CAPITULO IV. ARITMETICA DEL COMPUTADOR

1. REPRESENTACION DE LOS NUMEROS

Cuando se usa una calculadora o una computadora digital para realizar cálculos numéricos, se debe considerar un error inevitable, el llamado error de redondeo. Este error se origina porque la aritmética realizada en una máquina involucra números con sólo un número finito de dígitos, con el resultado de que muchos cálculos se realizan con representaciones aproximadas de los números verdaderos.

El ordenador recibe, normalmente, información en decimal, que es transformada a binario por un programa interno. Posteriormente efectúa las operaciones pertinentes, pasa el resultado a decimal e informa al usuario de este resultado.

Así pues, en principio deberíamos hablar de las representación de los números en binario (la forma usual de trabajar del ordenador), pero para facilitar la comprensión, usaremos la representación decimal.

La representación de los números en el sistema decimal no es única (considerar que $0.999... = 1.000...$). Esto también es cierto para otros sistemas de numeración, y en particular para el sistema binario. Para evitar estas ambiguedades, siempre nos referiremos a la representación finita (1 en vez de 0.999...).

En general los ordenadores digitales trabajan con un número fijo (finito) de posiciones, **la longitud de palabra**, cuando representan un número internamente. Esta longitud n depende de la máquina, y además algunas permiten extensiones a múltiplos enteros de n ($2n$, $3n$, ...) que posibilitan una mayor exactitud si se necesita. Una palabra de longitud n se puede utilizar de distintas formas para representar un número:

- la **representación de punto fijo** especifica un número fijo n_1 de lugares enteros, y un número fijo n_2 de decimales, de modo que $n = n_1 + n_2$. En esta representación, la posición del punto decimal está fija y son pocos los dispositivos que la utilizan (ciertas máquinas de calcular, o máquinas de tipo comercial).

Ejemplo 1.

Sea $n = 8$, $n_1 = 3$ y $n_2 = 5$. En el sistema decimal se tienen las siguientes representaciones:

$$81.46 \longrightarrow \underline{| \ 0 \ 8 \ 1 \ | \ 4 \ 6 \ 0 \ 0 \ 0 \ |}$$

$$0.0002 \longrightarrow \underline{| \ 0 \ 0 \ 0 \ | \ 0 \ 0 \ 0 \ 2 \ 0 \ |} \ .$$

- Más importante, sobre todo en el cálculo cientifico, es la **representación en punto flotante**. La posición del punto decimal con respecto al primer dígito se expresa con un número separado, denominado exponente. Así se obtiene la notación científica:

$$x = a * b^t, \qquad \text{con} \qquad |a| < 1, \qquad b \in \mathcal{N}, \qquad t \in \mathcal{Z}$$

donde b es la **base** del sistema de numeración, t es un exponente llamado **característica** y a se llama la **mantisa**. Además si $|a| \geq b^{-1}$, es decir que el primer dígito despues del punto raiz no es cero, se dice que la representación es **normalizada**.

Naturalmente, en un ordenador digital sólo se dispone de un número finito de posiciones para representar un número (n, la longitud de la palabra), por lo que cada ordenador tendrá reservadas m posiciones para la mantisa y c posiciones para la característica ($n = m + c$).

Ejemplo 2.

Sea $m = 5$, $c = 2$ ($\Rightarrow n = 7$), y $b = 10$. El número 5420.0 se representaría:

$$0.54200 * 10^4 \longrightarrow \underline{| \ 5 \ 4 \ 2 \ 0 \ 0 \ | \ 0 \ 4 \ |} \ .$$

Esta notación no es única porque ese número se podría haber expresado también como:

$$0.05420 * 10^5 \longrightarrow \underline{| \ 0 \ 5 \ 4 \ 2 \ 0 \ | \ 0 \ 5 \ |} \ .$$

La primera notación es la representación normalizada.

Son **dígitos significativos** de un número todos los dígitos de la mantisa sin contar los primeros ceros.

Los números m y c, junto con la base b de la representación de un número, determinan un conjunto $\mathcal{A} \subset \mathcal{R}$ de números reales que se pueden representar de forma exacta en una máquina; estos números se denominan **números de la máquina**.

Ejemplo 3.

Imaginemos que una computadora pueda representar exactamente el número decimal 179.015625 , y que el siguiente número de máquina más pequeño sea 179.015609 , mientras que el número de máquina más grande siguiente es 179.015640 . Esto significa que nuestro número de máquina original debe representar no solamente a 179.015625, sino a un número infinito de números reales que estén entre este número y su número de máquina más cercano.

Hay diversos conceptos, relacionados con estas representaciones, que describiremos a continuación: truncamiento, redondeo, underflow y overflow.

Los dos primeros conceptos, **truncamiento** y **redondeo**, son relativos a la forma de representar los números que no pertenecen al conjunto $\mathcal{A}$ definido anteriormente. Supongamos que tenemos una máquina con m dígitos de mantisa y c dígitos de característica. Ya sabemos que el conjunto $\mathcal{A}$ de los números reales que se pueden representar exactamente es un conjunto finito. Sea entonces $x \in \mathcal{R}$, $x \notin \mathcal{A}$ (por ejemplo, por el mayor número de dígitos de mantisa),

$$x = a * b^t, \qquad \text{donde} \qquad b^{-1} \leq |a| < 1$$

$$a = 0.\alpha_1 \ \alpha_2 \ ...\alpha_m \ \alpha_{m+1} \ ..., \qquad 0 \leq \alpha_i \leq b-1, \quad \alpha_1 \neq 0 \ .$$

Consideremos ahora

$$a' = \begin{cases} \pm 0.\alpha_1 \ \alpha_2 \ ...\alpha_m & 0 \leq \alpha_{m+1} \leq \frac{b}{2} - 1 \\ \pm 0.\alpha_1 \ \alpha_2 \ ...\alpha_m + b^{-m} & \frac{b}{2} \leq \alpha_{m+1} \leq b-1 \end{cases} \qquad (IV.1a)$$

esto es, se suprimen los dígitos que sigan al último representable si $0 \leq \alpha_{m+1} \leq \frac{b}{2} - 1$, y se aumenta en una unidad α_m si $\alpha_{m+1} \geq \frac{b}{2}$, y después se suprimen los dígitos que sigan al último representable. Entonces, está claro que

$$fl(x) = a' * b^t \in \mathcal{A} \ . \qquad (IV.1b)$$

Por ejemplo, supongamos que se desease utilizar el número $0.34826 * 10^4$ en una máquina en que $m = 4$ y $c = 2$. El truncamiento consiste en suprimir todos los dígitos que existen tras el último representable, sin mirar cuál es el dígito siguiente; por el contrario, el redondeo suprime los dígitos que sigan al último representable si el siguiente es menor o igual que 4 o aumentan en una unidad el último representable, suprimiendo los restantes, si el que sigue es mayor o igual que 5. Así el número anterior se representaría como

$$\underline{|\ 3\ 4\ 8\ 2\ |\ 0\ 4\ |} \quad \text{con truncamiento ,}$$
$$\underline{|\ 3\ 4\ 8\ 3\ |\ 0\ 4\ |} \quad \text{con redondeo .}$$

La forma usual de utilizar los números es la última, porque se usa el número de la máquina que está más próximo al que se necesita.

Entonces, desde un número $x \notin \mathcal{A}$ se puede construir otro número $fl(x) \in \mathcal{A}$, naturalmente haciendo un error de redondeo. Para el error relativo de $fl(x)$ se tiene

$$\Big|\frac{fl(x) - x}{x}\Big| = \Big|\frac{a' * b^t - a * b^t}{a * b^t}\Big| = \Big|\frac{a' - a}{a}\Big| < \frac{\frac{b}{2}\ b^{-(m+1)}}{|a|} \leq \frac{b}{2} b^{-m} \ ,$$

siendo $|a| \geq b^{-1}$. En el caso de $b = 10$

$$\Big|\frac{fl(x) - x}{x}\Big| < \frac{5.0 * 10^{-(m+1)}}{|a|} \leq 5.0 * 10^{-m} \ ,$$

y si ponemos $\nu = 5.0 * 10^{-m}$, se puede poner que

$$fl(x) = x\ (1 + \varepsilon) \ , \qquad \text{donde} \ \ |\varepsilon| \leq \nu \ .$$

El valor ν se llama **precisión de la máquina**.

