Clase 3

Cancelación de dígitos significativos

Recordemos la definición de dígitos significativos: el número $\overline{a}$ aproxima al número real a con r **dígitos significativos** si

$$\frac{\Delta a}{|a|} \leq 5\ 10^{-r} = \frac{1}{2}\ 10^{1-r}.$$

Un efecto no deseable en algoritmos numéricos es la gran cancelación de dígitos significativos que se produce en la resta de números próximos. Para fijar ideas veamos un ejemplo.

Sean $x_1 = 10.123455 \pm 0.5\ 10^{-6}$ y $x_2 = 10.123789 \pm 0.5\ 10^{-6}$.

x_1 y x_2 tienen error absoluto menor o igual a $0.5\ 10^{-6}$ y error relativo menor a $0.5\ 10^{-7} = 5\ 10^{-8}$, esto significa que ambos tienen 8 dígitos significativos.

Ahora bien, la resta $y = x_1 - x_2 = -0.000334 \pm 10^{-6}$, tiene un error absoluto pequeño, sin embargo el error relativo

$$\frac{\Delta y}{|y|} \leq \frac{10^{-6}}{0.000334} < 3\ 10^{-3} < 5\ 10^{-3},$$

por lo tanto la resta y tiene **sólo 3 dígitos significativos.**

Por lo tanto, es recomendable evitar restas de números próximos, siempre que sea posible.

Representación de números en una computadora

El ser humano está acostumbrado a utilizar un sistema de numeración decimal, el cual es un sistema posicional con base $\beta = 10$. La mayoría de las computadoras usa internamente la base β igual a 2 o 16.

Definición 1 *sea $\beta \in \mathbb{N}, \beta \geq 2$, todo número real r puede ser escrito en la forma:*

$$(\pm d_n d_{n-1} \ldots d_2 d_1 d_0 . d_{-1} d_{-2} \ldots)_\beta$$

donde $d_n, d_{n-1}, \ldots d_0, d_{-1} \ldots$ son números naturales entre 0 y $(\beta - 1)$. El valor del número r es:

$$\pm d_n \beta^n + d_{n-1}\beta^{n-1} + \cdots + d_2\beta^2 + d_1\beta^1 + d_0\beta^0 + d_{-1}\beta^{-1} + d_{-2}\beta^{-2} + \ldots$$

Ejemplos:

1. $(760)_8 = 7\ 8^2 + 6\ 8^1 + 0\ 8^0 = (496)_{10}$
2. $(101.101)_2 = 1\ 2^2 + 0\ 2^1 + 1\ 2^0 + 1\ 2^{-1} + 0\ 2^{-2} + 1\ 2^{-3} = (5.625)_{10}$
3. $(0.333\ldots)_{10} = 3\ 10^{-1} + 3\ 10^{-2} + \cdots = \frac{1}{3}$

4. $(0.1)_{10} = (0.0001100110011\ldots)_2$

Notar que en el último ejemplo $(0.1)_{10}$ no tiene representación binaria finita!

Observaciones:

1. la mayoría de los números reales no pueden ser representados exactamente en cualquier base;
2. aparecen errores de representación cuando un número es convertido de un sistema de numeración a otro;
3. aparecen errores debido a que la computadora usa aritmética finita.

¿cómo se representan los números en una computadora?

Básicamente, existen dos sistemas de representación de números en una computadora:

- sistema de punto fijo,
- sistema de punto flotante.

Sistema de punto fijo

El primero de ellos es el utilizado por las primeras computadoras (aproximadamente en 1940–1950) y donde los números se representan utilizando una cantidad fija de números enteros y de números fraccionarios. Por ejemplo, si usáramos la base β, $(s+1)$ dígitos para la parte entera y t para la parte fraccionaria, tendríamos:

$$\pm d_s\, d_{s-1}\, \ldots\, d_2\, d_1\, d_0\, .\, d_{-1}\, d_{-2}\, \ldots\, d_{-t},$$

donde cada $d_i \in \{0,\ldots,\beta-1\}$. En sistemas contables, aún hoy en día, suele usarse este sistema donde la cantidad de dígitos fraccionarios es $t=2$ para representar los centavos.

La principal desventaja de este sistema es que no es posible representar simultáneamente números reales muy pequeños y muy grandes, sin que la cantidad de dígitos s y t sean demasiados grandes. Por ejemplo si $s=3$ y $t=3$, el número más grande y el más que pequeño que se pueden representar en este sistema son 999.999 y 000.001, respectivamente. La manera de solucionar este problema es usar la notación científica y esto da origen al otro sistema.

Sistema de punto flotante

Definición 2 *Un* **sistema de punto flotante** (β,t,L,U) *es el conjunto de números normalizados en punto flotante en el sistema de numeración con base β, y t dígitos para la parte fraccionaria, es decir, números de la forma:*

$$x = m\,\beta^e$$

donde

$$m = \pm 0\,.\,d_{-1}\,d_{-2}\,\ldots\,d_{-t}$$

con $d_{-i} \in \{0,\ldots,\beta-1\}$ *para* $i=1,\ldots,t$*, con* $d_{-1} \neq 0$ *y* $L \leq e \leq U$*. Además,* β, e *y* m *se denominan base, exponente y mantisa, respectivamente. Es decir,* $1/\beta \leq |m| < 1$.

