TEMA I

APROXIMACIÓN NUMÉRICA

1.1. Números en coma (punto) flotante.

"Hay 10 *tipos de personas: Las que conocen el código binario y las que no".*

1.1.1. Algunos aspectos básicos del almacenamiento informático.

Los computadores trabajan en el sistema binario, en contraste con los humanos que utilizamos normalmente el sistema decimal.

En un número decimal, o en cualquier otra base, el valor de sus dígitos depende de la posición que ocupen; podemos resaltar esto expresando un número más explícitamente, por ejemplo

Base 10 : $(476.36)_{10} = 4\cdot 10^{2} + 7\cdot 10^{1} + 6\cdot 10^{0} + 3\cdot 10^{-1} + 6\cdot 10^{-2}$

En el sistema binario (base 2), sólo se utilizan los dígitos 0 y 1. También podemos representar un número binario en la forma expandida, tal como lo hemos hecho con el número decimal anterior:

Base 2: $(1010.11101)_2 = 1\cdot 2^{3} + 1\cdot 2^{1} + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 1\cdot 2^{-3} + 1\cdot 2^{-5} = (10.90625)_{10}$

Realizadas las operaciones anteriores obtenemos:

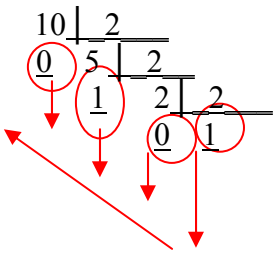
$$(1010.11101)_2 = (10.90625)_{10}$$

La anterior es la forma más fácil de pasar un número binario a decimal.

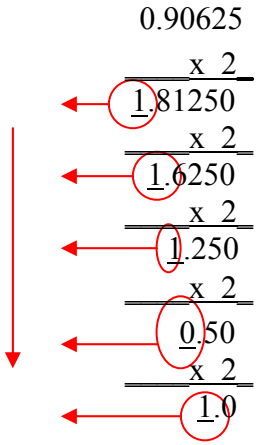
Para pasar un número de base 10 a base 2, la forma más simple es la siguiente: La parte entera se divide sucesivamente por dos y se van apartando los restos y el último cociente; mientras que la parte decimal se multiplica sucesivamente por dos y se van apartando las partes enteras. Ejemplo: Pasamos el número 10.90625 a base 2.

10 . 90625

divido entre 2 multiplico por 2



$(10)_{10} = (1010)_2$



$(0.900625)_{10}=(0.11101)_2$

$(10.900625)_{10}=(1010.11101)_2$

El usuario y el ordenador se comunican utilizando el sistema decimal. Después, mediante un proceso de conexión interno, la calculadora pasa los números a sistema binario. Aunque el usuario no debe de preocuparse de estas conversiones, si debe de tener conciencia de que implican procesos de redondeo y acotamiento de los números reales que podemos representar, ya que utilizamos una memoria finita para representar el conjunto infinito de los números.

Definición de bit: Un bit es el elemento más básico de una memoria. Puede estar en dos estados, a los que llamamos 0 y 1.

Bit es una palabra inglesa clásica que significa "trozo, pedacito, pieza pequeña", pero la palabra informática bit procede de la contracción de binary digit (número binario), haciéndose un juego de palabras con ambos significados.

En un disco duro o disquete, un bit se representa por el cambio o no cambio en el sentido de la imanación de dos bandas seguidas. Si la imanación tiene la misma dirección asociaremos con un 0, y si cambia de dirección, pondremos un 1.

Ejemplo de un bit

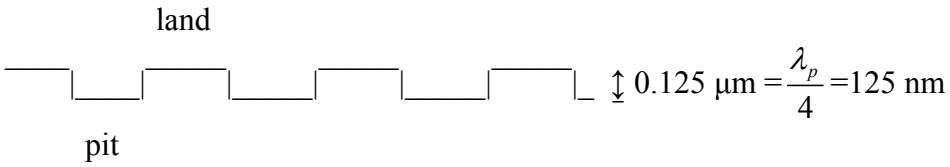
| → :→ | (La flecha indica el sentido de la imanación magnética)
0

Indicamos que pasamos de un bit a otro, mediante un cambio en la imanación o magnetización de las zonas

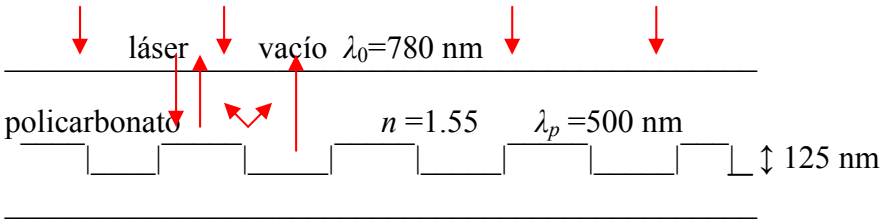
| → : → | ←: ← | →:← | →: ... |
0 1 0
↓
(el paso a otro bit se indica con un cambio en el sentido de la imanación)

El CD-ROM (Compact Disc-Read Only Memory) utiliza un sistema de almacenamiento distinto al de los discos magnéticos. Fue desarrollado conjuntamente por las multinacionales Philips y Sony con el fin de evitar las cuantiosas pérdidas económicas que suponía la fácil copia ilegal de los sistemas magnéticos (cassettes y discos).

La pista que contiene la información es una espiral que arranca del interior del disco, y que con espesor de 0.6 μm (100 veces más fina que un cabello), tiene 6 km de longitud sobre la superficie del disco. La pista, normalmente de aluminio, posee una serie de hondonadas (pits) y zonas llanas (lands) que forman una serie de escalones cuyo número varía entre 4 y 5 mil millones. La pista está protegida con una capa de 1.2 mm de espesor de policarbonato, material transparente con índice de refracción n=1.55. El CD-ROM se ilumina con un láser de longitud de onda en el vacío λ_0= 780 nm (zona fronteriza entre el rojo y el infrarrojo). Como la profundidad de los pits es de 0.125 μm, justamente $\lambda/4$ de la longitud de onda del láser en el policarbonato, cuando el rayo del láser ilumina entre un pit y un land, la luz reflejada por el pit ha recorrido una distancia $\lambda/2$ superior a la recorrida por la reflejada por el land, produciéndose una interferencia destructiva para la dirección perpendicular al plano que contiene al CD. Si el rayo del láser ilumina completamente un pit o un land, la luz es completamente reflejada. Asociamos un 1 a la escasa o nula energía luminosa reflejada en una transición pit-land y un cero a la luz reflejada, bien por un pit o por un land.



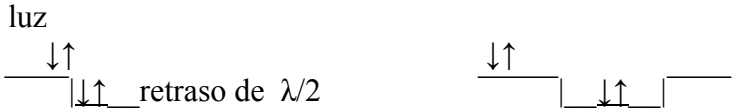
Iluminamos con un láser de longitud de onda en el vacío λ_0 = 780nm (zona entre el color rojo y la radiación infrarroja)



$\lambda_0 f = c$, $\lambda_p f = v_p$, $n=c/v_p$, n=1.55 $\Rightarrow$

λ_0 = 780 nm (en el vacío)

λ_p = 500 nm (en el policarbonato)



La luz reflejada es escasa o nula

Toda la luz se refleja

Las transiciones son los 1 (poca o nula luz) y las no transiciones son los 0 (se refleja toda la luz).

