ДОДАТОК А

Опубліковані результати

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
кафедра комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
Харківський національний університет міського господарства
імені О.М. Бекетова
Братиславський університет економіки та менеджменту
Громадська організація «Перспектива»
Угорський університет сільського господарства та природничих наук

МАТЕРІАЛИ

I International Conference
«Sustainable smart cities and communities: business and innovation solutions»
(Сталі розумні міста та спільноти: бізнес та інноваційні рішення)
SSC&C2025

21 квітня 2025

[електронне видання]

Харків 2025

Сталі розумні міста та спільноти: бізнес та інноваційні рішення 2025: матеріали І-ої Міжнародной конференції, Харків, 21 квітня 2025.: тези доповідей / [редкол. І.Ш. Невлюдов (відповідальний редактор)].-Харків: [електронний друк], 2025. – 68 с.

У збірник включені тези доповідей, які присвячені сучасним цифровим технологіям та автоматизації для сталого розвитку розумних міст; роботизованим системам та автономним технологіям у міському середовищі; циркулярной економіки та зеленой енергетики в автоматизованих системах; розумні транспортні системи та мобільність майбутнього; кіберфізичні системи та безпека даних у міській автоматизації; HMI та цифрові платформи для інтеграції міських послуг; автоматизація промисловості та міської інфраструктури: виклики та рішення ресурсоефективності.

Sustainable smart cities and communities: business and innovation solutions 2025: Proceedings of I st I International Conference, Kharkiv, April 21, 2025: Thesises of Reports / [Ed. I.Sh. Nevlyudov (chief editor).] .- Kharkiv .: [electronic version], 2025. - 68 p.

The collection includes abstracts of reports dedicated to modern digital technologies and automation for the sustainable development of smart cities; robotic systems and autonomous technologies in the urban environment; circular economy and green energy in automated systems; smart transport systems and mobility of the future; cyber-physical systems and data security in urban automation; HMI and digital platforms for the integration of urban services; automation of industry and urban infrastructure: challenges and solutions for resource efficiency.

Результати наукових досліджень, що представлені у збірнику, виконані в межах реалізації **міжнародного проєкту Еразмус+ Жан Моне Модуль «Україна-ЄС: рішення циклічної економіки для розумних та сталих міст» («Ukraine-EU: Circular Economy Solutions 4 Smart and Sustainable Cities (Eco4Smart)») – # 101127659)**

ЗМІСТ

FREE SOFTWARE AS A TOOL FOR TECHNOLOGICAL ADVANTAGE AND INDEPENDENCE IN THE DIGITAL ENVIRONMENT

Nazarii Piven
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, Nauky av., 14
E-mail: nazarii.piven@nure.ua

***Annotation*:** The paper examines the current issues of using non-commercial software as a tool for technological advantage and ensuring independence in the digital environment. An analysis of existing development environments, their effectiveness, relevance and application features, and a comparative analysis of their advantages and disadvantages are conducted.

Keywords: free software, development, automation, technological independence

The modern development of automated process control systems requires high-performance computing using open compilers and integrated development environments. Open software tools allow you to avoid licensing restrictions, reduce the costs of implementing new technologies, and ensure technological independence.

Using commercial development environments is often associated with significant financial costs, vendor lock-in, and limited customization. In contrast, free IDEs such as Code::Blocks, Eclipse, Visual Studio Code, and Geany offer a wide range of tools for programming, testing, and debugging without the need to purchase expensive licenses.

The purpose of the work is to study existing open development environments and analyze their effectiveness in the field of HPC for automated systems within the framework of the "smart city" concept.

The research was conducted by conducting a comparative analysis of development environments in terms of performance, integration capabilities, and flexibility for learning and development. The study includes an analysis of IDE functionality and their extensibility using plugins.

Based on the analysis, the main characteristics of the development environments were determined (Table 1).

Table 1 – Comparative characteristics of development environments.

Characteristic	Code:: Blocks	Eclipse	VS Code	Geany	Visual Studio (комерційне)	JetBrains IntelliJ IDEA (комерційне)
License	Free	Free	Free	Free	Paid	Paid
Programming languages	C/C++	Java, C++, Python	JavaScript, Python, C++	C, C++, Python	C#, C++, Python	Java, Kotlin, Python
Productivity	Low	High	Middle	Low	High	High
Integration with version control systems	Limited	Built-in	Built-in	Low	Built-in	Built-in
Extension possibilities	Limited	Wide	Wide	Limited	Wide	Wide
Price	Free	Free	Free	Free	From $45/mn	From $149/an
Owner dependency	None	None	Partly	None	Full	Full

A comparison of existing solutions was also conducted, the results are recorded in Table 2.

Table 2 – Comparative analysis of existing solutions.

Characteristic	Free IDE (Eclipse, VS Code)	Commercial IDE (VS,IJ)
Price	Free	From $100/an
Flexibility	Open source, modifications	Limited extention
Functionality	Large set of plugins, limitations	Premium tools, integration with CI/CD
Performance	Lighter, faster, more productive	Performance-dependent
Support	Developer community, forums	Official technical support
Dependency	Complete independence	Owner dependency

The result of the comparison of existing commercial solutions is recorded in Table 3.

Table 3 – Comparison of commercial solutions

Name	Owner	Languages	Price 2024
VS	Microsoft	C#, C++, Python, JS	$45-$250/monthly (Pro, Enterprise)
IJ	JetBrains	Java, Kotlin, Python, JS	$249/ann (Personal), $599/ann(Enterprise)
PyCharm	JetBrains	Python	$99/year (Personal), $249/year (Enterprise)
CLion	JetBrains	C, C++	$199/ann (Personal), $549/ann (Enterprise)
MATLAB	MathWorks	MATLAB	$860 base version + supplements

CONCLUSIONS: The results of the study confirm that the use of free IDEs is an effective solution for developing automated systems in smart cities, as they provide high performance, scalability, and vendor independence.

The results obtained can be used to select the optimal development environment in educational institutions and research projects related to the automation of technological processes.

References:

1. Van Rejsvud V., de Yager A. Vilne ta vidkryte prohramne zabezpechennya dlya rozvytku // arXiv preprint arXiv:0808.3717. (2008). https://arxiv.org/abs/0808.3717.
2. Stallman R. Vilne prohramne zabezpechennya, vilne suspilstvo. Vybrani ese / Per. z anhl. – K.: Fond Vidkryte suspilstvo, (2014). – 240 s.
3. Lupak R. L., Vasyltsiv T. H. Konkurentospromozhnist pidpryyemstva: navch. posib. Lviv: Vydavnytstvo LKA, (2016). 484 s.
4. KKU. St. 176. Porushennya avtors′koho prava ta sumizhnykh prav, (2025), https://zakon.rada.gov.ua/laws/show/2341-14#n1785.
5. Linfo.org. EULA. (2025), https://www.linfo.org/eula.html.
6. JetBrains. Official site(2025), https://www.jetbrains.com.
7. MathWorks. Official site (2025), https://www.mathworks.com.
8. Lumen Database. Piracy (2025), https://lumendatabase.org/topics/11.
9. Code::Blocks Team. Code::Blocks. (2025), https://www.codeblocks.org/.
10. ISO C++ Foundation. (2025), https://isocpp.org/ .
11. Amazon Web Services. What is an IDE?(2025), https://aws.amazon.com/what-is/ide/.
12. Eclipse Foundation. Eclipse IDE (2025), https://eclipseide.org/.
13. Microsoft. Visual Studio Code (2025), https://code.visualstudio.com/.
14. Geany. Geany (2025), https://www.geany.org/.

