

**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: Наумов Г.К.
Преподаватель: Пивоваров Д.Е.
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

3.1

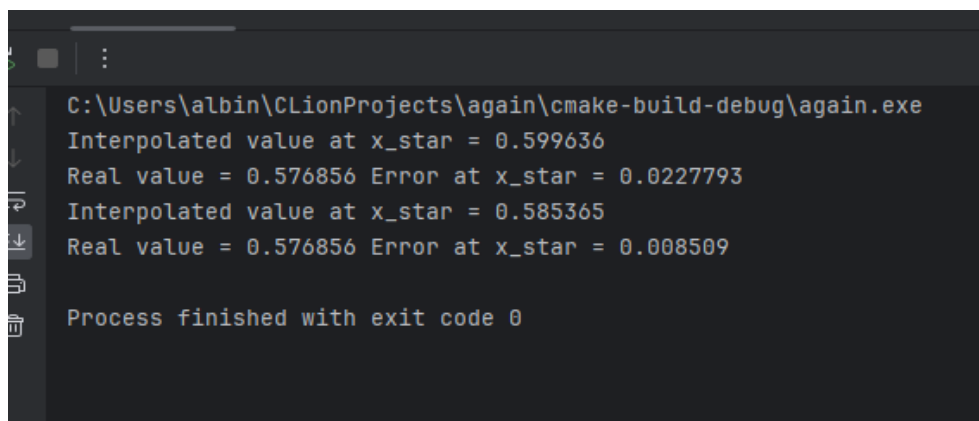
1 Постановка задачи

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: 16

$$16. y = \ln(x) + x, a) X_i = 0.1, 0.5, 0.9, 1.3; \quad б) X_i = 0.1, 0.5, 1.1, 1.3; \quad X^* = 0.8.$$

2 Результаты работы



```
C:\Users\albin\CLionProjects\again\cmake-build-debug\again.exe
Interpolated value at x_star = 0.599636
Real value = 0.576856 Error at x_star = 0.0227793
Interpolated value at x_star = 0.585365
Real value = 0.576856 Error at x_star = 0.008509

Process finished with exit code 0
```

Рис. 1: Вывод программы

3 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 |
5 | double lagrange_interpolation(const std::vector<double>& x, const std::vector<double>&
6 |     y, double x_star) {
7 |     double result = 0.0;
```

```

7   for (size_t i = 0; i < x.size(); ++i) {
8       double term = y[i];
9       for (size_t j = 0; j < x.size(); ++j) {
10          if (j != i) {
11              term *= (x_star - x[j]) / (x[i] - x[j]);
12          }
13      }
14      result += term;
15  }
16  return result;
17 }
18
19 double diff(const std::vector<double>& x, const std::vector<double>& y, size_t n) {
20     if (n == 0) {
21         return y[0];
22     } else {
23         return (diff(x, y, n - 1) - diff(x, y, n - 1)) / (x[n] - x[0]);
24     }
25 }
26
27 double newton_interpolation(const std::vector<double>& x, const std::vector<double>& y
    , double x_star) {
28     double result = y[0];
29     double term = 1.0;
30     for (size_t i = 1; i < x.size(); ++i) {
31         term *= (x_star - x[i - 1]);
32         result += diff(x, y, i) * term;
33     }
34     return result;
35 }
36
37 int main() {
38     std::vector<double> x = {0.1, 0.5, 0.9, 1.3};
39     std::vector<double> y;
40     for (double xi : x) {
41         y.push_back(std::log(xi) + xi);
42     }
43     double x_star = 0.8;
44
45     double interpolated_value = lagrange_interpolation(x, y, x_star);
46     std::cout << "Interpolated value at x_star = " << interpolated_value << std::endl;
47
48     double true_value = std::log(x_star) + x_star;
49     double error = std::abs(true_value - interpolated_value);
50     std::cout << "Real value = " << true_value << " Error at x_star = " << error << std
        ::endl;
51
52     x = {0.1, 0.5, 1.1, 1.3};
53     y.clear();

```