Los CD-ROM grabables (⊃ piratas) funcionan de manera distinta a los originales. Debajo de la capa de policarbonato, hay otra capa de una materia orgánica inicialmente transparente. Debajo de esta capa hay una superficie reflectante. Las transiciones del CD original ("*unos*", o no reflexión de la luz) se consiguen quemando

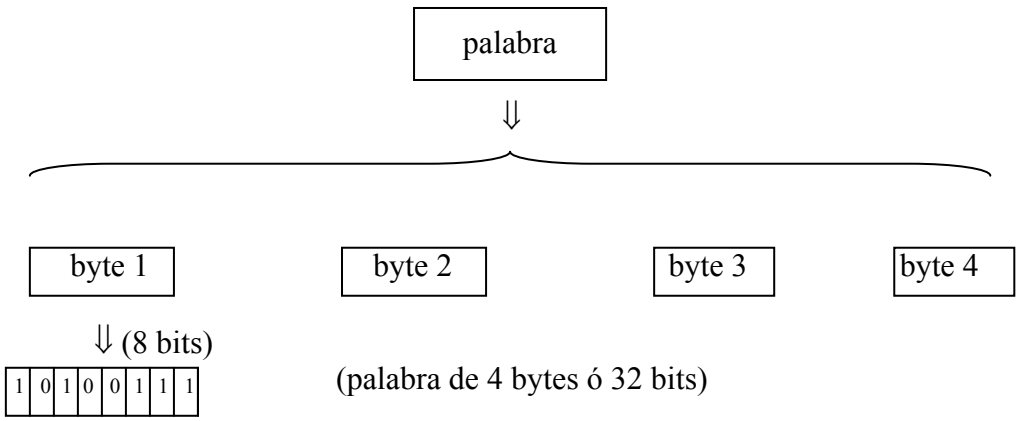
la sustancia orgánica mediante un láser de mayor potencia que el que se utiliza para la lectura de datos. Las zonas quemadas no reflejan bien y producen el mismo efecto que una transición pit-land. Actualmente se sintetizan sustancias orgánicas que cambian de transparentes a opacas si se las calienta con un láser y luego se las enfría lenta o rápidamente. Con estas sustancias se fabrican los CD regrabables.

Los bits se agrupan en bytes.

Definición de byte: 1 byte = 8 bits = Unidad básica de información.

En un CD-ROM, 1 byte está compuesto por 14 bits propiamente dichos y 3 bits de enlace entre bytes (17 bits) por la imposibilidad física de tener dos "*unos*" seguidos y porque al menos deben de ir dos "*ceros*" seguidos ya que la longitud mínima de un pit o un land es de dos "*ceros*". Después estos 14 bits se transforman en conjuntos de 8 bits que ya si pueden tener dos "*unos*" seguidos.

Definición de palabra: Una palabra o dato (por ejemplo, un número) se almacena utilizando un número determinado de bytes que pueden ser 1, 2, 4, 8, etc. Así, podremos tener palabras de 1 ó 2 ó 4 ó 8 bytes. El sistema operativo MS-DOS y el WIN 3.1 son sistemas operativos de 16 bits (2 bytes), mientras que el WIN95, WIN98, Millenium o WIN-XP son de 32 bits (4 bytes).



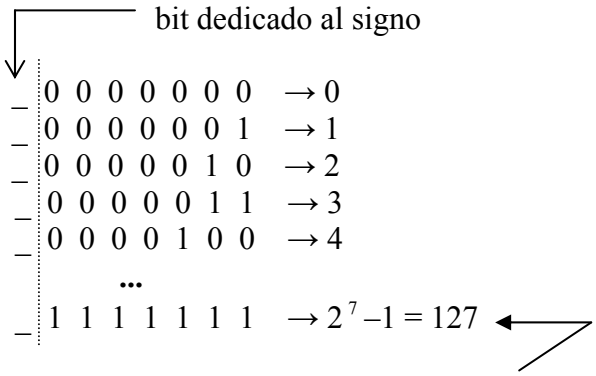
1.1.2. Almacenamiento de números enteros.

Vamos a calcular el número de números enteros distintos que podemos almacenar en una palabra de 1 byte (8 bits). Tendremos 8 posiciones que tendremos que rellenar, cada una de ellas, con un 1 ó un 0. Como los números enteros pueden ser positivos o negativos, reservamos el primer bit para el signo, asociando el + con el 0, y el – con el 1:

$$+ \leftrightarrow 0 \quad \text{ya que } (-1)^0 = 1$$
$$- \leftrightarrow 1 \quad \text{ya que } (-1)^1 = -1$$

| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | (en cada bit ponemos un 0 ó 1)

↑
signo (±)



(Si al último número, 1111111, le sumamos un 1, obtenemos 10000000=2^7)

Es decir, el rango de un número entero, representado con un solo byte, oscila entre –127 y 127. Si utilizamos palabras con más bytes obtendremos los resultados que se muestran en la siguiente tabla:

Entero representado por *n* bytes		Rango
8 bits (1 byte) →	$\pm(2^7-1) = \pm 127$	Entre –127 y +127
16 bits (2 bytes) →	$\pm(2^{15}-1) = \pm 32767$	Entre –32 767 y +32 767
32 bits (4 bytes)→	$\pm(2^{31}-1) = \pm 2\,147\,483\,647$	Entre ±2 147 483 647

Por defecto, si no se especifica lo contrario, se utilizan palabras de 4 bytes.

1.1.3. Almacenamiento de los números reales. Representación en coma flotante según el estándar IEEE-754 de 1985.

En el sistema decimal, cualquier número real puede expresarse en notación científica normalizada o *floating point* (coma o punto flotante), como un solo dígito a la izquierda del punto decimal, los decimales necesarios, y todo ello multiplicado por la correspondiente potencia de diez. Por ejemplo, la velocidad de la luz la expresamos en notación científica como

$$c = 2.998 \cdot 10^{8}\ \text{m/s}$$

Mantisa Exponente

No es habitual encontrarse un número del tipo 6451.43 ó 0.012, sino que nos los encontraremos escritos en la forma $6.45143\cdot10^{3}$ y $1.2\cdot10^{-2}$. En cada número expresado en forma científica distinguimos la mantisa (M) y el exponente (E). Para almacenar un número tendremos, pues, que almacenar su mantisa y su exponente.

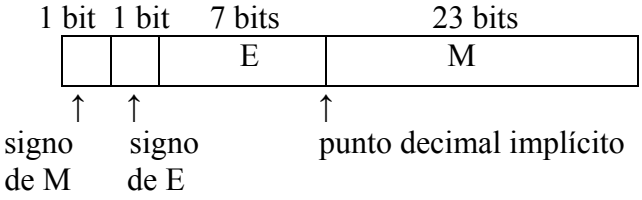
Pero como los ordenadores trabajan en base 2, debemos de utilizar la notación científica en base 2. Así, un número real tendrá la forma

$$n = M \cdot 2^{E}$$, con M y E en base 2.

Almacenamiento en simple precisión.

Siguiendo el estándar IEEE-754, (IEEE = Institute of Electrical and Electronics Engineers), vamos a describir el almacenamiento que se denomina en simple precisión y en el que se utiliza una palabra de 4 bytes (32 bits)

Utilizamos 4 bytes → 32 bits



Los dos primeros bits los reservamos al signo de la mantisa y del exponente, respectivamente. Los 7 siguientes bits para el exponente E y los restantes 23 para la mantisa M.

Representación del exponente.

El exponente, E, puede variar entre

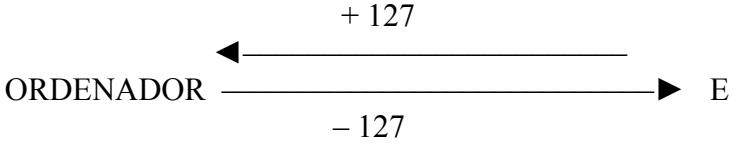
$0\,0\,0\,0\,0\,0\,0 \rightarrow 0 \quad \Rightarrow$ representa $2^0 = 1$
$0\,0\,0\,0\,0\,0\,1 \rightarrow 1 \quad \Rightarrow$ representa $2^1 = 2$
.....
.....
$1\,1\,1\,1\,1\,1\,1 \rightarrow 2^7 - 1 = 127 \Rightarrow$ representa $2^{127} \approx 10^{38}$

} 128 posiciones distintas.

