



Universidad de Chile

Departamento de Ingeniería Matemática

$$B_1 A_1 = E_1 A_1 + \rho_1 \sum_j B_j A_j F_{j1} \quad \nabla \times \vec{E} = - \frac{\partial \vec{B}}{\partial t} \quad \vec{F} = m \vec{E} + \frac{dm}{dt} \vec{v}$$

$$dU = \left( \frac{\partial U}{\partial S} \right)_V dS + \left( \frac{\partial U}{\partial V} \right)_S dV \quad \rho = \sum_j \rho_j e^{-E_j/kT}$$

$$F_j = \frac{\sum_{k=1}^{K_1} \frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t} \quad \nabla \times \vec{H} = \frac{\partial \vec{D}}{\partial t} + \vec{J} \quad \sum_i W_i B_i(t) P_i}{P_{n+1} = r P_n (1 - P_n)} \quad \sum_i W_i B_i(t)$$

$$0011 \ 0010 \ - \frac{h^2}{8m} \nabla^2 \psi(x,t) + V \psi(x,t) = - \frac{h}{2\pi i} \frac{\partial \psi(x,t)}{\partial t} \quad -\nabla^2 u + \lambda u = f$$

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla) \vec{u} = - \frac{1}{\rho} \nabla p + \nu \nabla^2 \vec{u} + \frac{1}{\rho} \vec{F} \quad \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = f$$

- NEWTON'S EQUATIONS • SCHRÖDINGER EQUATION (TIME-DEPENDENT) • NAVIER-STOKES EQUATION •
- POISSON EQUATION • HEAT EQUATION • HELMHOLTZ EQUATION • DISCRETE FOURIER TRANSFORM •
- MAXWELL'S EQUATIONS • PARTITION FUNCTION • POPULATION DYNAMICS •
- COMBINED 1ST AND 2ND LAWS OF THERMODYNAMICS • RADIOBITY • RATIONAL B-SPLINE •








Cálculo Numérico MA-33A

Gonzalo Hernández Oliva

# Representación Numérica y Errores

- 1) Motivación
- 2) Representación Numérica en el Computador
- 3) Tipos y Fuentes de Error
- 4) Representación Numérica y Error
- 5) Aritmética en Punto Flotante:  
Propagación de Errores en Operaciones Matemáticas
- 6) Bibliografía

# 1) Motivación 1: Números Egipcios !

Números egipcios antiguos						
1	10	100	1,000	10,000	100,000	1'000,000
						
Raya	Hueso del talón	Cuerda enrollada	Flor de loto	Dedo señalando	Pez	Hombre sorprendido

## 1) Motivación 2:

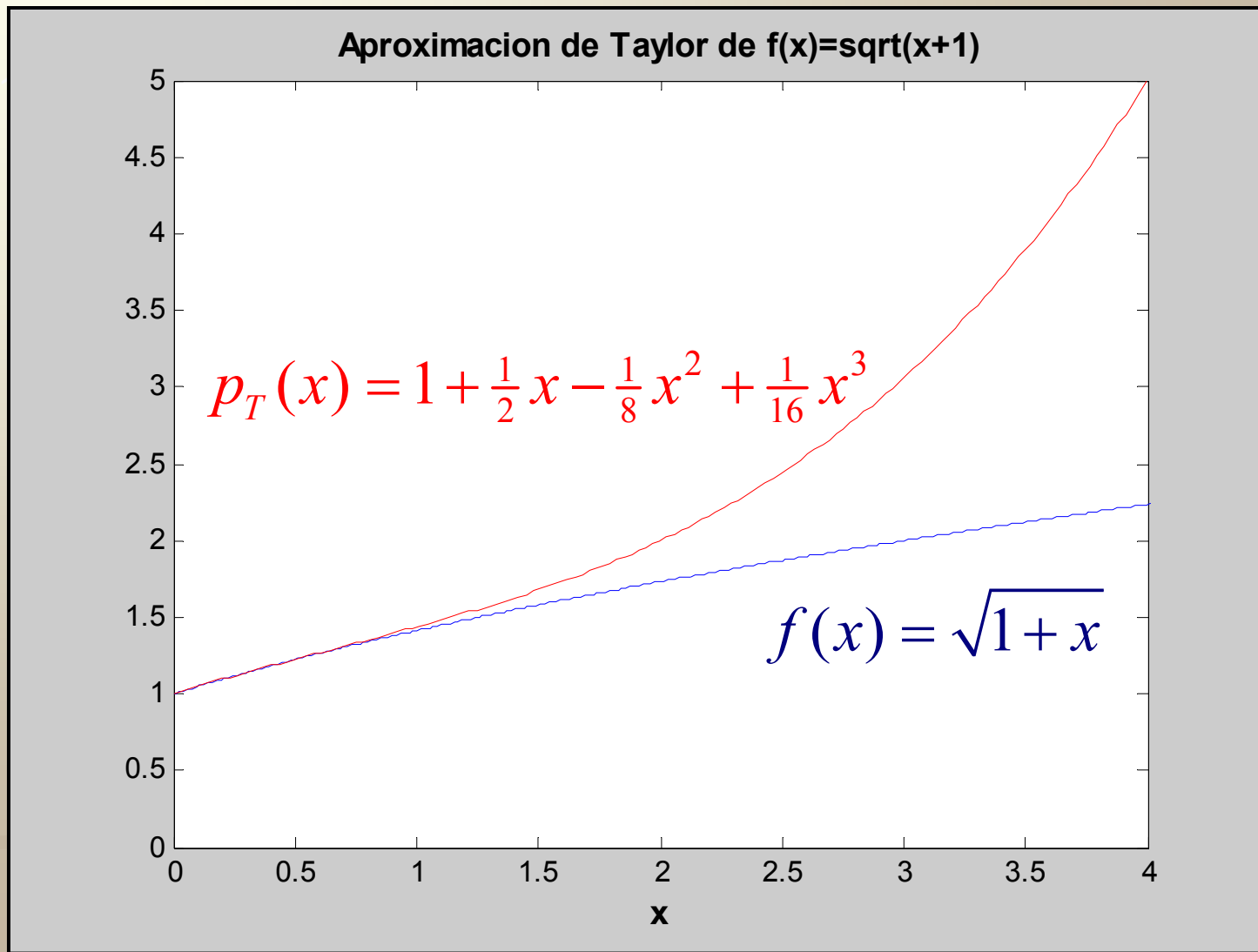
- Utilice el polinomio de Taylor de grado 3 de la función:

