Dr. M. Palacios
Primavera 1999-2001

1.I Introducción al Cálculo Numérico.

1 El Análisis Numérico.

El **Análisis Numérico** es la parte de las Matemáticas que se encarga de diseñar métodos para aproximar de forma eficiente las soluciones de problemas expresados matemáticamente; su objetivo no es sólo garantizar la existencia de solución de un problema concreto, sino también obtenerla mediante algún proceso constructivo.

Un **método constructivo** es un conjunto de instrucciones conducentes a calcular la solución de un problema, bien en un número finito de pasos o bien en un número infinito mediante un proceso de paso al límite.

Sin embargo, lo correcto sería que el método constructivo proporcionara la solución del problema en una forma eficiente. Para enterder esto, piénsese en el cálculo de un determinante de orden n definido como suma de $n!$ sumandos cada uno de los cuales es producto de un elemento de cada fila y columna. Este cálculo requiere $n! - 1$ sumas y $n - 1$ multiplicaciones. En un ordenador que hiciera cada operación aritmética elemental en $15 * 10^{-6}$ segundos se tendría el siguiente cuadro:

n	$num.operaciones$	$tiempo$
5	480	$7 * 10^{-3}\, segundos$
10	$3 * 10^{7}$	$8\, minutos$
20	$4.5 * 10^{19}$	$21 * 10^{6}\, años$

Por lo tanto, exigiremos que los métodos constructivos puedan ser implementados de manera razonable en el ordenador lo que definirá un **método numérico**.

La forma práctica en que un método es introducido en el ordenador es un algoritmo. Más concretamente, un **algoritmo** es una secuencia de operaciones aritméticas y lógicas que produce la aproximación a la solución de un problema con una tolerancia o precisión prefijada en un número finito de pasos.

Para medir la aproximación mencionada es usual utilizar las notaciones $\mathcal{O}(n)$ y $o(n)$ siguientes. Dadas dos sucesiones reales $\{x_n\}$ y $\{z_n\}$ se dice que $x_n = \mathcal{O}(z_n)$ ("$\mathcal{O}$ grande"), si existe una constante positiva C tal que $||x_n|| \leq C||z_n||$. Por el contrario, se dice que $x_n = o(z_n)$ ("o pequeña"), si $\lim ||x_n||/||z_n|| = 0$.

Por ejemplo, en el primer caso, si $\{z_n\}$ es sucesión convergente a 0, también $\{x_n\}$ lo será; en el segundo caso, si $\{x_n\}$ es un infinitésimo, $\{z_n\}$ también lo es pero de orden menor, y si $\{x_n\}$ es un infinito $\{z_n\}$ también pero orden mayor.

1.1 Convergencia

Una cuestión esencial es establecer la convergencia de la solución producida por el algoritmo (solución numérica) a la solución exacta del problema, y también analizar la velocidad de convergencia a dicha solución.

Veamos un ejemplo para entender estos dos conceptos. Supóngase que se desea encontrar una raíz de la ecuación $x^2 - a = 0$ mediante el método de Newton-Raphson:

$$\begin{aligned} &dado \quad x_0 \\ &x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})} \end{aligned}$$

Mediante este algoritmo, a partir del valor inicial x_0, se construye la sucesión x_n. Aplicando la definición, esta sucesión será convergente si se puede elegir un n_0 de forma que x_n sea tan cercano a $\sqrt{a}$ como se quiera para cualquier $n > n_0$. Esa cercanía se puede escribir en la forma:

$$\frac{x_n}{\sqrt{a}} = 1 + \epsilon_n$$

y el problema consiste en probar que al crecer n disminuye ϵ_n. En efecto, a partir del algoritmo de Newton-Raphson:

$$x_n = \frac{1}{2}\,(x_{n-1} + \frac{a}{x_{n-1}})$$

y sustituyendo resulta:

$$\epsilon_n = \frac{1}{2}\,\frac{\epsilon_{n-1}^2}{1 + \epsilon_{n-1}}$$

es decir,

$$\epsilon_n < \frac{1}{2}\,\epsilon_{n-1}$$

Por lo tanto, el error, ϵ_n, disminuye al aumentar n

Observemos que

$$\frac{|x_n - \sqrt{a}|}{|x_{n-1} - \sqrt{a}|^2} < cte$$

lo que da una idea de la velocidad de convergencia hacia la solución.

