MATEMÁTICA NUMÉRICA

Fernando Vadillo (*)

Este artículo persigue ofrecer un primer contacto con el mundo de la Matemática Numérica a los profesores de Matemáticas en la enseñanza secundaria y a quien tengan alguna curiosidad por esta Ciencia y la Computación. Estudiar Matemáticas es un proceso donde se debe aprender a resolver problemas; puede haber Matemáticas sin teoremas mas no sin problemas. Se trata de aprender a hacer algo y no sólo de adquirir conocimientos. En ese hacer algo es donde cada vez tienen más presencia los ordenadores en las aulas que obligan a los profesores a tener algunos conocimientos básicos en esta materia.

La Matemática Numérica es la rama de las Matemáticas que propone, desarrolla, analiza y aplica algoritmos y métodos numéricos para obtener soluciones aproximadas de problemas matemáticos. Trefethen en [Th92] diserta sobre varias definiciones del Análisis Numérico. En su opinión la mejor es la que dice que: El Análisis Numérico es el estudio de algoritmos para los problemas de las matemáticas continuas, después los algoritmos son implementados en computadoras. Se trata por tanto de una disciplica matemática totalmente aplicada, Sanz-Serna en [SS98] dice: "El Cálculo Numérico además de una disciplina académica que se estudia en las aulas, es parte de un esfuerzo humano global desarrollado a lo largo de la historia para resolver problemas científicos y técnicos".

En este artículo se han evitado matices y comentarios que aunque enriquecerían la exposición la alargarían excesivamente y quiza distraerían al lector de lo que es su objetivo fundamental: obtener una idea de tipo *impresionista* de la Matemática Numérica. Incluso se ha optado en ocasiones por utilizar argumentos cualitativos y no cuantitativos que obligarían a un mayor equipamiento matemático. Sólo se le suponen al lector conocimientos básicos de matemáticas salvo quizá en algun ejemplo que el lector no avezado puede evitar.

La primera parte del artículo está dedicada al tipo de problemas matemáticos que se resuelven. A continuación se analizan los métodos numéricos que aproximan las soluciones en un contexto muy general, se introducen los conceptos de consistencia, estabilidad y convergencia, y se dedica una especial atención a los errores de redondeo. Finalmente se valora la eficiencia de los métodos numéricos y se desarrolla un ejemplo.

En la bibliografía del final se ha seleccionado aquello que se ha considerado que hacía alguna aportación interesante, evidentemente existen otros textos que se pueden consultar, aunque tampoco sean muy numerosos en lengua española. En este punto destaca el texto del profesor Sanz-Serna por su claridad y concreción.

Todos los experimentos se han realizado en *MATLAB 6* para lo que se recomienda las referecias: [HH00], [mat6] y [manual].

(*) Profesor de la Universidad del País Vasco. Euskal Herriko Unibertsitatea.

1. CONDICIONAMIENTO DE UN PROBLEMA

Consideremos el problema de hallar x tal que

$$F(x, d) = 0 \tag{1}$$

donde d es un conjunto de datos y F es la función que relaciona los datos con el resultado x. Dependiendo del problema x , d serán variables de distintos tipos y dimensiones.

Se dice que el problema (1) está **bien condicionado** o **bien puesto**, si la solución x es única y depende continuamente de los datos en el sentido de que sus pequeñas pertubaciones provocan cambios también pequeños en los resultados.

Ejemplo.
El polinomio $p(x) = x^4 - x^2(2a - 1) + a(a - 1)$ tiene las raíces $x = \pm\sqrt{a}$ y $x = \pm\sqrt{a-1}$. El número de raíces reales diferentes depende del valor de a, si $a \geq 1$ tiene cuatro raíces reales distintas, si $0 \geq a < 1$ tiene dos y no existen raices reales si $a < 0$, por tanto buscar las raices reales de $p(x)$ en función de a es un problema mal puesto.

Para cuantificar la calidad de un problema definimos su número de condición relativo de la forma siguiente:

$$K(d) = \sup_{\delta d \in D} \frac{|\delta x| / |x|}{|\delta d| / |d|} \tag{2}$$

donde D es un entorno del origen que indica el conjunto de perturbaciones δd sobre los datos y δx el cambio en la solución, es decir $F(x + \delta x;\ d + \delta d) = 0$. Si $K(d)$ es un número grande hablaremos de un problema mal puesto o mal condicionado, y cuando su tamaño sea moderado el problema podrá tener un tratamiento numéricamente.

Cuando el problema (1) tiene una única solución x para cada dato o conjunto de datos d, necesariamente existe una función G llamada **resolvente** tal que:

$$x = G(d) \quad \rightarrow \quad F(G(d), d) = 0 \tag{3}$$

Si se supone ahora que G es diferenciable:

$$\delta x = (x + \delta x) - x = G(d + \delta d) - G(d) \approx G'(d)\delta d \tag{4}$$

de donde podemos estimar que el número de condición relativo es:

$$K(d) \approx |G'(d)| \frac{|d|}{|G(d)|} \tag{5}$$

Ejemplo.
Las soluciones de la ecuación algebraica $x^2 - 2px + 1 = 0$ con $p \geq 1$ son $x\pm = p\pm\sqrt{p^2 - 1}$, es decir, las raíces de la ecuación $F(x , p) = x^2 - 2px + 1 = 0$ son $\{x_+ , x_-\}$ y la resolvente $G\pm(Fp) = p\pm\sqrt{p^2 - 1}$.

