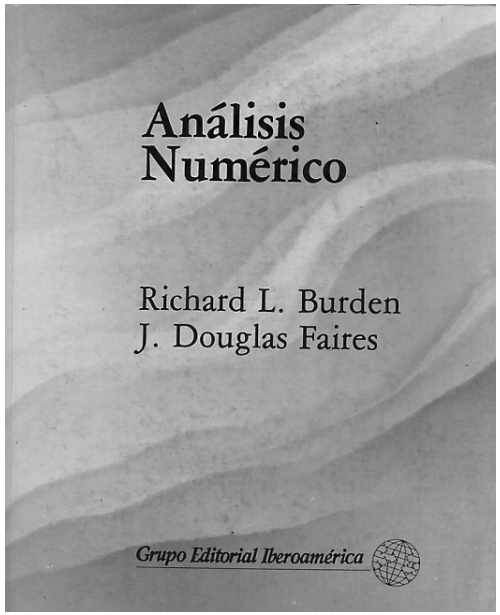
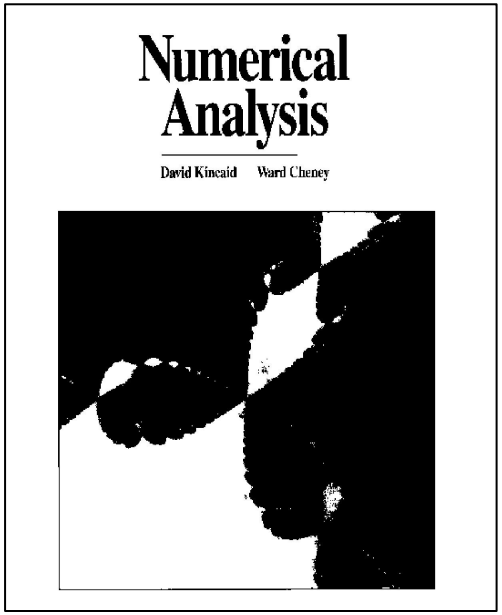
CO3211 – Cálculo Numérico

- Representación de números. Análisis de error
- Sistemas lineales. Métodos directos e iterativos.
- Autovalores y autovectores.
- Aproximación de Funciones. Mínimos cuadrados. Interpolación. Diferencias divididas. Splines. Curvas paramétricas

Referencias:

- Análisis Numérico. Richard Burden, Douglas Faires. Grupo Editorial Iberoamericano.
- Numerical Analysis. David Kincaid, Ward Cheney. Brooks-Cole.





CO3211 – Cálculo Numérico

Aula:

Teoría: martes 11:30 a 1:30 - AUL019

jueves 11:30 a 1:30 - AUL103

Laboratorio: viernes 7:30 a 9:30 - MYS Sala A

Evaluación:

2 Parciales con valor de 32 puntos cada uno (semanas 6 y 11)

8 Laboratorios evaluados con valor de 3 puntos cada uno.

1 proyecto con valor de 12 puntos

No hay recuperación de laboratorios evaluados.

Sólo recuperación de 1 parcial (justicado!!) en la semana 12 (toda la materia)

Representación de números

- Representación en base 10

$$0.101_{10} = 1 \cdot 10^{-1} + 0 \cdot 10^{-2} + 1 \cdot 10^{-3} = \frac{1}{10} + \frac{1}{1000} = 0.101$$

- Representación en base 2

$$0.101_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = \frac{1}{2} + \frac{1}{8} = 0.625_{10}$$

$$11011.01_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

$$= 16 + 8 + 2 + 1 + \frac{1}{4} = 27.25_{10}$$

En general

$$x = (a_n a_{n-1} \cdots a_1 a_0 \,.\, a_{-1} a_{-2} \cdots a_{-m})_2 =$$

$$a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0 + a_{-1} \cdot 2^{-1} + a_{-2} \cdot 2^{-2} + \cdots + a_{-m} \cdot 2^{-m}$$

Representación de números

- Representación en base 2 (cont.)

$$x = (a_n a_{n-1} \cdots a_1 a_0)_2 = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0 =$$

$$2(a_n \cdot 2^{n-1} + a_{n-1} \cdot 2^{n-2} + \cdots + a_1) + a_0$$

Luego a_0 es el resto de dividir x entre 2.

$$x = 2 \cdot x_1 + r_0$$

con

$$x_1 = a_n \cdot 2^{n-1} + a_{n-1} \cdot 2^{n-2} + \cdots + a_1 \quad \text{y} \quad r_0 = a_0$$

Para hallar el siguiente dígito a_1, aplicamos el mismo procedimiento a x_1

$$x_1 = a_n \cdot 2^{n-1} + a_{n-1} \cdot 2^{n-2} + \cdots + a_1 = 2 \cdot x_2 + r_1$$

con

$$x_2 = a_n \cdot 2^{n-2} + a_{n-1} \cdot 2^{n-3} + \cdots + a_2 \quad \text{y} \quad r_1 = a_1$$

Representación de números

- Representación en base 2 (cont.)

