Raíces de ecuaciones no lineales I

Raíz de una ecuación

Una ecuación escalar es una expresión de la forma

$$f(x) = 0$$

donde $f : \Omega \subset \mathbb{R} \to \mathbb{R}$ es una función.

Cualquier número α tal que $f(\alpha) = 0$ se dice que es una solución de la ecuación o una raíz de f.

Ejemplo 1. Calcular de forma aproximada las raíces de la ecuación:

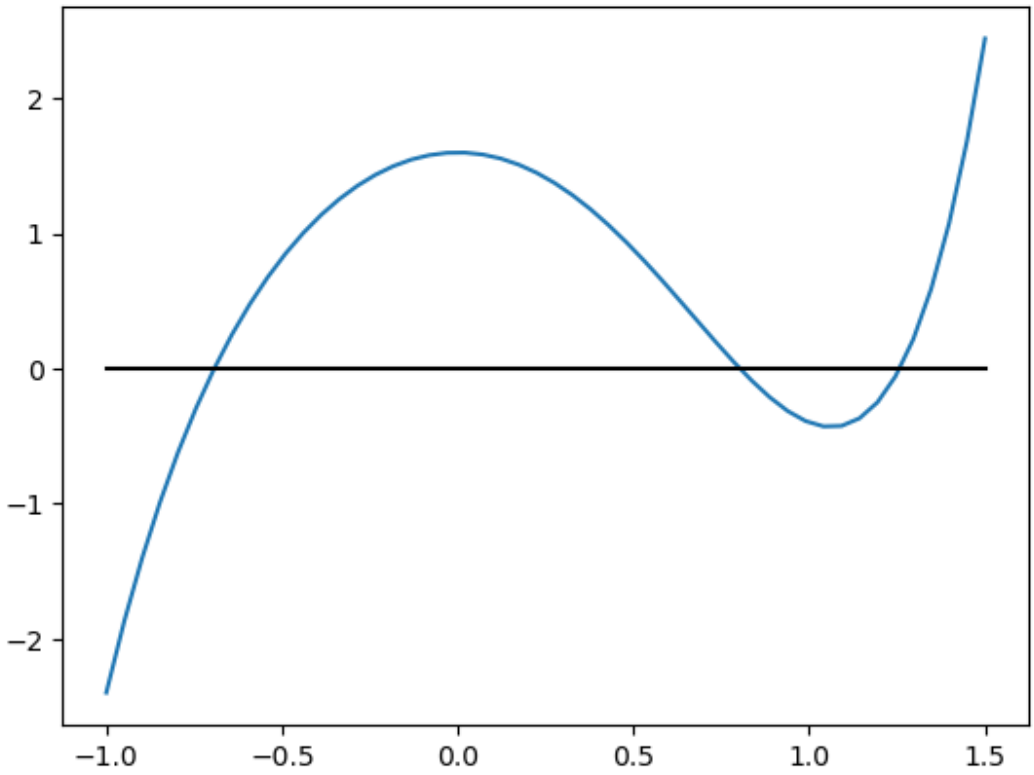
$$x^5 - 3x^2 + 1.6 = 0$$

Importamos los módulos `matplotlib.pyplot` y `numpy` .

```
import numpy as np
import matplotlib.pyplot as plt
```

```
f = lambda x : x**5 - 3 * x**2 + 1.6   # definimos la función
x = np.linspace(-1,1.5)                # definimos un vector con 50 elementos en
(-1,1.5)
ox = 0*x                               # definimos un vector de ceros del tamaño d
e x
```

```
plt.figure()
plt.plot(x,f(x))                    # dibujamos la función
plt.plot(x,ox,'k-')                 # dibujamos el eje X
plt.show()                          # hemos acabado este gráfico
```



Vemos que tiene tres raíces que son, aproximadamente, $x \simeq -0.7$, $x \simeq 0.8$ y $x \simeq 1.3$.

Separación de raíces: búsqueda incremental

A partir de la gráfica anterior podemos separar las raíces en intervalos que cumplan las condiciones del **Teorema de Bolzano**:

Sea $f : [a, b] \to \mathbb{R}$ una función contínua con $f(a)f(b) < 0$. Entonces existe al menos una raíz de f en $[a, b]$.

Es decir, buscamos intervalos donde f tome *signos distintos en los extremos*.

Y si, además, la función es *monótona creciente o decreciente* está garantizado que en el intervalo $[a, b]$ existe una única raíz.

