MATEMÁTICA SUPERIOR APLICADA

CAPÍTULO II

CAUSAS PRINCIPALES DE ERRORES EN LOS MÉTODOS NUMÉRICOS

II.1 Introducción

Las causas principales de errores en los métodos numéricos provienen del:

1) Truncamiento de las fórmulas matemáticas
2) Redondeo de los números almacenados en las computadoras

Los errores de truncamiento provienen de las aproximaciones utilizadas en las fórmulas matemáticas de los modelos que representan a los sistemas fisicoquímicos típicos en Ingeniería de Procesos.

Los errores de redondeo se asocian con el hecho que las computadoras digitales solo pueden almacenar números con un número finito de dígitos. Por consiguiente, *debemos conocer como almacenan números las computadoras y como se llevan a cabo las sumas y restas dentro de las mismas.*

II.2 Series de Taylor

Cuando hablemos de soluciones numéricas de los modelos matemáticos estamos haciendo referencia a las aproximaciones de las soluciones exactas obtenidas con las herramientas del Análisis Matemático clásico.

La mayoría de los métodos numéricos se basan en aproximaciones de las funciones mediante polinomios. Luego, toda vez que se quiera objetar la validez de los métodos numéricos se deberá investigar la precisión con que el polinomio aproxima a la función verdadera.

Desarrollo en serie de Taylor: Serie infinita de potencias que representa de manera exacta a una función dentro de un radio alrededor de un punto dado.

La comparación del desarrollo polinomial de la solución numérica con la serie de Taylor de la solución exacta nos permite evaluar el error de truncamiento.

La serie de Taylor se usa para desarrollar métodos numéricos; con excepción de algunos pocos, se puede obtener un polinomio que aproxime a la función verdadera. *A este polinomio se le llama serie truncada de Taylor*. Por consiguiente, el error del método numérico se origina en el truncamiento.

II.2.1 Desarrollo en serie de Taylor para funciones de una variable

Función analítica: Una función *f(x)* se dice analítica en *x = a* si *f(x)* se puede representar mediante una serie de potencias en términos de *h = x - a* dentro de un radio de convergencia *o*<|*x-a*|< *D*.

Condición necesaria de analiticidad de un punto: Deben existir todas las derivadas de la función y ser continuas tanto en el punto como en un entorno alrededor de él.

Un punto donde *f(x)* no es analítica recibe el nombre de *punto singular*. Si *f(x)* es derivable en un entorno de x_0 excepto *en* x_0, entonces x_0 es punto singular de *f(x)*.

Ejemplo: *tg (x)* es una función analítica, excepto en $x_n = \pm (n+½)\,\pi$; *n = 0, 1, 2, …,* ∞ los cuales son puntos singulares.

Desarrollo de una función f(x) en serie de Taylor alrededor de x = a:

$$f(x) = f(a) + hf'(a) + \frac{h^2}{2!}f''(a) + \frac{h^3}{3!}f'''(a) + \frac{h^4}{4!}f''''(a) + \frac{h^5}{5!}f'''''(a) + \cdots + \frac{h^m}{m!}f^{(m)}(a) + \cdots$$

donde $h = x - a$

Ejemplo 1: Desarrollo en serie de Taylor de $f(x) = e^{-x}$ alrededor de *x = 1*.

$$e^{-x} = e^{-1} - he^{-1} + \frac{h^2}{2}e^{-1} - \frac{h^3}{6}e^{-1} + \frac{h^4}{24}e^{-1} - \cdots$$

Ejemplo 2: Desarrollo en serie de Taylor de $f(x) = sen(x)$ alrededor de $x = \pi/4$.

$$sen(x) = sen(\frac{\pi}{4}) + h\cos(\frac{\pi}{4}) - \frac{h^2}{2}sen(\frac{\pi}{4}) - \frac{h^3}{6}\cos(\frac{\pi}{4}) + \frac{h^4}{24}sen(\frac{\pi}{4}) + \cdots$$

donde $h = x - 1$

La serie de Taylor es única, esto es, no existe otra serie de potencias de $h = x - a$ que represente a $f(x)$ en el entorno del punto.

