Práctica 1

por Juan Montes Cano, Cristian Pérez Corral y Juan Antonio Gordillo Gayo

1 Raíces de funciones no lineales - Parte 1

1.1 Descripción del algoritmo del método de la bisección.

El método de la bisección es convergente, para su funcionamiento debemos proporcionar una función continua f y un intervalo $[x_1, x_2]$ de manera que el signo de $f(x_1)$ y $f(x_2)$ sea opuesto, así podremos asegurar que existe un punto x_0 tal que $f(x_0) = 0$ (Teorema de Bolzano).

En ocasiones también podremos añadir una cota de error al algoritmo, que determinará el criterio de parada.

En primer lugar consideramos el punto intermedio:

$$x_m = \frac{(x_1 + x_2)}{2}$$

Ahora dependiendo del valor del $f(x_m)$ continuamos de la siguiente forma:

- Si $f(x_m) = 0$ entonces x_m es la raíz que buscamos. Esto será nuestro criterio de parada.

- Si $f(x_m)$ tiene el mismo signo que $f(x_1)$ entonces el valor de la raíz se encuentra en el intervalo (x_m, x_2) puesto que así $f(x_m)$ y $f(x_2)$ tienen valores de distinto signo.

  Ahora aplicaríamos el algoritmo de nuevo a este intervalo hasta encontrar el error.

- Si $f(x_m)$ tiene el mismo signo que $f(x_2)$ entonces el valor de la raíz se encuentra en el intervalo (x_1, x_m) puesto que así $f(x_1)$ y $f(x_m)$ tienen valores de distinto signo.

  Ahora aplicaríamos el algoritmo de nuevo a este intervalo hasta encontrar el error.

Este sería el algoritmo cuando queremos obtener raíces númericas exactas, es decir, con cota de error 0.

Si usamos el algoritmo con una cota de error distinta, el algoritmo sería el siguiente:

En primer lugar, definimos la cota de error como $\varepsilon = x_2 - x_1$, es decir, la cota de error es la distancia entre los dos valores que definen el intervalo.

Al igual que antes usamos una función continua f en un intervalo $[x_1, x_2]$ con una cota de error fija ε.

Así pues, nuestro algoritmo sería el siguiente:

- Si $x_2 - x_1 < \varepsilon$ entonces x_m es la raíz que buscamos. Esto será nuestro criterio de parada.

- Si $f(x_m)$ tiene el mismo signo que $f(x_1)$ entonces el valor de la raíz se encuentra en el

intervalo (x_m, x_2) puesto que así $f(x_m)$ y $f(x_2)$ tienen valores de distinto signo.

Ahora aplicaríamos el algoritmo de nuevo a este intervalo hasta encontrar el error.

– Si $f(x_m)$ tiene el mismo signo que $f(x_2)$ entonces el valor de la raíz se encuentra en el intervalo (x_1, x_m) puesto que así $f(x_1)$ y $f(x_m)$ tienen valores de distinto signo.

  Ahora aplicaríamos el algoritmo de nuevo a este intervalo hasta encontrar el error.

1.2 Implementación del algoritmo del método de la bisección.

```
# 1.- MÉTODO BISECCIÓN
def metodo_Biseccion(f, a, b, epsilon, ite=0):  # f es la funcion, a es el
punto a la izda y b el punto a la derecha (importante f(a) y f(b) deben ser de
signo opuesto)
    x_m = (a + b) / 2
    if ((b - a) < epsilon):
        return x_m, ite
    if (f.subs(x, x_m) == 0):
        return x_m,ite
    elif (f.subs(x, x_m) * f.subs(x, a) > 0):
        return metodo_Biseccion(f, x_m, b, epsilon, ite + 1)
    else:
        return metodo_Biseccion(f, a, x_m, epsilon, ite + 1)
```

Vemos que en nuestra función la implementamos con un esquema recursivo, de acuerdo con el algoritmo presentado. También devolvemos en que iteración la función converge y el valor.

Para recibir la raíz exacta de la función debemos usar el valor epsilon=0.

1.3 Resultados del algoritmo del método de la bisección.

Para comprobar resultados vamos a usar dos funciones:

i. $g(x) = x + \cos(x)$

ii. $h(x) = x - x^2$

Veamos las raíces de g:

Primero buscamos un intervalo que cumpla los requisitos del algoritmo:

Puesto que $g(-2) < 0$ y $g(10) > 10$ el intervalo $[-2, 10]$ es válido. Tomamos como cota de error $\varepsilon = 0.0001$.

Al aplicar la función con estos paramétros

```
g = x + sympy.cos(x)
resultado = metodo_Biseccion(g, -2, 10, 0.0001)
#RESULTADO
(-0.7390899658203125, 17) #RAÍZ,ITERACIONES
```

Nuestra raíz tiene valor -0.7390899658203125

Efectivamente, al calcular el valor de g en ese punto obtenemos -8.08791474471438e-6, que concuerda con lo esperado.

