Álgebra de Baldor: Laboratorio de Polinomios Polinomia

Práctica 4 de Programación Funcional

María Moronta Carrión, 22M111

Índice

1. Objetivo del proyecto

Esta práctica tiene como objetivo el desarrollo de un **laboratorio funcional** para realizar **operaciones algebraicas** sobre **polinomios**, utilizando el lenguaje **Haskell** como núcleo del programa. La temática está inspirada en el conocido libro *Álgebra de Baldor*, y plantea como reto extender el tratamiento básico de polinomios para incluir operaciones más complejas como **derivación**, **factorización**, **integración simbólica**, **evaluación**, **división mediante el método de Ruffini** y la **búsqueda de raíces enteras**. Como parte del desarrollo, se ha optado por complementar el sistema con una herramienta educativa adicional programada en **Python**, que genera **preguntas interactivas** y simula una experiencia lúdica para reforzar el aprendizaje.

2. Instrucciones de uso y estructura del proyecto

Para utilizar el programa es necesario contar con dos entornos principales: Haskell para la parte **lógica del álgebra** y Python para la **interfaz gráfica**. No se requieren librerías externas en Haskell, únicamente el compilador GHC en versión 8.0 o superior. En Python se utiliza la biblioteca estándar, y se recomienda verificar que el módulo `tkinter` está disponible en el sistema. También se hace uso de la librería externa `Pillow` que puede instalarse mediante:

```
pip install pillow
```

El primer paso consiste en compilar el archivo principal de Haskell, `PF_Practica4.hs`, con:

```
ghc -o polinomio PF_Practica4.hs
```

Esto generará un ejecutable llamado `polinomio`. Este ejecutable puede utilizarse de forma independiente desde la terminal, siguiendo los siguientes pasos:

1. Introducir en la primera línea los coeficientes del polinomio, escritos de menor a mayor grado y separados por espacios.
2. Introducir en la segunda línea el nombre de la operación a realizar. Las operaciones disponibles son: `raices`, `dividir`, `derivar`, `integrar`, `factorizar` o `evaluar`.
3. En caso de que la operación sea `dividir` o `evaluar`, introducir en la tercera línea el valor numérico correspondiente (por ejemplo, el punto en el que se evalúa el polinomio o el divisor de Ruffini).

Por ejemplo, para evaluar el polinomio $p(x) = 1+2x+3x^2$ en el punto $x = 2$, se debe escribir lo siguiente:

```
1 2 3
evaluar
2
```

Para derivar el mismo polinomio, basta con omitir la tercera línea:

```
1 2 3
derivar
```

Además del uso desde terminal, el sistema incluye una interfaz gráfica desarrollada en Python. Esta se ejecuta con el comando:

```
python ruleta.py
```

La **interfaz** presenta una ruleta que selecciona aleatoriamente una operación algebraica. A continuación, **se genera una pregunta** con varias opciones, una de las cuales es correcta. Para que la comunicación entre Python y Haskell funcione correctamente, es imprescindible que el ejecutable `polinomio` se encuentre en el mismo directorio que los archivos `.py`. También debe incluirse una imagen en la carpeta `Images/` con el nombre `Baldor.png`, utilizada en la interfaz como decoración.

Este sistema permite utilizar el programa tanto desde consola como desde un entorno visual interactivo, proporcionando flexibilidad para distintos tipos de usuario y contexto educativo.

A continuación se muestra la organización de los archivos que componen el proyecto:

```
Polinomia/
   PF_Practica4.hs        -- Programa principal
   Polinomios.hs          -- Módulo con operaciones sobre polinomios
   polinomio              -- Ejecutable generado por GHC (no incluido)

   ruleta.py              -- Interfaz gráfica con ruleta de operaciones
   preguntas.py           -- Generador de preguntas
   respuesta.py           -- Comunicación con ejecutable Haskell

   Images/
      Baldor.png          -- Imagen decorativa usada en la interfaz

   NombreApellidos.pdf    -- Documento explicativo (esta memoria)
   README.md              -- Manual técnico breve para ejecución
```

Como hemos mencionado con anterioridad, el ejecutable `polinomio` se genera automáticamente tras la compilación del fichero `PF_Practica4.hs` y debe estar presente en el mismo directorio para que la interfaz Python funcione correctamente.