En ocasiones, el número x no puede ser representado por la máquina al efectuar el redondeo, como indicamos en los cuatro casos siguientes:

$m = 4, c = 2, b = 10$
$fl(0.31794 * 10^{110}) = 0.3179 * 10^{110} \notin \mathcal{A}$
$fl(0.99997 * 10^{99}) = 0.1000 * 10^{100} \notin \mathcal{A}$
$fl(0.012345 * 10^{-99}) = 0.1235 * 10^{-100} \notin \mathcal{A}$
$fl(0.54321 * 10^{-110}) = 0.5432 * 10^{-110} \notin \mathcal{A}$.

En los dos primeros casos, el exponente es demasiado grande para caber en los lugares reservados para él, y se produce un **overflow** (rebasamiento del valor máximo), y en los dos últimos casos el exponente es demasiado pequeño para caber en los lugares reservados

para él, y se produce un **underflow** (rebasamiento del valor mínimo). Los dos últimos casos tienen una posible solución, y es la de prevenirlos definiendo:

$fl(0.012345 * 10^{-99}) = 0.0123 * 10^{-99} \in \mathcal{A}$, (no normalizado)

$fl(0.54321 * 10^{-110}) = 0.0 \in \mathcal{A}$,

pero ahora el redondeo puede no verificar que

$$fl(x) = x\ (1 + \varepsilon)\ , \qquad \text{con}\ \ |\varepsilon| \leq \nu\ .$$

Los ordenadores digitales tratan los fenómenos de overflow y de underflow de forma diferentes, y siempre como irregularidades del cálculo. En cierto casos, al producirse alguno de los rebasamientos, el ordenador continúa los cálculos con el mayor valor permitido (o cero si se trata de un underflow) mostrando o no un mensaje de aviso de lo que ha ocurrido; en otros se muestra un mensaje de error y detiene el programa. Los rebasamientos pueden ser evitados si se hacen escalados adecuados de los datos, y si durante los cálculos se hacen chequeos, efectuando reescalados si fuese preciso.

El uso frecuente de la aritmética de redondeo en computadoras lleva a la siguiente definición: se dice que **el número p^* aproxima a p con m dígitos significativos** (o cifras) si m es el entero más grande no negativo para el cual

$$\left|\frac{p^* - p}{p}\right| \leq 5.0 * 10^{-m}\ .$$

La razón por la cual se usa el error relativo en la definición es que se desea obtener un concepto continuo. Por ejemplo, para que p^* aproxime a 1000 con cuatro cifras significativas, p^* debe satisfacer

$$\left|\frac{p^* - 1000}{1000}\right| \leq 5.0 * 10^{-4}\ , \qquad \text{y eso implica que} \qquad 999.5 \leq p^* \leq 1000.5\ .$$

Pasamos ahora a ver brevemente algunas **representaciones internas** usadas por el ordenador para almacenar los números enteros y los reales.

1. **Representación interna de números enteros en "magnitud-signo"**.

La escritura de un número en el sistema binario es la manera más sencilla de representarlo mediante un patrón de bits. La idea más simple para representar el signo menos es reservar un bit para ello, de forma que si dicho bit vale 0 el número es positivo, y si vale 1 es negativo. Normalmente se suele utilizar el bit situato más a la izquierda.

Ejemplo 4.

Representar los número enteros 17 y -17 en magnitud-signo con 8 bits.

La representación binaria del número 17 es 10001. Entonces, en la representación magnitud-signo con 8 bits del número positivo 17 tendremos que el primer bit, que es el que denota el signo, es cero: 00010001. Para el número negativo -17 se obtiene 10010001.

Ejemplo 5.

¿Qué números decimales representan las series de 8 bits 11100010 y 00111011 codificados en magnitud-signo?

La primera serie 11100010 tiene un uno en el primer bit, indicando que el número representado es un número negativo. Los demás dígitos son la representación binaria del número

$$1100010 = 2 + 32 + 64 = 98 \ .$$

Entonces la primera serie representa al número entero -98.

La segunda serie 00111011 tiene un cero en el primer bit, indicando que el número representado es un número positivo. Los demás dígitos son la representación binaria del número

$$111011 = 1 + 2 + 8 + 16 + 32 = 59 \ .$$

Entonces la segunda serie representa al número entero 59.

2. Representación interna de números enteros en "notación en exceso".
La representación en magnitud-signo es una manera muy natural de codificar números en binario. Sin embargo, hay otras formas de codificación que permiten diseñar circuitos electrónicos más simples para interpretarlas. Una de éstas es la **notación en exceso**. La representación en exceso con p bits de un número decimal entero N consiste en codificar N como el equivalente binario del número $N + 2^{p-1}$, que se denomina **característica** de N con p bits.

Por ejemplo, la tabla siguiente muestra la notación en exceso con 4 bits

$0000 = -8$	$1000 = 0$
$0001 = -7$	$1001 = 1$
$0010 = -6$	$1010 = 2$
$0011 = -5$	$1011 = 3$
$0100 = -4$	$1100 = 4$
$0101 = -3$	$1101 = 5$
$0110 = -2$	$1110 = 6$
$0111 = -1$	$1111 = 7$

El nombre "notación en exceso" se debe a la diferencia que hay entre el número codificado y el número binario directo que representa el patrón de bits. Nótese que a diferencia de la codificación en magnitud-signo, en la notación en exceso los números positivos tienen el primer bit igual a 1, mientras que los números negativos tienen el primer bit igual a 0.

Ejemplo 6.

Representar en exceso con 8 bits los números enteros 23 y -49.

Para dar la representación en exceso del número 23 tenemos que hallar la representación binaria del número $23 + 2^{8-1} = 23 + 2^7 = 23 + 128 = 151$. Tal representación es 10010111 que coincide con la representación en exceso con 8 bits de 23. De manera parecida, para hallar la representación en exceso del número -49 tenemos que hallar la representación binaria del número $-49 + 2^{8-1} = -49 + 2^7 = -49 + 128 = 79$. Tal representación es 1001111, y ahora para tener la representación en exceso se necesitan añadir ceros a la izquierda; entonces, la representación en exceso con 8 bits de -49 es 01001111.

Ejemplo 7.

¿Qué números decimales representan los códigos en exceso 01100111 y 10010011?

Para el primer código: $01100111 = 1+2+4+32+64 = 103$, entonces $N+128 = 103$ lo cual implica $N = -25$.

Para el segundo código: $10010011 = 1+2+16+128 = 147$, entonces $N+128 = 147$ lo cual implica $N = 19$.

3. Representación interna de números enteros en "complemento a dos".

La representación en **complemento a dos** es una manera muy útil de codificar un número debido a que facilita enormemente las operaciones algebraicas. Para obtener el complemento a dos de un número binario hay que considerar en primer lugar el complemento a uno cuya definición es la siguiente: el **complemento a uno** de un número binario es el número que se obtiene al cambiar los ceros por unos y los unos por ceros. Conocido el complemento a uno, el complemento a dos se obtiene fácilmente: el **complemento a dos** de un número binario se obtiene sumando 1 al complemento a uno.

Ahora, la **representación en complemento a dos** de un número consiste en escribir los números positivos como su equivalente en el sistema binario, y los números negativos como el complemento a dos del equivalente en el sistema binario de su valor absoluto.

Para decodificar un número decimal representado en complemento a dos se procede del modo siguiente:

- si el primer bit de la izquierda es 0 el número es positivo. Entonces, el número representado es el equivalente del número binario que forma el resto de los bits.
- si el primer bit de la izquierda es 1 el número es negativo. Entonces el número representado es el opuesto del equivalente decimal del número binario que forma su complemento a dos.