1.2 Estabilidad

Otra propiedad que debiera poseer un algoritmo es la **estabilidad**, que en palabras llanas significa que pequeños cambios en los datos iniciales producen pequeñas variaciones en los resultados finales. Veamos un ejemplo que aclara este concepto.

Se trata de calcular la integral siguiente:

$$I_{10} = \int_0^1 \frac{x^{10}}{a + x}\,dx$$

pero teniendo en cuenta que:

$$I_n = \int_0^1 \frac{x^{n-1}\,(x+a-a)}{a+x}\,dx = \frac{1}{n} - aI_{n-1}$$

se puede calcular mediante el algoritmo:

$$I_0 = ln\frac{1+a}{a}$$
$$I_n = \frac{1}{n} - a\,I_{n-1}$$

Por ejemplo, si $a = 10$, $I_0 = 0.095310180....$ Sin embargo, este número no se puede introducir en el ordenador de manera exacta, es preciso cometer algún error ϵ_0, con lo que se tiene:

$$\bar{I}_0 = I_0 + \epsilon_0 \Longrightarrow \bar{I}_n = \frac{1}{n} - a\,\bar{I}_{n-1}$$

y si el error en la n-ésima aproximación es: $e_n = \bar{I}_n - I_n$, resulta:

$$e_n = -a\,e_{n-1} = (-a)^n\,e_0 = (-a)^n\epsilon_0 = (-10)^n\,\epsilon_0$$

Como se observa, en cada iteración el error inicial se multiplica por 10, de modo que el valor calculado, $\bar{I}_{10}$, si se han tomado 9 cifras decimales para $\bar{I}_0$, no tendrá ninguna cifra decimal correcta.

Este es un ejemplo claro de inestabilidad numérica.

1.3 Fuentes de error

Es evidente que para poder asegurar la precisión y la fiabilidad de los resultados numéricos es preciso analizar el origen de los errores que aparecen.

Aparte de los errores debidos a las simplificaciones inherentes al modelo matemático del problema físico, están los errores del método numérico que deben ser tenidos en cuenta, estos son:

- errores de iniciación
- errores de representación de los números
- errores de discretización

Los primeros son debidos tanto a la representación de los números, como a que en muchas ocasiones son obtenidos experimentalmente; los últimos son debidos a las simplificaciones que requiere el problema para ser implementado en un ordenador; en general, no se obtendrá la solución exacta, sino una aproximación.

2 Representación de los números en el ordenador

Para introducir los números en un ordenador *digital* se considera su representación en alguna base adecuada, la más habitual es la base 2; las cifras (dígitos) en base 2 se denominan **bits** ("binary digits").

Para representar los **números enteros** en la denominada *simple precisión* se utilizan dos *bytes* o palabras, es decir, 2 bloques de 8 bits; para las *doble precisión* se utilizan 4 bytes; en ambos casos el primer bit se reserva para el signo. En consecuencia, los números enteros más grande y más pequeño representables en simple precisión serán:

$$x_{Max} = (0111\,1111\,1111\,1111)_2 =$$
$$1 + 2 + 2^2 + \ldots + 2^{14} = 2^{15} - 1 = (32767)_{10}$$
$$x_{min} = (1111\,1111\,1111\,1111)_2 = -32767 = 2^{16} - 1$$

