Computació Numèrica

Tema 1. Conceptos básicos (II): Representación de números y estabilidad.

Irene Parada

irene.parada@upc.edu

Departamento de Matemáticas

Universitat Politècnica de Catalunya · BarcelonaTech

19 de febrero de 2024

Repaso

Breve recordatorio del Tema 1.1

- Función error $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ y entorno a 0:
  $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{n!(2n+1)}$
- Error absoluto y relativo.
- Cifras correctas y significativas.
- Precisión vs. exactitud.
- Fuentes de error: de modelización, experimental, de redondeo, y de truncamiento / discretización.
- Propagación del error al usar funciones de una o más variables.

Error absoluto propagado: $|\Delta f(x)| = |f(x) - f(\tilde{x})| \approx \sum_{i=1}^{n} \left| \frac{\partial f(\tilde{x})}{\partial x_i} \right| |\Delta x_i|$.

Error relativo propagado: $\left| \frac{\Delta f(x)}{f(\tilde{x})} \right| \approx \sum_{i=1}^{n} \underbrace{\left| \frac{\tilde{x}_i}{f(\tilde{x})} \frac{\partial f(\tilde{x})}{\partial x_i} \right|}_{\text{números de condición}} \left| \frac{\Delta x_i}{\tilde{x}_i} \right|$.

Propagación del error: ejemplos

Los ordenadores a menudo realizan cálculos de manera aproximada. En las sucesivas operaciones los errores se propagan. Además, el orden de las operaciones afecta al resultado.

Propagación del error: ejemplos

Los ordenadores a menudo realizan cálculos de manera aproximada. En las sucesivas operaciones los errores se propagan. Además, el orden de las operaciones afecta al resultado.

Ejemplo 1. Queremos calcular $a+b+c$, donde $a = 5.317 \times 10^{-3}$, $b = 3.387 \times 10^{1}$ y $c = -3.381 \times 10^{1}$, con 4 cifras significativas. Comparar $(a+b)+c$ con $a+(b+c)$ y con la solución exacta.

Propagación del error: ejemplos

Los ordenadores a menudo realizan cálculos de manera aproximada. En las sucesivas operaciones los errores se propagan. Además, el orden de las operaciones afecta al resultado.

Ejemplo 1. Queremos calcular $a + b + c$, donde $a = 5.317 \times 10^{-3}$, $b = 3.387 \times 10^{1}$ y $c = -3.381 \times 10^{1}$, con 4 cifras significativas. Comparar $(a + b) + c$ con $a + (b + c)$ y con la solución exacta.

$7.000 \cdot 10^{-}2$ $\qquad$ $6.532 \cdot 10^{-}2$

$6.5317 \cdot 10^{-}2$

Propagación del error: ejemplos

Los ordenadores a menudo realizan cálculos de manera aproximada. En las sucesivas operaciones los errores se propagan. Además, el orden de las operaciones afecta al resultado.

Ejemplo 1. Queremos calcular $a + b + c$, donde $a = 5.317 \times 10^{-3}$, $b = 3.387 \times 10^{1}$ y $c = -3.381 \times 10^{1}$, con 4 cifras significativas. Comparar $(a + b) + c$ con $a + (b + c)$ y con la solución exacta.

Ejemplo 2. Dada la relación de recurrencia $x_{n+1} = x_n + 3x_n(1 - x_n)$, con $x_0 = 0.01$ y $n \geq 0$, comparar las 50 primeras iteraciones hechas con 10 y con 12 cifras significativas.

Propagación del error: ejemplos

Los ordenadores a menudo realizan cálculos de manera aproximada. En las sucesivas operaciones los errores se propagan. Además, el orden de las operaciones afecta al resultado.

Ejemplo 1. Queremos calcular $a+b+c$, donde $a = 5.317 \times 10^{-3}$, $b = 3.387 \times 10^{1}$ y $c = -3.381 \times 10^{1}$, con 4 cifras significativas. Comparar $(a+b)+c$ con $a+(b+c)$ y con la solución exacta.

Ejemplo 2. Dada la relación de recurrencia $x_{n+1} = x_n + 3x_n(1-x_n)$, con $x_0 = 0.01$ y $n \geq 0$, comparar las 50 primeras iteraciones hechas con 10 y con 12 cifras significativas.

n	10 cifras sig.	12 cifras sig.
1	0.0397	0.0397
2	0.15407173	0.15407173
3	0.5450726260	0.545072626044
4	1.288978001	1.28897800119
5	0.1715191421	0.171519142100
10	0.7229143012	0.722914301711
15	1.270261775	1.27026178116
20	0.5965292447	0.596528770927
25	1.315587846	1.31558435183
30	0.3742092321	0.374647695060
35	0.9233215064	0.908845072341
40	0.0021143643	0.143971503996
45	1.219763115	1.23060086551
50	0.0036616295	0.225758993390

Propagación del error: ejemplos

Los ordenadores a menudo realizan cálculos de manera aproximada. En las sucesivas operaciones los errores se propagan. Además, el orden de las operaciones afecta al resultado.

Ejemplo 1. Queremos calcular $a+b+c$, donde $a = 5.317\times 10^{-3}$, $b = 3.387\times 10^{1}$ y $c = -3.381\times 10^{1}$, con 4 cifras significativas. Comparar $(a+b)+c$ con $a+(b+c)$ y con la solución exacta.

Ejemplo 2. Dada la relación de recurrencia $x_{n+1} = x_n + 3x_n(1-x_n)$, con $x_0 = 0.01$ y $n \geq 0$, comparar las 50 primeras iteraciones hechas con 10 y con 12 cifras significativas.

n	$x_n + 3x_n(1-x_n)$	$4x_n - 3x_n^2$
1	0.0397	0.0397
2	0.15407173	0.15407173
3	0.5450726260	0.5450726260
4	1.288978001	1.288978001
5	0.1715191421	0.1715191421
10	0.7229143012	0.7229143012
15	1.270261775	1.270261774
20	0.5965292447	0.5965293261
25	1.315587846	1.315588447
30	0.3742092321	0.3741338572
35	0.9233215064	0.9257966719
40	0.0021143643	0.0144387553
45	1.219763115	0.0497855318
50	0.0036616295	0.1917063297

Aritmética de punto/coma flotante

What Every Computer Scientist Should Know About Floating-Point Arithmetic

Representación de números en el ordenador

Existen diversas técnicas para representar los números reales en el ordenador.

Un vistazo: tanto a los conceptos generales como a la representación de números en MATLAB.

A Glimpse into Floating-Point Accuracy

Floating points. IEEE Standard unifies arithmetic model

Aritmética de punto/coma flotante

Representación de números

Representación de números (punto fijo)

$$x = \pm d_1 d_2 d_3 \ldots d_n \bullet d_{n+1} d_{n+2} \ldots d_{n+m}$$

Representa el número real x expresado en base 10, donde cada $d_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Ejemplos.

$$2345 = (2 \times 10^3) + (3 \times 10^2) + (4 \times 10^1) + (5 \times 10^0) = 2000 + 300 + 40 + 5$$

10^4	10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
	2	3	4	5 $\bullet$					

Representación de números (punto fijo)

$$x = \pm d_1 d_2 d_3 \ldots d_n \bullet d_{n+1} d_{n+2} \ldots d_{n+m}$$

Representa el número real x expresado en base 10, donde cada $d_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Ejemplos.

$$2345 = (2 \times 10^3) + (3 \times 10^2) + (4 \times 10^1) + (5 \times 10^0) = 2000 + 300 + 40 + 5$$

$$45.67 = (4 \times 10^1) + (5 \times 10^0) + (6 \times 10^{-1}) + (7 \times 10^{-2}) = 40 + 5 + 0.6 + 0.07$$

10^4	10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
			4	5 •	6	7			

Recordatorio: notación científica

La notación científica expresa un valor como un número entre 1 y 10 (mantisa), multiplicado por una potencia de 10 (orden de magnitud). Esta notación es muy usada cuando queremos expresar números muy grandes o muy pequeños.

En MATLAB, los valores en notación científica se designan con una "e" entre el número decimal y el exponente, reemplazando la expresión $\times 10$:
Por ejemplo, el número de Avogadro es 6.022×10^{23}, y el diámetro de un átomo de hierro es 1.4×10^{-10} metros. En MATLAB:

```
% Número de Avogadro
6.022e23
6.022*10^23
% Diámetro de un átomo
1.4e-10
1.4*10^(-10)
```

Representación de números (punto/coma flotante)

$$x = \pm 0.d_1 d_2 d_3 \ldots d_n \times 10^k \quad d_1 \neq 0$$

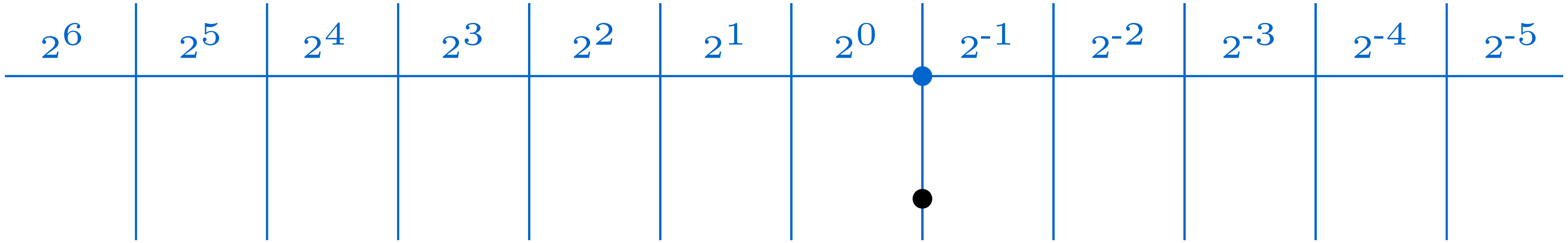
Representa el número real x expresado en base 10 y precisión n en notación de punto/coma flotante normalizada*, donde el exponente k es un número entero y cada dígito d_i es un entero entre 0 y 9.

Ejemplos.
El número -15.24 en coma flotante es -0.1524×10^2;
el número 0.000617 en coma flotante es 0.617×10^{-3};
el número 4.274952×10^{15} es 0.4274952×10^{16};
y el número -6543219 en coma flotante es -0.6543219×10^7.

* A veces el término normalización se usa para la notación en la que el dígito antes de la coma no es cero.

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.



Ejemplos.

$101.1101_{(2)}$

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.

2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
				1	0	1	1	1	0	1	

Ejemplos.

$101.1101_{(2)}$

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.

2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
				1	0	1	1	1	0	1	

Ejemplos.

$$
\begin{aligned}
101.1101_{(2)} &= (1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4})_{(10)} \\
&= (4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32})_{(10)} = (5.78125)_{(10)}
\end{aligned}
$$

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.

2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}

Ejemplos.

$$
\begin{aligned}
101.1101_{(2)} &= (1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4})_{(10)}\\
&= (4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32})_{(10)} = (5.78125)_{(10)}\\
97_{(10)} &=
\end{aligned}
$$

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.

$$\begin{aligned}
97 \div 2 &= 48 \text{ resto } 1\\
48 \div 2 &= 24 \text{ resto } 0\\
24 \div 2 &= 12 \text{ resto } 0\\
12 \div 2 &= 6 \text{ resto } 0\\
6 \div 2 &= 3 \text{ resto } 0\\
3 \div 2 &= 1 \text{ resto } 1\\
1 \div 2 &= 0 \text{ resto } 1
\end{aligned}$$

2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}

Ejemplos.

$$\begin{aligned}
101.1101_{(2)} &= (1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4})_{(10)}\\
&= (4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32})_{(10)} = (5.78125)_{(10)}\\
97_{(10)} &=
\end{aligned}$$

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.

$97 \div 2 = 48$ resto 1

$48 \div 2 = 24$ resto 0

$24 \div 2 = 12$ resto 0

$12 \div 2 = 6$ resto 0

$6 \div 2 = 3$ resto 0

$3 \div 2 = 1$ resto 1

$1 \div 2 = 0$ resto 1

2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
1	1	0	0	0	0	1					

Ejemplos.

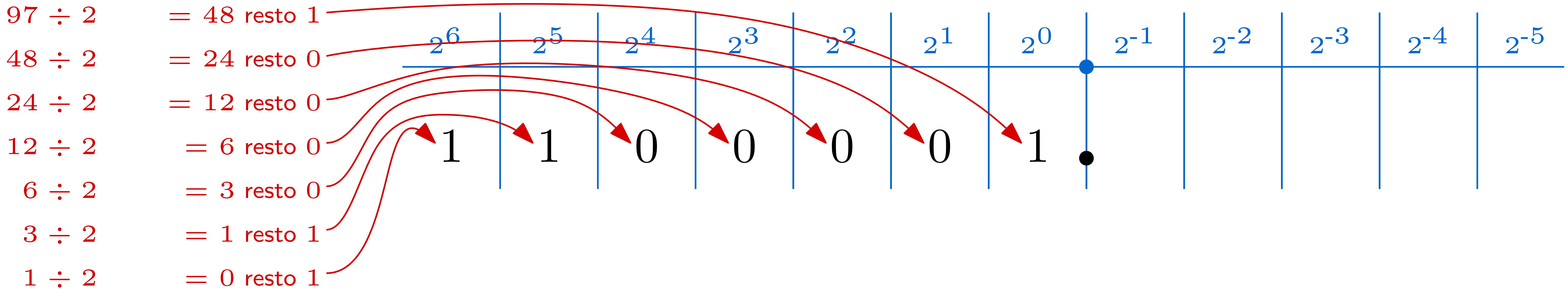
$$101.1101_{(2)} = (1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4})_{(10)}$$

$$= (4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32})_{(10)} = (5.78125)_{(10)}$$

$$97_{(10)} =$$

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.