Veamos la gráfico de la función y el resultado del algoritmo:

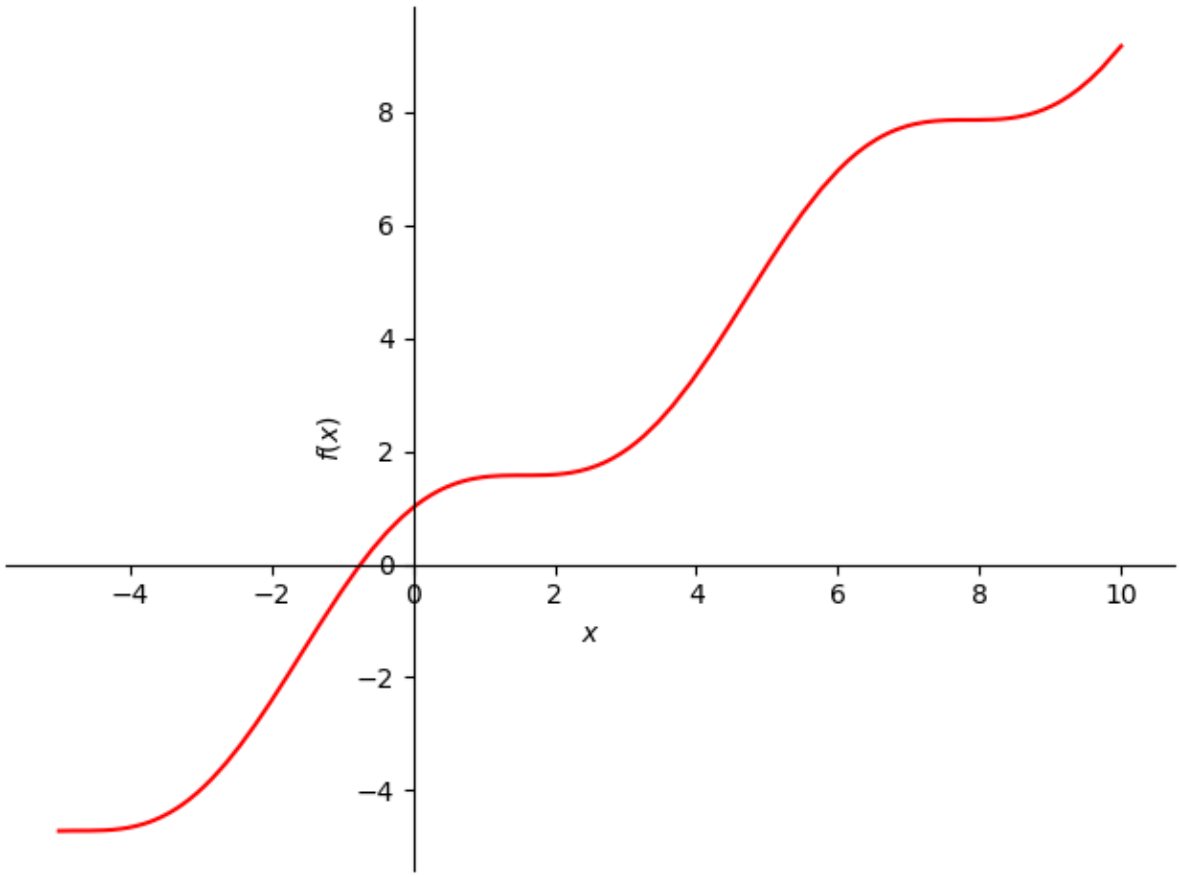

Figura 1.

Ahora analicemos la otra función h. Siguiendo el proceso anterior evaluamos en el intervalo $[2, 0.5]$ y tomaremos como $\varepsilon = 0.01$. Veamos los resultados:

```
h = x - x**2
resultado = metodo_Biseccion(h, 0.4, 2, 0.01)
#RESULTADO
(0.996875, 8) #RAIZ, ITERACIONES
```

Vemos que mientras mayor es la cota de error menor es el número de iteraciones.

Los resultados se corresponden con lo obtenido en la gráfica de h, nótese que la función presenta dos raíces en 0 y 1, como el intervalo cogido contiene a 1 encontró esa raíz, sin embargo, si hubiesemos tomados el intervalo $[-5, 0.5]$ el método nos habría devuelto la raíz 0.

La gráfica es la siguiente:

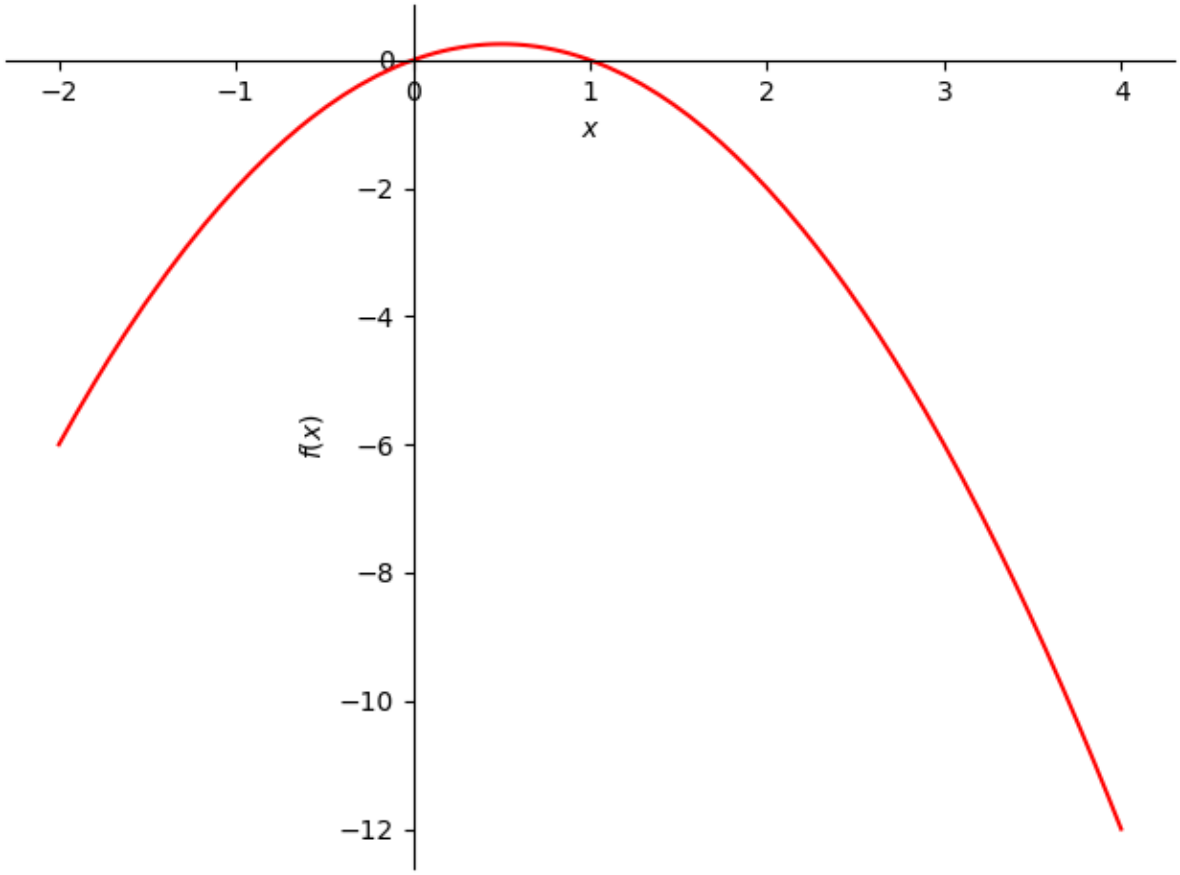

Figura 2.

1.4 Descripción del algoritmo del método de Newton-Raphson.

Es un método que tiene convergencia cuadrática en caso de existir. El algoritmo parte de un punto inicial $(x_0, f(x_0))$ y se busca una estimación del siguiente punto x_1 siguiendo la tangente de la función en el punto anterior.

Al igual que el método anterior es recursivo con un criterio de parada determinado.

Definimos como x_n el punto devuelto por el algoritmo en la iteración n.

El algoritmo recibe una función de clase $\mathcal{C}^1$ f, una valor de inicio x_0 y una cota de error ε.

El algoritmo sigue el siguiente esquema:

- Si $|x_{n+1} - x_n| < \varepsilon$ entonces devolvemos el valor x_{n+1}, es nuestro criterio de parada.
- En caso contrario, calculamos:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

1.5 Implementación del algoritmo del método de Newton-Raphson.

```
# 1.- MÉTODO NEWTON-RAPHSON
```

```
def metodo_NewtonRaphson(f, x_m, epsilon, ite=0,limit =-1):

    #Derivamos la funcion
    f_x = sympy.diff(f, x)
    x_Nuevo = x_m - (f.subs(x, x_m) / f_x.subs(x, x_m)) #CALCULO DEL NUEVO
    #SE CUMPLE CRITERIO DE PARADA
    if ((abs(x_m - x_Nuevo) < epsilon) or limit == 0):
        return x_Nuevo, ite
    #EN EL CASO DE QUE NO TENGAMOS ITERACIONES LIMITADAS
    if(limit < 0):
        return metodo_NewtonRaphson(f,  sympy.N(x_Nuevo), epsilon, ite + 1)
    else:#SI LAS TENEMOS POR CADA ITERACIONES RESTAMOS UNA
        return metodo_NewtonRaphson(f, sympy.N(x_Nuevo), epsilon, ite + 1,limit
-1)
```

Vemos como nuestros valores de entrada son la función f, el valor de inicio x_m, la cota de error ε y luego tenemos dos valores que nos permiten controlar tantas las iteraciones como el límite de las mismas. Estos últimos valores no son necesario introducirlos, pero nos permiten hacer la función mucho más manejable y adaptable a los ejercicios.