Ahora $G'_{\pm}(p) = 1 \pm \frac{p}{\sqrt{p^2-1}}$ y la estimación (5) resulta $K(p) \approx \frac{|p|}{\sqrt{p^2-1}}$.

Cuando $p \approx 1$, $\sqrt{p^2 - 1} \approx 0$ y $K(p)$ es muy grande por lo que el problema está mal condicionado.

Si ahora cambiamos a un nuevo parámetro $t = p + \sqrt{p^2 - 1}$ la función $F(x,t) = x^2 - \frac{1+t^2}{t}x + 1 = 0$ tiene las raices $\{x_- = t\ ,\ x_+ = 1/t\}$. La estimación (5) que resulta ahora es $K(t) \approx 1$ y el problema transformado está bien puesto. Aunque matemáticamente el problema es el mismo numéricamente son problemas muy diferentes.

Para resolver numéricamente un problema mal condicionado es imprescindible transformarlo. Es un error pensar que un método numérico pueda curar la patología de un problema mal puesto.

2. CONVERGENCIA DE MÉTODOS NUMÉRICOS

Suponiendo que el problema (1) está bien condicionado, *un método numérico* para calcular soluciones aproximadas, en general, consistirá en una sucesión de problemas aproximados:

$$F_n\,(x_n\,;\,d_n) = 0 \qquad \text{para} \qquad n \geq 1 \tag{6}$$

que dependen de un parámetro n diferente en cada situación. Ahora lo que se espera es que las soluciones numéricas x_n se vayan acercando a la solución exacta x, es decir, que haya una convergencia en algun sentido. Para demostrar esta convergencia se estudian los comportamientos de los **errores absolutos:**

$$E(x_n) = |\; x - x_n \;| \tag{7}$$

o también los **errores relativos** que son:

$$E_{rel}(x_n) = \frac{|\; x - x_n \;|}{|\; x \;|} \qquad \text{si.} \qquad x \neq 0 \tag{8}$$

una definición equivalente de error relativo es $E_{rel}\,(x_n) = |\;\rho\;|$ donde $x_n = x\,(1+\rho)$. La ventaja de los errores relativos es que son independientes de la escala, si $x \to \alpha x$ y $x_n \to \alpha x_n$ no cambia, mientras que los errores absolutos se multiplican por el factor de escala α.

El error relativo está relacionado con el concepto de **dígitos significativos correctos** en una aproximación, expresión muy utilizada pero muy poco precisa.

Se entiende por dígitos significativos de un número a sus primeros dígitos no cero, por ejemplo 1.7320 tiene cinco y 0.0491 tiene sólamente tres. Si se quisieran comparar sin más los dígitos de x y su aproximación x_n lo cosa siene su misterio, veamos los dos casos siguientes:

$$\begin{array}{lllll} (1) & x = 1{:}0000 & x_n = 1{:}00499; & E_{rel} = 4{:}99 \times 10^{-3} \\ (2) & x = 9{:}0000 & x_n = 8{:}99899; & E_{rel} = 1{:}12 \times 10^{-4} \end{array}$$

en el primer caso x y su aproximación tiene iguales los tres primeros dígitos mientras que el segundo no tiene ninguno a pesar de tener un error relativo menor. Parece razonable que algo se debe cambiar, quizá el significado de dígito correcto: una aproximación se dice que tienen p dígitos correctos, si al redondear a p dígitos, el número y su aproximación coinciden. Redondeo significa que un número se aproxima al número con p dígitos más próximo con alguna regla para cortar la cola cuando hay dos igualmente próximos.

Esta nueva versión resuelve el problema que se tenía porque en el primer ejemplo redondeado a cuatro dígitos da $x = x_n = 1.000$ y en el segundo $x = x_n = 9.000$. Pero tomado ahora $x = 0.9949$ y $x_n = 0.9951$ resulta que la aproximación tiene tres dígitos correctos pero no dos.

Pueden intentarse otras definiciones pero siempre se la puede encontrar la vuelta, y aunque se hable de número de dígitos correctos en términos coloquiales, siempre resulta poco preciso y es preferible hablar de errores.

Para estimar los errores, o bien se conoce la solución exacta x, en cuyo caso poco interés puede tener buscar soluciones aproximadas, o quizá se puedan acotar los errores por cantidades que van a cero cuando $n \to \infty$, si bien este segundo camino, en general, resulta muy complicado y poco práctico. Frecuentemente la convergencia de un método numérico se demuestra a través de dos nuevos conceptos que son la consistencia y la estabilidad que se comentan a continuación.