Ejemplos.

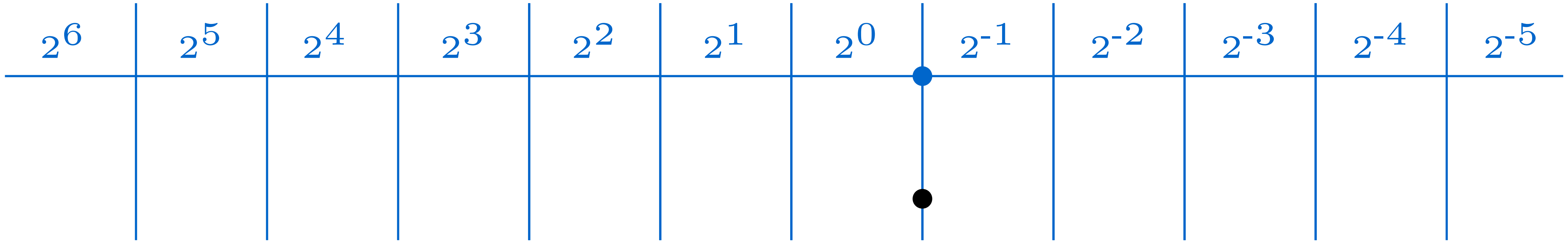
$$101.1101_{(2)} = (1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4})_{(10)}$$

$$= (4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32})_{(10)} = (5.78125)_{(10)}$$

$97_{(10)} = (1100001)_{(2)}$, usando restos al dividir por 2

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.



Ejemplos.

$$101.1101_{(2)} = (1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4})_{(10)}$$

$$= (4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32})_{(10)} = (5.78125)_{(10)}$$

$$97_{(10)} = (1100001)_{(2)}, \text{ usando restos al dividir por } 2$$

$$0.7_{(10)} =$$

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.

$0.7 \times 2 \quad = 1.4$ entero 1

$0.4 \times 2 \quad = 0.8$ entero 0

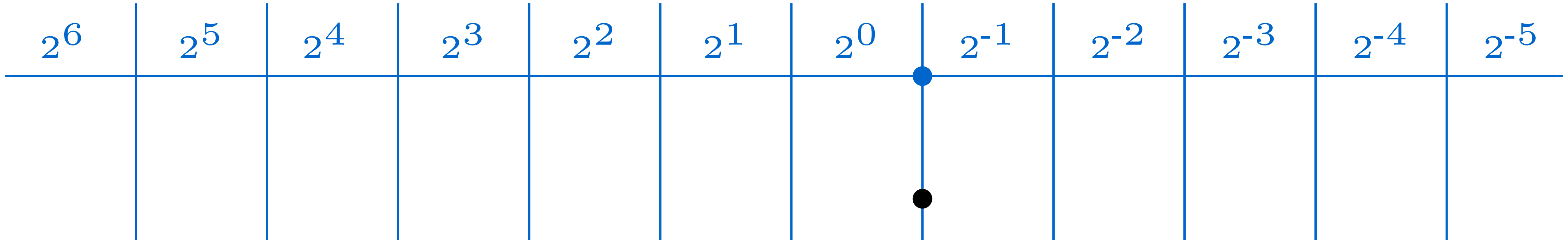
$0.8 \times 2 \quad = 1.6$ entero 1

$0.6 \times 2 \quad = 1.2$ entero 1

$0.2 \times 2 \quad = 0.4$ entero 0

$0.4 \times 2 \quad = 0.8$ entero 0

$\ldots$



Ejemplos.

$$101.1101_{(2)} = (1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4})_{(10)}$$

$$= (4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32})_{(10)} = (5.78125)_{(10)}$$

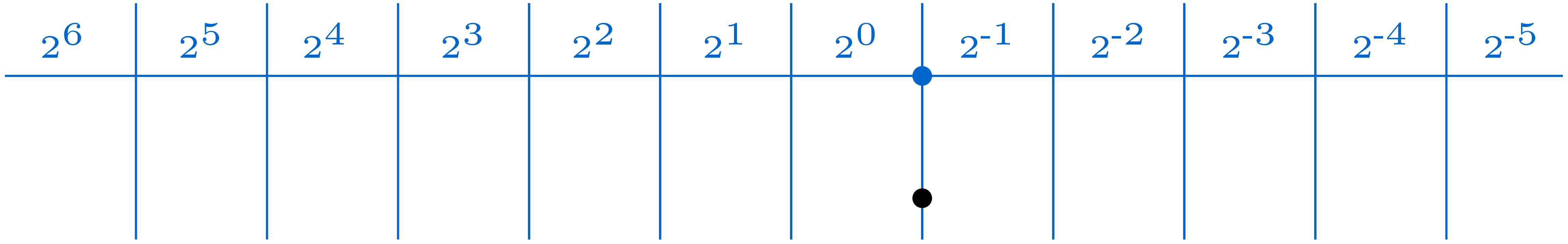
$$97_{(10)} = (1100001)_{(2)}, \text{ usando restos al dividir por } 2$$

$$0.7_{(10)} =$$

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.

$$
\begin{aligned}
0.7 \times 2 &= 1.4 \text{ entero } 1\\
0.4 \times 2 &= 0.8 \text{ entero } 0\\
0.8 \times 2 &= 1.6 \text{ entero } 1\\
0.6 \times 2 &= 1.2 \text{ entero } 1\\
0.2 \times 2 &= 0.4 \text{ entero } 0\\
0.4 \times 2 &= 0.8 \text{ entero } 0\\
&\ \ldots
\end{aligned}
$$

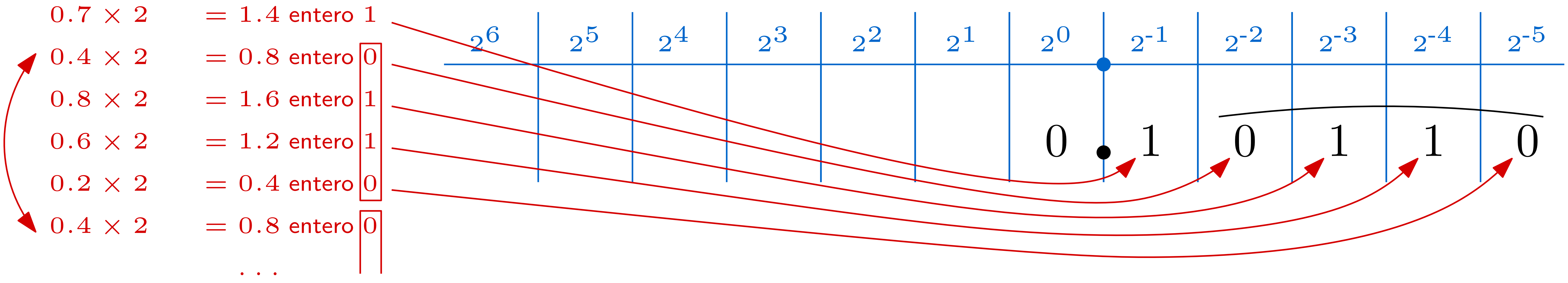


Ejemplos.

$$
\begin{aligned}
101.1101_{(2)} &= (1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4})_{(10)}\\
&= (4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32})_{(10)} = (5.78125)_{(10)}\\
97_{(10)} &= (1100001)_{(2)}, \text{usando restos al dividir por } 2\\
0.7_{(10)} &=
\end{aligned}
$$

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.



Ejemplos.

$$101.1101_{(2)} = (1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4})_{(10)}$$

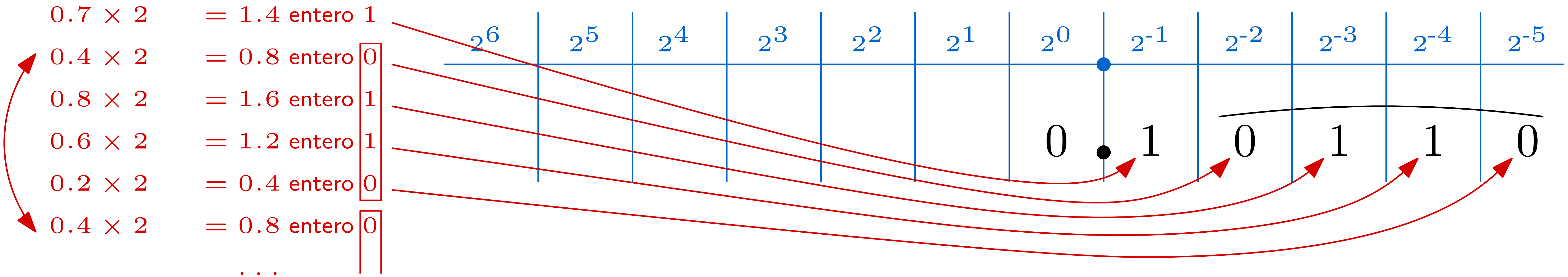
$$= (4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32})_{(10)} = (5.78125)_{(10)}$$

$$97_{(10)} = (1100001)_{(2)},$$ usando restos al dividir por 2

$$0.7_{(10)} =$$

Números binarios

El sistema binario es similar al decimal, pero solo utiliza los dígitos 0 y 1.



Ejemplos.

$$101.1101_{(2)} = (1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4})_{(10)}$$

$$= (4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32})_{(10)} = (5.78125)_{(10)}$$

$$97_{(10)} = (1100001)_{(2)},\text{ usando restos al dividir por } 2$$

$$0.7_{(10)} = (0.1\overline{0110})_{(2)},\text{ usando partes enteras al multiplicar por } 2$$

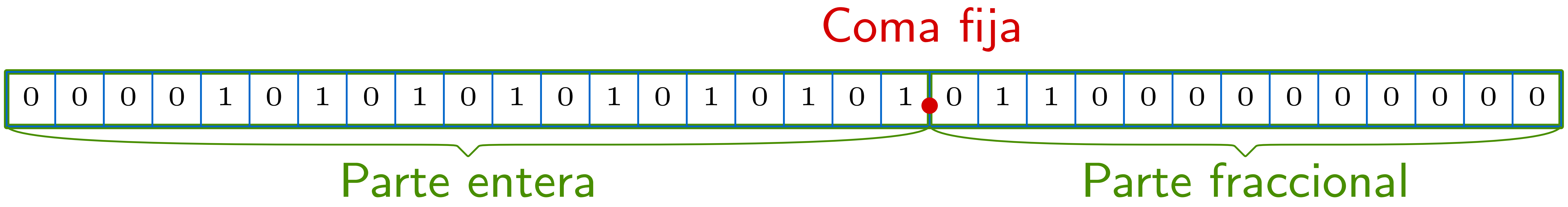
Aritmética de punto/coma flotante

Números en el ordenador

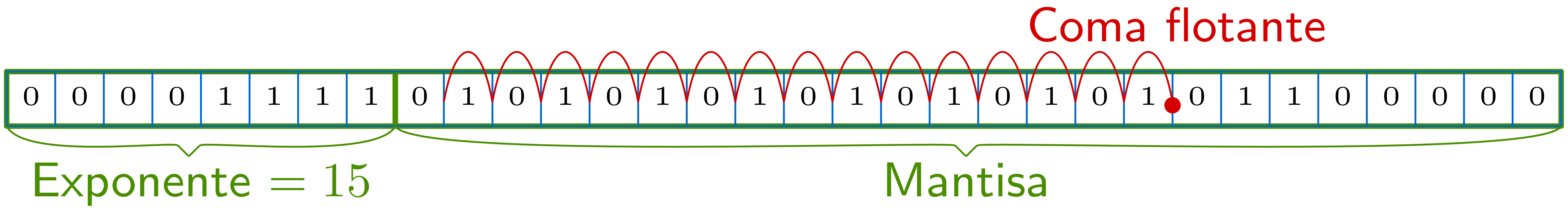
Representación de números reales en binario

$$21845.375_{(10)} = 101010101010101.011_{(2)}$$

Coma fija en binario



Coma flotante en binario



Historia

En las décadas de 1960 y 1970, las operaciones con números reales tenían implementaciones diferentes en cada ordenador: formato, precisión, redondeo, gestión de excepciones, etc. De esta manera, era muy difícil escribir código portátil.

En 1982, el Institute of Electrical and Electronics Engineers definió el estándar IEEE-754 y lo implementó en los procesadores Intel 8087. En todos los ordenadores que tenían este estándar implementado, los programas obtenían los mismos resultados.

En 2002, el estándar IEEE-754 se implementó universalmente en todos los ordenadores de propósito general.

Cleve's Corner: Cleve Moler on Mathematics and Computing, Floating Point Arithmetic Before IEEE 754

Norma 754 — 1985

En el año 1985, el Institute for Electrical and Electronic Engineers (IEEE) publicó el informe

Binary Floating Point Arithmetic Standard 754 – 1985,

en el que se especifican normas para representar números en punto/coma flotante con precisión simple, doble y extendida. El informe fue revisado y actualizado en el año 2008, IEEE Std 754-2008.

Hoy en día, casi todos los fabricantes de ordenadores han aceptado esta norma; por lo tanto, el ordenador almacena no el número real x, sino una aproximación binaria (octal o hexadecimal) en coma flotante a x.