Vemos que sigue un esquema recursivo con un caso base determinado por $|x_n - x_{n+1}| < \varepsilon$. Es decir, nuestra cota de error viene dada por la distancia entre los dos puntos de las últimas iteraciones.

1.6 Resultados del algoritmo del método de Newton-Raphson.

Para evaluar el funcionamiento de nuestro algoritmo, vamos a usar las funciones descritas anteriormente y así comparar los resultados con el método de la bisección.

Las funciones que usaremos son:

- $g(x) = x + \cos(x)$ con $\varepsilon = 0.0001$ y $x_0 = -2$
- $h(x) = x + x^2$ con $\varepsilon = 0.01$ y $x_0 = 0.4$

Comenzamos con g:

```
#NEWTON-RAPHSON
resultado = metodo_NewtonRaphson(g, -2, 0.0001)
print(resultado)
(-0.739085133219815, 2) #RESULTADO,ITERACIONES
```

Vemos que obtenemos prácticamente el mismo resultado que usando el método de la bisección pero con 15 iteraciones menos.

Veamos que sucede en el caso de h:

```
#NEWTON-RAPHSON
resultado = metodo_NewtonRaphson(h, 0.4, 0.01)
print(resultado)
(-2.31782539490059e-6, 4) #RESULTADO,ITERACIONES
```

Aquí obtenemos un resultado distinto al del método de la bisección, obtenemos en este caso la raíz correspondiente al número 0, en el método de la bisección encontramos el valor 1.

Que los valores sean distintos es completamente lógico, puesto que en nuestro punto de partida $x_0 = 0.4$ como vemos en la gráfica la pendiente de la tangente en $f(x_0)$ es positiva y $f(x_0) > 0$ entonces, por como está determinado el algoritmo, $x_1 < x_0$, por lo que el algoritmo encontrará la raíz a la izquierda de 0.4.

1.7 Aplicación de los métodos a la $f(x) = x\,\text{sen}\left(\frac{x^2}{2}\right) + e^{-x}$

Primero vamos a ver gráficamente como se comporta esta función:

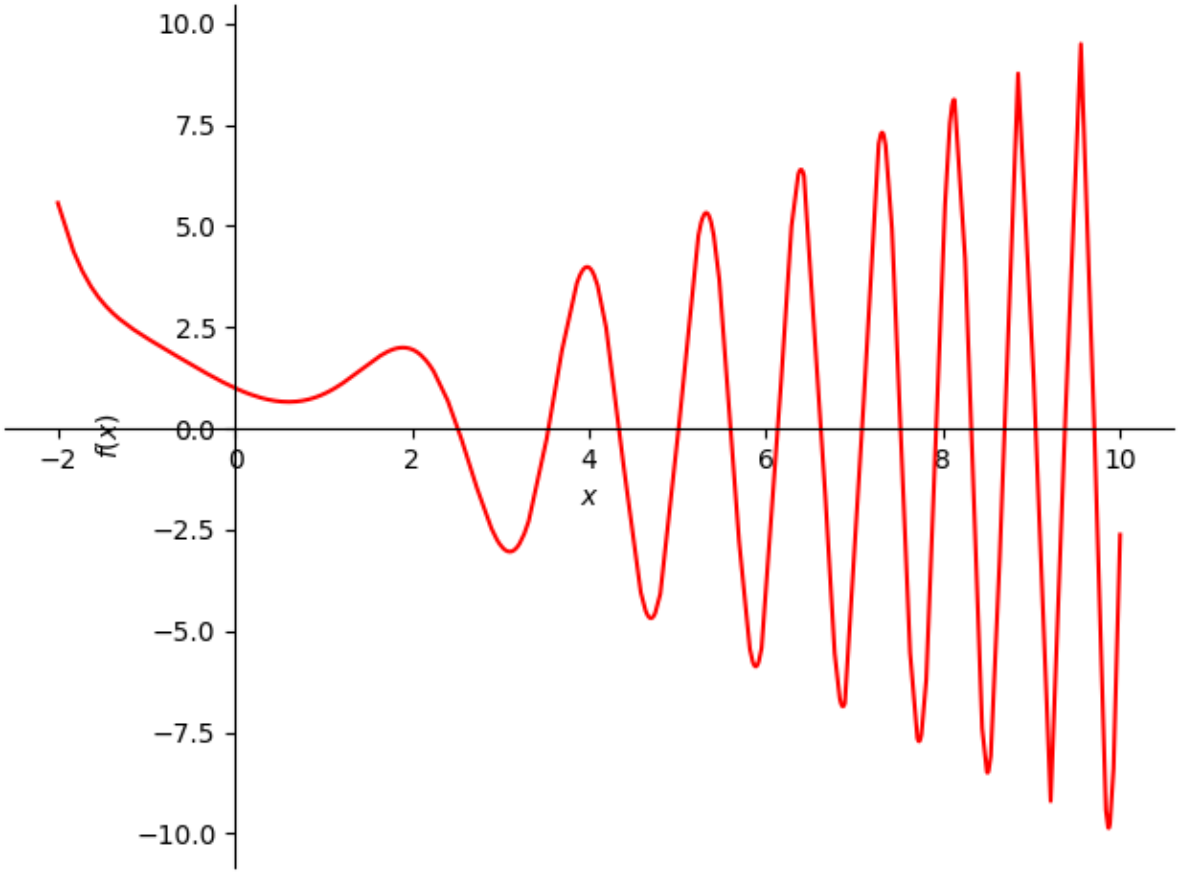

Figura 3.

