

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

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

**Кафедра 806 «Вычислительная математика и
программирование»**

Лабораторные работы по курсу «Численные методы»

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

Москва, 2024

1 Методы приближения функций. Численное дифференцирование и интегрирование

1 Постановка задачи

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

Вариант: 19

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

а)

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

б)

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

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

2 Результаты работы

```
1 Lagrange method
2 for x1
3 L(x)
4 [-0.81 -0.20 0.40 1.02 ]
5 y(x)
6 [-0.81 -0.20 0.40 1.02 ]
7 delta(x)
8 [0.00 0.00 0.00 0.00 ]
9
10
11 for x2
12 L(x)
13 [-0.81 -0.20 0.40 1.02 ]
14 y(x)
15 [-0.81 0.00 0.40 1.02 ]
16 delta(x)
17 [0.00 0.00 0.00 0.00 ]
18
19
20 Newton method
21 for x1
22 P(x)
23 [-0.81 -0.20 0.40 1.02 ]
24 y(x)
25 [-0.81 -0.20 0.40 1.02 ]
26 delta(x)
27 [0.00 0.00 0.00 0.00 ]
28
29
30 for x2
31 P(x)
32 [-0.81 0.00 0.40 1.02 ]
33 y(x)
34 [-0.81 0.00 0.40 1.02 ]
35 delta(x)
36 [0.00 0.00 0.00 0.00 ]
```

Рис. 1: Вывод в консоли

3 Исходный код

Lab3.1.cpp

```
1 | #include <iostream>
2 | #include <fstream>
3 | #include <cmath>
4 | #include <vector>
5 |
6 | using namespace std;
7 |
```

```

8 double func(double x1, double x2){
9     return ((asin(x1) + x1) - (asin(x2) + x2)) / (x1 - x2);
10 }
11
12 double func1(double x1, double x2, double x3){
13     return (func(x1,x2) - func(x2, x3)) / (x1 - x3);
14 }
15
16 double func2(double x1, double x2, double x3, double x4){
17     return (func1(x1, x2, x3) - func1(x2, x3, x4)) / (x1 - x4);
18 }
19
20 vector <double> omega_values(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_method(vector <double> x, double X, vector <double> &L, vector <double>
    &y, vector <double> &delta){
30     vector <vector <double>> table(5, vector<double>(4,0));
31     vector <double> omega = omega_values(x);
32     for (int i = 0; i < x.size(); i++){
33         table[0][i] = x[i];
34         table[1][i] = asin(x[i]) + x[i];
35         table[2][i] = omega[i];
36         table[3][i] = (asin(x[i]) + x[i]) / omega[i];
37         table[4][i] = X - x[i];
38     }
39
40     for (int i = 0; i < x.size(); i++){
41         L[i] = (table[3][0] * (x[i] - table[0][1]) * (x[i] - table[0][2]) * (x[i] -
            table[0][3]) \
42             + table[3][1] * (x[i] - table[0][0]) * (x[i] - table[0][2]) * (x[i] - table
                [0][3]) \
43             + table[3][2] * (x[i] - table[0][0]) * (x[i] - table[0][1]) * (x[i] - table
                [0][3]) \
44             + table[3][3] * (x[i] - table[0][0]) * (x[i] - table[0][1]) * (x[i] - table
                [0][2]));
45     }
46
47     for (int i = 0; i < x.size(); i++){
48         y[i] = asin(x[i]) + x[i];
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_method(vector <double> x, double X, vector <double> &P, vector <double> &y
    , vector <double> &delta){
58     vector <vector <double>> table(5, vector<double>(4,0));
59     for (int i = 0; i < x.size(); i++){
60         table[0][i] = x[i];
61         table[1][i] = asin(x[i]) + x[i];
62         if (i < 3){
63             table[2][i] = func(x[i], x[i+1]);
64         }
65         if (i < 2){
66             table[3][i] = func1(x[i], x[i+1], x[i+2]);
67         }
68     }
69     table[4][0] = func2(x[0], x[1], x[2], x[3]);
70
71
72     for (int i = 0; i < x.size(); i++){
73         P[i] = (table[1][0] + table[2][0] * (x[i] - table[0][0]) + table[3][0] * (x[i]
            - table[0][0]) * (x[i] - table[0][1]) \
74             + table[4][0] * (x[i] - table[0][0]) * (x[i] - table[0][1]) * (x[i] - table
                [0][2]));
75     }
76
77     for (int i = 0; i < x.size(); i++){
78         y[i] = asin(x[i]) + x[i];
79     }
80
81     for (int i = 0; i < x.size(); i++){
82         delta[i] = fabs(y[i] - P[i]);
83     }
84 }
85
86 int main(){
87     ofstream fout("answer1.txt");
88     fout.precision(2);
89     fout << fixed;
90     int n = 4;
91     vector <double> x1 = {-0.4, -0.1, 0.2, 0.5};
92     vector <double> x2 = {-0.4, 0, 0.2, 0.5};
93     double root = 0.1;
94     vector <double> L(n,0), y(n,0), delta(n,0), P(n,0);
95     fout << "Lagrange method" << endl;
96     Lagrange_method(x1, root, L, y, delta);
97     fout << "for x1" << endl << "L(x)\n" << "[";