Cleve's Corner: Cleve Moler on Mathematics and Computing, Floating Point Numbers

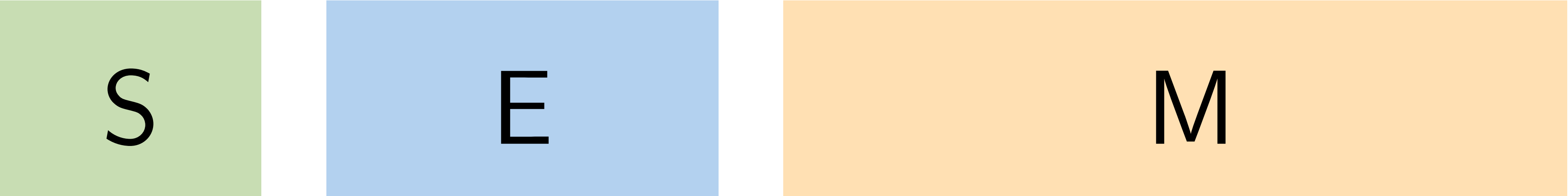
Formato de coma flotante

El formato de coma flotante usa tres campos binarios para la representación: signo (S), exponente (E) y fracción o mantisa (M).

S	E	M

Formato de coma flotante

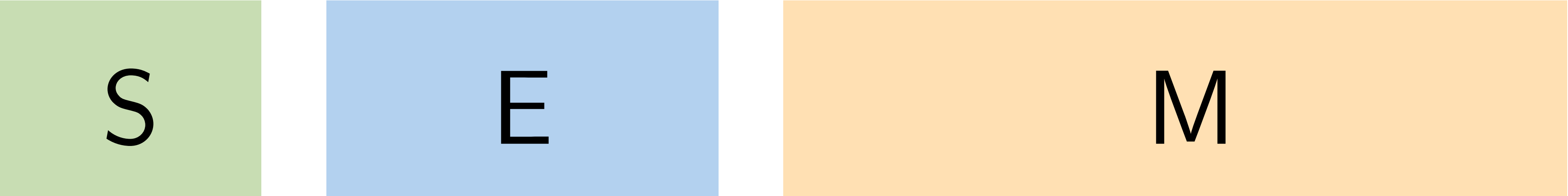
El formato de coma flotante usa tres campos binarios para la representación: signo (S), exponente (E) y fracción o mantisa (M).



El signo (S) ocupa un bit: 0 para números positivos y 1 para números negativos.

Formato de coma flotante

El formato de coma flotante usa tres campos binarios para la representación: signo (S), exponente (E) y fracción o mantisa (M).



El signo (S) ocupa un bit: 0 para números positivos y 1 para números negativos.

El exponente (E) se guarda desviado $2^{\text{bits de E}-1} - 1$.

Formato de coma flotante

El formato de coma flotante usa tres campos binarios para la representación: signo (S), exponente (E) y fracción o mantisa (M).

El signo (S) ocupa un bit: 0 para números positivos y 1 para números negativos.

El exponente (E) se guarda desviado $2^{\text{bits de E}-1} - 1$.

La mantisa (M) tiene un 1 implícito a la izquierda, es decir, la mantisa usa una notación científica normalizada.

Formato de coma flotante

El formato de coma flotante usa tres campos binarios para la representación: signo (S), exponente (E) y fracción o mantisa (M).

S	E	M

El signo (S) ocupa un bit: 0 para números positivos y 1 para números negativos.

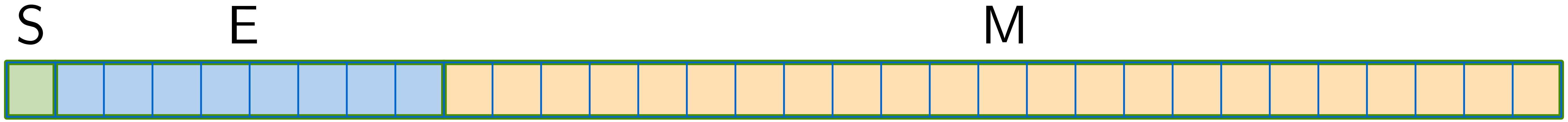
El exponente (E) se guarda desviado $2^{\text{bits de E}-1} - 1$.

La mantisa (M) tiene un 1 implícito a la izquierda, es decir, la mantisa usa una notación científica normalizada.

Formato IEEE 754	S	E	M	Desviación de E
32 bit precisión simple	1 bit	8 bits	23 bits (+1 impl.)	$2^{8-1} - 1 = 127$
64 bit precisión doble	1 bit	11 bits	52 bits (+1 impl.)	$2^{11-1} - 1 = 1023$
128 bit precisión cuádruple	1 bit	15 bits	112 bits (+1 impl.)	$2^{15-1} - 1 = 16383$

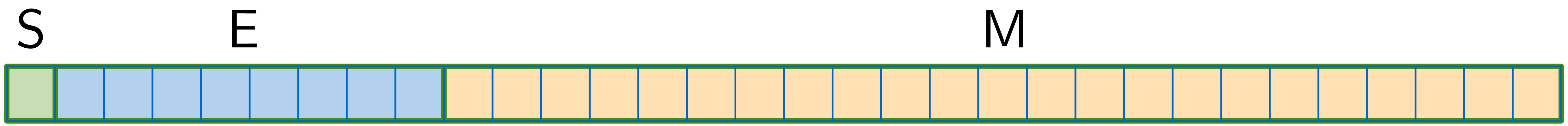
Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Ejemplo

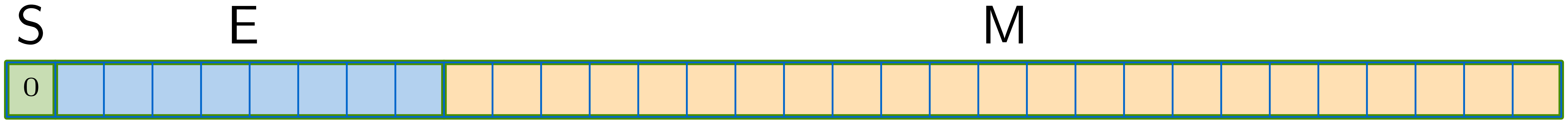
Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Ejemplo

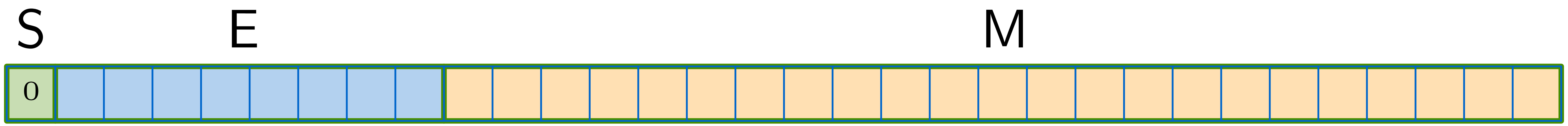
Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.

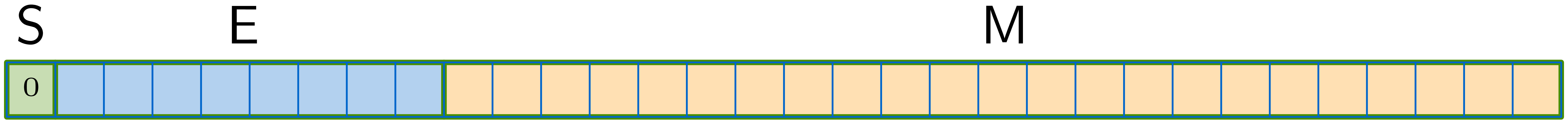


Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Paso 2: Convertir a binario puro.

Ejemplo

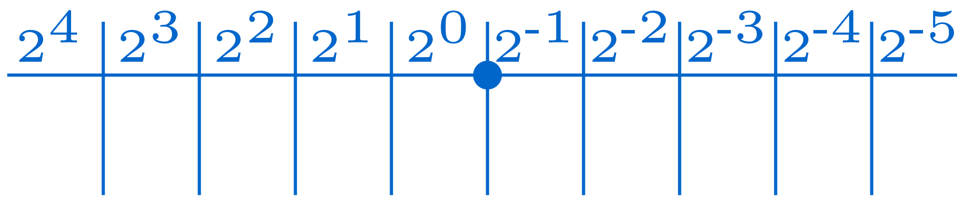
Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

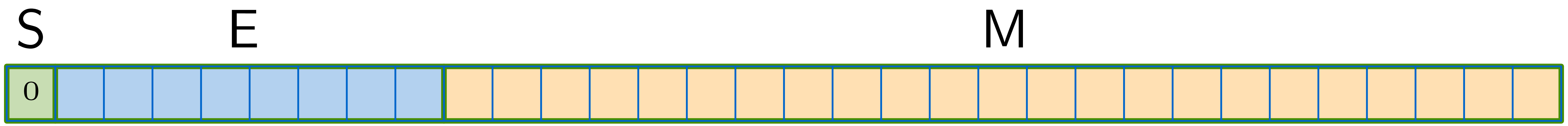
Paso 2: Convertir a binario puro.

$$19.59375_{(10)} =$$



Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Paso 2: Convertir a binario puro.

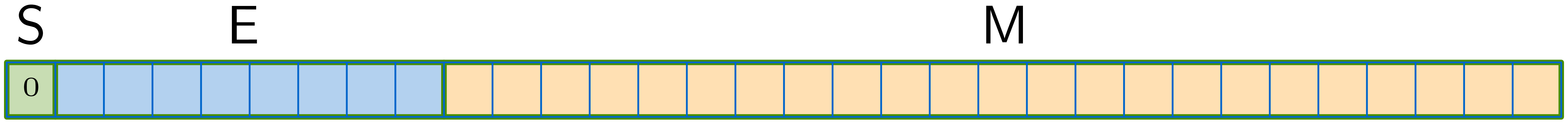
$19.59375_{(10)} =$

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}

$$19 \div 2 \quad = 9 \text{ resto } 1$$
$$9 \div 2 \quad = 4 \text{ resto } 1$$
$$4 \div 2 \quad = 2 \text{ resto } 0$$
$$2 \div 2 \quad = 1 \text{ resto } 0$$
$$\text{resto } 1$$

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

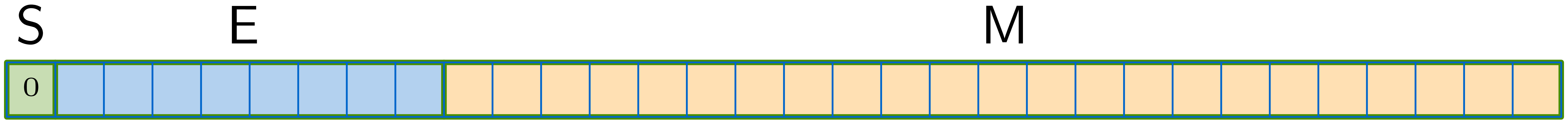
Paso 2: Convertir a binario puro.

$$19.59375_{(10)} = \begin{array}{c|c|c|c|c|c|c|c|c|c} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \\ \hline 1 & 0 & 0 & 1 & 1. & & & & & \end{array}$$

$$\begin{aligned} 19 \div 2 &= 9 \text{ resto } 1 \\ 9 \div 2 &= 4 \text{ resto } 1 \\ 4 \div 2 &= 2 \text{ resto } 0 \\ 2 \div 2 &= 1 \text{ resto } 0 \\ &\quad\;\; \text{resto } 1 \end{aligned}$$

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Paso 2: Convertir a binario puro.

$19.59375_{(10)} =$

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
1	0	0	1	1.					

$$
\begin{aligned}
19 \div 2 &= 9 \text{ resto } 1 & \quad 0.59375 \times 2 &= 1.1875 \text{ parte entera } 1 \\
9 \div 2 &= 4 \text{ resto } 1 & 0.1865 \times 2 &= 0.375 \text{ parte entera } 0 \\
4 \div 2 &= 2 \text{ resto } 0 & 0.375 \times 2 &= 0.75 \text{ parte entera } 0 \\
2 \div 2 &= 1 \text{ resto } 0 & 0.75 \times 2 &= 1.5 \text{ parte entera } 1 \\
&\text{resto } 1 & 0.5 \times 2 &= 1 \text{ parte entera } 1
\end{aligned}
$$

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.