Ejemplo 8.

Representar con 8 bits en complemento a dos los números decimales 17 y -17.

La representación binaria del número 17 es 10001, entonces la representación con 8 bits en complemento a dos de 17 se obtiene añadiendo ceros a la izquierda: 00010001.

Para la representación con 8 bits en complemento a dos de -17 tenemos que realizar el complemento a dos de la representación binaria del su valor absoluto 17. Primero se pasa de 00010001 a su complemento a uno: 11101110. Ahora, se hace el complemento a dos, es decir se le suma 1: $11101110 + 1 = 11101111$. Esta es la representación con 8 bits en complemento a dos de -17.

Ejemplo 9.

¿Qué números decimales representan las series de 8 bits 00101011 y 10101011 codificadas en complemento a dos?

La primera serie 00101011 tiene un cero en el primer bit, indicando que el número representado es un número positivo. Los demás dígitos son la representación binaria del número 43 ($101011 = 1+2+8+32 = 43$). Entonces la primera serie representa al número entero 43.

La segunda serie 10101011 tiene un uno en el primer bit, indicando que el número representado es un número negativo y que entonces tenemos que realizar la operación de complemento a dos. Es decir, primero tenemos que pasar la representación 10101011 a complemento a uno: 01010100 y ahora a complemento a dos añadiendo uno: $01010100 + 1 = 01010101$. Finalmente el número buscado es el opuesto del equivalente decimal: $01010101 = 1 + 4 + 16 + 64 = 85$, es decir -85.

4. Representación interna de números reales en "punto flotante".

Los números fraccionarios y reales se introducen en el ordenador **en punto flotante**. Esta representación consiste en escribirlos en forma exponencial binaria normalizada y codificar tres campos: el signo, el exponente y la matisa. Cada uno de los campos se codifica de la manera siguiente:

1. El bit de signo se pone a 0 cuando el número es positivo y a 1 cuando el número es negativo.
2. El campo exponente se codifica usualmente mediante la notación en exceso.
3. El campo mantisa se codifica como el equivalente binario directo del número decimal dado.

Si se usan 32 bits para la representación pueden dividirse del modo siguiente: 1 bit para el signo, 7 bits para el exponente y 24 bits para la mantisa:

1 bit (signo) | 7 bits (exponente) | 24 bits (mantisa)

Ejemplo 10.

Representar en punto flotante con 32 bits, 1 de signo, 7 de exponente y 24 de mantisa, los números decimales 104.3125 y -13506.96875.

El primer número 104.3125 es positivo, entonces el primer bit será un cero. La representación binaria del número es: 1101000.0101, cuya forma exponencial normalizada es 0.11010000101×2^7. El exponente (7) se codifica en exceso con 7 bits: $7+2^{7-1} = 7+2^6 = 7 + 64 = 71$ cuya represenatción binaria es 1000111. Finalmente, la mantisa tiene 11 bits (11010000101) y se completa con 13 ceros a la derecha. Entonces, la representación en punto flotante con 32 bits del número 104.3125 es

$$0 \mid 1000111 \mid 11010000101000000000000 \ .$$

De manera parecida, para el segundo número -13506.96875, notamos que es negativo, entonces el primer bit será un uno. La representación binaria del valor absoluto del número es: 11010011000010.11111. Su forma exponencial normalizada es $0.1101001100001011111 \times 2^{14}$. El exponente (14) se codifica en exceso con 7 bits: $14 + 2^{7-1} = 14 + 2^6 = 14 + 64 = 78$ cuya represenatción binaria es 1001110. Finalmente, la mantisa tiene 19 bits (1101001100001011111) y se completa con 5 ceros a la derecha. Entonces, la representación en punto flotante con 32 bits del número 104.3125 es

$$1 \mid 1001110 \mid 11010011000010111111000000 \ .$$

2. INTRODUCCION A LA ARITMETICA DE PUNTO FLOTANTE

Además de dar una representación inexacta de los números, la aritmética realizada en la computadora no es exacta. Sin embargo, usando números con representación en punto flotante con m dígitos de mantisa, las operaciones aritméticas elementales no se pueden siempre ejecutar de manera exacta, y los resultados de las operaciones no necesariamente son números de la máquina aunque los operandos lo sean. Por ello no se puede esperar reproducir de forma exacta las operaciones aritméticas en un ordenador digital. Deberemos contentarnos con sustituirlas por otras $(\oplus, \ominus, \otimes, \oslash)$ llamadas **operaciones de punto flotante**, que las aproximen tanto como sea posible. Esto se puede conseguir, por ejemplo, definiéndolas con la ayuda del redondeo:

suma: $x \oplus y = fl(fl(x) + fl(y))$
resta: $x \ominus y = fl(fl(x) - fl(y))$
multiplicación: $x \otimes y = fl(fl(x) * fl(y))$
división: $x \oslash y = fl(fl(x)/fl(y))$.

Esta aritmética idealizada corresponde a efectuar la aritmética exacta en la representación del punto flotante de x e y, y luego a la conversión del resultado exacto a su representación de punto flotante. Se pueden probar las relaciones

$$x \oplus y = (x + y)\ (1 + \varepsilon_1) \qquad (IV.2a)$$

$$x \ominus y = (x - y)\ (1 + \varepsilon_2) \qquad (IV.2b)$$

$$x \otimes y = (x * y)\ (1 + \varepsilon_3) \qquad (IV.2c)$$

$$x \oslash y = (x/y)\ (1 + \varepsilon_4)\ , \qquad (IV.2d)$$

donde $|\varepsilon_i| \leq \mu_i\ \nu$, y μ_i es un entero $\mu_i \geq 1$, que depende del tipo de máquina usada.

Podemos comprobar que las operaciones en punto flotante no verifican las reglas aritméticas normales:

a) $x \oplus y = x$ **no implica que** $y = 0$. Esta igualdad es cierta para todo y tal que $|y| < \frac{\nu}{b}|x|$ (b es la base del sistema de numeración usado). Según esto, la precisión de la máquina, ν, debería definirse como el menor número positivo g de la máquina, para el cual se cumple $1 \oplus g > 1$, $\nu = \min\{g \in \mathcal{A}\ /\ 1 \oplus g > 1 \ \text{ y } \ g > 0\}$.

Ejemplo 11.

$m = 3, c = 2, b = 10$
$x = 0.123 * 10^0$
$y = 0.000061 = 0.61 * 10^{-4} \leq 0.5 * 10^{-3} * 0.123 = 0.615 * 10^{-4}$

Entonces

$x \oplus y = x$

b) **no asociatividad**: $a \oplus (b \oplus c)$ puede ser diferente de $(a \oplus b) \oplus c$.

Ejemplo 12.

$m = 8, c = 2, b = 10$
$a = 0.23371258 * 10^{-4}$

$b = 0.33678429 * 10^2$

$c = -0.33677811 * 10^2$

Entonces

$a \oplus (b \oplus c) = 0.23371258 * 10^{-4} \oplus 0.61800000 * 10^{-3} = 0.64137126 * 10^{-3}$

$(a \oplus b) \oplus c = 0.33678452 * 10^2 \ominus 0.33677811 * 10^2 = 0.64100000 * 10^{-3}$

y el resultado exacto es

$a + b + c = 0.64137126 * 10^{-3}$.

Esto ha ocurrido porque cuando se restan dos números del mismo signo, se produce un efecto de cancelación si ambos coinciden en uno o más dígitos con respecto al mismo exponente (los dígitos comunes desaparecen). A pesar de que, cuando $x, y \in \mathcal{A}$, su diferencia también es un elemento de $\mathcal{A}$, y por lo tanto no hay errores de redondeo adicionales, veremos que la cancelación es un efecto peligroso cuando se trata de propagación de errores previos (cuando x e y provienen de cálculos que han necesitado redondeo).

c) **no distributividad**: $a \otimes (b \oplus c)$ puede ser diferente de $(a \otimes b) \oplus (a \otimes c)$.