ДОДАТОК Б

Програмна частина проєкту

Лістинг UserControl1.cs

```
using Microsoft.Web.WebView2.Core;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Text.Json;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.VisualStyles;
using System.Xml.Linq;
using static Formula_Window.UserControl1.HierarchicalNodeControl;

namespace Formula_Window
{
    public partial class UserControl1 : UserControl
    {
        private ComboBox _cmbMatrixOp;

        public abstract class ExprNode { }

        private string _pendingJsonData = null;

        public class ChainNode : ExprNode
        {
            public string Kind;

            public List<object> Items = new();
        }

        public class BigOpNode : ExprNode
        {
            public string Kind;
            public string Index;
            public ExprNode Lower;
            public ExprNode Upper;
            public ExprNode Body;
            public string Condition;
            public string IntVar;
        }

        private class NodeStyleControls
        {
            public ComboBox LineType;
            public ComboBox LineWidth;
            public ComboBox MarkerType;
            public ComboBox MarkerSize;
            public Button ColorBtn;
        }
        public class SubExpressionParams
        {
            public string Name { get; set; }
            public string Formula { get; set; }
            public string Color { get; set; }
            public string LineStr { get; set; }
            public string MarkerStr { get; set; }
        }
        private Dictionary<ExprNode, NodeStyleControls> _nodeStyles = new Dictionary<ExprNode, NodeStyleControls>();
```

```
class ValueNode : ExprNode
{

   public string Name;
   public bool IsVariable =>
     !string.IsNullOrEmpty(Name)

       && char.IsLetter(Name[0])

       && !Name.Contains(«-»)

       && Name != «incompatible-type»

       && Name != «number»

       && !Name.StartsWith(«list»)

       && !Name.StartsWith(«matrix»);
}
private Button _floatingButton;
public class MatrixNode : ExprNode
{
   public List<List<ExprNode>> Rows = new();
}
private void InitFloatingButton()
{
   _floatingButton = new Button();
   _floatingButton.Text = «Розрахувати»;
   _floatingButton.Size = new Size(140, 45);
   _floatingButton.FlatStyle = FlatStyle.Flat;
   _floatingButton.FlatAppearance.BorderSize = 0;
   _floatingButton.BackColor = Color.FromArgb(255, 100, 149, 237);
   _floatingButton.ForeColor = Color.White;
   _floatingButton.Font = new Font(«Segoe UI», 10f, FontStyle.Bold);
   _floatingButton.Cursor = Cursors.Hand;
   int margin = 20;
   _floatingButton.Location = new Point(
      this.Width - _floatingButton.Width - margin,
      this.Height - _floatingButton.Height - margin
   );
   _floatingButton.Anchor = AnchorStyles.Bottom | AnchorStyles.Right;
   _floatingButton.Visible = false;
   _floatingButton.Click += async (s, e) =>
   {
      if (webView21 != null && webView21.CoreWebView2 != null)
      {
         try
         {
            string rawJson = await webView21.CoreWebView2.ExecuteScriptAsync(«document.getElementById(‘mf’).getValue()»);
            if (!string.IsNullOrEmpty(rawJson) && rawJson != «null»)
            {
               string cleanLatex = System.Text.Json.JsonSerializer.Deserialize<string>(rawJson);
               currentLatexCode = cleanLatex;
            }
         }
         catch (Exception ex)
         {
            System.Diagnostics.Debug.WriteLine(«Помилка отримання LaTeX: « + ex.Message);
         }
      }
      RunPyCalculation();
   };
   this.Controls.Add(_floatingButton);
   _floatingButton.BringToFront();
}

public CalculationConfig GetCalculationConfig()
{
   var config = new CalculationConfig();
   var addedFormulas = new HashSet<string>();
```

```
if (_currentRootAst != null)
    config.Formula = HierarchicalNodeControl.AstToFormulaConverter.Convert(_currentRootAst);
else
    config.Formula = «0»;
var varNamesInConfig = new HashSet<string>();
foreach (var kvp in inputs)
{
    string varName = kvp.Key;
    var ctrl = kvp.Value;
    if (!double.TryParse(ctrl.start.Text.Replace(«,», «.»), NumberStyles.Any, CultureInfo.InvariantCulture, out double s)) s = -10.0;
    if (!double.TryParse(ctrl.end.Text.Replace(«,», «.»), NumberStyles.Any, CultureInfo.InvariantCulture, out double e)) e = 10.0;
    int lIdx = ctrl.cmblinestyle.SelectedIndex;
    if (lIdx < 0) lIdx = 1;
    string lW = ctrl.cmblinewidth.Text.Replace(«,», «.»);
    if (string.IsNullOrWhiteSpace(lW)) lW = «1.5»;
    int mIdx = ctrl.cmbmarker.SelectedIndex;
    if (mIdx < 0) mIdx = 1;
    string mSz = ctrl.cmbmarkerwidth.Text;
    if (string.IsNullOrWhiteSpace(mSz)) mSz = «6»;
    config.Variables.Add(new VariableParams
    {
        Name = varName,
        Start = s,
        End = e,
        Color = $»{ctrl.colorBtn.BackColor.R:D3}{ctrl.colorBtn.BackColor.G:D3}{ctrl.colorBtn.BackColor.B:D3}»,
        LineStr = $»{lIdx}{lW}»,
        MarkerStr = $»{mIdx:D2}{mSz}»
    });
}
var addedNames = new HashSet<string>();
foreach (var kvp in _nodeStyles)
{
    ExprNode node = kvp.Key;
    NodeStyleControls style = kvp.Value;
    if (node == _currentRootAst) continue;
    string subFormula = HierarchicalNodeControl.AstToFormulaConverter.Convert(node);
    string name = subFormula.ToUpper();
    if (string.IsNullOrEmpty(subFormula) || addedNames.Contains(name) || name.Contains(«ERROR»)) continue;
    addedNames.Add(name);
    int lineType = style.LineType.SelectedIndex;
    if (lineType < 0) lineType = 1;
    string lineWidth = style.LineWidth.Text.Replace(«,», «.»);
    if (string.IsNullOrWhiteSpace(lineWidth)) lineWidth = «1.5»;
    int markerType = style.MarkerType.SelectedIndex;
    if (markerType < 0) markerType = 0;
    string markerSize = style.MarkerSize.Text;
    if (string.IsNullOrWhiteSpace(markerSize)) markerSize = «6»;
    config.SubExpressions.Add(new SubExpressionParams
    {
        Name = name,
        Formula = subFormula,
        Color = $»{style.ColorBtn.BackColor.R:D3}{style.ColorBtn.BackColor.G:D3}{style.ColorBtn.BackColor.B:D3}»,
        LineStr = $»{lineType}{lineWidth}»,
        MarkerStr = $»{markerType:D2}{markerSize}»
    });
}
if (!double.TryParse(tb_Steps.Text.Replace(«,», «.»), NumberStyles.Any, CultureInfo.InvariantCulture, out double steps)) steps = 0.01;
if (!int.TryParse(tb_Iterations.Text, out int iter)) iter = 1000;
config.Steps = steps;
config.Iterations = iter;
config.ChartName = tb_ChartName.Text;
config.Latex = currentLatexCode;
config.XLabel = tb_xLabelName.Text;
config.YLabel = tb_yLabelName.Text;

Color resCol = Color.CornflowerBlue;
config.ResultColor = $»{resCol.R:D3}{resCol.G:D3}{resCol.B:D3}»;
config.ResultLineType = 1;
config.ResultLineWidth = 2.0;
config.ResultMarkerType = 1;
```

```
    config.ResultMarkerSize = 6;
    return config;
}
public class CalculationConfig
{
    public string Formula { get; set; }
    public double Steps { get; set; }
    public int Iterations { get; set; }
    public List<VariableParams> Variables { get; set; } = new();
    public string ChartName { get; set; }
    public string Latex { get; set; }
    public string XLabel { get; set; }
    public string YLabel { get; set; }

    public string ResultColor { get; set; }
    public int ResultLineType { get; set; }
    public double ResultLineWidth { get; set; }
    public int ResultMarkerType { get; set; }
    public int ResultMarkerSize { get; set; }
    public List<SubExpressionParams> SubExpressions { get; set; } = new();
}
public class VariableParams
{
    public string Name { get; set; }
    public double Start { get; set; }
    public double End { get; set; }

    public string Color { get; set; }
    public string LineStr { get; set; }
    public string MarkerStr { get; set; }
}
public class HierarchicalNodeControl : UserControl

{

    private Label _lblTitle;

    private FlowLayoutPanel _pnlContent;

    private Color _lineColor = Color.FromArgb(100, 149, 237);

    public string Title

    {

        get => _lblTitle.Text;

        set => _lblTitle.Text = value;

    }

    public ControlCollection ContentControls => _pnlContent.Controls;

    public HierarchicalNodeControl()

    {

        this.AutoSize = true;

        this.AutoSizeMode = AutoSizeMode.GrowAndShrink;

        this.Padding = new Padding(15, 0, 0, 10);

        this.DoubleBuffered = true;
        _lblTitle = new Label

        {

            AutoSize = true,
```

```
        Font = new Font(«Segoe UI», 9f, FontStyle.Bold),
        ForeColor = Color.DimGray,
        Margin = new Padding(0, 5, 0, 5)
    };
    _pnlContent = new FlowLayoutPanel
    {
        AutoSize = true,
        AutoSizeMode = AutoSizeMode.GrowAndShrink,
        FlowDirection = FlowDirection.TopDown,
        WrapContents = false,
        Margin = new Padding(0),
        Padding = new Padding(0)
    };
    var mainLayout = new FlowLayoutPanel
    {
        AutoSize = true,
        AutoSizeMode = AutoSizeMode.GrowAndShrink,
        FlowDirection = FlowDirection.TopDown,
        WrapContents = false,
        Dock = DockStyle.Fill
    };
    mainLayout.Controls.Add(_lblTitle);
    mainLayout.Controls.Add(_pnlContent);
    this.Controls.Add(mainLayout);
}
public static class AstToFormulaConverter
{
    public static string Convert(UserControl1.ExprNode node)
    {
        if (node == null) return «0»;
        switch (node)
        {
            case UserControl1.MatrixNode m:
                {
                    var rowsStr = m.Rows.Select(row =>
                        «[« + string.Join(«, «, row.Select(cell => Convert(cell))) + «]»
                    );
                    return $»Matrix([{string.Join(«, «, rowsStr)}])»;
                }
            case UserControl1.DerivativeNode d:
                return $»diff({Convert(d.Body)}, {d.Variable})»;
            case UserControl1.BigOpNode b:
                if (b.Kind == «Integral»)
                    return $»integrate({Convert(b.Body)}, ({b.IntVar}, {Convert(b.Lower)}, {Convert(b.Upper)}))»;
                if (b.Kind == «Sum»)
                    return $»Sum({Convert(b.Body)}, ({b.Index}, {Convert(b.Lower)}, {Convert(b.Upper)}))»;
```

```
                if (b.Kind == «Product»)
                    return $»Product({Convert(b.Body)}, ({b.Index}, {Convert(b.Lower)}, {Convert(b.Upper)}))»;
                return b.Kind;
            case UserControl1.ChainNode chain:
                List<string> parts = new List<string>();
                foreach (var item in chain.Items)
                {
                    if (item is UserControl1.ExprNode en)
                    {
                        string subExpr = Convert(en);
                        if (!string.IsNullOrEmpty(subExpr)) parts.Add(subExpr);
                    }
                    else if (item is string s)
                    {
                        if (parts.Count > 0) parts.Add(s);
                    }
                }
                if (parts.Count > 0)
                {
                    string last = parts[parts.Count - 1];
                    if (last == «+» || last == «-» || last == «*» || last == «/»)
                        parts.RemoveAt(parts.Count - 1);
                }
                return parts.Count > 0 ? $»({string.Join(««, parts)})» : «0»;
            case UserControl1.OperationNode op:
                string funcName = op.Op.ToLower();
                string symbol = op.Op switch
                {
                    «Add» => «+»,
                    «Subtract» => «-»,
                    «Multiply» => «*»,
                    «Divide» => «/»,
                    «Power» => «**»,
                    «List» => «, «,
                    «Sequence» => «, «,
                    _ => null
                };
                var args = op.Args.Select(a => Convert(a)).Where(s => !string.IsNullOrEmpty(s)).ToArray();
                if (symbol != null)
                {
                    return args.Length > 1 ? $»({string.Join(symbol, args)})» : (args.FirstOrDefault() ?? «0»);
                }
                return $»{funcName}({string.Join(«, «, args)})»;
            case UserControl1.ValueNode v:
                return string.IsNullOrEmpty(v.Name) ? «0» : v.Name;
            default:
                return «0»;
        }
    }
}

protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);
    e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
    using (Pen p = new Pen(_lineColor, 2))
    {
        e.Graphics.DrawLine(p, 5, 10, 5, this.Height - 10);
        e.Graphics.DrawLine(p, 5, this.Height - 10, 15, this.Height - 10);
        e.Graphics.DrawLine(p, 5, 10, 15, 10);

    }
}
public void AddControl(Control c)
{

    _pnlContent.Controls.Add(c);

}
}
```

```
class OperationNode : ExprNode
{
   public string Op;
   public List<ExprNode> Args = new();
}
class DerivativeNode : ExprNode
{
   public string Variable;
   public string Order;
   public ExprNode Body;
   public string Condition;
}

class NormalNode : ExprNode
{
   public JsonElement MathJson;
}
bool IsRealVariable(string name)
{
   if (string.IsNullOrWhiteSpace(name)) return false;
   if (double.TryParse(name, System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out _))
      return false;
   if (name.Length == 1 && !char.IsLetter(name[0]))
      return false;
   if (name == «+» || name == «-» || name == «*» || name == «/» || name == «^» || name == «=«)
      return false;
   if (name.StartsWith(«‘«) || name.Contains(«type») || name.Contains(«number»))
      return false;
   if (name is «incompatible-type» or «list» or «Error» or «Nothing»)
      return false;
   if (name.StartsWith(«\\»))
      return false;
   return true;
}
ExprNode ParseMathJson(JsonElement el)
{
   if (el.ValueKind == JsonValueKind.Number)
      return new ValueNode { Name = el.ToString() };

   if (el.ValueKind == JsonValueKind.String)
   {
      var s = el.GetString();
      if (string.IsNullOrWhiteSpace(s)) return null;
      return new ValueNode { Name = s };
   }

   if (el.ValueKind == JsonValueKind.Array)
   {
      var arr = el.EnumerateArray().ToList();
      if (arr.Count == 0) return null;
      string opName = arr[0].GetString();
      if (string.Equals(opName, «Matrix», StringComparison.OrdinalIgnoreCase))
      {
         var matrixNode = new MatrixNode();
         var args = arr.Skip(1).ToList();
         if (args.Count == 1 && args[0].ValueKind == JsonValueKind.Array)
         {
            var firstArgArr = args[0].EnumerateArray().ToList();
            if (firstArgArr.Count > 0 && firstArgArr[0].ValueKind == JsonValueKind.String && firstArgArr[0].GetString() == «List»)
            {
               args = firstArgArr.Skip(1).ToList();
            }
         }
         foreach (var rowElement in args)
         {
            var rowExprs = new List<ExprNode>();
            if (rowElement.ValueKind == JsonValueKind.Array)
            {
               var rowItems = rowElement.EnumerateArray().ToList();
               int startIndex = 0;
```

```
                if (rowItems.Count > 0 && rowItems[0].ValueKind == JsonValueKind.String && rowItems[0].GetString() == «List»)
                {
                    startIndex = 1;
                }

                for (int k = startIndex; k < rowItems.Count; k++)
                {
                    var cell = ParseMathJson(rowItems[k]);
                    if (cell != null) rowExprs.Add(cell);
                }
            }
            if (rowExprs.Count > 0)
            {
                matrixNode.Rows.Add(rowExprs);
            }
        }
        return matrixNode;
    }
    var node = new OperationNode { Op = opName };
    for (int i = 1; i < arr.Count; i++)
    {
        var child = ParseMathJson(arr[i]);
        if (child != null) node.Args.Add(child);
    }
    return node;
  }
  return null;
}
public string GlobalLatexValue { get; set; }
public string GlobalLatexEnc { get; set; }
private System.Windows.Forms.Timer _debounceTimer;
private Dictionary<string, Color> _variableColorMemory = new Dictionary<string, Color>();
private List<Color> _colorPalette = new List<Color>();
private int _nextColorIndex = 0;

private void GeneratePalette()
{
  _colorPalette.Clear();
  double goldenRatioConjugate = 0.618033988749895;
  double currentHue = 0.5;
  for (int i = 0; i < 1000; i++)
  {
    currentHue += goldenRatioConjugate;
    currentHue %= 1.0;
    Color c = ColorFromHSV(currentHue, 0.5, 0.95);
    _colorPalette.Add(c);
  }
}

private Color ColorFromHSV(double hue, double saturation, double value)
{
  int hi = Convert.ToInt32(Math.Floor(hue * 6)) % 6;
  double f = hue * 6 - Math.Floor(hue * 6);
  value = value * 255;
  int v = Convert.ToInt32(value);
  int p = Convert.ToInt32(value * (1 - saturation));
  int q = Convert.ToInt32(value * (1 - f * saturation));

  int t = Convert.ToInt32(value * (1 - (1 - f) * saturation));

  if (hi == 0) return Color.FromArgb(255, v, t, p);

  else if (hi == 1) return Color.FromArgb(255, q, v, p);

  else if (hi == 2) return Color.FromArgb(255, p, v, t);

  else if (hi == 3) return Color.FromArgb(255, p, q, v);

  else if (hi == 4) return Color.FromArgb(255, t, p, v);
```

```
            else return Color.FromArgb(255, v, p, q);

        }

        public string ProjectFilePath { get; private set; } = null;

        public bool IsDirty { get; set; } = false;

        public event EventHandler<string> ProjectSaved;
        private Dictionary<string, (TextBox start, TextBox end, ComboBox cmblinestyle, ComboBox cmblinewidth, ComboBox cmbmarker, ComboBox cmbmarkerwidth, ComboBox cmbType,
Button colorBtn)> inputs = new Dictionary<string, (TextBox, TextBox, ComboBox, ComboBox, ComboBox, ComboBox, ComboBox, Button)>();
        public Dictionary<string, VariableRange> ResultRanges { get; private set; }

        private string currentLatexCode = ««;

        private string _savedLatexCode = ««;

        private string _cachedCalculationResult = null;
        public UserControl1()
        {

            InitializeComponent();

            InitFloatingButton();

            GeneratePalette();
            _debounceTimer = new System.Windows.Forms.Timer();
            _debounceTimer.Interval = 400;
            _debounceTimer.Tick += DebounceTimer_Tick;
            try
            {
                PythonInterop.Initialize();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }

        private void RunPyCalculation()
        {
            var config = GetCalculationConfig();
            string formulaToSend = config.Formula;
            if (_cmbMatrixOp != null && _cmbMatrixOp.Visible && _cmbMatrixOp.SelectedItem != null)
            {
                string selectedOp = _cmbMatrixOp.SelectedItem.ToString();
                string safeFormula = $»({config.Formula})»;
                string cleanReal = «.replace(conjugate, Lambda(_v, _v))»;
                switch (selectedOp)
                {
                    case «Determinant (det)»: formulaToSend = $»det({safeFormula}){cleanReal}»; break;
                    case «Trace (tr)»: formulaToSend = $»trace({safeFormula}){cleanReal}»; break;
                    case «Norm (Frobenius)»: formulaToSend = $»{safeFormula}.norm(){cleanReal}»; break;
                    case «Norm (Spectral)»: formulaToSend = $»{safeFormula}.norm(2){cleanReal}»; break;
                    case «Singular Values (SVD)»: formulaToSend = $»({safeFormula}.singular_values())[0]{cleanReal}»; break;
                    case «Grammian Det (det(A.T*A))»: formulaToSend = $»det({safeFormula}.T * {safeFormula}){cleanReal}»; break;
                    case «Pseudo-Inverse (pinv)»: formulaToSend = $»{safeFormula}.pinv().norm(){cleanReal}»; break;
                    default: formulaToSend = $»det({safeFormula}){cleanReal}»; break;
                }
            }
            string jsonConfig = JsonSerializer.Serialize(config);
            string outputDataPath;
            if (!string.IsNullOrEmpty(this.ProjectFilePath))
            {
                outputDataPath = this.ProjectFilePath;
            }
            else
            {
                string tabTitle = «NewProject»;
```

```
    Control current = this.Parent;
    while (current != null)
    {
        if (current is TabPage page)
        {
            tabTitle = page.Text.Replace(«+», ««).Trim();
            if (string.IsNullOrEmpty(tabTitle)) tabTitle = «NewProject»;
            break;
        }
        current = current.Parent;
    }
    string projectDir = Path.Combine(Application.StartupPath, «Projects», tabTitle);
    if (!Directory.Exists(projectDir))
    {
        Directory.CreateDirectory(projectDir);
    }

    outputDataPath = Path.Combine(projectDir, tabTitle + «.UCH»);
}

string mainVar = «x»;
double start = -10, end = 10;
if (config.Variables.Count > 0)
{
    var xVar = config.Variables.FirstOrDefault(v => v.Name == «x»);
    if (xVar != null) { mainVar = xVar.Name; start = xVar.Start; end = xVar.End; }
    else { mainVar = config.Variables[0].Name; start = config.Variables[0].Start; end = config.Variables[0].End; }
}
Cursor.Current = Cursors.WaitCursor;
string result = PythonInterop.RunCalculationScript(
    formulaToSend,
    mainVar,
    start,
    end,
    config.Steps,
    jsonConfig,
    outputDataPath
);
Cursor.Current = Cursors.Default;
this.Invoke(new Action(() => {
    var owner = this.FindForm();
    if (result == «OK»)
    {
        string baseName = Path.GetFileNameWithoutExtension(outputDataPath);
        MessageBox.Show(owner,
            $»Розрахунок завершено!\n» +
            $»Файли збережено в:\n{Path.GetDirectoryName(outputDataPath)}\n\n» +
            $»Створено:\n» +
            $»- {Path.GetFileName(outputDataPath)}\n» +
            $»- {baseName}.f90\n» +
            $»- {baseName}.OUT»,
            «Успіх», MessageBoxButtons.OK, MessageBoxIcon.Information);
        try
        {
            string chartScript = Path.Combine(Application.StartupPath, «universal_chart.py»);
            if (File.Exists(chartScript))
            {
                var res = MessageBox.Show(owner, «Відкрити побудований графік?», «Графік», MessageBoxButtons.YesNo, MessageBoxIcon.Question);
                if (res == DialogResult.Yes)
                {
                    ProcessStartInfo psi = new ProcessStartInfo();
                    psi.FileName = «python»;
                    psi.Arguments = $»\»{chartScript}\» \»{outputDataPath}\»«;
                    psi.UseShellExecute = true;
                    psi.CreateNoWindow = false;
                    Process.Start(psi);
                }
            }
        }
        catch (Exception ex)
```

```
            {
                MessageBox.Show(«Не вдалося запустити перегляд графіка: « + ex.Message);
            }
        }
        else
        {
            MessageBox.Show(owner, $»Помилка при розрахунку:\n{result}»,
                    «Помилка», MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }));
}
private bool pageLoaded = false;

public void SaveProject()
{

    if (string.IsNullOrEmpty(ProjectFilePath))
    {
        throw new InvalidOperationException(«Шлях до файлу не встановлено. Використовуйте ‘Зберегти як’.»);
    }

    string content = GetUchContent();

    try
    {
        File.WriteAllText(ProjectFilePath, content, Encoding.UTF8);
        IsDirty = false;
        ProjectSaved?.Invoke(this, ProjectFilePath);
    }
    catch (Exception ex)
    {
        throw new IOException($»Не вдалося записати файл на диск: {ex.Message}», ex);
    }
}
public void UpdateProjectPath(string newPath)
{
    ProjectFilePath = newPath;
}
private void SetDefaultValues()
{
    tb_ChartName.Text = «My Chart»;
    tb_Iterations.Text = «1000»;
    tb_Steps.Text = «0,01»;
    tb_xLabelName.Text = «Time»;
    tb_yLabelName.Text = «Value»;
    currentLatexCode = ««;
}
public void LoadFromFile(string uchFilePath)
{
    if (!File.Exists(uchFilePath)) return;
    ProjectFilePath = uchFilePath;
    try
    {
        string[] lines = File.ReadAllLines(uchFilePath);
        if (lines.Length == 0 || (lines.Length == 1 && lines[0].Trim() == «{}»))
        {
            SetDefaultValues();

            return;
        }
        string latexVal = ««;
        foreach (string line in lines)
        {
            string cleanLine = line.Trim();
            if (cleanLine.StartsWith(«chart «))
            {
                tb_ChartName.Text = cleanLine.Substring(6).Trim();
            }
            else if (cleanLine.StartsWith(«latex «))
            {
```

```
                latexVal = cleanLine.Substring(6).Trim();
                currentLatexCode = latexVal;
                _originalLatexCode = latexVal;
                IsDirty = false;
            }
            else if (cleanLine.StartsWith(«xlabel «))
            {
                tb_xLabelName.Text = cleanLine.Substring(7).Trim();
            }
            else if (cleanLine.StartsWith(«ylabel «))
            {
                tb_yLabelName.Text = cleanLine.Substring(7).Trim();
            }
            else if (cleanLine.StartsWith(«iterations «))
            {
                tb_Iterations.Text = cleanLine.Substring(11).Trim();
            }
            else if (cleanLine.StartsWith(«steps «))
            {
                string sVal = cleanLine.Substring(6).Trim();
                tb_Steps.Text = sVal.Replace(«.», «,»);
            }
        }
        if (!string.IsNullOrEmpty(latexVal))
        {
            InjectLatexToWebView(latexVal);
        }
        IsDirty = false;
    }
    catch (Exception ex)
    {
        MessageBox.Show($»Ошибка чтения файла: {ex.Message}»);
    }
    finally
    {
    }
}

private async void UserControl1_Load(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(ProjectFilePath))
    {

        tb_ChartName.Text = «My Chart»;

        tb_Iterations.Text = «1000»;

        tb_Steps.Text = «0.01»;

        tb_xLabelName.Text = «Time»;

        tb_yLabelName.Text = «Value»;

    }
    if (webView21.CoreWebView2 != null) return;
    try
    {

        await webView21.EnsureCoreWebView2Async(null);

        webView21.CoreWebView2.WebMessageReceived += CoreWebView2_WebMessageReceived;

        if (this.IsDisposed || webView21.IsDisposed) return;

        webView21.CoreWebView2.NavigationCompleted += webView21_NavigationCompleted;

        string htmlFilePath = Path.Combine(Application.StartupPath, «Latex.html»);

        if (!File.Exists(htmlFilePath))
```

```
        {
            MessageBox.Show(«Ошибка: MathEditor.html не найден»);
            return;
        }