```

54 |     for (double xi : x) {
55 |         y.push_back(std::log(xi) + xi);
56 |     }
57 |     interpolated_value = newton_interpolation(x, y, x_star);
58 |     std::cout << "Interpolated value at x_star = " << interpolated_value << std::endl;
59 |
60 |     true_value = std::log(x_star) + x_star;
61 |     error = std::abs(true_value - interpolated_value);
62 |     std::cout << "Real value = " << true_value << " Error at x_star = " << error << std
        |         ::endl;
63 |
64 |     return 0;
65 | }

```

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 $x = X^*$.

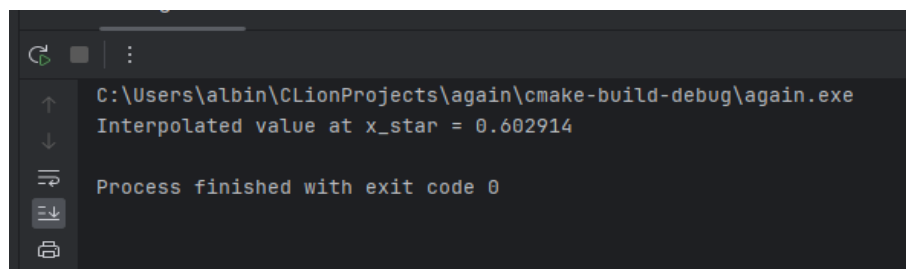
Вариант: 16

16. $X^* = 0.8$

i	0	1	2	3	4
x_i	0.1	0.5	0.9	1.3	1.7
f_i	-2.2026	-0.19315	0.79464	1.5624	2.2306

Рис. 2: Условие

5 Результаты работы



```
C:\Users\albin\CLionProjects\again\cmake-build-debug\again.exe
Interpolated value at x_star = 0.602914
Process finished with exit code 0
```

Рис. 3: Вывод программы

6 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 |
5 | void cubic_spline(const std::vector<double>& x, const std::vector<double>& f, double
   |     x_star) {
6 |     size_t n = x.size();
7 |
```

```

8   std::vector<double> h(n - 1);
9   std::vector<double> alpha(n - 1);
10  std::vector<double> l(n);
11  std::vector<double> mu(n - 1);
12  std::vector<double> z(n);
13
14  for (size_t i = 0; i < n - 1; ++i) {
15      h[i] = x[i + 1] - x[i];
16  }
17
18  for (size_t i = 1; i < n - 1; ++i) {
19      alpha[i] = 3 * (f[i + 1] - f[i]) / h[i] - 3 * (f[i] - f[i - 1]) / h[i - 1];
20  }
21
22  l[0] = 1;
23  mu[0] = 0;
24  z[0] = 0;
25
26  for (size_t i = 1; i < n - 1; ++i) {
27      l[i] = 2 * (x[i + 1] - x[i - 1]) - h[i - 1] * mu[i - 1];
28      mu[i] = h[i] / l[i];
29      z[i] = (alpha[i] - h[i - 1] * z[i - 1]) / l[i];
30  }
31
32  l[n - 1] = 1;
33  z[n - 1] = 0;
34  std::vector<double> c(n);
35  std::vector<double> b(n - 1);
36  std::vector<double> d(n - 1);
37
38  for (int j = n - 2; j >= 0; --j) {
39      c[j] = z[j] - mu[j] * c[j + 1];
40      b[j] = (f[j + 1] - f[j]) / h[j] - h[j] * (c[j + 1] + 2 * c[j]) / 3;
41      d[j] = (c[j + 1] - c[j]) / (3 * h[j]);
42  }
43
44  size_t interval_index = 0;
45  for (size_t i = 0; i < n - 1; ++i) {
46      if (x[i] <= x_star && x_star <= x[i + 1]) {
47          interval_index = i;
48          break;
49      }
50  }
51
52  double A = f[interval_index];
53  double B = b[interval_index];
54  double C = c[interval_index];
55  double D = d[interval_index];