En nuestro ejemplo, tres intervalos que cumplen estas condiciones son:

$$[-1, -0.1] \quad [0.5, 1] \quad [1.2, 1.5]$$

Como vemos, para localizar las raíces, lo mejor suele ser dibujar la función.

Otro método para localizar y separar raíces es el método de búsqueda incremetal. También se puede usar para calcular raíces, pero no suele ser un método eficiente, comparado con otros métodos que vamos a ver.

El método se basa en el teorema de Bolzano: buscamos, para una función continua, un intervalo donde la función tome signos opuestos en los extremos. Y si el intervalo es pequeño, es probable que contenga una única raíz. Por lo tanto, dividimos un intervalo $[a,b]$ en intervalos de pequeña longitud, dx, por medio de puntos $a=x_0, x_1, x_2, \ldots, x_{n-1}, x_n=b$, de forma que $[x_{k-1}, x_k]$ tiene longitud $dx = x_k - x_{k-1}$. Y empezando por la izquierda, vamos comprobando si el signo cambia en cada uno de estos intervalos.

El método tiene varios inconvenientes:

- Podemos pasar por alto dos raíces que disten entre si menos que dx.
- No detecta raíces dobles, cuádruples y, en general, de orden par.
- Puede confundir discontinuidades con raíces.

Ejercicio 1

Escribir una función **`busquedaIncremental(f,a,b,n)`** que tenga como argumentos de entrada una función lambda **`f`**, los extremos de un intervalo **`a`** y **`b`**, y el número de intervalos en que dividiremos $[a,b]$, **`n`**, para separar las raíces de f en intervalos de longitud $dx = \dfrac{b-a}{n}$. El argumento de salida será una matriz que contenga en cada fila los extremos de un intervalo de longitud **`dx`** donde se cumplen las condiciones del teorema de Bolzano.

- Usarla para separar las tres raíces de

  $$f_1(x) = x^5 - 3x^2 + 1.6$$

  buscando en el intervalo $[-1, 1.5]$ con 25 subintervalos.

- Usarla también para separar las raíces de la función

  $$f_2(x) = (x+2)\cos(2x)$$

  contenidas en el intervalo $[0, 10]$ con 100 subintervalos.

Nota:

- Crear una matriz **`intervalos = np.zeros((n,2))`** para almacenar los intervalos según los vamos encontrando.
- Ir contando los intervalos según los vamos guardando con una variable **`contador`**.
- Cuando hemos recorrido en intervalo **`[a,b]`** completo recortar la matriz **`intervalos`** de forma que la matriz de salida contenga solo los intervalos encontrados con **`intervalos = intervalos[:contador,:]`**, que será la variable de salida.

```
%run Ejercicio1
```

```
Intervalos que contienen raíces de f1

[[-0.7 -0.6]
 [ 0.8  0.9]
 [ 1.2  1.3]]

Intervalos que contienen raíces de f2

[[0.7 0.8]
 [2.3 2.4]
 [3.9 4. ]
 [5.4 5.5]
 [7.  7.1]
 [8.6 8.7]]
```

Método de bisección

El método de bisección parte de una función f **contínua** en un intervalo $[a,b]$ donde se cumplen las condiciones del teorema de Bolzano, es decir, la función tiene **distinto signo en los extremos**. Para que funcione no es necesario que las raíces estén separadas, pero sí es conveniente (si están las tres raíces en $[a,b]$ sólo aproximará una).

El método de bisección genera una **sucesión de intervalos** que:

- Cumplen las condiciones del teorema de Bolzano.
- Y por lo tanto contiene siempre al menos una raíz.
- Cada intervalo tiene una longitud que es la mitad del anterior.

En estas condiciones:

- La aproximación de la raíz es uno de los extremos del intervalo y por lo tanto
- El error máximo cometido es la longitud del intervalo.