II.2.2 Desarrollo de una función en serie de Mac Laurin

Es el desarrollo de Taylor de la función alrededor del origen (a = 0):

$$f(x) = f(0) + x f'(0) + \frac{x^2}{2!}f''(0) + \frac{x^3}{3!}f'''(0) + \cdots$$

Ejemplos:

1) $e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$

2) $sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$

3) $\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$

4) $ln(x+1) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$

Dado que solo podemos incluir un número finito de términos, en la práctica se procede a truncar o cortar la serie,

$$f(x) = f(a) + h f'(a) + \frac{h^2}{2!} f''(a) + \frac{h^3}{3!}f'''(a) + \frac{h^4}{4!}f''''(a) + \cdots$$

$$\cdots + \frac{h^N}{N!}f^{(N)}(a) + O(h^{N+1})$$

donde $h = x - a$

$O(h^{N+1})$ representa el error que se comete debido al truncamiento de los términos de orden $N+1$ y superiores.

Error global

$$O(h^{N+1}) = \frac{h^{N+1}}{(N+1)!}\ f^{(N+1)}(a + \xi\ h)$$
$$0 < \xi < 1$$

Dado que ξ no se conoce con exactitud,

$$O(h^{N+1}) = \frac{h^{N+1}}{(N+1)!} f^{(N+1)}(a)$$

Esto es, se hace $\xi = 0$ en la expresión del error global. Si $N = 1$,

$$f(x) \cong f(a) + f'(a)\, h$$

Si incluimos el error, entonces:

$$f(x) = f(a) + f'(a)\ h + \ O(h^2)$$

$$O(h^2) = f''(a + \xi\ h)\ \frac{h^2}{2}$$
$$0 < \xi < 1$$

II.2.3 Desarrollo en serie de Taylor de una función de dos variables

$$f(x,y) = f(a,b) + h\ \ f_x\big|_{a,b} + g\ f_y\big|_{a,b} + \frac{1}{2!}\left[h\frac{\partial}{\partial x} + g\frac{\partial}{\partial y}\right]^2 f\Big|_{a,b} + \frac{1}{3!}\left[h\frac{\partial}{\partial x} + g\frac{\partial}{\partial y}\right]^3 f\Big|_{a,b} + \cdots +$$

donde:

$h = x - a$

$g = y - b$

$$f_x = \frac{\partial}{\partial x} f(x,y)$$

$$f_y = \frac{\partial}{\partial y} f(x,y)$$

$\frac{\partial}{\partial x}$: Operador derivada parcial respecto a x.

$\frac{\partial}{\partial y}$: Operador derivada parcial respecto a y.

$$\left(h\frac{\partial}{\partial x}+g\frac{\partial}{\partial y}\right)^2 f=\left(h\frac{\partial}{\partial x}+g\frac{\partial}{\partial y}\right)\left(h\frac{\partial}{\partial x}+g\frac{\partial}{\partial y}\right) f=$$

$$=\left(h^2\frac{\partial^2}{\partial x^2}+h\ g\frac{\partial^2}{\partial x\,\partial y}+g\ h\frac{\partial^2}{\partial x\,\partial y}+g^2\frac{\partial^2}{\partial y^2}\right) f=$$

$$=h^2 f_{xx}+2\,h\,g\,f_{xy}+g^2 f_{yy}$$

Análogamente,

$$\left(h\frac{\partial}{\partial x}+g\frac{\partial}{\partial y}\right)^3 f=h^3 f_{xxx}+3\,h^2\ g\,f_{xx}f_y+3\,h\,g^2\,f_x f_{yy}+g^3 f_{yyy}$$

$$\left(h\frac{\partial}{\partial x}+g\frac{\partial}{\partial y}\right)^4 f=h^4 f_{xxxx}+4\,h^3\ g\,f_{xxx}f_y+6\,h^2\ g^2\,f_{xx}f_{yy}+4\,h\,g^3\,f_x f_{yyy}+g^4 f_{yyyy}$$

Y así sucesivamente.

II.3 Los Números en las Computadoras

Cuando resolvemos un problema de ingeniería con calculadora de escritorio somos concientes que los números decimales que calculamos no son exactos, sino que los redondeamos cuando los registramos. Aún cuando no se los redondee en forma intencional, el número limitado de dígitos de la calculadora puede acarrear errores de redondeo.

Calculadoras de bolsillo

- Barata: ~ de 6 dígitos.
- Cara: ~ entre 10 y 11 dígitos.

Computadoras personales (PC's)

Los errores de redondeo aparecen por las mismas razones que aparecen en las calculadoras de escritorio. Por consiguiente, resulta necesario conocer algunos aspectos básicos de operaciones aritméticas en computadoras y bajo qué circunstancias se pueden generar serios errores de redondeo.

II.3.1 Base de representación de los números

El sistema numérico usual es el sistema decimal, esto es, el sistema de numeración de base 10, así por ejemplo:

$1.321 = 1x10^3 + 3x10^2 + 2x10^1 + 1x10^0$

$= (1.321)_{10}$

Dígitos: 1, 2, 3, 4, 5, 6, 7, 8, 9 + cero (0)

Las calculadoras de escritorio y computadoras personales utilizan el sistema de numeración binario, basado en la lógica de dos estados:

off, on

apagado, encendido

Dígitos binarios: 0 y 1.

Otros sistemas numéricos que utilizan las computadoras:

- Octal (base 8). Dígitos: *1, 2, 3, 4, 5, 6, 7 + cero (0)*
- Hexadecimal (base 16).

  Dígitos: *1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F + cero (0).*

Los números representados en el sistema octal y el hexadecimal son más cortos que si se los representa en forma binaria, por lo tanto son más fáciles de leer y de comprender. En resumen los números se pueden representar en diferentes bases, en particular, las computadoras utilizan los siguientes sistemas de representación de números:

Base 2: Sistema binario
Base 8: Sistema octal
Base 16: Sistema hexadecimal

En general, el valor decimal de un número en base σ, por ejemplo,

$$(a\ b\ c\ d\ e\ f\ g\ .\ h\ i\ j\ k)_{\sigma}$$

se calcula como:

$$a\,\sigma^6 + b\,\sigma^5 + c\,\sigma^4 + d\,\sigma^3 + e\,\sigma^2 + f\,\sigma^1 + g\,\sigma^0 + h\,\sigma^{-1} + i\,\sigma^{-2} + j\,\sigma^{-3} + k\,\sigma^{-4}$$

Ejemplos:

$(1)_{10} = 1 \times 2^0 = (1)_2$

$(2)_{10} = 1 \times 2^1 + 0 \times 2^0 = (10)_2$

$(3)_{10} = 1 \times 2^1 + 1 \times 2^0 = (11)_2$

$(4)_{10} = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = (100)_2$

$(5)_{10} = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (101)_2$

$(6)_{10} = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (110)_2$

$(7)_{10} = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = (111)_2$

$(8)_{10} = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = (1000)_2$

$(9)_{10} = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (1001)_2$

$(10)_{10} = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (1010)_2$

+	0	1	2	3	4	5	6	7	8	9
0	0	1	10	11	100	101	110	111	1000	1001
1	1	10	11	100	101	110	111	1000	1001	1010
2	10	11	100	101	110	111	1000	1001	1010	1011
3	11	100	101	110	111	1000	1001	1010	1011	1100
4	100	101	110	111	1000	1001	1010	1011	1100	1101
5	101	110	111	1000	1001	1010	1011	1100	1101	1110
6	110	111	1000	1001	1010	1011	1100	1101	1110	1111
7	111	1000	1001	1010	1011	1100	1101	1110	1111	10000
8	1000	1001	1010	1011	1100	1101	1110	1111	10000	10001
9	1001	110	1011	1100	1101	1110	1111	10000	10001	10010

$(1)_{10}$	⟶	$(1)_8$
$(2)_{10}$	⟶	$(2)_8$
$(3)_{10}$	⟶	$(3)_8$
$(4)_{10}$	⟶	$(4)_8$
$(5)_{10}$	⟶	$(5)_8$
$(6)_{10}$	⟶	$(6)_8$

$(7)_{10}$	⟶	$(7)_8$
$(8)_{10}$	⟶	$(10)_8$
$(9)_{10}$	⟶	$(11)_8$
$(10)_{10}$	⟶	$(12)_8$

$(1)_{10}$	⟶	$(1)_{16}$
$(2)_{10}$	⟶	$(2)_{16}$
$(3)_{10}$	⟶	$(3)_{16}$
$(4)_{10}$	⟶	$(4)_{16}$
$(15)_{10}$	⟶	$(F)_{16}$
$(16)_{10}$	⟶	$(10)_{16}$

En lo que sigue, los números sin subíndice se supone que están representados en base 10, salvo que se indique lo contrario.

II.3.2 Rango de las constantes numéricas

Las constantes numéricas se clasifican en:

1) Enteras
2) Reales
3) Complejas

Los números utilizados en lenguajes de programación son decimales, a menos que se diga lo contrario.

***Rango de las constantes enteras*:**

BASIC para PC IBM

- Mayor número entero: $2^{15}-1 = 32.767$
- Menor número entero: $-2^{15} = -32.768$

FORTRAN para IBM 370

- Mayor número entero:
  $2^{31} - 1 = 2.147.483.647$
- Menor número entero:
  $-(2^{31} - 1) = -2.147.483.647$

Rango de las constantes reales

Con respecto a los números reales, la menor y mayor magnitud de los mismos dependen del:

1) Hardware de la computadora.
2) Software utilizado.

BASIC para PC IBM

- Menor número real positivo:
  $2{,}9 \times 10^{-39}$
- Mayor número real positivo:
  $1{,}7 \times 10^{38}$

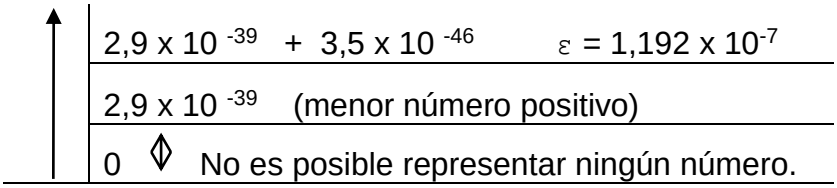
FORTRAN para IBM 370

- Menor número real positivo:
  $5{,}4 \times 10^{-79}$
- Mayor número real positivo:
  $7{,}2 \times 10^{75}$

Rango de los números reales en simple precisión

	Mínimo	Máximo
IBM PC (BASIC)	$2{,}9 \times 10^{-39}$	$1{,}7 \times 10^{38}$
IBM 370	$5{,}4 \times 10^{-79}$	$7{,}2 \times 10^{75}$
Cray XMP	$4{,}6 \times 10^{-2476}$	$5{,}4 \times 10^{2465}$
VAX	$2{,}9 \times 10^{-39}$	$1{,}7 \times 10^{38}$

Una cuestión muy importante: los números no son continuos en una computadora. Por ejemplo, el número positivo más pequeño en una IBM PC es $2{,}9 \times 10^{-39}$ por lo tanto no se pueden representar números entre 0 y $2{,}9 \times 10^{-39}$. Sin embargo, el intervalo entre el número positivo $2{,}9 \times 10^{-39}$ y el siguiente menor número positivo es $3{,}5 \times 10^{-46}$ que es mucho menor que $2{,}9 \times 10^{-39}$.



Epsilon de la máquina: Diferencia entre el menor número mayor que uno y uno. Intervalo entre cualquier número real y el siguiente:

$$\Delta = \varepsilon \times R$$

donde:

ε: épsilon de la máquina

R: número real

II.3.3 Números en el hardware de la computadora

Bit: dígito binario: 0 a 1 (off-on)

Byte: 8 bits: unidad de almacenamiento en memoria.

Representación de enteros (forma binaria)

$$I = \pm (a_k \; a_{k-1} \; a_{k-2} \; \cdots \; a_2 \; a_1 \; a_0)_2 \;\; con \;\; a_i = 0 \text{ ó } 1$$

Su representación decimal es:

$$I = \pm (a_k 2^k + a_{k-1} 2^{k-1} + \cdots + a_2 2^2 + a_1 2^1 + a_0)_{10}$$

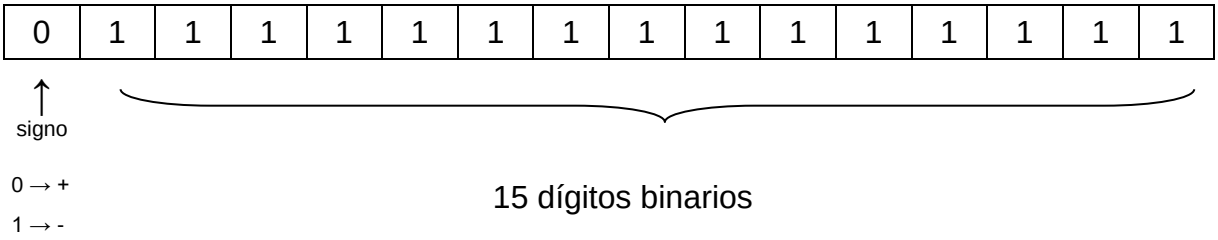
Ejemplo: Representar en forma decimal la expresión binaria $\pm (110101)_2$

$$I = \pm (1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)$$

$$= \pm (32 + 16 + 0 + 4 + 0 + 1)_{10} = \pm (53)_{10}$$

El valor máximo de k depende del diseño del hardware.

Enteros: IBM PC (BASIC) → 2 bytes = 16 bits, luego k = 14.



El valor decimal registrado en la expresión binaria anterior es:

$$I = \sum_{i=0}^{14} 2^i = 2^{15} - 1 = (32.767)_{10}$$

Formas de almacenamiento de un entero negativo:

1) Colocando un *1* en la posición del primer bit.

2) Utilizando el complemento a *2* del número.

Por ejemplo, el complemento a *2* para $(-32.767)_{10}$ es:

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

- Se cambian en la expresión del número entero (+) ceros por unos y unos por ceros y al resultado se le suma *1*.
- Si este número es menor que 2^{15} se lo interpreta positivo.
- Si es mayor o igual, entonces se transforma en un número negativo restándole 2^{16}. En nuestro ejemplo resulta que el número entero acumulado en forma complementaria es $Z = 2^{15} + 1$, luego:

  $$2^{15} + 1 - 2^{16} = 32.768 + 1 - 65.536 = (-32.767)_{10}$$

  Luego, el mayor entero negativo resulta:

  $$(1111\ 1111\ 1111\ 1111)_2 \rightarrow (2^{16} - 1) = (65.535)_{10}$$

  que es igual a $(2^{16}-1) - 2^{16} = (-1)_{10}$.

En la IBM 370 se usan *4 bytes* para representar a un número entero (*32 bits*). Por lo tanto el máximo número entero que se puede representar es:

$$2^{31} - 1 = 2.147.483.648 - 1 = 2.147.483.647$$

Representación de números reales (forma binaria)

Su presentación difiere según el diseño de hardware. Tomaremos como ejemplo la IBM PC (BASIC) y la IBM mainframe (FORTRAN 77).

IBM PC (BASIC): Formato punto flotante.

- Simple precisión: 4 bytes = 32 bits
- Doble precisión: 8 bytes = 64 bits

Al número decimal lo convierte al binario más cercano.

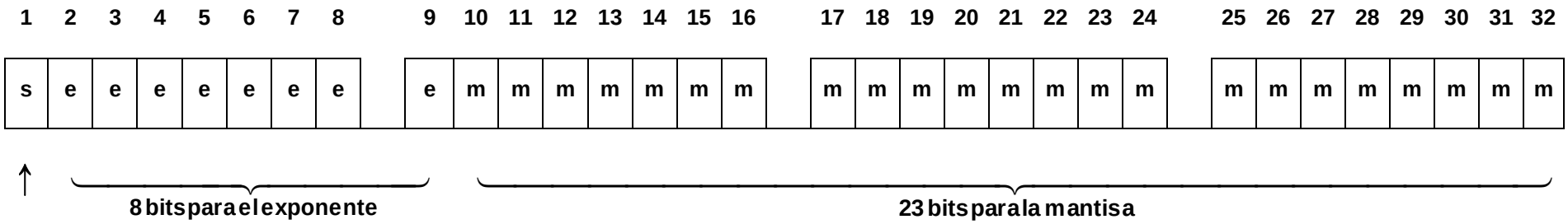
$$R = \pm (0,abbb \cdots bbb)_2 \times 2^k$$

donde:

a = 1

b = 0 ó 1

k = Exponente entero expresado como binario (número entero).



1 bit para el signo

0 → +

1→ -

En formato PF el primer dígito de la mantisa *siempre es 1*, por lo tanto no se lo almacena; en consecuencia se tienen 24 dígitos para la mantisa.

Si los 8 bits asignados se usan para almacenar enteros positivos, entonces el máximo exponente es:

$$k_{máx} = 2^8 - 1 = 255 \text{ (almacenado como binario)}$$

Para almacenar enteros negativos el exponente en decimales es sesgado en 128 y después convertido a binario (complemento a 2). Por ejemplo:

$(-3) \rightarrow (-3) + 128 = 125 \rightarrow$ se convierte a binario y se almacena en 8 bits, luego:

$$-128 \leq k \leq 127$$

IBM 370: Se utiliza formato PF normalizado en hexadecimal que se escribe como:

$$R = \pm (0,abbbbb)_{16} \times 16^k$$

donde:

a: dígito hexadecimal distinto de cero.

b: dígito hexadecimal que puede ser cero.

k: exponente entero expresado en binario.

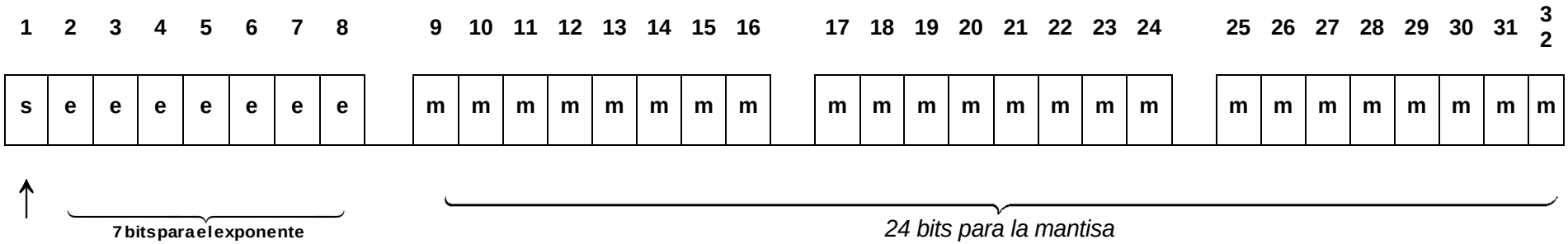
Si los 7 bits asignados al exponente se usan para almacenar enteros positivos, entonces el máximo exponente es:

$$k_{máx} = 2^7 - 1 = 127 \text{ (almacenado como binario)}$$

Para almacenar enteros negativos el exponente en decimales es sesgado en 64 y después convertido a binario (complemento a 2). Por consiguiente,

$$-64 \leq k \leq 63$$

La mantisa puede almacenar 6 dígitos hexadecimales. En simple precisión se utilizan 32 bits.



1 bit para el signo

0 → +

1→ -

Cabe preguntarse ahora: ¿Cuál es el máximo número real positivo que se puede representar en un mainframe IBM?

1ero.) La máxima mantisa positiva es:

$$(0,FFFFFF)_{16} = (1-16^{-6})_{10}$$

2do.) El máximo exponente que se puede representar en binario será:

$$127 - 64 = 63$$

Por lo tanto el máximo valor positivo del PF es:

$$(1-16^{-6})_{10} \times 16^{63} \approx (7.23 \times 10^{75})_{10}$$

¿Cuál es el menor número real positivo que se puede representar en un mainframe IBM?

$$(0,100000)_{16} \times 16^{-64}$$

Finalmente, se obtiene:

$$(0,100000)_{16} \times 16^{-64} = 16^{-1} \times 16^{-64} = 16^{-65} = (5,39 \times 10^{-79})_{10}$$

¿Cuál es el épsilon de la máquina?

El menor número mayor que uno que se puede escribir en formato PF es:

$$(0,100001)_{16} x16^{1}$$

luego,

$$\varepsilon = (0,100001)_{16} \times 16^{1} - (0,1)_{16} \times 16^{1} = 16^{-5} = (9,53 \times 10^{-7})_{10}$$

	Simple Precisión		**Doble Precisión**	
	Mantisa	**Exponente**	**Mantisa**	**Exponente**
IBM PC AT, XT	23	8	55	8
IBM 370	24	7	56	7
CDC 7600 y Cyber	48	11	96	11
Vax 11	23	8	55	8
Cray XPM	48	11	96	11

II.3.4 Errores de redondeo en una computadora

Ningún número real *R* se puede representar entre *1* y *(1+ε)*.

IBM PC (Basic): Si *α* es un número real tal que $|\alpha| < \varepsilon$

Entonces:

1. *si* $|\alpha| < (\varepsilon/2)$: $(1+\alpha) \rightarrow 1$
2. *si* $(\varepsilon/2) \leq |\alpha| < \varepsilon$: $(1+\alpha) \rightarrow 1 \pm \varepsilon$

Concluimos que *ε/2* es el máximo error de redondeo para *1*, por lo tanto cuando este valor se halla almacenado en la memoria, el valor original puede ser alguno entre:

$$1 - (\varepsilon/2) < x < 1 + (\varepsilon/2)$$

EPSILON'S				
Precisión	**IBM PC**	**IBM 370**	**VAX II**	**Cray XPM**
Simple	$1{,}19 \times 10^{-7}$	$9{,}53 \times 10^{-7}$	$5{,}96 \times 10^{-8}$	$3{,}55 \times 10^{-15}$
Doble	$2{,}77 \times 10^{-17}$	$2{,}22 \times 10^{-16}$	$1{,}38 \times 10^{-17}$	$1{,}26 \times 10^{-29}$

Errores de redondeo

Si R es un número real almacenado en memoria, el error de redondeo es aproximadamente:

1) ε R/2 si el número se redondea por exceso.
2) ε R si el número se redondea por defecto.

Efectos de los errores de redondeo

Si se suman o se restan dos números, es posible que la representación exacta del resultado requiera de un número de dígitos mayor. Situaciones que se plantean:

1) Cuando se suma (o se resta) a un número muy grande uno muy pequeño.
2) Cuando un número se resta de otro que es muy próximo.

Ejemplos:

Veamos la situación (1). Sumemos 10^4 veces $0,00001$ a la unidad. En una IBM PC la suma daría:

$$Suma = 1,100136$$

¿Qué error cometemos? El resultado exacto es:

$$Suma = 1,1$$

$$e_{rel}(\%) = 100 \times \left| 1,100136 - 1,1 \right| / 1,1 \approx 0,0124\%$$

Importante: El error de redondeo aumenta cuando el nro. de operaciones aritméticas se incrementa.

Analicemos la suma $1 + 0,00001$ en una IBM PC. Sus representaciones binarias son (simple precisión):

$$(1)_{10} = (0,1000\ 0000\ 0000\ 0000\ 0000\ 0000)_2 \times 2^1$$

$$(0,00001)_{10} = (0,1010\ 0111\ 1100\ 0101\ 1010\ 1100)_2 \times 2^{-16}$$

Además, $2^{-16} = 2^{-17} \times 2^1$. Luego,

$(1)_{10} + (0,00001)_{10} =$

$$= (0,1000\ 0000\ 0000\ 0000\ 0101\ 0011\ 1110\ 0010\ 1101\ 0110\ 0)_2 \times 2^1$$

↑ a partir de la posición 24 se redondea (mantisa de 24 bits).

Por lo tanto, el número que se guarda en memoria es:

$$(1)_{10} + (0,00001)_{10} = (0,1000\ 0000\ 0000\ 0000\ 0101\ 0100)_2 \times 2^1$$

que es equivalente a:

$$(2^{-1}+2^{-18}+2^{-20}+2^{-22}) \times 2^1 = 1+2^{-17}+2^{-19}+2^{-21} = 1,0000100136$$

Cada vez que se sume $0,00001$ a 1 se adiciona un error $e_{abs} = 0,0000000136$. Si se suma 10000 veces $0,00001$, el error acumulado será $e_{acum} = 0,000136$.

Veamos que sucede al restar dos números muy próximos (situación (2)). Supongamos que queremos restar 1 a $1,00001$. Entonces,

$$(1,00001)_{10} = (0,1000\ 0000\ 0000\ 0000\ 0101\ 0100)_2 \times 2^1$$

Luego,

$$(1,00001)_{10} - (1)_{10} = (0,1000\ 0000\ 0000\ 0000\ 0101\ 0100)_2 \times 2^1 - (0,1)_2 \times 2^1$$

$$= (0,10101)_2 \times 2^{-16} = (1,00136 \times 10^{-5})_{10}$$

El error relativo es:

$$e_{rel}(\%) = 100 \times (0,0000100136 - 0,00001)/(0,00001) = 0,136\%$$

Estrategias para evitar los errores de redondeo

1. Trabajar en doble precisión.
2. Agrupar términos.
3. Efectuar desarrollos de Taylor.
4. Cambiar la definición de las variables.
5. Reescribir las ecuaciones para evitar restas.