5.ª edición

MÉTODOS NUMÉRICOS

Teoría, problemas y prácticas con MATLAB



Juan Antonio Infante del Río
José María Rey Cabezas

PIRÁMIDE

5.ª edición

MÉTODOS NUMÉRICOS

Teoría, problemas y prácticas con MATLAB

JUAN ANTONIO INFANTE DEL RÍO
PROFESOR TITULAR DEL DEPARTAMENTO DE ANÁLISIS MATEMÁTICO
Y MATEMÁTICA APLICADA EN LA FACULTAD DE CIENCIAS MATEMÁTICAS
DE LA UNIVERSIDAD COMPLUTENSE DE MADRID

JOSÉ MARÍA REY CABEZAS
PROFESOR TITULAR DEL DEPARTAMENTO DE ANÁLISIS MATEMÁTICO
Y MATEMÁTICA APLICADA EN LA FACULTAD DE CIENCIAS QUÍMICAS
DE LA UNIVERSIDAD COMPLUTENSE DE MADRID

5.ª edición

MÉTODOS NUMÉRICOS

Teoría, problemas y prácticas con MATLAB

EDICIONES PIRÁMIDE

COLECCIÓN «CIENCIA Y TÉCNICA»

Edición en versión digital

Está prohibida la reproducción total o parcial de este libro electrónico, su transmisión, su descarga, su descompilación, su tratamiento informático, su almacenamiento o introducción en cualquier sistema de repositorio y recuperación, en cualquier forma o por cualquier medio, ya sea electrónico, mecánico, conocido o por inventar, sin el permiso expreso escrito de los titulares del copyright.

© Juan Antonio Infante del Río y José María Rey Cabezas, 2018
© Segunda edición electrónica publicada por Ediciones Pirámide (Grupo Anaya, S. A.), 2018
Para cualquier información pueden dirigirse a piramide_legal@anaya.es
Juan Ignacio Luca de Tena, 15. 28027 Madrid
Teléfono: 91 393 89 89
www.edicionespiramide.es
ISBN digital: 978-84-368-3984-5

A Julián Infante Lopesino,
Julián Infante del Río,
Carmen del Río Martín,
María Cabezas Pérez
y Julio Rey Vallejo.

In memoriam.

Índice

Prefacio 13

1. Análisis de errores 17
1.1. Introducción 17
1.2. Números máquina 19
1.3. Desbordamiento y redondeo 26
1.4. Aritmética en coma flotante 28
1.5. Propagación del error 33
1.5.1. Condicionamiento 34
1.5.2. Estabilidad 38
1.6. Problemas 44
1.6.1. Problemas resueltos 44
1.6.2. Problemas propuestos 48
1.7. Prácticas 49

2. Complementos de álgebra matricial 53
2.1. Introducción 53
2.2. Diversos tipos de matrices y propiedades 53
2.3. Normas matriciales 69
2.4. Convergencia de las iteraciones de una matriz 83
2.5. Problemas 87
2.5.1. Problemas resueltos 87
2.5.2. Problemas propuestos 101
2.6. Prácticas 104

3. Condicionamiento de un sistema lineal 105
3.1. Introducción 105
3.2. Condicionamiento de una matriz y de un sistema lineal 105
3.3. Problemas 111
3.3.1. Problemas resueltos 111
3.3.2. Problemas propuestos 112
3.4. Prácticas 113

© Ediciones Pirámide

4. Resolución de sistemas lineales: métodos directos 115
4.1. Introducción 115
4.2. Sistemas diagonales y triangulares 116
4.3. Eliminación gaussiana 118
4.3.1. Ejemplo para la formalización del método de Gauss 118
4.3.2. Estudio general del método de Gauss 121
4.3.3. Factorización PA=LU 130
4.3.4. Implementación del método de eliminación gaussiana 134
4.4. Factorización LU de una matriz 137
4.5. Método de Cholesky 144
4.6. Problemas 152
4.6.1. Problemas resueltos 152
4.6.2. Problemas propuestos 177
4.7. Prácticas 180

5. Resolución de sistemas lineales: métodos iterativos 183
5.1. Introducción 183
5.2. Estudio general 184
5.3. Métodos de Jacobi, Gauss–Seidel y relajación 187
5.3.1. Método de Jacobi 188
5.3.2. Método de Gauss–Seidel 189
5.3.3. Método de relajación 191
5.3.4. Métodos por bloques 195
5.4. Resultados de convergencia 200
5.5. Test de parada de las iteraciones 204
5.6. Problemas 207
5.6.1. Problemas resueltos 207
5.6.2. Problemas propuestos 230
5.7. Prácticas 236

6. Interpolación numérica 239
6.1. Introducción 239
6.2. Interpolación de Lagrange 240
6.2.1. El error de interpolación 244
6.2.2. Fórmula de interpolación de Newton 249
6.2.3. Minimización del error 258
6.3. Interpolación mediante funciones spline 266
6.3.1. Método de cálculo de las funciones spline cúbicas 267
6.3.2. Convergencia en la interpolación por funciones spline 276
6.4. Problemas 277
6.4.1. Problemas resueltos 277
6.4.2. Problemas propuestos 299
6.5. Prácticas 300

© Ediciones Pirámide

7. Diferenciación e integración numéricas 301
7.1. Introducción 301
7.2. Diferenciación numérica 301
7.2.1. El error en la diferenciación numérica 303
7.2.2. Ejemplos de fórmulas de derivación 305
7.3. Integración numérica 306
7.3.1. Fórmulas de Newton–Côtes 308
7.3.2. Fórmulas de integración compuesta 321
7.3.3. Fórmulas de cuadratura de Gauss 325
7.4. Problemas 333
7.4.1. Problemas resueltos 333
7.4.2. Problemas propuestos 344
7.5. Prácticas 345

8. Resolución de ecuaciones no lineales 347
8.1. Introducción 347
8.2. Método de la bisección 350
8.3. Métodos de punto fijo 352
8.4. Método de Newton 363
8.5. Variantes del método de Newton 371
8.5.1. Método de Whittaker 371
8.5.2. Método de las cuerdas 375
8.5.3. Método de la secante 379
8.5.4. Método de la Falsa Posición (o Regula Falsi) 381
8.6. Consideraciones finales 382
8.6.1. Test de parada de las iteraciones 382
8.6.2. Raíces múltiples 384
8.7. Problemas 387
8.7.1. Problemas resueltos 387
8.7.2. Problemas propuestos 419
8.8. Prácticas 421

9. Resolución de sistemas no lineales 423
9.1. Introducción 423
9.2. Método de Newton 423
9.2.1. Método de Newton–Jacobi de m pasos 427
9.2.2. Método de Newton–relajación de m pasos 428
9.3. Generalización de métodos lineales 431
9.3.1. Método de Jacobi no lineal 432
9.3.2. Método de Gauss–Seidel no lineal 433
9.3.3. Método de relajación no lineal 434
9.4. Prácticas 435

© Ediciones Pirámide

10. Cálculo de raíces de polinomios 437
10.1. Introducción 437
10.2. Algunas propiedades de los polinomios 438
10.3. Algoritmo de Horner 441
10.4. Métodos de acotación de raíces 442
10.5. Separación de raíces reales 446
10.5.1. Regla de los signos de Descartes 446
10.5.2. Método de Sturm 447
10.6. Ecuaciones con coeficientes racionales 452
10.7. Proceso de separación y cálculo de raíces reales de un polinomio 456
10.8. Raíces complejas: método de Bairstow 462
10.9. Problemas 467
10.9.1. Problemas resueltos 467
10.9.2. Problemas propuestos 483
10.10. Prácticas 485

11. Apéndice: Introducción al programa MATLAB 487
11.1. Generalidades 488
11.2. Vectores y matrices 490
11.3. Operaciones con vectores y matrices 494
11.4. Variables lógicas 497
11.5. Polinomios 497
11.6. Funciones matemáticas 498
11.7. Derivadas y primitivas 499
11.8. Gráficas de funciones 499
11.9. Programación con MATLAB 505

Bibliografía básica 509

Bibliografía de consulta 511

Índice alfabético 513

© Ediciones Pirámide

Prefacio

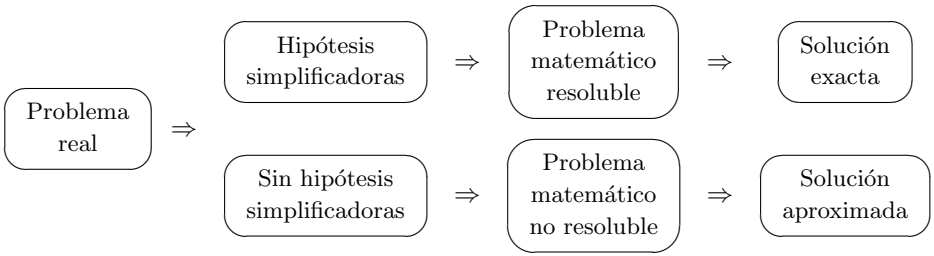
P. Henrici da una definición aproximada del Análisis Numérico como "*la teoría de los métodos constructivos en Análisis Matemático*", haciendo un especial énfasis en la palabra "constructivos". Durante mucho tiempo, las matemáticas fueron totalmente constructivas, pues su único objetivo era llegar a la solución de problemas concretos. No obstante, a medida que los problemas sujetos a la investigación matemática crecían en alcance y generalidad, los matemáticos fueron interesándose, cada vez más, por cuestiones como la *existencia*, *unicidad* y *propiedades cualitativas* de la solución, antes que por su construcción. Una de las causas que condujeron a esta situación fue la escasa capacidad de cálculo que hacía inútil el diseño de algoritmos constructivos de la solución de problemas complejos. No obstante, cuando parecía que las matemáticas habían olvidado cualquier matiz constructivo, surgieron los primeros *ordenadores*, los cuales devolvieron a los matemáticos la esperanza de poder construir las soluciones de los problemas. Fue entonces cuando nació lo que hoy denominamos *Análisis Numérico*. Aunque muchas de las ideas básicas en que se apoyan las técnicas numéricas actuales se conocen desde hace tiempo, ha sido la capacidad de cálculo aportada por los ordenadores (vertiginosamente acrecentada con el transcurso del tiempo) la que les ha dado mayor vigencia e importancia.

El Análisis Numérico es una herramienta fundamental en el campo de las ciencias aplicadas que trata de diseñar métodos que aproximen, de forma eficiente, las soluciones de problemas prácticos previamente formulados matemáticamente. En la mayoría de los casos, el problema matemático se deriva de un problema práctico en áreas experimentales como la Física, Química, Biología, Economía... Sobre él se aplican, típicamente, dos tipos de estrategias generales:

a) Se dan por supuestas algunas hipótesis de carácter simplificador que permiten llegar a una formulación matemática resoluble. (Así es como se procedió tradicionalmente, hasta que se contó con las técnicas numéricas.)

b) Se prescinde de alguna de estas hipótesis para llegar a una formulación matemática más complicada, que no se puede resolver explícitamente, pero cuya solución puede calcularse de forma aproximada.

© Ediciones Pirámide



Aunque de ninguna de las dos formas anteriores obtenemos la solución del problema original, a menudo resulta más apropiado utilizar la segunda. Para ello, se idea un *algoritmo*[1], es decir, una secuencia finita de operaciones algebraicas y lógicas, que se espera produzca una solución aproximada del problema matemático y, en consecuencia, del físico. La confirmación de esta esperanza es, precisamente, una de las principales tareas del Análisis Numérico. En otras palabras, el cometido de la disciplina que nos ocupa es el diseño de métodos que conduzcan (y no solamente de manera teórica) a la solución de los problemas planteados: estudiar los algoritmos en profundidad para que se pueda saber de antemano cuáles son sus ventajas e inconvenientes, qué dificultades presentan a la hora de llevar a cabo su programación efectiva y seleccionar, en cada caso, el algoritmo más *eficiente* en cuanto al almacenamiento de datos y al tiempo de cómputo. En definitiva, proporcionar métodos que permitan obtener realmente una aproximación de la solución buscada y conocer, en la medida de lo posible, el grado de aproximación entre la solución hallada y la real, es decir, dar una *estimación* del *error* cometido.

