Teoría de Errores y Aritmética del Computador

1.1 Introducción al Cálculo Numérico

El extendido uso del computador digital ha tenido un profundo efecto en la Ingeniería y Ciencia, esto ha traído muchos beneficios. Con computadoras y software apropiado podemos modelar y analizar complejos sistemas y problemas físicos. Sin embargo, el eficiente y preciso uso de los resultados numéricos obtenidos de programas de computador requiere considerable experiencia y avanzados conocimientos para evitar desatinos. En este curso intentamos proveer de algo de la experiencia y conocimientos necesarios para evitar tales dificultades.

En particular, se consideran varios de los más comúnmente usados métodos de aproximación empleados en la solución de problemas físicos. Una realista y exitosa solución de un problema de ingeniería, empieza con un preciso modelo físico del problema y una apropiada comprensión de las suposiciones empleadas. Luego este modelo es transformado en modelo o problema matemático. La solución del problema matemático es usualmente obtenida por métodos numéricos que por definición son aproximados. Excepto para casos realmente simples, el exitoso uso del método numérico depende considerablemente del uso del computador digital, el uso de microcomputadoras o supercomputadoras dependerá de la complejidad del problema.

En el proceso de resolución de problemas, es posible distinguir varias fases distintas. La primera fase es la formulación. Al formular un modelo matemático de una situación física, el ingeniero debe tomar en cuenta que espera resolver su problema en un computador. Por consiguiente determinará objetivos específicos, datos adecuados y verificados y tendrá en cuenta el tipo de resultados.

Una vez formulado un problema, deben diseñarse métodos numéricos, juntos con un análisis preliminar del error, para resolver el problema. Un método numérico que puede usarse para resolver un problema se llama algoritmo. Un algoritmo es un conjunto de procedimientos completo y carente de ambigüedad, que lleva a la solución de un problema matemático. La solución o construcción de algoritmos apropiados cae dentro del terreno del análisis numérico. Ya decidido el algoritmo o conjunto de algoritmos para resolver el problema, el analista debe considerar todas las fuentes de error que pueden afectar los resultados. Debe considerarse cuanta exactitud se requiere, estimar la magnitud del redondeo, y errores de discretización, determinar el número apropiado de iteraciones que se necesitan, proporcionar medios adecuados para verificar la exactitud y dejar campo para la acción correctiva.

La tercera fase de la solución de un problema es la programación. El programador debe transformar el algoritmo sugerido en un conjunto de instrucciones detallado y sin ambigüedad.

1.2 Teoría de Errores

En este capítulo citaremos las diferentes fuentes básicas de los errores, y las nociones importantes de la teoría de errores.

1.2.1 Fuentes básicas de los errores

Casi siempre la medida de las magnitudes al igual que las soluciones numéricas a problemas reales, contienen errores que tienen diferentes orígenes, que las podemos resumir en:

a) Error del modelo o error del problema
En los fenómenos de la naturaleza muchas veces efectuamos ciertas hipótesis, es decir aceptamos determinadas condiciones que nos dará una situación aproximada del fenómeno estudiado, de esta manera podemos plantear el comportamiento de dicho fenómeno por medio de un modelo matemático.

Por ejemplo en la caída libre de un cuerpo, si despreciamos la resistencia del aire la distancia "h" recorrida después de un tiempo "t" está modelado por la fórmula:
$h = \frac{1}{2} g\, t^2$ donde ***g*** es la gravedad

b) Error del método
Cuando un problema planteado en forma precisa no puede resolverse en forma exacta o es muy difícil de hallar la solución, se fórmula una aproximación del modelo, que ofrezca prácticamente los mismo resultados (método).

La vibración libre de una membrana elástica circular está modelada por la ecuación:
$$\frac{1}{r}\frac{\partial}{\partial r}\left[r\frac{\partial u}{\partial r}\right] + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} = \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2}$$
$u = u(r,\theta,t)$ satisface cierta condición de contorno.
Está ecuación puede aproximarse de manera adecuada (por ejemplo usando diferencias finitas) y de esta forma conocer una solución aproximada.

c) Error residual
Son los originados por las series infinitas, al considerar solo una parte finita.
Por ejemplo:
$e = 2 + 1/2! + 1/3! + \ldots$
$e = 2 + 1/2! + 1/3! + \ldots 1/n!$ para cierto valor ***n***.

d) Error inicial
Son los originados por los parámetros cuyos valores son conocidos aproximadamente:
Por ejemplo: La constante de Planck
$h = (6.62377 \pm 0.00027)10^{-27} \quad erg - seg$.

e) Errores de redondeo

Originados por la representación finita de los números, es el caso de las computadoras (notación de punto flotante).

Por ejemplo: $1/6 = 0.1666\cdots$ se redondea en un número finito de dígitos.

f) Error de operación

Podemos distinguir dos casos:

- Operación (exacta) de números aproximados:
  Por ejemplo: $\pi - 1/6 \approx 3.1416 - 0.1667 = 3.2083$.
- Operación (aproximada) de números exactos:
  $0.23x10^{8} - 1.36x10^{-8} \approx 0.23x10^{8}$, el valor verdadero es:
  22999999.9999999864

En la medición de magnitudes, sea directamente en un experimento o indirectamente mediante una relación (fórmula) aplicada a datos experimentales, distinguimos dos tipos de errores: sistemático y casual.

g) Error sistemático

Son aquellos, que sin variar las condiciones del ensayo entran de igual modo en cada resultado de las mediciones, pueden ser originados por:

- Defecto del instrumento
- Las condiciones del ambiente
- La metodología de la medición
- Precisión limitada del instrumento
- Las particularidades del experimentador

h) Error Casual o Accidental (fortuito)

Son los que están vinculados con los factores que sufren pequeñas variaciones (aleatorias) durante el experimento:

1.2.2 La estabilidad del problema

Significa que pequeños cambios en los datos producen pequeños cambios en la solución exacta del problema inicial. De los problemas que no verifican esta propiedad, se dicen que están mal condicionados.

