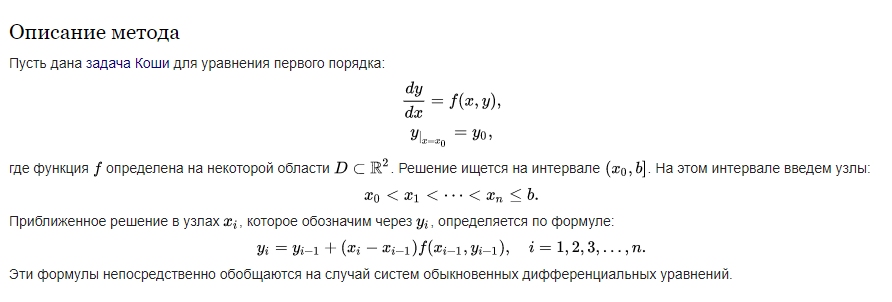
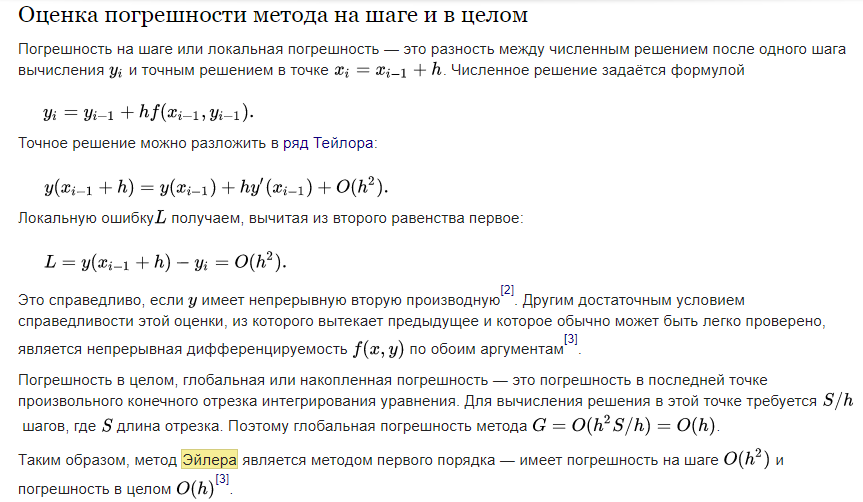
CАНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

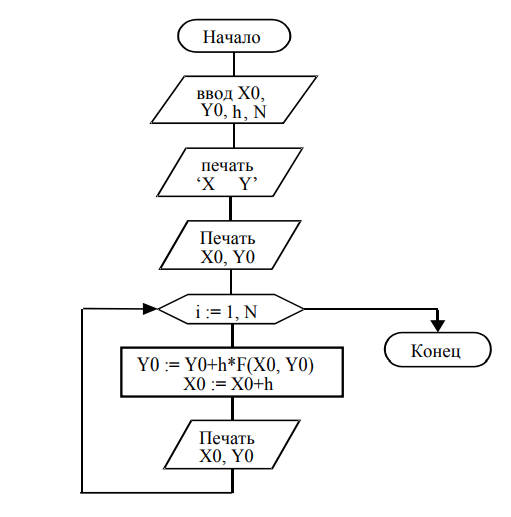
**Лабораторная работа по выч.математике №4**

2017 г.







 **Код программы:**

using System;

using System.Collections.Generic;

using System.Windows;

namespace LagrangeInterpol

{

internal static class KoshiSolver

{

public static Point[] RungeKutta(double x0, double y0, double xEnd, double accuracy, Func<double, double, double> f)

{

var h = Math.Pow(accuracy, 0.25);

h \*= 0.95;

var count = (int)Math.Truncate((xEnd - x0) / h);

return RungeKutta(x0, y0, h, count, f);

}

public static Point[] RungeKutta(double x0, double y0, double h, int count, Func<double, double, double> f)

{

var points = new List<Point>(count) { new Point(x0, y0) };

for (var i = 1; i < count; ++i)

