НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ВЫСШИЙ КОЛЛЕДЖ ИНФОРМАТИКИ
Кафедра информатики

Практическая работа

НАЗВАНИЕ

Разработка приложения для построения графиков математических функций

Отчет

Разработал Бланк Артём Владимирович

Группа 2407з2

Преподаватель Быков Виталий Валерьевич

Оценка

Дата

Новосибирск, 2025

Содержание отчета

1. Введение

Передо мной стоит задача создать приложение для построения графиков математических функций. В процессе выбора технологии я рассматривал различные варианты и пришел к выводу, что наилучшим решением будет использовать язык программирования C# в сочетании с фреймворком Avalonia UI. Это позволило создать кроссплатформенное приложение с удобным графическим интерфейсом и широкими возможностями для визуализации данных.

2. Стек технологий и пакетов

2.1 Avalonia UI

Avalonia UI – это кроссплатформенный фреймворк для создания пользовательского интерфейса на C#. Он поддерживает паттерн MVVM, обладает гибкой системой разметки XAML и позволяет разрабатывать приложения для Windows, Linux и macOS.

Плюсы Avalonia UI:

- Кроссплатформенность
- Поддержка MVVM
- Гибкость и настраиваемость
- Современный рендеринг с поддержкой аппаратного ускорения

2.2 OxyPlot

OxyPlot – это легковесная библиотека для построения графиков в C#. Она проста в использовании, обладает хорошей производительностью и поддерживает широкий спектр типов диаграмм, линейных графиков. В данном проекте библиотека использовалась для визуализации математических функций.

Плюсы OxyPlot:

- Простота интеграции
- Высокая производительность
- Поддержка различных типов графиков
- Кроссплатформенность

2.3 AngouriMath

AngouriMath – это мощная математическая библиотека для работы с алгебраическими выражениями, производными, интегралами и прочими математическими операциями. Она позволяет анализировать и вычислять выражения, что делает её отличным выбором для приложений, работающих с математическими формулами.

Плюсы AngouriMath:

- Поддержка символьных вычислений
- Работа с производными, интегралами, уравнениями
- Высокая точность вычислений
- Открытый исходный код

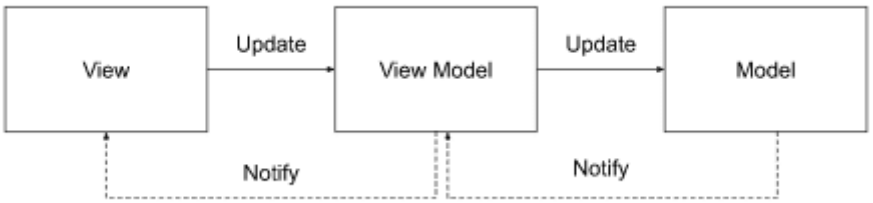
3. Паттерн проектирования MVVM

Приложение разработано с использованием архитектурного паттерна MVVM (Model-View-ViewModel). Этот паттерн позволяет разделить бизнес-логику, представление данных и пользовательский интерфейс.

Основные компоненты:

- Model (Модель) — содержит данные и бизнес-логику приложения.
- View (Представление) — отвечает за отображение данных и взаимодействие с пользователем.

- ViewModel (Модель представления) — связывает Model и View, обрабатывает команды и обновляет интерфейс.



Применение MVVM позволяет упростить тестирование и поддержку кода, а также сделать интерфейс более гибким.

