Estadística y Optimización

Algorítmica Numérica

Algorítmica numérica

Algorítmica numérica

Índice

Conceptos básicos

Algoritmo:

- ✓ Un algoritmo es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema.

- ✓ Ejemplos: Instrucciones de montaje de un aparato, recetas de cocina, algoritmo de la división para calcular el cociente de dos números, el algoritmo de Euclides para obtener el m.c.d. de dos enteros positivos …

- ✓ De un modo más formal, un algoritmo es una secuencia finita de instrucciones realizables, no ambiguas, cuya ejecución conduce a una resolución de un problema.

- ✓ Algoritmo matemático: Es un **proceso iterativo** que genera una **sucesión o de puntos** siguiendo **un conjunto de instrucciones** determinado y finaliza cuando se cumple **un criterio de terminación.**

APLICACIÓN ALGORÍTMICA:

Dado un punto x_k , aplicando las instrucciones de un algoritmo obtenemos un nuevo punto x_{k+1}. Este proceso puede describirse por medio de una función, que denominamos **aplicación algorítmica** y que denotamos por A.

$$A : D \to D, \text{ donde } D \text{ es el dominio.}$$

Por lo tanto, dado un punto inicial x_0, que denominamos **semilla**, la aplicación algorítmica A genera la secuencia de puntos $\{x_n\} = x_1, x_2, \ldots$, donde $x_{k+1} = A(x_k)$ para cada k. A la transformación de x_k en x_{k+1} a través de la aplicación A se le denomina una **iteración** del algoritmo.

Conceptos básicos

$$x_0$$

$$x_1$$

$$x_2$$

$$\vdots$$

$$x_n$$

$$x_{n+1}$$

$$\vdots$$

$$z$$

CONJUNTO SOLUCIÓN:

Una propiedad deseable de un algoritmo que se diseñe para resolver un determinado problema es que genere una sucesión de puntos que **converja a la (o a una) solución del problema**. Sin embargo, esto no siempre es posible por lo que el algoritmo debe detenerse cuando un punto pertenezca a un determinado conjunto que denominaremos **conjunto solución** y que denotaremos por *S*. Este conjunto estará formado por todos aquellos puntos que satisfagan alguna condición que los haga razonables.

EJEMPLO 1: Algoritmo para multiplicar dos números enteros a y b.

$$a_{n+1} = \left\lfloor \frac{a_n}{2} \right\rfloor, \quad b_{n+1} = 2b_n, \quad P_{n+1} = P_n + \left(\frac{a_n}{2} - a_{n+1} \right) b_{n+1}$$

Si $a_n = 0 \rightarrow P_n$ = solución

$\lfloor a \rfloor$ = parte entera de a

$$a_0 = a, b_0 = b, P_0 = 0$$

Multiplicación de 453 y 26:

Iteración	a_n	b_n	P_n
0	453	26	0
1	226	52	26
2	113	104	26
3	56	208	130
4	28	416	130
5	14	832	130
6	7	1664	130
7	3	3328	1794
8	1	6656	5122
9	0	13312	**11778**

Fin!!

EJEMPLO 2: Dado un número $x > 0$, un algoritmo que sirve para calcular la raíz cuadrada de x es el siguiente:

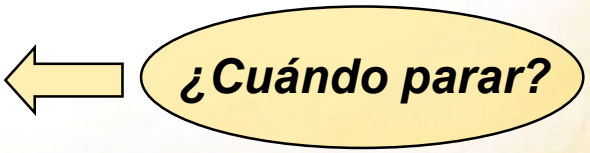
$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{x}{x_n} \right)$$

Calcula $\sqrt{7}$ con este algoritmo, empezando por la semilla $x_0 = 2$:

$$x_1 = \frac{1}{2}\left(2 + \frac{7}{2} \right) = 2.75 \qquad x_2 = \frac{1}{2}\left(2.75 + \frac{7}{2.75} \right) = 2.647727$$

$$x_3 = \frac{1}{2}\left(2.647727 + \frac{7}{2.647727} \right) = 2.645752$$

$$x_4 = \frac{1}{2}\left(2.645752 + \frac{7}{2.645752} \right) = 2.645751$$



CONVERGENCIA:

Una aplicación algorítmica $A : D \to D$ se dice que es **convergente** sobre un subconjunto C del dominio, $C \subset D$, si comenzando en cualquier punto semilla $x_0 \in C$, el límite de la sucesión de puntos $\{x_n\}$ generada por el algoritmo pertenece al conjunto solución S.

ERROR:

En general, los algoritmos no pueden aplicarse indefinidamente y, por lo tanto, debe detenerse por medio de algún criterio. Por ello, después de n iteraciones se alcanzará un punto x_n que tendrá alguna desviación respecto del punto solución z. A la diferencia $\varepsilon_n = |x_n - z|$ se le denomina **error**. El objetivo será, entonces, encontrar alguna **cota del error** que se comete después de un número finito de iteraciones del algoritmo.

- Nota: Si $z \in \mathbf{R}^n$, al definir el error se emplea el módulo en lugar del valor absoluto.

ORDEN DE CONVERGENCIA:

El **orden de convergencia** de forma intuitiva nos indica la velocidad o rapidez a la que una sucesión convergente $\{x_n\}$ converge hacia su límite z.

Si la sucesión $\{x_n\}$ converge a z, entonces se dice que el **orden de convergencia** es $p \geq 1$ si se satisface que

$$\lim_{n \to +\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^p} = \delta$$

donde $\varepsilon_n = |x_n - z|$ es el error.

Nótese que $\varepsilon_n \geq 0$ y, por lo tanto, $\delta \geq 0$.

ORDEN DE CONVERGENCIA:

Esto quiere decir que para n suficientemente grande tendremos que, aproximadamente,

$$\left|x_{n+1} - z\right| \cong \delta \left|x_n - z\right|^p$$

Vemos que, si la diferencia entre x_n y z es pequeña, cuanto más grande sea p más pequeña será la diferencia entre x_{n+1} y z. De este modo, a mayor p (orden de convergencia), mayor rapidez de convergencia (y también cuanto menor sea δ, pero esto es menos importante).

Convergencia, error y complejidad

A partir de un cierto n, donde la sucesión $\{x_n\}$ empiece ya a converger a z

$$\varepsilon_{n+1} \approx \delta \varepsilon_n^{\,p}$$

$$\varepsilon_n = 10^{-2}$$

$$p=2 \quad \rightarrow \quad \varepsilon_{n+1} \approx \delta\left(10^{-2}\right)^2 = \delta 10^{-4} \quad \rightarrow \quad \varepsilon_{n+2} \approx \delta\left(\delta 10^{-4}\right)^2 = \delta^3 10^{-8}$$

$$p=3 \quad \rightarrow \quad \varepsilon_{n+1} \approx \delta\left(10^{-2}\right)^3 = \delta 10^{-6} \quad \rightarrow \quad \varepsilon_{n+2} \approx \delta\left(\delta 10^{-6}\right)^3 = \delta^4 10^{-18}$$

La sucesión ε_n "*llega antes*" a 0 cuando p=3

Convergencia, error y complejidad

	p	δ		*convergencia*
+ lenta	1	$=1$	$\rightarrow$	*sublineal*
	1	<1	$\rightarrow$	*lineal*
	1	$=0$	$\rightarrow$	*superlineal*
	2	>0	$\rightarrow$	*cuadrática*
	3	>0	$\rightarrow$	*cúbica*
+ rápida	p	>0	$\rightarrow$	*orden p*

EJEMPLO 2: Podemos comprobar que el orden de convergencia del algoritmo anterior para calcular la raíz cuadrada de x es cuadrático:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{x}{x_n} \right) \quad y \quad \varepsilon_n = |x_n - z| = |x_n - \sqrt{x}| \quad \Rightarrow \quad ¿ \lim_{n\to+\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^2} = \text{constante?}$$

$$\lim_{n\to+\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^2} = \lim_{n\to+\infty} \frac{|x_{n+1} - \sqrt{x}|}{|x_n - \sqrt{x}|^2} = \lim_{n\to+\infty} \frac{\left|\frac{1}{2}\left(x_n + \frac{x}{x_n}\right) - \sqrt{x}\right|}{|x_n - \sqrt{x}|^2} =$$

$$\lim_{n\to+\infty} \frac{|x_n^2 + x - 2x_n\sqrt{x}|}{2|x_n - \sqrt{x}|^2 x_n} = \lim_{n\to+\infty} \frac{\left|\left(x_n - \sqrt{x}\right)^2\right|}{2|x_n - \sqrt{x}|^2 x_n} = \frac{1}{2\sqrt{x}}$$

nº real

Convergencia, error y complejidad

En relación al orden de convergencia, un algoritmo es **tanto mejor cuanto mayor sea el orden de convergencia p**. En caso de igual orden de convergencia, un algoritmo es mejor cuanto menor sea la constante asintótica del error δ.

EJERCICIO: Disponemos de dos algoritmos A1 y A2 para resolver un mismo problema. Determina cuál de ellos es preferible en las situaciones siguientes y por qué:

1. El orden de convergencia de A1 es 3 y su constante asintótica del error 30, mientras que el orden de convergencia de A2 es 2 y su constante asintótica del error 1.

EJERCICIO: Disponemos de dos algoritmos A1 y A2 para resolver un mismo problema. Determina cuál de ellos es preferible en las situaciones siguientes y por qué:

1. El orden de convergencia de A1 es 3 y su constante asintótica del error 30, mientras que el orden de convergencia de A2 es 2 y su constante asintótica del error 1.

$$A1 \rightarrow p = 3 \quad y \quad \delta = 30$$

$$A2 \rightarrow p = 2 \quad y \quad \delta = 1$$

EJERCICIO: Disponemos de dos algoritmos A1 y A2 para resolver un mismo problema. Determina cuál de ellos es preferible en las situaciones siguientes y por qué:

1. El orden de convergencia de A1 es 3 y su constante asintótica del error 30, mientras que el orden de convergencia de A2 es 2 y su constante asintótica del error 1.