```

```

98     for (int i = 0; i < L.size(); i++) fout << L[i] << "\t";
99     fout << "]\n" << "y(x)\n" << "[";
100    for (int i = 0; i < y.size(); i++) fout << y[i] << "\t";
101    fout << "]\n" << "delta(x)\n" << "[";
102    for (int i = 0; i < delta.size(); i++) fout << delta[i] << "\t";
103    fout << "]\n";
104    Lagrange_method(x2, root, P, y, delta);
105    fout << "\n\nfor x2" << endl << "L(x)\n" << "[";
106    for (int i = 0; i < L.size(); i++) fout << L[i] << "\t";
107    fout << "]\n" << "y(x)\n" << "[";
108    for (int i = 0; i < y.size(); i++) fout << y[i] << "\t";
109    fout << "]\n" << "delta(x)\n" << "[";
110    for (int i = 0; i < delta.size(); i++) fout << delta[i] << "\t";
111    fout << "]\n";
112    fout << "\n\nNewton method" << endl;
113    Newton_method(x1, root, P, y, delta);
114    fout << "for x1" << endl << "P(x)\n" << "[";
115    for (int i = 0; i < P.size(); i++) fout << P[i] << "\t";
116    fout << "]\n" << "y(x)\n" << "[";
117    for (int i = 0; i < y.size(); i++) fout << y[i] << "\t";
118    fout << "]\n" << "delta(x)\n" << "[";
119    for (int i = 0; i < delta.size(); i++) fout << delta[i] << "\t";
120    fout << "]\n";
121    Newton_method(x2, root, P, y, delta);
122    fout << "\n\nfor x2" << endl << "P(x)\n" << "[";
123    for (int i = 0; i < P.size(); i++) fout << P[i] << "\t";
124    fout << "]\n" << "y(x)\n" << "[";
125    for (int i = 0; i < y.size(); i++) fout << y[i] << "\t";
126    fout << "]\n" << "delta(x)\n" << "[";
127    for (int i = 0; i < delta.size(); i++) fout << delta[i] << "\t";
128    fout << "]\n";
129 }

```

4 Постановка задачи

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

Вариант: 19

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

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

5 Результаты работы

```
1 X = 0.1 in range: -0.1 - 0.2
2 Function value in this range: -0.20017 - 0.40136
3 Function value in X with cubix spline: -0.76488
```