$$f(x) = \sqrt{1+x}$$

en torno a  $x_0 = 0$  para calcular las siguientes raíces cuadradas de  $x = 2, 2.5, 3.0$  con una aritmética de redondeo 4 decimales.

- Calcule los errores cometidos al realizar esta aproximación

# 1) Motivación 2:



# 1) Motivación 3:

- Utilice el polinomio de Taylor de grado 4 y 8 de la función:

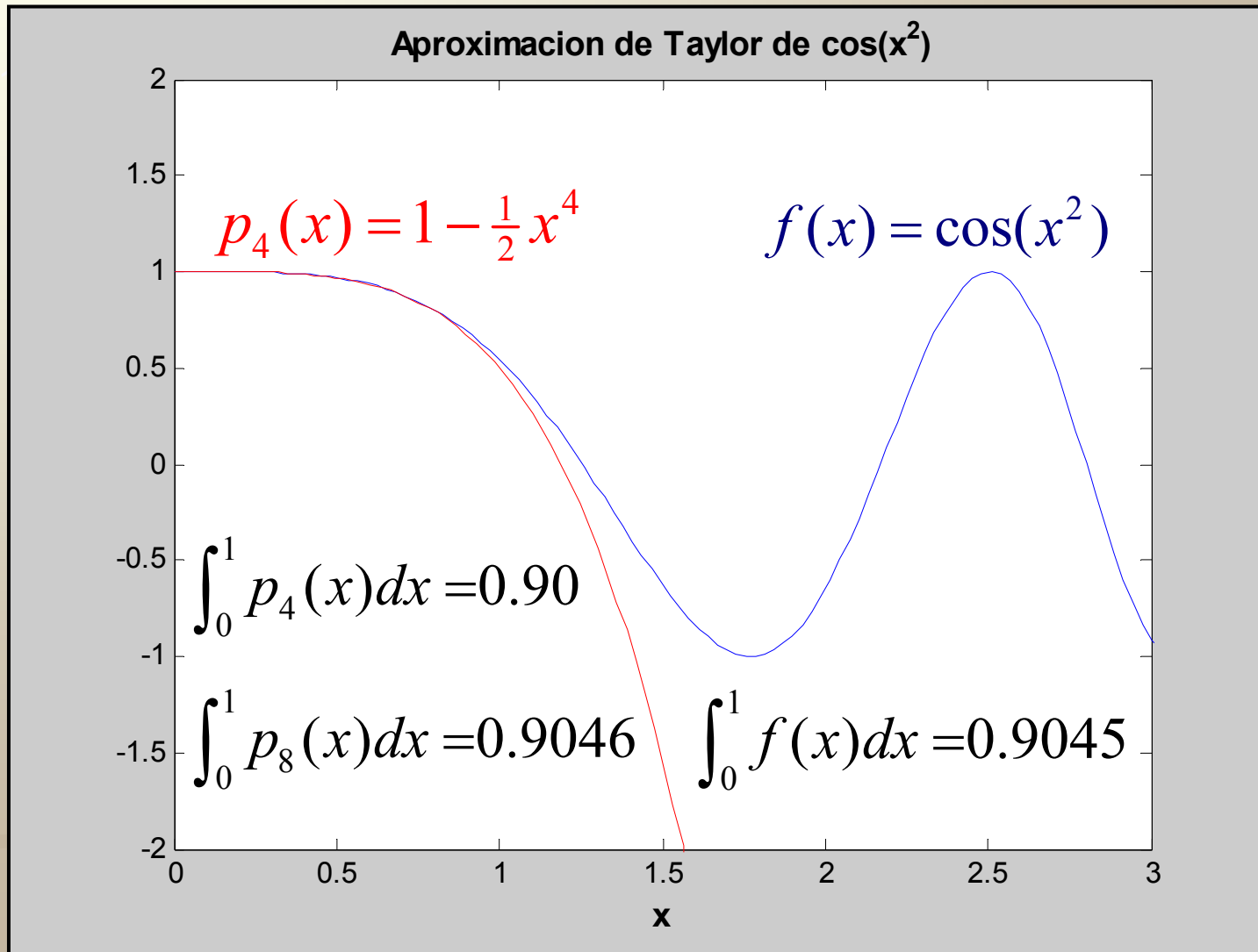
$$f(x) = \cos(x^2)$$

en torno a  $x_0 = 0$  para calcular:

$$I = \int_0^1 \cos(x^2) dx$$

Calcule los errores cometidos al realizar esta aproximación

# 1) Motivación 3:



## 1) Motivación 4:

- Al resolver la siguiente ecuación cuadrática:

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

Sus raíces exactas son aproximadamente:

$$x_1 = -0.01610723740897, x_2 = -62.08389276259103$$

Calcular las raíces con aritmética de 2 decimales:

$$x_1 = -0.02, x_2 = -62.09$$

Se puede mejorar la exactitud de estas raíces ???



## 2) Representación Numérica en el Computador

- No es posible representar todos los números en la memoria finita de un computador.
- Por lo tanto los cálculos computacionales se realizan con representaciones aproximadas de los números verdaderos (representación de punto flotante)
- Como se usa una pequeña proporción de números del sistema de los reales para representar a todo el sistema  $\Rightarrow$  se genera error en los cálculos

## 2) Codificación de Números Enteros

- Un número entero  $p$  se representa a través de su codificación binaria, decimal u octal

$$p = \sum_{k=n}^0 \alpha_k b^k \quad \text{donde } \alpha_k \in \{0, \dots, b-1\} \quad b = 2, 8, 10$$

- En general  $n = 30$  (palabra de 32 bits o 4 bytes para el tipo de dato int), pues se guarda un bit para el signo del entero. Por lo tanto, el intervalo de enteros representables es:  $[-(b^{31} - 1), (b^{31} - 1)]$