Este libro está escrito tras años de experiencia de los autores en la docencia de asignaturas relacionadas con el Análisis Numérico. En particular, de la asignatura *Métodos Numéricos* de los grados impartidos en la Facultad de Ciencias Matemáticas de la Universidad Complutense de Madrid. Asignaturas con similares contenidos forman parte del currículum de las titulaciones de Matemáticas, así como de estudios de Ingeniería, Ciencias experimentales, Economía...

El objetivo de este manual es modesto, pero no por ello carente de importancia. Por una parte, pretendemos proporcionar al alumno una primera toma de contacto con las técnicas numéricas que le sirva para conocer un amplio catálogo de métodos que aproximan las soluciones de los problemas abordados (esencialmente, ecuaciones y sistemas lineales y no lineales, interpolación, derivación e integración). Por otra, se intenta cimentar una sólida base teórica que permita conocer los límites de validez y condiciones de aplicación de los métodos, así como el ulterior estudio en profundidad de otras técnicas más sofisticadas.

[1]La palabra algoritmo proviene del matemático persa *Abu Jafar Mohammed ibn Musa al–Khowarizmi*, que vivió en Bagdad hacia el año 840 d.C.

© Ediciones Pirámide

Cada capítulo tiene una sección de problemas de los que, alrededor de la mitad, se resuelven con todo detalle; pensamos que la escasez de textos que incluyan una buena cantidad de problemas resueltos en su totalidad, puede ser un valor añadido de esta obra. En algunos problemas se recogen resultados complementarios a los expuestos en la parte teórica, enunciados de forma que su resolución sea abordable.

Finalmente, destacar que en la última sección de cada capítulo se enuncian una serie de prácticas de ordenador, pensadas para ser implementadas en `MATLAB`. En muchas de ellas se pide realizar programas que se podrían sustituir por un único comando (de hecho, en general se indica que se compare con el comando en cuestión). Si hacemos esto es porque estamos convencidos (y la experiencia con nuestros alumnos así lo confirma) de que sólo cuando uno se enfrenta a la programación efectiva de los métodos es capaz de entenderlos en profundidad. En las direcciones de internet

`http://www.mat.ucm.es/~infante` o `http://www.mat.ucm.es/~jrey`

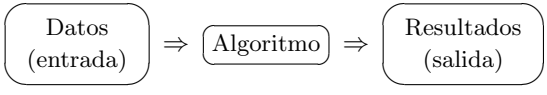
pueden encontrarse programas que resuelven algunas de las prácticas y servirán de ayuda, eventualmente, para resolver otras.

© Ediciones Pirámide

1 Análisis de errores

1.1. Introducción

El Análisis Numérico es una disciplina que contempla el desarrollo y evaluación de métodos para calcular, a partir de ciertos datos numéricos, los resultados requeridos. Se puede pensar el problema que se quiere resolver como una aplicación f que transforma datos x en resultados y; los datos constituyen la *información de entrada*, los resultados requeridos son la *información de salida* y el método de cálculo se conoce como *algoritmo*. Los ingredientes esenciales en un problema de Análisis Numérico pueden resumirse, pues, en el siguiente diagrama:



A modo de ejemplo, podemos pensar en el problema de hallar $\sqrt{17}$. En este caso, $x = 17$ es el *dato*, f la función raíz cuadrada e $y = \sqrt{17}$ el *resultado* deseado. Como *algoritmo* de cálculo podemos construir, para $\lambda = 17$, la sucesión

$$\begin{cases} x_0 = \lambda \\ x_n = \dfrac{1}{2}\left(x_{n-1} + \dfrac{\lambda}{x_{n-1}}\right), \ n \in \mathbb{N}, \end{cases}$$

cuyo límite es $\sqrt{\lambda}$ (compruébese), parando en un valor de n suficientemente grande.

Con bastante frecuencia nos encontramos con distintos algoritmos para construir la información de salida que se requiere. Así, volviendo al ejemplo anterior, para aproximar el número $\sqrt{17}$ podemos utilizar también el algoritmo que se obtiene al hacer un desarrollo de Taylor de orden 2 de la función raíz cuadrada

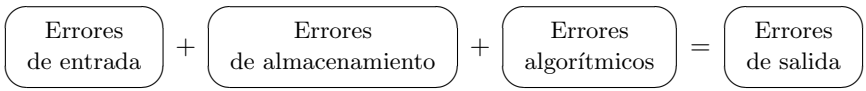
$$\sqrt{x} \simeq \sqrt{a} + \frac{1}{2\sqrt{a}}(x-a) - \frac{1}{8\sqrt{a^3}}(x-a)^2$$

© Ediciones Pirámide

particularizando en $x = 17$ y $a = 16$, es decir,

$$\sqrt{17} \simeq 4 + \frac{1}{8} - \frac{1}{512} = \frac{2111}{512} = 4.123046875.$$

Por tanto, para escoger entre los diversos algoritmos disponibles deben estudiarse los aspectos teóricos que contribuirán a la elección del algoritmo más adecuado a cada caso concreto. En general, los criterios fundamentales para preferir un algoritmo frente a otro son la *rapidez* y la *precisión*; a igualdad de precisión el algoritmo más rápido tendrá, obviamente, la preferencia. El objetivo de este capítulo es, de hecho, el estudio de la precisión o, equivalentemente, del *error*. En muy pocas ocasiones la información de entrada que se suministra es exacta pues se obtiene, en general, mediante instrumentos de medida; como, por otra parte, tanto el almacenamiento de los datos como el propio algoritmo de cálculo introducen también errores, la información de salida contendrá errores que provendrán de las tres fuentes. Esquemáticamente:



Sobre el primer tipo de errores nada podemos decir: están relacionados con el diseño de los aparatos de medición o la precisión de la percepción a través de los órganos sensoriales. Los otros dos tipos de errores son los que analizaremos en este capítulo: en las dos primeras secciones se aborda el estudio de los errores de almacenamiento y en las dos últimas los algorítmicos.

Antes de comenzar este estudio recordemos cierta terminología estándar en el tratamiento de errores. El error cometido al calibrar cierta magnitud puede ser medido bien en términos *absolutos*, bien en términos *relativos* al tamaño de la magnitud que se aproxima. Así, si $\tilde{z}$ es una aproximación de una cierta cantidad $z \neq 0$ y $||\cdot||$ es una norma (véase la definición 2.19), entonces

$$||\tilde{z} - z|| \quad \text{es el } \textit{error absoluto}$$

y

$$\frac{||\tilde{z} - z||}{||z||} \quad \text{es el } \textit{error relativo}$$

cometido en la aproximación anterior.[1]

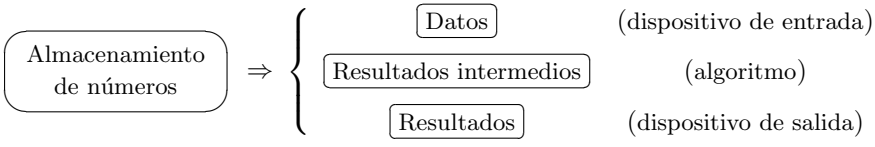
[1] Si $z = 0$ se trabaja sólo con errores absolutos.

© Ediciones Pirámide

Desde el punto de vista de las aplicaciones el error relativo resulta más relevante, ya que conocer el error absoluto no es muy útil si no se conoce la magnitud de la cantidad que se está midiendo. Por ejemplo, un error de $0'25$ cm al determinar la estatura de una persona puede ser irrelevante pero sería inaceptable en microcirugía.

1.2. Números máquina

Los problemas que aborda el Análisis Numérico se resuelven, fundamentalmente, mediante la realización de cadenas de operaciones aritméticas (más o menos sofisticadas) en un ordenador. Deberán, por tanto, almacenarse en la máquina los datos de partida (que introduciremos mediante un *dispositivo de entrada*, como puede ser un teclado o la lectura de un fichero) así como los resultados intermedios. Finalmente, deberán comunicarse los resultados finales (mediante un *dispositivo de salida*, como es la pantalla o la escritura en un fichero).



En esta sección vamos a ocuparnos del almacenamiento de los números en la máquina. Como es sabido, la capacidad de almacenamiento (la *memoria*) de los ordenadores crece vertiginosamente gracias a los progresos de la electrónica pero, por mucho que crezca, siempre será una capacidad finita. Esto implica que ninguna máquina es capaz de guardar ni siquiera un solo número irracional "completo" (las infinitas cifras decimales de π, por ejemplo). De hecho, cada número se representa en el ordenador con una cantidad máxima de cifras decimales; consecuentemente, sólo se guardarán de forma exacta los números que no excedan de ese máximo.

Vamos a intentar aclarar estas afirmaciones, que hemos introducido de forma un tanto vaga. Los números pueden representarse en la denominada *notación decimal en coma flotante*, que contiene la información relevante: signo, fracción, signo para el exponente y exponente. Por ejemplo, el número $-623'45$ admite, entre otras, las siguientes representaciones en coma flotante

$$-623'45+0,\ -62'345+1,\ -6234'5-1, -6'2345+2,\dots$$

que deben entenderse, respectivamente, como

$$-623'45\times 10^{0},\ -62'345\times 10^{1},\ -6234'5\times 10^{-1}, -6'2345\times 10^{2},\dots$$

© Ediciones Pirámide

De las infinitas representaciones posibles nos quedaremos con la última (denominada *notación decimal* en *coma flotante normalizada*), que está caracterizada por que la fracción es un número comprendido entre 1 y 10. Es decir, la notación decimal en coma flotante normalizada de un número real no nulo es

$$\pm m \pm E \ \text{ con } \ 1 \leq m < 10 \ \text{ y } \ E \in \mathbb{N} \cup \{0\}$$

que se corresponde con $\pm m \times 10^{\pm E}$.

Observación 1.1. A partir de ahora emplearemos, como hacen los ordenadores, la notación anglosajona de punto decimal en lugar de coma decimal, aunque seguiremos hablando de notación en coma flotante. □

Los ordenadores almacenan la información en cantidades ingentes de *posiciones de memoria* (o *bits*).[2] Éstas son entes físicos que sólo pueden tomar los valores 0/1, encendido/apagado, positivo/negativo o cualquier otra dicotomía electrónicamente viable. Es por esto por lo que no se utiliza la representación decimal para los números, sino la *representación binaria*.

Observación 1.2. El *sistema binario* utiliza el 2 como base, de la misma forma que el *sistema decimal* utiliza el 10. Con el propósito de hacer una comparación, recuérdese primero cómo funciona el sistema de representación decimal que nos es más familiar. Cuando escribimos el número 547.154 de forma más explícita tenemos

$$547.154 = 5 \times 10^2 + 4 \times 10^1 + 7 \times 10^0 + 1 \times 10^{-1} + 5 \times 10^{-2} + 4 \times 10^{-3}.$$

La expresión del segundo miembro utiliza potencias de 10 junto con los dígitos

$$0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9.$$

En el sistema binario sólo se utilizan los dígitos 0 y 1. Por ejemplo,

$$101.101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}.$$

Este número real se representa como 5.625 en notación decimal.

En general, cualquier número natural $\alpha > 1$ puede utilizarse como *base* para un *sistema numérico*. Los números representados en base α incluirán símbolos $s_0,\ s_1,\ s_2, \ldots,\ s_{\alpha-1}$ (véase el problema 1.8). De esta forma, todo número real x admite una representación en base α de la forma

$$\begin{aligned} x &= \pm \sum_{k=-\infty}^{\infty} c_k \alpha^k \\ &= \pm \left(\cdots + c_{-2}\alpha^{-2} + c_{-1}\alpha^{-1} + c_0\alpha^0 + c_1\alpha^1 + c_2\alpha^2 + \cdots \right) \end{aligned} \tag{1.1}$$

[2] La palabra *bit* es la abreviatura de *Binary Digit*.