Рис. 2: Вывод в консоли

6 Исходный код

Lab3.2.cpp

```
1 #include <iostream>
2 #include <vector>
3 #include <fstream>
4 #include <cmath>
5
6 using namespace std;
7
8 vector<double> solve_linear_system(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
30     vector<double> x(n);
31     for (int i = n - 1; i >= 0; i--) {
32         x[i] = b[i];
33         for (int j = i + 1; j < n; j++) {
34             x[i] -= A[i][j] * x[j];
35         }
36         x[i] /= A[i][i];
37     }
38
39     return x;
40 }
41
42
43 double Cubic_spline(vector<double>& x, vector<double>& y, double X) {
44     vector<double> A(3, 0);
45     vector<vector<double>> B(3, vector<double>(3, 0));
46     vector<vector<double>> results(4, vector<double>(4, 0));
47     vector<double> roots(3, 0);
48     double f = 0;
49
50     for (int i = 2; i < 5; i++) {
51         A[i - 2] = (3 * ((y[i] - y[i - 1]) / (x[i] - x[i - 1]) +
52             (y[i - 1] - y[i - 2]) / (x[i - 1] - x[i - 2]))));
53         for (int j = 0; j < B.size(); j++) {
54             for (int k = 0; k < B[0].size(); k++) {
55                 if (j == k) {
56                     B[j][k] = 2 * ((x[i - 1] - x[i - 2]) + (x[i] - x[i - 1]));
57                 } else if (j < k && (k != B.size() - 1 || j != 0)) {
58                     B[j][k] = (x[i] - x[i - 1]);
59                 } else if (j > k && (j != B.size() - 1 || k != 0)) {
60                     B[j][k] = (x[i - 1] - x[i - 2]);
61                 }
62             }
63         }
64     }
65
66     roots = solve_linear_system(B,A);
67     roots.insert(roots.begin(), 0);
68
69     for (int i = 0; i < results.size(); i++) {
70         for (int j = 0; j < results[0].size(); j++) {
71             if (i != results.size() - 1) {
72                 if (j == 0) {
73                     results[i][j] = y[i];
74                 }
75                 if (j == 1) {
76                     results[i][j] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]) - (

```

```

77         (x[i + 1] - x[i]) * roots[i + 1] + 2 * roots[i]) / 3;
78     }
79     if (j == 2) {
80         if (i == 0) {
81             results[i][j] = 0;
82         } else {
83             results[i][j] = roots[i];
84         }
85     }
86     if (j == 3) {
87         results[i][j] = (roots[i + 1] - roots[i]) / (3 * (x[i + 1] - x[i]));
88     }
89 } else {
90     if (j == 0) {
91         results[i][j] = y[i];
92     }
93     if (j == 1) {
94         results[i][j] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]) - (2 * (x[i +
95             1] - x[i]) * roots[i]) / 3;
96     }
97     if (j == 2) {
98         results[i][j] = roots[i];
99     }
100    if (j == 3) {
101        results[i][j] = -roots[i] / (3 * (x[i + 1] - x[i]));
102    }
103 }
104 }
105
106 f = results[1][0] + results[1][1] * (X - x[1]) + results[1][2] * pow((X - x[2]), 2)
    + results[1][3] * pow((X - x[3]), 3);
107
108 return f;
109 }
110
111
112 int main() {
113     ofstream fout("answer2.txt");
114     vector<double> x = {-0.4, -0.1, 0.2, 0.5, 0.8};
115     vector<double> y = {-0.81152, -0.20017, 0.40136, 1.0236, 1.7273};
116     double X = 0.1;
117     fout << "X = " << X << " in range: " << x[1] << " - " << x[2] << endl;
118     fout << "Function value in this range: " << y[1] << " - " << y[2] << endl;
119     fout << "Function value in X with cubix spline: " << Cubic_spline(x, y, X) << endl;
120
121     return 0;
122 }

```

7 Постановка задачи

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

Вариант: 19

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

8 Результаты работы

```
1 First-degree polynomial
2 -1.46917 -0.837155 -0.205144 0.426867 1.05888 1.69089
3
4 Second-degree polynomial
5 -1.45472 -0.840044 -0.216699 0.415312 1.05599 1.70533
6
7 Sum of squares of errors for the first-degree polynomial: 0.00394166
8 Sum of squares of errors for the second-degree polynomial: 0.00324066
```

