Índice

ÍNDICE

Capítulo 2

Análisis Numérico

El análisis numérico consiste en desarrollar métodos (algoritmos) para la solución aproximada de problemas matemáticos que no poseen una solución analítica simple.

En este proceso, deben tenerse en cuenta las distintas fuentes de error, a fin de evitar que dichos errores invaliden los resultados. Uno de estos errores son los **errores en los datos de entrada**, cuyo control está por fuera del algoritmo. Otro error con el que hay que lidiar es el **error de redondeo**, que surge cuando los cálculos son realizados con números cuya representación queda restringida a un número finito de dígitos, que es lo que lo que ocurre en una computadora. Veamos un ejemplo.

Supongamos que queremos evaluar la función

$$f(x) = \sqrt{x^2 + 1} - 1 \tag{2.1}$$

en $x = 0{,}25$, utilizando una computadora (ficticia) que hace las cuentas con 3 cifras significativas. Mostramos cómo se halla el resultado, paso a paso:

$$\begin{aligned} f(0{,}25) &= \sqrt{0{,}25^2 + 1} - 1 \\ &= \sqrt{0{,}0625 + 1} - 1 \\ &= \sqrt{1{,}06} - 1 \\ &= 1{,}03 - 1 \\ &= \boxed{0{,}03} \end{aligned} \tag{2.2}$$

Sin embargo, el resultado "correcto" (considerando 3 cifras significativas) debería ser 0,0308, teniendo en cuenta que el resultado exacto es 0,030776.

Si en lugar de trabajar con la expresión anterior la transformamos en su equivalente

$$f(x) = \frac{x^2}{\sqrt{1 + x^2} + 1} \tag{2.3}$$

obtenemos

$$\begin{aligned} f(0{,}25) &= \frac{0{,}25^2}{\sqrt{1 + 0{,}25^2} + 1} \\ &= \frac{0{,}0625}{\sqrt{1 + 0{,}0625} + 1} \\ &= \frac{0{,}0625}{1{,}03 + 1} \\ &= \frac{0{,}0625}{2{,}03} \\ &= \boxed{0{,}0308} \end{aligned} \tag{2.4}$$

que está más cerca del resultado exacto.

Conclusión: cuando se utiliza aritmética de precisión finita el resultado dependerá de cómo se realicen las cuentas, aún cuando se traten de expresiones equivalentes.

* * *

Otro tipo de errores son los llamados **errores de aproximación**, que surgen cuando un determinado método produce una solución a un problema que no es la exacta. Uno de esos errores es el llamado **error de truncamiento** que aparece, por ejemplo, cuando se aproxima una serie infinita por una suma finita de términos.

En otras ocasiones, un problema es aproximado por uno similar llevando a cabo una "discretización", por ejemplo

$$\int f(x)dx \quad \to \quad \sum_i f(x_i)\Delta x_i \tag{2.5}$$

$$\frac{df}{dx} \quad \to \quad \frac{f(x_i) - f(x_{i-1})}{\Delta x} \tag{2.6}$$

En estos casos el error de aproximación se conoce como **error de discretización**. De este tipo de errores nos vamos a encargar cuando veamos los distintos métodos para aproximar soluciones.

Comenzamos ahora con el estudio del efecto del redondeo en los cálculos, para lo cual necesitamos introducir algunas ideas.

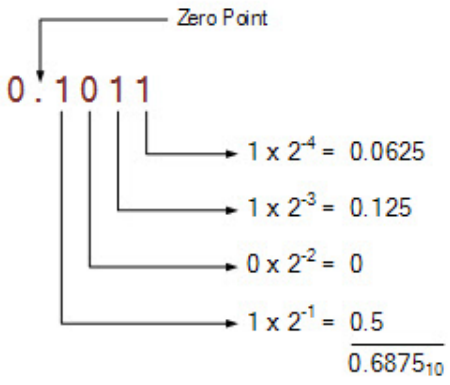
2.1. Representación de números: la notación posicional

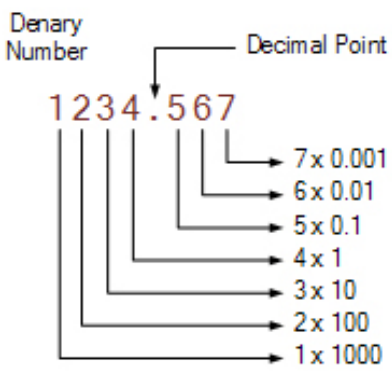
Revisemos primero la *notación posicional estándar*, en la cual el significado de cada dígito depende de su posición relativa con respecto a los demás.

Sea B cualquier entero positivo mayor que la unidad y sea d_i, $i \in \mathbb{Z}$, cualquiera de los números enteros entre 0 y $B-1$. Podemos representar entonces a cualquier real positivo como

$$d_n\, d_{n-1} \ldots d_1\, d_0\, .\, d_{-1}\, d_{-2} \ldots d_{-j} \ldots = \sum_{i=0}^{n} d_i B^i + \sum_{j=1}^{\infty} d_{-j} B^{-j}. \tag{2.7}$$

El número B se llama BASE de la representación. Nosotros estamos acostumbrados a trabajar con base $B = 10$ pero las computadoras lo hacen generalmente con base $B = 2$.





Si consideramos $B = 10$, sabemos que

$$347{,}29 = 3 \times 10^2 + 4 \times 10^1 + 7 \times 10^0 + 2 \times 10^{-1} + 9 \times 10^{-2}. \tag{2.8}$$

En cuanto al número infinito de cifras en la representación, está claro que hay casos, como éste, en donde sólo algunos de los coeficientes son distintos de cero.

Puede ocurrir que un número tenga una cantidad finita de cifras en una representación mientras que en otra requiera de infinitos dígitos. Por ejemplo,

$$B = 10 \qquad \frac{1}{10} = 0{,}1 \tag{2.9}$$

$$B = 2 \qquad \frac{1}{10} = 0{,}0001100110011... \tag{2.10}$$

por lo que 1/10 no tiene representación exacta en una computadora(!).

> Esta es una clara fuente de error al realizar operaciones con las computadoras, ya que uno le suele suministrar los datos en representación decimal y la computadora los debe convertir a representación binaria antes de comenzar a trabajar con ellos.

Por todo esto, conviene revisar cómo se realiza la conversión entre bases antes de seguir avanzando.

2.2. Conversión entre bases

Notación:

Si $x \in \mathbb{R}$, llamaremos $\phi_B(x)$ a su representación en la base B y $\phi_D(x)$ a la correspondiente a la base D. (Nota: usaremos notación decimal para describir a x, B y D.)

Normalmente el procedimiento de conversión se desdobla, tratándose las partes entera y fraccionaria por separado

$$x = N.F \tag{2.11}$$

Consideremos la conversión de la parte entera (N) desde la base D a la base B. Un número entero puede representarse en ambas bases de dos formas equivalentes:

$$N = \sum_{i=0}^{r} a_i D^i$$

$$= \sum_{j=0}^{s} c_j B^j \tag{2.12}$$

Suponemos los a_i conocidos, mientras que los c_j son las incógnitas. Dividimos N por la base a la que deseamos convertir:

$$\frac{N}{B} = \underbrace{\sum_{j=1}^{s} c_j B^{j-1}}_{=N_1} + \frac{c_0}{B} \tag{2.13}$$

En esta expresión, la sumatoria contiene la parte entera del cociente (N_1), mientras que c_0/B constituye la parte fraccionaria, en la cual c_0 es el resto.

