

**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: В. В. Хрушкова  
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**

$y = x \sin x, a) X_i = 0, \frac{\pi}{6}, \frac{2\pi}{6}, \frac{3\pi}{6}; ) X_i = 0, \frac{\pi}{6}, \frac{5\pi}{12}, \frac{\pi}{2}; X^* = 0.8$

### 2 Результаты работы

```
Lagrange algo
Result: 0.559218
The absolute difference: 0.00385809

Newton algo
Result: 0.559218
The absolute difference: 0.00385809

Lagrange algo
Result: 0.562287
The absolute difference: 0.00692646

Newton algo
Result: 0.562287
The absolute difference: 0.00692646
```

Рис. 1: Вывод программы в консоли

### 3 Исходный код

```
1 | #include <bits/stdc++.h>
2 | using namespace std;
3 |
4 | const double PI = 3.1415926535;
5 |
6 | double f(double x){
7 |     return x*sin(x);
8 | }
9 |
10 |
11 | int main() {
12 |     vector<double> x_vector = {0.0, PI/6, 2*PI/6, PI/2};
13 |     double target = PI / 4;
14 |
15 |     vector<pair<double, double>> crds;
16 |     for (double x : x_vector)
17 |         crds.emplace_back(x, f(x));
18 |
19 |     vector<double> coefs(crds.size());
20 |     int n = crds.size();
21 |
22 |
23 |     for (int i = 0; i < n; ++i)
24 |         coefs[i] = crds[i].second;
25 |     for (int i = 1; i < n; ++i)
26 |         for (int j = n - 1; j > i-1; --j)
27 |             coefs[j] = (coefs[j] - coefs[j - 1]) / (crds[j].first - crds[j - i].first);
28 |     for (int i = 1; i < n; ++i)
29 |         for (int j = 0; j < i; ++j)
30 |             coefs[i] *= target - crds[j].first;
31 |
32 |     double lagrange = 0;
33 |     for(auto val: coefs)
34 |         lagrange += val;
35 |
36 |
37 |     for (int i = 0; i < n; ++i)
38 |         coefs[i] = crds[i].second;
39 |     for (int i = 0; i < n; ++i)
40 |         for (int j = 0; j < n; ++j)
41 |             if (i != j)
42 |                 coefs[i] /= crds[i].first - crds[j].first;
43 |     for (int i = 0; i < n; ++i)
44 |         for (int j = 0; j < n; ++j)
45 |             if (i != j)
46 |                 coefs[i] *= target - crds[j].first;
47 |     double newton = 0;
```

```

48     for(auto val: coefs)
49         newton += val;
50
51
52     cout << "Lagrange algo" << "\nResult: " << lagrange << "\nThe absolute difference:
53         " << abs(lagrange - f(target)) << endl << endl;
54     cout << "Newton algo" << "\nResult: " << newton << "\nThe absolute difference: " <<
55         abs(newton - f(target)) << endl;
56     return 0;
57 }