Algoritmo

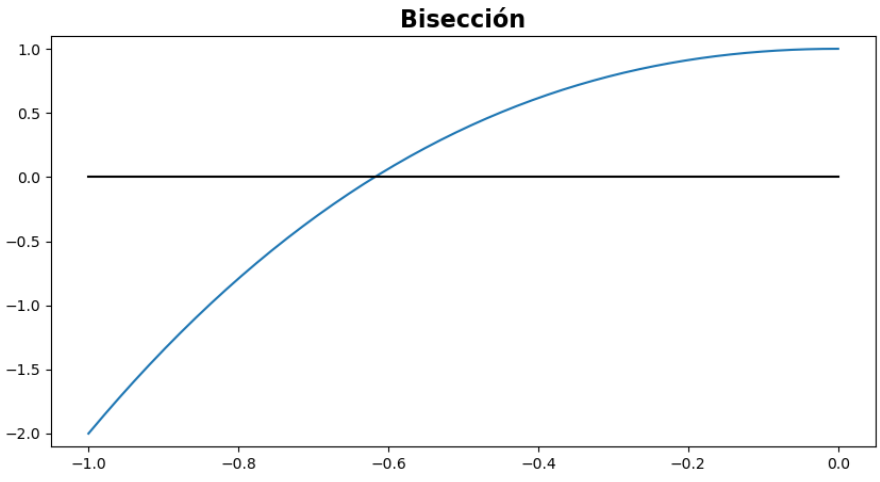
- Sea $a_1 = a$, $b_1 = b$.
- Para $k = 1, 2, \ldots, \mathrm{MaxIter}$
  - Calcular el punto medio $x_k = \dfrac{a_k + b_k}{2}$.
  - Si x_k satisface el criterio de parada, parar.
  - En el caso contrario,
    - si $f(a_k)f(x_k) < 0$ entonces:

      $a_{k+1} = a_k$, $b_{k+1} = x_k$,
    - si $f(x_k)f(b_k) < 0$ entonces:

      $a_{k+1} = x_k$, $b_{k+1} = b_k$.
    - en otro caso:

      acabar.

```
bisec
```



Once Loop Reflect

Criterios de parada

Como la sucesión x_k tal que $x_k \to \alpha$, con $f(\alpha) = 0$ es, en general, infinita, vamos a introducir un criterio para decidir cuando paramos.

Los criterios de parada se basan en:

- El error absoluto $|x_k - x_{k-1}| < tol$.
- El error relativo $\dfrac{|x_k - x_{k-1}|}{|x_k|} < tol$.
- El residual $|f(x_k)| < tol$.

Ejercicio 2

Escribir una función **`biseccion(f,a,b,tol=1.e-6,maxiter=100)`** que tenga como argumentos de entrada la función **`f`** , los extremos del intervalo **`a`** , **`b`** , la cota del error absoluto **`tol`** y el número máximo de iteraciones **`maxiter`** y como argumentos de salida la solución aproximada y el número de iteraciones realizadas.

(a) Usarla para aproximar las tres raíces de $f(x) = x^5 - 3x^2 + 1.6$, con $\mathrm{tol} = 10^{-6}$ y $\mathrm{MaxIter} = 100$. Escribir las tres raíces y el número de iteraciones realizadas para aproximarlas. Utilizar los intervalos obtenidos en el **Ejercicio 1**.

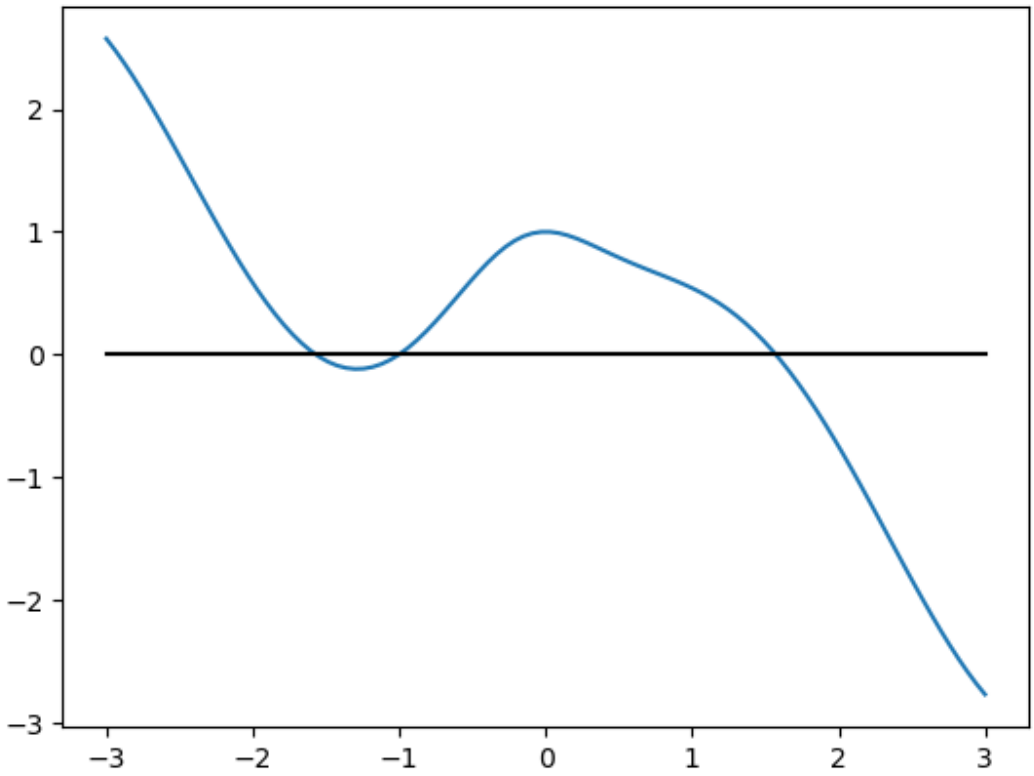
(b) Aproximar las tres raíces de la función

