Representación de números

MAT-251

Dr. Alonso Ramírez Manzanares
CIMAT A.C.
e-mail: alram@cimat.mx
web: http://www.cimat.mx/~alram/met_num/

Dr. Salvador Botello Rionda
CIMAT A.C.
e-mail: botello@cimat.mx

Aritmética de cómputadora

- La aritmetica que usamos en la computadora es distinta a la usada en la teoría de álgebra o cálculo

$$(\sqrt{7})^2 \neq 7$$

```
double n, test;
n=7.0;
test = sqrt(n)*sqrt(n);
if ( test == n)
    cout << endl << "iguales" << endl;
else {
    cout << endl << "diferentes" << endl;
}
```

Aritmética de cómputadora

- La aritmetica que usamos en la computadora es distinta a la usada en la teoría de álgebra o cálculo

$$(\sqrt{7})^2 \neq 7$$

```
double n, test;
n=7.0;
test = sqrt(n)*sqrt(n);
if ( test == n)
    cout << endl << "iguales" << endl;
else {
    cout << endl << "diferentes" << endl;
}
```

- ¿0.1 = 0.1 ? Estos errores se deben a la precisión finita de la máquina

Expansión en finita de 0.1 en base 2

Converting	Result
0.1	0.
$0.1 \times 2 = \mathbf{0.2} < 1$	0.0
$0.2 \times 2 = \mathbf{0.4} < 1$	0.00
$0.4 \times 2 = \mathbf{0.8} < 1$	0.000
$0.8 \times 2 = \mathbf{1.6} \geq 1$	0.0001
$0.6 \times 2 = \mathbf{1.2} \geq 1$	0.00011
$0.2 \times 2 = \mathbf{0.4} < 1$	0.000110
$0.4 \times 2 = \mathbf{0.8} < 1$	0.0001100
$0.8 \times 2 = \mathbf{1.6} \geq 1$	0.00011001
$0.6 \times 2 = \mathbf{1.2} \geq 1$	0.000110011
$0.2 \times 2 = \mathbf{0.4} < 1$	0.0001100110

Aritmética de cómputadora

- Tenemos un numero finito de dígitos para representar a los números reales

- Solo un subconjunto de los racionales se pueden representar de manera exacta.

- Los errores que se producen en la aritmética computacional se deben al ***error de redondeo***. Los cálculos son hechos con una aproximación de los números.

- Nosotros usamos el standard de la IEEE (1985) para precisión sencilla, doble y extendida (***float***, ***double*** y ***long double***).

- Ejemplo: doble precisión requiere 64 bits.

```
cout << endl << sizeof(float) << endl;
cout << endl << sizeof(double) << endl;
cout << endl << sizeof(long double) << endl;
```

Representación de números **reales** (de punto flotante)

- La representación en bits está dada de la siguiente manera

signo	exponente	fracción
s	c	f
1 bit	*q* bits	*n-1-q* bits

Representación de números **reales** (de punto flotante)

- La representación en bits está dada de la siguiente manera

signo	exponente	fracción
s	c	f
1 bit	*q* bits	*n-1-q* bits

La base del exponente es: ***b***=2

Representación de números **reales** (de punto flotante)

- La representación en bits está dada de la siguiente manera

signo	exponente	fracción
s	c	f
1 bit	*q* bits	*n-1-q* bits

La base del exponente es: ***b***=2

$$e = c - (2^{q-1} - 1)$$

Representación de números **reales** (de punto flotante)

- La representación en bits está dada de la siguiente manera

signo	exponente	fracción
s	c	f
1 bit	*q* bits	*n-1-q* bits

La base del exponente es: ***b***=2

$$e = c - (2^{q-1} - 1)$$

elemento	signo	exponente	mantisa
float	1	8	23
double	1	11	52

Representación de números **reales** (de punto flotante)

- La representación en bits está dada de la siguiente manera

signo	exponente	fracción
s	c	f
1 bit	*q* bits	*n-1-q* bits

La base del exponente es: ***b***=2

$$e = c - (2^{q-1} - 1)$$

elemento	signo	exponente	mantisa
float	1	8	23
double	1	11	52

para double

Representación de números **reales** (de punto flotante)