        string url = $»file:///{htmlFilePath.Replace(«\\», «/»)}»;
        webView21.CoreWebView2.Navigate(url);
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine($»WebView init error: {ex.Message}»);
    }
    tb_ChartName.TextChanged += MarkAsDirty;
    tb_Iterations.TextChanged += MarkAsDirty;
    tb_Steps.TextChanged += MarkAsDirty;
    tb_xLabelName.TextChanged += MarkAsDirty;
    tb_yLabelName.TextChanged += MarkAsDirty;
}

private string _cachedLatex = ««;

private string _cachedMathML = ««;
private ExprNode _currentRootAst = null;
void BuildVisualTree(ExprNode node, Control parent)
{
    if (node == null) return;
    var nodeControl = new HierarchicalNodeControl();
    Action<TextBox> styleBox = (tb) =>
    {
        tb.BorderStyle = BorderStyle.FixedSingle;
        tb.BackColor = Color.WhiteSmoke;
        tb.Font = new Font(«Consolas», 9f);
    };
    switch (node)
    {
        case ChainNode chain:
            {
                string title = chain.Kind == «AddChain» ? «Chain (Sum)» : «Chain (Prod)»;
                nodeControl.Title = title;

                var (stylePanel, _) = CreateRichControlTable(title, chain);
                nodeControl.AddControl(stylePanel);
                var flow = new FlowLayoutPanel { AutoSize = true, FlowDirection = FlowDirection.TopDown, Margin = new Padding(10, 0, 0, 0), WrapContents = false };
                foreach (var item in chain.Items)
                {
                    if (item is string opSymbol)
                    {
                        flow.Controls.Add(new Label { Text = $»[ {opSymbol} ]», AutoSize = true, Font = new Font(«Consolas», 10f, FontStyle.Bold), ForeColor = Color.CornflowerBlue });
                    }
                    else if (item is ExprNode subNode)
                    {
                        Panel wrapper = new Panel { AutoSize = true };
                        BuildVisualTree(subNode, wrapper);
                        flow.Controls.Add(wrapper);
```

```
            }
        }
        nodeControl.AddControl(flow);
        break;
    }
case BigOpNode bigOp:
    {
        var (stylePanel, _) = CreateRichControlTable(bigOp.Kind, bigOp);
        nodeControl.AddControl(stylePanel);
        break;
    }
case DerivativeNode d:
    {
        string title = «Похідна (d/d» + d.Variable + «)»;
        nodeControl.Title = title;
        var (stylePanel, _) = CreateRichControlTable(title, d);
        nodeControl.AddControl(stylePanel);
        var paramPanel = new FlowLayoutPanel
        {
            AutoSize = true,
            FlowDirection = FlowDirection.LeftToRight,

        };
        string labelText = «Order:»;
        bool isNumeric = double.TryParse(d.Order, out _);

        if (!isNumeric)
        {
            labelText = $»Order {d.Order}:»;
        }

        paramPanel.Controls.Add(new Label
        {
            Text = labelText,
            AutoSize = true,
            TextAlign = System.Drawing.ContentAlignment.MiddleLeft,
            Anchor = AnchorStyles.Left,
            Padding = new Padding(0, 6, 0, 0)
        });

        TextBox txtOrder = new TextBox
        {

            Text = isNumeric ? d.Order : ««,
            Width = 40,
            TextAlign = HorizontalAlignment.Center,
            Tag = d
        };
        styleBox(txtOrder);

        txtOrder.TextChanged += (s, args) =>
        {
            d.Order = string.IsNullOrWhiteSpace(txtOrder.Text) ? ««  : txtOrder.Text;
            MarkAsDirty(s, args);
        };

        paramPanel.Controls.Add(txtOrder);

        if (!string.IsNullOrEmpty(d.Condition))
        {
            var parts = d.Condition.Split(‘=‘);
            string rightVal = parts.Length > 1 ? parts[1].Trim() : ««;

            paramPanel.Controls.Add(new Label
            {
                Text = « at « + (parts.Length > 0 ? parts[0] : «x») + «=«,
                AutoSize = true,
                TextAlign = System.Drawing.ContentAlignment.MiddleLeft,
                Anchor = AnchorStyles.Left,
                Padding = new Padding(0, 6, 0, 0)
```

```
            });

            var tbCond = new TextBox { Width = 60, Text = rightVal };
            styleBox(tbCond);
            paramPanel.Controls.Add(tbCond);
        }

        if (paramPanel.Controls.Count > 0)
            nodeControl.AddControl(paramPanel);

        if (d.Body != null)
            BuildVisualTree(d.Body, nodeControl);

        break;
    }
case OperationNode op:
    {
        nodeControl.Title = op.Op;


        var (stylePanel, _) = CreateRichControlTable(op.Op, op);
        nodeControl.AddControl(stylePanel);
        foreach (var a in op.Args) BuildVisualTree(a, nodeControl);
        break;
    }
case MatrixNode m:
    {
        nodeControl.Title = «Матрица»;
        var (stylePanel, _) = CreateRichControlTable(«Матрица», m);
        nodeControl.AddControl(stylePanel);


        var container = new FlowLayoutPanel
        {
            AutoSize = true,
            AutoSizeMode = AutoSizeMode.GrowAndShrink,
            FlowDirection = FlowDirection.LeftToRight,
            WrapContents = false,
            VerticalScroll = { Visible = false }
        };

        var table = new TableLayoutPanel();
        table.AutoSize = true;
        table.CellBorderStyle = TableLayoutPanelCellBorderStyle.Single;
        int rowCount = m.Rows.Count;
        int colCount = rowCount > 0 ? m.Rows[0].Count : 0;
        table.RowCount = rowCount;
        table.ColumnCount = colCount;
        for (int r = 0; r < rowCount; r++)
        {
            for (int c = 0; c < colCount; c++)
            {
                if (c < m.Rows[r].Count)
                {
                    string cellText = AstToFormulaConverter.Convert(m.Rows[r][c]);
                    Label l = new Label
                    {
                        Text = cellText,
                        AutoSize = true,
                        Padding = new Padding(5),
                        Font = new Font(«Consolas», 10f)
                    };
                    table.Controls.Add(l, c, r);
                }
            }
        }
        container.Controls.Add(table);
        if (rowCount > 0)
        {
            var cmbOp = new ComboBox();
```

```
                cmbOp.DropDownStyle = ComboBoxStyle.DropDownList;
                cmbOp.Width = 180;
                cmbOp.Margin = new Padding(15, rowCount * 10, 0, 0);

                if (rowCount == colCount)
                {
                    cmbOp.Items.Add(«Determinant (det)»);
                    cmbOp.Items.Add(«Trace (tr)»);
                    cmbOp.Items.Add(«Norm (Frobenius)»);
                }
                else
                {
                    cmbOp.Items.Add(«Norm (Frobenius)»);

                    cmbOp.Items.Add(«Norm (Spectral)»);

                    cmbOp.Items.Add(«Singular Values (SVD)»);

                    cmbOp.Items.Add(«Grammian Det (det(A.T*A))»);
                    cmbOp.Items.Add(«Pseudo-Inverse (pinv)»);
                }
                if (cmbOp.Items.Count > 0) cmbOp.SelectedIndex = 0;
                container.Controls.Add(cmbOp);
                _cmbMatrixOp = cmbOp;
            }
            nodeControl.AddControl(container);
            break;
        }
    case ValueNode v:
        {
            nodeControl.AddControl(new Label { Text = v.Name, AutoSize = true, Font = new Font(«Consolas», 10f), Margin = new Padding(20, 5, 0, 5) });
            break;
        }
  }
  if (parent is HierarchicalNodeControl hParent) hParent.AddControl(nodeControl);
  else parent.Controls.Add(nodeControl);
}
private (TableLayoutPanel, NodeStyleControls) CreateRichControlTable(string name, ExprNode node = null)
{
  var table = new TableLayoutPanel();
  table.AutoSize = true;
  table.AutoSizeMode = AutoSizeMode.GrowAndShrink;
  table.Padding = new Padding(0, 0, 0, 5);
  table.Margin = new Padding(0);

  table.ColumnCount = 6;
  for (int i = 0; i < 6; i++) table.ColumnStyles.Add(new ColumnStyle(SizeType.AutoSize));
  table.RowCount = 2;
  Action<string, int> addHeader = (txt, col) => {
    table.Controls.Add(new Label
    {
      Text = txt,
      Font = new Font(«Segoe UI», 7f),
      ForeColor = Color.Gray,
      AutoSize = true,
      Margin = new Padding(0, 0, 0, 2)
    }, col, 0);
  };
  addHeader(««, 0);
  addHeader(«Line», 1);
  addHeader(«Width», 2);
  addHeader(«Marker», 3);
  addHeader(«Size», 4);
  addHeader(«Color», 5);
  Label lblName = new Label
  {
    Text = name,
    AutoSize = true,
    Font = new Font(«Segoe UI», 9f, FontStyle.Bold),
    Anchor = AnchorStyles.Left,
```

```
    Margin = new Padding(0, 5, 10, 0)
};
table.Controls.Add(lblName, 0, 1);

Action<ComboBox, string[], int> setupCmb = (cmb, items, idx) => {
    cmb.DropDownStyle = ComboBoxStyle.DropDownList;
    cmb.Items.AddRange(items);
    cmb.SelectedIndex = idx;
    cmb.Width = 45;
    cmb.Margin = new Padding(0, 3, 5, 0);
    cmb.SelectedIndexChanged += MarkAsDirty;
};
var cmbLine = new ComboBox();
setupCmb(cmbLine, new string[] { ««, «-», «- -», «-*-», «***» }, 1);
table.Controls.Add(cmbLine, 1, 1);
var cmbWidth = new ComboBox();
setupCmb(cmbWidth, new string[] { «0.5», «1.0», «1.5», «2.0», «3.0» }, 1);
table.Controls.Add(cmbWidth, 2, 1);

var cmbMarker = new ComboBox { Font = new Font(«MS Gothic», 9f) };
setupCmb(cmbMarker, new string[] { ««, «○», «□», «Δ», «◊» }, 0);
table.Controls.Add(cmbMarker, 3, 1);
cmbMarker.SelectedIndex = 1;

var cmbMSize = new ComboBox();
setupCmb(cmbMSize, new string[] { «2», «4», «6», «8» }, 2);
table.Controls.Add(cmbMSize, 4, 1);

var btnColor = new Button
{
    Size = new Size(22, 22),
    FlatStyle = FlatStyle.Flat,
    Margin = new Padding(0, 2, 0, 0)
};

Color newCol = (_colorPalette.Count > 0) ? _colorPalette[_nextColorIndex % _colorPalette.Count] : Color.Black;
if (node != null)
{
    int safeGuard = 0;
    while (_variableColorMemory.ContainsValue(newCol) && safeGuard < 100)
    {
        _nextColorIndex++;
        if (_colorPalette.Count > 0)
            newCol = _colorPalette[_nextColorIndex % _colorPalette.Count];
        safeGuard++;
    }
}
if (_colorPalette.Count > 0) _nextColorIndex++;
btnColor.BackColor = newCol;
btnColor.Click += (s, e) => {
    using (var cd = new ColorDialog())
    {
        cd.Color = btnColor.BackColor;
        if (cd.ShowDialog() == DialogResult.OK)
        {
            btnColor.BackColor = cd.Color;
            MarkAsDirty(s, e);
        }
    }
};
table.Controls.Add(btnColor, 5, 1);
var settings = new NodeStyleControls
{
    LineType = cmbLine,
    LineWidth = cmbWidth,
    MarkerType = cmbMarker,
    MarkerSize = cmbMSize,
    ColorBtn = btnColor
};
if (node != null)
```

```
    {
        _nodeStyles[node] = settings;
    }
    return (table, settings);
}

static bool IsComputeEngineType(string name)

{

    return name switch

    {

        «number» => true,

        «list-number» => true,

        «incompatible-type» => true,

        «List» => true,

        «Matrix» => true,

        «Add» => true,

        «Multiply» => true,

        «Power» => true,

        _ => false

    };

}

void CollectVariables(ExprNode node, HashSet<string> vars)
{
    if (node == null) return;
    switch (node)
    {
        case ChainNode chain:
            foreach (var item in chain.Items)
            {
                if (item is ExprNode en)
                {
                    CollectVariables(en, vars);
                }
                else if (item is string s)
                {
                    if (IsRealVariable(s))
                    {
                        vars.Add(s);
                    }
                }
            }
            break;
        case BigOpNode bigOp:
            if (!string.IsNullOrEmpty(bigOp.Index) && IsRealVariable(bigOp.Index))
            {
                vars.Add(bigOp.Index);
            }
            CollectVariables(bigOp.Lower, vars);
            CollectVariables(bigOp.Upper, vars);
            CollectVariables(bigOp.Body, vars);
            break;
        case ValueNode v:
            if (IsRealVariable(v.Name)) vars.Add(v.Name);
            break;
        case OperationNode op:
```

```
            foreach (var a in op.Args) CollectVariables(a, vars);
            break;
        case DerivativeNode d:
            CollectVariables(d.Body, vars);
            vars.Add(d.Variable);
            break;
        case MatrixNode m:
            foreach (var row in m.Rows)
                foreach (var cell in row) CollectVariables(cell, vars);
            break;
    }
}

bool IsNumber(string s) =>

double.TryParse(s,

System.Globalization.NumberStyles.Any,

System.Globalization.CultureInfo.InvariantCulture,

out _);

private void ScanForBigOpDefaults(ExprNode node, Dictionary<string, string> rules)
{
    if (node == null) return;
    if (node is BigOpNode big)
    {
        if (!string.IsNullOrEmpty(big.Index) && big.Lower != null)
        {
            string valStr = AstToFormulaConverter.Convert(big.Lower);
            if (double.TryParse(valStr, System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out _))
            {
                if (!rules.ContainsKey(big.Index))
                {
                    rules[big.Index] = valStr;
                }
            }
        }
        if (big.Upper is ValueNode vnUpper && IsRealVariable(vnUpper.Name))
        {
            if (!rules.ContainsKey(vnUpper.Name))
            {
                rules[vnUpper.Name] = «10»;
            }
        }
        if (big.Kind == «Integral» && big.Lower is ValueNode vnLower && IsRealVariable(vnLower.Name))
        {
            if (!rules.ContainsKey(vnLower.Name))
            {
                rules[vnLower.Name] = «-10»;
            }
        }
        ScanForBigOpDefaults(big.Body, rules);
        ScanForBigOpDefaults(big.Lower, rules);
        ScanForBigOpDefaults(big.Upper, rules);
    }
    else if (node is OperationNode op)
    {
        foreach (var arg in op.Args) ScanForBigOpDefaults(arg, rules);
    }
    else if (node is ChainNode chain)
    {
        foreach (var item in chain.Items)
            if (item is ExprNode en) ScanForBigOpDefaults(en, rules);
    }
    else if (node is DerivativeNode dn)
    {
        ScanForBigOpDefaults(dn.Body, rules);
    }
```

```
    else if (node is MatrixNode m)
    {
        foreach (var row in m.Rows)
            foreach (var cell in row) ScanForBigOpDefaults(cell, rules);
    }
}
ExprNode BuildExprNode(JsonElement el)
{
    if (el.ValueKind == JsonValueKind.Null || el.ValueKind == JsonValueKind.Undefined) return null;
    if (el.ValueKind == JsonValueKind.Array || el.ValueKind == JsonValueKind.String || el.ValueKind == JsonValueKind.Number)
    {
        return ParseMathJson(el);
    }
    if (el.ValueKind == JsonValueKind.Object && el.TryGetProperty(«kind», out var kindProp))
    {
        string kind = kindProp.GetString();
        if (kind == «Normal»)
            return ParseMathJson(el.GetProperty(«mathjson»));
        if (kind == «AddChain» || kind == «MulChain»)
        {
            var node = new ChainNode { Kind = kind };
            foreach (var item in el.GetProperty(«items»).EnumerateArray())
            {
                if (item.ValueKind == JsonValueKind.String) node.Items.Add(item.GetString());
                else node.Items.Add(BuildExprNode(item));
            }
            return node;
        }
        if (kind == «Derivative»)
        {
            return new DerivativeNode
            {
                Variable = el.GetProperty(«variable»).GetString(),
                Order = el.GetProperty(«order»).GetString(),
                Condition = el.TryGetProperty(«conditionLatex», out var c) ? c.GetString() : null,
                Body = BuildExprNode(el.GetProperty(«body»))
            };
        }
        if (kind == «Integral» || kind == «Sum» || kind == «Product»)
        {
            return new BigOpNode
            {
                Kind = kind,
                Index = el.TryGetProperty(«index», out var idx) ? idx.GetString() : null,
                IntVar = el.TryGetProperty(«intVar», out var iv) ? iv.GetString() : null,
                Lower = el.TryGetProperty(«lower», out var l) ? BuildExprNode(l) : null,
                Upper = el.TryGetProperty(«upper», out var u) ? BuildExprNode(u) : null,
                Body = el.TryGetProperty(«body», out var b) ? BuildExprNode(b) : null,
                Condition = el.TryGetProperty(«conditionLatex», out var c) ? c.GetString() : null
            };
        }
    }
    return ParseMathJson(el);
}
private void DebounceTimer_Tick(object sender, EventArgs e)
{
    _debounceTimer.Stop();
    if (string.IsNullOrEmpty(_pendingJsonData)) return;
    try
    {
        using var doc = JsonDocument.Parse(_pendingJsonData);
        var rootElement = doc.RootElement;
        if (!rootElement.TryGetProperty(«type», out var typeProp) || typeProp.GetString() != «Expression») return;
        var rootNodeJson = rootElement.GetProperty(«root»);
        if (rootNodeJson.ValueKind == JsonValueKind.Null)
        {
            Invoke(new Action(() => {
                flowLayoutPanel1.Controls.Clear();
                _currentRootAst = null;
                if (_floatingButton != null) _floatingButton.Visible = false;
```

```
            }));
            return;
        }
        var ast = BuildExprNode(rootNodeJson);
        _currentRootAst = ast;
        Invoke(new Action(() =>
        {
            flowLayoutPanel1.SuspendLayout();
            try
            {
                flowLayoutPanel1.Controls.Clear();
                _nodeStyles.Clear();
                inputs.Clear();
                _nextColorIndex = 0;
                BuildVisualTree(ast, flowLayoutPanel1);
                var vars = new HashSet<string>();
                CollectVariables(ast, vars);
                BuildVariablesPanel(vars, flowLayoutPanel1);
                if (_floatingButton != null)
                {
                    _floatingButton.Visible = true;
                    _floatingButton.BringToFront();
                }
            }
            finally { flowLayoutPanel1.ResumeLayout(); }
        }));
    }
    catch (Exception ex) { Debug.WriteLine(ex.Message); }
}

private void CoreWebView2_WebMessageReceived(object sender, CoreWebView2WebMessageReceivedEventArgs e)
{
    _pendingJsonData = e.WebMessageAsJson;
    _debounceTimer.Stop();
    _debounceTimer.Start();
    using var doc = JsonDocument.Parse(e.WebMessageAsJson);
    var rootElement = doc.RootElement;

    if (!rootElement.TryGetProperty(«type», out var typeProp) || typeProp.GetString() != «Expression»)
        return;

    var rootNodeJson = rootElement.GetProperty(«root»);

    if (rootNodeJson.ValueKind == JsonValueKind.Null)
    {
        Invoke(new Action(() =>
        {
            flowLayoutPanel1.Controls.Clear();
            currentLatexCode = ««;
            _currentRootAst = null;

            if (_floatingButton != null) _floatingButton.Visible = false;
            if (_cmbMatrixOp != null) _cmbMatrixOp.Visible = false;

        }));
        return;
    }

    var ast = BuildExprNode(rootNodeJson);
    _currentRootAst = ast;
    Invoke(new Action(() =>
    {
        _cmbMatrixOp = null;
        BuildVisualTree(ast, flowLayoutPanel1);

        var vars = new HashSet<string>();
        CollectVariables(ast, vars);
        this.Invoke(new Action(() =>
        {
            if (!IsDirty)
```

```
            {
                IsDirty = true;
            }
            _cachedCalculationResult = null;
        }));

        BuildVariablesPanel(vars, flowLayoutPanel1);
        if (_floatingButton != null)
        {
            _floatingButton.Visible = true;
            _floatingButton.BringToFront();
        }
    }));
}
private string _originalLatexCode = ««;

bool TryParseCondition(string condition, out string left, out string right)

{

    left = null;

    right = null;

    if (string.IsNullOrWhiteSpace(condition))

        return false;

    var parts = condition.Split('=');

    if (parts.Length != 2)

        return false;

    left = parts[0].Trim();

    right = parts[1].Trim();

    return true;

}


private void webView21_NavigationCompleted(object sender, CoreWebView2NavigationCompletedEventArgs e)

{

    pageLoaded = true;

    if (!string.IsNullOrEmpty(currentLatexCode))

    {

        InjectLatexToWebView(currentLatexCode);

    }

}
/*
                                private TableLayoutPanel CreateVariableGrid(List<string> vars)

                                {

                                        TableLayoutPanel grid = new TableLayoutPanel();

                                        grid.AutoSize = true;

                                        grid.Dock = DockStyle.Top;

                                        grid.ColumnCount = 9;
```

```
grid.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 20F));
grid.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 20F));
grid.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 20F));
grid.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 20F));
grid.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 20F));
grid.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 20F));
grid.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 20F));
grid.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 20F));
grid.RowCount = vars.Count + 1;
AddHeader(grid, «Var», 0, 0);
AddHeader(grid, «Min», 1, 0);
AddHeader(grid, «Max», 2, 0);
AddHeader(grid, «Лінія», 3, 0);
AddHeader(grid, «Товщина», 4, 0);
AddHeader(grid, «Маркер», 5, 0);
AddHeader(grid, «Товщина», 6, 0);
AddHeader(grid, «Колір», 7, 0);
int row = 1;
int varIndex = 0;
foreach (string varName in vars)
{
    Label lbl = new Label { Text = varName, TextAlign = System.Drawing.ContentAlignment.MiddleLeft, Font = new Font(this.Font, FontStyle.Bold), AutoSize = true };
    TextBox txtStart = new TextBox { Text = «-10», Dock = DockStyle.Fill };
    TextBox txtEnd = new TextBox { Text = «10», Dock = DockStyle.Fill };
    txtStart.TextChanged += MarkAsDirty;
    txtEnd.TextChanged += MarkAsDirty;
    ComboBox cmbLineStyle = new ComboBox { Font = new Font(«MS Gothic», 10f, FontStyle.Regular), DropDownStyle = ComboBoxStyle.DropDownList, Dock = DockStyle.Fill, Anchor = AnchorStyles.Left | AnchorStyles.Top };
    cmbLineStyle.Items.AddRange(new object[] { ««, «-», «- -», «-*-», «***» });
    cmbLineStyle.SelectedIndex = 1;
    ComboBox cmbLineWidth = new ComboBox { DropDownStyle = ComboBoxStyle.DropDownList, Dock = DockStyle.Fill, Anchor = AnchorStyles.Left | AnchorStyles.Top };
    cmbLineWidth.Items.AddRange(new object[] { «0.5», «1.0», «1.5», «2.0», «2.5», «3.0», «3.5», «4.0», «4.5», «5.0» });
    cmbLineWidth.SelectedIndex = 1;
    ComboBox cmbMarker = new ComboBox { Font = new Font(«MS Gothic», 10f, FontStyle.Bold), DropDownStyle = ComboBoxStyle.DropDownList, Dock = DockStyle.Fill, Anchor = AnchorStyles.Left | AnchorStyles.Top };
```

```
cmbMarker.Items.AddRange(new object[] { ««, «○», «□», «Δ», «◊» });
cmbMarker.SelectedIndex = 1;
ComboBox cmbMarkerWidth = new ComboBox { DropDownStyle = ComboBoxStyle.DropDownList, Dock = DockStyle.Fill, Anchor = AnchorStyles.Left | AnchorStyles.Top };
cmbMarkerWidth.Items.AddRange(new object[] { «2», «4», «6», «8», «10», «12», «15» });
cmbMarkerWidth.SelectedIndex = 1;
cmbLineStyle.SelectedIndexChanged += MarkAsDirty;
cmbLineWidth.SelectedIndexChanged += MarkAsDirty;
cmbMarker.SelectedIndexChanged += MarkAsDirty;
cmbMarkerWidth.SelectedIndexChanged += MarkAsDirty;
Button clrBtn = new Button();
Color assignedColor;
if (_variableColorMemory.ContainsKey(varName))
{
        assignedColor = _variableColorMemory[varName];
}
else
{
        if (_colorPalette.Count > 0)
        {
                assignedColor = _colorPalette[_nextColorIndex % _colorPalette.Count];
                _nextColorIndex++;
        }
        else
        {
                assignedColor = Color.Black;
        }
        _variableColorMemory[varName] = assignedColor;
}
clrBtn.BackColor = assignedColor;
clrBtn.FlatStyle = FlatStyle.Flat;
clrBtn.Size = new Size(20, 20);
clrBtn.MinimumSize = new Size(20, 20);
clrBtn.MaximumSize = new Size(20, 20);
clrBtn.Anchor = AnchorStyles.Left | AnchorStyles.Top;
```

```
                                        clrBtn.Margin = new Padding(3, 3, 0, 0);
                                        clrBtn.Click += (sender, e) =>
                                        {
                                                ColorDialog colorDialog = new ColorDialog();
                                                colorDialog.Color = clrBtn.BackColor;
                                                colorDialog.FullOpen = true;
                                                if (colorDialog.ShowDialog() == DialogResult.OK)
                                                {
                                                        clrBtn.BackColor = colorDialog.Color;
                                                        _variableColorMemory[varName] = colorDialog.Color;
                                                        MarkAsDirty(sender, e);
                                                }
                                        };
                                        if (varName != «x» && varName != «t»)
                                        {
                                                txtStart.Text = «-10»; txtEnd.Text = «10»;
                                        }
                                        grid.Controls.Add(lbl, 0, row);
                                        grid.Controls.Add(txtStart, 1, row);
                                        grid.Controls.Add(txtEnd, 2, row);
                                        grid.Controls.Add(cmbLineStyle, 3, row);
                                        grid.Controls.Add(cmbLineWidth, 4, row);
                                        grid.Controls.Add(cmbMarker, 5, row);
                                        grid.Controls.Add(cmbMarkerWidth, 6, row);
                                        grid.Controls.Add(clrBtn, 7, row);
                                        inputs[varName] = (txtStart, txtEnd, cmbLineStyle, cmbLineWidth, cmbMarker, cmbMarkerWidth, clrBtn);
                                        row++;
                                }
                                return grid;
                        }
                        */
private void AddHeader(TableLayoutPanel grid, string text, int col, int row)
{
    grid.Controls.Add(new Label { Text = text, AutoSize = true, ForeColor = Color.Black }, col, row);
