Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет Программной инженерии и компьютерной техники

Кафедра информатики и прикладной математики

**Дисциплина «Вычислительная математика»**

**Лабораторная работа №4**

**Усовершенствованный метод Эйлера**

Выполнил:

Съестов Дмитрий Вячеславович

Группа Р3217

Преподаватель:

Калёнова Ольга Вячеславовна

Санкт-Петербург, 2017 г.

**Описание метода**

Точность метода Эйлера можно повысить, если воспользоваться для аппроксимации интеграла более точной формулой интегрирования – формулой трапеций.

Основная идея этого метода: вычисляемое по формуле очередное значение

yi+1 = yi + h\*f(xi, yi) будет точнее, если значение производной, то есть угловой коэффициент прямой, замещающей интегральную кривую на отрезке [xi, xi+1] будет вычисляться не по левому краю (то есть в точке xi), а по центру отрезка. Но так как значение производной между точками xi и xi+1 не вычисляется, то перейдем к сдвоенным участкам [xi-1, xi+1], центром в которых является точка xi. Сначала вычисляется значение

в точке , а затем находится y1 с шагом h/2:

Таким образом, вычисления производятся по формуле

**Листинг программы**

using UnaryFunc = System.Func<double, double>;

using BinaryFunc = System.Func<double, double, double>;

public static class AdvancedEulerMethod

{

const int INITIAL\_STEP\_COUNT = 5;

//Разделённая разность (k - порядок)

private static double difference(Point[] points, int i, int k)

{

Assert.IsMore(k, -1);

if (k == 0) return points[i].Y;

return (difference(points, i + 1, k - 1) - difference(points, i, k - 1)) / (points[k].X - points[0].X);

}

private static UnaryFunc interpolation(Point[] points)

{

int nodeCount = points.Length;

double step = points[1].X - points[0].X;

Assert.IsNotNull(points);

Assert.IsPositive(step);

double[] coefficients = new double[nodeCount];

for (int i = 0; i < nodeCount; i++)

coefficients[i] = difference(points, 0, i);

UnaryFunc polynomial = x =>

{

double sum = 0, product = 1, xn = points[0].X;

double[] coefs = coefficients;

for (int i = 0; i < nodeCount; i++)

{

sum += coefs[i] \* product;

product \*= (x - xn);

xn += step;

}

return sum;

};

return polynomial;

}

//Вычисление 1-го элемента аппроксимации с N шагов

private static double quickY1(BinaryFunc f, double x0, double y0, double xn, double step)

{

double deltaY = step \* f(x0 + step/2, y0 + step/2 \* f(x0, y0));

return y0 + deltaY;

}

//Вычисление 2-го элемента аппроксимации с 2N шагов

private static double quickY2(BinaryFunc f, double x0, double y0, double xn, double step)

{

double deltaY = step \* f(x0 + step/2, y0 + step/2 \* f(x0, y0));

double y1 = y0 + deltaY;

deltaY = step \* f(x0 + step \* 1.5, y1 + step/2 \* f(x0 + step, y1));

return y1 + deltaY;

}

//Получение массива точек для интерполяции

private static Point[] approximate(BinaryFunc f, double x0, double y0, double xn, double step)

{

Assert.IsNotNull(f);

Assert.IsPositive(step);

Assert.IsMore(xn, x0);

int stepCount = (int)((xn - x0) / step);

Assert.IsTrue(stepCount % INITIAL\_STEP\_COUNT == 0);

var points = new Point[stepCount];

points[0] = new Point(x0, y0);

double yi, deltaY, xi = x0;

for (int i = 0; i < stepCount - 1; i++)

{

yi = points[i].Y;

deltaY = step \* f(xi + step/2, yi + step/2 \* f(xi, yi));

xi += step;

points[i + 1] = new Point(xi, yi + deltaY);

}

return points;

}

public static UnaryFunc Solve(BinaryFunc f, double x0, double y0, double xn, double precision)

{

if (f == null) throw new ArgumentNullException("производная не определена");

if (xn <= x0) throw new ArgumentException("xn должен быть больше x0");

if (precision <= 0) throw new ArgumentOutOfRangeException("точность должна быть положительной");

double step = (xn - x0) / INITIAL\_STEP\_COUNT;

