 4; j++)
162     {
163         tmp.push_back(-x[i]);
164         vector <double> v = open_brackets(tmp);
165         for (int k = 0; k < v.size(); k++)
166             res[k] += v[k];
167     }
168     vp[i] = polynomial(res);
169 }
170 return cubic_spline(x, vp);
171 }
172
173 int main()
174 {
175     setlocale(LC_ALL, "Rus");

```

```

176 | ofstream fout("answer3-2.txt");
177 | fout.precision(5);
178 | fout << fixed;
179 |
180 |
181 | vector <double> X = { -0.4, -0.1, 0.2, 0.5, 0.8 };
182 | vector <double> Y = { 1.5823, 1.5710, 1.5694, 1.5472, 1.4435 };
183 | cubic_spline cs = make_spline(X, Y);
184 | fout << cs;
185 | fout << "Значение сплайна в точке: " << cs.calculate(0.1);
186 |
187 | }

```

7 Постановка задачи

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

Вариант: 20

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

8 Результаты работы

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

Рис. 3: Вывод в консоли

9 Исходный код

3-3.cpp

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <fstream>
4 | #include <cmath>
5 | #include "matrix.h"
6 |
7 | using namespace std;
8 |
9 | struct polynomial {
10 | private:
11 |     vector<double> v;
12 |
13 | public:
14 |     polynomial(vector<double> _v = {}) {
15 |         v = _v;
16 |     }
17 |
18 |     double size() const {
19 |         return v.size();
20 |     }
21 |
22 |     double operator[](int idx) const {
23 |         return v[idx];
24 |     }
25 |
26 |     double& operator[](int idx) {
27 |         return v[idx];
28 |     }
29 |
30 |     double calculate(double x) const {
31 |         double res = 0;
32 |         double cur = 1;
```

```

33     for (int i = 0; i < v.size(); i++) {
34         res += cur * v[i];
35         cur = cur * x;
36     }
37     return res;
38 }
39 };
40
41 ostream& operator<<(ostream& stream, polynomial a)
42 {
43     for (int i = a.size() - 1; i > 0; i--)
44         stream << ((i == a.size() - 1) ? a[i] : abs(a[i])) << "*" << "x^" << i << ((a[i]
45             - 1] > 0) ? '+' : '-');
46     stream << abs(a[0]);
47     return stream;
48 }
49 polynomial least_squares(const vector<double>& x, const vector<double>& y, int m,
50     double& sum_sq_error) {
51     int n = x.size();
52     m++;
53     matrix phi(n, m);
54     for (int i = 0; i < n; i++) {
55         for (int j = 0; j < m; j++) {
56             phi[i][j] = pow(x[i], j);
57         }
58     }
59
60     matrix G = transposition(phi) * phi;
61     matrix Y(n, 1);
62     for (int i = 0; i < n; i++)
63         Y[i][0] = y[i];
64     matrix Z = transposition(phi) * Y;
65     matrix A = solve_gauss(lu_decomposition(G), Z);
66
67     vector<double> a(m);
68     for (int i = 0; i < m; i++)
69         a[i] = A[i][0];
70
71     sum_sq_error = 0.0;
72     for (int i = 0; i < n; i++) {
73         double approx_y = 0.0;
74         for (int j = 0; j < m; j++) {
75             approx_y += a[j] * pow(x[i], j);
76         }
77         sum_sq_error += pow(y[i] - approx_y, 2);
78     }
79 }

```

```

80     return polynomial(a);
81 }
82
83 int main() {
84     vector<double> x = {-0.7, -0.4, -0.1, 0.2, 0.5, 0.8};
85     vector<double> y = {1.6462, 1.5823, 1.571, 1.5694, 1.5472, 1.4435};
86
87     ofstream fout("answer3-3.txt");
88     fout.precision(5);
89     fout << fixed;
90
91     double error1;
92     polynomial p1 = least_squares(x, y, 1, error1);
93
94     double error2;
95     polynomial p2 = least_squares(x, y, 2, error2);
96
97     fout << "Приближающий многочленой1- степени: ";
98     fout << p1 << endl;
99
100    fout << "Приближающий многочленой2- степени: ";
101    fout << p2 << endl;
102
103    fout << "Сумма квадратовошибокдляой1- степени: " << error1 << endl;
104    fout << "Сумма квадратовошибокдляой2- степени: " << error2 << endl;
105 }

