Aritmética finita y análisis de error

Escuela de Ingeniería Informática de Oviedo

Contenidos

Tratamiento matemático de un problema físico

En este curso presentaremos los **métodos numéricos** básicos que resuelven un conjunto de problemas matemáticos clásicos.

Los **ordenadores** son una herramienta necesaria en el uso eficiente de los métodos numéricos.

Por lo tanto, en la clase de hoy, veremos **como los números**, que pueden tener infinitos dígitos, **se almacenan en el ordenador**, que es un dispositivo finito.

Esto nos lleva a tener en cuenta los **errores**, como definirlos y medirlos.

Outline

Almacenamiento de números

Los números se almacenan en los ordenadores en los siguientes formatos

- Entero
  Permite el almacenamiento exacto de
  - un conjunto de números enteros

- En punto flotante
  Permite el almacenamiento exacto de
  - un conjunto de números enteros
  - un conjunto de números no enteros

El formato usado más habitualmente es formato IEEE 754

Representación en punto flotante: decimal

La representación en punto flotante de un número real en base 10 $x \neq 0$ es

$$x = \sigma \times \bar{x} \times 10^e,$$

donde

- $\sigma = \pm 1$, el signo,
- $1 \leq \bar{x} < 10$, la mantisa,
- $e \in \mathbb{Z}$, el exponente

El número de dígitos en $\bar{x}$ es la precisión de la representación.

Ejemplo: Para la representación exacta en punto flotante decimal podemos escribir $x = 314.15 = 3.1415 \times 10^2$, y entonces

$$\sigma = +1, \quad \bar{x} = 3.1415, \quad e = 2.$$

que tiene una precisión de 5 dígitos.

Representación en punto flotante: binaria

La representación en punto flotante de un número real en base 2 $x \neq 0$ es

$$x = \sigma \times \bar{x} \times 2^e,$$

donde

- $\sigma = \pm 1$, el signo,
- $(1)_2 \leq \bar{x} < (10)_2$, la mantisa,
- $e \in \mathbb{Z}$, el exponente

Ejemplo: If $x = (10101.11001)_2 = (1.010111001)_2 \times 2^4$ entonces

$$\sigma = +1, \quad \bar{x} = (1.010111001)_2, \quad e = (4)_{10} = (100)_2.$$

Ejemplo

Para $x = (101.001101)_2 = (5.203125)_{10}$ tenemos

- representación decimal en punto flotante:

$$\sigma = +1, \quad \bar{x} = 5.203125, \quad e = 0,$$

- representación binaria en punto flotante:

$$\sigma = (1)_2, \quad \bar{x} = (1.0100)_2, \quad e = (2)_{10} = (10)_2.$$

Paso del sistema decimal a binario y viceversa

En el sistema decimal el número 107.625 significa:

$$107.625 = 1 \cdot 10^2 + 7 \cdot 10^0 + 6 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3}.$$

Los ordenadores usan el sistema binario. Sólo se almacenan 0 y 1. En el sistema binario, los números representan potencias de 2:

$$(107.625)_{10} = 2^6 + 2^5 + 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-3} = (1101011.101)_2$$

Y el paso de binario a decimal es directo:

$$(1101011.101)_2 = 2^6 + 2^5 + 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-3} = (107.625)_{10}$$

Paso del sistema decimal a binario y viceversa

Decimal a binario:

- Parte entera: Dividimos sucesivamente por 2 . Los restos son los dígitos en base 2, de menos a más significativos:

Cocientes	107	53	26	13	6	3	1	0
Restos		1	1	0	1	0	1	1

- Para convertir la parte fraccionaria dividimos por 2, restamos la parte entera y repetimos 1:

