# Capítulo 5

# Estabilidad

## 5.1. Introducción

## 5.2. Aritmética en punto flotante

Recordamos en esta sección, muy brevemente, las ideas expuestas en la sección con este mismo nombre en el documento **Prácticas de Análisis Matricial. Una Introducción a MATLAB**. En primer lugar, los números en punto flotante forman un subconjunto finito **F** de números racionales. Este conjunto está formado por el 0 y los números de la forma

$$x = \pm(1 + f) \cdot 2^e$$

con

$$0 \leqslant f < 1 \quad \text{y } e \text{ un número entero.}$$

La precisión del conjunto **F** queda determinada por un número llamado **épsilon de la máquina** que denotaremos con el símbolo $\epsilon_M$, y cuyo valor es

$$\epsilon_M = 2^{-53}.$$

Este número está muy relacionado con el error de redondeo al aproximar los números reales al número en **F** más próximo. Específicamente, si $\mathrm{fl}(x)$ es el número de **F** más próximo a $x \in \mathbb{R}$, $x \neq 0$, entonces

$$\frac{|\,\mathrm{fl}(x) - x|}{|x|} \leqslant 2^{-53} = \epsilon_M. \tag{5.1}$$

En algunos casos, como MATLAB, se toma como $\epsilon_M$ el doble de este valor; es decir, $2^{-52}$. Este número es la distancia de 1 al siguiente número en punto flotante. De hecho, en el intervalo $[1, 2]$ los números en punto flotante son equidistantes y la distancia entre dos de ellos es precisamente este número: $2^{-52}$.

La desigualdad (5.1) se puede escribir como

$$|\,\mathrm{fl}(x) - x| \leqslant \epsilon_M |x|,$$

en la que se puede incluir $x = 0$. A su vez, esta desigualdad permite establecer la siguiente norma que relaciona los números reales y los de **F**:

**Primer axioma para F**

Para cada $x \in \mathbb{R}$ hay un $\epsilon$ con $|\epsilon| \leqslant \epsilon_M$ tal que

$$\mathrm{fl}(x) = x(1 + \epsilon). \tag{5.2}$$

En un ordenador, todos los cálculos matemáticos se reducen a ciertas operaciones aritméticas elementales, de las cuales las clásicas son $+$, $-$, $\times$, y $\div$. Estos símbolos representan operaciones en $\mathbb{R}$. En un ordenador tenemos operaciones similares en **F**. Habitualmente estas operaciones aritméticas con números en punto flotante se representan con los símbolos $\oplus$, $\ominus$, $\otimes$ y $\oslash$, y en la práctica son implementadas mediante software.

Respecto de estas operaciones es razonable desear que los ordenadores se construyeran siguiendo el siguiente principio: si $x$, $y$ son números en punto flotante (i.e.,

$x, y \in \mathbf{F}$), $*$ representa una de las operaciones $+$, $-$, $\times$, o $\div$, y si $\circledast$ la operación correspondiente en punto flotante, entonces $x \circledast y$ debe ser exactamente:

$$x \circledast y = \mathrm{fl}(x * y) \tag{5.3}$$

Si se cumple esta condición, por (5.2) y (5.3), podemos concluir que el ordenador tiene la siguiente importante propiedad:

**Axioma fundamental de la aritmética en punto flotante**

Para todo $x, y \in \mathbf{F}$, existe $\epsilon$ con $|\epsilon| \leqslant \epsilon_M$ tal que

$$x \circledast y = (x * y)(1 + \epsilon) \tag{5.4}$$

Recordemos que si $x \in \mathbb{R}$ y $\mathrm{fl}(x)$ es el valor aproximado de $x$ en punto flotante, el error relativo al aproximar $x$ por $\mathrm{fl}(x)$ es

$$\frac{|\,\mathrm{fl}(x) - x|}{|x|}.$$

Por lo tanto, el axioma fundamental de la aritmética en punto flotante dice que *el error relativo en cada operación de la aritmética en punto flotante es a lo más* $\epsilon_M$.

El análisis del error de las operaciones con números en punto flotante está basado en las condiciones (5.2) y (5.4) para valores de $\epsilon_M$ que no tienen por qué ser $2^{-53}$. Así nuestro análisis incluirá, por ejemplo, ordenadores o calculadoras que en vez de redondeo utilizan truncamiento para aproximar un número real por el más próximo en punto flotante. En estos casos, el error relativo al aproximar un número real por el más cercano, truncando en vez de redondeamdo, es $2^{-52}$. En vez de pensar en $\epsilon_M$ como un valor fijo, pensaremos en $\epsilon_M$ de un ordenador o calculadora como el *menor número para el que se cumplen las condiciones* (5.2) y (5.4) en ese aparato.