**Underflow**

**Overflow**

## 2) Codificación Binaria de Números Enteros

- Un número entero  $p$  se representa a través de su codificación binaria por:

$$p = \sum_{k=n}^0 \alpha_k 2^k \quad \text{donde } \alpha_k \in \{0,1\}$$

- En general  $n = 30$  (palabras de 32 bits o 4 bytes para el tipo de dato int), pues se guarda un bit para el signo del entero. Por lo tanto, el intervalo de enteros representables binariamente con 32 bits es:

$$[-(2^{31}-1), (2^{31}-1)] = [-2147483647, 2147483647]$$

Underflow

Overflow

## 2) Cod. Binaria de Números Enteros: Ejemplo

Bit	Resto	Potencias de 2	Alfa	Resto
16	4605	65536	0	4605
15	4605	32768	0	4605
14	4605	16384	0	4605
13	4605	8192	0	4605
12	4605	4096	1	509
11	509	2048	0	509
10	509	1024	0	509
9	509	512	0	509
8	509	256	1	253
7	253	128	1	125
6	125	64	1	61
5	61	32	1	29
4	29	16	1	13
3	13	8	1	5
2	5	4	1	1
1	1	2	0	1
0	1	1	1	0

## 2) Codificación de Números Enteros

Ejercicios: 1101 0001 0100 1011

- a) Determinar un algoritmo para obtener la codificación binaria de un número natural (unsigned)
- b) Determinar un algoritmo para obtener la codificación en una base distinta a la binaria ( $b = 3, 5, 8$ ) de un número entero
- c) Calcular el número de ops de ambos algoritmos

## 2) Codificación de Números Reales en $[0,1]$

- Todo número real  $x \in [0,1]$  se representa en base a su codificación diádica (“binaria – negativa”)

$$x = \sum_{k=1}^n \alpha_{-k} 2^{-k} \quad \text{donde } \alpha_{-k} \in \{0,1\}$$

- Para obtener la representación floating point de números reales primero se debe convertir a un número en  $[0,1]$  (normalización) tal que su mantisa sea mayor que 0.1