Ejemplo 13.

$m = 2, c = 2, b = 10$

$a = 0.94 * 10^2$

$b = 0.33 * 10^2$

$c = -0.32 * 10^2$

Entonces

$a \otimes (b \oplus c) = 0.94 * 10^2 \otimes 0.1 * 10 = 0.94 * 10^2$

$(a \otimes b) \oplus (a \otimes c) = 0.31 * 10^4 \ominus 0.30 * 10^4 = 0.1 * 10^3$

y el resultado exacto es

$a * (b + c) = 0.94 * 10^2$.

Las operaciones aritméticas $+, -, *, /$, junto a las funciones para las que se hayan especificado sustituciones (por ejemplo raíz cuadrada, funciones trigonometricas, etc...) se llaman **funciones elementales**.

3. PROPAGACION DEL ERROR

Hemos comprobado que la no associatividad de la suma y la no distributividad del producto en un ordenador pueden provocar la obtención de resultados diferentes dependiendo de la técnica que se utilice para efectuar las operaciones. Entonces la propagación del error es un efecto muy importante a tener en cuenta, y se debe evitar en lo posible.

Generalmente, un problema matemático puede ser esquematizado en la manera siguiente: con un número finito de datos iniciales $x_1, x_2, ..., x_n \in \mathcal{R}$ queremos calcular un número finito $y_1, y_2, ..., y_m \in \mathcal{R}$ de resultados. Eso corresponde a asignar una función

$$\phi^{(i)}: D_i \to D_{i+1}, \qquad i = 0, ..., r, \qquad D_j \subseteq \mathcal{R}^{n_j} \tag{IV.3}$$

donde $\phi = \phi^{(r)} \circ \phi^{(r-1)} \circ ... \circ \phi^{(0)}$ y $D_0 = D \subset \mathcal{R}^n$, $D_{r+1} \subseteq \mathcal{R}^{n_r+1} \equiv \mathcal{R}^m$.

Entonces el problema es analizar como un error Δx y los errores de redondeo que se hacen en el cálculo se propagan y cambian el resultado final $y = \phi(x)$. Consideremos

$$\phi\colon D \subset \mathcal{R}^n \to \mathcal{R}^m, \qquad \phi(x) = \begin{pmatrix} \phi_1(x_1, \ldots, x_n) \\ \vdots \\ \phi_m(x_1, \ldots, x_n) \end{pmatrix}$$

con las funciones componentes continuas y con derivadas primeras continuas. Para hacer los calculos más sencillos, analicemos antes sólo la propagación del error sobre el dato inicial Δx, con un procedimiento del primer orden (si ε y η son dos números muy pequeños, entonces el producto $\varepsilon\ \eta$ se puede despreciar con respecto a ε y η).

Si x^* es una aproximación de x, los errores absolutos serán

$$\Delta x_i = x_i^* - x_i, \qquad \Delta x = x^* - x, \qquad \Delta y_i = \phi_i(x^*) - \phi_i(x)\ .$$

Si usamos el desarrollo en serie de Taylor hasta al primer orden

$$\Delta y_i = y_i^* - y_i = \phi_i(x^*) - \phi_i(x) \approx \sum_{j=1}^{n} (x_j^* - x_j)\ \frac{\partial \phi_i(x)}{\partial x_j} = \sum_{j=1}^{n} \Delta x_j\ \frac{\partial \phi_i(x)}{\partial x_j}\ , \qquad (IV.4a)$$

ó en notación matricial

$$\Delta y = \begin{pmatrix} \Delta y_1 \\ \vdots \\ \Delta y_m \end{pmatrix} \approx \begin{pmatrix} \frac{\partial \phi_1(x)}{\partial x_1} & \cdots & \frac{\partial \phi_1(x)}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial \phi_m(x)}{\partial x_1} & \cdots & \frac{\partial \phi_m(x)}{\partial x_n} \end{pmatrix} \begin{pmatrix} \Delta x_1 \\ \vdots \\ \Delta x_n \end{pmatrix} = D\phi(x) \cdot \Delta x\ , \qquad (IV.4b)$$

con $D\phi(x)$ la matriz Jacobiana. Aquí el factor de proporcionalidad $\frac{\partial \phi_i(x)}{\partial x_j}$ mide la sensibilidad con la cual y "reacciona" a las variaciones absolutas Δx_j de x_j. La fórmula análoga para la propagación de los errores relativos es:

$$RE_{y_i} \approx \sum_{j=1}^{n} \frac{x_j}{\phi_i(x)}\ \frac{\partial \phi_i(x)}{\partial x_j}\ RE_{x_j} = \sum_{j=1}^{n} \frac{\Delta x_j}{\phi_i(x)}\ \frac{\partial \phi_i(x)}{\partial x_j}\ . \qquad (IV.5)$$

Aquí el factor $\frac{x_i}{\phi(x)}\ \frac{\partial \phi(x)}{\partial x_i}$ (a menudo se le conoce como **índice de condicionamento**) indica cómo el error relativo en x_i repercute en el error relativo de y. Si el índice de condicionamento es de valor absoluto suficientemente grande, errores relativos pequeños en los datos iniciales producen errores relativos muy grandes en los resultados. En ese caso se dice que el problema está **mal planteado**.

La propagación del error relativo en las operaciones elementales viene dada por:

1. $\phi(x, y) = x * y \quad \Rightarrow \quad RE_{x*y} \approx RE_x + RE_y$
2. $\phi(x, y) = x/y \quad \Rightarrow \quad RE_{x/y} \approx RE_x - RE_y$
3. $\phi(x, y) = x \pm y \quad \Rightarrow \quad RE_{x\pm y} \approx \frac{x}{x\pm y}\ RE_x \pm \frac{y}{x\pm y}\ RE_y$
4. $\phi(x) = \sqrt{x} \quad \Rightarrow \quad RE_{\sqrt{x}} \approx \frac{1}{2}\ RE_x\ .$

Consideremos $\phi(x) = \sqrt{x}$. Entonces $\phi'(x) = \frac{1}{2\sqrt{x}}$ y el error relativo es:

$$\frac{|\phi(x) - \phi(x^*)|}{|\phi(x)|} \approx |\phi'(x^*)| \left| \frac{x - x^*}{\phi(x)} \right| = \frac{1}{2} \left| \frac{x - x^*}{\sqrt{x x^*}} \right| \approx \frac{1}{2} \left| \frac{x - x^*}{x} \right| ,$$

por lo que el error relativo en $\phi(x^*)$ es aproximadamente la mitad del error relativo en x^*, y por lo tanto, la operación de calcular la raíz cuadrada es, desde el punto de vista del error relativo, una operación segura.

Es más, también en la multiplicación, división y extracción de raíz, los errores relativos en los datos iniciales no se notan de manera fuerte en el resultado. Eso pasa también en la suma si los operandos x e y tienen el mismo signo: los indices de condicionamento $x/(x+y)$, $y/(x+y)$ tienen un valor entre cero y uno, y su suma es uno, luego

$$|RE_{x+y}| \leq \max(RE_x, RE_y) \ .$$

Si en la operación de suma los operandos x e y tienen signo contrario, por lo menos uno de los factores $x/(x+y)$, $y/(x+y)$, es mayor que uno, y entonces, por lo menos uno de los errores relativos RE_x, RE_y es mayor. Esa amplificación del error es todavía mayor si $x \approx -y$, porque en ese caso en la expresión de $x + y$ los dos terminos se cancelan.

Ejemplo 14.

Queremos estudiar el error obtenido para hallar la suma

$$\phi(\alpha, \beta, \gamma) = \alpha + \beta + \gamma$$

con $\phi : \mathcal{R}^3 \to \mathcal{R}$.

Para el calculo de ϕ se pueden usar los dos algoritmos:

Algoritmo 1	Algoritmo 2
$\eta = \alpha + \beta$	$\eta = \beta + \gamma$
$y = \phi(\alpha, \beta, \gamma) = \eta + \gamma$	$y = \phi(\alpha, \beta, \gamma) = \alpha + \eta$.