Durante mucho tiempo, y hoy todavía en algunos libros, se define el épsilon de la máquina como la distancia de 1 al siguiente número en punto flotante en el sistema $\mathbf{F}$ de dicha máquina. Hoy en día, sin embargo, se prefiere la definición dada más arriba por ser más independiente del ordenador concreto o sistema $\mathbf{F}$ que esté utilizando. Para la mayor parte de los ordenadores, incluyendo todos los que utilizan la aritmética del estándar IEEE, ambas definiciones de $\epsilon_M$ producen, aproximadamente, el mismo valor: en aritmética decimal $2^{-52}$ y $2^{-53}$ son ambos del orden de $10^{-16}$. En ocasiones, sin embargo, puede ser necesario un valor grande de $\epsilon_M$ para que se cumplan las condiciones (5.2) y (5.4). En 1994 se descubrió en los microprocesadores

Intel Pentium un error en la tabla que usaban para implementar el estándar IEEE de doble precisión; su precisión efectiva resultó ser de, aproximadamente, $6{,}1 \times 10^{-5}$, $10^{11}$ veces menor que la supuestamente deseada. Por supuesto, el error fué corregido muy pronto. Pero hay más ejemplos. La operación de restar en punto flotante en los ordenadores Cray que se produjeron hasta mediados de la década de 1990 necesitaba un $\epsilon_M = 1$para que se cumpliera (5.4). Esto era debido a que la substracción se realizaba sin lo que se llama el *dígito de guardia*. Este concepto se entiende mejor con un ejemplo. Supongamos que $\mathbf{F}$ es el conjunto de números en punto flotante representados en base 2 y con 3 posiciones en la fracción (no importa el rango del exponente para este ejemplo). Supongamos que queremos restar a 1 el anterior número de $\mathbf{F}$. Este número es 0,111. Escribimos estos números en forma normalizada

$$\begin{array}{rcl} 1 & \rightarrow & 0{,}100 \times 2^1 \\ 0{,}111 & \rightarrow & 0{,}111 \times 2^0, \end{array}$$

y los restamos

$$\begin{array}{rcl} 0{,}100 \times 2^1 & & \phantom{-}0{,}100 \times 2^1 \\ -0{,}111 \times 2^0 & \rightarrow & -0{,}0111 \times 2^1 \\ & & \hline \\ & & 0{,}0001 \times 2^1 = 0{,}100 \times 2^{-2} \end{array}$$

Observamos, en primer lugar, que la respuesta es correcta: en notación decimal, los números positivos de $\mathbf{F}$ son $0, 2^{-3}, 2 \cdot 2^{-3}, 3 \cdot 2^{-3}, \ldots, 7 \cdot 2^{-3}, 1$. Es decir, la diferencia entre 1 y su anterior número en $\mathbf{F}$ es $2^{-3} = 0{,}100 \times 2^{-2}$. En segundo lugar, al pasar de representar $0, 111$ como $-0{,}111 \times 2^0$ a $0{,}0001 \times 2^1$ hemos introducido un cuarto dígito en la fracción. Este es el dígito de guardia. Si no lo hubiéramos hecho así la operación habría sido como sigue:

$$\begin{array}{rcl} 0{,}100 \times 2^1 & & \phantom{-}0{,}100 \times 2^1 \\ -0{,}111 \times 2^0 & \rightarrow & -0{,}011 \times 2^1 \\ & & \hline \\ & & 0{,}001 \times 2^1 = 0{,}100 \times 2^{-1} \end{array}$$

¡El error relativo de esta respuesta es 1!:

$$\frac{|2^{-2} - 2^{-3}|}{|2^{-3}|} = 1.$$

Recordemos que para los antiguos ordenadores Cray $\epsilon_M = 1$. Estas máquinas no son inútiles pero el análisis del error debe ser diferente del que nosotros haremos. Afortunadamente el axioma (5.4) y la adopción del estándar IEEE para la aritmética de punto flotante han sido ampliamente admitidos por los productores de ordenadores;

y desde 1996 todos los ordenadores personales compatibles IBM y todas las estaciones de trabajo producidas por SUN, DEC, HP e IBM implementan el estándar IEEE que se expuso en el documento **Prácticas de Análisis Matricial. Una Introducción a MATLAB**.

## 5.3. Estabilidad de los algoritmos

Sería bueno que los algoritmos numéricos proporcionaran respuestas exactas a los problemas numéricos, pero como los números reales forman un continuo y los ordenadores sólo pueden utilizar un número finito de números racionales, ese objetivo es, en general, imposible de alcanzar. La noción de estabilidad es la forma estándar de caracterizar lo que es posible; lo que los analistas numéricos consideran que es la "respuesta correcta" aunque no sea la exacta.

En la Lección anterior definimos un *problema* como una función $f : X \to Y$ entre dos espacios vectoriales; uno de datos, $X$, y otro de soluciones, $Y$. Un *algoritmo* también puede verse como una función $\tilde{f} : X \to Y$ entre los mismos espacios vectoriales. Pero la definición es un poco más complicada: Supongamos que nos dan los siguientes datos:

- un problema $f : X \to Y$,
- un ordenador cuyo sistema en punto flotante cumple (5.4) ,
- un algoritmo para $f$ en el sentido intuitivo del término (una serie de mandatos implementables en el ordenador con el objetivo de resolver $f$), y
- una implementación de este algoritmo en la forma de un programa para el ordenador.

Dado un dato $x \in X$, sea $\mathrm{fl}(x)$ el valor redondeado de $x$ en el sistema de punto flotante de forma que se verifique (5.2), y proporcionemos este valor, $\mathrm{fl}(x)$, como entrada al programa del ordenador. El resultado es una colección de números en punto flotante que están en $Y$. El resultado obtenido es $\tilde{f}(x)$.

