**Desafio 2 - Analizador de Asíntotas (Funciones Racionales)**

**Materia:** Calculo Infinitesimal  
**Equipo:** Lucas Torres, Marco Vitali, Diego Santillán (Comisión F)  
**Repositorio GitHub:** *[agregar URL]*  
**Comunicador/a designado/a:** *Lucas Torres*  
**Fecha:** *14/09/2025*

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

***1. Introducción y alcance:***

El proyecto tiene como objetivo **desarrollar una aplicación en C# (WinForms)** que permita analizar **funciones racionales** de la forma:

Texto

El contenido generado por IA puede ser incorrecto.

El sistema debe identificar y mostrar:

* El **dominio** de la función.
* Las **asíntotas verticales**.
* Las **asíntotas horizontales**.
* Las **asíntotas oblicuas**.

Para lograrlo, se utilizará la librería **AngouriMath** (CAS – Computer Algebra System), que permite manejar expresiones simbólicas además de cálculos numéricos.

***2. Contexto, alternativas y decisión:***

***2.1 Análisis previo y checklist***

Antes de arrancar con el proyecto nos tomamos un tiempo para analizar los tres desafíos que nos proponían (1, 2 y 3). La idea fue evaluar cada uno en base a la dificultad, la parte matemática que requería, la viabilidad técnica y también qué tan interesante podía ser para mostrarlo después. Después de charlarlo en grupo, elegimos trabajar con el **Desafío 2 (Asíntotas de funciones racionales)**. Nos pareció el más completo porque mezcla cálculo algebraico con una forma de representación visual clara dentro de una aplicación, y además nos daba la posibilidad de aprovechar un CAS.

A partir de esa decisión, hicimos un relevamiento de lo que pedía puntualmente el desafío y armamos un checklist para organizarnos mejor y no dejar nada afuera:

* Checklist del desafío (Que debe hacer la aplicación):
  + Permitir ingresar una **función racional**.
  + Calcular y mostrar las **asíntotas verticales** (x = a).
  + Calcular y mostrar las **asíntotas horizontales** (y = L, aclarando si vale en +∞, −∞ o ambos).
  + Calcular y mostrar las **asíntotas oblicuas** (y = mx + b, con detalle de pendiente e intersección).
  + Acompañar los resultados con un **detalle explicativo.**
  + Mostrar mensajes claros en caso de **error o función inválida**.
* Checklist de implementación del desafío a la práctica :
* Crear el **repositorio en GitHub** (público, con README inicial).
* Configurar el proyecto en **C# WinForms** (decidimos WinForms para tener interfaz gráfica).
* Instalar la librería **AngouriMath** mediante NuGet, para resolver automáticamente operaciones algebraicas.
* Diseñar la interfaz:
  + - **ListBox** para mostrar asíntotas verticales y horizontales.
    - **ListView** con columnas para pendiente e intersección en las oblicuas.
    - **RichTextBox** para los detalles y explicaciones.
    - **StatusStrip** con ToolStripStatusLabel para los mensajes de estado.
  + Definir la lógica de cálculo usando **AngouriMath** (raíces, límites, simplificaciones).
  + Implementar validaciones para evitar entradas inválidas.
  + Documentar las pruebas realizadas y dejar constancia de las decisiones de diseño (por ejemplo, un solo TextBox para la función completa en lugar de dos separados).

***2.2 Alternativas de diseño:***

En la etapa inicial también discutimos distintas maneras de encarar la implementación. La decisión más importante fue definir **cómo ingresar la función racional** dentro de la aplicación:

* **Opción 1:** usar **un solo TextBox** para ingresar la función completa (ejemplo: (x^2+1)/(x-3)).
* **Opción 2:** usar **dos TextBox separados**, uno para el numerador (*p(x)*) y otro para el denominador (*q(x)*). (imagen 1)

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

(imagen 1)

Después de probar ambas ideas, vimos que trabajar con un solo TextBox resultaba más práctico y claro tanto para el usuario como para la implementación, ya que la librería **AngouriMath** se encarga de interpretar directamente la función completa. (imagen 2)

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.(imagen 2)

Otra alternativa que consideramos fue si hacerlo en **consola** o en **WinForms**:

* La opción consola era más sencilla para el cálculo, pero poco didáctica a la hora de mostrar los resultados.
* WinForms, en cambio, nos permitía organizar la información con pestañas, listas y cuadros de texto, lo cual mejora mucho la visualización.