Observamos que la función oscila, probemos primero usando el método de la bisección:

Vamos a tomar el conjunto $\{0, 3, 4, 5\}$ y vamos a usarlos como los puntos que forman los intervalores para el método de la bisección.

Así pues, obtenemos los siguientes intervalos, $[0, 3]$,$[3, 4]$,$[4, 5]$. Tomemos $\varepsilon = 0.01$.

Entonces obtenemos un código como el siguiente:

```
f = x * sympy.sin(0.5 * x ** 2) + sympy.exp(-x)
resultado = metodo_Biseccion(f, 0, 3, 0.01)
print("sol:{}␣␣(n:{})".format(sympy.N(resultado[0]),resultado[1]))
sol:2.51660156250000  (n:9)
resultado = metodo_Biseccion(f, 3, 4, 0.01)
print("sol:{}␣␣(n:{})".format(sympy.N(resultado[0]),resultado[1]))
sol:3.54296875000000  (n:7)
resultado = metodo_Biseccion(f, 4, 5, 0.01)
print("sol:{}␣␣(n:{})".format(sympy.N(resultado[0]),resultado[1]))
sol:4.33984375000000  (n:7)
```

Podemos ver que dependiendo del intervalo que escojamos obtenemos una raíz distinta.

Comparemos este comportamiento con los mismos puntos usando el método de Newton-Raphson.

```
f = x * sympy.sin(0.5 * x ** 2) + sympy.exp(-x)
resultado = metodo_NewtonRaphson(f,0,0.01)
print("sol:{}␣␣(n:{})".format(sympy.N(resultado[0]),resultado[1]))
sol:2.51935360957807  (n:7)
resultado = metodo_NewtonRaphson(f,3,0.01)
print("sol:{}␣␣(n:{})".format(sympy.N(resultado[0]),resultado[1]))
sol:5.01299206538645  (n:57)
resultado = metodo_NewtonRaphson(f,4,0.01)
print("sol:{}␣␣(n:{})".format(sympy.N(resultado[0]),resultado[1]))
sol:7.08979882780822  (n:5)
resultado = metodo_NewtonRaphson(f,5,0.01)
print("sol:{}␣␣(n:{})".format(sympy.N(resultado[0]),resultado[1]))
sol:5.01299189583714  (n:1)
```

Vemos que el valor de la raíz encontrada depende del punto y su tangente, al igual que en el caso de $x + x^2$. Prestemos atención el caso de $x_0 = 3$. Vemos que realiza 57 iteraciones, una diferencia muy grande con el resto de casos. Esto se debe a que oscila entorno a un punto.

2 Raíces de funciones no lineales - Parte 2

2.1 $f(x) = \text{sen}(x) - 0.3\,e^x$

Primero mostremos gráficamente como es la función:

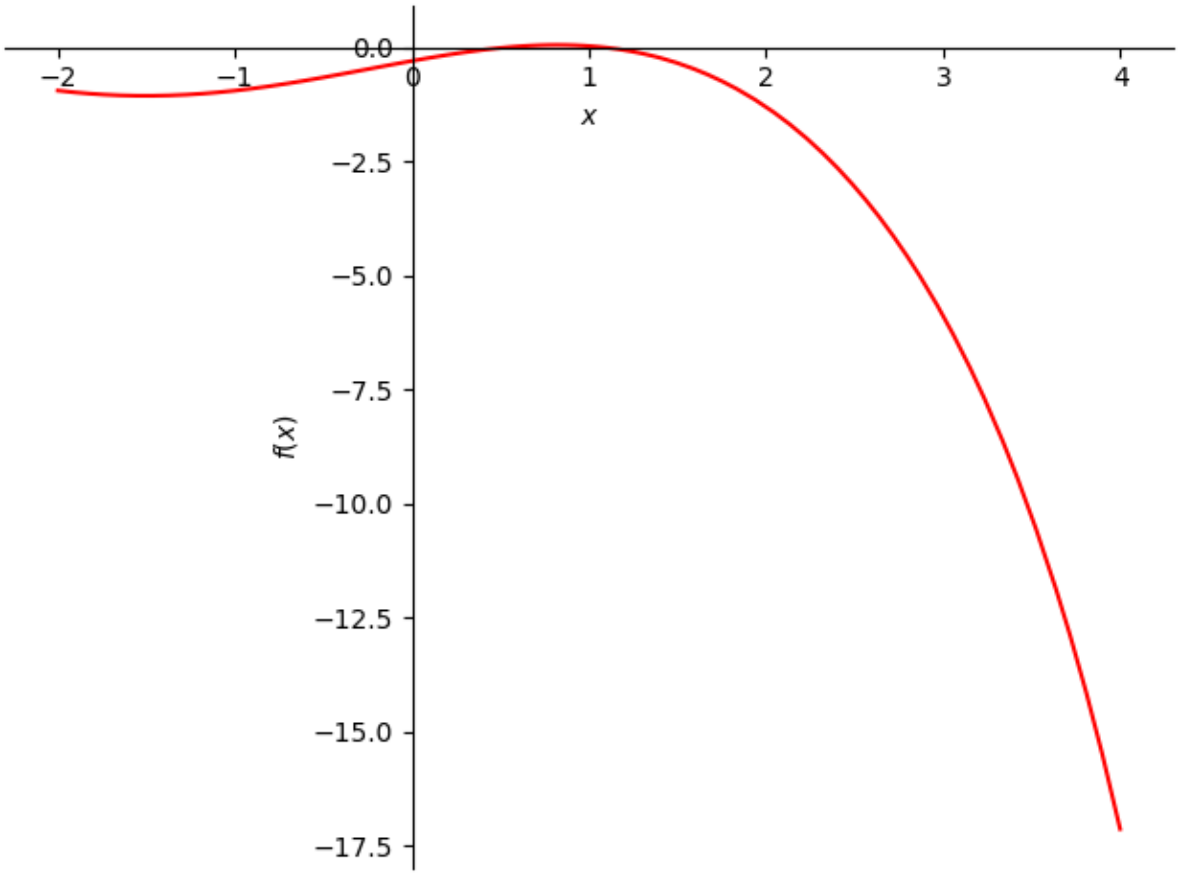

Figura 4.

Ahora aplicamos los métodos:

```
# 2.1 Bisección
aux = metodo_Biseccion(f, 1, 4, 0.01)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:1.07910156250000  (n:9) #Resultado obtenido
# 2.1  Newton-Rapshon
aux = metodo_NewtonRaphson(f, 1, 0.01)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:1.07646496412713  (n:3) #Resultado obtenido
```

Coinciden con los resultados que esperabamos.

2.2 $f(x) = \sqrt{x} - \cos(x)$

Veamos gráficamente la función:

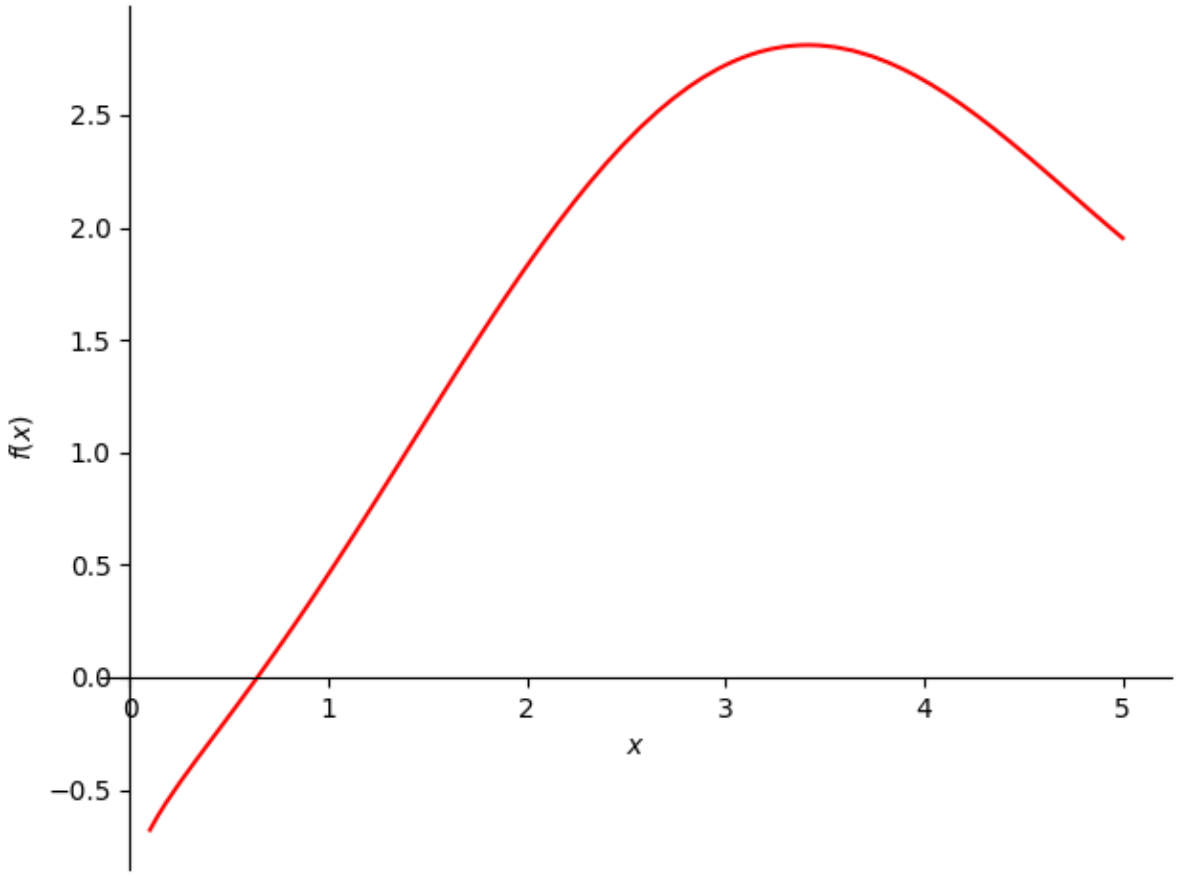

Figura 5.