$A1 \rightarrow p = 3 \quad y \quad \delta = 30$

$A2 \rightarrow p = 2 \quad y \quad \delta = 1$

Mejor A1 porque tiene orden de convergencia **MAYOR**
No importa δ

EJERCICIO: Disponemos de dos algoritmos A1 y A2 para resolver un mismo problema. Determina cuál de ellos es preferible en las situaciones siguientes y por qué:

2. El orden de convergencia de A1 es 3 y su constante asintótica del error 30, mientras que el orden de convergencia de A2 es 3 y su constante asintótica del error 1.

EJERCICIO: Disponemos de dos algoritmos A1 y A2 para resolver un mismo problema. Determina cuál de ellos es preferible en las situaciones siguientes y por qué:

2. El orden de convergencia de A1 es 3 y su constante asintótica del error 30, mientras que el orden de convergencia de A2 es 3 y su constante asintótica del error 1.

$$A1 \rightarrow p = 3 \quad y \quad \delta = 30$$

$$A2 \rightarrow p = 3 \quad y \quad \delta = 1$$

EJERCICIO: Disponemos de dos algoritmos A1 y A2 para resolver un mismo problema. Determina cuál de ellos es preferible en las situaciones siguientes y por qué:

2. El orden de convergencia de A1 es 3 y su constante asintótica del error 30, mientras que el orden de convergencia de A2 es 3 y su constante asintótica del error 1.

$A1 \rightarrow p = 3 \quad y \quad \delta = 30$

$A2 \rightarrow p = 3 \quad y \quad \delta = 1$

Mejor A2 porque tiene igual orden de convergencia que A1, pero **constante asintótica del error MENOR**

COMPLEJIDAD:

✓ Para resolver un problema suele ser habitual disponer de varios algoritmos.

Ej: Obtención del m.c.d. de dos números { Cálculo de divisores; Descomponer en factores primos; Algoritmo de Euclides }

✓ Objetivo: Elegir el algoritmo más eficiente.

✓ Complejidad: Medida del grado de dificultad de un algoritmo. Un algoritmo es más eficiente cuanto menos complejo sea.

✓ La complejidad suele medirse en términos de consumo de recursos:

- Complejidad espacial: Cantidad de memoria que requiere el algoritmo.
- **Complejidad temporal**: Tiempo que necesita el algoritmo para ejecutarse.

Convergencia, error y complejidad

- ✓ El tiempo que tarda un algoritmo en resolver un problema depende de:
  - El tamaño del problema (no se tarda lo mismo en ordenar 10 números que 1000)
  - Los datos de entrada (encontrar el valor 3 en la lista {2, 4, 3, 5, 6} o en {3, 2, 4, 1, 6})

- ✓ Tamaño de un problema: Valor o valores que se pueden obtener de los datos de entrada y que, si varían, repercuten en la duración del algoritmo para resolver dicho problema. (Ej: Ordenación de un vector → número de elementos; Factorizar un número en sus factores primos → valor del número)

- ✓ Unidad de medida de la complejidad: Ninguna de tiempo, porque variaría de un ordenador a otro → *Instrucciones que debe realizar el algoritmo*.
  (Simplificación: Suponemos que cada instrucción se ejecuta en un tiempo constante)

Objetivo: Obtener una función genérica $f(n)$ que determine cómo crece el nº de instrucciones necesarias para resolver el problema en función del tamaño, n.

EJEMPLO 1: Algoritmo para multiplicar dos números enteros a y b.

$$a_{n+1} = \left\lfloor \frac{a_n}{2} \right\rfloor, \quad b_{n+1} = 2b_n, \quad P_{n+1} = P_n + \left(\frac{a_n}{2} - a_{n+1} \right) b_{n+1} \quad \text{y} \quad \text{finaliza cuando } a_n = 0.$$

- En cada iteración se realizan 6 operaciones elementales (una división, un truncamiento, dos multiplicaciones, una resta y una suma).

EJEMPLO 1: Algoritmo para multiplicar dos números enteros a y b.

$$a_{n+1} = \left\lfloor \frac{a_n}{2} \right\rfloor, \qquad b_{n+1} = 2b_n, \qquad P_{n+1} = P_n + \left(\frac{a_n}{2} - a_{n+1} \right) b_{n+1}$$

1 división
1 truncamiento

1 multiplicación

1 resta
1 multiplicación
1 suma

EJEMPLO 1: Algoritmo para multiplicar dos números enteros a y b.

$$a_{n+1} = \left\lfloor \frac{a_n}{2} \right\rfloor, \quad b_{n+1} = 2b_n, \quad P_{n+1} = P_n + \left(\frac{a_n}{2} - a_{n+1} \right) b_{n+1} \quad \text{y} \quad \text{finaliza cuando } a_n = 0.$$

- En cada iteración se realizan 6 operaciones elementales (una división, un truncamiento, dos multiplicaciones, una resta y una suma).
- El número de iteraciones depende del valor $a \rightarrow$ Tamaño = a.
- Sabemos que existe un entero k tal que $2^k \leq a \leq 2^{k+1} \rightarrow n^o\, iteraciones = k+1$.
- Tomando logaritmos: $k \leq \log_2(a) \leq k+1 \rightarrow n^o\, iteraciones = \lfloor \log_2(a) \rfloor + 1$

$$f(a) = 6(\lfloor \log_2(a) \rfloor + 1)$$

EJEMPLO 1: Existe un entero k tal que $2^k \leq a \leq 2^{k+1} \rightarrow n^o\ iteraciones = k+1$

EJEMPLO 1: Existe un entero k tal que $2^k \leq a \leq 2^{k+1} \rightarrow n^o\ iteraciones = k+1$

$45 \rightarrow 32 = 2^5 \leq 45 \leq 2^6 = 64$

$103 \rightarrow 64 = 2^6 \leq 103 \leq 2^7 = 128$

$2020 \rightarrow 1024 = 2^{10} \leq 2020 \leq 2^{11} = 2048$

EJEMPLO 1: Existe un entero k tal que $2^k \leq a \leq 2^{k+1} \rightarrow n^o\ iteraciones = k+1$

$45 \rightarrow 32 = 2^5 \leq 45 \leq 2^6 = 64 \longrightarrow n^o\ iteraciones = 6$

$103 \rightarrow 64 = 2^6 \leq 103 \leq 2^7 = 128 \longrightarrow n^o\ iteraciones = 7$

$2020 \rightarrow 1024 = 2^{10} \leq 2020 \leq 2^{11} = 2048 \longrightarrow n^o\ iteraciones = 11$

EJEMPLO 1: Existe un entero k tal que $2^k \leq a \leq 2^{k+1} \rightarrow n^o\ iteraciones = k+1$

$$45 \rightarrow 32 = 2^5 \leq 45 \leq 2^6 = 64 \longrightarrow n^o\ iteraciones = 6$$

$$log_2(32) = 5 \leq log_2(45) = 5.49 \leq log_2(64) = 6$$

$$log_2(45) = 5.49 \rightarrow n^o\ iteraciones = 6$$

EJEMPLO 1: Existe un entero k tal que $2^k \leq a \leq 2^{k+1} \rightarrow n^o\ iteraciones = k+1$

$45 \rightarrow 32 = 2^5 \leq 45 \leq 2^6 = 64 \longrightarrow n^o\ iteraciones = 6$

$log_2(45) = 5.49$

$103 \rightarrow 64 = 2^6 \leq 103 \leq 2^7 = 128 \longrightarrow n^o\ iteraciones = 7$

$log_2(103) = 6.69$

$2020 \rightarrow 1024 = 2^{10} \leq 2020 \leq 2^{11} = 2048 \longrightarrow n^o\ iteraciones = 11$

$log_2(2020) = 10.98$

EJEMPLO 1: Algoritmo para multiplicar dos números enteros a y b.

$log_2(453) = 8.82$

n^o *iteraciones* = 9

Si $a_n = 0 \rightarrow P_n$ *= solución*

Multiplicación de 453 y 26:

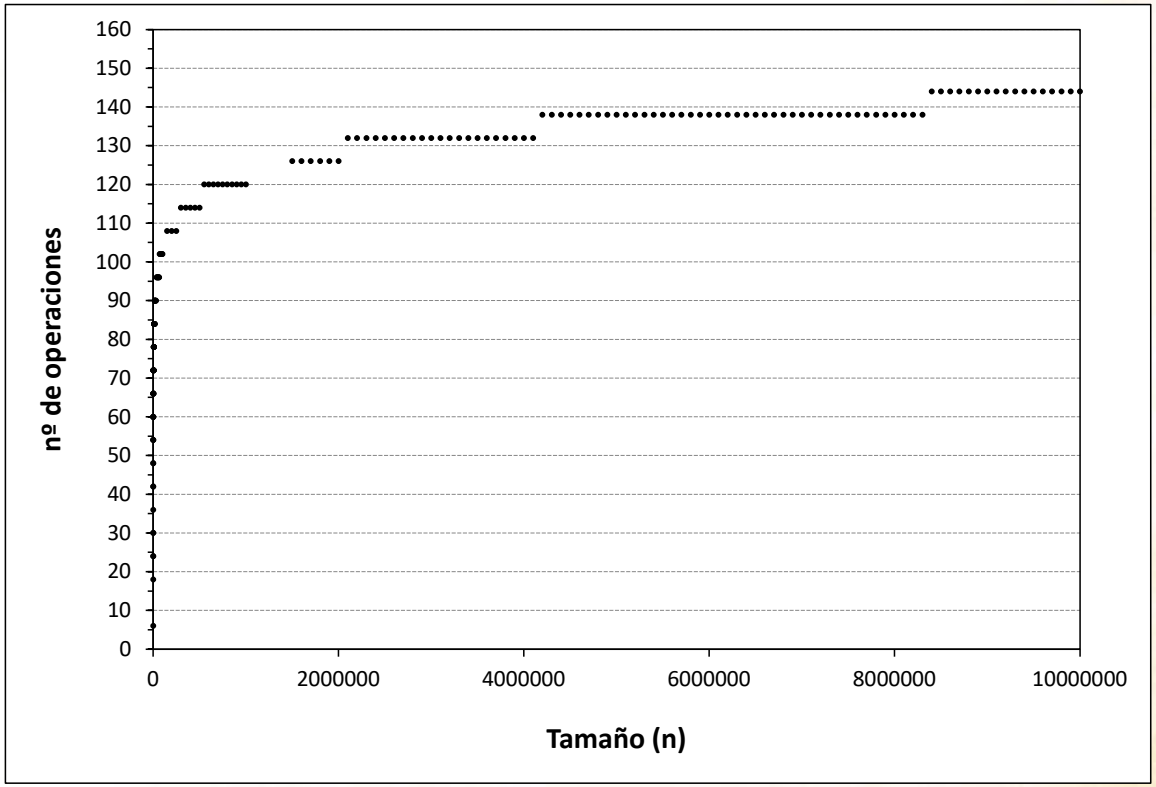
Iteración	a_n	b_n	P_n
0	453	26	0
1	226	52	26
2	113	104	26
3	56	208	130
4	28	416	130
5	14	832	130
6	7	1664	130
7	3	3328	1794
8	1	6656	5122
9	0	13312	**11778**

Fin!!