S	E								M																						
0																															

Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

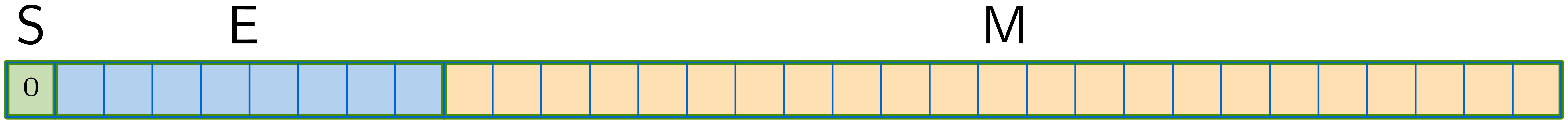
Paso 2: Convertir a binario puro.

$$19.59375_{(10)} = \begin{array}{c|c|c|c|c|c|c|c|c|c} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \\ \hline 1 & 0 & 0 & 1 & 1. & 1 & 0 & 0 & 1 & 1 \end{array}$$

$$\begin{aligned} 19 \div 2 &= 9 \text{ resto } \boxed{1} \\ 9 \div 2 &= 4 \text{ resto } \boxed{1} \\ 4 \div 2 &= 2 \text{ resto } \boxed{0} \\ 2 \div 2 &= 1 \text{ resto } \boxed{0} \\ & \text{resto } \boxed{1} \end{aligned} \qquad \begin{aligned} 0.59375 \times 2 &= 1.1875 \text{ parte entera } \boxed{1} \\ 0.1865 \times 2 &= 0.375 \text{ parte entera } \boxed{0} \\ 0.375 \times 2 &= 0.75 \text{ parte entera } \boxed{0} \\ 0.75 \times 2 &= 1.5 \text{ parte entera } \boxed{1} \\ 0.5 \times 2 &= 1 \text{ parte entera } \boxed{1} \end{aligned}$$

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

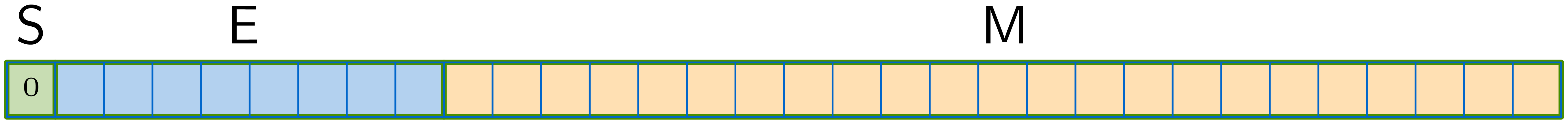
Paso 2: Convertir a binario puro.

$$19.59375_{(10)} = \begin{array}{c|c|c|c|c|c|c|c|c|c} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \\ \hline 1 & 0 & 0 & 1 & 1. & 1 & 0 & 0 & 1 & 1 \end{array}$$

Paso 3: Normalizar para determinar la mantisa y el exponente sin desviación (coma después del primer 1).

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Paso 2: Convertir a binario puro.

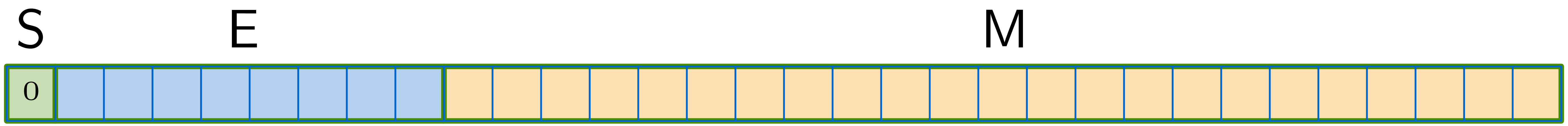
$19.59375_{(10)} =$

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
1	0	0	1	1.	1	0	0	1	1

Paso 3: Normalizar para determinar la mantisa y el exponente sin desviación (coma después del primer 1). $1.0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1 \times 2^4$

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Paso 2: Convertir a binario puro.

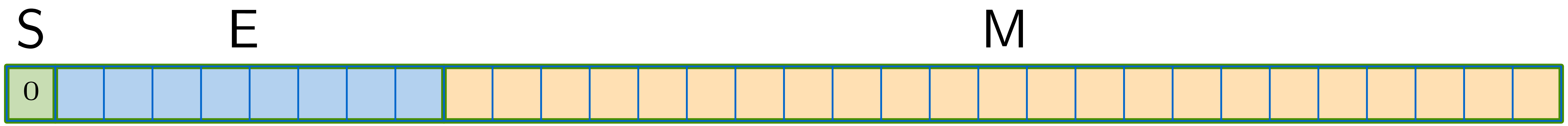
$$19.59375_{(10)} = \begin{array}{c|c|c|c|c|c|c|c|c|c} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \\ \hline 1 & 0 & 0 & 1 & 1. & 1 & 0 & 0 & 1 & 1 \end{array}$$

Paso 3: Normalizar para determinar la mantisa y el exponente sin desviación (coma después del primer 1). $1.0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1 \times 2^4$

Paso 4: Calcular el exponente desviado (sumar desviación y convertir a binario).

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.



Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Paso 2: Convertir a binario puro.

$19.59375_{(10)} =$

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
1	0	0	1	1.	1	0	0	1	1

Paso 3: Normalizar para determinar la mantisa y el exponente sin desviación (coma después del primer 1). 1.001110011×2^4

Paso 4: Calcular el exponente desviado (sumar desviación y convertir a binario).

$$4 + 127 = 131_{(10)} = 10000011_{(2)}$$

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.

S	E								M
0	1	0	0	0	0	0	1	1	

Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Paso 2: Convertir a binario puro.

$$19.59375_{(10)} = \begin{array}{c|c|c|c|c|c|c|c|c|c} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \\ \hline 1 & 0 & 0 & 1 & 1. & 1 & 0 & 0 & 1 & 1 \end{array}$$

Paso 3: Normalizar para determinar la mantisa y el exponente sin desviación (coma después del primer 1). $1.0\,0\,1\,1\,1\;0\;0\;1\;1 \times 2^4$

Paso 4: Calcular el exponente desviado (sumar desviación y convertir a binario).

$$4 + 127 = 131_{(10)} = 10000011_{(2)}$$

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.

S	E								M																						
0	1	0	0	0	0	0	1	1																							

Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Paso 2: Convertir a binario puro.

$$19.59375_{(10)} = \begin{array}{c|c|c|c|c|c|c|c|c|c} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \\ \hline 1 & 0 & 0 & 1 & 1. & 1 & 0 & 0 & 1 & 1 \end{array}$$

Paso 3: Normalizar para determinar la mantisa y el exponente sin desviación (coma después del primer 1). $1.0\;0\;1\;1\;1\;0\;0\;1\;1 \times 2^4$

Paso 4: Calcular el exponente desviado (sumar desviación y convertir a binario).

$$4 + 127 = 131_{(10)} = 10000011_{(2)}$$

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.

S	E								M																						
0	1	0	0	0	0	0	1	1																							

Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Paso 2: Convertir a binario puro.

$$19.59375_{(10)} = \begin{array}{c|c|c|c|c|c|c|c|c|c} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \\ \hline 1 & 0 & 0 & 1 & 1. & 1 & 0 & 0 & 1 & 1 \end{array}$$

Paso 3: Normalizar para determinar la mantisa y el exponente sin desviación (coma después del primer 1). $1.0\;0\;1\;1\;1\;0\;0\;1\;1 \times 2^4$

Paso 4: Calcular el exponente desviado (sumar desviación y convertir a binario).

$$4 + 127 = 131_{(10)} = 10000011_{(2)}$$

Paso 5: Quitar el primer 1 de la mantisa calculada (es implícito).

$$1.001110011 \rightarrow 001110011$$

Ejemplo

Convertir 19.59375 al estándar IEEE 754 de 32 bits de coma flotante en binario.

S	E								M																						
0	1	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Paso 1: Determinar el signo (0 si positivo, 1 si negativo).

Paso 2: Convertir a binario puro.

$19.59375_{(10)} =$

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
1	0	0	1	1.	1	0	0	1	1

Paso 3: Normalizar para determinar la mantisa y el exponente sin desviación (coma después del primer 1). $1.0\;0\;1\;1\;1\;0\;0\;1\;1 \times 2^4$

Paso 4: Calcular el exponente desviado (sumar desviación y convertir a binario).

$$4 + 127 = 131_{(10)} = 10000011_{(2)}$$

Paso 5: Quitar el primer 1 de la mantisa calculada (es implícito).

$$1.001110011 \to 001110011$$

Desviación del exponente: caso 4 bits

0000

0001

0010

0011

0100

0101

0110

0111

1000

1001

1010

1011

1100

1101

1110

1111

Desviación del exponente: caso 4 bits

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Desviación del exponente: caso 4 bits

0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Desviación del exponente: caso 4 bits

0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

El complemento a 2 es una forma habitual de representar números enteros con signo (positivo, negativo y cero) en el sistema binario. Utiliza el primer dígito binario como signo (1 si el signo es negativo y 0 en otro caso).

Desviación del exponente: caso 4 bits

0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
$\boxed{1010}$	10	$\boxed{-6}$
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

El complemento a 2 es una forma habitual de representar números enteros con signo (positivo, negativo y cero) en el sistema binario. Utiliza el primer dígito binario como signo (1 si el signo es negativo y 0 en otro caso).

$$1 \cdot -2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = -8 + 2 = -6$$

Desviación del exponente: caso 4 bits

0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

El complemento a 2 es una forma habitual de representar números enteros con signo (positivo, negativo y cero) en el sistema binario. Utiliza el primer dígito binario como signo (1 si el signo es negativo y 0 en otro caso).

$$1 \cdot -2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = -8 + 2 = -6$$

$$(-6+8)_{(10)} = 2_{(10)} = (010)_{(2)}$$

Desviación del exponente: caso 4 bits

0000	0	0	-5
0001	1	1	-4
0010	2	2	-3
0011	3	3	-2
0100	4	4	-1
0101	5	5	0
0110	6	6	1
0111	7	7	2
1000	8	-8	3
1001	9	-7	4
1010	10	-6	5
1011	11	-5	6
1100	12	-4	7
1101	13	-3	8
1110	14	-2	9
1111	15	-1	10

Desviación del exponente: caso 4 bits

0000	0	0	-5
0001	1	1	-4
0010	2	2	-3
0011	3	3	-2
0100	4	4	-1
0101	5	5	0
0110	6	6	1
0111	7	7	2
1000	8	-8	3
1001	9	-7	4
1010	10	-6	5
1011	11	-5	6
1100	12	-4	7
1101	13	-3	8
1110	14	-2	9
1111	15	-1	10

Más números positivos que negativos: la conversión depende de esta diferencia.

Desviación del exponente: caso 4 bits

0000	0	0	-5
0001	1	1	-4
$\boxed{0010}$	$\boxed{2}$	2	$\boxed{-3}$
0011	3	3	-2
0100	4	4	-1
0101	5	5	0
0110	6	6	1
0111	7	7	2
1000	8	-8	3
1001	9	-7	4
1010	10	-6	5
$\boxed{1011}$	$\boxed{11}$	-5	$\boxed{6}$
1100	12	-4	7
1101	13	-3	8
1110	14	-2	9
1111	15	-1	10

Más números positivos que negativos: la conversión depende de esta diferencia.

En este caso: sumar 5 y convertir a binario.

Desviación del exponente: caso 4 bits

0000	0	0	-5
0001	1	1	-4
$\boxed{0010}$	$\boxed{2}$	2	$\boxed{-3}$
0011	3	3	-2
0100	4	4	-1
0101	5	5	0
0110	6	6	1
0111	7	7	2
1000	8	-8	3
1001	9	-7	4
1010	10	-6	5
$\boxed{1011}$	$\boxed{11}$	-5	$\boxed{6}$
1100	12	-4	7
1101	13	-3	8
1110	14	-2	9
1111	15	-1	10

Más números positivos que negativos: la conversión depende de esta diferencia.

En este caso: sumar 5 y convertir a binario.

Los números de pueden comparar bit a bit (están ordenados lexicográficamente).

Desviación del exponente: caso 4 bits

0000	0	0	-5	-7
0001	1	1	-4	-6
0010	2	2	-3	-5
0011	3	3	-2	-4
0100	4	4	-1	-3
0101	5	5	0	-2
0110	6	6	1	-1
0111	7	7	2	0
1000	8	-8	3	1
1001	9	-7	4	2
1010	10	-6	5	3
1011	11	-5	6	4
1100	12	-4	7	5
1101	13	-3	8	6
1110	14	-2	9	7
1111	15	-1	10	8

Desviación del exponente: caso 4 bits

0000	0	0	-5	-7
0001	1	1	-4	-6
0010	2	2	-3	-5
0011	3	3	-2	-4
0100	4	4	-1	-3
0101	5	5	0	-2
0110	6	6	1	-1
0111	7	7	2	0
1000	8	-8	3	1
1001	9	-7	4	2
1010	10	-6	5	3
1011	11	-5	6	4
1100	12	-4	7	5
1101	13	-3	8	6
1110	14	-2	9	7
1111	15	-1	10	8

IEEE 754 utiliza este tipo de desviación.

$2^{(4-1)} - 1 = 7$ números negativos $\Rightarrow$ desviación 7.

El exponente (E) se guarda desviado $2^{\text{bits de E}-1} - 1$. Precisión simple (32 bits): E ocupa 8 bits y se suma $2^{8-1} - 1 = 127$.

Desviación del exponente: caso 4 bits

$\boxed{0000}$	0	0	-5	$\boxed{}$
0001	1	1	-4	-6
0010	2	2	-3	-5
0011	3	3	-2	-4
0100	4	4	-1	-3
0101	5	5	0	-2
0110	6	6	1	-1
0111	7	7	2	0
1000	8	-8	3	1
1001	9	-7	4	2
1010	10	-6	5	3
1011	11	-5	6	4
1100	12	-4	7	5
1101	13	-3	8	6
1110	14	-2	9	7
$\boxed{1111}$	15	-1	10	$\boxed{}$

IEEE 754 utiliza este tipo de desviación.

$2^{(4-1)} - 1 = 7$ números negativos $\Rightarrow$ desviación 7.

El exponente (E) se guarda desviado $2^{\text{bits de E}-1} - 1$. Precisión simple (32 bits): E ocupa 8 bits y se suma $2^{8-1} - 1 = 127$.

Ejercicios

Ejercicio 1. Convertir -123.3 a formato IEEE 754 binario de coma flotante de precisión simple.

Ejercicio 2. Convertir $110000100110101000000000000000$ a decimal.

