**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 $X_i, i = 0, ..3$ 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

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


double func(double x){
    return exp(x);
}

double lagrange(const std::vector<double> &x, double xu, int n){
    std::cout << "L(x) = ";
    double res = 0;
    for(int j = 0; j < n; j++) {
        double a = func(x[j]);
        double f = 1;
        for(int i = 0; i < n; i++) {
            if(i != j) a/=x[j] - x[i];
        }
        if(j && a >= 0) {
            std::cout << " + ";
        } else {
            std::cout << " - ";
        }
        std::cout << round(fabs(a)*10000)/10000.0;
        for(int i = 0; i < n; i++) {
            std::cout << "(x";
            if(x[i] >= 0) std::cout << "+";
            std::cout << x[i] << ")";
            if (i != j) f *= xu - x[i];
        }
        res += a*f;
    }
    std::cout << std::endl;
    return res;
}


double Newton(const std::vector<double> &x, double xu, int n){
    std::vector<std::vector<double>> d(n, std::vector<double>(n));
    for(int i = 0; i < n; i++){
        d[i][0] = func(x[i]);
    }
    for(int j = 1 ; j < n; j++){
        for(int i = 0; i < n-j; i++){
            d[i][j] = (d[i][j-1]-d[i+1][j-1])/float(x[i]-x[i+j]);
        }
    }
```

```
    std::cout << "N(x) = ";
    double res = 0;
    for(int j = 0; j < n; j++){
        double a = d[0][j];
        double f = 1;
        if(j && a >= 0) {
            std::cout << " + ";
        } else {
            std::cout << " - ";
        }
        std::cout << round(fabs(a)*10000)/10000.0;
        for(int i = 0; i < j; i++) {
            std::cout << "(x";
            if(x[i] >= 0) std::cout << "+";
            std::cout << x[i] << ")";
            f *= xu - x[i];
        }
        res += a*f;
    }
    std::cout << std::endl;
    return res;
}


int main(){
    const int n = 4;
    const double xu = -0.5;
    std::vector<double> Xa = {-2, -1, 0, 1};
    std::vector<double> Xb = {-2, -1, 0.2, 1};

    std::cout << " " << std::endl;
    std::cout << "a) ";
    double res = lagrange(Xa, xu, n);
    std::cout << "L(x*)=" << res << std::endl;
    std::cout << "y(x*)=" << func(xu) << std::endl;
    std::cout << ": " << fabs(res - func(xu)) << std::endl;

    std::cout << "b) ";
    res = lagrange(Xb, xu, n);
    std::cout << "L(x*)=" << res << std::endl;
    std::cout << "y(x*)=" << func(xu) << std::endl;
    std::cout << ": " << fabs(res - func(xu)) << std::endl;
    std::cout << std::endl;


    std::cout << " " << std::endl;
    std::cout << "a) ";
    res = Newton(Xa, xu, n);
    std::cout << "N(x*)=" << res << std::endl;
```

```
    std::cout << "y(x*)=" << func(xu) << std::endl;
    std::cout << ": " << fabs(res - func(xu)) << std::endl;

    std::cout << "b) ";
    res = Newton(Xb, xu, n);
    std::cout << "N(x*)=" << res << std::endl;
    std::cout << "y(x*)=" << func(xu) << std::endl;
    std::cout << ": " << fabs(res - func(xu)) << std::endl;
    std::cout << std::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:** 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

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

using InVector = const std::vector<double> &;


std::vector<double> threeDiagonal(InVector x, InVector y, int n) {
    std::vector<double> p(n-2);
    std::vector<double> q(n-2);
    p[0] = -(x[2]-x[1])/(2*x[3]-2*x[1]);
    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]);
    for(int i = 1; i < n - 2; i++){
        p[i] = -(x[i+2]-x[i+1])/((2*x[i+2]-2*x[i])+(x[i+1]-x[i])*p[i-1]);
        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]);
    }
    std::vector<double> res(n);
    res[n-1] = q[n-2];
    for(int i = n-2; i > 0; i--){
        res[i] = p[i-1]*res[i+1] + q[i-1];
    }
    return res;
}


std::string sign(double x) {
    return x? " + ": " - ";
}


double spline(InVector x, InVector y, double xu, int n) {
    std::vector<double> a(n-1);
    std::vector<double> b(n-1);
    std::vector<double> c(n-1);
    std::vector<double> d(n-1);
    for(int i = 0; i < n-1; i++) a[i] = y[i];
    c = threeDiagonal(x, y, n);
    for(int i = 0; i < n-2; i++) {
        b[i] = (y[i+1]-y[i])/(x[i+1]-x[i])-(x[i+1]-x[i])*(c[i+1]+2*c[i])/3;
        d[i] = (c[i+1]-c[i])/(x[i+1]-x[i])/3;
    }
    b[n-1] = (y[n]-y[n-1])/(x[n]-x[n-1])-(x[n]-x[n-1])*2*c[n-1]/3;
    d[n-1] = -c[n-1]/(x[n]-x[n-1])/3;

    for(int i = 0; i < n-1; i++){
        std::cout << "x  [" << x[i] << ", " << x[i+1] << "], =>" << std::endl;
```