© Ediciones Pirámide

con $c_k \in \{s_0, s_1, s_2, \ldots, s_{\alpha-1}\}$ para $k \in \mathbb{Z}$. Si el contexto no aclara cuál es la base numérica que se utiliza para el número x, suele emplearse la notación $(x)_\alpha$. Así, por ejemplo,

$$(101.101)_2 = (5.625)_{10}.$$

Ya que un ordenador se comunica con el usuario utilizando el sistema decimal (aun cuando internamente esté utilizando el sistema binario), debe efectuar procesos de conversión que ocurren durante la entrada o salida de la información. Normalmente el usuario no tiene que preocuparse de dichas conversiones, pero debe tener conciencia de que éstas implican errores de *redondeo*, como se verá en la sección 1.3. Por ejemplo, algunos números decimales sencillos toman expresiones más complicadas en el sistema binario; así,

$$(0.1)_{10} = (0.0\overbrace{0011}^{\frown})_2$$

(véase el problema 1.2). □

Observación 1.3. La representación en base $\alpha > 1$ dada en (1.1) puede no ser única como, por ejemplo, ocurre con el número representado en base 10 por $0.\overbrace{9}^{\frown}$ y 1. En efecto, al multiplicar $x = 0.\overbrace{9}^{\frown}$ por 10 obtenemos $10x = 9.\overbrace{9}^{\frown}$ de forma que al restar x a $10x$ queda $9x = 9$, de donde $x = 1$. □

Al igual que utilizábamos la notación decimal en coma flotante podemos también utilizar esta notación en el sistema binario. Un número binario representado en coma flotante normalizada tomará una expresión de la forma

$$(s)\ (S)\ E\ m$$

donde s es el signo del número (0 para los positivos y 1 para los negativos), S es el signo del exponente (también aquí 0 representará el signo + y 1 el signo −), el exponente E es un número entero en base 2 y m se denomina *mantisa* y es un número en base 2 verificando $1 \leq m < 2$ (es decir, $1 \leq m < (10)_2$). El número así representado será

$$(-1)^s \times m \times 2^{(-1)^S \times E_d}$$

donde E_d es la conversión decimal del número binario E (es decir, el número de lugares que hay que "correr la coma").

Ejemplo 1.1.

$$(1)\ (0)\ 10\ 1.01 \equiv -1.01 \times 2^2 \equiv -101\ (\equiv -(2^2 + 2^0) = -5 \text{ en decimal})$$

y

$$(0)\ (1)\ 1\ 1.1 \equiv 1.1 \times 2^{-1} \equiv 0.11\ (\equiv 2^{-1} + 2^{-2} = 0.75 \text{ en decimal}).$$ □

© Ediciones Pirámide

Cada número se almacena en la máquina en lo que se denomina una *palabra*. Cuando se trabaja en *precisión simple* cada palabra tiene una longitud de 32 posiciones de memoria. Estos 32 bits se distribuyen, como veremos a continuación, de la siguiente forma: 1 se usa para el signo del número, 8 para el exponente con signo y 23 para la mantisa.

Observación 1.4. El hecho de que la mantisa m verifique $1 \leq m < 10$ en binario, hace que la primera cifra de m (la que está a la izquierda del punto decimal) sea siempre un 1. Este 1, común a todos los números representados, no se guarda (se sobreentiende) y, por tanto, los 23 bits que se destinan a la mantisa sirven para almacenar las 24 primeras cifras. □

Obviamente, utilizando palabras de 32 bits, tan sólo puede ser representada una cantidad finita de números distintos: del orden de 2^{32} números diferentes.[3] Estos números, representables en el ordenador de forma *exacta*, se denominan *números máquina*.

Es fácil encontrar números que no son números máquina; por ejemplo, 1.0×2^{128} (128 es, en binario, 10000000, que no cabe en la parte destinada al exponente con signo) y $1.1 \overset{25)}{\cdots} 1$ (no cabe en el espacio destinado a la mantisa). El primero es un ejemplo en el que se sobrepasa la capacidad del ordenador (*desbordamiento*) y el segundo muestra que hay números que no se pueden usar de forma exacta y habrá que aproximarlos (*redondeo*). Ambos fenómenos serán estudiados más adelante.

Continuando con la representación de los números máquina, podemos preguntarnos por la representación del número real 0. La primera respuesta que nos viene a la cabeza es: "una palabra en la que los 32 bits tengan el valor 0". Pero el convenio de que se sobreentiende el primer 1 de la mantisa hace que la palabra anterior represente el número binario 1.0 (1 en notación decimal); de hecho, el 0 no sería un número máquina si se usara esta representación.

Este problema, junto con el de la doble representación de los números con exponente $+0$ y -0, se resuelve mediante el *formato estándar de representación* (*IEEE*[4] *Storage Format*): para almacenar la información correspondiente al exponente y su signo se suma 127 al exponente y se utilizan 8 bits para almacenarlo; el exponente verdadero se obtendrá al restar 127 a la conversión decimal del exponente almacenado. Así pues, el número queda representado en la forma

Signo del número	Exponente +127	Mantisa
1 bit	8 bits	23 bits

[3] Como los exponentes $+0$ y -0 dan lugar a los mismos números, el número total es la diferencia entre las variaciones con repetición de 2 elementos tomados de 32 en 32 y las variaciones con repetición de 2 elementos tomados de 24 en 24, es decir, $2^{32} - 2^{24} = 4278190080$.

[4] *Institute of Electrical and Electronic Engineers.*

© Ediciones Pirámide

Como el rango de números que pueden representarse con 8 bits es de 0 a 255 (ya que $|E| \leq (11111111)_2 = (255)_{10}$), los exponentes verdaderos correspondientes están en el rango comprendido entre -127 y 128. Los valores extremos 0 y 255 se usarán para números "especiales". En concreto, si s es el signo del número almacenado, m son los 23 dígitos de mantisa y E_d es la conversión decimal del número binario guardado en los 8 bits dedicados al exponente; cuando éste toma valores entre 1 y 254 (ambos inclusive), el número anteriormente representado corresponde al número binario

$$\boxed{(-1)^s \times 1.m \times 2^{E_d - 127}}$$

A este tipo de números se les denomina *números normales*.

Ejemplo 1.2. El número binario $101.11 = 1.0111 \times 2^2$ se representa de la siguiente forma

$$\overbrace{0}^{\text{Signo del número}} \quad \overbrace{10000001}^{2+127=129} \overbrace{01110000000000000000000}^{\text{Mantisa}}.$$

A la inversa, el número almacenado como

$$1\ 01111011\ 10100000000000000000000$$

es el número $-1.101 \times 2^{123-127} = -1.101 \times 2^{-4} = -0.0001101$ en base 2. □

Cuando el exponente toma el valor 0, el convenio es ligeramente distinto: el valor representado es

$$\boxed{(-1)^s \times 0.m \times 2^{-126}}$$

(obsérvese que aquí no se sobreentiende el valor 1 para el primer dígito de la mantisa y se ha desplazado la coma adecuadamente). Esta otra interpretación permite trabajar con números máquina mucho más pequeños, en valor absoluto, que los que se obtendrían con el convenio habitual. Así, al exponente 0 le corresponderían números positivos comprendidos entre

$$1.0 \times 2^{-127} \ \text{ y } \ 1.1 \overset{23)}{\cdots} 1 \times 2^{-127}$$

con el convenio habitual, y entre

$$0.0 \overset{22)}{\cdots} 01 \times 2^{-126} = 1.0 \times 2^{-149} \ \text{ y } \ 0.1 \overset{23)}{\cdots} 1 \times 2^{-126}$$

con esta nueva interpretación. De esta forma, se ve ampliada la capacidad de manejo, por parte de la máquina, de números "pequeños". Los números máquina

© Ediciones Pirámide

así representados se denominan *números subnormales*. Nótese que el 0 es uno de ellos, estando representado por una palabra enteramente formada por ceros, como pedía nuestra intuición.

Los números con exponente 255 representan las *excepciones*: $\pm\infty$ y `NaN` (*Not a Number*). Los valores $\pm\infty$ se producen a causa de un desbordamiento por exceso, mientras que el valor `NaN` surge al realizar ciertas operaciones como las clásicas indeterminaciones ($\frac{0}{0}$, $\infty - \infty, \ldots$), como se verá en la sección 1.3.

En la tabla 1.1 se recogen la representación y el valor decimal de algunos números especialmente interesantes (véase la práctica 1.3).

TABLA 1.1:
Representación de diversos números ($2^{127} \simeq 10^{38}$)

Número	Representación	Valor decimal
$+0$	$0\,00000000\,0\overset{23)}{\cdots}0$	0.0
-0	$1\,00000000\,0\overset{23)}{\cdots}0$	-0.0
Máximo número normal	$0\,11111110\,1\overset{23)}{\cdots}1$	$3.40282347 \times 10^{38}$
Mínimo número normal (positivo)	$0\,00000001\,0\overset{23)}{\cdots}0$	$1.17549435 \times 10^{-38}$
Máximo número subnormal	$0\,00000000\,1\overset{23)}{\cdots}1$	$1.17549421 \times 10^{-38}$
Mínimo número subnormal (positivo)	$0\,00000000\,0\overset{22)}{\cdots}01$	$1.40129846 \times 10^{-45}$
$+\infty$	$0\,11111111\,0\overset{23)}{\cdots}0$	infinito
$-\infty$	$1\,11111111\,0\overset{23)}{\cdots}0$	–infinito
`NaN`	$*\,11111111$ algún 1	no es un número

Observación 1.5.

1. Hasta ahora sólo hemos considerado la representación de números reales en precisión simple. En general, las máquinas también permiten trabajar en *doble precisión*: cada número se almacena en dos palabras unidas y, de los 64 bits disponibles, 1 se destina al signo, 11 al exponente y 52 a la mantisa. Así, los números normales en doble precisión se representan

Signo del número	Exponente +1023	Mantisa
1 bit	11 bits	52 bits

   lo que corresponde al número binario

   $$\boxed{(-1)^s \times 1.m \times 2^{E_d - 1023}}$$

   y determinan un rango del orden de 10^{-308} a 10^{308} (véase la práctica 1.4).

© Ediciones Pirámide

2. En general, los ordenadores (y los lenguajes de programación) distinguen entre números reales (tipo `REAL` o `FLOAT`) y números enteros (tipo `INTEGER`). Una vez declarado un número como entero, su almacenamiento se hace en una palabra, usando el primer bit para el signo y los 31 restantes para el número en binario. Por ejemplo, el número $(-254)_{10} \equiv (-11111110)_2$ se almacena como $10\overset{23)}{\cdots}011111110$. Como el mayor número entero representable es

$$0\,1\overset{31)}{\cdots}1 = \sum_{k=0}^{30} 2^k = (2^{31}-1)_{10},$$

esto hace que el rango de los números máquina enteros vaya desde $-(2^{31}-1)$ a $2^{31}-1$. También se pueden representar *enteros largos* mediante dos palabras unidas, usando 1 bit para el signo y 63 para el número en binario. Existen otras formas más sofisticadas de representar los números enteros que aquí no trataremos. En especial, la conocida con el nombre de *complemento a* 2, que está íntimamente relacionada con el tipo de almacenamiento con traslación que se usa para los exponentes.

3. Los números complejos se representan como un par de números reales (dos palabras en simple precisión y cuatro en doble precisión).

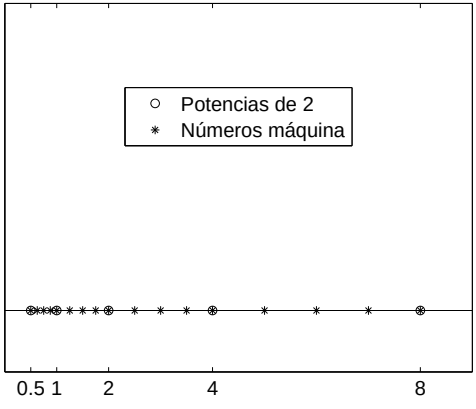


Figura 1.1: Distribución irregular de los números máquina.