Convergencia, error y complejidad

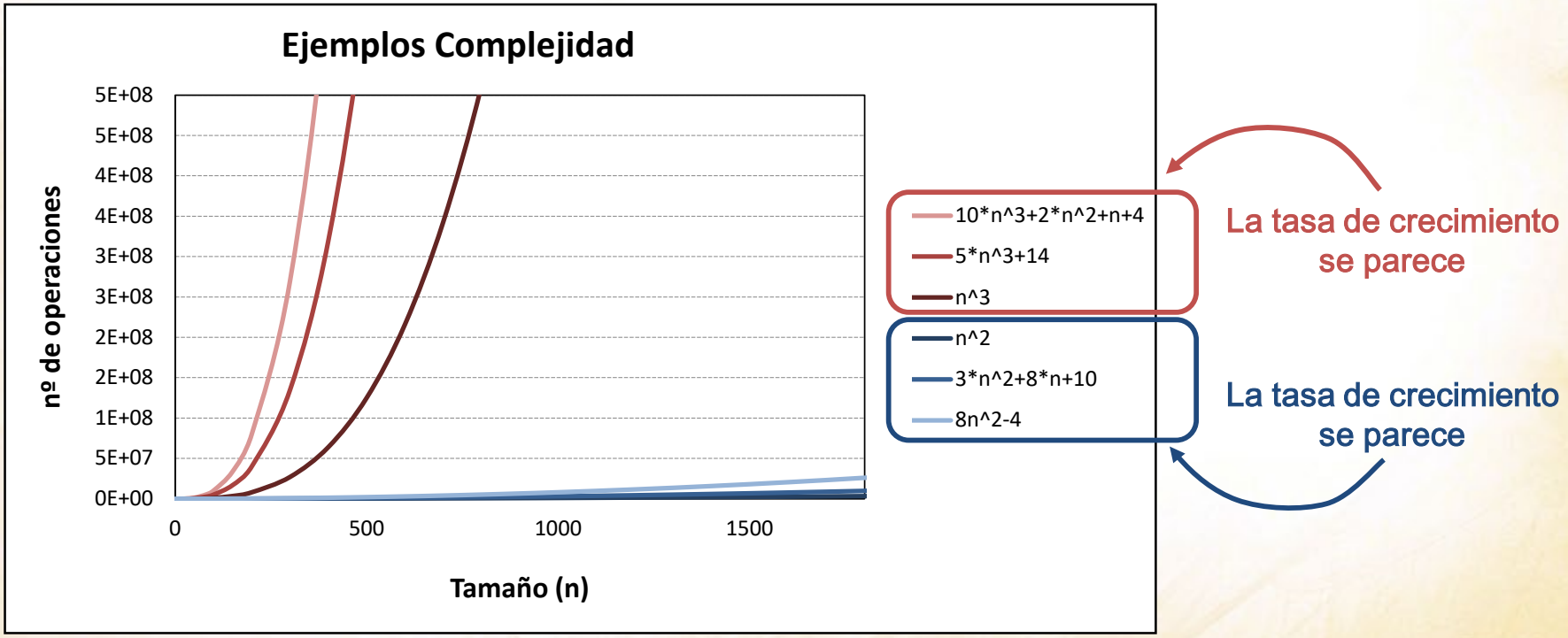
- Evolución del nº de operaciones necesarias para calcular $a \times b$ en función del tamaño del problema, n ($= a$):

n	$f(n)$
1	6
5	18
10	24
15	24
20	30
50	36
100	42
200	48
500	54
1000	60
5000	78
10000	84
50000	96
100000	102
1000000	120
10000000	144



¿Cómo se comparan distintos algoritmos?

- ✓ Se estudia cómo crece el tiempo de ejecución de un algoritmo en función del tamaño del problema, *n*, especialmente para valores de *n* muy grandes, y se agrupan todas las complejidades que crecen igual en un mismo grupo llamado **orden de complejidad**.



Para poder determinar y expresar correctamente el comportamiento asintótico de las funciones $f(n)$ es conveniente disponer de una adecuada notación $\rightarrow$ **Notación de Landau o notación de la O grande**.

Dadas las funciones $f(n)$ y $g(n)$ definidas de **N** en $\mathbf{R}_+$, se dice que $f(n) \in O(g(n))$ si existe una constante positiva c y un valor n_0 tales que :

$$f(n) \leq cg(n) \quad \forall n > n_0.$$

Ejemplos:

- $2n+10$ es $O(n)$: $2n+10 \leq 3n$ cuando $n \geq 10$
- $5n^3+3n^2+1 \in O(n^3)$: $5n^3+3n^2+1 \leq 6n^3,\ n \geq 3$

La sentencia "***f***(***n***) es ***O***(***g***(***n***))" significa que ***f***(***n***) no crece más deprisa que ***g***(***n***).

Nota: $5n^3 + 3n^2 + 1 \in O(n^4)$, pero $5n^3 + 3n^2 + 1 \notin O(n^2)$

- ✓ Todas las funciones del mismo orden de complejidad tienen un comportamiento asintótico que **está acotado superiormente, salvo constantes**.

- ✓ Estos grupos de funciones se denominan **O**, existiendo una infinidad de ellos. Para cada uno de ellos se suele identificar un miembro que se utiliza como representante de la clase.

- ✓ Propiedades de la notación:
  - Todas las funciones de tiempo constante son O(1).
  - $O(log_a n) = O(log_b n)$

    Por esta razón no es necesario especificar la base del logaritmo: $O(logn)$.

Orden	Nombre	Comentario
O(1)	Constante	Todos aquellos algoritmos que responden en un tiempo constante, sea cual sea el tamaño del problema. Son los que aplican alguna fórmula sencilla, por ejemplo, hallar el máximo de dos valores
O(log n)	Logarítmico	Los que el tiempo crece con un criterio logarítmico, independientemente de cuál sea la base mientras esta sea mayor que 1. Por eso, normalmente, ni siquiera se indica la base. No suelen ser muchos, y normalmente están bien considerados, ya que implican que una iteración realiza *menos* operaciones que el tamaño del problema, lo cual no suele ser muy común. Por ejemplo, el algoritmo para multiplicar dos enteros que hemos visto.
O(n)	Lineal	El tiempo crece linealmente con respecto al tamaño. Por ejemplo, encontrar el máximo de un vector de tamaño n.
O(n log n)	Enelogarímico, loglineal...	Este orden tiene muchos nombres. Es un orden relativamente bueno, porque la mayor parte de los algoritmos tienen un orden superior. En este orden está, por ejemplo, el algoritmo de ordenación Quicksort, o la transformada rápida de Fourier.
$O(n^p)$, con $p>1$	Polinómico	Aquí están muchos de los algoritmos más comunes. Cuando p es 2 se le llama *cuadrático*, cuando es 3 se le llama *cúbico*, y en general, polinómico. Intuitivamente podríamos decir que este orden es el último de los aceptables (siempre y cuando p sea relativamente bajo). A partir del siguiente, los algoritmos son complicados de tratar en la práctica cuando n es muy grande.
$O(c^n)$, con $c>1$	Exponencial	Es mucho peor que el anterior. Crece muchísimo más rápidamente.
O(n!)	Factorial	Es el típico de aquellos algoritmos que para un problema complejo prueban todas las combinaciones posibles.
$O(n^n)$	Combinatorio	Tan intratable como el anterior. A menudo no se hace distinción entre ellos.

P

NP

Convergencia, error y complejidad

Los problemas pueden clasificarse de acuerdo con su complejidad, medida ésta en términos de la existencia o no de algoritmos de determinada complejidad para su resolución.

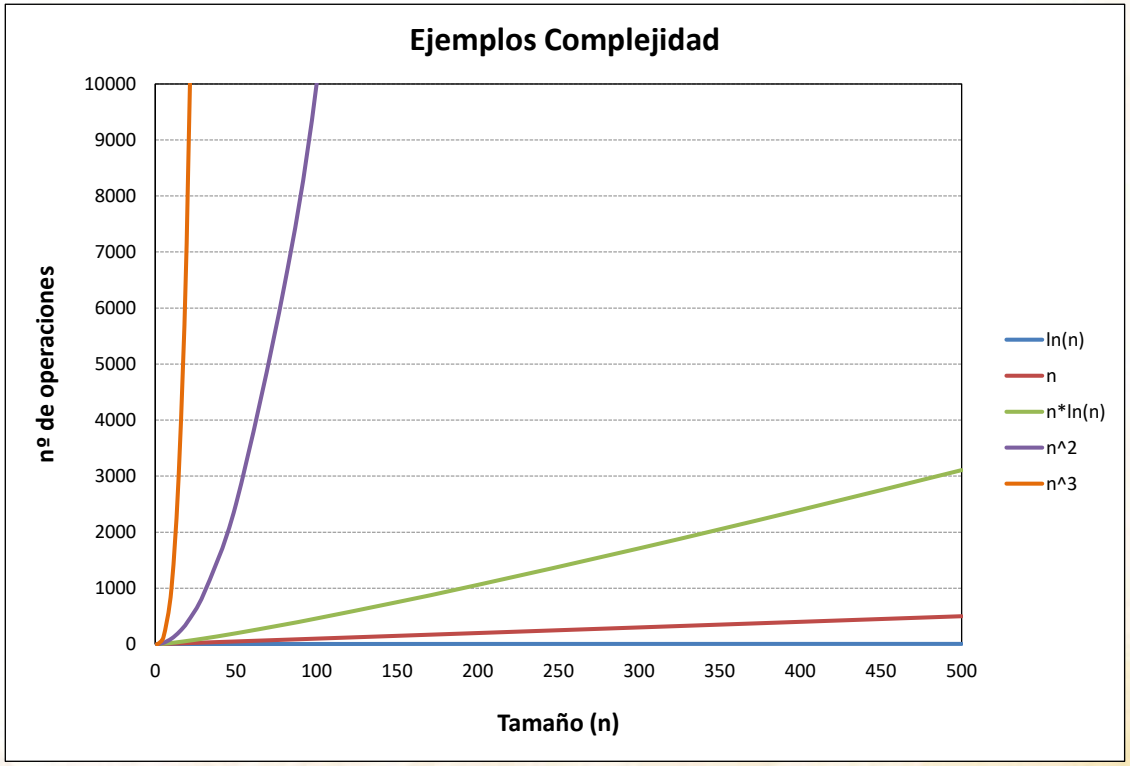
A grosso modo, podemos clasificar los problemas en *P* y *NP*, con las siguientes definiciones intuitivas:

P es la clase de todos los problemas para los que existe al menos un algoritmo que los resuelve al menos en tiempo polinomial o con una cantidad de recursos polinomial.