1.2.3 Error absoluto, relativo y precisión

Consideremos “A” el valor exacto de la medida de cierta magnitud (en general desconocida) y sea “a” un valor conocido que se llamará aproximación de “A”. Evidentemente la buena cualidad de la aproximación es de acuerdo a cuan próximo está “a” de “A”.

Definiciones:

a) Llamamos error absoluto del número aproximado "a" al valor:

$$\xi_a = |A - a|$$

y todo número $\xi_a^* \geq \xi_a$, se denominará cota del error absoluto.

b) Llamamos error relativo del número aproximado "a" al valor:

$$\delta_a = \frac{\xi_a}{|A|}, \quad A \neq 0$$

y todo número $\delta_a^* \geq \delta_a$, se denominará cota del error relativo.

c) Dado un $\varepsilon > 0$ (pequeño) decimos que el valor "a" aproxima a "A" con una precisión ε si:

$$\xi_a = |A - a| \leq \varepsilon$$

d) Sean A y "a" dos números reales. Se dice que "a" es una aproximación de A con "n" cifras decimales exactas (o que A y "a" coinciden en "n" cifras decimales), si "n" es el mayor entero no negativo tal que

$$\xi_a \leq 0.5x10^{-n}$$

Ejemplo:

Una aproximación de A = π = 3.141592… con tres cifras decimales exactas es $a = 3.1416$, ya que

$$|\pi - 3.1415| = 0.000092... < 0.5 \times 10^{-3} = 0.0005$$

$$|\pi - 3.1415| = 0.000092... > 0.5 \times 10^{-4} = 0.00005$$

Sin embargo, el número 3.141 coincide con π en sólo dos cifras decimales exactas, ya que

$$0.5 \times 10^{-3} = 0.0005 < |\pi - 3.141| = 0.000592... < 0.5 \times 10^{-2} = 0.005$$

e) Sean A y "a" dos números reales, con A≠0. Se dice que "a" es una aproximación de A con "n" cifras decimales significativas exactas (o que A y "a" coinciden en "n" cifras decimales significativas), si "n" es el mayor entero no negativo tal que

$$\delta_a \leq 5x10^{-n}.$$

Ejemplo:

El número 124.45 coincide con 123.45 en dos cifras significativas, ya que

$$5 \times 10^{-3} < \frac{|123.45 - 124.45|}{123.45} = \frac{1}{123.45} = 0.0081... < 5 \times 10^{-2}$$

El número 251.75 coincide con 250.75 en tres cifras significativas, ya que

$$5 \times 10^{-4} < \frac{|250.75 - 251.75|}{250.75} = \frac{1}{250.75} = 0.0039... < 5 \times 10^{-3}$$

f) Terminación por redondeo:
Representación decimal $\mathbf{0.\ a_1a_2a_3\ldots..a_t\ a_{t+1}}$ $\mathbf{0 \le a_i \le 9,\ \ a_1 \ne 0}$

Entonces se forma: $\mathbf{0.a_1a_2a_3\ldots..a_t}$ **si** $\mathbf{0 \le a_{t+1} \le 4}$

$\mathbf{0.a_1a_2a_3\ldots..a_t} + 10^{-t}$ **si** $\mathbf{a_{t+1} \ge 5.}$

g) Terminación por truncamiento:
Representación decimal $\mathbf{0.a_1\ a_2a_3\ldots..a_ka_{k+1}a_{k+2}\ldots...}$

Entonces para k dígitos de la mantisa se forma $\mathbf{0._1\ a_2a_3\ldots..a_{k-1}a_k.}$

1.2.4 Propagación del error de las funciones

Al resolver un problema utilizando métodos numéricos, en general el error será consecuencia de un cúmulo de errores ocurridos en pasos sucesivos, se debe estudiar la mecánica de "propagación" de los mismos a lo largo del cálculo.
Un mito común es que las computadoras modernas trabajan con tal grado de precisión que los usuarios no necesitan contemplar la posibilidad de resultados inexactos. Esto se ve reforzado cuando vemos en la pantalla los resultados con gran cantidad de cifras. Sin embargo, veremos a lo largo del curso que la falta de cuidado en cálculos aparentemente directos y triviales puede conducir a resultados catastróficos.

REGLAS DE PROPAGACIÓN
Los errores se propagan de acuerdo a las reglas que se presentan a continuación:

a) Suma: Propaga los errores *absolutos*

$$|\xi_{x+y}| \approx |\xi_x| + |\xi_y|$$

b) Producto y cociente (si δ_x y δ_y son pequeños): propaga los errores *relativos*

$$|\delta_{x.y}| \approx |\delta_x| + |\delta_y|$$
$$|\delta_{x/y}| \approx |\delta_x| + |\delta_y|$$

c) Funciones de una variable: $y = f(x)$

$$\xi_y \approx |y'|\xi_x$$

d) Funciones de varias variables: $y = f(x_1, x_2, \cdots, x_n)$

$$\xi_y \approx \sum_{i=1}^{n} \left|\frac{\partial f}{\partial x_i}\right| \xi_{x_i}$$

1.2.5 Errores de punto flotante

ARITMÉTICA DE PUNTO FLOTANTE

Las operaciones de suma, resta, multiplicación y división en el sistema de punto flotante (F), se denota por $\oplus$, $-, \otimes$ y $\div$ respectivamente. Estas operaciones están definidas por:

$$x \oplus y = fl(fl(x) + fl(y))$$
$$x - y = fl(fl(x) - fl(y))$$
$$x \otimes y = fl(fl(x) \,.\, fl(y))$$
$$x \div y = fl(fl(x) \div fl(y)),\ fl(y) \neq 0,\ y \neq 0$$

Estas operaciones no son cerradas sobre F, pues en algunos casos se genera underflow u overflow; más aún en otros casos producen una variedad de errores, que se indican más adelante. Esto nos lleva a situaciones inesperadas, por lo que se recomienda siempre tener presente estas situaciones en el momento que se programa o analizan los resultados numéricos de algún software en particular.

