

**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: Я. А. Хайруллина  
Преподаватель: Д. Е. Пивоваров  
Группа: М8О-303Б-21  
Дата:  
Оценка:  
Подпись:

**Москва, 2024**

## 3.1

### 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: 26**

$y = 1/x^2 + x^2, a) X_i = 0.1, 0.5, 0.9, 1.3; ) X_i = 0.1, 0.5, 1.1, 1.3; X^* = 0.8$

### 2 Результаты работы

```
Lagrange method
for x1
L(x)
[100.01 4.25      2.04457 2.28172 ]
y(x)
[100.01 4.25      2.04457 2.28172 ]
delta(x)
[0      0      0      4.44089e-16      ]

for x2
L(x)
[100.01 4.25      2.04457 2.28172 ]
y(x)
[100.01 4.25      2.03645 2.28172 ]
delta(x)
[1.42109e-14      8.88178e-16      0      0      ]

Newton method
for x1
P(x)
[100.01 4.25      2.04457 2.28172 ]
y(x)
[100.01 4.25      2.04457 2.28172 ]
delta(x)
[0      0      3.9968e-15      1.95399e-14      ]

for x2
P(x)
[100.01 4.25      2.03645 2.28172 ]
y(x)
[100.01 4.25      2.03645 2.28172 ]
delta(x)
[0      0      4.70735e-14      5.15143e-14      ]
```

Рис. 1: Вывод программы в консоли

### 3 Исходный код

```
1 #include <iostream>
2 #include <fstream>
3 #include <cmath>
4 #include <vector>
5
6 using namespace std;
7
8 double f(double x1, double x2) {
9     return ((pow(1 / x1, 2) + pow(x1, 2)) - (pow(1 / x2, 2) + pow(x2, 2))) / (x1 - x2);
10 }
11
12 double f1(double x1, double x2, double x3) {
13     return (f(x1, x2) - f(x2, x3)) / (x1 - x3);
14 }
15
16 double f2(double x1, double x2, double x3, double x4) {
17     return (f1(x1, x2, x3) - f1(x2, x3, x4)) / (x1 - x4);
18 }
19
20 vector <double> omega_val(vector <double> x) {
21     vector <double> omega(4, 0);
22     omega[0] = (x[0] - x[1]) * (x[0] - x[2]) * (x[0] - x[3]);
23     omega[1] = (x[1] - x[0]) * (x[1] - x[2]) * (x[1] - x[3]);
24     omega[2] = (x[2] - x[0]) * (x[2] - x[1]) * (x[2] - x[3]);
25     omega[3] = (x[3] - x[0]) * (x[3] - x[1]) * (x[3] - x[2]);
26     return omega;
27 }
28
29 void Lagrange(vector <double> x, double X, vector <double>& L, vector <double>& y,
30     vector <double>& delta) {
31     vector <vector <double>> ans(5, vector<double>(4, 0));
32     vector <double> omega = omega_val(x);
33     for (int i = 0; i < x.size(); i++) {
34         ans[0][i] = x[i];
35         ans[1][i] = pow(1 / x[i], 2) + pow(x[i], 2);
36         ans[2][i] = omega[i];
37         ans[3][i] = (pow(1 / x[i], 2) + pow(x[i], 2)) / omega[i];
38         ans[4][i] = X - x[i];
39     }
40     for (int i = 0; i < x.size(); i++) {
41         L[i] = (ans[3][0] * (x[i] - ans[0][1]) * (x[i] - ans[0][2]) * (x[i] - ans
42             [0][3]) \
43             + ans[3][1] * (x[i] - ans[0][0]) * (x[i] - ans[0][2]) * (x[i] - ans[0][3])
44             \
45             + ans[3][2] * (x[i] - ans[0][0]) * (x[i] - ans[0][1]) * (x[i] - ans[0][3])
46             \
47             + ans[3][3] * (x[i] - ans[0][0]) * (x[i] - ans[0][1]) * (x[i] - ans[0][2])
48             \
49             + ans[4][i] * ans[2][i]) / (ans[1][i] * ans[2][i]);
50         y[i] = ans[0][i];
51         delta[i] = ans[4][i];
52     }
53 }
```

```

44         + ans[3][3] * (x[i] - ans[0][0]) * (x[i] - ans[0][1]) * (x[i] - ans[0][2]))
45         ;
46     }
47     for (int i = 0; i < x.size(); i++) {
48         y[i] = pow(1 / x[i], 2) + pow(x[i], 2);
49     }
50
51     for (int i = 0; i < x.size(); i++) {
52         delta[i] = fabs(y[i] - L[i]);
53     }
54 }
55
56
57 void Newton(vector <double> x, double X, vector <double>& P, vector <double>& y,
58             vector <double>& delta) {
59     vector <vector <double>> ans(5, vector<double>(4, 0));
60     for (int i = 0; i < x.size(); i++) {
61         ans[0][i] = x[i];
62         ans[1][i] = pow(1 / x[i], 2) + pow(x[i], 2);
63         if (i < 3) {
64             ans[2][i] = f(x[i], x[i + 1]);
65         }
66         if (i < 2) {
67             ans[3][i] = f1(x[i], x[i + 1], x[i + 2]);
68         }
69     }
70     ans[4][0] = f2(x[0], x[1], x[2], x[3]);
71
72     for (int i = 0; i < x.size(); i++) {
73         P[i] = (ans[1][0] + ans[2][0] * (x[i] - ans[0][0]) + ans[3][0] * (x[i] - ans
74             [0][0]) * (x[i] - ans[0][1]) \
75             + ans[4][0] * (x[i] - ans[0][0]) * (x[i] - ans[0][1]) * (x[i] - ans[0][2]))
76             ;
77     }
78
79     for (int i = 0; i < x.size(); i++) {
80         y[i] = pow(1 / x[i], 2) + pow(x[i], 2);
81     }
82
83     for (int i = 0; i < x.size(); i++) {
84         delta[i] = fabs(y[i] - P[i]);
85     }
86 }
87
88 int main() {
89     int n = 4;
90     vector <double> x1 = { 0.1, 0.5, 0.9, 1.3 };