Con el signo ±, el exponente está comprendido entre ±127, que representa números entre 10^{38} y 10^{-39}, aproximadamente.

Representación de signo expresado en exceso para el exponente.

Pero la norma IEEE-754, que siguen la mayoría de los fabricantes de programas de ordenador, dice que para evitar el signo de E, se le sume al exponente 127 antes de guardarlo en memoria, y se utilicen los 8 bits para el almacenamiento de E



Esto se hace para que no haya números negativos.

Con 8 bits tenemos ahora:

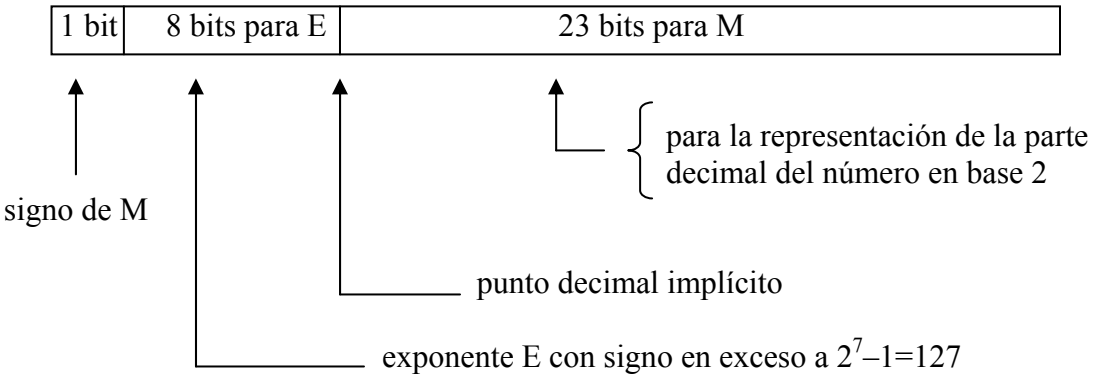
0 0 0 0 0 0 0 0 → 0 ⇒ Restando 127, representa a E=–127
0 0 0 0 0 0 0 1 → 1 ⇒ Restando 127, representa a E=–126
.....
0 1 1 1 1 1 1 1 → 127 ⇒ Restando 127, representa a E= 0

.....
1 1 1 1 1 1 1 1 → 2^8–1 = 255⇒ Restando 127, representa a E= 128

Ahora tenemos 256 posiciones diferentes. Antes teníamos 127x2+1=255, una menos que ahora, ya que antes se repetía ±0.

Con este tipo de representación para el exponente E, que se denomina representación con el signo expresado en exceso a $2^{n-1}-1$, (con n=8, $2^{n-1}-1$=127) siendo n el número de bits que tiene el exponente, se consigue ganar una posición, cambiándose el exponente repetido en el ±0 por el 128 que antes no teníamos.

Nos queda la palabra de 4 bytes estructurada en la siguiente forma:



Almacenamiento de la mantisa.

Volvamos a nuestro número en base 2, n=M· 2^E, para representar la mantisa M. La mantisa debe tener la forma 1._ _ _ _, ya que si tuviese la forma 0._ _ _ _, tendríamos que cambiar el exponente para conseguir la forma estándar 1._ _ _ _ (representación científica o en coma flotante).

El número a la izquierda del punto decimal es un 1 seguro y, como es conocido, no es necesario almacenarlo. Sólo almacenaremos la parte decimal de la mantisa en los 23 casilleros dedicados a M. Los iremos colocando en orden, a partir del punto decimal, el dígito de la izquierda del punto decimal ocupará el primer bit, el segundo dígito irá al segundo casillero o bit, etc.

2^{-1}	2^{-2}	(23 bits para M)	2^{-23}

De forma que el valor del dígito de la izquierda de la mantisa será 0 ó 1 por 2^{-1}, el segundo será 0 ó 1 por 2^{-2}, hasta el último que será el dígito 0 ó 1 por 2^{-23}.

Número de cifras decimales.

Un detalle importante que debemos de tener en cuenta es el número de cifras decimales que nos deja utilizar la simple precisión. El número de cifras decimales viene controlado por el número de bits dedicados a la mantisa. El dígito con menor valor es el último, el que ocupa la posición 23, asociado al valor $2^{-23}= 1.2\cdot10^{-7}$ ⇒ Los números digitales en simple precisión tienen su precisión limitada por seis cifras decimales. Los números con más cifras decimales serán objeto de redondeo.

Es por esto que no es conveniente quitar bits de la mantisa para dárselos al exponente, pues aunque aumentaríamos la zona de números reales, sería a cambio de perder precisión perdiendo cifras decimales.

Cantidad de números reales distintos que podemos representar en simple precisión. Números máquina.

De todas formas, hagamos lo que hagamos, para representar números reales con 32 bits, disponemos de 32 casilleros que podemos rellenar con ceros y unos. Si lo hacemos tendremos las siguientes posiciones:

1 2 .....31 32

0 0 .....0 0 → 0
0 0 .....0 1 → 1
0 0 .....1 0 → 2
.....

1 1 .....1 1 → $(2^{32}-1) = 4\ 294\ 967\ 255$

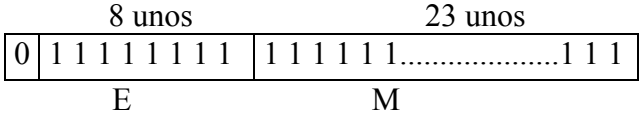
} 2^{32}=4 294 967 256 posiciones

⇒ Sólo podemos representar 4 294 967 256 números reales distintos de los infinitos existentes.

El conjunto de números reales que podemos almacenar en una computadora no tiene una estructura algebraica coherente y reciben el nombre de números máquina.

Ejercicio. Calcular el número real más grande que puede ser representado con 32 bits según la norma IEEE-754.

Ponemos el bit del signo de la mantisa en cero para que ésta sea positiva. El resto de bits los ponemos al máximo, es decir, los ponemos a 1



El exponente viene dado por

$$E = (\,1\,1\,1\,1\,1\,1\,1\,1\,)_2 - (\,127\,)_{10} = (2^8 - 1)_{10} - 127 = 255 - 127 = 128$$

↓

representado en exceso a 127

La mantisa viene dada por

$$M = (1.\overbrace{1\;1\;1\;1\;........1}^{23\text{ unos}}\,)_2 = (\,2 - 2^{-23}\,)_{10} = 1.999999881$$

↑ Este es el 1 que no se almacena.

El número buscado será:

$n = M \cdot 2^E = (\,2 - 2^{-23}\,) \cdot 2^{128} = 6.805\,646\,8 \cdot 10^{38}$

¿HASTA CUÁNTO PUEDE CONTAR USTED?

Según un cuento, dos aristócratas húngaros decidieron jugar a un juego en el cual ganaría quien dijera el número más alto.

-Bien -dijo uno de ellos-, di tú primeramente el número.

Después de algunos minutos de intenso trabajo mental, el segundo aristócrata dijo, finalmente, el mayor número en el que podía pensar.

-Tres -exclamó.

Entonces, le tocó el turno al primero para meditar, pero después de un cuarto de hora se dio por vencido.

-Has ganado -le dijo.

Por supuesto, los dos aristócratas húngaros no representan un grado de inteligencia muy alto y el cuento no es, probablemente, más que una calumnia maliciosa, pero podría haberse sostenido realmente una conversación semejante si los hombres hubieran sido, no húngaros, sino hotentotes. Sabemos positivamente, por los exploradores africanos, que muchas tribus de hotentotes no tienen en su vocabulario los nombres de los números superiores a tres. Si se pregunta a un nativo cuántos hijos tiene o cuántos enemigos ha matado, y si el número representa más de tres, contestará: muchos. Así, en el país de los hotentotes, en lo que respecta al arte de contar, los feroces guerreros serían vencidos por un niño de jardín de infancia que se pudiese jactar de contar ¡hasta diez!