```


10 Постановка задачи

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

Вариант: 20

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

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

11 Результаты работы

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

Рис. 4: Вывод в консоли

12 Исходный код

3-4.cpp

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 | #include <fstream>
5 | #include <functional>
6 | #include <algorithm>
7 | #include "matrix.h"
8 |
9 | using namespace std;
10 |
11 | struct polynomial
12 | {
13 | private:
14 |     vector <double> v;
15 |
16 | public:
17 |     polynomial(vector <double> _v = {})
18 |     {
19 |         v = _v;
20 |     }
21 |
22 |     double size()
23 |     {
24 |         return v.size();
25 |     }
26 |
27 |     double operator[](int idx)
28 |     {
29 |         return v[idx];
30 |     }
31 |
32 |     double calculate(double x)
33 |     {
34 |         double res = 0;
```

```

35     double cur = 1;
36     for (int i = 0; i < v.size(); i++)
37     {
38         res += cur * v[i];
39         cur = cur * x;
40     }
41     return res;
42 }
43 };
44
45 ostream& operator<<(ostream& stream, polynomial a)
46 {
47     for (int i = a.size() - 1; i > 0; i--)
48         stream << ((i == a.size() - 1) ? a[i] : abs(a[i])) << "*" << "x^" << i << ((a[i]
49             - 1] > 0) ? '+' : '-');
50     stream << abs(a[0]);
51     return stream;
52 }
53 struct cubic_spline
54 {
55 private:
56     vector <polynomial> p;
57     vector <double> xn;
58
59 public:
60     cubic_spline(vector <double> _xn = { 0 }, vector <polynomial> _p = {})
61     {
62         if (_p.size() == _xn.size() - 1)
63         {
64             p = _p;
65             xn = _xn;
66         }
67         else
68         {
69             p = {};
70             xn = { 0 };
71         }
72     }
73
74     int size()
75     {
76         return p.size();
77     }
78
79     pair <pair <double, double>, polynomial> operator[](int idx)
80     {
81         return { {xn[idx], xn[idx + 1]}, p[idx] };
82     }

```

```

83
84     double calculate(double x)
85     {
86         int idx = upper_bound(xn.begin(), xn.end(), x) - xn.begin();
87         idx = min(idx, (int)xn.size() - 1) - 1;
88         idx = max(0, idx);
89         return p[idx].calculate(x);
90     }
91 };
92
93 vector <double> open_brackets(vector <double> v)
94 {
95     if (v.size() == 1)
96         return { v[0], 1 };
97     int n = v.size();
98     double last = v.back();
99     v.erase(--v.end());
100    vector <double> res = open_brackets(v);
101    vector <double> tmp = res;
102    for (int i = 0; i < n; i++)
103        res[i] = res[i] * last;
104    res.push_back(0);
105    for (int i = 1; i <= n; i++)
106        res[i] += tmp[i - 1];
107    return res;
108 }
109
110 polynomial interpolation_lagrange(vector <double> x, vector <double> y)
111 {
112     int n = x.size();
113     vector <double> res(n);
114     for (int i = 0; i < n; i++)
115     {
116         vector <double> v;
117         double k = y[i];
118         for (int j = 0; j < n; j++)
119         {
120             if (i == j)
121                 continue;
122             v.push_back(-x[j]);
123             k = k / (x[i] - x[j]);
124         }
125         vector <double> tmp = open_brackets(v);
126         for (int j = 0; j < n; j++)
127             res[j] += k * tmp[j];
128     }
129     return polynomial(res);
130 }
131

```

```

132 polynomial derivative(polynomial p)
133 {
134     int n = p.size();
135     vector <double> res;
136     for (int i = 1; i < n; i++)
137         res.push_back(i * p[i]);
138     return polynomial(res);
139 }
140
141 function <double(double)> derivative(vector <double> x, vector <double> y, int m)
142 {
143     int n = x.size();
144     vector <pair <double, double>> xy(n);
145     for (int i = 0; i < n; i++)
146         xy[i] = { x[i], y[i] };
147     sort(xy.begin(), xy.end());
148     for (int i = 0; i < n; i++)
149     {
150         x[i] = xy[i].first;
151         y[i] = xy[i].second;
152     }
153     vector <polynomial> p(n - m);
154     for (int i = 0; i < n - m; i++)
155     {
156         vector <double> xm, ym;
157         for (int j = 0; j < m + 1; j++)
158         {
159             xm.push_back(x[i + j]);
160             ym.push_back(y[i + j]);
161         }
162         p[i] = interpolation_lagrange(xm, ym);
163         for (int j = 0; j < m; j++)
164             p[i] = derivative(p[i]);
165     }
166     auto res = [=](double _x) mutable
167     {
168         int idx = lower_bound(x.begin() + 1, x.end() - m, _x) - x.begin();
169         //int idx = upper_bound(x.begin() + 1, x.end() - m, _x) - x.begin();
170         idx = idx - 1;
171         return p[idx].calculate(_x);
172     };
173     return res;
174 }
175
176 int main()
177 {
178     setlocale(LC_ALL, "Rus");
179     ofstream fout("answer3-4.txt");
180     fout.precision(5);

```