4. En un ordenador binario, los números máquina están distribuidos de forma irregular, concentrándose más cuanto más cerca de cero se esté (véase la figura 1.1); téngase en cuenta que entre dos potencias consecutivas de 2 siempre existe la misma cantidad de números máquina: para un exponente fijo E y prefijado el signo, la cantidad total de números máquina normales

© Ediciones Pirámide

de la forma $\pm 1.m \times 2^E$ es 2^{23} (número de variaciones con repetición de 2 elementos tomados de 23 en 23). Dado que la distancia entre las potencias de 2 es más pequeña cerca del 0 y más grande lejos de él, se obtiene una distribución desigual de números máquina, siendo la densidad mayor cerca del origen. □

1.3. Desbordamiento y redondeo

Ya hemos visto la forma estándar de almacenar números máquina y hemos insistido en que éstos constituyen sólo una cantidad finita. ¿Qué podemos decir de los restantes números reales? El hecho de que un número real no sea un número máquina puede deberse a dos motivos:

a) Que su exponente (una vez normalizado el número) esté fuera del rango admitido (es decir, que el número sea demasiado grande o pequeño).

b) Que su mantisa (normalizada también) tenga más de 23 cifras (en cuyo caso el número tiene más cifras de las que se pueden almacenar).

Estas dos posibilidades se tratan de forma muy distinta:

a) Si el resultado de un cálculo o un número leído por la máquina tiene un exponente que se sale del rango admisible se produce el fenómeno conocido como *desbordamiento*, que puede ser de dos tipos: por *exceso* (*overflow*) o por *defecto* (*underflow*). La representación estándar en coma flotante trata el desbordamiento por exceso como una situación *excepcional*: toma el valor $+\infty$ (o $-\infty$ si el número es negativo)[5] y sigue trabajando, siempre que se pueda dar sentido a las operaciones siguientes (por ejemplo, $\frac{1}{\infty} = 0$). Para el desbordamiento por defecto se asigna el valor 0.

b) Cuando se necesita trabajar con números no máquina que no producen desbordamiento lo que se hace es aproximarlos por números máquina cercanos. Este proceso se denomina *redondeo*.

Veamos la forma en la que el redondeo se lleva a cabo: dado un número

$$x = \pm 1.a_1a_2 \cdots a_{23}a_{24}a_{25} \cdots \times 2^E$$

con E dentro del rango admisible, sean x_i y x_d los números máquina más próximos a x por la izquierda y por la derecha respectivamente, esto es,

$$x_i = x_d = x \text{ si } x \text{ es un número máquina}$$

[5] En la tabla 1.1 se vio cómo se representaban estos valores en la máquina.

© Ediciones Pirámide

y, en otro caso,

$$\begin{cases} x_i = 1.a_1 \cdots a_{23} \times 2^E & \text{y} \quad x_d = (1.a_1 \cdots a_{23} + 2^{-23}) \times 2^E \ \text{ si } \ x > 0 \\ x_i = -(1.a_1 \cdots a_{23} + 2^{-23}) \times 2^E & \text{y} \quad x_d = -1.a_1 \cdots a_{23} \times 2^E \ \text{ si } \ x < 0. \end{cases}$$

Nótese que, en cualquier caso, para todo número $x \in \mathbb{R}$ se verifica que

$$x_i \leq x \leq x_d.$$

En coma flotante estándar, para cada número real x, están definidos cuatro tipos de redondeo $r(x)$:

a) *Redondeo a la derecha* (o por *exceso*): $r(x) = x_d$.

b) *Redondeo a la izquierda* (o por *defecto*): $r(x) = x_i$.

c) *Redondeo a cero*: $r(x) = x_i$ si $x > 0$ y $r(x) = x_d$ si $x < 0$ (es decir, se elige de entre x_i y x_d el que está entre 0 y x).

d) *Redondeo al más próximo*: se elige de entre x_i y x_d el que está más cerca de x; en el caso de que estén a igual distancia, se toma el que tenga $a_{23} = 0$.

Observación 1.6.

1. Nótese que, con cualquiera de los criterios anteriores, si x es un número máquina entonces coincide con su redondeo $r(x)$.

2. Aunque en principio puede utilizarse cualquiera de las cuatro posibilidades anteriores (incluso puede cambiarse de una a otra dentro de la misma máquina) nosotros solamente utilizaremos la última de ellas, por ser la más útil y usada (de hecho es la opción por defecto) y la denominaremos, simplemente, *redondeo*. □

La primera cuestión que se nos plantea es qué error se comete cuando en lugar de trabajar con un número real x se trabaja (como es obligado) con su redondeo. En términos absolutos, y manteniendo la notación, se tiene la siguiente *cota del error de redondeo absoluto*

$$|r(x) - x| \leq \frac{x_d - x_i}{2} = \frac{2^{-23} \times 2^E}{2} = 2^{-24} \times 2^E$$

mientras que, en términos relativos, una *cota del error de redondeo relativo* (teniendo en cuenta que $|x| \geq 2^E$) es

$$\left| \frac{r(x) - x}{x} \right| \leq \frac{2^{-24} \times 2^E}{2^E} = 2^{-24}.$$

© Ediciones Pirámide

El valor de 2^{-24} obtenido en la cota del error de redondeo relativo es, exactamente, la mitad de la distancia entre 1 y el siguiente número máquina $1.0\overset{22)}{\cdots}01$. Esta distancia ($2^{-23} \simeq 1.2 \times 10^{-7}$) se denomina *precisión* o *épsilon* de la máquina y se denota por *eps*. Así pues, trabajando en coma flotante estándar, como

$$r(x) = \left(\frac{x + r(x) - x}{x}\right) x = \left(1 + \frac{r(x) - x}{x}\right) x,$$

se verifica que

$$r(x) = (1 + \delta)x \ \text{ con } \ |\delta| \leq 2^{-24} = \frac{eps}{2}$$

para todo $x \in \mathbb{R}$. Esto, en términos de la representación decimal de x, quiere decir que el redondeo de x tendrá alrededor de 7 *cifras significativas* correctas (de forma más precisa, un mínimo de 6 y un máximo de 8, dependiendo de en cuántas cifras decimales se transformen las 23 cifras decimales en binario).

Observación 1.7. Todas las consideraciones que se han hecho acerca del desbordamiento y el redondeo con precisión simple pueden trasladarse, con las correspondientes modificaciones, al caso en que se utilice doble precisión. En concreto, el épsilon de la máquina en doble precisión es 2^{-52} y el redondeo de un número en doble precisión tiene un mínimo de 15 y un máximo de 16 cifras significativas exactas. □

1.4. Aritmética en coma flotante

Para llevar a cabo una operación aritmética $(+, -, \times, /)$ en un ordenador, lo primero que se debe tener en cuenta es que los operandos con los que se trabaja no son exactamente los de partida, sino sus redondeos. Incluso aunque los operandos sean números máquina, el asunto es más complicado de lo que puede parecer a primera vista; el resultado de una operación con números máquina no tiene por qué ser necesariamente un número máquina y habrá, por tanto, que redondearlo. De hecho, la aritmética estándar en coma flotante está diseñada mediante operaciones $\oplus$, $\ominus$, $\otimes$, $\oslash$ que verifican, para cada par de números máquina x e y:

$$x \oplus y = r(x + y),\ x \ominus y = r(x - y),\ x \otimes y = r(x \times y),\ x \oslash y = r(x/y).$$

La implementación efectiva de las operaciones es bastante sofisticada, pues debe enfrentarse con diversos problemas de los que veremos algunos ejemplos seguidamente.

En primer lugar, para llevar a cabo una suma (o resta) de dos números máquina se siguen los siguientes pasos:

© Ediciones Pirámide

1. Se igualan los exponentes al mayor de ambos.
2. Se realiza la operación de forma exacta.
3. Se normaliza el resultado (modificando, si es necesario, el exponente de forma que la mantisa esté entre 1 y 2, es decir, de la forma $\pm 1.m \times 2^E$).
4. Se redondea (en caso de que sea necesario).
5. Se normaliza si es necesario.[6]

Ejemplo 1.3. Para sumar $1 \equiv 1.0 \times 2^0$ con $2 \equiv 1.0 \times 2^1$ se escribe

$$\left\{\begin{array}{rcl} 0.10 \overset{22)}{\cdots} 0 & \times & 2^1 \\ 1.00 \overset{22)}{\cdots} 0 & \times & 2^1 \\ \hline 1.10 \overset{22)}{\cdots} 0 & \times & 2^1 \end{array}\right.$$

Como se observa, sólo han hecho falta los dos primeros pasos. Sin embargo, si restamos a $1 \equiv 1.0 \times 2^0$ el número máquina más cercano a él por la izquierda, es decir, $1.1 \overset{23)}{\cdots} 1 \times 2^{-1}$, tenemos:

$$\text{En primer lugar:} \quad \left\{\begin{array}{rcl} 1.0 \overset{23)}{\cdots} 00 & \times & 2^0 \\ 0.1 \overset{23)}{\cdots} 11 & \times & 2^0 \\ \hline 0.0 \overset{23)}{\cdots} 01 & \times & 2^0 \end{array}\right.$$

$$\text{En segundo lugar:} \quad 1.0 \overset{23)}{\cdots} 0 \times 2^{-24}$$

Nótese que para realizar la operación de forma exacta hemos tenido que utilizar una posición más de memoria (24 bits) para guardar el segundo número y el resultado. Esto nos hace ver que las operaciones no se pueden llevar a cabo en palabras, sino que hace falta almacenar provisionalmente los operandos y el resultado en pedazos "más largos" de memoria. De hecho, en la aritmética estándar en coma flotante se utilizan *registros* de 80 bits para realizar las operaciones. □

Para la multiplicación (o división) de dos números no es necesario igualar los exponentes; basta multiplicar las dos mantisas (lo que dará un número de, a lo más, 48 cifras significativas, que cabe en un registro) y se suman los exponentes, normalizando el resultado, redondeándolo si es preciso y, eventualmente, volviéndolo a normalizar.

[6] Por ejemplo, el redondeo de $1.1 \overset{24)}{\cdots} 1 \times 2^E$ es 10.0×2^E que debe normalizarse a $1.0 \times 2^{E+1}$.

© Ediciones Pirámide

En cualquier caso, este tipo de precauciones garantizan, únicamente, que la aritmética en coma flotante se realiza siempre del modo correcto, en el sentido de que el resultado almacenado tras operar dos números máquina coincida con el redondeo del resultado real de la operación. No obstante, esto no impide que se produzcan situaciones anómalas que son intrínsecas al hecho de estar trabajando con una precisión determinada, como por ejemplo:

a) Las operaciones en coma flotante no verifican, en general, la propiedad asociativa. Así, para la suma,

$$(1 \oplus 2^{-24}) \oplus 2^{-24} = r\left(r\left(1 + 2^{-24}\right) + 2^{-24}\right) = r\left(1 + 2^{-24}\right) = 1$$

mientras que

$$\begin{aligned} 1 \oplus (2^{-24} \oplus 2^{-24}) &= r\left(1 + r\left(2^{-24} + 2^{-24}\right)\right) = r\left(1 + 2^{-23}\right) \\ &= 1 + 2^{-23} = 1.0\overset{22)}{\cdots}01 \times 2^0. \end{aligned}$$

b) Si dos números máquina x e y verifican $x \oplus y = x$, esto no implica, en general, que $y = 0$. De hecho, basta tomar cualquier número máquina y de la forma

$$y = \theta x \ \text{ con } \ 0 < \theta < 2^{-24}$$

para que se dé la igualdad $x \oplus y = x$. En efecto,

$$x \oplus y = r(x + y) = r(x + \theta x) = r(x) = x,$$

ya que al efectuar la suma entre

$$x = 1.a_1a_2 \cdots a_{23} \times 2^{E_x}$$

e

$$y = 1.b_1b_2 \cdots b_{23} \times 2^{E_y} \ \text{ con } \ E_y < E_x - 24$$

queda

$$\left\{\begin{array}{lcl} 1.a_1a_2 \cdots a_{23} & \times & 2^{E_x} \\ 0.0\overset{23)}{\cdots}00\overset{E_x-24-E_y)}{\cdots}01b_1b_2 \cdots b_{23} & \times & 2^{E_x} \\ \hline 1.a_1a_2 \cdots a_{23}0\overset{E_x-24-E_y)}{\cdots}01b_1b_2 \cdots b_{23} & \times & 2^{E_x}. \end{array}\right.$$

El resultado de la suma es un número cuyo redondeo es

$$1.a_1a_2 \cdots a_{23} \times 2^{E_x} = x.$$

Otro fenómeno a tener en cuenta es lo que se conoce como *cancelación* y aparece al restar dos números muy próximos entre sí, lo que puede producir grandes errores relativos.