Realmente esta definición no parece tener demasiado sentido. Para empezar $\tilde{f}(x)$ estará afectada por errores de redondeo; y, dependiendo de las situaciones, puede

verse afectada por todo tipo de circunstancias, como tolerancias para que haya convergencia o, incluso, otras tareas que pueda estar realizando el ordenador cuando se corre el programa. Esta "función" $\tilde{f}(x)$ puede tomar diferentes valores de una a otra ejecución del programa. En fin, todas estas complicaciones pueden hacernos pensar que no merece la pena intentar definir un algoritmo como una "función". Así y todo, hablar de $\tilde{f}(x)$ como una función es conveniente y sobre ella se pueden establecer unos cuantos resultados que permiten un mejor análisis de la exactitud de los algoritmos de Álgebra Lineal Numérica; y todo ello basado exclusivamente en los axiomas fundamentales (5.2) y (5.4).

El símbolo $\tilde{}$ es una notación que usaremos para expresar la cantidad calculada por el ordenador. Por ejemplo, la cantidad calculada por un ordenador cuyo sistema de aritmética en punto flotante cumpla los axiomas (5.2) y (5.4) del sistema lineal $Ax = b$ será denotada por $\tilde{x}$.

Lo más importante que podemos pedir a un buen algoritmo es que nos proporcione una buena aproximación al problema asociado $f$. Para precisar esta idea, consideremos el error absoluto del cálculo para un dato dado $x$, $\|f(x) - \tilde{f}(x)\|$, o el error relativo

$$\frac{\|f(x) - \tilde{f}(x)\|}{\|f(x)\|},$$

que será nuestra mediada habitual del error. Si $\tilde{f}$ es un buen algoritmo, se podría esperar un error relativo pequeño, del orden del $\epsilon_M$. Se podría decir que *un algoritmo $\tilde{f}$ para un problema $f$ es preciso, si para cada $x \in X$*

$$\frac{\|f(x) - \tilde{f}(x)\|}{\|f(x)\|} = O(\epsilon_M). \tag{5.5}$$

Hablando vagamente, el símbolo $O(\epsilon_M)$ significa "del orden del épsilon de la máquina". Sin embargo, $O(\epsilon_M)$ tiene un significado preciso. Recordémoslo. La notación

$$\varphi(t) = O(\phi(t))$$

es estándar en matemáticas y significa que existe una constante positiva $C$ tal que, para $t$ suficientemente próximo a un límite sobreentendido (como, por ejemplo $t \to 0$ o $t \to \infty$),

$$|\varphi(t)| \leqslant C|\phi(t)|.$$

Así, $\operatorname{sen}^2(t) = O(t^2)$ para $t \to 0$ porque existe una constante $C$ tal que $|\operatorname{sen}^2(t)| \leqslant C|t^2|$ para $t$ suficientemente pequeño como lo atestigua el desarrollo en serie de Taylor de $\operatorname{sen}^2(t)$.

También es estándar en matemáticas la expresión

$$\varphi(s,t) = O(\phi(t)) \text{ uniformemente en } s,$$

para indicar que existe una constante $C$ tal que

$$|\varphi(t,s)| \leqslant C|\phi(t)|$$

para todo $s$. Así, por ejemplo,

$$(\operatorname{sen}^2 t)(\operatorname{sen}^2 s) = O(t^2)$$

cuando $t \to 0$, pero la "uniformidad" se pierde si reemplazamos $\operatorname{sen}^2 s$ por $s^2$.

Aquí el uso de "$O$" sigue estos mismos criterios. Así, para un problema dado $f$ y un algoritmo $\tilde{f}$ para este problema, el resultado de aplicar el algoritmo a un dato $x \in X$, depende tanto de $x$ como del épsilon de la máquina, $\epsilon_M$, respecto del que se realizan los redondeos. La expresión (5.5) indica que existe una constante $C$ tal que el error relativo está acotado por $C\epsilon_M$ uniformemente para todo $x \in X$ y para $\epsilon_M \to 0$. Rara vez se menciona la uniformidad respecto de $x$; debe darse por sobreentendida.

En realidad $\epsilon_M$ es una cantidad fija para cada máquina. Cuando decimos que $\epsilon_M \to 0$ estamos pensando en una idealización de los ordenadores. La ecuación (5.5) significa que si corriéramos el algoritmo $\tilde{f}$ en unos ordenadores que satisfacen las condiciones (5.2) y (5.4) para una secuencia de valores de $\epsilon_M$ que converge a cero, entonces

$$\frac{\|f(x) - \tilde{f}(x)\|}{\|f(x)\|},$$

convergería a cero a la misma velocidad, al menos, que $\epsilon_M$.

Volviendo ahora a la precisión de los algoritmos, si el problema $f$ está mal condicionado para algún $x$, el objetivo de precisión definido por (5.5) es poco razonable porque el redondeo de los datos es inevitable en un ordenador digital. Incluso si todos los cálculos posteriores fueran hechos exactamente, esa perturbación inicial en los datos produciría grandes cambios (en términos relativos) en los resultados. En lugar de intentar la precisión en todos los casos, a lo más a lo que podemos aspirar, en general, es a la *estabilidad*.

**Definición 5.1** *Un algoritmo $\tilde{f}$ para un problema $f$ se dice que es* ***estable*** *si para cada $x \in X$ existe $\tilde{x}$ para el que*

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon_M)$$

*y*

$$\frac{\|f(\tilde{x}) - \tilde{f}(x)\|}{\|f(\tilde{x})\|} = O(\epsilon_M). \tag{5.6}$$

En palabras

> Un algoritmo estable es el que produce una respuesta casi correcta a una pregunta casi correcta.