Debido a las propiedades de las potencias de 2, en algunos ordenadores los números negativos se representan como el **complemento a 2**, es decir,

$$-x \to x + 2 - 2^{16}$$

Cuando se utiliza esta opción, primero se determina el valor decimal como si los 16 bits expresaran un número positivo; si este es menor que $2^{15} = 32768$, se le interpreta como positivo; caso contrario, se transforma en un número negativo restándole 2^{16}. Por ejemplo,

$$-x_{Max} \to 2^{15} - 1 + 2 - 2^{16} = 32768 + 1 - 65536 = -32767 = x_{min}$$

La representación binaria del complemento a dos de un número entero se obtiene fácilmente, basta restar la representación binaria de su valor absoluto de 2^{16}. Por ejemplo, la representación binaria del número 12 es

$$0000\,0000\,0000\,1100$$

por lo que la representación binaria del número -12 será

$$\begin{array}{r} 1\,0000\,0000\,0000\,0000 \\ -\,0000\,0000\,0000\,1100 \\ \hline 1111\,1111\,1111\,0100 \end{array}$$

Esta representación de los números enteros negativos simplifica y agiliza la adición y sustracción de enteros. Por ejemplo, 12 - 21 es construido en la forma siguiente:

$$\begin{array}{r} 0000\,0000\,0000\,1100 \\ -0000\,0000\,0001\,0101 \\ \hline 1111\,1111\,1111\,0111 \end{array}$$

que evidentemente es la representación binaria de -9 en complemento a 2, es decir,

$$\begin{array}{r} 1\,0000\,0000\,0000\,0000 \\ -\,0000\,0000\,0000\,1001 \\ \hline 1111\,1111\,1111\,0111 \end{array}$$

Los **números reales** se pueden representar de dos formas: **punto fijo** y **punto flotante**.

En punto fijo se considera un número fijo de decimales; es decir, si los números se representan mediante n dígitos, de ellos se toma $d < n$ para las cifras decimales. Por ejemplo, si se considera la base decimal y $n = 6$ y $d = 5$, el mayor número que podrá representarse es 9.99999 y el menor será 0.00001.

La representación en punto flotante se corresponde con la notación científica normalizada, por ejemplo,

$$732.5051 = 0.7325051 * 10^3$$

Representación en punto flotante significa que el número de dígitos usado para representar el número es fijo, así, si se utilizan 6 dígitos, 2 de los cuales se reservan para el exponente, el número anterior se puede escribir como

$$0.7325 * 10^{03}$$

En contraposición a la representación anterior, ahora, el número mayor respresentable en punto flotante será el $0.9999 * 10^{99}$, mientras que el número menor representable en punto flotante será el $0.0001 * 10^{-99}$.

Como se observa, se ha perdido precisión en la representación en punto flotante, pero se ha ganado en el espectro de números representables.

En una base de numeración cualquiera, β, un número en punto flotante se representará en la forma:

$$\pm\, q\, \beta^e$$

Esta representación se suele considerar normalizada en el sentido de que

$$\beta^{-1} \leq q < 1$$

Al número q se le denomina **mantisa o fracción** y al e **característica o exponente**.

Las bases de numeración más frecuentemente utilizadas son: $\beta = 2$ (binaria), $\beta = 8$ (octal) y $\beta = 16$ (hexadecimal), aunque los números son mostrados al usuario en base decimal.

La mayoría de las máquinas usan la base binaria. Para almacenar un número en esta forma, se requiere guardar el signo, la mantisa y la característica, y para ello se reserva en la memoria una **palabra**. Según el estadar del Institute of Electrical and Electronic Engineers (IEEE), en simple precisión, una palabra está constituida por 4 bytes, es decir, 32 bits, de los cuales el bit 1 se reserva para el signo de la mantisa, los 8 siguientes para el exponente y los 23 restantes para la mantisa. Como el primer bit de la mantisa q debe ser forzosamente 1, este no se almacena, con lo que se pueden almacenar los 23 bits siguientes y la mantisa queda definida con 24 bits. El exponente queda definido por 7 bits (uno para el signo), por lo que el mayor exponente representable es:

$$(1111111)_2 = 2^7 - 1 = 127;$$

de esta forma el número mayor representable será del orden de

$$2^{127} \approx 10^{38}$$

y su precisión, ya que para q sólo hay 24 bits, será

$$2^{-24} \approx 10^{-7},$$

es decir, un poco menor que la séptima cifra decimal. Habrá limitaciones: *overflow*, si el exponente resulta demasiado grande, *underflow*, si el exponente es demasiado pequeño.