Las decomposiciones (IV.3) de ϕ en este caso son:

$$\phi^{(0)} : \mathcal{R}^3 \to \mathcal{R}^2 \ , \qquad \phi^{(1)} : \mathcal{R}^2 \to \mathcal{R} \ .$$

Entonces los algoritmos son:

Algoritmo 1	Algoritmo 2
$\phi^{(0)}(\alpha, \beta, \gamma) = \begin{pmatrix} \alpha + \beta \\ \gamma \end{pmatrix} \in \mathcal{R}^2$	$\phi^{(0)}(\alpha, \beta, \gamma) = \begin{pmatrix} \beta + \gamma \\ \alpha \end{pmatrix} \in \mathcal{R}^2$
$\phi^{(1)}(u, v) = u + v \in \mathcal{R}$	$\phi^{(1)}(u, v) = u + v \in \mathcal{R}$.

Usando el cálculo en punto flotante, (IV.2), se obtiene para el primer algoritmo:

$\eta = fl(\alpha + \beta) = (\alpha + \beta)\ (1 + \varepsilon_1)$

$\overline{y} = fl(\eta + \gamma) = (\eta + \gamma)\ (1 + \varepsilon_2) = [(\alpha + \beta)\ (1 + \varepsilon_1) + \gamma]\ (1 + \varepsilon_2)$

$= \alpha + \beta + \gamma + (\alpha + \beta)\ \varepsilon_1 + (\alpha + \beta + \gamma)\ \varepsilon_2 + (\alpha + \beta)\ \varepsilon_1\ \varepsilon_2 =$

$= (\alpha+\beta+\gamma)\big[1+\frac{(\alpha+\beta)}{\alpha+\beta+\gamma}\ \varepsilon_1\ (1+\varepsilon_2)+\varepsilon_2\big]$

Y para el error relativo

$$RE_y = \Big|\frac{y-\overline{y}}{y}\Big| = \Big|\frac{\alpha+\beta}{\alpha+\beta+\gamma}\ \varepsilon_1\ (1+\varepsilon_2)+\varepsilon_2\Big|$$

Y despreciando los términos de orden superior (procedimento del primer orden):

$$RE_y \approx \Big|\frac{\alpha+\beta}{\alpha+\beta+\gamma}\ \varepsilon_1+\varepsilon_2\Big|\ .$$

Si hubiésemos usado el segundo algoritmo, tendríamos:

$$RE_y \approx \Big|\frac{\beta+\gamma}{\alpha+\beta+\gamma}\ \varepsilon_1+\varepsilon_2\Big|\ .$$

Los factores de amplificación $\frac{\alpha+\beta}{\alpha+\beta+\gamma}$ y $\frac{\beta+\gamma}{\alpha+\beta+\gamma}$, respectivamente, y 1, indican cómo los errores de redondeo ε_1 y ε_2 influyen sobre el error relativo RE_y del resultado. Dependiendo de cuál de las dos cantitades $(\alpha+\beta)$ o $(\beta+\gamma)$ es menor, se prefiere uno u otro algoritmo. En el caso del ejemplo 12:

$$\frac{\alpha+\beta}{\alpha+\beta+\gamma} \approx 0.5*10^5 \qquad \frac{\beta+\gamma}{\alpha+\beta+\gamma} \approx 0.97\ .$$

Y eso explica la mayor precisión del segundo algoritmo.

Por lo que concierne a la propagación del error relativo, desde la relación $(IV.5)$, se tiene:

$$RE_y \approx \frac{\alpha}{\alpha+\beta+\gamma}\ RE_\alpha + \frac{\beta}{\alpha+\beta+\gamma}\ RE_\beta + \frac{\gamma}{\alpha+\beta+\gamma}\ RE_\gamma\ .$$

Y se puede decir que el problema está bien planteado si cada sumando α, β, γ es pequeño con respecto a $(\alpha+\beta+\gamma)$.

Ejemplo 15.

Sabemos que las raices de $a\ x^2+b\ x+c=0$, cuando $a\neq 0$, son:

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \qquad x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}\ .$$

Consideremos la ecuación cuadrática

$$x^2+62.10\ x+1=0$$

con raíces aproximadas:

$$x_1 = -0.01610723 \quad \text{y} \quad x_2 = -62.08390\ .$$

Para esta ecuación, b^2 es mucho mayor que $4ac$, así que en el cálculo de x_1 y x_2 el numerador involucra la sustracción de números casi iguales. Supongamos que efectuamos los calculos para x_1 usando aritmética de redondeo con cuatro dígitos.

$$\sqrt{b^2-4ac} = \sqrt{(62.10)^2-4.000} = \sqrt{3856.-4.000} = \sqrt{3852.} = 62.06\ ,$$

así que

$$fl(x_1) = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-62.10 + 62.06}{2.000} = \frac{-0.040}{2.000} = -0.020$$

es una representación bastante pobre de $x_1 = -0.01611$ ($RE_{x_1} \approx 0.2415$). Por otro lado, los cálculos para x_2 implican la adición de dos números casi iguales, $-b$ y $-\sqrt{b^2 - 4ac}$, y no presentan ningún problema.

$$fl(x_2) = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{-62.10 - 62.06}{2.000} = \frac{-124.2}{2.000} = -62.10$$

es una aproximación precisa de $x_2 = -62.08$ ($RE_{x_2} \approx 0.0003222$).

Para obtener una aproximación más exacta de x_1, aún con redondeo de cuatro dígitos, cambiamos la forma de la fórmula cuadrática **racionalizando** el numerador. Entonces:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}\Big(\frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}}\Big) = \frac{-2c}{b + \sqrt{b^2 - 4ac}}$$

y desde luego

$$fl(x_1) = \frac{-2.000}{62.10 + 62.06} = \frac{-2.000}{124.2} = -0.0161\ .$$

La técnica de racionalización se puede aplicar para obtener una forma alternativa también para x_2

$$x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}}\ .$$

Esta sería la expresión a usar si b fuera un número negativo. En nuestro problema, sin embargo, el uso de esta fórmula resulta no sólamente en la sustracción de números casi iguales, sino también en la división entre el resultado pequeño de esta sustracción. La inexactitud que esto produce es dramática:

$$fl(x_2) = \frac{-2c}{b - \sqrt{b^2 - 4ac}} = \frac{-2.000}{62.10 - 62.06} = \frac{-2.000}{0.040} = -50.00\ .$$

Para comprender mejor ese "mal" resultado, hagamos

$$y = \phi(p, q) = p - \sqrt{p^2 + q}, \qquad p > 0$$

y determinemos el error relativo que se propaga en y. Dado que

$$\frac{\partial \phi}{\partial p} = 1 - \frac{p}{\sqrt{p^2 + q}} = \frac{-y}{\sqrt{p^2 + q}} \qquad \text{y} \qquad \frac{\partial \phi}{\partial q} = \frac{-1}{2\sqrt{p^2 + q}}$$

se sigue que

$$RE_y \approx \frac{p}{y}\Big(\frac{-y}{\sqrt{p^2 + q}}\Big)\ RE_p + \frac{q}{y}\Big(\frac{-1}{2\sqrt{p^2 + q}}\Big)\ RE_q =$$

$$= \frac{-p}{\sqrt{p^2 + q}}\ RE_p - \frac{q}{2y\sqrt{p^2 + q}}\ RE_q =$$

$$= \frac{-p}{\sqrt{p^2+q}}\ RE_p + \frac{p+\sqrt{p^2+q}}{2\sqrt{p^2+q}}\ RE_q\ .$$

Dado que, si $q \geq 0$:

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

entonces ϕ está bien planteada si $q > 0$, y mal planteada si $q \approx -p^2$. Además, si $|q|$ es muy pequeño con respecto a p^2, obtenemos el fenómeno de la cancelación, por el cual los errores de redondeo en el cálculo previo se amplifican notablemente.

