Documentación del Proyecto: Análisis de Funciones con AngouriMath

Segunda Entrega - Cálculo Inf.

Introducción

En esta aplicación desarrollada en C#, se utiliza la librería **AngouriMath** para realizar el análisis simbólico de funciones matemáticas dentro de una interfaz gráfica construida con **Windows Forms**. La herramienta permite calcular derivadas, puntos críticos, puntos de inflexión, intervalos de crecimiento y decrecimiento, así como también determinar la concavidad de una función. La interfaz de usuario permite ingresar una función en un cuadro de texto, ejecutar los cálculos mediante un botón y visualizar los resultados en etiquetas separadas dentro del formulario principal.

Código fuente y explicación detallada

Inclusión de librerías y definición del formulario principal

En esta primera parte se incluyen las librerías necesarias para realizar operaciones simbólicas y para la creación del entorno visual.

```
using AngouriMath;
using AngouriMath.Extensions;

namespace SegundaEntrega_Calculo
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

Inicialización de variables y estructuras de datos

Listing 1: Declaración de variables principales

```
// Variables para almacenar la funcion y sus derivadas
Entity funcion = Entity.Number.Integer.Zero;
Entity derivada = Entity.Number.Integer.Zero;
Entity derivada_segunda = Entity.Number.Integer.Zero;

// Esta lista se usara para calcular los intervalos de crecimiento y
    decrecimiento
```

```
7  double[] listaPuntosCriticos = new double[0];
8  double[] listaPuntosInflexion = new double[0];
```

Este bloque inicializa las variables principales que se utilizarán en el análisis matemático. Las entidades de tipo 'Entity' pertenecen a la librería **AngouriMath**, la cual permite representar expresiones simbólicas (funciones, derivadas, ecuaciones, etc.). Las listas 'listaPuntosCriticos' y 'listaPuntosInflexion' almacenarán los valores numéricos obtenidos al resolver las ecuaciones derivadas para determinar los intervalos de análisis.

Evento principal del botón Calcular

El evento `btnCalcular_Click` es el núcleo del programa. Aquí se toma la función ingresada por el usuario, se convierte en una entidad simbólica y se realizan los distintos cálculos.

```
1  private void btnCalcular_Click(object sender, EventArgs e)
2  {
3      try
4      {
5          var funcionTxt = txtFuncion.Text.Trim();
6
7          if (string.IsNullOrEmpty(funcionTxt))
8          {
9              MessageBox.Show("Ingrese una funcion valida");
10             return;
11         }
12
13         Entity funcion = MathS.FromString(funcionTxt);
14         var x = MathS.Var("x");
```

Conversión y validación de la función

Listing 2: Ingreso y validación de la función

```
1  // Ingreso de funcion
2  string funcion_texto = txtFuncion.Text.Trim();
3
4  // Verifica que el campo de texto no este vacio
5  if (string.IsNullOrEmpty(funcion_texto))
6  {
7      MessageBox.Show("Ingrese una funcion valida");
8      return;
9  }
10
11 // Convierte el texto de la funcion a un objeto Entity y simplifica
12 funcion = funcion_texto.ToEntity().Simplify();
13
14 // Verifica que la funcion contenga la variable 'x'
15 if (!funcion.Vars.Contains("x"))
16 {
17     MessageBox.Show("La funcion debe contener la variable 'x'");
18     return;
19 }
```

Aquí se toman los datos del *TextBox* de la interfaz gráfica y se transforman en una expresión matemática simbólica mediante 'ToEntity()'. La función es simplificada automáticamente para evitar errores al calcular derivadas y se valida que contenga la variable independiente x.

Cálculo de derivadas y comprobación de validez

Listing 3: Cálculo de derivadas con AngouriMath

```
// Calculo de la primera derivada
derivada = funcion.Differentiate("x").Simplify();

// Si la derivada es cero, la funcion es constante
if (derivada == Entity.Number.Integer.Zero)
{
    MessageBox.Show("La funcion ingresada es constante o no contiene 'x'");
    return;
}

// Calculo de la segunda derivada
derivada_segunda = derivada.Differentiate("x").Simplify();

// Muestra las derivadas en el formulario
Resul_Derivada.Text = $"f'(x) = {derivada}\nf''(x) = {derivada_segunda}";
```

En este bloque se calculan la primera y segunda derivada de la función ingresada. La comprobación evita que el programa continúe si la derivada es nula (función constante). Los resultados se muestran en la interfaz, facilitando la verificación visual.

Cálculo de puntos críticos e inflexión

Listing 4: Cálculo y almacenamiento de puntos críticos e inflexión

```
var puntosCriticos = MathS.SolveEquation(derivada, "x").Simplify();

if (puntosCriticos is AngouriMath.Entity.Set.FiniteSet setCriticos && setCriticos.Count > 0)
{
    listaPuntosCriticos = new double[setCriticos.Count];
    int i = 0;
    foreach (var punto in setCriticos)
    {
        listaPuntosCriticos[i] = (double)punto.EvalNumerical();
        i++;
    }

    // Redondeo, orden y eliminacion de duplicados
    for (int f = 0; f < listaPuntosCriticos.Length; f++)
        listaPuntosCriticos[f] = Math.Round(listaPuntosCriticos[f], 2);

    Array.Sort(listaPuntosCriticos);
    listaPuntosCriticos = listaPuntosCriticos.Distinct().ToArray();
    Resul_PuntoCritico.Text = "x => " + string.Join("; ", listaPuntosCriticos.Select(p => p.ToString("F2")));
```