© Ediciones Pirámide

Ejemplo 1.4. Al realizar la resta en el ordenador de los números

$$x = 1.1 \overset{23)}{\cdots} 101 \overset{15)}{\cdots} 1 \times 2^E \quad \text{e} \quad y = 1.1 \overset{15)}{\cdots} 1 \times 2^E$$

(teniendo en cuenta que y es ya un número máquina) se obtiene

$$\begin{aligned} x \ominus y = r(r(x) - y) &= r\left(1.1 \overset{23)}{\cdots} 1 \times 2^E - 1.1 \overset{15)}{\cdots} 1 \times 2^E\right) \\ &= r\left(0.0 \overset{15)}{\cdots} 01 \overset{8)}{\cdots} 1 \times 2^E\right) = 1.1 \overset{7)}{\cdots} 1 \times 2^{E-16}, \end{aligned}$$

mientras que el resultado exacto es

$$x - y = 0.0 \overset{15)}{\cdots} 01 \overset{8)}{\cdots} 101 \overset{15)}{\cdots} 1 \times 2^E = 1.1 \overset{7)}{\cdots} 101 \overset{15)}{\cdots} 1 \times 2^{E-16},$$

¡que es un número máquina! Como se observa, el resultado hallado coincide con el real en tan sólo las 9 primeras cifras significativas; esto se produce porque al normalizar $0.0 \overset{15)}{\cdots} 01 \overset{8)}{\cdots} 1 \times 2^E$ se "corre la coma" 16 lugares, añadiendo 16 ceros por la derecha que son independientes de los valores de partida. □

Veamos otro ejemplo patológico:

Ejemplo 1.5. Dada la función $f(x) = \operatorname{sen} x$, si se utiliza el cociente incremental

$$\frac{\operatorname{sen}(x+h) - \operatorname{sen} x}{h}$$

en $x = 1$ para aproximar

$$\cos 1 = f'(1) = \lim_{h \to 0} \frac{\operatorname{sen}(1+h) - \operatorname{sen} 1}{h}$$

se puede comprobar que, trabajando en doble precisión, se obtienen los resultados de la tabla 1.2 para diversos valores de h, donde se han comparado los resultados obtenidos con el valor de $\cos 1 \simeq 0.54030230586814$. ¿Qué es lo que está ocurriendo? Lo que sucede es que a medida que h va decreciendo, los números $\operatorname{sen}(1+h)$ y $\operatorname{sen} 1$ van teniendo, cada vez, más cifras significativas iguales y, por tanto, su diferencia va teniendo cada vez menor número de cifras exactas (de hecho, para valores de h menores o iguales a 10^{-16} la diferencia $\operatorname{sen}(1+h) - \operatorname{sen} 1$ da, en el ordenador, el valor 0). □

La consecuencia que podemos extraer de los ejemplos 1.4 y 1.5 es que, en la medida de lo posible, deben intentar evitarse sustracciones de números muy cercanos. A veces hay que prever estos problemas y cambiar el orden de las operaciones con vistas a obtener representaciones matemáticas equivalentes que eviten las cancelaciones.

© Ediciones Pirámide

TABLA 1.2:
Valores aproximados de $\cos 1$ *mediante el cociente* $\frac{\operatorname{sen}(1+h)-\operatorname{sen} 1}{h}$

h	Valor aproximado	Error absoluto	Error relativo (%)
10^{-5}	0.54029809850586	0.00000420736228	0.00077870522286
10^{-6}	0.54030188512133	0.00000042074681	0.00007787248080
10^{-7}	0.54030226404045	0.00000004182769	0.00000774153481
10^{-8}	0.54030230289825	0.00000000296989	0.00000054967103
10^{-9}	0.54030235840941	0.00000005254127	0.00000972442009
10^{-10}	0.54030224738710	0.00000005848104	0.00001082376215
10^{-11}	0.54030113716408	0.00000116870406	0.00021630558456
10^{-12}	0.54034554608506	0.00004324021692	0.00800296731187
10^{-13}	0.53956838996783	0.00073391590031	0.13583430837565
10^{-14}	0.54400928206633	0.00370697619819	0.68609298126735
10^{-15}	0.55511151231258	0.01480920644444	2.74091120537484
10^{-16}	0	0.54030230586814	100

Ejemplo 1.6.

1. Como es sabido, las raíces de la ecuación de segundo grado

$$ax^2 + bx + c = 0 \quad (a, b, c \in \mathbb{R},\ a \neq 0)$$

son

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{y} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Ahora bien, cuando $b > 0$ y $0 < 4ac \ll b^2$, resulta conveniente utilizar la siguiente expresión equivalente para el cálculo de la primera raíz:

$$x_1 = \frac{\left(\sqrt{b^2 - 4ac} - b\right)\left(\sqrt{b^2 - 4ac} + b\right)}{2a\left(\sqrt{b^2 - 4ac} + b\right)} = -\frac{2c}{b + \sqrt{b^2 - 4ac}}.$$

2. En lugar de trabajar con la función

$$f(x) = \frac{x^4 - x^3 - x + 1}{(x-1)^2},\ x \in \mathbb{R} \tag{1.2}$$

cuando $x \simeq 1$, podemos escribir ésta en forma equivalente como

$$f(x) = \frac{(x-1)^2(x^2 + x + 1)}{(x-1)^2} = x^2 + x + 1. \tag{1.3}$$

© Ediciones Pirámide

Como se observa en la tabla 1.3, mediante la expresión (1.2) evaluamos f de forma errónea, debido al fenómeno de la cancelación. □

TABLA 1.3:
Valores de f en puntos próximos a $x = 1$

Abscisa x	Expresión (1.2)	Expresión (1.3)
1.00000762939453	3.00002288818359	3.00002288824180
1.00000381469727	3.00001525878906	3.00001144410635
1.00000190734863	3.00000000000000	3.00000572204954
1.00000095367432	3.00000000000000	3.00000286102386
1.00000047683716	3.00000000000000	3.00000143051170
1.00000023841858	3.00000000000000	3.00000071525579
1.00000011920929	3.00000000000000	3.00000035762788
1.00000005960464	3.00000000000000	3.00000017881394
1.00000002980232	3.00000000000000	3.00000008940697
1.00000001490116	3.00000000000000	3.00000004470348
1.00000000745058	4.00000000000000	3.00000002235174
1.00000000372529	0	3.00000001117587

1.5. Propagación del error

Según hemos visto, a la hora de resolver un problema lo primero que hay que tener en cuenta es que los datos de partida con los que va a trabajar el ordenador no tienen por qué ser exactamente los datos originales (debido al redondeo). La segunda cuestión relevante es que, aunque los datos se almacenaran de forma exacta en el ordenador, en el momento que empezamos a trabajar con ellos se comienzan a cometer errores, de forma que, en algunas ocasiones, el resultado obtenido dista mucho del deseado. Para clarificar lo anterior analicemos una simple operación aritmética como es la suma de dos números: dados dos números $x, y \in \mathbb{R}$, para calcular su suma $x + y$ el ordenador halla

$$\begin{aligned} x \oplus y &= r\left(r(x) + r(y)\right) = r\left((1+\delta_1)x + (1+\delta_2)y\right) \\ &= (1+\delta_3)\left[(1+\delta_1)x + (1+\delta_2)y\right] \\ &= (1+\delta_3)(x+y) + (1+\delta_3)\delta_1 x + (1+\delta_3)\delta_2 y \end{aligned} \tag{1.4}$$

con

$$|\delta_i| \leq 2^{-24} = \frac{eps}{2}$$

para $i = 1, 2, 3$. Vemos así que los errores de redondeo en los datos se propagan al resultado de la operación. Si ahora este resultado fuera un operando de otra

© Ediciones Pirámide

operación, este error se propagaría al consiguiente resultado, y así sucesivamente. De esta forma, si pensamos en un algoritmo que involucre una gran cantidad de operaciones elementales, las perspectivas pueden parecer no muy buenas.

En general, si denotamos por $*$ cualquier operación elemental $(+, -, \cdot, /)$, para cada par de números máquina x e y, se tendrá

$$r(x * y) = (1 + \delta)(x * y) \;\text{ con }\; |\delta| \leq \frac{eps}{2}.$$

Si x e y no son números máquina, se tiene

$$r(r(x) * r(y)) = r\left([(1 + \delta_1)x] * [(1 + \delta_2)]y\right) = (1 + \delta_3)\left((1 + \delta_1)x * (1 + \delta_2)y\right).$$

Para operaciones compuestas las cosas se complican. Supongamos que queremos calcular $x(y + z)$ donde x, y y z son números máquina. El resultado es

$$\begin{aligned} x \otimes (y \oplus z) &= r\left(xr(y + z)\right) = (1 + \delta_1)\left(xr(y + z)\right) \\ &= (1 + \delta_1)\left[(1 + \delta_2)x(y + z)\right] = (1 + \delta_1 + \delta_2 + \delta_1\delta_2)x(y + z) \\ &\simeq (1 + \delta_1 + \delta_2)x(y + z) = (1 + \delta_3)x(y + z) \end{aligned}$$

con $|\delta_3| \leq |\delta_1| + |\delta_2| \leq 2^{-24} + 2^{-24} = 2^{-23}$.

Parece claro, por tanto, que debemos estudiar cuánto influye la propagación del error en el resultado final del problema. Dos son los principales conceptos ligados a este estudio:

a) El *condicionamiento*. Mide la influencia que tendrían en el resultado eventuales errores en los datos, en el caso ideal de que se pudiese trabajar con aritmética exacta; está ligado, por tanto, al problema en sí y no depende del algoritmo que se utilice para resolverlo.

b) La *estabilidad*. Está relacionada con la influencia que tiene en los resultados finales la acumulación de los errores que se producen en las sucesivas operaciones elementales que se llevan a cabo para resolver el problema; es decir, depende del algoritmo que se utilice para obtener la solución.

Así pues, hablaremos de un problema bien o mal condicionado, mientras que, dado un algoritmo para resolver un problema, diremos si es o no numéricamente estable. Ambos conceptos son, en general, difíciles de analizar.

1.5.1. Condicionamiento

Diremos que un problema está *mal condicionado* cuando pequeños cambios en los datos dan lugar a grandes cambios en las respuestas. Las técnicas que se

© Ediciones Pirámide

emplean en el estudio del condicionamiento de un problema están fuertemente ligadas a la estructura del mismo. En general, a la hora de resolver un problema $y = f(x)$ se intenta definir un *número de condición* $\kappa = \kappa(x) \geq 0$ de forma que

$$\frac{||f(\widetilde{x}) - f(x)||}{||f(x)||} \simeq \kappa(x) \frac{||\widetilde{x} - x||}{||x||}. \tag{1.5}$$

Este número indicará, según sea cercano a 1 o alejado de éste, si el problema está bien o mal condicionado, respectivamente. Si el número de condición es menor que 1 o está próximo a 1, el error del dato no se amplificará mucho y el error del resultado será, a lo sumo, del mismo orden que el error en el dato; por el contrario, si este número de condición toma valores muy grandes, el error final será una gran amplificación del error en el dato.