Repetimos el proceso, ahora para N_1

$$\frac{N_1}{B} = \underbrace{\sum_{j=2}^{s} c_j B^{j-1}}_{=N_2} + \frac{c_1}{B} \tag{2.14}$$

y así se van obteniendo c_0, c_1,..., c_s.

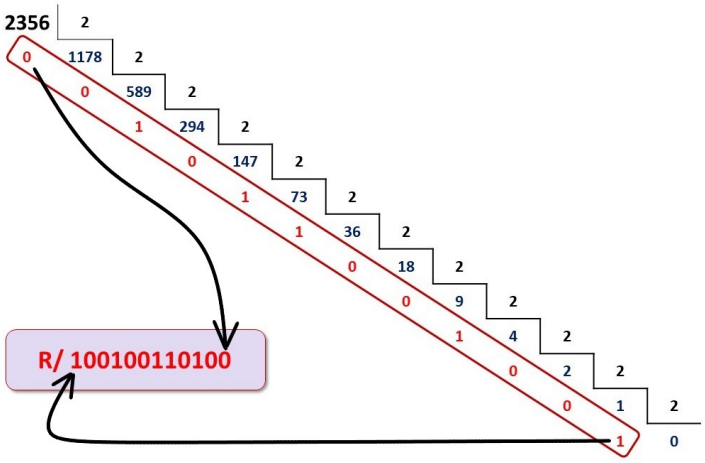


Figura 2.1: Conversión de la parte entera (de decimal a binario)

Para convertir la parte fraccionaria (F) de la base D a la base B escribimos primero su representación en ambas bases

$$F = \sum_{i=1}^{\infty} f_{-i} D^{-i}$$

$$= \sum_{j=1}^{\infty} g_{-j} B^{-j} \tag{2.15}$$

donde consideramos a los f_{-i} como dato y nos interesa hallar los g_{-j}, que son las incógnitas.

Multiplicando F por B

$$BF = g_{-1} + \underbrace{\sum_{j=2}^{\infty} g_{-j} B^{-j+1}}_{=F_1} \tag{2.16}$$

nos queda g_{-1} como la parte entera y el resto la parte fraccionaria (F_1). Podemos repetir el proceso, ahora para F_1

$$BF_1 = g_{-2} + \underbrace{\sum_{j=3}^{\infty} g_{-j} B^{-j+2}}_{=F_2} \tag{2.17}$$

donde g_{-2} es la parte entera y lo demás, la parte fraccionaria. Repitiendo el proceso obtenemos los demás coeficientes.

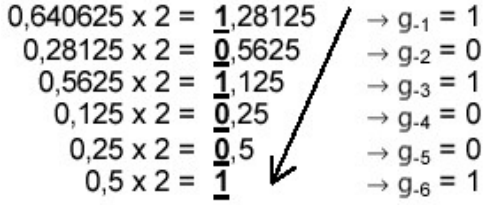


Figura 2.2: Conversión de la parte fraccionaria (de decimal a binario)

(En clase practicaremos el procedimiento convirtiendo dos números, $\phi_{10}(75{,}8)$ y $\phi_{10}(0{,}1)$, a base 2.)

Cuando convertimos desde un sistema con base $D \neq 10$ a la decimal conviene, en vez de utilizar este algoritmo, usar directamente la notación posicional. Por ejemplo, si deseáramos convertir $\phi_2(x) = 1110{,}101$ a decimal $\phi_{10}(x)$, haríamos directamente lo siguiente:

$$\begin{aligned}\phi_{10}(x) &= 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 8 + 4 + 2 + \frac{1}{2} + \frac{1}{8} \\ &= 14{,}625 \end{aligned} \tag{2.18}$$

2.3. Representación de números enteros en una computadora binaria

Aunque en general puede elegirse una base B cualquiera, vamos a trabajara con $B = 2$ que es la que usan las computadoras. Consideremos que disponemos de k dígitos con los que deseamos representar números (enteros) positivos y negativos utilizando las 2^k posibilidades que tenemos. Existen 4 métodos distintos para hacerlo, pero nosotros sólo vamos a ver dos.

2.3.1. Signo y magnitud

En este método se toman los $k-1$ dígitos menos significativos para guardar el valor absoluto del número en cuestión y el bit más significativo se preserva para el signo (0: signo +; 1: signo −).
Ejemplo ($k = 3$): en este caso tenemos 2 lugares para el valor absoluto.

000	(0)	100	(−0)
001	(1)	101	(−1)
010	(2)	110	(−2)
011	(3)	111	(−3)

Es decir, podemos representar 7 enteros distintos, desde -3 hasta 3, aunque hay dos formas de representar el cero. Esto puede considerarse una desventaja del método.

2.3.2. Representación en exceso N (o sesgada)

En esta representación se utiliza un número N preestablecido como sesgo. Luego, un valor es representado por el número sin signo que es mayor por N que el valor a obtener.

Por ejemplo, si $N = 3$, la representación binaria del cero corresponde a -3, de 1 a -2 y así siguiendo:

0	000	(−3)
1	001	(−2)
2	010	(−1)
3	011	(0)

Esta representación de enteros se utiliza para codificar el exponente de números de punto flotante.

2.4. Representación de punto flotante (números reales)

Las cantidades fraccionarias se representan en la computadora en la forma de punto flotante. Como se dijo anteriormente, las computadoras usan alguna forma de representación binaria para estos números; nosotros vamos a trabajar alternativamente con la base 10 por una cuestión de simplicidad.

Un número N en representación de punto flotante se escribe como

$$N = \pm a \times 10^b \tag{2.19}$$

donde a es el *significando*, b el *exponente* ($b \in \mathbb{Z}$) y 10 la base.

Consideremos el número 156,78. En notación de punto flotante este número se representa como

$$0{,}15678 \times 10^3 \tag{2.20}$$

aunque, evidentemente, podría haberse escrito también como

$$0{,}0015678 \times 10^5. \tag{2.21}$$

Sin embargo, esta última elección queda descartada porque los ceros de más ocupan lugares inútilmente. Así, se conviene utilizar una representación *normalizada*, para la cual se verifica que

$$\frac{1}{10} \leq a < 1. \tag{2.22}$$

La estructura de un número de punto flotante en una computadora es la siguiente:

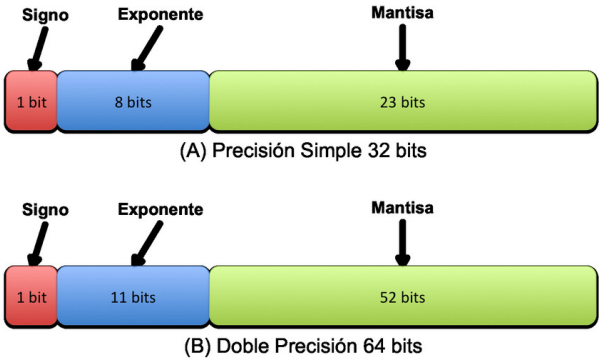


Figura 2.3: Representación de acuerdo con la norma IEEE 754

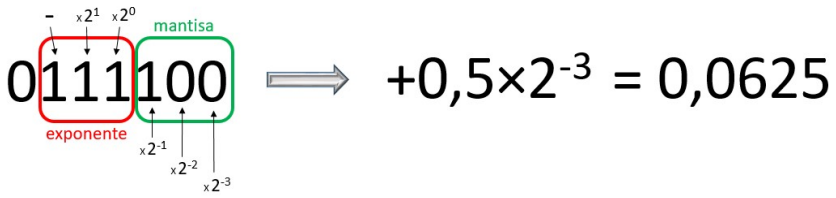
Donde uno establece de antemano el campo total (cantidad de bytes) que va a destinarse para cada número real.