$$f(x) = \frac{x^3 + 1}{x^2 + 1}\cos(x)$$

en el intervalo $[-3, 3]$ con $\mathrm{tol} = 10^{-6}$ y $\mathrm{MaxIter} = 100$. Para ello, dibujar la función en el intervalo $[-3, 3]$ y estimar sobre la gráfica un intervalo $[a, b]$ que contenga la raíz y cumpla las condiciones del teorema de Bolzano. Imprimir las raíces calculadas con solo cinco cifras decimales usando **`print('%.5f' % sol)`**

```
%run Ejercicio2
```

```
-0.6928901672363279 17
0.8027961730957034 17
1.257399749755859 17
```




```
-1.57080
-1.00000
1.57080
```

El método de Newton-Raphson

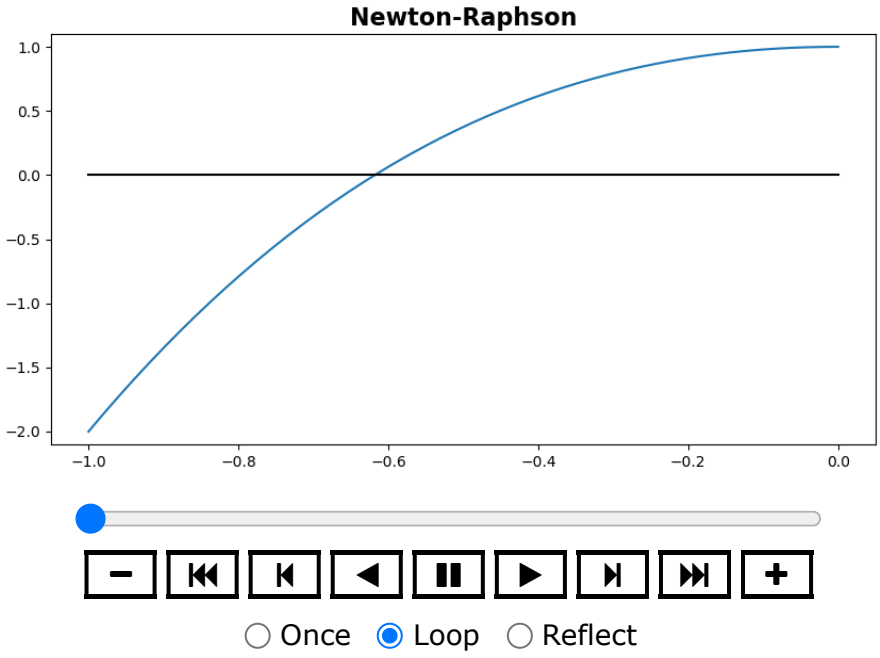
Es el método más usado cuando podemos calcular la derivada exacta. En cada paso, se sustituye la función por su recta tangente en el punto x_k y se calcula su raíz, que es una aproximación de la raíz de la función.

Algoritmo

- Sea x_0 un punto inicial.
- Para $k = 1, 2, \ldots, \text{MaxIter}$:
  - Calcular $x_k = x_{k-1} - \dfrac{f(x_{k-1})}{f'(x_{k-1})}$
  - Si x_k satisface el criterio de parada, parar.
  - En el caso contrario, hacer otra iteración.

En el dibujo vemos la sucesión de rectas tangentes (en rojo) para calcular la primera raíz tomando $x_0 = -1$.

```
newton_raphson
```



Ejercicio 3

Escribir un programa **`newton(f,df,x0,tol=1.e-6,maxiter=100)`** , que tenga como argumentos de entrada la función **`f`** , su derivada **`df`** , el punto inicial **`x0`** , la cota de error absoluto **`tol`** y el número máximo de iteraciones **`maxiter`** y como argumentos de salida la solución aproximada y el número de iteraciones realizadas.