Para casos concretos, podremos definir fácilmente un número de condición. Comencemos con un problema sencillo: la evaluación de una función diferenciable $f : \mathbb{R} \to \mathbb{R}$ en un punto x. Si en lugar de x tomamos una aproximación suya $\widetilde{x}$ con $|\widetilde{x} - x| \ll 1$ (por ejemplo su redondeo), el teorema del Valor Medio asegura que

$$f(\widetilde{x}) - f(x) = f'(\xi)(\widetilde{x} - x) \simeq f'(x)(\widetilde{x} - x).$$

De esta forma, si $f'(x)$ no es muy grande, el efecto de la perturbación sobre $f(x)$ es pequeño. Concretamente, el error relativo de la perturbación viene dado por

$$\overbrace{\left|\frac{f(\widetilde{x}) - f(x)}{f(x)}\right|}^{\text{Error relativo (resultados)}} \simeq \left|\frac{f'(x)}{f(x)}\right| |\widetilde{x} - x| = \underbrace{\left|\frac{f'(x)}{f(x)} x\right|}_{\text{Número de condición}} \overbrace{\left|\frac{\widetilde{x} - x}{x}\right|}^{\text{Error relativo (datos)}}$$

Ejemplo 1.7. Consideremos la función

$$f(x) = 1 - \sqrt{1 - x^2},\ -1 \leq x \leq 1.$$

Como

$$f'(x) = \frac{x}{\sqrt{1 - x^2}},\ -1 < x < 1$$

© Ediciones Pirámide

entonces el número de condición de f es

$$\left|\frac{f'(x)}{f(x)}x\right| = \frac{x^2}{\sqrt{1-x^2}(1-\sqrt{1-x^2})}$$
$$= \frac{x^2(1+\sqrt{1-x^2})}{\sqrt{1-x^2}(1-\sqrt{1-x^2})(1+\sqrt{1-x^2})}$$
$$= \frac{1+\sqrt{1-x^2}}{\sqrt{1-x^2}}$$

que es muy grande para valores de x próximos a ± 1. □

Ejemplo 1.8. Supongamos que $\widetilde{x}_1$ y $\widetilde{x}_2$ son aproximaciones de x_1 y x_2 con errores ε_1 y ε_2 respectivamente, es decir, $\widetilde{x}_1 = x_1 + \varepsilon_1$ y $\widetilde{x}_2 = x_2 + \varepsilon_2$. Veamos que el error relativo del producto es, aproximadamente, igual a la suma de los errores relativos de los factores, esto es,

$$\frac{\widetilde{x}_1\widetilde{x}_2 - x_1x_2}{x_1x_2} \sim \frac{\varepsilon_1}{x_1} + \frac{\varepsilon_2}{x_2} \tag{1.6}$$

y, por tanto, el producto de dos números es siempre un problema bien condicionado. En efecto, como

$$\widetilde{x}_1\widetilde{x}_2 = (x_1+\varepsilon_1)(x_2+\varepsilon_2) = x_1x_2 + x_1\varepsilon_2 + x_2\varepsilon_1 + \varepsilon_1\varepsilon_2$$

entonces

$$\widetilde{x}_1\widetilde{x}_2 - x_1x_2 \sim x_1\varepsilon_2 + x_2\varepsilon_1,$$

de donde se sigue (1.6).

Sin embargo, la suma de dos números no va a ser siempre un problema bien condicionado. Teniendo en cuenta que

$$\widetilde{x}_1 + \widetilde{x}_2 = (x_1+\varepsilon_1) + (x_2+\varepsilon_2) = x_1 + x_2 + \varepsilon_1 + \varepsilon_2$$

se verifica que

$$(\widetilde{x}_1 + \widetilde{x}_2) - (x_1 + x_2) = \varepsilon_1 + \varepsilon_2,$$

de donde

$$\frac{(\widetilde{x}_1 + \widetilde{x}_2) - (x_1 + x_2)}{x_1 + x_2} = \frac{\varepsilon_1}{x_1 + x_2} + \frac{\varepsilon_2}{x_1 + x_2} = \frac{x_1}{x_1 + x_2}\frac{\varepsilon_1}{x_1} + \frac{x_2}{x_1 + x_2}\frac{\varepsilon_2}{x_2}.$$

Nótese que el problema para la suma de dos números está bien condicionado salvo cuando $x_1 \simeq -x_2$, en cuyo caso el condicionamiento puede ser muy malo. Esto explica el fenómeno ya estudiado de la cancelación. □

© Ediciones Pirámide

Otro tipo de problemas para los que se puede dar un número de condición de forma relativamente sencilla es la resolución de sistemas lineales $Ax = b$, como veremos en el capítulo 3. Anticipamos, a continuación, un ejemplo de sistema lineal mal condicionado.

Ejemplo 1.9 (R. S. Wilson). Consideremos el sistema lineal $Ax = b$ donde b es el vector

$$b = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

y A es la aparentemente "inofensiva" matriz simétrica

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$$

que tiene por matriz inversa a

$$A^{-1} = \begin{pmatrix} 25 & -41 & 10 & -6 \\ -41 & 68 & -17 & 10 \\ 10 & -17 & 5 & -3 \\ -6 & 10 & -3 & 2 \end{pmatrix}$$

y cuyo determinante vale 1. La solución exacta de dicho sistema es

$$x = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

Si consideramos las siguientes perturbaciones de los datos A y b

$$\widetilde{A} = \begin{pmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{pmatrix} \quad \text{y} \quad \widetilde{b} = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix},$$

las soluciones exactas de los sistemas lineales $\widetilde{A}y = b$ y $Az = \widetilde{b}$ vienen dadas, respectivamente, por

$$y = \begin{pmatrix} -81 \\ 137 \\ -34 \\ 22 \end{pmatrix} \quad \text{y} \quad z = \begin{pmatrix} 9.2 \\ -12.6 \\ 4.5 \\ -1.1 \end{pmatrix}.$$

© Ediciones Pirámide

Como se aprecia, pequeños cambios en el dato A proporcionan un resultado muy alejado de la solución original, x. Análogamente, cuando se perturba ligeramente el dato b se obtiene también un resultado z también distante de x.

La justificación de estas propiedades sorprendentes, así como la forma precisa de medir el tamaño de las perturbaciones y de los errores, se verán en el capítulo 3 (véase, en particular, el ejemplo 3.1). □

1.5.2. Estabilidad

En términos generales, diremos que un algoritmo es *inestable* cuando los errores que se van produciendo en las diversas etapas van aumentando en etapas posteriores y afectan, en gran manera, a la exactitud del cálculo en su conjunto; un algoritmo es *estable* cuando no es *inestable*. Otra forma de aproximarse al concepto de estabilidad es pensar un *algoritmo estable* como aquel que consigue resultados cercanos a los exactos cuando se parte de datos próximos a los verdaderos.

Ejemplo 1.10. Para ilustrar el comentario anterior vamos a considerar dos algoritmos para el cálculo de $\left(\frac{1}{7}\right)^{100}$. Cada uno de ellos es una aplicación particular de dos algoritmos más generales que sirven para calcular las potencias sucesivas de un número positivo $\lambda > 0$, a saber

$$\begin{cases} x_0 = 1 \\ x_n = \lambda x_{n-1},\ n \in \mathbb{N} \end{cases} \quad \text{y} \quad \begin{cases} x_0 = 1,\ x_1 = \lambda \\ x_{n+1} = (3+\lambda)x_n - 3\lambda x_{n-1},\ n \in \mathbb{N}. \end{cases}$$

En ambos casos, el término general de la sucesión $\{x_n\}_{n=0}^{\infty}$ es

$$x_n = \lambda^n,\ n \in \mathbb{N} \cup \{0\}.$$

En efecto, en el primero de ellos se comprueba de forma inmediata y en el segundo basta observar que $x_0 = 1$, $x_1 = \lambda$ y para cada $n \in \mathbb{N}$ se verifica que

$$(3+\lambda)x_n - 3\lambda x_{n-1} = (3+\lambda)\lambda^n - 3\lambda\lambda^{n-1} = 3\lambda^n + \lambda^{n+1} - 3\lambda^n = \lambda^{n+1} = x_{n+1}.$$

En nuestro caso concreto queremos hallar el término x_{100} para la elección $\lambda = \frac{1}{7}$, por lo que basta calcular los 100 primeros términos de las sucesiones anteriores para dicho valor de λ, esto es,

$$(\varphi_1) \begin{cases} x_0 = 1 \\ x_n = \dfrac{x_{n-1}}{7},\ n = 1, 2, \ldots, 100 \end{cases}$$

© Ediciones Pirámide

y

$$(\varphi_2)\begin{cases} x_0 = 1,\ x_1 = \dfrac{1}{7} \\ x_{n+1} = \dfrac{22}{7}x_n - \dfrac{3}{7}x_{n-1},\ n = 1, 2, \ldots, 99. \end{cases}$$

Si se calculan los sucesivos valores de x_n mediante ambos algoritmos se obtienen los resultados expuestos en la tabla 1.4. Estos resultados muestran el buen comportamiento del algoritmo φ_1 frente al pésimo comportamiento de φ_2. □

TABLA 1.4:
Algoritmos φ_1 y φ_2 para el cálculo de 7^{-100} en doble precisión

n	Valor de x_n (φ_1)	Valor de x_n (φ_2)
0	1	1
1	0.1428571428571	0.1428571428571
2	$2.0408163265306 \times 10^{-2}$	$2.0408163265306 \times 10^{-2}$
3	$2.9154518950437 \times 10^{-3}$	$2.9154518950436 \times 10^{-3}$
4	$4.1649312786339 \times 10^{-4}$	$4.1649312786308 \times 10^{-4}$
11	$5.0573331066306 \times 10^{-10}$	$5.0505834009817 \times 10^{-10}$
12	$7.2247615809009 \times 10^{-11}$	$7.0222704114337 \times 10^{-11}$
13	$1.0321087972716 \times 10^{-11}$	$4.2463528886988 \times 10^{-12}$
14	$1.4744411389594 \times 10^{-12}$	$-1.6749764113091 \times 10^{-11}$
15	$2.1063444842277 \times 10^{-13}$	$-5.4461981307728 \times 10^{-11}$
16	$3.0090635488967 \times 10^{-14}$	$-1.6398775663296 \times 10^{-10}$
25	$7.4567399858374 \times 10^{-22}$	$-3.2283632877849 \times 10^{-6}$
30	$4.4366870862363 \times 10^{-26}$	$-7.8449227893174 \times 10^{-4}$
34	$1.8478496818977 \times 10^{-29}$	$-6.3543874593471 \times 10^{-2}$
35	$2.6397852598538 \times 10^{-30}$	-0.1906316237804
38	$7.6961669383493 \times 10^{-33}$	-5.1470538420712
40	$1.5706463139488 \times 10^{-34}$	$-4.6323484578640 \times 10$
45	$9.3451913723379 \times 10^{-39}$	$-1.1256606752610 \times 10^{4}$
50	$5.5602971216386 \times 10^{-43}$	$-2.7353554408841 \times 10^{6}$
60	$1.9684192301176 \times 10^{-51}$	$-1.6152000342877 \times 10^{11}$
70	$6.9684662181415 \times 10^{-60}$	$-9.5375946824653 \times 10^{15}$
80	$2.4669298435211 \times 10^{-68}$	$-5.6318542840489 \times 10^{20}$
90	$8.7332601785620 \times 10^{-77}$	$-3.3255536361880 \times 10^{25}$
100	$3.0916904080902 \times 10^{-85}$	$-1.9637061666327 \times 10^{30}$

© Ediciones Pirámide

Formalicemos a continuación el concepto de estabilidad. Para ello introducimos primero la siguiente notación debida a *Landau*.