Рис. 3: Вывод в консоли

9 Исходный код

Lab3.3.cpp

```
1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 #include <fstream>
5
6 using namespace std;
7
8 vector<double> solve_linear_system(vector<vector<double>>& A, vector<double>& b) {
9     int n = A.size();
10
11
12     for (int i = 0; i < n; i++) {
13         int max_row = i;
14         for (int k = i + 1; k < n; k++) {
15             if (abs(A[k][i]) > abs(A[max_row][i])) {
16                 max_row = k;
17             }
18         }
19         swap(A[i], A[max_row]);
20         swap(b[i], b[max_row]);
21         for (int k = i + 1; k < n; k++) {
22             double factor = A[k][i] / A[i][i];
23             for (int j = i; j < n; j++) {
24                 A[k][j] -= factor * A[i][j];
25             }
26             b[k] -= factor * b[i];
27         }
28     }
29 }
```

```

30     vector<double> x(n);
31     for (int i = n - 1; i >= 0; i--) {
32         x[i] = b[i];
33         for (int j = i + 1; j < n; j++) {
34             x[i] -= A[i][j] * x[j];
35         }
36         x[i] /= A[i][i];
37     }
38
39     return x;
40 }
41
42 pair<vector<double>, vector<double>> method_least_squares(vector<double> x, vector<
43     double> y) {
44     vector <vector <double>> A(2, vector<double>(3,0));
45     double sumx = 0, sumy = 0, sumx2 = 0, sumxy = 0;
46     for (int i = 0; i < x.size(); ++i) {
47         sumx += x[i];
48         sumy += y[i];
49         sumx2 += pow(x[i], 2);
50         sumxy += x[i] * y[i];
51     }
52     A[0][0] = x.size();
53     A[0][1] = sumx;
54     A[0][2] = sumy;
55     A[1][0] = sumx;
56     A[1][1] = sumx2;
57     A[1][2] = sumxy;
58
59     vector <vector <double>> first_slice_A = {{A[0][0], A[0][1]}, {A[1][0], A[1][1]}};
60     vector <double> second_slice_A = {A[0][2], A[1][2]};
61
62     vector <double> roots1 = solve_linear_system(first_slice_A, second_slice_A);
63
64     vector <double> y1 (x.size(), 0);
65     for (int i = 0; i < y1.size(); i++){
66         y1[i] = roots1[0] + x[i] * roots1[1];
67     }
68
69     vector <vector <double>> A2(3, vector<double>(4,0));
70     sumx = 0, sumy = 0, sumx2 = 0, sumxy = 0;
71     double sumx3 = 0, sumx4 = 0, sumx2y = 0;
72     for (int i = 0; i < x.size(); ++i) {
73         sumx += x[i];
74         sumy += y[i];
75         sumx2 += pow(x[i],2);
76         sumx3 += pow(x[i],3);
77         sumx4 += pow(x[i],4);
78         sumxy += x[i] * y[i];
79         sumx2y += pow(x[i],2) * y[i];

```

```

78     }
79     A2[0][0] = x.size();
80     A2[0][1] = sumx;
81     A2[0][2] = sumx2;
82     A2[0][3] = sumy;
83     A2[1][0] = sumx;
84     A2[1][1] = sumx2;
85     A2[1][2] = sumx3;
86     A2[1][3] = sumxy;
87     A2[2][0] = sumx2;
88     A2[2][1] = sumx3;
89     A2[2][2] = sumx4;
90     A2[2][3] = sumx2y;
91
92     vector <vector <double>> first_slice_A2 = {{A2[0][0], A2[0][1], A2[0][2]}, {A2
        [1][0], A2[1][1], A2[1][2]}, {A2[2][0], A2[2][1], A2[2][2]}};
93     vector <double> second_slice_A2 = {A2[0][3], A2[1][3], A2[2][3]};
94
95     vector <double> roots2 = solve_linear_system(first_slice_A2, second_slice_A2);
96
97     vector <double> y2(x.size(), 0);
98
99     for (int i = 0; i < y2.size(); i++){
100         y2[i] = roots2[0] + x[i] * roots2[1] + pow(x[i],2) * roots2[2];
101     }
102
103     return make_pair(y1, y2);
104 }
105
106 int main() {
107     ofstream fout("answer3.txt");
108     vector<double> x = {-0.7, -0.4, -0.1, 0.2, 0.5, 0.8};
109     vector<double> y = {-1.4754, -0.81152, -0.20017, 0.40136, 1.0236, 1.7273};
110
111     vector <double> y1 = method_least_squares(x,y).first;
112     vector <double> y2 = method_least_squares(x,y).second;
113
114     double F1 = 0.0, F2 = 0.0;
115     for (int i = 0; i < y.size(); ++i) {
116         F1 += pow((y[i] - y1[i]), 2);
117         F2 += pow((y[i] - y2[i]), 2);
118     }
119
120     fout << "First-degree polynomial" << endl;
121     for (int i = 0; i < y1.size(); i++){
122         fout << y1[i] << " ";
123     }
124
125     fout << endl << endl;

```

```

126 |     fout << "Second-degree polynomial" << endl;
127 |     for (int i = 0; i < y2.size(); i++){
128 |         fout << y2[i] << " ";
129 |     }
130 |     fout << endl << endl;
131 |     fout << "Sum of squares of errors for the first-degree polynomial: " << F1 << endl;
132 |     fout << "Sum of squares of errors for the second-degree polynomial: " << F2 << endl;
133 |     ;
134 |     return 0;
135 | }