El concepto de estabilidad tiene diferentes significados en otras partes del análisis numéricos. La definición que acabamos de dar es buena para analizar los algoritmos propios del Álgebra Lineal Numérica, pero puede no ser apropiada para otras áreas, como las ecuaciones diferenciales.

Muchos algoritmos de Álgebra Lineal Numérica satisfacen una condición de estabilidad que es a la vez más fuerte y más sencilla:

**Definición 5.2** *Un algoritmo $\tilde{f}$ para un problema $f$ se dice que es* ***estable hacia atrás*** *(*backward stable*) si para cada $x \in X$ existe $\tilde{x}$ para el que*

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon_M)$$

*y*

$$\tilde{f}(x) = f(\tilde{x}).$$

La condición de estabilidad hacia atrás es más fuerte que la de estabilidad dada en la Definición 5.1 porque el $O(\epsilon_m)$ de la condición (5.6) ha sido sustituído por 0. En palabras:

> Un algoritmo estable hacia atrás es el que produce una respuesta correcta a una pregunta casi correcta.

O dicho de otra forma: *un algoritmo es estable hacia atrás si al ser aplicado a unos datos exactos produce el mismo resultado que si se resolviera exactamente el problema con unos datos aproximados*. El error relativo en estos datos aproximados debe ser del orden $O(\epsilon_M)$.

Antes de ver algunos ejemplos generales de algoritmos estables, quizá sea ilustrativo estudiar lo que significa la estabilidad hacia atrás con un ejemplo numérico.

**Ejemplo 5.3** Supongamos que queremos reducir la matriz

$$A = \begin{bmatrix} 3{,}000 & 2{,}000 \\ 1{,}000 & 2{,}000 \end{bmatrix}$$

a forma triangular superior mediante operaciones elementales. Este es el principio del algoritmo de eliminación gaussiana para luego resolver el sistema correspondiente mediante sustitución hacia atrás. Lo haremos usando aritmética decimal de cuatro dígitos en la fracción. ¿Qué significa esto? Es importante aclararlo para saber hacer las operaciones. Al hablar de aritmética decimal nos referimos a que la base de numeración es 10. Y al decir que la fracción es de cuatro dígitos queremos decir que todos los números se aproximan por redondeo a uno de la forma

$$\pm f \cdot 10^e, \quad f = 0.x_1x_2x_3x_4, \quad 0 \leqslant x_i \leqslant 9$$

Así, los números en el intervalo $[1, 10)$ en este sistema serían (en forma no normalizada pero fácilmente normalizable)

$$\begin{array}{cccccc} 1 & 1{,}001 & 1{,}002 & \cdots & 1{,}998 & 1{,}999 \\ 2 & 2{,}001 & 2{,}002 & \cdots & 2{,}998 & 2{,}999 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 9 & 9{,}001 & 9{,}002 & \cdots & 9{,}998 & 9{,}999 \end{array}$$

Y los números en el intervalo $[10^j, 10^{j+1})$ se obtienen al multiplicar cada uno de los números de esta tabla por $10^j$, $mine \leqslant j \leqslant maxe$, siendo $mine$ y $maxe$ los valores mínimo y máximo asignados a $e$ (que no vamos a especificar porque no los necesitaremos en nuestro cálculo, pero que se puede pensar que son tan pequeño y grande, respectivamente, como se necesite).

En este sistema tomando, como es habitual salvo que tengamos otra referencia, $\epsilon_M$ como la distancia de 1 al siguiente número en punto flotante 1,001, resulta que

$$\epsilon_M = 10^{-3}. \tag{5.7}$$

Y, al operar, cualquier resultado obtenido que sea menor que $\epsilon_M$ podrá ser considerado como cero. Por supuesto, si $mine < -3$ hay números menores que $\epsilon_M$ que son representables en este sistema y no son cero. Lo que se quiere decir es que si al hacer, digamos, una resta el resultado es menor que $\epsilon_M$, el número será redondeado a cero. Esto es una práctica habitual en los ordenadores en los procesos cuyo objetivo es converger a cero o hacer cero ciertas cantidades mediante operaciones. En ocasiones, estos ceros no son absolutos y se redondea a cero cuando dichas cantidades son menores que un cierto valor que se suele llamar tolerancia. Así, en el sistema elegido la diferencia $1 - 0{,}9999$ se tendría que hacer con el dígito de guardia:

$$\begin{array}{rcl} 0{,}1000 \times 10^1 & & 0{,}1000 \times 10^1 \\ -0{,}9999 \times 10^0 & \rightarrow & -0{,}09999 \times 10^1 \\ \hline & & 0{,}00001 \times 10^1 = 0{,}1000 \times 10^{-3} \end{array}$$

que sería considerado como $0,000$ (si ésta fuera la aritmética de MATLAB, el número que presentaría en pantalla sería $0,0000$) .

Volviendo a la matriz $A$, notamos que ya la hemos escrito en el sistema aritmético especificado más arriba, aunque los números no están normalizados.

