Clase 6

Ceros de Funciones de una Variable

6.1 Introducción

Cuando hablamos de los ceros de una función, podemos acercarnos a un tipo de funciones muy conocidas y son los polinomios. Los polinomio tienen la forma

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

Teorema 1:

Todo polinomio con coeficientes reales puede ser factorizado como un producto de factores lineales y cuadráticos irreducibles, donde:

- **Factores Lineales:** Son de la forma $(x - r)$, donde r es una raíz real del polinomio.
- **Factores Cuadráticos Irreducibles:** Son de la forma ax^2+bx+c, donde a, b y c son coeficientes reales, y el discriminante $b^2 - 4ac$ es negativo. Estos factores no tienen raíces reales.

Ejemplo 1:

Calcular el polinomio de grado tres (3) que tiene raíces en $x = 2$, $x = 3$ y $x = 5$

Para encontrar este polinomio simplemente tenemos

$$\begin{aligned} P_3(x) &= (x-2)(x-3)(x-5) \\ &= \left(x^2 - 3x - 2x + 6\right)(x-5) \\ &= \left(x^2 - 5x + 6\right)(x-5) \\ &= x^3 - 5x^2 - 5x^2 + 25x + 6x - 30 \\ &= x^3 - 10x^2 + 31x - 30 \end{aligned}$$

Ejercicio 1:

Crear un *programa* en **Python** que grafique el polinomio $P_3(x) = x^3 - 10x^2 + 31x - 30$ para $x \in [1, 6]$

```python
# Importar las librerias necesarias
import numpy as np
import matplotlib.pyplot as plt

# Definir el polinomio
def poly(x):
    return x**3 - 10*x**2 + 31*x - 30

# Crear el dominio de graficación
x = np.linspace(1, 6, 100)

# Evaluar el polinomio en x
y = poly(x)

# Graficar y configurar la gráfica
plt.plot(x, y)
plt.xlabel('$x$')
plt.ylabel('$P_n(x)$')
plt.title('Gráfico del Polinomio $P_n(x)$')
plt.grid(linestyle='--')
plt.show()
```

Archivo de Programa

El
archivo de programa se encuentra en la ruta: *Unidad-003-Solucion-Ecuaciones-Una-Variable\Programas\Grafica-Polinomio.ipynb*

La gráfica del polinomio se presenta en la siguiente imagen.