El método (6) se dice que es **consistente** con el problema (1) si:

$$F_n\,(x,d) \to 0 \qquad \text{cuando} \qquad n \to \infty\,,$$

es decir, se comprueba en qué medida los datos y la solución exacta verifican la ecuación aproximada (6), y si el error que cometen tiende a cero cuando $n \to \infty$ tenemos la consistencia. Si se hiciera al revés, es decir, se calculara $F(x_n, d_n) \neq 0$, la exactitud con la que las soluciones aproximadas verifican la ecuación exacta (1) se obtendrían los *residuos* que es otra forma de calibrar la calidad una aproximación.

En algunos casos, por ejemplo en los métodos iterativos, la ecuación (6) puede tener otra forma

$$F_n(x_n, x_{n-1}, \ldots x_{n-q}, d_n) = 0 \tag{9}$$

donde $x_0, x_1, \ldots, x_{q-1}$ son datos. En este caso la consistencia medirá $F_n(x, x, \ldots, x, d)$ para $n \geq q$

Ejemplo.

El método de Newton es habitual para aproximar una raiz simple α de una función $f : \mathbb{R} \to \mathbb{R}$ y consiste en la iteración:

$$\begin{cases} x_0 \quad \text{dado;} \\ x_n = x_{n-1} - \dfrac{f(x_{n-1})}{f'(x_{n-1})}, \qquad n \geq 1, \end{cases} \tag{10}$$

Ahora $F_n(x_n, x_{n-1}, f) = x_n - x_{n-1} + \dfrac{f(x_{n-1})}{f'(x_{n-1})}$, $F(a, a, f) = 0$ y el método es siempre consistente.

Por otra parte, para que un método numérico (6) sea útil, para cada dato o conjunto de datos d_n deberá existir un única solución x_n, por tanto debe existir otra resolvente G_n tal que:

$$x_n = G_n(d_n) \tag{11}$$

con $F_n(G_n(d_n), d_n) = 0$ para todo n. Además las soluciones aproximadas x_n deberán depender continuamente de los datos d_n en el sentido que comentamos en el párrafo anterior: pequeñas perturbaciones de éstos deben provocar leves cambios en los resultados. Los métodos numéricos con este comportamiento se dice que son estables.

Para cuantificar la **estabilidad** de un método se define su **número de condición** de manera similar a como ya se hizo para los problemas:

$$K_n(d_n) = \sup_{d_n \in D_n} \frac{|\delta x_n| / |x_n|}{|\delta d_n| / |d_n|}; \tag{12}$$

y cuando esta cantidad sea grande el método es inestable y por tanto de dudosa utilidad práctica. Si además la resolvente G_n es diferenciable, repitiendo los cálculos ya explicitados en el párrafo anterior se consigue una aproximación del número de condición que es:

$$K_n(d_n) = |G_n'(d_n)| \frac{|d_n|}{|G_n(d_n)|}. \tag{13}$$

Ejemplo.

El problema de las cancelaciones. La función $f : \mathbb{R}^2 \to \mathbb{R}$, $f(a, b) = a + b$ es lineal y su gradiente (derivada) es $(1,1)^T$. Usando la norma $\| \cdot \|_1$ se calcula fácilmente su número de condición $K(a, b) = \dfrac{|a|+|b|}{|a+b|}$. Por tanto, sumar dos números de igual signo es estable porque

$$K(a,b) \approx 1,$$

pero al restar dos números casi iguales $|a + b| << |a| + |b|$ el número de condición es muy grande y resulta una operación inestable.

Los conceptos de estabilidad y convergencia están fuertemente conectados. En primer lugar, siempre que el problema esté bien condicionado, la estabilidad de un método numérico es necesaria para la convergencia, es decir, no existen métodos convergentes que sean inestables.

Si se supone que el método (6) es convergente, y si $x_n(d)$ denota la solución numérica con el dato d:

$$\begin{aligned} |\delta x_n| &= |x_n(d) - x_n(d+\delta d_n)| \\ &\leq |x_n(d) - x(d)| + |x(d) - x(d+\delta d_n)| + |x(d+\delta d_n) - x_n(d+\delta d_n)| \end{aligned} \tag{14}$$

de estos tres sumandos, el primero y tercero son pequeños por la convergencia y el segundo está acotado porque hemos supuesto que el problema está bien condicionado.

Sin embargo nuestro objetivo no era el resultado anterior, si no mas bien el contrario, es decir, demostrar un resultado de convergencia a través de la estabilidad. El resultado que habitualmente se denomina **teorema de convergencia** dice que si un método es consistente, la estabilidad es suficiente para la convergencia, dicho que otra manera, **consistencia más estabilidad implica convergencia**, porque:

$$\begin{aligned} |E(x_n)| &= |x(d+\delta d_n) - x_n(d+\delta d_n)| \\ &\leq |x(d+\delta d_n) - x(d)| + |x(d) - x_n(d)| + |x_n(d) - x_n(d+\delta d_n)|, \end{aligned} \tag{15}$$

y de estos tres sumandos, el primero está acotado porque el problema está bien puesto, el segundo por la consistencia y el tercero por la estabilidad. En conclusión: para demostrar la convergencia de un método es suficiente con comprobar su consistecia con el problema y su estabilidad numérica.

