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:	Кучмистов Д.Р.
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 $X_i, i = 0, ..3$ построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки $\{X_i, Y_i\}$. Вычислить значение погрешности интерполяции в точке X^*.

Вариант: 14

14. $y = tg(x) + x$,а) $X_i = 0, \frac{\pi}{8}, \frac{2\pi}{8}, \frac{3\pi}{8}$; б) $X_i = 0, \frac{\pi}{8}, \frac{\pi}{3}, \frac{3\pi}{8}$; $X^* = \frac{3\pi}{16}$.

2 Результаты работы

```
For a):
True value: 1.25723
Lagrange polynomial: 1.23366, Error: 0.0235719
Newton polynomial: 1.23366, Error: 0.0235719

For b):
True value: 1.25723
Lagrange polynomial: 1.1743, Error: 0.0829278
Newton polynomial: 1.1743, Error: 0.0829278
```

Рис. 1: Вывод программы

3 Исходный код

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

double lagrangeInterpolation(vector<double> X, vector<double> Y, double x) {
    double result = 0;
```

```
    for (int i = 0; i < X.size(); ++i) {
        double term = Y[i];
        for (int j = 0; j < X.size(); ++j) {
            if (j != i) {
                term = term * (x - X[j]) / (X[i] - X[j]);
            }
        }
        result += term;
    }
    return result;
}

double newtonInterpolation(vector<double> X, vector<double> Y, double x) {
    int n = X.size();
    double result = 0;
    vector<vector<double>> f(n, vector<double>(n, 0));
    for (int i = 0; i < n; ++i) {
        f[i][0] = Y[i];
    }
    for (int i = 1; i < n; ++i) {
        for (int j = 0; j < n - i; ++j) {
            f[j][i] = (f[j + 1][i - 1] - f[j][i - 1]) / (X[i + j] - X[j]);
        }
    }
    for (int i = 0; i < n; ++i) {
        double term = f[0][i];
        for (int j = 0; j < i; ++j) {
            term *= (x - X[j]);
        }
        result += term;
    }
    return result;
}

int main() {
    vector<double> X_a = {0, M_PI / 8, 2 * M_PI / 8, 3 * M_PI / 8};
    vector<double> Y_a;
    for (auto x : X_a) {
        Y_a.push_back(tan(x) + x);
    }

    vector<double> X_b = {0, M_PI / 8, M_PI / 3, 3 * M_PI / 8};
    vector<double> Y_b;
    for (auto x : X_b) {
        Y_b.push_back(tan(x) + x);
    }

    double X_star = 3 * M_PI / 16;

```

```
    double trueValue_a = tan(X_star) + X_star;
    double lagrangeResult_a = lagrangeInterpolation(X_a, Y_a, X_star);
    double newtonResult_a = newtonInterpolation(X_a, Y_a, X_star);
    double trueValue_b = tan(X_star) + X_star;
    double lagrangeResult_b = lagrangeInterpolation(X_b, Y_b, X_star);
    double newtonResult_b = newtonInterpolation(X_b, Y_b, X_star);

    double error_lagrange_a = abs(lagrangeResult_a - trueValue_a);
    double error_newton_a = abs(newtonResult_a - trueValue_a);
    double error_lagrange_b = abs(lagrangeResult_b - trueValue_b);
    double error_newton_b = abs(newtonResult_b - trueValue_b);

    cout << "For a):" << endl;
    cout << "True value: " << trueValue_a << endl;
    cout << "Lagrange polynomial: " << lagrangeResult_a << ", Error: " << error_lagrange_a << endl;
    cout << "Newton polynomial: " << newtonResult_a << ", Error: " << error_newton_a << endl;
    cout << endl;
    cout << "For b):" << endl;
    cout << "True value: " << trueValue_b << endl;
    cout << "Lagrange polynomial: " << lagrangeResult_b << ", Error: " << error_lagrange_b << endl;
    cout << "Newton polynomial: " << newtonResult_b << ", Error: " << error_newton_b << endl;

    return 0;
}
```

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: 14

14. $X^* = 1.5$

i	0	1	2	3	4
x_i	0.0	0.9	1.8	2.7	3.6
f_i	0.0	0.72235	1.5609	2.8459	7.7275

Рис. 2: Условие

5 Результаты работы