Para fijar ideas, supongamos que se destinan 7 bits por cada número real: 1 bit para el signo, 3 bits para el exponente y 3 bits para el significando.

```
 sgn   e1   e2   e3   s1   s2   s3
|----|----|----|----|----|----|----|
 bit
```

Para facilitar la descripción supondremos que el exponente se representa mediante el método de signo y magnitud. Por lo tanto, con 3 bits para el exponente podremos representar $2^3 - 1$ ($= 7$) números enteros (desde -3 hasta 3). Además, sólo trataremos los números positivos (los negativos se obtienen de forma análoga).

Analicemos cuál es el número más pequeño que podemos representar. Dicho número corresponderá al de exponente más pequeño (-3) y menor significando, compatible con la normalización (100). Por lo tanto, en notación binaria, tendremos que el número más pequeño representable (en módulo) será



Los siguientes valores serán

$$0111101 \rightarrow 0{,}078125 \tag{2.23}$$

$$0111110 \rightarrow 0{,}093750 \tag{2.24}$$

$$0111111 \rightarrow 0{,}109375 \tag{2.25}$$

La diferencia entre números consecutivos es de 0,015625, que es lo que aporta un cambio de una unidad en el bit menos significativo, cuando el exponente es el menor posible ($1 \times 2^{-3} \times 2^{-3} = 2^{-6} = 0{,}015625$).

Para seguir generando los demás valores representables, hay que incrementar en uno el exponente. Así, los números que continúan son

$$0110100 \rightarrow 0{,}125000 \tag{2.26}$$

$$0110101 \rightarrow 0{,}156250 \tag{2.27}$$

$$0110110 \rightarrow 0{,}187500 \tag{2.28}$$

$$0110111 \rightarrow 0{,}218750 \tag{2.29}$$

Ahora la diferencia entre números estará dada por el cambio del bit menos significativo cuando el exponente es (-2), es decir, $1 \times 2^{-3} \times 2^{-2} = 2^{-5} = 0{,}03125$. Como se ve, ahora el salto entre números consecutivos aumenta al doble.

Este patrón se repite hasta alcanzar el exponente máximo. Para él, los números consecutivos serán

$$0011100 \rightarrow 4 \tag{2.30}$$

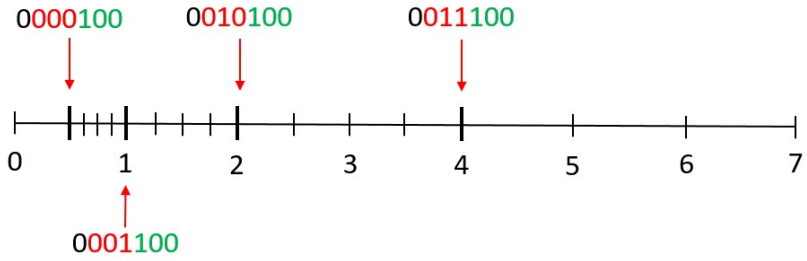
$$0011101 \rightarrow 5 \tag{2.31}$$

$$0011110 \rightarrow 6 \tag{2.32}$$

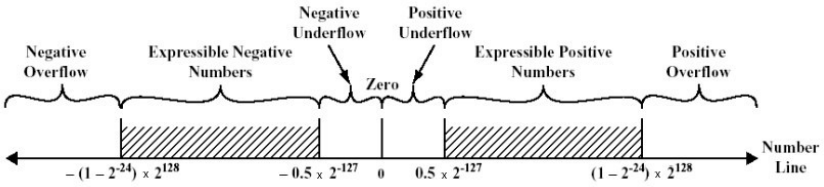
$$0011111 \rightarrow 7 \tag{2.33}$$

2.4.1. Características generales de la representación de punto flotante

- El conjunto de números representable es finito.
- El cero no está incluido, si nos atenemos a la normalización.
- El intervalo del eje real que se representa es acotado.
- La separación entre números aumenta al aumentar su valor absoluto. Esta característica es la que permite que la representación de punto flotante conserve los dígitos significativos. Esquemáticamente, el muestreo de la recta real será:



- Si, como resultado de un cálculo, obtenemos un valor por encima del máximo representable → *overflow*. (El máximo valor representable con reales de 4 bytes es de $\approx 3{,}4 \times 10^{38}$; con reales de 8 bytes $\approx 1{,}8 \times 10^{308}$.) Si obtenemos un valor por debajo del mínimo representable (en módulo) → *underflow*. Gráficamente, para la representación de 32 bits tenemos:



- En la figura 2.3 se muestra la distribución de bits en reales de 32 y 64 bits (4 y 8 bytes, respectivamente), según la norma IEEE 754. Al duplicar la cantidad de bits quien se lleva la mayor parte es el significando. (¿por qué?)

Como hemos visto, no todos los números reales tienen una representación exacta dentro de este esquema. Esto provoca que, en general, debamos trabajar con valores aproximados cuando realizamos cálculos con una computadora. Para decidir qué valor aproximado elegir, el esquema que se suele utilizar (aunque no siempre) es el de redondeo. A continuación estudiaremos el error de redondeo en la aritmética de punto flotante.

2.4.2. Errores de redondeo y aritmética de punto flotante

Sea $x \in \mathbb{R}$, $x \notin \mathcal{A}$ ($\mathcal{A}$: conjunto de números representables en punto flotante en una computadora). Sea $\mathrm{RD}(x)$ el número al cual se aproxima x y que está en $\mathcal{A}$. Lo que, en general, se le pide a $\mathrm{RD}(x)$ es que verifique

$$|x - \mathrm{RD}(x)| \leq |x - y| \qquad \forall y \in \mathcal{A}, \tag{2.34}$$

es decir, el valor de $\mathrm{RD}(x)$ es, de todos los números de $\mathcal{A}$, el más próximo.

En general, para obtener $\mathrm{RD}(x)$ de un número real cualquiera

$$x = \pm a \times 10^b, \quad \text{con} \quad a = 0.\alpha_1\alpha_2...\alpha_i\alpha_{i+1}... \quad (0 \leq \alpha_i \leq 9, \alpha_1 \neq 0) \tag{2.35}$$

en una computadora (decimal) de t dígitos hacemos

$$\mathrm{RD}(x) = \pm a' \times 10^b \tag{2.36}$$

donde

$$a' = \begin{cases} 0.\alpha_1\alpha_2...\alpha_t & \text{si} \quad 0 \leq \alpha_{t+1} < 5 \\ 0.\alpha_1\alpha_2...\alpha_t + 1 & \text{si} \quad 5 \leq \alpha_{t+1} \leq 9 \end{cases} \tag{2.37}$$

Esquemáticamente,

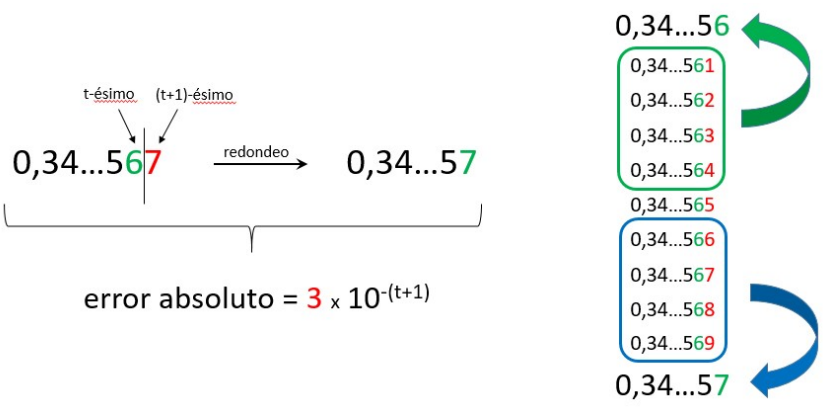


Figura 2.4: Redondeo y error absoluto asociado.