La única operación que tenemos que hacer es restar a la segunda fila la primera multiplicada por $1/3$, pero con la aritmética que estamos utilizando:

$$m_{21} = a_{21}/a_{11} = \text{fl}(1{,}000/3{,}000) = 0{,}3333 \quad (\text{o } 3{,}333 \cdot 10^{-1} \text{ en forma normalizada})$$

Si sumamos a la segunda fila de $A$ la primera multiplicada por $m_{21}$ tenemos que si $A' = [a'_{ij}]$ representa la matriz obtenida entonces

$$a'_{21} = \text{fl}(1{,}000 - 0{,}3333 \times 3{,}000) = \text{fl}(1{,}000 - 0{,}9999) = 0{,}000,$$

tal y como hemos visto más arriba. Y

$$a'_{22} = \text{fl}(2{,}000 - 0{,}3333 \times 2{,}000) = \text{fl}(2{,}000 - 0{,}6666) = \text{fl}(1{,}3334) = 1{,}333$$

Así

$$A' = \begin{pmatrix} 1{,}000 & 2{,}000 \\ 0 & 1{,}333 \end{pmatrix}$$

El objetivo es encontrar una matriz $\tilde{A}$ que al hacer esta misma operación en aritmética exacta nos de como resultado $A'$. Procediendo paso a paso con todo detalle, escogemos

$$\tilde{a}_{21} = 0{,}9999,$$

Así, exactamente

$$m_{21} = \tilde{a}_{21}/a_{11} = 0{,}9999/3{,}000 = 0{,}3333.$$

y

$$\tilde{a}_{21} - m_{21}a_{11} = 0$$

exactamente. Ahora debemos escoger $\tilde{a}_{22}$ de forma que

$$\tilde{a}_{22} - m_{21}a_{12} = 1{,}333$$

exactamente. Es decir,

$$\tilde{a}_{22} = 1{,}333 + 0{,}3333 \cdot 2 = 1{,}9996$$

La matriz $\tilde{A}$ deseada es

$$\tilde{A} = \begin{bmatrix} 3 & 2 \\ 0{,}9999 & 1{,}9996 \end{bmatrix}$$

En símbolos

$$f(\tilde{A}) = \tilde{f}(A).$$

Para que el algoritmo sea estable haca atrás debe ser

$$\frac{\|A - \tilde{A}\|}{\|A\|} = O(\epsilon_M)$$

Como todas las normas son equivalentes basta que lo veamos en una de ellas. Escogemos, por sencillez, la norma $\ell_\infty$ vectorial; i.e. $\|A\| = \text{máx}_{1 \leqslant i,j \leqslant 2} |a_{ij}|$. Así, como

$$A - \tilde{A} = \begin{pmatrix} 0 & 0 \\ 10^{-4} & 4 \cdot 10^{-4} \end{pmatrix}$$

resulta que $\|A - \tilde{A}\| = 4 \cdot 10^{-4}$ y $\|A\| = 3$. Por lo tanto

$$\frac{\|A - \tilde{A}\|}{\|A\|} = \frac{4}{3} 10^{-4} = O(\epsilon_M)$$

tal y como se deseaba mostrar.

En todos los ejemplos numéricos de esta lección asumiremos que la máquina sobre la que los hacemos opera con la aritmética que acabamos de decir y, que como consecuencia, su épsilon es el de (5.7).

## 5.4. Ejemplos de algoritmos estables

Veremos en esta sección tres ejemplos de algoritmos estables hacia atrás y uno que es estable pero no lo es hacia atrás.

### 5.4.1. Estabilidad de las operaciones aritméticas elementales

Comencemos con las cuatro operaciones aritméticas elementales: $+$, $-$, $\times$, $\div$. No hay mucho que decir sobre la posibilidad de elegir algoritmos para realizarlas: vienen implícitas en los ordenadores que las implementas con las operaciones $\oplus$, $\ominus$, $\otimes$ y $\oslash$, respectivamente. En un ordenador que satisfaga los axiomas (5.2) y (5.4) los algoritmos implícitos son estables hacia atrás. Lo mostraremos con la resta, que es la operación más susceptible de inestabilidad.

En este caso $X = \mathbb{R}^2$ y $Y = \mathbb{R}$ (en sección supondremos que la aritmética se hace con números reales). Para $(x_1, x_2) \in X$, $f(x_1, x_2) = x_1 - x_2$ es el problema a resolver y el algoritmo es

$$\tilde{f}(x_1, x_2) = \mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2). \tag{5.8}$$

Hagamos primero un ejemplo en aritmética decimal de punto flotante con 4 dígitos en la fracción. Sea $x_1 = \pi$ y $x_2 = \sqrt{2}/2$. Entonces

$$\mathrm{fl}(x_1) = 3{,}142, \quad \mathrm{fl}(x_2) = 0{,}7071 = 7{,}071 \cdot 10^{-1}$$

y utilizando el dígito de guardia:

$$\mathrm{fl}(x_1) - \mathrm{fl}(x_2) = 2{,}4349.$$

Es decir,

$$\mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) = 2{,}435.$$

Tomando, por ejemplo, $\tilde{x}_1 = 3{,}142$ y $\tilde{x}_2 = 0{,}7070$ tenemos que

$$\tilde{x}_1 - \tilde{x}_2 = 2{,}435$$

en aritmética exacta y el error de esta aproximación es

$$\frac{|x_1 - \tilde{x}_1|}{|x_1|} \leqslant 1{,}3 \cdot 10^{-4}, \quad \frac{|x_2 - \tilde{x}_2|}{|x_2|} \leqslant 1{,}6 \cdot 10^{-4}$$

ambos del orden de $\epsilon_M$, (5.7).

