analisis_numerico (/github/andressotov/analisis_numerico/tree/main)
/ Errores Numéricos.ipynb (/github/andressotov/analisis_numerico/tree/main/Errores Numéricos.ipynb)

Errores Numéricos

Representación de números

Los sistemas de numeración han progresado desde el uso de dedos y marcas de conteo, quizás hace más de 40.000 años, hasta el uso de conjuntos de glifos o pictogramas capaces de representar cualquier número concebible de manera eficiente. Las notaciones inequívocas más antiguas conocidas para números surgieron en Mesopotamia hace unos 5000 o 6000 años.

La escritura más antigua conocida para el mantenimiento de registros surgió de un sistema de contabilidad que usaba pequeñas fichas de arcilla. Los primeros artefactos son de Tell Abu Hureyra, un sitio en el valle del Alto Éufrates en Siria que data del décimo milenio a. C., y Ganj-i-Dareh Tepe, un sitio en la región de Zagros en Irán que data de el noveno milenio a. C.

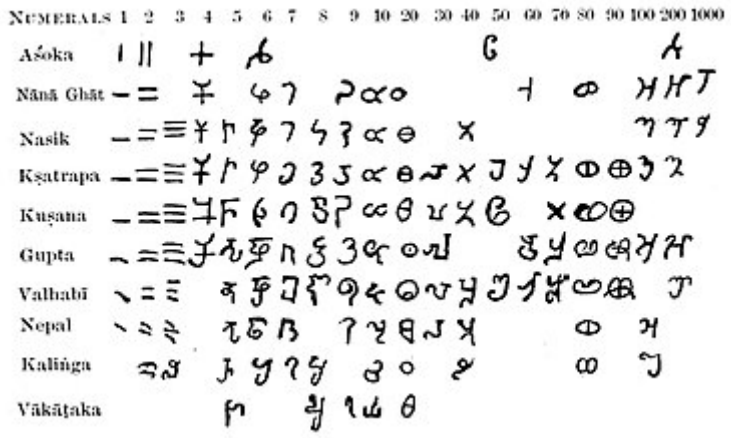
Para crear un registro que representaba "dos ovejas", se usaron dos fichas, cada una de las cuales representaba una unidad. Los diferentes tipos de objetos también se contaron de manera diferente. Dentro del sistema de conteo utilizado con la mayoría de los objetos discretos (incluidos los animales como las ovejas), había una ficha para un elemento (unidades), una ficha diferente para diez elementos (decenas), una ficha diferente para seis decenas (sesenta), etc.

A mediados y finales del cuarto milenio a. C., las impresiones numéricas utilizadas previamente fueron reemplazadas por tablillas numéricas con números proto-cuneiformes impresos en arcilla con un estilete redondo sostenido en diferentes ángulos para producir las diversas formas utilizadas para los signos numéricos. Cada signo numérico representaba tanto la mercancía que se contaba como la cantidad o volumen de esa mercancía. Estos números pronto fueron acompañados por pequeñas imágenes que identificaban el producto enumerado.

Alrededor de 2100 a. C., se desarrolló un sistema numérico sexagesimal común con valor posicional y se usó para ayudar a las conversiones entre sistemas de conteo especificados por objetos.

Los números sexagesimales constituían un sistema de base mixto con bases alternas de 10 y 6 que caracterizaban las fichas, las impresiones numéricas y los signos numéricos protocuneiformes. Los números sexagesimales se usaban en el comercio, así como para cálculos astronómicos y de otro tipo. En números arábigos, la numeración sexagesimal todavía se usa hoy para contar el tiempo (segundo por minuto; minutos por hora) y ángulos (grados).

El sistema numérico hindú-árabe es un sistema numérico de valor posicional decimal que utiliza un glifo cero como en "205". Sus glifos descienden de los números indios Brahmi.



El sistema completo surgió entre los siglos VIII y IX, y se describe por primera vez fuera de la India en On the Calculation with Hindu Numerals de Al-Khwarizmi (ca. 825), y en el segundo trabajo de cuatro volúmenes de Al-Kindi On the Use of the Indian Numerals (ca. 830).

Los números romanos se desarrollaron a partir de símbolos etruscos a mediados del primer milenio a. C. En el sistema etrusco, el símbolo 1 era una única marca vertical, el símbolo 10 eran dos marcas de conteo cruzadas perpendicularmente y el símbolo 100 eran tres marcas de conteo cruzadas (de forma similar a un asterisco moderno *). El 5 (en forma de V invertida) y el 50 (una V invertida dividida por una sola marca vertical) tal vez se derivaron de las mitades inferiores de los signos de 10 y 100. No hay una explicación convincente de cómo el símbolo romano de 100, C, se derivó de su antecedente etrusco en forma de asterisco.

La notación posicional es un sistema de numeración en el cual cada dígito posee un valor que depende de su posición relativa, la cual está determinada por la base, que es el número de dígitos necesarios para escribir cualquier número.

Un ejemplo de numeración posicional es el sistema decimal (base 10), que utiliza diez dígitos diferentes constituidos por un símbolo (grafema), cuyo valor en orden creciente es: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Utilizando la notación posicional, el mismo dígito 5 toma diferente valor en los números 5, 50 y 500.