¿Qué número producirá el mayor error de redondeo en el esquema anterior? ¿De cuánto sería?

MALA SUERTE. ¿Podría ocurrir que el número redondeado no perteneciera a $\mathcal{A}$? Técnicamente, sí. Por ejemplo, si tuviéramos que representar el número $x = 0{,}99997 \times 10^{99}$ en una computadora con 2 dígitos para el exponente y el valor redondeado fuera $\mathrm{RD}(x) = 0{,}10 \times 10^{100}$, éste no sería representable y se daría una condición de *overflow*.

Si bien estos casos existen, se presentan esporádicamente, por lo que vamos a suponer que $\mathrm{RD}(x)$ siempre pertenece a $\mathcal{A}$.

Epsilon de la máquina

Resulta importante conocer una cota del error **relativo** que contiene $\mathrm{RD}(x)$. Para ello, si tenemos presente que $|a| \geq 10^{-1}$, podemos hacer la siguiente estimación

$$|\varepsilon| = \frac{|x - \mathrm{RD}(x)|}{|x|} = \frac{|a - a'| \times \cancel{10^{b}}}{|a| \times \cancel{10^{b}}} \leq \frac{5 \times 10^{-(t+\cancel{1})}}{\cancel{10^{-1}}} = 5 \times 10^{-t} = \mathrm{eps} \tag{2.38}$$

Esta cantidad se denomina el *epsilon de la máquina* (eps) y caracteriza el error *relativo* que se comete con el redondeo. Así, se cumple que

$$\mathrm{RD}(x) = x(1 + \varepsilon), \qquad |\varepsilon| \leq \mathrm{eps} \tag{2.39}$$

Una consecuencia de su definición es que el epsilon de la máquina es el número más pequeño que, sumado a uno, da un número mayor que uno.

* * *

Un problema importante que viene de la mano de la representación de punto flotante es que, aún cuando operemos con números x, $y \in \mathcal{A}$, el resultado de las operaciones aritméticas (*,/,+,-), en general, no pertenecerá a $\mathcal{A}$. Para explicitar este hecho, denotaremos las operaciones aritméticas realizadas con error de redondeo con símbolos especiales. Así, tendremos que

$$\begin{aligned} x \otimes y &= \mathrm{RD}(x * y), \\ x \oslash y &= \mathrm{RD}(x / y), \\ x \oplus y &= \mathrm{RD}(x + y), \\ x \ominus y &= \mathrm{RD}(x - y), \end{aligned}$$

donde hemos reservado la simbología habitual para referirnos a la operación "exacta", esto es, sin error de redondeo. De acuerdo con lo visto más arriba, podemos escribir estas expresiones como

$$\begin{aligned} x \otimes y &= (x * y)(1 + \varepsilon_1), \\ x \oslash y &= (x / y)(1 + \varepsilon_2), \\ x \oplus y &= (x + y)(1 + \varepsilon_3), \\ x \ominus y &= (x - y)(1 + \varepsilon_4), \end{aligned}$$

donde $|\varepsilon_i| \leq \mathrm{eps}$, $i = 1, 2, 3, 4$. Las operaciones de punto flotante no satisfacen propiedades básicas a las que estamos muy acostumbrados. Así, ahora resulta que

$$\begin{aligned} x \otimes (y \otimes z) &\neq (x \otimes y) \otimes z, \\ x \oplus (y \oplus z) &\neq (x \oplus y) \oplus z, \\ x \otimes (y \oplus z) &\neq x \otimes y \oplus x \otimes z. \end{aligned}$$

2.4.3. Ejemplos de operaciones en aritmética de punto flotante

Consideremos una computadora con un significando de 4 dígitos y un exponente de un dígito en base 10.

Para sumar y restar debemos tener en cuenta que, si los exponentes de los números difieren, el de menor exponente será modificado hasta coincidir con el del mayor (ajustando el significando consecuentemente):

$$\begin{array}{rr} & 0{,}1557 \times 10^{1} \\ + & 0{,}4381 \times 10^{-1} \\ \hline \end{array} \quad \rightarrow \quad \begin{array}{rr} & 0{,}1557 \times 10^{1} \\ + & 0{,}0044 \times 10^{1} \\ \hline & 0{,}1601 \times 10^{1} \end{array} \tag{2.40}$$

El valor exacto es $0{,}160081 \times 10^{1}$. Esta forma de realizar la suma puede provocar lo siguiente:

$$\begin{array}{rr} & 0{,}4000 \times 10^{4} \\ + & 0{,}1000 \times 10^{-3} \\ \hline \end{array} \quad \rightarrow \quad \begin{array}{rr} & 0{,}4000 \times 10^{4} \\ + & 0{,}0000 \times 10^{4} \\ \hline & 0{,}4000 \times 10^{4} \end{array} \tag{2.41}$$

que implica que, si dos números difieren en varios órdenes de magnitud, el número menor no aporte a la suma.

Veamos un ejemplo de sustracción entre números de igual exponente

$$\begin{array}{rr} & 0{,}3641 \times 10^{2} \\ - & 0{,}2686 \times 10^{2} \\ \hline & 0{,}0955 \times 10^{2} \end{array} \quad \rightarrow \quad 0{,}955\boxed{?} \times 10^{1} \tag{2.42}$$

¿qué valor se espera que tenga el dígito recuadrado?

De especial interés es el caso en el que la resta se da entre números muy parecidos

$$\begin{array}{rr} & 0{,}7642 \times 10^{5} \\ - & 0{,}7641 \times 10^{5} \\ \hline & 0{,}0001 \times 10^{5} \end{array} \quad \rightarrow \quad 0{,}1\boxed{?}\boxed{?}\boxed{?} \times 10^{2} \tag{2.43}$$

El problema en este caso es que se pierden los dígitos más significativos, con lo cual pasan a serlo números que (pueden) provenir de cálculos previos, con errores de redondeo. A este fenómenos se denomina "cancelación", y es una de las fuentes más peligrosas de errores numéricos.

En cuanto al producto, como la multiplicación de dos significandos de n dígitos da por resultado uno de $2n$ (dígitos), muchas computadoras ofrecen la posibilidad de guardar resultados intermedios con mayor precisión para luego redondear.

$$\begin{array}{rr} & 0{,}1363 \times 10^{3} \\ \times & 0{,}6423 \times 10^{-1} \\ \hline & 0{,}08754549 \times 10^{2} \end{array} \quad \rightarrow \quad 0{,}8755 \times 10^{1} \tag{2.44}$$

La multiplicación no suele generar problemas, siempre que no nos salgamos del rango representable (casos de *underflow* y *overflow*).

2.5. Breve introducción a la teoría de errores

2.5.1. Definiciones preliminares

Dado un número x, que supondremos exacto (aunque desconocido), y otro número $\widetilde{x}$, que será una aproximación a x, llamaremos *error absoluto* de $\widetilde{x}$ a

$$\boxed{\Delta = \widetilde{x} - x.} \tag{2.45}$$

Por su parte, el *error relativo* de $\widetilde{x}$ estará dado por

$$\boxed{\varepsilon = \frac{\widetilde{x} - x}{x}.} \tag{2.46}$$

Por completitud, definimos también el *error relativo porcentual*:

$$\delta = 100 \cdot \varepsilon. \tag{2.47}$$

Como, en general, no se conoce x, no se pueden calcular Δ y ε en forma exacta y nos tenemos que contentar con establecer cotas para los errores.