```

## 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  $x = X^*$ .

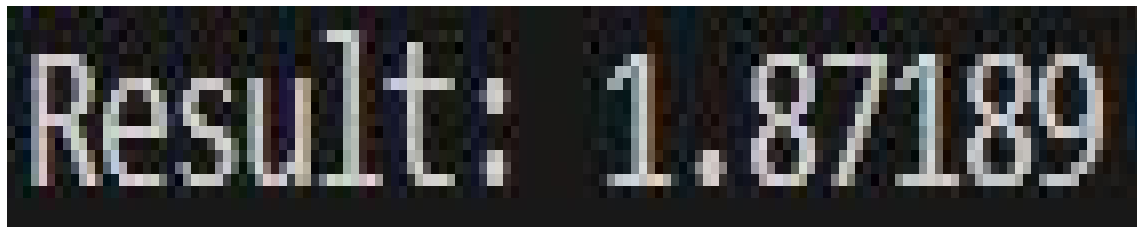
**Вариант: 27**

27.  $X^* = 1.5$

$i$	0	1	2	3	4
$x_i$	0.0	1.0	2.0	3.0	5.0
$f_i$	0.0	0.26180	0.90690	1.5708	1.3090

Рис. 2: Условие

### 5 Результаты работы



```
Result: 1.87189
```

Рис. 3: Вывод программы в консоли

## 6 Исходный код

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 // 1.2
5 vector<vector<double>>> tridiagonal(vector<vector<double>>& coefficients, vector<vector<double>>& results) {
6     int n = coefficients.size();
7     double a = 0, b = coefficients[0][0], c = coefficients[0][1], d = results[0][0];
8     vector<double> P(n, 0), Q(n, 0);
9
10    P[0] = -c/b;
11    Q[0] = d/b;
12    for (int i=1; i < n-1; i++){
13        a = coefficients[i][i-1];
14        b = coefficients[i][i];
15        c = coefficients[i][i+1];
16        d = results[i][0];
17
18        P[i] = -c/(b + a*P[i-1]);
19        Q[i] = (d - a*Q[i-1])/(b + a*P[i-1]);
20    }
21
22    a = coefficients[n-1][n-2];
23    b = coefficients[n-1][n-1];
24    c = 0;
25    d = results[n-1][0];
26
27    Q[n-1] = (d - a * Q[n-2]) / (b + a * P[n-2]);
28
29    vector<vector<double>>> result(n);
30    for(int i=0; i<n; i++)
31        result[i].push_back(0);
32
33    result[n-1][0] = Q[n-1];
34    for (int i = n-2; i > -1; i--)
35        result[i][0] = P[i]*result[i+1][0] + Q[i];
36
37    return result;
38 }
39
40 int main() {
41     double star_x = 1.5;
42     vector<double> x = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0}, y = {0.0, 2.6180, 0.90690,
43         1.5708, 1.3090}, h = {0.0};
44
45     for (int i = 0; i < 4; ++i)
46         h.push_back(x[i + 1] - x[i]);
```

```

46 vector<vector<double>> X = {{2 * (h[1] + h[2]), h[2], 0}}, Y = {};
47
48 for (int i=3; i<4; i++)
49     X.push_back({h[i - 1], 2 * (h[i - 1] + h[i]), h[i]});
50 for (int i=0; i<3; i++)
51     Y.push_back({3 * ((y[i + 2] - y[i + 1]) / h[i + 2] - (y[i + 1] - y[i]) / h[i +
52     1]))});
53 X.push_back({0, h[3], 2 * (h[3] + h[4])});
54
55 vector<double> a(y.begin(), y.end() - 1), b, c = {0}, d;
56 auto result = tridiagonal(X, Y);
57
58 for (auto val : result)
59     c.push_back(val[0]);
60
61 for (int i = 1; i < 4; ++i)
62     b.push_back((y[i] - y[i - 1]) / h[i] - h[i] * (c[i] + 2 * c[i - 1]) / 3);
63 b.push_back((y[4] - y[3]) / h[4] - 2 * h[4] * c[3] / 3);
64
65 for (int i = 0; i < 3; ++i)
66     d.push_back((c[i + 1] - c[i]) / (3 * h[i + 1]));
67 d.push_back(-c[3] / (3 * h[4]));
68
69 for (int i = 0; i < 4; ++i)
70     if (x[i] <= star_x && star_x <= x[i + 1]) {
71         double res = a[i] + b[i]*(star_x-x[i]) + c[i]*(star_x-x[i])*(star_x-x[i]) +
72             d[i]*(star_x-x[i])*(star_x-x[i])*(star_x-x[i]);
73         cout << "Result: " << res << endl;
74         break;
75     }
76 return 0;
77 }

```

### 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**

27.

$i$	0	1	2	3	4	5
$x_i$	-1.0	0.0	1.0	2.0	3.0	5.0
$y_i$	0.5	0.0	0.5	1.7321	3.0	2.5

Рис. 4: Условия

#### 8 Результаты работы