```
f(1.5) = 1.31356
```

Рис. 3: Вывод программы

6 Исходный код

```
#include <iostream>
#include <vector>

using namespace std;

class CubicSpline {
private:
    vector<double> x, y;
    vector<double> h, alpha, l, mu, z, c, b, d;

public:
    CubicSpline(const vector<double> &x, const vector<double> &y) : x(x), y(y) {
        int n = x.size();
        h.resize(n);
        alpha.resize(n);
```

```
        l.resize(n);
        mu.resize(n);
        z.resize(n);
        c.resize(n);
        b.resize(n);
        d.resize(n);

        for (int i = 0; i < n - 1; ++i) {
            h[i] = x[i + 1] - x[i];
        }

        for (int i = 1; i < n - 1; ++i) {
            alpha[i] = (3.0 / h[i]) * (y[i + 1] - y[i]) - (3.0 / h[i - 1]) * (y[i] - y[i - 1]);
        }

        l[0] = 1;
        mu[0] = 0;
        z[0] = 0;

        for (int i = 1; i < n - 1; ++i) {
            l[i] = 2 * (x[i + 1] - x[i - 1]) - h[i - 1] * mu[i - 1];
            mu[i] = h[i] / l[i];
            z[i] = (alpha[i] - h[i - 1] * z[i - 1]) / l[i];
        }

        l[n - 1] = 1;
        z[n - 1] = 0;
        c[n - 1] = 0;

        for (int j = n - 2; j >= 0; --j) {
            c[j] = z[j] - mu[j] * c[j + 1];
            b[j] = (y[j + 1] - y[j]) / h[j] - h[j] * (c[j + 1] + 2 * c[j]) / 3;
            d[j] = (c[j + 1] - c[j]) / (3 * h[j]);
        }
    }

    double interpolate(double x_star) {
        int n = x.size();
        int j = 0;
        while (j < n && x[j] < x_star)
            j++;
        if (j >= n)
            j = n - 1;
        double dx = x_star - x[j - 1];
        return y[j - 1] + b[j - 1] * dx + c[j - 1] * dx * dx + d[j - 1] * dx * dx * dx;
    }
};

```

```cpp
int main() {
    vector<double> x = {0.0, 0.9, 1.8, 2.7, 3.6};
    vector<double> y = {0.0, 0.72235, 1.5609, 2.8459, 7.7275};

    CubicSpline spline(x, y);

    double x_star = 1.5;
    double interpolated_value = spline.interpolate(x_star);

    cout << "f(" << x_star << ") = " << interpolated_value << endl;

    return 0;
}
```

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: 14

14.

i	0	1	2	3	4	5
x_i	-0.9	0.0	0.9	1.8	2.7	3.6
y_i	-1.2689	0.0	1.2689	2.6541	4.4856	9.9138

Рис. 4: Условия

8 Результаты работы

```
First-degree polynomial coefficients:
a0 = -0.1901, a1 = 2.2462
Sum of squared errors for first-degree polynomial: 8.6790

Second-degree polynomial coefficients:
a0 = -0.4645, a1 = 0.8744, a2 = 0.5081
Sum of squared errors for second-degree polynomial: 2.3557
```

Рис. 5: Вывод программы

9 Исходный код

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>

