# Лабораторная работа № 3 Разработка приложений для работы с графикой

**Цель лабораторной работы**

Изучить возможности Visual Studio по созданию простейших графических изображений, анимации и рисованию графиков.

**Постановка задачи**

Класс для работы с таймером Timer формирует в приложении повторяющиеся события. События повторяются с периодичностью, указанной в миллисекундах в свойстве Interval. Установка свойства Enabled в значение true запускает таймер.

Для форм в C# предусмотрен способ, позволяющий приложению при необходимости перерисовывать окно формы в любой момент времени. Когда вся клиентская область окна формы или часть этой области требует перерисовки, форме передается событие Paint. Все, что требуется от программиста, - это создать обработчик данного события, наполнив его необходимой функциональностью.

Для рисования линий и фигур, отображения текста, вывода изображений и т.д. нужно использовать объект Graphics. Этот объект предоставляет поверхность рисования и используется для создания графических изображения. Ниже представлены два этапа работы с графикой.

Создание или получение объекта Graphics.

Рисовать возможно на любом компоненте, но быстрее всего происходит отрисовка на **pictureBox-**е.

Добавим на форму pictureBox и установим свойство Anchor: Top, Left, Right, Bottom (рисунки 1 и 2).

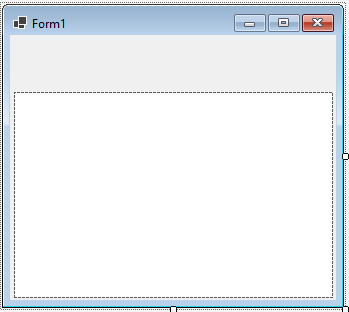


Рисунок 1 – Конструктор формы с PictureBox

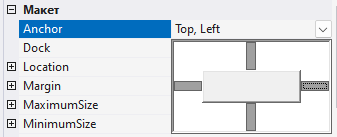


Рисунок 2 – Свойство Anchor у PictureBox

Рисование необходимо проводить в обработчике события Paint. Нарисуем круг радиуса 100 в середине полотна для рисования (рисунок 3).

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// Для рисования необходимо создать кисть Pen.

var pen = new Pen(Color.Red, 2);

int width = 100, height = 100;

var cx = (pictureBox.Width - width) / 2;

var cy = (pictureBox.Height - height) / 2;

e.Graphics.DrawEllipse(

pen, cx, cy, width, height);

}

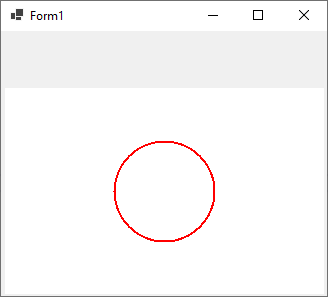


Рисунок 3 – Рисование круга в PictureBox

Нарисуем закрашенный прямоугольник шириной 100 и высотой 50 (рисунок 4). Здесь для рисования используется кисть и метод начинается не с Draw… а с Fill… (FillRectangle).

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// Для закрашенных фигур необходима кисть

var brush = new SolidBrush(Color.Red);

int width = 100, height = 50;

var cx = (pictureBox.Width - width) / 2;

var cy = (pictureBox.Height - height) / 2;

e.Graphics.FillRectangle(

brush, cx, cy, width, height);

}

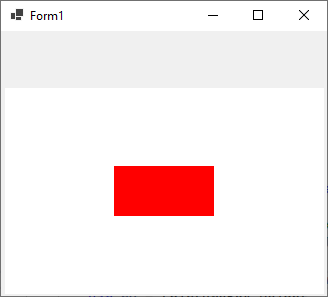


Рисунок 4 – Рисование закрашенного прямоугольника  
в PictureBox

Изменим код чтобы прямоугольник рисовался там, где была нажата ЛКМ (рисунок 5).

public partial class Form1 : Form

{

private int mouseX, mouseY; // Координаты мыши

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// Для закрашенных фигур необходима кисть

var brush = new SolidBrush(Color.Red);

int width = 100, height = 50;

var cx = mouseX - width / 2;

var cy = mouseY - height / 2;

e.Graphics.FillRectangle(

brush, cx, cy, width, height);

}

private void pictureBox\_MouseClick(object sender, MouseEventArgs e)

{

mouseX = e.X; // Считываем координаты мыши

mouseY = e.Y;

pictureBox.Refresh(); // Для вызова отрисовки

}

}

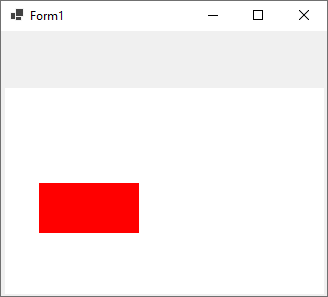


Рисунок 5 – Рисование закрашенного прямоугольника в PictureBox в точке нажатия левой кнопки мыши

Напишем код для рисования линий (рисунок 6):

public partial class Form1 : Form

{

private List<Point> points = new(); // Координаты точек

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

if (points.Count < 2) // Недостаточно точек

return;

var pen = new Pen(Color.Red, 2); e.Graphics.DrawLines(pen, points.ToArray());

}

private void pictureBox\_MouseClick(object sender, MouseEventArgs e)

{

points.Add(new(e.X, e.Y)); // Добавляем точку

pictureBox.Refresh();

}

}

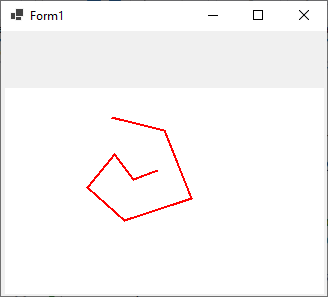


Рисунок 6 – Рисунок последовательности линий

Добавим возможность **выбора цвета** с помощью компонента ColorDialog:

private Color color = Color.Red; private void buttonColorDialog\_Click(object sender, EventArgs e)

{

var cd = new ColorDialog(); // Выбор цвета

if (cd.ShowDialog() == DialogResult.OK)

color = cd.Color;

}

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

if (points.Count < 2)

return;

var pen = new Pen(color, 2);

e.Graphics.DrawLines(pen, points.ToArray());

}

После нажатия на кнопку «Выбрать цвет» появляется диалоговое окно с палитрой (рисунок 7).

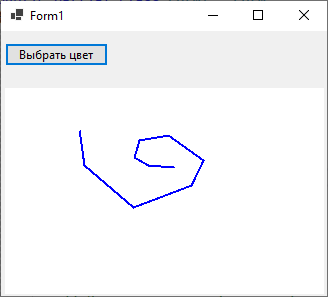
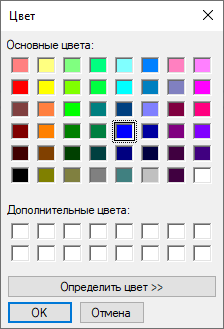
 

Рисунок 7 – Изменения цвета при рисовании

Имена большого количества методов, определенных в классе Graphics, начинаются с префикса Draw\* и Fill\*. Первые из них предназначены для рисования текста, линий и незакрашенных фигур (таких, как прямоугольные рамки), а вторые – для рисования закрашенных геометрических фигур.