Aplicamos los métodos en los puntos que nos piden:

```
# 2.2 a)biseccion
aux = metodo_Biseccion(f, 0, 4, 0.001)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:0.642089843750000  (n:12) #Resultado obtenido
# 2.2 a)Newton-Rapshon
aux = metodo_NewtonRaphson(f, 1, 0.001)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:0.641714371002502  (n:3) #Resultado obtenido
```

```
# 2.2 b)biseccion
print("b)")
aux = metodo_Biseccion(f, 0.5, 1, 0.001)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:0.642089843750000  (n:9) #Resultado obtenido
# 2.2 b)Newton-Rapshon
aux = metodo_NewtonRaphson(f, 0.5, 0.001)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:0.641714370872914  (n:3) #Resultado obtenido
```

Coinciden con los resultados esperados.

2.3 $f(x) = 2\,x^3 - 11.7\,x^2 + 17.7\,x^5$

Veamos gráficamente la función:

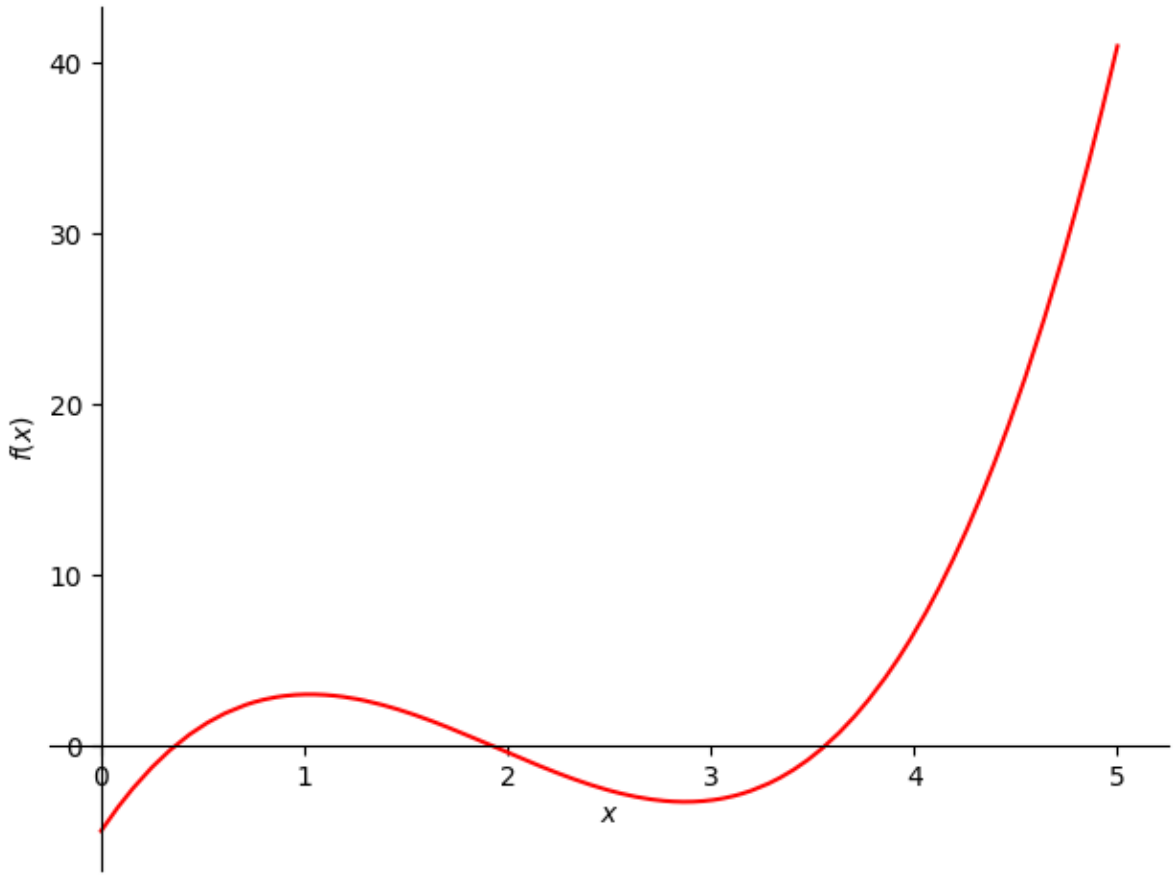

Figura 6.