```

```

56     double interpolated_value = A + B * (x_star - x[interval_index]) + C * pow((x_star
    - x[interval_index]), 2) + D * pow((x_star - x[interval_index]), 3);
57
58     std::cout << "Interpolated value at x_star = " << interpolated_value << std::endl;
59 }
60
61 int main() {
62     std::vector<double> x = {0.1, 0.5, 0.9, 1.3, 1.7};
63     std::vector<double> f = {-2.2026, -0.19315, 0.79464, 1.5624, 2.2306};
64     double x_star = 0.8;
65
66     cubic_spline(x, f, x_star);
67
68     return 0;
69 }

```

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.

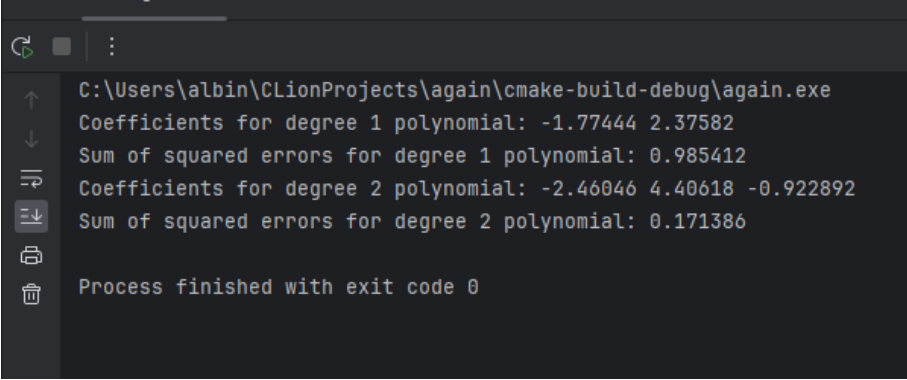
Вариант: 16

16.

i	0	1	2	3	4	5
x_i	0.1	0.5	0.9	1.3	1.7	2.1
y_i	-2.2026	-0.19315	0.79464	1.5624	2.2306	2.8419

Рис. 4: Условия

8 Результаты работы



```
C:\Users\albin\CLionProjects\again\cmake-build-debug\again.exe
Coefficients for degree 1 polynomial: -1.77444 2.37582
Sum of squared errors for degree 1 polynomial: 0.985412
Coefficients for degree 2 polynomial: -2.46046 4.40618 -0.922892
Sum of squared errors for degree 2 polynomial: 0.171386

Process finished with exit code 0
```

Рис. 5: Вывод программы

9 Исходный код

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



```

4
5
6 void solve_system(std::vector<std::vector<double>>& A, std::vector<double>& b, std::
  vector<double>& x) {
7   int n = A.size();
8   for (int i = 0; i < n; ++i) {
9     int pivot = i;
10    for (int j = i + 1; j < n; ++j) {
11      if (abs(A[j][i]) > abs(A[pivot][i])) {
12        pivot = j;
13      }
14    }
15    std::swap(A[i], A[pivot]);
16    std::swap(b[i], b[pivot]);
17    for (int j = i + 1; j < n; ++j) {
18      double factor = A[j][i] / A[i][i];
19      for (int k = i; k < n; ++k) {
20        A[j][k] -= factor * A[i][k];
21      }
22      b[j] -= factor * b[i];
23    }
24  }
25  x.assign(n, 0);
26  for (int i = n - 1; i >= 0; --i) {
27    double sum = 0;
28    for (int j = i + 1; j < n; ++j) {
29      sum += A[i][j] * x[j];
30    }
31    x[i] = (b[i] - sum) / A[i][i];
32  }
33 }
34
35 std::vector<double> least_squares_polynomial(const std::vector<double>& x, const std::
  vector<double>& f, int degree) {
36   int n = x.size();
37   std::vector<std::vector<double>> A(degree + 1, std::vector<double>(degree + 1, 0));
38   std::vector<double> b(degree + 1, 0);
39   for (int i = 0; i <= degree; ++i) {
40     for (int j = 0; j <= degree; ++j) {
41       for (int k = 0; k < n; ++k) {
42         A[i][j] += pow(x[k], i + j);
43       }
44     }
45     for (int k = 0; k < n; ++k) {
46       b[i] += pow(x[k], i) * f[k];
47     }
48   }
49   std::vector<double> coefficients;
50   solve_system(A, b, coefficients);

