

Métodos Numéricos

Departamento de Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Lección 1: Introducción al análisis numérico

Índice

1. Errores	2
1.1. Errores relativos y errores absolutos	2
1.2. Aritmética de punto flotante	3
1.3. El formato IEEE de doble precisión	6
2. Condicionamiento y estabilidad en análisis numérico	10
2.1. Condicionamiento de los problemas numéricos	10
2.2. Estabilidad de los métodos numéricos	12
3. Apéndice A: Estructura del sistema numérico de punto flotante	20
4. Apéndice B: Dígitos significativos	22
5. Apéndice C: Dígitos significativos correctos	24
6. Apéndice D: Propagación del error	26
7. Soluciones de los proyectos	29

El análisis numérico es la rama de las matemáticas que se encarga de encontrar aproximaciones a las soluciones de problemas cuyos resultados exactos son, o bien imposibles, o bien muy complicados de calcular. Puesto que las soluciones obtenidas mediante los métodos del análisis numérico son de carácter aproximado, se hace necesario controlar su nivel de precisión. Por lo tanto, además de proporcionar la aproximación a la solución, los procedimientos del análisis numérico deben establecer un límite realista para el error asociado.

Entre los cometidos del análisis numérico se encuentra el diseño de algoritmos que permitan estimar, con un nivel razonable de precisión, las soluciones de los problemas tratados. Los algoritmos están contruidos de forma que se puedan implementar computacionalmente y constan de una secuencia de operaciones simples que conducen en última instancia a la aproximación de la solución.

El avance extraordinario del análisis numérico ha ido de la mano del excepcional desarrollo que han experimentado los ordenadores desde su nacimiento en los años cuarenta del siglo veinte. El tratamiento computacional de los algoritmos propuestos por el análisis numérico se ha convertido en una herramienta esencial, debido al número considerablemente alto de operaciones y manipulaciones de datos implicados en los procedimientos numéricos.

1. Errores

En la creación de modelos matemáticos susceptibles de tratamiento computacional, los errores pueden proceder de muy diversas fuentes. Hay errores que se originan en la propia creación del modelo. Ejemplo de ello son los errores que se derivan de las imperfecciones inherentes a las medidas físicas. Por otro lado, están los errores específicamente computacionales, que surgen al implementar los distintos métodos matemáticos. En este curso nos fijaremos exclusivamente en este último tipo de errores.

1.1. Errores relativos y errores absolutos

Resolver un problema numérico significa, a grandes rasgos, obtener cierta aproximación x_a a cierto valor teórico o exacto x_e .

Si x_e y x_a son números reales¹, se define el **error absoluto** cometido al aproximar x_e por x_a como

$$\text{Abs}(x_a) := |x_e - x_a|.$$

Para muchos propósitos, sin embargo, es preferible considerar el porcentaje de error en x_a , o **error relativo**, dado por

$$\text{Rel}(x_a) := \left| \frac{x_e - x_a}{x_e} \right|, \quad x_e \neq 0.$$

En términos generales, la utilidad de manejar el error relativo radica en que nos protege contra afirmaciones precipitadas sobre la bondad de una aproximación, sobre todo cuando nos movemos en escalas extremas².

¹Si x_e y x_a son vectores, los errores absoluto y relativo se definen mediante la norma de vector, esto es,

$$\text{Abs}(x_a) := \|x_e - x_a\|, \quad \text{Rel}(x_a) := \frac{\|x_e - x_a\|}{\|x_e\|}, \quad x_e \neq 0.$$

Los errores absoluto y relativo en el caso de matrices se definen por medio de la norma de matriz.

²Es decir, cuando la magnitud, medida en valor absoluto o en norma, es muy grande o muy pequeña.

Ejemplo 1.1 ► Errores absoluto y relativo

Consideremos el problema de aproximar $x_e = 8.8861105216 \times 10^6$. Debido al tamaño de x_e , incluso aproximaciones intuitivamente buenas dan lugar a errores absolutos apreciables. Supongamos que $x_a = 8.8861105207 \times 10^6$ es una aproximación de x_e . Se tiene que $|x_e - x_a| = 9 \times 10^{-4}$, a pesar de que x_e y x_a comparten nueve dígitos. Sin embargo, el error relativo

$$\left| \frac{x_e - x_a}{x_e} \right| \approx 1.0128 \times 10^{-10}.$$

proporciona una información más realista de la bondad de la aproximación.

Otro tanto ocurre cuando los números son pequeños en valor absoluto. En efecto, consideremos ahora $x_e = 3.4120569685 \times 10^{-9}$ y $x_a = 3.4120569678 \times 10^{-9}$. Se tiene que $|x_e - x_a| = 7 \times 10^{-19}$. De nuevo, el error relativo

$$\left| \frac{x_e - x_a}{x_e} \right| \approx 2.0515 \times 10^{-10}$$

proporciona una información más ajustada a la realidad, puesto que x_e y x_a comparten nueve dígitos.

1.2. Aritmética de punto flotante

En el tratamiento computacional de problemas matemáticos, una fuente básica de error surge al considerar la representación de un número en un ordenador. Cuando escribimos 3.1416, lo que estamos realmente representando es el número racional

$$3 + \frac{1}{10} + \frac{4}{10^2} + \frac{1}{10^3} + \frac{6}{10^4}.$$

Utilizando series numéricas, este proceso conduce al conocido concepto de **representación binaria** o en base 2. En concreto, si $x \in (0, 1]$, siempre existe una sucesión $\{d_j\}$, con $d_j \in \{0, 1\}$, tal que

$$x = \frac{d_1}{2} + \frac{d_2}{2^2} + \cdots + \frac{d_j}{2^j} + \cdots.$$

La sucesión $\{d_j\}$ es esencialmente única³ y permite, por tanto, introducir la notación

$$x = (0.d_1d_2 \dots d_j \dots)_2.$$

La restricción al intervalo $(0, 1]$ no es relevante. En efecto, si $x \in \mathbb{R}$ y $x \neq 0$, existen un número entero e denominado **exponente** y una sucesión $\{d_j\}$, con $d_1 = 1$ y $d_j \in \{0, 1\}$, para $j \geq 2$, tales que

$$x = \text{sig}(x) \times (1.d_2 \dots d_j \dots)_2 \times 2^e,$$

donde $\text{sig}(x)$ denota el signo de x . Los números d_j son conocidos como «bits»⁴.

³Algunos números admiten dos representaciones diferentes. Por ejemplo, $0.\widehat{9} = 1$ (en base 10) y $0.0\widehat{1} = 0.1$ (en base 2). En general, todo número no nulo con parte decimal finita tiene, en cualquier base entera $\beta \geq 2$, un «gemelo» en el que figura $\beta - 1$ infinitas veces.

⁴Los números d_j se denominan «dígitos» o «cifras» cuando la base es 10.

Puesto que la memoria de cualquier ordenador es finita, es evidente que no todos los bits d_j de la representación anterior pueden almacenarse. De hecho, los ordenadores manejan exclusivamente números reales cuyas representaciones binarias constan de un número finito t de bits y cuyo exponente pertenece a cierto intervalo finito $[e_{\min}, e_{\max}]$. Se trata de los denominados **números en punto flotante** (*floating point*) o números de la máquina. El conjunto de tales números suele representarse por

$$\mathbb{M}(2, t, e_{\min}, e_{\max})$$

y se conoce como **sistema numérico de punto flotante** o conjunto de números de la máquina⁵. De este modo, si x es un número en punto flotante, esto es, si $x \in \mathbb{M}$, se tiene que⁶

$$x = \text{sig}(x) \times (1.d_2 \dots d_t)_2 \times 2^e$$

La secuencia $(d_2 \dots d_t)_2$ es conocida como **fracción**. La parte entera junto con la fracción $(1.d_2 \dots d_t)_2$ se denomina **coeficiente**⁷ del número.

Dado un número real x arbitrario, es muy probable que x no pertenezca a \mathbb{M} . Para encontrar un «sustituto» de x que esté en \mathbb{M} , se utiliza el **redondeo** de x a t bits. La idea es simplemente determinar el número en punto flotante más próximo a x .

En concreto, consideremos un número real x cuya representación binaria sea

$$x = \text{sig}(x) \times (1.d_2 \dots d_t d_{t+1} \dots)_2 \times 2^e.$$

El número en punto flotante más próximo a x se denota por $\text{fl}(x)$ y se obtiene redondeando x a t bits:

$$\text{fl}(x) = \text{sig}(x) \times 2^e \times \begin{cases} (1.d_2 \dots d_t)_2, & \text{si } d_{t+1} = 0, \\ (1.d_2 \dots d_t)_2 + 2^{-(t-1)}, & \text{si } d_{t+1} = 1. \end{cases}$$

Notemos que en el caso en que d_{t+1} sea igual a uno, el redondeo puede expresarse como⁸

$$\text{fl}(x) = (1.d_2 \dots d_t)_2 + (0.0 \overset{t}{\dots} 01)_2.$$

Ejemplo 1.2 ► Representación en punto flotante

Consideremos el número $x = \frac{91}{8}$ y el sistema numérico de punto flotante $\mathbb{M}(2, 6)$. Vamos a comprobar en primer lugar que x no pertenece a $\mathbb{M}(2, 6)$. Operando, se tiene que

$$\frac{91}{8} = 11.375 = 2^3 + 2 + 1 + \frac{1}{2^2} + \frac{1}{2^3},$$

lo que nos permite expresar este número en base 2. Así,

$$x = (1011.011)_2 = (1.011011)_2 \times 2^3.$$

El coeficiente de x consta de siete bits, por lo que, efectivamente, x no pertenece

⁵Si no hay ambigüedad, se puede representar de forma más simple por $\mathbb{M}(2, t)$ o por \mathbb{M} .

⁶Se trata de la denominada representación en punto flotante «normalizada». Esta representación es única.

⁷O «mantisa». No obstante, para evitar confusión, se desaconseja el uso de este término, dado que denota también la parte fraccionaria de un logaritmo. En el ámbito anglosajón, el vocablo más utilizado es *significand*.

⁸La letra t indica el número total de bits de los que consta el coeficiente, no la fracción.

a $\mathbb{M}(2, 6)$. Nuestro objetivo es, por tanto, calcular $\text{fl}(x)$, esto es, el número en punto flotante (en otras palabras, el elemento de $\mathbb{M}(2, 6)$) más próximo a x . Para ello, efectuamos el redondeo de x a 6 bits:

$$\begin{aligned}\text{fl}(x) &= ((1.01101)_2 + 2^{-5}) \times 2^3 \\ &= ((1.01101)_2 + (0.00001)_2) \times 2^3 \\ &= (1.01110)_2 \times 2^3 \in \mathbb{M}(2, 6).\end{aligned}$$

Desde el punto de vista del error relativo, el proceso de redondeo está controlado por la denominada **unidad de redondeo** $u := 2^{-t}$. Específicamente, si $x \in \mathbb{R}$, $x \neq 0$, y escribimos

$$\text{fl}(x) = x(1 + \delta),$$

se puede probar que el error de redondeo $\text{Rel}(\text{fl}(x))$ está acotado por u , es decir,

$$\text{Rel}(\text{fl}(x)) = \left| \frac{\text{fl}(x) - x}{x} \right| = |\delta| \leq u.$$

En muchos contextos, en lugar de u , suele manejarse otra cantidad, $\varepsilon_{\mathbb{M}}$, llamada **precisión de la máquina**⁹ (*machine epsilon*) o «épsilon» de la máquina, cuyo valor es¹⁰ $\varepsilon_{\mathbb{M}} = 2u$, esto es, $\varepsilon_{\mathbb{M}} = 2^{1-t}$.

Proyecto 1.1 (solución en página 29)

La particularidad del valor $\varepsilon_{\mathbb{M}}$ es que es exactamente la distancia del número uno al número siguiente^a a uno representable en \mathbb{M} . Demuéstralo.

^aSiguiente en el sentido de consecutivo mayor.

MATLAB® opera principalmente (véase la nota 14) con el sistema numérico de punto flotante $\mathbb{M}(2, 53)$ y, por tanto, la correspondiente unidad de redondeo vale

$$u = 2^{-53} \approx 1.1102 \times 10^{-16}.$$

El programa incorpora la variable especial **eps**, que representa el épsilon de la máquina.

Desde un punto de vista práctico, el valor de **eps** nos indica el máximo nivel de precisión que podemos alcanzar al trabajar en MATLAB®. Puesto que **eps** está en torno a 10^{-16} , no es posible obtener valores numéricos en MATLAB® con más de 16 dígitos (o cifras) significativos correctos (o exactos)¹¹.

⁹Es usual llamar «precisión» también a t , el número de bits. Ambas cantidades, t y $\varepsilon_{\mathbb{M}}$, expresan diferentes formas de medir el nivel de exactitud que se puede alcanzar en $\mathbb{M}(2, t)$.

¹⁰En general, cuando se considera el sistema numérico de punto flotante $\mathbb{M}(\beta, t)$, donde $\beta \in \mathbb{Z}$, $\beta \geq 2$, los valores de la unidad de redondeo y de la precisión de la máquina son, respectivamente,

$$u = \frac{1}{2}\beta^{1-t}, \quad \varepsilon_{\mathbb{M}} = \beta^{1-t}.$$

¹¹Las reglas para determinar si un dígito es o no significativo figuran en el apéndice B. En el apéndice C se pueden encontrar resultados sobre dígitos significativos correctos.

Las operaciones aritméticas (suma, resta, multiplicación y división) no pueden ser reproducidas de forma exacta en un ordenador. Es necesario, por tanto, definir en \mathbb{M} unos «sustitutos» de estas operaciones. Surgen, así, las denominadas «operaciones en punto flotante» o *flop* (de *F*loating *O*perations), que dan forma a la **aritmética de punto flotante** o aritmética de precisión finita.

En concreto, dados $x, y \in \mathbb{M}$, el valor $x \text{ op } y$ (donde *op* denota cualquiera de las operaciones aritméticas, esto es, $\text{op} = +, -, *, /$) no tiene necesariamente que pertenecer a \mathbb{M} . Las correspondientes operaciones en punto flotante, denotadas por op_{fl} ($= +_{\text{fl}}, -_{\text{fl}}, *_{\text{fl}}, /_{\text{fl}}$) y definidas en la forma

$$x \text{ op}_{\text{fl}} y := \text{fl}(x \text{ op } y),$$

salvan esta dificultad, ya que su resultado es siempre un número de \mathbb{M} ¹². El redondeo de $x \text{ op } y$ sigue, lógicamente, el mismo patrón de control del error relativo cometido, es decir,

$$\text{fl}(x \text{ op } y) = (x \text{ op } y)(1 + \delta), \quad |\delta| \leq u.$$

Puede comprobarse además que las leyes habituales de la aritmética exacta, como, por ejemplo, la propiedad asociativa de la suma, no se verifican en general en este contexto de aritmética de punto flotante.