Los errores cometidos al realizar operaciones aritméticas en el sistema de punto flotante, los podemos tipificar según los siguientes criterios:

1. **Error de Redondeo:** Ya se menciono que éste tipo de error es el obtenido por el proceso de redondeo (exceso o defecto) de los números reales en su representación por números punto flotante ó de máquina.

2. **Error Significativo:** Este error ocurre con frecuencia cuando se restan números casi iguales o cuando se suman números "casi iguales" en magnitud, pero de signos contrarios. También se presentan cuando se divide por un divisor relativamente pequeño.

   Ejemplo 1:
   Sea un ordenador (hipotético) que trabaje con números expresados en base 10, con cuatro dígitos de precisión, y con aproximación por redondeo.

   a) La suma de dos números
   Sean $x = 4/3$ e $y = 2/9$; se pide calcular $x + y$. Evidentemente, si se efectúa un cálculo exacto, se obtiene $x + y = 14/9 = 1.5555...$ con lo que
   $fl(x + y) = 0.1556 \times 10^1$
   Sin embargo, los cálculos del ordenador se harían todos en la correspondiente aritmética de punto flotante, es decir, como $x = 1.3333...$ e $y = 0.2222...,$ resulta
   $fl(x) = 0.1333 \times 10^1, \quad fl(y) = 0.2222 \times 10^0$
   y
   $fl(fl(x) + fl(y)) = fl(1.3333 + 0.2222) = fl(1.5552) = 0.155 \times 10^1$
   Se observa pos consiguiente que $fl(fl(x) + fl(y)) \neq fl(x + y)$

b) La resta de dos números muy próximos entre sí

Sean:

$x = 0.3721478693$

$y = 0.3720230572$

Se pide calcular $x - y$. Evidentemente, si se efectúa un cálculo exacto, se obtiene

$x - y = 0.0001248121$

Si los cálculos se realizan con precisión de 4 decimales:

$fl(x) = 0.3721$

$fl(y) = 0.3720$

$fl(x) - fl(y) = 0.0001$

Error relativo = $\left| \dfrac{(x-y)-(fl(x)-fl(y))}{x-y} \right| = 0.1988$

Es decir, del orden del 19.88%, error relativo que es por tanto muy elevado

En consecuencia, hay que evitar que en el ordenador se produzcan operaciones de sustracción de sumandos casi iguales. Esto puede lograrse de diversas maneras, y a continuación exponemos un ejemplo

Ejemplo 2:

Al calcular: $y = \sqrt{x^2+1} - 1$, cuando $x = 10^{-10}$, se obtiene $y = 0$.

Se puede reescribir de otra manera:

$$y = \frac{\sqrt{x^2+1}-1}{\sqrt{x^2+1}+1}\left(\sqrt{x^2+1}+1\right) = \frac{x^2}{\sqrt{x^2+1}+1} \approx 5x10^{-21}$$

1.3 Aritmética del computador

1.3.1 Representación de números en el computador

Los computadores trabajan con aritmética real usando un sistema denominado de "punto flotante". Suponen un número real que tiene la expansión binaria:

$x = \pm m \; x \; 2^{Ee}$, donde $1 \le m < 2$

y $m = (b_0.b_1b_2b_3\cdots)_2 \quad b_i = 0 \; o \; 1$

Cuando $b_0 = 1$, se dice que es un número normalizado.

Exponente externo (Ee), donde Ee=Ei-bias

bias=2^{k-1}-1, con k= # de bits del Exponente interno (Ei)

Para almacenar un número en representación de punto flotante, el computador usa 3 campos:

Signo (S)	Exponente Interno (Ei)	Mantisa (M)

Signo (S): El signo del numero, 0=positivo, 1=negativo.

Exponente (Ei): Potencia de la base 2, mas adelante se explica con más detalle

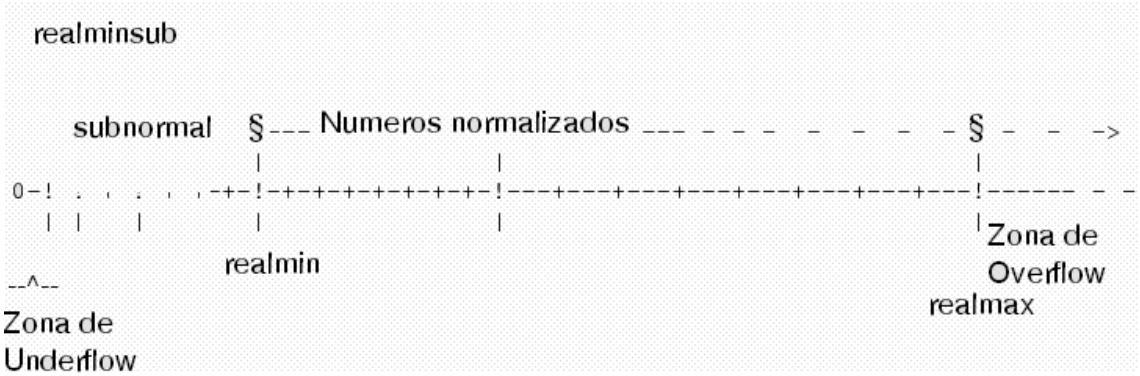
Mantisa (M): Dígitos de la parte decimal $M = b_1b_2b_3\cdots \quad b_i = 0 \; o \; 1$.

Dado que un número en punto flotante puede expresarse de distintas formas que son equivalentes, es necesario establecer una única representación. Es por ello que se trabaja con números ***normalizados***. Decimos que un número está normalizado si el dígito a la izquierda del punto o coma está entre 0 y la base (0 < dígito a la izquierda del punto < *b*).