Ejercicios

Ejercicio 1. Convertir -123.3 a formato IEEE 754 binario de coma flotante de precisión simple.

> Resultado: $1\ 10000101\ 11101101001100110011010$ (si se redondea) o $1\ 10000101\ 11101101001100110011001$ si se trunca.

Ejercicio 2. Convertir $11000010011010100000000000000000$ a decimal.

> Resultado: 58.5.

Valores reservados

Exponente (E)	Mantisa (M)	Representa
11111111	Todo ceros	Infinito (∞)
11111111	No todo ceros	Not a Number (NaN)
00000000	Todo ceros	Cero (0)
00000000	No todo ceros	Número denormal (muy pequeño)

IEEE 754 precisión doble (64 bits)

La representación en coma flotante con doble precisión es

$$\mathrm{fl}(x) = (-1)^S \times (1 + M) \times 2^{E-1023}.$$

Se usa $1 + M$ para aumentar en un bit la precisión.

La representación tiene 64 dígitos binarios, asignados de la siguiente manera:

S	E	M
1 bit	11 bits	52 bits

IEEE 754 precisión doble (64 bits)

La representación en coma flotante con doble precisión es

$$\mathrm{fl}(x) = (-1)^S \times (1+M) \times 2^{E-1023}.$$

Se usa $1+M$ para aumentar en un bit la precisión.

La representación tiene 64 dígitos binarios, asignados de la siguiente manera:

S	E	M
1 bit	11 bits	52 bits
$S \in \{0,1\}$	Entero $-2^{10}-2 \leq E \leq 2^{10}-1$ $-1022 \leq E \leq 1023$	Real, $0 \leq M < 1$ $2^{52}M$ es un entero en el intervalo $0 \leq 2^{52}M < 2^{53}$

IEEE 754 precisión doble (64 bits)

La representación en coma flotante con doble precisión es

$$\mathrm{fl}(x) = (-1)^S \times (1 + M) \times 2^{E-1023}.$$

Se usa $1 + M$ para aumentar en un bit la precisión.

La representación tiene 64 dígitos binarios, asignados de la siguiente manera:

S	E	M
1 bit	11 bits	52 bits
$S \in \{0, 1\}$	Entero $-2^{10}-2 \leq E \leq 2^{10}-1$ $-1022 \leq E \leq 1023$	Real, $0 \leq M < 1$ $2^{52}M$ es un entero en el intervalo $0 \leq 2^{52}M < 2^{53}$
Signo	Rango	Precisión

IEEE 754 precisión doble (64 bits)

La representación en coma flotante con doble precisión es

$$\mathrm{fl}(x) = (-1)^S \times (1 + M) \times 2^{E-1023}.$$

Se usa $1 + M$ para aumentar en un bit la precisión.

La representación tiene 64 dígitos binarios, asignados de la siguiente manera:

S	E	M
1 bit	11 bits	52 bits
$S \in \{0, 1\}$	Entero	Real, $0 \le M < 1$
	$-2^{10}-2 \le E \le 2^{10}-1$	$2^{52}M$ es un entero en el intervalo $0 \le 2^{52}M < 2^{53}$
	$-1022 \le E \le 1023$	
Signo	Rango	Precisión

Los números así representables están distribuidos de forma irregular, concentrándose más cerca de cero; entre dos potencias consecutivas de 2 siempre existe la misma cantidad de números.

Underflow, overflow, y epsilon de la máquina en precisión doble

La representación en coma flotante con doble precisión es

$$\mathsf{fl}(x) = (-1)^S \times (1 + M) \times 2^{E-1023}.$$

S	E	M
1 bit	11 bits	52 bits

Underflow, overflow, y epsilon de la máquina en precisión doble

La representación en coma flotante con doble precisión es

$$\mathsf{fl}(x) = (-1)^S \times (1 + M) \times 2^{E-1023}.$$

S	E	M
1 bit	11 bits	52 bits

- Entre potencias consecutivas de 2 hay $2^{52} - 1$ números máquina equidistantes.

Underflow, overflow, y epsilon de la máquina en precisión doble

La representación en coma flotante con doble precisión es

$$\mathrm{fl}(x) = (-1)^S \times (1 + M) \times 2^{E-1023}.$$

- Entre potencias consecutivas de 2 hay $2^{52} - 1$ números máquina equidistantes.

- Épsilon de la máquina (eps): diferencia entre el menor número máquina mayor que 1 y 1. ¿Cuánto vale?
- Número de overflow (realmax): el número máquina positivo más grande. ¿Cuánto vale?
- Número de underflow (realmin): el número máquina positivo más pequeño. ¿Cuánto vale?

Underflow, overflow, y epsilon de la máquina en precisión doble

La representación en coma flotante con doble precisión es

$$\mathrm{fl}(x) = (-1)^S \times (1 + M) \times 2^{E-1023}.$$

- Entre potencias consecutivas de 2 hay $2^{52} - 1$ números máquina equidistantes.

- Épsilon de la máquina (eps): diferencia entre el menor número máquina mayor que 1 y 1. ¿Cuánto vale? eps $= 2^{-52}$
- Número de overflow (realmax): el número máquina positivo más grande. ¿Cuánto vale?
- Número de underflow (realmin): el número máquina positivo más pequeño. ¿Cuánto vale?

Underflow, overflow, y epsilon de la máquina en precisión doble

La representación en coma flotante con doble precisión es

$$\mathrm{fl}(x) = (-1)^S \times (1 + M) \times 2^{E-1023}.$$

- Entre potencias consecutivas de 2 hay $2^{52} - 1$ números máquina equidistantes.

- Épsilon de la máquina (eps): diferencia entre el menor número máquina mayor que 1 y 1. ¿Cuánto vale? $\text{eps} = 2^{-52}$
- Número de overflow (realmax): el número máquina positivo más grande. ¿Cuánto vale? $\text{realmax} = (2 - \text{eps}) \times 2^{(2^{10}-1)} = (2 - \text{eps}) \times 2^{1023}$
- Número de underflow (realmin): el número máquina positivo más pequeño. ¿Cuánto vale?

Underflow, overflow, y epsilon de la máquina en precisión doble

La representación en coma flotante con doble precisión es

$$\mathrm{fl}(x) = (-1)^S \times (1 + M) \times 2^{E-1023}.$$

- Entre potencias consecutivas de 2 hay $2^{52} - 1$ números máquina equidistantes.

- Épsilon de la máquina (eps): diferencia entre el menor número máquina mayor que 1 y 1. ¿Cuánto vale? $\mathrm{eps} = 2^{-52}$
- Número de overflow (realmax): el número máquina positivo más grande. ¿Cuánto vale? $\mathrm{realmax} = (2 - \mathrm{eps}) \times 2^{(2^{10}-1)} = (2 - \mathrm{eps}) \times 2^{1023}$
- Número de underflow (realmin): el número máquina positivo más pequeño. ¿Cuánto vale? $\mathrm{realmin} = (1) \times 2^{-(2^{10}-2)} = 2^{-1022}$

Underflow, overflow, y epsilon de la máquina en precisión doble

La representación en coma flotante con doble precisión es

$$\text{fl}(x) = (-1)^S \times (1 + M) \times 2^{E-1023}.$$

S	E	M
1 bit	11 bits	52 bits

- Entre potencias consecutivas de 2 hay $2^{52} - 1$ números máquina equidistantes.

- Épsilon de la máquina (eps): diferencia entre el menor número máquina mayor que 1 y 1. ¿Cuánto vale? $\text{eps} = 2^{-52}$ $\approx 2.2204 \times 10^{-16}$
- Número de overflow (realmax): el número máquina positivo más grande. ¿Cuánto vale? $\text{realmax} = (2 - \text{eps}) \times 2^{(2^{10}-1)} = (2 - \text{eps}) \times 2^{1023}$ $\approx 1.7977 \times 10^{308}$
- Número de underflow (realmin): el número máquina positivo más pequeño. ¿Cuánto vale? $\text{realmin} = (1) \times 2^{-(2^{10}-2)} = 2^{-1022}$ $\approx 2.2251 \times 10^{-308}$

Precisión del ordenador/software

La precisión de un ordenador dependerá del fabricante y del tipo de variable que se defina; la unidad de información viene dada por el número de dígitos binarios o longitud de la palabra (word): 1 byte $= 8$ bits, 1 word $= 2$ bytes, or 4 bytes, or 8 bytes...

El ordenador almacena no el número real x, sino una aproximación binaria a este número, generalmente designada como $\mathrm{fl}(x)$.

Cleve's Corner: Cleve Moler on Mathematics and Computing, Floating Point Arithmetic Before IEEE 754

El conjunto $F(\beta, t, L, U)$

El conjunto de números en coma flotante representables en el ordenador lo designaremos como $F(\beta, t, L, U)$, donde β representa la base, t la precisión (o número de dígitos representados o significativos) y el intervalo $[L, U]$ es el rango del exponente e.

Para todo número real x expresado en el conjunto $F(\beta, t, L, U)$ existen t cifras $d_i \in \mathbb{N}$, $0 \leq d_i < \beta$, con $i = [1 : t]$ y un exponente e, $L \leq e \leq U$, tal que:

$$\mathrm{fl}(x) = \underbrace{\pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_t}{\beta^t} \right)}_{\text{Parte fraccionaria}} \cdot \beta^e,$$

El conjunto $F(\beta, t, L, U)$

El conjunto de números en coma flotante representables en el ordenador lo designaremos como $F(\beta, t, L, U)$, donde β representa la base, t la precisión (o número de dígitos representados o significativos) y el intervalo $[L, U]$ es el rango del exponente e.

Para todo número real x expresado en el conjunto $F(\beta, t, L, U)$ existen t cifras $d_i \in \mathbb{N}$, $0 \leq d_i < \beta$, con $i = [1 : t]$ y un exponente e, $L \leq e \leq U$, tal que:

$$\text{fl}(x) = \underbrace{\pm\left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_t}{\beta^t}\right)}_{\text{Parte fraccionaria}} \cdot \beta^e,$$

- Puede haber $U - L + 1$ exponentes diferentes, por lo que representa el rango.

El conjunto $F(\beta, t, L, U)$

El conjunto de números en coma flotante representables en el ordenador lo designaremos como $F(\beta, t, L, U)$, donde β representa la base, t la precisión (o número de dígitos representados o significativos) y el intervalo $[L, U]$ es el rango del exponente e.

Para todo número real x expresado en el conjunto $F(\beta, t, L, U)$ existen t cifras $d_i \in \mathbb{N}$, $0 \leq d_i < \beta$, con $i = [1 : t]$ y un exponente e, $L \leq e \leq U$, tal que:

$$\mathrm{fl}(x) = \underbrace{\pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_t}{\beta^t} \right)}_{\text{Parte fraccionaria}} \cdot \beta^e,$$

- Puede haber $U - L + 1$ exponentes diferentes, por lo que representa el rango.
- Si exigimos $d_1 \neq 0$ para $x \neq 0$, se dice que la representación está normalizada.

El conjunto $F(\beta, t, L, U)$

El conjunto de números en coma flotante representables en el ordenador lo designaremos como $F(\beta, t, L, U)$, donde β representa la base, t la precisión (o número de dígitos representados o significativos) y el intervalo $[L, U]$ es el rango del exponente e.

Para todo número real x expresado en el conjunto $F(\beta, t, L, U)$ existen t cifras $d_i \in \mathbb{N}$, $0 \leq d_i < \beta$, con $i = [1 : t]$ y un exponente e, $L \leq e \leq U$, tal que:

$$\mathrm{fl}(x) = \underbrace{\pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_t}{\beta^t} \right)}_{\text{Parte fraccionaria}} \cdot \beta^e,$$

- Puede haber $U - L + 1$ exponentes diferentes, por lo que representa el rango.
- Si exigimos $d_1 \neq 0$ para $x \neq 0$, se dice que la representación está normalizada.
- La cantidad $m = d_1 d_2 d_3 \ldots d_t$ se llama mantisa. Puede haber β^t mantisas diferentes.

Épsilon de la máquina

Si el ordenador usa la aritmética $F(\beta, t, L, U)$, entonces

$$\mathrm{fl}(x) = x(1+\delta) \qquad\qquad |\delta| \leq \mathsf{eps}_M = \frac{1}{2}\beta^{1-t}.$$

La precisión de una aritmética de coma flotante se caracteriza por el épsilon de la máquina eps_M. No es el número más pequeño representable, pero proporciona una medida relativa de hasta qué punto dos números muy cercanos serán diferentes.

Épsilon de la máquina

Si el ordenador usa la aritmética $F(\beta, t, L, U)$, entonces

$$\mathsf{fl}(x) = x(1+\delta) \qquad\qquad |\delta| \leq \mathsf{eps}_M = \frac{1}{2}\beta^{1-t}.$$

La precisión de una aritmética de coma flotante se caracteriza por el épsilon de la máquina eps_M. No es el número más pequeño representable, pero proporciona una medida relativa de hasta qué punto dos números muy cercanos serán diferentes.

En Matlab es el épsilon que hemos visto para la aritmética de punto flotante IEEE 754 en precisión doble, $\mathsf{eps}_{\mathsf{Matlab}} = 2^{-52} \approx 2.2204 \times 10^{-16}$.