Def. Llamamos *cota superior del error absoluto* Δ de $\widetilde{x}$ a todo número real positivo Δ^* que verifique

$$|\Delta| \leq \Delta^*. \tag{2.48}$$

De aquí resulta que $\widetilde{x} - \Delta^* \leq x \leq \widetilde{x} + \Delta^*$, lo que suele denotarse como

$$x = \widetilde{x} \pm \Delta^*. \tag{2.49}$$

Def. Llamamos *cota superior del error relativo* ε de $\widetilde{x}$ a todo número real positivo ε^* tal que

$$|\varepsilon| \leq \varepsilon^*. \tag{2.50}$$

2.5.2. Relación entre el error relativo y el número de cifras significativas de un número aproximado

Decimos que el número $\widetilde{x}$ aproxima a x con k cifras significativas si

$$|\varepsilon| \leq 5 \times 10^{-k}, \tag{2.51}$$

donde k es el **mayor** entero positivo para el que se verifica esta desigualdad.

Ejemplo. Supongamos que hemos obtenido como resultado de una cuenta

$$0{,}127439855 \times 10^5 \tag{2.52}$$

con $|\varepsilon| \leq 10^{-4}$, entonces, el número de cifras confiables, de acuerdo a la expresión anterior, será:

$$|10^{-4}| \leq 5 \times 10^{-(k=4)}. \tag{2.53}$$

Por lo tanto, del número original serán útiles sólo las primeras 4 cifras

$$0{,}1274\,|\,39855 \times 10^5 \tag{2.54}$$

2.6. Propagación de errores (sin incluir redondeo)

Consideremos los datos de entrada x que producirán a través de la aplicación de una función φ resultados y, es decir,

$$y = \varphi(x). \tag{2.55}$$

Para darle generalidad al problema, supondremos que

$$x = (x_1, x_2, ..., x_n)$$

$$y = (y_1, y_2, ..., y_m)$$

$$\varphi : D \in \mathbb{R}^n \to \mathbb{R}^m \tag{2.56}$$

es decir, que φ es una función vectorial de m componentes, donde

$$y_i = \varphi_i(x_1, x_2, ..., x_n), \qquad i = 1, 2, ..., m. \tag{2.57}$$

La idea es estudiar cómo se propagan los errores presentes en los datos de entrada. Por el momento supondremos que no existen los errores de redondeo, es decir, que nuestra computadora tiene **precisión infinita**. También supondremos que las funciones φ_i tienen derivadas primeras continuas en D y, de acuerdo con las definiciones previas, denotaremos

$$\Delta x_i = \widetilde{x}_i - x_i, \qquad i = 1, 2, ..., n \tag{2.58}$$

a los errores absolutos de las componentes del vector de datos de entrada, que podemos juntar en un vector de errores absolutos

$$\Delta x = \widetilde{x} - x. \tag{2.59}$$

En forma similar, los errores relativos serán

$$\varepsilon_{x_i} = \frac{\widetilde{x}_i - x_i}{x_i}, \qquad \text{si} \quad x_i \neq 0. \tag{2.60}$$

Si los datos de entrada contienen errores, en lugar de suministrarle a φ valores exactos x ingresaremos otros que son aproximados $\widetilde{x}$ y, consecuentemente, la salida reflejará esta diferencia

$$\widetilde{y} = \varphi(\widetilde{x}). \tag{2.61}$$

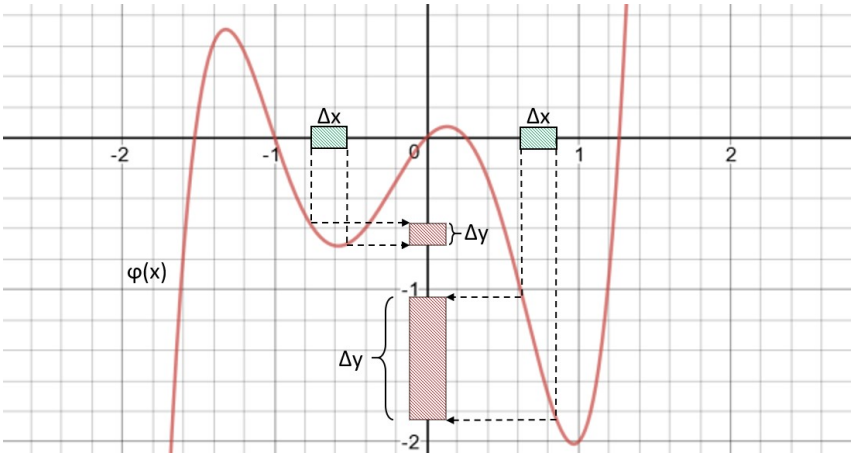


Figura 2.5: Impacto de los errores en x en el cálculo de y.

Notar que la función no cambia, lo que cambia es el resultado por haber modificado los valores de entrada.

Si desarrollamos en serie de Taylor alrededor de los valores verdaderos (desconocidos) y despreciamos términos de orden superior tendremos

$$\Delta y_i = \widetilde{y}_i - y_i = \varphi_i(\widetilde{x}) - \varphi_i(x) \approx \cancel{\varphi_i(x)} + \sum_{j=1}^{n} (\widetilde{x}_j - x_j) \frac{\partial \varphi_i(x)}{\partial x_j} - \cancel{\varphi_i(x)}, \tag{2.62}$$

es decir,

$$\Delta y_i \approx \sum_{j=1}^{n} \frac{\partial \varphi_i(x)}{\partial x_j} \Delta x_j, \qquad i = 1, 2, ..., m. \tag{2.63}$$

Esta expresión puede escribirse de manera más compacta como

$$\Delta y \approx J\varphi(x)\, \Delta x \tag{2.64}$$

donde la matriz

$$J\varphi(x) = \begin{pmatrix} \frac{\partial \varphi_1}{\partial x_1} & \frac{\partial \varphi_1}{\partial x_2} & \cdots & \frac{\partial \varphi_1}{\partial x_n} \\ \vdots & \ddots & & \vdots \\ \frac{\partial \varphi_m}{\partial x_1} & \frac{\partial \varphi_m}{\partial x_2} & \cdots & \frac{\partial \varphi_m}{\partial x_n} \end{pmatrix} \tag{2.65}$$

se denomina la *matriz jacobiana*.

Notar que la cantidad

$$\frac{\partial \varphi_i(x)}{\partial x_j} \tag{2.66}$$

representa la sensibilidad con que el resultado y_i reacciona a la perturbación Δx_j de x_j. Si $y_i \neq 0$, para $i = 1, 2, ..., m$ y $x_j \neq 0$ para $j = 1, 2, ..., n$, puede verse que una fórmula similar vale para los errores relativos

$$\varepsilon_{y_i} \approx \sum_{j=1}^{n} \boxed{\frac{x_j}{\varphi_i(x)} \frac{\partial \varphi_i(x)}{\partial x_j}} \varepsilon_{x_j}. \tag{2.67}$$

donde hemos recuadrado el factor que indica cuán fuertemente afecta el error relativo en x_j al error relativo del resultado y_i.

La diferencia con la fórmula anterior es que estos *factores de amplificación* no dependen de las escalas de x_j e y_i. Los factores de amplificación se llaman **números de condición**. En general, números de condición con valor absoluto grande ($\gg 1$) implican **problemas mal condicionados**[1].