En particular, decimos que un número binario está normalizado si el dígito a la izquierda del punto es igual a 1

Ejemplo:
1.00×10^{-1} normalizado
0.01×10^{2} no normalizado

1.3.2 Distribución de los datos en la recta numérica



- Si en los cálculos se genera un resultado inferior al menor valor positivo que tenga almacenamiento exacto en el computador se producirá un desbordamiento denominado **Underflow**. El menor valor positivo es un valor no normalizado o subnormal (**realminsub**).
- También existe un menor valor normalizado (**realmin**).
- El número positivo mas grande que se puede almacenar se denomina **realmax**, si en los cálculos se genera un resultado mayor que este valor adoptara el valor **inf**, lo que indica un desbordamiento de rango.
- La precisión de la maquina (**epsilon**) según la IEEE se define como la distancia de **1** al siguiente numero que tenga almacenamiento exacto.
- Puesto que la cantidad de números a almacenar es una cantidad finita, la mayoría de números reales tendrán que ser aproximados a aquellos que tienen una representación exacta en el sistema de punto flotante empleado. Esto origina las perdidas de precisión por redondeo.

1.3.3 Estándar IEEE-754 para representación de Punto Flotante

Este estándar se desarrolló para facilitar la portabilidad de los programas de un procesadora otro y para alentar el desarrollo de programas numéricos sofisticados. Este estándar ha sido ampliamente adoptado y se utiliza prácticamente en todos los procesadores y coprocesadores aritméticos actuales. El estándar del IEEE define el formato para precisión simple de 32 bits y para precisión doble de 64 bits.

Hasta la década de los 90 cada computador utilizaba su propio formato en punto flotante, en 1985 se introduce el estándar IEEE-754 con la finalidad de uniformizarlos.

1.3.4 Números de simple y doble precisión

Precisión simple: (32 bits)

Signo (S) **1 bit**	**Exponente (E)** **8 bits**	**Mantisa (M)** **23 bits**

Precisión doble: (64 bits)

Signo (S) **1 bit**	**Exponente (E)** **11 bits**	**Mantisa (M)** **52 bits**

La doble precisión permite manejar mas cifras significativas y exponentes de mayor rango que en simple precisión.

A continuación se muestra una tabla de conversión muy importante:

Precisión simple (32 bits): Exponente Exceso 127(Ei)

Exponente interno(Ei)	**Mantisa**	**Representa**
-127 (combinación binaria 0)	0	± 0
-127 (combinación binaria 0) {*Notación Subnormal*}	$\neq 0$	$(-1)^S (0.m_1 m_2 \cdots m_{23})_2 .2^{-126}$ {*Notación Subnormal*}
[-126,127] (combinación binaria [1, 254]) {*Notación Normal*}	----	$(-1)^S (1.m_1 m_2 \cdots m_{23})_2 .2^{(e_1 e_2 \cdots e_8)_2 - 127}$ {*Notación Normal*}
128 (combinación binaria 255)	0	$\pm \inf$
128 (combinación binaria 255)	$\neq 0$	*NaN*

Precisión doble (64 bits): Exponente Exceso 1023 (Ei)

Exponente	**Mantisa**	**Representa**
-1023	0	± 0
-1023	$\neq 0$	$(-1)^S (0.m_1 m_2 \cdots m_{52})_2 .2^{-1022}$
[-1022,1023]	----	$(-1)^S (1.m_1 m_2 \cdots m_{52})_2 .2^{(e_1 e_2 \cdots e_{11})_2 - 1023}$
1024	0	$\pm \inf$

1024	$\neq 0$	*NaN*

1.3.5 Consideraciones especiales

- **Definición del cero:** puesto que el significando se supone almacenado en forma normalizada, no es posible representar el cero (se supone siempre precedido de un 1). Por esta razón se convino que el cero se representaría con valores 0 en el exponente y en la mantisa. Ejemplo:

  0 00000000 00000000000000000000000 = +0

  1 00000000 00000000000000000000000 = -0

  Observe que en estas condiciones el bit de signo S aún permite distinguir +0 de -0. Sin embargo el Estándar establece que al comparar ambos "ceros" el resultado debe indicar que son iguales.
- **Infinitos:** se ha convenido que cuando todos los bits del exponente están a 1 y todos los de la mantisa a 0, el valor es +/- infinito (según el valor S). Esta distinción ha permitido al Estándar definir procedimientos para continuar las operaciones después que se ha alcanzado uno de estos valores (después de un overflow). Ejemplo:

  0 11111111 00000000000000000000000 = +Infinito

  1 11111111 00000000000000000000000 = -Infinito
- **Valores no-normalizados:** (denominados también "subnormales"). Se identifican porque todos los bits del exponente son 0 pero la mantisa presenta un valor distinto de cero (en caso contrario se trataría de un cero).

  Ejemplo:
  1 00000000 00100010001001010101010
- **Valores no-numéricos**: Denominados NaN ("Not-a-number"). Se identifican por un exponente con todos sus valores a 1, y una mantisa distinta de cero.

  Ejemplo:
  0 11111111 10000100000000000000000
  1 11111111 00100010001001010101010

1.3.6 Sistemas de números y conversiones

- Sistema de numeración decimal utiliza como base de numeración el 10 (dígitos 0…9)
- Un número **en base q** se denota como: $(a_n a_{n-1} \ldots a_1 a_0 . b_1 b_2 \ldots b_k \ldots)_q$ donde q toma valores desde 0 hasta q-1
- La conversión a decimales es, por definición:

$$(a_n a_{n-1} \ldots a_1 a_0 . b_1 b_2 \ldots b_k \ldots)_q = a_n q^n + a_{n-1} q^{n-1} + \ldots + a_1 q^1 + a_0 q^0 + \\ + b_1 q^{-1} + b_2 q^{-2} + \ldots + b_k q^{-k} + \ldots$$

- El sistema natural de numeración digital es el binario (base 2), utilizando sólo los dígitos 0 y 1.