(a) Utilizarlo para aproximar las tres raíces de la función $f(x) = x^5 - 3x^2 + 1.6$, con $tol = 10^{-6}$ y $\mathrm{MaxIter} = 100$. Utilizar como valor inicial el borde izquierdo de los intervalos obtenidos en el **Ejercicio 1**.

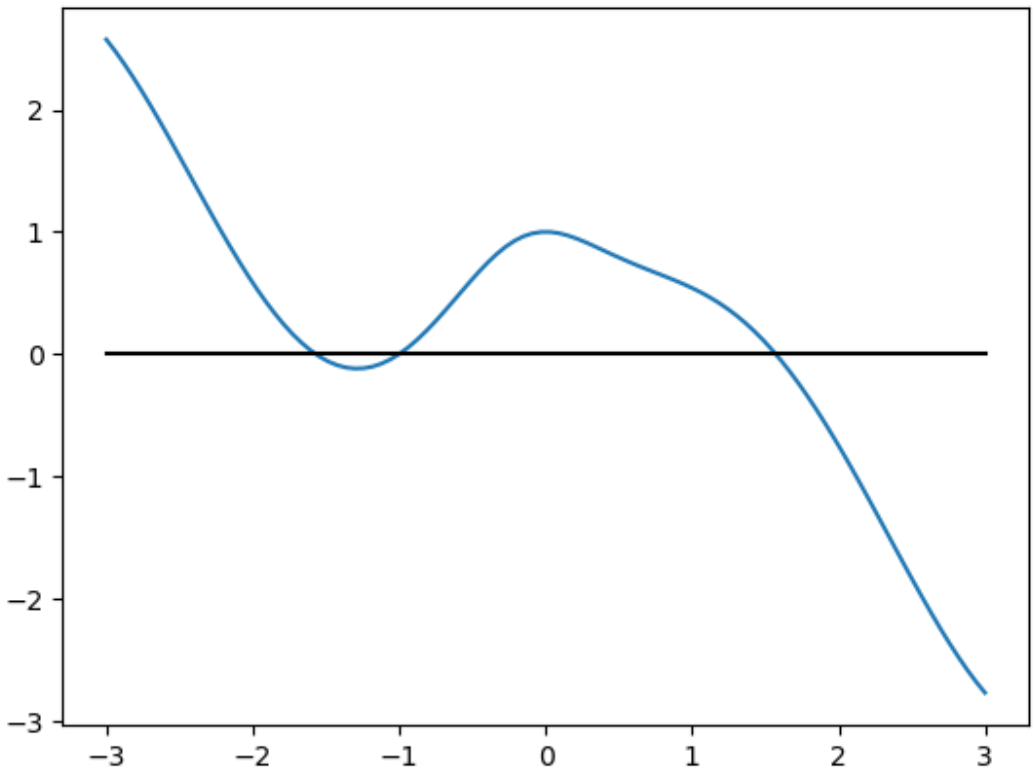
(b) Aproximar las tres raíces de la función

$$f(x) = \frac{x^3 + 1}{x^2 + 1}\cos(x)$$

en el intervalo $[-3, 3]$ con $\mathrm{tol} = 10^{-6}$ y $\mathrm{MaxIter} = 100$. Para ello, dibujar la función en el intervalo $[-3, 3]$ y estimar sobre la gráfica un punto inicial próximo a cada raíz. Imprimir las raíces calculadas con solo cinco cifras decimales usando **`print('%.5f' % sol)`**

```
%run Ejercicio3
```

```
-0.692890801771933 3
0.8027967742655664 3
1.2573997082536856 5
```




```
-1.57080
-1.00000
1.57080
```

Ejercicios propuestos

Ejercicio 4

Escribir una función **`raices_bisec(f,a,b)`** que llame a las funciones de los **ejercicios 1** y **2** y que a partir de un intervalo $[a,b]$ y una función continua f, calcule todas las raíces reales de la función contenidas en el intervalo $[a,b]$.

La cota del error absoluto será **`tol = 1.e-6`** , el número máximo de iteraciones **`maxiter = 100`** , y el número de subintervalos **`n = 100`** .