Метод DrawLine рисует линию, соединяющую две точки с заданными координатами. У метода есть несколько перегруженных версий:

public void DrawLine(Pen, Point, Point);

public void DrawLine(Pen, PointF, PointF);

public void DrawLine(Pen, int, int, int, int);

public void DrawLine(Pen, float, float, float, float);

Первый параметр задает инструмент для рисования линии – перо. Перья создаются как объекты класса Pen, например:

Pen p = new Pen(Brushes.Black, 2);

Здесь создается черное перо толщиной 2 пикселя. При создании пера можно выбрать его цвет, толщину и тип линии, а также другие атрибуты.

Остальные параметры перегруженных методов DrawLine задают координаты соединяемых точек. Эти координаты могут быть заданы как объекты класса Point и PointF, а также в виде целых числе и числе с плавающей десятичной точкой.

В классах Point и PointF определены свойства X и Y, задающие, соответственно, координаты точки по горизонтальной и горизонтальной оси. При этом в классе Point эти свойства имеют целочисленные значения, а в классе PointF – значения с плавающей десятичной точкой.

Третий и четвертый варианты метода DrawLine позволяют задавать координаты соединяемых точек в виде двух пар чисел. Первая пара определяет координаты первой точки по горизонтальной и вертикальной оси, а вторая – координаты второй точки по этим же осям. Разница между третьим и четвертым методом заключается в использовании координат различных типов.

Метод DrawEllipse рисует эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров. При помощи метода DrawArc программа может нарисовать сегмент эллипса. Сегмент задается при помощи координат прямоугольной области, в которую вписан эллипс, а также двух углов, отсчитываемых в направлении против часовой стрелки. Первый угол Angle1 задает расположение одного конца сегмента, а второй Angle2 – расположение другого конца сегмента (рисунок 8).

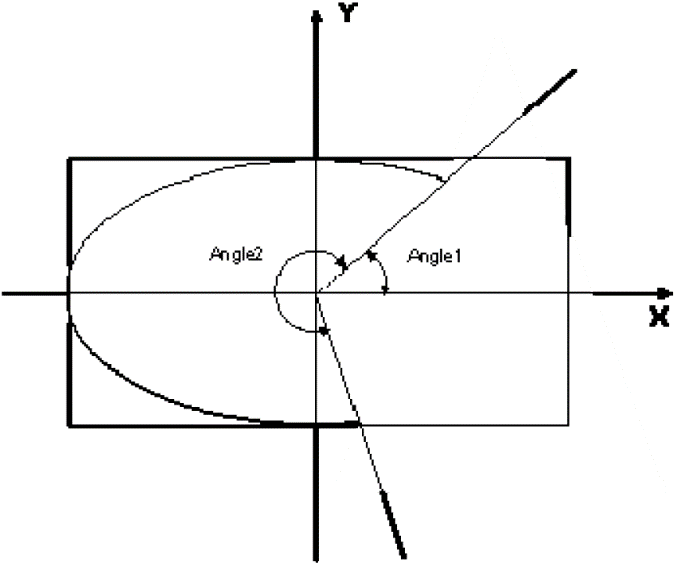


Рисунок 8 – Углы и прямоугольник, задающие сегмент эллипса

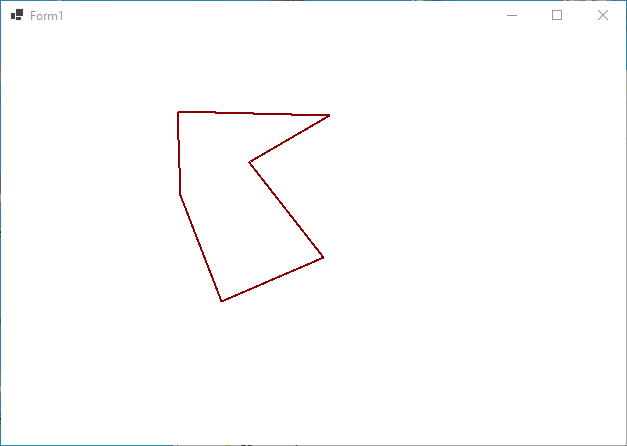


Рисунок 9 – Нарисованный многоугольник

**Создание анимации.**

Для создания простой анимации достаточно использовать таймер, при тике которого будут изменяться параметры изображения (например, координаты концов отрезка) и вызываться обработчик события Paint для рисования по новым параметрам. При таком подходе не надо заботиться об удалении старого изображения, ведь оно создается в окне заново.

Добавим таймер на форму, установим интервал 10 мс. И сразу включим **Enabled** = True. Определим основные переменные, такие как: скорость, размеры и т.д. При срабатывании таймера изменяем координаты прямоугольника, выполняем проверку на выход за границы области и рисуем.

public partial class Form1 : Form

{

private Color color = Color.Red;

private Point rectPoint = new Point(100, 100);

private int width = 100, height = 50;

private int speedX = 5, speedY = 5;

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

var brush = new SolidBrush(color);

var cx = rectPoint.X - width / 2;

var cy = rectPoint.Y - height / 2;

e.Graphics.FillRectangle(

brush, cx, cy, width, height);

}

private void timer\_Tick(object sender, EventArgs e)

{

rectPoint.X += speedX;

rectPoint.Y += speedY;

if (rectPoint.X <= 0 || rectPoint.X >= pictureBox.Width)

speedX \*= -1;

if (rectPoint.Y <= 0 || rectPoint.Y >= pictureBox.Height)

speedY \*= -1;

// Для вызова отрисовки

pictureBox.Refresh();

}

}

В результате на экране увидим двигающийся и отскакивающий от границ прямоугольник.

Движение по траектории реализуется аналогично. Для реализации движения по прямой нужно увеличивать переменные, являющиеся узловыми точками, на определенные константы: в приведенном выше примере это переменные x2 и y2. Для задания более сложной траектории можно использовать различные параметрические кривые.

В случае движения на плоскости обычно изменению подвергается один параметр. Рассмотрим пример реализации движения окружности по декартову листу. Декартов лист (рисунок 10) – это плоская кривая третьего порядка, удовлетворяющая уравнению в прямоугольной системе . Параметр 3*a* определяется как диагональ квадрата, сторона которого равна наибольшей хорде петли.

При переходе к параметрическому виду получаем:



где .