```

10 Постановка задачи

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

Вариант: 19

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

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

11 Результаты работы

```
First diff
left diff:  1.7854
right diff: 1.3217
diff with second order precision:  1.55355

Second diff
-0.4637
```

Рис. 4: Вывод в консоли

12 Исходный код

Lab3.4.cpp

```
1  #include <iostream>
2  #include <vector>
3  #include <fstream>
4  #include <cmath>
5
6  using namespace std;
7
8  vector <double> first_diff(vector<double> x, vector <double> y, double root_index){
9      int n = x.size();
10     double left_diff, right_diff, diff;
11     if (root_index != 1 && root_index != (n - 2)){
12         left_diff = (y[root_index] - y[root_index - 1]) / (x[root_index] - x[root_index
13         - 1]);
14         right_diff = (y[root_index + 1] - y[root_index]) / (x[root_index + 1] - x[
15         root_index]);
16
17         diff = (y[root_index] - y[root_index - 1]) / (x[root_index] - x[root_index -
18         1]) \
19         + ((y[root_index + 1] - y[root_index]) / (x[root_index + 1] - x[root_index]) - (y[
20         root_index] - y[root_index - 1]) / (x[root_index] - x[root_index - 1])) \
21         / (x[root_index + 1] - x[root_index - 1]) \
22         * (2 * x[root_index] - x[root_index] - x[root_index - 1]);
23     }
24     return vector <double> {left_diff, right_diff, diff};
25 }
26
27 cout << "Error!" << endl;
28 return vector <double> (0,0);
```

```

23 }
24
25 double second_diff(vector <double> x, vector <double> y, double root_index){
26     int n = x.size();
27     if (root_index != 1 && root_index != (n - 2)){
28         double dx2 = 2 * ((y[root_index + 1] - y[root_index])/(x[root_index+1] - x[
                root_index]) - (y[root_index] - y[root_index - 1])/(x[root_index] - x[
                root_index-1])) \
29         / (x[root_index + 1] - x[root_index - 1]) ;
30         return dx2;
31     }
32     cout << "Error!" << endl;
33     return 0.0;
34
35 }
36
37
38 int main(){
39     ofstream fout("answer4.txt");
40     // fout.precision(2);
41     // fout << fixed;
42     int n = 5;
43     vector <double> x = {-1, 0, 1, 2, 3};
44     vector <double> y = {-1.7854, 0.0, 1.7854, 3.1071, 4.249};
45     double X = 2; // 1- x
46
47     vector <double> first_diffs = first_diff(x,y,X);
48     double dx2 = second_diff(x,y,X);
49     fout << "First diff" << endl;
50     fout << "left diff:\t" << first_diffs[0] << endl;
51     fout << "right diff:\t" << first_diffs[1] << endl;
52     fout << "diff with second order precision:\t" << first_diffs[2] << endl << endl;
53     fout << "Second diff" << endl;
54     fout << dx2 << endl;
55 }

```

13 Постановка задачи

3.5. Вычислить определенный интеграл

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

, , ,

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

Вариант: 19

$$y = x^2, 625 - x^4; X_0 = 0, X_k = 4, h_1 = 1.0, h_2 = 0.5 \quad (7)$$

14 Результаты работы

```
Rectangle with h = 1
0.040489
Rectangle with h = 0.5
0.0418683
Trapez with h = 1
0.046395
Trapez with h = 0.5
0.043442
Simpson with h = 1
0.0424577
Simpson with h = 0.5
0.0423929
with Runge-Romberg-Richardson method
Rectangle with h = 1
0.040489
погрешность:0.00189816
Rectangle with h = 0.5
0.0418683
погрешность:0.000518844
Trapez with h = 1
0.046395
погрешность:0.00400791
Trapez with h = 0.5
0.043442
погрешность:0.00105487
Simpson with h = 1
0.0424577
погрешность:7.05285e-05
Simpson with h = 0.5
0.0423929
погрешность:5.72845e-06
```

Рис. 5: Вывод в консоли

15 Исходный код

Lab3.5.cpp

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <fstream>
```

```

4  #include <cmath>
5
6  using namespace std;
7
8  double func(double x){
9      return (x * x) / (625 - x * x * x * x);
10 }
11
12
13 double rectangle_method(double x0, double xk, double h){
14     double F = 0;
15     double n = (int) ((xk - x0) / h);
16     n += 1;
17     vector <double> x_values(n, 0);
18     for (int i = 0; i < n; i++){
19         x_values[i] = x0 + h*i;
20     }
21     for (int i = 1; i < n; i++){
22         F += h * func((x_values[i] + x_values[i-1])/2);
23     }
24     return F;
25 }
26
27 double trapez_method(double x0, double xk, double h){
28     double F = 0;
29     double n = (int) ((xk - x0) / h);
30     n += 1;
31     vector <double> x_values(n, 0);
32     for (int i = 0; i < n; i++){
33         x_values[i] = x0 + h*i;
34     }
35     for (int i = 1; i < n; i++){
36         F += (func(x_values[i]) + func(x_values[i-1])) / 2 * h;
37     }
38     return F;
39 }
40
41 double simps_method(double x0, double xk, double h){
42     int n = (int)((xk - x0) / h);
43     double F = 0;
44     for (int i = 0; i < n; i++) {
45         double x1 = x0 + i * h;
46         double x2 = x0 + (i + 1) * h;
47         double x3 = x0 + (i + 0.5) * h;
48         F += (h / 6) * (func(x1) + 4 * func(x3) + func(x2));
49     }
50     return F;
51 }
52

```

```

53 vector <double> runge_romb_rich(double x0, double xk, double h){
54     double F = 0;
55     vector <double> results(3,0);
56     results[0] = rectangle_method(x0, xk, h) + (rectangle_method(x0, xk, h/2) -
57         rectangle_method(x0, xk, h/2))/(1-0.5*0.5);
58     results[1] = trapez_method(x0, xk, h) + (trapez_method(x0, xk, h/2) - trapez_method
59         (x0, xk, h/2))/(1-0.5*0.5);
60     results[2] = simps_method(x0, xk, h) + (simps_method(x0, xk, h/2) - simps_method(x0
61         , xk, h/2))/(1-0.5*0.5*0.5*0.5);
62     return results;
63 }
64
65 int main(){
66     ofstream fout("answer5.txt");
67     // fout.precision(2);
68     // fout << fixed;
69     int n = 5;
70     double x0 = 0, xk = 4, h1 = 1.0, h2 = 0.5;
71     double integral = 0.042387134;
72     fout << "Rectangle with h = 1\n";
73     fout << rectangle_method(x0, xk, h1);
74     fout << "\nRectangle with h = 0.5\n";
75     fout << rectangle_method(x0, xk, h2);
76     fout << "\n\nTrapez with h = 1\n";
77     fout << trapez_method(x0,xk,h1);
78     fout << "\nTrapez with h = 0.5\n";
79     fout << trapez_method(x0,xk,h2);
80     fout << "\n\nSimpson with h = 1\n";
81     fout << simps_method(x0,xk,h1);
82     fout << "\nSimpson with h = 0.5\n";
83     fout << simps_method(x0,xk,h2) << endl << endl;
84     vector <double> RRR = runge_romb_rich(x0, xk, h1);
85     vector <double> RRR2 = runge_romb_rich(x0,xk,h2);
86     fout << "with Runge-Romberg-Richardson method" << endl;
87     fout << "Rectangle with h = 1\n";
88     fout << RRR[0];
89     fout << "\n:" << fabs(integral - RRR[0]);
90     fout << "\nRectangle with h = 0.5\n";
91     fout << RRR2[0];
92     fout << "\n:" << fabs(integral - RRR2[0]);
93     fout << "\n\nTrapez with h = 1\n";
94     fout << RRR[1];
95     fout << "\n:" << fabs(integral - RRR[1]);
96     fout << "\nTrapez with h = 0.5\n";
97     fout << RRR2[1];
98     fout << "\n:" << fabs(integral - RRR2[1]);
99     fout << "\n\nSimpson with h = 1\n";
100    fout << RRR[2];

```

```

99  || fout << "\n:" << fabs(integral - RRR[2]);
100 || fout << "\nSimpson with h = 0.5\n";
101 || fout << RRR2[2];
102 || fout << "\n:" << fabs(integral - RRR2[2]);
103 || }

```