Por convenio, los dígitos en esta notación se escriben de izquierda a derecha (incluso en idiomas que normalmente escriben de derecha a izquierda), empezando por los órdenes superiores y acabando en la unidad como tal, marcando la carencia de unidades con un 0 (cero). Así, en sistema decimal:

$$505 = 5 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0$$

Si existen órdenes menores que la unidad, se escribe una coma (',') o un punto ('.') según el idioma, para separarlos de las unidades, y se continúa escribiendo de mayor a menor, acabando con las unidades de menor orden.

Otros Sistemas de Numeración

Sistema binario

El sistema binario es un sistema de numeración en el que los números son representados utilizando únicamente dos cifras: 0 (cero) y 1 (uno). Es uno de los sistemas que se utilizan en las computadoras, debido a que estas trabajan internamente con dos niveles de voltaje, por lo cual su sistema de numeración natural es el sistema binario. Un número binario puede ser representado por cualquier secuencia de bits (binary digit).

Los números binarios se escriben a menudo con subíndices, prefijos o sufijos para indicar su base. Las notaciones siguientes son equivalentes:

- 100101 binario (declaración explícita de formato)
- 100101b (un sufijo que indica formato binario)
- 100101_2 (un subíndice que indica base 2, notación binaria)

entre otros.

Conversión entre binario y decimal

Conversión de decimal a binario

Supongamos que se quiere convertir el valor X que corresponde a un número del sistema decimal a sistema binario. Para ello se divide X entre 2, obteniendo un cociente c_1 y un residuo r_1. Si el cociente c_1 es mayor que 2, se divide este por 2, obteniendo así un nuevo cociente c_2 y un residuo r_2. Este proceso se repite hasta que el cociente obtenido c_i sea menor que 2.

A continuación se ordena desde el último cociente hasta el primer resto, simplemente se colocan en orden inverso a como aparecen en la división. Este será el número binario que buscamos.

Ejemplo: Transformar el número decimal 131_{10} en binario.

131 dividido entre 2 da cociente=65 con residuo igual a 1

65 dividido entre 2 da cociente=32 con residuo igual a 1

32 dividido entre 2 da cociente=16 con residuo igual a 0

16 dividido entre 2 da cociente=8 con residuo igual a 0

8 dividido entre 2 da cociente=4 con residuo igual a 0

4 dividido entre 2 da cociente=2 con residuo igual a 0

2 dividido entre 2 da cociente=1 con residuo igual a 0

```
el último cociente es 1
```

$\Rightarrow$ Ordenamos los residuos, del último al primero: 10000011 En sistema binario, 131_{10} se escribe 10000011_2.

¿Y qué hacemos si es un número decimal con parte fraccionaria?

Se convierten de forma independiente la parte entera y la parte fraccionaria. La parte entera se convierte como habíamos visto anteriormente. Luego, se sigue con la parte fraccionaria.

Para la parte fraccionaria, en lugar de dividir por dos como hicimos con la parte entera, lo que se hace es multiplicar por dos la parte fraccionaria. Como resultado de la multiplicación, pueden ocurrir dos cosas:

```
a)el resultado obtenido es mayor o igual a 1

b)el resultado obtenido es menor que 1
```

Si el resultado obtenido es mayor o igual a 1, se anota como un uno (1) binario. Si es menor que 1 se anota como un 0 binario. Por ejemplo, al multiplicar 0.6 por 2 obtenemos como resultado 1.2, lo cual indica que nuestro resultado es un uno (1) en binario, solo se toma la parte entera del resultado.

A continuación, tomamos nuevamente la parte fraccionaria de la multiplicación obtenida y se vuelve a multiplicar por 2, repitiendo el proceso mientras la parte fraccionaria no se anule (i.e. no se haga cero) o que consideremos que tenemos suficientes dígitos calculados.

NOTA algunos valores fraccionarios no tienen una representación exacta al convertirlos a binario, por lo cual tienen una representación donde uno o más dígitos se repiten periodicamente, como por ejemplo, el 0.1.

Después de concluir las multiplicaciones, se colocan los números en el mismo orden de su obtención.

Ejemplo: supongamos que se quiere convertir el número $0,3125_10$ a binario

Proceso:

0,3125 * 2 = 0,625 $\Rightarrow$ 0

0,625 * 2 = 1,25 $\Rightarrow$ 1

0,25 * 2 = 0,5 $\Rightarrow$ 0

0,5 * 2 = 1 $\Rightarrow$ 1

En orden de su obtención: 0101 $\Rightarrow 0,0101_2$

Otro ejemplo: se quiere convertir el número $0,1_10$ a binario

Proceso:

0,1 * 2 = 0,2 $\Rightarrow$ 0

0,2 * 2 = 0,4 $\Rightarrow$ 0

0,4 * 2 = 0,8 $\Rightarrow$ 0

0,8 * 2 = 1,6 $\Rightarrow$ 1

0,6 * 2 = 1,2 $\Rightarrow$ 1

0,2 * 2 = 0,4 $\Rightarrow$ 0 $\leftarrow$ se repiten las últimas cuatro cifras, periódicamente