Notación 1.1. Utilizaremos la expresión $\phi(r) = O(r)$ para denotar que la cantidad $\phi(r)$ es del orden de r para r pequeño, es decir, cuando existe una constante $C > 0$ tal que

$$|\phi(r)| \leq Cr,\ 0 < r \ll 1. \quad \square$$

Definición 1.1. Un algoritmo φ de resolución del problema $y = f(x)$ es *estable* si existe $\widetilde{x}$ con

$$\frac{||\widetilde{x} - x||}{||x||} = O(eps)$$

verificando

$$\frac{||\varphi(x) - f(\widetilde{x})||}{||f(\widetilde{x})||} = O(eps). \tag{1.7}$$

En caso contrario se dice que el algoritmo φ es *inestable*. $\square$

Observación 1.8. Nótese que la expresión (1.7) indica que un algoritmo estable φ proporciona un resultado relativamente cercano (*i.e.*, con error relativo del orden del épsilon de la máquina) al resultado correspondiente a un dato relativamente cercano (en el sentido anterior) al dato original. $\square$

Ejemplo 1.11. Estudiemos la estabilidad de los algoritmos φ_1 y φ_2 del ejemplo 1.10. En el algoritmo φ_2 es claro que $f(\widetilde{\lambda}) = \widetilde{\lambda}^{100}$ es muy pequeño para cualquier valor $\widetilde{\lambda}$ que esté cerca de $\frac{1}{7}$. De esta forma, con los datos de la tabla 1.4, se puede comprobar que

$$\frac{\left|\varphi_2\left(\frac{1}{7}\right) - f(\widetilde{\lambda})\right|}{\left|f(\widetilde{\lambda})\right|} \simeq 10^{114}$$

para cualquier $\widetilde{\lambda}$ próximo a $\frac{1}{7}$, lo que hace que φ_2 sea un algoritmo inestable.

En el estudio de la estabilidad del algoritmo φ_1 vamos a afinar un poco más;

© Ediciones Pirámide

los valores sucesivos que se van obteniendo con él son:

$$\begin{cases} x_0 = 1 \\ x_1 = r\left(\dfrac{1}{7}\right) \\ x_2 = \left(r\left(\dfrac{1}{7}\right)\right)^2 (1+\delta_1) \\ x_3 = \left(\left(r\left(\dfrac{1}{7}\right)\right)^2 (1+\delta_1) r\left(\dfrac{1}{7}\right)\right)(1+\delta_2) = \left(r\left(\dfrac{1}{7}\right)\right)^3 (1+\delta_1)(1+\delta_2) \\ \quad \cdots \\ x_{100} = \left(r\left(\dfrac{1}{7}\right)\right)^{100} (1+\delta_1)(1+\delta_2)\cdots(1+\delta_{99}) \end{cases}$$

donde $r\left(\dfrac{1}{7}\right)$ denota el redondeo de $\dfrac{1}{7}$ y

$$|\delta_i| \leq \frac{eps}{2}$$

para $i = 1, 2, \ldots, 99$. De esta forma, eligiendo

$$\widetilde{\lambda} = r\left(\frac{1}{7}\right) \sqrt[100]{(1+\delta_1)(1+\delta_2)\cdots(1+\delta_{99})}$$

se tiene que

$$\frac{\left|\widetilde{\lambda} - \dfrac{1}{7}\right|}{\left|\dfrac{1}{7}\right|} = O(eps) \;\text{ y }\; \varphi_1\left(\frac{1}{7}\right) = f(\widetilde{\lambda}) \tag{1.8}$$

con lo que

$$\frac{\left|\varphi_1\left(\dfrac{1}{7}\right) - f(\widetilde{\lambda})\right|}{\left|f(\widetilde{\lambda})\right|} = 0$$

y, por tanto, el algoritmo φ_1 es estable. □

Observación 1.9. El comportamiento asintótico hacia $-\infty$ de la sucesión definida en el algoritmo φ_2 puede justificarse de la siguiente forma. Dicho algoritmo es, a su vez, una particularización de la ley de recurrencia más general

$$\begin{cases} x_0 = \alpha + \beta,\ x_1 = \dfrac{\alpha}{7} + 3\beta \\ x_{n+1} = \dfrac{22}{7}x_n - \dfrac{3}{7}x_{n-1},\ n \in \mathbb{N} \end{cases} \qquad (\alpha, \beta \in \mathbb{R})$$

© Ediciones Pirámide

que tiene como término general

$$x_n = \alpha \left(\frac{1}{7}\right)^n + \beta 3^n,\ n \in \mathbb{N} \cup \{0\}.$$

Nuestro caso se corresponde con la elección $\alpha = 1$ y $\beta = 0$. No obstante, al tomar la máquina los datos iniciales redondeados

$$\widetilde{x}_0 = 1 \ \text{ y } \ \widetilde{x}_1 = r\left(\frac{1}{7}\right) = 0.14285714285714,$$

los valores de α y β correspondientes vienen dados por la solución del sistema lineal

$$\begin{cases} \alpha + \beta = 1 \\ \dfrac{\alpha}{7} + 3\beta = 0.14285714285714, \end{cases}$$

es decir,

$$\alpha = \frac{21 - 7r\left(\frac{1}{7}\right)}{20} = 1 + 10^{-15} \ \text{ y } \ \beta = \frac{7r\left(\frac{1}{7}\right) - 1}{20} = -10^{-15}.$$

El hecho de que $\beta \neq 0$ hace que para valores grandes de n la cantidad $\beta 3^n$ sea, en valor absoluto, muy grande y perturbe desmesuradamente los resultados finales. □

Una técnica estándar para estudiar la estabilidad de un algoritmo es lo que se denomina *análisis regresivo del error*: mediante las manipulaciones necesarias, se intenta escribir el resultado obtenido por el algoritmo como el verdadero valor que se obtendría si se aplicasen operaciones exactas a otros datos distintos; si la diferencia entre estos últimos y los reales no es grande, el algoritmo será estable y, si es nula, será *regresivamente estable*. Más concretamente,

Definición 1.2. Un algoritmo φ de resolución del problema $y = f(x)$ es *regresivamente estable* si existe $\widetilde{x}$ con

$$\frac{||\widetilde{x} - x||}{||x||} = O(eps)$$

verificando

$$\varphi(x) = f(\widetilde{x}). \quad \square \tag{1.9}$$

Observación 1.10. Claramente, un algoritmo regresivamente estable es estable puesto que (1.9) implica

$$\frac{||\varphi(x) - f(\widetilde{x})||}{||f(\widetilde{x})||} = 0$$

© Ediciones Pirámide

que, obviamente, es del orden del épsilon de la máquina. Un algoritmo regresivamente estable φ proporciona, por tanto, un resultado exacto correspondiente a un dato relativamente cercano (en el sentido anteriormente explicado) al dato verdadero. Es precisamente el hecho de que el error cometido se proyecte hacia atrás sobre los datos lo que justifica el calificativo de regresivo. □

Ejemplo 1.12. El algoritmo φ_1 considerado en el ejemplo 1.10 es regresivamente estable (véase (1.8)). □

Los conceptos de condicionamiento y estabilidad regresiva nos permiten estudiar, de manera sencilla, la *precisión* que un algoritmo proporciona a la hora de resolver un problema concreto.

Teorema 1.1. *Supongamos que queremos resolver un problema $y = f(x)$, cuyo número de condición es $\kappa(x) \geq 0$, mediante un algoritmo φ regresivamente estable. Entonces se verifica que*

$$\frac{||\varphi(x) - f(x)||}{||f(x)||} = O(\kappa(x)\ eps),$$

es decir, el error relativo cometido es del orden del número de condición multiplicado por el épsilon de la máquina.

Demostración. Por ser φ un algoritmo regresivamente estable se tiene que

$$\varphi(x) = f(\widetilde{x})$$

para un cierto $\widetilde{x}$ verificando

$$\frac{||\widetilde{x} - x||}{||x||} = O(eps)$$

(véase (1.9)). A partir de la relación (1.5), se concluye que

$$\frac{||\varphi(x) - f(x)||}{||f(x)||} = \frac{||f(\widetilde{x}) - f(x)||}{||f(x)||} \simeq \kappa(x)\frac{||\widetilde{x} - x||}{||x||} = O(\kappa(x)\ eps). \quad \square$$

Observación 1.11. El teorema 1.1 aporta información sobre la precisión del algoritmo en el siguiente sentido: determina el orden del error relativo que se comete cuando se toma el resultado que el algoritmo proporciona a partir del dato exacto, en lugar de tomar el resultado exacto del problema (que, como ya se ha indicado, es imposible de calcular en la mayoría de los casos). □

© Ediciones Pirámide

Ejemplo 1.13. Como el número de condición de la función

$$\psi(x) = x^{100},\ x > 0$$

es constante para todo $x > 0$ e igual a

$$\kappa = \left| x\frac{\psi'(x)}{\psi(x)} \right| = 100,$$

se tiene que el error relativo cometido al usar el algoritmo φ_1 considerado en el ejemplo 1.10 es del orden de 100 veces el épsilon de la máquina. □

1.6. Problemas

1.6.1. Problemas resueltos

1.1. Encontrar la expresión decimal de los números binarios 11.11 y $111.\overline{101}$.

SOLUCIÓN. Claramente se tiene que

$$(11.11)_2 = 1\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2} = 2 + 1 + \frac{1}{2} + \frac{1}{4} = (3.75)_{10}$$

y

$$\begin{aligned}\left(111.\overline{101}\right)_2 &= 1\times 2^2 + 1\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} \\ &\quad + 1\times 2^{-3} + 1\times 2^{-4} + 0\times 2^{-5} + 1\times 2^{-6} + \cdots \\ &= 4 + 2 + 1 + (4+0+1)\times 2^{-3} + (4+0+1)\times 2^{-6} \\ &\quad + (4+0+1)\times 2^{-9} + (4+0+1)\times 2^{-12} + \cdots \\ &= 7 + 5\sum_{n=1}^{\infty}\frac{1}{2^{3n}} = 7 + 5\sum_{n=1}^{\infty}\left(\frac{1}{8}\right)^n = 7 + 5\frac{\frac{1}{8}}{1-\frac{1}{8}} \\ &= 7 + \frac{5}{7} = \left(7.\overline{714285}\right)_{10}. \quad \square\end{aligned}$$

1.2. Encontrar la expresión binaria de los números decimales 0.1 y $5.\overline{3}$.

SOLUCIÓN. En primer lugar, veamos el algoritmo general para convertir un número decimal x con $0 < x < 1$ a binario en la forma

$$x = (0.\alpha_1\alpha_2\ldots\alpha_p\ldots)_2.$$

© Ediciones Pirámide

Como

$$x = \frac{\alpha_1}{2} + \frac{\alpha_2}{2^2} + \cdots + \frac{\alpha_p}{2^p} + \cdots$$

se tiene que

$$\alpha_1 = 2x - \left(\frac{\alpha_2}{2} + \frac{\alpha_3}{2^2} + \cdots + \frac{\alpha_{p+1}}{2^p} + \cdots\right) = 2x - (0.\alpha_2\alpha_3 \ldots \alpha_{p+1} \ldots)_2,$$

es decir, la primera cifra decimal α_1 de x en base 2 se halla tomando la parte entera de $2x$. Ahora bien, como el número $(0.\alpha_2\alpha_3 \ldots \alpha_{p+1} \ldots)_2$ está también entre 0 y 1, para hallar la siguiente cifra α_2 basta repetir el proceso y, así, sucesivamente. De esta forma, a partir de la siguiente tabla

$0.1 \times 2 = 0.2$	$\rightarrow$	0
$0.2 \times 2 = 0.4$	$\rightarrow$	0
$0.4 \times 2 = 0.8$	$\rightarrow$	0
$0.8 \times 2 = 1.6$	$\rightarrow$	1
$0.6 \times 2 = 1.2$	$\rightarrow$	1
$0.2 \times 2 = 0.4$	$\rightarrow$	0
$\cdots$	$\cdots$	$\cdots$

se obtiene que

$$(0.1)_{10} = (0.0\overbrace{0011})_2$$

y, como $5.\overbrace{3} = 5 + 0.\overbrace{3}$, a partir de

$0.\overbrace{3} \times 2 = 0.\overbrace{6}$	$\rightarrow$	0
$0.\overbrace{6} \times 2 = 1.\overbrace{3}$	$\rightarrow$	1
$0.\overbrace{3} \times 2 = 0.\overbrace{6}$	$\rightarrow$	0
$\cdots$	$\cdots$	$\cdots$

se concluye que

$$(5.\overbrace{3})_{10} = (5)_{10} + (0.\overbrace{3})_{10} = (101)_2 + (0.\overbrace{01})_2 = (101.\overbrace{01})_2. \quad \square$$

1.3. Determinar los números decimales que en simple precisión tienen la siguiente representación en coma flotante estándar:

a) 11100101110010010000000000000000.

b) 00000011110001111000000000000000.

c) 11111111100000000000000000000000.

d) 10000000010000000000000000000000.