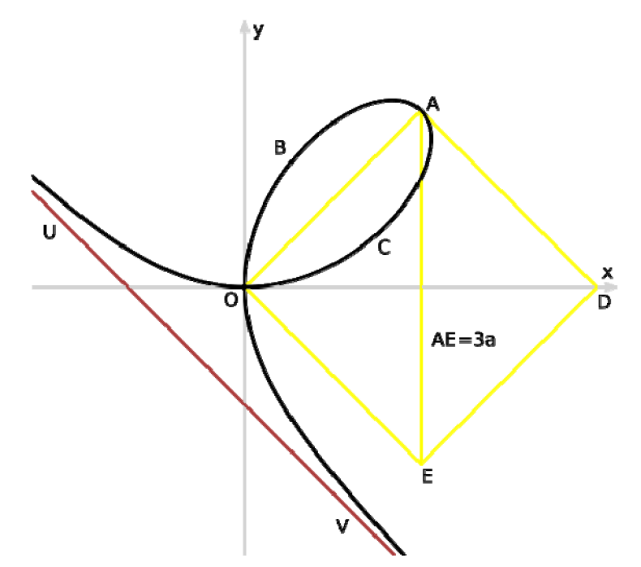
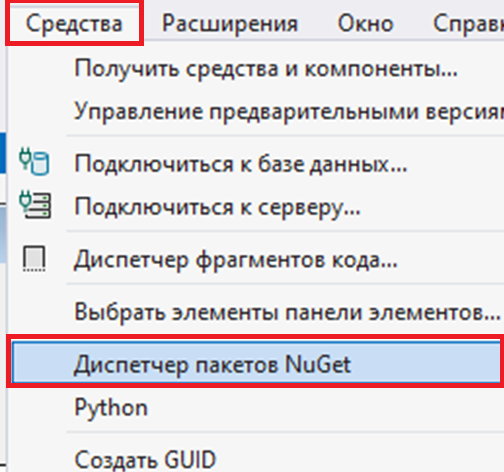


Рисунок 10 – Декартов лист

**Построение графиков с помощью ScottPlot.**

Рассмотрим пример добавления к проекту библиотеки для построения графиков ScottPlot. Необходимо открыть диспетчер пакетов **NuGet** (рисунок 11).



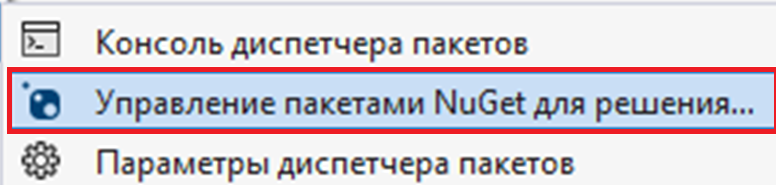


Рисунок 11 – Установка пакетов с помощью NuGet

Необходимо перейти на вкладку «Обзор» найти библиотеки ScottPlot и ScottPlot.WinForms, и нажать «Установить» (рисунок 12).

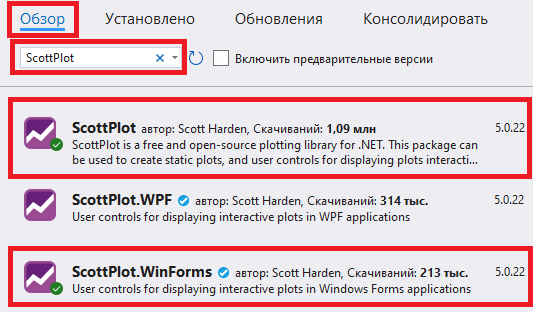


Рисунок 12 – Установка пакета ScottPlot

Элемент для построения графиков (FormsPlot) появится в панели элементов (рисунок 13). После установки пакета **возможно придется перезапустить приложение** (если появляется ошибка при добавлении элемента FormsPlot).

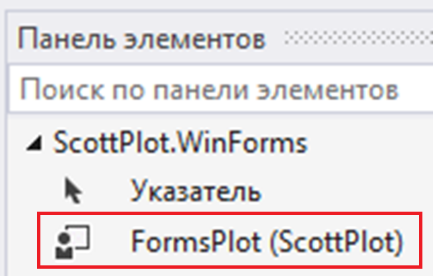


Рисунок 13 – Отображение ScottPlot в панели элементов

Добавим подписи осей, название графика и отобразим что получилось (рисунок 14).

formsPlot1.Plot.XLabel("X, м");

formsPlot1.Plot.YLabel("Y, м");

formsPlot1.Plot.Title("График");

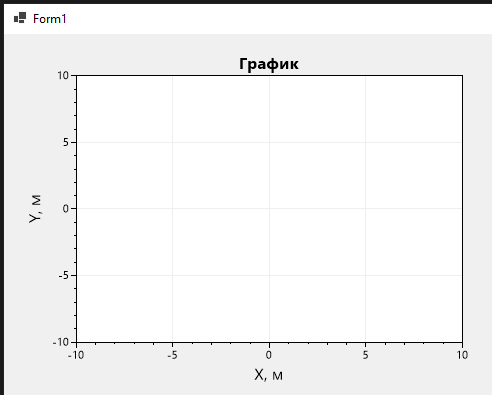


Рисунок 14 – Пустой график ScottPlot на форме

Построим график функции *f(x) = x*2 на участке от 0 до 100 (рисунок 15).

var xs = new List<double>();

var ys = new List<double>();

for (int i = 0; i <= 100; i++)

{

xs.Add(i);

ys.Add(i \* i);

}

var scatter = formsPlot1.Plot.Add.Scatter(xs, ys);

// Убираем маркеры (точки) и оставляем линию

scatter.MarkerStyle = ScottPlot.MarkerStyle.None;

scatter.Label = "x^2"; // Подписии линии графика

formsPlot1.Plot.ShowLegend();

plot.Axes.AutoScale(); // Автоматически приближать к области графика

formsPlot.Refresh(); // Вызов метода перерисовки

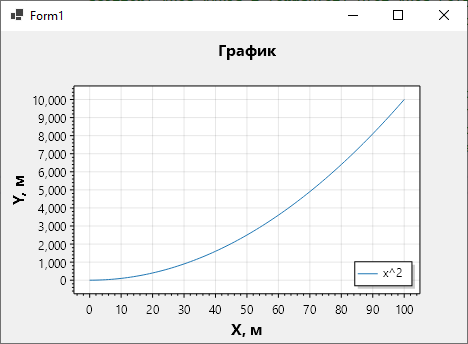


Рисунок 15 – Отображение графика функции на форме

Основное преимущество ScottPlot это **интерактивность**, график можно масштабировать и перемещаться по графику, также он сам определяет интервалы подписей у осей.