```

```

51     return coefficients;
52 }
53
54 double evaluate_polynomial(const std::vector<double>& coefficients, double x) {
55     double result = 0;
56     for (size_t i = 0; i < coefficients.size(); ++i) {
57         result += coefficients[i] * pow(x, i);
58     }
59     return result;
60 }
61
62 double sum_of_squared_errors(const std::vector<double>& x, const std::vector<double>&
    f, const std::vector<double>& coefficients) {
63     double sum = 0;
64     for (size_t i = 0; i < x.size(); ++i) {
65         double error = f[i] - evaluate_polynomial(coefficients, x[i]);
66         sum += error * error;
67     }
68     return sum;
69 }
70
71 int main() {
72     std::vector<double> x = {0.1, 0.5, 0.9, 1.3, 1.7, 2.1};
73     std::vector<double> f = {-2.2026, -0.19315, 0.79464, 1.5624, 2.2306, 2.8419};
74
75     std::vector<double> coefficients_degree_1 = least_squares_polynomial(x, f, 1);
76     double sum_of_squared_errors_degree_1 = sum_of_squared_errors(x, f,
        coefficients_degree_1);
77     std::cout << "Coefficients for degree 1 polynomial: ";
78     for (size_t i = 0; i < coefficients_degree_1.size(); ++i) {
79         std::cout << coefficients_degree_1[i] << " ";
80     }
81     std::cout << std::endl;
82     std::cout << "Sum of squared errors for degree 1 polynomial: " <<
        sum_of_squared_errors_degree_1 << std::endl;
83
84     std::vector<double> coefficients_degree_2 = least_squares_polynomial(x, f, 2);
85     double sum_of_squared_errors_degree_2 = sum_of_squared_errors(x, f,
        coefficients_degree_2);
86     std::cout << "Coefficients for degree 2 polynomial: ";
87     for (size_t i = 0; i < coefficients_degree_2.size(); ++i) {
88         std::cout << coefficients_degree_2[i] << " ";
89     }
90     std::cout << std::endl;
91     std::cout << "Sum of squared errors for degree 2 polynomial: " <<
        sum_of_squared_errors_degree_2 << std::endl;
92
93     return 0;
94 }

```

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: 16

16. $X^* = 2.0$

i	0	1	2	3	4
x_i	0.0	1.0	2.0	3.0	4.0
y_i	0.0	2.0	3.4142	4.7321	6.0

Рис. 6: Условия

11 Результаты работы



```
C:\Users\albin\CLionProjects\again\cmake-build-debug\again.exe
First derivative at x_star = 1.36605
Second derivative at x_star = -0.0963

Process finished with exit code 0
```

Рис. 7: Вывод программы

12 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 |
4 | double first_derivative(const std::vector<double>& x, const std::vector<double>& y,
   |     double x_star) {
5 |     int idx = -1;
6 |     for (int i = 0; i < x.size(); ++i) {
7 |         if (x[i] <= x_star)
8 |             idx = i;
```