© Ediciones Pirámide

Solución.

a) $-1.1001001 \times 2^{203-127} = -1.1001001 \times 2^{76} \simeq -1.186494578820998 \times 10^{23}$.

b) $1.10001111 \times 2^{7-127} = 1.10001111 \times 2^{-120} \simeq 1.172555614945232 \times 10^{-36}$.

c) $-\infty$.

d) $-0.1 \times 2^{-126} = -1.0 \times 2^{-127} \simeq -5.877471754111438 \times 10^{-39}$. □

1.4. Supongamos que tenemos un ordenador que almacena los números en base 10 con tan sólo dos dígitos de mantisa. Queremos calcular con esta máquina la menor raíz de la ecuación $x^2 - 20x + 1 = 0$.

a) ¿Qué valor se obtendría al calcularla como $10 - \sqrt{99}$?

b) Ídem calculándola como $\dfrac{1}{10 + \sqrt{99}}$.

Solución.

a) Las raíces de la ecuación $x^2 - 20x + 1 = 0$ son

$$10 - \sqrt{99} = 0.05012562893429393\ldots \text{ y } 10 + \sqrt{99} = 19.94987437106570\ldots .$$

Trabajando en una máquina con una mantisa de 2 dígitos decimales, 99 es un número máquina para ella (9.9×10^1). Como

$$\sqrt{99} = 9.949874371065706\ldots$$

se tiene que $r(\sqrt{99}) = 9.9 \times 10^0$, con lo que, al ser también 10 un número máquina $(10 = 1.0 \times 10^1)$,

$$r(10 - \sqrt{99}) = 10 - 9.9 = 0.1.$$

b) Podemos evitar la cancelación anterior escribiendo

$$10 - \sqrt{99} = \frac{1}{10 + \sqrt{99}}.$$

Puesto que $r\left(10 + \sqrt{99}\right) = r\left(10 + 9.9\right) = 20$, dado que 19.9 no es un número máquina y hay que tomar su redondeo, se verifica

$$r\left(\frac{1}{10 + \sqrt{99}}\right) = r\left(\frac{1}{r\left(10 + \sqrt{99}\right)}\right) = r\left(\frac{1}{20}\right) = 0.05.$$

Nótese que éste es el mejor resultado que puede obtenerse en esa máquina para el cálculo de $10 - \sqrt{99}$ puesto que coincide con el redondeo del resultado exacto. □

© Ediciones Pirámide

1.5. Se considera el problema de sumar tres números reales $x, y, z \in \mathbb{R}$ con el algoritmo

$$(x + y) + z,$$

es decir, se suman los dos primeros y al resultado obtenido se le suma el tercero. Demostrar que este algoritmo es regresivamente estable.

SOLUCIÓN. Basta observar que, a partir de la relación (1.4), se tiene que

$$\begin{aligned}(x \oplus y) \oplus z &= r\left((x \oplus y) + r(z)\right) = (1+\delta_5)\left((x \oplus y) + (1+\delta_4)z\right)\\ &= (1+\delta_5)\left((1+\delta_3)(x+y) + (1+\delta_3)\delta_1 x + (1+\delta_3)\delta_2 y + (1+\delta_4)z\right)\\ &= \widetilde{x} + \widetilde{y} + \widetilde{z}\end{aligned}$$

donde

$$|\delta_i| \leq \frac{eps}{2}$$

para $i = 1, 2, 3, 4, 5$, y los números

$$\begin{cases} \widetilde{x} = (1 + \delta_1 + \delta_3 + \delta_5 + \delta_1\delta_3 + \delta_1\delta_5 + \delta_3\delta_5 + \delta_1\delta_3\delta_5)\, x \\ \widetilde{y} = (1 + \delta_2 + \delta_3 + \delta_5 + \delta_2\delta_3 + \delta_2\delta_5 + \delta_3\delta_5 + \delta_2\delta_3\delta_5)\, y \\ \widetilde{z} = (1 + \delta_4 + \delta_5 + \delta_4\delta_5)\, z \end{cases}$$

verifican

$$\left|\frac{\widetilde{x} - x}{x}\right| = O(eps), \left|\frac{\widetilde{y} - y}{y}\right| = O(eps) \text{ y } \left|\frac{\widetilde{z} - z}{z}\right| = O(eps). \quad \square$$

1.6. Estudiar el condicionamiento de la suma de n números $x_1 + x_2 + \cdots + x_n$.

SOLUCIÓN. Consideremos $\widetilde{x}_i$, perturbaciones de x_i,

$$\widetilde{x}_i = x_i + \varepsilon_i$$

con $i = 1, 2, \ldots, n$. El error absoluto en la suma es

$$\left|\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \widetilde{x}_i\right| = \left|\sum_{i=1}^{n} (x_i - \widetilde{x}_i)\right| = \left|\sum_{i=1}^{n} \varepsilon_i\right|$$

y una cota del error relativo viene dada por

$$\frac{\left|\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \widetilde{x}_i\right|}{\left|\sum_{i=1}^{n} x_i\right|} = \frac{\left|\sum_{i=1}^{n} \varepsilon_i\right|}{\left|\sum_{i=1}^{n} x_i\right|} \leq \frac{\sum_{i=1}^{n} |\varepsilon_i|}{\left|\sum_{i=1}^{n} x_i\right|} = \sum_{i=1}^{n} \frac{|x_i|}{\left|\sum_{i=1}^{n} x_i\right|} \left|\frac{\varepsilon_i}{x_i}\right|$$

© Ediciones Pirámide

por lo que los números de condición son

$$\kappa(x_i) = \frac{|x_i|}{\left|\sum_{i=1}^{n} x_i\right|}$$

para $i = 1, 2, \ldots, n$. Si todos los números $\{x_1, x_2, \ldots, x_n\}$ tienen el mismo signo entonces el problema está bien condicionado. En otro caso, si

$$\sum_{i=1}^{n} x_i \simeq 0$$

los números de condición pueden ser muy grandes, lo que ocurre, en particular, cuando los números están próximos a cancelarse entre sí. □

1.6.2. Problemas propuestos

1.7. *Arquímedes* (278–212 a.C.) obtuvo las siguientes acotaciones del número π

$$\frac{223}{71} < \pi < \frac{22}{7}.$$

Determinar los errores absolutos y relativos cometidos en esas aproximaciones.

1.8. Algunos ordenadores utilizan, en lugar del sistema binario, el sistema hexadecimal, es decir, utilizan como base el 16 y los dígitos que se emplean son

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

Encontrar la expresión decimal de los números hexadecimales E, 1A, A9B.A1 y A.$\widehat{\mathrm{A}}$, así como su expresión binaria. Comprobar lo cómodo que resulta expresar números hexadecimales en binario y viceversa.

1.9. Hallar la representación en coma flotante estándar en precisión simple de:

a) Los números máquina de los problemas 1.1 y 1.2.

b) El redondeo de los números de los problemas 1.1 y 1.2 que no son números máquina.

1.10. Hallar el número de condición para las siguientes funciones:

$$f(x) = x^{\alpha}\ (\alpha \in \mathbb{R}),\ g(x) = \operatorname{sen} x \ \text{ y } \ h(x) = e^{x}.$$

© Ediciones Pirámide

1.11. Se considera el problema de sumar n números reales $x_1, x_2, \ldots, x_n \in \mathbb{R}$ con el algoritmo

$$\begin{cases} s_1 = x_1 \\ s_i = s_{i-1} + x_i,\ i = 2, 3, \ldots, n. \end{cases}$$

Demostrar que el algoritmo anterior es regresivamente estable.

1.7. Prácticas

1.1. Teniendo en cuenta que `MATLAB` trabaja en doble precisión, calcular el número máquina inmediatamente anterior a 1 y comprobar que dista 2^{-53} de 1.

1.2. Calcular $1 \oplus 2^{-52}$, $1 \oplus 2^{-53}$, $1 \ominus 2^{-53}$ y $1 \ominus 2^{-54}$ y comprobar que los resultados coinciden con los que teóricamente deben obtenerse.

1.3. Determinar el mayor y menor número máquina normal positivo, así como el mayor y menor número subnormal positivo, cuando se trabaja en simple precisión. Comprobar que coinciden con los expuestos en la tabla 1.1.

1.4. Ídem para doble precisión. Comparar con los comandos `realmax` y `realmin` de `MATLAB`.

1.5. Hallar la suma y la resta de dos números con exponentes muy dispares y comprobar que el resultado obtenido concuerda con el previsto según la aritmética en coma flotante estándar.

1.6. Hallar las potencias sucesivas de 10 con exponente desde 1 hasta 309. Comprobar que el valor de 10^{309} es infinito y que si $x = 10^{309}$ entonces $\dfrac{1}{x} = 0$.

1.7. Determinar el épsilon de la máquina. Para ello, calcular $1 + x$ con $x = 2^{-i}$ para $i = 1, 2, \ldots$ mientras que $1 + x > 1$. Comparar con el comando `eps` de `MATLAB`.

1.8. Consideremos la *sucesión de Fibonacci*

$$\begin{cases} x_0 = 1,\ x_1 = 1 \\ x_{n+1} = x_n + x_{n-1},\ n \in \mathbb{N}. \end{cases}$$

a) A partir de la sucesión $\{x_n\}_{n=0}^{\infty}$ se define

$$y_n = \frac{x_n}{x_{n-1}},\ n \in \mathbb{N}.$$

© Ediciones Pirámide

Obtener la ley de recurrencia de la sucesión $\{y_n\}_{n=1}^{\infty}$ y demostrar que

$$\lim_{n\to+\infty} y_n = \frac{1+\sqrt{5}}{2}.$$

El número $\phi = \dfrac{1+\sqrt{5}}{2} \simeq 1.61803$ se denomina *razón áurea*.

b) Determinar los errores absoluto y relativo,

$$|y_n - \phi| \quad \text{y} \quad \left|\frac{y_n - \phi}{\phi}\right|,$$

para valores $n = 1, 2, \ldots, 100$.

1.9. Se considera la sucesión de números reales $\{\alpha_n\}_{n=0}^{\infty}$ siendo

$$\alpha_n = \int_0^1 \frac{x^n}{x+10}\, dx,\ n \in \mathbb{N} \cup \{0\}.$$

a) Demostrar que los números α_n verifican la ley de recurrencia

$$\alpha_n + 10\alpha_{n-1} = \frac{1}{n},\ n \in \mathbb{N}.$$

b) Comprobar que los valores de $\{\alpha_0, \alpha_1, \ldots, \alpha_{25}\}$ obtenidos a partir de las secuencias

$$\begin{cases} \alpha_0 = \log\dfrac{11}{10} \\ \alpha_n = \dfrac{1}{n} - 10\alpha_{n-1}, \quad n = 1, 2, \ldots, 25 \end{cases} \tag{1.10}$$

y

$$\begin{cases} \alpha_{25} = 3.509 \times 10^{-3} \\ \alpha_{n-1} = \dfrac{1}{10}\left(\dfrac{1}{n} - \alpha_n\right), \quad n = 25, 24, \ldots, 1 \end{cases} \tag{1.11}$$

son los que se obtienen en la tabla 1.5. Dar una explicación a los mismos.

1.10. Aproximar la derivada de la función $\operatorname{sen} x$ en $x = 1$ mediante la fórmula

$$\frac{\operatorname{sen}(1+h) - \operatorname{sen} 1}{h}$$

con $h = 10^{-i}$ para $i = 5, 6, \ldots, 16$ comparando los resultados obtenidos con el valor exacto como se hace en la tabla 1.2. Comprobar la pérdida de precisión por cancelación.

© Ediciones Pirámide