```
Coefficients 0.471371 1.31767          Loss = 0.48743
Coefficients 0.129389 1.61941 -0.0354999  Loss = 0.0944722
```

Рис. 5: Вывод программы в консоли



## 9 Исходный код

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 // 1.1
5 pair<vector<vector<double>>, vector<vector<double>>>> LU(vector<vector<double>>& X,
6     vector<vector<double>>& Y, int n) {
7     vector<vector<double>> L(n);
8     vector<vector<double>> U = X;
9
10    //
11    for (int i=0; i<n; i++)
12        for (int j=0; j<n; j++)
13            L[i].push_back(0);
14
15    for (int k=0; k<n; k++) {
16        //
17        if (U[k][k] == 0) {
18            for (int i=k+1; i<n; i++) {
19                if (U[i][k] != 0) {
20                    swap(X[k], X[i]);
21                    swap(Y[k], Y[i]);
22                    swap(U[k], U[i]);
23                    swap(L[k], L[i]);
24                    break;
25                }
26            }
27        }
28        //
29        //
30        L[k][k] = 1;
31        for (int i=k+1; i<n; i++) {
32            L[i][k] = U[i][k]/U[k][k];
33            if (U[i][k] == 0)
34                continue;
35            for(int j=k; j<n; j++)
36                U[i][j] -= L[i][k]*U[k][j];
37        }
38    }
39 }
40
41 return make_pair(L, U);
42 }
43
44
45 vector<vector<double>> calculate_result(vector<vector<double>>& X, vector<vector<
46     double>>& Y, int n) {
```

```

46     vector<vector<double>> L, U;
47     tie(L, U) = LU(X, Y, n);
48     vector<vector<double>> res = Y;
49
50     int m = res[0].size();
51     //
52     for (int k=0; k<m; k++)
53         for (int i=0; i<n; i++)
54             for (int j=0; j<i; j++)
55                 res[i][k] -= res[j][k]*L[i][j];
56     for (int k=0; k<m; k++) {
57         for (int i=n-1; i>-1; i--) {
58             for (int j=i+1; j<n; j++) {
59                 res[i][k] -= res[j][k]*U[i][j];
60             }
61             res[i][k] /= U[i][i];
62         }
63     }
64
65     return res;
66 }
67
68 int main() {
69     vector<double> x = {0.0, 1.7, 3.4, 5.1, 6.8, 8.5}, y = {0.0, 3.0038, 5.2439,
70         7.3583, 9.4077, 11.415};
71
72     vector<double> s(7, 0);
73     for (int i=0; i<6; i++){
74         for (int j=0; j<4; j++)
75             s[j] += pow(x[i], j+1);
76         for (int j=0; j<3; j++)
77             s[j+4] += y[i]*pow(x[i], j);
78     }
79
80     vector<vector<double>> X = {
81         {6.0, s[0]},
82         {s[0], s[1]}
83     };
84
85     vector<vector<double>> Y = {
86         {s[4]},
87         {s[5]}
88     };
89
90     vector<vector<double>> coefficients = calculate_result(X, Y, 2);
91     cout << "Coefficients " << coefficients[0][0] << " " << coefficients[1][0];
92     double loss = 0;
93     for (int i = 0; i < 6; i++)
94         loss += pow(coefficients[0][0] + coefficients[1][0] * x[i] - y[i], 2);

```

```

94     cout << "\t\tLoss = " << loss << endl;
95
96     X = {
97         {6, s[0], s[1]},
98         {s[0], s[1], s[2]},
99         {s[1], s[2], s[3]}
100    };
101
102     Y = {
103         {s[4]},
104         {s[5]},
105         {s[6]}
106    };
107
108     coefficients = calculate_result(X, Y, 3);
109     cout << "Coefficients " << coefficients[0][0] << " " << coefficients[1][0] << " " <<
        coefficients[2][0];
110     loss = 0;
111     for (int i = 0; i < 6; i++)
112         loss += pow(coefficients[0][0] + coefficients[1][0] * x[i] + coefficients[2][0] *
            pow(x[i], 2) - y[i], 2);
113     cout << "\t\tLoss = " << loss << endl;
114
115     return 0;
116 }

```

## 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**  $X^* = 2.0$

27.  $X^* = 0.0$

$\bar{i}$	0	1	2	3	4
$x_i$	-1.0	-0.5	0.0	0.5	1.0
$y_i$	-0.36788	-0.30327	0.0	0.82436	2.7183