- La representación en bits está dada de la siguiente manera

signo	exponente	fracción
s	c	f
1 bit	*q* bits	*n-1-q* bits

La base del exponente es: ***b***=2

$$e = c - (2^{q-1} - 1)$$

elemento	signo	exponente	mantisa
float	1	8	23
double	1	11	52

para double

$$(-1)^s * 2^{c-1023} * (1+f)$$

Representación de números **reales** (de punto flotante)

- La representación en bits está dada de la siguiente manera

signo	exponente	fracción
s	c	f
1 bit	*q* bits	*n-1-q* bits

La base del exponente es: ***b***=2

$$e = c - (2^{q-1} - 1)$$

elemento	signo	exponente	mantisa
float	1	8	23
double	1	11	52

para double

$$(-1)^s * 2^{c-1023} * (1 + f)$$

Para float, *e = c-127, por lo tanto c = e+127*

Representación de números **reales** (de punto flotante)

- La representación en bits está dada de la siguiente manera

signo	exponente	fracción
s	c	f
1 bit	*q* bits	*n-1-q* bits

La base del exponente es: ***b***=2

$$e = c - (2^{q-1} - 1)$$

elemento	signo	exponente	mantisa
float	1	8	23
double	1	11	52

para double

$$(-1)^s * 2^{c-1023} * (1 + f)$$

Para float, *e = c-127, por lo tanto c = e+127*

exponentes de -127 (todos 0s) and +128 (c=255 -> todos 1s) están reservados para números especiales

Representación de números **reales** (de punto flotante)

- Casos especiales

Clase	Exp	Fracción
Ceros	0	0
Números desnormalizados	0	distinto de 0
Números normalizados	1-254	cualquiera
Infinitos	255	0
NaN (Not a Number)	255	distinto de 0

- Ver ejemplo de float en http://www.h-schmidt.net/FloatConverter/IEEE754.html

Representación de números **reales** (de punto flotante), hablemos en particular de los doubles

- En los doubles tenemos 64 bits, de tal forma $(-1)^s * 2^{c-1023} * (1+f)$

- Para el número de ejemplo

$$0\ 10000000011\ 1011100100010000000000000000000000000000000000000000.$$

  - tenemos:

$$c = 1 \cdot 2^{10} + 0 \cdot 2^9 + \cdots + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1024 + 2 + 1 = 1027.$$

$$f = 1 \cdot \left(\frac{1}{2}\right)^1 + 1 \cdot \left(\frac{1}{2}\right)^3 + 1 \cdot \left(\frac{1}{2}\right)^4 + 1 \cdot \left(\frac{1}{2}\right)^5 + 1 \cdot \left(\frac{1}{2}\right)^8 + 1 \cdot \left(\frac{1}{2}\right)^{12}$$

  - de tal forma que el valor final es:

$$\begin{aligned} &(-1)^s * 2^{c-1023} * (1+f) \\ &= (-1)^0 \cdot 2^{1027-1023} \left(1 + \left(\frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096}\right)\right) \\ &= 27.56640625. \end{aligned}$$

Representación de números **reales** (de punto flotante), hablemos ahora de los doubles

- De tal forma que para este número 27.56640625

$$0\ 10000000011\ 1011100100010000000000000000000000000000000000000000.$$

- El número más grande que es menor que el número original es

$$0\ 10000000011\ 1011100100001111111111111111111111111111111111111111$$

- El número mas pequeño que es mas grande que el # original es:

$$0\ 10000000011\ 1011100100010000000000000000000000000000000000000001$$

- Entonces cada número representa en realidad un intervalo, en este caso:

$$[27.5664062499999988897769753748434595763683319091796875,$$
$$27.5664062500000011102230246251565404236316680908203125).$$

Representación de números **reales** (de punto flotante), hablemos ahora de los doubles

$$(-1)^s * 2^{c-1023} * (1+f)$$

- El número normalizado más pequeño positivo es (c = 1)

$$2^{-1022} \cdot (1+0) \approx 0.225 \times 10^{-307},$$

- El número normalizado más grande positivo es (c = 2046)

$$2^{1023} \cdot (1 + (1 - 2^{-52})) \approx 0.17977 \times 10^{309}$$

- Con 53 posiciones en la mantisa binaria, tenemos aproximadamente 16 dígitos de precisión en base 10.
- Por la forma exponencial tenemos la misma cantidad de números entre exponentes 2^n y 2^{n+1}

Truncamiento y redondeo

- Números mas pequeños producen un subdesbordamiento (underflow) y se tratan como un cero a nivel máquina.