Observaciones:

1. aunque el sistema de punto flotante permite representar magnitudes de órdenes muy variados, a diferencia del sistema de punto fijo, también puede ocurrir *overflow* si $e > U$ o *underflow* si $e < L$;

2. **el cero no puede representarse en este sistema de números normalizados.**

Errores de redondeo en aritmética de punto flotante

Al representar números en un sistema de punto flotante (β, t, L, U) se producen errores de redondeo debido a la precisión limitada. Asumiendo redondeo, estimaremos una cota de los errores absoluto y relativo.

Supongamos que podemos escribir un número real (exacto) en la forma:

$$x = m\beta^e, \qquad \frac{1}{\beta} \leq |m| < 1,$$

donde el exponente e es tal que $L \leq e \leq U$.

Escribimos ahora su representante en el sistema de punto flotante:

$$fl(x) = x_r = m_r\beta^e, \qquad \frac{1}{\beta} \leq |m_r| < 1,$$

donde m_r es la mantisa que se obtiene redondeando a t dígitos la parte fraccionaria de m. Entonces, es claro que

$$|m_r - m| \leq \frac{1}{2}\beta^{-t},$$

y por lo tanto, una cota del error del absoluto de representación en x es

$$|x_r - x| \leq \frac{1}{2}\beta^{-t}\beta^e.$$

Para el error relativo, tenemos lo siguiente:

$$\frac{|x_r - x|}{|x|} \leq \frac{\frac{1}{2}\beta^{-t}\beta^e}{|m|\beta^e} = \frac{1}{2|m|}\beta^{-t} \leq \frac{1}{2}\beta^{1-t},$$

pues si $|m| \geq \dfrac{1}{\beta}$ entonces $\dfrac{1}{|m|} \leq \beta$.

Luego el error relativo debido al redondeo en la representación en el sistema de punto flotante está acotado por:

$$\frac{|x_r - x|}{|x|} \leq \frac{1}{2}\beta^{1-t} = \mu,$$

donde μ se llama unidad de redondeo.

Notar que el error absoluto de representación en punto flotante depende del orden de la magnitud, en cambio el error relativo no.

¿Cómo se realiza la suma en aritmética de punto flotante?

Para fijar ideas veremos dos ejemplos con el sistema de punto flotante dado por $(\beta, t, L, U) = (10, 4, -9, 9)$. Sean

$$x = m_x \beta^{e_x}, \qquad y = m_y \beta^{e_y},$$

con $x \geq y$. Queremos calcular $z = fl(x+y)$

Ejemplo 1: sean $x = 0.1234\ 10^0$, $\quad y = 0.4567\ 10^{-2}$, entonces

$$\begin{aligned} x+y &= 0.1234\ 10^0 + 0.4567\ 10^{-2} \\ &= 0.1234\ 10^0 + 0.004567\ 10^0 \\ &= (0.1234 + 0.004567)\ 10^0 = 0.127967\ 10^0, \end{aligned}$$

por lo tanto, $fl(x+y) = 0.1280\ 10^0$.

Ejemplo 2: sean $x = 0.1234\ 10^0$, $\quad y = 0.5678\ 10^{-5}$, entonces

$$\begin{aligned} x+y &= 0.1234\ 10^0 + 0.5678\ 10^{-5} \\ &= 0.1234\ 10^0 + 0.000005678\ 10^0 \\ &= (0.1234 + 0.000005678)\ 10^0 = 0.123405678\ 10^0, \end{aligned}$$

por lo tanto, $fl(x+y) = 0.1234\ 10^0 = x$.

Esto ocurre porque el orden de magnitud de x es con respecto a y es muy grande.

Observación: algunas propiedades o axiomas de la aritmética infinita dejan de valer en aritmética de punto flotante. Veamos con un ejemplo que la propiedad asociativa $((a+b)+c = a+(b+c))$ no es válida en un sistema de punto flotante, es decir: $fl(fl(a+b)+c) \neq fl(a+fl(b+c))$.

Dado el sistema de punto flotante dado por $(\beta, t, L, U) = (10, 4, -9, 9)$ consideremos los números $a = 0.9876\ 10^4, b = -0.9880\ 10^4$ y $c = 0.3456\ 10^1$. Entonces, por un lado,

$$\begin{aligned} fl(fl(a+b)+c) &= fl(fl(0.9876\ 10^4 - 0.9880\ 10^4) + c) \\ &= fl(fl(-0.0004\ 10^4) + c) \\ &= fl(-0.4000\ 10^1 + 0.3456\ 10^1) \\ &= fl(-0.0544\ 10^1) \\ &= -0.5440\ 10^0 \end{aligned}$$

Por otro lado,

$$\begin{aligned} fl(a+fl(b+c)) &= fl(a + fl(-0.9880\ 10^4 + 0.0003456\ 10^4)) \\ &= fl(a - fl(0.9876544\ 10^4)) \\ &= fl(0.9876\ 10^4 - 0.9877\ 10^4) \\ &= fl(-0.0001\ 10^4) \\ &= -0.1000\ 10^1 \end{aligned}$$

Observaciones de implementación:

1. dado que en una implementación o programa se realizan muchas operaciones, cada una con su correspondiente error, es conveniente prestar atención en las operaciones que se realizan;

2. si x e y son números reales y en una programa se tiene una sentencia del tipo

   ```
   if x == y then . . .
   ```

   es más conveniente reemplazarla por una sentencia del tipo

   ```
   if (abs(x-y)) < epsilon then . . .
   ```

   para algún valor de epsilon dado por el usuario, puesto que es casi imposible que se verifique la primera sentencia.