## 2) Codificación de Números Reales en [0,1]

Bit	Mantisa	Potencias de 0.5	Alfa	Resto
1	0.6992797852000000	0.5000000000000000	1	0.1992797852000000
2	0.1992797852000000	0.2500000000000000	0	0.1992797852000000
3	0.1992797852000000	0.1250000000000000	1	0.0742797852000000
4	0.0742797852000000	0.0625000000000000	1	0.0117797852000000
5	0.0117797852000000	0.0312500000000000	0	0.0117797852000000
6	0.0117797852000000	0.0156250000000000	0	0.0117797852000000
7	0.0117797852000000	0.0078125000000000	1	0.0039672852000000
8	0.0039672852000000	0.0039062500000000	1	0.0000610352000000
9	0.0000610352000000	0.0019531250000000	0	0.0000610352000000
10	0.0000610352000000	0.0009765625000000	0	0.0000610352000000
11	0.0000610352000000	0.0004882812500000	0	0.0000610352000000
12	0.0000610352000000	0.0002441406250000	0	0.0000610352000000
13	0.0000610352000000	0.0001220703125000	0	0.0000610352000000
14	0.0000610352000000	0.0000610351562500	1	0.0000000000437500
15	0.0000000000437500	0.0000305175781250	0	0.0000000000437500
16	0.0000000000437500	0.0000152587890625	0	0.0000000000437500

## 2) Codificación de Números Reales

### (Primer Floating Point)

0011 0010 1010 1101 0001 0100 1011

En general, todo número real se puede escribir como:

$$x = \pm m \cdot b^{\pm z}$$

donde

b : base es un número natural

m : mantisa es un número real en  $[0,1]$

z : exponente es un número entero



## 2) Codificación FP de Números Reales

En un computador la mantisa debe tener un número finito de dígitos. Esto genera la representación floating point (fp) de números reales:

$$x = \pm 0.m_1m_2m_3\dots m_t \cdot b^{\pm z} \quad 0 \leq m_k \leq b - 1$$

En computadores inicialmente se utilizó:

**b = 2** (base binaria)

**t = 23 o 52** (single - double precisión)

**z = 7 o 11 bits** (número entero)

## 2) Codificación FP de Números Reales:

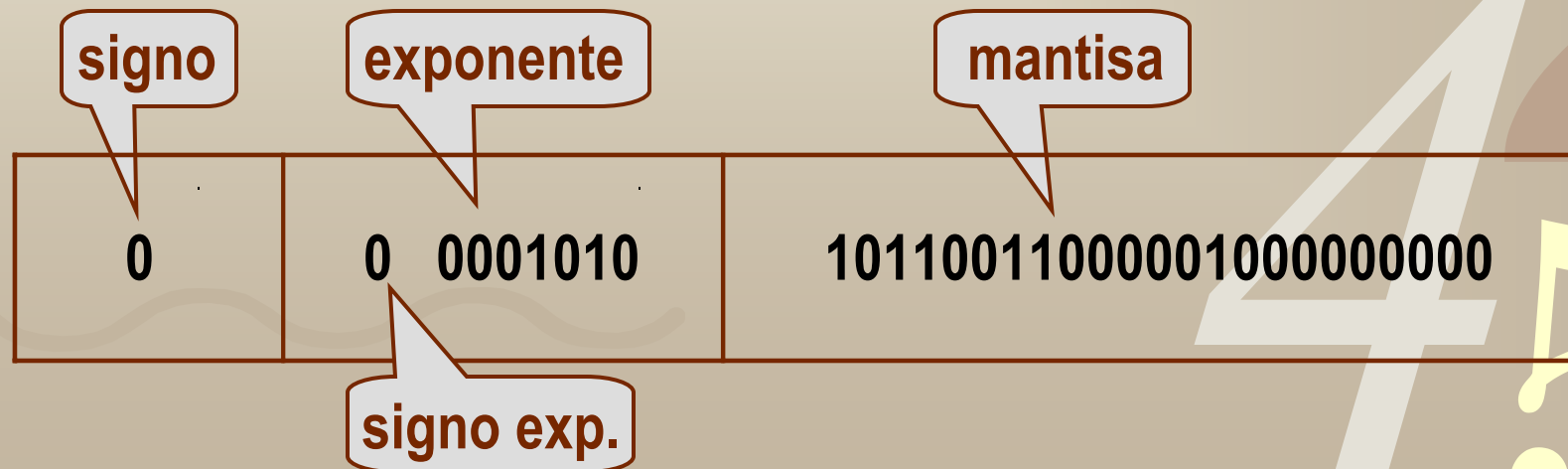
Un ejemplo: Float de 32 bit

Considerando los siguientes parámetros:

**Signo ( + o - ) :** 1 dígito binario ( 1 bit )

**Exponente:** base  $b = 2$  de 7 bit más bit signo exp.

**Mantisa:** 23 bit



## 2) Codificación FP de Números Reales:

Un ejemplo

<b>Signo</b> 0	positivo (+)
<b>Exponente</b> 0 0001010	$0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = +10$
<b>Mantisa</b> 101100110000010000000000	$1 \times \left(\frac{1}{2}\right)^1 + 1 \times \left(\frac{1}{2}\right)^3 + 1 \times \left(\frac{1}{2}\right)^4 + 1 \times \left(\frac{1}{2}\right)^7 + 1 \times \left(\frac{1}{2}\right)^8 + 1 \times \left(\frac{1}{2}\right)^{14}$

**Número Representado = + 0.699279785 x 2<sup>10</sup> = 716.0625000448**

## 2) Codificación FP de Números Reales

Algoritmo para obtener la codificación de un real  $x$ :

1) Determinar la notación científica de  $x$  con mantisa normalizada:  $x = \pm m \cdot 10^{\pm z}$

2) Cambiar base 10 a base 2 dividiendo por potencia de 2 mayor más cercana a la potencia de 10:

$$x = \pm m \cdot \left( 10^{\pm z} / 2^{\pm \lceil \log_2(10^z) \rceil} \right) 2^{\pm \lceil \log_2(10^z) \rceil}$$

3) Recalcular mantisa:  $m^* = m \cdot 10^{\pm z} / 2^{\lceil \log_2(10^z) \rceil}$

4) Determinar codificación binaria exponente  $\lceil \log_2(10^z) \rceil$  y codificación diádica mantisa  $m^*$

## 2) Codificación FP de Números Reales

### Ejemplos

Cuál es el número representable en el computador mayor más cercano al de nuestro ejemplo ?

716.0625000448		
0	0 0001010	101100110000010000000000

716.0626221056		
0	0 0001010	101100110000010000000001



## 2) Codificación FP de Números Reales

### Ejercicios

0011 0010 1010 1101 0001 0100 1011

- Obtener codificación FP de:  $10/3$ ,  $\sqrt{2}$ , etc
- Cuál es el intervalo representado por un real en la codificación floating point ?
- Determinar el número real máximo y mínimo representado por la codificación inicial floating point
- Calcular el número de ops del algoritmo codificación números reales

## 2) Codificación de Números Reales (1985)

- En los computadores actuales se utiliza la siguiente representación floating point (fl) de números reales (IEEE BFLAS 754-1985)

$$x = (-1)^s 2^{(z-c)} (1 + m)$$

- Donde:

s:	Signo del real (1 bit)
c:	Corrección del exponente (Entero)
z:	Exponente (Entero Binario)
m:	Mantisa (Digito real $\varepsilon [0,1]$ )

## 2) Codificación de Números Reales (1985)

- La normativa IEEE BFLAS 754-1985 define las siguientes cantidades:

s: 1 bit (0 = fl positivo , 1 = fl negativo)

c: 1023

z: 11 bit  $\Rightarrow$  Intervalo Exponente: 0 a  $(2^{11} - 1) = 2047$   
 $\Rightarrow (z - c)$  de -1023 a 1024

m: 52 bit  $\Rightarrow$  16 a 17 dígitos decimales  
(precisión double)



## 2) Codificación de Números Reales:

Un ejemplo de la codificación actual

<b>Signo</b>	0	positivo (+)
<b>Exponente</b>	10000000011	$1 \times 2^{10} + 0 \times 2^9 + 0 \times 2^8 + \dots + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 1027$
<b>Mantisa</b>	1011100100010000 ... 00000	$1 \times \left(\frac{1}{2}\right)^1 + 1 \times \left(\frac{1}{2}\right)^3 + 1 \times \left(\frac{1}{2}\right)^4 + 1 \times \left(\frac{1}{2}\right)^5 + 1 \times \left(\frac{1}{2}\right)^8 + 1 \times \left(\frac{1}{2}\right)^{12}$

$$\begin{aligned}
 x &= (-1)^s 2^{(z - c)} (1 + m) = (-1)^0 2^{(1027 - 1023)} (1 + 0.7229003) \\
 &= 2^4 (1.7229003) = 27.566404
 \end{aligned}$$

## 2) Codificación de Números Reales: Límites Actuales de FL

0011 0010 1010 1101 0001 0100 1011

- Determinar un algoritmo para obtener la codificación fl actual de un número real
- Cuál es el número menor y mayor más cercano al real de nuestro ejemplo, según la codificación fl actual ?
- Cuál es el intervalo representado por el real del ejemplo en la codificación floating point actual ?
- Determinar el número real máximo y mínimo representado por la codificación actual floating point

### 3) Tipos y Fuentes de Error

Existen los siguientes tipos de error:

- 1) Errores en los datos de input: Debidos a imperfección en las mediciones físicas (por ejemplo)
- 2) Errores de redondeo: Debidos a la representación finita (y por lo tanto inexacta) de números reales. Ello origina errores de redondeo y truncación.
- 3) Errores de aproximación: Debidos a la utilización de métodos con error inexactos para la solución de ecuaciones matemáticas. En esta clase encontramos los errores de truncación y discretización

### 3) Tipos de Error: Absoluto y Relativo

- Si un número real  $x$  tiene la representación en el computador de floating point  $fl(x)$ , se definen los siguientes tipos de error:

Error Absoluto:

$$E_{abs}(x, fl(x)) = |x - fl(x)|$$

Error Relativo:

$$E_{rel}(x, fl(x)) = \frac{|x - fl(x)|}{|x|}$$

## 4) Representación Numérica y Error

- El conjunto de números reales representables en el computador  $\mathcal{R}_c$  es finito
- Una pregunta importante es cómo aproximamos un número  $x \notin \mathcal{R}_c$  por un número  $y \in \mathcal{R}_c$
- Este problema se encuentra no sólo al introducir datos de un problema a resolver en un computador sino también al representar el resultado de operaciones aritméticas (por ejemplo).
- El resultado de operar aritméticamente 2 números reales  $u, v \in \mathcal{R}_c$  no necesariamente está en  $\mathcal{R}_c$

## 4) Representación Numérica y Error

- Es natural postular que la aproximación de cualquier número  $x \notin \mathcal{R}_C$  por un número  $fl(x) \in \mathcal{R}_C$  debe satisfacer:

$$|x - fl(x)| \leq |x - y| \quad \forall y \in \mathcal{R}_C$$

- Se puede obtener  $fl(x)$  en un computador de t-dígitos de la siguiente forma:
  - a) Suponiendo que  $x$  está representado en forma normalizada decimal:

$$x = (-1)^{\text{sgn}(x)} m \cdot 10^z$$

donde se tiene que:

## 4) Representación Numérica y Error

a)  $m = 0.\alpha_1 \alpha_2 \alpha_3 \alpha_4 \dots \alpha_i \dots \quad 0 \leq \alpha_i \leq 9$   
 $\alpha_1 \geq 0.1$

b) Entonces se forma por redondeo:

$$m' = \begin{cases} 0.\alpha_1 \alpha_2 \dots \alpha_t & \text{si } \alpha_{t+1} \leq 4 \\ 0.\alpha_1 \alpha_2 \dots \alpha_t + 10^{-t} & \text{si } \alpha_{t+1} \geq 5 \end{cases}$$

Por ejemplo si  $t = 10$  y:

$$m = 0.3267897657\textcircled{3}2 \dots \Rightarrow m' = 0.3267897657$$

$$m = 0.3267897657\textcircled{6}2 \dots \Rightarrow m' = 0.3267897658$$

## 4) Representación Numérica y Error

c) Finalmente:

$$fl(x) = (-1)^{\text{sgn}(x)} m' \cdot 10^z$$

- En esta representación se redondea el número original. Se puede demostrar que:

$$fl(x) = x(1 + \delta)$$

Donde  $\delta$  puede ser cero, negativo o positivo.

- $\delta$  es el error relativo cometido al representar  $x$  en el computador. Se tiene que:  $|\delta| \leq 5 \times 10^{-t}$
- $\text{eps} = 5 \times 10^{-t}$  es la precisión de máquina



## 4) Representación Numérica y Error

- El proceso de aproximación anterior para obtener la representación floating point de un número  $x \notin \mathcal{R}_c$  funciona en la mayoría de los casos.
- Sin embargo hay casos en que este proceso no obtiene un número de máquina, es decir  $x \notin \mathcal{R}_c$  y  $\text{fl}(x) \notin \mathcal{R}_c$
- Estos casos ocurren con exponentes  $z$  cerca del overflow y underflow. En estas situaciones se obtiene el  $\text{fl}(x)$  aproximando al mayor o menor número de máquina representado según corresponda.

## 5) Aritmética en FP: Propagación de Errores

- Sean  $x$ ,  $y$  números con representación de igual precisión, luego si se operan por una operación  $\otimes$ :

$$fl(x \otimes y) = (x \otimes y)(1 + \varepsilon) \quad \text{para } \otimes = +', -', *, /'$$

son las operaciones aritméticas en el computador

- Es decir:

$$x +' y = fl(x + y) = (x + y)(1 + \varepsilon_s)$$

$$x -' y = fl(x - y) = (x - y)(1 + \varepsilon_r)$$

$$x *' y = fl(x * y) = (x * y)(1 + \varepsilon_m)$$

$$x /' y = fl(x / y) = (x / y)(1 + \varepsilon_d)$$

$\varepsilon_s, \varepsilon_r, \varepsilon_m, \varepsilon_d$   
Son los errores  
relativos de cada  
operación

## 5) Aritmética en FP: Propagación de Errores

- Debido a los errores incurridos en la representación de los números y las operaciones aritméticas en el computador las leyes usuales de la matemática dejan de ser válidas.
- Veamos un ejemplo: Asociatividad de la suma para:

$$x = 0.23371258 \times 10^{-4}$$

$$y = 0.33678429 \times 10^2$$

$$z = -0.33677811 \times 10^2$$

Considerando  $t = 8$   
dígitos significativos

El resultado exacto de la suma de estos 3 números es:  $x + y + z = 0.641371258 \times 10^{-3}$  pero ...