3. ANÁLISIS A PRIORI Y A POSTERIORI DE LOS ERRORES

Para analizar la estabilidad de un método númerico se pueden usan dos estrategias:

1. **Análisis progresivo:** Se buscan cotas de los errores $|\delta x_n|$ en función de las perturbaciones de los datos y los errores intrínsecos al método.
2. **Análisis regresivo:** Se estiman las perturbaciones sobre los datos que darían la solución obtenida suponiendo que la aritmética es exacta, es decir, se busca δd_n tal que $F_n(\hat{x}_n, d_n + \delta d_n) = 0$ donde $\hat{x}_n$ es la solución computada.

Ejemplo.

Suponiendo que $\alpha_1, \ldots\ldots, \alpha_n$ son las raices del polinomio $p(x) = \sum_{k=0}^{n} a_k x^k$ y $\hat{\alpha}_1, \ldots, \hat{\alpha}_n$ son las raices del polinomio perturbado $p(x) = \sum_{k=0}^{n} (a_k + \delta a_k) x^k$ Un análisis progresivo estimará los errores $\alpha_i - \hat{\alpha}_i$ en función de los δa_k, mientras que un análisis regresivo estimará δa_k tales que

$$\sum_{k=0}^{n} (a_k + \delta a_k) \hat{\alpha}_i^k = 0.$$

Los análisis progresivos y regresivos de los errores son ambos **análisis a priori**, los **análisis a posteriori** evaluan los errores $x - x_n$ en función de los residuos $r_n = F(x_n, d)$. En el ejemplo anterior un análisis a posteriori estimaría $\alpha_i - \hat{\alpha}_i$ en función de $p(\hat{\alpha}_i)$.

Los análisis a posteriori se utilizan para controlar los errores, cambiando el parámetro de discretización se puede obligar a que los errores sean menores que una tolerancia. Los métodos con estrategias de este tipo se denominan **métodos adaptativos** de los que recomendamos la referencia [EEHJ96].

4. FUENTES DE ERROR EN LA COMPUTACIÓN

El método numérico (6) es una aproximación del problema matemático (1) que a su vez, es un modelo de un problema físico; por tanto, las soluciones aproximadas $\hat{x}_n$ están *contaminadas* de varias fuentes de error que se esquematizan a continuación:

Problema físico $\overset{(1)(2)}{\longrightarrow}$ Modelo matemático $\overset{(3)}{\longrightarrow}$
Método numérico $\overset{(4)}{\longrightarrow}$ Solución computada

1. Errores debido al modelo que deben ser controlados por una adecuada construcción de este.
2. Errores en los datos, que se pueden mejorar con mediciones más precisas
3. **Errores de discretización** que es el resultado de cambiar la ecuación (1) por la (6).
4. **Errores de redondeo** cometidos en la operaciones de la ecuación (6).

Los errores 3 y 4 son los **errores de computación**, el error 3 lo controla la convergencia del método, un método es convergente cuando los errores de discretización se hacen tan pequeños como se quiera con una elección adecuada del parámetro de discretización.

Los errores de redondeo son debidos a la computación finita, en un ordenador sólo se pueden usar un número finito de cifras. El efecto del redondeo sobre los algoritmos numéricos puede ser dramático como se puede apreciar en el ejemplo siguiente: tome un ordenador, un número cualquiera $x > 0$ y haga la siguiente sucesión de operaciones:

para $i = 1 : 60$
$x = \sqrt{x}$
final
para $i = 1 : 60$
$x = x^2$
final

esto es: calcule primero 60 raices cuadradas y después otros 60 potencias cuadradas, observará que el ordenador sorprendentemente no devuelve el valor correcto que es x, el resultado es 0 cuando $0 < x < 1$ y 1 para $1 \leq x$. El motivo fundamental de este comportamiento es que estos cálculos superan la capacidad de casi cualquier máquina, en [Hi96, pp.18] puede encontrar los detalles.

5. REPRESENTACIÓN DE LOS NÚMEROS EN UN ORDENADOR

Los errores de redondeo se deben a que en un ordenador sólo se pueden representar un número finito de cifras, asi pues, para poder operar con números de infinitas cifras diferentes (cualquier número irracional) se deben aproximar dichos números por otro con finitas cifras y por tanto se está condenado inevitablemente a cometer errores. Lo importante en este punto es conocer como se propagan estos errores.

Tomemos una base de numeración $\beta \geq 2$, por supuesto que β es un número natural, y sea x un número real con un número finito de dígitos, escrito x en la base β con la habitual notación posicional:

$$x = (-1)^s \cdot x_n x_{n-1} \ldots x_0 . x_{-1} \ldots x_{-m} = (-1)^s \sum_{k=-m}^{n} x_k \beta^k, \tag{16}$$

donde $0 \leq x_k < \beta$ para $k = -m, \ldots, n$ y s depende el signo de x, $s = 0$ para los números positivos y $s = 1$ para los negativos.