Ejercicio 1.1

Usando *format long*, ejecuta en MATLAB® la expresión $3 \left(\frac{4}{3} - 1\right) - 1$ y comprueba que el resultado no es cero. ¿Qué crees que está ocurriendo?

1.3. El formato IEEE de doble precisión

El IEEE (*Institute of Electrical and Electronics Engineers*) es una asociación del mundo de la Ingeniería dedicada, entre otras muchas cosas, a la estandarización. En 1985, el IEEE creó el denominado **formato IEEE de doble precisión** como parte de un estándar conocido como IEEE 754-2019¹³. El formato IEEE de doble precisión emplea $\mathbb{M}(2, 53)$ como sistema numérico de punto flotante y establece el procedimiento para representar los números de este conjunto. Por defecto, MATLAB® utiliza este formato¹⁴.

El sistema numérico de punto flotante $\mathbb{M}(2, 53)$ maneja los números

$$x = \text{sig}(x) \times (1.d_2d_3 \dots d_{53})_2 \times 2^e, \quad (1)$$

donde el exponente e varía en un rango que especificaremos de inmediato.

En el formato IEEE de doble precisión cada uno de estos números se representa mediante una secuencia de 64 bits¹⁵, de forma que

- 1 bit se destina al signo (en concreto, 0 si el número es positivo y 1 si es negativo);
- 52 bits se emplean en la fracción;
- los 11 bits restantes se asignan al exponente.

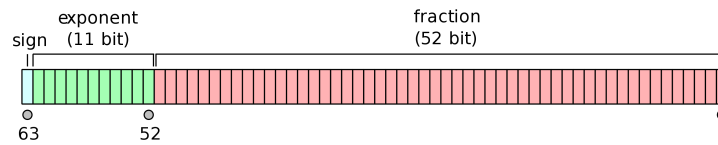
¹²En general, si $x, y \in \mathbb{R}$, las operaciones en punto flotante vienen dadas por $x \text{ op}_{\text{fl}} y = \text{fl}(\text{fl}(x) \text{ op } \text{fl}(y))$.

¹³El estándar ha sido objeto de revisiones posteriores; la última, en 2019.

¹⁴MATLAB® puede utilizar, si así se lo indicamos, el sistema numérico de «precisión simple» $\mathbb{M}(2, 24)$, cuyos números se representan mediante secuencias de 32 bits.

¹⁵El bit $d_1 = 1$ no se incluye en la secuencia; se trata del denominado «bit oculto».

Los 64 bits se disponen en el orden que muestra la siguiente imagen:



Ahora bien, lo que realmente se almacena en los 11 bits del exponente no es e , sino un valor sesgado¹⁶,

$$e_s := e + 1023.$$

Al número 1023 se le conoce como «sesgo» del exponente. El valor de e_s puede ir desde 0 (todos los bits nulos) hasta 2047 (todos los bits iguales a 1):

```
Command Window
>> bin2dec('11111111111') % Once veces el bit 1
ans =
    2047
```

Por tanto, el exponente e varía entre -1023 y $+1024$. Los valores

- $e = -1023$ ($e_s = 0$),
- $e = +1024$ ($e_s = 2047$),

se reservan para casos especiales, que enseguida detallaremos.

Ejemplo 1.3 ► Representación en formato IEEE de doble precisión

Vamos a representar el número 0.25 en formato IEEE de doble precisión, es decir, como una secuencia de 64 bits.

En primer lugar, escribimos 0.25 en base 2:

$$0.25 = \frac{1}{4} = \frac{1}{2^2} = (0.01)_2.$$

A continuación, expresamos $(0.01)_2$ como número en punto flotante, esto es, como elemento del conjunto $\mathbb{M}(2, 53)$, al cual pertenece:

$$(0.01)_2 = (1.00 \overset{53}{\dots} 00)_2 \times 2^{-2}.$$

Una vez introducido el sesgo, el exponente a almacenar es

$$e = -2 + 1023 = 1021 = (0111111101)_2.$$

Por tanto, la representación de 0.25 en formato IEEE de doble precisión es

$$0 \mid 0111111101 \mid 00 \overset{52}{\dots} 00.$$

¹⁶El sesgo se introduce para evitar exponentes negativos. Los exponentes sesgados optimizan los algoritmos de comparación de números.

El menor número positivo que maneja MATLAB® puede obtenerse con la variable `realmin` y el mayor, con `realmax`¹⁷. En concreto,

```

Command Window
>> realmin

ans =

    2.2251e-308

>> realmax

ans =

    1.7977e+308

```

Estos valores se deducen fácilmente si tenemos presente (1):

$$\text{realmin} = (1.0 \overset{53}{\dots} 0)_2 \times 2^{-1022} = 2^{-1022} \approx 2.2251 \times 10^{-308}.$$

$$\begin{aligned} \text{realmax} &= (1.1 \overset{53}{\dots} 1)_2 \times 2^{1023} = \left(1 + \sum_{j=1}^{52} \frac{1}{2^j}\right) \times 2^{1023} \\ &= \left(1 + \frac{\frac{1}{2}(1 - (\frac{1}{2})^{52})}{1 - \frac{1}{2}}\right) \times 2^{1023} \\ &= (2 - 2^{-52}) \times 2^{1023} = (2 - \varepsilon_{\mathbb{M}}) \times 2^{1023} \approx 1.7977 \times 10^{308}. \end{aligned}$$

Cuando se obtiene o se genera un valor superior a `realmax` se produce el fenómeno denominado **overflow**. Cuando eso ocurre, se asigna a ese resultado un valor excepcional denominado *infinite* (Inf o inf en MATLAB®). Por otro lado, cuando se manejan valores inferiores a `realmin` se produce el fenómeno conocido como **underflow** y, en ese caso, la regla usual es considerar dicho resultado como cero¹⁸.

Para manejar indeterminaciones del tipo Inf/Inf, 0/0, Inf–Inf, ... se introduce otro valor excepcional denominado *Not-a-Number* (NaN en MATLAB®).

¹⁷Los números pertenecientes al conjunto $[-\text{realmax}, -\text{realmin}] \cup [\text{realmin}, \text{realmax}]$ se denominan números «normales».

¹⁸No obstante, desde la versión 7 de MATLAB® en adelante aparecen ciertos valores inferiores a `realmin` llamados números «denormales» o «desnormalizados», cuyo objeto es simular una aproximación a cero menos abrupta. En el formato IEEE de doble precisión estos números se representan con exponente nulo y fracción distinta de cero, pero se asume que el bit oculto es 0, en lugar de 1. Una representación lógica de los números denormales sería, pues, $\pm(0.f)_2 \times 2^{-1023}$, donde $f \neq 0$ denota la fracción. Sin embargo, el estándar utiliza como exponente $1 - 1023 = -1022$ en lugar de -1023 , de modo que estos números se describen en la forma

$$\pm(0.f)_2 \times 2^{1-1023} = \pm(0.f)_2 \times 2^{-1022},$$

con $f \neq 0$. No hay una razón obvia para preferir este exponente, salvo, quizás, que al hacerlo de esta manera el salto entre el mayor número denormal y el menor número normal positivo es más pequeño. Los números denormales positivos menor y mayor son, respectivamente, `eps*realmin` y `(1-eps)*realmin`. Así, todo cálculo cuyo resultado sea inferior a `eps*realmin` se redondea a cero (auténtico *underflow*) y todo cálculo que esté entre `eps*realmin` y `(1-eps)*realmin` se redondea a un número denormal.

Como se señaló más arriba, los exponentes $e_s = 0$ ($e = -1023$) y $e_s = 2047$ ($e = +1024$) se reservan para casos especiales. En concreto,

- $e = -1023$ ($e_s = 0$) se reserva para cero¹⁹, si la fracción es nula, y para los números denormales (véase la nota 18), si la fracción es distinta de cero.
- $e = +1024$ ($e_s = 2047$) se reserva para los símbolos **Inf**, si la fracción es nula, y **NaN**, si la fracción es distinta de cero.

La siguiente tabla reúne de forma esquemática los resultados vistos anteriormente:

Exponente en base 2	Exponente en base 10 con sesgo	Exponente en base 10 sin sesgo	Fracción en base 2	Elementos del sistema numérico de punto flotante $\mathbb{M}(2, 53)$
00...00	0	-1023	00...00	± 0
00...00	0	-1023	$d_2 d_3 \dots d_{53}$ $\exists i: d_i \neq 0$	$\pm(0.00\dots 01)_2 \times 2^{1-1023} = \pm \text{eps} * \text{realmin}$ \vdots $\pm(0.11\dots 11)_2 \times 2^{1-1023} = \pm(1-\text{eps}) * \text{realmin}$
00...01	1	$e_{\text{mín}} = -1022$	$d_2 d_3 \dots d_{53}$ $d_i \in \{0,1\}$	$\pm(1.00\dots 00)_2 \times 2^{-1022} = \pm \text{realmin}$ $\pm(1.00\dots 01)_2 \times 2^{-1022}$ \vdots $\pm(1.11\dots 11)_2 \times 2^{-1022}$
00...10 \vdots 11...01	2 \vdots 2045	-1021 \vdots 1022	$d_2 d_3 \dots d_{53}$ $d_i \in \{0,1\}$	$\pm(1.00\dots 00)_2 \times 2^{-1021}$ \vdots $\pm(1.11\dots 11)_2 \times 2^{1022}$
11...10	2046	$e_{\text{máx}} = 1023$	$d_2 d_3 \dots d_{53}$ $d_i \in \{0,1\}$	$\pm(1.00\dots 00)_2 \times 2^{1023}$ \vdots $\pm(1.11\dots 10)_2 \times 2^{1023}$ $\pm(1.11\dots 11)_2 \times 2^{1023} = \pm \text{realmax}$
11...11	2047	1024	00...00	$\pm(1.00\dots 00)_2 \times 2^{1024} \rightarrow \pm \text{Inf}$
11...11	2047	1024	$d_2 d_3 \dots d_{53}$ $\exists i: d_i \neq 0$	$\pm(1.d_2 d_3 \dots d_{53})_2 \times 2^{1024} \rightarrow \text{NaN}$

 Números normales
 Números denormales
 Símbolos especiales

Proyecto 1.2 (solución en página 31)

Explica brevemente qué hace cada uno de los tres siguientes conjuntos de instrucciones al ser ejecutados en MATLAB®:

```
x=1;
while 1+x > 1
    x=x/2;
end
```

```
x=1;
while x+x > x
    x=2*x;
end
```

```
x=1;
while x+x > x
    x=x/2;
end
```

¹⁹El número cero no admite una representación en punto flotante normalizada, de ahí que sea necesario elegir un número de \mathbb{M} que lo suplante.

Es interesante mencionar finalmente que la orden de MATLAB® `format hex` permite visualizar en pantalla la representación del formato IEEE de doble precisión. En concreto, MATLAB® agrupa los 64 bits de los que consta este formato en bloques de cuatro bits, creando los llamados bloques hexadecimales. Puesto que en cada uno de estos bloques pueden aparecer todos los números del 0 al 15, MATLAB® utiliza la representación hexadecimal para mostrarlos²⁰. De este modo, basta con 16 símbolos hexadecimales para representar cada número escrito en formato IEEE de doble precisión.

Proyecto 1.3 (solución en página 32)

Al ejecutar en MATLAB® las instrucciones

```
Command Window
>> format hex
>> 0.1
```

se obtiene `3fb9999999999999a`. Explica este resultado.

2. Condicionamiento y estabilidad en análisis numérico

El objeto del análisis numérico es el diseño y estudio de métodos que permitan obtener, con la mayor precisión posible, soluciones numéricas de ciertos problemas. Estos métodos resuelven dichos problemas mediante algoritmos, esto es, por medio de una lista ordenada de operaciones aritméticas y lógicas que transforman ciertos «datos de entrada» en ciertos «datos de salida». Además de su correcto funcionamiento en aritmética exacta, la idoneidad de un algoritmo está también ligada a su coste temporal y a su comportamiento respecto a los errores de redondeo.

La implementación de un algoritmo mediante algún lenguaje de programación tiene en cuenta numerosos factores; algunos no propiamente matemáticos, como el uso de la memoria del ordenador. Estos factores de indudable sesgo informático no serán tratados en el presente curso.

2.1. Condicionamiento de los problemas numéricos

Los resultados o datos de salida de los problemas tratados en análisis numérico deben (o, al menos, deberían) estar perfectamente determinados por los datos de entrada. En la inmensa mayoría de los casos puede asumirse incluso que los datos de salida dependen continuamente de los datos de entrada. En algunas ocasiones, el grado de continuidad entre esos datos de entrada y los correspondientes resultados es muy bajo, es decir, pequeños cambios en los datos de entrada generan grandes variaciones en los datos de salida. Aquellos problemas que poseen esta propiedad indeseable se denominan **problemas mal condicionados** (o problemas con una mala condición). Si, por el contrario, el grado de continuidad entre los

²⁰En el formato hexadecimal, los números de 10 a 15 se representan, respectivamente, por las letras *a*, *b*, *c*, *d*, *e*, *f*.

datos de entrada y los de salida es alto, esto es, si pequeños cambios en los datos de entrada conducen a pequeñas variaciones en los resultados, se habla de **problemas bien condicionados** (o problemas con una buena condición).

La gravedad de un problema mal condicionado reside, por tanto, en que su resolución puede producir soluciones muy dispares si los datos de entrada varían levemente. Por otro lado, esta variabilidad es un hecho común e inevitable en todo tipo de aplicaciones a problemas reales.

Es importante subrayar que la condición es una propiedad inherente al problema y no depende del algoritmo utilizado en su resolución. No obstante, cuando un problema está mal condicionado, su mala condición se manifiesta con nitidez cuando se opera en aritmética de punto flotante, dado que en este caso se producen inevitablemente variaciones (en la forma de errores de redondeo) en los datos de entrada. Operando en aritmética exacta, la mala condición pasa inadvertida: al no existir errores en los datos de entrada, no se producen errores en los datos de salida.

Para medir el grado de continuidad mencionado anteriormente y dilucidar hasta qué punto un problema está bien o mal condicionado, suelen introducirse los denominados **números de condición**.

Ejemplo 2.1 ► Número de condición

Supongamos que queremos evaluar cierta función $f : [a, b] \subset \mathbb{R} \rightarrow \mathbb{R}$ en un punto $x \in (a, b)$. Asumimos que f es suficientemente derivable en el intervalo (a, b) .

