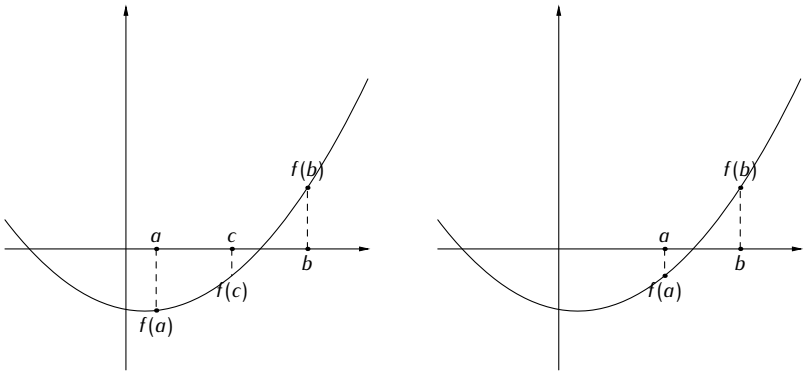
6.6.4. El método de la bisección

El método de la bisección permite encontrar un cero de una función matemática $f(x)$ en un intervalo $[a, b]$ si $f(x)$ es continua en dicho intervalo y $f(a)$ y $f(b)$ son de distinto signo.

El método de la bisección consiste en dividir el intervalo en dos partes iguales. Llamemos c al punto medio del intervalo. Si el signo de $f(c)$ tiene el mismo signo que $f(a)$, aplicamos el mismo método al intervalo $[c, b]$. Si $f(c)$ tiene el mismo signo que $f(b)$, aplicamos el método de la bisección al intervalo $[a, c]$. El método finaliza cuando hallamos un punto c tal que $f(c) = 0$ o cuando la longitud del intervalo de búsqueda es menor que un ϵ determinado.

En la figura de la izquierda te mostramos el instante inicial de la búsqueda: nos piden hallar un cero de una función continua f entre a y b y ha de haberlo porque el signo de $f(a)$ es distinto del de $f(b)$. Calcular entonces el punto medio c entre a y b. $f(a)$ y $f(c)$ presentan el mismo signo, así que el cero no se encuentra entre a y c, sino entre c y b. La figura de la derecha te muestra la nueva zona de interés: a ha cambiado su valor y ha tomado el que tenía c.



Deseamos diseñar un programa que aplique el método de la bisección a la búsqueda de un cero de la función $f(x) = x^2 - 2x - 2$ en el intervalo $[0.5, 3.5]$. No debemos considerar intervalos de búsqueda mayores que 10^{-5}.

Parece claro que implementaremos dos funciones: una para la función matemática $f(x)$ y otra para el método de la bisección. Esta última tendrá tres parámetros: los dos extremos del intervalo y el valor de ϵ que determina el tamaño del (sub)intervalo de búsqueda más pequeño que queremos considerar:

biseccion.py

```
def f(x):
    return x**2 - 2*x -2

def biseccion(a, b, epsilon):
    ...
```

El método de la bisección es un método iterativo: aplica un mismo procedimiento repetidas veces hasta satisfacer cierta condición. Utilizaremos un bucle, pero ¿un **while** o un **for–in**? Obviamente, un bucle **while**: no sabemos a priori cuántas veces iteraremos. ¿Cómo decidimos cuándo hay que volver a iterar? Hay que volver a iterar mientras no hayamos hallado el cero *y, además,* el intervalo de búsqueda sea mayor que ϵ:

biseccion.py

```
def f(x):
    return x**2 - 2*x -2

def biseccion(a, b, epsilon):
    while f(c) != 0 and b - a > epsilon:
        ...
```

Para que la primera comparación funcione c ha de tener asignado algún valor:

biseccion.py

```
def f(x):
    return x**2 - 2*x -2

def biseccion(a, b, epsilon):
    c = (a + b) / 2.0
    while f(c) != 0 and b - a > epsilon:
        ...
```

Dentro del bucle hemos de actualizar el intervalo de búsqueda:

biseccion.py

```
def f(x):
    return x**2 - 2*x -2
```

Parámetros con valor por defecto

La función *biseccion* trabaja con tres parámetros. El tercero está relacionado con el margen de error que aceptamos en la respuesta. Supón que el noventa por cien de las veces trabajamos con un valor de ϵ fijo, pongamos que igual a 10^{-5}. Puede resultar pesado proporcionar explícitamente ese valor en todas y cada una de las llamadas a la función. Python nos permite proporcionar parámetros con un valor por defecto. Si damos un valor por defecto al parámetro *epsilon*, podremos llamar a la función *biseccion* con tres argumentos, como siempre, o con sólo dos.

El valor por defecto de un parámetro se declara en la definición de la función:

```
def biseccion(a, b, epsilon=1e-5):
    ...
```

Si llamamos a la función con *biseccion*(1, 2), es como si la llamásemos así: *biseccion*(1, 2, 1e-5). Al no indicar valor para *epsilon*, Python toma su valor por defecto.

```

def biseccion(a, b, epsilon):
    c = (a + b) / 2.0
    while f(c) != 0 and b - a > epsilon:
        if (f(a) < 0 and f(c) < 0 ) or (f(a) > 0 and f(c) > 0):
            a = c
        elif (f(b) < 0 and f(c) < 0 ) or (f(b) > 0 and f(c) > 0):
            b = c
        ...
```

Las condiciones del **if-elif** son complicadas. Podemos simplificarlas con una idea feliz: dos números x e y tienen el mismo signo si su producto es positivo.

biseccion.py

```
def f(x):
    return x**2 - 2*x -2

def biseccion(a, b, epsilon):
    c = (a + b) / 2.0
    while f(c) != 0 and b - a > epsilon:
        if f(a)*f(c) > 0:
            a = c
        elif f(b)*f(c) > 0:
            b = c
        ...
```

Aún nos queda «preparar» la siguiente iteración. Si no actualizamos el valor de c, la función quedará atrapada en un bucle sin fin. ¡Ah! Y al finalizar el bucle hemos de devolver el cero de la función:

biseccion_3.py — biseccion.py

```
def f(x):
    return x**2 - 2*x -2

def biseccion(a, b, epsilon):
    c = (a + b) / 2.0
    while f(c) != 0 and b - a > epsilon:
        if f(a)*f(c) > 0:
            a = c
        elif f(b)*f(c) > 0:
            b = c
```

```
11        c = (a + b) / 2.0
12    return c
```

Ya podemos completar el programa introduciendo el intervalo de búsqueda y el valor de ϵ:

biseccion.py

```
    ⋮
14 print 'El cero está en:', biseccion(0.5, 3.5, 1e-5)
```

EJERCICIOS

▶ 365 La función *biseccion* aún no está acabada del todo. ¿Qué ocurre si el usuario introduce un intervalo $[a, b]$ tal que $f(a)$ y $f(b)$ tienen el mismo signo? ¿Y si $f(a)$ o $f(b)$ valen 0? Modifica la función para que sólo inicie la búsqueda cuando procede y, en caso contrario, devuelva el valor especial *None*. Si $f(a)$ o $f(b)$ valen cero, *biseccion* devolverá el valor de a o b, según proceda.

▶ 366 Modifica el programa para que solicite al usuario los valores a, b y ϵ. El programa sólo aceptará valores de a y b tales que $a < b$.