{

var dy = h / 6;C:

var k1 = f(points[i - 1].X, points[i - 1].Y);

var k2 = f(points[i - 1].X + h / 2, points[i - 1].Y + (h \* k1) / 2);

var k3 = f(points[i - 1].X + h / 2, points[i - 1].Y + (h \* k2) / 2);

var k4 = f(points[i - 1].X + h, points[i - 1].Y + h \* k3);

dy \*= (k1 + 2 \* k2 + 2 \* k3 + k4);

points.Add(new Point(points[i - 1].X + h, points[i - 1].Y + dy));

}

return points.ToArray();

}

public static Point[] Euler(double x0, double y0, double xEndValue, double accuracy, Func<double, double, double> f)

{

var points = new List<Point>();

var continueFlag = true;

var y = y0;

var h = 0.1;

while (continueFlag)

{

double x;

for (x = x0 + h; x < xEndValue; x = x + h)

{

y = y + f(x, y) \* h;

points.Add(new Point(x, y));

}

try

{

if (points[points.Count - 2].Y - points[points.Count - 1].Y < accuracy)

{

continueFlag = false;

}

else

{

h = h / 2;

}

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

return points.ToArray();

}

public static Point[] Predictor\_and\_corrector(double x0, double y0, double xEndValue, double accuracy, Func<double, double, double> f)

{

var points = new List<Point>();

var continueFlag = true;

var y = y0;

var h = 0.1;

points.Add(new Point(x0, y));

while (continueFlag)

{

var x1 = x0 + h;

y = y + f(x1, y);

points.Add(new Point(x1, y));

double x;

for (x = x1 + h; x < xEndValue; x = x + h)

{

var index = points.Count - 1;

y = y + (h / 2) \* (f(x1, points[index].Y + 2 \* h \* f(x, y)) + f(x, y)) + ((points[index].Y + 2 \* h \* f(x, y) - y + (h / 2) \* (f(x1, points[index].Y + 2 \* h \* f(x, y)) + f(x, y))) / 5);

points.Add(new Point(x1, y));

}

try

{

if (points[points.Count - 2].Y - points[points.Count - 1].Y < accuracy)

{

continueFlag = false;

}

else

{

h = h / 2;

}

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

return points.ToArray();

}

public static Point[] Adams(double x0, double y0, double xEndValue, double accuracy,

Func<double, double, double> f)

{

var points = new List<Point>();

var continueFlag = true;

var y = y0;

var h = 0.1;

points.Add(new Point(x0,y0));

points.Add(new Point(x0, y0));

while (continueFlag)

{

var x1 = x0 + h;

double k1 = y + f(x0, y) \* h;

points.Add(new Point(x1, k1));

double x;

int count;

for (x = x1; x < xEndValue; x = x + h)

{

x1 = x + h;

count = points.Count;

y = y + h \* ((3 / 2) \* f(x, y) - (1 / 2) \* f(points[count - 3].X, points[count

- 3].Y));

points.Add(new Point(x1, y));

}

try

{

if (points[points.Count - 2].Y - points[points.Count - 1].Y < accuracy)

{

continueFlag = false;

}

else

{

h = h / 2;

}

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

return points.ToArray();

}

public static Point[] ImprovedEuler(double x0, double y0, double xEndValue, double accuracy, Func<double, double, double> f)

{

var points = new List<Point>();

var continueFlag = true;

var y = y0;

var h = 0.1;

while (continueFlag)

{

double x;

for (x = x0; x < xEndValue; x = x + h)

{

y = y + (0.5) \* (f(x, y) + f(x + h, y + f(x, y) \* h)) \* h;

points.Add(new Point(x + h, y));

}

try

{

if (points[points.Count - 2].X - points[points.Count - 1].X < accuracy)

{

continueFlag = false;

}

else

{

h = h / 2;

}

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

return points.ToArray();

}

public static Point[] Milne(double x0, double y0, double xEndValue, double accuracy, Func<double, double, double> f)