Al definir un número de condición es fundamental saber qué datos de entrada van a sufrir la perturbación. En este caso, supondremos f «fija» y consideraremos un número de condición respecto a perturbaciones en x^a . Sean $y := f(x)$, $\tilde{y} := f(\tilde{x})$, donde \tilde{x} es la perturbación en x , esto es, $\tilde{x} = x + \delta_x$. Aplicando la fórmula de Taylor, se tiene que

$$\tilde{y} = f(\tilde{x}) = f(x + \delta_x) \approx f(x) + f'(x)\delta_x,$$

de donde $\tilde{y} - y \approx f'(x)\delta_x = f'(x)(\tilde{x} - x)$. Dividiendo por $y = f(x) \neq 0$,

$$\frac{\tilde{y} - y}{y} \approx \frac{1}{f(x)} f'(x)(\tilde{x} - x),$$

y multiplicando y dividiendo por $x \neq 0$ en el segundo miembro,

$$\frac{\tilde{y} - y}{y} \approx \frac{x}{f(x)} f'(x) \frac{\tilde{x} - x}{x},$$

se obtiene, finalmente,

$$\text{Rel}(\tilde{y}) \approx \frac{|x|}{|f(x)|} |f'(x)| \times \text{Rel}(\tilde{x}).$$

Por tanto, es razonable considerar la cantidad (cuando esté definida)

$$\text{cond}(f, x) := \frac{|x|}{|f(x)|} |f'(x)|$$

como un número de condición respecto a x para el problema anterior, dado que representa un factor de amplificación del error relativo cometido en el dato de entrada.

^aEn el apéndice C se describe la situación general en la que, junto al error en el dato de entrada, se producen errores de redondeo debidos al algoritmo utilizado para evaluar la función f .

Ejercicio 2.1

Considera las siguientes expresiones:

$$0.99 - 0.7\sqrt{2}, \quad \frac{10^{-4}}{0.99 + 0.7\sqrt{2}}.$$

1. Comprueba que, en aritmética exacta, son equivalentes.
2. Cálculalas en MATLAB®. ¿Qué valor de los proporcionados por el programa piensas que es más fiable? Justifica tu respuesta.

Proyecto 2.1 (solución en página 33)

Dado $\alpha \in (0, 1)$, considera el sistema lineal de ecuaciones

$$\begin{cases} x + \alpha y = 1, \\ \alpha x + y = 0. \end{cases}$$

El cálculo del punto de corte de las rectas que definen el sistema es un problema que está mejor condicionado cuanto más cerca se encuentra α de cero y peor condicionado cuanto más cerca se encuentra α de uno.

Comprueba este fenómeno observando qué le ocurre al punto de corte cuando introducimos una perturbación en α . En concreto, toma $\alpha = 0.01$ y $\alpha = 0.99$ y en ambos casos introduce la perturbación $\tilde{\alpha} = \alpha + 0.002$. Calcula los puntos de corte para α y $\tilde{\alpha}$ y determina su distancia. ¿Qué observas?

2.2. Estabilidad de los métodos numéricos

Al ejecutar un programa que implemente cierto algoritmo y estar, por tanto, en un contexto de aritmética de punto flotante, podemos dar casi por seguro que en cada paso que efectúe dicho algoritmo se producirá un error de redondeo. En otras palabras, inevitablemente el error se propaga²¹. Sin embargo, en muchos casos la acumulación de todos esos errores tiene un efecto despreciable en el resultado final, desde el punto de vista de la precisión requerida. Aquellos métodos numéricos con esta propiedad deseable se dice que son **métodos numéricamente estables** y, en caso contrario, se dice que son **métodos numéricamente inestables**²².

Es importante mencionar que en numerosas ocasiones la inestabilidad numérica puede solventarse tras realizar un análisis en profundidad del método numérico elegido.

Cuando un problema está bien condicionado y aplicamos en su resolución un algoritmo numéricamente estable, el resultado obtenido suele tener un gran nivel de precisión. Por el contrario, si un problema está mal condicionado, todos los algoritmos empleados para estimar su solución, aún siendo numéricamente estables, generan resultados que tienen poca precisión.

²¹En el apéndice D se puede encontrar una descripción básica de la propagación del error.

²²Definiciones más rigurosas de estabilidad e inestabilidad numéricas se encuentran en el apéndice C.

Uno de los fenómenos más conocidos y relacionados con la inestabilidad es la llamada **cancelación numérica**²³. Este fenómeno se observa cuando, durante un cálculo en aritmética de punto flotante, se restan dos números parecidos. El error generado por la sustracción es más pronunciado cuanto más cercanos son los números. Además, cuando dicha diferencia se usa en operaciones posteriores, se genera un efecto de propagación del error que suele reducir fuertemente la exactitud del resultado final.

Para ser más precisos, consideremos la sustracción, en aritmética exacta, de dos cantidades,

$$x = a - b,$$

donde a y b tienen el mismo signo. En aritmética de punto flotante la operación se expresa, de manera simplificada, como

$$\tilde{x} = \text{fl}(a) - \text{fl}(b),$$

donde

$$\text{fl}(a) = a(1 + \delta_a), \quad \text{fl}(b) = b(1 + \delta_b), \quad (2)$$

siendo $|\delta_a|$ y $|\delta_b|$, respectivamente, los errores relativos de redondeo asociados a las cantidades a y b que queremos restar. El error relativo generado por el cálculo de la resta en aritmética de punto flotante verifica

$$\text{Rel}(\tilde{x}) = \left| \frac{\tilde{x} - x}{x} \right| = \left| \frac{a\delta_a - b\delta_b}{a - b} \right| \leq \max\{|\delta_a|, |\delta_b|\} \frac{|a| + |b|}{|a - b|}. \quad (3)$$

Por consiguiente, el error relativo en x depende (como es lógico) de los errores relativos en a y en b , pero puede ser amplificado por la cantidad

$$\frac{|a| + |b|}{|a - b|}. \quad (4)$$

De esta forma, si $|a - b| \ll |a| + |b|$, el cociente (4) puede ser muy grande y generarse, en consecuencia, inestabilidad numérica.

Proyecto 2.2 (solución en página 37)

En el texto anterior se menciona que, en aritmética de punto flotante, la operación $x = a - b$ se expresa, de manera simplificada, como

$$\tilde{x} = \text{fl}(a) - \text{fl}(b).$$

¿Puedes explicar en qué consiste esta simplificación?

Haz un análisis del error relativo que ocasiona el cálculo en aritmética de punto flotante de las demás operaciones algebraicas básicas (suma, multiplicación y división), de forma análoga al efectuado previamente para la sustracción, y comprueba que ninguna de ellas genera crecimiento del error, esto es, ninguna presenta inestabilidad numérica. Realiza este análisis de dos maneras: primero, asumiendo para cada operación una simplificación similar a la introducida para la resta y, después, sin suponer ninguna simplificación.

²³O, también, cancelación catastrófica.

Ejemplo 2.2 ► Cancelación numérica

Consideremos dos números que compartan sus diez primeros dígitos:

$$a = 1.333333338,$$

$$b = \frac{4}{3} = 1.\widehat{3} = 1.333333333 \dots$$

Sus respectivas representaciones en punto flotante de 16 dígitos vienen dadas por

$$\text{fl}(a) = a = \underbrace{1.333333333800000}_{16 \text{ dígitos significativos}},$$

$$\text{fl}(b) = \underbrace{1.333333333333333}_{16 \text{ dígitos significativos}}.$$

La resta de a y b vale

$$\begin{array}{r} a = 1.333333333800000 \dots \\ - b = 1.333333333333333 \dots \\ \hline 0.000000000466666 \dots = 4.\widehat{6} \times 10^{-10} \end{array}$$

Por su parte, la resta de $\text{fl}(a)$ y $\text{fl}(b)$ vale

$$\begin{array}{r} \text{fl}(a) = \overbrace{1.333333333}^{10 \text{ dígitos comunes}}800000|0 \\ - \text{fl}(b) = \overbrace{1.333333333333333} \\ \hline \underbrace{0.000000000}_{\text{dígitos no significativos}} \underbrace{466666|7}_{\text{dígitos significativos}} \end{array}$$

Observemos que la secuencia de infinitos dígitos 6 presentes en la resta exacta ha desaparecido y en su lugar figura una única cifra 7.

Ahora bien, la representación del resultado de la sustracción como número en punto flotante da lugar a que los diez ceros no significativos situados a la izquierda del primer dígito significativo se desvanezcan:

$$\underbrace{0.000000000}_{\text{desaparecen}} 4666667 = \underbrace{4.666667}_{7 \text{ dígitos significativos}} \times 10^{-10}.$$

En consecuencia, el resultado de la resta realizada en aritmética de punto flotante tiene únicamente 7 dígitos significativos, a diferencia de $\text{fl}(a)$ y $\text{fl}(b)$, que constan de 16. Han desaparecido, por tanto, 9 dígitos, lo que causa la pérdida de precisión que observamos en la aproximación de la resta:

$$a - b = 4.\widehat{6} \times 10^{-10},$$

$$\text{fl}(a) - \text{fl}(b) = 4.666667 \times 10^{-10}.$$

El error relativo generado por el cálculo de la resta en aritmética de punto flotante viene dado por

$$\begin{aligned} \text{Rel}(\tilde{x}) &= \left| \frac{\tilde{x} - x}{x} \right| = \frac{4.6666667 - 4.\widehat{6}}{4.\widehat{6}} \\ &\approx 7.1429 \times 10^{-8}, \end{aligned} \quad (5)$$

donde

$$\begin{aligned} x &= a - b, \\ \tilde{x} &= \text{fl}(a) - \text{fl}(b). \end{aligned}$$

Calculamos finalmente el valor del factor de amplificación del error relativo (4):

$$\begin{aligned} \frac{|a| + |b|}{|a - b|} &= \frac{2.66666666671\widehat{3}}{4.\widehat{6}} \times 10^{10} \\ &\approx 5.7143 \times 10^9. \end{aligned}$$

Un cálculo trivial permite comprobar que se verifica la desigualdad dada en la previsión teórica (3). Efectivamente, al ser $|\delta_a|$ y $|\delta_b|$ en (2) los errores de redondeo de a y b , respectivamente, ambas cantidades valen, como mucho, la unidad de redondeo, $u \approx 1.1102 \times 10^{-16}$. Así,

$$\begin{aligned} 7.1429 \times 10^{-8} \approx \text{Rel}(\tilde{x}) &\leq \max\{|\delta_a|, |\delta_b|\} \frac{|a| + |b|}{|a - b|} \\ &\leq u \frac{|a| + |b|}{|a - b|} \\ &\approx 1.1102 \times 10^{-16} \times 5.7143 \times 10^9 \\ &\approx 6.3441 \times 10^{-7}. \end{aligned}$$

Ejemplo 2.3 ► Cancelación numérica

Consideremos los números $a = 1$ y $b = 1 + 10^{-15}$. Su diferencia, en aritmética exacta, es $b - a = 10^{-15}$.

Sin embargo, si realizamos este cálculo en MATLAB® obtenemos:

Command Window

```
>> format long
>> a=1;
>> b=1+10^(-15);
>> b-a
```

ans =

```
1.110223024625157e-15
```

Comparemos este resultado con el valor exacto:

$$1 \text{ dígito significativo correcto} \rightarrow \boxed{1}. \underbrace{110223024625157}_{15 \text{ dígitos significativos no correctos}} \times 10^{-15},$$

$$16 \text{ dígitos significativos correctos} \rightarrow \boxed{1.0000000000000000} \times 10^{-15}.$$

Han desaparecido 15 de los 16 dígitos significativos correctos de los que consta el valor de la resta en aritmética exacta. Calculada en aritmética de punto flotante, la sustracción tiene un único dígito significativo correcto. El origen de este crecimiento dramático del error se encuentra en la proximidad de los números que se sustraen. Su resta genera un fortísimo efecto de cancelación numérica.

El error relativo cometido al realizar esta operación en aritmética de punto flotante ha sido:

```

Command Window
>> format short e
>> y=b-a; % Valor aproximado de la resta
>> x=10^(-15); % Valor exacto de la resta
>> error_relativo=abs(x-y)/abs(x)

error_relativo =

    1.1022e-01
  
```

El error relativo se ha amplificado por un factor de 10^{15} , lo que implica, como acabamos de ver, una pérdida de 15 dígitos significativos correctos.

El valor del factor de amplificación del error relativo (4) es, en este caso,

$$\frac{|a| + |b|}{|a - b|} = \frac{2 + 10^{-15}}{10^{-15}} = 2 \times 10^{15} + 1.$$

El resultado numérico obtenido para el error relativo se ajusta muy bien a la previsión teórica (3), puesto que

$$\begin{aligned} \underbrace{1.1022 \times 10^{-1}}_{\text{error relativo}} &\approx \left| \frac{x - y}{x} \right| \leq \overbrace{\max\{|\delta_a|, |\delta_b|\}}^{\leq u} \frac{|a| + |b|}{|a - b|} \\ &\leq u \times (2 \times 10^{15} + 1) \\ &\approx 1.1102 \times 10^{-16} \times (2 \times 10^{15} + 1) \\ &\approx \underbrace{2.2204 \times 10^{-1}}_{\text{previsión teórica}}. \end{aligned}$$

Problema 2.1

Considera las funciones

$$f(x) = \frac{1 - \cos(x)}{x^2}, \quad x \neq 0,$$

y

$$g(x) = \frac{e^x - 1}{x}, \quad x \neq 0,$$

1. Comprueba analíticamente que las funciones anteriores son continuas en $x = 0$, esto es, comprueba que

$$\lim_{x \rightarrow 0} f(x) = \frac{1}{2}$$

y

$$\lim_{x \rightarrow 0} g(x) = 1.$$

2. Eligiendo adecuadamente valores de x próximos a cero, por ejemplo

$$x = 10^{-1}, 10^{-2}, \dots, 10^{-15},$$

muestra en MATLAB® que tanto $f(x)$ como $g(x)$ presentan cancelación numérica.

Para cada una de las dos funciones, construye una tabla donde se muestre la evolución del error, de forma que en la primera columna aparezca x , en la segunda columna figure el valor de la función en x y en la tercera columna se exponga el error cometido.

3. Obtén dos expresiones equivalentes a $f(x)$ que eliminen el problema visto en el apartado anterior.^a
4. Haciendo el cambio de variable $u = e^x$ en la función $g(x)$, se obtiene

$$h(u) = \frac{u - 1}{\log(u)}, \quad u > 0.$$

Comprueba que, cuando u se aproxima a uno, el problema visto en el apartado segundo desaparece, a pesar de que el numerador de la nueva función h sigue presentando cancelación numérica. Analiza este fenómeno y trata de entender qué está ocurriendo.

^aPuedes deducir la primera de ellas utilizando la identidad trigonométrica

$$\operatorname{sen}^2\left(\frac{\alpha}{2}\right) = \frac{1 - \cos(\alpha)}{2}.$$

Puedes deducir la segunda multiplicando y dividiendo por

$$1 + \cos(x)$$

en la expresión de la función.

Proyecto 2.3 (solución en página 40)

Un ejemplo simple, pero importante, donde surge de modo natural la cancelación numérica es la resolución de ecuaciones de segundo grado.

En aritmética exacta, el problema de resolver $ax^2 + bx + c = 0$ ($a \neq 0$) es muy conocido y sencillo: hay dos raíces dadas por la fórmula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

1. Usando la fórmula anterior en MATLAB®, calcula las correspondientes raíces de

$$x^2 - 10^n x + 1 = 0,$$

para $n = 1, 2, \dots, 10$.