Calculamos los valores pedidos:

```
print("a)")
# 2.3 a)biseccion (0,1)
aux = metodo_Biseccion(f, 0, 1, 0.001)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:0.364746093750000  (n:10)
# 2.3 a)biseccion (1,3)
```

```
aux = metodo_Biseccion(f, 1, 3, 0.001)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:1.92138671875000  (n:11)
# 2.3 a)biseccion (3,5)
aux = metodo_Biseccion(f, 3, 5, 0.001)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:3.56298828125000  (n:11)
print("b)")
# 2.3 b)Newton-Rapshon 0.5
aux = metodo_NewtonRaphson(f, 0.5, 10 ** (-10))
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:0.365098243362236  (n:5)
# 2.3 b)Newton-Rapshon 1.5
aux = metodo_NewtonRaphson(f, 1.5, 10 ** (-10))
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:1.92174093177571  (n:5)
# 2.3 b)Newton-Rapshon 4
aux = metodo_NewtonRaphson(f, 4, 10 ** (-10))
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:3.56316082486206  (n:6)
```

Los resultados coinciden con lo esperado.

2.4 $f(x) = e^{x/2} + 5\,x - 5$

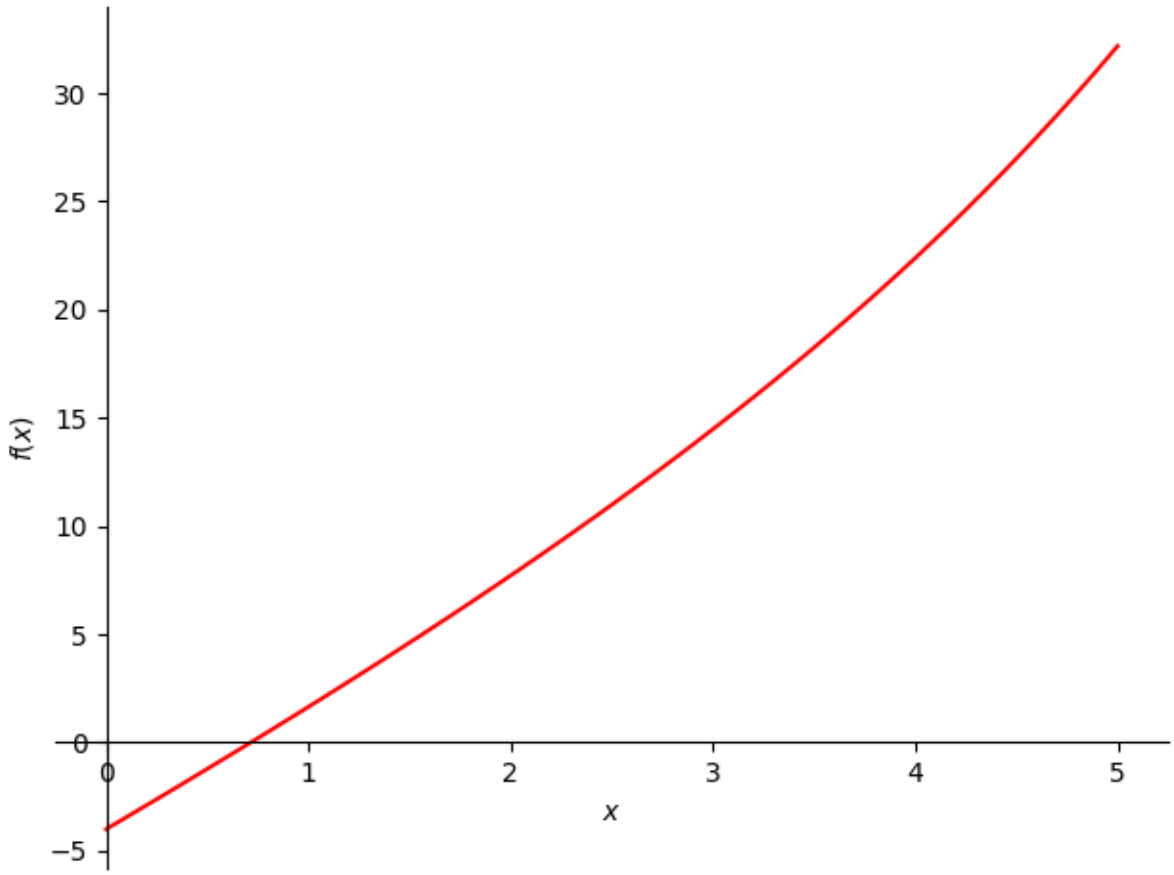

Figura 7.

Calculamos los valores:

```
# 2.4 a)biseccion
print("a)")
aux = metodo_Biseccion(f, 0, 4, 0.001)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:0.714355468750000  (n:12)
# 2.4 b)Newton-Raphson
print("b)")
aux = metodo_NewtonRaphson(f, 1, 0.001)
print("sol:{}␣␣(n:{})".format(sympy.N(aux[0]), aux[1]))
sol:0.714168715029391  (n:3)
```