Ejemplo1:

$$x = 25 \quad \Rightarrow \quad x = 2 \cdot 12 + 1 \quad \Rightarrow \quad a_0 = 1$$

$$x_1 = 12 \quad \Rightarrow \quad x_1 = 2 \cdot 6 + 0 \quad \Rightarrow \quad a_1 = 0$$

$$x_2 = 6 \quad \Rightarrow \quad x_2 = 2 \cdot 3 + 0 \quad \Rightarrow \quad a_2 = 0$$

$$x_3 = 3 \quad \Rightarrow \quad x_3 = 2 \cdot 1 + 1 \quad \Rightarrow \quad a_3 = 1$$

$$x_4 = 1 \quad \Rightarrow \quad x_4 = 2 \cdot 0 + 1 \quad \Rightarrow \quad a_4 = 1$$

$$x_5 = 0 \qquad \text{finaliza}$$

por lo tanto $(25)_{10} = (11001)_2$

Representación de números

- Representación en base 2 (cont.)

$$x = (.a_{-1}a_{-2}\cdots a_{-m})_2 = a_{-1}\cdot 2^{-1} + a_{-2}\cdot 2^{-2} + \cdots + a_{-m}\cdot 2^{-m}$$

$$\frac{1}{2}(a_{-1} + a_{-2}\cdot 2^{-1} + \cdots + a_{-m}\cdot 2^{-m+1})$$

luego $\quad 2x = a_{-1} + a_{-2}\cdot 2^{-1} + \cdots + a_{-m}\cdot 2^{-m+1}$

así, a_{-1} es la parte entera de 2x, y repetimos el mismo proceso para calcular el siguiente dígito

Ejemplo2:

$$x = 0.8125 \quad \Rightarrow \quad 2x = 1.625 \quad \Rightarrow \quad a_{-1} = 1$$

$$x_1 = 0.625 \quad \Rightarrow \quad 2x_1 = 1.25 \quad \Rightarrow \quad a_{-2} = 1$$

$$x_2 = 0.25 \quad \Rightarrow \quad 2x_2 = 0.50 \quad \Rightarrow \quad a_{-3} = 0$$

$$x_3 = 0.50 \quad \Rightarrow \quad 2x_2 = 1.0 \quad \Rightarrow \quad a_{-4} = 1$$

$$x_4 = 0.0 \qquad \text{finaliza}$$

por lo tanto $\quad (0.8125)_{10} = (0.1101)_2$

Representación de números

- Representación en base 8

decimal	0	1	2	3	4	5	6	7
binario	000	001	010	011	100	101	110	111
octal	0	1	2	3	4	5	6	7

así, el número se divide en bloques de 3 dígitos (la parte entera de derecha a izquierda y la parte decimal de izquierda a derecha) desde el punto decimal

$$(101|101|001.110|010|100)_2 = (551.624)_8$$

Representación de números

- Representación en base 8 (cont.)

  Justificación del cálculo

$$(101101001.1100101)_2 = (101|101|001.110|010|100)_2 = (551.624)_8$$

$$\begin{aligned} x &= (.b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}b_{-6}\cdots)_2 \\ &= b_{-1}\cdot 2^{-1} + b_{-2}\cdot 2^{-2} + b_{-3}\cdot 2^{-3} + b_{-4}\cdot 2^{-4} + b_{-5}\cdot 2^{-5} + b_{-6}\cdot 2^{-6} + \cdots \\ &= (4b_{-1} + 2b_{-2} + b_{-3})\cdot 2^{-3} + (4b_{-4} + 2b_{-5} + b_{-6})\cdot 2^{-6} + \cdots \\ &= (4b_{-1} + 2b_{-2} + b_{-3})\cdot 8^{-1} + (4b_{-4} + 2b_{-5} + b_{6})\cdot 8^{-2} + \cdots \end{aligned}$$

notar que los b_{-i}, $i=1,2,3,\ldots$, son los números 0 o 1, y la operación entre paréntesis produce dígitos entre 0 y 7

Representación de números en el computador

Sea β la base usada en el computador, β=2,8,16.
Sea x un número real, con x distinto de cero y normalizado

$$x = \sigma(a_0.a_1a_2\cdots a_t)_\beta \beta^E$$

donde

σ representa el signo de *x*

$(a_1a_2\cdots a_t)_\beta$ la mantisa en base β

$(E)_\beta$ el exponente en base β

a_0 es un número entero entre 1 y β-1

Ejemplo de normalización de un número en base 2:

Sea el número x = 49.8125

$$x = (110001.1101)_2 = (1.\underbrace{10001}1101)_2 2^5$$

5 posiciones

Representación de números en el computador

Una palabra en el computador esta formada por 32 bits.

Así, el número x puede ser representado usando:

- 1 bit para el signo
- 8 bits para el exponente (la característica)
- 23 bits para la parte fraccionaria (la mantisa)

Observación:

El exponente de 8 dígitos representa un número del 0 al 2^8-1 = 255

1	1	1	1	1	1	1	1

$$= 1 \cdot 2^7 + 1 \cdot 2^6 + \cdots + 1 \cdot 2^1 + 1 \cdot 2^0 =$$

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$$

A fin de que estos números se puedan representar, se resta 127 del exponente, de manera que el intervalo del exponente es en realidad [-127, 128].

El exponente se almacena sumándole el sesgo correspondiente al número de bits de la representación usada, en este caso el sesgo corresponde a 127.
La razón es que el exponente va ser un número entre -127 y 128.

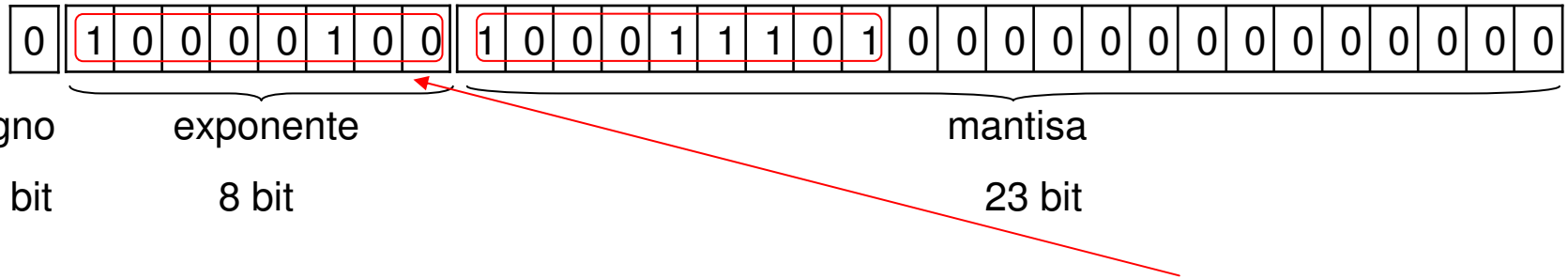
Representación de números en el computador