```

```
using namespace std;

double sumOfSquaredErrors(const vector<double>& x, const vector<double>& y, const vector<double>& y_approx) {
    double sum = 0.0;
    for (size_t i = 0; i < x.size(); ++i) {
        double error = y[i] - y_approx[i];
        sum += error * error;
    }
    return sum;
}

vector<double> findPolynomialCoefficients(const vector<double>& x, const vector<double>& y, int degree) {
    int n = x.size();
    vector<double> sumX(2 * degree + 1, 0.0);
    vector<double> sumY(degree + 1, 0.0);
    vector<double> a(degree + 1, 0.0);

    for (int i = 0; i < n; ++i) {
        double xi = x[i];
        double yi = y[i];
        for (int j = 0; j <= 2 * degree; ++j) {
            sumX[j] += pow(xi, j);
        }
        for (int j = 0; j <= degree; ++j) {
            sumY[j] += yi * pow(xi, j);
        }
    }

    vector<vector<double>> A(degree + 1, vector<double>(degree + 2, 0.0));

    for (int i = 0; i <= degree; ++i) {
        for (int j = 0; j <= degree; ++j) {
            A[i][j] = sumX[i + j];
        }
        A[i][degree + 1] = sumY[i];
    }

    for (int i = 0; i < degree; ++i) {
        for (int k = i + 1; k <= degree; ++k) {
            double ratio = A[k][i] / A[i][i];
            for (int j = 0; j <= degree + 1; ++j) {
                A[k][j] -= ratio * A[i][j];
            }
        }
    }

    for (int i = degree; i >= 0; --i) {
```

```
        a[i] = A[i][degree + 1];
        for (int j = i + 1; j <= degree; ++j) {
            if (j != degree + 1) {
                a[i] -= A[i][j] * a[j];
            }
        }
        a[i] /= A[i][i];
    }
    return a;
}

double evaluatePolynomial(const vector<double>& a, double x) {
    double result = 0.0;
    for (size_t i = 0; i < a.size(); ++i) {
        result += a[i] * pow(x, i);
    }
    return result;
}

int main() {
    vector<double> x = {-0.9, 0.0, 0.9, 1.8, 2.7, 3.6};
    vector<double> y = {-1.2689, 0.0, 1.2689, 2.6541, 4.4856, 9.9138};
    int n = x.size();

    vector<double> a1 = findPolynomialCoefficients(x, y, 1);
    cout << "First-degree polynomial coefficients:" << endl;
    cout << "a0 = " << fixed << setprecision(4) << a1[0] << ", a1 = " << a1[1] << endl;

    vector<double> y_approx1(n);
    for (int i = 0; i < n; ++i) {
        y_approx1[i] = evaluatePolynomial(a1, x[i]);
    }

    double sum_sq_error_1 = sumOfSquaredErrors(x, y, y_approx1);
    cout << "Sum of squared errors for first-degree polynomial: " << sum_sq_error_1 << endl;

    vector<double> a2 = findPolynomialCoefficients(x, y, 2);
    cout << "\nSecond-degree polynomial coefficients:" << endl;
    cout << "a0 = " << a2[0] << ", a1 = " << a2[1] << ", a2 = " << a2[2] << endl;

    vector<double> y_approx2(n);
    for (int i = 0; i < n; ++i) {
        y_approx2[i] = evaluatePolynomial(a2, x[i]);
    }

    double sum_sq_error_2 = sumOfSquaredErrors(x, y, y_approx2);
    cout << "Sum of squared errors for second-degree polynomial: " << sum_sq_error_2 << endl;
```

```

    return 0;
}
```

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: 14

14. $X^* = 3.0$

i	0	1	2	3	4
x_i	1.0	2.0	3.0	4.0	5.0
y_i	1.0	2.6931	4.0986	5.3863	6.6094

Рис. 6: Условия

11 Результаты работы

```
First derivative at x = 3: 1.4055
Second derivative at x = 3: -0.1178
```

Рис. 7: Вывод программы

12 Исходный код

```cpp
#include <iostream>
#include <vector>

using namespace std;

vector<double> x = {1.0, 2.0, 3.0, 4.0, 5.0};
vector<double> y = {1.0, 2.6931, 4.0986, 5.3863, 6.6094};
double X_star = 3.0;

double first_derivative(vector<double>& x, vector<double>& y, double X_star) {
    int n = x.size();
    double h = x[1] - x[0];

    int idx = 0;
    for (int i = 0; i < n; ++i) {
```

```
        if (x[i] < X_star)
            idx = i;
        else
            break;
    }

    double first_derivative = (y[idx + 1] - y[idx]) / h;

    return first_derivative;
}

double second_derivative(vector<double>& x, vector<double>& y, double X_star) {
    int n = x.size();
    double h = x[1] - x[0];

    int idx = 0;
    for (int i = 0; i < n; ++i) {
        if (x[i] < X_star)
            idx = i;
        else
            break;
    }

    double second_derivative = (y[idx + 2] - 2 * y[idx + 1] + y[idx]) / (h * h);

    return second_derivative;
}

int main() {
    double first = first_derivative(x, y, X_star);
    double second = second_derivative(x, y, X_star);

    cout << "First derivative at x = " << X_star << ": " << first << endl;
    cout << "Second derivative at x = " << X_star << ": " << second << endl;

    return 0;
}
```