}
private async void InjectLatexToWebView(string latex)
```

```
{
    if (webView21.CoreWebView2 == null || !pageLoaded) return;
    if (string.IsNullOrWhiteSpace(latex)) return;
    try
    {

        await webView21.CoreWebView2.ExecuteScriptAsync($»document.getElementById(‘mf’).setValue(‘{latex}’);»);
    }
    catch (Exception ex)
    {
        Debug.WriteLine(«Ошибка отправки в WebView: « + ex.Message);
    }
}

private void UserControl1_DragEnter(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop))
    {
        e.Effect = DragDropEffects.Copy;
    }
    else
    {
        e.Effect = DragDropEffects.None;
    }
}
private void UserControl1_DragDrop(object sender, DragEventArgs e)
{
    string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);
    foreach (string file in files)
    {
        MessageBox.Show(«Файл: « + file);
    }
}
private void MarkAsDirty(object sender, EventArgs e)
{
    if (!IsDirty)
```

```
    {
        IsDirty = true;
    }
    _cachedCalculationResult = null;
}
public string ChartNameText => tb_ChartName.Text;
public string IterationsText => tb_Iterations.Text;
public string StepsText => tb_Steps.Text;
public string XLabelText => tb_xLabelName.Text;
public string YLabelText => tb_yLabelName.Text;
public bool HasEmptyCriticalFields
{
    get
    {
        return string.IsNullOrWhiteSpace(tb_ChartName.Text) ||
               string.IsNullOrWhiteSpace(tb_Iterations.Text) ||
               string.IsNullOrWhiteSpace(tb_Steps.Text);
    }
}
public string GetUchContent()
{
    StringBuilder sb = new StringBuilder();
    string chartName = string.IsNullOrWhiteSpace(tb_ChartName.Text) ? «My Chart» : tb_ChartName.Text.Trim();
    string xLab = string.IsNullOrWhiteSpace(tb_xLabelName.Text) ? «Time» : tb_xLabelName.Text.Trim();
    string yLab = string.IsNullOrWhiteSpace(tb_yLabelName.Text) ? «Value» : tb_yLabelName.Text.Trim();
    string latexToSave = string.IsNullOrWhiteSpace(currentLatexCode) ? «« : currentLatexCode;
    sb.AppendLine($»chart {chartName}»);
    sb.AppendLine($»latex {latexToSave}»);
    sb.AppendLine($»xlabel {xLab}»);
    sb.AppendLine($»ylabel {yLab}»);

    sb.AppendLine(«xgrid 110;10;1»);
    sb.AppendLine(«ygrid 110;10;1»);
    sb.AppendLine(«legend 0; 1»);

    sb.AppendLine($»curves {inputs.Count:D9}»);
```

```
if (!int.TryParse(tb_Iterations.Text, out int gIter)) gIter = 1000;

string stepText = tb_Steps.Text.Replace(«,», «.»);
if (!double.TryParse(stepText, System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out double gStep)) gStep = 0.01;
sb.AppendLine($»iterations {gIter}»);
sb.AppendLine($»steps {gStep.ToString(System.Globalization.CultureInfo.InvariantCulture)}»);

foreach (var kvp in inputs)
{
    string varName = kvp.Key;
    var controls = kvp.Value;
    sb.AppendLine($»curve {varName.ToUpper()}»);

    string sStart = controls.start.Text.Replace(«,», «.»);
    string sEnd = controls.end.Text.Replace(«,», «.»);
    if (!double.TryParse(sStart, System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out double dStart)) dStart = -10.0;
    if (!double.TryParse(sEnd, System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out double dEnd)) dEnd = 10.0;

    sb.AppendLine($»min {dStart.ToString(System.Globalization.CultureInfo.InvariantCulture)}»);
    sb.AppendLine($»max {dEnd.ToString(System.Globalization.CultureInfo.InvariantCulture)}»);
    string typeVal = controls.cmbType.SelectedItem?.ToString() ?? «Dynamic»;
    sb.AppendLine($»type {typeVal}»);

    Color c = controls.colorBtn.BackColor;
    sb.AppendLine($»color {c.R:D3}{c.G:D3}{c.B:D3}»);
    int lType = controls.cmblinestyle.SelectedIndex;
    if (lType < 0) lType = 1;
    string lWidth = controls.cmblinewidth.SelectedItem?.ToString() ?? «1.5»;
    lWidth = lWidth.Replace(«,», «.»);
    sb.AppendLine($»line {lType}{lWidth}»);

    int mType = controls.cmbmarker.SelectedIndex;
    if (mType < 0) mType = 1;
    string mSize = controls.cmbmarkerwidth.SelectedItem?.ToString() ?? «6»;
    mSize = mSize.Replace(«,», «.»);
    sb.AppendLine($»markers {mType:D2}{mSize}»);

    sb.AppendLine(«data 000000002»);
    sb.AppendLine($»{dStart.ToString(«E15», System.Globalization.CultureInfo.InvariantCulture)};0.000000000000000E+000»);
```

```
            sb.AppendLine($»{dEnd.ToString(«E15», System.Globalization.CultureInfo.InvariantCulture)};0.000000000000000E+000»);

        }

        return sb.ToString();

    }

    public class MathLivePayload

    {

        public string type { get; set; }

        public string latex { get; set; }

        public string mathml { get; set; }

        public System.Text.Json.JsonElement mathjson { get; set; }

    }

    void BuildVariablesPanel(HashSet<string> vars, Control parent)
    {
        var derivativeRules = new Dictionary<string, string>();
        ScanForDerivativeRules(_currentRootAst, derivativeRules, vars);
        ScanForBigOpDefaults(_currentRootAst, derivativeRules);
        if (vars == null || vars.Count == 0) return;
        var dependencyMap = new Dictionary<string, HashSet<string>>();
        foreach (var rule in derivativeRules)
        {
            string v1 = rule.Key;
            string v2 = rule.Value;
            if (double.TryParse(v2, out _)) continue;
            if (!dependencyMap.ContainsKey(v1)) dependencyMap[v1] = new HashSet<string>();
            if (!dependencyMap.ContainsKey(v2)) dependencyMap[v2] = new HashSet<string>();
            dependencyMap[v1].Add(v2);
            dependencyMap[v2].Add(v1);
        }

        var variablesNode = new HierarchicalNodeControl();
        variablesNode.Title = «Налаштування змінних (Parsed)»;
        TableLayoutPanel table = new TableLayoutPanel();
        table.AutoSize = true;
        table.AutoSizeMode = AutoSizeMode.GrowAndShrink;
        table.Padding = new Padding(0, 5, 0, 0);
        table.Margin = new Padding(0);
        table.ColumnCount = 9;
        for (int i = 0; i < 9; i++) table.ColumnStyles.Add(new ColumnStyle(SizeType.AutoSize));
        table.RowCount = vars.Count + 1;

        int r = 0;
        AddHeaderToTable(table, «Var», 0, r);
        AddHeaderToTable(table, «Min», 1, r);
        AddHeaderToTable(table, «Max», 2, r);
        AddHeaderToTable(table, «Line», 3, r);
        AddHeaderToTable(table, «Width», 4, r);
        AddHeaderToTable(table, «Marker», 5, r);
        AddHeaderToTable(table, «Width», 6, r);
        AddHeaderToTable(table, «Type», 7, r);
        AddHeaderToTable(table, «Color», 8, r);

        r = 1;
        inputs.Clear();
        _variableColorMemory = _variableColorMemory ?? new Dictionary<string, Color>();
        bool isSyncing = false;

        Action<string, Action<(TextBox start, TextBox end, ComboBox cmblinestyle, ComboBox cmblinewidth, ComboBox cmbmarker, ComboBox cmbmarkerwidth, ComboBox cmbType,
Button colorBtn)>> performSync = (sourceVar, applyChange) =>
```

```
                {
                    if (isSyncing) return;
                    isSyncing = true;
                    try
                    {
                        if (dependencyMap.ContainsKey(sourceVar))
                        {
                            foreach (var targetVar in dependencyMap[sourceVar])
                            {
                                if (inputs.ContainsKey(targetVar))
                                {

                                    var targetControls = inputs[targetVar];
                                    applyChange(targetControls);
                                }
                            }
                        }
                        MarkAsDirty(null, EventArgs.Empty);
                    }
                    finally
                    {
                        isSyncing = false;
                    }
                };

                foreach (var v in vars.OrderBy(x => x))
                {
                    Label lblName = new Label
                    {
                        Text = v,
                        AutoSize = true,
                        Anchor = AnchorStyles.Left | AnchorStyles.Top,
                        Font = new Font(«Segoe UI», 9f, FontStyle.Bold),
                        Margin = new Padding(0, 6, 10, 0)
                    };

                    TextBox txtMin = new TextBox { Text = «-10», Width = 40, BorderStyle = BorderStyle.FixedSingle, BackColor = Color.WhiteSmoke, Margin = new Padding(0, 3, 5, 0) };
                    TextBox txtMax = new TextBox { Text = «10», Width = 40, BorderStyle = BorderStyle.FixedSingle, BackColor = Color.WhiteSmoke, Margin = new Padding(0, 3, 5, 0) };

                    ComboBox cmbLineStyle = new ComboBox { Width = 45, DropDownStyle = ComboBoxStyle.DropDownList, Margin = new Padding(0, 3, 5, 0) };
                    cmbLineStyle.Items.AddRange(new string[] { ««, «-», «- -», «-*-», «***» });
                    cmbLineStyle.SelectedIndex = 1;

                    ComboBox cmbLineWidth = new ComboBox { Width = 45, DropDownStyle = ComboBoxStyle.DropDownList, Margin = new Padding(0, 3, 5, 0) };
                    cmbLineWidth.Items.AddRange(new string[] { «0.5», «1.0», «1.5», «2.0», «3.0» });
                    cmbLineWidth.SelectedIndex = 1;

                    ComboBox cmbMarker = new ComboBox { Width = 45, DropDownStyle = ComboBoxStyle.DropDownList, Font = new Font(«MS Gothic», 9f), Margin = new Padding(0, 3, 5, 0)
};
                    cmbMarker.Items.AddRange(new object[] { ««, «○», «□», «Δ», «◊» });
                    cmbMarker.SelectedIndex = 1;

                    ComboBox cmbMarkerSize = new ComboBox { Width = 45, DropDownStyle = ComboBoxStyle.DropDownList, Margin = new Padding(0, 3, 5, 0) };
                    cmbMarkerSize.Items.AddRange(new string[] { «2», «4», «6», «8», «10» });
                    cmbMarkerSize.SelectedIndex = 2;

                    ComboBox cmbType = new ComboBox { Width = 75, DropDownStyle = ComboBoxStyle.DropDownList, Margin = new Padding(0, 3, 5, 0) };
                    cmbType.Items.AddRange(new string[] { «Static», «Dynamic» });
                    cmbType.SelectedIndex = 1;

                    Button btnColor = new Button { Size = new Size(22, 22), FlatStyle = FlatStyle.Flat, Margin = new Padding(3, 2, 0, 0), UseVisualStyleBackColor = false };

                    if (_variableColorMemory.TryGetValue(v, out Color varColor))
                    {
                        btnColor.BackColor = varColor;
                    }
                    else
                    {
                        Color newCol = _colorPalette[_nextColorIndex % _colorPalette.Count];
                        btnColor.BackColor = newCol;
```

```
            _variableColorMemory[v] = newCol;
            _nextColorIndex++;
        }

        txtMin.TextChanged += (s, e) =>
        {
            performSync(v, (t) => t.start.Text = txtMin.Text);
            if (cmbType.SelectedItem?.ToString() == «Static») txtMax.Text = txtMin.Text;
            MarkAsDirty(s, e);
        };
        txtMax.TextChanged += (s, e) =>
        {
            performSync(v, (t) => t.end.Text = txtMax.Text);
            MarkAsDirty(s, e);
        };
        cmbLineStyle.SelectedIndexChanged += (s, e) =>
        {
            performSync(v, (t) => t.cmblinestyle.SelectedIndex = cmbLineStyle.SelectedIndex);
            MarkAsDirty(s, e);
        };
        cmbLineWidth.SelectedIndexChanged += (s, e) =>
        {
            performSync(v, (t) => t.cmblinewidth.SelectedIndex = cmbLineWidth.SelectedIndex);
            MarkAsDirty(s, e);
        };
        cmbMarker.SelectedIndexChanged += (s, e) =>
        {
            performSync(v, (t) => t.cmbmarker.SelectedIndex = cmbMarker.SelectedIndex);
            MarkAsDirty(s, e);
        };
        cmbMarkerSize.SelectedIndexChanged += (s, e) =>
        {
            performSync(v, (t) => t.cmbmarkerwidth.SelectedIndex = cmbMarkerSize.SelectedIndex);
            MarkAsDirty(s, e);
        };
        btnColor.Click += (s, e) =>
        {
            using (var cd = new ColorDialog())
            {
                cd.Color = btnColor.BackColor;
                if (cd.ShowDialog() == DialogResult.OK)
                {
                    btnColor.BackColor = cd.Color;
                    _variableColorMemory[v] = cd.Color;
                    performSync(v, (t) => {
                        t.colorBtn.BackColor = cd.Color;
                    });
                    MarkAsDirty(s, e);
                }
            }
        };
        Action<(TextBox start, TextBox end, ComboBox cmblinestyle, ComboBox cmblinewidth, ComboBox cmbmarker, ComboBox cmbmarkerwidth, ComboBox cmbType, Button
colorBtn)> applyVisualState = (controls) =>
        {
            string selected = controls.cmbType.SelectedItem?.ToString();
            if (selected == «Static»)
            {
                controls.end.Visible = false;
                controls.start.Width = 85;
                controls.end.Text = controls.start.Text;
            }
            else
            {
                controls.end.Visible = true;
                controls.start.Width = 40;
                controls.end.Width = 40;
            }
        };
        cmbType.SelectedIndexChanged += (s, e) =>
        {
```

```
        string selected = cmbType.SelectedItem?.ToString();
        if (selected == «Static»)
        {
            txtMax.Visible = false;
            txtMin.Width = 85;
            table.SetColumnSpan(txtMin, 2);
            txtMax.Text = txtMin.Text;
        }
        else
        {
            txtMax.Visible = true;
            txtMin.Width = 40;
            txtMax.Width = 40;
            table.SetColumnSpan(txtMin, 1);
        }
        performSync(v, (t) =>
        {
            t.cmbType.SelectedIndex = cmbType.SelectedIndex;
            applyVisualState(t);
        });
        MarkAsDirty(s, e);
    };

    table.Controls.Add(lblName, 0, r);
    table.Controls.Add(txtMin, 1, r);
    table.Controls.Add(txtMax, 2, r);
    table.Controls.Add(cmbLineStyle, 3, r);
    table.Controls.Add(cmbLineWidth, 4, r);
    table.Controls.Add(cmbMarker, 5, r);
    table.Controls.Add(cmbMarkerSize, 6, r);
    table.Controls.Add(cmbType, 7, r);
    table.Controls.Add(btnColor, 8, r);
    inputs[v] = (txtMin, txtMax, cmbLineStyle, cmbLineWidth, cmbMarker, cmbMarkerSize, cmbType, btnColor);
    if (cmbType.SelectedIndex == 0)
    {
        txtMax.Visible = false;
        txtMin.Width = 85;
        table.SetColumnSpan(txtMin, 2);
    }
    r++;
}

foreach (var rule in derivativeRules)
{
    string mainVar = rule.Key;
    string value = rule.Value;
    if (inputs.ContainsKey(mainVar))
    {
        var mainControls = inputs[mainVar];

        if (double.TryParse(value, System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out double numVal))
        {
            mainControls.start.Text = numVal.ToString(System.Globalization.CultureInfo.InvariantCulture);
            mainControls.cmbType.SelectedItem = «Static»;
        }

        else if (inputs.ContainsKey(value))
        {
            var secondControls = inputs[value];
            isSyncing = true;
            try
            {
                mainControls.cmbType.SelectedItem = «Static»;
                secondControls.cmbType.SelectedItem = «Static»;
                secondControls.start.Text = mainControls.start.Text;
                secondControls.cmblinestyle.SelectedIndex = mainControls.cmblinestyle.SelectedIndex;
                secondControls.cmblinewidth.SelectedIndex = mainControls.cmblinewidth.SelectedIndex;
                secondControls.cmbmarker.SelectedIndex = mainControls.cmbmarker.SelectedIndex;
                secondControls.cmbmarkerwidth.SelectedIndex = mainControls.cmbmarkerwidth.SelectedIndex;
                secondControls.colorBtn.BackColor = mainControls.colorBtn.BackColor;
```

```
                _variableColorMemory[value] = mainControls.colorBtn.BackColor;

            }
            finally
            {
                isSyncing = false;
            }
        }
    }
    variablesNode.AddControl(table);
    parent.Controls.Add(variablesNode);
}

private void AddHeaderToTable(TableLayoutPanel table, string text, int col, int row)
{
    Label l = new Label
    {
        Text = text,
        AutoSize = true,
        Font = new Font(«Segoe UI», 8f, FontStyle.Regular),
        ForeColor = Color.DimGray,
        Margin = new Padding(0, 0, 0, 5)
    };
    table.Controls.Add(l, col, row);
}
void BuildVariablesBox(IEnumerable<string> vars, Control parent)

{

    var box = new GroupBox

    {

        Text = «Variables»,

        AutoSize = true,

        Padding = new Padding(10),

        Margin = new Padding(10)

    };

    var panel = new FlowLayoutPanel

    {

        AutoSize = true,

        FlowDirection = FlowDirection.TopDown

    };

    foreach (var v in vars.OrderBy(x => x))

    {

        var row = new FlowLayoutPanel { AutoSize = true };

        row.Controls.Add(new Label

        {

            Text = v,

            Width = 30,

            TextAlign = System.Drawing.ContentAlignment.MiddleLeft
```

```
        });

        row.Controls.Add(new TextBox

        {

            Width = 100,

            Tag = v

        });

        panel.Controls.Add(row);

    }

    box.Controls.Add(panel);

    parent.Controls.Add(box);

}
private void AddComputeButton(Control parent)
{
    FlowLayoutPanel btnContainer = new FlowLayoutPanel
    {
        AutoSize = true,
        FlowDirection = FlowDirection.LeftToRight,
        Padding = new Padding(20, 10, 0, 10),
        Width = parent.Width
    };

    Button btn = new Button
    {
        Text = «Розрахувати»,
        AutoSize = true,
        FlatStyle = FlatStyle.Flat,
        BackColor = Color.FromArgb(100, 149, 237),
        ForeColor = Color.White,
        Font = new Font(«Segoe UI», 10f, FontStyle.Bold),
        Cursor = Cursors.Hand,
        Padding = new Padding(10, 5, 10, 5)
    };


    btn.FlatAppearance.BorderSize = 0;


    btn.Click += (s, e) =>
    {

        MessageBox.Show(«Формула не пуста, начинаем расчет!»);

    };

    btnContainer.Controls.Add(btn);
    parent.Controls.Add(btnContainer);
}
private void ScanForBigOpRules(ExprNode node, Dictionary<string, string> staticRules, HashSet<string> vars)
{
    if (node == null) return;
    if (node is BigOpNode big)
    {
        if (big.Upper is ValueNode uVal && IsRealVariable(uVal.Name))
        {
            vars.Add(uVal.Name);
        }
        if (big.Lower is OperationNode op && op.Op == «Equal» && op.Args.Count == 2)
        {

            var left = op.Args[0] as ValueNode;
```

```
            var right = op.Args[1];
            if (left != null && IsRealVariable(left.Name))
            {
                vars.Add(left.Name);
                string valStr = AstToFormulaConverter.Convert(right);

                if (!staticRules.ContainsKey(left.Name))
                {
                    staticRules[left.Name] = valStr;
                }
            }
        }

        ScanForBigOpRules(big.Body, staticRules, vars);
        ScanForBigOpRules(big.Lower, staticRules, vars);
        ScanForBigOpRules(big.Upper, staticRules, vars);
    }
    else if (node is OperationNode op)
    {
        foreach (var arg in op.Args) ScanForBigOpRules(arg, staticRules, vars);
    }
    else if (node is ChainNode chain)
    {
        foreach (var item in chain.Items)
            if (item is ExprNode en) ScanForBigOpRules(en, staticRules, vars);
    }
    else if (node is DerivativeNode dn)
    {
        ScanForBigOpRules(dn.Body, staticRules, vars);
    }
    else if (node is MatrixNode m)
    {
        foreach (var row in m.Rows)
            foreach (var cell in row) ScanForBigOpRules(cell, staticRules, vars);
    }
}

private void ScanForDerivativeRules(ExprNode node, Dictionary<string, string> rules, HashSet<string> vars)
{
    if (node == null) return;

    if (node is DerivativeNode d && !string.IsNullOrEmpty(d.Condition))
    {
        var parts = d.Condition.Split('=');
        if (parts.Length == 2)
        {
            string leftVar = parts[0].Trim();
            string rightVal = parts[1].Trim();

            if (!string.IsNullOrEmpty(rightVal))
            {

                if (!rules.ContainsKey(leftVar))
                {
                    rules[leftVar] = rightVal;
                }

                if (!double.TryParse(rightVal, System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out _))
                {
                    vars.Add(rightVal);
                }
            }
        }
    }

    if (node is OperationNode op)
    {
        foreach (var arg in op.Args) ScanForDerivativeRules(arg, rules, vars);
    }
```

```
    else if (node is ChainNode chain)
    {
        foreach (var item in chain.Items)
            if (item is ExprNode en) ScanForDerivativeRules(en, rules, vars);
    }
    else if (node is BigOpNode big)
    {
        ScanForDerivativeRules(big.Body, rules, vars);
        ScanForDerivativeRules(big.Lower, rules, vars);
        ScanForDerivativeRules(big.Upper, rules, vars);
    }
    else if (node is DerivativeNode dn)
    {
        ScanForDerivativeRules(dn.Body, rules, vars);
    }
    else if (node is MatrixNode m)
    {
        foreach (var row in m.Rows)
            foreach (var cell in row) ScanForDerivativeRules(cell, rules, vars);
    }
}


private MatrixNode FindSquareMatrix(ExprNode node)
{
    if (node == null) return null;

    if (node is MatrixNode m)
    {
        int rows = m.Rows.Count;

        int cols = (rows > 0) ? m.Rows[0].Count : 0;

        if (rows > 0 && rows == cols)
        {
            return m;
        }

        return null;
    }

    if (node is OperationNode op)
    {
        foreach (var arg in op.Args)
        {
            var found = FindSquareMatrix(arg);
            if (found != null) return found;
        }
    }

    if (node is ChainNode chain)
    {
        foreach (var item in chain.Items)
        {
            if (item is ExprNode en)
            {
                var found = FindSquareMatrix(en);
                if (found != null) return found;
            }
        }
    }

    if (node is DerivativeNode d)
    {
        return FindSquareMatrix(d.Body);
    }

    if (node is BigOpNode big)
    {
        return FindSquareMatrix(big.Body);
```

```
    }
    return null;
}
private FlowLayoutPanel CreateRichControlRow(string name, bool includeMinMax)
{

    var panel = new FlowLayoutPanel
    {
        AutoSize = true,
        FlowDirection = FlowDirection.LeftToRight,
        Padding = new Padding(0),
        Margin = new Padding(0),
        WrapContents = false,
        VerticalScroll = { Visible = false }
    };


    var lbl = new Label
    {
        Text = $»{name}»,
        AutoSize = true,
        Anchor = AnchorStyles.Left,
        Font = new Font(«Segoe UI», 9f, FontStyle.Bold),
        Margin = new Padding(0, 8, 5, 0)
    };
    panel.Controls.Add(lbl);


    if (includeMinMax)
    {
        var txtMin = new TextBox { Text = «-10», Width = 40, Tag = «min» };
        var txtMax = new TextBox { Text = «10», Width = 40, Tag = «max» };

        txtMin.BorderStyle = BorderStyle.FixedSingle; txtMin.BackColor = Color.WhiteSmoke;
        txtMax.BorderStyle = BorderStyle.FixedSingle; txtMax.BackColor = Color.WhiteSmoke;

        txtMin.TextChanged += MarkAsDirty;
        txtMax.TextChanged += MarkAsDirty;

        panel.Controls.Add(new Label { Text = «Min:», AutoSize = true, Anchor = AnchorStyles.Left, Margin = new Padding(0, 8, 0, 0) });
        panel.Controls.Add(txtMin);
        panel.Controls.Add(new Label { Text = «Max:», AutoSize = true, Anchor = AnchorStyles.Left, Margin = new Padding(5, 8, 0, 0) });
        panel.Controls.Add(txtMax);
    }

    Label createPropLabel(string text)
    {
        return new Label
        {
            Text = text,
            AutoSize = true,
            Anchor = AnchorStyles.Left,

            Margin = new Padding(10, 8, 0, 0),
            Font = new Font(«Segoe UI», 8f, FontStyle.Regular),
            ForeColor = Color.DarkGray
        };
    }


    Action<ComboBox, string[], int> setupCombo = (cmb, items, selIndex) =>
    {
        cmb.DropDownStyle = ComboBoxStyle.DropDownList;
        cmb.Items.AddRange(items);
        cmb.SelectedIndex = selIndex;
        cmb.Width = 50;
        cmb.Margin = new Padding(3, 3, 0, 0);
        cmb.SelectedIndexChanged += MarkAsDirty;
    };
```

```
        panel.Controls.Add(createPropLabel(«line»));
        var cmbLineStyle = new ComboBox();
        setupCombo(cmbLineStyle, new string[] { ««, «-», «- -», «-*-», «***» }, 1);
        panel.Controls.Add(cmbLineStyle);

        panel.Controls.Add(createPropLabel(«line width»));
        var cmbLineWidth = new ComboBox();
        setupCombo(cmbLineWidth, new string[] { «0.5», «1.0», «1.5», «2.0», «3.0» }, 1);
        panel.Controls.Add(cmbLineWidth);

        panel.Controls.Add(createPropLabel(«marker»));
        var cmbMarker = new ComboBox { Font = new Font(«MS Gothic», 9f) };
        setupCombo(cmbMarker, new string[] { ««, «○», «□», «Δ», «◊» }, 1);
        panel.Controls.Add(cmbMarker);

        panel.Controls.Add(createPropLabel(«marker width»));
        var cmbMarkerSize = new ComboBox();
        setupCombo(cmbMarkerSize, new string[] { «2», «4», «6», «8», «10» }, 2);
        panel.Controls.Add(cmbMarkerSize);

        panel.Controls.Add(createPropLabel(«color»));
        var btnColor = new Button
        {
          Size = new Size(22, 22),
          FlatStyle = FlatStyle.Flat,
          Margin = new Padding(3, 3, 0, 0),
          UseVisualStyleBackColor = false
        };
        if (_variableColorMemory.ContainsKey(name))
        {
          btnColor.BackColor = _variableColorMemory[name];
        }
        else
        {
          Color newCol = (_colorPalette.Count > 0) ? _colorPalette[_nextColorIndex % _colorPalette.Count] : Color.Black;
          if (_colorPalette.Count > 0) _nextColorIndex++;
          _variableColorMemory[name] = newCol;
          btnColor.BackColor = newCol;
        }

        btnColor.Click += (s, e) =>
        {
          using (var cd = new ColorDialog())
          {
            cd.Color = btnColor.BackColor;
            if (cd.ShowDialog() == DialogResult.OK)
            {
              btnColor.BackColor = cd.Color;
              _variableColorMemory[name] = cd.Color;
              MarkAsDirty(s, e);
            }
          }
        };
        panel.Controls.Add(btnColor);

        return panel;
      }

    }

}
```