Retomando el ejemplo x = 49.8125

$$x = (110001.1101)_2 = (1.100011101)_2 2^5$$

El número normalizado

$x = (1.100011101)_2 2^5$ se representa como



el exponente *e* = (5 + 127 (sesgo))$_2$: $e = 5 + 127 = 132 = (10000100)_2$

el signo *s* se representa como 0 si es positivo y 1 si es negativo, *s* = 0

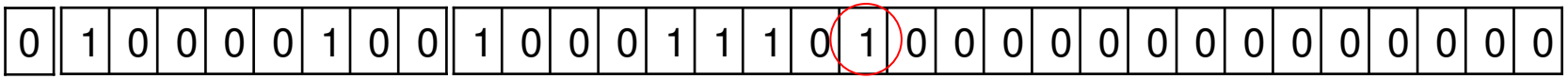
la mantisa es $(100011101)_2$, de donde m = $(0.100011101)_2$

El uso de este sistema proporciona un número en punto flotante de la forma

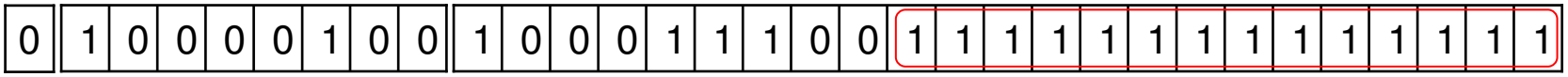
$$(-1)^s 2^{e-127}(1+m)$$

Representación de números en el computador

El número de máquina menor que le sigue a

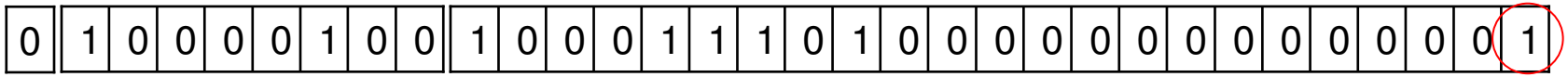


es



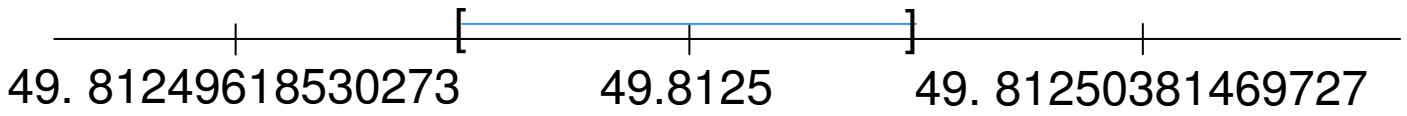
49. 81249618530273

y el siguiente número de máquina mayor es



49. 81250381469727

Esto significa que nuestro número de máquina original representa no sólo a 49.8125 sino también a los números que están en el intervalo entre corchetes (línea azul).



Para computadores de 64 bits (una palabra), la representación es

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

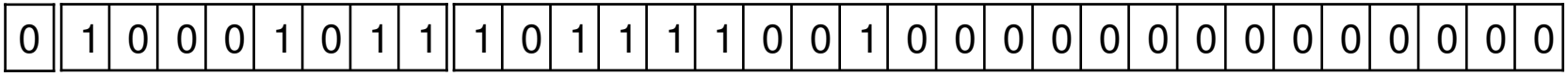
Representación de números en el computador

Determinar el número decimal que corresponde al número de máquina (representación de 32 bits) siguiente:

$$(45DE4000)_{16}$$

El número en binario correspondiente es

$$(0100\ 0101\ 1101\ 1110\ 0100\ 0000\ 0000\ 0000)_2$$



exponente: $(10001011)_2 = (213)_8 = (139)_{10}$, restando el sesgo: 139-127=12

mantiza: 101111001 , de donde el número asociado es

$$(1.101111001)_2 \times 2^{12} = (1101111001000.)_2 = (15710)_8 =$$

$$0 \times 1 + 1 \times 8 + 7 \times 8^2 + 5 \times 8^3 + 1 \times 8^4 =$$

$$(7112)_{10}$$

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Errores de redondeo y aritmética del computador

El error de redondeo se origina porque la aritmética realizada en un computador 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.

En un computador, sólo un subconjunto relativamente pequeño de los números reales, los números de punto flotante o números de máquina decimal, se usan para representar a todos los números reales.

Para explicar los problemas que pueden surgir en la manipulación de números de máquina decimal, supondremos que estos se representan como (aritmética de k dígitos)

$$\pm(0.d_1 d_2 \cdots d_k) \times 10^n$$

$$\text{con} \quad 1 \le d_1 \le 9, \quad 0 \le d_i \le 9 \quad \text{para} \quad i = 2, \ldots, k$$

Los computadores usan aproximadamente $k = 6$ y $-77 \le n \le 76.$

Errores de redondeo y aritmética del computador

Cualquier número real *y* que este dentro del rango numérico del computador se le puede asociar una representación de punto flotante *fl(y)*.

$$y = (0.d_1 d_2 \cdots d_k d_{k+1} \cdots) \times 10^n$$

Hay 2 formas de llevar a cabo esto:

- Truncando: cortar los dígitos d_{k+1}, d_{k+2}, …, y así

$$fl(y) = 0.d_1 d_2 \cdots d_k \times 10^n$$

- Redondeando: añadir $5 \times 10^{n-(k+1)}$ y luego cortar, obteniéndose

$$fl(y) = 0.\delta_1 \delta_2 \cdots \delta_k \times 10^n$$

Este método funciona así:

Si d_{k+1} es mayor o igual a 5, agregamos 1 a d_k

Si d_{k+1} es menor que 5, cortamos los dígitos *k+1* en adelante

Errores de redondeo y aritmética del computador

Las imprecisiones que resultan de redondear se conocen como

errores por redondeo.