```

```

89     vector<double> x2 = { 0.1, 0.5, 1.1, 1.3 };
90     double root = 0.8;
91     vector<double> L(n, 0), y(n, 0), delta(n, 0), P(n, 0);
92     std::cout << "Lagrange method" << std::endl;
93     Lagrange(x1, root, L, y, delta);
94     std::cout << "for x1" << std::endl << "L(x)\n" << "[";
95     for (int i = 0; i < L.size(); i++) std::cout << L[i] << "\t";
96     std::cout << "]\n" << "y(x)\n" << "[";
97     for (int i = 0; i < y.size(); i++) std::cout << y[i] << "\t";
98     std::cout << "]\n" << "delta(x)\n" << "[";
99     for (int i = 0; i < delta.size(); i++) std::cout << delta[i] << "\t";
100    std::cout << "]\n";
101    Lagrange(x2, root, P, y, delta);
102    std::cout << "\n\nfor x2" << std::endl << "L(x)\n" << "[";
103    for (int i = 0; i < L.size(); i++) std::cout << L[i] << "\t";
104    std::cout << "]\n" << "y(x)\n" << "[";
105    for (int i = 0; i < y.size(); i++) std::cout << y[i] << "\t";
106    std::cout << "]\n" << "delta(x)\n" << "[";
107    for (int i = 0; i < delta.size(); i++) std::cout << delta[i] << "\t";
108    std::cout << "]\n";
109    std::cout << "\n\nNewton method" << std::endl;
110    Newton(x1, root, P, y, delta);
111    std::cout << "for x1" << std::endl << "P(x)\n" << "[";
112    for (int i = 0; i < P.size(); i++) std::cout << P[i] << "\t";
113    std::cout << "]\n" << "y(x)\n" << "[";
114    for (int i = 0; i < y.size(); i++) std::cout << y[i] << "\t";
115    std::cout << "]\n" << "delta(x)\n" << "[";
116    for (int i = 0; i < delta.size(); i++) std::cout << delta[i] << "\t";
117    std::cout << "]\n";
118    Newton(x2, root, P, y, delta);
119    std::cout << "\n\nfor x2" << std::endl << "P(x)\n" << "[";
120    for (int i = 0; i < P.size(); i++) std::cout << P[i] << "\t";
121    std::cout << "]\n" << "y(x)\n" << "[";
122    for (int i = 0; i < y.size(); i++) std::cout << y[i] << "\t";
123    std::cout << "]\n" << "delta(x)\n" << "[";
124    for (int i = 0; i < delta.size(); i++) std::cout << delta[i] << "\t";
125    std::cout << "]\n";
126 }