3.5

13 Постановка задачи

Вычислить определенный интеграл $\int\limits_{X_0}^{X_1} ydx$, методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга:
Вариант: 14

14. $y = \dfrac{1}{x^4 + 16}$, $X_0 = 0$, $X_k = 2$, $h_1 = 0.5$, $h_2 = 0.25$;

14 Результаты работы

```
Rectangle method (h=0.5): 0.115529
Rectangle method (h=0.25): 0.112115
Error estimate (rectangles): 0.00113806

Trapezoidal method (h=0.5): 0.107717
Trapezoidal method (h=0.25): 0.108209
Error estimate (trapezoids): -0.000164022

Simpson's method (h=0.5): 0.108389
Simpson's method (h=0.25): 0.108373
Error estimate (Simpson): 1.10543e-06
```

Рис. 8: Вывод программы

15 Исходный код

```
#include <iostream>
#include <cmath>

using namespace std;

double func(double x) {
    return 1.0 / (pow(x, 4) + 16);
}

double rectangle_method(double a, double b, int n) {
    double h = (b - a) / n;
    double sum = 0;
    for (int i = 0; i < n; i++) {
        sum += func(a + i * h);
    }
    return h * sum;
}

double trapezoidal_method(double a, double b, int n) {
    double h = (b - a) / n;
    double sum = (func(a) + func(b)) / 2.0;
    for (int i = 1; i < n; ++i) {
        sum += func(a + i * h);
    }
    return h * sum;
}

double simpson_method(double a, double b, int n) {
    if (n % 2 != 0) {
        cout << "Number of intervals must be even for Simpson's method." << endl;
        return 0;
    }
    double h = (b - a) / n;
    double sum = func(a) + func(b);
    for (int i = 1; i < n; ++i) {
        if (i % 2 == 0)
            sum += 2 * func(a + i * h);
        else
            sum += 4 * func(a + i * h);
    }
    return h * sum / 3.0;
}

double runge_romberg(double I_h, double I_2h, int p) {
    return (I_h - I_2h) / (pow(2, p) - 1);
}

```

```
int main() {
    double X_0 = 0.0;
    double X_1 = 2.0;
    double h_1 = 0.5;
    double h_2 = 0.25;

    double I_rectangle_h1 = rectangle_method(X_0, X_1, (X_1 - X_0) / h_1);
    double I_rectangle_h2 = rectangle_method(X_0, X_1, (X_1 - X_0) / h_2);

    double I_trapezoidal_h1 = trapezoidal_method(X_0, X_1, (X_1 - X_0) / h_1);
    double I_trapezoidal_h2 = trapezoidal_method(X_0, X_1, (X_1 - X_0) / h_2);

    double I_simpson_h1 = simpson_method(X_0, X_1, (X_1 - X_0) / h_1);
    double I_simpson_h2 = simpson_method(X_0, X_1, (X_1 - X_0) / h_2);

    double error_rectangle = runge_romberg(I_rectangle_h1, I_rectangle_h2, 2);
    double error_trapezoidal = runge_romberg(I_trapezoidal_h1, I_trapezoidal_h2, 2);
    double error_simpson = runge_romberg(I_simpson_h1, I_simpson_h2, 4);

    cout << "Rectangle method (h=0.5): " << I_rectangle_h1 << endl;
    cout << "Rectangle method (h=0.25): " << I_rectangle_h2 << endl;
    cout << "Error estimate (rectangles): " << error_rectangle << endl << endl;

    cout << "Trapezoidal method (h=0.5): " << I_trapezoidal_h1 << endl;
    cout << "Trapezoidal method (h=0.25): " << I_trapezoidal_h2 << endl;
    cout << "Error estimate (trapezoids): " << error_trapezoidal << endl << endl;

    cout << "Simpson's method (h=0.5): " << I_simpson_h1 << endl;
    cout << "Simpson's method (h=0.25): " << I_simpson_h2 << endl;
    cout << "Error estimate (Simpson): " << error_simpson << endl;

    return 0;
}
```