Procedamos a ver que lo que nos indica este ejemplo es verdad en general. En primer lugar la condición (5.8) significa que primero redondeamos $x_1$ y $x_2$ a sus valores más próximos en punto flotante y luego realizamos la operación $\ominus$. Por (5.2)

$$\mathrm{fl}(x_1) = x_1(1 + \epsilon_1), \quad \mathrm{fl}(x_2) = x_2(1 + \epsilon_2)$$

para algunos $|\epsilon_1|,\ |\epsilon_2| \leqslant \epsilon_M$. Y por (5.4)

$$\mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) = (\mathrm{fl}(x_1) - \mathrm{fl}(x_2))(1 + \epsilon_3)$$

para algún $|\epsilon_3| < \epsilon_M$. Combinando estas dos ecuaciones

$$\begin{aligned} \mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) &= [x_1(1 + \epsilon_1) - x_2(1 + \epsilon_2)](1 + \epsilon_3) = \\ &= x_1(1 + \epsilon_1)(1 + \epsilon_3) - x_2(1 + \epsilon_2)(1 + \epsilon_3) = \\ &= x_1(1 + \epsilon_1 + \epsilon_3 + \epsilon_1\epsilon_3) - x_2(1 + \epsilon_2 + \epsilon_3 + \epsilon_2\epsilon_3) \end{aligned}$$

Pongamos

$$\tilde{x}_1 = x_1(1 + \epsilon_1 + \epsilon_3 + \epsilon_1\epsilon_3), \tilde{x}_2 = x_2(1 + \epsilon_2 + \epsilon_3 + \epsilon_2\epsilon_3).$$

Tenemos que

$$\frac{|\tilde{x}_1 - x_1|}{|x_1|} \leqslant |\epsilon_1| + |\epsilon_3| + |\epsilon_1||\epsilon_3| \leqslant 2\epsilon_M + \epsilon_M^2.$$

Para $\epsilon_M$ suficientemente pequeño $\epsilon_M^2 \leqslant \epsilon_M$ por lo que

$$\frac{|\tilde{x}_1 - x_1|}{|x_1|} = O(\epsilon_M).$$

Y de la misma manera

$$\frac{|\tilde{x}_2 - x_2|}{|x_2|} = O(\epsilon_M).$$

Esto significa que $\tilde{f}(x) = \mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) = \tilde{x}_1 - \tilde{x}_2 = f(\tilde{x})$ con

$$\frac{\|\tilde{x} - x\|_\infty}{\|x\|_\infty} = O(\epsilon_M)$$

Pero como todas las normas en $\mathbb{F}^2$ son equivalentes podemos sustituir la norma $\ell_\infty$ por cualquier norma. Por lo tanto se verifica la condición de estabilidad hacia atrás de la Definición 5.2.

**Observaciones 5.4** En el análisis de la estabilidad hacia atrás aparecen muy a menudo expresiones de la forma

$$(1+\epsilon_1)(1+\epsilon_2)\cdot\ldots\cdot(1+\epsilon_n)$$

para algún valor de $n$. De hecho, en el análisis recién realizado para la resta hemos visto que

$$(1+\epsilon_1)(1+\epsilon_2) \leqslant 1+2\epsilon_M+O(\epsilon_M^2).$$

Y como, para $\epsilon_M < 1$ tenemos que $\epsilon_M^2 \leqslant \epsilon_M$ podríamos poner

$$(1+\epsilon_1)(1+\epsilon_2) \leqslant 3\epsilon_M.$$

Pero en realidad tenemos una cota mucho mejor que nos la proporciona el siguiente resultado, que damos sin demostración (ver Problema 4).

**Proposición 5.5** *Si* $n\epsilon_M \leqslant 0{,}1$ *y para* $i=1,\ldots,n$ $|\epsilon_i| \leqslant \epsilon_M$, *entonces*

$$(1+\epsilon_1)(1+\epsilon_2)\cdot\ldots\cdot(1+\epsilon_n) = 1+\eta \tag{5.9}$$

*con*

$$|\eta| \leqslant 1{,}06n\epsilon_M.$$

A la cantidad

$$\epsilon'_M = 1{,}05\epsilon_M$$

se le llama, a veces, *unidad de redondeo ajustada*.

La condición $n\epsilon_M \leqslant 0{,}1$ puede parecer, si no se analiza de cerca, una suposición bastante restrictiva. Pero no lo es en manera alguna. En efecto, un ordenador que tardara en realizar cada suma $1\mu$segundos$=10^{-6}$ segundos con un $\epsilon_M = 10^{-15}$ necesitaría un $n = 10^{14}$ para que $n\epsilon_M \leqslant 0{,}1$. El tiempo que necesitaría para realizar este número de sumas sería

$$10^8 \text{ segundos } = 3{,}2 \text{ años }.$$

Como dice G. Stewart "no contengas la respiración esperando que $n\epsilon_M$ sea mayor que 0,1".