Ejemplo:

Decimal: $(123.25)_{10} = 1\times10^{2} + 2\times10^{1} + 3\times10^{0} + 2\times10^{-1} + 5\times10^{-2}$

Binario: $(1011.01)_2 = 1\times2^{3} + 0\times2^{2} + 1\times2^{1} + 1\times2^{0} + 0\times2^{-1} + 1\times2^{-2} = 11.25$

- La conversión de **base decimal** a **base q** es:

  C*onversión de la parte entera*:

  $(a_n a_{n-1} \ldots a_1 a_0)_q = (a_n a_{n-1} \ldots a_1)_q \times q + (a_0)_q$

  Al dividir un número entero entre q el resto es el dígito menos significativo del número en base q. Dividiendo sucesivamente (hasta llegar al cociente 0) obtenemos los sucesivos dígitos del número en base q.

  Conversión de la parte fraccionaria:

  $(.b_1 b_2 \ldots b_k)_q \times q = (b_1)_q + (.b_2 \ldots b_k)_q$

  La parte entera del resultado de multiplicar nuestro número por q es el primer dígito tras el punto decimal. Reteniendo la parte fraccionaria que resulta en cada paso y repitiendo sucesivamente el proceso, vamos obteniendo el resto de cifras tras el punto.

  Ejemplo

  Escribamos $(26.1)_{10}$ en base 2

  Solución:

  Parte entera: Dividiendo sucesivamente, tenemos que:

  $26 = 2\times13 + \underline{0}; \quad 13 = 2\times6 + \underline{1}; \quad 6 = 2\times3 + \underline{0}; \quad 3 = 2\times1 + \underline{1}; \quad 1 = 2\times0 + \underline{1}$

  Leyendo de izquierda a derecha los números subrayados:

  $(26)_{10} = (11010)_2$

  Parte fraccionaria: Multiplicando sucesivamente por dos y separando la parte fraccionaria:

  $0.1\times2 = \underline{0}.2; \quad 0.2\times2 = \underline{0}.4; \quad 0.4\times2 = \underline{0}.8; \quad 0.8\times2 = \underline{1}.6; \quad 0.6\times2 = \underline{1}.2;$

  $0.2\times2 = \ldots$

  Leyendo de izquierda a derecha tenemos los dígitos de la parte fraccionaria (subrayados) luego $(0.1)_{10} = (0.0\underline{0011})_2$.

  Donde las cifras subrayadas son las cifras periódicas.

  Por lo tanto: $(26)_{10} = (11010.0\underline{0011})_2$

1.3.7 Ejercicios resueltos

1. Representar según el estándar IEEE de punto flotante precisión simple para los siguiente valores:
   a) 7
   b) 21

   Solución:
   (a)
   Convertimos el número a binario: $7_{10}= 111_2$
   Normalizamos el número: $1.11_2 \times 10_2^2$
   Calculamos el exponente con exceso 127 para precisión simple
   $2+127=129_{10} = 1000\ 0001_2$
   El número 7_{10} en el estándar IEEE es representado como:

   0 10000001 11000000000000000000000

   (b)
   $21_{10} = 10101_2 = 1.0101_2 \times 10_2^4$
   exponente: $4+127 = 131_{10} = 10000011_2$

   0 10000011 01010000000000000000000

2. Convertir 0.5_{10} a binario y hallar su representación en IEEE precisión simple

   Solución:
   Convertimos el número a binario: $0.5 \times 2 = \underline{1}.0; \quad 0.0 \times 2 = \underline{0}.0; \quad \rightarrow 0.5_{10} = 0.1_2$
   Normalizamos el número: $0.5_{10} = 1.0_2 \times 10_2^{-1}$
   Calculamos el exponente con exceso 127 para precisión simple
   $-1+127=126_{10} = 01111110$
   El número 0.5_{10} en el estándar IEEE es representado como:

   0 01111110 00000000000000000000000

3. Cual es el valor decimal de: 1 01111100 11000000000000000000000?

   Solución:
   El bit de signo es 1: Numero negativo
   El exponente contiene 01111100 = 124
   La mantisa es 0.1100…. = 0.75

   El valor es: $(-1)x(1+0.75)x\left(2^{124-127}\right) = -0.21875$

4. Convertir 3.75 a binario y hallar su representación en IEEE precisión simple

   Solución:

   Convertimos el número a binario: $3.75_{10} = 11.11_2$

   Normalizamos el número: $3.75_{10} = 1.111_2 \times 2^1$

   Calculamos el exponente con exceso 127 para precisión simple
   $1+127=128_{10} = 1000\ 0000_2$
   El número 3.75_{10} en el estándar IEEE es representado como:

   0 1000 0000 11100000000000000000000

5. Cual es la representación en simple precisión de 347.625?

   Solución:

   Convertir a binario: 347.625 = 101011011.101
   Normalizar el numero (mover el punto decimal hasta que haya un solo 1 a la izquierda) :
   $101011011.101=1.01011011101x2^8$
   Mantisa: 01011011101
   Calculamos el exponente con exceso 127 para precisión simple
   $8 + 127 = 135_{10} = 10000111_2$
   El numero es positivo: bit de signo 0

   0 10000111 01011011101000000000000

6. Si suponemos una representación en punto flotante como a continuación se detalla, donde *v* indica el valor del número representado, determinar:

   a) La representación del 1,
   b) El mínimo número no nulo normalizado representable en valor absoluto,
   c) El mínimo número representable en valor absoluto (distinto de 0), y
   d) El máximo número representable en valor absoluto.