```

## 3.2

### 4 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: 26**

$X_i = 0.8$

$i$	0	1	2	3	4
$x_i$	0.1	0.5	0.9	1.3	1.7
$f_i$	100.01	4.2500	2.0446	2.2817	3.2360

Рис. 2: Условие

### 5 Результаты работы

```
Function value in X with spline: 27.4016
```

Рис. 3: Вывод программы в консоли

## 6 Исходный код

```
1 #include <iostream>
2 #include <vector>
3 #include <fstream>
4 #include <cmath>
5
6 using namespace std;
7
8 vector<double> solution(vector<vector<double>>& A, vector<double>& b) {
9     int n = A.size();
10
11     for (int i = 0; i < n; i++) {
12         int max_row = i;
13         for (int k = i + 1; k < n; k++) {
14             if (abs(A[k][i]) > abs(A[max_row][i])) {
15                 max_row = k;
16             }
17         }
18         swap(A[i], A[max_row]);
19         swap(b[i], b[max_row]);
20         for (int k = i + 1; k < n; k++) {
21             double factor = A[k][i] / A[i][i];
22             for (int j = i; j < n; j++) {
23                 A[k][j] -= factor * A[i][j];
24             }
25             b[k] -= factor * b[i];
26         }
27     }
28
29     vector<double> x(n);
30     for (int i = n - 1; i >= 0; i--) {
31         x[i] = b[i];
32         for (int j = i + 1; j < n; j++) {
33             x[i] -= A[i][j] * x[j];
34         }
35         x[i] /= A[i][i];
36     }
37
38     return x;
39 }
40
41
42 double Spline(vector<double>& x, vector<double>& y, double X) {
43     vector<double> A(3, 0);
44     vector<vector<double>> B(3, vector<double>(3, 0));
45     vector<vector<double>> ans(4, vector<double>(4, 0));
46     vector<double> roots(3, 0);
47     double f = 0;
```

```

48
49     for (int i = 2; i < 5; i++) {
50         A[i - 2] = (3 * ((y[i] - y[i - 1]) / (x[i] - x[i - 1]) +
51             (y[i - 1] - y[i - 2]) / (x[i - 1] - x[i - 2]))));
52         for (int j = 0; j < B.size(); j++) {
53             for (int k = 0; k < B[0].size(); k++) {
54                 if (j == k) {
55                     B[j][k] = 2 * ((x[i - 1] - x[i - 2]) + (x[i] - x[i - 1]));
56                 }
57                 else if (j < k && (k != B.size() - 1 || j != 0)) {
58                     B[j][k] = (x[i] - x[i - 1]);
59                 }
60                 else if (j > k && (j != B.size() - 1 || k != 0)) {
61                     B[j][k] = (x[i - 1] - x[i - 2]);
62                 }
63             }
64         }
65     }
66
67     roots = solution(B, A);
68     roots.insert(roots.begin(), 0);
69
70     for (int i = 0; i < ans.size(); i++) {
71         for (int j = 0; j < ans[0].size(); j++) {
72             if (i != ans.size() - 1) {
73                 if (j == 0) {
74                     ans[i][j] = y[i];
75                 }
76                 if (j == 1) {
77                     ans[i][j] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]) - (
78                         (x[i + 1] - x[i]) * roots[i + 1] + 2 * roots[i]) / 3;
79                 }
80                 if (j == 2) {
81                     if (i == 0) {
82                         ans[i][j] = 0;
83                     }
84                     else {
85                         ans[i][j] = roots[i];
86                     }
87                 }
88                 if (j == 3) {
89                     ans[i][j] = (roots[i + 1] - roots[i]) / (3 * (x[i + 1] - x[i]));
90                 }
91             }
92             else {
93                 if (j == 0) {
94                     ans[i][j] = y[i];
95                 }
96                 if (j == 1) {