Veamos ahora la forma en que se propagan los errores en algunos casos, para poder deducir la forma correcta en que deberían realizarse las operaciones.

a) $S = a_1 + a_2 + a_3 + a_4 + a_5$

$$fl[((((a_1 + a_2) + a_3) + a_4) + a_5)] = fl(fl(fl(fl(a_1 + a_2) + a_3) + a_4) + a_5) =$$

$$= ((((a_1 + a_2)(1+\varepsilon_2) + a_3)(1+\varepsilon_3) + a_4)(1+\varepsilon_4) + a_5)(1+\varepsilon_5) =$$

$$(a_1 + a_2 + a_3 + a_4 + a_5)(1+\delta)$$

y hay que acotar

$$|\delta| = \left|\frac{S - fl[...]}{S}\right|\ ,$$

sabiendo que $|\varepsilon_i| < \nu$.

$$\begin{aligned} fl[...] = {} & a_1\ (1+\varepsilon_2)\ (1+\varepsilon_3)\ (1+\varepsilon_4)\ (1+\varepsilon_5) + \\ & a_2\ (1+\varepsilon_2)\ (1+\varepsilon_3)\ (1+\varepsilon_4)\ (1+\varepsilon_5) + \\ & a_3\ (1+\varepsilon_3)\ (1+\varepsilon_4)\ (1+\varepsilon_5) + \\ & a_4\ (1+\varepsilon_4)\ (1+\varepsilon_5) + \\ & a_5\ (1+\varepsilon_5) \approx \\ \approx {} & a_1 + a_2 + a_3 + a_4 + a_5 + \\ & a_1\ (\varepsilon_2 + \varepsilon_3 + \varepsilon_4 + \varepsilon_5) + a_2\ (\varepsilon_2 + \varepsilon_3 + \varepsilon_4 + \varepsilon_5) + \\ & a_3\ (\varepsilon_3 + \varepsilon_4 + \varepsilon_5) + a_4\ (\varepsilon_4 + \varepsilon_5) + a_5\ \varepsilon_5\ , \end{aligned}$$

donde se han desestimado los sumandos $\varepsilon_i\ \varepsilon_j$ que son despreciables respecto a ε_i.

Si ahora consideramos la situación más desfavorable (todos los ε_i iguales en signo y con el mayor valor absoluto ν), tenemos la siguiente acotación:

$$|\delta| \leq \left|\frac{4\ \nu\ a_1}{S}\right| + \left|\frac{4\ \nu\ a_2}{S}\right| + \left|\frac{3\ \nu\ a_3}{S}\right| + \left|\frac{2\ \nu\ a_4}{S}\right| + \left|\frac{\nu\ a_5}{S}\right| =$$

$$= \frac{\nu}{S}\ \left[|4\ a_1| + |4\ a_2| + |3\ a_3| + |2\ a_4| + |a_5|\ \right]\ .$$

En general, al sumar progresivamente $a_1 + a_2 + ... + a_n$, el error relativo máximo que se comete es, aproximadamente:

$$\left|\frac{S - fl[...]}{S}\right| \leq \frac{\nu}{S}\ \left[(n-1)\ (|a_1| + |a_2|) + (n-2)\ |a_3| + ... + 2\ |a_{n-1}| + |a_n|\ \right]\ .$$

Está claro, pues, que esta acotación es menor si los números $a_1, ..., a_n$ se ordenan de menor a mayor antes de sumarlos. Obtenemos así la siguiente regla práctica:
si se desea hallar $\sum_{i=0}^{n} a_i$, con n grande, y se trata de una serie convergente, entonces $\lim_{n\to\infty} a_i = 0$, y debe efectuarse la suma en orden inverso.

b) Se pretende calcular $\sum_{i=1}^{4} x_i\ y_i$. El resultado que se obtiene es:

$$\begin{aligned} fl\Big[\sum_{i=1}^{4} x_i\ y_i\Big] = \Big(\Big\{\big[x_1\ y_1\ (1+\delta_1) + x_2\ y_2\ (1+\delta_2)\big]\ (1+\delta_5) + \\ x_3\ y_3\ (1+\delta_3)\Big\}\ (1+\delta_6) + x_4\ y_4\ (1+\delta_4)\Big)\ (1+\delta_7) = \\ = x_1\ y_1\ (1+\delta_1)\ (1+\delta_5)\ (1+\delta_6)\ (1+\delta_7) + \\ x_2\ y_2\ (1+\delta_2)\ (1+\delta_5)\ (1+\delta_6)\ (1+\delta_7) + \\ x_3\ y_3\ (1+\delta_3)\ (1+\delta_6)\ (1+\delta_7) + \\ x_4\ y_4\ (1+\delta_4)\ (1+\delta_7)\ . \end{aligned} \qquad (IV.6)$$

Observamos la falta de simetría en los resultados, debida a la no conmutatividad y no asociatividad de las operaciones de punto flotante.

Para simplificar la expresión anterior, vamos a obtener unas cotas manejables de los productos $(1+\delta_i)$.

Lema. Si $|\delta_i| \le u, i = 1, ..., n$ y $n\ u \le 0.01$, entonces

$$\prod_{i=1}^{n} (1+\delta_i) \le 1 + 1.01\ n\ u\ .$$

Demostración. Antes, consideremos $0 \le x \le 0.01$, y entonces

$$1 + x \le e^x \le 1 + 1.01\ x\ .$$

La primera desigualdad es inmediata, así que sólo veremos la segunda:

$$\begin{aligned} e^x = \sum_{r=0}^{\infty} \tfrac{x^r}{r!} = 1 + x\ \big(1 + \tfrac{x}{2} + \tfrac{x^2}{3!} + ... + \tfrac{x^n}{(n+1)!} + ...\big) \le \\ \le 1 + x\ \big(1 + \tfrac{x}{2} + \tfrac{x^2}{4} + ... + \tfrac{x^n}{2^n} + ...\big) \le \\ \le 1 + x\ \big(1 + x\ (\tfrac{1}{2} + \tfrac{x}{4} + ...)\big) \le \\ \le 1 + x\ (1+x) \le 1 + 1.01\ x\ . \end{aligned}$$

Entonces, si $n \in \mathcal{N}$ y $0 \le n\ u \le 0.01$

$$(1+u)^n \le (e^u)^n = e^{n\ u} \le 1 + 1.01\ n\ u\ .$$

Ahora está claro que

$$\prod_{i=1}^{n} (1+\delta_i) \le \prod_{i=1}^{n} (1+u) = (1+u)^n \le 1 + 1.01\ n\ u\ .$$

c.q.d.

El resultado de este lema puede expresarse también como:

$$\prod_{i=1}^{n}(1+\delta_i) = 1 + 1.01\ n\ \theta\ u\ , \qquad \text{donde} \qquad |\theta| \leq 1.$$

Entonces, volviendo a (*IV*.6), y suponiendo que $n\ u \leq 0.01$ (lo que se cumple en todas las situaciones reales):

$$fl\Big(\sum_{i=1}^{n} x_i\ y_i\Big) = x_1\ y_1\ (1 + 4.04\ \theta_1\ u) + x_2\ y_2\ (1 + 4.04\ \theta_2\ u) + x_3\ y_3\ (1 + 3.03\ \theta_3\ u) + x_4\ y_4\ (1 + 2.02\ \theta_4\ u)\ , \qquad |\theta_i| \leq 1\ .$$

En general se verifica:

Teorema: Si $n\ u \leq 0.01$ entonces

$$fl\Big(\sum_{i=1}^{n} x_i\ y_i\Big) = x_1\ y_1\ (1 + 1.01\ n\ \theta_1\ u) + \sum_{i=2}^{n} x_i\ y_i\ (1 + 1.01\ (n+2-i)\ \theta_i\ u)\ , \qquad |\theta_i| \leq 1\ .$$

Muchos computadores tienen la ventaja de que en la evaluación de productos escalares $\sum_{i=1}^{n} x_i\ y_i$, pueden ir acumulando los productos parciales $x_1\ y_1$, $x_1\ y_1 + x_2\ y_2$, ..., en doble precisión, de modo que la única vez que se redondea un número con precisión simple es cuando se da el resultado final. Con esta acumulación en doble precisión, el error en el cálculo del producto interno es aproximadamente el de una sola operación.