## 5) Aritmética en FP: Propagación de Errores

$$x +' (y +' z) = 0.64137126 \times 10^{-3}$$

$$(x +' y) +' z = 0.64100000 \times 10^{-3}$$

- En general al operar aritméticamente 2 números reales representados en el computador según la codificación floating point se produce una propagación de errores que se puede aproximar por las relaciones:

Si  $fl(x) = x(1 + \varepsilon_x)$  ,  $fl(y) = y(1 + \varepsilon_y)$  entonces

$$x \pm' y = fl(x \pm y) = (x \pm y)(1 + \varepsilon_{sr})$$

$$x *' y = fl(x * y) = (x * y)(1 + \varepsilon_m)$$

$$x /' y = fl(x / y) = (x / y)(1 + \varepsilon_d)$$

## 5) Aritmética en FP: Propagación de Errores

- Error de la Suma y Resta  $\varepsilon_{sr}$  verifica:

$$\varepsilon_{sr} = \frac{x}{(x \pm y)} \varepsilon_x \pm \frac{y}{(x \pm y)} \varepsilon_y$$

- Error de la Multiplicación  $\varepsilon_m$  verifica:

$$\varepsilon_m = \varepsilon_x + \varepsilon_y$$

- Error de la División  $\varepsilon_d$  verifica:

$$\varepsilon_d = \varepsilon_x - \varepsilon_y$$

- Estas relaciones (obtenidas por aproximación de primer orden de las operaciones) indican que las operaciones que más generan error es la suma y resta

## 5) Aritmética en FP: Propagación de Errores

### ■ Propagación Grave de Errores:

- ✗ Suma (resta) de números de distinto signo
- ✗ Suma de números distintos en magnitud
- ✗ Resta números similares
- ✗ Overflow y Underflow
- ✗ División por un número pequeño

- Es posible realizar un análisis de propagación de errores de primer orden para cualquier operación matemática.

## 5) Aritmética en FP: Propagación de Errores

- Si  $z = \phi(x,y)$  representa una secuencia de operaciones matemáticas entre 2 números reales
- El error absoluto de la operación  $\phi$  se puede calcular aproximadamente en función de los errores absolutos de representación de los números  $x$  e  $y$

$$\Delta\Phi(x, y) \cong \frac{\partial\Phi}{\partial x} \Delta x + \frac{\partial\Phi}{\partial y} \Delta y$$

Aproximación de Taylor de primer orden

- Análogamente se puede calcular el error relativo de la operación  $\phi$

## 5) Aritmética en FP: Propagación de Errores

- El error relativo de la operación  $\phi$ ,  $\varepsilon_{\Phi}$ , se puede calcular aproximadamente en función de los errores relativos de la representación fl de los números  $x$  e  $y$

$$\varepsilon_{\Phi} = \frac{x}{\Phi(x, y)} \frac{\partial \Phi}{\partial x} \varepsilon_x + \frac{y}{\Phi(x, y)} \frac{\partial \Phi}{\partial y} \varepsilon_y$$

Aproximación de Taylor de 1<sup>er</sup> orden

- Ejemplos simples:

$\phi(x, y) = xy$  (fórmulas para error relativo de ops. aritm.)

$\phi(x) = \sqrt{x}$

$\phi(x, y) = x^2 - y^2$



## 5) Aritmética en FP: Propagación de Errores

- Ejercicios de propagación de errores:

a)  $\Phi(x, y) = x \pm y, xy, \frac{x}{y}$

b)  $\Phi(x, y) = x^2 y, \sqrt{xy}, \frac{x^2}{y}, \frac{x}{y^2}, \frac{x-y}{x+y}$

c)  $\Phi(x, y) = \sin(x^2 + y^2), \log(x^2 + y^2), xe^y, e^{-x^2}$

d)  $\Phi(x, y, z) = \frac{-y \pm \sqrt{y^2 - 4xz}}{2x}$

- Para todos los ejemplos anteriores determine las condiciones (si existen) bajo las cuales el error de la operación  $\phi$  crece linealmente

## 5) Aritmética en FP: Propagación de Errores

- Para una operación  $\phi(x,y)$  las cantidades  $\mu_x$  y  $\mu_y$  definidas según:

$$\mu_x = \frac{x}{\Phi(x,y)} \frac{\partial \Phi}{\partial x}$$
$$\mu_y = \frac{y}{\Phi(x,y)} \frac{\partial \Phi}{\partial y}$$