Decimal	0.625	0.25	0.5	1
Entera		1	0	1

El resultado es:

$$(107.625)_{10} = (1101011.101)_2.$$

Ejemplo: representación de enteros con 4 bits

Representación binaria ($m = 4$ bits)	Enteros sin signo	Enteros con signo (signo en 1^{er} bit)	Enteros con signo $sesgo = 2^{m-1}$	Enteros con signo (Expo.) $sesgo = 2^{m-1} - 1$
0000	0	$+0$	-8	Reservado
0001	1	$+1$	-7	-6
0010	2	$+2$	-6	-5
0011	3	$+3$	-5	-4
0100	4	$+4$	-4	-3
0101	5	$+5$	-3	-2
0110	6	$+6$	-2	-1
0111	7	$+7$	-1	0
1000	8	-0	0	1
1001	9	-1	1	2
1010	10	-2	2	3
1011	11	-3	3	4
1100	12	-4	4	5
1101	13	-5	5	6
1110	14	-6	6	7
1111	15	-7	7	Reservado

Outline

La norma IEEE 754

IEEE significa Institute of Electrical and Electronics Engineers.
El estandar IEEE 754 es la representación en punto flotante binaria y es el usado por casi todos los procesadores.

Precisión simple:

$$x = \sigma \times (1.a_1 a_2 \dots a_{23}) \times 2^e$$

Utiliza 32 bits (4 bytes) distribuídos:

- 1 bit para el signo.
- 8 bits para el exponente.
- 23 bits para la mantisa.

Tiene una precisión de 24 dígitos binarios.
El exponente toma valores en $[-126, 127]$ con sesgo 127.

La norma IEEE 754

Precisión doble:

$$x = \sigma \times (1.a_1 a_2 \ldots a_{52}) \times 2^e$$

Utiliza 64 bits (8 bytes) distribuídos:

- 1 bit para el signo.
- 11 bits para el exponente.
- 52 bits para la mantisa.

Tiene una precisión de 53 dígitos binarios. El exponente toma valores en $[-1022, 1023]$ con sesgo 1023.

IEEE 754. Precisión simple

El número

$$x = \sigma \times (1.a_1 a_2 \ldots a_{23}) \times 2^e, \quad \text{con } e \in [-126, 127]$$

se almacena usando 32 bits (4 bytes):

$$b_1 b_2 \ldots b_9 b_{10} \ldots b_{32}$$

Aquí

- $b_1 = \begin{cases} 0 & \text{si } \sigma = +1, \\ 1 & \text{si } \sigma = -1. \end{cases}$
- $b_2 \ldots b_9$ para almacenar $E = e + 127 (> 0)$,
- $b_{10} \ldots b_{32}$ para almacenar la parte decimal, m de la mantisa $\bar{x}$. El entero parte de $\bar{x}$ es 1 siempre que $x \neq 0$ o un número no normalizado.

	$E = 0$	$0 < E < (255)_{10}$	$E = (255)_{10}$
$m = 0$	0	$(-1)^\sigma \times 2^{E-127}$	$\pm\infty$
$m \neq 0$	números no normalizados		*NaN*

IEEE 754. Doble precisión

El número

$$x = \sigma \times (1.a_1 a_2 \ldots a_{52}) \times 2^e, \quad \text{with } e \in [-1022, 1023]$$

se almacena usando 64 bits (8 bytes):

$$b_1 b_2 \ldots b_{12} b_{13} \ldots b_{64}$$

Aquí

- $b_1 = \begin{cases} 0 & \text{si } \sigma = +1, \\ 1 & \text{si } \sigma = -1. \end{cases}$
- $b_2 \ldots b_{12}$ para almacenar $E = e + 1023 (> 0)$,
- $b_{13} \ldots b_{64}$ para almacenar la parte decimal, m de la mantisa $\bar{x}$. La parte entera de $\bar{x}$ es 1 siempre que $x \neq 0$ o un número no normalizado.

	$E = 0$	$0 < E < (2047)_{10}$	$E = (2047)_{10}$
$m = 0$	0	$(-1)^\sigma \times 2^{E-1023}$	$\pm\infty$
$m \neq 0$	números no normalizados		*NaN*

Ejemplo. De base 10 a binario IEEE 754

Vamos a convertir 10.25 en base 10 a binario IEEE 754 en precisión simple. Los pasos son:

1. Convertimos la parte entera a base 2: $(10)_{10} = (1010)_2$.
2. Convertimos los decimales a base 2: $(.25)_{10} = (.01)_2$.
3. Los sumamos: $1010 + 0.01 = 1010.01$.
4. Lo escribimos en binario normalizado: 1.01001×2^3.
5. Convertimos el 3 añadiéndole el sesgo correspondiente. En este caso 127. Por lo tanto tenemos $127 + 3 = 130$, que convertimos a binario 1000 0010.
6. Escribimos el número en el orden (signo exponente mantisa)

$$0\ 1000\ 0010\ 0100\ 1000\ 0000\ 0000\ 0000\ 000$$

Fijarse que el bit escondido "1" no está representado.