0,4 * 2 = 0,8 $\Rightarrow$ 0 $\leftarrow$

0,8 * 2 = 1,6 $\Rightarrow$ 1 $\leftarrow$

0,6 * 2 = 1,2 $\Rightarrow$ 1 $\leftarrow ...$

En orden de su obtención: 0 0011 0011 ... $\Rightarrow 0,000110011..._2$

Conversión de binario a decimal

Al igual que la conversión de decimal a binario, la conversión de binario a decimal se subdivide en dos partes:

```
1.Conversión de la parte entera del número

2.Conversión de la parte fraccionaria
```

Para realizar la conversión de binario a decimal de la parte entera del número, realice lo siguiente:

- Comience por el lado derecho del número en binario. Considere que el dígito del extremo derecho ocupa la posición 0 correspondiente a las unidades, el dígito siguiente (siempre de derecha a izquierda) ocupa la posición 1 (que correspondería a las decenas si fuera en decimal), el siguiente por la izquierda la posición 2 y así sucesivamente.
- Multiplique cada dígito por 2 elevado a la potencia correspondiente a la posición que ocupa el dígito. De esta forma, el dígito en la posición i-ésima se multiplica por 2^i
- Después de realizar cada una de las multiplicaciones, súmelas todas y el número resultante será el equivalente en sistema decimal.

Ejemplo convertir el número 110101_2 a decimal

$$110101_2 = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 53$$

La conversión de binario a decimal de la parte fraccionaria del número se realiza de forma similar:

- Comenzando por posición adjunta a la coma (o punto) decimal, numere las posiciones igual que hicimos anteriormente, pero de izquierda a derecha en este caso. Así, la posición adjunta a coma (o punto) decimal sería la posición 1, La siguiente posición de izquierda a derecha sería la 2 y así sucesivamente.
- Multiplique cada dígito por 2 elevado a la potencia negativa correspondiente a la posición que ocupa el dígito. De esta forma, el dígito en la posición i-ésima se multiplica por 2^{-i}

- Después de realizar cada una de las multiplicaciones, súmelas todas y el número resultante será el número correspondiente en sistema decimal.

Ejemplo convertir el número 0.101_2 a decimal

$$0.101_2 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 1 \times 0.5 + 0 \times 0.25 + 1 \times 0.125 = 0.625$$

Sistema octal

El sistema octal es el sistema de numeración posicional cuya base es igual 8, utilizando los dígitos indo-arábigos: 0, 1, 2, 3, 4, 5, 6, 7, donde cada dígito tiene el mismo valor que en el sistema de numeración decimal. Como 8 es una potencia de 2, la conversión a binario o viceversa sea bastante simple. El sistema octal es utilizado como una forma abreviada de representar números binarios. Por otra parte, los cálculos para convertir de decimal a octal y viceversa se reducen con relación a los requeridos ra convertir entre decimal y binario.

A continuación, presentamos una tabla con la conversión de decimal a binario y octal. Observe que cada dígito octal se representa mediante exactamente 3 dígitos binarios.

Decimal	Binario	Octal
0	000000	0
1	000001	1
2	000010	2
3	000011	3
4	000100	4
5	000101	5
6	000110	6
7	000111	7
8	001000	10
9	001001	11
10	001010	12
11	001011	13
12	001100	14
13	001101	15
14	001110	16
15	001111	17
16	010000	20

Para pasar de binario a octal, solo hay que agrupar de 3 en 3 los dígitos binarios comenzando a partir del extremo derecho. Así, el número 1001010_2 (74_{10}), se agrupa como $(1)(001)(010)$. Dado que, en la posición más significativa solo tenemos un dígito, se completa con ceros a la izquierda para completar el trío, de modo que quedaría: $(001)(001)(010)$, lo cual equivale al número octal 112_8. De la misma manera, el número octal 357_8 se convierte en binario como 011101111_2

Sistema hexadecimal

El sistema hexadecimal es el sistema de numeración posicional que tiene como base el valor 16. Se emplea fundamentalmente porque las operaciones de la CPU Central Processing Unit) suelen usar el byte u octeto como unidad básica de memoria. Un byte representa 2^8 valores posibles, que puede representarse como dos dígitos hexadecimales.

$$2^8 = 2^4 \cdot 2^4 = 16 \cdot 16$$

Por otra parte, utilizar el sistema hexadecimal ayuda a reducir la cantidad de dígitos a emplear para expresar una cantidad determinada. Por ejemplo,

$$2^8 = 256_{10} = 1 \cdot 16^2 + 0 \cdot 16^1 + 0 \cdot 16^0 1 \cdot 16^2 + 0 \cdot 16^1 + 0 \cdot 16^0 = 100_{16}$$

,

Dado que el sistema de numeración usual es de base decimal y solo se disponía de diez dígitos, se adoptó la convención de usar las seis primeras letras del alfabeto latino para suplir los dígitos que faltaban. El conjunto de símbolos es el siguiente:

$$S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

,

donde las letras de la A a la F corresponden a los siguientes valores numéricos decimales: $A = 10, B = 11, C = 12, D = 13, E = 14$ y $F = 15$.

