Método de bisección (búsqueda binaria)

Estos apuntes están redactados por Maria de los Angeles Isidro Pérez y Egor Maximenko.

Objetivos. Conocer el método de bisección para resolver ecuaciones no lineales.

Requisitos. Teorema del valor intermedio y elementos de programación: comando If, el ciclo `While` y la recursión (opcional).

Teorema del valor intermedio (repaso)

1. Primer teorema del valor intermedio (Bernhard Bolzano y Augustin-Louis Cauchy). Sean $a, b \in \mathbb{R}$ tales que $a < b$, sea $f\colon [a, b] \to \mathbb{R}$ una función continua que toma valores de signos opuestos en a y b, esto es,

$$\big(f(a) < 0 \quad \text{y} \quad f(b) > 0\big) \quad \text{o} \quad \big(f(a) > 0 \quad \text{y} \quad f(b) < 0\big). \tag{1}$$

Entonces existe al menos un punto x en el intervalo (a, b) tal que $f(x) = 0$.

2. Primer teorema del valor intermedio, sin fórmulas. Si una función continua cambia su signo en un intervalo entonces esta necesariamente se anula en este intervalo.

3. Forma breve para escribir la condición que la función tiene signos opuestos en los extremos del intervalo. La condición (1) se puede escribir brevemente de la siguiente manera:

$$f(a)f(b) < 0. \tag{2}$$

4. Multiplicar los signos en vez de valores. Puede ser que los números $f(a)$ y $f(b)$ son muy pequeños y el número $|f(a)f(b)|$ es menor que el número mínimo positivo que se puede representar en la máquina, así que el producto $f(a)f(b)$ se representa en la máquina como el cero ("arithmetic underflow"). Para evitar esta situación, en vez de (2) se usa la siguiente condición que matematicamente es equivalente a (2), pero es más segura para los cálculos:

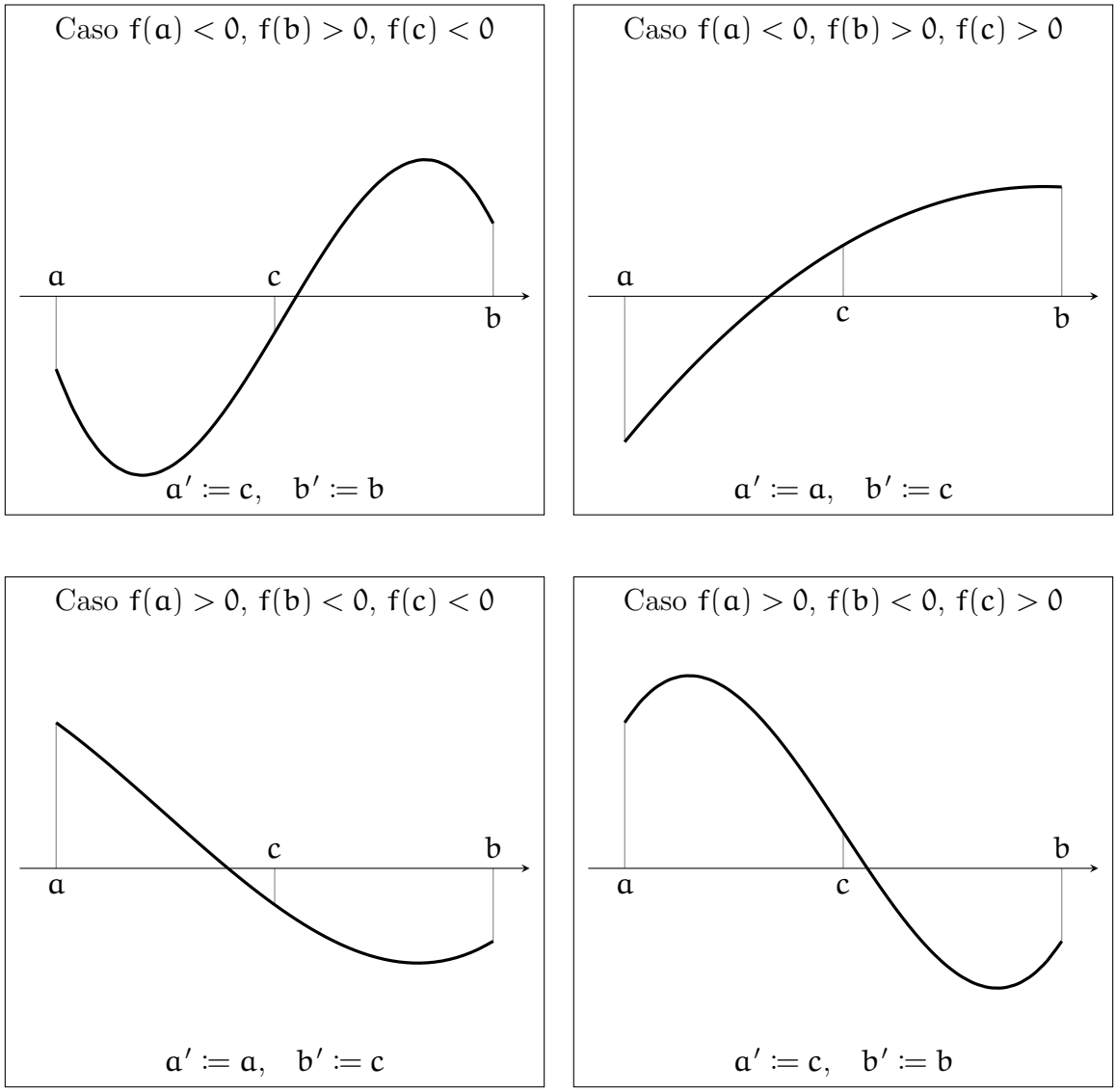
$$\operatorname{sgn}(f(a)) \operatorname{sgn}(f(b)) < 0. \tag{3}$$