Outline

Exactitud en punto flotante

Queremos medir la exactitud del almacenamiento en punto flotante.

Las medidas habituales son:

- El epsilon de la máquina: Es la diferencia entre 1 y el número siguiente $x > 1$ que se puede almacenar de forma exacta.

- El entero más grande: Es el entero más grande M tal que todos los enteros x, donde $0 \leq x \leq M$, se almacenan de la misma forma.

Exactitud en punto flotante: el epsilon de la máquina

- Precición simple IEEE :

$$\epsilon = (00\underbrace{\ldots}_{22}01)_2 = 2^{-23} \approx 1.19 \times 10^{-7},$$

y podemos almacenar aproximadamente 7 dígitos de un número en base 10.

- Precisión doble IEEE:

$$\epsilon = (00\underbrace{\ldots}_{51}01)_2 = 2^{-52} \approx 2.22 \times 10^{-16},$$

y podemos almacenar aproximadamente 16 dígitos de un número en base 10.

Exactitud en punto flotante: el entero más grande

Si n es el número de dígitos binarios en la mantisa, entonce, el entero más grande es

$$M = 2^n$$

porque

- Todos los enteros x con

$$0 \leq x \leq (11\underbrace{\ldots}_{n-1}1)_2 \times 2^{n-1} = 2^{n-1} + 2^{n-2} + \ldots + 2^1 + 2^0 = \frac{2^n - 1}{2 - 1} = 2^n - 1,$$

  se pueden almacenar de forma exacta.

- $x = (10\underbrace{\ldots}_{n-1}0)_2 \times 2^n = 2^n$ se puede almacenar de forma exacta.

- $x = (10\underbrace{\ldots}_{n-1}0)_2 \times 2^n + 1$ no se puede almacenar de forma exacta.

Exactitud en punto flotante: el entero más grande

- Precisión simple IEEE:

$$M = 2^{24} = 1677216,$$

y podemos almacenar los enteros con 7 dígitos.

- Precisión doble IEEE:

$$M = 2^{53} \approx 9.0 \times 10^{15},$$

y podemos almacenar todos los enteros con 15 dígitos y casi todos los enteros con 16 dígitos.

Exactitud IEEE

¿Qué sucede si intentamos almacenar un número mayor que M?

Error de overflow:

Un error de overflow se produce cuando intentamos usar un número demasiado grande.

- En la mayor parte de los ordenadores se aborta la ejecución.
- El formato IEEE puede darle soporte asignándole los valores simbólicos $\pm\infty$ or NaN.
- A menudo, se debe a errores de programación, que deben ser corregidos.

Exactitud IEEE

¿Y qué sucede si intentamos almacenar un número más pequeño que ϵ?

Error de underflow:

Un error de underflow se produce cuando intentamos usar un número demasiado pequeño:

- En casi todos los ordenadores, se identifica con 0, y la ejecución continua.

Outline

Redondeo decimal

Escribamos un número x con notación en punto flotante en base 10 como

$$x = \pm d_0.d_1 d_2 \ldots \times 10^n = \pm \left(\sum_{k=0}^{\infty} d_k 10^{-k} \right) \times 10^n,$$

donde $d_0 \neq 0$ y $0 \leq d_k \leq 9$, para $k = 1, 2, \ldots$

Si la mantisa tiene más de p dígitos decimales, es decir,

$$d_k \neq 0 \quad \text{para algunos} \quad k > p - 1,$$

entonces x no tiene un representación en punto flotante exacta con precisión p.

En esta situación, se produce el redondeo.