Ejemplo. Vamos a estudiar el condicionamiento de

$$y = \varphi(p, q) = -p + \sqrt{p^2 + q}. \tag{2.68}$$

El error relativo ε_y será

$$\varepsilon_y \approx \frac{p}{\varphi(p,q)} \frac{\partial \varphi(p,q)}{\partial p} \varepsilon_p + \frac{q}{\varphi(p,q)} \frac{\partial \varphi(p,q)}{\partial q} \varepsilon_q. \tag{2.69}$$

Calculamos las derivadas parciales

$$\frac{\partial \varphi}{\partial p} = -1 + \frac{p}{\sqrt{p^2 + q}} = \frac{(-\varphi)}{\sqrt{p^2 + q}}, \tag{2.70}$$

$$\frac{\partial \varphi}{\partial q} = \frac{1}{2\sqrt{p^2 + q}}. \tag{2.71}$$

Resulta útil reescribir φ de la siguiente manera

$$\varphi(p, q) = (-p + \sqrt{p^2 + q}) \frac{(p + \sqrt{p^2 + q})}{(p + \sqrt{p^2 + q})} = \frac{q}{p + \sqrt{p^2 + q}} \tag{2.72}$$

lo cual será válido siempre que $p^2 \geq q$.

Reemplazando en la expresión del error

$$\begin{aligned} \varepsilon_y &\approx \frac{p}{\cancel{\varphi}} \frac{(-\cancel{\varphi})}{\sqrt{p^2 + q}} \varepsilon_p + \frac{q}{\varphi} \frac{1}{2\sqrt{p^2 + q}} \varepsilon_q \\ &= -\frac{p}{\sqrt{p^2 + q}} \varepsilon_p + \frac{\cancel{q}}{\cancel{q}/(p + \sqrt{p^2 + q})} \frac{1}{2\sqrt{p^2 + q}} \varepsilon_q \\ &= \boxed{-\frac{p}{\sqrt{p^2 + q}} \varepsilon_p + \frac{p + \sqrt{p^2 + q}}{2\sqrt{p^2 + q}} \varepsilon_q} \end{aligned} \tag{2.73}$$

[1] Hablamos de un problema mal condicionado cuando pequeñas perturbaciones en los datos de entrada generan resultados muy distintos entre sí. El buen o mal condicionamiento de un problema no depende del algoritmo utilizado para resolverlo sino del problema en sí.

Si $q > 0$

$$\left|\frac{p}{\sqrt{p^2+q}}\right| \leq 1, \qquad \text{y} \qquad \left|\frac{p+\sqrt{p^2+q}}{2\sqrt{p^2+q}}\right| \leq 1, \tag{2.74}$$

es decir, en este caso φ está bien condicionada.

Si, en cambio, $q \approx -p^2$, muy probablemente la función estará mal condicionada.

2.6.1. Acotando el error

Si sólo se conocen cotas de los errores absolutos o relativos de las variables x_i, es posible encontrar cotas para los respectivos errores en las variables y_i; por ejemplo, para el caso de los errores relativos tenemos

$$|\varepsilon_{y_i}| \approx \left|\sum_{j=1}^{n} \frac{x_j}{\varphi_i(x)} \frac{\partial \varphi_i(x)}{\partial x_j} \varepsilon_{x_j}\right| \leq \sum_{j=1}^{n} \left|\frac{x_j}{\varphi_i(x)} \frac{\partial \varphi_i(x)}{\partial x_j}\right| \left|\varepsilon_{x_j}\right|, \qquad (i = 1, 2, ..., m) \tag{2.75}$$

Utilizando la definición de cota superior del error relativo (que vimos antes) podemos escribir

$$\varepsilon^*_{y_i} = \sum_{j=1}^{n} \left|\frac{x_j}{\varphi_i(x)} \frac{\partial \varphi_i(x)}{\partial x_j}\right| \varepsilon^*_{x_j}, \qquad (i = 1, 2, ..., m) \tag{2.76}$$

que será una cota superior al error relativo en ε_{y_i}.

Podemos proceder de forma análoga para los errores absolutos, y obtendremos

$$\Delta^* y_i = \sum_{j=1}^{n} \left|\frac{\partial \varphi_i(x)}{\partial x_j}\right| \Delta^* x_j, \qquad (i = 1, 2, ..., m) \tag{2.77}$$

2.6.2. Ejemplo de propagación de errores para casos elementales

Sean $x, y \neq 0$. Calculemos cuál es la propagación del error para cuatro casos elementales.

Producto

$$\varphi(x, y) = x \cdot y \quad \Rightarrow \quad \varepsilon_\varphi \approx \frac{x}{\varphi}\frac{\partial \varphi}{\partial x}\varepsilon_x + \frac{y}{\varphi}\frac{\partial \varphi}{\partial y}\varepsilon_y = \boxed{\varepsilon_x + \varepsilon_y} \tag{2.78}$$

División

$$\varphi(x, y) = x/y \quad \Rightarrow \quad \varepsilon_\varphi \approx \frac{x}{(x/y)}\frac{1}{y}\varepsilon_x + \frac{y}{(x/y)}\frac{(-x)}{y^2}\varepsilon_y = \boxed{\varepsilon_x - \varepsilon_y} \tag{2.79}$$

Suma y resta

$$\varphi(x, y) = x \pm y \quad \Rightarrow \quad \varepsilon_\varphi \approx \frac{x}{x \pm y}(1)\,\varepsilon_x + \frac{y}{x \pm y}(\pm 1)\,\varepsilon_y = \boxed{\frac{x}{x \pm y}\,\varepsilon_x \pm \frac{y}{x \pm y}\,\varepsilon_y} \tag{2.80}$$

(vale si $x \pm y \neq 0$).

Radicación

$$\varphi(x) = \sqrt{x} \quad \Rightarrow \quad \varepsilon_\varphi \approx \frac{x}{\sqrt{x}}\frac{1}{2\sqrt{x}}\varepsilon_x = \boxed{\frac{\varepsilon_x}{2}} \tag{2.81}$$

El análisis de estos resultados, y sus consecuencias, serán hechos en clase.

2.7. Propagación de errores (incluyendo redondeo)

Consideremos ahora el caso más general, en el que incluimos los errores de redondeo que se producen cuando operamos con precisión finita.

Un algoritmo para calcular la función $\varphi : D \to \mathbb{R}^m$, $D \subseteq \mathbb{R}^n$, para algún $x = (x_1, x_2, ..., x_n) \in D$ puede pensarse como una concatenación de funciones o transformaciones en las que φ puede ser descompuesta

$$\varphi = \varphi^{(r)} \circ \varphi^{(r-1)} \circ ... \circ \varphi^{(0)}, \tag{2.82}$$

que nos lleva desde $x^{(0)} \equiv x$, a través de una cadena de resultados intermedios,

$$x = x^{(0)} \to \varphi^{(0)}(x^{(0)}) = x^{(1)} \to ...\varphi^{(r)}(x^{(r)}) = x^{(r+1)} = y \tag{2.83}$$

al resultado y. Las funciones $\varphi^{(i)}$ se llaman transformaciones *elementales*.