**Чтобы переместиться по графику** нужно зажать ЛКМ и двигать мышью, чтобы приблизиться (рисунок 16) или отдалиться – вращать колесико мыши, также чтобы приблизить некоторую область можно зажать колесико и выделить прямоугольник. Чтобы вернуться к исходному виду нужно нажать на колесико.

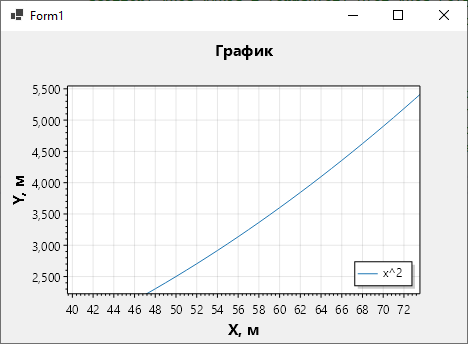


Рисунок 16 – Приближение участка графика

ScottPlot поддерживает рисование по второстепенным осям, это полезно, когда 2 набора данных сильно отличаются по значениям, но их требуется представить на одном графике (Например, давление и температура).

var xs = new List<double>();

var ys = new List<double>();

var ys2 = new List<double>();

for (int i = 0; i <= 100; i++)

{

xs.Add(i);

ys.Add(i \* i);

ys2.Add(Math.Sin(i));

}

Отобразим *x*2 на основной оси, а sin(*x*) на второстепенной оси *Oy.* Также сделаем график синуса пунктирным (рисунок 17).

var scatter1 = formsPlot1.Plot.Add.Scatter(xs, ys);

scatter1.Axes.XAxis = formsPlot1.Plot.Axes.Bottom;

scatter1.Axes.YAxis = formsPlot1.Plot.Axes.Left;

scatter1.MarkerStyle = ScottPlot.MarkerStyle.None;

scatter1.Label = "x^2";

var scatter2 = formsPlot1.Plot.Add.Scatter(xs, ys2);

scatter2.Axes.XAxis = formsPlot1.Plot.Axes.Bottom;

scatter2.Axes.YAxis = formsPlot1.Plot.Axes.Right;

scatter2.MarkerStyle = ScottPlot.MarkerStyle.None;

scatter2.LinePattern = ScottPlot.LinePattern.Dashed;

scatter2.Label = "sin(x)";

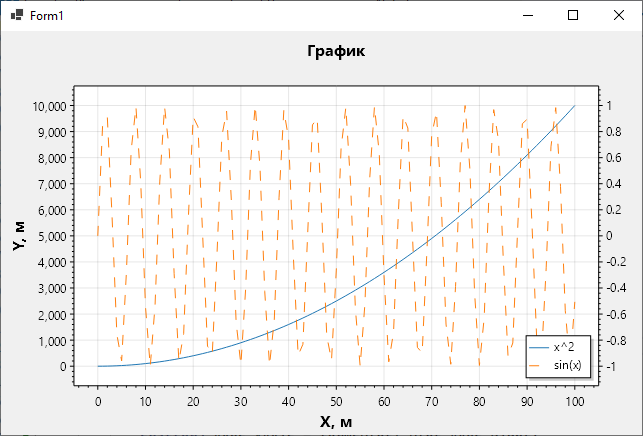


Рисунок 17 – Отображение графиков  
на главной и второстепенной осях

Рассмотрим пример построения гистограмм (рисунок 18):

var plot = formsPlot1.Plot;

double[] xs1 = { 1, 2, 3, 4 };

double[] ys1 = { 5, 10, 7, 13 };

var bars1 = plot.Add.Bars(xs1, ys1);

bars1.Label = "Alpha";

double[] xs2 = { 6, 7, 8, 9 };

double[] ys2 = { 7, 12, 9, 15 };

var bars2 = plot.Add.Bars(xs2, ys2);

bars2.Label = "Beta";

plot.ShowLegend(Alignment.UpperLeft);

// Убираем отсут от нижней границы

plot.Axes.Margins(bottom: 0);

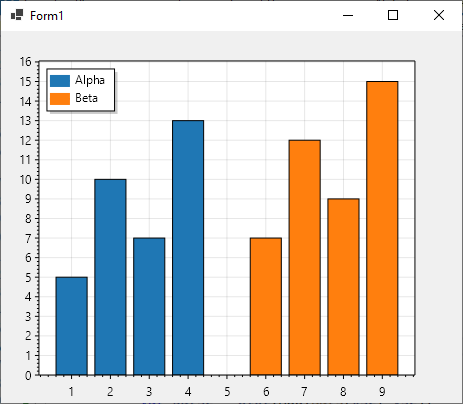


Рисунок 18 – Отображение гистограммы

Рассмотрим пример построения круговой диаграммы (рисунок 19):

var plot = formsPlot1.Plot;

List<PieSlice> slices = new()

{

new PieSlice(){ Value = 5, FillColor = Colors.Red, Label = "Red" },

new PieSlice(){ Value = 2, FillColor = Colors.Orange, Label ="Orange"},

new PieSlice(){ Value = 8, FillColor = Colors.Gold, Label ="Yellow"},

new PieSlice(){ Value = 4, FillColor = Colors.Green, Label ="Green"},

new PieSlice(){ Value = 8, FillColor = Colors.Blue, Label ="Blue" }};

var pie = plot.Add.Pie(slices);

pie.ExplodeFraction = 0.1;

plot.ShowLegend();

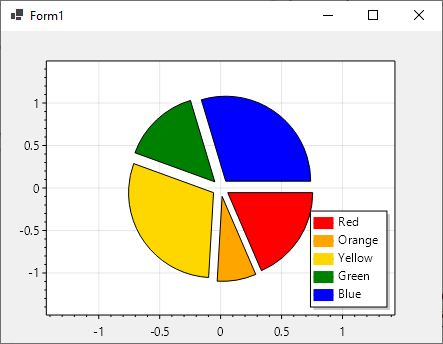


Рисунок 19 – Отображение гистограммы

**Анимация.**

Здесь необходим компонент Timer. Требуется добавить 2 кнопки – для запуска и остановки движения.

public partial class Form1 : Form

{

private double x = 0, speed = 0.1;

private List<double> xs = new();

private List<double> ys = new();

public Form1()

{

InitializeComponent();

}

private void buttonStart\_Click(object sender, EventArgs e)

{

timer.Start();

}

private void buttonStop\_Click(object sender, EventArgs e)

{

timer.Stop();

}

private double getF(double x) => Math.Sin(x);

private void timer\_Tick(object sender, EventArgs e)

{

x += speed;

xs.Add(x);

ys.Add(getF(x));

var plot = formsPlot.Plot;

plot.Clear();

var scatter = plot.Add.Scatter(xs, ys);