Redondeo decimal

Dos formas habituales de redondeo

$$x = \pm d_0.d_1 d_2 \ldots \times 10^n = \pm \left(\sum_{k=0}^{\infty} d_k 10^{-k} \right) \times 10^n :$$

- Redondeo a cero o truncado con $p+1$ dígitos:

$$x^* = \pm d_0.d_1 d_2 \ldots d_p \times 10^p,$$

- Redondeo al (par) más cercano con p dígitos:

$$x^* = \begin{cases} \pm d_0.d_1 d_2 \ldots d_{p-1} \times 10^n & \text{si } 0 \leq d_p \leq 4, \\ \pm (d_0.d_1 d_2 \ldots d_{p-1} + 10^{-p+1}) \times 10^n & \text{si } 5 \leq d_p \leq 9, \\ \text{al número acabado en par más cercano} & \text{si } d_p = 5 \text{ y } d_{p+k} = 0. \end{cases}$$

Ejemplos redondeo decimal

Ejemplo: Para $x = 0.999953$ y $p = 4$

- Truncando $x^* = 0.9999$.
- Redondeando $x^* = 1.000$.

Ejemplo: Para $x = 0.433309$ y $p = 3$

- Truncando $x^* = 0.433$.
- Redondeando $x^* = 0.433$.

Ejemplo: Para $x = 0.433500$ y $p = 3$

- Truncando $x^* = 0.433$.
- Redondeando $x^* = 0.434$. (Hacia el n^o acabado en par más cercano)

Ejemplo: Para $x = 0.434500$ y $p = 3$

- Truncando $x^* = 0.434$.
- Redondeando $x^* = 0.434$. (Hacia el n^o acabado en par más cercano)

Redondeo binario

Escribamos un número x en punto flotante en base dos como

$$x = \pm(1.d_1 d_2 \ldots)_2 \times 2^e = \pm\left(1 + \sum_{k=1}^{\infty} d_k 2^{-k}\right) \times 2^e,$$

donde $0 \leq d_k \leq (1)_2$, para $k = 1, 2, \ldots$

Si la mantisa tiene más de $p+1$ dígitos binarios, es decir,

$$d_k \neq 0 \quad \text{para algunos} \quad k > p-1,$$

entonces x no tiene un representación en punto flotante exacta con precisión $p+1$.

Otra vez, se produce redondeo.

Redondeo binario

Dos formas de redondear

$$x = \pm(1.d_1 d_2 \ldots)_2 \times 2^e = \pm\left(1 + \sum_{k=1}^{\infty} d_k 2^{-k}\right) \times 2^e :$$

- Redondeo a cero o truncado con p dígitos:

$$x^* = \pm(1.d_1 d_2 \ldots d_{p-1})_2 \times 2^e,$$

- Redondeo al (par) más cercano con p dígitos:

$$x^* = \begin{cases} \pm(1.d_1 d_2 \ldots d_{p-1})_2 \times 2^e & \text{si } d_p = 0, \\ \pm((1.d_1 d_2 \ldots d_{p-1})_2 + 2^{-p+1}) \times 2^e & \text{si } d_p = 1, \\ \text{al número acabado en cero más cercano} & \text{si } d_p = 1 \text{ y } d_{p+k} = 0. \end{cases}$$

Ejemplos redondeo binario

Ejemplo: Para $x = 1.1111$ y $p = 3$

- Truncando $x^* = 1.11$.
- Redondeando $x^* = 10.0$.

Ejemplo: Para $x = 1.1101$ y $p = 3$

- Truncando $x^* = 1.11$.
- Redondeando $x^* = 1.11$.

Ejemplo: Para $x = 1.0010$ y $p = 3$

- Truncando $x^* = 1.00$.
- Redondeando $x^* = 1.00$. (Hacia el n^o acabado en cero más cercano)

Ejemplo: Para $x = 1.0110$ y $p = 3$

- Truncando $x^* = 1.01$.
- Redondeando $x^* = 1.10$. (Hacia el n^o acabado en cero más cercano)

Ejemplo: Para $x = 1.1101$ y $p = 3$

- Truncando $x^* = 1.11$.
- Redondeando $x^* = 1.11$.