NP es la clase de todos los problemas para los que no existe o no se conoce un algoritmo que los resuelve al menos en tiempo polinomial o con una cantidad de recursos polinomial.

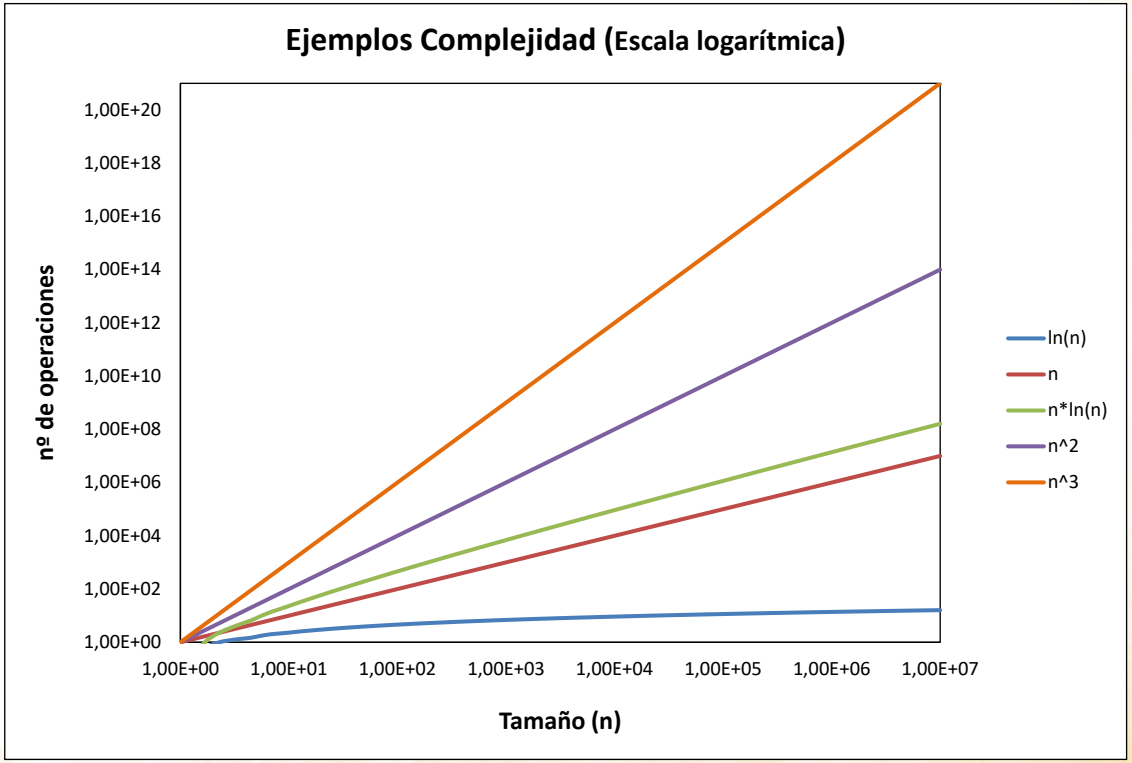
Convergencia, error y complejidad

n	ln(n)	n	n*ln(n)	n^2	n^3
1	0	1	0	1	1
10	2,30	10	23,03	100	1000
100	4,61	100	460,52	10000	1000000
1000	6,91	1000	6907,76	1000000	1000000000
10000	9,21	10000	92103,40	100000000	1E+12
100000	11,51	100000	1151292,55	1E+10	1E+15
1000000	13,82	1000000	13815510,56	1E+12	1E+18
10000000	16,12	10000000	161180956,51	1E+14	1E+21



Convergencia, error y complejidad

n	ln(n)	n	n*ln(n)	n^2	n^3
1	0	1	0	1	1
10	2,30	10	23,03	100	1000
100	4,61	100	460,52	10000	1000000
1000	6,91	1000	6907,76	1000000	1000000000
10000	9,21	10000	92103,40	100000000	1E+12
100000	11,51	100000	1151292,55	1E+10	1E+15
1000000	13,82	1000000	13815510,56	1E+12	1E+18
10000000	16,12	10000000	161180956,51	1E+14	1E+21



Convergencia, error y complejidad

- ✓ Tiempos empleados para el cálculo de algoritmos con distintos ordenes, considerando que el computador en cuestión ejecuta 1 Millón de operaciones por segundo (1MHz).

n	Tiempo(n), en segundos						
	log n	n	nlogn	n^2	n^3	2^n	n!
10	2,3E-06	1,0E-05	2,3E-05	0,0001	0,001	0,0010	3,6288
50	3,9E-06	5,0E-05	2,0E-04	0,0025	0,125	1,1E+09	3,0E+58
100	4,6E-06	1,0E-04	4,6E-04	0,01	1	1,3E+24	9,3E+151
1000	6,9E-06	0,001	0,01	1	1000	1,1E+295	#¡NUM!
10000	9,2E-06	0,01	0,09	100	1000000	#¡NUM!	#¡NUM!
100000	1,2E-05	0,1	1,15	10000	1E+09	#¡NUM!	#¡NUM!
1000000	1,4E-05	1	13,82	1000000	1E+12	#¡NUM!	#¡NUM!

INTRATABLES

1E+09 seg ≈ 31.71 años!!!

EJERCICIO: Disponemos de dos algoritmos A1 y A2 para resolver un mismo problema. Determina cuál de ellos es preferible en las situaciones siguientes y por qué:

1. La complejidad de A1 es O(log n) y la de A2 es O(n!).

EJERCICIO: Disponemos de dos algoritmos A1 y A2 para resolver un mismo problema. Determina cuál de ellos es preferible en las situaciones siguientes y por qué:

1. La complejidad de A1 es O(log n) y la de A2 es O(n!).

A1, porque el número de operaciones que se tienen que realizar para resolver un problema un tamaño n crece mucho más lentamente al aumentar n que el correspondiente para A2

EJERCICIO: Disponemos de dos algoritmos A1 y A2 para resolver un mismo problema. Determina cuál de ellos es preferible en las situaciones siguientes y por qué:

2) La complejidad de A1 es lineal y la de A2 es loglineal

EJERCICIO: Disponemos de dos algoritmos A1 y A2 para resolver un mismo problema. Determina cuál de ellos es preferible en las situaciones siguientes y por qué:

2) La complejidad de A1 es lineal y la de A2 es loglineal

A1, porque el número de operaciones que se tienen que realizar para resolver un problema un tamaño n crece más lentamente al aumentar n que el correspondiente para A2

EJERCICIO: Determina el orden de complejidad de las siguientes funciones:

1) $f(n) = 13n^3 - 8n + 12$

2) $f(n) = 5n^2 + n\ln(n) + 3$

3) $f(n) = 13n! - 8\log_2(n) + 12$

EJERCICIO: Determina el orden de complejidad de las siguientes funciones:

1) $f(n) = 13n^3 - 8n + 12 \longrightarrow f(n) \in O(n^3)$

2) $f(n) = 5n^2 + n\ln(n) + 3$

3) $f(n) = 13n! - 8\log_2(n) + 12$

EJERCICIO: Determina el orden de complejidad de las siguientes funciones:

1) $f(n) = 13n^3 - 8n + 12 \longrightarrow f(n) \in O(n^3)$

2) $f(n) = 5n^2 + n\ln(n) + 3 \longrightarrow f(n) \in O(n^2)$

3) $f(n) = 13n! - 8\log_2(n) + 12$

EJERCICIO: Determina el orden de complejidad de las siguientes funciones:

1) $f(n) = 13n^3 - 8n + 12 \longrightarrow f(n) \in O(n^3)$

2) $f(n) = 5n^2 + n\ln(n) + 3 \longrightarrow f(n) \in O(n^2)$

3) $f(n) = 13n! - 8\log_2(n) + 12 \longrightarrow f(n) \in O(n!)$

EJERCICIO: Determina el orden de complejidad de las siguientes funciones:

1) $f(n) = 13n^3 - 8n + 12 \longrightarrow f(n) \in O(n^3)$

2) $f(n) = 5n^2 + n\ln(n) + 3 \longrightarrow f(n) \in O(n^2)$

3) $f(n) = 13n! - 8\log_2(n) + 12 \longrightarrow f(n) \in O(n!)$

EJERCICIO: Si los algoritmos A1, A2 y A3 tuvieran, respectivamente, los órdenes de convergencia anteriores, ¿cuál sería su ordenación de mejor a peor?

EJERCICIO: Determina el orden de complejidad de las siguientes funciones:

1) $f(n) = 13n^3 - 8n + 12 \longrightarrow f(n) \in O(n^3)$

2) $f(n) = 5n^2 + n\ln(n) + 3 \longrightarrow f(n) \in O(n^2)$

3) $f(n) = 13n! - 8\log_2(n) + 12 \longrightarrow f(n) \in O(n!)$

EJERCICIO: Si los algoritmos A1, A2 y A3 tuvieran, respectivamente, los órdenes de convergencia anteriores, ¿cuál sería su ordenación de mejor a peor?