/\* Так как для оценки Рунге используется только один элемент от каждой аппроксимации, вычисляем только этот элемент, а не весь массив \*/

double error; //погрешность

double yn; //1-й элемент аппроксимации с N шагов

double y2n; //2-й элемент аппроксимации с 2N шагов

do

{

yn = quickY1(f, x0, y0, xn, step);

step /= 2;

y2n = quickY2(f, x0, y0, xn, step);

error = Math.Abs(yn - y2n) / 3;

} while (error >= precision);

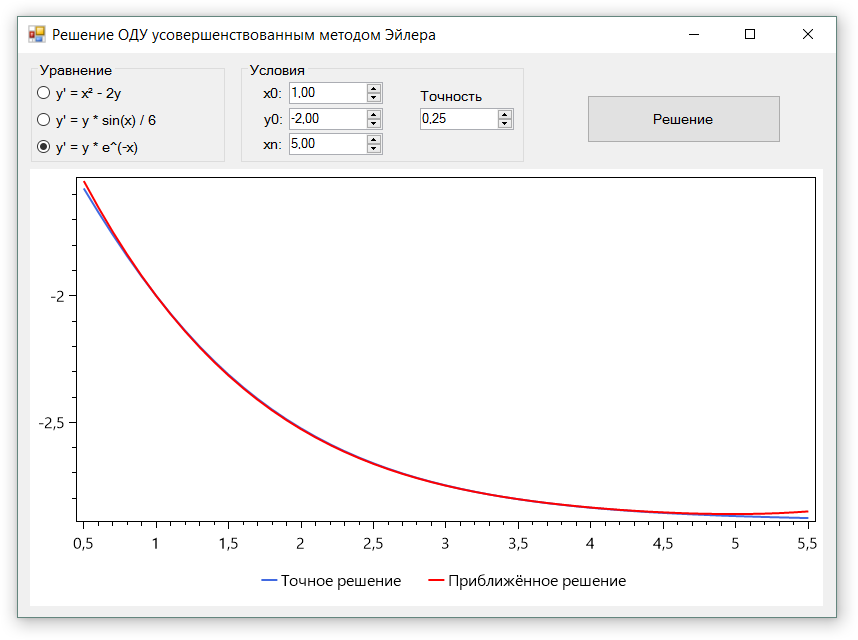
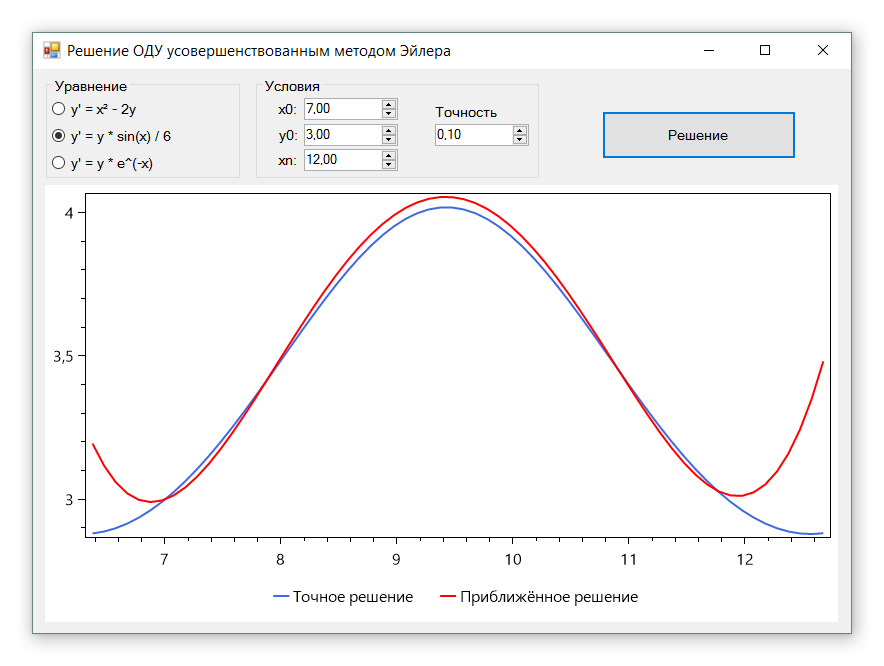
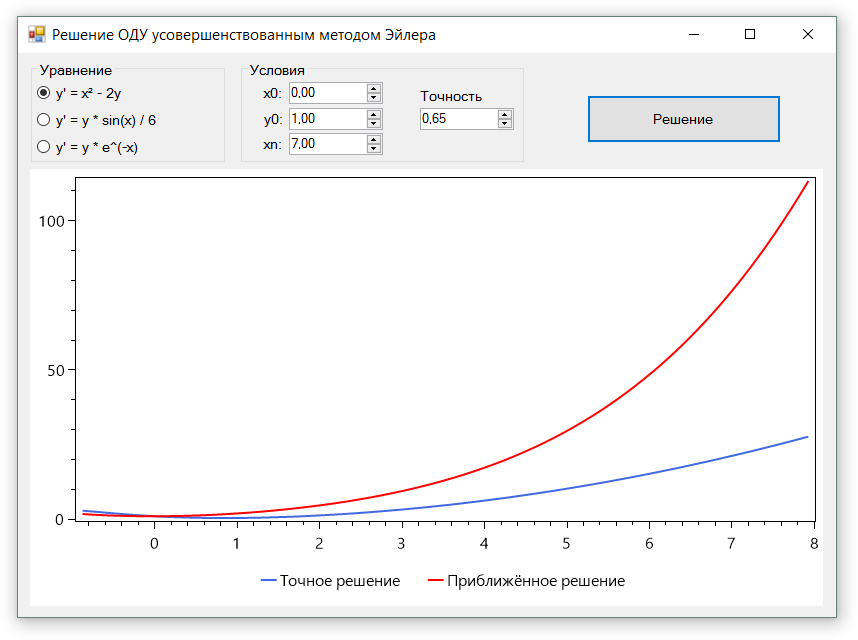
step \*= 2;