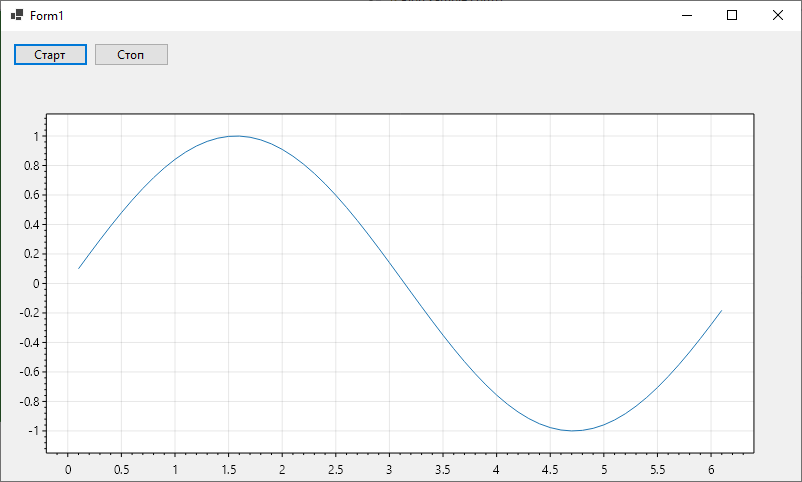
scatter.MarkerStyle = MarkerStyle.None;

plot.Axes.AutoScale();

formsPlot.Refresh();

}

}

****

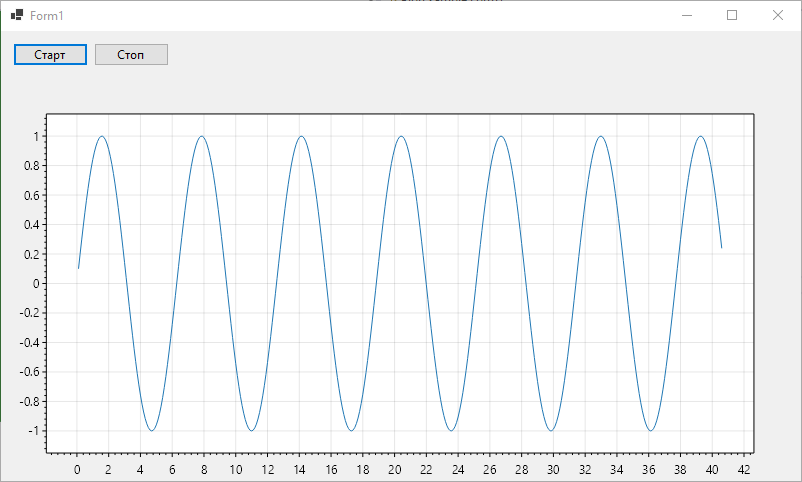
****

Рисунок 20 – Отображение анимации синусоиды

Примеры построения графиков и работы в ScottPlot приводятся в документации на сайте: <https://scottplot.net/cookbook/5.0/>

**Задание на лабораторную работу**

1. Разработать Windows-приложение в соответствии с вариантом задания, соблюдая принципы объектно-ориентированного программирования (ООП) и лучшие практики проектирования программного обеспечения.
2. Заголовок окна приложения должен содержать ФИО исполнителя, группу и номер варианта.
3. В программе должна быть предусмотрена обработка исключений, возникающих из-за ошибочного ввода пользователя.
4. Написать XML комментарии, поясняющие работу программы.

**Варианты заданий на лабораторную работу**

**Задание 1.**

* 1. Создайте программу, показывающую пульсирующее сердце.
  2. Создайте приложение, отображающее вращающийся винт самолета.
  3. Разработайте программу анимации двигающегося человечка.
  4. Создайте программу, показывающую движение окружности по синусоиде.
  5. Создайте приложение, отображающее движение окружности по спирали.
  6. Разработайте программу анимации падения снежинки.
  7. Создайте программу, показывающую скачущий мячик.
  8. Создайте приложение, отображение движение окружности вдоль границы окна. Учтите возможность изменения размеров окна.
  9. Разработайте программу анимации летающего бумеранга.
  10. Создайте программу, показывающую падение нескольких звезд одновременно.
  11. Создайте приложение, отображающее хаотичное движение звезды в окне.
  12. Разработайте программу анимации взлета ракеты. Старт осуществляется по нажатию специальной «красной» кнопки.
  13. Создайте программу, показывающую движение окружности вдоль многоугольника. Число вершин вводится пользователем до анимации.
  14. Создайте приложение, отображающее броуновское движение молекулы в окне.
  15. Разработайте программу анимации движения планет в Солнечной системе.
  16. Создайте программу, показывающую движение квадратика по траектории, состоящей из 100 точек, хранящихся в специальном массиве.
  17. Создайте приложение, имитирующее механические часы.
  18. Разработайте программу анимации падения нескольких листков с дерева. Движение не должно быть линейным.
  19. Создайте программу. показывающую движение окружности по спирали с плавно изменяющейся скоростью.
  20. Создайте приложение, отображающее движение автомобиля с вращающимися колесами.

**Задание 2.**

Необходимо построить 2D графики в среде MS Visual Studio на языке C# (компонент ScottPlot).

1. Построить график функции *y=f(x)*. Должны быть подписаны оси координат с единицами измерения, отображены линии сеток. Предусмотреть возможность задания цвета и толщины графика.
2. **Дополнительно.** Реализовать возможность установки (и удаления) маркера на график, предусмотреть ввод имени маркера. Маркер должен появляться в легенде.
3. **Дополнительно.** Реализовать возможность отображения значения функции рядом с мышью при наведении мыши на график.

Переменные являются осями координат для графиков, параметры вводятся с формы.

|  |  |
| --- | --- |
| **№ Варианта** | **График** |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |

**Задание 3.**

Используя метод CubicSpline.InterpolateNatural найти промежуточные значения табличных данных и построить график функции.

В качестве данных используем таблицу стандартной атмосферы. Независимая переменная (*x*) – высота, м. Зависимая переменная (*y*) – по варианту.