Como en cualquier sistema de numeración posicional, el valor numérico de cada dígito es alterado dependiendo de su posición en la cadena de dígitos, quedando multiplicado por una cierta potencia de la base del sistema, que en este caso es 16. Por ejemplo:

$$\begin{aligned} 3E0A_{16} &= 3 \times 16^3 + E \times 16^2 + 0 \times 16^1 + A \times 16^0 \\ &= 3 \times 4096 + 14 \times 256 + 0 \times 16 + 10 \times 1 = 15882_{10} \end{aligned}$$

.

A continuación, presentamos una tabla con la conversión de decimal a binario, hexadecimal y octal.

Decimal	Binario	Hexadecimal	octal
0	000000	0	00
1	000001	1	01
2	000010	2	02
3	000011	3	03
4	000100	4	04
5	000101	5	05
6	000110	6	06
7	000111	7	07
8	001000	8	10

Decimal	Binario	Hexadecimal	octal
9	001001	9	11
10	001010	A	12
11	001011	B	13
12	001100	C	14
13	001101	D	15
14	001110	E	16
15	001111	F	17
16	010000	10	20
17	010001	11	21
18	010010	12	22
19	010011	13	23
20	010100	14	24
21	010101	15	25
22	010110	16	26
23	010111	17	27
24	011000	18	30
25	011001	19	31
26	011010	1A	32
27	011011	1B	33
28	011100	1C	34
29	011101	1D	35
30	011110	1E	36
31	011111	1F	37
32	100000	20	40
33	100001	21	41

El sistema hexadecimal fue introducido en el ámbito de la computación por primera vez por IBM en 1963.

¿Cómo se almacenan los números en el ordenador?

Ya sabemos que los números se almacenan utilizando el sistema binario, pero ¿en dónde se guardan? En la memoria del ordenador, la cual se subdivide en una gran cantidad de celdas denominadas localizaciones. Dichas localizaciones tienen un tamaño fijo determinado por el fabricante del ordenador. Por lo cual, cada localización puede contener un número fijo de bits, i.e., digitos binarios. Por lo cual, una localización de memoria solo puede contener números en un determinado rango de valores. Al contenido de una localización se le denomina palabra y muchas veces se usan ambos términos de forma indiferenciada.

¿Cómo se almacena el signo del número

En la vida cotidiana, los números negativos se representan con un signo menos ('-') delante del número. Pero, en el ordenador esto no es posible. Para representarlo, existen diferentes formas como signo y magnitud, complemento a uno, complemento a dos y exceso K, donde normalmente K equivale a $b^{n-1}-1$. Para la mayoría de usos, las computadoras modernas utilizan típicamente la representación en complemento a dos.

El complemento a dos de un número N , expresado en el sistema binario con n dígitos, se define como:

$$C_2(N) = 2^n - N$$

donde total de números positivos será $2^{n-1}-1$ y el de negativos 2^{n-1}, siendo n el número de bits. El 0 contaría como positivo, ya que los positivos son los que empiezan por 0 y los negativos los que empiezan por 1 .

Veamos un ejemplo:

tomemos el número $N = 45$ expresado en binario es $N = 101101_2$, con 6 dígitos, y calculemos su complemento a dos:

$$N = 45, n = 6 \Rightarrow C_2(N) = 2^n - N = 2^6 - 45 = 64 - 45 = 19 = 010011_2$$

Al representarlo con 6 dígitos el bit más significativo es 0. Este proceso puede parecer muy complicado, pero es muy fácil obtener el complemento a dos de un número a partir de su complemento a uno, porque el complemento a dos de un número binario es igual a su complemento a uno más uno, es decir:

$$C_2^N = C_1^N + 1$$

El complemento a uno de un número binario se define como el valor obtenido al invertir todos los bits en la representación binaria del número (intercambiando los 0 por 1 y los 1 por 0). Calcular el complemento a uno de un número binario es una operación muy sencilla en cualquier ordenador.

De donde:

$$C_1(101101) = 010010 \Rightarrow C_2(45) = C_1(101101) + 1 = 010010 + 000001 = 010011_2$$

En este punto conviene introducir el concepto de desbordamiento (overflow) en aritmética entera. Un desbordamiento de enteros ocurre cuando una operación aritmética intenta crear un valor numérico que está fuera del rango que se puede representar con un número determinado de dígitos, ya sea mayor que el valor máximo o menor que el valor mínimo representable. El resultado más común de un desbordamiento es que solo se almacenen los dígitos menos significativos del resultado. Normalmente, esta es una situación que debe ser detectada mediante el hardware y el software del equipo y tratada apropiadamente indicando un error.

Estándar IEEE 754

La representación de coma (o punto) flotante (floating point) es una forma de notación científica usada en las computadoras con la cual se pueden representar números reales extremadamente grandes y pequeños de una manera muy eficiente y compacta y con la que

se pueden realizar operaciones aritméticas. La notación científica, también denominada notación exponencial, es una forma de organizar o escribir los números basada en potencias de 10.