Actualmente estamos muy acostumbrados a la idea de que podemos escribir el número más alto que nos agrade -ya represente los gastos de guerra en centavos, o distancias estelares en pulgadas- mediante la simple colocación de un número suficiente de ceros en el lado derecho de la misma cifra. Usted puede poner ceros hasta que se le canse la mano, y antes de que lo sepa tendrá un número aún más grande que el número total de átomos que existen en el universo, el cual, aproximadamente, es 300.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.

O puede escribirlo de esta manera más breve: 3×10^{74}.

Aquí el número más pequeño, 74, colocado arriba y a la derecha del 10, indica que se deben escribir muchos ceros, o en otras palabras: que 3 se debe multiplicar por 10 setenta y cuatro veces.

Pero este sistema «de aritmética hecha fácil» no se conocía en la Antigüedad. En realidad, lo inventó hace menos de dos mil años algún matemático hindú desconocido. Antes de este gran descubrimiento -y fue un gran descubrimiento, aunque generalmente no nos demos cuenta- los números se escribían utilizando un símbolo especial para cada una de las que ahora llamamos unidades decimales, y se repetía este símbolo tantas veces como unidades hubiera. Por ejemplo, el número 8732 era escrito por los antiguos egipcios en esta forma:

ЪЪЪЪЪЪCCCCCCCЋЋЋЛЛ

George Gamow. "Un, dos, tres…infinito". Capítulo I: Los grandes números, pp. 21-22. RBA Editores, Barcelona, 1993.

Ejercicio. Calcular el número real positivo más pequeño que puede ser representado con 32 bits según la norma IEEE-754.

Si lo buscamos positivo, ponemos el primer bit, correspondiente al signo de M en 0. El resto de los bits los ponemos al mínimo⇒ todos a 0.

0	0 0 0 0 0 0 0 0	0 0 0 0 0 0

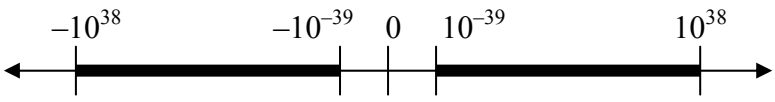
8 bits 23 bits

Debemos de llevar en mente el 1 de la mantisa que no representamos y que expresamos el exponente en exceso a 127. Con esto

E = 0 – 127 = –127
M = 1.0 0 0 .....0 (un *uno*, seguido de 23 *ceros*)
$n = 1.0 \cdot 2^{-127} = 5.877\,747\,2 \cdot 10^{-39}$

<u>Región ocupada por los números reales en simple precisión.</u>

Procediendo de manera similar con las mantisas negativas, observamos que los números reales en simple precisión están en dos regiones.



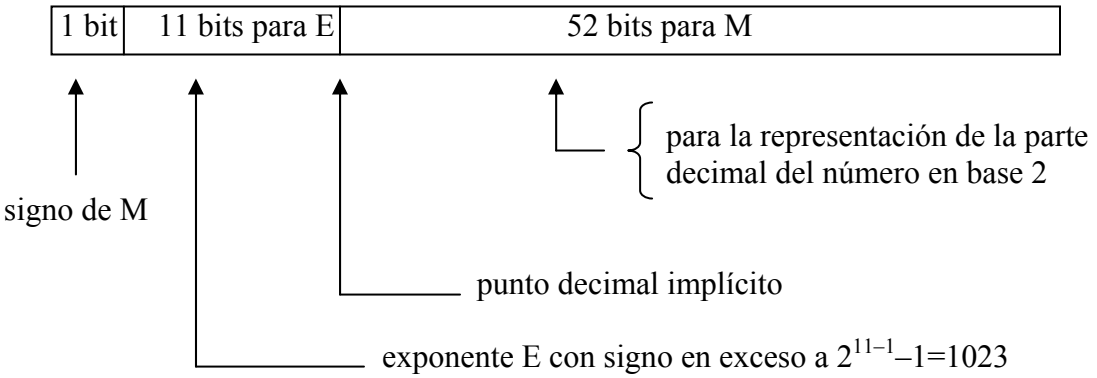
Los compiladores también incluyen el cero, ya que no permiten E y M llenos completamente de ceros, tal como hemos hecho nosotros en el cálculo del número real más pequeño, y exigen que al menos un bit de la mantisa sea 1. Si todos los bits de E y M son ceros, el compilador asigna el cero directamente. Algunos compiladores exigen que al menos uno de los cuatro bits principales sea distinto de cero, para que no se asigne el cero.

En el compilador FORTRAN de Microsoft aparece como número real positivo mayor, la mitad del que hemos calculado aquí. Este número mitad es el que se obtiene si en lugar de hacer una representación del signo del exponente en exceso, se guarda un bit para el signo de E. Para este compilador, el número real positivo más pequeño se corresponde con una mantisa llena de *unos* en lugar de *ceros* y un exponente lleno de *ceros*.

Si en el desarrollo de un programa aparece un número de magnitud inferior a la frontera 10^{-39}, se produce un *underflow* (desbordamiento por defecto) y, normalmente, se le asigna el valor 0. Los números superiores a la frontera máxima, 10^{38}, producen un *overflow* (desbordamiento por exceso) y, normalmente, traen como consecuencia un aviso de error y la interrupción de los cálculos.

Almacenamiento en doble precisión.

Cuando necesitamos aumentar el campo y/o precisión de almacenamiento de números reales en simple precisión, bien porque necesitamos más decimales, o bien porque los números con los que estamos trabajando pudiesen exceder de las fronteras que impone la simple precisión, debemos recurrir a utilizar más bytes para nuestra palabra. Con lo que llamamos doble precisión utilizamos 8 bytes (64 bits) para un solo número. El estándar IEEE-754 aconseja dedicar 11 para el exponente y 53 para la mantisa, de estos últimos uno se dedica al signo y los 52 restantes para la magnitud de la mantisa. La palabra de 8 bytes queda estructurada en la siguiente forma:



Repitiendo los consideraciones que ya hicimos para la simple precisión, podemos encontrar las principales características de este almacenamiento:

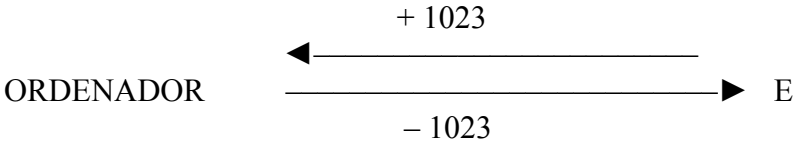
Frontera superior

Con 11 bits para E $\Rightarrow 2^{11} = 2048$ situaciones distintas$\Rightarrow$ desde –1023,...,0,...,1024.

0 0 0 0 0 0 0 0 0 0 0 → –1023
0 0 0 0 0 0 0 0 0 0 1 → –1022
.....
.....
0 1 1 1 1 1 1 1 1 1 1 → 0
.....
.....
1 1 1 1 1 1 1 1 1 1 1 → 1024

2048 posiciones distintas

Se representa con signo expresado en exceso a $2^{11-1}-1=1023$, evitando así los exponentes negativos.



La frontera superior está ligada al exponente más grande, 1024. En base decimal, el número ligado a este exponente es

$$2^{1024}=(2^{256})^4=(1.157920892\cdot 10^{77})^4=1.797693135\cdot 10^{308}$$

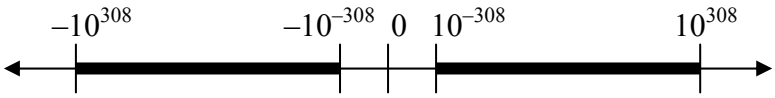
Frontera inferior.