Варианты:

1. температура,
2. скорость звука,
3. давление,
4. плотность,
5. кинематическая вязкость.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Высота, H, м | Температура, T, К | Скорость звука, a, м/с | Давление, P, Па | Плотность, ρ, кг/м3 | Кинематическая вязкость, ν, м2/с |
| -2000 | 301,2 | 347,9 | 127783 | 1,4782 | 1,2525⋅10−5 |
| -1500 | 297,9 | 346,0 | 120696 | 1,4114 | 1,3009⋅10−5 |
| -1000 | 294,7 | 344,1 | 113931 | 1,3470 | 1,3516⋅10−5 |
| -500 | 291,4 | 342,2 | 107478 | 1,2849 | 1,4048⋅10−5 |
| 0 | 288,2 | 340,3 | 101330 | 1,2250 | 1,46⋅10−5 |
| 500 | 284,9 | 338,4 | 95464 | 1,1673 | 1,52⋅10−5 |
| 1000 | 281,7 | 336,4 | 89877 | 1,1117 | 1,58⋅10−5 |
| 1500 | 278,4 | 334,5 | 84559 | 1,0581 | 1,65⋅10−5 |
| 2000 | 275,2 | 332,5 | 79499 | 1,0065 | 1,71⋅10−5 |
| 2500 | 271,9 | 330,6 | 74690 | 0,9569 | 1,79⋅10−5 |
| 3000 | 268,7 | 328,6 | 70123 | 0,9093 | 1,86⋅10−5 |
| 4000 | 262,2 | 324,6 | 61661 | 0,8194 | 2,03⋅10−5 |
| 5000 | 255,7 | 320,6 | 54052 | 0,7365 | 2,21⋅10−5 |
| 6000 | 249,2 | 316,5 | 47217 | 0,6601 | 2,42⋅10−5 |
| 7000 | 242,7 | 312,3 | 41106 | 0,59 | 2,65⋅10−5 |
| 8000 | 236,2 | 308,1 | 35653 | 0,5258 | 2,9⋅10−5 |
| 9000 | 229,7 | 303,9 | 30801 | 0,4671 | 3,2⋅10−5 |
| 10 000 | 223,3 | 299,6 | 26500 | 0,4135 | 3,53⋅10−5 |
| 11 000 | 216,8 | 295,2 | 22700 | 0,3648 | 3,9⋅10−5 |
| 12 000 | 216,7 | 295,1 | 19399 | 0,3119 | 4,56⋅10−5 |
| 14 000 | 216,7 | 295,1 | 14170 | 0,2279 | 6,24⋅10−5 |
| 16 000 | 216,7 | 295,1 | 10353 | 0,1665 | 8,54⋅10−5 |
| 18 000 | 216,7 | 295,1 | 7565 | 0,1216 | 1,17⋅10−4 |
| 20 000 | 216,7 | 295,1 | 5529 | 0,0889 | 1,6⋅10−4 |
| 24 000 | 220,6 | 297,7 | 2971 | 0,0469 | 3,07⋅10−4 |
| 28 000 | 224,5 | 300,4 | 1616 | 0,0251 | 5,84⋅10−4 |
| 32 000 | 228,5 | 303 | 889 | 0,0136 | 1,1⋅10−3 |
| 36 000 | 239,3 | 310,1 | 499 | 7,26⋅10−3 | 2,13⋅10−3 |
| 40 000 | 250,4 | 317,2 | 287 | 4,00⋅10−3 | 4,01⋅10−3 |
| 50 000 | 270,7 | 329,8 | 80 | 1,03⋅10−3 | 0,0166 |
| 60 000 | 247 | 315,1 | 22 | 3,00⋅10−4 | 0,0511 |
| 80 000 | 198,6 | 282,5 | 1 | 1,85⋅10−5 | 0,716 |

**Пример разработки приложения**

**Задание.** Рассмотрим построение графика

.

**Шаг 1: Установка необходимых библиотек**

Перед началом работы необходимо установить два NuGet пакета:

ScottPlot - основная библиотека для построения графиков

ScottPlot.WinForms - компоненты для интеграции с Windows Forms

**Шаг 2: Подготовка главной формы (рисунок 21)**

На форме должны быть расположены следующие элементы:

* Элемент FormsPlot (из ScottPlot.WinForms) для отображения графика
* NumericUpDown элементы для настройки параметров:
  + fromNumeric - начальное значение x
  + toNumeric - конечное значение x
  + pointsNumeric - количество точек
  + widthNumeric - толщина линии графика
* Кнопка setColorButton для выбора цвета линии

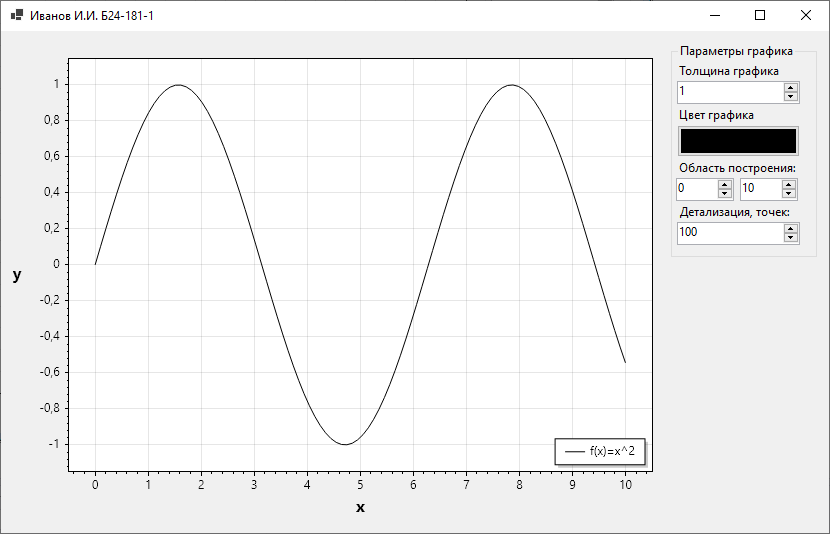


Рисунок 21 – Главное окно программы для построения графика

**Шаг 3: Основной функционал формы (MainForm)**

1. **Инициализация**:
   * В конструкторе создаются экземпляры ScatterPlotter и MarkerPlotter
   * Настраивается отображение легенды на графике
2. **Загрузка формы**:
   * Устанавливаются подписи осей X и Y
   * Вызывается метод BuildPlot() для первоначального построения графика
3. **Функция для графика**:
   * Метод f(double x) определяет функцию для построения (в примере - sin(x))
4. **Обработчики событий**:
   * Изменение параметров (fromNumeric, toNumeric, pointsNumeric) вызывает перестроение графика
   * Изменение толщины линии (widthNumeric) обновляет отображение
   * Кнопка выбора цвета (setColorButton) открывает диалог выбора цвета
5. **Конфигурация графиков**:
   * ScatterConfigurate - настраивает внешний вид линии графика
   * MarkerConfigurate - настраивает внешний вид маркера
6. **Построение графика**:
   * BuildPlot() создает данные для графика с помощью DataBuilder
   * formsPlot\_MouseDown обрабатывает клики по графику для отображения маркера в ближайшей точке

public partial class MainForm : Form

{

    private ScottPlot.Color lineColor = Colors.Black;

    private readonly ScatterPlotter scatterPlotter;

    private readonly MarkerPlotter markerPlotter;

    public MainForm()

    {

        InitializeComponent();

        formsPlot.Plot.ShowLegend();

        scatterPlotter = new ScatterPlotter(formsPlot);

        scatterPlotter.Configuration += ScatterConfigurate;

        markerPlotter = new MarkerPlotter(formsPlot);

        markerPlotter.Configuration += MarkerConfigurate;