3. Decisiones de implementación

Durante el desarrollo, se tomó la decisión de construir un **TAD** (Tipo Abstracto de Datos) llamado `Polinomio`, con los constructores `PolCero` y `ConsPol`, que permiten representar polinomios de forma dispersa, almacenando únicamente los **coeficientes no nulos** y sus grados correspondientes. Para facilitar operaciones más eficientes, se añadió una segunda representación mediante **listas de coeficientes**, encapsulada en el tipo `PolinomioL`, lo cual permite aplicar técnicas como el método de Ruffini de forma más directa.

Para trabajar con operaciones como la **integración indefinida** y la **división**, se optó por representar los coeficientes de los polinomios con el tipo `Double`. Esta elección permite realizar cálculos con **números reales**, aunque introduce cierta imprecisión inherente al uso de coma flotante. Para solucionar este efecto, se incorporó una **tolerancia numérica** de 10^{-6} al comparar resultados con cero, especialmente en funciones como la detección de raíces o la limpieza de términos nulos.

En cuanto a la presentación, se definió una instancia personalizada de la clase `Show`, para mostrar los polinomios con **notación algebraica** convencional, incluyendo exponentes unicode y símbolos más agradables visualmente.

Para el desarrollo de la interfaz de usuario se optó por utilizar **Python** en lugar de **Haskell**. Esta decisión se tomó por razones prácticas: Python cuenta con bibliotecas maduras y ampliamente documentadas como `tkinter`, que permiten construir **interfaces gráficas** de manera rápida y visualmente atractiva. Además, al separar la lógica algebraica en Haskell del entorno gráfico en Python, se favorece una arquitectura **modular** donde cada lenguaje se utiliza para lo que mejor sabe hacer.

4. Descripción del funcionamiento

El programa principal en **Haskell** se encuentra en el archivo `PF_Practica4.hs`, que ha sido renombrado desde `Main.hs` para cumplir con los requisitos de entrega. Este archivo contiene un `main` interactivo que permite introducir los coeficientes del polinomio, elegir una operación y, en su caso, un valor numérico. A partir de esta entrada, se ejecutan las funciones necesarias para devolver los resultados correspondientes: **derivada**, **integral**, **factorización**, resultado de una **división** o **evaluación** numérica. Las operaciones se han implementado en un módulo auxiliar, `Polinomios.hs`, que contiene tanto la **lógica algebraica** como funciones auxiliares para **visualización** y manipulación simbólica.

Además del núcleo funcional en Haskell, se desarrolló una **interfaz gráfica** en Python como herramienta educativa adicional. Esta interfaz se lanza mediante `ruleta.py`, que muestra una **ruleta animada** creada con `tkinter`, que selecciona de forma aleatoria una operación algebraica entre derivar, integrar, dividir, factorizar, buscar raíces o evaluar. Una vez seleccionada la operación, el sistema genera automáticamente un polinomio factorizable aleatorio, lo procesa mediante una llamada al ejecutable Haskell usando el módulo `subprocess` y obtiene el resultado correspondiente. A partir de esta solución, el programa construye una **pregunta de opción múltiple** con una **respuesta correcta** y varias **opciones erróneas** generadas de forma automática o escogidas al azar entre respuestas predefinidas. La pregunta, el polinomio y las respuestas se presentan visualmente al usuario, quien debe seleccionar la opción que considere correcta. El sistema responde inmediatamente indicando si la respuesta fue acertada.

Además del componente funcional, la interfaz incluye un apartado visual en el que se muestra una imagen del libro *Álgebra de Baldor* junto con un **dato aleatorio** relacionado con su autor, su historia o curiosidades del texto. Esta información se elige automáticamente de un conjunto de frases predefinidas al inicio de la ejecución. El propósito de este bloque es puramente decorativo, pero **sirve de guiño** al enunciado de la práctica.