Definimos ahora la "transformación restante" como

$$\psi^{(i)} = \varphi^{(r)} \circ \varphi^{(r-1)} \circ ... \circ \varphi^{(i)} : D_i \to \mathbb{R}^m \qquad i = 0, 1, ..., r \tag{2.84}$$

De esta definición, es claro que

$$\psi^{(0)} \equiv \varphi. \tag{2.85}$$

Sean ahora $J\varphi^{(i)}$ y $J\psi^{(i)}$, las matrices jacobianas de las funciones $\varphi^{(i)}$ y $\psi^{(i)}$, respectivamente. Utilizando la siguiente propiedad del jacobianos para la composición de funciones (sin demostración)

$$J(f \circ g)(x) = Jf(g(x)) \cdot Jg(x). \tag{2.86}$$

tenemos entonces que

$$J\varphi(x) = J\varphi^{(r)}(x^{(r)}) \cdot J\varphi^{(r-1)}(x^{(r-1)}) ... J\varphi^{(0)}(x^{(0)}) \tag{2.87}$$

y

$$J\psi^{(i)}(x^{(i)}) = J\varphi^{(r)}(x^{(r)}) \cdot J\varphi^{(r-1)}(x^{(r-1)}) ... J\varphi^{(i)}(x^{(i)}). \qquad (i = 0, 1, ..., r) \tag{2.88}$$

Usando aritmética de punto flotante, a los errores de entrada se sumarán ahora los de redondeo, que perturbarán tanto a los valores iniciales como a los resultados intermedios. Así, tendremos que

$$\widetilde{x}^{(i+1)} = \mathrm{FL}\left(\varphi^{(i)}(\widetilde{x}^{(i)})\right) \tag{2.89}$$

donde hemos usado "FL" para simbolizar el efecto de la aproximación de punto flotante. Para analizar el error absoluto de $\widetilde{x}^{(i+1)}$ podemos escribir

$$\begin{aligned} \Delta x^{(i+1)} &= \widetilde{x}^{(i+1)} - x^{(i+1)} \\ &= \mathrm{FL}\left(\varphi^{(i)}(\widetilde{x}^{(i)})\right) - x^{(i+1)} \\ &= \underbrace{\left[\mathrm{FL}\left(\varphi^{(i)}(\widetilde{x}^{(i)})\right) - \varphi^{(i)}(\widetilde{x}^{(i)})\right]}_{\text{por redondeo}} + \underbrace{\left[\varphi^{(i)}(\widetilde{x}^{(i)}) - \varphi^{(i)}(x^{(i)})\right]}_{\text{por error en } x^{(i)}} \end{aligned} \tag{2.90}$$

2.7.1. Aporte por el error en $x^{(i)}$

Utilizando la teoría previa (propagación de errores en los datos) podemos estimar el error que aporta la inexactitud de $x^{(i)}$ como

$$\varphi^{(i)}(\widetilde{x}^{(i)}) - \varphi^{(i)}(x^{(i)}) \approx \boxed{J\varphi^{(i)}(x^{(i)})\,\Delta x^{(i)}}. \tag{2.91}$$

2.7.2. Aporte por el error de redondeo

Identificando el error por representación de punto flotante con el redondeo

$$\mathrm{FL}\left(\varphi^{(i)}(u)\right) = \mathrm{RD}\left(\varphi^{(i)}(u)\right), \tag{2.92}$$

y teniendo en cuenta que

$$\varphi^{(i)}(u) = \left(\varphi_1^{(i)}(u),\, \varphi_2^{(i)}(u), \ldots,\, \varphi_{n_{i+1}}^{(i)}(u)\right), \tag{2.93}$$

donde $\varphi^{(i)} : D_i \to D_{i+1}$, con $D_{i+1} \subseteq \mathbb{R}^{n_{i+1}}$.

Para cada componente, tenemos entonces que

$$\mathrm{FL}\left(\varphi_j^{(i)}(u)\right) = \mathrm{RD}\left(\varphi_j^{(i)}(u)\right) = (1+\varepsilon_j)\,\varphi_j^{(i)}(u), \tag{2.94}$$

con $|\varepsilon_j| \leq$eps y $1 \leq j \leq n_{i+1}$. Así, podemos reescribir

$$\mathrm{FL}\left(\varphi^{(i)}(u)\right) = (I + E_{i+1})\varphi^{(i)}(u), \tag{2.95}$$

donde

$$E_{i+1} = \begin{pmatrix} \varepsilon_1 & \ldots & 0 \\ \vdots & \ddots & \\ 0 & & \varepsilon_{n_{i+1}} \end{pmatrix}. \tag{2.96}$$

Por lo tanto, los primeros dos términos del error absoluto podrán escribirse como

$$\mathrm{FL}\left(\varphi^{(i)}(\widetilde{x}^{(i)})\right) - \varphi^{(i)}(\widetilde{x}^{(i)}) = (I + E_{i+1})\varphi^{(i)}(\widetilde{x}^{(i)}) - \cancel{\varphi^{(i)}(\widetilde{x}^{(i)})} = E_{i+1}\varphi^{(i)}(\widetilde{x}^{(i)}), \tag{2.97}$$

que puede ser aproximado por

$$E_{i+1}\,\varphi^{(i)}(\widetilde{x}^{(i)}) \approx E_{i+1}\,\varphi^{(i)}(x^{(i)}), \tag{2.98}$$

ya que los términos de error por los cuales difieren $\varphi^{(i)}(\widetilde{x}^{(i)})$ y $\varphi^{(i)}(x^{(i)})$ son multiplicados por ε en la diagonal de E_{i+1}, dando lugar a términos cuadráticos en el error, que despreciaremos. Así, tendremos que

$$\mathrm{FL}\left(\varphi^{(i)}(\widetilde{x}^{(i)})\right) - \varphi^{(i)}(\widetilde{x}^{(i)}) \approx E_{i+1}\,\varphi^{(i)}(x^{(i)}) = E_{i+1}\,x^{(i+1)} \equiv \boxed{\alpha_{i+1}}. \tag{2.99}$$

Por lo tanto, describiremos con α_{i+1} a la aproximación de los *errores intermedios de redondeo en el paso* $i+1$.

2.7.3. Uniendo todo

Uniendo estos resultados, podremos escribir el error absoluto en el paso $i+1$ como

$$\Delta x^{(i+1)} \approx \alpha_{i+1} + J\varphi^{(i)}(x^{(i)})\,\Delta x^{(i)}, \tag{2.100}$$

para $i = 0, 1, ..., r$ y $\Delta x^{(0)} = \Delta x$. Para hallar el error absoluto total (al finalizar el cálculo) necesitamos ir calculando los errores previos hasta llegar al último paso

$$\Delta x^{(1)} \approx J\varphi^{(0)}(x)\Delta x + \alpha_1 \tag{2.101}$$

$$\begin{aligned} \Delta x^{(2)} &\approx J\varphi^{(1)}(x^{(1)})\Delta x^{(1)} + \alpha_2 \\ &\approx J\varphi^{(1)}(x^{(1)})\left[J\varphi^{(0)}(x)\Delta x + \alpha_1\right] + \alpha_2 \end{aligned} \tag{2.102}$$

$$\begin{aligned} \Delta x^{(3)} &\approx J\varphi^{(2)}(x^{(2)})\Delta x^{(2)} + \alpha_3 \\ &\approx J\varphi^{(2)}(x^{(2)})\left[J\varphi^{(1)}(x^{(1)})\left[J\varphi^{(0)}(x)\Delta x + \alpha_1\right] + \alpha_2\right] + \alpha_3 \end{aligned} \tag{2.103}$$

$$\vdots$$

$$\begin{aligned}\Delta x^{(r+1)} &\approx J\varphi^{(r)}(x^{(r)})\Delta x^{(r)} + \alpha_{r+1} \\ &\approx J\varphi^{(r)}\, J\varphi^{(r-1)} \dots J\varphi^{(0)}\Delta x + J\varphi^{(r)} \dots J\varphi^{(1)}\alpha_1 + \cdots + \alpha_{r+1}\end{aligned} \tag{2.104}$$