Лістинг universal_chart.py

```
import sys
import os
import matplotlib.pyplot as plt

def parse_uch(filepath):
    data = {
        'title': 'Chart',
        'xlabel': 'X',
        'ylabel': 'Y',
        'curves': []
    }

    if not os.path.exists(filepath):
        print(f»File not found: {filepath}»)
        return None

    try:
        with open(filepath, 'r', encoding='utf-8', errors='replace') as f:
            lines = f.readlines()
    except Exception as e:
        print(f»Error reading file: {e}»)
        return None

    i = 0
    current_curve = None

    line_styles = {'0': '-', '1': '-', '2': '--', '3': '-.', '4': ':'}
    markers_map = {
        '00': None, '01': 'o', '02': 's', '03': '^', '04': 'D',
        '05': 'v', '06': '<', '07': '>', '08': 'p', '09': '*',
        '10': '+', '11': 'x', '12': '.', '13': ',', '14': 'o',
        '15': 'o', '16': 'o'
    }

    while i < len(lines):
        line = lines[i].strip()
        if not line:
            i += 1
            continue

        if line.startswith('chart '):
            data['title'] = line[6:].strip()
        elif line.startswith('xlabel '):
            data['xlabel'] = line[7:].strip().replace('$', '')
        elif line.startswith('ylabel '):
            data['ylabel'] = line[7:].strip().replace('$', '')

        elif line.startswith('curve '):
            if current_curve:
                data['curves'].append(current_curve)

            current_curve = {
                'name': line[6:].strip(),
                'color': 'black',
                'linestyle': '-',
                'linewidth': 1.5,
                'marker': None,
                'markersize': 0,
                'x': [],
                'y': []
            }
```

```
        elif line.startswith('color ') and current_curve:
            val = line[6:].strip()
            if len(val) == 9 and val.isdigit():
                r = int(val[0:3]) / 255.0
                g = int(val[3:6]) / 255.0
                b = int(val[6:9]) / 255.0
                current_curve['color'] = (r, g, b)
            else:
                current_curve['color'] = 'black'

        elif line.startswith('line ') and current_curve:
            val = line[5:].strip()
            ltype = '1'
            lwidth = 1.5

            if len(val) > 1:
                ltype = val[0]
                try: lwidth = float(val[1:])
                except: pass
            elif len(val) == 1:
                ltype = val

            current_curve['linestyle'] = line_styles.get(ltype, '-')
            current_curve['linewidth'] = lwidth

        elif line.startswith('markers ') and current_curve:
            val = line[8:].strip()
            if len(val) >= 2:
                mtype = val[0:2]
                msize = 6.0
                if len(val) > 2:
                    try: msize = float(val[2:])
                    except: pass

                current_curve['marker'] = markers_map.get(mtype, None)
                current_curve['markersize'] = msize

        elif line.startswith('data ') and current_curve:
            try:
                count_str = line[5:].strip()
                count = int(count_str)

                for _ in range(count):
                    i += 1
                    if i >= len(lines): break

                    row = lines[i].strip()
                    if not row: continue

                    parts = row.split(';')
                    if len(parts) >= 2:
                        try:
                            vx = float(parts[0].replace('D', 'E').replace(',', '.'))
                            vy = float(parts[1].replace('D', 'E').replace(',', '.'))
                            current_curve['x'].append(vx)
                            current_curve['y'].append(vy)
                        except:
                            pass
            except:
                pass

        i += 1

    if current_curve:
```

```
            data['curves'].append(current_curve)

        return data

    def plot_chart(data):
        if not data or not data['curves']:
            print("No data to plot.")
            return

        plt.figure(figsize=(10, 6))
        plt.title(data['title'], fontsize=14)
        plt.xlabel(data['xlabel'], fontsize=12)
        plt.ylabel(data['ylabel'], fontsize=12)

        for c in data['curves']:
            if not c['x']: continue

            mk = c['marker']
            if mk == 'none': mk = None

            plt.plot(c['x'], c['y'],
                     label=c['name'],
                     color=c['color'],
                     linestyle=c['linestyle'],
                     linewidth=c['linewidth'],
                     marker=mk,
                     markersize=c['markersize'])

        plt.grid(True, linestyle='--', alpha=0.7)
        plt.legend()
        plt.tight_layout()
        plt.show()

    if __name__ == "__main__":
        if len(sys.argv) < 2:
            print("Usage: python universal_chart.py <path_to_uch_file>")
            input("Press Enter to exit...")
        else:
            file_path = sys.argv[1]
            chart_data = parse_uch(file_path)
            if chart_data:
                plot_chart(chart_data)
            else:
                print("Failed to parse UCH file.")
                      input("Press Enter...")
```