Рис. 6: Условия

### 11 Результаты работы

```
Left derivative = 0.60654    Right derivative = 1.64872    Second derivative = 2.08436
```

Рис. 7: Вывод программы в консоли

## 12 Исходный код

```
1 | #include <bits/stdc++.h>
2 | using namespace std;
3 |
4 | int main() {
5 |     vector<double> x = {-1.0, -0.5, 0.0, 0.5, 1.0}, y = {-0.36788, -0.30327, 0.0,
6 |         0.82436, 2.7183}, p, pp;
7 |     double target = 0.0;
8 |
9 |     for (int i = 0; i < 5; i++)
10 |         p.push_back((y[i + 1] - y[i]) / (x[i + 1] - x[i]));
11 |     for (int i = 0; i < 4; i++)
12 |         pp.push_back(2 * ((y[i + 2] - y[i + 1]) / (x[i + 2] - x[i + 1]) - (y[i + 1] - y
13 |             [i]) / (x[i + 1] - x[i])) / (x[i + 2] - x[i]));
14 |
15 |     for (int i = 0; i < 5; i++)
16 |         if (x[i] == target) {
17 |             cout << "Left derivative = " << p[i - 1] << "\tRight derivative = " << p[i
18 |                 ];
19 |             break;
20 |         }
21 |
22 |     for (int i = 0; i < 4; i++)
23 |         if (x[i] <= target && target <= x[i + 1]) {
24 |             cout << "\tSecond derivative = " << pp[i] << endl;
25 |             break;
26 |         }
27 |     return 0;
28 | }
```

## 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: 27**

$$y = \sqrt{9 + x^2} \quad X_0 = 1, X_k = 5, h_1 = 1.0, h_2 = 0.5$$

### 14 Результаты работы

```
Rectangle
1 result = 17.2773      2 result = 17.9782      3 result: 18.2119

Trapeze
1 result = 17.3448      2 result = 17.3111      3 result: 17.2998

Simpson
1 result = 17.3002      2 result = 17.2998      3 result: 17.2997
```

Рис. 8: Вывод программы в консоли

## 15 Исходный код

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  double my_function(double x) {
5      return sqrt(x*x + 9);
6  }
7
8  int main() {
9      double start_x = 1, end_x = 5, step_1 = 1.0, step_2 = 0.5;
10     double F1, F2;
11
12     cout << "Rectangle" << endl;
13     double x = start_x, t = 0;
14     while (x < end_x){
15         t += my_function((2*x + step_1)/2);
16         x += step_1;
17     }
18     F1 = step_1*t;
19     x = start_x, t = 0;
20     while (x < end_x){
21         t += my_function((2*x + step_1)/2);
22         x += step_2;
23     }
24     F2 = step_2*t;
25     cout << "1 result = " << F1 << "\t2 result = " << F2 << "\t3 result: " << F1 + (F1
        - F2)/(pow((step_2/step_1), 2) - 1) << endl << endl << "Trapeze" << endl;
26
27     x = start_x+step_1, t = my_function(start_x)/2 + my_function(end_x)/2;
28     while (x < end_x){
29         t += my_function(x);
30         x += step_1;
31     }
32     F1 = step_1 * t;
33     x = start_x+step_2, t = my_function(start_x)/2 + my_function(end_x)/2;
34     while (x < end_x){
35         t += my_function(x);
36         x += step_2;
37     }
38     F2 = step_2 * t;
39     cout << "1 result = " << F1 << "\t2 result = " << F2 << "\t3 result: " << F1 + (F1
        - F2)/(pow((step_2/step_1), 2) - 1) << endl << endl << "Simpson" << endl;
40
41     x = start_x + step_1, t = my_function(start_x) + my_function(end_x);
42     bool flag = true;
43     while (x < end_x){
44         t += my_function(x) * ((flag) ? 4 : 2);
45         x += step_1;
```

```

46     flag = !flag;
47 }
48 F1 = step_1 * t / 3;
49 x = start_x + step_2, t = my_function(start_x) + my_function(end_x);
50 flag = true;
51 while (x < end_x){
52     t += my_function(x) * ((flag) ? 4 : 2);
53     x += step_2;
54     flag = !flag;
55 }
56 F2 = step_2 * t / 3;
57 cout << "1 result = " << F1 << "\t2 result = " << F2 << "\t3 result: " << F1 + (F1
    - F2)/(pow((step_2/step_1), 2) - 1) << endl << endl;
58 return 0;
59 }

```