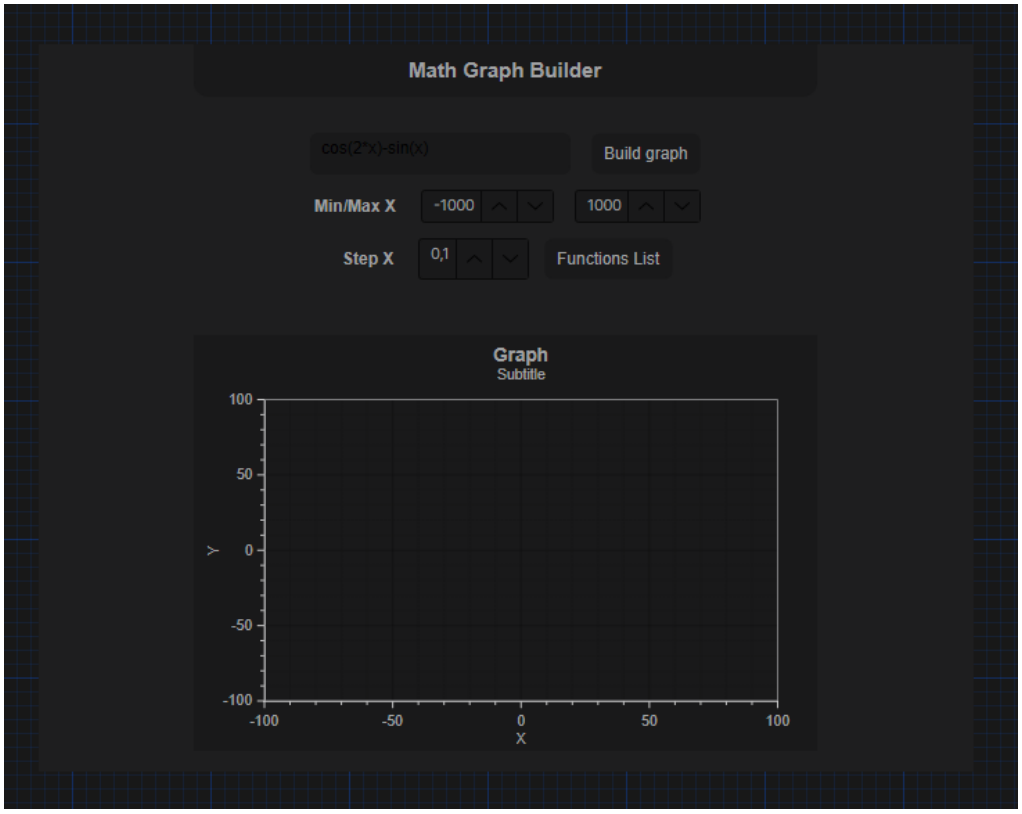
4. Разметка окон

В качестве разметки окна в Avalonia используется axaml(расширенный xaml). XAML — расширяемый язык разметки для приложений — основанный на XML язык разметки для декларативного программирования приложений, разработанный