Queremos ahora usar la fórmula matricial (*IV*.4*b*) para describir la propagación del error de redondeo en un algoritmo. Como hemos ya visto, un algoritmo para calcular una función $\phi\colon D \subset \mathcal{R}^n \to \mathcal{R}^m$, para un dado $x = (x_1, \ldots, x_n)^t \in D$ corresponde a una descomposición de la aplicación ϕ en aplicaciones elementales, diferenciables continuamente, $\phi^{(i)}$, (ver (*IV*.1)), y nos lleva desde x hasta y con resultados intermedios

$$x = x^{(0)} \to \phi^{(0)}(x^{(0)}) = x^{(1)} \to \ldots \to \phi^{(r)}(x^{(r)}) = x^{(r+1)} = y\ .$$

Denotamos con $\psi^{(i)}$ la **aplicación resto**

$$\psi^{(i)} = \phi^{(r)} \circ \phi^{(r-1)} \circ \ldots \circ \phi^{(i)} : D_i \to \mathcal{R}^m\ , \qquad i = 0, 1, 2, \ldots, r\ .$$

Entonces, $\psi^{(0)} \equiv \phi$. $D\phi^{(i)}$ y $D\psi^{(i)}$ son las matrices Jacobianas de las aplicaciones $\phi^{(i)}$ y $\psi^{(i)}$, respectivamente. Dado que las matrices Jacobianas son multiplicativa con respecto de la composición de funciones, tenemos, para $i = 0, 1, 2 \ldots r$

$$D(f \circ g)(x) = Df(g(x)) \cdot Dg(x)\ ,$$

$$D\phi(x) = D\phi^{(r)}(x^{(r)}) \cdot D\phi^{(r-1)}(x^{(r-1)}) \ldots D\phi^{(0)}(x^{(0)})\ ,$$

$$D\psi^{(i)}(x^{(i)}) = D\phi^{(r)}(x^{(r)}) \cdot D\phi^{(r-1)}(x^{(r-1)}) \dots D\phi^{(i)}(x^{(i)}) \ .$$

Con aritmética de punto flotante, los errores iniciales y de redondeo perturberán los resultados intermedios $x^{(i)}$, de manera que se obtendrá el valor aproximado $x^{*(i+1)} = fl(\phi^{(i)}(x^{*(i)}))$. Para los errores absolutos obtenemos

$$\begin{aligned} \Delta x^{(i+1)} =& x^{*(i+1)} - x^{(i+1)} = \\ =& [fl(\phi^{(i)}(x^{*(i)})) - \phi^{(i)}(x^{*(i)})] + [\phi^{(i)}(x^{*(i)}) - \phi^{(i)}(x^{(i)})] \ . \end{aligned} \tag{IV.7}$$

Desde (*IV*.4*b*) sigue

$$\phi^{(i)}(x^{*(i)}) - \phi^{(i)}(x^{(i)}) \approx D\phi^{(i)}(x^{(i)})\Delta x^{(i)} \tag{IV.8}$$

Nótese que la aplicación $\phi^{(i)}: D_i \to D_{i+1} \subseteq \mathcal{R}^{n_i+1}$ es un vector de funciones componentes $\phi_j^{(i)}: D_i \to \mathcal{R}$, $j = 1, \dots, n_i + 1$. Entonces, podemos escribir

$$fl(\phi^{(i)}(u)) = (I + E_{i+1}) \cdot \phi^{(i)}(u) \ ,$$

con I la matriz identidad y E_{i+1} la matriz diagonal cuyos elementos son los errores ε_j, $j = 1, \dots, n_i + 1$, $|\varepsilon_j| \leq \nu$. Entonces, para el primer paréntesis de la expresión (*IV*.7) sigue

$$\begin{aligned} fl(\phi^{(i)}(x^{*(i)})) - \phi^{(i)}(x^{*(i)}) =& E_{i+1} \cdot \phi^{(i)}(x^{*(i)}) \\ \approx& E_{i+1} \cdot \phi^{(i)}(x^{(i)}) = E_{i+1} \cdot x^{(i+1)} = \alpha_{i+1} \ . \end{aligned} \tag{IV.9}$$

La cantidad α_{i+1} se puede interpretar como el error absoluto de redondeo creado cuando $\phi^{(i)}$ es evaluada en aritmética de punto flotante, y los elementos diagonales de E_{i+1} se pueden interpretar como los correspondientes errores relativos de redondeo.

Unendo (*IV*.7), (*IV*.8) y (*IV*.9), se puede expresar $\Delta x^{(i+1)}$ como aproximación del primer orden de la maniera siguiente

$$\Delta x^{(i+1)} \approx \alpha_{i+1} + D\phi^{(i)}(x^{(i)}) \cdot \Delta x^{(i)} = E_{i+1} \cdot x^{(i+1)} + D\phi^{(i)}(x^{(i)}) \cdot \Delta x^{(i)} \ .$$

Sigue entonces que

$$\begin{aligned} \Delta x^{(1)} \approx& D\phi^{(0)}(x) \cdot \Delta x + \alpha_1 \ , \\ \Delta x^{(2)} \approx& D\phi^{(1)}(x^{(1)})[D\phi^{(0)}(x) \cdot \Delta x + \alpha_1] + \alpha_2 \ , \\ & \dots\dots\dots \\ \Delta y = \Delta x^{(r+1)} \approx& D\phi^{(r)}(x^{(r)}) \dots D\phi^{(0)}(x) \cdot \Delta x + \\ & + D\phi^{(r)}(x^{(r)}) \dots D\phi^{(1)}(x^{(1)}) \cdot \alpha_1 + \dots + \alpha_{r+1} \ , \end{aligned}$$

que se puede escribir como

$$\begin{aligned} \Delta y \approx& D\phi(x) \cdot \Delta x + D\psi^{(1)}(x^{(1)}) \cdot \alpha_1 + \dots + D\psi^{(r)}(x^{(r)}) \cdot \alpha_r + \alpha_{r+1} \\ \approx& D\phi(x) \cdot \Delta x + D\psi^{(1)}(x^{(1)}) \cdot E_1 x^{(1)} + \dots + D\psi^{(r)}(x^{(r)}) \cdot E_r x^{(r)} + E_{r+1} y \ . \end{aligned} \tag{IV.10}$$

Es entonces la medida de la matriz Jacobiana $D\psi^{(i)}$ de la aplicación resto $\psi^{(i)}$ que es critica para el efecto de los errores de redondeo intermedios α_i ó E_i sobre el resultado final. Está claro que si para hallar el mismo resultados $\phi(x)$ se usan dos algoritmos diferentes, $D\phi(x)$ queda igual mientras que las matrices Jacobianas $D\psi^{(i)}$ que miden la propagación del error de redondeo serán diferentes. Un algoritmo se dirá **numéricamente más fiable** que otro si, por un dado conjunto de datos, el efecto total de redondeo, dado por $D\psi^{(1)}(x^{(1)}) \cdot \alpha_1 + \ldots + D\psi^{(r)}(x^{(r)}) \cdot \alpha_r + \alpha_{r+1}$ es menor por el primer algoritmo que por el segundo.

Ejemplo 16.

Queremos estudiar el error obtenido para hallar la operacion

$$\phi(a,b) = a^2 - b^2$$

con $\phi : \mathcal{R}^2 \to \mathcal{R}$. Dado que $a^2 - b^2 = (a+b)(a-b)$, se pueden usar para el calculo de ϕ los dos algoritmos

Algoritmo 1	Algoritmo 2
$\eta_1 = a \times a$	$\eta_1 = a + b$
$\eta_2 = b \times b$	$\eta_2 = a - b$
$y = \phi(a,b) = \eta_1 - \eta_2$	$y = \phi(a,b) = \eta_1 \times \eta_2$.