Лістинг calculator_bridge.py

```
import sympy
import sys
import os
import subprocess
import json
from sympy import conjugate
def write_out_metadata(config, uch_path, var_name, start_val, end_val, step_val):
    try:
        base_name = os.path.basename(uch_path)
        proj_name = os.path.splitext(base_name)[0]
        out_path = os.path.splitext(uch_path)[0] + «.OUT»

        chart_name = config.get('ChartName', 'Chart')
        latex_eq = config.get('Latex', 'Equation')

        t0_str = f»{float(start_val):.1f}»
        tf_str = f»{float(end_val):.1f}»
        dt_str = str(step_val)

        separator = « ..... .....
.....»
        header_line = «
.....»

        def format_row(col1, col2, col3):
            return f»{col1:>17}       {col2:<50} {col3:>25}»

        content = []
        content.append(f»{proj_name}»)
        content.append(«INPUT DATA GENERALISED REPRESENTATION»)
        content.append(header_line)

        input_count = 4
        content.append(f»    COUNT OF THE VARIABLES = {input_count:02d}»)
        content.append(f»            VALUES'S LABEL = {chart_name}»)
        content.append(f»          VALUES'S CAPTION = CALCULATION FOR EQUATION {latex_eq}»)
        content.append(separator)
        content.append(f» VARIABLES' NAMES                                        VARIABLES' MEANNINGS         VARIABLES'
VALUES»)
        content.append(separator)

        content.append(format_row(«t0», «INITIAL VALUE (Start)», t0_str))
        content.append(format_row(«tf», «FINAL VALUE (End)», tf_str))
        content.append(format_row(«dt», «INTEGRATING STEP», dt_str))
        content.append(format_row(var_name.upper(), «INDEPENDENT VARIABLE», base_name))

        content.append(separator)
        content.append(header_line)
        content.append(«OUTPUT DATA GENERALISED REPRESENTATION»)
        content.append(header_line)

        subs = config.get('SubExpressions', [])
        output_count = 1 + len(subs)

        content.append(f»    COUNT OF THE VARIABLES = {output_count:02d}»)
        content.append(f»            VALUES'S LABEL = {chart_name}»)
        content.append(f»          VALUES'S CAPTION = RESULT OF {latex_eq}»)
        content.append(separator)
        content.append(f» VARIABLES' NAMES                                        VARIABLES' MEANINGS         VARIABLES'
VALUES»)
        content.append(separator)

        content.append(format_row(«RESULT», «CALCULATED FUNCTION VALUE», base_name))

        for sub in subs:
            s_name = sub['Name']
            content.append(format_row(s_name, «SUB-EXPRESSION VALUE», base_name))

        content.append(separator)
        content.append(header_line)

        with open(out_path, «w», encoding=«utf-8») as f:
            f.write(«\n».join(content))

        return True
    except Exception as e:
        print(f»Warning: Could not write .OUT file: {e}»)
        return False
def get_fortran_code(f_str, main_var):

    f_str = f_str.strip()
    if f_str.endswith(('+', '-', '*', '/')):
        f_str = f_str[:-1]
```

```
    try:
        expr = sympy.sympify(f_str)

        expr = expr.doit()
        expr = expr.replace(conjugate, lambda x: x)

        user_s = sympy.Symbol(main_var)
        loop_s = sympy.Symbol('loop_var')
        expr_subs = expr.subs(user_s, loop_s)

        return sympy.printing.fcode(expr_subs, source_format='free', standard=95)

    except Exception:

        return «0.0d0»

def generate_and_calculate(formula_str, var_name, start_val, end_val, step_val, full_config_json, output_path):
    try:
        config = json.loads(full_config_json)

        extra_vars = {}
        all_variables = config.get('Variables', [])
        for v in all_variables:
            if v['Name'] != var_name:
                extra_vars[v['Name']] = v['Start']

        base_vars_upper = [v['Name'].upper() for v in all_variables]
        sub_expr_codes = []
        seen_formulas = set()

        for sub in config.get('SubExpressions', []):
            name = sub['Name']

            if name == «RESULT» or name in seen_formulas:
                continue

            code = get_fortran_code(sub['Formula'], var_name)

            seen_formulas.add(name)
            sub_expr_codes.append({
                'Name': name,
                'Code': code,
                'Color': sub.get('Color', '150150150'),
                'Line': sub.get('LineStr', '11.5'),
                'Marker': sub.get('MarkerStr', '006')
            })

        declarations = []
        initializations = []
        for p_name, p_val in extra_vars.items():
            declarations.append(f»  double precision :: {p_name}»)
            val_str = str(p_val).replace(',', '.')
            initializations.append(f»  {p_name} = {val_str}d0»)

        res_color = config.get('ResultColor', '100149237')
        res_line = f»{config.get('ResultLineType', 1)}{str(config.get('ResultLineWidth', 2.0)).replace(',', '.')}»
        res_markers = f»{config.get('ResultMarkerType', 0):02d}{config.get('ResultMarkerSize', 6)}»

        total_curves = len(all_variables) + len(sub_expr_codes) + 1

        vars_block = ««
        for v in all_variables:
            v_name = v['Name']
            color = v.get('Color', '000000000')
            line = v.get('LineStr', '11.0')
            marker = v.get('MarkerStr', '016')
            val_expr = «loop_var» if v_name == var_name else f»{v_name}»

            vars_block += f»««
  write(10, '(A)') «curve {v_name.upper()}»
  write(10, '(A)') «min {str(v['Start']).replace(',', '.')}»
  write(10, '(A)') «max {str(v['End']).replace(',', '.')}»
  write(10, '(A)') «color {color}»
  write(10, '(A)') «line {line}»
  write(10, '(A)') «markers {marker}»
  write(10, '(A, I9.9)') «data «, (f_N + 1)
  do f_idx = 0, f_N
      loop_var = start_v + f_idx * step_v
      write(10, '(ES24.15E3, «;», ES24.15E3)') loop_var, {val_expr}
  end do
«««

        subs_block = ««
        for item in sub_expr_codes:
            subs_block += f»««
  write(10, '(A)') «curve {item['Name']}»
  write(10, '(A)') «color {item['Color']}»
  write(10, '(A)') «line {item['Line']}»
```

```
  write(10, '(A)') «markers {item['Marker']}»
  write(10, '(A, I9.9)') «data «, (f_N + 1)
  do f_idx = 0, f_N
      loop_var = start_v + f_idx * step_v
      result_val = {item['Code']}
      write(10, '(ES24.15E3, «;», ES24.15E3)') loop_var, result_val
  end do
««««

        filename_only = os.path.basename(output_path)
        file_root_name = os.path.splitext(filename_only)[0]
        chart_name = config.get('ChartName', 'My Chart')
        latex_code = config.get('Latex', '')
        x_label = config.get('XLabel', 'Time')
        y_label = config.get('YLabel', 'Value')
        iterations = config.get('Iterations', 1000)
        steps_str = str(config.get('Steps', 0.01)).replace(',', '.')

        fortran_src = f»««
program main
  implicit none
  integer :: f_idx, f_N
  double precision :: loop_var, result_val, start_v, end_v, step_v
{chr(10).join(declarations)}

  f_N = {iterations}
  start_v = {str(start_val).replace(',', '.')}d0
  end_v = {str(end_val).replace(',', '.')}d0
  step_v = {steps_str}d0
{chr(10).join(initializations)}

  open(unit=10, file='{filename_only}', status='replace', action='write')

  write(10, '(A)') «chart {chart_name}»
  write(10, '(A)') «latex {latex_code}»
  write(10, '(A)') «xlabel {x_label}»
  write(10, '(A)') «ylabel {y_label}»
  write(10, '(A)') «xgrid 110;10;1»
  write(10, '(A)') «ygrid 110;10;1»
  write(10, '(A)') «legend 0;1»
  write(10, '(A, I9.9)') «curves «, {total_curves}
  write(10, '(A, I0)') «iterations «, f_N
  write(10, '(A, A)') «steps «, «{steps_str}»

{vars_block}
{subs_block}

  write(10, '(A)') «curve RESULT»
  write(10, '(A)') «color {res_color}»
  write(10, '(A)') «line {res_line}»
  write(10, '(A)') «markers {res_markers}»
  write(10, '(A, I9.9)') «data «, (f_N + 1)
  do f_idx = 0, f_N
      loop_var = start_v + f_idx * step_v
      result_val = {get_fortran_code(formula_str, var_name)}
      write(10, '(ES24.15E3, «;», ES24.15E3)') loop_var, result_val
  end do

  close(10)
end program main
««««
        base_dir = os.path.dirname(os.path.abspath(output_path))
        f90_path = os.path.join(base_dir, f»{file_root_name}.f90»)
        exe_path = os.path.join(base_dir, f»{file_root_name}.exe»)

        with open(f90_path, «w», encoding='utf-8') as f:
            f.write(fortran_src)

        startupinfo = None
        if os.name == 'nt':
            startupinfo = subprocess.STARTUPINFO()
            startupinfo.dwFlags |= subprocess.STARTF_USESHOWWINDOW

        res_compile = subprocess.run([«gfortran», f90_path, «-o», exe_path, «-O3»],
                                     capture_output=True, text=True, startupinfo=startupinfo)
        if res_compile.returncode != 0:
            return f»Error: Compilation failed.\\n{res_compile.stderr}»

        res_run = subprocess.run([exe_path], capture_output=True, text=True, startupinfo=startupinfo, cwd=base_dir)

        if res_run.returncode == 0:
            write_out_metadata(config, output_path, var_name, start_val, end_val, step_val)
        return «OK» if res_run.returncode == 0 else f»Error: Runtime failed.\\n{res_run.stderr}»

    except Exception as e:
                return f»Error: Python exception: {str(e)}»
```