Comparación entre truncado y redondeo en binario

La representación en punto flotante con precisión p de x puede expresarse como

$$x^* = x(1+\gamma), \quad \text{donde} \quad \gamma = \begin{cases} [-2^{-p+1}, 0] & \text{si truncamos,} \\ [-2^{-p}, 2^{-p}] & \text{si redondeamos.} \end{cases}$$

Consecuencias:

- El mayor error de truncamiento es el doble que el mayor error de redondeo.
- El error de truncamiento es siempre no positivo, mientras que el error de redondeo puede cambiar de signo.

Por lo tanto, lo errores se amplifican menos si usamos redondeo.

Ejemplos

Si $x = (1.1001101)_2 = (1.6015625)_{10}$ lo aproximamos con

- truncamiento a 5 dígitos binarios,

$$x^* = (1.1001)_2 = (1.5625)_{10}$$

y entonces $\gamma = -\dfrac{x - x^*}{x} = -0.0243902\ldots \in \left[-2^{-4}, 0\right]$.

- redondeo a 5 dígitos binarios,

$$x^* = (1.1010)_2 = (1.625)_{10}$$

y entonces $\gamma = -\dfrac{x - x^*}{x} = 0.0146341\ldots \in \left[-2^{-5}, 2^{-5}\right]$.

Outline

Inestabilidad numérica

Errores de redondeo que se deben a que la aritmética de la computación es finita

- son *pequeños* en cada operación, **pero**
- pueden acumularse y propagarse si un algoritmo tiene muchas operaciones o iteraciones, resultando en una diferencia grande entre la solución exacta y la solución calculada numéricamente.

Este efecto se conoce como inestabilidad numérica del algoritmo.

Ejemplo

Para la sucesión $s_k = 1 + 2 + \ldots + k$, for $k = 1, 2, \ldots$, calcular

$$x_k = \frac{1}{s_k} + \frac{2}{s_k} + \ldots + \frac{k}{s_k},$$

cuyo resultado es

$$x_k = 1 \quad \text{para todos los } k = 1, 2, \ldots$$

Sin embargo, en precisión simple obtenemos

k	x_k^*	$\lvert x_k - x_k^* \rvert$
10^1	1.000000	0.0
10^3	0.999999	1.0×10^{-7}
10^6	0.9998996	1.004×10^{-4}
10^7	1.002663	2.663×10^{-3}

Error absoluto y relativo

Hay dos medidas principales del error cometido cuando aproximamos un número x con x^*:

- Error absoluto:

$$e_a = |x - x^*|.$$

- Error relativo:

$$e_r = \frac{|x - x^*|}{|x|}.$$

El error relativo es independiente de la escala y por tanto es más significativo que el error absoluto, como podemos ver en el siguiente **Ejemplo:**

x	$x*$	e_a	e_r
0.3×10^1	0.31×10^1	0.1	$0.333... \times 10^{-1}$
0.3×10^{-3}	0.31×10^{-3}	0.1×10^{-4}	$0.333... \times 10^{-1}$
0.3×10^4	0.31×10^4	0.1×10^3	$0.333... \times 10^{-1}$

Dígitos significativos

Decimos que x^* aproxima a x con p dígitos significativos si p es el mayor entero no negativo tal que el error relativo satisface

$$\frac{|x - x^*|}{|x|} \leq 5 \times 10^{-p}.$$

Ejemplos:

- $x^* = 124.45$ aproxima $x = 123.45$ con $p = 2$ dígitos significativos:

$$\frac{|x - x^*|}{|x|} = \frac{1}{123.45} = 0.0081 \leq 0.05 = 5 \times 10^{-2}.$$

- $x^* = 0.0012445$ aproxima $x = 0.0012345$ con $p = 2$ dígitos significativos:

$$\frac{|x - x^*|}{|x|} = \frac{0.00001}{0.0012345} = 0.0081 \leq 0.05 = 5 \times 10^{-2}.$$

- $x^* = 999.8$ aproxima $x = 1000$ con $p = 4$ dígitos significativoss:

$$\frac{|x - x^*|}{|x|} = \frac{0.2}{1000} = 0.0002 \leq 0.0005 = 5 \times 10^{-4}.$$