Aunque teóricamente todas las bases son equivalentes en computación se usan tres:

1. Binaria ($\beta = 2$). Los dígitos se reducen a los símbolos 0 y 1 llamados **bits** (binary digits).
2. Decimal ($\beta = 10$).
3. Hexadecimal ($\beta = 16$) con los dígitos 0,1,2,..,9,A,B,C,D,E y F.

Suponiendo que un ordenador tiene N posiciones de memoria para almacenar cualquier número, lo más natural sería fijar una posición para guardar el signo, $N - k - 1$ posiciones para la parte entera y las k restantes para la parte decimal. Esta representación denominada **sistema de punto fijo** porque fija el número de dígitos a la izquierda y la derecha de la coma, tiene una fuerte limitación sobre las cantidades que se pueden representar, salvo que N sea muy grande en cuyo caso se necesitaría una memoria enorme. Para evitar este grave inconveniente se utiliza la representación en **punto flotante** en donde el número x se escribe en notación exponencial de la forma:

$$x = (-1)^s \cdot (0.a_1 \cdots a_t) \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t}, \tag{17}$$

donde t es el número de dígitos significativos, $m = a_1 \cdots a_t$ es la **mantisa** con $a_1 \neq 0$ y e es el **exponente** que estará en un intervalo finito $L \leq e \leq U$. El **conjunto de números de punto flotante** es:

$$F(\beta, t, L, U) = \{0\} \bigcup \left\{ x \in \mathbb{R} : x = (-1)^s \beta^e \sum_{i=1}^{t} a_i \beta^{-i} \right\} \tag{18}$$

Cuando se utilizan representaciones en punto flotante las N posiciones de memoria se distribuyen de manera diferente: una posición para el signo, t posiciones para la mantisa y las $N - t - 1$ posiciones restantes para el exponente. Las dos formas más utilizadas son las denominadas **precisión simple y doble** que en caso binario se distribuyen de la siguiente forma:

precisión simple:	$N = 32$	1 bit para s	8 bits para e	23 bits para m
precisión doble:	$N = 64$	1 bit para s	11 bits para e	52 bits para m

La posibilidad de construir conjuntos de números flotantes con diferentes bases, número de digitos significativos en la mantisa y distintos rangos en el exponente, llevó al IEEE (Institute of Electrical and Electronics Engineers) a desarrollar en 1985 un sistema estándar que fue aprobado en 1989 por la IEC (International Electronical Commision) que ahora se conoce con el nombre de aritmética IEC/IEEE:

en precisión simple:	F(2,24,-125,128)
en precisión doble:	F(2,53,-1021,1028)

el 24 = 23 + 1 y el 53 = 52 + 1 se deben a que el primer dígito de la mantisa es siempre 1 y es innecesario almacenarlo. Además incorporan representaciones para ± 0 , $\pm\infty$ y el NaN (not a number) que representa por ejemplo el 0/0, $\infty - \infty$, $0 \cdot \infty$ $\cdots$ que son:

valor	exponente	mantisa
± 0	L-1	0
$\pm\infty$	U+1	0
NaN	U+1	$\neq 0$

Como el conjunto de números de punto flotante $F(\beta, t, L, U)$ es un subconjunto del conjunto de números reales, se debe representar en F cualquier número con el que se quiera operar, incluso si dos números x e y están en F el resultado de una operación entre ellos no necesariamente también lo estará, es decir, se necesita alguna forma de aproximar en F aquellos números que no entra en su rango de representación. Para cualquier $x \in \mathbb{R}$ se elige una representación flotante en F que se suele denotar por $fl(x)$. Lo habitual es hacer un redondeo de la siguiente forma:

$$fl(x) = (-1)^s \cdot (0.a_1 \cdots a_t) \cdot \beta^e \quad \text{donde} \quad a_t = \begin{cases} a_t & \text{si} \quad a_{t+1} < \beta/2 \\ a_t + 1 & \text{si} \quad a_{t+1} \geq \beta/2 \end{cases}$$

Esta representación será válida cuando el $L \leq e \leq U$ y si el resultado de alguna operación obtiene un exponente menor que L se obtiene un **underflow** y cuando $e > U$ un **overflow** que habitualmente interrumpe la ejecución del programa. Evidentemente, si $x \in F$ entonces $fl(x) = x$.

Los errores de redondeo están acotados porque se demuestra que ([Hi96], theorem 2.2):

$$fl(x) = x(1+\delta) \qquad \text{con} \qquad |\delta| \leq u = \frac{1}{2}\beta^{1-t}.$$

Esto indica que el error relativo es menor que el valor u llamado **precisión de la maquina**; en precisión simple $u = 2^{-24} \approx 5.96 \times 10^{-8}$ y en la doble $u = 2^{-53} \approx 1.11 \times 10^{-16}$.