```
}
else
{
    Resul_PuntoCritico.Text = "No hay puntos criticos";
}
```

Esta parte utiliza el método 'SolveEquation()' para resolver la ecuación $f'(x) = 0$ y obtener los puntos críticos. Luego convierte los resultados simbólicos en valores numéricos ('EvalNumerical()'), los ordena, redondea y elimina duplicados antes de mostrarlos en pantalla.

Análisis de crecimiento y decrecimiento

Listing 5: Determinación de intervalos de crecimiento y decrecimiento

```
if (puntosCriticos != null)
{
    string resultado = "";
    bool primer = true;
    double ultimoPunto = double.NegativeInfinity;

    foreach (var punto in listaPuntosCriticos)
    {
        if (primer)
        {
            double valor = (double)derivada.Substitute("x", punto - 1).
                EvalNumerical();
            resultado += valor > 0
                ? $"En el intervalo (-inf, {punto:F2}) f(x) => Crece\n"
                : $"En el intervalo (-inf, {punto:F2}) f(x) => Decrece\
                    n";
            primer = false;
        }
        else
        {
            double valor = (double)derivada.Substitute("x", (
                ultimoPunto + punto) / 2).EvalNumerical();
            resultado += valor > 0
                ? $"En el intervalo ({ultimoPunto:F2}, {punto:F2}) f(x)
                    => Crece\n"
                : $"En el intervalo ({ultimoPunto:F2}, {punto:F2}) f(x)
                    => Decrece\n";
        }
        ultimoPunto = punto;
    }

    double valorFinal = (double)derivada.Substitute("x", ultimoPunto +
        1).EvalNumerical();
    resultado += valorFinal > 0
        ? $"En el intervalo ({ultimoPunto:F2}, +inf) f(x) => Crece\n"
        : $"En el intervalo ({ultimoPunto:F2}, +inf) f(x) => Decrece\n"
            ;

    Resul_CrecDecre.Text = resultado;
}
else
```

```
35      Resul_CrecDecre.Text = "No se pudo calcular crecimiento/
            decrecimiento";
```

Aquí se evalúa el signo de la derivada en diferentes intervalos definidos por los puntos críticos. Si la derivada es positiva, la función crece en ese intervalo; si es negativa, decrece. Los intervalos son construidos dinámicamente y presentados en formato legible en la interfaz.

Determinación de concavidad

Listing 6: Cálculo de concavidad e intervalos

```
1  if (puntosInflexion != null)
2  {
3      string resultado = "";
4      bool primer = true;
5      double ultimoPunto = double.NegativeInfinity;
6
7      foreach (var punto in listaPuntosInflexion)
8      {
9          if (primer)
10         {
11             double valor = (double)derivada_segunda.Substitute("x",
                   punto - 1).EvalNumerical();
12             resultado += valor > 0
13                 ? $"En el intervalo (-inf, {punto:F2}) f(x) => Concava
                       hacia arriba\n"
14                 : $"En el intervalo (-inf, {punto:F2}) f(x) => Concava
                       hacia abajo\n";
15             primer = false;
16         }
17         else
18         {
19             double valor = (double)derivada_segunda.Substitute("x", (
                   ultimoPunto + punto) / 2).EvalNumerical();
20             resultado += valor > 0
21                 ? $"En el intervalo ({ultimoPunto:F2}, {punto:F2}) f(x)
                       => Concava hacia arriba\n"
22                 : $"En el intervalo ({ultimoPunto:F2}, {punto:F2}) f(x)
                       => Concava hacia abajo\n";
23         }
24         ultimoPunto = punto;
25     }
26
27     double valorFinal = (double)derivada_segunda.Substitute("x",
           ultimoPunto + 1).EvalNumerical();
28     resultado += valorFinal > 0
29         ? $"En el intervalo ({ultimoPunto:F2}, +inf) f(x) => Concava
               hacia arriba\n"
30         : $"En el intervalo ({ultimoPunto:F2}, +inf) f(x) => Concava
               hacia abajo\n";
31
32     ResulConcavidad.Text = resultado;
33 }
34 else
35     ResulConcavidad.Text = "No se pudo calcular la concavidad";
```

En esta última parte se analizan los intervalos de concavidad usando el signo de la segunda derivada. Si $f''(x) > 0$, la función es cóncava hacia arriba; si $f''(x) < 0$, es cóncava hacia abajo. El procedimiento es análogo al cálculo de crecimiento y decrecimiento.

Manejo de errores y formulario adicional

Finalmente, se manejan los posibles errores en la ejecución y se incluye un botón adicional para mostrar información del proyecto.

```
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Error al analizar la funcion: {ex.Message}",
                        "Error", MessageBoxButtons.OK, MessageBoxIcon.
                           Error);
    }
}

private void btnData_Click(object sender, EventArgs e)
{
    Info form = new Info();
    form.ShowDialog();
}
```

Conclusión

El proyecto logra integrar de forma exitosa el análisis simbólico proporcionado por **AngouriMath** con una interfaz gráfica intuitiva. El usuario puede ingresar una función y obtener de manera automática su análisis completo: derivadas, puntos críticos, puntos de inflexión, intervalos de crecimiento y concavidad. Esto permite una visualización práctica del comportamiento de las funciones sin necesidad de cálculos manuales.