Utilizar dicho programa para calcular todas las raíces reales de la función

$$f_1(x) = \cos^2(x) + x/10$$

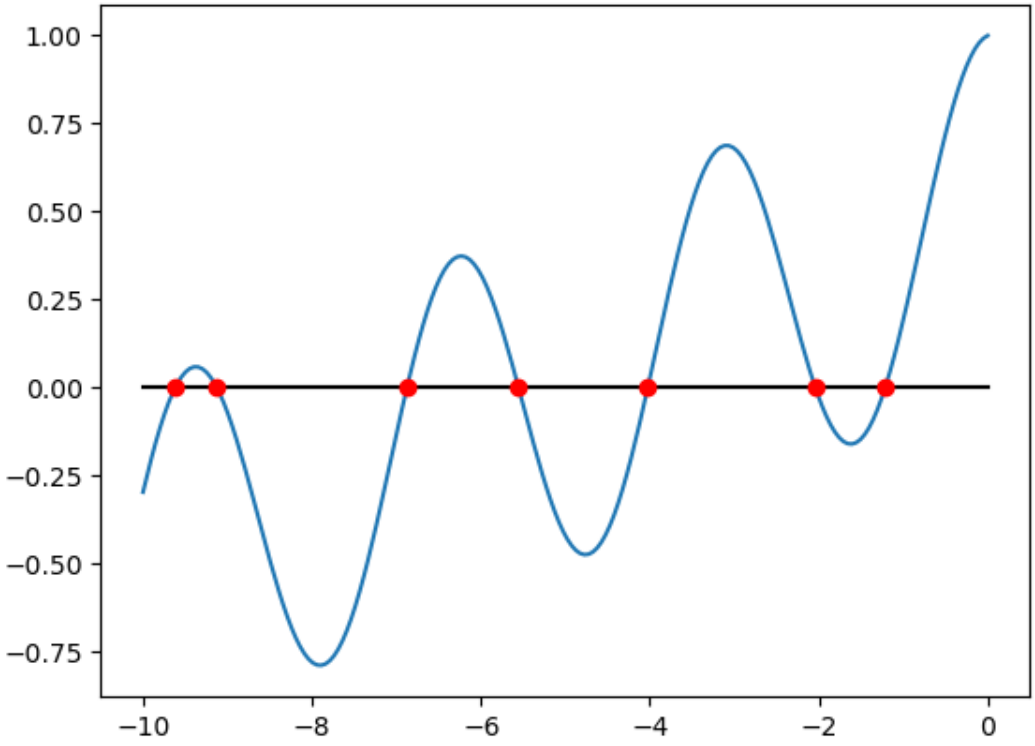
Dibujar la función. El intervalo inicial se encontrará a tanteo, es decir, se probará con varios intervalos iniciales y por el gráfico de la función se decidirá que no hay más raíces.