```

9         else
10             break;
11     }
12     if (idx == -1 || idx == x.size() - 1) {
13         std::cerr << "x_star is outside the range of the table." << std::endl;
14         return 0.0;
15     }
16     double h = x[1] - x[0];
17     double derivative = (y[idx + 1] - y[idx - 1]) / (2 * h);
18
19     return derivative;
20 }
21
22 double second_derivative(const std::vector<double>& x, const std::vector<double>& y,
23     double x_star) {
24     int idx = -1;
25     for (int i = 0; i < x.size(); ++i) {
26         if (x[i] <= x_star)
27             idx = i;
28         else
29             break;
30     }
31
32     if (idx == -1 || idx == x.size() - 1) {
33         std::cerr << "x_star is outside the range of the table." << std::endl;
34         return 0.0;
35     }
36
37     double h = x[1] - x[0];
38     double derivative = (y[idx + 1] - 2 * y[idx] + y[idx - 1]) / (h * h);
39
40     return derivative;
41 }
42
43 int main() {
44     std::vector<double> x = {0.0, 1.0, 2.0, 3.0, 4.0};
45     std::vector<double> y = {0.0, 2.0, 3.4142, 4.7321, 6.0};
46     double x_star = 2.0;
47
48     double first_deriv = first_derivative(x, y, x_star);
49     std::cout << "First derivative at x_star = " << first_deriv << std::endl;
50
51     double second_deriv = second_derivative(x, y, x_star);
52     std::cout << "Second derivative at x_star = " << second_deriv << std::endl;
53
54     return 0;
55 }

```

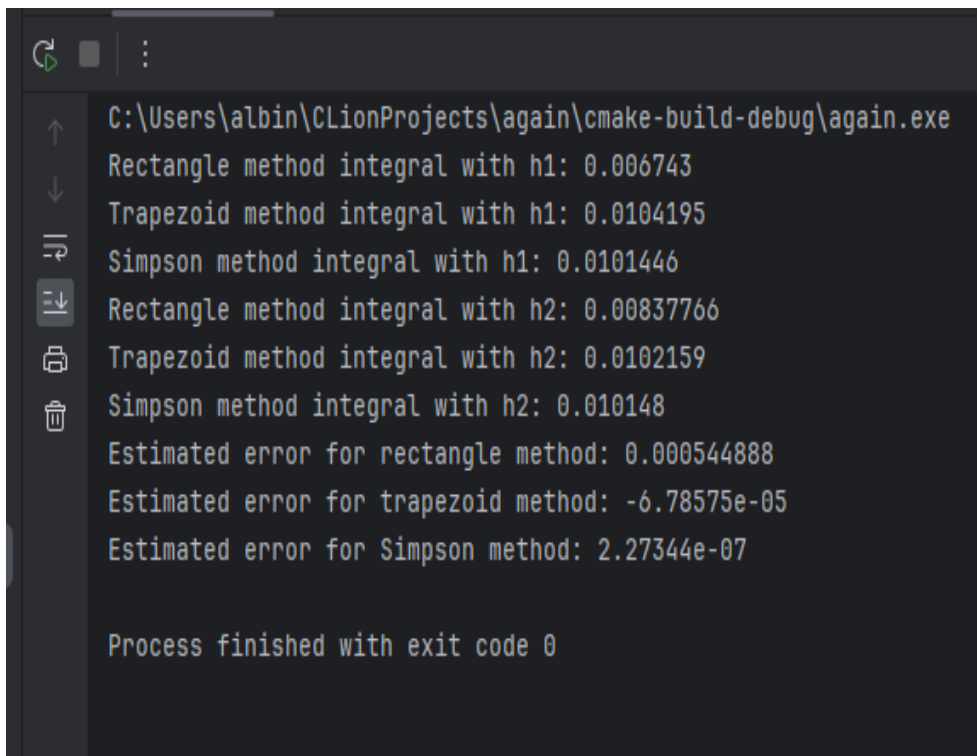
3.5

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: 16

$$16. \quad y = \frac{x^2}{x^4 + 256}, \quad X_0 = 0, \quad X_k = 2, \quad h_1 = 0.5, \quad h_2 = 0.25;$$

14 Результаты работы