Cuando se opera en punto flotante entre dos números x e y primero se deben redondear los números y después el resultado, por tanto:

$$x \odot y = fl(fl(x) \cdot fl(y)) \tag{21}$$

donde $\cdot$ ahora indica una suma, resta, multiplicación o división, y $\odot$ denota la correspondiente operación en la aritmética de punto flotante. En esta nueva aritmética algunas de las propiedades de la aritmética clásica se conservan, como por ejemplo la conmutatividad, pero otras se pierden, por ejemplo la asociatividad de la suma:

$$x \oplus (y \oplus z) \neq (x \oplus y) \oplus z,$$

lo que significa que el orden de sumación es relevante.

Es bién conocido que $\sum_{k=1}^{\infty} k^{-2} = \pi^2/6 = 1.64493306684873$, pero imagine que desconoce esta identidad y quiere aproximar esta suma. En MATLAB la suma de un millón de términos de mayor a menor es 1.64493306684877 y sumados al revés da 1.64493306684873, la diferencia es apenas apreciable porque se opera en doble precisión. Con precisiones menores en algún momento se obtiene mejores aproximaciones sumando de menor a mayor que al revés, el número de términos que debe tomar para apreciar la diferencia dependerá de la precisión que utilice. A la hora de sumar muchos términos siempre se aconseja que se haga de menor a mayor [Hi96, capítulo 4].

No es frecuente que los errores de redondeo se compensen unos con otros para dar mejores resultados finales que intermedios, pero es posible. Considere la función:

$$f(x) = \frac{e^x - 1}{x} = \sum_{i=0}^{\infty} \frac{x^i}{(i+1)!} \tag{23}$$

si usa dos algoritmos diferentes aunque equivalentes:

algoritmo 1		algoritmo 2	
si	$x = 0$	$y = e^x$	
	$f = 1$	si	$y = 1$
si no			$f = 1$
	$f = \frac{e^x - 1}{x}$	si no	
final			$f = \frac{y-1}{\log y}$
		final	

MATLAB obtiene los siguientes resultados:

x	algoritmo1	algoritmo 2
10^{-5}	1.00000500000696	1.00000500001667
10^{-8}	0.99999999392253	1.00000000500000
10^{-10}	1.00000008274037	1.00000000005000
10^{-12}	1.00008890058234	1.00000000000050
10^{-14}	0.99920072216264	1.00000000000001
10^{-15}	1.11022302462516	1.00000000000000
10^{-16}	0	1

en [Hi96 pg 22] se demuestra que el algoritmo 2 tiene mejores resultados porque hay una cancelación de errores en la división que no ocurre en el algoritmo 1.

Los errores de redondeo pueden incluso beneficiar el comportamiento de algunos algoritmos. El método de la potencia se utiliza para calcular autovectores asociados a autovalores dominantes de matrices, como se trata de un método iterativo, para comenzar debe tomarse un vector arbitrario. Si el vector inicial no tiene componente en la dirección del vector buscado, teóricamente el método no converge, pero los errores de redondeo hacen la componente, aunque pequeña, diferente de cero y esto corrigen la divergencia del método.

6. COSTO OPERATIVO Y EFICIENCIA

Habitualmente un mismo problema puede resolverse usando varios métodos numéricos. La pregunta ahora es con cuál de ellos quedarse. El tamaño del error es muy importante, se podría elegir el método con menor error. Sin embargo, existe un criterio mejor: *nos quedaremos con el método más eficiente en el sentido de que dando un error menor que una tolera-cia predeterminada, necesite el menor trabajo.* La eficiencia depende del tamaño de los errores y del *costo operativo* del método que es el número de operaciones necesaria para llegar a la solución. La eficiencia o no de un método numérico es un rasgo muy importante que siempre se debe analizar.

Ejemplo.
Algoritmo de Horner. Suponga que se debe evaluar para un valor dado z un polinomio cuártico $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$. El método ingenuo calcularía los sucesivos productos $z^2 = z \cdot z$, $z^3 = z^2 \cdot z$, $z^4 = z^3 \cdot z$ que son tres multiplicaciones, despues calcularía los productos por los coeficientes que son otras cuatro multiplicaciones, y finalmente las cuatro sumas o restas dependiendo del signo del coeficiente. En total son cuatro sumas o restas y siete multiplicaciones.

Si ahora opera en otro orden:

$$a_0 + z \cdot (a_1 + z \cdot (a_2 + z \cdot (a_3 + z \cdot a_4))) \tag{24}$$

el resultado es el mismo pero sólo precisa cuatro sumas o restas y cuatro multiplicaciones, se ha reducido de siete a cuatro el número de multiplicaciones.

Para un polinomio de grado N el método ingenuo requiere N sumas y 2N-1 multiplicaciones mientras que el método (24) llamado de Horner o de multiplicaciones encajadas necesita N sumas y N multiplicaciones. Como ninguno de los dos métodos genera errores salvo los de redondeo, el de Horner es más eficiente. Si como es frecuente en un programa se deben evaluar polinomios en miles de puntos diferentes, usar un método u otro puede suponer mucho tiempo de computación.