```



```

97         ans[i][j] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]) - (2 * (x[i + 1] -
98             x[i]) * roots[i]) / 3;
99     }
100     if (j == 2) {
101         ans[i][j] = roots[i];
102     }
103     if (j == 3) {
104         ans[i][j] = -roots[i] / (3 * (x[i + 1] - x[i]));
105     }
106 }
107 }
108
109 f = ans[1][0] + ans[1][1] * (X - x[1]) + ans[1][2] * pow((X - x[2]), 2) + ans[1][3]
    * pow((X - x[3]), 3);
110
111 return f;
112 }
113
114
115 int main() {
116     vector<double> x = { 0.1, 0.5, 0.9, 1.3, 1.7 };
117     vector<double> y = { 100.01, 4.25, 2.0446, 2.2817, 3.236 };
118     double X = 0.8;
119     std::cout << "Function value in X with spline: " << Spline(x, y, X) << std::endl;
120
121     return 0;
122 }

```

### 3.3

#### 7 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: 26**

$i$	0	1	2	3	4	5
$x_i$	0.1	0.5	0.9	1.3	1.7	2.1
$y_i$	100.01	4.250	2.0446	2.2817	3.236	4.6368

Рис. 4: Условия

#### 8 Результаты работы

```
Least squares method, 1st degree
Coefficients: [ 57.0983 -34.2622 ]
Sum of the error square: 4514.05
2st degree
Coefficients: [ 98.4497 -156.648 55.6298 ]
Sum of the error square: 1556.37
```

Рис. 5: Вывод программы в консоли

## 9 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 |
5 | std::vector<double> x = { 0.1, 0.5, 0.9, 1.3, 1.7, 2.1 };
6 | std::vector<double> y = { 100.01, 4.25, 2.0446, 2.2817, 3.236, 4.6368 };
7 |
8 | double pol(std::vector<double> a, double x) {
9 |     double sum = 0;
10 |    for (int i = 0; i < a.size(); ++i) {
11 |        sum += pow(x, i) * a[i];
12 |    }
13 |    return sum;
14 | }
15 |
16 | double error(std::vector<double> a, std::vector<double> x, std::vector<double> y) {
17 |     double sum = 0;
18 |     for (int i = 0; i < x.size(); ++i) {
19 |         sum += pow(pol(a, x[i]) - y[i], 2);
20 |     }
21 |     return sum;
22 | }
23 |
24 |
25 |
26 | std::vector<std::vector<std::vector<double>>>> getLU(std::vector<std::vector<double>>> A
27 | ) {
28 |     std::vector<std::vector<double>>> U = A;
29 |
30 |     std::vector<std::vector<double>>> L(U.size(), std::vector<double>(U.size()));
31 |     for (auto& l : L) std::fill(l.begin(), l.end(), 0.0);
32 |
33 |     for (int k = 0; k < U.size(); ++k) {
34 |         for (int i = k; i < U.size(); ++i) {
35 |             L[i][k] = U[i][k] / U[k][k];
36 |         }
37 |
38 |         for (int i = k + 1; i < U.size(); ++i) {
39 |             for (int j = k; j < U.size(); ++j) {
40 |                 U[i][j] = U[i][j] - L[i][k] * U[k][j];
41 |             }
42 |         }
43 |     }
44 |
45 |     return { L, U };
46 | }
```