Recordando que

$$\Delta y \equiv \Delta x^{(r+1)} \tag{2.105}$$

y usando las funciones "transformación restante" (que definimos antes) podemos escribir

$$\Delta y \approx J\varphi(x)\Delta x + J\psi^{(1)}(x^{(1)})\alpha_1 + \cdots + J\psi^{(r)}(x^{(r)})\alpha_r + \alpha_{r+1} \tag{2.106}$$

y, finalmente,

$$\boxed{\Delta y \approx J\varphi(x)\,\Delta x + E_{r+1}\cdot y + \sum_{k=1}^{r} J\psi^{(k)}(x^{(k)})\,\alpha_k} \tag{2.107}$$

Observaciones

- Notar que la magnitud de los elementos de las matrices jacobianas $J\psi^{(k)}(x^{(k)})$ es la que determina el efecto de los errores intermedios de redondeo α_i en el resultado final.
- Diferentes algoritmos para calcular el mismo resultado (en otras palabras, diferentes descomposiciones de φ) resultarán en diferentes jacobianos $J\psi^{(k)}(x^{(k)})$ pero no modificarán $J\varphi(x)$. En consecuencia, la elección del algoritmo influirá en el efecto total del redondeo. Siempre deberemos elegir el algoritmo *numéricamente más confiable*, es decir, el de menores errores de redondeo.
- El aporte del error de redondeo introducido en el último paso puede ser acotado

  $$|E_{r+1}\cdot y| \le |y|\,\text{eps} \tag{2.108}$$

  Así, independientemente del algoritmo utilizado para calcular $y = \varphi(x)$, siempre existirá un error de al menos $|y|$ eps en el resultado.
- Por otra parte, el redondeo producido sobre los datos de entrada x causará un error de entrada que podemos acotar

  $$|\Delta^{(0)}x| \le |x|\,\text{eps} \tag{2.109}$$
- De las dos observaciones previas se desprende que TODO algoritmo para computar $y = \varphi(x)$ tendrá este error incorporado, que podemos escribir así

  $$\boxed{|\Delta^{(0)}y| = (|J\varphi(x)|\cdot|x| + |y|)\,\text{eps}.} \tag{2.110}$$

  Este error recibe el nombre de *error inherente*. (OJO, en la última expresión las barras de módulo deben interpretarse componente a componente.)
- Llamaremos a los errores intermedios de redondeo α_i como *inofensivos* si su contribución al error total Δy es, como máximo, del orden de magnitud del error inherente, es decir, si se cumple que

  $$|J\psi^{(i)}(x^{(i)})\,\alpha_i| \approx |\Delta^{(0)}y|. \tag{2.111}$$
- Si todos los errores de redondeo son inofensivos, el algoritmo es **bien comportado** o **numéricamente estable**.
- De acuerdo con las definiciones previas, un algoritmo puede ser numéricamente más confiable que otro, pero ninguno de los dos ser numéricamente estable. Naturalmente, si ambos son estables, siempre deberemos utilizar aquél que sea numéricamente más confiable.

2.7.4. Ejemplo

Dados $p > 0$, $q > 0$ y $p \gg q$, determínese la raíz

$$y = \varphi(p, q) = -p + \sqrt{p^2 + q} \tag{2.112}$$

con el menor valor absoluto de la ecuación cuadrática

$$y^2 + 2p\,y - q = 0 \tag{2.113}$$

Antes, cuando estudiamos la propagación de errores sin tener en cuenta el redondeo, vimos que el error relativo del resultado quedaba determinado por la siguiente expresión

$$\varepsilon_y \approx \frac{-p}{\sqrt{p^2+q}}\,\varepsilon_p + \frac{p+\sqrt{p^2+q}}{2\sqrt{p^2+q}}\,\varepsilon_q \tag{2.114}$$

Ahora, como $p > 0$ y $q > 0$ tenemos que

$$\left|\frac{-p}{\sqrt{p^2+q}}\right| < 1, \qquad \text{y} \qquad \left|\frac{p+\sqrt{p^2+q}}{2\sqrt{p^2+q}}\right| < 1. \tag{2.115}$$

Por lo tanto, estamos tratando un **problema bien comportado**. El error inherente será

$$\Delta y^{(0)} = \left(\left|\frac{\partial\varphi}{\partial p}\right| p + \left|\frac{\partial\varphi}{\partial q}\right| q + |y|\right) \text{eps} \tag{2.116}$$

que podemos fácilmente transformar en un error inherente relativo

$$\begin{aligned} \varepsilon_{y^{(0)}} &= \left(\left|\frac{\partial\varphi}{\partial p}\right| \frac{p}{|y|} + \left|\frac{\partial\varphi}{\partial q}\right| \frac{q}{|y|} + 1\right) \text{eps} \\ &\leq \boxed{3\,\text{eps}} \end{aligned} \tag{2.117}$$

Ahora que tenemos el error inherente, podremos determinar si los errores de redondeo del algoritmo que utilicemos son inofensivos o no. Vamos a considerar dos algoritmos para el cálculo de $y = -p + \sqrt{p^2+q}$. Con cada algoritmos seguiremos el siguiente orden en las operaciones:

`Algoritmo 1`	`Algoritmo 2`
$s = p^2$	$s = p^2$
$t = s + q$	$t = s + q$
$u = \sqrt{t}$	$u = \sqrt{t}$
$y = -p + u$	$v = p + u$
	$y = q/v$

Los primeros tres pasos son iguales, por lo que vamos a concentrar nuestro análisis a las operaciones siguientes. En ambos casos, el cálculo de u incluirá error de redondeo que podemos escribir como

$$\text{FL}(u) = \text{FL}(\sqrt{t}) = (1-\varepsilon)\sqrt{t}, \qquad |\varepsilon| \leq \text{eps} \tag{2.118}$$

por lo que

$$\Delta u = \varepsilon\sqrt{t} = \varepsilon\sqrt{p^2+q}. \tag{2.119}$$

Consideremos primero el Algoritmo 1. La función "transformación restante" es, en este caso,

$$\psi(u) = -p + u, \tag{2.120}$$

Ahora podemos calcular la propagación del error de redondeo y su impacto en el resultado final haciendo

$$\frac{\not{u}}{\psi}\frac{\partial\psi(u)}{\partial u}\frac{\Delta u}{\not{u}} = \frac{\sqrt{p^2+q}}{-p+\sqrt{p^2+q}}\,\varepsilon = \frac{1}{q}\left(p\sqrt{p^2+q} + p^2 + q\right)\,\varepsilon = k\,\varepsilon. \tag{2.121}$$

Recordando que $p, q > 0$, tenemos que

$$k > \frac{2p^2}{q} > 0 \tag{2.122}$$

y como estamos trabajando en el caso $p \gg q$, este número será grande. En consecuencia, se ve que el algoritmo 1 es numéricamente inestable ya que la propagación de un error de redondeo intermedio resulta muy superior al error inherente.

Consideremos ahora el Algoritmo 2. En este caso, la función transformación restante será

$$\psi(u) = \frac{q}{p+u}, \tag{2.123}$$

por lo cual, el error Δu se propagará ahora de la siguiente manera

$$\varepsilon_y = \frac{\cancel{\psi}}{\psi} \frac{\partial \psi(u)}{\partial u} \frac{\Delta u}{\cancel{\psi}} = \frac{\cancel{p + \sqrt{p^2+q}}}{\cancel{q}} \frac{-\cancel{q}}{(p+\sqrt{p^2+q})^{\cancel{2}}} \sqrt{p^2+q}\,\varepsilon = \frac{-\sqrt{p^2+q}}{p+\sqrt{p^2+q}}\,\varepsilon = k\,\varepsilon. \tag{2.124}$$

Evidentemente

$$|k| < 1, \tag{2.125}$$

y, por lo tanto, se ve que el algoritmo 2 es numéricamente estable.

En clase relacionaremos este problema con el de la cancelación numérica.