Microsoft. Модель приложений Vista включает объект Application. Его набор свойств, методов и событий позволяет объединить веб-документы в связанное приложение.

4.1 Главное окно (MainWindow)

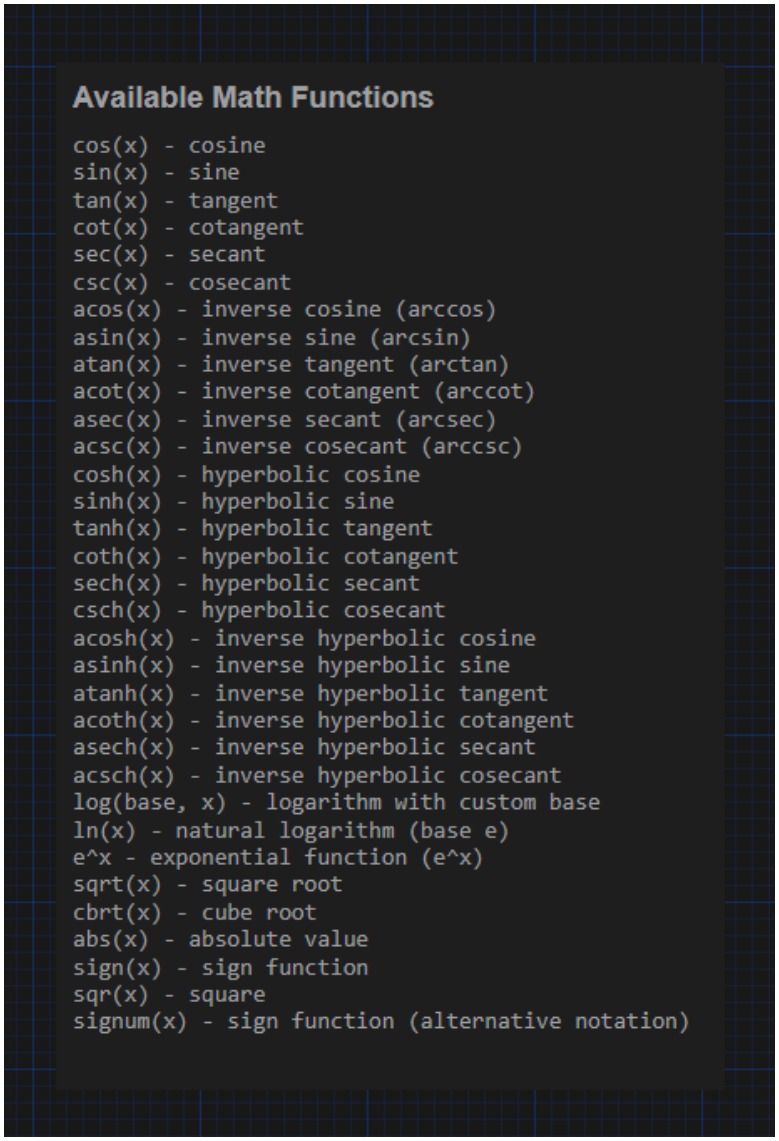
Главное окно содержит:

- Поле ввода математической функции
- Поле ввода нижней границы параметра X
- Поле ввода верхней границы параметра X
- Поле ввода шага параметра X
- Кнопку построения графика
- Кнопку вызова окна со списком функций
- Область для отображения графика



4.2 Окно списка доступных функций (FunctionsWindow)

Это дополнительное окно, содержащее список доступных математических функций, поддерживаемых AngouriMath. Оно позволяет пользователю узнать, какие функции можно использовать при построении графика.

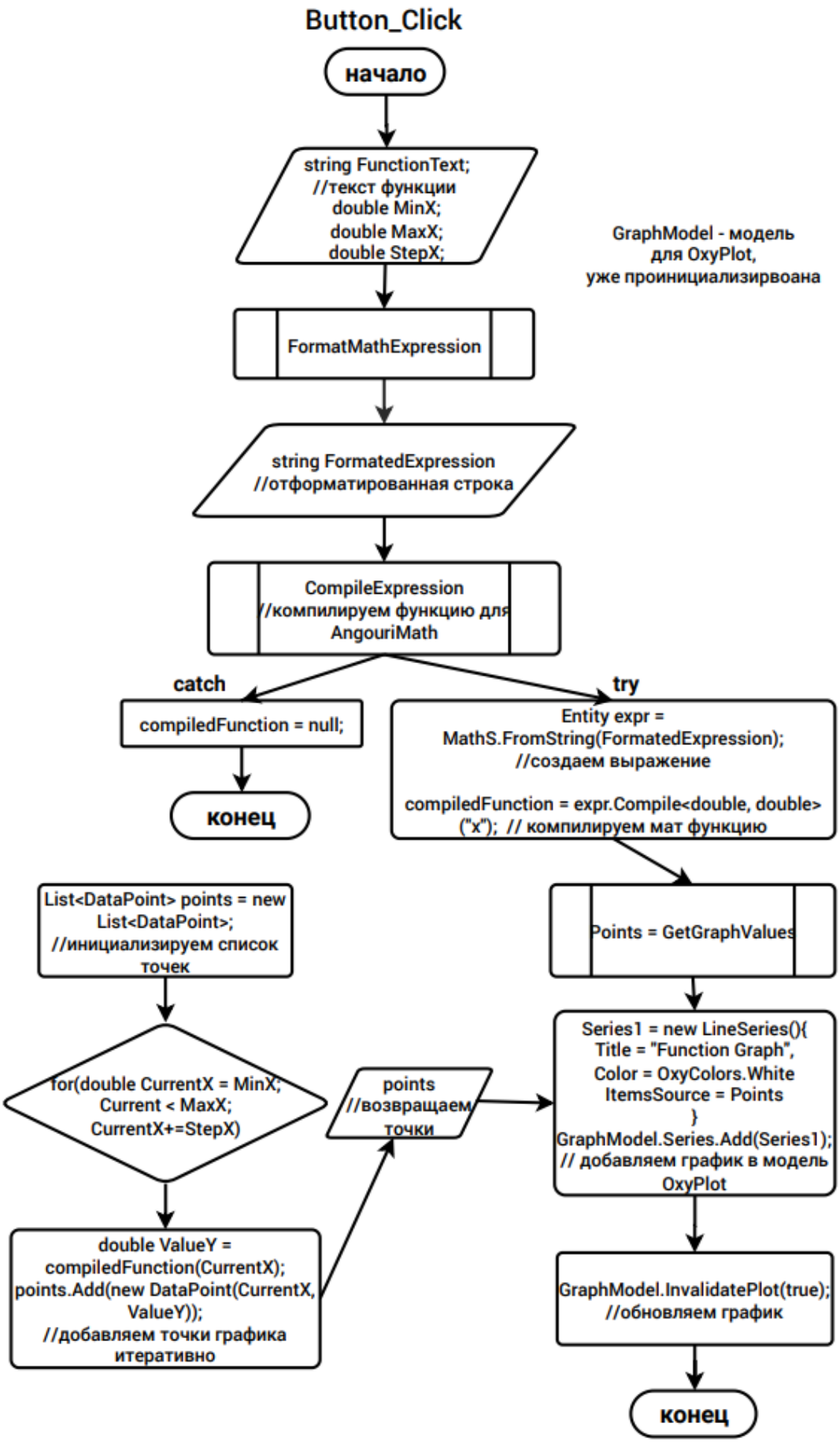


5. Алгоритм работы

Алгоритм работы приложения состоит из следующих шагов:

- Пользователь вводит математическое выражение в текстовое поле.
- Программа парсит выражение с помощью AngouriMath и проверяет его корректность.
- На основе введенного выражения создается вычисляемая функция.
- Генерируются точки для построения графика в заданном диапазоне (MinX, MaxX) с определенным шагом (StepX).
- Точки передаются в OxyPlot, который строит график.
- График отображается в главном окне приложения.
- Пользователь может изменить параметры построения и обновить график.