En general, los errores por redondeo se acumulan menos (durante los cálculos repetidos) que los errores producidos por truncamiento, ya que el valor verdadero es mayor que el valor redondeado en alrededor de la mitad de las veces y menor también alrededor de la mitad de las veces.

Además, para la operación de truncamiento, el mayor error absoluto que pudiera producirse sería del doble del de redondeo.

Por otra parte, el truncamiento no requiere de decisión alguna acerca de cambiar o no el último dígito retenido.

Errores de redondeo y aritmética del computador

Ejemplo: usando aritmética de 4 dígitos, realicemos la suma de 2 números para los casos de redondeo y truncamiento:

	número	=	redondeo	+	error	=	truncamiento	+	error
	1374.8	=	1375	-	0.2	=	1374	+	0.8
	3856.4	=	3856	+	0.4	=	3856	+	0.4
Total	5231.2	=	5231	+	0.2	=	5230	+	1.2

Los errores durante el proceso de redondeo tienen signos opuestos y se cancelan parcialmente. Para el caso de truncamiento, sin embargo, los errores tienen el mismo signo y por lo tanto se suman.

Los errores por redondeo se acumulan menos que los errores producidos por truncamiento.

Errores de redondeo y aritmética del computador

Si p* es una aproximación de p, definimos dos tipos de errores:

- El error absoluto, que viene dado por $EA = | p^* - p |$
- El error relativo, que esta dado por $ER = \frac{| p^* - p |}{| p |}$ siempre y cuando p sea distinto de cero.

Ejemplo:

$p = 0.3000 \times 10^{1}$	$p = 0.3000 \times 10^{-3}$
$p^* = 0.3100 \times 10^{1}$	$p^* = 0.3100 \times 10^{-3}$
$EA = 0.1$	$EA = 0.1 \times 10^{-4}$
$ER = 0.3333 \times 10^{-1}$	$ER = 0.3333 \times 10^{-1}$

Observaciones:

- el error relativo en ambos casos es el mismo, mientras los errores absolutos son diferentes
- como medida de precisión el error absoluto puede ser engañoso, en cambio el error relativo puede ser más significativo

Errores de redondeo y aritmética del computador

Error relativo para la representación de punto flotante *fl(y)* de *y*

$$y = (0.d_1 d_2 \cdots d_k d_{k+1} \cdots) \times 10^n$$

- Si se usan k dígitos decimales "cortando"

$$fl(y) = 0.d_1 d_2 \cdots d_k \times 10^n$$

el error relativo es

$$\left| \frac{y - fl(y)}{y} \right| = \left| \frac{0.d_1 d_2 \cdots d_k d_{k+1} \cdots \times 10^n - 0.d_1 d_2 \cdots d_k \times 10^n}{0.d_1 d_2 \cdots d_k d_{k+1} \cdots \times 10^n} \right|$$

$$= \left| \frac{0.d_{k+1} \cdots \times 10^{n-k}}{0.d_1 d_2 \cdots d_k d_{k+1} \cdots \times 10^n} \right| = \left| \frac{0.d_{k+1} \cdots}{0.d_1 d_2 \cdots d_k d_{k+1} \cdots} \right| 10^{-k}$$

Como $d_1 \neq 0$, el mínimo valor del denominador es 0.1, y el numerador está acotado por 1, obteniéndose

$$\left| \frac{y - fl(y)}{y} \right| \leq \frac{1}{0.1} \times 10^{-k} = 10^{-k+1}$$

Obs. La cota para el error relativo cuando se usa aritmética de k dígitos, es independiente del número que se está representando.

Errores de redondeo y aritmética del computador

- De manera similar, una cota para el error relativo, cuando se usa aritmética de redondeo de *k* dígitos,

$$\text{si} \quad d_{k+1} < 5, \quad \text{entonces} \qquad fl(y) = 0.d_1 d_2 \cdots d_k \times 10^n$$

$$\text{si} \quad d_{k+1} \geq 5, \quad \text{entonces} \qquad fl(y) = 0.d_1 d_2 \cdots d_k \times 10^n + 10^{n-k}$$

Para el segundo caso,

$$\left| \frac{y - fl(y)}{y} \right| = \left| \frac{0.d_{k+1} \cdots \times 10^{n-k} - 10^{n-k}}{0.d_1 d_2 \cdots d_k d_{k+1} \cdots \times 10^n} \right| = \left| \frac{0.d_{k+1} \cdots - 1}{0.d_1 d_2 \cdots d_k d_{k+1} \cdots} \right| 10^{-k}$$

$$\leq \left| \frac{0.d_{k+1} \cdots - 1}{0.1} \right| 10^{-k} = |\, 0.d_{k+1} \cdots - 1 \,| \, 10^{-k+1} \leq 0.5 \cdot 10^{-k+1}$$

finalmente

$$\left| \frac{y - fl(y)}{y} \right| \leq 0.5 \cdot 10^{-k+1}$$

Obs. La cota para el error relativo cuando se usa aritmética de k dígitos, es independiente del número que se está representando.

Errores de redondeo y aritmética del computador

Las cotas para el error relativo cuando se usa aritmética de *k* dígitos, son independientes del número que se está representando.

