*Boletín de Matemáticas*
Nueva Serie, Volumen III (1996), pp.1-23

# RESULTADOS FALSOS CON EL COMPUTADOR

IGNACIO MANTILLA PRADA(*)

**Resumen.** El uso generalizado del computador como herramienta de cómputo debe estar acompañado del conocimiento de la aritmética de máquina y del análisis teórico de los diferentes tipos de errores que se producen en cualquier proceso que involucre operaciones aritméticas con una máquina. Mediante la presentación de algunos ejemplos seleccionados se muestra el efecto que tienen en el resultado final de un cálculo numérico, factores que no siempre son tenidos en cuenta, tales como el error de aproximación inicial, el error de redondeo debido a la aritmética de punto flotante, la capacidad máxima del computador, la estabilidad del método numérico elegido y el condicionamiento del problema a resolver.

*Abstract. Computer aided calculations are often affected by the machine arithmetic, and a theoretic analysis of errors and approximations is often mandatory. Using selected examples we study some of these factors: initial approximations, stability, conditionings, roundoff-errors.*
*Keywords. Machine arithmetic, numerical methods, errors and approximations.*

## 1. Introducción

Aun cuando actualmente existen programas que permiten hacer cálculo simbólico (Mathematica, Maple, Derive, etc.), se presenta un gran espacio de tareas matemáticas de carácter netamente aproximativo, cuando, por diferentes razones, se debe abandonar la idea de obtener una solución exacta. Mediante un proceso analítico podemos demostrar, por ejemplo, existencia, unicidad, quizá

(*)Texto recibido 12/11/96, revisado 29/1/97. Ignacio Mantilla, Departamento de Matemáticas y Estadística, Universidad Nacional, e-mail: igmantil@ciencias.ciencias.unal.edu.co.

también el comportamiento monótono de la solución de una ecuación diferencial. Para objetivos prácticos, sin embargo, se desea tener un conocimiento concreto sobre la solución. En ausencia de una combinación explícita de funciones que represente la solución, conocer al menos de manera aproximada la trayectoria, un valor máximo o mínimo en un intervalo, o poder evaluar la solución en un conjunto de puntos de interés, puede resultar más útil. Tareas de cálculo numérico tendientes a obtener aproximaciones de resultados reales y exactos, implican errores de diferente índole, debidos en gran parte a las aproximaciones sucesivas originadas en las limitaciones de cómputo, pues aunque ya en 1985 el computador Cray-2 podía ejecutar cerca de 1.5 millones de operaciones aritméticas por segundo [19], los números elegidos que pueden ser involucrados en éstas son muy pocos y de características muy especiales, teniendo en cuenta que las calculadoras o los computadores (grandes y pequeños) sólo pueden trabajar con números que son representados por una cantidad limitada de cifras. Más concretamente, sólo un subconjunto finito de números racionales debe cumplir la difícil tarea de representar a todos los números reales. La precisión y cobertura totales son entonces factores que están aún muy lejos de poder ser implementados, de manera que el usuario del computador pueda despreocuparse por completo de la exactitud de los resultados y pronunciar con certeza, la frase común: *con el computador cualquier cálculo es posible y sencillo.* Hay resultados incorrectos que en algunos casos son inevitables, pero ¿hasta qué punto podemos confiar en el resultado que nos arroja el computador? ¿Cuándo podemos abandonar la duda sobre la exactitud de la respuesta que obtenemos en él?

Algunos sencillos ejemplos van a ilustrar el comportamiento de pequeños errores introducidos en un proceso de cálculo y cómo éstos pueden propagarse hasta lograr finalmente afectar el resultado para conducirnos a respuestas sin ningún sentido. También ilustraremos la forma como en algunos casos puede controlarse este fenómeno y mostraremos herramientas teóricas que nos brindan total confianza en los resultados finales. Finalmente investigaremos la razón por la cual, por ejemplo, dos métodos diferentes que deberían arrojar el mismo resultado en el computador, algunas veces conducen a resultados completamente diferentes.

**Notación.** En adelante usaremos la forma $x_{b^m}$ para expresar "$x$ x $b^m$ ($x$ por $b$ a la $m$)". También usaremos la notación " := " para reemplazar el símbolo " = " cuando se trate de una definición. Si $[x]$ denota la función *parte entera de* $x$, la función $Round(x)$ denotará: $[x+0.5]$ para $x \geq 0$ y $[x-0.5]$ para $x < 0$.

## 2. Cálculos con el computador

Un número entero que puede ser calificado de *grande* es el siguiente:

$$p := 2^{216091} - 1 = 0.7460931030646613...6204103815528447\ _{10}{}^{65050}. \tag{2.1}$$

Éste era, hasta hace diez años, el número primo más grande conocido [15]. Actualmente (sept/96) el número $2^{1257787} - 1$ es el mayor primo conocido [12]. $p$ tiene, como se observa, 65050 cifras y si lo escribiésemos aquí en su totalidad, sus cifras llenarían cerca de 25 de estas páginas. Si intentamos calcular $2^{216091} - 1$ con una calculadora o un computador, sin haber implementado previamente un software especial, éstos van a reaccionar con parpadeos o algún mensaje de error. Diremos entonces que no hay que sorprenderse, pues el número, en efecto, es demasiado grande. En cambio, nadie está en contra de efectuar en cualquier computador, aun en una pequeña calculadora, la suma de dos enteros "razonables" como $1.2\ _{10}{}^{25}$ y 7. De manera instantánea obtenemos como resultado:

$$1.2\ _{10}{}^{25} + 7 = 1.2\ _{10}{}^{25}. \tag{2.2}$$

Puesto que 7 no es el elemento neutro para la suma, finalmente nos alegramos de comprobar que el hombre es, "a veces", más listo que el computador.

Examinemos ahora un ejemplo que requiere algo de análisis. Consideremos la sucesión

$$\left(\frac{n!}{x^n}\right)_{n\in\mathbb{N}}, \quad x > 0 \quad \text{fijo}. \tag{2.3}$$

Si elegimos $x = 1000$ y deseamos observar en el computador el comportamiento de la sucesión para $n \to \infty$, el resultado de evaluar los primeros términos es el siguiente:

| $n$ | $n!/1000^n$ | $n$ | $n!/1000^n$ |
|---|---|---|---|
| 1 | $1.0000000000000000E-0003$ | 21 | $1.19511449599999993E-0054$ |
| 2 | $2.0000000000000000E-0006$ | 22 | $5.22715135999999931E-0058$ |
| 3 | $6.0000000000000000E-0009$ | 23 | $8.62453759999999937E-0061$ |
| 4 | $2.4000000000000000E-0011$ | 24 | $7.75946239999999891E-0064$ |
| 5 | $1.2000000000000000E-0013$ | 25 | $2.07618047999999974E-0066$ |
| 6 | $7.2000000000000000E-0016$ | 26 | $1.85388236799999960E-0069$ |
| 7 | $5.0400000000000000E-0018$ | 27 | $1.48478361599999980E-0072$ |
| 8 | $4.0320000000000007E-0020$ | 28 | $1.37573171199999992E-0075$ |
| 9 | $3.6287999999999995E-0022$ | 29 | $1.24151398400000005E-0078$ |
| 10 | $3.6287999999999993E-0024$ | 30 | $1.40928614400000005E-0081$ |
| 11 | $3.9916799999999964E-0026$ | 31 | $7.38197504000000053E-0085$ |
| 12 | $4.7900159999999980E-0028$ | 32 | $2.14748364800000015E-0087$ |
| 13 | $1.9320535039999982E-0030$ | 33 | $2.14748364800000007E-0090$ |
| 14 | $1.2789452799999994E-0033$ | 34 | $0.00000000000000000E+0000$ |
| 15 | $2.0043100159999982E-0036$ | 35 | $0.00000000000000000E+0000$ |
| 16 | $2.0041891839999991E-0039$ | 36 | $0.00000000000000000E+0000$ |
| 17 | $2.8852224000000002E-0043$ | 37 | $0.00000000000000000E+0000$ |
| 18 | $8.9843023999999930E-0046$ | 38 | $0.00000000000000000E+0000$ |
| 19 | $1.0964172799999995E-0049$ | 39 | $0.00000000000000000E+0000$ |
| 20 | $2.1021327359999973E-0051$ | 40 | $0.00000000000000000E+0000$ |

La sucesión, para $n > 33$, es 0; es decir, converge a 0 rápidamente. Es la respuesta que arroja el computador. Pero un sencillo análisis teórico permite demostrar que esta sucesión diverge a $+\infty$ [18, cap.I, ej.4].

Un programa implementado hace algunos años en la Universidad Nacional, facilitaba al profesor el procesamiento de las notas de sus estudiantes, llenando tres casillas correspondientes a las notas obtenidas en exámenes parciales y una casilla correspondiente a la nota del examen final. Las casillas debían llenarse con números enteros entre 0 y 50. El programa lograba que el computador automáticamente calculara el 70% del promedio de previas, el 30% del examen final y la respectiva suma. Un desafortunado estudiante, al finalizar el semestre, discutía con el profesor y sostenía que su nota definitiva era aprobatoria y no 29. Sus notas previas eran, $p_1 = 30$, $p_2 = 35$, $p_3 = 32$ y en el examen final había obtenido $e = 24$. Un sencillo análisis muestra que ambos tenían razón; sólo estaban usando una aritmética diferente. La nota definitiva *real* se obtiene de

$$d := 0.7\left(\frac{1}{3}\sum_{i=1}^{3} p_i\right) + 0.3e, \quad p_i, e \in \mathbb{Z}, \quad i = 1(1)3, \tag{2.4}$$

que para el ejemplo sería, $d = 29.8333...$ Puesto que ésta debe ser entera, aproximando correctamente tenemos, $D = 30$ $(= Round(d))$. Así que el estudiante tenía razón. Pero de otra parte, el formulario de registro exigía cantidades enteras en todas sus casillas y el computador no evaluaba entonces (2.4), sino, (2.5)

$$D := Round\underbrace{\left\{\underbrace{0.7\underbrace{\left[\underbrace{Round\underbrace{\left(\frac{1}{3}\sum_{i=1}^{3} p_i\right)}_{32.333...}}_{32}\right]}_{22.4}}\right\}}_{} + \underbrace{Round(0.3e)}_{7.2}, \quad p_i, e \in \mathbb{Z},\ i = 1(1)3.$$

Por lo tanto,

$$D = Round(22.4) + Round(7.2) = 22 + 7 = 29.$$

Esta pérdida de cifras decimales en cada operación, debida al redondeo, era entonces la causa del erróneo resultado final, pero justificaba al profesor en su obligación de realizar sólo una "aritmética entera".

En [15] se menciona que hace algunos años fue descubierto un empleado bancario, quien había modificado el programa del Banco para que todos los recortes debidos al redondeo de los centavos, en el cálculo de los intereses de las cuentas de los clientes, fueran transferidos directamente a su propia cuenta.

Algunos cálculos simples, tales como la evaluación de $x^n$, $x \in \mathbb{R}$, $n \in \mathbb{N}$, $n \gg 1$, pueden arrojar respuestas diferentes en computadores distintos. El resultado de multiplicar $n$ veces una cantidad real $x$, generalmente es diferente al resultado de calcular $e^{n \ln x}$. Este tipo de cálculos aparecen frecuentemente en múltiples aplicaciones. Por ejemplo, cada consignación $S_0$ va a rendir, después de $n$ días, con un interés diario del $r\%$, una suma,

$$S_n := S_{n-1}\left(1 + \frac{r}{100}\right) = S_0\left(1 + \frac{r}{100}\right)^n, n = 1(1)\cdots.$$

Si la suma inicial, $S_0$, es grande, una inadecuada aproximación en la evaluación de la función $x^n = e^{n \ln x}$, puede significar la pérdida o ganancia de unos cuantos miles de pesos después de algunos años. Una expresión similar aparece en la solución del siguiente problema. *Después de muchos años de observaciones se ha determinado que la probabilidad de que caiga nieve sobre París el $1^0$ de Mayo es $\frac{1}{250}$. ¿Con qué probabilidad caerá nieve sobre París el $1^0$ de Mayo de alguno de los próximos 500 años ?* (Solución: $1 - (1 - \frac{1}{250})^{500}$ ).

Como puede observarse, muy frecuentemente debemos hacer cálculo de potencias (al evaluar cualquier polinomio), que algunas veces requiere el conocimiento y el uso de estrategias diferentes a oprimir la tecla $\boxed{x^y}$ en una calculadora de bolsillo. Más ejemplos se encuentran en [15], [9], [13] y [14].

Para poder analizar los fenómenos ilustrados en esta primera parte, vamos a considerar los números con los cuales trabaja el computador y la aritmética que es usada.

## 3. Números de máquina y aritmética de punto flotante

Cada número real $x$ puede ser representado en un sistema numérico de base $B \in \mathbb{N}$, $B > 1$, en la forma:

$$x = \pm \left( \sum_{i=1}^{\infty} a_i \ {}_{B^{-i}} \right) \cdot {}_{B^L} = \pm 0.a_1 a_2 \ldots \ {}_{B^L}, \tag{3.1}$$

con $a_i \in \{0, 1, \ldots, B-1\}$, $i = 1, 2, \ldots$, y $L \in \mathbb{Z}$.

La representación de $x$ en (3.1) se llama *representación de punto flotante de $x$ para la base $B$* (ver [20]). $L$ se llama *exponente* y $a_1 a_2 \ldots$ se llama *mantisa* de la representación (3.1) en base $B$ de $x$. Si $x \neq 0$, a través de cambios en $L$ puede lograrse siempre que la primera cifra de la mantisa no sea 0 ($0.001 \ {}_{B^L} = 0.1 \ {}_{B^{L-2}}$). En este caso se habla de una representación de punto flotante *normalizada*. Esta forma de punto flotante va a ser usada ahora para representar los números en un computador, pero con dos decisivas restricciones, debidas a la limitada memoria de la que disponemos: el espacio para el exponente y la longitud de la mantisa.

El conjunto $\mathcal{M}$ de los números, en forma de punto flotante normalizados, que pueden ser representados en un computador, llamados *números de máquina*, es entonces dependiente de la base $B$, de la longitud de la mantisa $M \in \mathbb{N}$ y del intervalo del exponente $[k, K]^* := [k, K] \cap \mathbb{Z}$, con $k, K \in \mathbb{N}$. $\mathcal{M}$ queda completamente determinado por estas cantidades y está dado explícitamente por:

$$\begin{aligned} \mathcal{M} = \mathcal{M}(B, M, k, K) = \{0\} \cup \{ & \pm 0.a_1 a_2 \ldots a_M \ {}_{B^L} \mid \\ & L \in \{k, k+1, \ldots K\}, a_1, a_2, \ldots, a_M \in \{0, 1, \ldots, B-1\}, a_1 \neq 0\}. \end{aligned} \tag{3.2}$$

$\mathcal{M}$ es un subconjunto finito de los números racionales. El computador únicamente puede calcular con números de $\mathcal{M}$ y solamente puede arrojar como resultados números de $\mathcal{M}$.

Un sencillo modelo que nos permite observar los números de $\mathcal{M}$ es

$$\mathcal{M}(2,2,-2,2) = \{0\} \cup \{\pm 0.1a\ _{2^L} \mid L \in \{-2,-1,0,1,2\}, a \in \{0,1\}\}.$$

El menor número positivo en $\mathcal{M}$ es

$$0.10\ _{2^{-2}} = 1\ _{2^{-3}} = \frac{1}{8},$$

y el mayor es

$$0.11\ _{2^2} = 11 = 2^0 + 2^1 = 3.$$

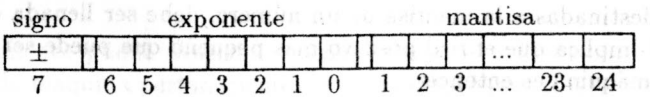
En el conjunto

$$\mathcal{M}^+ = \left\{0, \frac{1}{8}, \frac{3}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{2}, \frac{3}{4}, 1, \frac{3}{2}, 2, 3\right\},$$

están todos los números no negativos, en orden ascendente, de $\mathcal{M}$. Se ve que su distribución en la recta real no es uniforme y que en cercanías de 0 ésta se hace más densa. Los números mayores que 3 no pueden ser dominados por este "mini-microcomputador" y conducen a un mensaje de error. En este caso se habla de *overflow*. Los números entre $-3$ y 3, que no son números de máquina, van a ser aproximados al siguiente número de máquina más cercano: $2.6 \to 3$, $-0.7 \to -0.75\ (= -\frac{3}{4})$.

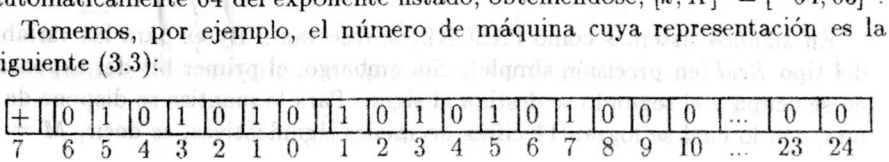
Para el cálculo de la expresión (2.1), por ejemplo, $K$ es muy pequeño y se presenta *overflow*. En (2.3), en cambio, $|k|$ es muy grande, de tal manera que para $n \geq 34$, $\frac{n!}{1000^n}$ es reemplazado por 0 (*underflow*).

Examinemos ahora un modelo más real: tomemos, a manera de ejemplo, el computador IBM 360/370. Para el uso de un número de punto flotante, con precisión simple, este computador reserva 4 Bytes (1 Byte = 8 bits) de memoria y cada dígito binario ocupa un bit. Para la representación de estos números de máquina se usa el modelo $\mathcal{M}(B, M, k, K)$, con $B = 16$, $M = 6$, $k = -64$ y $K = 63$. De los 32 bits disponibles, el primero se reserva para el signo, los 7 siguientes para el exponente y los restantes 24 para la mantisa, tal como se ilustra a continuación:




El exponente de 7 bits nos proporciona entonces un rango entre 0 y $\sum_{k=0}^{6} 2^k = 2^7 - 1 = 127$. Para poder representar exponentes negativos, se resta automáticamente 64 del exponente listado, obteniéndose, $[k, K]^* = [-64, 63]^*$.

Tomemos, por ejemplo, el número de máquina cuya representación es la siguiente (3.3):



Los bits que se usan para describir el exponente, 0101010, indican que éste es:

$$0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 42.$$

Los 24 bits finales indican que la mantisa es:

$$(\frac{1}{2})^1 + (\frac{1}{2})^3 + (\frac{1}{2})^5 + (\frac{1}{2})^7,$$

entonces el número de máquina dado en (3.3) representa al número real

$$(3.4)\quad y = \left[(\frac{1}{2})^1 + (\frac{1}{2})^3 + (\frac{1}{2})^5 + (\frac{1}{2})^7\right] \cdot 16^{42-64} = 0.2145701662201152_{10^{-26}}.$$

Obsérvese que, de (3.4), el siguiente número de máquina más pequeño que $y$ tiene la representación (3.5):

| + | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | ... | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... | 23 | 24 |

la cual conduce al número real

$$\begin{aligned} x &= \left[(\frac{1}{2})^1 + (\frac{1}{2})^3 + (\frac{1}{2})^5 + (\frac{1}{2})^7 - \left(\frac{1}{2}\right)^{24}\right] \cdot 16^{42-64} \\ &= \left[(\frac{1}{2})^1 + (\frac{1}{2})^3 + (\frac{1}{2})^5 + \sum_{k=8}^{24} (\frac{1}{2})^k\right] \cdot 16^{42-64} \\ (3.6)\qquad &= 0.2145701469608157672_{10^{-26}}. \end{aligned}$$

Comparando (3.4) con (3.6) puede observarse que los números $x$ y $y$ sólo coinciden hasta la sexta cifra decimal. Nuestro número de máquina $y$ es usado para representar a todos los números reales en el intervalo $(x, y]$.

Una restricción adicional exige que por lo menos una de las cuatro primeras casillas, destinadas a la mantisa de un número, debe ser llenada con el dígito 1, lo cual implica que el real positivo más pequeño que puede ser almacenado por esta máquina es entonces:

$$\left(\frac{1}{2}\right)^4 \cdot 16^{-64} = 16^{-65} \approx 10^{-78}.$$

En algunos sistemas como PASCAL, se reservan 6 Bytes para las variables del tipo *Real* (en precisión simple). Sin embargo, el primer bit del exponente no se ocupa y el segundo se destina al signo. Para la mantisa se dispone de 40 bits, con lo cual se logran 11 cifras decimales significativas, es decir, $M = 11$.

El rango del exponente controlado por $[k, K]^*$ permite, en este caso, el uso de reales positivos mayores que $2.9\ _{10^{-39}}$ y menores que $1.7\ _{10^{38}}$. Los números reales positivos menores que $2.9\ _{10^{-39}}$ son aproximados a cero y en la ejecución de un proceso aritmético éste no se detiene por esta causa, produciendo a veces alteraciones significativas en el resultado final, tal como lo muestra el cómputo de (2.3) en la seccion anterior.

Para el cálculo con variables de tipo entero (*Integer* en PASCAL o FORTRAN, *Int* en C) se destinan 16 bits, de los cuales el primero está reservado al signo. Por lo tanto, el entero positivo más grande, de este tipo, que puede ser usado es $\sum_{k=0}^{14} 2^k = 2^{15} - 1 = 32767$. Las variables que son declaradas como enteros "normales", no deben ser entonces mayores que este número, ni menores que $-32768$. Cualquier cantidad entera fuera de este rango conduce a un error. Así por ejemplo, si $m$ es declarada como una variable entera y le asignamos el valor, $m := 185$, el resultado de calcular $m^2$ será $-31311$ ($\neq 34225 = m^2$). La razón de este absurdo resultado es que en el computador se ejecuta la siguiente aritmética:

$$m^2 = \underbrace{(32767 + 1)}_{= -32768} + 1457 = -31311.$$

Naturalmente, éste es un error que puede evitarse de múltiples maneras. Primero, declarando $m$ como un "entero largo" (*LongInt*), de tal manera que la máquina reserve 4 Bytes para nuestra variable $m$ y nos ofrezca entonces un rango entre $-2.147.483.648$ y $2.147.483.647$. Otra forma de evitar el error, es calcular $m^2$ como el producto de $1.0 \cdot m \cdot m$, ya que la introducción del factor real 1.0, reemplaza el *producto entero*: $(\mathbb{Z} \times \mathbb{Z}) \to \mathbb{Z}$, por un producto $(\mathbb{Z} \times \mathbb{Z}) \to \mathbb{R}$ con resultado real. Sin embargo, debe tenerse en cuenta que si este resultado está definiendo una nueva variable de tipo entero, se va a producir un error que interrumpirá la ejecución.

## 4. Correcto redondeo

Formalmente puede describirse, como sigue, el redondeo correcto de los números de máquina pertenecientes al conjunto $\mathcal{M}$ (ver (3.2)).

Sea $z := 0.1\ _{B^k}$ el número de máquina más pequeño. Entonces,

$$Z := \sum_{i=1}^{M} (B-1)\ _{B^{-i}} \cdot B^K = (B-1)B^K \left[ \frac{B^{M+1} - 1}{(B-1)B^M} - 1 \right] = (1 - B^{-M})B^K$$

es el número de máquina más grande en $\mathcal{M}$. Llamemos

$$I_{\mathcal{M}} := [-Z, -z] \cup [z, Z],$$

de tal manera que para $x \in I_{\mathcal{M}}$, $x = \pm 0.a_1a_2 \ldots a_M a_{M+1} \cdots \cdot B^L$, $-k \leq L \leq K$, $a_1 \neq 0$, la aplicación de redondeo $rd : I_{\mathcal{M}} \to \mathcal{M}$ asigne a $x$ el número de máquina definido por (4.1):

$$rd(x) := \pm \begin{cases} 0.a_1a_2 \ldots a_M \cdot B^L, & si \ \ a_{M+1} < \frac{B}{2} \quad \text{(defecto)} \\ (0.a_1a_2 \ldots a_M + B^{-M}) \cdot B^L, & si \ \ a_{M+1} \geq \frac{B}{2} \quad \text{(exceso)}. \end{cases}$$

Si $x \in (-z, z)$ (underflow), entonces $x$ va a ser tomado como el cero y se permite, en la mayoría de los casos, continuar la ejecución. Si $|x| > Z$ (overflow), hay un mensaje de error que interrumpe cualquier ejecución. Eventualmente, $x$ puede ser reemplazado por $Z$ o $-Z$, pero en ambos casos, una aproximación razonable a un número de máquina resulta muy dudosa. Para todos los demás casos, en cambio, el error relativo que se comete, debido al redondeo, puede acotarse por medio de:

$$(4.2) \qquad error\ relativo \ := \frac{|rd(x) - x|}{|x|} < \frac{1}{2} \cdot B^{1-M} =: eps, \ \text{para} \ x \in I_{\mathcal{M}}.$$

*eps* se llama *precisión de máquina* y no depende de $x$.

*Demostración (4.2).* De (4.1) tenemos que,

$$\begin{aligned} &|rd(x) - x| \\ &= \begin{cases} |0.a_1a_2 \ldots a_M \cdot B^L - 0.a_1a_2 \ldots a_M a_{M+1} \cdots \cdot B^L| & \text{si } a_{M+1} < \frac{B}{2} \\ |(0.a_1a_2 \ldots a_M + B^{-M}) \cdot B^L - 0.a_1a_2 \ldots a_M a_{M+1} \cdots \cdot B^L| & \text{si } a_{M+1} \geq \frac{B}{2} \end{cases} \\ &= \begin{cases} |0.a_{M+1}a_{M+2} \cdots \cdot B^{(L-M)}| \text{ si } a_{M+1} < \frac{B}{2} \\ |B^{(L-M)} + 0.a_1a_2 \ldots a_M \cdot B^L - 0.a_1a_2 \ldots a_M a_{M+1} \cdots \cdot B^L| & \text{si } a_{M+1} \geq \frac{B}{2} \end{cases} \\ &= \begin{cases} |0.a_{M+1}a_{M+2} \cdots| \cdot B^{(L-M)} \text{ si } a_{M+1} < \frac{B}{2} \\ |1 - 0.a_M a_{M+1} \cdots| \cdot B^{(L-M)} & \text{si } a_{M+1} \geq \frac{B}{2}. \end{cases} \end{aligned}$$

Si $a_{M+1} < \frac{B}{2}$ entonces $0.a_{M+1}a_{M+2} \cdots < \frac{B}{2}B^{-1} = \frac{1}{2}$. Si $a_{M+1} \geq \frac{B}{2}$, entonces $1 - 0.a_{M+1}a_{M+2} \cdots \leq 1 - \frac{B}{2}B^{-1} = \frac{1}{2}$. Por lo tanto $|rd(x) - x| \leq \frac{1}{2} \cdot B^{(L-M)}$, y puesto que $x \in I_{\mathcal{M}}$, se tiene $a_1 \geq 1$, lo cual nos permite afirmar que $|x| \geq 0.1 \ _{B^L}$, con lo cual se consigue

$$\frac{|rd(x) - x|}{|x| + -} \leq \frac{\frac{1}{2} \cdot B^{(L-M)}}{0.1 \cdot B^L},$$

y se concluye de manera inmediata (4.2).

Cuando se calcula con números de máquina, el resultado puede, fácilmente, sobrepasar este límite para el error relativo, pero mientras no se presente ningún

overflow o underflow, el redondeo conduce siempre a números de $\mathcal{M}$. Por eso, la aritmética con la que trabaja (exactamente: debería trabajar) el computador es la siguiente.

Sea $\circ \in \{*, /, -, +\}$. Definimos la *Aritmética de Punto Flotante* $\boxdot$ de la siguiente manera:

$$\begin{array}{rccl} \boxdot: & \mathcal{M} \times \mathcal{M} & \to & \mathcal{M} \\ & (a,b) & \mapsto & a \boxdot b := rd(a \circ b). \end{array}$$

$a \circ b$ va a ser (mientras sea necesario) evaluado de manera exacta, eventualmente normalizado, después redondeado y finalmente, si es necesario, de nuevo normalizado. Desafortunadamente no está construida esta aritmética ideal en todos los computadores y no necesariamente se tiene, como en (4.2), la afirmación

$$a \circ b \in I_{\mathcal{M}} \Rightarrow \frac{|(a \boxdot b) - (a \circ b)|}{|a \circ b|} < eps,$$

para el error relativo de *Aritmética de Punto Flotante*.

Si examinamos nuevamente la aritmética que interviene en (2.2), eligiendo, por ejemplo, $B = 10$, $M = 10$, $K \geq 30$, tendremos

$$7 \boxplus 1.2_{10^{25}} = rd(0.12000 \cdots \underset{\substack{\uparrow \\ \text{posición } M}}{0} \cdots 007)_{10^{26}} = 1.2_{10^{25}}.$$

Pero ¿qué tamaños tienen $B$, $M$, $K$ y $k$ en cada computador? Generalmente, la base $B$, usada en los computadores, es una potencia de 2. El siguiente programa, escrito en PASCAL, permite determinar la base $B$ (siempre que $B$ no sea divisible por 3) y la longitud de la mantisa $M$, implementadas en un computador.

```
program test;
uses crt;
var      u, x, base, L        : Real;
begin
u := abs(3 * (4/3 - 1) - 1);
x := abs(3 * (2/3 - 0.5) - 0.5);
base := u/x;
L := -ln(x)/ln(base);
writeln('Base para el sistema numérico : ',round(base):4);
writeln('Longitud de la Mantisa        : ' ,round(L):4);
repeat until keypressed;
end.
```

Examinemos el programa brevemente, con el fin de aclarar el algoritmo programado. Hagamos el ejercicio, suponiendo que de antemano sabemos que $B = 10$. Para el cálculo de u necesitamos en primer lugar, el número $\frac{4}{3}$ como número de máquina:

$$\begin{array}{rcccccc} \frac{4}{3} & = & 1.333\ldots & = & 1.1333\ldots_{10^1}. & & \\ rd(\frac{4}{3}) & = & 0.1333\ldots\ 3_{10^1} & & = 1.333\ldots 3 & & 0, \\ & & \uparrow & & & & \uparrow \\ & & M & & & & M \end{array}$$

entonces

$$rd(\tfrac{4}{3}) \boxminus 1 = rd(\tfrac{4}{3}) - 1 = 0.333\ldots 3 \quad \begin{array}{c} 0 \\ \uparrow \\ M \end{array}$$

y

$$3 * \left(rd(\tfrac{4}{3}) \boxminus 1\right) = 0.999\ldots 9 \quad \begin{array}{c} 0. \\ \uparrow \\ M \end{array}$$

Por lo tanto,

$$\begin{array}{rl} u = & |0.999\ldots 9 \quad \begin{array}{c} 0 \\ \uparrow \\ M \end{array} \quad \boxminus 1| \\ = & 0.999\ldots \quad \begin{array}{c} 9 \\ \uparrow \\ M \end{array} \quad \ldots 999\ldots - 0.999\ldots 9 \quad \begin{array}{c} 0 \\ \uparrow \\ M \end{array} \\ = & 0.999\ldots_{10^{(1-M)}} = 10^{(1-M)}. \end{array}$$

De manera análoga se obtiene,

$$\mathrm{x} = 10^{-M};$$

entonces para la variable *base* tenemos

$$base = \frac{\mathrm{u}}{\mathrm{x}} = \frac{10^{(1-M)}}{10^{-M}} = 10.$$

La longitud de la mantisa es

$$L = \frac{-\ln(\mathrm{x})}{\ln(base)} = \frac{-\ln(10^{-M})}{\ln(10)} = M.$$

Los valores de $K$ y $k$ pueden finalmente determinarse, potenciando $B$ y $\frac{1}{B}$ hasta que se presente overflow y underflow (o aproximación a cero) respectivamente.

## 5. Condición y estabilidad

En esta sección se estudian otros dos problemas del cálculo numérico: el condicionamiento del problema a resolver y la estabilidad del método empleado para encontrar su solución. La presentación se hará de la mano de algunos ejemplos ilustrativos.

Consideremos los números $x := 0.41274$ e $y := 0.40860$. Si realizamos la diferencia de estos dos números, con una aritmética de punto flotante de cinco decimales, el resultado es:

$$x - y = 0.00414 = 0.414\ _{10^{-2}} =: z. \tag{5.1}$$

Como siempre, cuando se restan dos números muy cercanos, las primeras cifras desaparecen (se transforman en cero). Reemplacemos $x$ por el número, $x^* := 0.4127$, y efectuemos la diferencia, $x^* - y$. El resultado es ahora:

$$x^* - y = 0.41\ _{10^{-2}} =: z^*.$$

El pequeño cambio relativo en $x$,

$$\left|\frac{x^* - x}{x}\right| = 0.969\ _{10^{-4}} \approx 10^{-4},$$

conduce a un cambio relativo en $z$,

$$\left|\frac{z^* - z}{z}\right| = 0.966\ _{10^{-2}} \approx 10^{-2},$$

lo cual indica que la introducción de un error relativo del orden de $10^{-4}$ en un dato inicial es capaz de producir un error relativo del orden de $10^{-2}$, es decir 100 veces mayor, en el resultado final.

Examinemos ahora, como un segundo ejemplo, el sistema,

$$\begin{aligned} x + y &= 2 \\ x + 1.0001y &= 2.0001. \end{aligned} \tag{5.2}$$

Fácilmente puede verificarse que su solución es $x = 1$, $y = 1$. Si se modifica la segunda ecuación de (5.2) y se plantea el nuevo sistema,

$$\begin{aligned} x^* + y^* &= 2 \\ x^* + 1.0001y^* &= 2.0002, \end{aligned}$$

entonces su solución exacta es $x^* = 0$, $y^* = 2$. Obsérvese que en la modificación de la segunda ecuación de (5.2) se ha introducido un error relativo inicial

$$\left| \frac{2.0001 - 2.0002}{2.0001} \right|,$$

cercano a $\frac{1}{2}\ 10^{-5}$, y éste ha conducido a un resultado, con un error relativo en las dos incógnitas,

$$\left| \frac{x - x^*}{x} \right| = \left| \frac{y - y^*}{y} \right| = 1,$$

que es 200.000 veces mayor que el introducido originalmente.

En los dos ejemplos anteriores se observa cómo pequeñas modificaciones en los datos iniciales (de entrada), conducen a resultados que se apartan considerablemente del resultado final previo. Los problemas en los que se presenta este efecto se llaman *problemas mal condicionados*. Este concepto describe una propiedad de ciertos problemas cuyas soluciones son extremadamente sensibles a cambios, aún muy pequeños, en los datos de entrada. Con datos iniciales dudosos o provenientes de mediciones aproximadas, los problemas mal condicionados conducen a soluciones erróneas, independientemente del computador y el método empleados.

Para examinar en forma general el mal condicionamiento ilustrado en el primer ejemplo (ver (5.1)), podemos calcular fácilmente la *condición* del problema *Resta de dos números*. El error relativo en el resultado de la diferencia se expresa como:

$$\frac{z^* - z}{z} = \frac{(x^* - y^*) - (x - y)}{x - y} = \frac{x^* - x}{x - y} - \frac{y^* - y}{x - y}$$
$$= \left(\frac{x}{x - y}\right)\left(\frac{x^* - x}{x}\right) - \left(\frac{y}{x - y}\right)\left(\frac{y^* - y}{y}\right).$$

Las expresiones $\frac{x}{z}$ y $\frac{y}{z}$, que aparecen como factores de los errores relativos de $x$ y de $y$ respectivamente, se llaman *números relativos de condición*. Estas cantidades nos dan información sobre la forma como el error relativo de los datos iniciales $x$ e $y$, respectivamente, se fortifica o se debilita. Si $x \approx y$ entonces estos factores van a ser grandes y el problema está mal condicionado. En caso contrario, el problema está más o menos bien condicionado, dependiendo de qué tan cercanos estén $\left|\frac{x}{z}\right|$ y $\left|\frac{y}{z}\right|$ de 1. Para el primer ejemplo se tiene, $\left|\frac{x}{z}\right| \approx 100 \gg 1$, con lo cual no cabe ninguna duda sobre el mal condicionamiento del problema (5.1). Para el segundo ejemplo, la matriz $A$, asociada al sistema (5.2), tiene un *número de condición* ($Cond(A) = \|A\| \cdot \|A^{-1}\|$, [13, cap.4, §4]) significativamente distante de 1 ( $Cond(A) > 4\ _{10^4} \gg 1$ ).

Ahora veremos cómo una desafortunada elección del método puede conducirnos a resultados totalmente erróneos, independientemente del computador y aún tratándose de problemas bien condicionados.

Cuando $x \approx y$, al realizar su diferencia aparecen ceros que después de normalizar el resultado van a desaparecer para permitir el traslado a la izquierda de las últimas cifras de la derecha, causando la fortificación del error relativo. Si esta situación se presenta en medio de un algoritmo, puede entonces conducir a un resultado completamente falso. Para ilustrar este hecho consideremos simultáneamente las funciones

$$f(x) := \frac{1}{1 + 2x} - \frac{1 - x}{1 + x}$$

y

$$g(x) := \frac{2x^2}{(1 + 2x)(1 + x)} = f(x).$$

La evaluación de $f$ y $g$ para diferentes valores de $x$ es presentada en la siguiente tabla:

| $x$ | $f(x)$ | $g(x)$ |
|---|---|---|
| $2.36475946E-030$ | $0.00000000000000E+000$ | $1.11841746073190E-059$ |
| $2.36475946E-018$ | $0.00000000000000E+000$ | $1.11841746073190E-035$ |
| $2.36475946E-009$ | $1.11672823766007E-017$ | $1.11841745279753E-017$ |
| $2.36475946E-006$ | $1.11840952623198E-011$ | $1.11840952641087E-011$ |
| $2.36475946E-003$ | $1.11052665516771E-005$ | $1.11052665516771E-005$ |
| $2.36475946E+000$ | $5.80138593831970E-001$ | $5.80138593831970E-001$ |

Los dos primeros resultados de evaluar $f$ en $x$ son claramente falsos, pues para valores positivos de $x$, $f$ es positiva. Los siguientes dos valores obtenidos, son también dudosos, ya que no coinciden con los resultado de evaluar $g$ en $x$. En las dos últimas líneas podemos tener más confianza en la veracidad de los resultados. El fenómeno observado en este ejemplo merece ser analizado más detalladamente. Veamos que el problema: *evaluar $f$ en un valor $x > 0$*, está bien condicionado. En efecto,

$$\begin{aligned} f(x^*) - f(x) &= f'(x)(x^* - x) + O\left(|x^* - x|^2\right) \\ &\doteq f'(x)(x^* - x). \end{aligned}$$

Por lo tanto,

$$\begin{aligned} \frac{f(x^*) - f(x)}{f(x)} &\doteq \left(x\frac{f'(x)}{f(x)}\right)\left(\frac{x^* - x}{x}\right) \\ &= \underbrace{\left(\frac{1}{1+x} + \frac{1}{1+2x}\right)}_{=:\ \mathcal{C}}\left(\frac{x^* - x}{x}\right). \end{aligned}$$

Para $x \geq 0$, el número relativo de condición del problema, denotado por $\mathcal{C}$, satisface $0 < \mathcal{C} \leq 2$, lo que nos permite afirmar que el problema no está mal condicionado. En el algoritmo, sin embargo, se puede observar que para pequeños valores de $x$, las evaluaciones de $\frac{1}{1+2x}$ y $\frac{1-x}{1+x}$ producen cantidades muy cercanas que son restadas, lo cual, debido a la aritmética de punto flotante, introduce pequeños errores relativos en ambas cantidades y éstos conducen a un gran error relativo en el resultado final. Para lograr mejores resultados, debe buscarse entonces, en lo posible, otro algoritmo. Si examinamos los resultados de la evaluación de la función $g$ para pequeños valores de $x$, notamos que el algoritmo impide el error antes mencionado y nos arroja resultados confiables.

El resultado que muestra la tabla anterior fue obtenido mediante un sencillo programa escrito en PASCAL. En cualquier otro lenguaje de programación el resultado será similar, al igual que en programas tales como MATHEMATICA, pues como hemos visto, el problema que aquí se presenta aparece en el método empleado. Con MATHEMATICA, por ejemplo, la evaluación de $f$ en $x = 2.36475946\ _{10-9}$ arroja 0 como resultado.

Puesto que el principal objetivo debe ser la obtención de resultados confiables, nuestro propósito debe ahora encaminarse al desarrollo de algoritmos *estables*. Diremos, en forma vaga, que un algoritmo es estable si, durante el proceso de cálculo, los errores de redondeo generados por la aritmética de punto flotante, no alcanzan a falsificar "de manera significativa" el resultado final. El algoritmo se llama inestable, cuando el error de redondeo se propaga, fuera de control, afectando el resultado final.

Para aclarar el concepto de estabilidad, analizaremos el siguiente ejemplo. Consideremos el sistema

$$Ax = \begin{pmatrix} 0.0005 & 0.9006 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.4508 \\ 1.5 \end{pmatrix} = b.$$

Fácilmente puede verificarse que su solución exacta es $x_1 = 1$, $x_2 = 0.5$. También puede mostrarse que el sistema está bien condicionado [3, cap.3]. Si usamos el algoritmo de Gauss y elegimos $a_{11}$ como elemento pivote, al multiplicar por $q := -\frac{1}{a_{11}} = -2000$ la primera fila y hacer la suma de la segunda fila más la primera, tenemos, calculando con $M = 4$ cifras decimales:

$$\left(\begin{array}{cc|c} 0.0005 & 0.9006 & 0.4508 \\ 0 & -1800.2 & -900.1 \end{array}\right) \underline{rd} \left(\begin{array}{cc|c} 0.0005 & 0.9006 & 0.4508 \\ 0 & -1800 & -900.1 \end{array}\right).$$

Entonces

$$x_2 = \frac{900.1}{1800} = 0.5000555\ldots \underline{rd}\ \ 0.5001, \quad \text{(aceptable, en lugar de 0.5)}.$$

Si ahora reemplazamos el valor encontrado de $x_2$ en la primera ecuación, para encontrar $x_1$, tenemos:

$$\begin{aligned} x_1 &= \frac{0.4508 - 0.9006 \cdot 0.5001}{0.0005} \\ &= \frac{0.4508 - 0.45039006}{0.0005} \ \underline{rd}\ \ \frac{0.4508 - 0.4504}{0.0005} \\ &= \frac{0.0004}{0.0005} \\ &= 0.8 \qquad \text{(en lugar de 1)}. \end{aligned}$$

Es claro que el valor encontrado para $x_1$ es totalmente falso. Debido al pequeño numerador de $x_1$, que resulta de hacer la diferencia entre números muy cercanos, se presenta una considerable pérdida de exactitud. Al elegir como pivote un elemento pequeño en la primera columna, se produjo un factor $q$ grande y dominante, que hizo que la información a partir de la segunda cifra después de la coma prácticamente desapareciera.

Si inicialmente, en cambio, hacemos la permutación de las dos filas de $A$, al multiplicar por $q := 0.0005$ la nueva primera fila y hacer la diferencia de la segunda fila menos la primera, tenemos:

$$\left(\begin{array}{cc|c} 1 & 1 & 1.5 \\ 0.0005 & 0.9006 & 0.4508 \end{array}\right) \rightarrow \left(\begin{array}{cc|c} 1 & 1 & 1.5 \\ 0 & 0.9001 & 0.45005 \end{array}\right)$$

$$\underline{rd} \left(\begin{array}{cc|c} 1 & 1 & 1.5 \\ 0 & 0.9001 & 0.4501 \end{array}\right).$$

Entonces

$$x_2 = \frac{0.4501}{0.9001} = 0.5000555... \underline{rd}\ \ 0.5001 \qquad \text{(aceptable en lugar de 0.5)}$$

y

$$x_1 = \frac{1.5 - 1 \cdot 0.5001}{1} = 0.9999 \qquad \text{(aceptable en lugar de 1)}.$$

Puede apreciarse claramente que la exactitud del resultado en el segundo caso es significativamente mayor. Este ejemplo pone de presente la conveniencia de elegir un elemento pivote lo más grande posible, de tal manera que $q$ resulte pequeño y la pérdida de información debida al redondeo en la eliminación sea mínima. (Para un análisis detallado del error de redondeo en el método de eliminación de Gauss, ver ([17, cap.4, §5].)

En el siguiente ejemplo ilustraremos cómo el uso del cálculo integral, combinado con el empleo de métodos numéricos, puede aprovecharse para impedir la obtención de resultados completamente erróneos. Calculemos

$$\int_{-1}^{1} t^{20} e^{-t} dt. \tag{5.3}$$

Si esta integral se evalúa con una *Fórmula de Cuadratura*, se obtiene rápidamente, y en forma segura, una buena aproximación. Pero si recurrimos únicamente a nuestros conocimientos del cálculo, o a una tabla de integrales, encontraremos:

$$\int_{-1}^{1} t^n e^{-t} dt = n \int_{-1}^{1} t^{n-1} e^{-t} dt + (-1)^n e - \frac{1}{e}, \quad n \geq 0. \tag{5.4}$$

Denotemos por

$$\mu_n := \int_{-1}^{1} t^n e^{-t} dt, \quad n \geq 0,$$

y construyamos la fórmula de recursión:

$$\mu_0 = e - \frac{1}{e},$$

$$\mu_n = n\mu_{n-1} + (-1)^n e - \frac{1}{e}, \quad n \geq 1. \tag{5.5}$$

Fácilmente puede verificarse que

$$\mu_n > 0 \quad \text{para } n \text{ par}, \qquad \mu_n < 0 \quad \text{para } n \text{ impar}, \tag{5.6}$$

(5.7) $|\mu_{2n+1}|$ y $|\mu_{2n}|$ son decrecientes.

Si evaluamos (5.3) utilizando la fórmula de recurrencia (5.5), el computador arroja los siguientes resultados

| $n$ | $\mu_n$ | $n$ | $\mu_n$ |
|---|---|---|---|
| 0 | $2.3504023550E+00$ | 10 | $1.8847405167E-01$ |
| 1 | $-7.3575889005E-01$ | 11 | $-1.0129466767E+00$ |
| 2 | $8.7888457488E-01$ | 12 | $-9.8049577648E+00$ |
| 3 | $-4.4950752038E-01$ | 13 | $-1.3055061219E+02$ |
| 4 | $5.5237227347E-01$ | 14 | $-1.8253581683E+03$ |
| 5 | $-3.2429987766E-01$ | 15 | $-2.7383458685E+04$ |
| 6 | $4.0460308901E-01$ | 16 | $-4.3813298856E+05$ |
| 7 | $-2.5393962195E-01$ | 17 | $-7.4482638917E+06$ |
| 8 | $3.1888537941E-01$ | 18 | $-1.3406874770E+08$ |
| 9 | $-2.1619283033E-01$ | 19 | $-2.5473062094E+09$ |
| | | 20 | $-5.0946124186E+10$ |

Si examinamos detenidamente los valores obtenidos, se nota que a partir de $n = 12$, las propiedades (5.6) y (5.7) ya no se satisfacen y que, por lo tanto, los últimos valores son falsos. La aproximación de la integral (5.3), por medio de $\mu_{20} = -5.0946124186\ _{10^{-10}}$, no tiene entonces ningún sentido. La causa de este resultado desastroso, tal como lo muestra el siguiente sencillo análisis, se manifiesta en la obtención de $\mu_{20}$ con el uso de un algoritmo extremadamente inestable. En efecto, para poder calcular (5.3) usando (5.5), se ha elegido una fórmula de recursión de la forma:

$$\text{(5.8)} \qquad \mu_0, \quad \mu_{n+1} = a_{n+1}\mu_n + b_{n+1}, \quad a_{n+1} \neq 0, \quad \mu_n \neq 0, \quad n \geq 0.$$

Si evaluamos las cantidades

$$\text{(5.9)} \qquad y_0 = \mu_0(1+\varepsilon_0), \quad y_{n+1} = (a_{n+1}y_n + b_{n+1})(1+\varepsilon_{n+1}), \quad n \geq 0,$$

donde $\varepsilon_0$ denota el error relativo en el valor inicial $\mu_0$, y $\varepsilon_{n+1}$ denota el error relativo que se presenta en el cálculo de $a_{n+1}y_n + b_{n+1}$ con una aritmética de punto flotante, encontramos que para nuestro ejemplo se satisface, de (5.9), (5.8) y (5.5):

$$\mu_0 = e - \frac{1}{e}, \quad y_0 = rd(e) \boxminus (1 \boxslash rd(e)), \quad a_n = n, \quad n \geq 1.$$

Bajo la fuerte hipótesis de suponer, para mayor sencillez, $\varepsilon_{n+1} = 0$, se obtiene:

$$y_0 = \mu_0(1+\varepsilon_0)$$
$$y_1 = a_1\mu_0(1+\varepsilon_0) + b_1 = \mu_1 + a_1\mu_0\varepsilon_0$$
$$\vdots$$
$$y_n = \mu_n + a_n a_{n-1} \cdots a_1\mu_0\varepsilon_0.$$

Por lo tanto, en nuestro ejemplo se cumple:

$$\frac{y_n - \mu_n}{\mu_n} = \left(a_n a_{n-1} \cdots a_1 \frac{\mu_0}{\mu_n}\right)\varepsilon_0$$
$$= \left(n!\frac{\mu_0}{\mu_n}\right)\varepsilon_0, \quad n \geq 0.$$

Entoces, el error inicial

$$\varepsilon_0 = \frac{y_0 - \mu_0}{\mu_0}$$

se fortifica enormemente con el factor

$$\left(20!\frac{\mu_0}{\mu_{20}}\right) \approx 4 \ 10^{19}.$$

De manera análoga, el error $\varepsilon_{n+1}$, que en la realidad no desaparece, como lo hemos supuesto, tiene gran influencia en el resultado final. Sin embargo, aún en este caso, en el cual hemos observado un algoritmo totalmente inestable, podemos recurrir a una ayuda. De (5.5) podemos plantear la siguiente fórmula de recursión hacia atrás:

$$\mu_{n-1} = \frac{1}{n}\left[\mu_n - (-1)^n e + \frac{1}{e}\right], \quad n = N, N-1, \ \ldots, 1. \tag{5.10}$$

Pero para poder usar (5.10) con el fin de evaluar $\mu_{20}$, se necesita un valor inicial $\mu_N$, $N > 20$, que desafortunadamente nos es desconocido. Las siguientes tablas muestran, sin embargo, los resultados obtenidos con valores iniciales ($N = 40$) que pueden ser más o menos falsos: $\mu_{40} = 0$, $\mu_{40} = -4000$.

| $n$ | $\mu_n$ | $n$ | $\mu_n$ |
|---|---|---|---|
| 40 | $0.0000000000E+00$ | 40 | $-4.0000000000E+03$ |
| 39 | $-5.8760058874E-02$ | 39 | $-1.0005876006E+02$ |
| 38 | $7.7625671440E-02$ | 38 | $-2.4864768927E+00$ |
| 37 | $-5.9809912725E-02$ | 37 | $-1.2728629599E-01$ |
| 36 | $8.1793279251E-02$ | 36 | $7.9969593217E-02$ |
| 35 | $-6.3016918770E-02$ | 35 | $-6.3067576716E-02$ |
| 34 | $8.6375552179E-02$ | 34 | $8.6374104809E-02$ |
| 33 | $-6.6589023612E-02$ | 33 | $-6.6589066181E-02$ |
| 32 | $9.1502188528E-02$ | 32 | $9.1502187238E-02$ |
| 31 | $-7.0590630202E-02$ | 31 | $-7.0590630242E-02$ |
| 30 | $9.7276471446E-02$ | 30 | $9.7276471445E-02$ |
| 29 | $-7.5104196118E-02$ | 29 | $-7.5104196118E-02$ |
| 28 | $1.0382955341E-01$ | 28 | $1.0382955341E-01$ |
| 27 | $-8.0234742913E-02$ | 27 | $-8.0234742913E-02$ |
| 26 | $1.1133061119E-01$ | 26 | $1.1133061119E-01$ |
| 25 | $-8.6118143992E-02$ | 25 | $-8.6118143992E-02$ |
| 24 | $1.2000172404E-01$ | 24 | $1.2000172404E-01$ |
| 23 | $-9.2933359622E-02$ | 23 | $-9.2933359622E-02$ |
| 22 | $1.3014034284E-01$ | 22 | $1.3014034284E-01$ |
| 21 | $-1.0092100055E-01$ | 21 | $-1.0092100055E-01$ |
| 20 | $1.4215429736E-01$ | 20 | $1.4215429736E-01$ |