- masa del electrón

$$0.00000000000000000000000000000911kg = 9.11 \times 10^{-31} kg$$

- mayor distancia observable del universo:

$$74000000000000000000000000m = 7.4 \times 10^{26} m$$

El estándar actual para la representación en coma flotante es el IEEE 754, un estándar técnico para la aritmética de coma flotante establecido en 1985 por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE). El estándar abordó muchos de los problemas encontrados en su momento en las diversas implementaciones de punto flotante. Muchas unidades de punto flotante de hardware utilizan el estándar IEEE 754. La versión actual, IEEE 754-2019, se publicó en julio de 2019. Es una revisión menor de la versión anterior, que incorpora principalmente aclaraciones, correcciones de defectos y nuevas operaciones recomendadas.

El estándar IEEE 754 define:

- formatos aritméticos: conjuntos de datos binarios y decimales de coma flotante, que consisten en números finitos (incluidos ceros con signo y números subnormales), infinitos y valores especiales (NaN: Not a Number)
- reglas de redondeo: propiedades que deben cumplirse al redondear números durante la aritmética y las conversiones
- operaciones: aritmética y otras operaciones (como funciones trigonométricas) en formatos aritméticos
- manejo de excepciones: indicaciones de condiciones excepcionales (como división por cero, desbordamiento, etc.)

Un formato de coma flotante se especifica mediante

- una base (también llamada raíz) b, que es 2 (binario) o 10 (decimal) en IEEE 754;
- una precisión p;
- un rango de exponente de emin a emax, con emin = 1 − emax para todos los formatos IEEE 754.

Un formato comprende

- Números finitos, que pueden describirse mediante tres números enteros:
  - $s =$ un signo (cero o uno),
  - $c =$ una mantisa (o coeficiente) que no tiene más de p dígitos cuando se escribe en base b (es decir, un número entero en el rango hasta 0 a $b^p - 1$), y
  - $q =$ un exponente tal que $e_{min} \leq q + p - 1 \leq e_{max}$.

El valor numérico de dicho número finito es $(-1)^s \times c \times b^q$.

Además, hay

- dos valores cero, llamados ceros con signo: el bit de signo especifica si un cero es +0 (cero positivo) o − 0 (cero negativo).
- dos infinitos: +∞ y −∞.
- dos tipos de NaN: un NaN silencioso (qNaN) y un NaN de señalización (sNaN).

Por ejemplo, si $b = 10$, $p = 7$ y $e_{max} = 96$, entonces $e_{min} = -95$, la mantisa satisface $0 \leq c \leq 9999999$ y el exponente satisface $-101 \leq q \leq 90$.

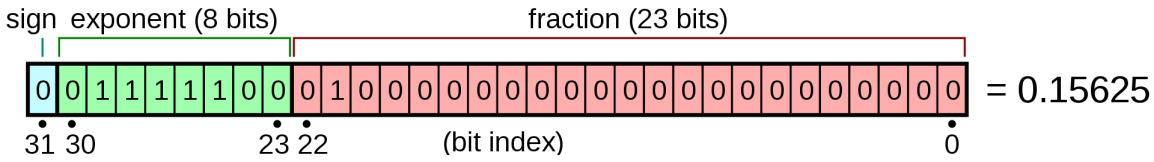
En consecuencia, el menor número positivo distinto de cero que se puede representar es 1×10^{-101}, y el mayor es $9999999 \times 10^{90} (9,999999 \times 10^{96})$, por lo que el rango completo de números es $-9,999999 \times 10^{96}$ a $9,999999 \times 10^{96}$.

Los números $-b^{1-e_{max}}$ y $b^{1-e_{max}}$ (aquí, -1×10^{-95} y 1×10^{-95}) son los números normales más pequeños (en magnitud). Los números distintos de cero entre estos números más pequeños se denominan números subnormales.

Los números pueden tener diversas representaciones posibles en formato exponencial. Por ejemplo, si $b = 10$ y $p = 7$, entonces $-12,345$ se puede representar mediante -12345×10^{-3}, -123450×10^{-4} y -1234500×10^{-5}.

Para los números binarios, se utiliza una única representación, en la cual se elige el exponente más pequeño. Para los números con un exponente en el rango normal (no todos unos o todos ceros), el bit inicial del significando siempre será 1. En consecuencia, el bit 1 principal puede ser implícito en lugar de estar explícitamente presente en la codificación de la memoria. Esta regla se denomina convención de bit principal, o también convención de bits implícita o convención de bits ocultos. La regla permite que el formato de memoria tenga un poco más de precisión. La convención de bit principal no se utiliza para los números subnormales ya que tienen un exponente fuera del rango del exponente normal.

El estándar IEEE 754 define cinco formatos básicos que se denominan por su base numérica y el número de bits utilizados en su codificación de intercambio. Existen tres formatos básicos binarios de punto flotante (codificados con 32, 64 o 128 bits) y dos formatos básicos de punto flotante decimal (codificados con 64 o 128 bits). Los formatos binary32 y binary64 son los formatos simple y doble del estándar original IEEE 754-1985. Una implementación conforme debe implementar completamente al menos uno de los formatos básicos.