- Números mas grandes producen un desbordamiento (overflow) y se genera un error que detiene los cálculos. A menos que ... ¿Recuerdan el try-catch?

- Supongamos la forma normalizada de cualquier número en su forma decimal

$$\pm 0.d_1 d_2 \ldots d_k \times 10^n, \quad 1 \leq d_1 \leq 9, \quad 0 \leq d_i \leq 9 \quad \textit{para } i = 2,..,k$$

Truncamiento y redondeo

- Podemos ponerlo en esta notación

$$y = 0.d_1 d_2 \ldots d_k d_{k+1} d_{k+2} \ldots \times 10^n.$$

- La forma de punto flotante de ***y*** que denotamos ***fl(y)*** se obtiene terminando la mantisa de ***y*** en los primero ***k*** dígitos decimales, esto se llama ***truncamiento***:

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

- La otra forma de obtenerlo es mediante redondeo: Si el (***k+1***)-ésimo dígito es menor que 5 se hace truncamiento (redondeo hacia abajo), de lo contrario se agrega un 1 al ***k***-ésimo digito y se le aplica truncamiento al número resultante (redondeo hacia arriba). Esto se obtiene agregando $5 \times 10^{n-(k+1)}$ a ***y*** y luego truncando.

- Ejemplo, con el número irracional $\pi = 0.314159265 \ldots \times 10^1.$

Error de redondeo

$\neq$

Error de redondeo

- El ***error de redondeo*** es aquel que resulta de sustituir un número por su forma de punto flotante (ojo con la notación, no importa si se hizo **redondeo** o **truncamiento**).

$\neq$

Error de redondeo

- El ***error de redondeo*** es aquel que resulta de sustituir un número por su forma de punto flotante (ojo con la notación, no importa si se hizo **redondeo** o **truncamiento**).

- Sea p^* una aproximación de p, entonces el error absoluto es $e_a = |p - p^*|$

$\neq$

Error de redondeo

- El ***error de redondeo*** es aquel que resulta de sustituir un número por su forma de punto flotante (ojo con la notación, no importa si se hizo **redondeo** o **truncamiento**).

- Sea p^* una aproximación de p, entonces el error absoluto es $e_a = |p - p^*|$

- El error relativo se define como $e_r = |p - p^*| / |p|$ siempre y cuando $p \neq 0$

Error de redondeo

- El ***error de redondeo*** es aquel que resulta de sustituir un número por su forma de punto flotante (ojo con la notación, no importa si se hizo **redondeo** o **truncamiento**).

- Sea $\boldsymbol{p^*}$ una aproximación de $\boldsymbol{p}$, entonces el error absoluto es $\mathbf{e_a = |p - p^*|}$

- El error relativo se define como $\mathbf{e_r = |p - p^*| / |p|}$ siempre y cuando $\mathbf{p \neq 0}$

- ¿Cual es la ventaja de usar el error relativo?, que es consistente:

Error de redondeo

- El ***error de redondeo*** es aquel que resulta de sustituir un número por su forma de punto flotante (ojo con la notación, no importa si se hizo **redondeo** o **truncamiento**).

- Sea p^* una aproximación de p, entonces el error absoluto es $e_a = |p - p^*|$

- El error relativo se define como $e_r = |p - p^*| / |p|$ siempre y cuando $p \neq 0$

- ¿Cual es la ventaja de usar el error relativo?, que es consistente:
  - $p=0.3\times10^{1}$, $p^*=0.31\times10^{1}$ entonces $e_a = 0.1$ y $e_r = 0.3333\times10^{-1}$

Error de redondeo

- El ***error de redondeo*** es aquel que resulta de sustituir un número por su forma de punto flotante (ojo con la notación, no importa si se hizo **redondeo** o **truncamiento**).