Лістинг WebView.cs

```
using Microsoft.Web.WebView2.Core;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Text.Json;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Linq;

namespace Formula_Window
{
        public partial class MainForm : Form
        {
          private int _clickedTabIndex = -1;
          private bool isLoading = true;
          private Dictionary<string, (TextBox start, TextBox end,ComboBox cmblinestyle,ComboBox cmblinewidth,ComboBox cmbmarker, ComboBox cmbmarkerwidth, Button colorBtn)> inputs = new Dictionary<string, (TextBox, TextBox,ComboBox,ComboBox, ComboBox, ComboBox, Button)>();

          public Dictionary<string, VariableRange> ResultRanges { get; private set; }

          private string currentLatexCode = ««;
          private ContextMenuStrip _tabContextMenu;
          public MainForm()
          {
                InitializeComponent();
                tabControl1.TabPages.Add(«+»);
```

```
            InitTabContextMenu();
        }
        private void InitTabContextMenu()
        {
            _tabContextMenu = new ContextMenuStrip();

            var itemRename = _tabContextMenu.Items.Add(«Rename / Save As...»);
            itemRename.Click += (s, e) => { RenameViaSaveAs(_clickedTabIndex); };

            var itemClose = _tabContextMenu.Items.Add(«Close»);
            itemClose.Click += (s, e) => { CloseTab(_clickedTabIndex); };

            _tabContextMenu.Items.Add(new ToolStripSeparator());

            var itemOpenFolder = _tabContextMenu.Items.Add(«Відкрити папку проєкту»);
            itemOpenFolder.Click += (s, e) => { OpenTabFolder(_clickedTabIndex); };
            var itemShowOUT= _tabContextMenu.Items.Add(«Відкрити .OUT файл»);
            itemShowOUT.Click += (s, e) => { OpenOUT(_clickedTabIndex); };
            var itemShowF90= _tabContextMenu.Items.Add(«Відкрити код .F90»);
            itemShowF90.Click += (s, e) => { OpenF90(_clickedTabIndex); };
            var itemShowGraph= _tabContextMenu.Items.Add(«Показати графік .UCH»);
            itemShowGraph.Click += (s, e) => { ShowUCH(_clickedTabIndex); };
            var itemOpenUCH= _tabContextMenu.Items.Add(«Відкрити .UCH»);
            itemOpenUCH.Click += (s, e) => { OpenUCH(_clickedTabIndex); };
        }
        private bool pageLoaded = false;
        private void EnsurePlusTab()
        {
            bool hasPlus = false;
            foreach (TabPage tab in tabControl1.TabPages)
            {
                if (tab.Text == «+») hasPlus = true;
            }

            if (!hasPlus)
            {
                tabControl1.TabPages.Add(«+»);
```

```
        }
    }
    private void ShowUCH(int index)
    {
        if (index < 0 || index >= tabControl1.TabCount) return;

        if (tabControl1.TabPages[index].Controls.Count > 0 &&
            tabControl1.TabPages[index].Controls[0] is UserControl1 uc)
        {
            string uchPath = uc.ProjectFilePath;

            if (string.IsNullOrEmpty(uchPath))
            {
                MessageBox.Show(«Проєкт ще не збережено. Спочатку збережіть файл.», «Увага», MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
            if (!File.Exists(uchPath))
            {
                MessageBox.Show($»Файл не знайдено:\n{uchPath}», «Помилка», MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            try
            {
                string scriptPath = Path.Combine(Application.StartupPath, «universal_chart.py»);

                if (!File.Exists(scriptPath))
                {
                    MessageBox.Show(«Скрипт universal_chart.py не знайдено в папці програми.», «Помилка»);
                    return;
                }

                ProcessStartInfo psi = new ProcessStartInfo();
                psi.FileName = «python»;
```

```
                psi.Arguments = $»\»{scriptPath}\» \»{uchPath}\»«;
                psi.UseShellExecute = true;
                psi.CreateNoWindow = false;

                Process.Start(psi);
            }
            catch (Exception ex)
            {
                MessageBox.Show($»Помилка запуску Python: {ex.Message}», «Помилка»);
            }
        }
    }
    private bool LoadSession()
    {
        string sessionFile = Path.Combine(Application.StartupPath, «session.txt»);

        if (!File.Exists(sessionFile)) return false;

        try
        {
            string[] lines = File.ReadAllLines(sessionFile);
            if (lines.Length <= 1) return false;

            int activeIndex = 0;
            int.TryParse(lines[0], out activeIndex);

            bool filesLoaded = false;

            for (int i = 1; i < lines.Length; i++)
            {
                string path = lines[i].Trim();
                if (File.Exists(path))
                {
                    CreateNewTab(path);
                    filesLoaded = true;
                }
            }
```

```
                    if (activeIndex >= 0 && activeIndex < tabControl1.TabCount - 1)
                    {
                        tabControl1.SelectedIndex = activeIndex;
                    }

                    return filesLoaded;
                }
                catch
                {
                    return false;
                }
            }
            private bool RestoreSession()
            {
                string sessionFile = Path.Combine(Application.StartupPath, «session.txt»);
                try
                {
                    string[] lines = File.ReadAllLines(sessionFile);
                    if (lines.Length < 2) return false;
                    int activeIndex = 0;
                    int.TryParse(lines[0], out activeIndex);

                    bool anyTabLoaded = false;

                    for (int i = 1; i < lines.Length; i++)
                    {
                        string path = lines[i].Trim();
                        if (File.Exists(path))
                        {
                            CreateNewTab(path);
                            anyTabLoaded = true;
                        }
                    }

                    if (anyTabLoaded && activeIndex >= 0 && activeIndex <
tabControl1.TabCount - 1)
                    {
```

```
                    tabControl1.SelectedIndex = activeIndex;
                }

                return anyTabLoaded;
            }
            catch
            {
                return false;
            }
        }
        private void PerformFirstLaunchSetup()
        {
            string baseDir = Path.Combine(Application.StartupPath, «Projects»);
            if (!Directory.Exists(baseDir)) Directory.CreateDirectory(baseDir);

            string projName = «NewProject1»;
            string projFolder = Path.Combine(baseDir, projName);
            string fullPath = Path.Combine(projFolder, projName + «.UCH»);

            CreateNewTab(fullPath);

            SaveSession();
        }
        private void MainForm_Load(object sender, EventArgs e)
        {

            EnsurePlusTab();

            string sessionFile = Path.Combine(Application.StartupPath, «session.txt»);

            if (!File.Exists(sessionFile) || new FileInfo(sessionFile).Length == 0)
            {
                PerformFirstLaunchSetup();
            }
            else
            {
                bool success = RestoreSession();
```

```
                    if (!success)
                    {
                        PerformFirstLaunchSetup();
                    }
                }
            }
            private void webView21_NavigationCompleted(object sender,
CoreWebView2NavigationCompletedEventArgs e)
            {
                pageLoaded = true;
            }


            private void MainForm_DragDrop(object sender, DragEventArgs e)
            {
                string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);

                foreach (string file in files)
                {
                    MessageBox.Show(«Файл: « + file);
                }
            }

            private void MainForm_DragEnter(object sender, DragEventArgs e)
            {
                if (e.Data.GetDataPresent(DataFormats.FileDrop))
                {
                    e.Effect = DragDropEffects.Copy;
                }
                else
                {
                    e.Effect = DragDropEffects.None;
                }
            }


            protected override bool ProcessCmdKey(ref Message msg, Keys keyData)
            {
```

```
            bool isCtrlTab = keyData == (Keys.Control | Keys.Tab);
            bool isCtrlShiftTab = keyData == (Keys.Control | Keys.Shift | Keys.Tab);

            if (isCtrlTab || isCtrlShiftTab)
            {
                int realTabCount = tabControl1.TabCount - 1;
                if (realTabCount <= 1) return true;

                int currentIndex = tabControl1.SelectedIndex;
                int nextIndex = currentIndex;

                if (isCtrlTab)
                {
                    nextIndex++;
                    if (nextIndex >= realTabCount) nextIndex = 0;
                }
                else
                {
                    nextIndex--;
                    if (nextIndex < 0) nextIndex = realTabCount - 1;
                }

                tabControl1.SelectedIndex = nextIndex;

                tabControl1.Focus();


                return true;
            }

            return base.ProcessCmdKey(ref msg, keyData);
        }
        private void tabControl1_DrawItem(object sender, DrawItemEventArgs e)
        {
            var tabControl = sender as TabControl;
            var tabPage = tabControl.TabPages[e.Index];
            Rectangle tabRect = tabControl.GetTabRect(e.Index);
```

```
                bool isPlusTab = (e.Index == tabControl.TabCount - 1);

                if (isPlusTab)
                {
                    TextRenderer.DrawText(e.Graphics, «+», new Font(e.Font.FontFamily,
12, FontStyle.Bold),
                        tabRect, Color.Black,
                        TextFormatFlags.HorizontalCenter |
TextFormatFlags.VerticalCenter);
                }
                else
                {
                    const int closeBtnWidth = 20;

                    Rectangle closeRect = new Rectangle(
                        tabRect.Right - closeBtnWidth,
                        tabRect.Top,
                        closeBtnWidth,
                        tabRect.Height);

                    Rectangle textRect = new Rectangle(
                        tabRect.X,
                        tabRect.Top,
                        tabRect.Width - closeBtnWidth,
                        tabRect.Height);

                    TextRenderer.DrawText(e.Graphics, tabPage.Text, e.Font, textRect,
                        Color.Black,
                        TextFormatFlags.VerticalCenter | TextFormatFlags.Left |
TextFormatFlags.EndEllipsis);

                    using (var fontX = new Font(e.Font.FontFamily, 9, FontStyle.Bold))
                    {
                        TextRenderer.DrawText(e.Graphics, «X», fontX, closeRect,
                            Color.Red,
                            TextFormatFlags.VerticalCenter |
TextFormatFlags.HorizontalCenter);
                    }
```

```
        }
    }

    private void OpenTabFolder(int index)
    {
        if (index < 0 || index >= tabControl1.TabCount) return;

        TabPage page = tabControl1.TabPages[index];
        if (page.Controls.Count > 0 && page.Controls[0] is UserControl1 uc)
        {
            string path = uc.ProjectFilePath;
            if (!string.IsNullOrEmpty(path) && File.Exists(path))
            {
                Process.Start(«explorer.exe», $»/select, \»{path}\»«);
            }
            else
            {
                MessageBox.Show(«Файл ще не збережно на диску.», «Помилка», MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
    }
    private void OpenOUT(int index)
    {
        if (index < 0 || index >= tabControl1.TabCount) return;

        TabPage page = tabControl1.TabPages[index];

        if (page.Controls.Count > 0 && page.Controls[0] is UserControl1 uc)
        {
            string uchPath = uc.ProjectFilePath;

            if (string.IsNullOrEmpty(uchPath))
            {
                MessageBox.Show(«Спочатку збережіть проєкт.», «Увага», MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
```

```
                string outPath = Path.ChangeExtension(uchPath, «.OUT»);

                if (File.Exists(outPath))
                {
                    try
                    {
                        Process.Start(«notepad.exe», outPath);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($»Не вдалося відкрити блокнот: {ex.Message}»);
                    }
                }
                else
                {
                    MessageBox.Show($»Файл .OUT не знайдено:\n{outPath}», «Помилка», MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }
        private void OpenF90(int index)
        {
            if (index < 0 || index >= tabControl1.TabCount) return;
            if (tabControl1.TabPages[index].Controls[0] is UserControl1 uc)
            {
                if (string.IsNullOrEmpty(uc.ProjectFilePath)) return;

                string f90Path = Path.ChangeExtension(uc.ProjectFilePath, «.f90»);

                if (File.Exists(f90Path))
                {
                    Process.Start(«notepad.exe», f90Path);
                }
                else
                {
                    MessageBox.Show(«Файл коду .f90 не знайдено.»);
```

```
                }
            }
        }
        private void OpenUCH(int index)
        {
            if (index < 0 || index >= tabControl1.TabCount) return;
            if (tabControl1.TabPages[index].Controls[0] is UserControl1 uc)
            {
                string path = uc.ProjectFilePath;
                if (!string.IsNullOrEmpty(path) && File.Exists(path))
                {
                    Process.Start(«notepad.exe», path);
                }
            }
        }

        private void tabControl1_MouseDown(object sender, MouseEventArgs e)
        {
            for (int i = 0; i < tabControl1.TabCount; i++)
            {
                Rectangle tabRect = tabControl1.GetTabRect(i);

                if (tabRect.Contains(e.Location))
                {
                    if (e.Button == MouseButtons.Left)
                    {
                        Rectangle closeButton = new Rectangle(tabRect.Right -
20, tabRect.Top + 4, 15, 15);
                        if (closeButton.Contains(e.Location))
                        {
                            CloseTab(i);
                            return;
                        }

                        if (i == tabControl1.TabCount - 1) return;
```

```
                                        tabControl1.DoDragDrop(tabControl1.TabPages[i],
DragDropEffects.Move);
                                        return;
                                    }

                                    else if (e.Button == MouseButtons.Right)
                                    {

                                        if (i == tabControl1.TabCount - 1) return;

                                        _clickedTabIndex = i;
                                        tabControl1.SelectedIndex = i;


                                        _tabContextMenu.Show(tabControl1, e.Location);
                                    }
                                }
                            }
                        }
                        private string EnsureProjectStructure(string fullUchPath)
                        {
                            string folderPath = Path.GetDirectoryName(fullUchPath);
                            string fileNameNoExt = Path.GetFileNameWithoutExtension(fullUchPath);

                            if (!Directory.Exists(folderPath))
                            {
                                Directory.CreateDirectory(folderPath);
                            }

                            string uchFile = Path.Combine(folderPath, fileNameNoExt + «.UCH»);
                            string outFile = Path.Combine(folderPath, fileNameNoExt + «.OUT»);
                            string f90File = Path.Combine(folderPath, fileNameNoExt + «.f90»);

                            if (!File.Exists(uchFile)) File.WriteAllText(uchFile, «{}»);
                            if (!File.Exists(outFile)) File.WriteAllText(outFile, ««);
                            if (!File.Exists(f90File)) File.WriteAllText(f90File, «! код фортран»);

                            return uchFile;
```

```
}
private void CreateNewTab(string filePath = null)
{
    if (filePath == null)
    {
        string baseDir = Path.Combine(Application.StartupPath, «Projects»);
        if (!Directory.Exists(baseDir)) Directory.CreateDirectory(baseDir);

        int i = 1;
        while (true)
        {
            string potentialName = «NewProject» + i;
            string potentialFolder = Path.Combine(baseDir, potentialName);
            if (!Directory.Exists(potentialFolder))
            {
                string uFile = Path.Combine(potentialFolder,
potentialName + «.UCH»);
                filePath = uFile;
                break;
            }
            i++;
        }
    }

    EnsureProjectStructure(filePath);

    string title = Path.GetFileName(filePath);
    TabPage newTab = new TabPage(title);
    newTab.Tag = filePath;
    newTab.ToolTipText = filePath;

    var uc = new UserControl1();
    uc.Dock = DockStyle.Fill;
    uc.ProjectSaved += (s, newPath) => {
        newTab.Text = Path.GetFileName(newPath);
        newTab.Tag = newPath;
        SaveSession();
    };
```

```
                uc.LoadFromFile(filePath);
                newTab.Controls.Add(uc);
                int insertIdx = tabControl1.TabCount;
                if (tabControl1.TabCount > 0 && tabControl1.TabPages[tabControl1.TabCount
- 1].Text == «+»)
                        insertIdx = tabControl1.TabCount - 1;

                tabControl1.TabPages.Insert(insertIdx, newTab);
                tabControl1.SelectedTab = newTab;
                if (newTab.Controls.Count > 0)
                {
                        newTab.Controls[0].Focus();
                }
        }
        private void tabControl1_Selecting(object sender, TabControlCancelEventArgs e)
        {
                if (e.TabPage != null && e.TabPage.Text == «+»)
                {
                        e.Cancel = true;

                        this.BeginInvoke(new Action(() =>
                        {
                                CreateNewTab(null);
                        }));
                }
        }
        private TabPage GetTabAt(Point point)
        {
                for (int i = 0; i < tabControl1.TabCount; i++)
                {
                        if (tabControl1.GetTabRect(i).Contains(point))
                        {
                                return tabControl1.TabPages[i];
                        }
                }
                return null;
        }
```

```
private void tabControl1_DragOver(object sender, DragEventArgs e)
{
    if (!e.Data.GetDataPresent(typeof(TabPage)))
    {
        e.Effect = DragDropEffects.None;
        return;
    }
    Point clientPoint = tabControl1.PointToClient(new Point(e.X, e.Y));

    TabPage hoverTab = GetTabAt(clientPoint);

    if (hoverTab == null || tabControl1.TabPages.IndexOf(hoverTab) ==
tabControl1.TabCount - 1)
    {
        e.Effect = DragDropEffects.None;
    }
    else
    {
        e.Effect = DragDropEffects.Move;
    }
}

private void tabControl1_DragDrop(object sender, DragEventArgs e)
{
    Point clientPoint = tabControl1.PointToClient(new Point(e.X, e.Y));

    TabPage targetTab = GetTabAt(clientPoint);
    TabPage draggedTab = (TabPage)e.Data.GetData(typeof(TabPage));
    if (targetTab != null && draggedTab != null && targetTab != draggedTab)
    {
        int targetIndex = tabControl1.TabPages.IndexOf(targetTab);

        if (targetIndex >= tabControl1.TabCount - 1) return;

        tabControl1.TabPages.Remove(draggedTab);

        tabControl1.TabPages.Insert(targetIndex, draggedTab);
```

```
                tabControl1.SelectedTab = draggedTab;
        }
}
private void TabControl1_Selecting(object sender, TabControlCancelEventArgs e)
{
        if (e.TabPage != null && e.TabPage.Text == «+»)
        {
                e.Cancel = true;

                CreateNewTab(null);
        }
}
private void SaveSession()
{
        var linesToSave = new List<string>();

        int idx = tabControl1.SelectedIndex;
        if (idx >= tabControl1.TabCount - 1 && tabControl1.TabCount > 1)
                idx = tabControl1.TabCount - 2;

        linesToSave.Add(idx.ToString());

        foreach (TabPage tab in tabControl1.TabPages)
        {
                if (tab.Text == «+») continue;

                if (tab.Tag != null)
                {
                        linesToSave.Add(tab.Tag.ToString());
                }
        }

        string sessionFile = Path.Combine(Application.StartupPath, «session.txt»);
        File.WriteAllLines(sessionFile, linesToSave);
}
private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
{
        bool hasUnsavedChanges = false;
```

```
            string unsavedFilesList = ««;

            foreach (TabPage page in tabControl1.TabPages)
            {
                if (page.Text == «+») continue;

                if (page.Controls.Count > 0 && page.Controls[0] is UserControl1 uc)
                {
                    if (uc.IsDirty)
                    {
                        hasUnsavedChanges = true;
                        unsavedFilesList += $»\n- {page.Text}»;
                    }
                }
            }
            if (hasUnsavedChanges)
            {
                var result = MessageBox.Show(
                    $»У вас є незбережені зміни у файлах:{unsavedFilesList}\n\nВи точно хочете вийти? (Зміни буде втрачено)»,
                    «Незбережені дані»,
                    MessageBoxButtons.YesNo,
                    MessageBoxIcon.Warning,
                    MessageBoxDefaultButton.Button2);
                if (result == DialogResult.No)
                {
                    e.Cancel = true;
                    return;
                }
            }

            SaveSession();
        }

        private void OpenToolStripMenuItem_Click(object sender, EventArgs e)
        {
            using (var ofd = new OpenFileDialog())
            {
```

```
                ofd.Filter = «Файлы графиков (*.UCH)|*.UCH»;

                if (ofd.ShowDialog() == DialogResult.OK)
                {
                    foreach (TabPage tab in tabControl1.TabPages)
                    {
                        if (tab.Tag != null && tab.Tag.ToString() == ofd.FileName)
                        {
                            tabControl1.SelectedTab = tab;
                            return;
                        }
                    }

                  CreateNewTab(ofd.FileName);
                }
            }
        }
        private void NewToolStripMenuItem_Click(object sender, EventArgs e)
        {
            using (var fbd = new FolderBrowserDialog())
            {
                fbd.Description = «Выберите папку для создания проекта»;

                if (fbd.ShowDialog() == DialogResult.OK)
                {
                    string projName = «NewProject_» + DateTime.Now.Ticks;

                    string fullPath = Path.Combine(fbd.SelectedPath, projName, projName + «.UCH»);

                    CreateNewTab(fullPath);
                }
            }
        }


        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
    this.Close();
}

private void SaveAsToolStripMenuItem_Click(object sender, EventArgs e)
{
    RenameViaSaveAs(tabControl1.SelectedIndex);
}
private void RenameViaSaveAs(int index)
{
    if (index < 0 || index >= tabControl1.TabCount) return;
    if (tabControl1.TabPages[index].Text == «+») return;

    TabPage page = tabControl1.TabPages[index];

    if (page.Controls.Count > 0 && page.Controls[0] is UserControl1 uc)
    {
        using (var sfd = new SaveFileDialog())
        {
            sfd.Title = «Зберегти проєкт як...»;
            sfd.Filter = «UCH Project (*.UCH)|*.UCH»;
            sfd.FileName = Path.GetFileName(uc.ProjectFilePath);

            if (sfd.ShowDialog() == DialogResult.OK)
            {
                string newUchPath = sfd.FileName;
                string newDirectory = Path.GetDirectoryName(newUchPath);
                string newFileNameNoExt = Path.GetFileNameWithoutExtension(newUchPath);

                string oldUchPath = uc.ProjectFilePath;

                if (!string.IsNullOrEmpty(oldUchPath) && File.Exists(oldUchPath))
                {
                    string oldDirectory = Path.GetDirectoryName(oldUchPath);
```

```
                string oldFileNameNoExt = Path.GetFileNameWithoutExtension(oldUchPath);

                string[] extensionsToCopy = { «.f90», «.OUT», «.exe» };

                foreach (var ext in extensionsToCopy)
                {
                    string oldFile = Path.Combine(oldDirectory, oldFileNameNoExt + ext);
                    string newFile = Path.Combine(newDirectory, newFileNameNoExt + ext);

                    if (File.Exists(oldFile))
                    {
                        try
                        {
                            File.Copy(oldFile, newFile, true);
                        }
                        catch (Exception ex)
                        {
                            MessageBox.Show($»Не вдалося скопіювати файл {ext}: {ex.Message}»);
                        }
                    }
                }
            }
            else
            {
                string f90Path = Path.Combine(newDirectory, newFileNameNoExt + «.f90»);
                string outPath = Path.Combine(newDirectory, newFileNameNoExt + «.OUT»);

                if (!File.Exists(f90Path))
                File.WriteAllText(f90Path, «! Fortran code here»);
```

```
                                        if (!File.Exists(outPath))
File.WriteAllText(outPath, ««);
                                        }

                                        uc.UpdateProjectPath(newUchPath);

                                        uc.SaveProject();

                                        page.Text = newFileNameNoExt;
                                        page.Tag = newUchPath;
                                        page.ToolTipText = newDirectory;


                                        SaveSession();

                                        MessageBox.Show($»Проєкт успішно збережено як:\n{newFileNameNoExt}», «Успіх», MessageBoxButtons.OK, MessageBoxIcon.Information);
                                    }
                                }
                            }
                        }

                        private void SaveToolStripMenuItem_Click(object sender, EventArgs e)
                        {

                            if (tabControl1.SelectedIndex < 0 || tabControl1.SelectedIndex >= tabControl1.TabCount) return;
                            if (tabControl1.SelectedTab.Text == «+») return;


                            if (tabControl1.SelectedTab.Controls.Count > 0 && tabControl1.SelectedTab.Controls[0] is UserControl1 uc)
                            {

                                if (string.IsNullOrEmpty(uc.ProjectFilePath))
                                {
                                    SaveAsToolStripMenuItem_Click(sender, e);
                                    return;
```

```
                }


                try
                {
                    uc.SaveProject();
                    string cleanTitle = tabControl1.SelectedTab.Text.Replace(«*», ««).Trim();
                    tabControl1.SelectedTab.Text = cleanTitle;

                }
                catch (Exception ex)
                {
                    MessageBox.Show($»Помилка при збереженні файлу:\n{ex.Message}», «Помилка», MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }

        private bool CloseTab(int index)
        {

            if (index < 0 || index >= tabControl1.TabCount) return false;
            if (tabControl1.TabPages[index].Text == «+») return false;

            TabPage page = tabControl1.TabPages[index];


            if (page.Controls.Count > 0 && page.Controls[0] is UserControl1 uc)
            {
                string projectPath = uc.ProjectFilePath;
                string projectDir = Path.GetDirectoryName(projectPath);
                string folderName = new DirectoryInfo(projectDir).Name;


                if (uc.IsDirty)
                {
                    tabControl1.SelectedIndex = index;
```

```
                var result = MessageBox.Show(
                    $»Проєкт ‘{page.Text}’ має незбережені зміни.\nЗберегти їх?»,
                    «Збереження»,
                    MessageBoxButtons.YesNoCancel,
                    MessageBoxIcon.Warning);

                if (result == DialogResult.Cancel) return false;

                if (result == DialogResult.Yes)
                {
                    if (string.IsNullOrEmpty(projectPath) || folderName.StartsWith(«NewProject»))
                        RenameViaSaveAs(index);
                    else
                        uc.SaveProject();

                    if (uc.IsDirty) return false;
                }
            }

            if (folderName.StartsWith(«NewProject»))
            {
                try
                {
                    if (Directory.Exists(projectDir))
                    {
                        Directory.Delete(projectDir, true);
                    }
                }
                catch (Exception ex)
                {
                    Debug.WriteLine($»Не вдалося почистити папку {folderName}: {ex.Message}»);
                }
            }
        }
```

```
            tabControl1.TabPages.RemoveAt(index);
            page.Dispose();

            int realTabsCount = 0;
            foreach (TabPage t in tabControl1.TabPages)
                if (t.Text != «+») realTabsCount++;

            if (realTabsCount == 0)
            {
                this.Close();
                return true;
            }

            return true;
  }
  private void closeMultipleToolStripMenuItem_Click(object sender, EventArgs e)
  {
            for (int i = tabControl1.TabCount - 1; i >= 0; i--)
            {
                if (tabControl1.TabPages[i].Text == «+») continue;

                bool closed = CloseTab(i);
                if (!closed) break;
            }
  }

  private void closeToolStripMenuItem_Click(object sender, EventArgs e)
  {
            CloseTab(_clickedTabIndex);
  }
}

public class VariableRange
{
  public double Start { get; set; }
  public double End { get; set; }
  public double Step { get; set; }
}}
```