2. Compara los resultados obtenidos con los proporcionados por la orden de MATLAB® `roots`.
3. Describe dónde se están produciendo fenómenos de cancelación numérica. ¿Cómo podrían solventarse dichos problemas?

Proyecto 2.4 (solución en página 44)

Queremos calcular el número π usando el método de Arquímedes, que consiste en inscribir polígonos regulares P_n de $n \geq 3$ lados en una circunferencia de radio unidad y hacer tender n hacia infinito.

1. Demuestra que

$$f(n) = n \operatorname{sen} \left(\frac{\pi}{n} \right),$$

donde $f(n)$ denota la mitad del perímetro de P_n .

Verifica que

$$\lim_{n \rightarrow \infty} f(n) = \pi.$$

Para cada $k \geq 2$, considera $y_k := f(2^k)$. Demuestra que

$$y_{k+1} = 2^{k+1} \sqrt{\frac{1 - \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2}}{2}}.$$

2. Utilizando la recurrencia anterior, implementa una función en MATLAB® que evalúe el número π . Muestra una tabla de resultados con las 30 primeras iteraciones. ¿Qué problema surge?
3. Modifica la recurrencia anterior para obtener un método numérico más estable. Aproxima el número π utilizando esta nueva recurrencia.

Los fenómenos numéricos que hemos estado comentando son algo más que curiosidades matemáticas desde el punto de vista de las aplicaciones. En este sentido, vamos a finalizar la lección mencionando dos incidentes que ejemplifican hasta qué punto no se deben subestimar detalles menores en los algoritmos implicados en problemas reales²⁴.

Ejemplo 2.4 ► El fallo del misil *Patriot*

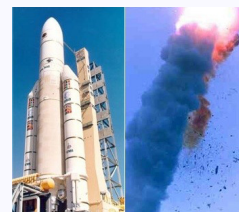
Durante la Primera Guerra del Golfo, en febrero de 1991, un misil *Patriot* estadounidense se desvió de su objetivo inicial y falló en detectar un misil *Scud* iraquí, causando este último veintiocho muertos entre las tropas aliadas. El error fue consecuencia, en última instancia, de una pobre gestión de los errores de redondeo generados en el programa principal.



En concreto, el reloj interno del sistema efectuaba una multiplicación por el factor 0.1 cada décima de segundo. Ahora bien, el número 0.1 es periódico en base 2, luego en cada una de estas multiplicaciones se generaba cierto error de redondeo. En la fecha fatídica, el *software* que controlaba el misil *Patriot* había estado ejecutándose ininterrumpidamente durante 100 horas. El efecto acumulado de los errores de redondeo durante un período de tiempo tan prolongado ocasionó un retraso de 34 centésimas de segundo en el despegue del misil, lo que impidió que alcanzara su objetivo.

Ejemplo 2.5 ► La explosión del cohete *Ariane 5*

El 4 de junio de 1996, el cohete *Ariane 5*, integrante de un programa de la Agencia Espacial Europea cuyo objetivo era poner en órbita varios satélites artificiales, explotó a los cuarenta segundos de su despegue desde la Guayana Francesa, ocasionando pérdidas multimillonarias (afortunadamente, la lanzadera no iba tripulada). La causa de la explosión fue un error de *software* en el sistema inercial del cohete.



Específicamente, cierto número escrito en formato IEEE de doble precisión y relacionado con la velocidad horizontal del cohete con respecto a la plataforma de despegue fue convertido a formato de número entero de 16 bits. El desdichado número resultó ser mayor que la máxima cantidad que se puede obtener utilizando 16 bits, lo que originó una aparición inesperada de *overflow*. A partir de ahí, una cadena de circunstancias condujo al fatal desenlace.

²⁴En cualquier caso, es importante subrayar que sucesos como los mencionados en los ejemplos anteriores son la excepción, no la norma. De hecho, durante el curso, solo ocasionalmente tendremos que fijarnos en fenómenos ligados a *roundoff* (errores de redondeo), *overflow* o *underflow*.

3. Apéndice A: Estructura del sistema numérico de punto flotante

Vamos a describir la estructura del sistema numérico de punto flotante $\mathbb{M}(2, t, e_{\min}, e_{\max})$ (\mathbb{M} en lo sucesivo), examinando únicamente los números positivos (la disposición de los negativos es, evidentemente, simétrica).

Denotamos por $\mathbb{M}^+(2, t, e_{\min}, e_{\max})$ (\mathbb{M}^+ en lo sucesivo) el conjunto de los números positivos que pertenecen a \mathbb{M} .

El símbolo $[a, b)_{\mathbb{M}}$ representa los números del intervalo $[a, b)$ que están en \mathbb{M} , esto es,

$$[a, b)_{\mathbb{M}} = [a, b) \cap \mathbb{M}.$$

El conjunto $[1, 2)_{\mathbb{M}}$ está compuesto por los números²⁵:

$$\begin{aligned} (1.00 \overset{t}{\dots} 00)_2 &= 1, \\ (1.00 \overset{t}{\dots} 01)_2 &= 1 + \varepsilon_{\mathbb{M}}, \\ (1.00 \overset{t}{\dots} 10)_2 &= 1 + 2\varepsilon_{\mathbb{M}}, \\ &\vdots \\ (1.11 \overset{t}{\dots} 11)_2 &= 1 + (2^{t-1} - 1)\varepsilon_{\mathbb{M}} = 1 + (1 - \varepsilon_{\mathbb{M}}) = 2 - \varepsilon_{\mathbb{M}}. \end{aligned}$$

La distancia entre elementos consecutivos de $[1, 2)_{\mathbb{M}}$ es constante y coincide con la precisión de la máquina $\varepsilon_{\mathbb{M}}$.

El conjunto $[1, 2)_{\mathbb{M}}$ es la base sobre la que se edifica la estructura completa de \mathbb{M}^+ . En efecto,

$$\begin{aligned} \mathbb{M}^+ &= \dots \cup \left[\frac{1}{4}, \frac{1}{2}\right)_{\mathbb{M}} \cup \left[\frac{1}{2}, 1\right)_{\mathbb{M}} \cup [1, 2)_{\mathbb{M}} \cup [2, 4)_{\mathbb{M}} \cup \dots \\ &= \dots \cup \frac{1}{4}[1, 2)_{\mathbb{M}} \cup \frac{1}{2}[1, 2)_{\mathbb{M}} \cup [1, 2)_{\mathbb{M}} \cup 2[1, 2)_{\mathbb{M}} \cup \dots \end{aligned}$$

Para ser más precisos,

$$\mathbb{M}^+ = \bigcup_{n=1}^{-e_{\min}} I_n \cup \bigcup_{n=0}^{e_{\max}} J_n$$

donde

$$\begin{aligned} I_n &= \left[\frac{1}{2^n}, \frac{1}{2^{n-1}}\right)_{\mathbb{M}} = \frac{1}{2^n}[1, 2)_{\mathbb{M}}, \quad n = 1, 2, \dots, -e_{\min}, \\ J_n &= [2^n, 2^{n+1})_{\mathbb{M}} = 2^n[1, 2)_{\mathbb{M}}, \quad n = 0, 1, \dots, e_{\max}. \end{aligned}$$

Cada uno de estos conjuntos consta de la misma cantidad de números²⁶. Además, la distancia entre números consecutivos dentro de cada conjunto es constante²⁷.

En otras palabras, los intervalos de la recta real

$$\dots \left[\frac{1}{4}, \frac{1}{2}\right), \left[\frac{1}{2}, 1\right), [1, 2), [2, 4), [4, 8) \dots$$

van duplicando sucesivamente su longitud, pero cada uno de ellos contiene la misma cantidad de números de \mathbb{M} . Además, en cada intervalo, la distancia entre números de \mathbb{M} consecutivos

²⁵Se trata de un total de $\frac{1}{\varepsilon_{\mathbb{M}}} = 2^{t-1}$ números.

²⁶Cada uno de ellos tiene exactamente $\frac{1}{\varepsilon_{\mathbb{M}}} = 2^{t-1}$ elementos.

²⁷El valor de esa distancia es $2^n \varepsilon_{\mathbb{M}}$ para el conjunto I_n y $\frac{\varepsilon_{\mathbb{M}}}{2^n}$ para el conjunto J_n .

es constante. Por tanto, los números de \mathbb{M} situados en un intervalo distan entre sí el doble de lo que distan los números de \mathbb{M} situados en el intervalo anterior.

En consecuencia, los números de \mathbb{M} se van alejando unos de otros a medida que nos distanciamos de cero y se van acercando unos a otros a medida que nos aproximamos a cero.

Ejemplo 3.1 ► Sistema numérico de punto flotante

Consideremos el sistema numérico de punto flotante

$$\mathbb{M}(2, t, e_{\min}, e_{\max}) = \mathbb{M}(2, 3, -2, 2).$$

El conjunto de sus números positivos viene dado por

$$\mathbb{M}^+(2, 3, -2, 2) = \left[\frac{1}{4}, \frac{1}{2}\right)_{\mathbb{M}} \cup \left[\frac{1}{2}, 1\right)_{\mathbb{M}} \cup [1, 2)_{\mathbb{M}} \cup [2, 4)_{\mathbb{M}} \cup [4, 8)_{\mathbb{M}}.$$

Cada uno de los conjuntos que integran la unión anterior está compuesto de

$$\frac{1}{\varepsilon_{\mathbb{M}}} = 2^{t-1} = 2^2 = 4$$

números.

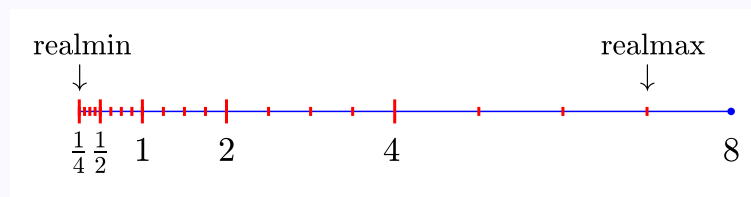
Observa que

$$\varepsilon_{\mathbb{M}} = 2^{1-3} = \frac{1}{4},$$

$$\text{realmin} = 2^{-2} = \frac{1}{4},$$

$$\text{realmax} = 1.11 \times 2^2 = 7.$$

La siguiente imagen muestra la distribución de los números de $\mathbb{M}^+(2, 3, -2, 2)$.



4. Apéndice B: Dígitos significativos

Reglas para los dígitos significativos

1. Dígitos no nulos. Todos los dígitos distintos de cero **son significativos**.

Ejemplo 4.1 ► Dígitos no nulos

46.7 tiene tres cifras significativas.

7253 tiene cuatro dígitos significativos.

2. Dígitos nulos:

- a) Los ceros situados entre dos dígitos no nulos **son significativos**.

Ejemplo 4.2 ► Dígitos nulos entre dígitos no nulos

40.09003 tiene siete cifras significativas.

- b) Los ceros ubicados a la izquierda del primer dígito no nulo (es decir, los ceros iniciales) **no son significativos**.

Ejemplo 4.3 ► Ceros iniciales

0.67 tiene dos cifras significativas.

0.00000067 también tiene dos cifras significativas.

- c) Los ceros localizados a la derecha del último dígito no nulo (es decir, los ceros finales):

- 1) **Son significativos** si en el número figura el punto decimal.

Ejemplo 4.4 ► Ceros finales con punto decimal

830. tiene tres cifras significativas.

570.0 tiene cuatro dígitos significativos.

46.00 tiene cuatro cifras significativas.

0.0084000 tiene cinco dígitos significativos.

- 2) **No son significativos** si en el número no aparece el punto decimal.

Ejemplo 4.5 ► Ceros finales sin punto decimal

830 tiene dos cifras significativas.

En resumen:

Son dígitos o cifras significativos:

1. Todos los dígitos distintos de cero.
2. Los ceros situados entre dos dígitos no nulos.
3. Los ceros finales de un número en cuya expresión figura el punto decimal.

No son dígitos o cifras significativos:

1. Los ceros iniciales.
2. Los ceros finales de un número en cuya expresión no aparece el punto decimal.

5. Apéndice C: Dígitos significativos correctos

Un dígito significativo de cierto número es (o no es) correcto dependiendo de un segundo número del cual el primero es una aproximación²⁸.

Cuando un número aproxima a otro, la intuición sugiere que los dígitos compartidos son significativos correctos. No obstante, el adjetivo correcto denota aquí una categoría «escurridiza», pues una cifra puede no ser compartida pero ser correcta. Por ejemplo, 0.9999 tiene, como aproximación al número 1, un mínimo de tres dígitos significativos correctos (como enseguida veremos), a pesar de que ambos números no comparten ninguna cifra.

La definición formal de dígito significativo correcto es inevitablemente «algo enrevesada». En base 10, se dice que x_a comparte $m \in \mathbb{N}$ dígitos significativos correctos con x_t si se verifica que $x_t - x_a$ tiene magnitud menor o igual que cinco en la posición digital $(m + 1)$ -ésima de x_t (y cero en las anteriores), contando a la derecha del primer dígito no nulo de x_t . El significado intuitivo de este concepto es que los m primeros dígitos de x_a , a partir del primero no nulo, son considerados como dígitos «verdaderos» de la representación decimal de x_t que, como es lógico, se supone desconocida.

Ejemplo 5.1 ► Número de dígitos significativos correctos

Sean $x_a = 0.02138$ y $x_t = 0.02143$. Se verifica que

$$x_t - x_a = 0.00005.$$

Contando hacia la derecha a partir de la posición donde aparece la primera cifra no nula de x_t (señalada por la flecha), el primer dígito no nulo se encuentra en la posición $m + 1 = 4$. Puesto que este dígito es 5, se tiene que x_a comparte $m = 3$ dígitos significativos correctos con x_t , esto es, las cifras 2, 1 y 3.

Se enuncian a continuación dos resultados sobre dígitos significativos correctos.

Proposición 1. Si x_a tiene m dígitos significativos correctos de x_e , entonces

$$\text{Rel}(x_a) \leq 5 \times 10^{-m}.$$

Ejemplo 5.2 ► Acotación del error relativo

$x_a = 3.141582$ tiene 5 dígitos significativos correctos de $\pi = 3.141592\dots$, luego debe ser cierto que

$$\text{Rel}(x_a) \leq 5 \times 10^{-5}.$$

En efecto, $\text{Rel}(x_a) \approx 3.3911 \times 10^{-6}$.

²⁸Por el contrario, un dígito es (o no es) significativo dependiendo únicamente del número al que pertenece.

Proposición 2. Si el error relativo cometido al aproximar x_e por x_a verifica

$$\text{Rel}(x_a) \leq 5 \times 10^{-m}$$

entonces x_a tiene, al menos, $m - 1$ dígitos significativos correctos de x_e .

Ejemplo 5.3 ► Número mínimo de dígitos significativos correctos

x_a	x_e	$\text{Rel}(x_a)$	$m - 1$
6.3	6	5×10^{-2}	1
3.9999	4	2.5×10^{-5}	4
0.9999	1	1×10^{-4}	3
0.000199	0.0002	5×10^{-3}	2
8.0000649	8.000065	1.25×10^{-8}	7
8.0000649	8.000064	1.125×10^{-7}	6

6. Apéndice D: Propagación del error

Llamamos «algoritmo» a una sucesión finita de operaciones elementales que prescriben cómo calcular la solución de un problema a partir de unos datos de entrada.