Las correspondientes decomposiciones (IV.3) de ϕ en este caso son

Algoritmo 1	Algoritmo 2
$\phi^{(0)}(a,b) = \begin{pmatrix} a^2 \\ b \end{pmatrix} \in \mathcal{R}^2$	$\phi^{(0)}(a,b) = \begin{pmatrix} a+b \\ a-b \end{pmatrix} \in \mathcal{R}^2$
$\phi^{(1)}(u,v) = \begin{pmatrix} u \\ v^2 \end{pmatrix} \in \mathcal{R}^2$	$\phi^{(1)}(u,v) = u \cdot v \in \mathcal{R}$
$\phi^{(2)}(\alpha,\beta) = \alpha - \beta \in \mathcal{R}$.	

Para el algoritmo 1 obtenemos

$$x = x^{(0)} = \begin{pmatrix} a \\ b \end{pmatrix} \ , \quad x^{(1)} = \begin{pmatrix} a^2 \\ b \end{pmatrix} \ , \quad x^{(2)} = \begin{pmatrix} a^2 \\ b^2 \end{pmatrix} \ , \quad x^{(3)} = y = a^2 - b^2 \ ,$$

$$\psi^{(1)}(u,v) = u - v^2 \ , \quad \psi^{(2)}(u,v) = u - v \ ,$$

$$D\phi(x) = (2a, -2b) \ , \quad D\psi^{(1)}(x^{(1)}) = (1, -2b) \ , \quad D\psi^{(2)}(x^{(2)}) = (1, -1) \ .$$

Además, dado que $fl(\phi^{(0)}(x^{(0)})) - \phi^{(0)}(x^{(0)}) = \begin{pmatrix} a \otimes a \\ b \end{pmatrix} - \begin{pmatrix} a^2 \\ b \end{pmatrix}$, tenemos, con $|\varepsilon_i| < \nu$

$$E_1 = \begin{pmatrix} \varepsilon_1 & 0 \\ 0 & 0 \end{pmatrix} \ , \ \alpha_1 = \begin{pmatrix} \varepsilon_1 \ a^2 \\ 0 \end{pmatrix} \ , \ E_2 = \begin{pmatrix} 0 & 0 \\ 0 & \varepsilon_2 \end{pmatrix} \ , \ \alpha_2 = \begin{pmatrix} 0 \\ \varepsilon_2 \ b^2 \end{pmatrix} \ , \ \alpha_3 = \varepsilon_3(a^2 - b^2) \ .$$

Desde (IV.10) con $\Delta x = (\Delta a, \Delta b)^t$ sigue

$$\Delta y \approx 2a\Delta a - 2b\Delta b + a^2\varepsilon_1 - b^2\varepsilon_2 + (a^2 - b^2)\varepsilon_3 \ . \qquad (IV.11)$$

De la misma manera para el algoritmo 2 sigue

$$x = x^{(0)} = \begin{pmatrix} a \\ b \end{pmatrix} \ , \quad x^{(1)} = \begin{pmatrix} a+b \\ a-b \end{pmatrix} \ , \quad x^{(2)} = y = a^2 - b^2 \ ,$$

$$\psi^{(1)}(u,v) = u \cdot v \ , \quad D\phi(x) = (2a, -2b) \ , \quad D\psi^{(1)}(x^{(1)}) = (a-b, a+b) \ ,$$

$$E_1 = \begin{pmatrix} \varepsilon_1 & 0 \\ 0 & \varepsilon_2 \end{pmatrix} \ , \quad \alpha_1 = \begin{pmatrix} \varepsilon_1(a+b) \\ \varepsilon_2(a-b) \end{pmatrix} \ , \quad \alpha_2 = \varepsilon_3(a^2 - b^2) \ .$$

Y entonces desde (IV.10) sigue

$$\Delta y \approx 2a\Delta a - 2b\Delta b + (a^2 - b^2)(\varepsilon_1 + \varepsilon_2 + \varepsilon_3) \ . \qquad (IV.12)$$

Desde las ecuaciones (IV.11) y (IV.12) se obtienen los siguientes efectos totales de redondeo:

$$|a^2\varepsilon_1 - b^2\varepsilon_2 + (a^2 - b^2)\varepsilon_3| \leq (a^2 + b^2 + |a^2 - b^2|)\nu \ ,$$

para el algoritmo 1, y

$$|(a^2 - b^2)(\varepsilon_1 + \varepsilon_2 + \varepsilon_3)| \leq 3|a^2 - b^2|\nu \ ,$$

para el algoritmo 2. Entonces, podemos decir que el algoritmo 2 es numéricamente más fiable que el algoritmo 1 cada vez que $\frac{1}{3} < |\frac{a}{b}|^2 < 3$; en los otros casos el algoritmo 1 es más fiable. Esto sigue desde la equivalencia de las dos relaciones $\frac{1}{3} \leq |\frac{a}{b}|^2 \leq 3$ y $3|a^2 - b^2| \leq a^2 + b^2 + |a^2 - b^2|$.

Por ejemplo para $a = 0.3237$ y $b = 0.3134$, con aritmética de cuatro dígitos significativos, se obtienen los siguientes resultados:

Algoritmo 1: $a \otimes a = 0.1048$, $b \otimes b = 0.9822 \times 10^{-1}$

$a \otimes a - b \otimes b = 0.6580 \times 10^{-2}$.

Algoritmo 2: $a \oplus b = 0.6371$, $a \ominus b = 0.1030 \times 10^{-1}$

$a^2 - b^2 = 0.6562 \times 10^{-2}$.

Resultado exacto: $a^2 - b^2 = 0.656213 \times 10^{-2}$.

EJERCICIOS.

1. Encuentre cotas para x, si x es una aproximación con cuatro cifras significativas de

a) π ;
b) e .

2. Suponga que p^* aproxima a p con 3 dígitos significativos. Encuentre el intervalo en el cual p^* debe estar, si p es

a) 150 ; 900 ;
b) 1500 ; 90 .

3. Considere los siguientes valores de p y p^*. ¿Con cuántas cifras significativas aproxima p^* a p?

a) $p = \pi, \quad p^* = 3.1$;
b) $p = \frac{1}{3}, \quad p^* = 0.333$;
c) $p = \frac{\pi}{1000}, \quad p^* = 0.0031$;
d) $p = \frac{100}{3}, \quad p^* = 33.3$;
e) $p = \pi, \quad p^* = 3.141$;
f) $p = \frac{1}{3}, \quad p^* = 0.33333$;
g) $p = \frac{\pi}{1000}, \quad p^* = 0.00314$;
h) $p = \frac{100}{3}, \quad p^* = 33.33$.

4. Efectúe los siguientes cálculos (i) exactamente, (ii) usando aritmética cortando a tres dígitos, (iii) usando aritmética de redondeo a tres dígitos. Determine luego las pérdidas de dígitos significativos suponiendo que los números dados son exactos.

a) $14.1 + 0.0981$;
b) $0.0218 * 179.$;
c) $(164. + 0.913) - (143. + 21.0)$;
d) $(164. - 143.) + (0.913 - 21.0)$.

5. Efectúe los siguientes cálculos (i) exactamente, (ii) usando aritmética cortando a tres dígitos, (iii) usando aritmética de redondeo a tres dígitos.

a) $\frac{4}{5} + \frac{1}{3}$;
b) $\frac{4}{5} * \frac{1}{3}$;
c) $\left(\frac{1}{3} - \frac{3}{11}\right) + \frac{3}{20}$;
d) $\left(\frac{1}{3} + \frac{3}{11}\right) - \frac{3}{20}$.

6. Representar en (a) magnitud-signo, (b) notación en exceso, (c) complemento a dos, todos con 8 bits, los siguientes números: 18, -23, 100, 127, -127, 0, -55 y -99.

7. Representar en punto flotante con 32 bits (1 bit para el signo, 7 bits para el exponente y 24 bits para la mantisa) los números: 1234.5675, -1234.5675 y -234.25.