Aritmética en $F(\beta, t, L, U)$

Les operaciones aritméticas en coma flotante son

$$
\begin{aligned}
x + y &= \longleftrightarrow & x \oplus y &= \mathsf{fl}(\mathsf{fl}(x) + \mathsf{fl}(y)) \\
x - y &= \longleftrightarrow & x \ominus y &= \mathsf{fl}(\mathsf{fl}(x) - \mathsf{fl}(y)) \\
x \cdot y &= \longleftrightarrow & x \otimes y &= \mathsf{fl}(\mathsf{fl}(x) \cdot \mathsf{fl}(y)) \\
x \div y &= \longleftrightarrow & x \oslash y &= \mathsf{fl}(\mathsf{fl}(x) \div \mathsf{fl}(y))
\end{aligned}
$$

Teorema. En todas las operaciones tenemos

$$(x \circledast y) \cdot (1 \pm \delta) = \mathsf{fl}(x \circledast y)$$

con $0 < \delta < \mathrm{eps}_M$, asumiendo que los registros aritméticos admiten $2t + 2$ dígitos. Eso quiere decir que los resultados de sumar y multiplicar antes de normalizar y redondear son exactos.

Aritmética en $F(\beta, t, L, U)$: ejemplos

Imaginemos un ordenador $F(10, 5, 0, 127)$, y los números $x = 0.31426 \times 10^3$ e $y = 0.92577 \times 10^5$.

$$x \times y = 0.2909324802 \times 10^8 \qquad x \otimes y = 0.29093 \times 10^8$$

$$x + y = 0.9289126 \times 10^5 \qquad x \oplus y = 0.92891 \times 10^5$$

$$x - y = -0.92262740 \times 10^5 \qquad x \ominus y = -0.92263 \times 10^5$$

$$x \div y = 0.3394579647 \times 10^{-2} \qquad x \oslash y = 0.33946 \times 10^{-2}$$

En estos resultados, el error relativo es de 8.5×10^{-6}, 2.3×10^{-6}, 2.8×10^{-6}, 6.0×10^{-6}, respectivamente, todos por debajo de 10^{-5}.

Aritmética de punto/coma flotante

Aritmética de Matlab

Aritmètica de Matlab

Leer los documentos:

- "Floating points" by Cleve Moler.
- Cleve's Corner: Cleve Moler on Mathematics and Computing, Floating Point Arithmetic Before IEEE 754.
- Cleve's Corner: Cleve Moler on Mathematics and Computing, Floating Point Numbers.
- Cleve's Corner: Cleve Moler on Mathematics and Computing, Floating Point Denormals, Insignificant But Controversial.
- MathWorks Documentation Center, Floating-Point Numbers.

Estabilidad numérica y problemas bien condicionados

Some disasters attributable to bad numerical computing

Sensibilidad a las condiciones iniciales

Muchos problemas son especialmente sensibles a los datos iniciales, independientemente de los errores de redondeo y del algoritmo empleado.

Ejemplo. Polinomio de Wilkinson.

Sea $p(x) = (x-1)(x-2)(x-3)\dots(x-20)$, el polinomio con raíces los veinte primeros números naturales, definimos el polinomio $q(x) = p(x) + \frac{1}{2^{23}}x^{19}$, modificando ligeramente el coeficiente de x^{19} respecto de $p(x)$. ¿Cómo deberían ser las raíces del polinomio $q(x)$? Cálculelas.

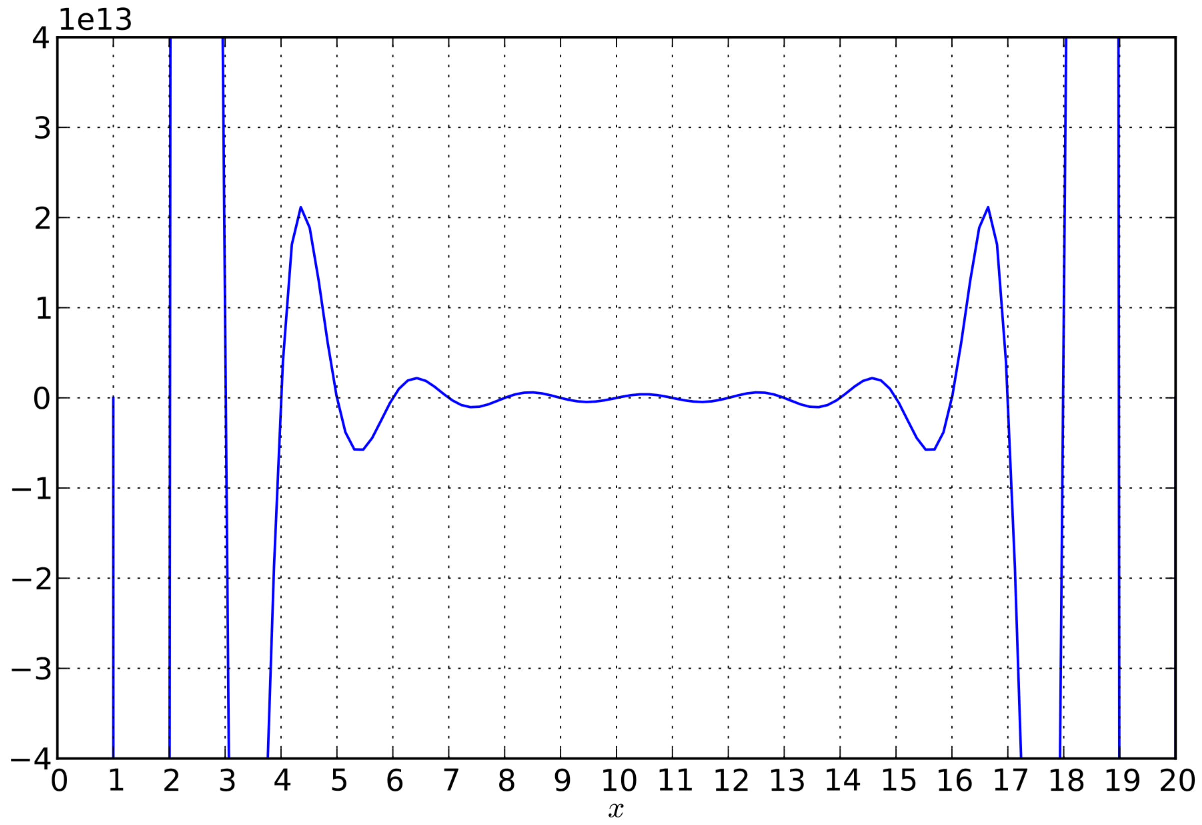
Sensibilidad a las condiciones iniciales

Muchos problemas son especialmente sensibles a los datos iniciales, independientemente de los errores de redondeo y del algoritmo empleado.

Ejemplo. Polinomio de Wilkinson.
Sea $p(x) = (x-1)(x-2)(x-3)\dots(x-20)$, el polinomio con raíces los veinte primeros números naturales, definimos el polinomio $q(x) = p(x) + \frac{1}{2^{23}}x^{19}$, modificando ligeramente el coeficiente de x^{19} respecto de $p(x)$. ¿Cómo deberían ser las raíces del polinomio $q(x)$? Cálculelas.

Solución. Las raíces de $q(x)$ son (5 decimales):

1.00000	2.00000	3.00000	4.00000
5.00000	6.00001	6.99970	8.00727
8.91725	20.84691	$10.09527 \pm 0.64350i$	$11.79363 \pm 1.65233i$
$13.99236 \pm 2.51883i$	$16.73074 \pm 2.81262i$	$19.50244 \pm 1.94033i$	



Problemas mal condicionados

Si pequeñas variaciones de los datos provocan grandes variaciones en la solución, se dice que el problema está mal condicionado.

Ejemplo. (Lo haremos en la práctica.)

Resolver los sistemas

$$\begin{cases} 2x - 4y & = 1 \\ -2.998x + 6.001y & = 2 \end{cases} \qquad \begin{cases} 2x - 4y & = 1 \\ -2.998x + 6y & = 2 \end{cases}$$

Número de condición

El número de condición de una función mide cuánto se modifica el valor de salida al realizar un pequeño cambio en el valor de entrada.

$$|f(x) - f(x^*)|$$

Número de condición

El número de condición de una función mide cuánto se modifica el valor de salida al realizar un pequeño cambio en el valor de entrada.

$$|f(x) - f(x^*)|$$

- Los n valores

$$\left| \frac{\widetilde{x}_i}{f(\widetilde{x})} \frac{\partial f(\widetilde{x})}{\partial x_i} \right|$$

  se denominan números de condición o factores de propagación. Estos proporcionan una medida de cuán mal condicionado es un problema.

Número de condición

El número de condición de una función mide cuánto se modifica el valor de salida al realizar un pequeño cambio en el valor de entrada.

$$|f(x) - f(x^*)|$$

- Los n valores

$$\left| \frac{\widetilde{x}_i}{f(\widetilde{x})} \frac{\partial f(\widetilde{x})}{\partial x_i} \right|$$

  se denominan números de condición o factores de propagación. Estos proporcionan una medida de cuán mal condicionado es un problema.

- Para sistemas lineales $Ax = b$, el número de condición de una matriz es:

$$\text{Cond}(A) = \|A\| \cdot \|A^{-1}\|$$

  Se puedes usar distintas normas matriciales. $\text{Cond}(A)$ es siempre un número mayor que 1, y el sistema $Ax = b$ estará tanto mejor condicionado cuanto más próximo a 1 sea el número de condición.

Estabilidad numérica: intuición

Un algoritmo lo clasificaremos como numéricamente estable si un error no crece *mucho* en el proceso de cálculo.

La estabilidad numérica se ve afectada por el número de cifras significativas; pocas cifras o la pérdida en pasos intermedios del cálculo disminuye la fiabilidad de los resultados obtenidos.

Estabilidad numérica: intuición

Un algoritmo lo clasificaremos como numéricamente estable si un error no crece *mucho* en el proceso de cálculo.

La estabilidad numérica se ve afectada por el número de cifras significativas; pocas cifras o la pérdida en pasos intermedios del cálculo disminuye la fiabilidad de los resultados obtenidos.

Ejemplo (ejercicio hecho en Matlab).

$$f(x) = \sqrt{x^2+1} - 1,$$
$$g(x) = \frac{x^2}{\sqrt{x^2+1}+1}.$$

Aunque $f(x) = g(x)$, al evaluarlas en 8^{-n}, $1 \leq n \leq 13$ los resultados que da matlab son distintos.

k	8^(-k)	f(8^-k)	g(8^-k)
1	0.125	0.0077822	0.0077822
2	0.015625	0.00012206	0.00012206
3	0.0019531	1.9073e-06	1.9073e-06
4	0.00024414	2.9802e-08	2.9802e-08
5	3.0518e-05	4.6566e-10	4.6566e-10
6	3.8147e-06	7.276e-12	7.276e-12
7	4.7684e-07	1.1369e-13	1.1369e-13
8	5.9605e-08	1.7764e-15	1.7764e-15
9	7.4506e-09	0	2.7756e-17
10	9.3132e-10	0	4.3368e-19
11	1.1642e-10	0	6.7763e-21
12	1.4552e-11	0	1.0588e-22
13	1.819e-12	0	1.6544e-24

Inestabilidad numérica

Sin rigor, decimos que un proceso numérico es inestable cuando los pequeños errores que se producen en una de sus etapas se agrandan en etapas posteriores, hasta tal punto que no podemos confiar en el cálculo global.

Inestabilidad numérica

Sin rigor, decimos que un proceso numérico es inestable cuando los pequeños errores que se producen en una de sus etapas se agrandan en etapas posteriores, hasta tal punto que no podemos confiar en el cálculo global.

Ejemplo. (Parte de los ejercicios a hacer.)

Para calcular las integrales $I_n = \int_0^1 x^n e^{x-1}\,dx, \quad n \geq 1$, disponemos de dos métodos iterativos diferentes:

a) $I_{n-1} = \dfrac{1 - I_n}{n}, \qquad n \geq 2$ donde $I_{50} = 0$,

b) $I_n = 1 - nI_{n-1}, \qquad n \geq 2$ donde $I_1 = \dfrac{1}{e}$.

Inestabilidad numérica

Sin rigor, decimos que un proceso numérico es inestable cuando los pequeños errores que se producen en una de sus etapas se agrandan en etapas posteriores, hasta tal punto que no podemos confiar en el cálculo global.

Ejemplo. (Parte de los ejercicios a hacer.)

Para calcular las integrales $I_n = \int_0^1 x^n e^{x-1}\, dx, \quad n \geq 1$, disponemos de dos métodos iterativos diferentes:

a) $I_{n-1} = \dfrac{1 - I_n}{n}, \qquad n \geq 2$ donde $I_{50} = 0$,

b) $I_n = 1 - nI_{n-1}, \qquad n \geq 2$ donde $I_1 = \dfrac{1}{e}$.