Se denominan números de condicionamiento del error relativo e indican en que forma el error relativo en x e y afecta al error relativo de  $\phi(x,y)$

## 5) Propagación de Errores: Estabilidad

- Si los números de condicionamiento  $\mu_x$  y  $\mu_y$  se pueden acotar por constantes independientes de  $x$  e  $y$ , diremos entonces que la secuencia de operaciones matemáticas es estable. En otro caso diremos que es inestable.
- Ejemplos de algoritmos estables son:  
 $\phi(x,y) = xy$  ,  $\phi(x,y) = x / y$  ,  $\phi(x) = \sqrt{x}$
- Ejemplos de algoritmos inestables son:  
 $\phi(x,y) = e^{xy}$  ,  $\phi(x,y) = x^2 - y^2$  ,  $\phi(x,y) = \ln(x^2 + y^2)$

## 5) Propagación de Errores en Algoritmos

- Si una operación matemática se puede realizar mediante más de un algoritmo (secuencia de instrucciones) es de utilidad determinar la propagación de errores de cada algoritmo. Por ej.:

$$\varphi(x, y) = x^2 - y^2$$

$$\text{Alg 1: } \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x+y \\ x-y \end{bmatrix} \rightarrow (x+y)(x-y) = x^2 - y^2$$

$$\text{Alg 2: } \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x^2 \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x^2 \\ y^2 \end{bmatrix} \rightarrow x^2 - y^2$$

## 5) Propagación de Errores en Algoritmos

- Consideremos una operación matemática  $\varphi$  que se calcula mediante una sucesión de operaciones elementales:  $\varphi^{(r)}, \varphi^{(r-1)}, \dots, \varphi^{(1)}, \varphi^{(0)}$

$$z = \varphi(x) = \varphi^{(r)} \circ \varphi^{(r-1)} \circ \dots \circ \varphi^{(1)} \circ \varphi^{(0)}(x)$$

$$x^{(k+1)} = \varphi^{(k)}(x^{(k)}) \quad \forall k = 0, 1, \dots, r \quad \text{con } x^{(0)} = x$$

- El error aportado en cada etapa del algoritmo es:

$$\begin{aligned} \Delta x^{(k+1)} &= \tilde{x}^{(k+1)} - x^{(k+1)} = fl(\varphi(\tilde{x}^{(k)})) - \varphi(x^{(k)}) \\ &= [fl(\varphi^{(k)}(\tilde{x}^{(k)})) - \varphi^{(k)}(\tilde{x}^{(k)})] + [\varphi^{(k)}(\tilde{x}^{(k)}) - \varphi^{(k)}(x^{(k)})] \end{aligned}$$

## 5) Propagación de Errores en Algoritmos

- Aproximamos el segundo sumando por Taylor:

$$\varphi^{(k)}(\tilde{x}^{(k)}) - \varphi^{(k)}(x^{(k)}) = D\varphi^{(k)}(x^{(k)})(\tilde{x}^{(k)} - x^{(k)})$$

- Para el primer sumando, se tiene que:

$$fl(\varphi^{(k)}(\tilde{x}^{(k)})) = (I + E^{(k+1)})\varphi^{(k)}(\tilde{x}^{(k)})$$

- Luego:

$$\Delta x^{(k+1)} = E^{(k+1)}\varphi^{(k)}(\tilde{x}^{(k)}) + D\varphi^{(k)}(x^{(k)})\Delta x^{(k)}$$

- Aproximamos la matriz  $E^{(k+1)}$  por una diagonal.

## 5) Propagación de Errores en Algoritmos

- Y aproximamos también:

$$E^{(k+1)} \varphi^{(k)}(\tilde{x}^{(k)}) = E^{(k+1)} x^{(k+1)} = \alpha_{k+1}$$

- Luego:

$$\Delta x^{(k+1)} = \alpha_{k+1} + D\varphi^{(k)}(x^{(k)})\Delta x^{(k)}$$

- El error acumulado en todo el algoritmo es finalmente:

$$\Delta z = \Delta x^{(r+1)} \approx D\varphi^{(r)} \cdot \dots \cdot D\varphi^{(0)} \Delta x + D\varphi^{(r)} \cdot \dots \cdot D\varphi^{(1)} \alpha_1 + \\ D\varphi^{(r)} \cdot \dots \cdot D\varphi^{(2)} \alpha_2 + \dots + \alpha_{k+1}$$

- Veamos un ejemplo

## 6) Bibliografía

- 1) R. Burden & J. D. Faires, Análisis Numérico, Séptima Edición, Thomson Learning, 2002
- 2) J. Stoer & R. Burlisch, Introduction to Numerical Analysis, Second Edition, Springer, 1992.
- 3) G. Hernández O.: Apuntes de Cálculo Numérico