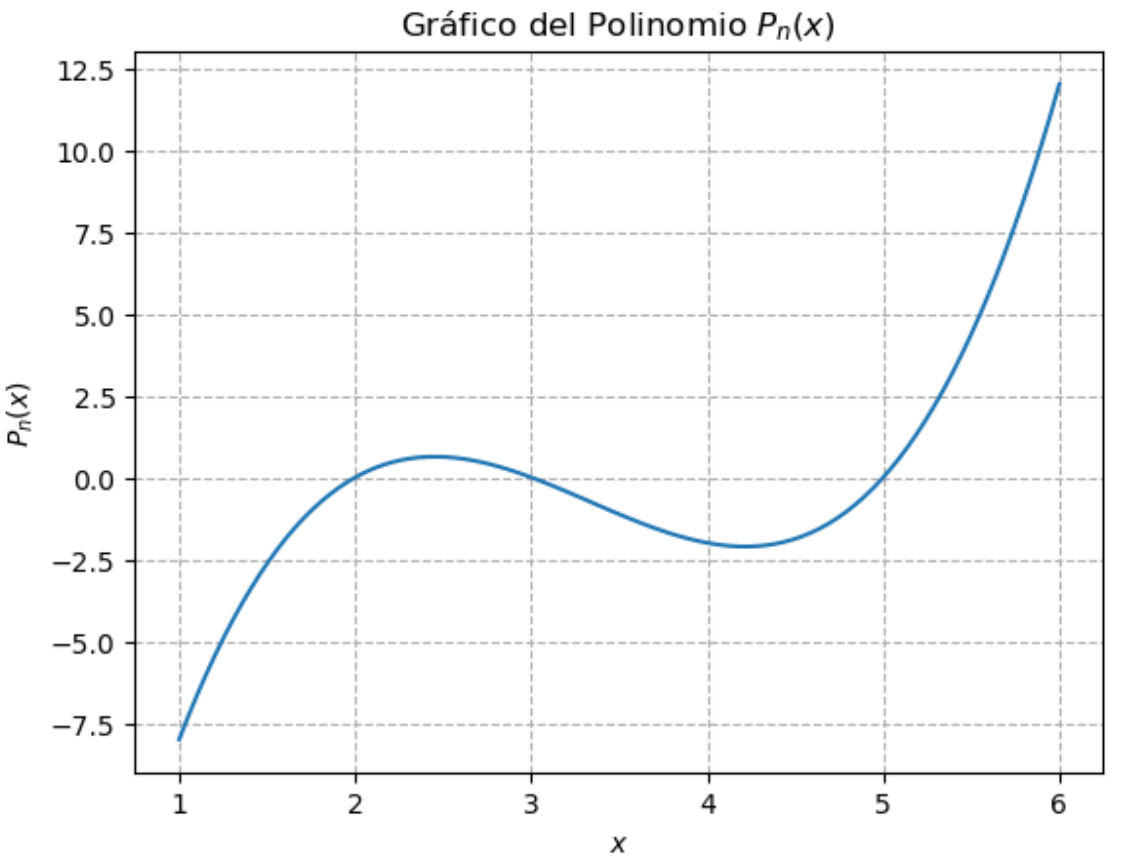

Figura 6.1: Gráfica del Polinomio $P_3(x) = x^3 - 10x^2 + 31x - 30$

En muchos problemas de ingeniería, inteligencia artificial u otras disciplinas afines tenemos que resolver ecuaciones del tipo $f(x) = 0$, donde f es una función que dada una cantidad x nos devuelve $f(x)$.

El valor $\hat{x}$ que cumple que $f(\hat{x}) = 0$ se le llama **solución** de la ecuación, **raíz de la función** o **cero** de la misma. Como vimos en la introducción, en el caso de $P_3(x) = x^3 - 10x^2 + 31x - 30$ podemos verificar que $\hat{x_1} = 2$, $\hat{x_2} = 3$ y $\hat{x_3} = 5$ ya que al evaluar estos valores en la función da como resultado cero (se puede comprobar sobre la gráfica anterior)

En dichos problemas, **saber explícitamente la función f en muchos casos no es posible**, solo tenemos un algoritmo que dado x, nos devuelve $f(x)$.

Entonces, dependiendo de nuestro conocimiento de la función f, podremos aplicar un método numérico u otro.

Todos los métodos numéricos que hallan aproximaciones de ceros construyen una sucesión $(x_n)_n$ que queremos que converja hacia el cero $\hat{x}$ de la función f.

Cuanto mayor sea la velocidad de convergencia de la sucesión $(x_n)_n$, mejor será el método usado.

6.2 Método de la Bisección

Vamos a empezar por el algoritmo menos óptimo de la serie de algoritmos para encontrar los ceros de una función. El método de la bisección no es óptimo pero asegura la convergencia.

El método está basado en el **Teorema de Bolzano** donde recordemos que dice que si la función f es continua y tenemos dos valores a y b $(a < b)$ tal que $f(a) \cdot f(b) < 0$, o sea, hay un cambio de signo entre a y b o en el intervalo (a, b), podemos asegurar que existe un valor $\hat{x}$ tal que $f(\hat{x}) = 0$.

El método, también llamado método de los intervalos encajados, va construyendo una sucesión de intervalos encajados:

$$[a_0, b_0] \supset [a_1, b1] \supset \cdots \supset [a_n, b_n] \supset \cdots,$$

tal que el cero $\hat{x}$ siempre está en todos los intervalos $[a_n, b_n]$ y la longitud de cada intervalo $[a_n, b_n]$ vale $\frac{b_0 - a_0}{2^n}$.De esta manera el cero $\hat{x}$ se calcula de la forma más precisa.

6.2.1 ¿Cómo se construyen los intervalos $[a_n, b_n]$?

El primer intervalo $[a_0, b_0]$ vale

$$[a_0, b_0] = [a, b]$$

A continuación, sea $c = \frac{a_0 + b_0}{2}$ el punto medio del intervalo $[a_0, b_0]$.