Con el signo expresado en exceso a 1023, el exponente más pequeño, ligado con la frontera inferior es E=0–1023=–1023, que se corresponderá con el número decimal

$$2^{-1023}=(2/2)\ 2^{-1023}=2\cdot 2^{-1024}=2(2^{-256})^4=2(8.636168576\cdot 10^{-78})^4=$$

$$=2\cdot(8.636168576)^4\cdot 10^{-312}\approx 10^{-308}$$

La región, sobre la recta real, que ocupa la doble precisión es



Numero de cifras decimales.

El número de cifras decimales está unido al dígito menos significativo de la mantisa que es le que ocupa la posición 52, en base 10 su valor será:

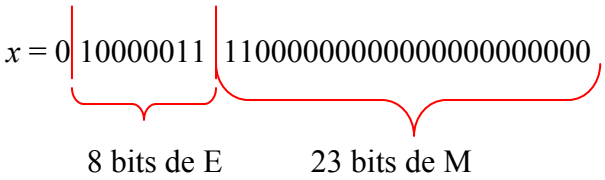
$$2^{-52}=2.22\cdot 10^{-16}\ \Rightarrow$$

$\Rightarrow$ en doble precisión podemos almacenar números con 15 cifras decimales exactas.

Ejemplo: Averiguar el número decimal que corresponde con el número x siguiente, dado en formato binario de coma flotante siguiendo el estándar IEEE-754.

$$x = 01000001111000000000000000000000$$

Dividimos el número separándolo en tres partes: signo de M, E y M



El bit del signo de M es un cero $\Rightarrow$ la mantisa es +

El exponente almacenado en exceso a 127, será en base 10:

$E = (10000011)_2 - (127)_{10} = 2^7 + 2^1 + 2^0 - 127 = 4$

Mantisa: $(11000000000000000000000)_2 \Rightarrow M = 1 + 2^{-1} + 2^{-2} = 1.75$

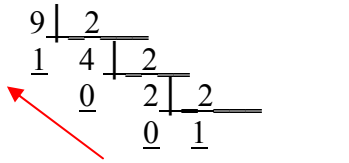
Por tanto, $(x)_{10} = (1.75 \cdot 2^4)_{10} = 28$

Ejercicio: Expresar en formato IEEE-754, en simple precisión, el número –9.

Antes de almacenarlo, lo vamos a expresar en formato de coma flotante, para después pasar el exponente y la mantisa a la norma IEEE-754

$-9 = M \cdot 2^E$

Para calcular E y M, expresamos 9 en binario. El signo negativo sólo lo tenemos que tener en cuenta al darle valor al primer bit que es el que lleva esta información. Este bit tendrá el valor 1 que es el que se corresponde con el signo –.



$(9)_{10} = (1001)_2 = (1.001)_2 \cdot 2^3$

directamente a la mantisa

El exponente se introduce con signo expresado en exceso a 127 $\Rightarrow$

$(127)_{10} = (1111111)_2$
$(3)_{10} = (0000011)_2$
$(130)_{10} = (10000010)_2$

Con todo lo anterior

$(-9)_{10} \equiv$ | 1 | 10000010 | 00100000000000000000000 |

1.1.4. Números máquina próximos.

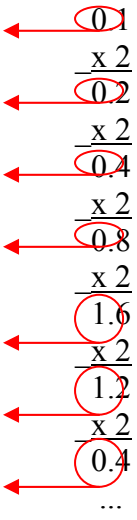
Llamamos números máquinas al conjunto de números reales que pueden ser almacenados exactamente en un ordenador. Ya que el conjunto de números máquina es finito, deben esperarse errores de redondeo siempre que se introduzcan datos en un ordenador.

Por ejemplo, ni siquiera un número sencillo en base 10, como 0.1, puede almacenarse exactamente. Para comprobar esto vamos a expresar este número en coma

flotante en base 2. Para esto lo pasamos en primer lugar a base 2, y lo podemos hacer bien dividiendo $(1)_{10}=(1)_2$ entre $(10)_{10}=(1010)_2$, ambos expresados en base 2:

```
1.0000     |1010
0 0110     0.0001100110 0
  0010000
    00110
     001000
```

obteniéndose una división no exacta que se repite cíclicamente; o bien, podemos pasar a base 2 en la forma habitual



y así, continuamos multiplicando por 2, obteniendo una cadena indefinida de dígitos para expresar $(0.1)_{10}$ en base 2. Por ambos caminos hemos obtenido, lógicamente, el mismo resultado:

$$(0.1)_{10} = \frac{(1)_{10}}{(10)_{10}} = \frac{(1)_2}{(1010)_2} = (0.000\ 110\ 011\ 001\ 100\ 110\ 0\ ...)_2 =$$

$$= (1.100\ 110\ 011\ 001\ 100\ 110\ 011\ 001\ 100\ ...)_2 \cdot 2^{-4}$$

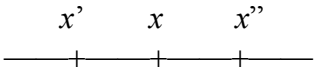
Ya que la mantisa es indefinida y sólo podemos guardar los primeros 23 dígitos, no existe un número maquina que represente a $(0.1)_{10}$ exactamente. Este hecho puede verificarse fácilmente sumando 10^7 veces el número 0.1 con el programa:

```
X=0.1
S=0.0
DO I=1, 10000000
S=S+X
END DO
PRINT *, S
STOP
END
```

El resultado es 1087937 y no 10^6, como era de esperar si $(0.1)_{10}$ fuese un número máquina y no se cometiesen errores de redondeo.

Nos formulamos ahora la pregunta: ¿Cuál es el número máquina más cercano a un número dado x? Al número máquina más cercano a x lo llamaremos $fl(x)$.

Nuestro número x ocupará un punto en la recta real y estará situado entre dos números máquina, uno por debajo al que llamaremos x' y otro por arriba al que llamamos x''.



El número x tendrá la siguiente forma general

$$x = (1.\, a_1\ a_2\ a_3\ \ldots\ a_{23} \mid a_{24}\ a_{25}\ \ldots) \cdot 2^E$$

El número x' se obtiene por truncamiento de la mantisa, al eliminar los dígitos a_{24}, a_{25}, ... que no pueden ser almacenados:

$$x' = (1.\, a_1\ a_2\ a_3\ \ldots\ a_{23}) \cdot 2^E$$

El número x'' se obtiene mediante redondeo por exceso. Esto significa que eliminamos el exceso de dígitos, pero aumentamos en una unidad el último dígito que es el que ocupa la posición 23.

$$x'' = (1.\, a_1\ a_2\ a_3 \ldots\ldots\ldots a_{23}) \cdot 2^E + 1 \cdot 2^{-23} \cdot 2^E$$

que podemos poner en la forma

$$x'' = x' + 1 \cdot 2^{-23} \cdot 2^E \quad \Rightarrow \boxed{x'' - x' = 2^{E-23}}$$

Si x está más cerca de x' que de x'', el error que cometemos es

$$|x - x'| \le \frac{1}{2} |x'' - x'| = \frac{1}{2} 2^{E-23} = \boxed{2^{E-24}}$$

Pero, si x está más cerca de x'' que a x', el error es

$$|x'' - x| \le \frac{1}{2} |x'' - x'| = \boxed{2^{E-24}}$$

En cualquiera de los casos, el error cometido es igual o inferior a 2^{E-24},

$$|fl(x) - x| \le 2^{E-24}$$

Ejemplo. ¿Cuál es la forma binaria del número decimal $x = \frac{2}{3}$? ¿Cuáles son los números máquina *x'* y *x''* más cercanos a *x*? ¿Cuál de estos dos números se elegirá como *fl*(*x*) (número más cercano a *x*)? ¿Cuál es el error absoluto y relativo que se comete al representar *x* por *fl*(*x*)? ¿Cuál será el formato IEEE-754 para *x*?