Signo(1 bit)	Exponente (8 bits)	Mantisa (32 bits)
S	E	f

$$0 < e < 255 \rightarrow v = (-1)^s * 1.f * 2^{e-127}$$
$$e = 0 \rightarrow v = (-1)^s * 0.f * 2^{-126}$$

   Solución

   a) Representación del 1

0	01111111	00……0

$$v=1.0*2^{127-127}$$

b) Mínimo número no nulo normalizado

0	00000001	00…….0

$v=1.0*2^{1-127}$

c) Mínimo número representable no nulo

0	00000000	00……01

$v=0.0\ldots\ldots1*2^{-126}$

d) Máximo número representable en valor absoluto

0	11111110	1 1 1…….111

$v=1.1\ldots.1*2^{254-127}$

7. Sea un sistema binario de punto flotante de 16 bits:

s	$e_1e_2e_3e_4e_5$	$m_1m_2m_3m_4m_5m_6m_7m_8m_9m_{10}$

Donde los números normalizados son de la forma:

$$x=(-1)^s 1.m_1m_2m_3\cdots m_{10}\,2^{(e_1e_2e_3e_4e_5)_2-15}$$

Además:

$00000_2<(e_1e_2e_3e_4e_5)_2<11111_2$

Muestre la representación en punto flotante y los 16 bits del:

a) Numero 1
b) El numero 1+eps
c) Mayor valor positivo normalizado
d) Menor valor positivo normalizado
e) El numero: 475.65625

Solución

a)

$(-1)^0 1.0000000000x2^{01111_2-15}$

0 01111 0000000000

b)

$(-1)^0 1.0000000001x2^{01111_2-15}$

0 01111 0000000001

c)
$(-1)^0 1.1111111111 x 2^{11110_2 - 15}$

0 11110 1111111111

d)
$(-1)^0 1.0000000000 x 2^{00001_2 - 15}$

0 00001 0000000000

e)
$475.65625 = 111011011.10101_2 = 1.1101101110 \text{x} 2^8$
$(e_1 e_2 e_3 e_4 e_5)_2 - 15 = 8$
$(e_1 e_2 e_3 e_4 e_5)_2 = 23 = 10111_2$
0 10111 1101101110

1.3.8 Ejercicios propuestos

1. Convertir 0.3_{10} a binario y hallar su representación en IEEE precisión simple.
2. ¿Cual es el número siguiente a 143.1 que tiene representación exacta en simple precisión?
3. ¿Qué número decimal representa el siguiente patrón de bits en IEEE precisión simple?
   0 00001100 01000000000000000000000

4. Determine para los sistemas de simple y doble precisión:
   - El menor valor positivo normalizado
   - El menor valor positivo no normalizado
   - El mayor valor positivo
   - El epsilón de la maquina

1.4 Puntos flotantes en MATLAB

A continuación reproducimos un interesantísimo artículo de Cleve Moler http://www.mathworks.com/company/newsletters/news_notes/pdf/Fall96Cleve.pdf
Uno de los creadores del MATLAB (vs. Traducida Por R.M.G.J.):

Puntos Flotantes

Si vemos cuidadosamente las definiciones de las principales operaciones aritméticas como la suma y la multiplicación, rápidamente encontraremos la abstracción matemática conocida como los *números reales*. Pero el cómputo efectivo con números reales no es muy práctico porque ello implica límites e infinitos. En cambio, MATLAB y muchos otros ambientes de computación técnica usan aritmética de punto flotante, lo cual implica un conjunto finito de números con precisión finita. Esto conduce a fenómenos tales como errores de redondeo, *underflow* y *overflow.* La mayor parte de las veces, MATLAB puede ser usado efectivamente sin preocuparse por estos detalles, pero algunas veces, es conveniente conocer algo acerca de las propiedades y limitaciones de los numero de punto flotante.
Hace veinte años la situación era más complicada de lo que es actualmente. Cada computadora tenía su propio sistema numérico de punto flotante. Algunos eran binarios; otros eran decimales. Incluso había una computadora rusa que utilizaba aritmética ternaria. Entre las computadoras binarias, algunas usaban 2 como base; otras utilizaban 8 o 16. Y cada una tenía una precisión diferente.

En 1985, el IEEE Standards Board y la American National Standards Institute adoptaron la norma ANSI/IEEE Standard 754-1985 para la aritmética binaria de punto flotante. Esto fue la culminación de casi una década de trabajo de un grupo de 92 personas, matemáticos, científicos de computadoras e ingenieros provenientes de Universidades y compañías fabricantes de computadoras y microprocesadores.

Todas las computadoras diseñadas en los últimos 15 años usan la aritmética flotante de la IEEE. Esto no significa que todas obtienen exactamente los mismos resultados, porque hay cierta flexibilidad dentro del estándar. Pero si que tendremos un modelo independiente de la máquina de cómo la aritmética de punto flotante se comporta.
MATLAB utiliza el formato de doble precisión de la IEEE. También hay un formato de simple precisión el cual ahorra espacio pero no es mucho más rápido en las máquinas modernas. Y, hay un formato de precisión extendida, el cual es opcional y por consiguiente es una de las razones para la ausencia de uniformidad entre diferentes máquinas.
Muchos números de punto flotante están normalizados. Esto significa que ellos pueden ser expresados como:

$$x = \pm (1 + f) \bullet 2^{e}$$

donde f es la fracción o mantisa y e es el exponente. La fracción debe satisfacer:

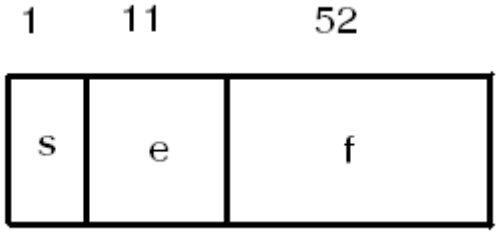
$$0 \le f < 1$$

y debe ser representable en binario usando a lo más 52 bits. En otras palabras $2^{52} f$ debe ser un entero en el intervalo:

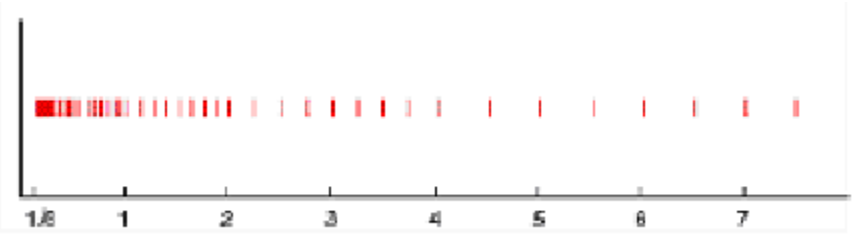
$$0 \le 2^{52} f < 2^{53}$$

El exponente debe ser un entero en el intervalo:
$-1022 \le e \le 1023$

El que *f* sea finito es una limitación sobre la *precisión*. El que *e* sea finito es una limitación sobre el *rango*. Los números que no cumplen estas limitaciones deben ser aproximados por otros que sí:



Los números de punto flotante de doble precisión pueden se almacenados en una extensión de 64 bits, con 52 bits para ***f***, 11 bits para *e* y 01 bit para el signo del número. El signo de ***e*** es acomodado almacenando *e*+1023, el cual esta entre 1 y $2^{11} - 2$. Los dos valores extremos para el campo del exponente, 0 y $2^{11} - 1$, están reservados para números de punto flotante excepcionales, los cuales describiremos posteriormente:



La figura anterior muestra la distribución de los números positivos en un sistema ficticio de punto flotante con solo 03 bits para ***f*** y para ***e*** entre 2^e y 2^{e+1} los números están igualmente espaciados con un incremento de 2^{e-3}.Cuando ***e*** crece, el espaciamiento crece. El espaciamiento de los números entre 1 y 2 en nuestro sistema es 2^3 o 1/8. En el sistema de IEEE, el espaciamiento es 2^{-52} .MATLAB denota a esta cantidad ***eps***, el cual es el ***epsilón de la máquina***.

Eps=2^(-52)

Antes del estándar de la IEEE, diferentes máquinas tenían diferentes de ***eps***.

El valor decimal aproximado de ***eps*** es: 2.2204 e^{-16}. Tanto *eps*/2 o *eps* pueden ser llamados el nivel de redondeo. El máximo error relativo incurrido cuando el resultado de una operación aritmética simple es redondeado al más cercano número de punto flotante es ***eps*/2**. **El máximo espaciamiento relativo entre números es *eps***. En cualquier caso, se puede decir que el nivel de redondeo es alrededor de 16 dígitos decimales.
Un ejemplo muy importante ocurre con la simple asignación en MATLAB
t=0.1
El valor almacenado en t no es exactamente 0.1 porque expresar la fracción decimal 1/10 en binario requiere una serie infinita:

$$1/10 = 1/2^4 + 1/2^5 + 0/2^6 + 0/2^7 + 1/2^8 + 1/2^9 + 0/2^{10} + 0/2^{11} + 1/2^{12} + \cdots$$

Después del primer término, la secuencia de coeficientes 1,0,0,1 se repite indefinidamente. El número de puntos flotantes más cercano a 0.1 es obtenido redondeando esta serie a 53 términos incluyendo el redondeo de los cuatro últimos coeficientes al binario 1010. Agrupando los términos resultantes de cuatro cada vez se expresa la aproximación como una serie de base 16 o hexadecimal. Luego el valor resultante para t es:

$$t = \left(1 + 9/16 + 9/16^2 + 9/16^3 + \cdots 9/16^{12} + 10/16^3\right)2^{-4}$$

El comando de MATLAB
format hex
Ocasiona que t se imprima como

$$3fb999999999999a$$

Los primeros tres caracteres 3fb, proporcionan la representación hexadecimal del exponente alterado, $e+1023$, donde e es –4. Los otros 13 caracteres son la representación hexadecimal de la fracción f.
Luego el valor almacenado en t es muy cercano, pero no exactamente igual, a 0.1. Esta distinción es ocasionalmente importante. Por ejemplo, la cantidad:
0.3/0.1
No es exactamente igual a 3 porque el numerador actual es un poco menor que 0.3 y el denominador actual es un poco mayor que 0.1.
Diez pasos de longitud t no son precisamente lo mismo que un paso de longitud 1. MATLAB es cuidadoso para arreglar que el último elemento del vector:

0:0.1:1

es exactamente igual a 1, pero si Ud. forma este vector por si mismo por incrementos adicionales de 0.1, Ud. no llegara a 1 exactamente.
Otro ejemplo es previsto por el código MATLAB siguiente:

```
a = 4/3
b = a – 1
c = b + b + b
e = 1 – c
```

Un cálculo exacto nos llevaría a que debe ser cero. Por el punto flotante el valor de e calculado no es cero. Se desprende que el valor de redondeo ocurre en la primera sentencia. El valor almacenado en a no es exactamente 4/3, excepto en la computadora trinaria rusa. El valor almacenado en b es cercano a 1/3, pero su último bit es cero. El valor almacenado en c no es exactamente igual a 1, porque las sumas son hechas con algún error. Luego el valor almacenado en e no es cero. En realidad, e es igual a eps. Antes del estándar IEEE, este cogido era usado como una forma fácil para estimar los errores de redondeo en varias computadoras.

El nivel de redondeo eps es algunas veces llamado "*punto flotante cero*", pero esto es un nombre inapropiado. Existen muchos números de punto flotante más pequeños que el **eps**. El más pequeño número positivo de punto flotante normalizado tiene f=0 y e=-1022. El mayor numero de punto flotante tiene un f es un poco menor que uno y e=1023. MATLAB llama a estos números realmin y realmax respectivamente. Estos números juntos con **eps**, caracterizan al sistema estándar.

Nombre	Binario	Decimal
eps	2^{-52}	2.2204e-16
realmin	2^{-1022}	2.2251e-308
realmax	$(2-eps)*2^{1023}$	1.7977e+308

Cuándo un cálculo trata de producir un valor mayor que realmax, esto se llama *overflow*. El resultado es un valor excepcional de punto flotante llamado Inf, o infinito. Esto es representado tomando *f*=0 y *e*=1024 y satisface relaciones como 1/inf=0 e inf+inf=inf.