- Sea p^* una aproximación de p, entonces el error absoluto es $e_a = |p - p^*|$

- El error relativo se define como $e_r = |p - p^*| / |p|$ siempre y cuando $p \neq 0$

- ¿Cual es la ventaja de usar el error relativo?, que es consistente:

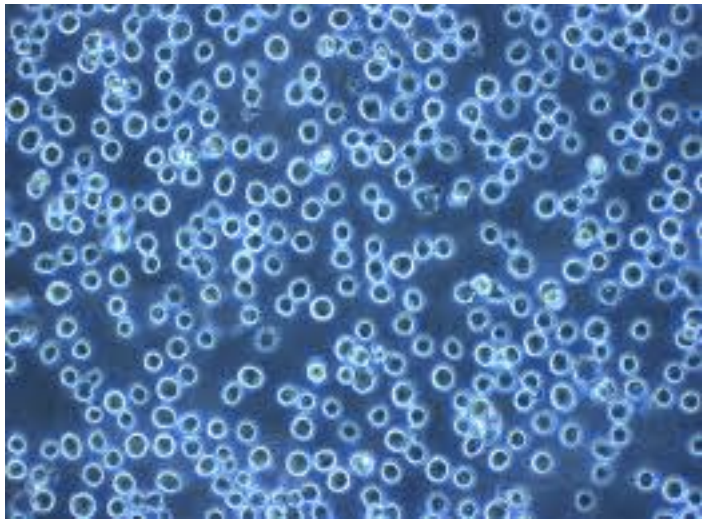
  - $p=0.3\times10^{1}$, $p^*=0.31\times10^{1}$ entonces $e_a = 0.1$ y $e_r = 0.3333\times10^{-1}$

  - $p=0.3\times10^{-3}$, $p^*=0.31\times10^{-3}$ entonces $e_a = 0.1\times10^{-4}$ y $e_r = 0.3333\times10^{-1}$

Error de redondeo

- El ***error de redondeo*** es aquel que resulta de sustituir un número por su forma de punto flotante (ojo con la notación, no importa si se hizo **redondeo** o **truncamiento**).
- Sea $\boldsymbol{p^*}$ una aproximación de $\boldsymbol{p}$, entonces el error absoluto es $\boldsymbol{e_a = |p - p^*|}$
- El error relativo se define como $\boldsymbol{e_r = |p - p^*| / |p|}$ siempre y cuando $\mathbf{p \neq 0}$
- ¿Cual es la ventaja de usar el error relativo?, que es consistente:
  - $\boldsymbol{p}=0.3\text{x}10^{1}$, $p^*=0.31\text{x}10^{1}$ entonces $\boldsymbol{e_a} = 0.1$ y $\boldsymbol{e_r} = 0.3333\text{x}10^{-1}$
  - $\boldsymbol{p}=0.3\text{x}10^{-3}$, $p^*=0.31\text{x}10^{-3}$ entonces $\boldsymbol{e_a} = 0.1\text{x}10^{-4}$ y $\boldsymbol{e_r} = 0.3333\text{x}10^{-1}$
  - $\boldsymbol{p}=0.3\text{x}10^{4}$, $p^*=0.31\text{x}10^{4}$ entonces $\boldsymbol{e_a} = 0.1\text{x}10^{3}$ y $\boldsymbol{e_r} = 0.3333\text{x}10^{-1}$

¿cuál error es mas grave?



- ¿reportar que solo hay un borrego cuando en realidad hay 2
- ¿reportar que hay 1001 celulas cuando en realidad hay 1000?

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

$$\left|\frac{y-fl(y)}{y}\right|.$$

Si se usan k cifras decimales y el truncamiento para la representación en la máquina de

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

$$\left|\frac{y - fl(y)}{y}\right|.$$

Si se usan k cifras decimales y el truncamiento para la representación en la máquina de

$$y = 0.d_1 d_2 \ldots d_k d_{k+1} \ldots \times 10^n,$$

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

$$\left|\frac{y - fl(y)}{y}\right|.$$

Si se usan k cifras decimales y el truncamiento para la representación en la máquina de

$$y = 0.d_1 d_2 \ldots d_k d_{k+1} \ldots \times 10^n,$$

entonces

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

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

$$\left|\frac{y - fl(y)}{y}\right|.$$

Si se usan k cifras decimales y el truncamiento para la representación en la máquina de

$$y = 0.d_1 d_2 \ldots d_k d_{k+1} \ldots \times 10^n,$$

entonces

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

$$= \left|\frac{0.d_{k+1} d_{k+2} \ldots \times 10^{n-k}}{0.d_1 d_2 \ldots \times 10^n}\right| = \left|\frac{0.d_{k+1}\, d_{k+2} \ldots}{0.d_1 d_2 \ldots}\right| \times 10^{-k}.$$