A2 mejor que A1 mejor que A3, ya que:

n^2 crece más lentamente que n^3 y, este, a su vez, que $n!$

EJERCICIO: Determina si son ciertas las siguientes afirmaciones

1) $f(n) = 8n^2 - 32n + 1 \in O(n!)$

2) $f(n) = 5n^2 + n\ln(n) + 3 \in O(n\ln(n))$

3) Si $g(n)$ crece más rápido que $f(n)$, entonces $g(n) \in O(f(n))$

EJERCICIO: Determina si son ciertas las siguientes afirmaciones

1) $f(n) = 8n^2 - 32n + 1 \in O(n!)$ → Sí. Porque n^2 crece más lento que $n!$

2) $f(n) = 5n^2 + n\ln(n) + 3 \in O(n\ln(n))$

3) Si $g(n)$ crece más rápido que $f(n)$, entonces $g(n) \in O(f(n))$

EJERCICIO: Determina si son ciertas las siguientes afirmaciones

1) $f(n) = 8n^2 - 32n + 1 \in O(n!)$ → Sí. Porque n^2 crece más lento que $n!$

2) $f(n) = 5n^2 + n\ln(n) + 3 \in O(n\ln(n))$ → No. Porque n^2 crece más rápido que $n\ln(n)$

3) Si $g(n)$ crece más rápido que $f(n)$, entonces $g(n) \in O(f(n))$

EJERCICIO: Determina si son ciertas las siguientes afirmaciones

1) $f(n) = 8n^2 - 32n + 1 \in O(n!)$ → Sí. Porque n^2 crece más lento que $n!$

2) $f(n) = 5n^2 + n\ln(n) + 3 \in O(n\ln(n))$ → No. Porque n^2 crece más rápido que $n\ln(n)$

3) Si $g(n)$ crece más rápido que $f(n)$, entonces $g(n) \in O(f(n))$

↓

No. Sería justo al revés: $f(n) \in O(g(n))$

EJERCICIO: Si una computadora es capaz de realizar 2 millones de operaciones elementales por segundo, calcula el orden de los tiempos necesarios para resolver un problema de tamaño $n = 10000$ con algoritmos de las siguientes complejidades:

1) $f(n) \in O(n^3)$

EJERCICIO: Si una computadora es capaz de realizar 2 millones de operaciones elementales por segundo, calcula el orden de los tiempos necesarios para resolver un problema de tamaño $n = 10000$ con algoritmos de las siguientes complejidades:

1) $f(n) \in O(n^3)$ ⟶ nº de operaciones del orden de $(10^4)^3 = 10^{12}$

EJERCICIO: Si una computadora es capaz de realizar 2 millones de operaciones elementales por segundo, calcula el orden de los tiempos necesarios para resolver un problema de tamaño $n = 10000$ con algoritmos de las siguientes complejidades:

1) $f(n) \in O(n^3)$ → nº de operaciones del orden de $(10^4)^3 = 10^{12}$

$$\text{Orden de tiempo} = \frac{10^{12}}{2 \times 10^6} = 5 \times 10^5$$

EJERCICIO: Si una computadora es capaz de realizar 2 millones de operaciones elementales por segundo, calcula el orden de los tiempos necesarios para resolver un problema de tamaño $n = 10000$ con algoritmos de las siguientes complejidades:

2) $f(n) \in O(n \ln n)$

EJERCICIO: Si una computadora es capaz de realizar 2 millones de operaciones elementales por segundo, calcula el orden de los tiempos necesarios para resolver un problema de tamaño $n = 10000$ con algoritmos de las siguientes complejidades:

2) $f(n) \in O(n \ln n)$ ⟶

nº de operaciones del orden de $10^4 \log(10^4) = 4 \times 10^4$

EJERCICIO: Si una computadora es capaz de realizar 2 millones de operaciones elementales por segundo, calcula el orden de los tiempos necesarios para resolver un problema de tamaño $n = 10000$ con algoritmos de las siguientes complejidades:

2) $f(n) \in O(n \ln n)$ ⟶

nº de operaciones del orden de $10^4 \log(10^4) = 4 \times 10^4$

$$\text{Orden de tiempo} = \frac{4 \times 10^4}{2 \times 10^6} = 2 \times 10^{-2}$$

CRITERIO DE TERMINACIÓN:

Los siguientes criterios son frecuentemente utilizados para detener la ejecución de un algoritmo. Sean $\varepsilon > 0$, N un entero positivo y α una función de interés para el problema, todos ellos prefijados antes de la ejecución del algoritmo:

1. $|x_{k+N} - x_k| < \varepsilon$
2. $\dfrac{|x_{k+1} - x_k|}{|x_k|} < \varepsilon$
3. $|\alpha(x_k) - \alpha(x_{k+N})| < \varepsilon$
4. $\dfrac{|\alpha(x_k) - \alpha(x_{k+1})|}{|\alpha(x_k)|} < \varepsilon$
5. $|\alpha(x_k) - \alpha(x^*)| < \varepsilon, x^* \in S$

Un ejemplo típico de función de interés en problemas de optimización es el gradiente.

Criterios de terminación

x_0

x_1

x_2

x_3

$\vdots$

x_n

x_{n+1}

$\vdots$

z

$¿|x_3 - x_0| < 10^{-6}?$

$k = 3, \quad \varepsilon = 10^{-6}$

$|x_{k+N} - x_k| < \varepsilon$

Criterios de terminación

EJEMPLO 3: Dada una función continua $f : [a, b] \to [a, b]$ un problema de interés consiste en encontrar un punto fijo de la función, es decir, una solución de la ecuación $x = f(x)$. Un algoritmo sencillo para ello es el siguiente:

$$x_{n+1} = f(x_n)$$

Nota: Este algoritmo no siempre converge a la solución, pero bajo determinadas condiciones sí puede garantizarse su convergencia.

Resuelve la ecuación $x = e^{-x}$ con $x \in [0, 1]$ de forma aproximada utilizando como criterios de parada o terminación 1 y 2 con $\varepsilon = 0.004$ y 0.0004 y $N = 3$.

Criterios de terminación

Criterio 1: $|x_{k+3} - x_k| < 0,004\ (0,0004)$

Criterio 2: $\dfrac{|x_{k+1} - x_k|}{|x_k|} < 0,004\ (0,0004)$

n	x_n	*Criterio 1*	*Criterio 2*	n	x_n	*Criterio 1*	*Criterio 2*
0	0,5						
1	0,6065307		0,2130613	11	0,5672772	0,0008677	0,0006526
2	0,5452392		0,1010525	12	0,5670674	0,0004923	0,0003699
3	0,5797031	0,0797031	0,0632087	13	0,5671864	0,0002791	0,0002099
4	0,5600646	0,046466	0,0338768	14	0,5671189	0,0001583	0,000119
5	0,5711721	0,0259329	0,0198326	15	0,5671571	8,979E-05	6,75E-05
6	0,5648629	0,0148401	0,0110461	16	0,5671354	5,093E-05	3,828E-05
7	0,568438	0,0083734	0,0063291	17	0,5671477	2,888E-05	2,171E-05
8	0,5664095	0,0047627	0,0035687	18	0,5671408	1,638E-05	1,231E-05
9	0,5675596	0,0026967	0,0020307	19	0,5671447	9,29E-06	6,983E-06
10	0,5669072	0,0015308	0,0011495	20	0,5671425	5,269E-06	3,96E-06

CRITERIOS PARA LA ELECCIÓN O DISEÑO DE UN ALGORITMO:

1. **Generalidad**: variedad de problemas que se pueden resolver con el algoritmo y condiciones necesarias para su convergencia.

2. **Fiabilidad o robustez**: solución efectiva de los problemas para los que se diseñó. El tamaño y la estructura del problema son elementos importantes a tener en cuenta en este criterio.

3. **Precisión**: calidad de las soluciones obtenidas por el algoritmo después de un número de iteraciones.

4. **Sensibilidad a los parámetros**: dependencia de los valores iniciales de los parámetros necesarios para el inicio del algoritmo.

CRITERIOS PARA LA ELECCIÓN O DISEÑO DE UN ALGORITMO:

5. **Sensibilidad a los datos**: dependencia de los datos del problema.

6. **Esfuerzo preparatorio**: tiempo y recursos necesarios para preparar el algoritmo.

7. **Esfuerzo computacional**: tiempo y recursos necesarios para ejecutar el algoritmo.

8. **Orden de convergencia**: rapidez con la que el algoritmo converge a una solución del problema.

Un último ejemplo

EJEMPLO 4: El algoritmo del punto medio para la búsqueda de las raíces de un polinomio $p(x)$ en un intervalo $[a, b]$ donde se sabe que hay un cambio de signo.

Paso 0: Hacer $a_1 = a$ y $b_1 = b$.

Paso 1: Evaluar $p(a_1)$ y $p(b_1)$.

- Si $p(a_1)\times p(b_1) > 0$ entonces PARAR $\rightarrow$ Elegir otro intervalo $[a, b]$.

- Si $p(a_1)\times p(b_1) = 0$ entonces PARAR $\rightarrow$ a y/o b son raíces de $p(x)$ en $[a, b]$.

- Si $p(a_1)\times p(b_1) < 0$ y $b_1 - a_1 < \varepsilon$ entonces PARAR $\rightarrow$ $(a_1 + b_1)/2$ es una aproximación a una raíz de $p(x)$ en $[a, b]$.

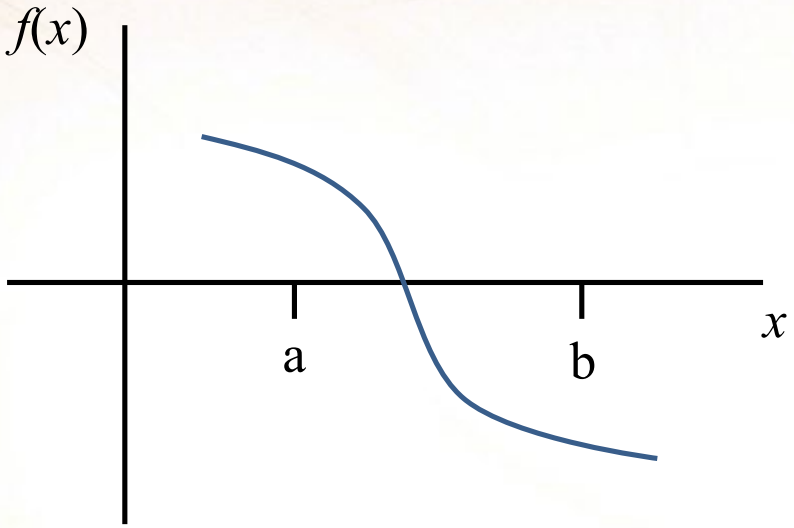
- En otro caso, hacer $m_1 = (a_1 + b_1)/2$ e ir al Paso 2.

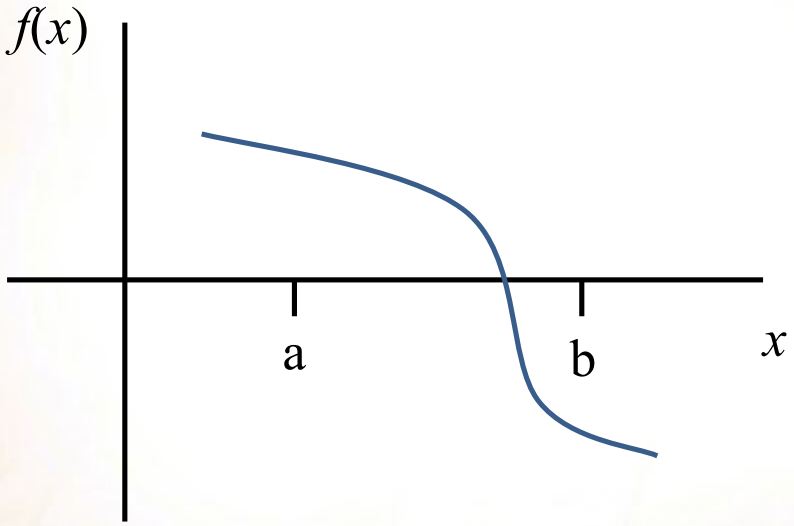
Paso 2: Evaluar $p(m_1)$.

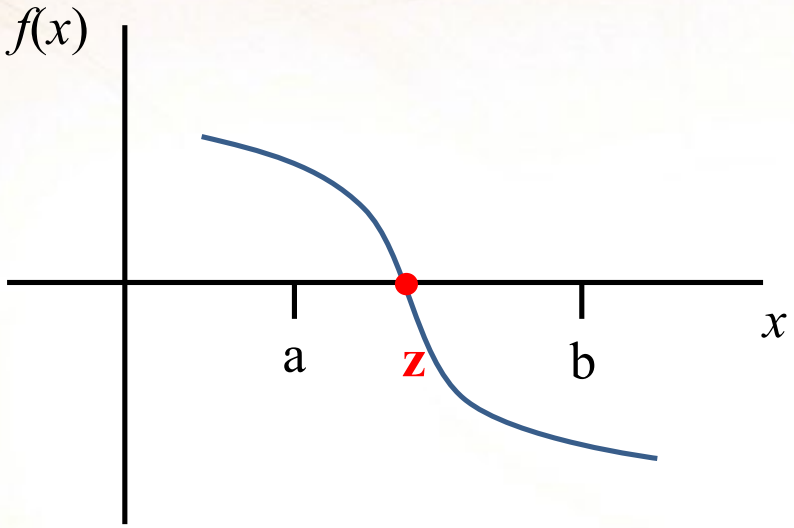
- Si $p(a_1)\times p(m_1) < 0$, entonces HACER $a_1 := a_1$ y $b_1 := m_1$ e IR al Paso 1.

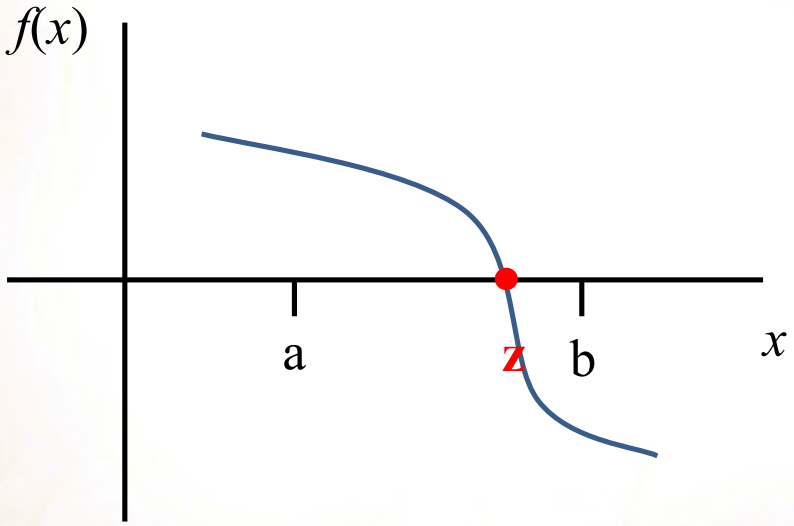
- Si $p(a_1) \times p(m_1) > 0$, entonces HACER $a_1 := m_1$ y $b_1 := b_1$ e IR al Paso 1.

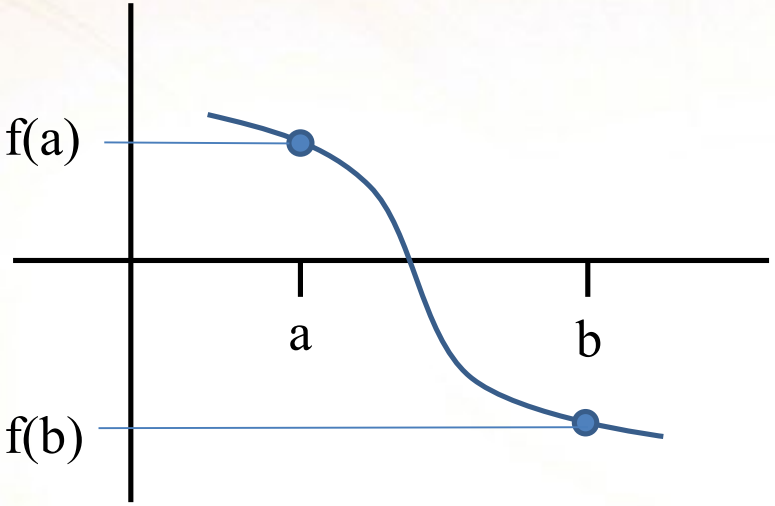
- Si $p(m_1) = 0$, entonces PARAR $\rightarrow$ m_1 es una raíz de $p(x)$ en $[a, b]$.

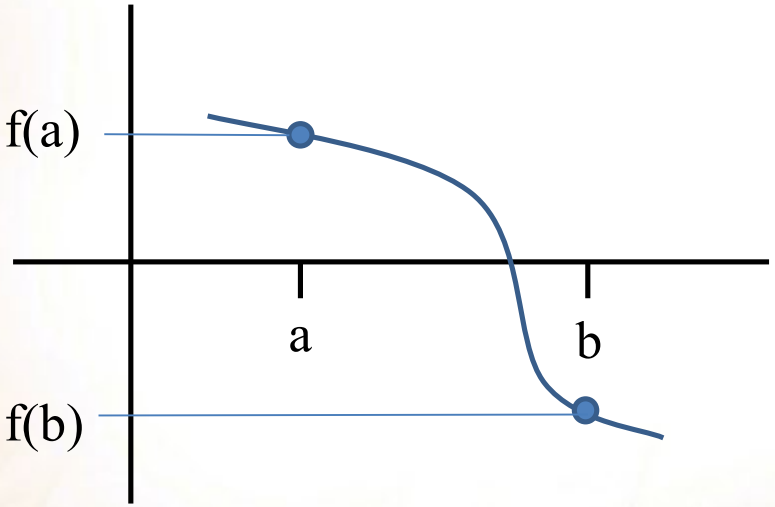

f(x)
x
a
b


f(x)
x
a
b


f(x)
x
a
z
b

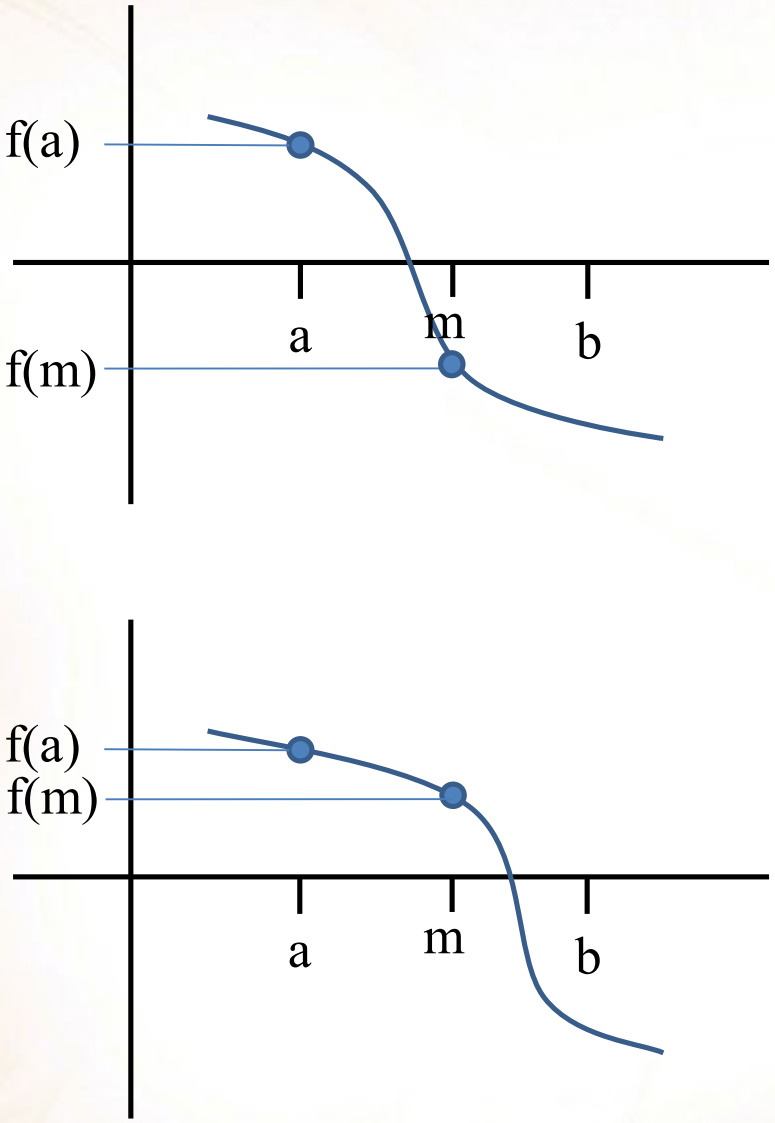

f(x)
x
a
z
b



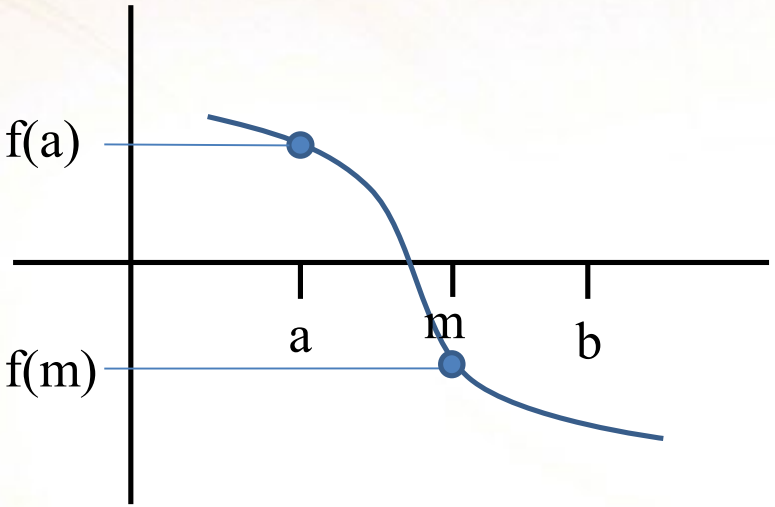


Si f(x) es continua, entonces hay una raíz en el intervalo [a, b] si $f(a)f(b) < 0$

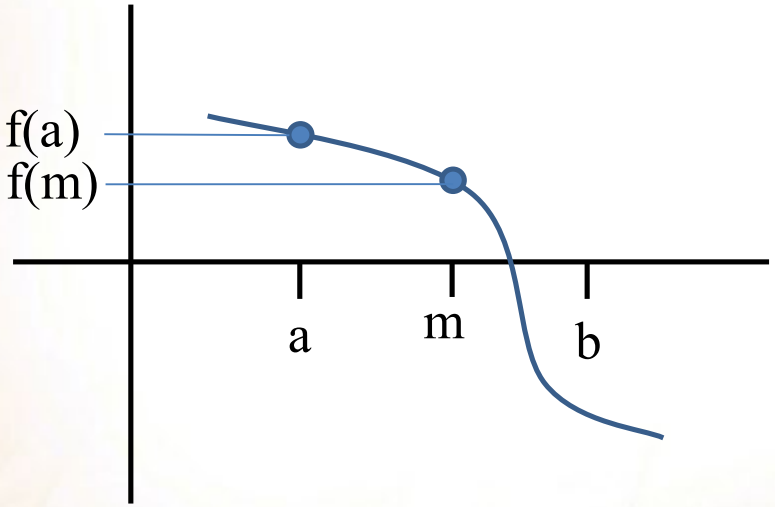
Calculamos m=punto medio entre a y b



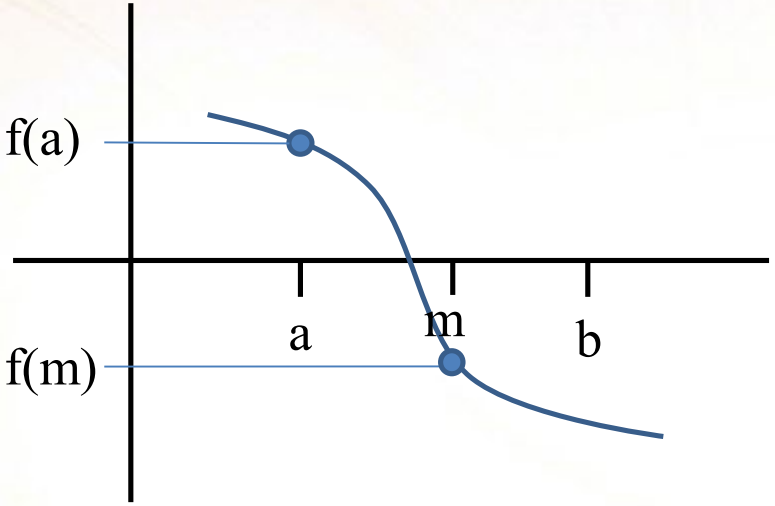
Calculamos m=punto medio entre a y b



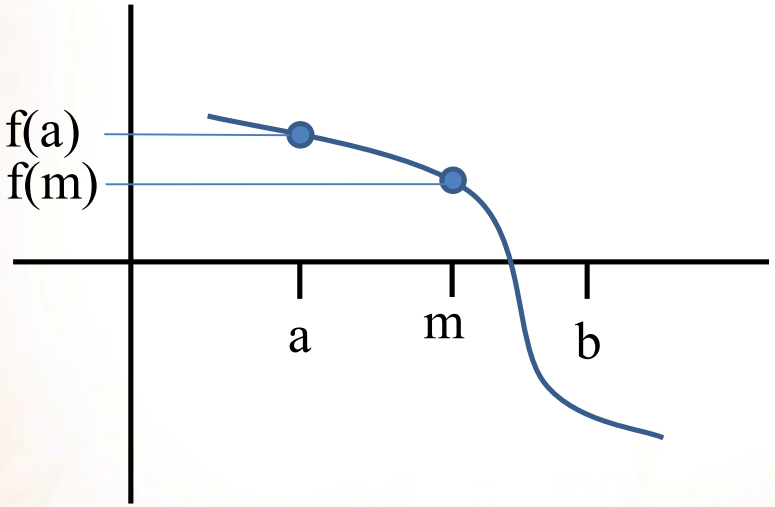
Si f(a)f(m) < 0, entonces la raíz está en la primera mitad del intervalo [a, b]



Calculamos m=punto medio entre a y b

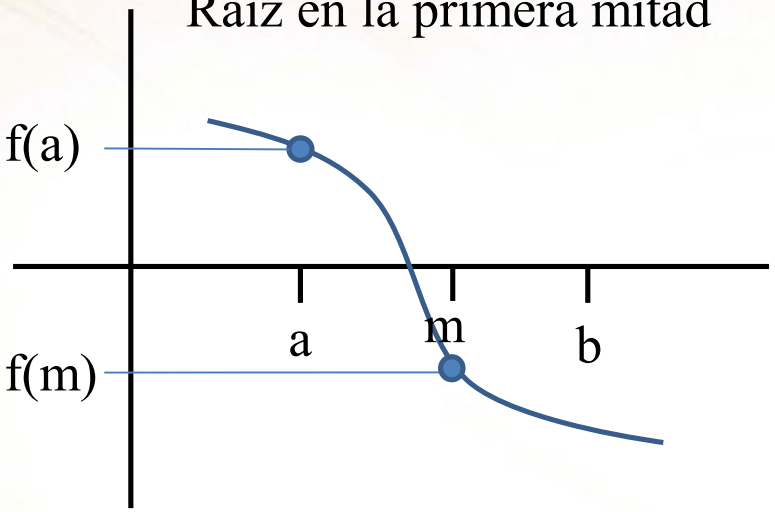


Si f(a)f(m) < 0, entonces la raíz está en la primera mitad del intervalo [a, b]

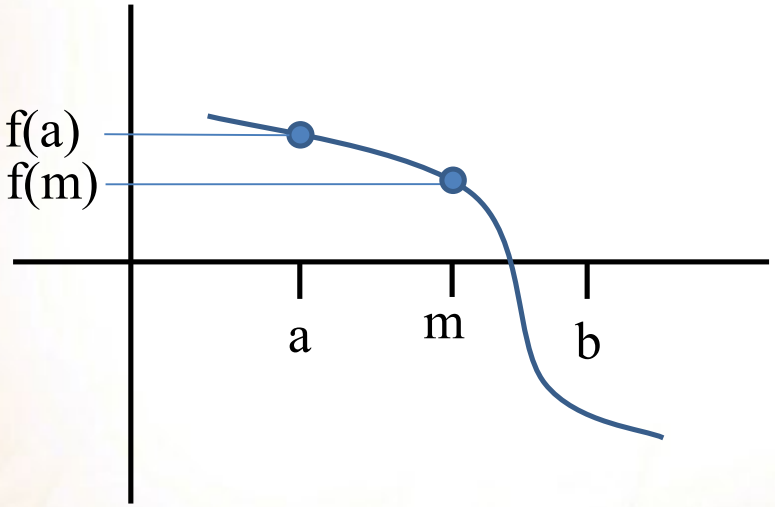


Si f(a)f(m) > 0, entonces la raíz está en la segunda mitad del intervalo [a, b]