    }

    private void MainForm\_Load(object sender, EventArgs e)

    {

        formsPlot.Plot.Axes.Bottom.Label.Text = "x";

        formsPlot.Plot.Axes.Left.Label.Text = "y";

        formsPlot.Plot.Axes.Left.Label.Rotation = 0;

        BuildPlot();

    }

    private double f(double x) => Math.Sin(x);

    private void widthNumeric\_ValueChanged(object sender, EventArgs e)

    {

        scatterPlotter.Draw();

    }

    private void setColorButton\_Click(object sender, EventArgs e)

    {

        var cd = new ColorDialog();

        if (cd.ShowDialog() == DialogResult.OK)

        {

            lineColor = ScottPlot.Color.FromColor(cd.Color);

            setColorButton.BackColor = cd.Color;

        }

        scatterPlotter.Draw();

    }

    private void fromNumeric\_ValueChanged(object sender, Even-tArgs e)

    {

        if (fromNumeric.Value > toNumeric.Value)

            toNumeric.Value = fromNumeric.Value;

        BuildPlot();

    }

    private void toNumeric\_ValueChanged(object sender, Even-tArgs e)

    {

        if (toNumeric.Value < fromNumeric.Value)

            fromNumeric.Value = toNumeric.Value;

        BuildPlot();

    }

    private void pointsNumeric\_ValueChanged(object sender, EventArgs e)

    {

        BuildPlot();

    }

    private void ScatterConfigurate(Scatter scatter)

    {

        scatter.LineWidth = (int)widthNumeric.Value;

        scatter.LineColor = lineColor;

        scatter.MarkerStyle = MarkerStyle.None;

        scatter.LegendText = "f(x)=x^2";

    }

    private void MarkerConfigurate(Marker marker)

    {

        marker.MarkerColor = Colors.Red;

        marker.LegendText = $"x: {marker.X:0.###} y: {mark-er.Y:0.###}";

    }

    private void BuildPlot()

    {

        var from = (double)fromNumeric.Value;

        var to = (double)toNumeric.Value;

        var pointsCount = (int)pointsNumeric.Value;

        var builder = new DataBuilder(f, new DataRange(from, to), pointsCount);

        scatterPlotter.UpdateData(builder.Build());

    }

    private void formsPlot\_MouseDown(object sender, MouseEven-tArgs e)

    {

        if (scatterPlotter.Scatter == null)

            return;

        var mousePixel = new Pixel(e.Location.X, e.Location.Y);

        var mouseLocation = formsPlot.Plot.GetCoordinates(mousePixel);

        var nearest = scatterPlot-ter.Scatter.Data.GetNearest(mouseLocation, formsPlot.Plot.LastRender);

        if (nearest.IsReal)

        {

            markerPlotter.SetPosition(nearest.X, nearest.Y);

        }

    }

}

**Шаг 4: Вспомогательные классы**

DataBuilder

Генерирует данные для построения графика:

* DataRange - запись для хранения диапазона значений
* DataXY - запись для хранения пар значений X и Y
* Метод Build() создает равномерно распределенные точки в заданном диапазоне и вычисляет значения функции в этих точках

public record DataRange(double Min, double Max);

public record DataXY(ICollection<double> X, ICollec-tion<double> Y);

public class DataBuilder

{

    public DataBuilder(

        Func<double, double> function,

        DataRange range,

        int pointsCount)

    {

        Function = function;

        Range = range;

        PointsCount = pointsCount;

    }

    public Func<double, double> Function { get; set; }

    public DataRange Range { get; set; }

    public int PointsCount { get; set; }

    public DataXY Build()

    {

        var xs = new double[PointsCount];

        var ys = new double[PointsCount];

        var dx = (Range.Max - Range.Min) / (PointsCount - 1);

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

        {

            xs[i] = Range.Min + i \* dx;

            ys[i] = Function(xs[i]);

        }

        return new DataXY(xs, ys);

    }

}

Создадим класс для построения графика ScatterPlotter

Он отвечает за отображение линии графика:

* UpdateData() - обновляет данные графика
* Draw() - перерисовывает график с текущими параметрами
* Позволяет настраивать внешний вид через событие Configuration

public class ScatterPlotter

{

    private readonly FormsPlot formsPlot;

    private DataXY? data;

    public Scatter? Scatter { get; private set; }

    public Action<Scatter>? Configuration { get; set; }

    public ScatterPlotter(FormsPlot formsPlot)

    {

        this.formsPlot = formsPlot;

    }

    public void Draw()

    {

        if (data == null)

            throw new ArgumentException("Не заданы данные для графика");

        if (Scatter != null)

            formsPlot.Plot.Remove(Scatter);

        Scatter = formsPlot.Plot.Add.Scatter(data.X.ToArray(), data.Y.ToArray());

        Configuration?.Invoke(Scatter);

        formsPlot.Refresh();

    }

    public void UpdateData(DataXY data)

    {

        this.data = data;

        Draw();

        formsPlot.Plot.Axes.AutoScale();

    }

}

Создадим класс для добавления маркера на график MarkerPlotter

Он отвечает за отображение маркера на графике:

* SetPosition() - устанавливает позицию маркера
* Draw() - перерисовывает маркер
* Позволяет настраивать внешний вид через событие Configuration

public class MarkerPlotter

{

    private readonly FormsPlot formsPlot;

    private double x;

    private double y;

    public Marker? Marker { get; private set; }

    public Action<Marker>? Configuration { get; set; }

    public MarkerPlotter(FormsPlot formsPlot)

    {

        this.formsPlot = formsPlot;

    }

    public void Draw()

    {

        if (Marker != null)

            formsPlot.Plot.Remove(Marker);

        Marker = formsPlot.Plot.Add.Marker(x, y);

        Configuration?.Invoke(Marker);

        formsPlot.Refresh();

    }

    public void SetPosition(double x, double y)

    {

        this.x = x;

        this.y = y;

        Draw();

    }

}

**Задание.**

Пусть данные метеоизмерений заданы таблицей:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y, м | 0 | 1 | 2 | 5 | 10 | 15 | 25 | 50 | 100 |
| T, C | 30 | 30,5 | 30,7 | 30 | 29 | 28 | 27,5 | 27 | 25 |

**Шаг 1: Установка необходимых библиотек**

Перед началом работы необходимо установить NuGet пакеты:

1. ScottPlot - для визуализации данных
2. ScottPlot.WinForms - компоненты для интеграции с Windows Forms
3. MathNet.Numerics - для математических вычислений и интерполяции

**Шаг 2: Подготовка главной формы (рисунок 22)**

На форме должны быть расположены следующие элементы:

* Элемент FormsPlot (из ScottPlot.WinForms) для отображения графика
* Кнопка loadDataButton для загрузки данных из файла