Otro aspecto a considerar es el siguiente: Obsérvese que números reales sencillos no se pueden representar de forma exacta en máquina. Por ejemplo, ¿cúal es el número de máquina que representa al número $x = 2/3$?. Para ello, primero se transforma $2/3$ a base binaria siguiendo, por ejemplo, el proceso siguiente:

$$\frac{2}{3} = (0.b_1b_2b_3...)_2$$

Multiplicando por 2 resulta:

$$1 + \frac{1}{3} = (b_1.b_2b_3..)_2 \Longrightarrow b_1 = 1$$

Restando la parte entera y repitiendo el proceso se llega a que

$$\frac{2}{3} = (0.10101010...)_2$$

Los dos números de máquina más próximos con 24 bits serán:

$$\begin{aligned} x' &= (0.10101010...10)_2 \\ x'' &= (0.10101010...11)_2 \end{aligned}$$

El error cometido en ambos casos será:

$$x - x' = (0.10101010...)_2 * 2^{-24} = \frac{2}{3} * 2^{-24}$$

$$x'' - x = (x'' - x') - (x - x') = 2^{-24} - \frac{2}{3} * 2^{-24} = \frac{1}{3} * 2^{-24}$$

En consecuencia, debe tomarse x'' como valor de máquina en punto flotante; este valor se representa por $fl(x)$ y el error anterior se denomina **error de redondeo**. Por lo tanto, el error de redondeo absoluto será:

$$|fl(x) - x| = \frac{1}{3} * 2^{-24}$$

mientras que el error de redondeo relativo será:

$$\frac{|fl(x) - x|}{|x|} = \frac{\frac{1}{3} * 2^{-24}}{\frac{2}{3}} = 2^{-25} < 2^{-24}$$

Es importante conocer que al trabajar con punto flotante las operaciones aritméticas elementales (sumas, restas, multiplicaciones y divisiones), que se pueden representar genericamente por $\odot$, verifican la siguiente propiedad:

$$fl(x \odot y) = (x \odot y)\,(1 + \delta), \quad |\delta| < \epsilon$$

siendo ϵ el *cero de la máquina* (menor número de máquina positivo).

También hay que saber que, debido a la representación de los números en la máquina, las operaciones con números en punto flotante no siguen siempre las reglas típicas de la aritmética real (propiedades asociativa, distributiva, etc.)

3 Elección de un algoritmo

A la hora de elegir un algoritmo entre varios para resolver cierto problema es preciso tener en cuenta, además de las características teóricas de los mismos, algunas consideraciones de tipo práctico, como son:

- facilidad de implementación
- precision buscada
- coste computacional (medido, por ejemplo, en el número de operaciones aritméticas y lógicas requeridas)
- necesidad de almacenamiento (cantidad de memoria requerida)
- eficiencia

3.1 Criterios de parada

Desde luego, antes de poner en funcionamiento un algoritmo (no debe olvidarse que construye una sucesión de soluciones aproximadas de la solución exacta) es necesario establecer un criterio de parada. Por ejemplo, se puede establecer el **número máximo de iteraciones** o repeticiones del algoritmo; esto será oportuno si no se conoce la convergencia del algoritmo o la velocidad de convergencia. Si se sabe que el algoritmo es convergente, se puede definir una **tolerancia** y exigir que bien el error absoluto o el error relativo no la superen. Algunos otros criterios se pueden utilizar para algoritmos concretos.

References

[1] Nakamura, S.: Methodos numéricos aplicados con software, Prentice Hall, 1992.

[2] Woodford, C.: Solving linear and non-linear equations, Ellis Horwood, 1992.