El formato en coma flotante de simple precisión es un formato de número de computador u ordenador que ocupa 4 bytes (32 bits) en su memoria y representa un amplio rango dinámico de valores mediante el uso de la coma flotante. En la norma o estándar IEEE 754-

2008, el formato de 32 bits de base 2 se conoce oficialmente como binary32.

El estándar IEEE 754 especifica que un formato binary32 consta de:4

- Bits de signo (S): 1 bit.
- Exponente desplazado (E): 8 bits.
- Significando o Mantisa (T): 24 bits (23 almacenados explícitamente).

Este formato proporciona una precisión de 6 a 9 dígitos decimales significativos. Si una cadena decimal de hasta 6 dígitos decimales significativos se convierte en formato IEEE 754 de precisión simple y luego se convierte de nuevo al mismo número de dígitos decimales significativos, la cadena final debe coincidir con el original y si un número de precisión simple IEEE 754 se convierte en una cadena decimal con al menos 9 dígitos decimales significativos y luego se convierte de nuevo a un número de precisión simple, entonces el número final debe coincidir con el original.

El bit de signo (S) determina el signo del número, que también es el signo de la mantisa o significando. El exponente (E) es, tanto un número entero de 8 bits con signo en el rango de -126 a 127 (expresado en la forma de complemento a 2) como un entero sin signo (e) de 8 bits comprendido de 0 a 255 que es la forma sesgada aceptada en la definición del formato binary32 de IEEE 754. Si se utiliza el formato de número entero sin signo, el valor del exponente utilizado en la aritmética es el exponente desplazado por un sesgo (E).

Para el caso del formato binary32, un valor de exponente desplazado de 127 representa el cero real (es decir, para que 2^{e-127} sea uno, e debe valer 127). El exponente desplazado $(E = e - 127)$ abarca desde −126 hasta +127 ya que los valores de −127 (todos ceros) y 128 (todos unos) son reservados para números especiales.

La mantisa real del formato incluye 23 bits de la fracción a la derecha de la coma binaria y un bit de encabezado implícito (a la izquierda de la coma binaria) con valor de "1" a menos que el exponente se almacene con sus bits en cero. Por lo tanto sólo 23 bits de la mantisa aparecen en el formato de la memoria, pero la precisión total es de 24 bits (equivalente a $log_{10}(2^{24})$ dígitos ≈ 7225 decimales).

Errores numéricos

El concepto de error es intrínseco al cálculo numérico. En todos los problemas es fundamental poder determinar y cuantificar los errores cometidos, a fin de poder estimar el grado de aproximación de la solución que se obtiene.

Los errores asociados a los cálculos numéricos tienen su origen en dos grandes factores:

- Aquellos que son inherentes a la formulación del problema.
- Los que son consecuencia del método empleado para encontrar la solución del problema.

Dentro del grupo de los primeros, se incluyen aquellos en los que la definición matemática del problema es sólo una aproximación a la situación física real. Estos errores son normalmente despreciables; por ejemplo, el que se comete al obviar los efectos relativistas en la solución de un problema de mecánica clásica. En aquellos casos en que estos errores no son realmente despreciables, nuestra solución será poco precisa independientemente de la precisión empleada para encontrar las soluciones numéricas. Otra fuente de este tipo de errores tiene su origen en la imprecisión de los datos físicos: constantes físicas y datos empíricos. En el caso de errores en la medida de los datos empíricos y teniendo en cuenta su carácter generalmente aleatorio, su tratamiento analítico es especialmente complejo pero imprescindible para contrastar el resultado obtenido computacionalmente.

En lo que se refiere al segundo tipo de error (error computacional), existen tres fuentes principales:

1. Equivocaciones en la realización de las operaciones. Esta fuente de error es bien conocida por cualquiera que haya realizado cálculos manualmente o empleado una calculadora. El empleo de ordenadores ha reducido enormemente la frecuencia de estos errores. Sin embargo, no es despreciable la probabilidad de que el programador cometa uno de estos errores (calculando correctamente el resultado erróneo). Más aún, la presencia de bugs no detectados en el compilador o en el software del sistema no es inusual. Cuando no resulta posible verificar que la solución calculada es razonablemente correcta, la probabilidad de que se haya cometido un error de estos no puede ser ignorada. Sin embargo, no es esta la fuente de error que más nos preocupa en estos momentos.

2. El error causado por el uso de algún tipo de aproximación para resolver el problema debido a la sustitución de un proceso infinito (sumatoria o integración) o un infinitesimal (diferenciación) por una aproximación finita. Algunos ejemplos son:

   El cálculo de una función elemental (por ejemplo, $Seno(x)$) empleando sólo n términos de los infinitos que constituyen la expansión en serie de Taylor.

   Aproximación de la integral de una función por una suma finita de los valores de la función, como la empleada en la regla de los trapecios.

   Resolución de una ecuación diferencial reemplazando las derivadas por una aproximación (diferencias finitas).

