

**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: Ерофеева Е.С.
Преподаватель: Пивоваров Д.Е.
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: 6

$y = \exp x$

a) $X_i = -2, -1, 0, 1;$

b) $X_i = -2, -1, 0.2, 1;$

$X^* = -0.5$

2 Результаты работы

```
Многочлен Лагранжа
a) L(x) = - 0.0226(x-2)(x-1)(x+0)(x+1) + 0.1839(x-2)(x-1)(x+0)(x+1) - 0.5(x-2)(x-1)(x+0)(x+1) + 0.453(x-2)(x-1)(x+0)(x+1)
L(x*)=0.591081
y(x*)=0.606531
Погрешность: 0.0154495
b) L(x) = - 0.0205(x-2)(x-1)(x+0.2)(x+1) + 0.1533(x-2)(x-1)(x+0.2)(x+1) - 0.5783(x-2)(x-1)(x+0.2)(x+1) + 0.5663(x-2)(x-1)(x+0.2)(x+1)
L(x*)=0.583949
y(x*)=0.606531
Погрешность: 0.022582

Многочлен Ньютона
a) N(x) = - 0.1353 + 0.2325(x-2) + 0.1998(x-2)(x-1) + 0.1144(x-2)(x-1)(x+0)
N(x*)=0.591081
y(x*)=0.606531
Погрешность: 0.0154495
b) N(x) = - 0.1353 + 0.2325(x-2) + 0.2176(x-2)(x-1) + 0.1208(x-2)(x-1)(x+0.2)
N(x*)=0.583949
y(x*)=0.606531
Погрешность: 0.022582
```

Рис. 1: Вывод программы в консоли

3 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4
5
6  double func(double x){
7      return exp(x);
8  }
9
10 double lagrange(const std::vector<double> &x, double xu, int n){
11     std::cout << "L(x) = ";
12     double res = 0;
13     for(int j = 0; j < n; j++) {
14         double a = func(x[j]);
15         double f = 1;
16         for(int i = 0; i < n; i++) {
17             if(i != j) a/=x[j] - x[i];
18         }
19         if(j && a >= 0) {
20             std::cout << " + ";
21         } else {
22             std::cout << " - ";
23         }
24         std::cout << round(fabs(a)*10000)/10000.0;
25         for(int i = 0; i < n; i++) {
26             std::cout << "(x";
27             if(x[i] >= 0) std::cout << "+";
28             std::cout << x[i] << ")";
29             if (i != j) f *= xu - x[i];
30         }
31         res += a*f;
32     }
33     std::cout << std::endl;
34     return res;
35 }
36
37
38 double Newton(const std::vector<double> &x, double xu, int n){
39     std::vector<std::vector<double>> d(n, std::vector<double>(n));
40     for(int i = 0; i < n; i++){
41         d[i][0] = func(x[i]);
42     }
43     for(int j = 1 ; j < n; j++){
44         for(int i = 0; i < n-j; i++){
45             d[i][j] = (d[i][j-1]-d[i+1][j-1])/float(x[i]-x[i+j]);
46         }
47     }
```

```

48     std::cout << "N(x) = ";
49     double res = 0;
50     for(int j = 0; j < n; j++){
51         double a = d[0][j];
52         double f = 1;
53         if(j && a >= 0) {
54             std::cout << " + ";
55         } else {
56             std::cout << " - ";
57         }
58         std::cout << round(fabs(a)*10000)/10000.0;
59         for(int i = 0; i < j; i++) {
60             std::cout << "(x";
61             if(x[i] >= 0) std::cout << "+";
62             std::cout << x[i] << ")";
63             f *= xu - x[i];
64         }
65         res += a*f;
66     }
67     std::cout << std::endl;
68     return res;
69 }
70
71
72 int main(){
73     const int n = 4;
74     const double xu = -0.5;
75     std::vector<double> Xa = {-2, -1, 0, 1};
76     std::vector<double> Xb = {-2, -1, 0.2, 1};
77
78     std::cout << " " << std::endl;
79     std::cout << "a) ";
80     double res = lagrange(Xa, xu, n);
81     std::cout << "L(x*)=" << res << std::endl;
82     std::cout << "y(x*)=" << func(xu) << std::endl;
83     std::cout << ": " << fabs(res - func(xu)) << std::endl;
84
85     std::cout << "b) ";
86     res = lagrange(Xb, xu, n);
87     std::cout << "L(x*)=" << res << std::endl;
88     std::cout << "y(x*)=" << func(xu) << std::endl;
89     std::cout << ": " << fabs(res - func(xu)) << std::endl;
90     std::cout << std::endl;
91
92
93     std::cout << " " << std::endl;
94     std::cout << "a) ";
95     res = Newton(Xa, xu, n);
96     std::cout << "N(x*)=" << res << std::endl;

```

```

97 | std::cout << "y(x*)=" << func(xu) << std::endl;
98 | std::cout << ": " << fabs(res - func(xu)) << std::endl;
99 |
100 | std::cout << "b) ";
101 | res = Newton(Xb, xu, n);
102 | std::cout << "N(x*)=" << res << std::endl;
103 | std::cout << "y(x*)=" << func(xu) << std::endl;
104 | std::cout << ": " << fabs(res - func(xu)) << std::endl;
105 | std::cout << std::endl;
106 |
107 |
108 | return 0;
109 | }

```

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: 6

6. $X^* = -0.5$

i	0	1	2	3	4
x_i	-2.0	-1.0	0.0	1.0	2.0
f_i	0.13534	0.36788	1.0	2.7183	7.3891

Рис. 2: Условия

5 Результаты работы

```
x ∈ [-2, -1], =>
    s(x) = 0.13534 + 0.150371(x + 2) - 0(x + 2)^2 + 0.0821693(x + 2)^3

x ∈ [-1, 0], =>
    s(x) = 0.36788 + 0.396879(x + 1) + 0.246508(x + 1)^2 + 0.0112664(x + 1)^3

x ∈ [0, 1], =>
    s(x) = 1 + 0.856095(x - 0) + 0.212709(x - 0)^2 + 0.649496(x - 0)^3

x ∈ [1, 2], =>
    s(x) = 2.7183 - 0(x + 1) + 2.1612(x + 1)^2 - 0(x + 1)^3

s(x*) = 0.626538
```

Рис. 3: Вывод программы в консоли

6 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4
5  using InVector = const std::vector<double> &;
6
7
8  std::vector<double> threeDiagonal(InVector x, InVector y, int n) {
9      std::vector<double> p(n-2);
10     std::vector<double> q(n-2);
11     p[0] = -(x[2]-x[1])/(2*x[3]-2*x[1]);
12     q[0] = 3*((y[2]-y[1])/(x[2]-x[1])-(y[1]-y[0])/(x[1]-x[0]))/(2*x[2]-2*x[0]);
13     for(int i = 1; i < n - 2; i++){
14         p[i] = -(x[i+2]-x[i+1])/((2*x[i+2]-2*x[i])+(x[i+1]-x[i])*p[i-1]);
15         q[i] = (3*((y[i+2]-y[i+1])/(x[i+2]-x[i+1])-(y[i+1]-y[i])/(x[i+1]-x[i]))-(x[i+1]-x[i])*q[i-1])/((2*x[i+2]-2*x[i])+(x[i+1]-x[i])*p[i-1]);
16     }
17     std::vector<double> res(n);
18     res[n-1] = q[n-2];
19     for(int i = n-2; i > 0; i--){
20         res[i] = p[i-1]*res[i+1] + q[i-1];
21     }
22     return res;
23 }
24
25
26 std::string sign(double x) {
27     return x? " + ": " - ";
28 }
29
30
31 double spline(InVector x, InVector y, double xu, int n) {
32     std::vector<double> a(n-1);
33     std::vector<double> b(n-1);
34     std::vector<double> c(n-1);
35     std::vector<double> d(n-1);
36     for(int i = 0; i < n-1; i++) a[i] = y[i];
37     c = threeDiagonal(x, y, n);
38     for(int i = 0; i < n-2; i++) {
39         b[i] = (y[i+1]-y[i])/(x[i+1]-x[i])-(x[i+1]-x[i])*(c[i+1]+2*c[i])/3;
40         d[i] = (c[i+1]-c[i])/(x[i+1]-x[i])/3;
41     }
42     b[n-1] = (y[n]-y[n-1])/(x[n]-x[n-1])-(x[n]-x[n-1])*2*c[n-1]/3;
43     d[n-1] = -c[n-1]/(x[n]-x[n-1])/3;
44
45     for(int i = 0; i < n-1; i++){
46         std::cout << "x  [" << x[i] << ", " << x[i+1] << "], =>" << std::endl;
```

```

47     std::cout << "\t s(x) = ";
48     std::cout << a[i];
49     std::cout << sign(b[i]) << fabs(b[i]) << "(x" << sign(-x[i]) << fabs(x[i]) << "
50         )";
51     std::cout << sign(c[i]) << fabs(c[i]) << "(x" << sign(-x[i]) << fabs(x[i]) << "
52         )^2";
53     std::cout << sign(d[i]) << fabs(d[i]) << "(x" << sign(-x[i]) << fabs(x[i]) << "
54         )^3";
55     std::cout << std::endl << std::endl;
56 }
57
58 int j = 0;
59 for(int i = 0; i < n-1; i++) {
60     if(xu >= x[i] && xu <= x[i+1]) j = i;
61 }
62 if(!j) return xu;
63 xu -= x[j];
64 return a[j]+b[j]*xu+c[j]*xu*xu + d[j]*xu*xu*xu;
65 }
66
67 int main(){
68     const int n = 5;
69     const double xu = -0.5;
70     std::vector<double> x = { -2, -1, 0, 1, 2 };
71     std::vector<double> y = { 0.13534, 0.36788, 1, 2.7183, 7.3891 };
72     double Sxu = spline(x, y, xu, n);
73     std::cout << "s(x*) = " << Sxu << std::endl;
74     return 0;
75 }

```


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: 6

6.

i	0	1	2	3	4	5
x_i	-3.0	-2.0	-1.0	0.0	1.0	2.0
y_i	0.04979	0.13534	0.36788	1.0	2.7183	7.3891

Рис. 4: Условия

8 Графики аппроксимаций

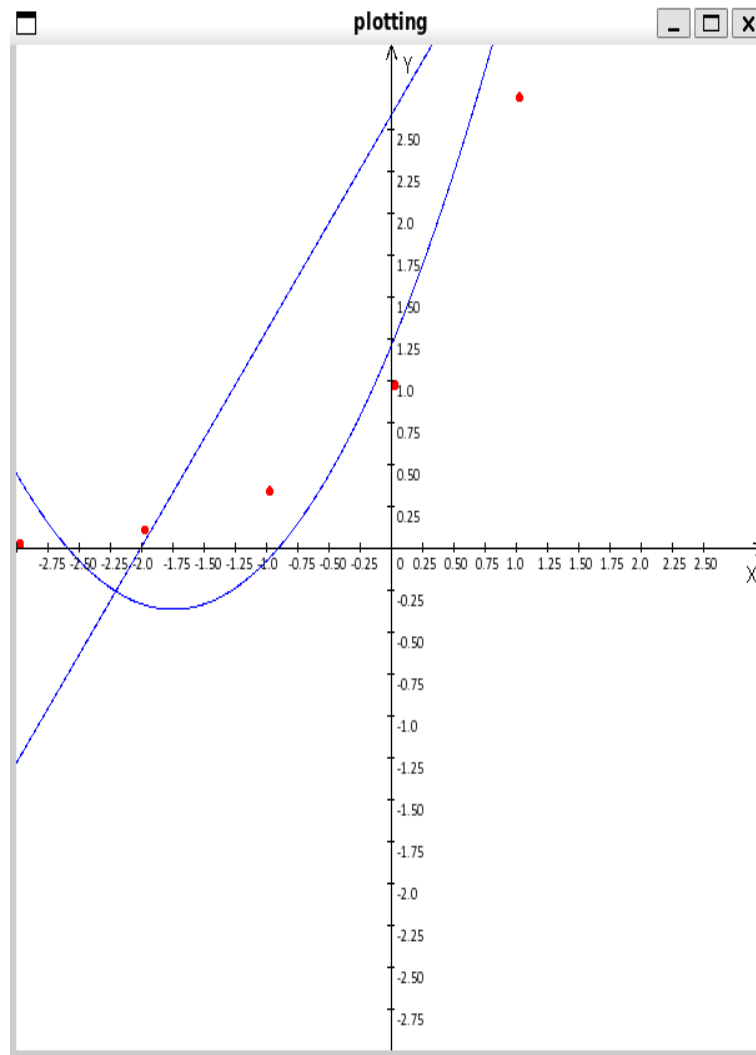


Рис. 5: Вывод программы в консоли

9 Результаты работы

```

МНК, Многочлен 1-ой степени:
P(x) = 2.58737 + 1.28793x
Сумма квадратов ошибок: 11.4549

МНК, Многочлен 2-ой степени:
P(x) = 1.21264 + 1.80345x + 0.515523x^2
Сумма квадратов ошибок: 1.533

```

Рис. 6: Вывод программы в консоли

10 Исходный код

```

1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4
5
6  using InVector = std::vector<double> &;
7  using InMatrix = std::vector<std::vector<double>> &;
8
9
10 double det(InMatrix a){
11     double D = 0;
12     int size = a.size();
13     if(size == 1) return a[0][0];
14     double sign = 1;
15     for(int k = 0; k < size; k++) {
16         std::vector<std::vector<double>> minor(size-1, std::vector<double>(size-1));
17         for(int i = 0; i < size-1; i++) {
18             for(int j = 0; j < size-1; j++) {
19                 int mj = (j < k)? j: j+1;
20                 minor[i][j] = a[i+1][mj];
21             }
22         }
23         double temp = det(minor);
24         D += sign*a[0][k]*temp;
25         sign *= -1;
26     }
27     return D;
28 }
29
30 std::vector<double> Cramer(InMatrix a, InVector b, int m){
31     std::vector<double> x(m);
32     for(int i = 0; i < m; i++) {
33         std::vector<std::vector<double>> ai(a);
34         for(int j = 0; j < m; j++) {

```

```

35         ai[j][i] = b[j];
36     }
37     x[i] = det(ai)/det(a);
38 }
39 return x;
40 }
41
42
43 std::string sign(double x) {
44     return (x < 0)? " - ": " + ";
45 }
46
47
48 double minSquare(InVector x, InVector y, int m, int n){
49     std::vector<std::vector<double>> a(m, std::vector<double>(m));
50     std::vector<double> b(m);
51     for (int i = 0; i < m; i++) {
52         for (int j = 0; j < m; j++) {
53             for(int k = 0; k < n; k++) {
54                 a[i][j] += pow(x[k], i+j);
55             }
56         }
57         for(int k = 0; k < n; k++) {
58             b[i] += y[k]*pow(x[k], i);
59         }
60     }
61     std::vector<double> z(Cramer(a, b, m));
62     double err = 0;
63     for(int k = 0; k < n; k++){
64         double px = 0;
65         for(int i = 0; i < m; i++) {
66             px += z[i]*pow(x[k], i);
67         }
68         err += (px - y[k])*(px - y[k]);
69     }
70     std::cout << "P(x) = " << z[0];
71     for(int i = 1; i < m; i++) {
72         std::cout << sign(z[i]) << fabs(z[i]) << "x";
73         if(i > 1) std::cout << "^" << i;
74     }
75     std::cout << std::endl;
76     return err;
77 }
78
79
80 int main(){
81     const int n = 6;
82     std::vector<double> x = { -3, -2, -1, 0, 1, 2 };
83     std::vector<double> y = { 0.04979, 0.13534, 0.36788, 1, 2.7183, 7.3891 };

```

```

84
85     std::cout << ", 1- :" << std::endl;
86     double res = minSquare(x, y, 2, n);
87     std::cout << " : " << res << std::endl;
88     std::cout << std::endl;
89
90     std::cout << ", 2- :" << std::endl;
91     res = minSquare(x, y, 3, n);
92     std::cout << " : " << res << std::endl;
93     std::cout << std::endl;
94     return 0;
95 }

```

3.4

11 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: 6

6. $X^* = 0.2$

i	0	1	2	3	4
x_i	-0.2	0.0	0.2	0.4	0.6
y_i	-0.20136	0.0	0.20136	0.41152	0.64350

Рис. 7: Условия

12 Результаты работы

```
Производные по безразностным формулам
Левая первая производная: 1.0508
Правая первая производная: 1.0068
Центральная первая производная: 1.0288
Вторая производная: 0.22

Производные от интерполяционных полиномов
f' (x*) = 1.01971
f''(x*) = 0.211208
```

Рис. 8: Вывод программы в консоли

13 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4
5
6  using InVector = std::vector<double> &;
7  using Matrix = std::vector<std::vector<double>>>;
8
9
10 Matrix getPolinom(InVector x, InVector y, int n){
11     Matrix D(n, std::vector<double>(n));
12     for(int i = 0; i < n; i++) {
13         D[i][0] = y[i];
14     }
15     for(int j = 1; j < n; j++) {
16         for(int i = 0; i < n-j; i++) {
17             D[i][j] = (D[i][j-1] - D[i+1][j-1])/(x[i]-x[i+j]);
18         }
19     }
20     return D;
21 }
22
23
24 double firstDerivative(InVector x, Matrix &D, double xu, int n) {
25     double dydx = 0;
26     for(int j = 1; j < n; j++) {
27         double f = 0;
28         for(int k = 0; k < j; k++) {
29             double g = D[0][j];
30             for(int i = 0; i < j; i++) {
31                 if(i!=k) g*=xu-x[i];
32             }
33             f += g;
34         }
35         dydx += f;
36     }
37     return dydx;
38 }
39
40
41 double secondDerivative(InVector x, Matrix &D, double xu, int n) {
42     double ddydxdx = 0;
43     for(int j = 2; j < n; j++) {
44         double f = 0;
45         for(int k = 0; k < j; k++) {
46             for(int h = 0; h < j; h++) {
47                 if (h == k) continue;
```

```

48         double g = D[0][j];
49         for(int i = 0; i < j; i++) {
50             if(i == k || i == h) continue;
51             g *= xu - x[i];
52         }
53         f += g;
54     }
55 }
56 ddydxdx+= f;
57 }
58 return ddydxdx;
59 }
60
61
62 int findInd(InVector x, double xu) {
63     double diff = fabs(x[0] - xu);
64     for(int i = 1; i < x.size(); i++) {
65         if(fabs(x[i] - xu) > diff) return i - 1;
66         diff = fabs(x[i] - xu);
67     }
68     return -1;
69 }
70
71
72 double nonDiffLeft(InVector x, InVector y, int i) {
73     return (y[i+1] - y[i])/(x[i+1] - x[i]);
74 }
75
76
77 double nonDiffRight(InVector x, InVector y, int i) {
78     return (y[i] - y[i-1])/(x[i] - x[i-1]);
79 }
80
81
82 double nonDiffCentre(InVector x, InVector y, int i) {
83     return (y[i+1] - y[i-1])/(x[i+1] - x[i-1]);
84 }
85
86
87 double nonDiffSecond(InVector x, InVector y, int i){
88     return (y[i+1] - 2*y[i] + y[i-1])/((x[i+1] - x[i])*(x[i] - x[i-1]));
89 }
90
91
92 int main(){
93     const int n = 5;
94     const double xu = 0.2;
95     std::vector<double> x = { -0.2, 0, 0.2, 0.4, 0.6 };
96     std::vector<double> y = { -0.20136, 0, 0.20136, 0.41152, 0.6435 };

```