5. Idea del método: "divide y vencerás". Sea $f\colon [a,b] \to \mathbb{R}$ una función continua que cumple con (1). Tomamos la aproximación a la raiz c como el punto medio del intervalo [a,b]:

$$c = \frac{a+b}{2}.$$

Si $f(c) = 0$, entonces ya tenemos una solución. Si $f(c)$ tiene el mismo signo que $f(a)$, entonces f cambia su signo en $[c,b]$, y tiene que existir una raíz en este intervalo. Si $f(c)$ tiene el mismo signo que $f(b)$, entonces f cambia su signo en $[a,c]$, y tiene que existir una raíz en este intervalo. En los últimos dos casos, hemos reducido el intervalo de busqueda a la mitad: $[a,c]$ o $[c,b]$.

6. Los siguientes dibujos muestran la elección de los extremos nuevos del intervalo, denotados por a' y b', dependiendo de los signos de $f(a)$, $f(b)$ y $f(c)$.



Condiciones de terminación

7. Tolerancia de las abscisas. ¿Hasta qué momento continuar el proceso de bisección? Vamos a suponer que desde el inicio nos dan una precisión requerida de la respuesta que vamos a denotar por `xtol` (*tolerancia de las abscisas*), y si encontramos un intervalo de longitud $\leq$ `xtol` que contiene una raíz de la función, entonces ya podemos terminar los cálculos.

8. Número máximo de iteraciones. Para aumentar la seguridad podemos elegir también el número máximo de iteraciones, `maxiter`.

9. Problema de precisión. Habitualmente calculamos los valores de `f` aproximadamente, con cierta precisión `ytol` (*tolerancia de los valores*). Si en un punto `c` se tiene que $|\texttt{f}(\texttt{c})| < \texttt{ytol}$, entonces no hay sentido tomar en cuenta el signo de $\texttt{f}(\texttt{c})$, y lo más natural que se puede hacer es terminar los cálculos y regresar `c` como una aproximación a la raíz.

10. Condiciones de terminación. Hay que terminar el proceso de bisección si

$$|\texttt{b} - \texttt{a}| < \texttt{xtol} \qquad \text{o} \qquad |\texttt{f}(\texttt{c})| < \texttt{ytol} \qquad \text{o} \qquad \texttt{i} > \texttt{maxiter},$$

donde `i` es el número de las iteraciones hechas. Usando la ley de De Morgan escriba la condición de continuación.

11. Regresar la aproximación a la raíz y el número de las iteraciones hechas. Para comparar la eficiencia de varios algoritmos vamos a regresar como resultado no solamente la aproximación a la raíz, sino también el número de las iteraciones hechas.

Algorimo de bisección

12. Algoritmo Bisection.

```
Entrada: f, a, b, xtol, ytol, maxiter;
Variables locales: c, fa, fb, fc, i;
fa := f(a); fb := f(b); c := (a + b) / 2; fc := f(c); i = 1;
Mientras (i <= maxiter) y (abs(c - a) >= xtol) y (abs(fc) >= ytol):
   Si sign(fb) * sign(fc) > 0, entonces:
      b = c; fb = fc;
   En otro caso:
      a = c; fa = fc;
   i := i + 1; c = (a + b) / 2; fc := f(c);
Regresar el par de los números c, i.
```

Versión recursiva del algoritmo de bisección

En vez del ciclo `While` se puede usar la recursión. Para no calcular varias veces los valores de la función en un mismo punto vamos a pasar los valores calculados anteriormente como argumentos de la función recursiva.

13. Algoritmo BisectionRecur.

```
Función BisectionR(f, a, b, fa, fb, xtol, ytol, maxiter):
   Vabiables locales: c, fc;
   c := (a + b) / 2; fc := f(c);
   Si (abs(c - a) < xtol) o (abs(fc) < ytol) o (maxiter <= 1):
      Regresar (c, 1);
   En otro caso:
      Si sign(fc) * sign(fa) < 0:
         (c, i) := BisectionR(f, a, c, fa, fc, xtol, ytol, maxiter - 1);
      En otro caso:
         (c, i) := BisectionR(f, c, b, fc, fb, xtol, ytol, maxiter - 1);
      Regresar (c, i + 1);

Función BisectionRecur(f, a, b, xtol, ytol, maxiter):
   Variables locales: fa, fb;
   fa := f(a); fb := f(b);
   Si sign(fa) * sign(fb) > 0:
      Regresar (a, -1);
   En otro caso:
      Regresar BisectionR(f, a, b, f(a), f(b), xtol, ytol, maxiter);
```