A partir de $\mu_{30}$ coinciden, a pesar de los diferentes datos iniciales (falsos), todos los respectivos valores de $\mu_n$ y en especial $\mu_{20}$ es calculada en forma completamente correcta. Un análisis de error, similar al desarrollado anteriormente, muestra que

$$\frac{y_n - \mu_n}{\mu_n} = \left(\frac{n!}{N!}\right)\left(\frac{\mu_N}{\mu_n}\right)\varepsilon_N, \qquad 0 \le n \le N,$$

con lo cual el error relativo se reduce significativamente, gracias al factor

$$\frac{20!}{40!} \approx 3\ _{10^{-30}}.$$

Estas observaciones nos permiten concluir que:

valores iniciales absurdos, falsos y sin sentido pueden arrojar resultados buenos, verdaderos y exactos;
valores iniciales buenos, verdaderos y exactos pueden arrojar resultados absurdos, falsos y sin sentido,
*... pero sólo si ignoramos la matemática numérica.*

## 6. Sobre la matemática numérica

Añadimos a continuación una serie de citas de científicos y matemáticos, traducidas libremente, acerca del interés que presenta la matemática numérica.

P. Henrici [8,19-20]: *Las palabras "análisis numérico" tenían connotaciones que las relacionaban con pesadísimos cálculos aritméticos efectuados por funcionarios matemáticos armados de un lápiz, una tremenda hoja de papel y el indispensable borrador. Después de Euler, la fe en la utilidad numérica de un algoritmo fue decreciendo lenta, pero firmemente. Los matemáticos fueron interesándose en cuestiones de existencia de soluciones con preferencia a su construcción, debido a las difíciles exigencias computacionales que los métodos constructivos presentaban. Es difícil imaginar, por ejemplo, que Émile Picard (1856-1941) pensase jamás en efectuar los pasos requeridos por su método de iteración para resolver, digamos, una ecuación diferencial parcial no lineal. En vista de este sentimiento de impotencia algorítmica es comprensible que los matemáticos se inclinasen cada vez más por el uso de métodos puramente lógicos. Hacia finales de la primera mitad del siglo XX, las matemáticas finalmente parecen llegar al punto de hacer válida la orgullosa afirmación de Jacobi (1804-1851): "las matemáticas sólo sirven al honor del espíritu humano". Pero, por una extraña coincidencia, los algoritmos matemáticos se han visto liberados del yugo del penoso trabajo operatorio en el preciso momento en que las matemáticas puras parecían al fin haberse liberado de sus últimos lazos con el pensamiento algorítmico. La velocidad de cómputo ha aumentado de tal manera, que realizar los más complicados algoritmos, hoy en día, no presenta dificultad alguna.*

J. Albrecht, L. Collatz [1,298]: *La matemática aplicada no debe ocuparse tanto de las cantidades aproximadas, sino de las afirmaciones exactas sobre las cantidades buscadas.*

W. Gander [5,46]: *La matemática numérica y la informática, pero también la matemática numérica y la matemática, se han distanciado. Lo común es la utilización del mismo computador, con el que frecuentemente hablan en diferentes idiomas.*

H.Freudenthal [2,104]: *No es la aproximación algo poco frecuente o desacostumbrado en la Matemática. La excepción es el resultado exacto.*

M. Born [2,158]: *Las cantidades físicas, con excepción de las constantes universales, sólo pueden ser definidas estadísticamente y son por eso en su exactitud de medida, por principio, limitadas.*

R. Furth [2,158]: *La exactitud de una medida está limitada por una constante de error, proporcional a la temperatura absoluta. El único camino para disminuir el error en una medida está entonces en la disminución de la temperatura del aparato para medir.*

Gauss [2,104]: *La carencia en la formación matemática no se deja reconocer*

*en nada, como en la desmedida falta de rigor en el cálculo numérico.*

## Referencias


1. J. Albrecht, L. Collatz,"Beispiele für numerische Mathematik im Schulunterricht", Der Math. Nat. Unt. (1959), 298.
2. J. Blanhenagel, *Numerische Mathematik im Rahmen der Schulmath*, Mannheim/Wien/Zurich: B.I. Wissenschaftverlag, 1985.
3. W. Börsch-Supan, *Numerische Mathematik I*, Johannes Gutenberg-Universität, Mainz, 1989.
4. R. L. Burden, J. D. Faires, *Análisis Numérico*, Grupo Editorial Iberoamérica, 1985.
5. W. Gander,"Numerische Mathematik - Mutter der Informatik", Zeitsch. für den Unterricht in Natwiss. und. Math. (1981), 46.
6. N. Hagander, Y. Sundblad, *Aufgabensammlung Numerische Methoden*, München Wien: R. Oldenbourg Verlag, 1972.
7. G. Hämmerlin, K. H. Hoffmann, *Numerical Mathematics*, New York: Springer Verlag, 1991.
8. P. Henrici, *Elemente der Numerischen Analysis*, Mannheim/Wien/Zurich: B.I. Wissenschaftverlag, 1972.
9. W. Kahan, B. N. Parlett, "Können Sie sich auf Ihrem Rechner verlassen?", Jahrbuch Überblicke Mathematik (1978), 199-216.
10. U. Kulisch, *Wissenschaftliches Rechnen und Programiersprachen*. German Chapter of the ACM, Berichte 10, Teubner Stuttgart, 1982.
11. S. Linnainmaa, "Software for Double-Precision Floating-Point Computation", ACM TOMS 7 (1981), 272-283.
12. *Miami Herald*, Septiembre 3 (1996).
13. G. Miel, *Calculator, calculus and Roundoff Errors*, The American Mathematical Monthly 87 (1980), 243-252.
14. S. M. Rump, "Wie zuverlässig sind die ergebnisse unserer Rechenanlagen?", Jahrbuch Überblicke Mathematik (1983), 163-168.
15. C. Schneider, *Fallstricke beim Rechnen mit Computern*, Numerisches Rechnen, Algorithmen und Computer (Tagungsvorträge), Johannes Gutenberg-Universität, Mainz, 1985.
16. R. D. Skeel, J. B. Keiper, *Elementary Numerical Computing with Mathematica*, New York: McGraw-Hill, 1993.
17. J. Stoer, *Numerische Mathematik 1*, New York: Springer-Lehrbuch, 1990.
18. Y. Takeuchi, *Sucesiones y series I*,8 Departamento de Matemáticas y Estadística, Universidad Nacional de Colombia, Bogotá, 1971.
19. *Time*, Junio 17 (1985).
20. J. S. Vandergraft, *Introduction to Numerical Computations*, New York: Academic Press, 1978.