```

97
98     std::cout << "    " << std::endl;
99     int ind = findInd(x, xu);
100    double dydxL = nonDiffLeft(x, y, ind);
101    double dydxR = nonDiffRight(x, y, ind);
102    double dydxC = nonDiffCentre(x, y, ind);
103    double ddydxndxnd = nonDiffSecond(x, y, ind);
104    std::cout << "    : " << dydxL << std::endl;
105    std::cout << "    : " << dydxR << std::endl;
106    std::cout << "    : " << dydxC << std::endl;
107    std::cout << "    : " << ddydxndxnd << std::endl;
108    std::cout << std::endl;
109
110    std::cout << "    " << std::endl;
111    Matrix D = getPolinom(x, y, n);
112    double dydx = firstDerivative(x, D, xu, n);
113    double ddydxdx = secondDerivative(x, D, xu, n);
114    std::cout << "f' (x*) = " << dydx << std::endl;
115    std::cout << "f''(x*) = " << ddydxdx << std::endl;
116    std::cout << std::endl;
117
118    return 0;
119 }

```

3.5

14 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: 6

$$y = x / ((2x + 7)(3x + 4))$$

$$X_0 = -1, X_k = 1, h_1 = 0.5, h_2 = 0.25$$

15 Результаты работы

```
Метод прямоугольников:  
при h1: -0.03431  
при h2: -0.03924  
уточнение Рунге-Ромберга: -0.04089  
Метод трапеций:  
при h1: -0.05702  
при h2: -0.04566  
уточнение Рунге-Ромберга: -0.04188  
Метод Симпсона:  
при h1: -0.04533  
при h2: -0.04188  
уточнение Рунге-Ромберга: -0.04165
```

Рис. 9: Вывод программы в консоли

16 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 | #include <fstream>
5 | #include <functional>
6 |
7 | using namespace std;
8 |
9 |
10 | double integrate_rectangles(function <double(double)> f, double x0, double x1, double
    h)
11 | {
12 |     double res = 0;
13 |     while (x0 < x1)
14 |     {
15 |         double x = x0 + h;
16 |         res += f((x + x0) / 2) * h;
17 |         x0 = x;
18 |     }
19 |     return res;
20 | }
21 |
22 | double integrate_trapezoids(function <double(double)> f, double x0, double x1, double
    h)
23 | {
24 |     double res = 0;
25 |     while (x0 < x1)
26 |     {
27 |         double x = x0 + h;
28 |         res += (f(x0) + f(x)) * h;
29 |         x0 = x;
30 |     }
31 |     return res / 2;
32 | }
33 |
34 | double integrate_simpson(function <double(double)> f, double x0, double x1, double h)
35 | {
36 |     double res = 0;
37 |     while (x0 < x1)
38 |     {
39 |         double x = x0 + 2 * h;
40 |         double xm = x0 + h;
41 |         res += (f(x0) + 4 * f(xm) + f(x)) * h;
42 |         x0 = x;
43 |     }
44 |     return res / 3;
45 | }
```

```

46
47 // rectangles -> p = 2
48 // trapezoids -> p = 2
49 // simpson -> p = 4
50 double method_runge(double i1, double i2, double h1, double h2, double p)
51 {
52     double k = h2 / h1;
53     return i1 + (i1 - i2) / (pow(k, p) - 1);
54 }
55
56 double f2(double x)
57 {
58     return x / ((2*x + 7) * (3*x + 4));
59 }
60
61 int main()
62 {
63     setlocale(LC_ALL, "Rus");
64     ofstream fout("answer3-5.txt");
65     fout.precision(5);
66     fout << fixed;
67
68     double h1 = 0.5, h2 = 0.25;
69     fout << " : \n";
70     double i1 = integrate_rectangles(f2, -1, 1, h1);
71     double i2 = integrate_rectangles(f2, -1, 1, h2);
72     fout << " h1: " << i1;
73     fout << "\n h2: " << i2;
74     fout << "\n -: " << method_runge(i1, i2, h1, h2, 2);
75     fout << "\n : \n";
76     i1 = integrate_trapezoids(f2, -1, 1, h1);
77     i2 = integrate_trapezoids(f2, -1, 1, h2);
78     fout << " h1: " << i1;
79     fout << "\n h2: " << i2;
80     fout << "\n -: " << method_runge(i1, i2, h1, h2, 2);
81     fout << "\n : \n";
82     i1 = integrate_simpson(f2, -1, 1, h1);
83     i2 = integrate_simpson(f2, -1, 1, h2);
84     fout << " h1: " << i1;
85     fout << "\n h2: " << i2;
86     fout << "\n -: " << method_runge(i1, i2, h1, h2, 4);
87 }

```