7. UN PROBLEMA DE DEPREDADOR-PRESA

El Análisis Numérico ha desarrollado un número muy importante de algoritmos para resolver una gran variedad de problemas:

Resolución de sistemas de ecuaciones lineales.
Solución de ecuaciones y sistemas no lineales.
Aproximación de funciones.
Diferenciación e integración de funciones.
Problemas de optimización.
Problemas de mínimos cuadrados.
Resolución de ecuaciones diferenciales.
.....

Quizá la novedad más sobresaliente de los últimos años haya sido la aparición de un nuevo lenguaje de programación *MATLAB* que permite implementar los algoritmos que ya se conocían de una manera mucho más eficaz y rápida.

MATLAB (MAThematical LABoratory) es un lenguaje para la computación científica cuyos datos fundamentales son vectores y matrices. Se distinge de otros lenguajes como Fortran o C porque opera en un nivel muy alto incluyendo cientos de operaciones en un sólo comando. Desde las primeras versiones de Cleve Moler, a finales de los años setenta, *MATLAB* ha llegado a ser una poderosa herramienta para investigar y resolver problemas prácticos que ha sido introducido en los programas de estudio de las escuelas científicas y técnicas de las más prestigiosas universidades de los paises avanzados. Personalidades de la autoridad de L.N.Trefethen afirman en el prefacio de su libro [Tr00] que con el *MATLAB* ha nacido una nueva era en la computación científica, porque hace una cantidad astronómica de cálculos con muy pocas lineas de programa y en muy pocos segundos de tiempo. Charles F. Van Loan otra autoridad indiscutible en la Computación Matricial, en [Vl00] dice que *MATLAB* ha cambiado la forma de computar e investigr en las Ciencias de la Computación.

Para que el lector llege a formarse una idea de la capacidad de resolución del Análisis Numérico con *MATLAB* se considera el siguiente ejemplo: se supone que un conejo sigue un camino prefijado ($r_1(t)$, $r_2(t)$) y que un zorro le está persiguiendo con dos hipótesis:

- En cada instante la tangente del camino del zorro apunta al conejo.
- La velocidad del zorro es k veces la del conejo.

Sea ($y_1(t)$, $y_2(t)$) la ruta que sigue el zorro, según las hipótesis anteriores deben ser la solución del sistema de ecuaciones diferenciales:

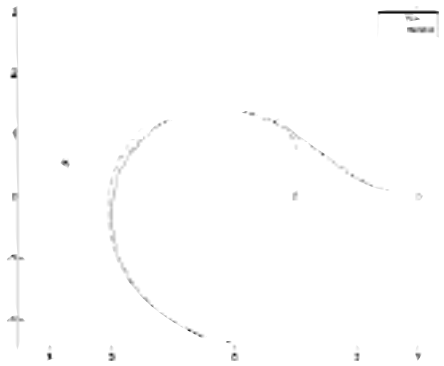
$$\begin{aligned} \frac{d}{dt}y_1(t) &= s(t)(r_1(t) - y_1(t)), \\ \frac{d}{dt}y_2(t) &= s(t)(r_2(t) - y_2(t)), \end{aligned}$$

donde:

$$s(t) = \frac{k\sqrt{(\frac{d}{dt}r_1(t))^2 + (\frac{d}{dt}r_2(t))^2}}{\sqrt{(r_1(t) - y_1(t))^2 + (r_2(t) - y_2(t))^2}}$$

Cuando el zorro es más rápido que el conejo ($k > 1$), el zorro debe alcanzar a la presa, de forma que el denominador de s(t) se hace muy pequeño y no puede completarse la integración numérica.

Para estos casos 'ode45' permite introducir una opcion llamada 'Event' de forma que si durante la ejecución del programa se da una situación concreta se detiene la ejecución y se devuelven los resultados en ese momento. Por ejemplo el evento pudiera definirse diciendo que la distancia del zorro al conejo es menor que 10^{-4}. El programa en *MATLAB* para resolver y dibujar las trayectorias está detallado en [HH00] y apenas ocupa veinte lineas. Las trayectorias de los dos animales con la caza que tiene lugar en el tiempo $t = 5{:}071$ y en el punto $(.8646, -2.3073)$ es la que sigue:



8. BIBLIOGRAFÍA

QSS00 – **A. Quarteroni, R. Saccco y F. Saleri:** "Numerical Mathematics". *Springer*, 2000.

EEHJ96 – **K. Erikson, D. Estep, P. Hansbo y C. Johnson:** "Computational Differential Equations". *Cambridge University Press*, 1996.

Fa99 – **L. V. Fausett:** "Applied Numerical Analysis Using MATLAB". *Prentice-Hall*, 1999.

Hi96 – **N. J. Higham:** "Accuracy and Stability of Numerical Algoritms". Society for Industrial and Applied Mathematics, *Philadelphia*, 2000.

HH00 – **D. J. Higham y N.J. Higham:** "MATLAB Guide". Society for Industrial and Applied Mathematics, *Philadelphia*, 2000.

mat6 – "MATLAB. The Language of Technical Computing". Disponible en http://www.mathworks.com.