Cuando algún calculo trata de producir un valor menor que mineral, esto se llama underflow. Esto envuelve uno de los aspectos opcionales y controversiales del estándar IEEE. Muchas, pero no todas, las maquinas permiten números excepcionales denormal o subnormal de punto flotante en el intervalo entre realmin y **eps***realmin. El más pequeño ***número subnormal*** positivo es el alrededor **0.494e-^{323}**. Cualquier resultado menor que este numero es asignado como cero. En máquinas sin subnormales, todo resultado menor que realmin es asignado como cero. ***Los números subnormales*** caen en el caso que se puede ver en nuestro sistema ficticio entre cero y el más pequeño numero positivo. Ellos proveen un modelo elegante para manejar el underflow, pero su importancia práctica para el estilo de computación de MATLAB es muy raro. Cuando un cálculo trata de producir un valor que es indefinido en el sistema de números reales, el resultado es un valor excepcional conocido como Not-a-Number, o NaN. Ejemplos incluyen 0/0 y Inf – Inf.

MATLAB usa el sistema de punto flotante para manejar los enteros. Matemáticamente, los números 3 y 3.0 son los mismos, pero en muchos lenguajes de programación usan diferentes representaciones para los dos. MATLAB no distingue entre ambos. Nos gusta usar el término *flint* para describir un número flotante cuyo valor es un entero. Las operaciones del punto flotante sobre *flints* no introducen errores de redondeo, desde que los resultados no son tan grandes. La suma, la resta y la multiplicación de *flints* produce el resultado *flint* exacto, si este no es mayor que 2^{53}. La división y la raíz cuadrada que involucran flints también producen un *flint* cuando el resultado es un entero. Por ejemplo, sqrt(363/3) produce 11, sin error de redondeo.
Como un ejemplo de cómo el error de redondeo afecta en el cálculo de las matrices, considere el sistema lineal de dos ecuaciones con dos incógnitas:

$$10x_1 + x_2 = 11$$
$$3x_1 + 0.3x_2 = 3.3$$

La solución obvia es x_1=1, y x_2=1. Pero las sentencias de MATLAB

```
A=[ 10 1 ; 3 0.3 ]
b=[11 3.3]'
x=A\b
```

produce:

x=

-0.5000
16.0000 {*Según el artículo,*
pero en la vs 7.0 del Matlab esto produce x_1=*NaN y* x_2=*NaN*}

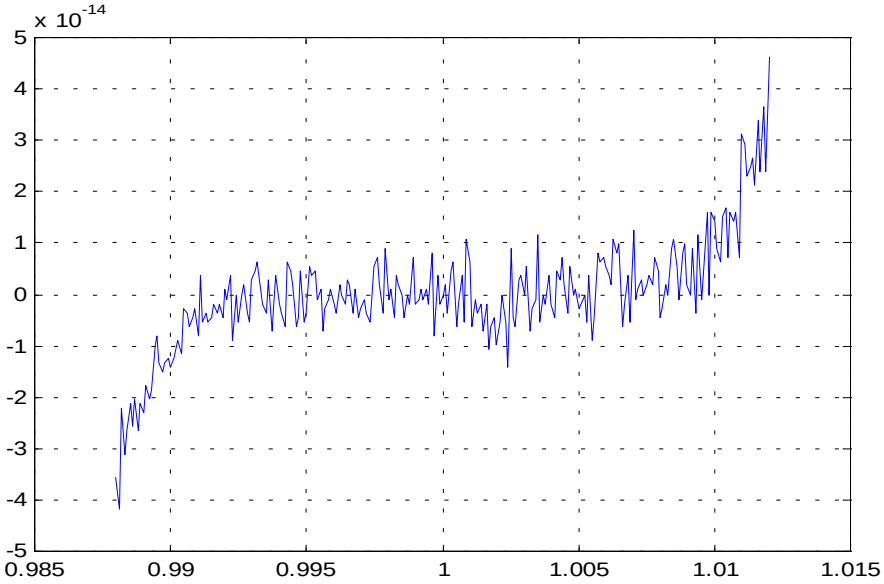
¿Porque? Bueno, las ecuaciones son singulares. La segunda ecuación es justamente 0.3 veces que la primera. Pero la representación en punto flotante de la matriz A no es exactamente singular porque A(2,2) no es exactamente 0.3.

La eliminación Gaussiana transforma las ecuaciones al sistema triangular superior
U*x=c

Donde
U(2,2) = 0.3-3*(0.1)
= -5.5551e-17
y
c(2) = 3.3 – 33*(0.1)
= $-4.4409e^{-16}$
MATLAB detecta el pequeño valor de U(2,2) e imprime un mensaje de advertencia que la matriz es cercana a la singular, Entonces calcula la relación de los dos errores de redondeo
X(2) = c(2)/U(2,2)
= 16
Este valor es substituido en la primera ecuación para dar
X(1) = (11 - x(2))/10
= - 0.5
Las ecuaciones singulares son consistentes. Hay un número infinito de soluciones. Los detalles del error de redondeo determinan cual solución particular será calculada.

Nuestro ejemplo final grafica un polinomio de séptimo grado:

```
x=0.988:.0001:1.012;
y=x.^7-7*x.^6+21*x.^5-35*x.^4+35*x.^3-21*x.^2+7*x-1;
plot(x,y)
```



Pero el gráfico resultante no se parece nada a un polinomio. No es suave. Estamos viendo el error de redondeo en acción. El factor de escala del eje Y es muy pequeño, 10^{14}. Los pequeños valores de y son calculados tomando sumas y diferencias de números tan grandes como 35x 1.012^4. Existe una severa cancelación por diferencia. El ejemplo fue ideado usando Toolbox Simbólic para expandir $(x - 1)^7$ y cuidadosamente elegido el rango del eje x para que sea cercano a x=1. Si los valores de y son calculados en cambio por:
Y= (x – 1).^7;
Entonces una grafica suave (pero muy plana) es obtenida.