Esto se debe a la manera en que los "números de máquina decimal" o "números de punto flotante" están distribuidos a lo largo de R.

Debido a la forma exponencial de la característica, se usa la misma cantidad de "números de máquina decimal" para representar cada uno de los intervalos

$$[0.1,1],\ [1,10],\ [10,100],\ \ldots,\ [10^{n-1},10^{n}]$$

$$[0.1,1] = [0.1,1]\times 10^{0} \ \rightarrow\ \pm(0.d_1 d_2 \ldots d_k)\times 10^{0}$$
$$[1,10] = [0.1,1]\times 10^{1} \ \rightarrow\ \pm(0.d_1 d_2 \ldots d_k)\times 10^{1}$$
$$\ldots$$
$$[10^{n-1},10^{n}] = [0.1,1]\times 10^{n} \ \rightarrow\ \pm(0.d_1 d_2 \ldots d_k)\times 10^{n}$$

La cantidad de números es constante para todo entero n.

Errores de redondeo y aritmética del computador

Se dice que el número p^* aproxima a p con t **dígitos significativos** (o cifras), si t es el entero más grande no negativo para el cual

$$\left|\frac{p-p^*}{p}\right| < 0.5 \cdot 10^{-t}$$

Ejemplos:

Supongamos que p* aproxima a 1000 al menos con 3 cifras significativas, entonces

$$\left|\frac{1000-p^*}{1000}\right| < 5 \cdot 10^{-4}$$

$$\Leftrightarrow -0.5 < p^* - 1000 < 0.5$$

$$\Leftrightarrow 999.5 < p^* < 1000.5$$

Supongamos que p* aproxima a 5000 al menos con 3 cifras significativas, entonces

$$\left|\frac{5000-p^*}{5000}\right| < 5 \cdot 10^{-4}$$

$$\Leftrightarrow -2.5 < p^* - 5000 < 2.5$$

$$\Leftrightarrow 4997.5 < p^* < 5002.5$$

Errores de redondeo y aritmética del computador

La tabla siguiente ilustra la naturaleza continua del concepto de **dígitos significativos**, listando, para varios valores de *p*, la mínima cota superior de

$$| p - p^* |$$

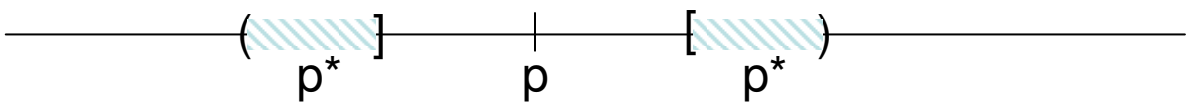
cuando p^* concuerda con p en tres cifras significativas

p	0.1	0.5	100	1000	5000	9990	10000
$máx \mid p - p^* \mid$	0.00005	0.00025	0.05	0.5	2.5	4.995	5

Errores de redondeo y aritmética del computador

Si el número p* tiene 3 cifras significativas respecto a otro número p, determinar la región exacta donde esta ubicado p*.

$$\left|\frac{p-p^*}{p}\right| < 0.5\cdot 10^{-3} \qquad \text{y} \qquad \left|\frac{p-p^*}{p}\right| \geq 0.5\cdot 10^{-3-1}$$



$$\left|\frac{p-p^*}{p}\right| < 0.5\cdot 10^{-3} \quad \Rightarrow \quad p^* \in (p - |p| 5\cdot 10^{-4}, p + |p| 5\cdot 10^{-4})$$

$$\left|\frac{p-p^*}{p}\right| \geq 0.5\cdot 10^{-3-1} \quad \Rightarrow \quad p^* \in (-\infty, p - |p| 5\cdot 10^{-5}] \cup [p + |p| 5\cdot 10^{-5}, +\infty)$$

Finalmente

$$p^* \in (p - |p| 5\cdot 10^{-4}, p - |p| 5\cdot 10^{-5}] \cup [p + |p| 5\cdot 10^{-5}, p + |p| 5\cdot 10^{-4})$$

Representación de números en el computador

Además de tener una representación inexacta de los números reales, la aritmética realizada en el computador no es exacta.

Supongamos que $\oplus$, $\ominus$, $\otimes$ y $\oslash$ representan los operaciones de suma, resta, multiplicación y división en el computador (aritmética idealizada):

$$x \oplus y = fl(fl(x) + fl(y))$$
$$x \ominus y = fl(fl(x) - fl(y))$$
$$x \otimes y = fl(fl(x) \times fl(y))$$
$$x \oslash y = fl(fl(x) / fl(y))$$

Representación de números en el computador

Ejemplo:

Consideremos la siguiente suma:

$$0.99 + 0.0044 + 0.0042.$$

Con aritmética exacta, el resultado es 0.9986.

Sin embargo, si usamos una aritmética de tres dígitos, y las operaciones se realizan siguiendo el orden de izquierda a derecha, encontramos que

$$(0.99+0.0044)+0.0042 = 0.994+0.0042 = 0.998.$$

Por otra parte, si operamos primero los dos últimos números, tenemos que:

$$0.99+(0.0044+0.0042) = 0.99+0.0086 = 0.999,$$

lo cual demuestra el efecto del error por redondeo en un caso tan simple como éste.

(¿Por qué sucede esto?).

Desde el punto de vista numérico es importante el orden en que sumamos.

Representación de números en el computador

Ejemplo: usando aritmética de cinco dígitos truncando, para $x = 1/3,\ \ y = 5/7$

	resultado	valor real	error abs	error rel
x⊕y	0.10476×10^{1}	22/21	0.190×10^{-4}	0.182×10^{-4}
xΘy	0.38095×10^{0}	8/21	0.238×10^{-5}	0.625×10^{-5}
x⊗y	0.23809×10^{0}	5/27	0.524×10^{-5}	0.220×10^{-4}
x∅y	0.21428×10^{1}	15/7	0.571×10^{-4}	0.267×10^{-4}

Error relativo máximo obtenido 0.267×10^{-4}, para lo cual la aritmética de cinco dígitos cortando produce resultados satisfactorios.