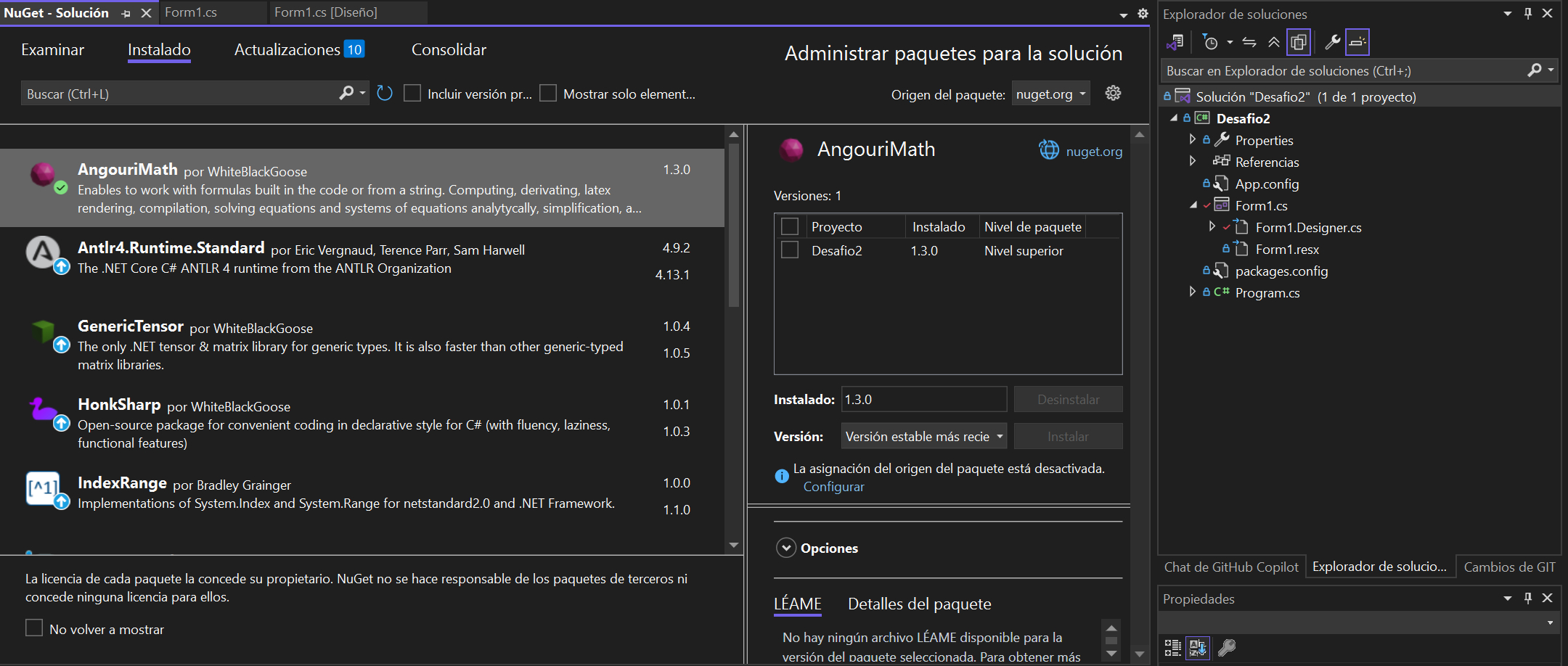
Finalmente, elegimos **WinForms con un solo TextBox para la función completa**, porque era la combinación que nos permitía cumplir mejor con el objetivo didáctico del desafío, al mismo tiempo que mantenía la implementación clara y ordenada. (Imagen 3)

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.(Imagen 3)

***3.1 Configuración inicial***

* Creamos el **repositorio en GitHub** con README.
* Definimos la **solución en C# WinForms**, ya que nos daba la flexibilidad de trabajar con controles gráficos.
* Instalamos la librería **AngouriMath** desde NuGet, fundamental para manipular expresiones algebraicas y calcular límites de forma automática.(imagen 4)

(imagen 4)

***3.2 Diseño de la interfaz***

La interfaz se pensó para que fuera clara y didáctica. Los elementos principales son: (imagen 3)

* Tres pestañas:
  + Verticales → ListBox para mostrar ecuaciones del tipo x = a.
  + Horizontales → ListBox para mostrar límites en +∞ y −∞.
  + Oblicuas → ListView con columnas para pendiente (m) e intersección (b).
* **RichTextBox** donde se detalla el procedimiento (raíces del denominador, si se cancelan o no, etc.).
* **StatusStrip** con un ToolStripStatusLabel que muestra mensajes como “Listo” o “Función inválida”.
* **TextBox de entrada** para que el usuario escriba la función racional completa.