Point[] points = approximate(f, x0, y0, xn, step);

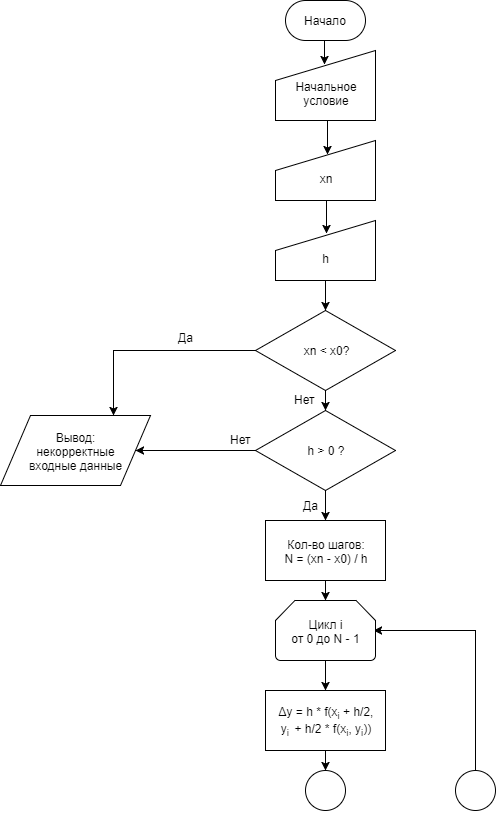
return interpolation(points);

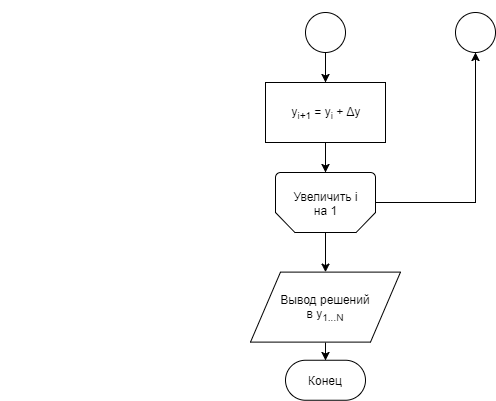
}

}

**Тестовые данные**

**Блок-схема**

****

****

**Вывод**

Усовершенствованный метод Эйлера точнее простого, так как его порядок точности – 2 (у простого – 1), но менее точен, чем метод Рунге-Кутты 4 порядка.

Усовершенствованный метод Эйлера относится к явным одношаговым методам, поэтому для него требуется больший объём вычислений, чем для многошаговых той же точности. Явные методы позволяют выразить yi через предыдущие значения yi-1, yi-2, …, в то время как неявные требуют решения дополнительных уравнений.