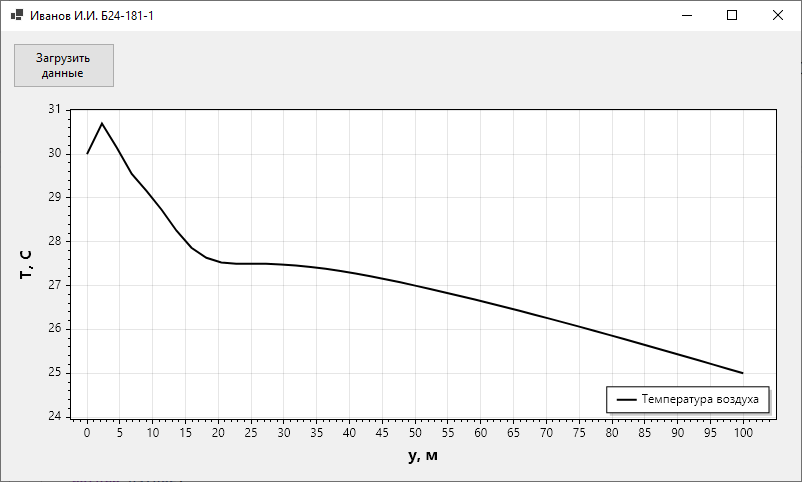


Рисунок 22 – Главное окно программы

**Шаг 3: Основной функционал формы (MainForm)**

Выделим отдельно методы для считывания данных, вычисления промежуточных значений и построения графика

1. Инициализация формы

* В методе MainForm\_Load устанавливаются подписи осей:
  + Ось X: "y, м" (высота в метрах)
  + Ось Y: "T, С" (температура в градусах Цельсия)
* Включается отображение легенды на графике

2. Загрузка данных

* При нажатии на кнопку loadDataButton открывается диалог выбора файла (поддерживаются форматы .txt и .csv)
* Файл должен содержать две строки:
  1. Первая строка - значения высот (в метрах)
  2. Вторая строка - соответствующие значения температур (в градусах Цельсия)
* Данные разделяются пробелами или табуляциями

3. Обработка данных

* Метод ParseLine разбивает строку на отдельные числовые значения
* Создается объект PlotData, содержащий пары значений высота-температура

4. Интерполяция данных

* Константа INTERPOLATION\_FACTOR = 5 определяет, во сколько раз увеличится количество точек после интерполяции
* Метод Interpolate использует натуральный кубический сплайн (CubicSpline.InterpolateNatural) для создания плавного графика
* Создаются новые массивы данных с увеличенным количеством точек

5. Визуализация данных

* Метод DrawTemperaturesScatter отображает интерполированные данные на графике:
  + Черная линия толщиной 2 пикселя
  + Без маркеров в точках данных
  + Подпись в легенде: "Температура воздуха"
* График автоматически масштабируется под данные

using MathNet.Numerics.Interpolation;

using ScottPlot;

namespace WinFormsApp2;

public partial class MainForm : Form

{

    private const int INTERPOLATION\_FACTOR = 5;

    public MainForm()

    {

        InitializeComponent();

    }

    private void MainForm\_Load(object sender, EventArgs e)

    {

        formsPlot.Plot.Axes.Bottom.Label.Text = "y, м";

        formsPlot.Plot.Axes.Left.Label.Text = "T, С";

        formsPlot.Plot.ShowLegend();

    }

    private void loadDataButton\_Click(object sender, EventArgs e)

    {

        var ofd = new OpenFileDialog

        {

            Filter = "Text Files|\*.txt;\*.csv"

        };

        if (ofd.ShowDialog() != DialogResult.OK)

            return;

        var lines = File.ReadAllLines(ofd.FileName);

        var heights = ParseLine(lines[0]);

        var temperatures = ParseLine(lines[1]);

        var input = new PlotData(heights, temperatures);

        var interpolated = Interpolate(input, input.X.Count \* INTERPOLATION\_FACTOR);

        DrawTemperaturesScatter(interpolated);

    }

    private PlotData Interpolate(PlotData data, int interpolatedPointsCount)

    {

        var spline = CubicSpline.InterpolateNatural(data.X, data.Y);

        var dx = (data.X.Last() - data.X.First()) / (interpolatedPointsCount - 1);

        var interpolatedX = new double[interpolatedPointsCount];

        var interpolatedY = new double[interpolatedPointsCount];

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

        {

            interpolatedX[i] = data.X.First() + i \* dx;

            interpolatedY[i] = spline.Interpolate(interpolatedX[i]);

        }

        return new PlotData(interpolatedX, interpolatedY);

    }

    private void DrawTemperaturesScatter(PlotData tempera-tures)

    {

        var scatter = formsPlot.Plot.Add.Scatter(temperatures.X.ToArray(), temperatures.Y.ToArray());

        scatter.LineColor = Colors.Black;

        scatter.MarkerShape = MarkerShape.None;

        scatter.LineWidth = 2;

        scatter.LegendText = "Температура воздуха";

        formsPlot.Plot.Axes.AutoScale();

        formsPlot.Refresh();

    }

    private double[] ParseLine(string line)

    {

        var strValues = line.Split(new[] { ' ', '\t' });

        var values = new double[strValues.Length];

        for (int i = 0; i < values.Length; i++)

        {

            values[i] = double.Parse(strValues[i]);

        }

        return values;

    }

}

**Шаг 4: Вспомогательный класс PlotData**

* Простая запись (record) для хранения пар значений X (высота) и Y (температура)
* Позволяет удобно передавать данные между методами

public record PlotData(ICollection<double> X, ICollection<double> Y);

**Контрольные вопросы**

1. Что такое ScottPlot и для чего он используется?
2. Как установить библиотеку ScottPlot в проект C#?
3. Как создать простой график с использованием ScottPlot?
4. Какой метод используется для отображения графика в WinForms?
5. Какие типы графиков поддерживает ScottPlot?
6. Как построить линейный график и чем он отличается от точечного графика?
7. Как изменить заголовок графика и подписи осей?
8. Как изменить цвет и стиль линий на графике?
9. Как обновить данные на уже существующем графике без его перерисовки?
10. Как добавить возможность масштабирования?
11. Как сохранить график в файл (например, в формате PNG или SVG)?
12. Какие методы доступны для экспорта графиков в другие форматы?
13. Как добавить несколько графиков на одну ось?
14. Как настроить отображение легенды для нескольких графиков?
15. Как реализовать анимацию графика, чтобы данные обновлялись в реальном времени?
16. Как изменить шрифт и размер текста на графике?
17. Как добавить аннотации или текстовые метки к определенным точкам на графике?
18. Как загрузить данные из файла (например, CSV) и построить график на их основе?