```
        std::cout << "\t s(x) = ";
        std::cout << a[i];
        std::cout << sign(b[i]) << fabs(b[i]) << "(x" << sign(-x[i]) << fabs(x[i]) << "
            )";
        std::cout << sign(c[i]) << fabs(c[i]) << "(x" << sign(-x[i]) << fabs(x[i]) << "
            )^2";
        std::cout << sign(d[i]) << fabs(d[i]) << "(x" << sign(-x[i]) << fabs(x[i]) << "
            )^3";
        std::cout << std::endl << std::endl;
    }

    int j = 0;
    for(int i = 0; i < n-1; i++) {
        if(xu >= x[i] && xu <= x[i+1]) j = i;
    }
    if(!j) return xu;
    xu -= x[j];
    return a[j]+b[j]*xu+c[j]*xu*xu + d[j]*xu*xu*xu;
}


int main(){
    const int n = 5;
    const double xu = -0.5;
    std::vector<double> x = { -2, -1, 0, 1, 2 };
    std::vector<double> y = { 0.13534, 0.36788, 1, 2.7183, 7.3891 };
    double Sxu = spline(x, y, xu, n);
    std::cout << "s(x*) = " << Sxu << std::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:** 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

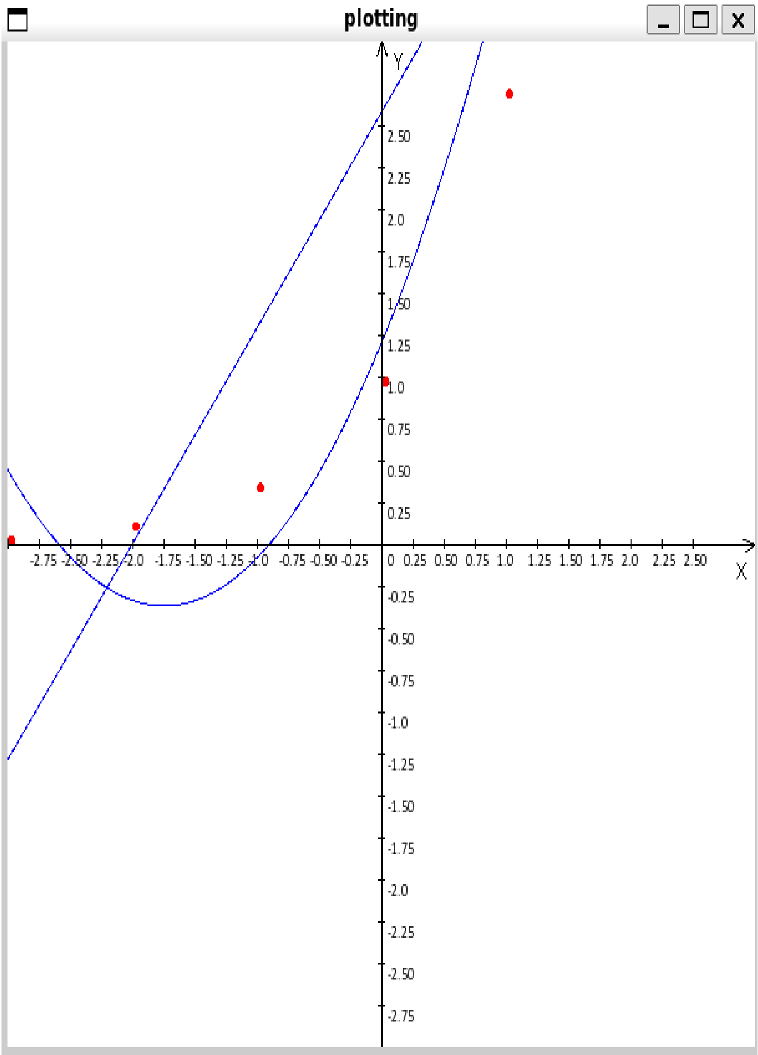


Рис. 5: Вывод программы в консоли

# 9 Результаты работы

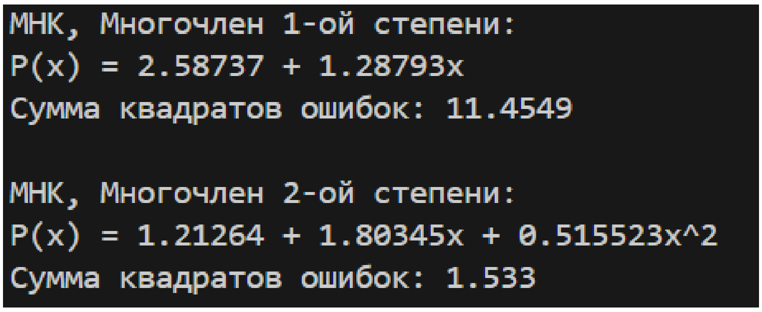


Рис. 6: Вывод программы в консоли

## 10 Исходный код

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


using InVector = std::vector<double> &;
using InMatrix = std::vector<std::vector<double>> &;


double det(InMatrix a){
    double D = 0;
    int size = a.size();
    if(size == 1) return a[0][0];
    double sign = 1;
    for(int k = 0; k < size; k++) {
        std::vector<std::vector<double>> minor(size-1, std::vector<double>(size-1));
        for(int i = 0; i < size-1; i++) {
            for(int j = 0; j < size-1; j++) {
                int mj = (j < k)? j: j+1;