5. Limitaciones

Actualmente, la **búsqueda de raíces enteras** se encuentra limitada al intervalo $[-10, 10]$, lo que implica que no se detectarán raíces fuera de ese rango, ni raíces racionales o complejas. Esta decisión se tomó para garantizar un **tiempo de respuesta rápido** y evitar el uso de algoritmos más complejos o librerías algebraicas externas. De igual manera, la **factorización** del polinomio está restringida a expresiones que se puedan descomponer mediante raíces reales enteras, sin aplicar técnicas de factorización simbólica más avanzadas.

Por otro lado, aunque el sistema admite polinomios con **coeficientes reales** utilizando el tipo `Double`, esta elección puede conllevar **pequeñas imprecisiones numéricas**, especialmente al realizar operaciones como la división mediante Ruffini o la evaluación numérica. Para mitigar estos efectos, se aplican técnicas de redondeo y comparaciones con tolerancia, pero sigue existiendo un **margen de error** inherente al uso de aritmética en coma flotante.

En lo referente a la operación de integración, se realiza una antiderivada simbólica de los términos del polinomio, pero no se incluye la **constante de integración** habitual, ya que esta no afecta al propósito didáctico del resultado ni al resto de operaciones disponibles.

Respecto a la **interfaz gráfica** desarrollada en Python, actualmente funciona como un **generador de preguntas tipo test** que se presentan de forma visual al usuario. No se lleva un registro del historial de respuestas, ni se permite seleccionar un nivel de dificultad o configurar el tipo de operación manualmente, aunque estas funcionalidades podrían incorporarse en futuras versiones para enriquecer la experiencia del usuario y adaptar el entorno a contextos educativos más exigentes. Concretamente, para la entrega del 5 de junio se implementará un **contador de preguntas seguidas contestadas correctamente**, con el objetivo de **gamificar** más la aplicación.

6. Pruebas realizadas

Se han probado manualmente todas las **operaciones principales** (derivación, integración, factorización, Ruffini y evaluación) con polinomios de grado 2 a 4. En la interfaz gráfica se ha comprobado que las preguntas se generan correctamente, que las **respuestas correctas** se detectan y que las **respuestas erróneas** no coinciden accidentalmente. La comunicación entre **Python** y **Haskell** ha sido probada en distintos sistemas con Python 3.10 y GHC 8.10.

A continuación se muestran algunas capturas de la interfaz gráfica en funcionamiento, donde se puede observar la ruleta de operaciones, la generación de varias preguntas tipo test y la presentación de resultados al usuario:

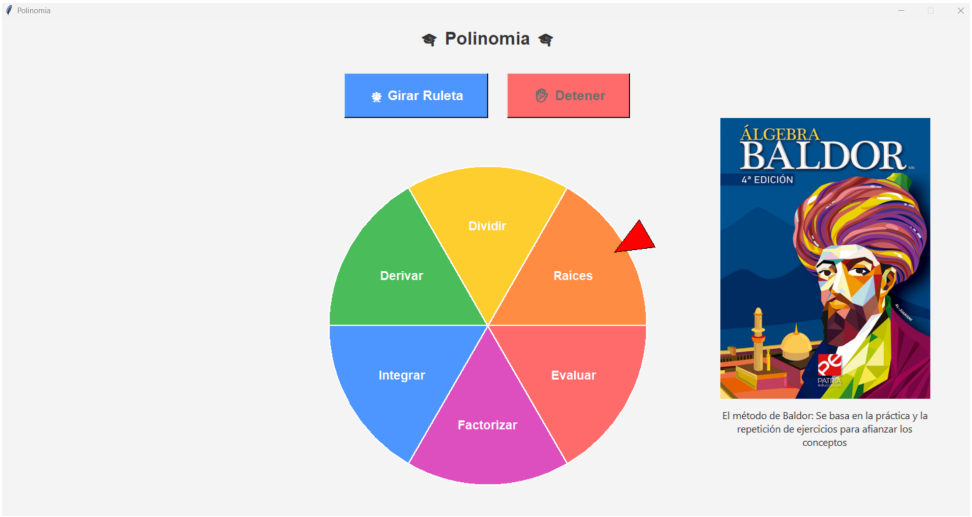

Figura 1: Pantalla de inicio

Figura 2: Ejemplo de pregunta de integración.

Figura 3: Ejemplo de pregunta de división de polinomios

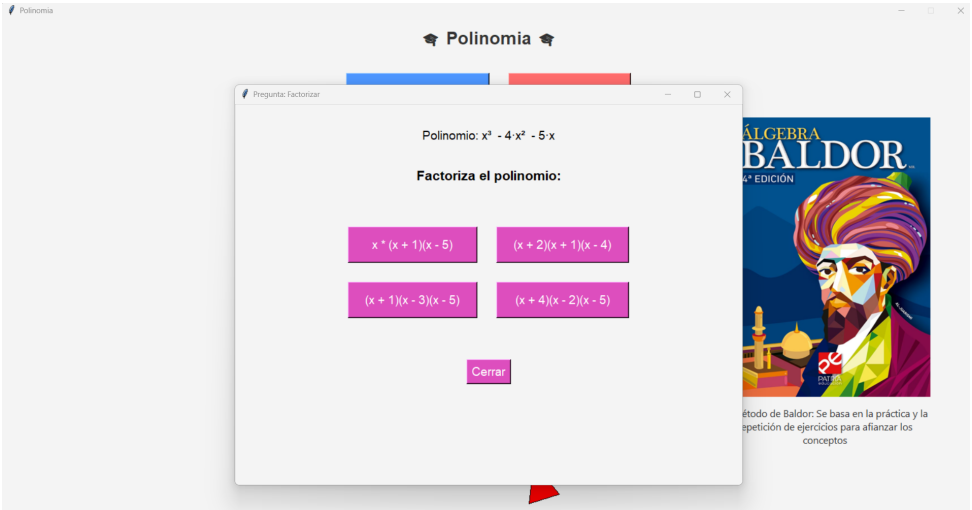

Figura 4: Ejemplo de pregunta de factorización

7. Conclusión

Este proyecto cumple con los **requisitos establecidos** en el enunciado, incluyendo tanto el **contenido mínimo** como parte del **contenido ampliado**. Integra adecuadamente **Haskell** y **Python** para ofrecer una **experiencia educativa** completa y lúdica, combinando **precisión simbólica** con **dinamismo visual**. La implementación propuesta es **fácilmente extensible** y puede servir como base para una **herramienta didáctica** de apoyo al aprendizaje del **álgebra**.