ДОДАТОК В

Вміст обчислювального ядра f90

```
program main
  implicit none
  integer :: f_idx, f_N
  double precision :: loop_var, result_val, start_v, end_v, step_v


  f_N = 1000
  start_v = -10.0d0
  end_v = 10.0d0
  step_v = 0.01d0


  open(unit=10, file='NewProject5.UCH', status='replace', action='write')

  write(10, '(A)') «chart My Chart»
  write(10, '(A)') «latex x+x»
  write(10, '(A)') «xlabel Time»
  write(10, '(A)') «ylabel Value»
  write(10, '(A)') «xgrid 110;10;1»
  write(10, '(A)') «ygrid 110;10;1»
  write(10, '(A)') «legend 0;1»
  write(10, '(A, I9.9)') «curves «, 2
  write(10, '(A, I0)') «iterations «, f_N
  write(10, '(A, A)') «steps «, «0.01»


  write(10, '(A)') «curve X»
  write(10, '(A)') «min -10»
  write(10, '(A)') «max 10»
```

```
  write(10, ‘(A)’) «color 172121242»
  write(10, ‘(A)’) «line 11.0»
  write(10, ‘(A)’) «markers 016»
  write(10, ‘(A, I9.9)’) «data «, (f_N + 1)
  do f_idx = 0, f_N
     loop_var = start_v + f_idx * step_v
     write(10, ‘(ES24.15E3, «;», ES24.15E3)’) loop_var, loop_var
  end do



  write(10, ‘(A)’) «curve RESULT»
  write(10, ‘(A)’) «color 100149237»
  write(10, ‘(A)’) «line 12»
  write(10, ‘(A)’) «markers 016»
  write(10, ‘(A, I9.9)’) «data «, (f_N + 1)
  do f_idx = 0, f_N
     loop_var = start_v + f_idx * step_v
     result_val = 2*loop_var
     write(10, ‘(ES24.15E3, «;», ES24.15E3)’) loop_var, result_val
  end do

  close(10)
end program main
```

ДОДАТОК Г

Програмний код мовою python для вирішення рівняння методом Гауса

Програмний код мовою Python для вирішення рівняння методом Гауса

```
import numpy as np
def gaussian_elimination(A, b):
n = len(b)
M = np.hstack((A, b.reshape(-1, 1))).astype(float)
print(«Початкова розширена матриця:»)
print(M)
print()
for i in range(n):
max_row = np.argmax(np.abs(M[i:, i])) + i
M[[i, max_row]] = M[[max_row, i]]
print(f»Доповнена матриця після заміни рядків {i}:»)
print(M)
print()
for j in range(i+1, n):
factor = M[j, i] / M[i, i]
M[j, i:] -= factor * M[i, i:]
print(f»Доповнена матриця після видалення рядка {j} {i}:»)
print(M)
print()
x = np.zeros(n)
for i in range(n-1, -1, -1):
x[i] = (M[i, -1] - np.dot(M[i, i+1:n], x[i+1:])) / M[i, i]
print(f»Вектор рішення після кроку зворотної заміни {n-i}:»)
print(x)
print()
return x
A = np.array([
[2, -1, 0, 0, 0, 0, 0, 0, 0],
[-1, 2, -1, 0, 0, 0, 0, 0, 0],
[0, -1, 2, -1, 0, 0, 0, 0, 0],
[0, 0, -1, 2, -1, 0, 0, 0, 0],
[0, 0, 0, -1, 2, -1, 0, 0, 0],
[0, 0, 0, 0, -1, 2, -1, 0, 0],
[0, 0, 0, 0, 0, -1, 2, -1, 0],
```

```
[0, 0, 0, 0, 0, 0, -1, 2, -1],
[0, 0, 0, 0, 0, 0, 0, -1, 2]
])
77
C_u = np.array([
[1, 0],
[0, 1],
[1, 0],
[0, 1],
[1, 0],
[0, 1],
[1, 0],
[0, 1],
[1, 0]
])
C_v = np.array([
[1, 0, 1, 0, 1, 0, 1, 0, 1],
[0, 1, 0, 1, 0, 1, 0, 1, 0]
])
B = np.array([
[2, -1],
[-1, 2]
])
f_u = np.array([
[1],
[0],
[1],
[0],
[1],
[0],
[1],
[0],
[1]
])
f_v = np.array([
[1],
```

```
[0]
])
M = np.block([
[A, C_u],
[C_v, B]
])
f = np.vstack([f_u, f_v]).flatten()
solution = gaussian_elimination(M, f)
a = solution[:9]
b = solution[9:]
print(«Вектор рішення a:»)
print(a)
print()
78
print(«Вектор рішення b:»)
print(b)
print()
a, b
```

ДОДАТОК Д

Демонстраційний матеріал

№ документа	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК.ХХХХХХ.03 ПЗ	Пояснювальна записка	А4, 85 с.
		Додаткові матеріали	
		Додаток А	
2		Опубліковані результати	А4, 5 с.
3		Додаток Б	.
		Програмна частина проєкту	А4, 70 с.
		Додаток В	
		Вміст обчислювального ядра f90	А4, 3 с.
		Додаток Г	
		Програмний код мовою python	А4, 4 с.
		для вирішення рівняння	
		Додаток Д	
		Демонстраційний матеріал	А4, 10 с.

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК.ХХХХХХ.03 ВД			
Розроб.		Півень Н.П.			Розробка системи розпізнавання інформаційних позначок для автономних навігації мобільного робота на складському комплексі	Літера	Аркуш	Аркушів
Перевір.		Ромашов Ю.В.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Стародубцев М.Г.						
Затв.		Невлюдов І.Ш.						