Supongamos que el problema consiste en evaluar cierta función²⁹

$$\varphi : [a, b] \subset \mathbb{R} \rightarrow \mathbb{R}$$

en un punto $x \in (a, b)$. Asumimos que φ es suficientemente derivable en el intervalo (a, b) .

Todo algoritmo que calcule $y = \varphi(x)$ puede descomponerse en una sucesión de operaciones elementales. En particular, la función φ se puede expresar como la composición de un conjunto de funciones «elementales» $\varphi^{(i)}$, $i = 0, 1, \dots, n$,

$$\varphi = \varphi^{(n)} \circ \varphi^{(n-1)} \circ \dots \circ \varphi^{(0)}.$$

Nuestro objetivo es describir, en líneas generales y sin entrar en consideraciones técnicas, cómo se propagan:

- i) el error de redondeo del dato de entrada,
- ii) los errores de redondeo asociados al algoritmo elegido.

En otras palabras, nuestra finalidad es determinar de qué manera afectan al resultado final $y = \varphi(x)$ los errores de redondeo acumulados durante el curso del algoritmo.

El algoritmo conduce desde el dato de entrada $x^{(0)} := x$, vía una cadena de resultados intermedios

$$x = x^{(0)} \rightarrow \varphi^{(0)}(x^{(0)}) = x^{(1)} \rightarrow \dots \rightarrow \varphi^{(n)}(x^{(n)}) = x^{(n+1)} = y$$

al resultado final y .

Los errores de redondeo que surgen en el contexto de la aritmética de punto flotante perturban los resultados intermedios (exactos) $x^{(i)}$, de modo que el algoritmo opera con los valores aproximados $\tilde{x}^{(i)}$, donde

$$i) \tilde{x}^{(i+1)} = \text{fl}(\varphi^{(i)}(\tilde{x}^{(i)})), \quad i = 0, 1, \dots, n,$$

$$ii) \tilde{x}^{(0)} = \text{fl}(x^{(0)}),$$

esto es,

$$\tilde{x}^{(0)} = \text{fl}(x^{(0)}) \rightarrow \text{fl}(\varphi^{(0)}(\tilde{x}^{(0)})) = \tilde{x}^{(1)} \rightarrow \dots \rightarrow \text{fl}(\varphi^{(n)}(\tilde{x}^{(n)})) = \tilde{x}^{(n+1)} := \tilde{y}.$$

²⁹Todo lo que figura en este apéndice se puede generalizar a una función vectorial de varias variables.

Si introducimos la notación

$$\Delta x^{(i)} = \tilde{x}^{(i)} - x^{(i)}, \quad i = 0, 1, \dots, n + 1,$$

no es difícil probar la siguiente fórmula, que describe, a primer orden, el efecto de los errores de redondeo en el resultado final $y = x^{(n+1)} = \varphi(x)$.

Fórmula de propagación del error³⁰:

$$\Delta y \approx \varphi'(x)\Delta x + C_1\delta_1x^{(1)} + \dots + C_n\delta_nx^{(n)} + \delta_{n+1}y$$

donde

- i) $|\Delta y| = |\Delta x^{(n+1)}| = |\tilde{x}^{(n+1)} - x^{(n+1)}| = |\tilde{y} - y|$ es el error absoluto del resultado final y ,
- ii) $|\Delta x| = |\Delta x^{(0)}| = |\tilde{x}^{(0)} - x^{(0)}|$ es el error absoluto de redondeo del dato de entrada x ³¹,
- iii) C_i , $i = 1, \dots, n$, es una constante que depende de los valores de las derivadas de las funciones elementales,
- iv) $|\delta_i|$, $i = 1, \dots, n + 1$, es el error de redondeo de $\varphi^{(i)}(\tilde{x}^{(i)})$, es decir,

$$\text{fl}(\varphi^{(i)}(\tilde{x}^{(i)})) = (1 + \delta_i)\varphi^{(i)}(\tilde{x}^{(i)}),$$

de donde se sigue que $|\delta_i| \leq u$, siendo u la unidad de redondeo.

Visto lo anterior, estamos en condiciones de exponer qué elementos clave ejercen una influencia crítica en la propagación del error:

1. El efecto que el error de redondeo del dato de entrada x ejerce en el resultado final y depende del tamaño de $\varphi'(x)$ en la fórmula de propagación del error,
2. El efecto que los errores de redondeo δ_i , $i = 1, \dots, n$, de los resultados intermedios ejercen en el resultado final y depende de los tamaños de las constantes C_i .

Los sumandos participantes en la fórmula de propagación del error se pueden clasificar en dos categorías:

1. El primer término, $\varphi'(x)\Delta x$, proviene del redondeo efectuado sobre el dato de entrada x , luego está presente independientemente del algoritmo utilizado para calcular $y = \varphi(x)$. Más aún, está presente incluso en ausencia de cualquier tipo de algoritmo. Notemos que el error absoluto de redondeo del dato de entrada verifica

$$|\Delta x| \leq |x|u,$$

³⁰Observemos que en ausencia de errores de redondeo de las funciones elementales $\varphi^{(i)}$, $i = 0, 1, \dots, n$, se tendría $\Delta y \approx \varphi'(x)\Delta x$, es decir, nos encontraríamos en la situación del ejemplo 2.1.

³¹En la fórmula de propagación del error, $|\Delta x|$ puede ser, de hecho, cualquier error o perturbación del dato de entrada. No obstante, al definir más adelante el error inherente de y , se requiere que $|\Delta x|$ sea el error absoluto de redondeo.

de donde se sigue que

$$|\varphi'(x)\Delta x| \leq |\varphi'(x)||x|u.$$

Por consiguiente, hay que asumir un error de magnitud

$$|\varphi'(x)||x|u$$

independientemente del algoritmo involucrado.

2. El resto de términos de la fórmula de propagación del error surge del algoritmo empleado para calcular $y = \varphi(x)$, si bien el último de ellos, $\delta_{n+1}y$, admite la acotación

$$|\delta_{n+1}y| \leq |y|u,$$

con independencia del algoritmo que se haya usado. En este sentido, podemos decir que, al igual que ocurre con $\varphi'(x)\Delta x$, se trata de un término de error independiente del algoritmo. En consecuencia, hay que contar con un error de magnitud

$$|y|u$$

sea cual sea el algoritmo utilizado.

$$\begin{array}{c} \text{Error de salida} \longrightarrow \Delta y \approx \underbrace{\varphi'(x)\Delta x + \delta_{n+1}y}_{\text{Términos de error independientes del algoritmo}} + \underbrace{C_1\delta_1x^{(1)} + \dots + C_n\delta_nx^{(n)}}_{\text{Términos de error dependientes del algoritmo}} \\ \begin{array}{c} \text{Error de entrada} \\ \downarrow \\ \varphi'(x) \end{array} \end{array}$$

En conjunto, para todo algoritmo se debe esperar un error de magnitud

$$\Delta^{(0)}y := (|\varphi'(x)||x| + |y|)u.$$

Llamamos «error inherente» de y a $\Delta^{(0)}y$.

Con las ideas expuestas hasta este momento, es posible dar una definición rigurosa de algoritmo numéricamente estable (y, por tanto, también de algoritmo numéricamente inestable):

Si la contribución al error total Δy de cada error de redondeo intermedio de un algoritmo, δ_i , $i = 1, \dots, n$, es, como mucho, del mismo orden de magnitud que el error inherente $\Delta^{(0)}y$, esto es,

$$|C_i\delta_ix^{(i)}| \lesssim \Delta^{(0)}y, \quad i = 1, \dots, n,$$

se dice que el algoritmo es «numéricamente estable». En caso contrario, se dice que el algoritmo es «numéricamente inestable».

$$\boxed{\varphi'(x)\Delta x + \delta_{n+1}y} + \boxed{C_1\delta_1x^{(1)} + \dots + C_n\delta_nx^{(n)}} \longleftarrow \text{algoritmo numéricamente estable}$$

$$\boxed{\varphi'(x)\Delta x + \delta_{n+1}y} + \boxed{C_1\delta_1x^{(1)} + \dots + C_n\delta_nx^{(n)}} \longleftarrow \text{algoritmo numéricamente inestable}$$

7. Soluciones de los proyectos

Proyecto 1.1

En todo sistema numérico de punto flotante $\mathbb{M}(\beta, t)$, donde $\beta \in \mathbb{Z}$, $\beta \geq 2$, la precisión de la máquina $\varepsilon_{\mathbb{M}} = \beta^{1-t}$ es exactamente la distancia del número uno al número siguiente³² a uno representable en $\mathbb{M}(\beta, t)$.

Vamos a probar esta propiedad en el caso³³ $\beta = 2$.

Por definición,

$$\varepsilon_{\mathbb{M}} = 2u,$$

donde $u = 2^{-t}$ es la unidad de redondeo.

El número siguiente a uno representable en $\mathbb{M}(2, t)$, denotado por 1^+ , se obtiene sustituyendo el último 0 de la fracción del número uno por 1, esto es,

$$1 = (1.0 \overset{t}{\dots} 00)_2 \times 2^0 \quad \longrightarrow \quad 1^+ = (1.0 \overset{t}{\dots} 01)_2 \times 2^0.$$

Observa que, efectivamente, los números 1 y 1^+ son consecutivos en $\mathbb{M}(2, t)$, puesto que la representación de un número comprendido entre ambos exigiría más de t bits en el coeficiente.

Por consiguiente,

$$\begin{aligned} 1^+ &= (1.0 \overset{t}{\dots} 01)_2 \times 2^0 \\ &= (1.0 \overset{t}{\dots} 00)_2 \times 2^0 + (0.00 \overset{t}{\dots} 01)_2 \times 2^0 \\ &= 1 + 2^{-(t-1)} \\ &= 1 + 2 \cdot 2^{-t} \\ &= 1 + 2u \\ &= 1 + \varepsilon_{\mathbb{M}}. \end{aligned}$$

Esto es, $1^+ = 1 + \varepsilon_{\mathbb{M}}$. Por lo tanto, $\varepsilon_{\mathbb{M}}$ es exactamente la distancia del número uno al número siguiente a uno representable en $\mathbb{M}(2, t)$.

La propiedad demostrada establece una conexión directa entre la precisión de la máquina y el número uno. No parece difícil concluir, por tanto, que esta constante está ligada exclusivamente a (y depende únicamente de) la unidad. Pero no es así.

La referencia al número uno en relación a la precisión de la máquina es irrelevante. Es decir, el número uno no ocupa ningún lugar «privilegiado». La precisión de la máquina no depende ni del número uno ni de ningún otro número. Se trata, en realidad, de una constante «universal» en $\mathbb{M}(2, t)$. Por consiguiente, la propiedad antes demostrada se puede enunciar prescindiendo del número uno y empleando en su lugar cualquier otro número de $\mathbb{M}(2, t)$, si bien en ese caso habría que expresarla en la forma:

³²Siguiente en el sentido de consecutivo mayor.

³³La demostración es la misma en cualquier base.

La precisión de la máquina $\varepsilon_{\mathbb{M}}$ es la cantidad que hay que sumar al coeficiente de cualquier número de $\mathbb{M}(2, t)$ para obtener el siguiente número representable en $\mathbb{M}(2, t)$.

No obstante, es más simple de enunciar utilizando el número uno, debido a que este número tiene la peculiaridad de ser igual a su coeficiente (puesto que su exponente es cero), característica que comparte con todos los números del intervalo $[1, 2)$ que pertenecen a $\mathbb{M}(2, t)$.

Proyecto 1.2

1. El primer código calcula la unidad de redondeo. Efectivamente, la salida del bucle `while` se producirá cuando x tome un valor tal que $1 + x$ se redondee a 1. Esto ocurrirá cuando x tome precisamente el valor de la unidad de redondeo. De hecho, los dos últimos valores de x son la precisión de la máquina $\varepsilon_{\mathbb{M}}$ y la unidad de redondeo $u = \frac{\varepsilon_{\mathbb{M}}}{2}$.
2. El segundo código muestra «overflow». Los dos últimos valores de x son $8.9885 \times 10^{307} > \text{realmax}/2$ e `Inf`. La salida del bucle `while` se producirá cuando x tome un valor tal que $x + x = x$, lo que solamente es posible (siendo $x \neq 0$ y tomando valores positivos y crecientes) cuando x es igual a `Inf`.
3. El tercer código muestra «underflow». Los dos últimos valores de x son $\text{eps} * \text{realmin} = 4.9407 \times 10^{-324}$ y 0. La salida del bucle `while` se producirá cuando x tome un valor tal que $x + x = x$, lo que solamente es posible (siendo x positivo y decreciente) cuando $x = 0$.

Proyecto 1.3

En primer lugar representamos 0.1 en formato IEEE de doble precisión. En base 2, el número 0.1 es periódico y se expresa en la forma

$$(0.\widehat{00011})_2 = (0.000110011\dots)_2.$$

Desplazando el punto decimal de forma que a su izquierda quede un único bit no nulo se obtiene

$$(0.\widehat{00011})_2 = (1.\widehat{10011})_2 \times 2^{-4}.$$

El exponente a almacenar, una vez introducido el sesgo, es

$$-4 + 1023 = 1019 = (0111111011)_2.$$

Efectuamos el redondeo del coeficiente $(1.\widehat{10011})_2$ a 53 bits:

$$\left(\overbrace{(1.1001\ 1001\dots1001\ 1001)}^{53} \mid 1001\dots \right)_2 \xrightarrow{1001+0001=1010} (1.1001\ 1001 \overbrace{\dots}^{53} 1001\ 1010)_2.$$

En consecuencia, la representación del número 0.1 en formato IEEE de doble precisión es

$$0 \mid 0111111011 \mid 1001\ 1001 \overbrace{\dots}^{52} 1001\ 1010.$$

A continuación, se agrupan los bits en bloques de cuatro, esto es, en bloques hexadecimales, obteniendo

$$\underbrace{0011}_3 \mid \underbrace{1111}_{15 \rightarrow f} \mid \underbrace{1011}_{11 \rightarrow b} \mid \overbrace{\underbrace{1001}_9 \mid \underbrace{1001}_9 \mid \dots \mid \underbrace{1001}_9}_{12} \mid \underbrace{1010}_{10 \rightarrow a},$$

que, en formato hexadecimal, se expresa como

$$3fb9999999999999a.$$

Proyecto 2.1

Dado el sistema lineal

$$\begin{cases} x + \alpha y = 1, \\ \alpha x + y = 0, \end{cases}$$

es muy fácil comprobar que el punto de corte de las rectas que definen el sistema es

$$\left(\frac{1}{1 - \alpha^2}, \frac{-\alpha}{1 - \alpha^2} \right).$$

El siguiente código de MATLAB® dibuja las rectas correspondientes a

$$\alpha = 0.01 \quad \text{y} \quad \tilde{\alpha} = \alpha + 0.002 = 0.012,$$

calculando, además, la distancia entre los puntos de corte para ambos valores.