6. Блок-схема алгоритма



7. Код логики

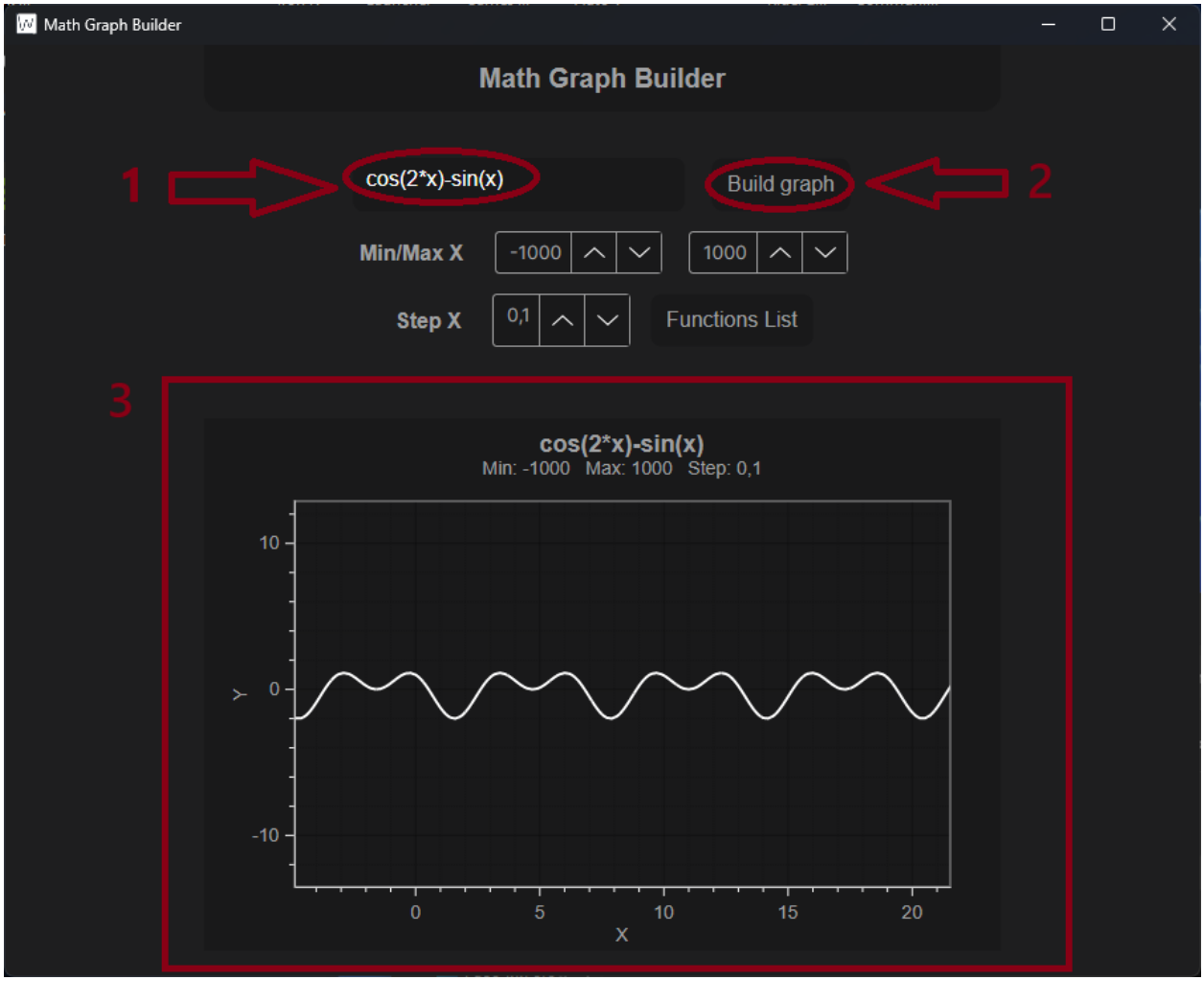
```
using System;
using System.Collections.Generic;
using OxyPlot;
using OxyPlot.Series;
using OxyPlot.Axes;
using System.Text.RegularExpressions;
using AngouriMath;
using MathGraph.Views;


namespace MathGraph.ViewModels;

public partial class MainWindowViewModel : ViewModelBase
{
    //конструктор
    public MainWindowViewModel()
    {
        InitializePlotModel();

        //инициализация значений
        this.FunctionText = "cos(2*x)-sin(x)";
        this.MinX = -1000;
        this.MaxX = 1000;
        this.StepX = 0.1;
    }

    public string FunctionText { get; set;  } //текст функции

    public PlotModel GraphModel { get; private set; } //модель графика
    public LineSeries Series1 { get; private set; } // функция на графике
    public List<DataPoint> Points { get; set; } = new List<DataPoint>(); //
список точек для графика
    public double MinX { get; set; }  // нижняя граница
    public double MaxX { get; set; } // верхняя граница
    public double StepX { get; set; } // шаг функции
    public string FormatedExpression { get; set; } // мат выражение
    public Func<double, double>? compiledFunction { get; set; }


    //точки для инициализации графика
    private List<DataPoint> ExamplePoints()
    {
        return new List<DataPoint>()
        {
            new DataPoint(0, 0),
        };
    }

    //инициализация графика
    private void InitializePlotModel()
    {
        this.GraphModel = new PlotModel();
        GraphModel.Title = "Graph";
        GraphModel.Subtitle = "Subtitle";

        GraphModel.Axes.Add(new LinearAxis {
            Position = AxisPosition.Bottom,
            Title = "X",
            Minimum = -100,
            Maximum = 100,
            MajorGridlineStyle = LineStyle.Solid,
```

```
            MinorGridlineStyle = LineStyle.Dot,
            TicklineColor = OxyColors.White,
            AxislineStyle = LineStyle.Solid,
            AxislineColor = OxyColors.White,
        });
        GraphModel.Axes.Add(new LinearAxis
        {
            Position = AxisPosition.Left,
            Title = "Y",
            Minimum = -100,
            Maximum = 100,
            MajorGridlineStyle = LineStyle.Solid,
            MinorGridlineStyle = LineStyle.Dot,
            TicklineColor = OxyColors.White,
            AxislineStyle = LineStyle.Solid,
            AxislineColor = OxyColors.White,
        });

        GraphModel.PlotMargins = new OxyThickness(60, 10, 30, 40);

        this.Points = ExamplePoints();

        Series1 = new LineSeries
        {
            Title = "Function graph",
            Color = OxyColors.White,
            ItemsSource = Points,
        };
        Series1.MarkerType = MarkerType.Cross;

        GraphModel.Series.Add(Series1);

        GraphModel.DefaultFont = "Arial";
        GraphModel.DefaultFontSize = 14;
        GraphModel.Background = OxyColor.FromRgb(25, 25, 26);