Repetir la ejecución usando

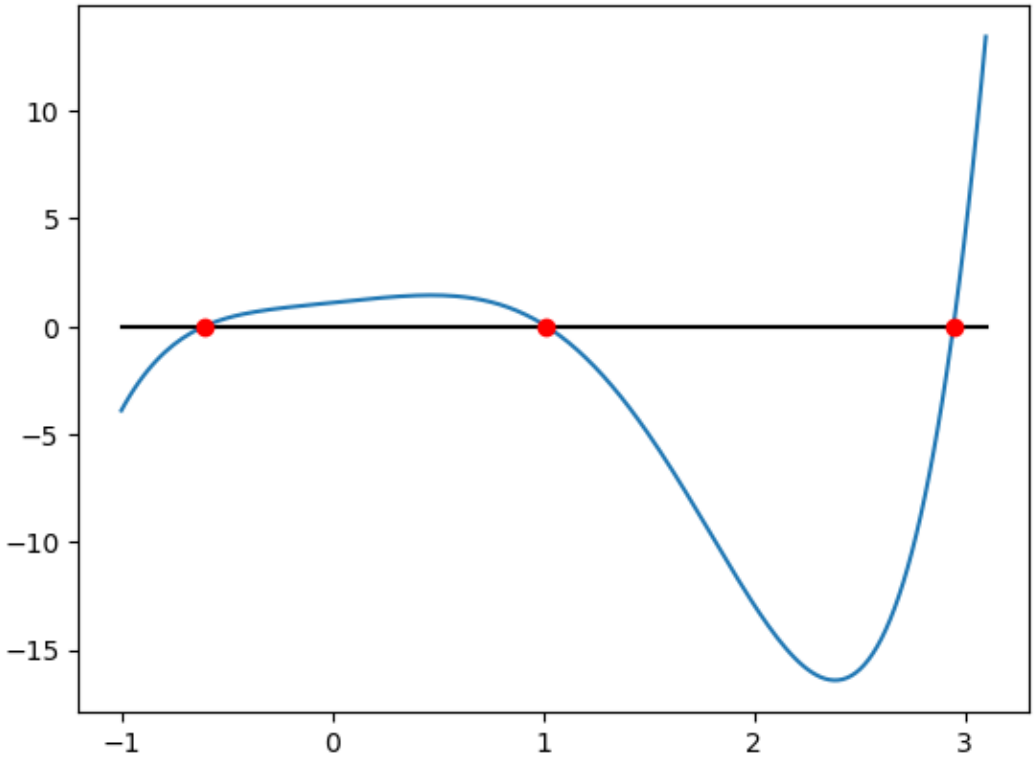
$$f_2(x) = x^5 - 3x^4 + x + 1.1$$

Tener en cuenta que el resultado puede variar ligeramente dependiendo del intervalo $[a,b]$ inicial.

```
%run Ejercicio4
```




```
[-9.6207695  -9.12436905 -6.87625046 -5.55322342 -4.0251091  -2.0393
4402
 -1.214785  ]
```




```
[-0.60776295  1.01631026  2.94630318]
```

Ejercicio 5

Escribir una función **`raices_newton(f,df,a,b)`** que llame a las funciones de los **ejercicios 1** y **3**, y a partir de un intervalo $[a,b]$ y una función continua f, calcule todas las raíces reales de la función contenidas en el intervalo $[a,b]$.

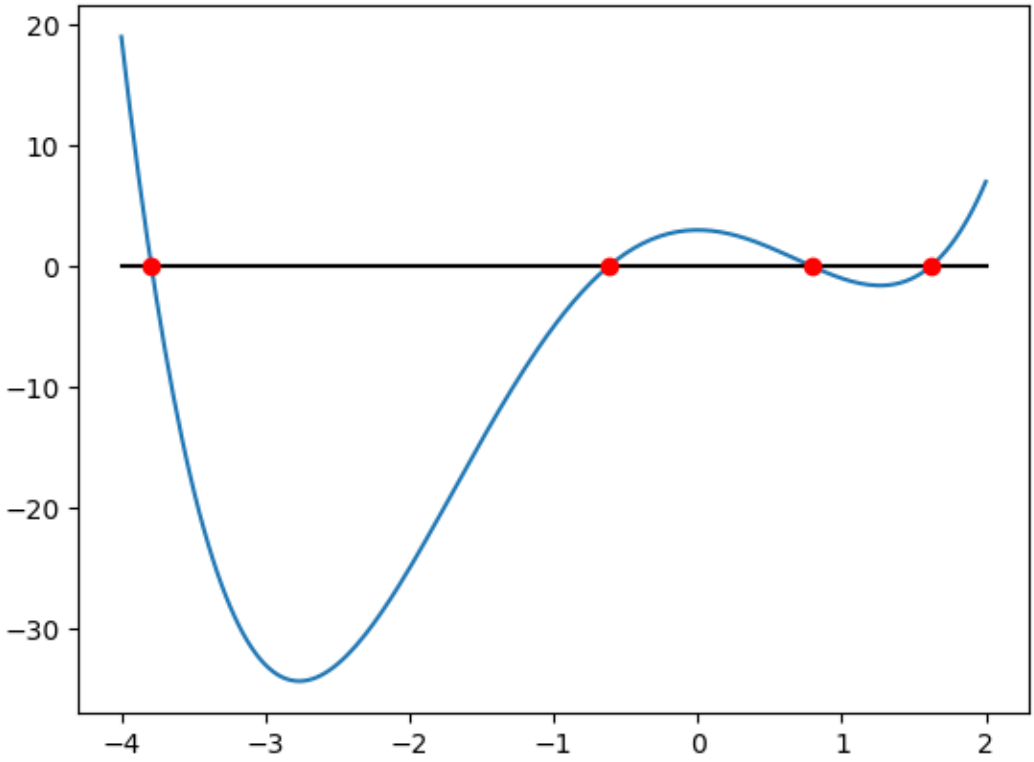
Utilizar dicho programa para calcular todas las raíces reales de los polinomios

$$P_4(x) = x^4 + 2x^3 - 7x^2 + 3 \qquad P_6(x) = x^6 - 0.1x^5 - 17x^4 + x^3 + 73x^2 - 4x - 68$$