Pasamos nuestro número *x* a base 2, en formato estándar $x = M \cdot 2^E$

$$x = \frac{(2)_{10}}{(3)_{10}} = \frac{(10)_2}{(11)_2} = (0.101010...)_2 = (1.010101...)_2 \cdot 2^{-1}$$

```
1 0 . 0          |11
  0  1 0 0       0.10101
     0 1 0 0
```

Debido a que los números de la mantisa se repiten periódicamente, este número no pertenece al conjunto de números máquina. Los números máquina más próximos serán:

$x' = (1.010101 \ldots 10)_2 \cdot 2^{-1}$ (obtenido por truncamiento)

posición 23 después del punto decimal

$x'' = (1.010101 \ldots 11)_2 \cdot 2^{-1}$ (obtenido por redondeo por exceso)

Para saber cuál de los dos anteriores números es *fl*(*x*), calculamos las diferencias *x*–*x'* y *x''*–*x*:

$$\begin{array}{r} x = (1.0101010101010101010101010 \ldots) \cdot 2^{-1} \\ -x' = (1.010101010101010101010) \cdot 2^{-1} \\ \hline (0.00000000000000000000000101010 \ldots) \cdot 2^{-1} \end{array}$$

$$\Rightarrow \quad x - x' = (1.01010\ldots) \cdot 2^{-24} \cdot 2^{-1} = \frac{2}{3} \cdot 2^{-24}$$

Por su parte, $x'' - x = (x'' - x') - (x - x') = 2^{-24} - \frac{2}{3} \cdot 2^{-24} = \frac{1}{3} \cdot 2^{-24}$

Y por tanto, $fl(x) = x''$

Los errores absoluto y relativo vienen dados por

$$|fl(x) - x| = |x'' - x| = \frac{1}{3} \cdot 2^{-24} \approx 2 \cdot 10^{-8}$$

y

$$\frac{|fl(x) - x|}{|x|} = \frac{\frac{1}{3} \cdot 2^{-24}}{\frac{2}{3}} = 2^{-25} \approx 3 \cdot 10^{-8}$$

En formato de coma flotante estándar IEEE-754, el número x es reemplazado por x". De este número sabemos ya la mantisa y el exponente, –1, con signo expresado en exceso a 127, tendrá el valor E = 127–1 = 126.

$$\begin{array}{rl} (127)_{10} = & (1111111)_2 \qquad \text{(7 unos)} \\ -\ (1)_{10} = & -(0000001)_2 \\ \hline (126)_{10} = & (1111110)_2 \end{array}$$

Completando todos los bits:

0	01111110	01010101010101010101011

1.1. 5. Representación de caracteres.

Para representar caracteres se utiliza el código ASCII (American Standard Code for Information Interchange). Utilizando este estándar a cada carácter se le asocia un número entero que se representa en 8 bits (1 byte). Algunos códigos ASCII son:

0→ 48, 1→ 49, 2 → 50, ..., 9 → 57
A → 65, B → 66, C → 67, ...
a → 97, b → 98, c → 99, ...
(→ 40,) → 41, * → 42, + → 43, – → 45, . → 46, / → 47, = → 61

1.2. Inestabilidad Numérica.

Un proceso numérico se dice inestable cuando los pequeños errores que se producen en alguna de sus etapas se agrandan en etapas posteriores y degradan seriamente la exactitud del cálculo en su conjunto.

Ejemplo 1. Un ejemplo nos permitirá comprender este concepto. Consideremos la sucesión de números reales definida inductivamente mediante las siguientes expresiones:

$$\begin{cases} x_0 = 1, \qquad x_1 = \dfrac{1}{3}, \\ x_n = \dfrac{13}{3}\, x_{n-1} - \dfrac{4}{3}\, x_{n-2} \qquad (n \geq 2) \end{cases}$$

Tomando una calculadora de mano, podemos ir encontrando los diferentes números integrantes de esta sucesión y completar la columna titulada "Sucesión inductiva" de la siguiente tabla.

Por otra parte, vamos a demostrar que la sucesión inductiva anterior es idéntica a la sucesión $x_n = \left(\frac{1}{3}\right)^n$, en la que es evidente que $x_n \to 0$ con $n \to \infty$.

Demostramos que ambas sucesiones son equivalentes utilizando el método de recurrencia o de inducción matemática, en el que comprobamos que se cumple lo que queremos demostrar para $m=1$, suponemos que es válido para un m arbitrario y demostramos que se cumple para $m+1$.

Es claro que $x_0 = \left(\frac{1}{3}\right)^0 = 1$, también $x_1 = \frac{1}{3}$, y analogamente $x_2 = \frac{13}{3}\frac{1}{3} - \frac{4}{3}1 = \frac{13}{9} - \frac{12}{9} = \frac{1}{9} = \left(\frac{1}{3}\right)^2$.

Supongamos que son iguales hasta orden m, es decir $x_m = \left(\frac{1}{3}\right)^m$, y demostramos la validez para $m+1 \Rightarrow$ Nos preguntamos $\text{¿}\, x_{m+1} = \left(\frac{1}{3}\right)^{m+1}$?

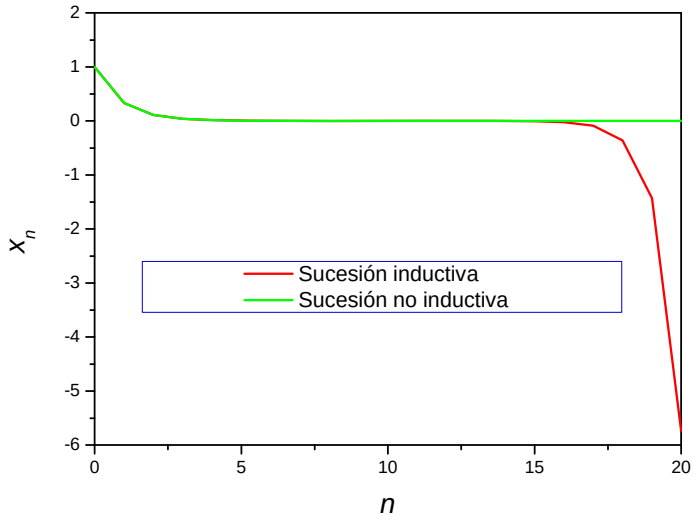
$$x_{m+1} = \frac{13}{3}x_m - \frac{4}{3}x_{m-1} = \frac{13}{3}\left(\frac{1}{3}\right)^m - \frac{4}{3}\left(\frac{1}{3}\right)^{m-1} = \left(\frac{1}{3}\right)^{m-1}\left[\frac{13}{3}\frac{1}{3} - \frac{4}{3}\right] = \left(\frac{1}{3}\right)^{m-1}\frac{1}{9} = \left(\frac{1}{3}\right)^{m+1}$$

Con los valores que aporta la sucesión $x_n = \left(\frac{1}{3}\right)^n$ rellenamos la columna titulada "Sucesión no inductiva". Ambas columnas deberían de ser iguales puesto que las sucesiones son idénticas, y a medida que n aumenta deberíamos obtener números más pequeños puesto que las sucesiones tienden a cero, con n tendiendo a infinito.

	Sucesión inductiva	Sucesión no inductiva	Comparativa entre sucesiones
$x_0 =$	1	1	Los 9 dígitos coinciden
$x_1 =$	0. 333 333 333	0. 333 333 333	Los 9 dígitos coinciden
$x_2 =$	0. 111 111 111	0. 111 111 111	Los 9 dígitos coinciden
$x_3 =$	0. 037 037 037	0. 037 037 037	Los 9 dígitos coinciden
$x_4 =$	0. 012 345 678	0. 012 345 679	El último dígito es diferente
$x_5 =$	0. 004 115 221	0. 004 115 226	El último dígito es diferente
$x_6 =$	0. 001 371 721	0. 001 371 742	Los 2 últimos son diferentes
$x_7 =$	0. 000 457 162	0. 000 457 247	Los 3 últimos son diferentes
$x_8 =$	0. 000 152 074	0. 000 152 415	Los 3 últimos son diferentes
$x_9 =$	0. 000 049 439	0. 000 050 805	Sólo 1 bien por redondeo
$x_{10} =$	0. 000 011 468	0. 000 016 935	Uno coincide sin redondeo
$x_{11} =$	–0. 000 016 224	0. 000 005 645	Ninguno coincide
$x_{12} =$	–0. 000 085 595	0. 000 001 881	Ninguno coincide
$x_{13} =$	–0. 000 349 278	0. 000 000 627	Ninguno coincide
$x_{14} =$	–0. 001 399 412	0. 000 000 209	Ninguno coincide

$x_{15}=$	–0. 005 598 416	0. 000 000 069	Ninguno coincide
$x_{16}=$	–0. 022 393 920	0. 000 000 023	Ninguno coincide
$x_{17}=$	–0. 089 575 764	0. 000 000 007	Ninguno coincide
$x_{18}=$	–0. 358 303 086	0. 000 000 002	Ninguno coincide
$x_{19}=$	–1. 433 212 353	$8.603916\cdot 10^{-10}$	Ninguno coincide
$x_{20}=$	–5. 732 849 415	$2.867972\cdot 10^{-10}$	Ninguno coincide

¿Qué ha ocurrido? El procedimiento inductivo a resultado ser un algoritmo numérico inestable. Debería haber sido una serie que tendiese a cero, pero en una etapa determinada ha roto su tendencia a ser cada vez más pequeño en magnitud, se ha vuelto negativo y a comenzado a crecer en magnitud.



La razón es la siguiente: Cualquier error que se comete en el cálculo de x_1, se multiplica por $\frac{13}{3}$ en el cálculo x_2, por $\left(\frac{13}{3}\right)^2$ en el cálculo de x_3, ..., y por $\left(\frac{13}{3}\right)^{19}$ en x_{20}. Ya que el error absoluto en x_1 es aproximadamente de $\delta=10^{-11}$ porque la calculadora utiliza un dígito más de los que posee en pantalla para redondeo, los errores en las diferentes etapas serán:

$$x_1 \rightarrow \delta$$
$$x_2 \rightarrow \left(\frac{13}{3}\right)\delta$$
$$x_3 \rightarrow \left(\frac{13}{3}\right)^2\delta$$
$$\cdots$$
$$x_{20} \rightarrow \left(\frac{13}{3}\right)^{19}\delta$$

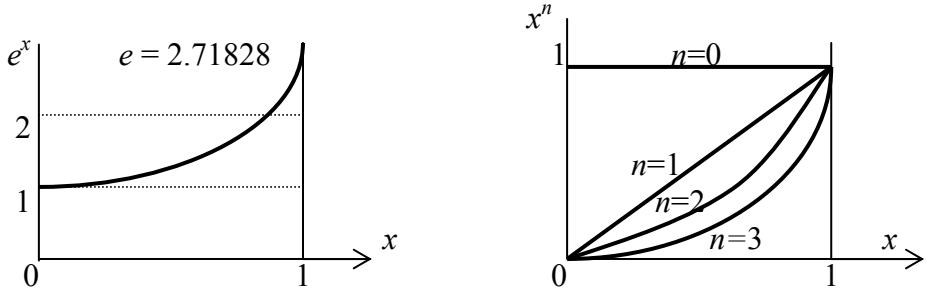
Si $\delta \approx 10^{-11}$ (dígito oculto en la calculadora), el error de $x_{20} \approx \left(\frac{13}{3}\right)^{19} \cdot 10^{-11} \approx 10$.

Otros errores adicionales, del término de $\frac{4}{3}$, o surgidos al calcular x_2, x_3, ... se suman al generado en x_1 y también llegan multiplicados a x_{20}.

Ejemplo 2. Evaluación de la sucesión y_n dada por

$$y_n = \int_0^1 x^n e^x dx \qquad \text{con} \quad n \geq 0$$

Representamos, en primer lugar, las funciones e^x y x^n, de manera aproximada en el intervalo de integración



Vemos, de estas gráficas, que la integral disminuye conforme aumenta $n \Rightarrow$ $y_1 > y_2 > ... > y_n$ y que el $\lim_{n \to \infty} y_n = 0$.

Para evaluar estas integrales vamos a generar una relación de recurrencia, y para ello la integramos por partes:

$$y_n = \int_0^1 x^n e^x dx = \left[x^n e^x\right]_0^1 - \int_0^1 n\, x^{n-1} e^x dx = e - n \underbrace{\boxed{\int_0^1 x^{n-1} e^x dx}}_{\Downarrow\ y_n}$$

donde hemos utilizado $\begin{cases} u = x^n & \Rightarrow \quad du = n\, x^{n-1} dx \\ dv = e^x dx & \Rightarrow \quad v = e^x \end{cases}$

quedándonos finalmente la relación de recurrencia $y_n = e - n\, y_{n-1}$.

$y_0 = e - 1$ (haciendo $n=0$ en la expresión inicial de la integral)

$y_1 = e - y_0 = e - (e-1) = 1$

...

$y_n = e - n\, y_{n-1}$

Utilizando la calculadora podemos generar la siguiente tabla de valores para la sucesión y_n:

$y_1 = 1$
$y_2 = e-2 = 0.\ 718\ 281\ 828$
$y_3 = \ 0.\ 563\ 436\ 344$
$y_4 = \ 0.\ 464\ 536\ 452$
$y_5 = \ 0.\ 395\ 599\ 568$
$y_6 = \ 0.\ 344\ 684\ 420$
$y_7 = \ 0.\ 305\ 490\ 888$
$y_8 = \ 0.\ 274\ 354\ 724$
$y_9 = \ 0.\ 249\ 089\ 312$
$y_{10} = \ 0.\ 227\ 388\ 708$
$y_{11} = \ 0.\ 217\ 006\ 040$
$y_{12} = \ 0.\ 114\ 209\ 348$
$y_{13} = \ 1.\ 233\ 560\ 304$
$y_{14} = -14.\ 551\ 62\ 43$
$y_{15} = \ 220.\ 99\ 17\ 183$

Estos números, lógicamente, no pueden ser correctos. Comienzan disminuyendo, como es de esperar del comportamiento que hemos estudiado inicialmente para la serie. Pero después aumentan (comportamiento no esperado) y pasan de positivos a negativos en un claro signo de inestabilidad. La inestabilidad surge del pequeño error, δ, que se comete al evaluar el número irracional e (con infinitas cifras decimales) en y_2. Al evaluar y_3, este error se multiplica por 3, convirtiéndose en 3δ. Al evaluar y_4, el error se multiplica por 4, convirtiéndose en 12δ, y así sucesivamente como se indica a continuación.

En y_2 hemos cometido un error δ porque e tiene ∞ decimales.

En y_3, el error es $3\,\delta$.

En y_4, el error es $4{\cdot}3\,\delta$.

En y_5, el error es $5{\cdot}4{\cdot}3\,\delta$.

...

En y_n, el error es $\frac{1}{2}n!\ \delta$.

Si estamos trabajando en simple precisión, inicialmente nuestro error está ligado al bit en la última posición de la mantisa (posición 23), por tanto el error δ es aproximadamente

$$\delta \approx 2^{-23} \approx 10^{-7},$$

y por tanto el error para y_{14} es $\frac{1}{2}14!10^{-7} \cong 4000,$

y para y_{20} es $\frac{1}{2}\ 20!10^{-7} \cong 10^{11}$

Tipos básicos de crecimiento de error en un cálculo numérico.

Llamemos:

$\varepsilon \equiv$ error cometido en una etapa

$n \equiv$ número de etapas
$E_n \equiv$ error acumulado después de n etapas

Si $|E_n| = C\, n\, \varepsilon$, con $C =$ una constante independiente de n, el error es lineal.
Si $|E_n| = K^n\, \varepsilon$ con $K > 1$, el crecimiento del error es de tipo exponencial.

El crecimiento lineal suele ser inevitable, el exponencial debe evitarse.

1.3. Condicionamiento: Mal condicionamiento.

La palabra condicionamiento se utiliza para indicar lo sensible que es una solución numérica de un problema a pequeños cambios en los datos de entrada. Un problema está mal condicionado si pequeños cambios en la entrada, pueden dar lugar a grandes cambios en la respuesta. El mal condicionamiento suele ir unido a la no linealidad del sistema, o de la expresión matemática o procedimiento que se está utilizando. Un ejemplo popular de problema mal condicionado es el llamado efecto mariposa. En Física, un problema clásico bien condicionado es la mecánica celeste, comenzada a desarrollar por Newton y completada, entre otros, por Laplace. Las orbitas de los planetas tienen la regularidad de un reloj bien hecho. Aun eventos a largo plazo como eclipses solares o retorno de cometas pueden ser predichos con total exactitud. Esta filosofía del "buen comportamiento" se extendió como posible por décadas a todos los campos de la Física. Pero a principios de los años 60 del pasado siglo, el metereólogo Edward Lorenz del MIT (Massachusetts Institute of Technology) dijo que el tiempo (el pronostico del metereológico) era intrínsecamente impredecible, y no por su complejidad, sino por el mal condicionamiento de las ecuaciones no lineales que lo gobiernan. Lorenz formuló un modelo simple para la predicción meteorológica reduciendo el problema a un sistema de doce ecuaciones diferenciales no lineales extremadamente sensibles a las condiciones iniciales.

Consideremos algunos ejemplos particulares:

Ejemplo 1.
Supongamos que estamos evaluando la función f en un punto x, y nos preguntamos: Si se perturba ligeramente x, ¿cuál es el efecto sobre $f(x)$? Es decir,

si $x \to x+h$, $f(x+h) \to ?$

Supuesto que el cambio en x es pequeño, podemos recurrir al desarrollo en serie de Taylor y quedarnos con una aproximación de orden 1:

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

El cambio de la función, dado como error absoluto sería:

$$f(x+h) - f(x) = h\, f'(x)$$

El cambio dado en términos de error relativo sería:

$$\frac{f(x+h)-f(x)}{f(x)} = \frac{hf'(x)}{f(x)} = \left[\frac{xf'(x)}{f(x)}\right]\left(\frac{h}{x}\right)$$

Cambio de la función Número de condición Cambio de la variable independiente

Dándonos cuenta que $\frac{h}{x} = \frac{(x+h)-x}{x}$ es el cambio relativo en la variable independiente, resulta que el cambio relativo en la función es proporcional al cambio relativo en la variable a través de un factor $\left[\frac{xf'(x)}{f(x)}\right]$, al que llamamos número de condición. Si el número de condición es alto tenemos un problema mal condicionado.

Aplicamos lo anterior a la evaluación de la función arcoseno:

$$f(x) = \text{arcsen}\, x = \text{sen}^{-1} x \quad \Rightarrow \quad f'(x) = \frac{1}{\sqrt{1-x^2}}$$

y el número de condición tomará la forma:

$$\text{número de condición} = \left[\frac{xf'(x)}{f(x)}\right] = \frac{x}{\left(\sqrt{1-x^2}\right)\text{arcsen}\, x}$$

Para valores de x próximos a 1, $\text{arcsen}\, x \approx \frac{\pi}{2}$ y el número de condición se vuelve infinito conforme x se acerca a 1$\Rightarrow$ Por tanto, pequeños errores relativos en x pueden producir grandes errores relativos en la función arcsen x, cerca de $x = 1$.

Ejemplo 2.

Consideremos ahora el problema de localizar una raíz (un cero) de una función f. Supongamos que hemos encontrado una raíz simple de esta función en el punto $x = r$, de forma que $f(r) = 0$ con $f'(r) \neq 0$. Si perturbamos la función $f(x)$, sumándole $\varepsilon g(x)$, siendo $g(x)$ una función derivable en la vecindad de r, tal como también lo es $f(x)$, ¿dónde se encontrará la nueva raíz de la función perturbada $f(x) + \varepsilon g(x)$?

Construimos la función $F(x) = f(x) + \varepsilon g(x)$, y supongamos que la nueva raíz es $r+h \Rightarrow F(r+h) = 0$.

Haciendo un desarrollo en serie de Taylor hasta primer orden

$$F(r+h) \approx F(r) + h\, F'(r) = f(r) + \varepsilon g(r) + h\,[f'(r) + \varepsilon g'(r)] = 0$$

$$\text{Con } f(r) = 0 \;\Rightarrow\; h = -\varepsilon \frac{g(r)}{f'(r) + \varepsilon g'(r)}$$

$$\text{Y con } \varepsilon \text{ lo suficientemente pequeño, } h \approx \frac{-\varepsilon g(r)}{f'(r)}$$

Aplicamos lo anterior al ejemplo clásico, dado por Wilkinson en 1.984, del polinomio pérfido

$$f(x) = \prod_{k=1}^{20}(x-k) = (x\text{-}1)(x\text{-}2)\ldots(x\text{-}20)$$

con la función $g(x) = x^{20}$.

Obviamente, las raíces de $f(x)$ son los enteros 1, 2, 3, …, 20, y nos preguntamos como se altera la raíz $r = 20$ cuando se perturba a $f(x)$ como $f(x)+\varepsilon\, g(x)$. Con la expresión anterior para h

$$h = -\,\varepsilon\,\frac{g(20)}{f'(20)}$$

Donde sólo queda por determinar la derivada del polinomio pérfido:

$$\begin{aligned} f'(x) = {} & (x\text{-}2)\,(x\text{-}3)\,(x\text{-}4)\ldots(x\text{-}20) + \\ & + (x\text{-}1)\,(x\text{-}3)\,(x\text{-}4)\ldots(x\text{-}20) + \\ & + (x\text{-}1)\,(x\text{-}2)\,(x\text{-}4)\ldots(x\text{-}20) + \\ & \ldots \\ & +(x\text{-}1)\,(x\text{-}2)\,(x\text{-}3)\ldots..(x\text{-}19) = \sum_{i=1}^{20}\prod_{\substack{j=1\\ j\neq i}}^{20}(x-j) \Rightarrow \end{aligned}$$

$f'(20) = 19\cdot 18\cdot 17\ldots 2\cdot 1 = 19!$ (ya que se anulan todos los sumandos menos el último) $\Rightarrow$

$$h = -\varepsilon\,\frac{20^{20}}{19!} \approx -\varepsilon\; 10^{9}$$

pudiéndose producir grandes cambios en la posición de la raíz al perturbar la función, y siendo este un ejemplo de problema mal condicionado.

Ejemplo 3

Consideremos el sistema

$$\begin{pmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2.00 \\ 2.00 \end{pmatrix},$$

resultando evidente que la solución es $x = 1$, $y = 1$.

Supóngase que se modifica ligeramente la matriz columna de los términos independientes, quedando el sistema

$$\begin{pmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2.02 \\ 1.98 \end{pmatrix},$$

la solución obvia es $x = 2$, $y = 0$.

Por último, si los términos independientes vuelven a cambiar en la forma siguiente:

$$\begin{pmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1.98 \\ 2.02 \end{pmatrix},$$

la solución sería $x = 0, y = 2$.

Claramente, este sistema es un ejemplo sencillo de problema o sistema mal condicionado porque pequeños cambios, del orden de un uno por ciento, en los términos independientes, producen relativamente grande cambios en la solución.