```

47 std::vector<double> solution(std::vector<std::vector<double>>> A, std::vector<double> b
   ) {
48     int n = A.size();
49     std::vector<std::vector<double>>> LU = getLU(A);
50     std::vector<std::vector<double>>> L = LU[0];
51     std::vector<std::vector<double>>> U = LU[1];
52
53     std::vector<double> z(n);
54     z[0] = b[0];
55     for (int i = 1; i < n; ++i) {
56         double sum = 0;
57         for (int j = 0; j < i; ++j) {
58             sum += L[i][j] * z[j];
59         }
60         z[i] = b[i] - sum;
61     }
62
63     std::vector<double> x(n);
64     x[n - 1] = z[n - 1] / U[n - 1][n - 1];
65     for (int i = n - 2; i >= 0; --i) {
66         double sum = 0;
67         for (int j = i + 1; j < n; ++j) {
68             sum += U[i][j] * x[j];
69         }
70
71         x[i] = (z[i] - sum) / U[i][i];
72     }
73     return x;
74 }
75
76 std::vector<double> MNK(std::vector<double> x, std::vector<double> y, int m) {
77     std::vector<std::vector<double>>> A(m + 1, std::vector<double>(m + 1));
78     std::vector<double> b(m + 1);
79
80     for (int k = 0; k < m + 1; ++k) {
81         for (int j = 0; j < m + 1; ++j) {
82             double sum = 0;
83             for (int i = 0; i < x.size(); ++i) {
84                 sum += pow(x[i], k + j);
85             }
86             A[k][j] = sum;
87         }
88
89         double sum = 0;
90         for (int i = 0; i < x.size(); ++i) {
91             sum += y[i] * pow(x[i], k);
92         }
93         b[k] = sum;
94     }

```

```

95 |
96 |     std::vector<double> a = solution(A, b);
97 |     return a;
98 | }
99 |
100 |
101 |
102 | int main() {
103 |     std::cout << "Least squares method, 1st degree\n";
104 |     std::vector<double> a = MNK(x, y, 1);
105 |     std::cout << "Coefficients: [";
106 |     for (int i = 0; i < a.size(); ++i) std::cout << " " << a[i] << " ";
107 |     std::cout << "]\n";
108 |     std::cout << "Sum of the error square: " << error(a, x, y) << "\n";
109 |
110 |     std::cout << "2st degree\n";
111 |     a = MNK(x, y, 2);
112 |     std::cout << "Coefficients: [";
113 |     for (int i = 0; i < a.size(); ++i) std::cout << " " << a[i] << " ";
114 |     std::cout << "]\n";
115 |     std::cout << "Sum of the error square: " << error(a, x, y) << "\n";
116 |
117 |     return 0;
118 | }

```

## 3.4

### 10 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: 26**  $X^* = 2.0$

$i$	0	1	2	3	4
$x_i$	0.0	1.0	2.0	3.0	4.0
$y_i$	0.0	0.86603	1.0	0.0	-2.0

Рис. 6: Условия

### 11 Результаты работы

```
First derivative in X: -0.23206
Second derivative in X: -0.73206
```

Рис. 7: Вывод программы в консоли

## 12 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 |
4 | double derivative(std::vector<double> x, std::vector<double> y, double X) {
5 |     int n = 0;
6 |     double h = x[1] - x[0];
7 |     for (int i = 0; i < x.size() - 1; ++i)
8 |         if (X > x[i] && X < x[i + 1]) {
9 |             if (i == 0) n = i;
10 |             else if (i + 2 > x.size() - 1)
11 |                 n = i - (i + 2 - (x.size() - 1));
12 |             else n = i - 1;
13 |         }
14 |     double a1 = (y[n + 1] - y[n]) / h;
15 |     double a2 = ((y[n + 2] - 2 * y[n + 1] + y[n]) / (2 * h * h)) * (2 * X - x[n] - x[n
16 |         + 1]);
17 |     return a1 + a2;
18 | }
19 |
20 | double derivative2(std::vector<double> x, std::vector<double> y, double X) {
21 |     int n = 0;
22 |     double h = x[1] - x[0];
23 |     for (int i = 0; i < x.size() - 1; ++i)
24 |         if (X > x[i] && X < x[i + 1]) {
25 |             if (i == 0) n = i;
26 |             else if (i + 2 > x.size() - 1)
27 |                 n = i - (i + 2 - (x.size() - 1));
28 |             else n = i - 1;
29 |         }
30 |     return (y[n + 2] - 2 * y[n + 1] + y[n]) / (h * h);
31 | }
32 |
33 | int main() {
34 |     std::vector<double> x = { 0.0, 1.0, 2.0, 3.0, 4.0 };
35 |     std::vector<double> y = { 0.0, 0.86603, 1.0, 0.0, -2.0 };
36 |     double X = 2.0;
37 |     std::cout << "First derivative in X: " << derivative(x, y, X) << "\n";
38 |     std::cout << "Second derivative in X: " << derivative2(x, y, X) << "\n";
39 |
40 |     return 0;
41 | }
```