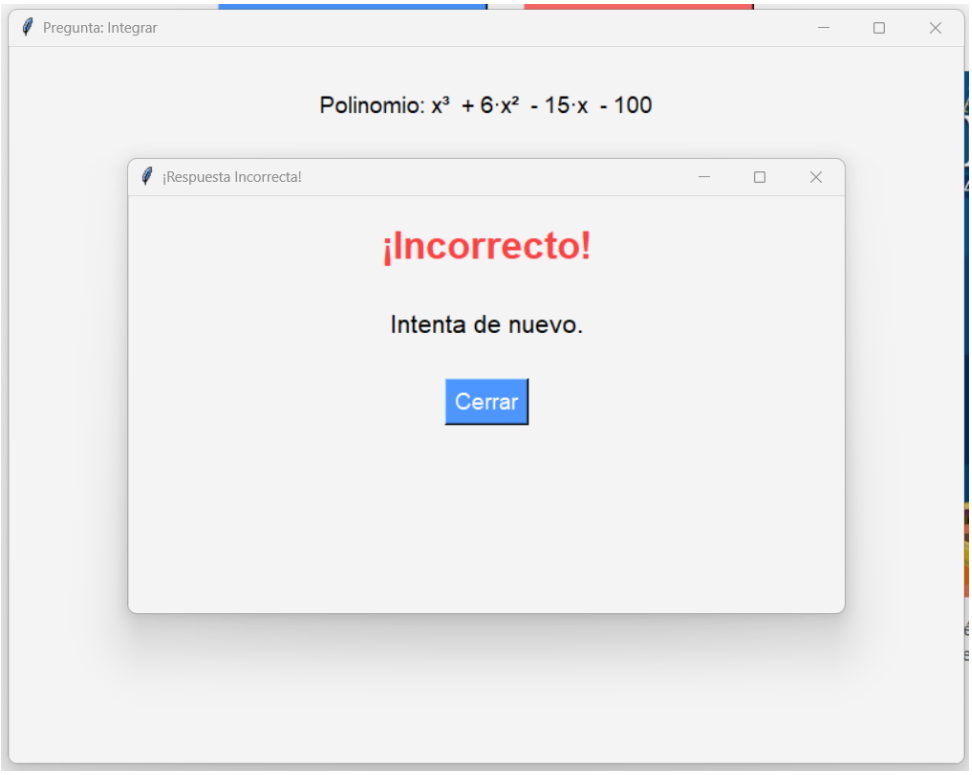

Figura 5: Respuesta incorrecta y retroalimentación visual

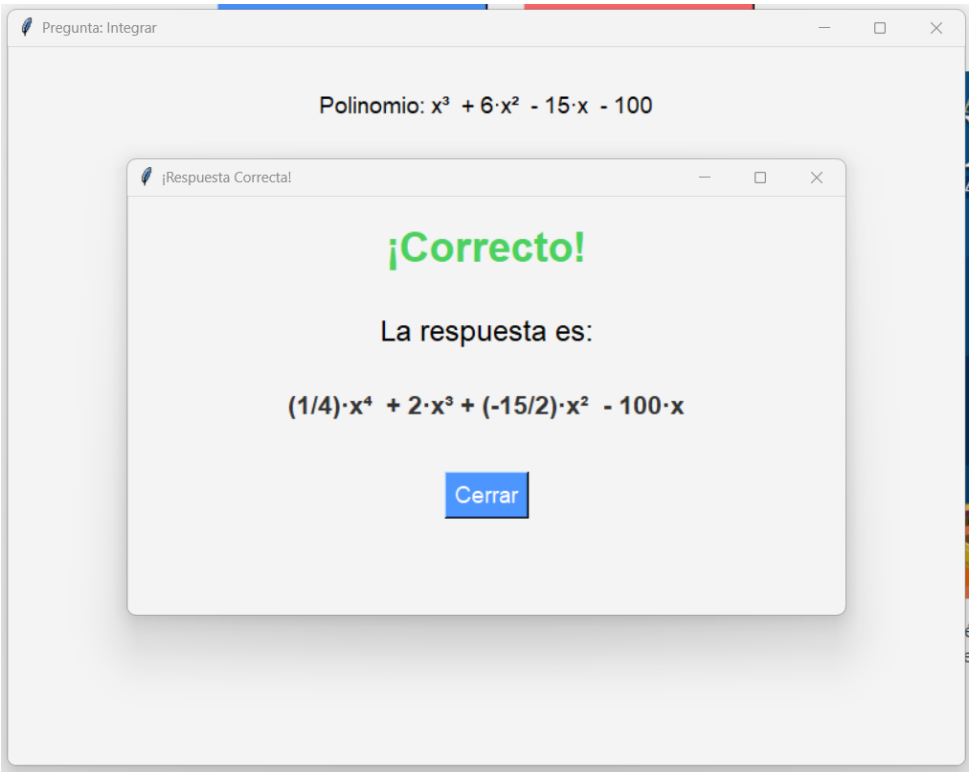

Figura 6: Respuesta acertada por el usuario y retroalimentación visual