```
C:\Users\albin\CLionProjects\again\cmake-build-debug\again.exe
Rectangle method integral with h1: 0.006743
Trapezoid method integral with h1: 0.0104195
Simpson method integral with h1: 0.0101446
Rectangle method integral with h2: 0.00837766
Trapezoid method integral with h2: 0.0102159
Simpson method integral with h2: 0.010148
Estimated error for rectangle method: 0.000544888
Estimated error for trapezoid method: -6.78575e-05
Estimated error for Simpson method: 2.27344e-07

Process finished with exit code 0
```

Рис. 8: Вывод программы

15 Исходный код

```
1 | #include <iostream>
2 | #include <cmath>
3 |
4 | double y(double x) {
5 |     return x * x / (pow(x, 4) + 256);
6 | }
7 |
8 | double rectangle_method(double x0, double x1, double h) {
9 |     double integral = 0.0;
10 |    for (double x = x0; x < x1; x += h) {
11 |        integral += y(x) * h;
12 |    }
13 |    return integral;
14 | }
15 |
16 | double trapezoid_method(double x0, double x1, double h) {
17 |     double integral = 0.0;
18 |     for (double x = x0; x < x1; x += h) {
19 |         integral += (y(x) + y(x + h)) * h / 2.0;
20 |     }
21 |     return integral;
22 | }
23 |
24 | double simpson_method(double x0, double x1, double h) {
25 |     double integral = 0.0;
26 |     for (double x = x0; x < x1; x += 2 * h) {
27 |         integral += (y(x) + 4 * y(x + h) + y(x + 2 * h)) * h / 3.0;
28 |     }
29 |     return integral;
30 | }
31 |
32 | double runge_romberg_method(double I1, double I2, double p) {
33 |     return (I2 - I1) / (pow(2, p) - 1);
34 | }
35 |
36 | int main() {
37 |     double x0 = 0.0;
38 |     double x1 = 2.0;
39 |     double h1 = 0.5;
40 |     double h2 = 0.25;
41 |
42 |     double integral_rect_h1 = rectangle_method(x0, x1, h1);
43 |     double integral_rect_h2 = rectangle_method(x0, x1, h2);
44 |
45 |     double integral_trap_h1 = trapezoid_method(x0, x1, h1);
46 |     double integral_trap_h2 = trapezoid_method(x0, x1, h2);
47 | }
```

```

48     double integral_simpson_h1 = simpson_method(x0, x1, h1);
49     double integral_simpson_h2 = simpson_method(x0, x1, h2);
50
51     double error_rect = runge_romberg_method(integral_rect_h1, integral_rect_h2, 2);
52     double error_trap = runge_romberg_method(integral_trap_h1, integral_trap_h2, 2);
53     double error_simpson = runge_romberg_method(integral_simpson_h1,
54         integral_simpson_h2, 4);
55
56     std::cout << "Rectangle method integral with h1: " << integral_rect_h1 << std::endl
57         ;
58     std::cout << "Trapezoid method integral with h1: " << integral_trap_h1 << std::endl
59         ;
60     std::cout << "Simpson method integral with h1: " << integral_simpson_h1 << std::
61         endl;
62
63     std::cout << "Rectangle method integral with h2: " << integral_rect_h2 << std::endl
64         ;
65     std::cout << "Trapezoid method integral with h2: " << integral_trap_h2 << std::endl
66         ;
67     std::cout << "Simpson method integral with h2: " << integral_simpson_h2 << std::
68         endl;
69
70     std::cout << "Estimated error for rectangle method: " << error_rect << std::endl;
71     std::cout << "Estimated error for trapezoid method: " << error_trap << std::endl;
72     std::cout << "Estimated error for Simpson method: " << error_simpson << std::endl;
73
74     return 0;
75 }

```