Si $f(a_0) \cdot f(c) < 0$, consideremos $b_1 = c$ y $[a_1, b_1] = [a_0, c]$ y si $f(b_0) \cdot f(c) < 0$, consideramos $a_1 = c$ y $[a_1, b_1] = [c, b_0]$.

En general, veamos cómo construir $[a_{n+1}, b_{n+1}]$ en función del intervalo $[a_n, b_n]$. Observemos que se cumplirá siempre que $f(a_n) \cdot f(b_n) < 0$.

Sea $c = \dfrac{a_n + b_n}{2}$ el punto medio del intervalo $[a_n, b_n]$. Si $f(a_n) \cdot f(c) < 0$, consideremos $b_{n+1} = c$ y $[a_{n+1}, b_{n+1}] = [a_n, c]$, y si $f(b_n) \cdot f(c) < 0$, consideremos $a_{n+1} = c$ y $[a_{n+1}, b_{n+1}] = [c, b_n]$

6.2.2 Descripción Gráfica

En la siguiente gráfica se presenta el comportamiento del método de la bisección.

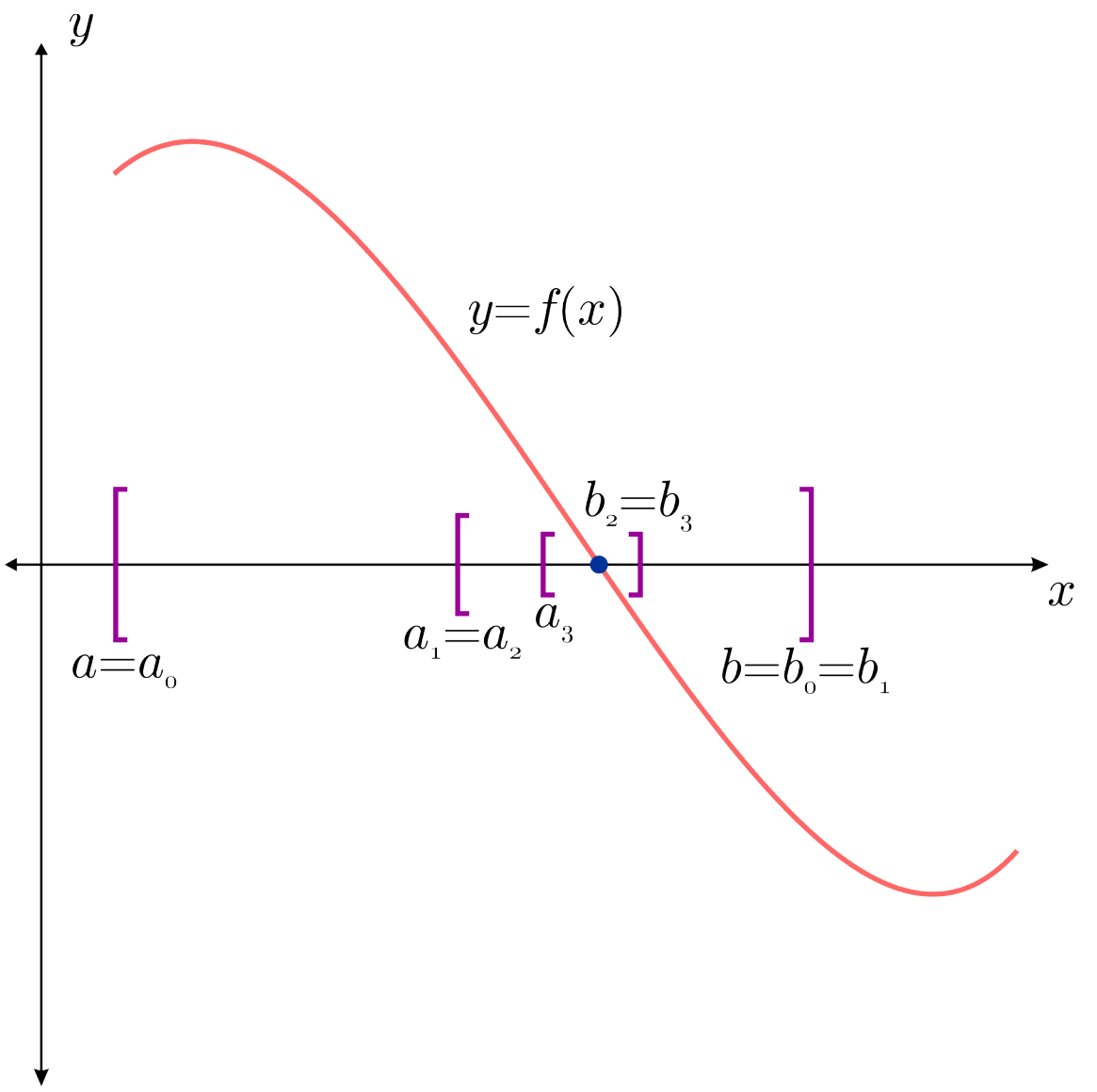

Figura 6.2: Método de la Bisección

6.2.3 Código 1

```
1 # Importar las librerias necesarias
2 import numpy as np
3
4 # Definir la función Biseccion
5 def Biseccion(funcion, a, b, tol, console=False):
6
7     """
```

```
    ## ***Función:*** Biseccion
    - **Descripción:** Calcula la raíz o cero de la función $f$ en el intervalo dado $[a,b]$ con una tolerancia dada.
    - **Parámetros:**
        - *funcion:* Función a la cual se le quiere calcular la raíz o cero
        - *a:* inicio del intervalo de evaluación
        - *b:* fin del intervalo de evaluación
        - *tol:* Máximo error permitido entre la aproximación de la raíz y el valor real.
        - *console:* Valor booleano que permite (si True) mostrar los mensajes de procesamiento de la función
    """