{

var points = new List<Point>();

var h = accuracy;

points.AddRange(RungeKutta(x0, y0, h, 4, f)); // get the first 4 points

h = 0.1;

bool converged;

do

{

converged = true;

var count = (int)((xEndValue - x0) / h);

if (points.Count != 4)

points.RemoveRange(4, points.Count - 4);

for (var i = 4; i < count; ++i)

{

var predictor = 4.0 \* h / 3.0;

predictor \*= 2 \* f(points[i - 3].X, points[i - 3].Y) +

f(points[i - 2].X, points[i - 2].Y) +

2 \* f(points[i - 1].X, points[i - 1].Y);

predictor += points[i - 4].Y;

var corrector = h / 3.0;

corrector \*= f(points[i - 1].X + h, predictor) +

4 \* f(points[i - 1].X, points[i - 1].Y) +

f(points[i - 2].X, points[i - 2].Y);

corrector += points[i - 2].Y;

var error = 1.0 / 29.0;

error \*= predictor - corrector;

if (Math.Abs(error) >= accuracy)

{

converged = false;

h /= 2;

break;

}

points.Add(new Point(points[i - 1].X + h, corrector));

}

} while (!converged);

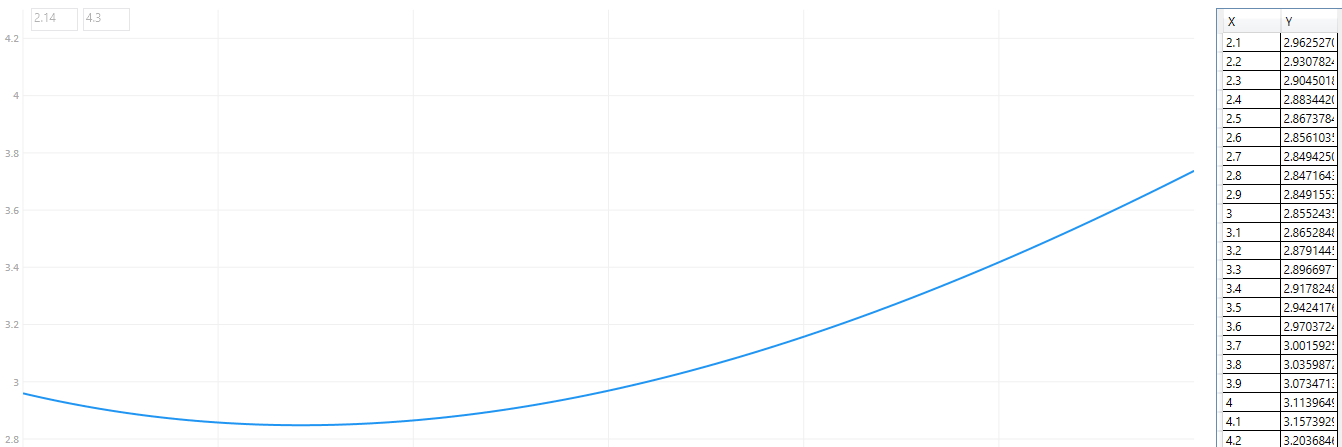
return points.ToArray();

}

}

}

Результаты программа:





Вывод . Метод Эйлера являлся исторически первым методом численного решения задачи Коши. [О. Коши](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%88%D0%B8,_%D0%9E%D0%B3%D1%8E%D1%81%D1%82%D0%B5%D0%BD_%D0%9B%D1%83%D0%B8) использовал этот метод для доказательства существования решения задачи Коши. Ввиду невысокой точности и вычислительной неустойчивости для практического нахождения решений задачи Коши метод Эйлера применяется редко. Однако в виду своей простоты метод Эйлера находит своё применение в теоретических исследованиях дифференциальных уравнений, задач [вариационного исчисления](https://ru.wikipedia.org/wiki/%D0%92%D0%B0%D1%80%D0%B8%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B8%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) и ряда других математических проблем.