        GraphModel.TextColor = OxyColor.FromRgb(160, 161, 163);
        GraphModel.PlotAreaBorderColor = OxyColor.FromRgb(160, 161, 163);

        GraphModel.InvalidatePlot(true);
    }

    // нажатие кнопки
    public void Button_Click()
    {
        BuildGraph();
    }

    //построение графика
    private void BuildGraph()
    {
        FormatedExpression = FormatMathExpression(FunctionText);

        CompileExpression();

        // обработка ошибки компиляции мат выражения
        if (compiledFunction == null)
        {
            Console.WriteLine("Error: Could not compile expression.");

            GraphModel.Title = FormatedExpression;
            GraphModel.Subtitle = $"Error: Could not compile expression.";
            GraphModel.Series.Clear();

            GraphModel.InvalidatePlot(true);
```

```
            return;
        }

        //обновление текста
        GraphModel.Title = FormatedExpression;
        GraphModel.Subtitle = $"Min: {MinX}   Max: {MaxX}   Step: {StepX}";
        GraphModel.Series.Clear();

        //обновление точек графика
        Points = GetGraphValues();

        //добавление нового графика
        Series1 = new LineSeries()
        {
            Title = "Function graph",
            Color = OxyColors.White,
            ItemsSource = Points,
        };
        Series1.MarkerType = MarkerType.Cross;
        GraphModel.Series.Add(Series1);

        // обновление графика
        GraphModel.InvalidatePlot(true);
    }

    // компиляция мат выражения
    private void CompileExpression()
    {
        try
        {
            Entity expr = MathS.FromString(FormatedExpression);
            compiledFunction = expr.Compile<double, double>("x");
        }
        catch (Exception ex)
        {
            compiledFunction = null;
            Console.WriteLine("Ошибка компиляции: " + ex.Message);
        }
    }

    // вычисление точек графика
    private List<DataPoint> GetGraphValues()
    {
        Entity expr = MathS.FromString(FormatedExpression); // Парсим
выражение

        List<DataPoint> points = new List<DataPoint>();

        for (double CurrentX = MinX; CurrentX <= MaxX; CurrentX += StepX)
        {
            double ValueY = compiledFunction(CurrentX);
            points.Add(new DataPoint(CurrentX, ValueY));
        }

        return points;
    }

    //форматируем математическое выражение
    static string FormatMathExpression(string expression)
    {
        expression = Regex.Replace(expression, @"\s+", " ");  // Убираем
лишние пробелы
        expression = Regex.Replace(expression, @"\s*([\+\-\*/\^\(\)])\s*",
```

```
"$1"); // Убираем пробелы вокруг операторов
        return expression.ToLower(); // нижний регистр
    }

    // вызов окна со списком функций
    public void FunctionListButton_Click()
    {
        var functionsWindow = new FunctionsWindow();
        functionsWindow.Show();
    }

}
```