Ejemplo: usando aritmética de cinco dígitos truncando, para
$x = 1/3,\ \ y = 5/7,\ u = 0.714251,\ v = 98765.9,\ w = 0.11111 \times 10^{-4}$

	resultado	valor real	error abs	error rel
yΘu	0.30000×10^{-4}	0.34714×10^{-4}	0.471×10^{-5}	0.136×10^{0}
(yΘu)Øw	0.27000×10^{1}	0.31243×10^{1}	0.424×10^{0}	0.136×10^{0}
(yΘu)⊗v	0.29629×10^{1}	0.34285×10^{1}	0.465×10^{0}	0.136×10^{0}
u⊕v	0.98765×10^{5}	0.98766×10^{5}	0.161×10^{1}	0.163×10^{-4}

Pueden ocurrir errores significativos

Representación de números en el computador

Operaciones que producen errores:

- la suma de muchos números siempre genera errores de redondeo, lo importante es que estos errores no inutilicen el resultado!
- la división de un número con dígitos finitos entre un número de magnitud muy pequeña
- la multiplicación de un número con dígitos finitos por un número relativamente grande
- la sustracción de números casi iguales

$$fl(x) = 0.a_1a_2 \cdots a_p a_{p+1} \cdots a_k \times 10^n \qquad fl(y) = 0.a_1a_2 \cdots a_p b_{p+1} \cdots b_k \times 10^n$$

$$fl(fl(x) - fl(y)) = 0.d_{p+1} \cdots d_k \times 10^{n-p}$$

así, el número usado para representar *x-y* tendrá sólo *k-p* cifras significativas (en la mayoría de los computadores, se le asignarán *k* dígitos, pero los últimos *p* serán asignados al azar).

Cualquier cálculo adicional que involucre *x-y* retendría el problema de tener solamente *k-p* cifras significativas.

Con el objeto de evitar esta dificultad se recomienda buscar una reformulación del algoritmo inicialmente considerado.

Representación de números en el computador

Ejemplo:
Consideremos $x^2 + 62.10\,x + 1 = 0$, cuyas raíces tienen los valores aproximados

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

En esta ecuación, b^2 es mucho mayor que 4, de modo que el numerador en el cálculo de x_1 implica la resta de números casi iguales.
Ahora bien, si suponemos una aritmética de redondeo a cuatro cifras, y como

$$\Delta = \sqrt{b^2 - 4} = \sqrt{3856.0 - 4.000} = \sqrt{3852.} = 62.06,$$

tenemos que

$$fl(x_1) = \frac{-62.10 + 62.06}{2.000} = \frac{-0.0400}{2.000} = -0.0200,$$

la cual es una mala aproximación a $x_1 = 0.01611$, con un error relativo grande de $\approx 2.4 \times 10^{-1}$.

Representación de números en el computador

Ejemplo (cont.):
Esta dificultad puede evitarse racionalizando el numerador en la fórmula cuadrática

$$x_1 = \frac{(\Delta - b)(\Delta + b)}{2(\Delta + b)} = \frac{(\Delta^2 - b^2)}{2(\Delta + b)} = \frac{-4}{2(\Delta + b)} = \frac{-2}{(\Delta + b)}.$$

Se obtiene en este caso

$$x_1 = \frac{-2.000}{62.10 + 62.06} = \frac{-2.000}{124.2} = -0.01610,$$

con un error relativo de $\approx 6.2 \times 10^{-4}$, mucho menor que el anterior.

Por otro lado, el cálculo de x_2 implica la suma de dos números casi iguales.

$$-b \quad \text{y} \quad -\sqrt{b^2 - 4}.$$

Pero esto no representa problema alguno, pues

$$fl(x_2) = \frac{-62.10 - 62.06}{2.000} = \frac{-124.2}{2.000} = -62.10,$$

que tiene un error relativo pequeño de $\approx 3.2 \times 10^{-4}$.

Algoritmos y convergencia

Un algoritmo es un procedimiento que describe, sin ninguna ambigüedad, una sucesión finita de pasos a realizar en un orden especifico.

El objetivo de un algoritmo será generalmente el de implantar un procedimiento numérico para resolver un problema o aproximar una solución del problema.

Como vehículo para describir algoritmos usaremos un “pseudocódigo”.

Los pasos en los algoritmos se arreglan de tal manera que la dificultad de traducir en un lenguaje de programación (MATLAB, FORTRAN, C, …) apropiado para aplicaciones científicas sea mínimo.

Algoritmos y convergencia

Ejemplo:

algoritmo para calcular la suma de los números x_1 al x_n, es decir,

$$\sum_{i=1}^{n} x_i = x_1 + x_2 + \cdots + x_n$$

donde n y los números x_1 al x_n están dados.

Entrada: n, x_1, x_2, …, x_n

P1: sum = 0

P2: para i = 1 hasta n

sum = sum + x_i

fin para

P3: escribir sum

P4: parar

Obs. Considerar la suma

s = 9.87 + 0.78 + 0.05 + 0.01

Al realizar esta operación de izquierda a derecha y de derecha a izquierda usando aritmética de 3 dígitos con redondeo, se obtienen los resultados:

10.8 y 10.7

Algoritmos y convergencia

Un **algoritmo** es una secuencia finita de operaciones algebraicas y lógicas que producen una solución aproximada de un problema matemático

Análisis Numérico $\rightarrow$ diseño de algoritmos y estudio de su eficiencia

Eficiencia $\rightarrow$
- requerimiento de memoria,
- tiempo de cálculo (rapidez)
- estimación del error (precisión)

$$\underbrace{\text{errores de entrada (en medidas)} + \underbrace{\text{errores de almacenamiento} + \text{errores de algoritmo}}_{\text{a analizar}}}_{} = \text{errores de salida}$$

Algoritmos y convergencia

Los **errores** se propagan a través de los cálculos, debido a la estructura propia del algoritmo. Para estudiar esta propagación y por lo tanto el error final, atendemos a los conceptos de condicionamiento y estabilidad.