## 3.5

### 13 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: 26**

$y = x^2 \sqrt{36 - x^2}$   $X_0 = 1, X_k = 5, h_1 = 1.0, h_2 = 0.5$

### 14 Результаты работы

```
h = 1
Integral rectangle: 106.86
Integral trapeze: 108.128
Integral simpson: 47.7634
h = 0.5
Integral rectangle: 145.435
Integral trapeze: 145.474
Integral simpson: 107.288
Runge Rombert:
Rombert rectangle: 184.009
Rombert trapeze: 157.922
Rombert simpson: 111.256
```

Рис. 8: Вывод программы в консоли



## 15 Исходный код

```
1 | #include <iostream>
2 | #include <cmath>
3 |
4 | double f(double x) {
5 |     return pow(x, 2) * pow(36 - pow(x, 2), 0.5);
6 | }
7 |
8 | double rectangle_method(double a, double b, double h) {
9 |     double res = 0;
10 |    while (a + h < b) {
11 |        res += f(a + h / 2);
12 |        a += h;
13 |    }
14 |    return res * h;
15 | }
16 |
17 | double trapez_method(double a, double b, double h) {
18 |     double res = 0;
19 |     while (a + h < b) {
20 |         res += (f(a + h) + f(a));
21 |         a += h;
22 |     }
23 |     return res * h * 0.5;
24 | }
25 |
26 | double simpson_method(double a, double b, double h) {
27 |     double res = 0;
28 |     a += h;
29 |     while (a + h < b) {
30 |         res += f(a - h) + 4 * f(a - h / 2) + f(a);
31 |         a += h;
32 |     }
33 |     return res * h / 6;
34 | }
35 |
36 | double rungeRombert(double h1, double h2, double i1, double i2, double p) {
37 |     return i1 + (i1 - i2) / (pow((h2 / h1), p) - 1);
38 | }
39 |
40 | int main() {
41 |     double a = 1;
42 |     double b = 5;
43 |     double h1 = 1.0;
44 |     double h2 = 0.5;
45 |     std::cout << "h = " << h1 << "\n";
46 |     std::cout << "Integral rectangle: " << rectangle_method(a, b, h1) << "\n";
47 |     std::cout << "Integral trapeze: " << trapez_method(a, b, h1) << "\n";
```

```

48 | std::cout << "Integral simpson: " << simpson_method(a, b, h1) << "\n";
49 | std::cout << "h = " << h2 << "\n";
50 | std::cout << "Integral rectangle: " << rectangle_method(a, b, h2) << "\n";
51 | std::cout << "Integral trapeze: " << trapez_method(a, b, h2) << "\n";
52 | std::cout << "Integral simpson: " << simpson_method(a, b, h2) << "\n";
53 | std::cout << "Runge Rombert:\n";
54 | std::cout << "Rombert rectangle: " << rungeRombert(h1, h2, rectangle_method(a, b,
    | h1), rectangle_method(a, b, h2), 1) << "\n";
55 | std::cout << "Rombert trapeze: " << rungeRombert(h1, h2, trapez_method(a, b, h1),
    | trapez_method(a, b, h2), 2) << "\n";
56 | std::cout << "Rombert simpson: " << rungeRombert(h1, h2, simpson_method(a, b, h1),
    | simpson_method(a, b, h2), 4) << "\n";
57 |
58 | return 0;
59 | }

```