```

181 | fout << fixed;
182 | function <double(double)> df = derivative({-1, 0, 1, 2, 3}, {1.3562, 1.5708,
    | 1.7854, 2.4636, 3.3218}, 1);
183 | function <double(double)> ddf = derivative({-1, 0, 1, 2, 3}, {1.3562, 1.5708,
    | 1.7854, 2.4636, 3.3218}, 2);
184 | fout << "Значение первойпроизводной: " << df(1);
185 | fout << "\Значениен второйпроизводной: " << ddf(1);
186 | }

```

13 Постановка задачи

3.5. Вычислить определенный интеграл

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

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

Вариант: 20

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

14 Результаты работы

```
Метод прямоугольников:  
при h1: 0.53182  
при h2: 0.53139  
уточнение Рунге-Ромберга: 0.53125  
Метод трапеций:  
при h1: 0.52993  
при h2: 0.53087  
уточнение Рунге-Ромберга: 0.53119  
Метод Симпсона:  
при h1: 0.53090  
при h2: 0.53119  
уточнение Рунге-Ромберга: 0.53121
```

Рис. 5: Вывод в консоли

15 Исходный код

3-5.cpp

```
1 | #include <iostream>  
2 | #include <vector>  
3 | #include <cmath>  
4 | #include <fstream>  
5 | #include <functional>  
6 | #include <algorithm>  
7 | #include "matrix.h"  
8 |  
9 | using namespace std;  
10 |  
11 |  
12 | double integrate_rectangles(function <double(double)> f, double x0, double x1, double  
    | h)  
13 | {  
14 |     double res = 0;  
15 |     while (x0 < x1)  
16 |     {  
17 |         double x = x0 + h;  
18 |         res += f((x + x0) / 2) * h;  
19 |         x0 = x;  
20 |     }
```



```

21     return res;
22 }
23
24 double integrate_trapezoids(function <double(double)> f, double x0, double x1, double
    h)
25 {
26     double res = 0;
27     while (x0 < x1)
28     {
29         double x = x0 + h;
30         res += (f(x0) + f(x)) * h;
31         x0 = x;
32     }
33     return res / 2;
34 }
35
36 double integrate_simpson(function <double(double)> f, double x0, double x1, double h)
37 {
38     double res = 0;
39     while (x0 < x1)
40     {
41         double x = x0 + 2 * h;
42         double xm = x0 + h;
43         res += (f(x0) + 4 * f(xm) + f(x)) * h;
44         x0 = x;
45     }
46     return res / 3;
47 }
48
49 // rectangles -> p = 2
50 // trapezoids -> p = 2
51 // simpson -> p = 4
52 double method_runge(double i1, double i2, double h1, double h2, double p)
53 {
54     double k = h2 / h1;
55     return i1 + (i1 - i2) / (pow(k, p) - 1);
56 }
57
58 double f2(double x)
59 {
60     return sqrt(x) / (4 + 3 * x);
61 }
62
63 int main()
64 {
65     setlocale(LC_ALL, "Rus");
66     ofstream fout("answer3-5.txt");
67     fout.precision(5);
68     fout << fixed;

```

```

69
70     double h1 = 1, h2 = 0.5;
71     fout << "Метод прямоугольников:\n";
72     double i1 = integrate_rectangles(f2, 1, 5, h1);
73     double i2 = integrate_rectangles(f2, 1, 5, h2);
74     fout << "при h1: " << i1;
75     fout << "\nпри h2: " << i2;
76     fout << "\nУточнение Рунге-Ромберга-: " << method_runge(i1, i2, h1, h2, 2);
77     fout << "\nМетод трапеций:\n";
78     i1 = integrate_trapezoids(f2, 1, 5, h1);
79     i2 = integrate_trapezoids(f2, 1, 5, h2);
80     fout << "при h1: " << i1;
81     fout << "\nпри h2: " << i2;
82     fout << "\nУточнение Рунге-Ромберга-: " << method_runge(i1, i2, h1, h2, 2);
83     fout << "\nМетод Симпсона:\n";
84     i1 = integrate_simpson(f2, 1, 5, h1);
85     i2 = integrate_simpson(f2, 1, 5, h2);
86     fout << "при h1: " << i1;
87     fout << "\nпри h2: " << i2;
88     fout << "\nУточнение Рунге-Ромберга-: " << method_runge(i1, i2, h1, h2, 4);
89 }

```