   Solución de la ecuación $f(x) = 0$ por un proceso iterativo que, en general, converge sólo cuando el número de iteraciones tiende a infinito como el método de Newton-Raphson.

   Este tipo de error se denomina error por truncamiento, ya que resulta de truncar un proceso infinito utilizando en su lugar un proceso finito. Obviamente, es importante poder estimar y/o acotar, este tipo de error en los procedimientos numéricos.

3. La otra fuente de error de importancia ocurre como producto de que los cálculos aritméticos no pueden realizarse con precisión ilimitada. Muchos números requieren infinitos dígitos (i.e., números periódicos) para su representación, sin embargo, en la

práctica no se puede operar con todos ellos, especialmente por el uso de los ordenadores y su capacidad de almacenamiento finita. Por ello, es necesario redondearlos o truncarlos.

Incluso en el caso en que un número pueda representarse de manera exacta, al realizar cálculos con el mismo puede dar lugar a otros números que no tengan representación exacta provocando la aparición de errores. Por ejemplo, las divisiones pueden producir números que deben ser redondeados y las multiplicaciones pueden dar lugar a más dígitos de los que se pueden almacenar. El error que se introduce al redondear un número se denomina error de redondeo.

Veamos algunas definiciones. Generalmente, no conocemos el valor de una cierta magnitud $\bar{x}$ y hemos de conformarnos con un valor aproximado x. Para estimar la magnitud de este error necesitamos dos definiciones básicas:

Error absoluto de x viene dado por el valor absoluto de la diferencia entre el valor real y el valor aproximado:

$$e_a(x) = |x - \bar{x}|$$

Error relativo de x (si x es diferente de cero):

$$e_r(x) = \frac{e_a(x)}{\bar{x}}; (\bar{x} \neq 0)$$

En la práctica, se emplea la expresión:

$$e_r(x) = \frac{e_a(x)}{x}; (x \neq 0)$$

En general, no conocemos el valor de este error, ya que no es habitual disponer del valor exacto de la magnitud, sino sólo de una acotación de su valor, esto es, un número $\varepsilon(x)$, tal que:

$$|e_a(x)| \leq \varepsilon_a(x)$$

o bien:

$$|e_r(x)| \leq \varepsilon_r(x)$$

De acuerdo con lo anterior, un numero se representará del siguiente modo:

$$\bar{x} = x \pm \varepsilon_a(x)$$
$$\bar{x} = x(1 \pm \varepsilon_r(x))$$

Sea x un número real con una representación periódica. Diremos que x está adecuadamente redondeado a un número x_d con d decimales, si el error de redondeo, ε es tal que:

$$|\varepsilon| = |x - x_d| \leq \frac{1}{2} \cdot 10^{-d}$$

Ejemplo: sea $x = 3,141592, x_d = 3,14, d = 2$ entonces

$$|x - x_d| = 0,001592 < \frac{1}{2} \cdot 10^{-2} = 0,005$$

Otra forma de acotar el número de cifras significativas es eliminando los dígitos de orden inferior. El error en est caso se denomina error de truncamiento y se puede medir mediante la siguiente fórmula:

$$|\varepsilon| = |x - x_d| \leq 1 \cdot 10^{-d}$$

Este método produce, por tanto, peores resultados que el método anterior.

Error de propagación.

El error de representación o de redondeo afecta a las operaciones aritméticas que se realicen con el ordenador. El resultado de cada operación se redondea de acuerdo con el número máximo de dígitos que puede representar el ordenador. De esta forma, en una sucesión de operaciones se pueden ir produciendo sucesivos errores, que se van arrastrando (propagando), lo que ocasiona una pérdida general de exactitud, puesto que se van perdiendo las cifras desechadas en cada paso.

Ejemplo: calcular la expresión $((1 + 10^{-20} - 1) \cdot 10^{20})$. Esta expresión, operada en el orden que se indica, da en la mayoría de los ordenadores como resultado 0, en contraste con su resultado correcto que es 1.

In [1]:
```
x = ((1 + 1e-20 - 1) * 1e20)
x
```

Out[1]:
```
0.0
```

In [2]:
```
# Las características del tipo float en nuestra ordenador se obtienen con:
import sys

sys.float_info
```

Out[2]:
```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.225
0738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.
220446049250313e-16, radix=2, rounds=1)
```

donde:

- epsilon: diferencia entre 1.0 y el menor valor mayor que 1.0 que es representable como flotante.
- dig: número máximo de dígitos decimales que se pueden representar fielmente en un flotante.
- mant_dig: precisión de flotantes: el número de dígitos base-radix en el significando de un flotante.
- max: máximo punto flotante positivo representable.
- max_exp: entero máximo e tal que radix**(e-1) es un flotante finito representable.
- max_10_exp: entero máximo e tal que 10**e está en el rango de flotantes finitos representables.
- min: flotante normalizado mínimo representable positivo.

- min_exp: entero mínimo e tal que radix**(e-1) es un flotante normalizado.
- min_10_exp: entero mínimo e tal que 10**e es un valor flotante normalizado.
- radix: base de representación de exponente.
- rounds: entero que representa el modo de redondeo para la aritmética de coma flotante.