Editor ► Problema bien condicionado

proyecto21a.m

```
alpha=0.01;
perturbacion=0.002;

% Punto de corte del sistema (alpha)

pcx=1./(1-alpha^2);
pcy=-alpha/(1-alpha^2);

% Dibujo del par de rectas (alpha)

t=0:0.01:2;
r1=(1-t)/alpha;
r2=-alpha*t;
plot(t,r1,'b',t,r2,'b',pcx,pcy,'o','MarkerEdgeColor','k',...
      'MarkerFaceColor','b'),shg
axis([0 2 -20 20])

pause

% Punto de corte del sistema (nalpna=alpha+perturbacion)

nalpna=alpha+perturbacion;
npcx=1./(1-nalpna^2);
npcy=-nalpna/(1-nalpna^2);

% Dibujo conjunto de los dos pares de rectas (alpha,nalpna)

t=0:0.01:2;
nr1=(1-t)/alpha;
nr2=-nalpna*t;
```

```

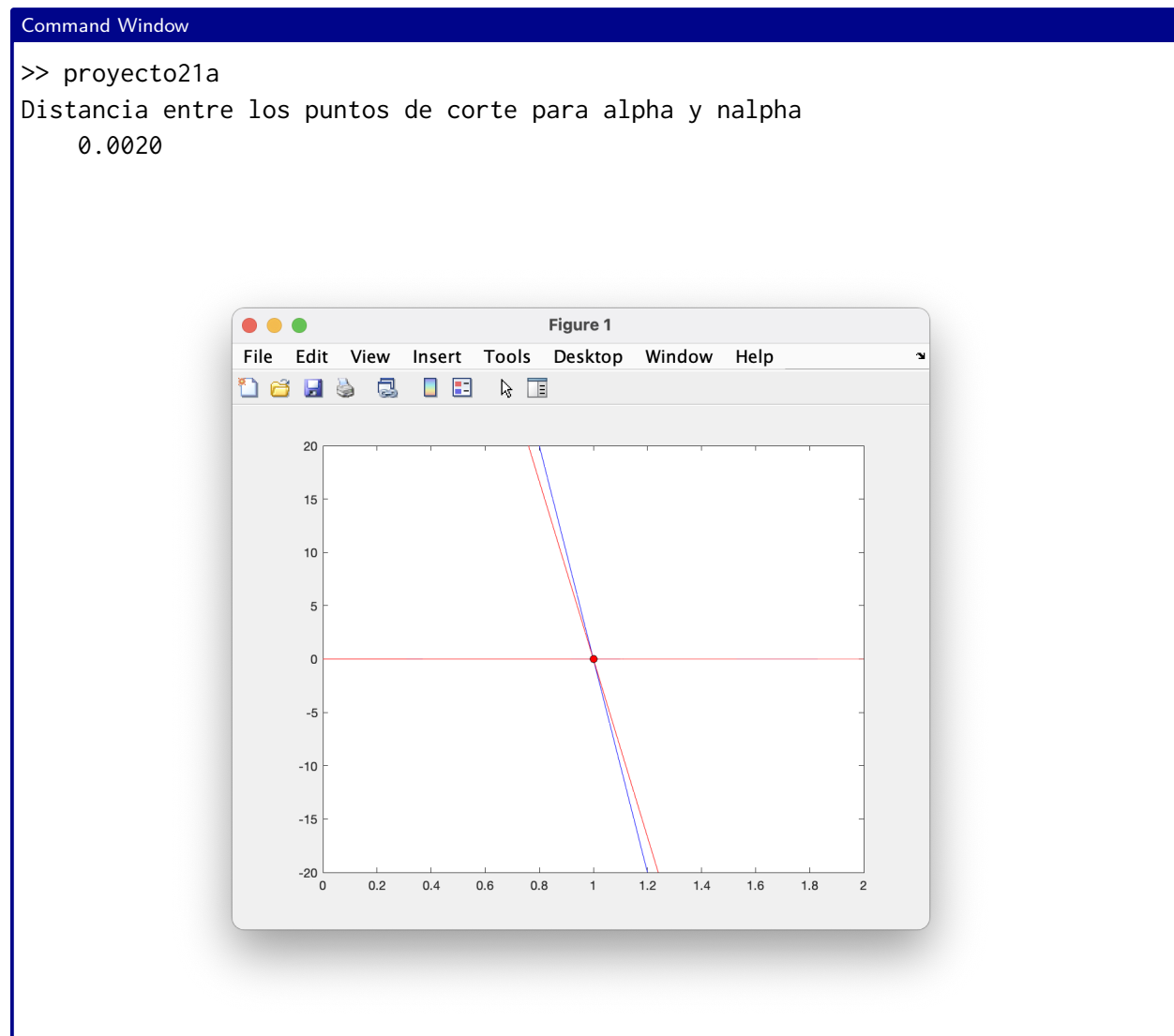
hold on
plot(t,nr1,'r',t,nr2,'r',npcx,npcy,'o','MarkerEdgeColor','k',...
     'MarkerFaceColor','r'),shg
hold off

% Distancia entre los puntos de corte para alpha y nalpha

disp('Distancia entre los puntos de corte para alpha y nalpha')
d=norm([pcx pcy]-[npcx npcy]);
format short e
disp(d)

```

Al ejecutarlo en MATLAB® se obtiene:



Observa que la distancia entre los puntos de corte (para $\alpha = 0.01$ y $\tilde{\alpha} = \alpha + 0.002 = 0.012$) es del orden de la perturbación introducida (coinciden en formato corto). De hecho, ambos puntos son, en la imagen, visualmente indistinguibles.

El código de MATLAB® que aparece a continuación es muy similar al anterior, pero en

esta ocasión dibuja las rectas correspondientes a

$$\alpha = 0.99 \quad \text{y} \quad \tilde{\alpha} = \alpha + 0.002 = 0.992,$$

calculando, además, la distancia entre los puntos de corte para ambos valores.

Editor ► Problema mal condicionado

proyecto21b.m

```
alpha=0.99;
perturbacion=0.002;

% Punto de corte del sistema (alpha)

pcx=1./(1-alpha^2);
pcy=-alpha/(1-alpha^2);

% Dibujo del par de rectas (alpha)

t=40:0.01:70;
r1=(1-t)/alpha;
r2=-alpha*t;
plot(t,r1,'b',t,r2,'b',pcx,pcy,'o','MarkerEdgeColor','k',...
      'MarkerFaceColor','b'),shg
axis([40 70 -70 -40])

pause

% Punto de corte del sistema (nalpna=alpha+perturbacion)

nalpna=alpha+perturbacion;
npcx=1./(1-nalpna^2);
npcy=-nalpna/(1-nalpna^2);

% Dibujo conjunto de los dos pares de rectas (alpha,nalpna)

t=40:0.01:70;
nr1=(1-t)/nalpna;
nr2=-nalpna*t;
hold on
plot(t,nr1,'r',t,nr2,'r',npcx,npcy,'o','MarkerEdgeColor','k',...
      'MarkerFaceColor','r'),shg
hold off

% Distancia entre los puntos de corte para alpha y nalpna

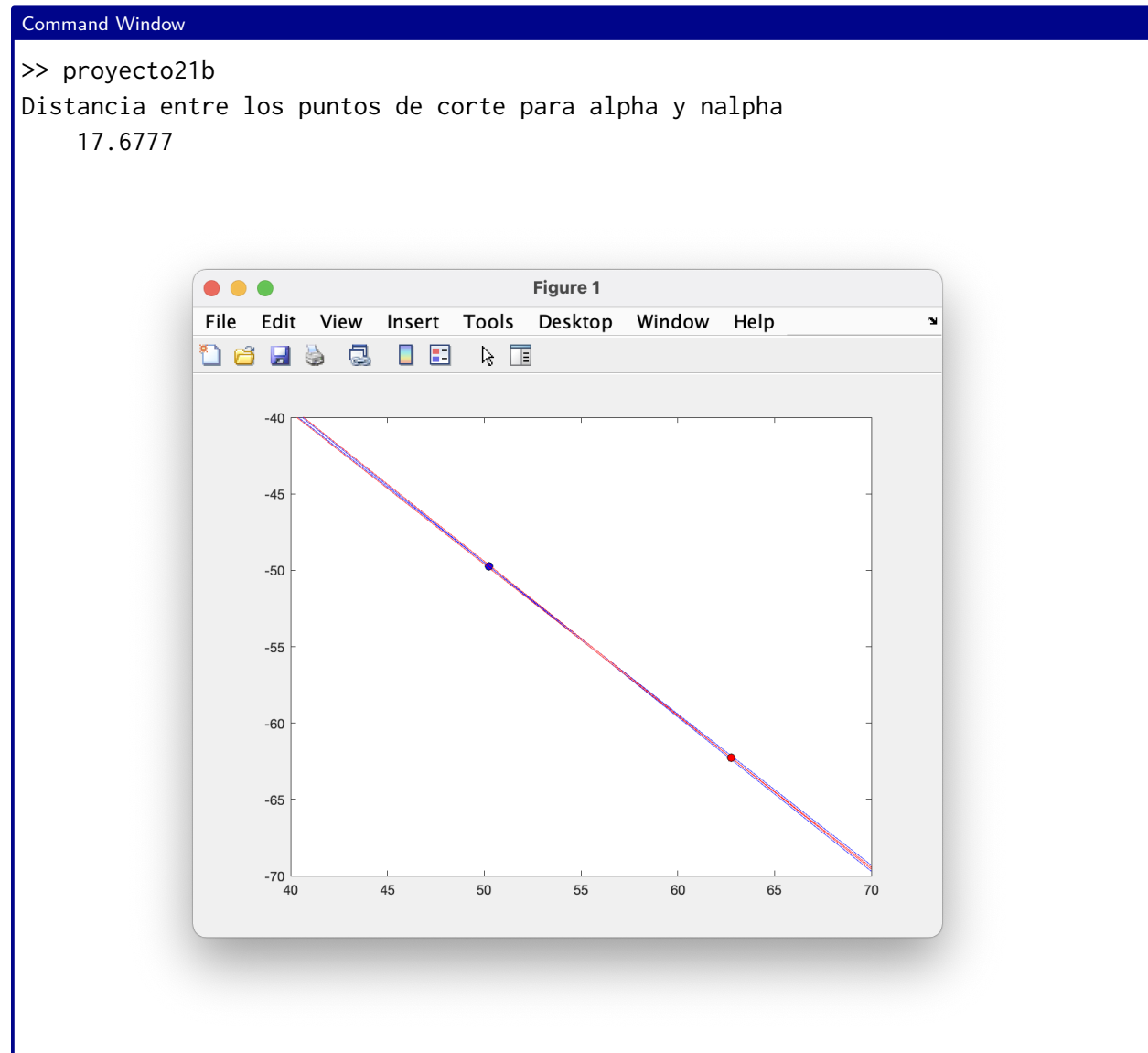
disp('Distancia entre los puntos de corte para alpha y nalpna')
d=norm([pcx pcy]-[npcx npcy]);
format short
disp(d)
```

Una vez ejecutado el código en MATLAB®, puedes observar que la distancia entre los

puntos de corte (para $\alpha = 0.99$ y $\tilde{\alpha} = \alpha + 0.002 = 0.992$) es ahora casi nueve mil veces la perturbación introducida, puesto que

$$d = \frac{17.6777}{0.002} = 8.8389 \times 10^3.$$

Fíjate en la imagen y observa la separación evidente entre ambos puntos de corte.



Proyecto 2.2

El texto que inicia el razonamiento relativo a la transmisión del error en la sustracción debería decir:

Consideremos la sustracción, en aritmética exacta, de dos cantidades,

$$x = a - b,$$

donde a y b tienen el mismo signo. En aritmética de punto flotante, la operación se expresa como

$$\tilde{x} = \text{fl}(\text{fl}(a) - \text{fl}(b)).$$

La simplificación mencionada en el guion consiste en asumir que

$$\text{fl}(\text{fl}(a) - \text{fl}(b)) = \text{fl}(a) - \text{fl}(b).$$

Si prescindimos de esta suposición y denotamos por δ_r el error relativo de redondeo de la cantidad $\text{fl}(a) - \text{fl}(b)$, es decir,

$$\text{fl}(\text{fl}(a) - \text{fl}(b)) = (\text{fl}(a) - \text{fl}(b))(1 + \delta_r),$$

se deduce que

$$\left| \frac{\tilde{x} - x}{x} \right| = \left| \frac{a\delta_a - b\delta_b}{a - b}(1 + \delta_r) + \delta_r \right| \approx \left| \frac{a\delta_a - b\delta_b}{a - b} + \delta_r \right|,$$

pues $|\delta_a\delta_r|, |\delta_b\delta_r| \ll \min\{|\delta_a|, |\delta_b|, |\delta_r|\}$.

Así,

$$\left| \frac{\tilde{x} - x}{x} \right| \lesssim \max\{|\delta_a|, |\delta_b|\} \frac{|a| + |b|}{|a - b|} + |\delta_r|,$$

Las acotaciones de los errores relativos de la suma, multiplicación y división las realizaremos asumiendo simplificaciones similares a la anterior; no obstante, indicaremos también los resultados que se obtendrían omitiendo dichas simplificaciones.

De manera similar a la sustracción, al sumar, multiplicar y dividir dos números reales a y b , las correspondientes operaciones en punto flotante se efectúan sobre los resultados de redondear estos números, esto es, $\text{fl}(a) = a(1 + \delta_a)$ y $\text{fl}(b) = b(1 + \delta_b)$, siendo $|\delta_a|$ y $|\delta_b|$, respectivamente, los errores relativos de redondeo asociados a las cantidades a y b .

- La suma, en aritmética exacta, se expresa como $x = a + b$, donde a y b tienen el mismo signo. En aritmética de punto flotante la operación se expresa como $\tilde{x} = \text{fl}(a) + \text{fl}(b)$.

Se verifica que

$$\left| \frac{\tilde{x} - x}{x} \right| = \left| \frac{a\delta_a + b\delta_b}{a + b} \right| \leq \max\{|\delta_a|, |\delta_b|\} \frac{|a| + |b|}{|a + b|}.$$

Pero, dado que a y b tienen el mismo signo,

$$\frac{|a| + |b|}{|a + b|} = 1.$$

Así,

$$\left| \frac{\tilde{x} - x}{x} \right| \leq \max\{|\delta_a|, |\delta_b|\}.$$

Si no asumimos la simplificación $\text{fl}(\text{fl}(a) + \text{fl}(b)) = \text{fl}(a) + \text{fl}(b)$ y llamamos δ_s al error relativo de redondeo de $\text{fl}(a) + \text{fl}(b)$, es decir,

$$\text{fl}(\text{fl}(a) + \text{fl}(b)) = (\text{fl}(a) + \text{fl}(b))(1 + \delta_s),$$

se obtiene

$$\left| \frac{\tilde{x} - x}{x} \right| = \left| \frac{a\delta_a + b\delta_b}{a + b}(1 + \delta_s) + \delta_s \right| \approx \left| \frac{a\delta_a + b\delta_b}{a + b} + \delta_s \right|,$$

pues $|\delta_a\delta_s|, |\delta_b\delta_s| \ll \min\{|\delta_a|, |\delta_b|, |\delta_s|\}$.