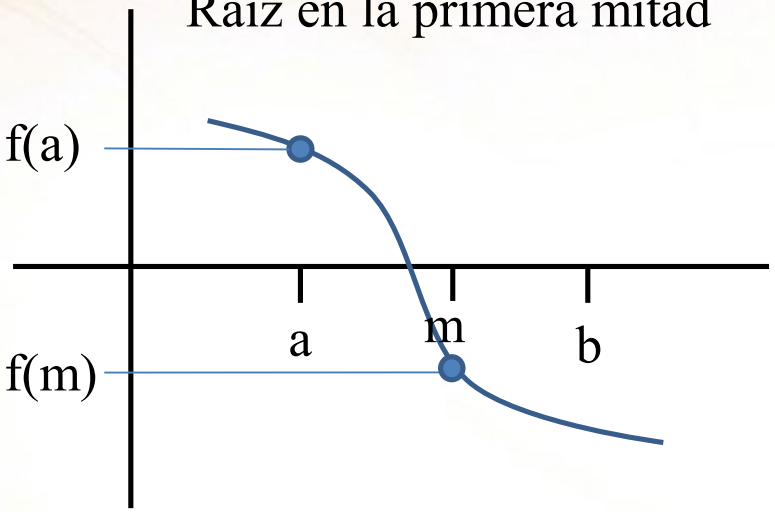
Raíz en la primera mitad

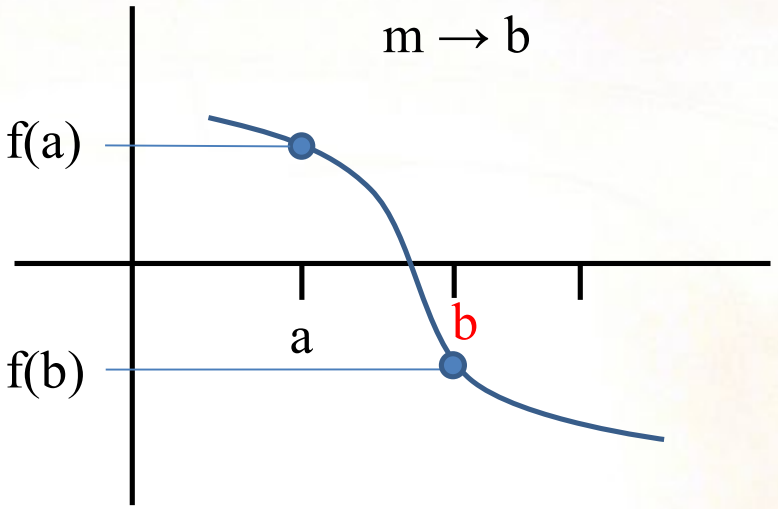


Raíz en la segunda mitad

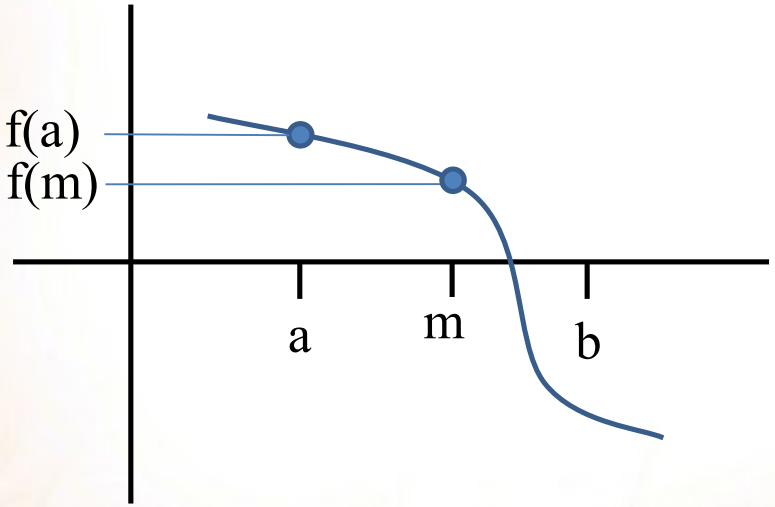


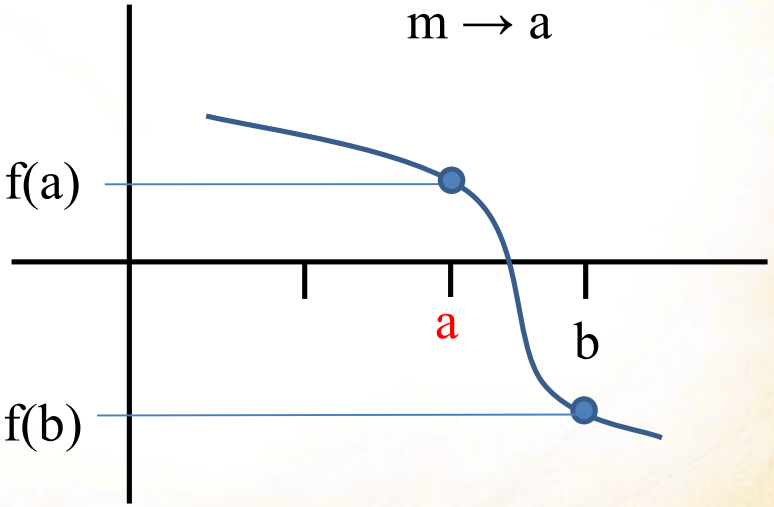
Raíz en la primera mitad

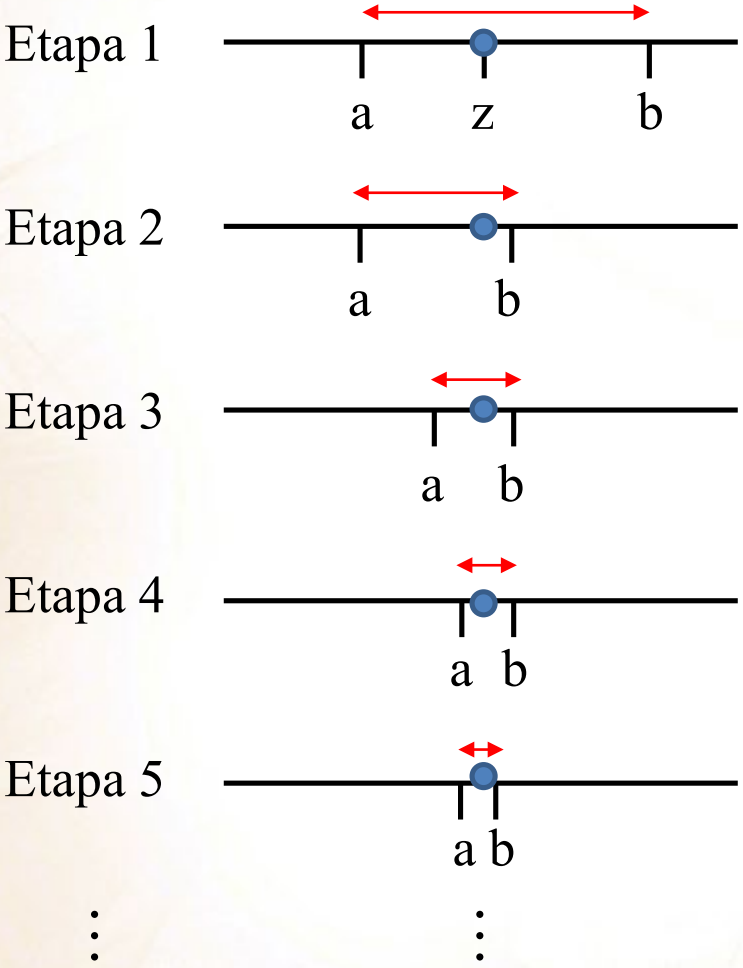


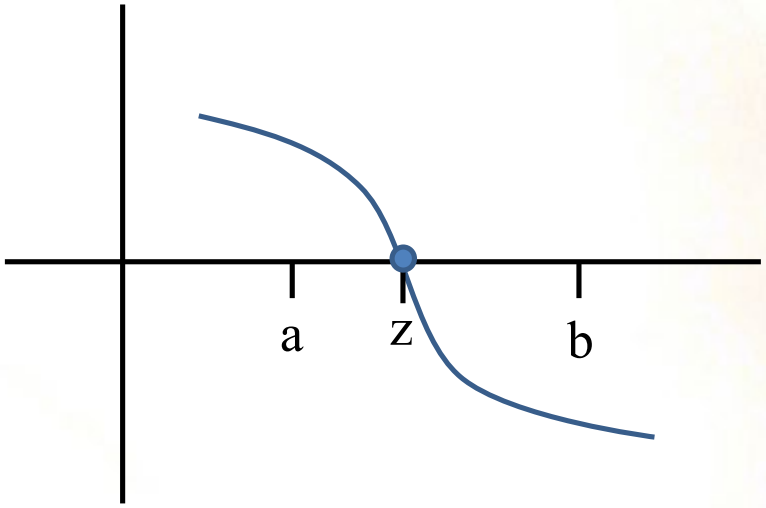


Raíz en la segunda mitad






Etapa 1
a
z
b
Etapa 2
a
b
Etapa 3
a
b
Etapa 4
a
b
Etapa 5
a b


a
z
b

Un último ejemplo

Encontrar una raíz del polinomio $p(x) = x^3 + 2x^2 - 13x - 6$ en el intervalo $[1, 7]$ aplicando el algoritmo del punto medio con $\varepsilon = 0.0001$.

Iteración	a1	b1	p(a1)	p(b1)	p(a1)p(b1)	ε = b1-a1	m1	p(m1)	p(a1)xp(m1)
1	1	7	-16	344	-5504	6	4	38	-608
2	1	4	-16	38	-608	3	2,5	-10,375	166
3	2,5	4	-10,375	38	-394,25	1,5	3,25	7,2031	-74,7324
4	2,5	3,25	-10,375	7,2031	-74,7324	0,75	2,875	-3,0801	31,9558
5	2,875	3,25	-3,0801	7,2031	-22,1862	0,375	3,0625	1,6682	-5,1382
6	2,875	3,0625	-3,0801	1,6682	-5,1382	0,1875	2,96875	-0,8018	2,4696
7	2,96875	3,0625	-0,8018	1,6682	-1,3376	0,09375	3,015625	0,4089	-0,3279
8	2,96875	3,015625	-0,8018	0,4089	-0,3279	0,04688	2,9921875	-0,2025	0,1623
9	2,9921875	3,015625	-0,2025	0,4089	-0,0828	0,02344	3,0039063	0,1017	-0,0206
10	2,9921875	3,0039063	-0,2025	0,1017	-0,0206	0,01172	2,9980469	-0,0507	0,0103
11	2,9980469	3,0039063	-0,0507	0,1017	-0,0052	0,00586	3,0009766	0,0254	-0,0013
12	2,9980469	3,0009766	-0,0507	0,0254	-0,0013	0,00293	2,9995117	-0,0127	0,0006
13	2,9995117	3,0009766	-0,0127	0,0254	-0,0003	0,00146	3,0002441	0,0063	-8,06E-05
14	2,9995117	3,0002441	-0,0127	0,0063	-8,06E-05	0,00073	2,9998779	-0,0032	4,03E-05
15	2,9998779	3,0002441	-0,0032	0,0063	-2,01E-05	0,00037	3,0000610	0,0016	-5,04E-06
16	2,9998779	3,0000610	-0,0032	0,0016	-5,04E-06	0,00018	2,9999695	-0,0008	2,52E-06
17	2,9999695	3,0000610	-0,0008	0,0016	-1,26E-06	0,00009	3,0000153	0,0004	

Se da **x = 3.0000153** como solución, con $p(3.0000153) = 0.0003967$
(La raíz exacta es 3)