Análisis numérico

La ciencia y la tecnología describen los fenómenos reales mediante modelos matemáticos. El estudio de estos modelos permite un conocimiento más profundo del fenómeno, así como de su evolución. La matemática aplicada es la rama de las matemáticas que se dedica a buscar y aplicar las herramientas más adecuadas a los problemas basados en estos modelos. Desafortunadamente, no siempre es posible aplicar métodos analíticos clásicos por diferentes razones:

- No se adecúan al modelo concreto.
- Su aplicación resulta excesivamente compleja.
- La solución formal es tan complicada que hace imposible cualquier interpretación posterior.
- Simplemente no existen métodos analíticos capaces de proporcionar soluciones al problema.

En estos casos son útiles las técnicas numéricas, que mediante una labor de cálculo más o menos intensa, conducen a soluciones aproximadas que son siempre numéricas. El importante esfuerzo de cálculo que implica la mayoría de estos métodos hace que su uso esté íntimamente ligado al empleo de ordenadores. De hecho, sin el desarrollo que se ha producido en el campo de la informática resultaría difícilmente imaginable el nivel actual de utilización de las técnicas numéricas en ámbitos cada día más diversos.

Los métodos numéricos son procedimientos matemáticos cuyo objetivo es la resolución numérica de problemas que carecen de expresión analítica para su resolución exacta. Estos procedimientos se expresan, en general, mediante algoritmos que especifican la secuencia de operaciones lógicas y aritméticas que conducen a la solución (normalmente aproximada) del problema planteado.

Los métodos numéricos se implementan usualmente mediante lenguajes de programación para su ejecución en un sistema computacional. Este sistema puede estar formado por un único ordenador, o por múltiples ordenadores conectados en red. Cualquiera que sea el caso, los sistemas computacionales tienen limitaciones inherentes que deben ser tenidas en cuenta en el diseño de los algoritmos.

Por ejemplo, los números irracionales como $\sqrt{2}, \pi, e$, etc., tienen un desarrollo de infinitas cifras decimales, por lo que no pueden representarse exactamente en la memoria de un ordenador.

Bibliografía

- History of ancient numeral systems (https://en.wikipedia.org/wiki/History_of_ancient_numeral_systems)
- History of the Hindu–Arabic numeral system (https://en.wikipedia.org/wiki/History_of_the_Hindu%E2%80%93Arabic_numeral_system)
- Las representaciones de los números. Un acercamiento antropológico (https://ciencia.unam.mx/leer/38/Las_representaciones_de_los_numeros_Un_acercamiento_a
- Etruscan numerals (https://en.wikipedia.org/wiki/Etruscan_numerals)
- Brahmi numerals (https://en.wikipedia.org/wiki/Brahmi_numerals)
- Notación posicional (https://es.wikipedia.org/wiki/Notaci%C3%B3n_posicional)
- Sistema binario (https://es.wikipedia.org/wiki/Sistema_binario)
- Representación de números con signo (https://es.wikipedia.org/wiki/Representaci%C3%B3n_de_n%C3%BAmeros_con_signo)
- Complemento a dos (https://es.wikipedia.org/wiki/Complemento_a_dos)
- Complemento a uno (https://es.wikipedia.org/wiki/Complemento_a_uno)
- Integer overflow (https://en.wikipedia.org/wiki/Integer_overflow)
- Coma flotante (https://es.wikipedia.org/wiki/Coma_flotante)
- Notación científica (https://es.wikipedia.org/wiki/Notaci%C3%B3n_cient%C3%ADfica)
- Standard IEEE 754 (https://en.wikipedia.org/wiki/IEEE_754)
- Estándar IEEE 754 (https://es.wikipedia.org/wiki/IEEE_754)
- Arithmetic underflow (https://en.wikipedia.org/wiki/Arithmetic_underflow)
- Formato en coma flotante de simple precisión (https://es.wikipedia.org/wiki/Formato_en_coma_flotante_de_simple_precisi%C3%B3n)
- Formato en coma flotante de doble precisión. (https://es.wikipedia.org/wiki/Formato_en_coma_flotante_de_doble_precisi%C3%B3n)
- Error de aproximación. (https://es.wikipedia.org/wiki/Error_de_aproximaci%C3%B3n)
- Métodos Numéricos 1: fuentes de error. (https://estadistica-dma.ulpgc.es/FCC/05-1-Generalidades-Metodos-Numericos.html#error-de-representacion-o-de-redondeo)
- Errores. (https://www.uv.es/diaz/mn/node2.html)
- Número decimal periódico. (https://es.wikipedia.org/wiki/N%C3%BAmero_decimal_peri%C3%B3dico#:~:text=Un%20n%

- Approximation error. (https://en.wikipedia.org/wiki/Approximation_error)
- Round-off error. (https://en.wikipedia.org/wiki/Round-off_error)
- sys Parámetros y funciones específicos del sistema. (https://docs.python.org/es/3.10/library/sys.html)
- Aritmética finita y fuentes del error (https://www.unioviedo.es/compnum/laboratorios_py/Aritmetica_finita/Aritmetica_finita_y_er

In [ ]: