Proyecto Matemáticas Discretas

Estudiantes:
Omar Alejandro Salazar Carvajal
Luis Rodrigo Rivera Rivera
Profesor:
Jonatan Gómez Perdomo

21 de julio de 2025

1. Introducción

El siguiente programa se hizo con la intención académica de proporcionar un punto de vista desde una manera recursiva para la derivación de un polinomio en forma factorial, por ejemplo, un polinomio que sea de la forma $(x+a)(x+b)$ se puede ver como el producto de $A \cdot B$ siendo A, B polinomios de la forma $P(x) = \alpha_n x^n + \ \alpha_{n-1}x^{n-1} + \ldots + \alpha_1 x^{\ 1} + \alpha_0$ siendo $\alpha \in \mathbf{R}$, $A = (x+a)$ y $B = (x+b)$ los polinomios correspondientes en el ejemplo dado. Para hallar su derivada lo que haremos es remitirnos a la fórmula correspondiente para la derivada de un producto de $f(x) \cdot g(x)$ (la derivada de un producto de dos funciones es igual a la primera función multiplicada por la derivada de la segunda, más la segunda función multiplicada por la derivada de la primera)[1] siendo nuestras 2 funciones los polinomio que definimos como A y B y aplicando dicha regla que resulta de la forma $A' \cdot B + A \cdot B'$, ahora bien, este caso se puede generalizar si tuvieramos n polinomios que quisieramos tratar, visto de otra forma, podemos intercambiar la derivada del producto de los n polinomios por la derivada del primer polinomio multiplicado por los $n-1$ polinomios restantes más el producto del primer polinomio por la derivada de los $n-1$ polinomios restantes.

2. Desarrollo del problema

Para empezar tenemos que cualquier polinomio es derivable, con esta definición en cuenta podemos partir de la derivada del mismo para descifrar cómo podría crearse una serie de pasos para convertirlo en código.
Entonces supondriamos que:
Sea $P(x)$ un polinomio de grado hasta n

$$\vdash \frac{\frac{d(P(x))}{dx}(s1)}{\frac{d}{dx}(P_1(x)\cdot P_2(x))}(\text{Paso 1})$$

$$\vdash \frac{\frac{d}{dx}(P_1(x)\cdot P_2(x)\cdot \ldots \cdot P_n(x))}{\frac{d}{dx}(P_1(x)\cdot (P_2(x)\cdot \ldots \cdot P_n(x)))}(\text{Paso 3})$$

$$\vdash \frac{\frac{d}{dx}(P_1(x)\cdot (P_2(x)\cdot \ldots \cdot P_n(x)))}{\frac{d}{dx}(P_1(x)\cdot P_\beta(x))}(\text{Paso 4})$$

$$\vdash \frac{\frac{d}{dx}(P_1(x)\cdot P_\beta(x))}{\frac{d(P_1(x))}{dx}\cdot P_\beta(x) + P_1(x)\cdot \frac{d(P_\beta(x))}{dx}}(\text{Paso 5})$$

$$\vdash \frac{\frac{d(P_1(x))}{dx}\cdot P_\beta(x) + P_1(x)\cdot \frac{d(P_\beta(x))}{dx}}{\frac{d(P_1(x))}{dx}\cdot (P_2(x)\cdot \ldots \cdot P_n(x))) + P_1(x)\cdot \frac{d(P_2(x)\cdot \ldots \cdot P_n(x))}{dx}}(\text{Paso 1})$$

Paso 1: Descomposición de un P(x)
Paso 2: Generalización de la descomposición
Paso 3: Asociatividad
Paso 4: Composición de un P(x)
Paso 5: Derivada del producto de dos funciones (considerando P_1 y P_2 como 2 funciones)

Como podemos observar podriamos aplicar este método tantas veces sean necesarias para desarrollar la derivada de dicho polinomio $P(x)$. Dicho de otra forma, podriamos computarizar este problema con un algoritmo recursivo que retorne la derivada del polinomio tras una serie de iteraciones de los pasos ya mencionados.

3. Desarrollo del código

Ya con una imagen más ilustrada del algoritmo que debería seguir el programa, definimos varias funciones que extraigan los componentes que necesitaremos para calcular la derivada del producto de dos polinomios, empezando por calcular la derivada de cada polinomio durante el proceso, para esto también debería ser extraido los componentes de cada polinomio con el fin de operar con ellos, quiere decir que de cada polinomio también extraeremos el coeficiente que lo acompaña (incluido el síbolo) y el exponente máximo de cada polinomio. Para agilizar las operaciones que se realicen con los exponentes y coeficientes se pensó en dejarlo en forma de tuplas.

3.1. Programa y Librerías

Con esto dicho el programa se realizará en **python3** por su versatibilidad y sobretodo por una de sus librerias que nos permitirá extraer los datos de manera más sencilla, además de permitirnos parsear la expresión que ingresemos en código LaTeX a una expresión que pueda entender el lenguaje para así procesar cada operación. La libreria se llama **sympy**, hecha para leer expresiones simbólicas, matemáticas en su mayoría.
De esta librería importaremos los siguientes métodos que extraerán y que crearán variables simbólicas para manipular expresiones matemáticas simbólicas: ***symbols, diff, sympify, latex, Mul, Poly, Add***.
Adicionalmente usaremos **parse_latex** de **sympy.parsing.latex** que funcionará como nuestro parseador dde LaTeX a una expresión simbólica que pueda procesar sympy.
Como segunda librería usaremos **matplotlib**, una librería creada para ver gráficamente dada una función.
Seguimos con la librería **Pillow** o PIL (por sus siglas Python Imaging Library) permite procesar imágenes al interprete de python, esto junto con *matplotlib* hace que una gráfica pueda visualizarce en la GUI del programa, entre otras funciones como poder mostrar pequeños símbolos o imágenes para la familiaridad del usuario con el programa.
Usaremos la librería de **numpy** en conjunto con las anteriores 2 mencionadas (sobretodo matplotlib) con el fin de crear un dominio y evaluarlo, un método que aplica a vectores, arreglos y matrices pero que usaeremos para gestionar la parte gráfica de nuestro programa.
Por último para nuestra interfaz estaremos usando **tkinter** (Toolkit Interface) para manejar justamente una ventana que funcione como interfaz, la cual, pueda interactuar con el código y usuario para que muestre el resultado fuera de la consola.

3.2. Funciones

def. obtener_coeficientes(polinomio)
Su principal objetivo es extraer todos los coeficientes y exponentes del polinomio, devolviéndolos como una lista de tuplas dado un parámetro *polinomio* que puede ser cualquier expresión simbólica, pero en este caso en particular lo trataremos como un polinomio, por ejemplo, $3x^2 + 2x + 1$. Esta función Convierte la expresión en un objeto **Poly**, que facilita el análisis de polinomios, luego usa **.all_coeffs()** para obtener los coeficientes ordenados de mayor a menor grado, luego con **.degree()** determina los exponentes correspondientes y finalmente entrega una tupla de la forma [(coef1, exp1), (coef2, exp2), ...,(coefn,expn)].

def regla_derivada(expresion_parseada)
Aplica la regla del producto y la linealidad de la derivada recursivamente (linealidad que anteriormente explicamos) para derivar una expresión simbólica (para nuestro caso, polinomios), el parámetro *expresion_parseada* es una expresión simbólica ya convertida desde LaTeX mediante parse_latex. Se divide en 3 casos:
Si la expresión es un producto (lo evalua usando **Mul**) aplica la regla del producto recorriendo cada factor, derivando uno a la vez, y multiplicando por los demás sin derivar.

Si la expresión es una suma (lo evalua usando **Add**) deriva cada término y los vuelve a sumar.
Por último si la expresión es un solo polinomio xtrae sus coeficientes y exponentes con obtener_coeficientes y aplica la regla de potencia para derivar ($\frac{d}{dx}(ax^n) = n \cdot ax^{n-1}$) y retorna la derivada en forma de expresión simbólica.

def derivar(expresion_latex)
Función principal que recibe una expresión en formato LaTeX, la convierte a SymPy, aplica la derivación definida por *regla_derivada*, y devuelve el resultado en formato LaTeX. Su parámetro es justamente una cadena en formato LaTeX que representa un polinomio o combinación de polinomios. Usa *parse_latex()* para transformar la cadena LaTeX en una expresión de SymPy y luego llama a *regla_derivada()* para obtener su derivada.

def. calcular_derivada(tkinter)
Una vez es introducida la funcion en formato LaTeX, esta se encarga de obtenerla mediante un get, verificando que este escrita correctamente y poder enviarla a la funcion derivar, una vez se completa esta parte, si se derivó de manera correcta, muestra los resultados obtenidos en derivar, de lo contrario se muestra un error enviado por la funcion derivar.

3.3. GUI

La interfaz gráfica funciona principalmente gracias a la libreria tkinter, permitiendo una mejor interacción entre el usuario y el programa mostrando de manera simple y detallada la función y la derivada de la misma mediante una gráfica, una vez calculada la derivada, y mostrando su equivalente en lenguaje LaTeX.

Funcionamiento de la GUI
Cuando el usuario ingresa una expresión y presiona ' Calcular Derivada ', la expresión en LaTeX es interpretada mediante sympy y convertida en una expresión simbólica. Luego, se aplica una función de derivación personalizada basada en reglas de derivación para sumas y productos. El resultado se muestra en pantalla tanto en formato LaTeX como simbólico. Si el usuario opta por graficar, la expresión y su derivada se convierten en funciones numéricas mediante lambdify, evaluadas sobre un rango definido de valores (usando numpy), y graficadas con matplotlib. Los gráficos se insertan directamente dentro de la ventana tkinter.

def. cargar_ejmplo(tkinter)
Se encarga de mostrar el funcionamiento del programa mediante ejemplos previamente cargados mostrando el correcto funcionamiento y uso del programa al usuario.

def. limpiar_campos(tkinter)
Una vez se oprime ' Limpiar ' la funcion borra los valores introducidos previamente y deja los campos en blanco para poder usarse nuevamente.

def. crear_interfaz(tkinter)
Mediante tkinter se establecen las dimenciones que van a tener la ventana, botones, entradas de texto, que van a ser visibles para el usuario, una vez se tiene estos valores procede a crear la interfaz, con lo antes establecido

def. graficar_derivada(tkinter)
Mediante la libreria matplotlib, se grafican la función y la derivada, primero se comprueba si la derivada ha sido calculada, de no ser así manda un mensaje solicitando que sea calculada la derivada, después procede a convertir la función y la derivada en funciones numéricas que puedan ser aceptadas en numpy, debido a que esta calculará sus valores en X y Y para posteriormente ser graficados por matplotlib evaluándolas en un dominio previamente establecido, luego se envian estos resultados a tkinter para mostrarlos al usuario mediante una gráfica.

3.4. Adicionales

Dentro del código mismo se encuentran también varios auxiliares comentados que permiten determinar el progreso del programa paso a paso, entre ellos se encuentran principalmente algunas salidas por consola que muestran el proceso, ya sea de cada derivada, cofactores o la expresión en LaTeX o simbólica del polinomio ingresado.
También al final hay un ejemplo, con su respectiva ejecución y se utiliza el método **diff()** para comprobar si el resultado es correcto con respecto al programa.

4. Conclusiones

El código progresa según lo esperado y su salida (ya derivada) se expresa en forma de producto de factores, no en la expresión mínima posible, cumpliendo así con el objetivo propuesto por el profesor. Además de mostrar una forma de ver las derivadas de polinomios como un proceso recursivo se muestra una forma diferente de tratar LaTeX en python para tratar problemas tanto matemáticos como recursivos.
Cabe destacar que esta idea de derivada es posible aplicarla más allá de polinomios, por ejemplo, el programa tiene el potencial de alcanzar derivadas de la forma $\frac{x+a}{x+b}$, sin embargo se tienen que considerar varios casos (entre ellos, por ejemplo, el dominio de la función), así que por esto por el momento solo es posible tratar con polinomio que consideramos "sencillos"para el fin académico.
Este documento no incluye el instructivo del programa, por lo que algunas recomendaciones de su uso pueden estar en él, ubicado en el repositorio en el que se encuentre este documento.