Condicionamiento: mide la influencia que tendrían los errores en los datos en el caso en que se pueda trabajar con aritmética exacta. No depende del algoritmo sino del problema en si.

Estabilidad: está relacionada con la influencia que tienen en los resultados finales la acumulación de errores que se producen al realizar las diferentes operaciones elementales que constituyen el algoritmo.

condicionamiento y estabilidad	$\rightarrow$	permite estudiar la precisión de un algoritmo para un problema concreto

Algoritmos y convergencia

Condicionamiento

Diremos que un problema está mal condicionado cuando pequeños cambios en los datos dan lugar a grandes cambios es las respuestas. Para estudiar el condicionamiento de un problema se introduce el llamado número de condición de ese problema, especifico del problema, que es mejor cuando más cerca de 1 (el problema está bien condicionado) y peor cuando más grande sea (mal condicionado).

> La gravedad de un problema mal condicionado reside en que su resolución puede producir soluciones muy dispares en cuanto los datos cambien muy poco.

Estabilidad

Todo algoritmo que resuelve un problema numéricamente produce en cada paso un error numérico.
Un algoritmo se dice inestable cuando los errores que se cometen en cada etapa del mismo van aumentando de forma progresiva, de manera que el resultado final pierde gran parte de su exactitud.
Un algoritmo es estable cuando no es inestable (está controlado).

Algoritmos y convergencia

Sistema mal condicionado

Ejemplo:

sistema 1

$$\begin{aligned} 10x_1 + 7x_2 + 8x_3 + 7x_4 &= 32 \\ 7x_1 + 5x_2 + 6x_3 + 5x_4 &= 23 \\ 8x_1 + 6x_2 + 10x_3 + 9x_4 &= 33 \\ 7x_1 + 5x_2 + 9x_3 + 10x_4 &= 31 \end{aligned} \quad \Rightarrow \quad \begin{aligned} x_1 &= 1 \\ x_2 &= 1 \\ x_3 &= 1 \\ x_4 &= 1 \end{aligned}$$

sistema 2

$$\begin{aligned} 10x_1 + 7x_2 + 7.983x_3 + 7.019x_4 &= 32 \\ 7.08x_1 + 5.04x_2 + 6x_3 + 5x_4 &= 23 \\ 8x_1 + 5.98x_2 + 9.89x_3 + 9x_4 &= 33 \\ 6.99x_1 + 4.99x_2 + 9x_3 + 9.98x_4 &= 31 \end{aligned}$$

$$\Downarrow$$

$$x_1 = -81,\ x_2 = 137,\ x_3 = -34,\ x_4 = 22$$

sistema 3

$$\begin{aligned} 10x_1 + 7x_2 + 8x_3 + 7x_4 &= 32.1 \\ 7x_1 + 5x_2 + 6x_3 + 5x_4 &= 22.9 \\ 8x_1 + 6x_2 + 10x_3 + 9x_4 &= 32.98 \\ 7x_1 + 5x_2 + 9x_3 + 10x_4 &= 31.02 \end{aligned}$$

$$\Downarrow$$

$$x_1 = 7.28,\ x_2 = -9.36,\ x_3 = 3.54,\ x_4 = -0.5$$

Pequeños cambios en los datos en algunos elementos producen grandes cambios en las soluciones:

- entre los sistemas 1 y 2 cambios del orden de 11 centésimas (en la matriz) producen variaciones de hasta 136 unidades en la solución,
- entre los sistemas 1 y 3 cambios del orden de 1 décima (en el término de la derecha) producen variaciones de hasta de 10 unidades en la solución.

condicionamiento.m

Algoritmos y convergencia

Estabilidad

Supongamos que E_n representa el crecimiento del error después de *n* operaciones subsecuentes.

Si $|E_n| \approx c\, n + \varepsilon$, donde *c* es una constante independiente de n, diremos que el crecimiento del error es lineal.

Si $|E_n| \approx k^n\, \varepsilon$, para $k > 1$, el crecimiento del error es exponencial.

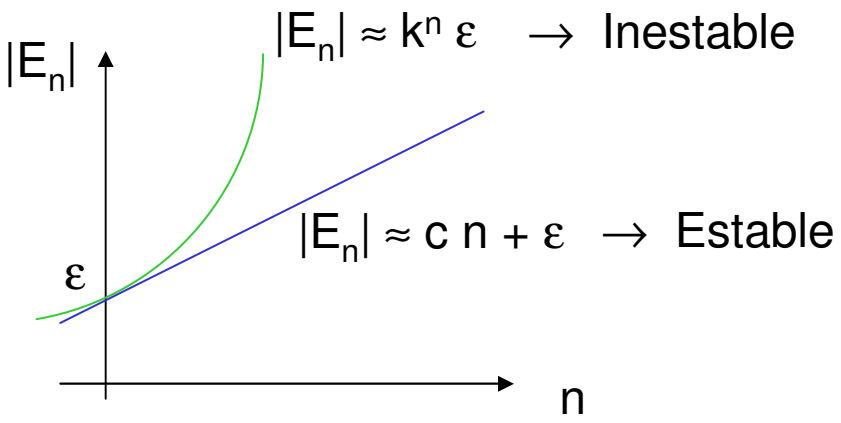
- El crecimiento del error es inevitable, y cuando el crecimiento es lineal y *c* y ε son pequeños, los resultados son generalmente aceptables.
- El crecimiento exponencial del error debe ser evitado, ya que el término k^n será grande aún para valores relativamente pequeños de ε.