                minor[i][j] = a[i+1][mj];
            }
        }
        double temp = det(minor);
        D += sign*a[0][k]*temp;
        sign *= -1;
    }
    return D;
}

std::vector<double> Cramer(InMatrix a, InVector b, int m){
    std::vector<double> x(m);
    for(int i = 0; i < m; i++) {
        std::vector<std::vector<double>> ai(a);
        for(int j = 0; j < m; j++) {
```

```
            ai[j][i] = b[j];
        }
        x[i] = det(ai)/det(a);
    }
    return x;
}


std::string sign(double x) {
    return (x < 0)? " - ": " + ";
}


double minSquare(InVector x, InVector y, int m, int n){
    std::vector<std::vector<double>> a(m, std::vector<double>(m));
    std::vector<double> b(m);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            for(int k = 0; k < n; k++) {
                a[i][j] += pow(x[k], i+j);
            }
        }
        for(int k = 0; k < n; k++) {
            b[i] += y[k]*pow(x[k], i);
        }
    }
    std::vector<double> z(Cramer(a, b, m));
    double err = 0;
    for(int k = 0; k < n; k++){
        double px = 0;
        for(int i = 0; i < m; i++) {
            px += z[i]*pow(x[k], i);
        }
        err += (px - y[k])*(px - y[k]);
    }
    std::cout << "P(x) = " << z[0];
    for(int i = 1; i < m; i++) {
        std::cout << sign(z[i]) << fabs(z[i]) << "x";
        if(i > 1) std::cout << "^" << i;
    }
    std::cout << std::endl;
    return err;
}


int main(){
    const int n = 6;
    std::vector<double> x = { -3, -2, -1, 0, 1, 2 };
    std::vector<double> y = { 0.04979, 0.13534, 0.36788, 1, 2.7183, 7.3891 };
```

```
    std::cout << ",  1- :" << std::endl;
    double res = minSquare(x, y, 2, n);
    std::cout << "  : " << res << std::endl;
    std::cout << std::endl;