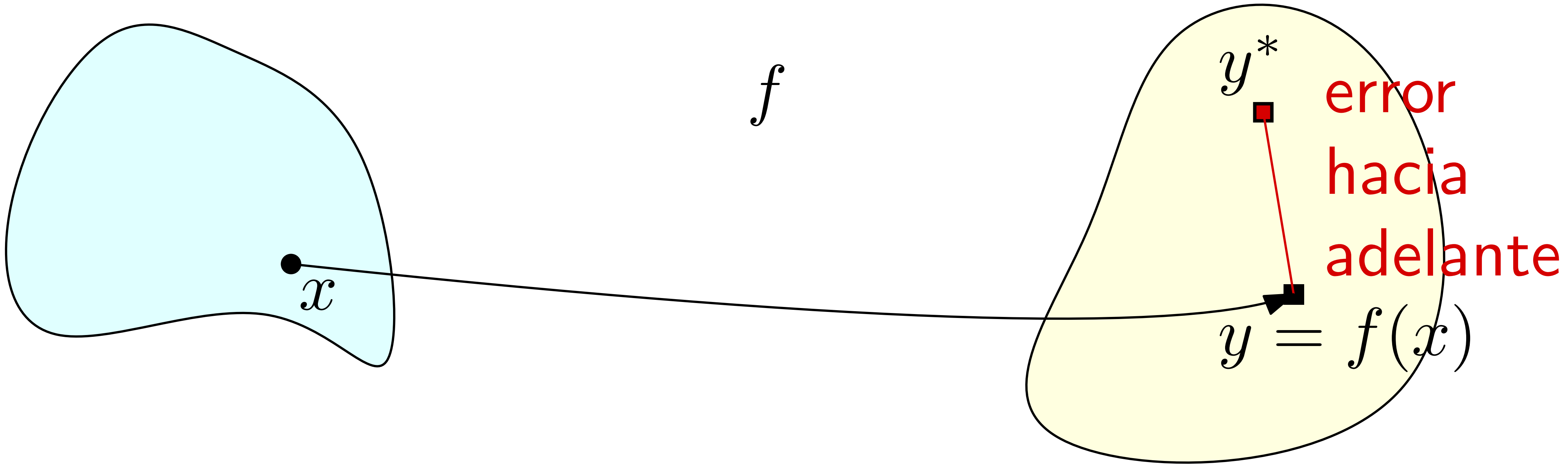
$$I_n = \int_0^1 x^n e^{x-1}\, dx = x^n e^{x-1}\Big|_0^1 - \int_0^1 n x^{n-1} e^{x-1}\, dx = 1 - nI_{n-1}$$

Estabilidad numérica

La definición de estabilidad numérica depende del contexto: álgebra lineal, ecuaciones diferenciales,. . .

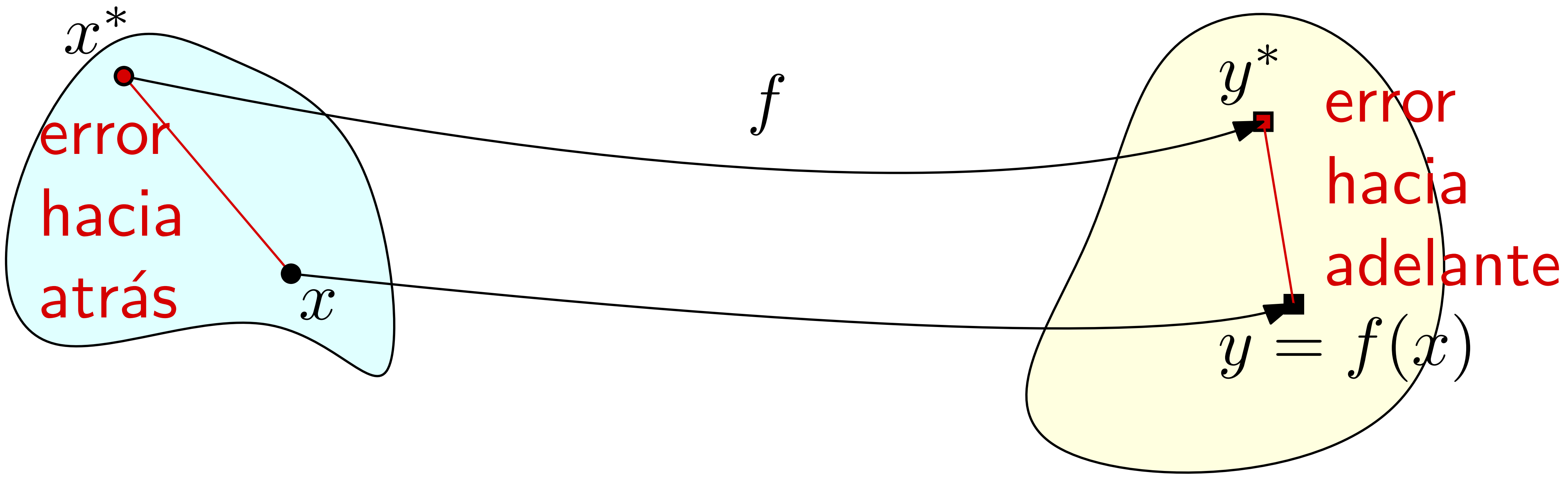
Estabilidad numérica

La definición de estabilidad numérica depende del contexto: álgebra lineal, ecuaciones diferenciales,. . .



Estabilidad numérica

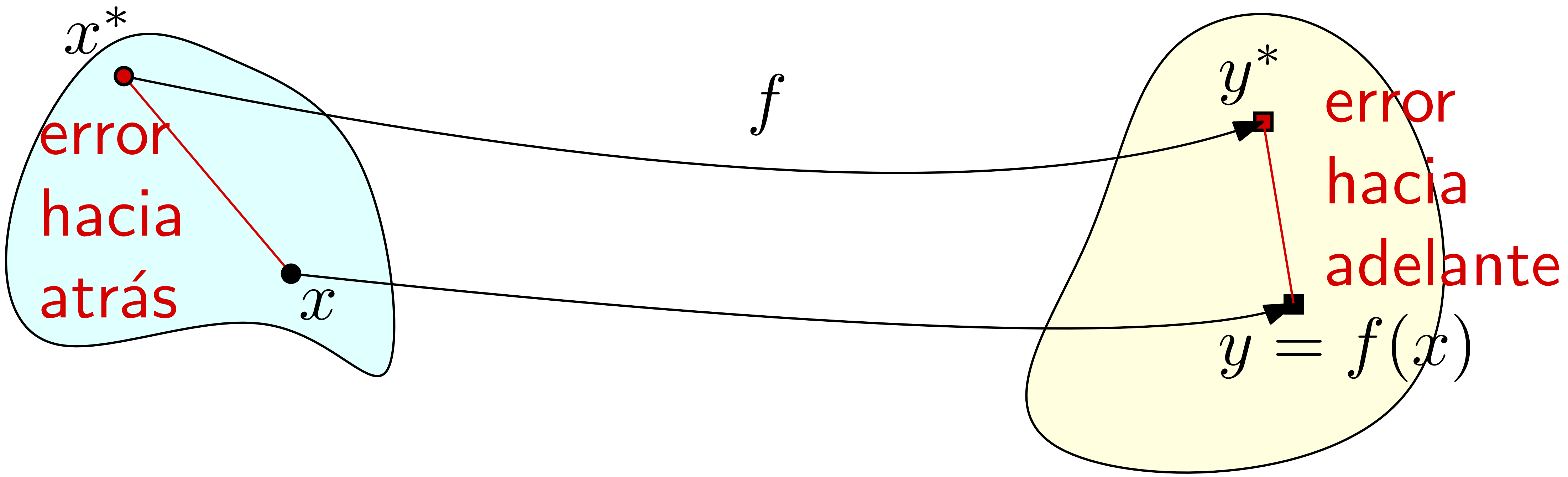
La definición de estabilidad numérica depende del contexto: álgebra lineal, ecuaciones diferenciales,. . .



Estabilidad numérica

La definición de estabilidad numérica depende del contexto: álgebra lineal, ecuaciones diferenciales,. . .

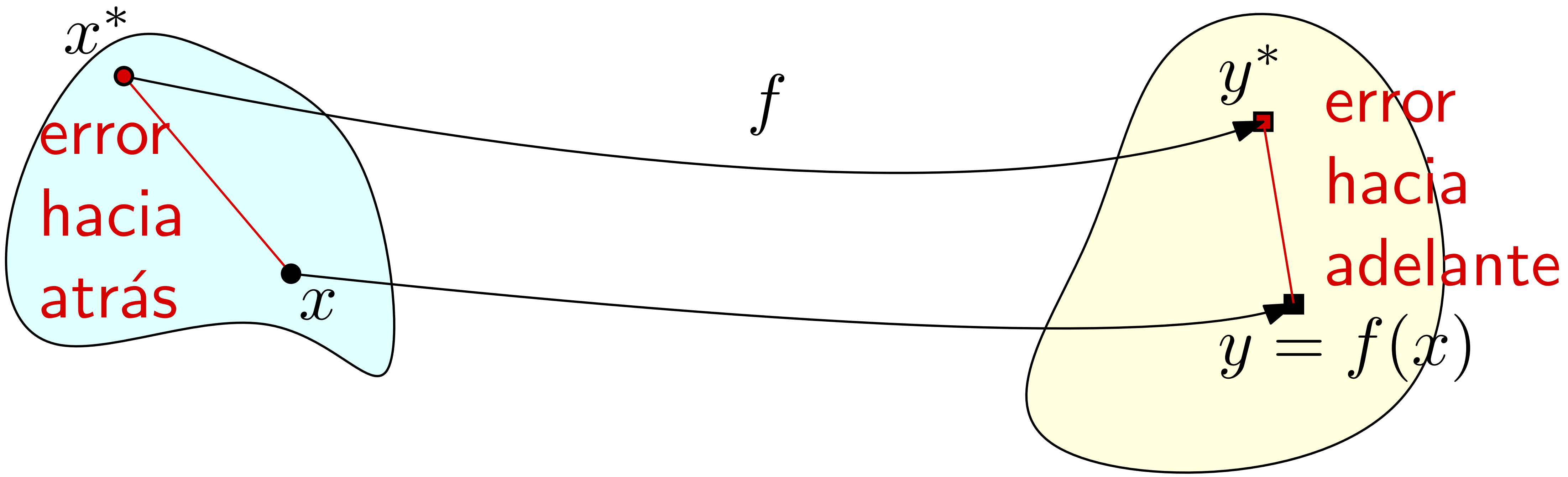
x^* es el punto más cercano a x tal que $f(x^*) = y^*$



Estabilidad numérica

La definición de estabilidad numérica depende del contexto: álgebra lineal, ecuaciones diferenciales,...

x^* es el punto más cercano a x tal que $f(x^*) = y^*$

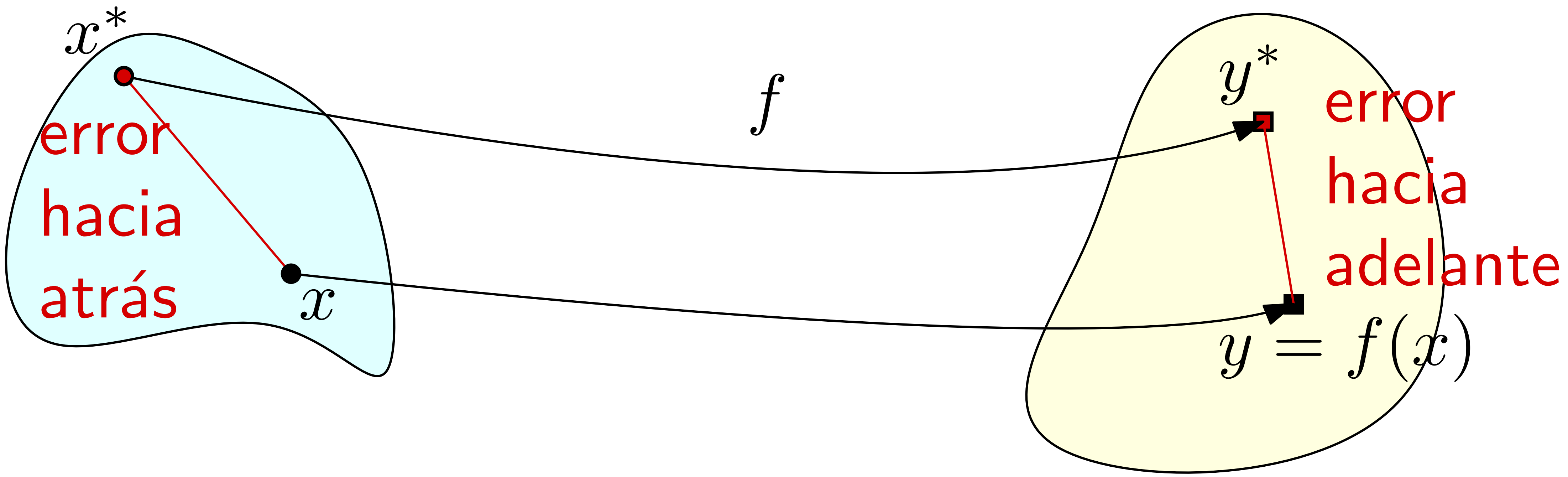


- Un algoritmo es estable hacia adelante si su error hacia adelante dividido por el número de condición del problema es pequeño.