***3.3 Lógica de cálculo***

Con la ayuda de AngouriMath implementamos la lógica para:

* **Detectar asíntotas verticales:** buscando raíces del denominador que no se cancelen con el numerador.
* **Calcular asíntotas horizontales:** evaluando límites cuando x → +∞ y x → −∞.
* **Calcular asíntotas oblicuas:** aplicando el método de división de polinomios o evaluando el cociente de grados.

Además, se programaron **validaciones** para:

* Evitar que el usuario deje el campo vacío.
* Manejar expresiones inválidas o que no correspondan a una función racional.
* Mostrar mensajes de error claros y no permitir que la aplicación se cierre inesperadamente.

***3.4 Pruebas y ajustes***

Se hicieron distintas pruebas (imagen 5) con funciones de diferente complejidad para comprobar que la aplicación respondiera correctamente. También ajustamos la interfaz para que los resultados se muestren ordenados y fáciles de leer (imagen 6 y 7).

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.(imagen 5)

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.(Imagen 6)

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

(imagen 7)

***4. Resultados***

Al finalizar la implementación y las pruebas, obtuvimos los siguientes resultados:

* **Correcta identificación de asíntotas verticales** en distintos casos, distinguiendo cuando una raíz del denominador genera asíntota o cuando se cancela con el numerador.
* **Cálculo de asíntotas horizontales** mostrando los valores de los límites tanto al infinito positivo como al negativo.
* **Detección de asíntotas oblicuas** en funciones donde el grado del numerador es mayor al del denominador, mostrando pendiente e intersección.
* **Dominio de la función:** la aplicación especifica claramente los valores excluidos (raíces del denominador).
* **Explicación detallada:** el RichTextBox entrega un desglose paso a paso, lo que facilita la comprensión del procedimiento.
* **Interfaz amigable:** la organización en pestañas y el uso del StatusStrip mejoraron la experiencia de usuario, mostrando mensajes claros (“Listo”, “Función inválida”).

***5. Qué faltó incorporar y por qué***

Al proyecto le podrían haber sumado algunas funciones extra, como por ejemplo:

* Un **gráfico de la función con sus asíntotas**, que hubiese hecho la app más visual.
* Mejorar la interfaz visual.
* La opción de **exportar los resultados**.

No lo incluimos principalmente por dos motivos: el tiempo disponible y porque no eran requisitos obligatorios para cumplir con el desafío. Decidimos priorizar que lo que pedía el enunciado estuviera bien resuelto, que la app fuera estable y que los cálculos estuvieran correctos.

**6. Conclusión final**

Con este proyecto aprendimos bastante, no solo sobre programación en C#, sino también sobre cómo aplicar la matemática de manera práctica. Al principio nos costó decidir cómo encarar el desafío, pero la elección de hacerlo en WinForms y usar la librería AngouriMath fue clave porque nos simplificó mucho la parte de los cálculos y nos permitió enfocarnos en la lógica y en que la aplicación sea entendible.

Algo que nos sirvió mucho fue armar el checklist antes de empezar. Eso nos ordenó y evitó que se nos pase por alto algún requisito. También nos dimos cuenta de lo importante que es pensar la interfaz desde el lado del usuario: que los resultados estén bien organizados y que el programa avise si algo está mal escrito hace que la experiencia sea mucho mejor.

En líneas generales estamos conformes porque la aplicación cumple lo que pedía el desafío, y además nos permitió reforzar temas de matemática que no usábamos hace rato. Como mejora a futuro estaría bueno agregar gráficos para que sea todavía más visual, y quizás exportar los resultados para tenerlos guardados.

En conclusión, sentimos que el trabajo salió bien, logramos el alcance B y lo más importante es que entendimos mejor cómo conectar la teoría con la práctica.