En consecuencia, ante una expresión de la forma (5.9) siempre escribiremos

$$(1+\epsilon_1)(1+\epsilon_2)\cdot\ldots\cdot(1+\epsilon_n) = 1+O(\epsilon_M),$$

o también, cuando queramos poner de manifiesto el número de productos involucrados

$$(1+\epsilon_1)(1+\epsilon_2)\cdot\ldots\cdot(1+\epsilon_n) \leqslant 1+n\epsilon_M+O(\epsilon_M^2),$$

interpretando que $O(\epsilon_M^2)$ es despreciable respecto a $\epsilon_M$.

### 5.4.2. Producto escalar

Supongamos que nos dan dos vectores $x, y \in \mathbb{R}^n$ y queremos calcular su producto escalar $\alpha = x^T y$. El algoritmo obvio es calcular los productos $x_i y_i$ con $\otimes$ y sumar los resultados con $\oplus$. Todavía quedaría la cuestión del orden en el que se realiza esta suma; supongamos que es de izquierda a derecha. Es decir:

$$(((x_1y_1 + x_2y_2) + x_3y_3) + \cdots).$$

Por sencillez supondremos que los vectores $x$, $y$ ya están redondeados; i.e. $x_i, y_i \in \mathbf{F}$, $i = 1, \ldots, n$ (esto nos evita un "$\epsilon$"para cada componente). Suponiendo que se verifican las condiciones (5.2) y (5.4) procederíamos de la siguiente forma:

$$s_1 = \mathrm{fl}(x_1y_1)x_1 \otimes y_1 = x_1y_1(1+\epsilon_1)$$

con $|\epsilon_1| < \epsilon_M$. De la misma forma

$$\begin{aligned}
s_2 &= \mathrm{fl}(s_1 + x_2y_2) = s_1 \oplus (x_2 \otimes y_2) = (s_1 + x_2y_2(1+\epsilon_{22}))(1+\epsilon_2) = \\
&= x_1y_1(1+\epsilon_1)(1+\epsilon_2) + x_2y_2(1+\epsilon_{22})(1+\epsilon_2). \\
s_3 &= \mathrm{fl}(s_2 + x_3y_3) = s_2 \oplus (x_3 \otimes y_3) = (s_2 + x_3y_3(1+\epsilon_{33}))(1+\epsilon_3) = \\
&= x_1y_1(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3) + x_2y_2(1+\epsilon_{22})(1+\epsilon_2)(1+\epsilon_3) \\
&\quad + x_3y_3(1+\epsilon_{33})(1+\epsilon_3).
\end{aligned}$$

con $|\epsilon_2|, |\epsilon_3|, |\epsilon_{22}|, |\epsilon_{33}| < \epsilon_M$. SE seguiría así sucesivamente hasta obtener la expresión de $s_n$ que sería la representación en punto flotante del producto escalar $x^T y$ cuando se usa el orden fijado más arriba. A fin de dar una expresión simplificada para este número haremos el siguiente convenio: $(1 \pm \epsilon)^j$ será una abreviatura para representar el producto de $j$ $\epsilon$s que pueden ser diferentes. Así $(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3)$ lo abreviaríamos a $(1 \pm \epsilon)^3$, pero $(1+\epsilon_{22})(1+\epsilon_2)(1+\epsilon_3)$ también lo abreviaríamos a $(1 \pm \epsilon)^3$. Utilizando este convenio es fácil ver que

$$s_n = x_1y_1(1 \pm \epsilon)^n + x_2y_2(1 \pm \epsilon)^n + x_3y_3(1 \pm \epsilon)^{n-1} + \cdots + x_ny_n(1 \pm \epsilon)^2.$$

Usando ahora la Proposición 5.5, tenemos

$$s_n = x_1y_1(1+\eta_n) + x_2y_2(1+\eta'_n) + x_3y_3(1+\eta_{n-1}) + \cdots + x_ny_n(1+\eta_2)$$

con $|\eta_i| \leqslant 1,06i\epsilon_M$ i $|\eta'_n| \leqslant 1,06n\epsilon_M$.

Poniendo, por ejemplo, $\tilde{x}_1 = x_1(1+\eta_n)$, $\tilde{x}_2 = x_2(1+\eta'_n)$, $\ldots$, $\tilde{x}_n = x_n(1+\eta_2)$ y $\tilde{y}_i = y_i$, $i = 1, \ldots, n$ (o al revés), tendríamos que para $i = 1, \ldots, n$,

$$\frac{|\tilde{x}_i - x_i|}{|x_i|} = O(\epsilon_M), \quad \frac{|\tilde{y}_i - y_i|}{|y_i|} = O(\epsilon_M)$$

y $\tilde{f}(x^Ty) = \tilde{x}^T\tilde{y}$. Es decir, el resultado devuelto por el algoritmo es el resultado exacto del producto escalar de dos vectores cuyas componentes difieren, en valor relativo, del orden de $\epsilon_M$ de los vectores dados. En otra palabras, el algoritmo es estable hacia atrás.