Estabilidad numérica

La definición de estabilidad numérica depende del contexto: álgebra lineal, ecuaciones diferenciales,...

x^* es el punto más cercano a x tal que $f(x^*) = y^*$

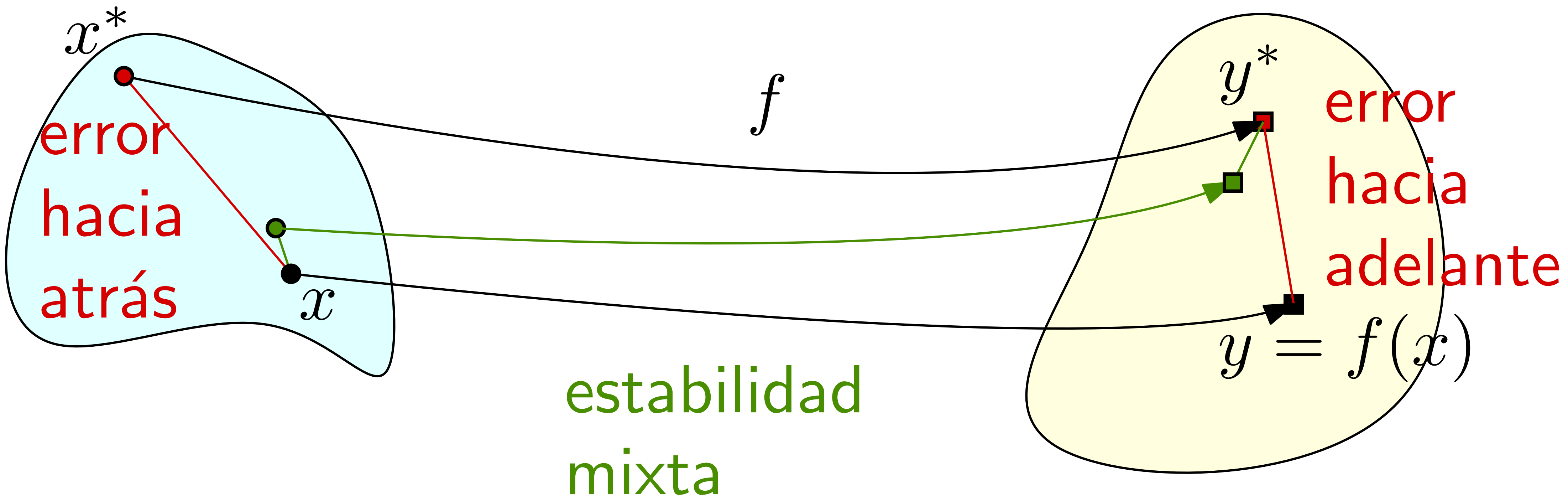


- Un algoritmo es estable hacia adelante si su error hacia adelante dividido por el número de condición del problema es pequeño.
- Se dice que un algoritmo es estable hacia atrás si el error hacia atrás es pequeño para todo x (respecto a la unidad de redondeo). Tal algoritmo eventualmente calcula la solución exacta para un problema cercano.

Estabilidad numérica

La definición de estabilidad numérica depende del contexto: álgebra lineal, ecuaciones diferenciales,...

x^* es el punto más cercano a x tal que $f(x^*) = y^*$



- Un algoritmo es estable hacia adelante si su error hacia adelante dividido por el número de condición del problema es pequeño.
- Se dice que un algoritmo es estable hacia atrás si el error hacia atrás es pequeño para todo x (respecto a la unidad de redondeo). Tal algoritmo eventualmente calcula la solución exacta para un problema cercano.

Algoritmos con cancelación

La pérdida de cifras significativas por cancelación se produce al restar dos números muy cercanos. La situación se puede resumir en

$$g(x+\delta) - g(x) \quad \text{con } |\delta| \ll 1;$$

Algoritmos con cancelación

La pérdida de cifras significativas por cancelación se produce al restar dos números muy cercanos. La situación se puede resumir en

$$g(x+\delta) - g(x) \ \text{ con } |\delta| \ll 1;$$

Ejemplo.
Las soluciones de $x^2 - 18x + 1 = 0$ son $x_{1,2} = 9 \pm \sqrt{80}$.
Si $\sqrt{80} = 8.9443 \pm 0.5 \cdot 10^{-4}$ entonces $x_1 = 17.9443 \pm 0.5 \cdot 10^{-4}$ tiene 6 cifras significativas, mientras que $x_2 = 0.0557 \pm 0.5 \cdot 10^{-4}$ solo tiene 3.

Propagación de los errores en coma flotante

Con el fin de reducir o evitar la propagación de errores, se recomienda minimizar el número de operaciones, reordenar las operaciones y replantear el problema en otros términos.

Propagación de los errores en coma flotante

Con el fin de reducir o evitar la propagación de errores, se recomienda minimizar el número de operaciones, reordenar las operaciones y replantear el problema en otros términos.

Ejemplo. Resolver la ecuación $x^2 + 62.10x + 1 = 0$ trabajando con cuatro dígitos y redondeando.

Propagación de los errores en coma flotante

Con el fin de reducir o evitar la propagación de errores, se recomienda minimizar el número de operaciones, reordenar las operaciones y replantear el problema en otros términos.

Ejemplo. Resolver la ecuación $x^2 + 62.10x + 1 = 0$ trabajando con cuatro dígitos y redondeando.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Donde los coeficientes son $a = 1$, $b = 62.10$, y $c = 1$. Sustituyendo estos valores:

$$x = \frac{-62.10 \pm \sqrt{(62.10)^2 - 4}}{2} = \frac{-62.10 \pm 62.06}{2} \qquad \begin{matrix} x_1 = 62.08 \\ x_2 = 0.02 \end{matrix}$$

Fórmula mejorada para ecuaciones de segundo grado

Consideramos la ecuación $ax^2 + bx + c = 0$ y supongamos que $a \neq 0$ y que $b^2 - 4ac > 0$. Las raíces pueden calcularse por medio de la conocida fórmula:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \qquad \text{(i)}$$

Fórmula mejorada para ecuaciones de segundo grado

Consideramos la ecuación $ax^2 + bx + c = 0$ y supongamos que $a \neq 0$ y que $b^2 - 4ac > 0$. Las raíces pueden calcularse por medio de la conocida fórmula:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \qquad \text{(i)}$$

Ejercicio. comprobar que estas raíces pueden calcularse mediantes estas fórmulas equivalentes:

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \qquad x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}} \qquad \text{(ii)}$$

Fórmula mejorada para ecuaciones de segundo grado

Consideramos la ecuación $ax^2 + bx + c = 0$ y supongamos que $a \neq 0$ y que $b^2 - 4ac > 0$. Las raíces pueden calcularse por medio de la conocida fórmula:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \qquad \text{(i)}$$

Ejercicio. comprobar que estas raíces pueden calcularse mediantes estas fórmulas equivalentes:

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \qquad x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}} \qquad \text{(ii)}$$

Cuando $b \approx \sqrt{b^2 - 4ac}$ hay que proceder con cuidado para evitar la pérdida de precisión por cancelación.

Fórmula mejorada para ecuaciones de segundo grado

Consideramos la ecuación $ax^2 + bx + c = 0$ y supongamos que $a \neq 0$ y que $b^2 - 4ac > 0$. Las raíces pueden calcularse por medio de la conocida fórmula:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \qquad \text{(i)}$$

Ejercicio. comprobar que estas raíces pueden calcularse mediantes estas fórmulas equivalentes:

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \qquad x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}} \qquad \text{(ii)}$$

Cuando $b \approx \sqrt{b^2 - 4ac}$ hay que proceder con cuidado para evitar la pérdida de precisión por cancelación.

Si $b > 0$: usar (ii) para x_1 y (i) para x_2. Si $b < 0$: usar (i) para x_1 y (ii) para x_2.

Regla de Horner

Ejemplo. Evaluar el polinomio $P(x) = x^3 - 6.1x^2 + 3.2x + 1.5$ en $x = 4.71$ utilizando una aritmética de tres dígitos.

Regla de Horner

Ejemplo. Evaluar el polinomio $P(x) = x^3 - 6.1x^2 + 3.2x + 1.5$ en $x = 4.71$ utilizando una aritmética de tres dígitos.

$$P(4.71) = 104 - 135 + 15.1 + 1.5 = -14.4$$

Regla de Horner

Ejemplo. Evaluar el polinomio $P(x) = x^3 - 6.1x^2 + 3.2x + 1.5$ en $x = 4.71$ utilizando una aritmética de tres dígitos.

$$P(4.71) = 104 - 135 + 15.1 + 1.5 = -14.\boxed{4}$$

Resultado correcto: -14.2639.

Regla de Horner

Ejemplo. Evaluar el polinomio $P(x) = x^3 - 6.1x^2 + 3.2x + 1.5$ en $x = 4.71$ utilizando una aritmética de tres dígitos.

$$P(4.71) = 104 - 135 + 15.1 + 1.5 = -14.\boxed{4}$$

Resultado correcto: -14.2639.

Regla de Horner

Para evaluar el polinomio $P(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ en $x = c$ la regla de Horner es:

$$P(c) = (((((a_n \cdot c + a_{n-1}) \cdot c + a_{n-2}) \cdot c + \ldots) \cdot c + a_1) \cdot c + a_0.$$

Evitar la propagación de los errores

Con el fin de reducir o evitar la propagación de errores, se recomienda minimizar el número de operaciones, reordenar las operaciones y replantear el problema en otros términos.

- Evitar restas de cantidades muy próximas.
- Evitar la división por cantidades pequeñas.
- Sumar en orden creciente de valor absoluto.
- Usar la regla de Horner.
- ...

Ejercicios

Operaciones en coma flotante

En una aritmética de cinco dígitos, representar los números $x = \frac{1}{3}$ e $y = \frac{5}{7}$ y calcular:

a) $x \times y$, $x \otimes y$.

b) $x + y$, $x \oplus y$.

c) $x - y$, $x \ominus y$.

d) $x \div y$, $x \oslash y$.

Comprobar que el error se mantiene por debajo de 0.5×10^{-4}.

Respuestas: Operaciones en coma flotante

Si $\mathrm{fl}(1/3) = 0.33333$ y $\mathrm{fl}(5/7) = 0.71428$ entonces:

a) $x \times y = \frac{5}{21}$ y $x \otimes y = \mathrm{fl}(0.2380909524) = 0.23809$
y los errores son $e_a = 0.5 \times 10^{-5}$ y $e_r = 0.2 \times 10^{-4}$.

b) $x + y = \frac{22}{21}$ y $x \oplus y = \mathrm{fl}(1.04761) = 1.04761$
y los errores son $e_a = 0.9 \times 10^{-5}$ y $e_r = 0.9 \times 10^{-5}$.

c) $x - y = -\frac{8}{21}$ y $x \ominus y = \mathrm{fl}(-0.38095) = -0.38095$
y los errores son $e_a = 0.2 \times 10^{-5}$ y $e_r = 0.6 \times 10^{-5}$.

d) $x \div y = \frac{7}{15}$ y $x \oslash y = \mathrm{fl}(0.466665733325867) = 0.46666$
y los errores son $e_a = 0.7 \times 10^{-5}$ y $e_r = 0.1 \times 10^{-4}$.

Problemas con operaciones

En una aritmética de cinco dígitos, representar los números $y = \frac{5}{7}$, $u = 0.714251$, $v = 98765.9$ y $w = 0.111111 \times 10^{-4}$ y calcular:

a) $y \ominus u$. (restar dos cantidades muy cercanas)

b) $(y \ominus u) \oslash w$. (dividir por una cantidad pequeña)

c) $(y \ominus u) \otimes w$. (multiplicar por una cantidad grande)

d) $u \oplus v$.

e) $y \ominus w$.

Comprobar que el error NO se mantiene por debajo de 0.5×10^{-4}.

Respuestas: Problemas con operaciones

$$
\begin{aligned}
y &= 5/7 & &\Rightarrow \mathsf{fl}(y) = 0.71428 \\
u &= 0.714251 & &\Rightarrow \mathsf{fl}(u) = 0.71425 \\
v &= 98765.9 & &\Rightarrow \mathsf{fl}(v) = 0.98765 \times 10^5 \\
w &= 0.111111 \times 10^{-4} & &\Rightarrow \mathsf{fl}(w) = 0.11111 \times 10^{-4}
\end{aligned}
$$

Operation	e_a	e_r
$y \ominus u$	0.472×10^{-5}	0.136
$(y \ominus u) \oslash w$	0.425	0.136
$(y \ominus u) \otimes w$	0.466	0.136
$u \oplus v$	0.162×10^1	0.164×10^{-4}
$y \ominus w$	0.779	0.122×10^{-4}

Autoavaluación

Ejercicio 1. Realizar las operaciones aritméticas:

$$\frac{4}{5} + \frac{1}{3}; \quad \left(\frac{1}{3} - \frac{3}{11}\right) + \frac{3}{20}; \quad \left(\frac{1}{3} + \frac{3}{11}\right) - \frac{3}{20};$$

a) Haciendo uso de una aritmética de tres dígitos y truncando los números.
b) Haciendo uso de una aritmética de tres dígitos y redondeando los números.
c) Calcular los errores relativos de los apartados a) y b).

Autoavaluación

Ejercicio 2. Calcular, respetando el orden de los sumandos:

$$\sum_{k=1}^{6} \frac{1}{3^k} \text{ y } \sum_{k=1}^{6} \frac{1}{3^{(7-k)}}$$

a) Haciendo uso de la aritmética de tres dígitos y redondeando.

b) Haciendo uso de la aritmética de cuatro dígitos y redondeando.

c) ¿Por qué dan resultados diferentes? Calcular en cada caso el error relativo porcentual.

Guia de estudio

Libro *Càlcul numèric: teoria i pràctica* de M. Grau Sánchez y M. Noguera Batlle

- Conceptos asociados: capítulo 1, de la página 2 a la 30.
- Problemas propuestos: 2 y 9.

Libro *Cálculo numérico* de M. Grau Sánchez, y M. Noguera Batlle

- Conceptos asociados: capítulo 1, de la página 13 a la 53.
- Problemas propuestos: 2 y 9.

Otros libros de consulta

- *Cálculo Científico con MATLAB y Octave* de A. Quarteroni y F. Saleri.
- *Métodos Numéricos con MATLAB* de J. H. Mathews y K. D. Fink.
- *Numerical Computing with MATLAB* de C. Moler.