Por tanto,

$$\left| \frac{\tilde{x} - x}{x} \right| \lesssim \max\{|\delta_a|, |\delta_b|\} + |\delta_s|.$$

- Consideremos ahora los productos $x = a \cdot b$, en aritmética exacta, y $\tilde{x} = \text{fl}(a) \cdot \text{fl}(b)$, en aritmética de punto flotante.

Operando,

$$\left| \frac{\tilde{x} - x}{x} \right| = \left| \frac{ab(\delta_a + \delta_b + \delta_a\delta_b)}{ab} \right| = |\delta_a + \delta_b + \delta_a\delta_b| \approx |\delta_a + \delta_b|,$$

pues $|\delta_a\delta_b| \ll \min\{|\delta_a|, |\delta_b|\}$.

Así,

$$\left| \frac{\tilde{x} - x}{x} \right| \approx |\delta_a + \delta_b|.$$

Si descartamos la simplificación $\text{fl}(\text{fl}(a) \cdot \text{fl}(b)) = \text{fl}(a) \cdot \text{fl}(b)$ y denotamos por δ_p el error relativo de redondeo de $\text{fl}(a) \cdot \text{fl}(b)$, es decir,

$$\text{fl}(\text{fl}(a) \cdot \text{fl}(b)) = (\text{fl}(a) \cdot \text{fl}(b))(1 + \delta_p).$$

se verifica que

$$\left| \frac{\tilde{x} - x}{x} \right| = |(\delta_a + \delta_b + \delta_a\delta_b)(1 + \delta_p) + \delta_p| \approx |\delta_a + \delta_b + \delta_p|,$$

pues $|\delta_a\delta_p|, |\delta_b\delta_p|, |\delta_a\delta_b\delta_p| \ll \min\{|\delta_a|, |\delta_b|, |\delta_p|\}$.

Por tanto,

$$\left| \frac{\tilde{x} - x}{x} \right| \approx |\delta_a + \delta_b + \delta_p|.$$

- Por último, consideremos los cocientes $x = \frac{a}{b}$, en aritmética exacta, y $\tilde{x} = \frac{\text{fl}(a)}{\text{fl}(b)}$, en aritmética de punto flotante (suponiendo que b y $\text{fl}(b)$ son distintos de cero).

Una conocida propiedad de las series geométricas afirma que

$$\frac{1}{1 + \delta_b} = 1 - \delta_b + \delta_b^2 - \delta_b^3 + \dots,$$

siempre que $|\delta_b| < 1$.

Por otra parte,

$$1 - \delta_b + \delta_b^2 - \delta_b^3 + \dots \approx 1 - \delta_b,$$

dado que $|\delta_b|^n \ll |\delta_b|$, para $n \geq 2$.

En consecuencia,

$$\begin{aligned} \left| \frac{\tilde{x} - x}{x} \right| &= \left| \frac{\delta_a - \delta_b}{1 + \delta_b} \right| \\ &\approx |\delta_a - \delta_b| |1 - \delta_b| \\ &\approx |\delta_a - \delta_b|, \end{aligned}$$

puesto que $|\delta_b|^2, |\delta_a \delta_b| \ll \min\{|\delta_a|, |\delta_b|\}$.

Así,

$$\left| \frac{\tilde{x} - x}{x} \right| \approx |\delta_a - \delta_b|.$$

Si omitimos la simplificación

$$\text{fl} \left(\frac{\text{fl}(a)}{\text{fl}(b)} \right) = \frac{\text{fl}(a)}{\text{fl}(b)},$$

y representamos por δ_c el error relativo de redondeo de $\frac{\text{fl}(a)}{\text{fl}(b)}$, es decir,

$$\text{fl} \left(\frac{\text{fl}(a)}{\text{fl}(b)} \right) = \frac{\text{fl}(a)}{\text{fl}(b)} (1 + \delta_c),$$

se tiene que

$$\left| \frac{\tilde{x} - x}{x} \right| = |(\delta_a - \delta_b - \delta_a \delta_b)(1 + \delta_c) + \delta_c| \approx |\delta_a - \delta_b + \delta_c|,$$

pues $|\delta_a \delta_c|, |\delta_b \delta_c|, |\delta_a \delta_b \delta_c| \ll \min\{|\delta_a|, |\delta_b|, |\delta_c|\}$.

Por tanto,

$$\left| \frac{\tilde{x} - x}{x} \right| \approx |\delta_a - \delta_b + \delta_c|.$$

En resumen:

- Cuando se introducen simplificaciones al escribir las operaciones en aritmética de punto flotante, los errores relativos de dichas operaciones están acotados, o bien por el máximo de los errores relativos asociados a a y b , o bien por la suma de estos dos errores relativos. Es decir, en ninguno de los casos se produce un fenómeno de inestabilidad numérica.
- Cuando no se asumen simplificaciones al escribir las operaciones en aritmética de punto flotante, el error relativo de cada operación se ve afectado por la aparición de un sumando cuyo tamaño en ningún caso supera la unidad de redondeo y, por tanto, no genera cambios en la estabilidad.

Proyecto 2.3

Considera el siguiente código de MATLAB®.

Editor ► Raíces de la ecuación de segundo grado proyecto23.m

```

disp(' ')
disp('Tabla 1:')
fprintf('----- \n')
fprintf(' n   error1           x1           x(1)   \n' )
fprintf('----- \n')

for n=1:10
    x=roots([1 -10^n 1]);
    x1=(10^n+sqrt(10^(2*n)-4))/2;
    err1=abs(x(1)-x1)/x(1);
    fprintf('%2d %.4e %.15e %.15e \n',n,err1,x1,x(1))
end

disp(' ')
disp(' ')
disp('Tabla 2:')
fprintf('----- \n')
fprintf(' n   error2           x2           x(2)   \n' )
fprintf('----- \n')

for n=1:10
    x=roots([1 -10^n 1]);
    x2=(10^n-sqrt(10^(2*n)-4))/2;
    err2=abs(x(2)-x2)/x(2);
    fprintf('%2d %.4e %.15e %.15e \n',n,err2,x2,x(2))
end

disp(' ')
disp(' ')
disp('Tabla 3:')
fprintf('----- \n')
fprintf(' n   error2           x2           x(2)   \n' )
fprintf('----- \n')

for n=1:10
    x=roots([1 -10^n 1]);
    x1=(10^n+sqrt(10^(2*n)-4))/2;
    x2=1/x1;
    err2=abs(x(2)-x2)/x(2);
    fprintf('%2d %.4e %.15e %.15e \n',n,err2,x2,x(2))
end

```

Su ejecución genera tres tablas:

```

Command Window
>> proyecto23

Tabla 1:
-----
n   error1          x1          x(1)
-----
1  0.0000e+00  9.898979485566356e+00  9.898979485566356e+00
2  0.0000e+00  9.998999899979995e+01  9.998999899979995e+01
3  0.0000e+00  9.999989999990000e+02  9.999989999990000e+02
4  0.0000e+00  9.99999899999999e+03  9.99999899999999e+03
5  0.0000e+00  9.999999999000000e+04  9.999999999000000e+04
6  0.0000e+00  9.99999999990000e+05  9.99999999990000e+05
7  0.0000e+00  9.9999999999899e+06  9.9999999999899e+06
8  1.4901e-16  1.000000000000000e+08  9.9999999999999e+07
9  0.0000e+00  1.000000000000000e+09  1.000000000000000e+09
10 0.0000e+00  1.000000000000000e+10  1.000000000000000e+10

Tabla 2:
-----
n   error2          x2          x(2)
-----
1  4.2587e-15  1.010205144336442e-01  1.010205144336438e-01
2  1.2211e-13  1.000100020004879e-02  1.000100020005001e-02
3  2.0621e-11  1.000001000022621e-03  1.00000100002000e-03
4  1.1177e-09  1.000000011117663e-04  1.000000010000000e-04
5  3.3844e-07  1.000000338535756e-05  1.00000000100000e-05
6  7.6145e-06  1.000007614493370e-06  1.000000000001000e-06
7  3.4848e-03  9.965151548385620e-08  1.000000000000010e-07
8  2.5494e-01  7.450580596923828e-09  1.000000000000000e-08
9  1.0000e+00  0.000000000000000e+00  1.000000000000000e-09
10 1.0000e+00  0.000000000000000e+00  1.000000000000000e-10

Tabla 3:
-----
n   error2          x2          x(2)
-----
1  0.0000e+00  1.010205144336438e-01  1.010205144336438e-01
2  0.0000e+00  1.000100020005001e-02  1.000100020005001e-02
3  0.0000e+00  1.00000100002000e-03  1.00000100002000e-03
4  0.0000e+00  1.000000010000000e-04  1.000000010000000e-04
5  0.0000e+00  1.00000000100000e-05  1.00000000100000e-05

```

6	0.0000e+00	1.0000000000001000e-06	1.000000000001000e-06
7	0.0000e+00	1.00000000000010e-07	1.0000000000010e-07
8	1.6544e-16	1.00000000000000e-08	1.00000000000000e-08
9	0.0000e+00	1.00000000000000e-09	1.00000000000000e-09
10	0.0000e+00	1.00000000000000e-10	1.00000000000000e-10

En la primera tabla se muestra el cálculo de la raíz

$$x^+ = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{10^n + \sqrt{10^{2n} - 4}}{2}, \quad (6)$$

para $n = 1, 2, \dots, 10$.

A su lado aparece esta misma raíz, pero estimada con la orden de MATLAB® `roots`. Observa su coincidencia, lo que significa que el cálculo efectuado en (6) es correcto. Fíjate también en la columna de los errores (son todos cero salvo uno, que es del orden de la precisión de la máquina). Estos resultados eran previsible, ya que la única resta presente en (6), $10^{2n} - 4$, se realiza entre números de magnitud muy diferente, lo que no ocasiona cancelación numérica.

En la segunda tabla se muestra el cálculo de la otra raíz,

$$x^- = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{10^n - \sqrt{10^{2n} - 4}}{2}, \quad (7)$$

para $n = 1, 2, \dots, 10$.

Fíjate en que ahora sí se produce cancelación numérica. Observa la columna del error y comprueba su crecimiento a medida que n aumenta. Esta amplificación del error se debe a que (7) contiene una resta (la que aparece fuera de la raíz cuadrada) entre números cercanos; de hecho la proximidad de estos números se acentúa a medida que n crece.

En la tercera tabla reaparece la raíz x^- , pero esta vez su cálculo se ha realizado utilizando un procedimiento alternativo a (7) que evita la cancelación numérica y que surge de la propiedad que establece que el producto de las raíces x_1 y x_2 de una ecuación de segundo grado $ax^2 + bx + c = 0$ verifica

$$x_1 x_2 = \frac{c}{a},$$

de donde

$$x_2 = \frac{c}{ax_1}. \quad (8)$$

Así, en el caso de nuestra ecuación, $x^2 - 10^n x + 1 = 0$, se tiene que

$$x^- = \frac{1}{x^+}.$$

Por último, comprobamos que (8) puede deducirse si, en la expresión

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a},$$

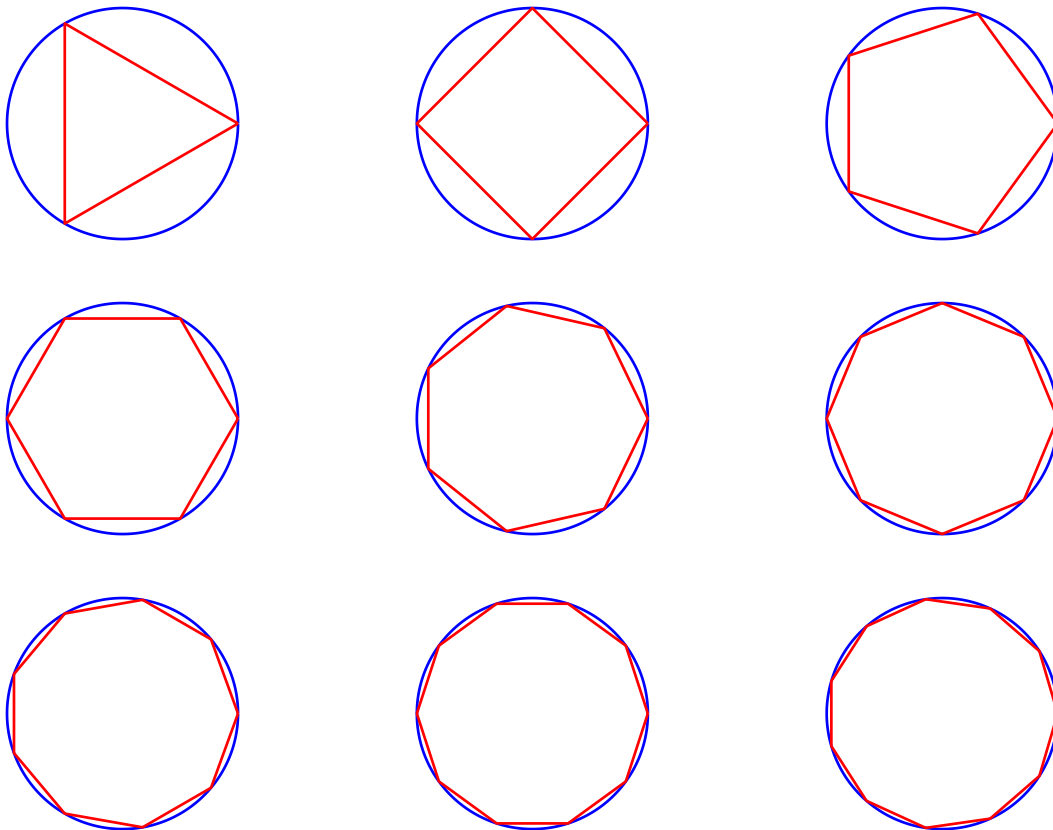
multiplicamos y dividimos por $-b + \sqrt{b^2 - 4ac}$.

En efecto,

$$\begin{aligned}x_2 &= \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{(-b - \sqrt{b^2 - 4ac})(-b + \sqrt{b^2 - 4ac})}{2a(-b + \sqrt{b^2 - 4ac})} \\&= \frac{b^2 - (b^2 - 4ac)}{2a(-b + \sqrt{b^2 - 4ac})} = \frac{4ac}{2a(-b + \sqrt{b^2 - 4ac})} \\&= \frac{2c}{-b + \sqrt{b^2 - 4ac}} = \frac{c}{a \frac{-b + \sqrt{b^2 - 4ac}}{2a}} = \frac{c}{ax_1}.\end{aligned}$$

Proyecto 2.4

La siguiente imagen muestra de manera intuitiva el método de Arquímedes: los polígonos regulares se van acercando a la circunferencia a medida que aumenta el número de lados. Esto sugiere que podemos aproximar la longitud de la circunferencia y por tanto el número π calculando el perímetro de un polígono que tenga un número de lados suficientemente grande.