Algoritmos y convergencia

Estabilidad

Consecuencias:

- Un algoritmo que exhibe crecimiento lineal del error es ESTABLE
- Un algoritmo en el que el crecimiento del error es exponencial es INESTABLE



Algoritmos y convergencia

Diremos que un software o subprograma es **robusto** si es capaz de enfrentarse a una amplia variedad de situaciones numéricas distintas sin la intervención del usuario.

Es un hecho conocido que cuando estudiamos diferentes métodos del cálculo científico se presenta una interdependencia entre la **velocidad** y la **confiabilidad** (la primera está directamente relacionada con los costos).

Así, para algunos problemas que involucran cálculo intensivo (como la solución numérica de ecuaciones en derivadas parciales), la velocidad es lo más importante.

Sin embargo, cuando nos referimos a un software de propósito general para ser utilizado por una amplia gama de usuarios, la confiabilidad y la robustez son las características que marcan la pauta.

Por muchos años se han realizado esfuerzos por generar paquetes numéricos de propósito general que reúnan ambas características, confiabilidad y robustez.

Algoritmos y convergencia

El siguiente ejemplo muestra lo que puede suceder cuando escogemos un algoritmo inadecuado. El error por redondeo puede acabar completamente con el resultado de un cálculo.

Supongamos que queremos estimar para n = 0, 1,…, 8, la integral

$$y_n = \int_0^1 \frac{x^n}{x+5}\,dx.$$

Observamos que

$$y_n + 5y_{n-1} = \int_0^1 \frac{x^n + 5x^{n-1}}{x+5}\,dx = \int_0^1 \frac{x^{n-1}(x+5)}{x+5}\,dx = \frac{1}{n}.$$

Supongamos que en nuestros cálculos usamos sólo tres dígitos

$$y_0 = \int_0^1 \frac{1}{x+5}\,dx = \ln(x+5)\,|_0^1 \approx 0.182.$$

error en este cálculo $\delta = |\frac{p-p^*}{p}| < 0.5\cdot 10^{-3} = 5\cdot 10^{-4}$

Algoritmos y convergencia

Observamos que la sucesión en decreciente y todos sus términos son positivos

$$y_n \geq y_{n+1} \qquad \text{y} \qquad y_n \geq 0$$

$x \in [0,1]$

$$n \in N \quad \Rightarrow \quad x^n \geq x^{n+1} \quad \Rightarrow \quad \frac{x^n}{x+5} \geq \frac{x^{n+1}}{x+5}$$

$$\Rightarrow \quad \int_0^1 \frac{x^n}{x+5}\,dx \geq \int_0^1 \frac{x^{n+1}}{x+5}\,dx \quad \Rightarrow \quad y_n \geq y_{n+1}$$

$x \in [0,1]$

$$\frac{x^n}{x+5} \geq 0$$

así, la integral de una función positiva es positiva, es decir,

$$y_n = \int_0^1 \frac{x^n}{x+5}\,dx \geq 0$$

Algoritmos y convergencia

- Consideremos el algoritmo

$$y_1 = 1 - 5y_0 = 1 - 0.910 \approx 0.090$$

$$y_2 = \frac{1}{2} - 5y_1 \approx 0.050$$

$$y_3 = \frac{1}{3} - 5y_2 \approx 0.083 \qquad (\text{¡}y_3 > y_2\text{!})$$

$$y_4 = \frac{1}{4} - 5y_3 \approx -0.165 \qquad \text{¡sin sentido!}$$

La causa de este resultado está en que el error por redondeo δ en y_0 (cuya magnitud es del orden de 5×10^{-4}) se multiplica por -5 en el cálculo de y_1, el cual tendrá entonces un error de -5δ.

Ese error produce, a su vez, un error en y_2 de 25δ, en y3 de -125δ, y en y_4 de 625δ (en el que el error será tan grande como $625 \times 5 \times 10^{-4} = 0.3125$).

Si usáramos una mayor precisión, con más lugares decimales, los resultados "sin sentido" aparecerán en una etapa posterior.

Este fenómeno (por supuesto, indeseable) se conoce como **inestabilidad numérica**. La inestabilidad numérica puede corregirse si encontramos un algoritmo más adecuado.

inestable_progresivo.m

Algoritmos y convergencia

- Consideremos el algoritmo $$y_{n-1} = \frac{1}{5n} - \frac{y_n}{5}.$$

En este caso el error estaría, en cada paso, dividido por -5 (aunque necesitaremos un valor de entrada).

$$y_9 + 5y_9 = \frac{1}{10} \Rightarrow y_9 \approx \frac{1}{60} \approx 0.017$$

Observemos directamente de la definición de y_n , que la misma decrece cuando n aumenta.

Por lo que podemos asumir que, cuando n es grande, $y_{n+1} \approx y_n$.

De esta manera, podemos por conveniencia suponer que $y_{10} \approx y_9$, de donde sigue que

$$y_8 = \frac{1}{45} - \frac{y_9}{5} \approx 0.019$$

$$y_7 = \frac{1}{40} - \frac{y_8}{5} \approx 0.021$$

$$y_6 \approx 0.025, \quad y_5 \approx 0.028,$$

$$y_4 \approx 0.034, \quad y_3 \approx 0.043,$$

$$y_2 \approx 0.058, \quad y_1 \approx 0.088,$$

$$y_0 \approx 0.182$$

¡bueno!

estable_regresivo.m