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

$$\left| \frac{y - fl(y)}{y} \right|.$$

Si se usan k cifras decimales y el truncamiento para la representación en la máquina de

$$y = 0.d_1 d_2 \ldots d_k d_{k+1} \ldots \times 10^n,$$

entonces

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

$$= \left| \frac{0.d_{k+1} d_{k+2} \ldots \times 10^{n-k}}{0.d_1 d_2 \ldots \times 10^n} \right| = \left| \frac{0.d_{k+1}\, d_{k+2} \ldots}{0.d_1 d_2 \ldots} \right| \times 10^{-k}.$$

Como $d_1 \neq 0$, el valor mínimo del denominador es 0.1. El 1 es la cota superior del numerador. En consecuencia,

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

Error de redondeo

- Relación entre el error relativo de redondeo y las ***k*** cifras significativas de la aproximación numérica

$$\left|\frac{y-fl(y)}{y}\right|.$$

Si se usan k cifras decimales y el truncamiento para la representación en la máquina de

$$y = 0.d_1d_2\ldots d_kd_{k+1}\ldots \times 10^n,$$

entonces

$$\left|\frac{y-fl(y)}{y}\right| = \left|\frac{0.d_1d_2\ldots d_kd_{k+1}\ldots \times 10^n - 0.d_1d_2\ldots d_k \times 10^n}{0.d_1d_2\ldots \times 10^n}\right|$$

$$= \left|\frac{0.d_{k+1}d_{k+2}\ldots \times 10^{n-k}}{0.d_1d_2\ldots \times 10^n}\right| = \left|\frac{0.d_{k+1}\,d_{k+2}\ldots}{0.d_1d_2\ldots}\right| \times 10^{-k}.$$

Como $d_1 \neq 0$, el valor mínimo del denominador es 0.1. El 1 es la cota superior del numerador. En consecuencia,

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

Error de redondeo

- sea **u** = 0.714251, **v** = 98765.9 y **w** = 0.111111x10^{-4} , las operaciones de máquina, por ejemplo la suma, se expresa como:

  - ***$x \oplus y$ = fl(fl(x) + fl(y)),***

  - ***x = 5/7 , fl(x) = 0.71428****, usaremos 5 cifras*

Operación	Resultado	Valor real	Error absoluto	Error relativo
$x \ominus u$	0.30000×10^{-4}	0.34714×10^{-4}	0.471×10^{-5}	0.136
$(x \ominus u) \oplus w$	0.29629×10^{1}	0.34285×10^{1}	0.465	0.136
$(x \ominus u) \otimes v$	0.29629×10^{1}	0.34285×10^{1}	0.465	0.136
$u \oplus v$	0.98765×10^{5}	0.98766×10^{5}	0.161×10^{1}	0.163×10^{-4}

Error de redondeo

- Como es de esperarse, las operaciones aritméticas de suma, resta, multiplicacion, etc, introducen errores de redondeo (corrimientos y operaciones lógicas a bits).
- Todo esto es con aritmética de numero finito de digitos: *finite-digit arithmetic*.
- Veamos un ejemplo con 4 dígitos de precisión en forma normalizada, con la resta de números que son muy parecidos

	A	B	C	D
1		"exactos"	aproximaciones	Errores relativos
2	pi	3.14159265358979	3.141	0.000188647496713457
3	22/7	3.14285714285714	3.142	0.000272727272727283
4	diferencia	-0.00126448926734968	-0.001	**0.209166874072524**

Error de redondeo

- Las cifras significativas de una resta de números “parecidos” ***x*** y ***y***

$$fl(x) = 0.d_1 d_2 \ldots d_p \alpha_{p+1} \alpha_{p+2} \ldots \alpha_k \times 10^n,$$

y

$$fl(y) = 0.d_1 d_2 \ldots d_p \beta_{p+1} \beta_{p+2} \ldots \beta_k \times 10^n,$$