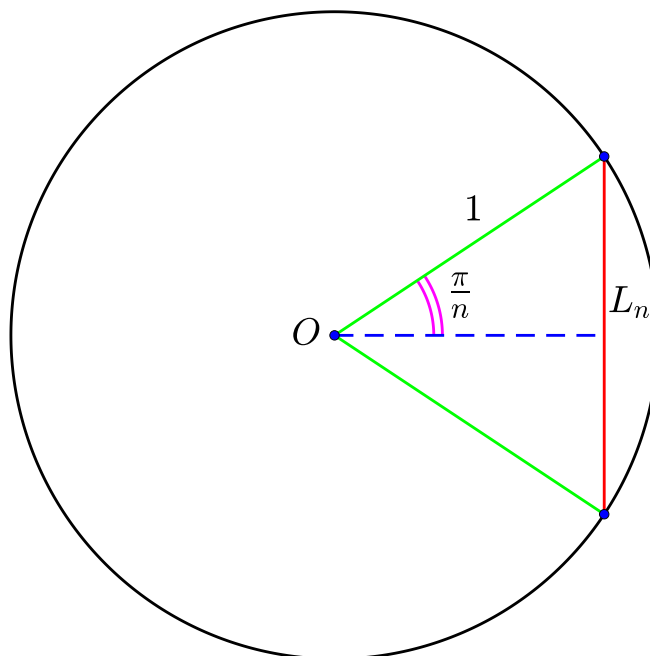


1. Vamos a demostrar, en primer lugar, que

$$f(n) = n \operatorname{sen} \left(\frac{\pi}{n} \right).$$

En la imagen que aparece debajo se representa (en color rojo) un lado cualquiera L_n de un polígono regular de $n \geq 3$ lados inscrito en la circunferencia de radio uno. Trazando desde los puntos extremos de L_n sendos radios (en color verde) y partiendo por la mitad, a través de la línea discontinua (en color azul), el triángulo isósceles formado, se generan dos triángulos rectángulos. El lado opuesto al ángulo $\frac{\pi}{n}$ en cualquiera de los dos mide $\operatorname{sen} \left(\frac{\pi}{n} \right)$ y coincide con la mitad de la longitud de L_n .

Puesto que la mitad del polígono mide n veces la mitad de la longitud del lado L_n , se tiene que $f(n)$ vale n veces $\operatorname{sen} \left(\frac{\pi}{n} \right)$, que es, precisamente, lo que queríamos demostrar.



Veamos ahora que

$$\lim_{n \rightarrow \infty} f(n) = \pi.$$

En efecto,

$$\lim_{n \rightarrow \infty} f(n) = \lim_{n \rightarrow \infty} n \operatorname{sen} \left(\frac{\pi}{n} \right),$$

que es una indeterminación de tipo $0 \cdot \infty$. Transformamos esta indeterminación en otra de tipo $\frac{0}{0}$:

$$\lim_{n \rightarrow \infty} n \operatorname{sen} \left(\frac{\pi}{n} \right) = \lim_{n \rightarrow \infty} \frac{\operatorname{sen} \left(\frac{\pi}{n} \right)}{\frac{1}{n}}.$$

Este último límite coincide con el límite

$$\lim_{x \rightarrow \infty} \frac{\operatorname{sen} \left(\frac{\pi}{x} \right)}{\frac{1}{x}},$$

donde $x \in \mathbb{R}$. Aplicando la regla de L'Hôpital, se tiene que

$$\lim_{x \rightarrow \infty} \frac{\operatorname{sen} \left(\frac{\pi}{x} \right)}{\frac{1}{x}} = \lim_{x \rightarrow \infty} \frac{-\frac{\pi}{x^2} \cos \left(\frac{\pi}{x} \right)}{-\frac{1}{x^2}} = \lim_{x \rightarrow \infty} \pi \cos \left(\frac{\pi}{x} \right) = \pi.$$

Por último, vamos a comprobar que, dado $y_k = f(2^k)$, con $k \geq 2$, se verifica la recurrencia

$$y_{k+1} = 2^{k+1} \sqrt{\frac{1 - \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2}}{2}}.$$

Observa que al hacer $y_k = f(2^k)$, con $k \geq 2$, estamos considerando únicamente aquellos polígonos cuyo número de lados coincide con una potencia de 2. Estamos introduciendo, en última instancia, una aceleración en la convergencia, de modo que podamos aproximarnos al número π con más rapidez.

Se tiene que

$$y_{k+1} = f(2^{k+1}) = 2^{k+1} \operatorname{sen} \left(\frac{\pi}{2^{k+1}} \right).$$

Utilizando la identidad trigonométrica

$$\operatorname{sen}^2 \left(\frac{\alpha}{2} \right) = \frac{1 - \cos(\alpha)}{2},$$

se deduce que

$$\operatorname{sen} \left(\frac{\pi}{2^{k+1}} \right) = \operatorname{sen} \left(\frac{\pi}{2^k 2} \right) = \operatorname{sen} \left(\frac{\pi/2^k}{2} \right) = \sqrt{\frac{1 - \cos \left(\frac{\pi}{2^k} \right)}{2}}.$$

Observa que la raíz cuadrada es positiva, pues todos los ángulos implicados son del primer cuadrante.

A continuación, sustituimos $\cos \left(\frac{\pi}{2^k} \right)$ por

$$\sqrt{1 - \operatorname{sen}^2 \left(\frac{\pi}{2^k} \right)},$$

haciendo uso de la conocida identidad $\operatorname{sen}^2(\alpha) + \cos^2(\alpha) = 1$.

Por consiguiente,

$$\operatorname{sen} \left(\frac{\pi}{2^{k+1}} \right) = \sqrt{\frac{1 - \sqrt{1 - \operatorname{sen}^2 \left(\frac{\pi}{2^k} \right)}}{2}}.$$

Puesto que

$$y_k = f(2^k) = 2^k \operatorname{sen} \left(\frac{\pi}{2^k} \right),$$

se verifica que

$$\operatorname{sen} \left(\frac{\pi}{2^k} \right) = \frac{y_k}{2^k},$$

de donde se obtiene finalmente que

$$y_{k+1} = 2^{k+1} \sqrt{\frac{1 - \sqrt{1 - \left(\frac{y_k}{2^k} \right)^2}}{2}}. \quad (9)$$

2. El siguiente código de MATLAB® implementa la recurrencia (9), generando 30 iteraciones.

Fíjate en que el punto de arranque de la recurrencia es

$$y_2 = f(2^2) = f(4) = 4 \operatorname{sen} \left(\frac{\pi}{4} \right) = 4 \frac{\sqrt{2}}{2} = 2\sqrt{2},$$

es decir, la mitad de la longitud del primer polígono cuyo número de lados es una potencia de 2. Este polígono no es otro que el cuadrado regular inscrito en la circunferencia de radio uno.

Editor ► Método de Arquímedes inestable

proyecto24a.m

```

y=2*sqrt(2); % Valor inicial de la recurrencia
for k=2:31
    y=2^(k+1)*sqrt(0.5*(1-sqrt(1-(y/2^k)^2)));
    err=abs(pi-y)/pi;
    fprintf('k = %2d   pi ~ %17.15f   error = %.4e \n',k+1,y,err)
end

```

Al ejecutar este código en MATLAB® se obtiene:

```

Command Window
>> proyecto24a
k = 3   pi ~ 3.061467458920719   error = 2.5505e-02
k = 4   pi ~ 3.121445152258053   error = 6.4131e-03
k = 5   pi ~ 3.136548490545941   error = 1.6056e-03
k = 6   pi ~ 3.140331156954739   error = 4.0155e-04
k = 7   pi ~ 3.141277250932757   error = 1.0040e-04
k = 8   pi ~ 3.141513801144145   error = 2.5100e-05
k = 9   pi ~ 3.141572940367883   error = 6.2749e-06
k = 10  pi ~ 3.141587725279961   error = 1.5687e-06
k = 11  pi ~ 3.141591421504635   error = 3.9218e-07
k = 12  pi ~ 3.141592345611077   error = 9.8033e-08
k = 13  pi ~ 3.141592576545004   error = 2.4524e-08
k = 14  pi ~ 3.141592633463248   error = 6.4065e-09
k = 15  pi ~ 3.141592654807589   error = 3.8764e-10
k = 16  pi ~ 3.141592645321215   error = 2.6320e-09
k = 17  pi ~ 3.141592607375720   error = 1.4710e-08
k = 18  pi ~ 3.141592910939673   error = 8.1917e-08
k = 19  pi ~ 3.141594125195191   error = 4.6843e-07
k = 20  pi ~ 3.141596553704820   error = 1.2414e-06
k = 21  pi ~ 3.141596553704820   error = 1.2414e-06
k = 22  pi ~ 3.141674265021758   error = 2.5978e-05
k = 23  pi ~ 3.141829681889202   error = 7.5448e-05
k = 24  pi ~ 3.142451272494134   error = 2.7331e-04
k = 25  pi ~ 3.142451272494134   error = 2.7331e-04
k = 26  pi ~ 3.162277660168380   error = 6.5842e-03
k = 27  pi ~ 3.162277660168380   error = 6.5842e-03
k = 28  pi ~ 3.464101615137754   error = 1.0266e-01
k = 29  pi ~ 4.000000000000000   error = 2.7324e-01
k = 30  pi ~ 0.000000000000000   error = 1.0000e+00
k = 31  pi ~ 0.000000000000000   error = 1.0000e+00
k = 32  pi ~ 0.000000000000000   error = 1.0000e+00

```

Observa la aparición de inestabilidad numérica. Aunque el error comienza disminuyendo en las primeras iteraciones, termina aumentando a partir de la iteración decimosexta.

Para detectar la causa de esta inestabilidad numérica, observemos la expresión

$$1 - \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2}. \quad (10)$$

La resta interior a la raíz cuadrada no presenta ningún problema, puesto que se sustraen números de magnitudes muy diferentes:

$$1 \gg \left(\frac{y_k}{2^k}\right)^2 \xrightarrow{k \rightarrow \infty} 0.$$

Sin embargo, la resta exterior a la raíz cuadrada genera cancelación numérica, dado que, a medida que k crece, los números que se sustraen son cada vez más parecidos. En efecto,

$$1 - \left(\frac{y_k}{2^k}\right)^2 \xrightarrow{k \rightarrow \infty} 0$$

de donde

$$1 \approx \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2},$$

para k suficientemente grande.

3. El objetivo de este último apartado es transformar el algoritmo numéricamente inestable generado a partir de (9) en otro numéricamente estable.

Puesto que la resta exterior a la raíz cuadrada en (10) es la causante de la cancelación numérica, nuestra estrategia consistirá en hacer desaparecer esa resta. Para ello, es suficiente con multiplicar y dividir en (10) por

$$1 + \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2}.$$

Se obtiene

$$\begin{aligned} 1 - \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2} &= \frac{\left(1 - \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2}\right) \left(1 + \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2}\right)}{1 + \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2}} \\ &= \frac{1^2 - \left(1 - \left(\frac{y_k}{2^k}\right)^2\right)}{1 + \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2}} = \frac{\left(\frac{y_k}{2^k}\right)^2}{1 + \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2}}. \end{aligned}$$

Sustituyendo en (9) y simplificando se deduce que

$$y_{k+1} = \sqrt{2} \frac{y_k}{1 + \sqrt{1 - \left(\frac{y_k}{2^k}\right)^2}}.$$

Fíjate en que esta nueva recurrencia no contiene ninguna resta que genere cancelación numérica. El siguiente código la implementa, creando una tabla con los resultados obtenidos con 30 iteraciones.

Editor ► Método de Arquímedes estable

proyecto24b.m

```

y=2*sqrt(2); % Valor inicial de la recurrencia
for k=2:31
    y=sqrt(2)*y/sqrt(1+sqrt(1-(y/2^k)^2));
    err=abs(pi-y)/pi;
    fprintf('k = %2d   pi ~ %17.15f   error = %.4e \n',k+1,y,err)
end

```

Su ejecución en MATLAB® produce la siguiente tabla:

```

Command Window
>> proyecto24b
k = 3   pi ~ 3.061467458920719   error = 2.5505e-02
k = 4   pi ~ 3.121445152258053   error = 6.4131e-03
k = 5   pi ~ 3.136548490545940   error = 1.6056e-03
k = 6   pi ~ 3.140331156954754   error = 4.0155e-04
k = 7   pi ~ 3.141277250932774   error = 1.0040e-04
k = 8   pi ~ 3.141513801144303   error = 2.5100e-05
k = 9   pi ~ 3.141572940367093   error = 6.2749e-06
k = 10  pi ~ 3.141587725277162   error = 1.5687e-06
k = 11  pi ~ 3.141591421511202   error = 3.9218e-07
k = 12  pi ~ 3.141592345570119   error = 9.8046e-08
k = 13  pi ~ 3.141592576584875   error = 2.4511e-08
k = 14  pi ~ 3.141592634338565   error = 6.1279e-09
k = 15  pi ~ 3.141592648776988   error = 1.5320e-09
k = 16  pi ~ 3.141592652386594   error = 3.8299e-10
k = 17  pi ~ 3.141592653288996   error = 9.5747e-11
k = 18  pi ~ 3.141592653514596   error = 2.3936e-11
k = 19  pi ~ 3.141592653570997   error = 5.9831e-12
k = 20  pi ~ 3.141592653585097   error = 1.4949e-12
k = 21  pi ~ 3.141592653588622   error = 3.7262e-13
k = 22  pi ~ 3.141592653589504   error = 9.2165e-14
k = 23  pi ~ 3.141592653589724   error = 2.2052e-14
k = 24  pi ~ 3.141592653589779   error = 4.5235e-15
k = 25  pi ~ 3.141592653589793   error = 0.0000e+00
k = 26  pi ~ 3.141592653589797   error = 1.1309e-15
k = 27  pi ~ 3.141592653589798   error = 1.4136e-15
k = 28  pi ~ 3.141592653589798   error = 1.5549e-15
k = 29  pi ~ 3.141592653589798   error = 1.5549e-15
k = 30  pi ~ 3.141592653589798   error = 1.5549e-15
k = 31  pi ~ 3.141592653589798   error = 1.5549e-15
k = 32  pi ~ 3.141592653589798   error = 1.5549e-15

```

Observa el decrecimiento continuo del error hasta estabilizarse en torno a la precisión de la máquina. Se trata de un algoritmo numéricamente estable³⁴, pues permite cal-

³⁴La estabilidad no es perfecta, ya que el último dígito no es correcto.

cular el número π con 15 dígitos significativos correctos. El valor de π con 16 dígitos significativos correctos es

$$\pi \approx 3.141592653589793.$$