MF99 – **J. H. Mathews y K.D. Fink**: "Métodos Numéricos con MATLAB". Tercera Edición. *Prentice- Hall*, 1999.

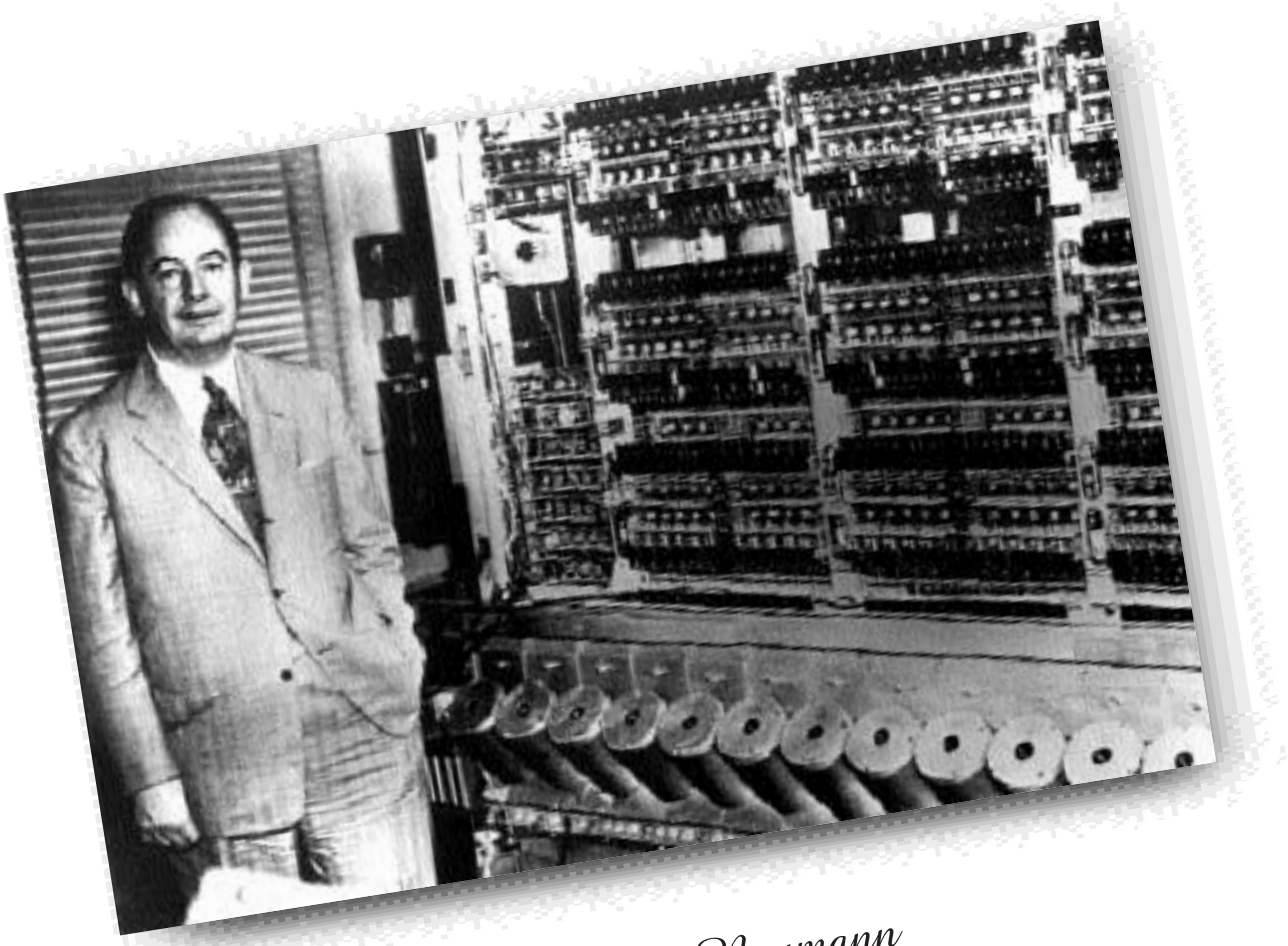
manual – "The Student Edition of MATLAB". *The Mathworks, Inc.* Published by *Prentice-Hall*, 1997.

SS98 – **J. M. Sanz-Serna:** "Diez lecciones de Cálculo Numérico". *Universidad de Valladolid*, Valladolid, 1998.

Th92 – **L. N. Trefeten:** "The Definition of Numerical Analysis". Disponible en http://www.comlab.ax. ac.uk/oucl/work/nick.trefethen.

Th00 – **L. N. Trefethen:** "Spectral Methods in MATLAB". *Society for Industrial and Applied Mathematics*, Philadelphia, 2000.

Vl00 – **C. F. Van Loan**: "Introduction to Scientific Computing. A Matrix-Vector Approach Using MATLAB". 2nd Edition *Prentice-Hall*, 2000.



John von Neumann
(1903-1957)