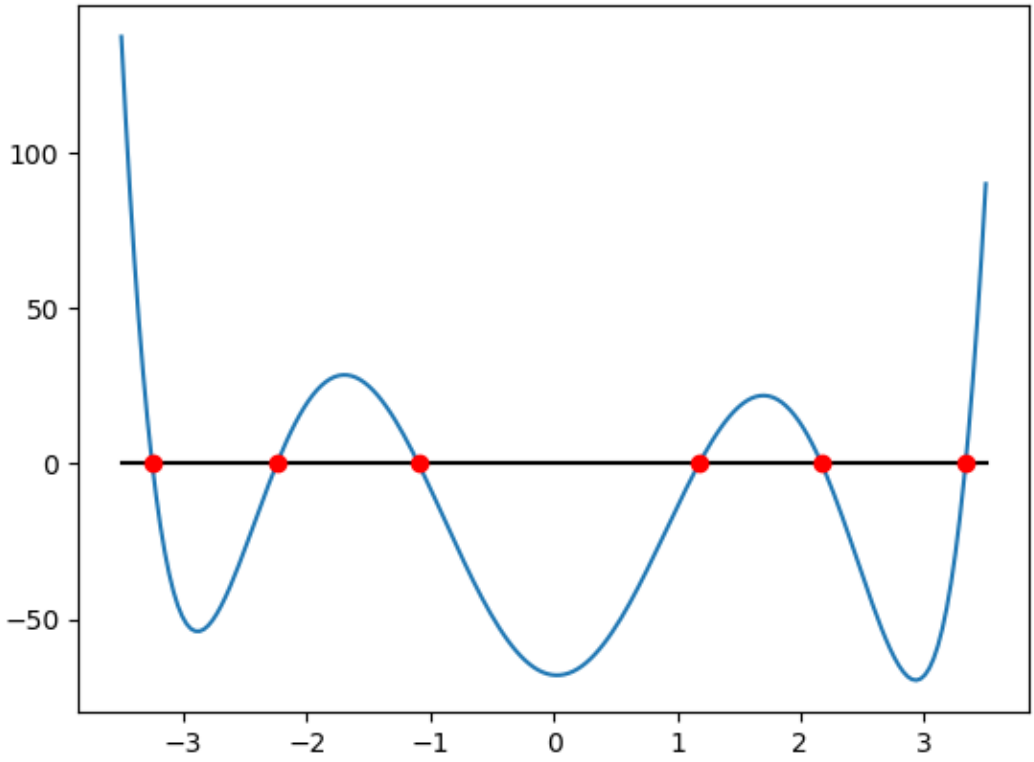
La cota del error absoluto será **`tol = 1.e-6`** , el número máximo de iteraciones **`maxiter = 100`** , y el número de subintervalos **`n = 100`** .

Tener en cuenta que el resultado puede variar ligeramente dependiendo del intervalo $[a,b]$ inicial.

```
%run Ejercicio5
```




```
[-3.79128785 -0.61803399  0.79128785  1.61803399]
```




```
[-3.25062535 -2.23971491 -1.09454575  1.17891488  2.16929806  3.3366
7307]
```

NOTA: Calcular la derivada exacta de una función

Para calcular derivadas de forma simbólica podemos utilizar el módulo sympy.

```
import sympy as sym
```

Comenzamos declarando una variable simbólica y la función

```
x = sym.Symbol('x', real=True)
```

Declaramos la función simbólica y podemos calcular sus derivadas

```
f_sim   = sym.cos(x)*(x**3+1)/(x**2+1)
df_sim  = sym.diff(f_sim,x)
d2f_sim = sym.diff(df_sim,x)
```

Podemos imprimirlas

```
print(df_sim)
```

```
3*x**2*cos(x)/(x**2 + 1) - 2*x*(x**3 + 1)*cos(x)/(x**2 + 1)**2 - (x*
*3 + 1)*sin(x)/(x**2 + 1)
```

Para usarlas como funciones numéricas las *lambificamos*.

```
f   = sym.lambdify([x], f_sim,'numpy')
df  = sym.lambdify([x], df_sim,'numpy')
d2f = sym.lambdify([x], d2f_sim,'numpy')
```

Y ya podemos, por ejemplo, dibujarlas con `plot`

```
x = np.linspace(-3,3,100)

plt.figure()
plt.plot(x,f(x),x,df(x),x,d2f(x))
plt.plot(x,0*x,'k')
plt.legend(['f','df','d2f'])
plt.show()
```