Error de redondeo

- Las cifras significativas de una resta de números "parecidos" ***x*** y ***y***

$$fl(x) = 0.d_1d_2 \ldots d_p\alpha_{p+1}\alpha_{p+2} \ldots \alpha_k \times 10^n,$$

y

$$fl(y) = 0.d_1d_2 \ldots d_p\beta_{p+1}\beta_{p+2} \ldots \beta_k \times 10^n,$$

La forma de punto flotante de $x - y$ es

$$fl(fl(x) - fl(y)) = 0.\sigma_{p+1}\sigma_{p+2} \ldots \sigma_k \times 10^{n-p},$$

donde

$$0.\sigma_{p+1}\sigma_{p+2} \ldots \sigma_k = 0.\alpha_{p+1}\alpha_{p+2} \ldots \alpha_k - 0.\beta_{p+1}\beta_{p+2} \ldots \beta_k.$$

- Lo malo es que se le asignarán *k* cifras significativas y por lo tanto *p* cifras tendrán ceros o peor aún ¡¡basura!!.

Error de redondeo

- Si tenemos un error, este aumenta al dividir por un número pequeño o al multiplicar por uno grande. Supongamos que por resultado de un cálculo o por la representación tenemos el error en ***z*** como ***z + d*** y dividimos por un número pequeño epsilon, ***10***

$$\frac{z}{\epsilon} \approx= fl\left(\frac{fl(z)}{fl(\epsilon)}\right) = (z+d) \times 10^n$$

Error de redondeo

- Si tenemos un error, este aumenta al dividir por un número pequeño o al multiplicar por uno grande. Supongamos que por resultado de un cálculo o por la representación tenemos el error en ***z*** como ***z + d*** y dividimos por un número pequeño epsilon, $\boldsymbol{10^{-n}}$ con ***n***>0:

$$\frac{z}{\epsilon} \approx= fl\left(\frac{fl(z)}{fl(\epsilon)}\right) = (z+d) \times 10^{n}$$

Error de redondeo

- Si tenemos un error, este aumenta al dividir por un número pequeño o al multiplicar por uno grande. Supongamos que por resultado de un cálculo o por la representación tenemos el error en ***z*** como ***z + d*** y dividimos por un número pequeño epsilon, $\boldsymbol{10^{-n}}$ con ***n***>0:

$$\frac{z}{\epsilon} \approx= fl\left(\frac{fl(z)}{fl(\epsilon)}\right) = (z+d) \times 10^{n}$$

El error se magnifica

Casos **graves** de errores de redondeo

Casos **graves** de errores de redondeo

- Ariane 5 Flight 501, http://en.wikipedia.org/wiki/Ariane_5_Flight_501

Casos **graves** de errores de redondeo

- Ariane 5 Flight 501, http://en.wikipedia.org/wiki/Ariane_5_Flight_501
  - Conversión sin protección (sin protección por motivos de eficiencia) de float (64 bits) a entero con signo (16 bits)

Casos **graves** de errores de redondeo

- Ariane 5 Flight 501, http://en.wikipedia.org/wiki/Ariane_5_Flight_501
    - Conversión sin protección (sin protección por motivos de eficiencia) de float (64 bits) a entero con signo (16 bits)
- Misil Patriot, donde se guarda un factor de 1/10 en 24 bits para convertir de décimas-de-segundos a segundos, un error de .3 seg = 500 metros.

Casos **graves** de errores de redondeo

- Ariane 5 Flight 501, http://en.wikipedia.org/wiki/Ariane_5_Flight_501
  - Conversión sin protección (sin protección por motivos de eficiencia) de float (64 bits) a entero con signo (16 bits)
- Misil Patriot, donde se guarda un factor de 1/10 en 24 bits para convertir de décimas-de-segundos a segundos, un error de .3 seg = 500 metros.
  - Ver http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html

Casos **graves** de errores de redondeo

- Ustedes tienen que tener cuidado con errores pequeños.
  - A veces replantear algebraicamente las ecuaciones evita restas de número casi iguales y divisiones y/o multiplicaciones de ellas.
  - Tarea: Reproducir el ejemplo 5 capítulo 1, sobre calculo de raíces de una ecuación de 2º grado en página 24 (ver archivo de tarea).