    # Evaluar la función en el punto medio
    c = (a + b) / 2
    feval = funcion(c)

    # Mientras el punto medio del intervalo no cumpla con la condición de cero aproximado
    while abs(feval) >= tol:

        # Determinar el comportamiento de la función
        if funcion(a)*funcion(c) < 0:
            b = c
        else:
            a = c

        # Evaluar la función en el punto medio
        c = (a + b) / 2
        feval = funcion(c)

        if console:
            print("|f(c)| = ", abs(feval))
            print("[a,b] = [",a,",",b,"]")
```

```
38
39      # Damos como valor aproximado del cero el último
          valor de c hallado
40      return c
41
42  # ### Pruebas
43  # Probar la función de bisección para calcular la raíz
      de la función $f(x) = \sin(x) + \cos(x)$ en el
      intervalo para $x\in[1,4]$
44
45  # Probar la función
46  def f(x):
47      return np.sin(x) + np.cos(x)
48
49  x = Biseccion(f, 1, 4, 1e-3, console=True)
50  print("Raíz: ",x)
```

Archivo de Programa

El archivo de programa se encuentra en la ruta: *Unidad-003-Solucion-Ecuaciones-Una-Variable\Programas\Metodo-Biseccion-1.ipynb*

6.2.4 Intervalos encajados

La sucesión $(x_n)_n$ del método de la bisección serán los puntos medios de los intervalos encajados, $x_n = \frac{a_n+b_n}{2}$.

Como $x_n, \hat{x} \in [a_n, b_n]$ tendremos que $|x_n - \hat{x}| \leq b_n - a_n = \frac{b-a}{2^n}$.

Por tanto el orden de convergencia de la sucesión $(x_n)_n$ será la de la sucesión $\frac{1}{2^n}$:

$$|x - \hat{x}| \leq K \cdot \frac{1}{2^n}$$

con $K = b - a$

6.2.5 Código 2

```python
# Definir la función Biseccion
def Biseccion(funcion, a, b, tol, console=False):

    """
    ## ***Función:*** Biseccion
    - **Descripción:** Calcula la raíz o cero de la función $f$ en el intervalo dado $[a,b]$ con una tolerancia dada.
    - **Parámetros:**
        - *funcion:* Función a la cual se le quiere calcular la raíz o cero
        - *a:* inicio del intervalo de evaluación
        - *b:* fin del intervalo de evaluación
        - *tol:* Máximo error permitido entre la aproximación de la raíz y el valor real.
        - *console:* Valor booleano que permite (si True) mostrar los mensajes de procesamiento de la función
    """

    # Calculamos el valor x0 (inicial)
    x = (a + b) / 2

    # Definimos el siguiente valor de la sucesión. Agregando un 1 nos aseguramos que entre en el ciclo while
    y = x + 1

    # Crear el ciclo condicional para calcular los intervalos. Mientras el punto medio del intervalo
    # no cumpla con la condición de cero aproximadoo
    while abs(x - y) >= tol:

        # Calculamos el punto medio
        x = (a + b ) / 2

        # Determinar el comportamiento de la función
```

```
29          if funcion(a)*funcion(x) < 0:
30              # Se define el nuevo b como x
31              b = x
32          else:
33              # Se define el nuevo a como x
34              a = x
35
36          # Calcular el nuevo valor de y
37          y = (a + b) / 2
38
39          if console:
40              print("x = ", x, "; y = ", y)
41
42      # Damos como valor aproximado del cero el último
          valor de x hallado
43      return x
```

6.2.6 Criterios de parada

¿Cuál de los dos métodos de parada es el mejor?

Los dos métodos son equivalentes pero en el caso en que evaluar la función f sea muy costoso es mejor usar el segundo método de parada ya que nos evita una evaluación de la función f.

En cambio, si evaluar la función f no es costoso, podemos usar cualquiera de los dos métodos indistintamente.

Ejemplo 2:

Aproximar el cero de la función $f(x) = x^3 - x + 1$ entre $x = -2$ y $x = -1$, o sea, aproximar el valor $\hat{x} \in (-2, -1)$ tal que $f(\hat{x}) = \hat{x}^3 - \hat{x} + 1 = 0$.

Comprobemos que hay un cambio de signo entre $x = -2$ y $x = -1$:

$$\begin{aligned} f(-2) &= (-2)^3 - (-2) + 1 \\ &= -8 + 2 + 1 = -5 \\ f(-1) &= (-1)^3 - (-1) + 1 \\ &= -1 + 2 = 1 \end{aligned}$$

Aplicando el método de bisección, obtenemos los valores siguientes:

n	a_n	b_n	$x_n = \frac{a_n+b_n}{2}$	$\lvert f(x_n)\rvert$	$\lvert x_n - x_{n-1}\rvert$
0	-2	-1	-1.5	0.875	1
1	-1.5	-1	-1.5	0.875	0.25
2	-1.5	-1.25	-1.25	0.296875	0.125
3	-1.375	-1.25	-1.375	0.224609375	0.0625
4	-1.375	-1.3125	-1.3125	0.051513671875	0.03125
5	-1.34375	-1.3125	-1.34375	0.082611083984375	0.015625
6	-1.328125	-1.3125	-1.328125	0.0145759582519531	0.0078125
7	-1.328125	-1.3203125	-1.3203125	0.0187106132507324	0.00390625
8	-1.328125	-1.32421875	-1.32421875	0.00212794542312622	0.001953125
9	-1.326171875	-1.32421875	-1.326171875	0.00620882958173752	0.0009765625
10	-1.3251953125	-1.32421875	-1.3251953125	0.00203665066510439	0.00048828125
11	-1.3251953125	-1.32470703125	-1.32470703125	$4.65948833152652E-5$	0.000244140625
12	-1.324951171875	-1.32470703125	-1.324951171875	0.000994790971162729	0.0001220703125
13	-1.3248291015625	-1.32470703125	-1.3248291015625	0.000474038819447742	6.103515625E-5

Observaciones

- Vemos que el método, tal como se comento anteriormente, es lento. Por ejemplo, para pasar de un error menor que 0.1 a un error menor que 0.01, usando el método de parada $|f(x_n)| < \epsilon$, necesitamos cuatro iteraciones (de la iteración $n = 3$ a la iteración $n = 7$), o sea, necesitamos cuatro iteraciones para "ganar" una cifra significativa en la aproximación del cero.
- Los dos criterios de parada (dos últimas columnas) son equivalentes en el sentido que necesitan aproximadamente el mismo número de iteraciones para que se "gane" una cifra significativa en el cero. Además los valores que se obtienen con los dos criterios son del mismo orden, es decir, que si en la iteración n, $|f(x_{10})| \approx 4.7 \times 10^{-5}$ **y** $|x_{10} - x_9| \approx 4.9 \times 10^{-5}$