    std::cout << ",  2- :" << std::endl;
    res = minSquare(x, y, 3, n);
    std::cout << "  : " << res << std::endl;
    std::cout << std::endl;
    return 0;
}
```

# 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

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


using InVector = std::vector<double> &;
using Matrix = std::vector<std::vector<double>>;


Matrix getPolinom(InVector x, InVector y, int n){
    Matrix D(n, std::vector<double>(n));
    for(int i = 0; i < n; i++) {
        D[i][0] = y[i];
    }
    for(int j = 1; j < n; j++) {
        for(int i = 0; i < n-j; i++) {
            D[i][j] = (D[i][j-1] - D[i+1][j-1])/(x[i]-x[i+j]);
        }
    }
    return D;
}


double firstDerivative(InVector x, Matrix &D, double xu, int n) {
    double dydx = 0;
    for(int j = 1; j < n; j++) {
        double f = 0;
        for(int k = 0; k < j; k++) {
            double g = D[0][j];
            for(int i = 0; i < j; i++) {
                if(i!=k) g*=xu-x[i];
            }
            f += g;
        }
        dydx += f;
    }
    return dydx;
}


double secondDerivative(InVector x, Matrix &D, double xu, int n) {
    double ddydxdx = 0;
    for(int j = 2; j < n; j++) {
        double f = 0;
        for(int k = 0; k < j; k++) {
            for(int h = 0; h < j; h++) {
                if (h == k) continue;
```

```
                double g = D[0][j];
                for(int i = 0; i < j; i++) {
                    if(i == k || i == h) continue;
                    g *= xu - x[i];
                }
                f += g;
            }
        }
        ddydxdx+= f;
    }
    return ddydxdx;
}


int findInd(InVector x, double xu) {
    double diff = fabs(x[0] - xu);
    for(int i = 1; i < x.size(); i++) {
        if(fabs(x[i] - xu) > diff) return i - 1;
        diff = fabs(x[i] - xu);
    }
    return -1;
}


double nonDiffLeft(InVector x, InVector y, int i) {
    return (y[i+1] - y[i])/(x[i+1] - x[i]);
}


double nonDiffRight(InVector x, InVector y, int i) {
    return (y[i] - y[i-1])/(x[i] - x[i-1]);
}


double nonDiffCentre(InVector x, InVector y, int i) {
    return (y[i+1] - y[i-1])/(x[i+1] - x[i-1]);
}


double nonDiffSecond(InVector x, InVector y, int i){
    return (y[i+1] - 2*y[i] + y[i-1])/((x[i+1] - x[i])*(x[i] - x[i-1]));
}


int main(){
    const int n = 5;
    const double xu = 0.2;
    std::vector<double> x = { -0.2, 0, 0.2, 0.4, 0.6 };
    std::vector<double> y = { -0.20136, 0, 0.20136, 0.41152, 0.6435 };
```

```

    std::cout << "    " << std::endl;
    int ind = findInd(x, xu);
    double dydxL = nonDiffLeft(x, y, ind);
    double dydxR = nonDiffRight(x, y, ind);
    double dydxC = nonDiffCentre(x, y, ind);
    double ddydxdxnd = nonDiffSecond(x, y, ind);
    std::cout << "   : " << dydxL << std::endl;
    std::cout << "   : " << dydxR << std::endl;
    std::cout << "   : " << dydxC << std::endl;
    std::cout << " : " << ddydxdxnd << std::endl;
    std::cout << std::endl;

    std::cout << "    " << std::endl;
    Matrix D = getPolinom(x, y, n);
    double dydx = firstDerivative(x, D, xu, n);
    double ddydxdx = secondDerivative(x, D, xu, n);
    std::cout << "f' (x*) = " << dydx << std::endl;
    std::cout << "f''(x*) = " << ddydxdx << std::endl;
    std::cout << std::endl;

    return 0;
}
```

## 3.5

# 14 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:** 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

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

using namespace std;


double integrate_rectangles(function <double(double)> f, double x0, double x1, double
    h)
{
    double res = 0;
    while (x0 < x1)
    {
        double x = x0 + h;
        res += f((x + x0) / 2) * h;
        x0 = x;
    }
    return res;
}

double integrate_trapezoids(function <double(double)> f, double x0, double x1, double
    h)
{
    double res = 0;
    while (x0 < x1)
    {
        double x = x0 + h;
        res += (f(x0) + f(x)) * h;
        x0 = x;
    }
    return res / 2;
}

double integrate_simpson(function <double(double)> f, double x0, double x1, double h)
{
    double res = 0;
    while (x0 < x1)
    {
        double x = x0 + 2 * h;
        double xm = x0 + h;
        res += (f(x0) + 4 * f(xm) + f(x)) * h;
        x0 = x;
    }
    return res / 3;
}
```

```
// rectangles -> p = 2
// trapezoids -> p = 2
// simpson -> p = 4
double method_runge(double i1, double i2, double h1, double h2, double p)
{
    double k = h2 / h1;
    return i1 + (i1 - i2) / (pow(k, p) - 1);
}

double f2(double x)
{
    return x / ((2*x + 7) * (3*x + 4));
}

int main()
{
    setlocale(LC_ALL, "Rus");
    ofstream fout("answer3-5.txt");
    fout.precision(5);
    fout << fixed;

    double h1 = 0.5, h2 = 0.25;
    fout << " :\n";
    double i1 = integrate_rectangles(f2, -1, 1, h1);
    double i2 = integrate_rectangles(f2, -1, 1, h2);
    fout << " h1: " << i1;
    fout << "\n h2: " << i2;
    fout << "\n -: " << method_runge(i1, i2, h1, h2, 2);
    fout << "\n :\n";
    i1 = integrate_trapezoids(f2, -1, 1, h1);
    i2 = integrate_trapezoids(f2, -1, 1, h2);
    fout << " h1: " << i1;
    fout << "\n h2: " << i2;
    fout << "\n -: " << method_runge(i1, i2, h1, h2, 2);
    fout << "\n :\n";
    i1 = integrate_simpson(f2, -1, 1, h1);
    i2 = integrate_simpson(f2, -1, 1, h2);
    fout << " h1: " << i1;
    fout << "\n h2: " << i2;
    fout << "\n -: " << method_runge(i1, i2, h1, h2, 4);
}
```