8. Работа программы



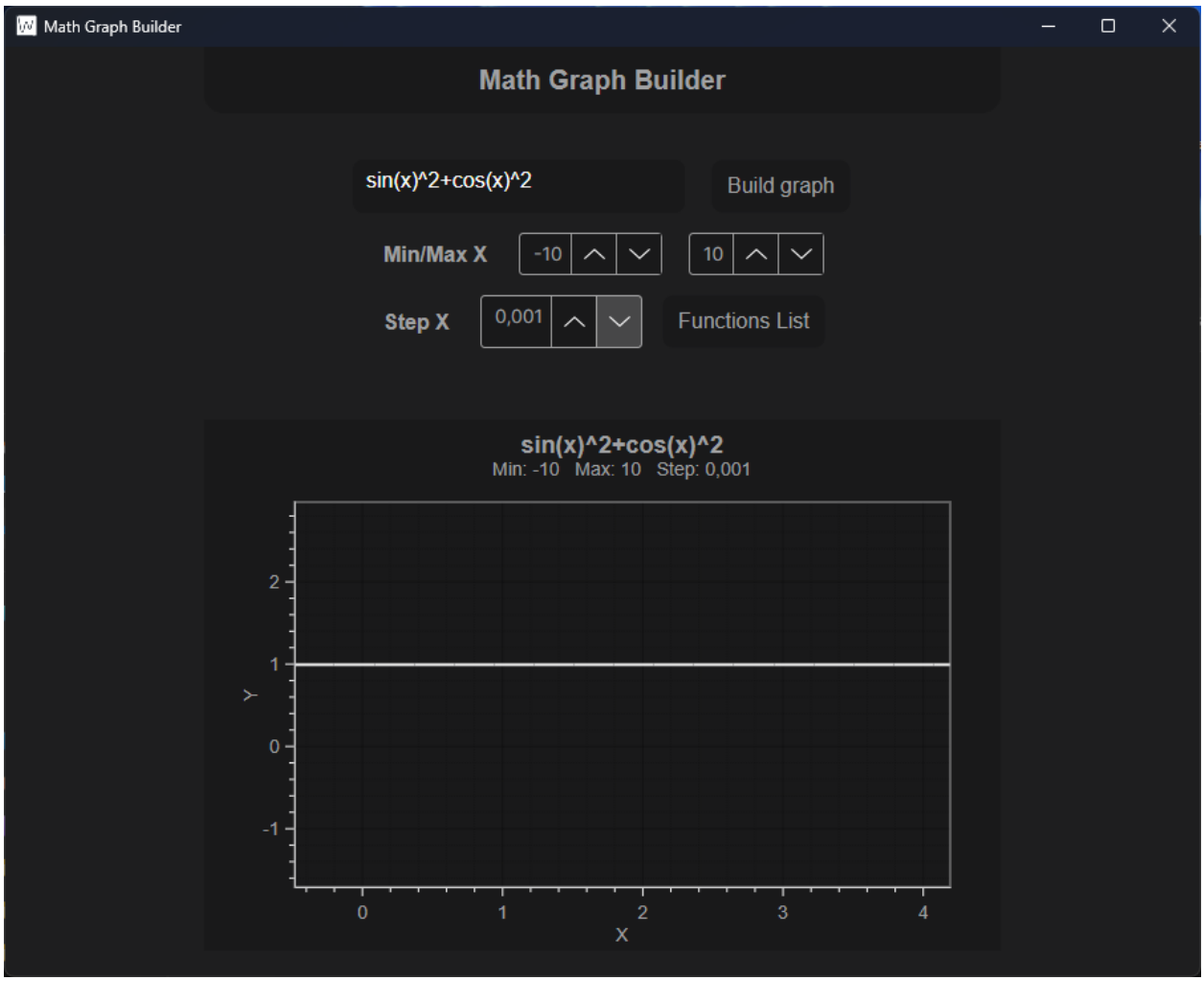
1. Вводим математическое выражение (по необходимости - Min/Max и Step)
2. Нажимаем кнопку “Build graph”
3. Получаем построенный график

9. Контрольный пример

В качестве контрольного примера возьмем основное тригонометрическое тождество.

$$sin^2\,x + cos^2\,x = 1$$

То есть, это выражение должно равняться 1 при любом значении параметра X.



Действительно, функция возвращает 1 при любом значении параметра X, значит программа работает и верно строит график.

10. Источники

- Документация Avalonia UI - https://docs.avaloniaui.net/
- Страница OxyPlot на GitHub - https://github.com/oxyplot/oxyplot-avalonia
- Документация OxyPlot - https://app.readthedocs.org/projects/oxyplot/downloads/pdf/latest/