Un ejemplo numérico con $n = 2$ es el siguiente: Sean

$$x = \begin{bmatrix} 2/3 \\ 16/17 \end{bmatrix} \quad y = \begin{bmatrix} 1/6 \\ 125/126 \end{bmatrix}.$$

En un ordenador con aritmética decimal de 4 dígitos en la fracción tendríamos

$$\text{fl}(\text{fl}(x_1) \cdot \text{fl}(y_1)) = \text{fl}(0{,}6667 \cdot 0{,}1667) = 0{,}1111,$$

y

$$\text{fl}(\text{fl}(x_2) \cdot \text{fl}(y_2)) = \text{fl}(0{,}9412 \cdot 0{,}9921) = 0{,}9338.$$

Así

$$\text{fl}(\text{fl}(\text{fl}(x_1) \cdot \text{fl}(y_1)) + \text{fl}(\text{fl}(x_2) \cdot \text{fl}(y_2))) = 1{,}045.$$

Hay muchas formas de escoger $\tilde{x}$ y $\tilde{y}$ para que $\tilde{x}\tilde{y}^T = 1{,}045$ y que el error relativo sea del orden de $\epsilon_M = 10^{-3}$. Por ejemplo, si ponemos

$$\tilde{x}_1 = 0{,}6666, \quad \tilde{y}_1 = \frac{0{,}1111}{0{,}6666} = \frac{1}{6},$$

entonces

$$\tilde{x}_1\tilde{y}_1 = 0{,}1111 \text{ y } \frac{|x_1 - \tilde{x}_1|}{|x_1|} = 10^{-4}, \frac{|y_1 - \tilde{y}_1|}{|y_1|} = 0.$$

Y si ponemos

$$\tilde{x}_2 = \frac{16}{17} \quad \tilde{y}_2 = \frac{0{,}9339 \cdot 17}{16},$$

entonces

$$\tilde{x}_2\tilde{y}_2 = 0{,}9339 \text{ y } \frac{|x_2 - \tilde{x}_2|}{|x_2|} = 0, \quad \frac{|y_1 - \tilde{y}_1|}{|y_1|} \leqslant 2{,}07 \cdot 10^{-4}.$$

Además

$$\tilde{x}_1\tilde{y}_1 + \tilde{x}_2\tilde{y}_2 = 1{,}045.$$

### 5.4.3. Producto exterior

Supongamos ahora que queremos calcular el producto exterior de dos vectores. Es decir, para $x, y \in \mathbb{R}^n$, la matriz $A = xy^T$ que tiene rango 1. El algoritmo obvio para hacerlo es calcular cada uno de los productos

$$x_i y_j$$

y construir la matriz $\tilde{A}$ cuyo elemento en la posición $(i, j)$ sea $\mathrm{fl}(x_i) \otimes \mathrm{fl}(y_j)$. Este algoritmo es estable porque por (5.2) y (5.4)

$$\mathrm{fl}(x_i) \otimes \mathrm{fl}(y_j) = x_i(1 + \epsilon_i)y_j(1 + \epsilon_j)(1 + \epsilon_{ij})$$

con $|\epsilon_i|, |\epsilon_j|, |\epsilon_{ij}| \leqslant \epsilon_M$. Operando como más arriba vemos que

$$\mathrm{fl}(x_i) \otimes \mathrm{fl}(y_j) = x_i\overline{y_j}(1 + \eta), \quad |\eta| \leqslant 3, 18\epsilon_M,$$

lo que indica que

$$\frac{\|a_{ij} - \tilde{a}_{ij}\|}{\|a_{ij}\|} = O(\epsilon_M)$$

siendo $a_{ij} = x_i y_j$ y $\tilde{a}_{ij} = \mathrm{fl}(x_i) \otimes \mathrm{fl}(y_j)$ los elementos en la posición $(i, j)$ de $A = xy^T$ y $\tilde{A}$, respectivamente.

Por lo tanto

$$\frac{\|\tilde{A} - A\|_\infty}{\|A\|_\infty} = O(\epsilon_M),$$

y esta expresión es válida para cualquier norma por la equivalencia de normas. De acuerdo con la definición de estabilidad de los algoritmos resulta que este algoritmo es estable.

Con los vectores $x$, $y$ del ejemplo anterior

$$\tilde{f}(x, y) = \tilde{A} = \begin{bmatrix} 0{,}1111 & 0{,}6614 \\ 0{,}1569 & 0{,}9938 \end{bmatrix}$$

Si ponemos

$$\tilde{x}_1 = 0{,}6666, \tilde{x}_2 = \frac{16}{17}, \tilde{y}_1 = \frac{1}{6}, \tilde{y}_2 = \frac{0{,}9338 \cdot 17}{16}$$

resulta que, en aritmética exacta,

$$f(\tilde{x}, \tilde{y}) = \tilde{x}\tilde{y}^T = \begin{bmatrix} 0{,}1111 & 264550209/(4 \cdot 10^8) \\ 8/51 & 0{,}9338 \end{bmatrix}.$$

Usando la norma espectral

$$\frac{\|f(\tilde{x}, \tilde{y}) - \tilde{f}(x, y)\|}{\|f(\tilde{x}, \tilde{y})\|} \leqslant 1{,}183 \cdot 10^{-4}$$

Sin embargo, el algoritmo no es estable hacia atrás. La explicación es la siguiente: La matriz calculada, $\tilde{A}$, es una pequeña perturbación de la matriz $A = \left[x_i y_j\right]$ que tiene rango 1. Sabemos que el conjunto de las matrices de rango completo es un conjunto abierto y denso en $\mathbb{R}^{n \times n}$. Esto significa que tan cerca como queramos de $A$ hay matrices de rango superior a 1. Es muy probable que $\tilde{A}$ sea una de tales matrices. En efecto, para que $\tilde{A}$ fuera de rango 1, se debería poder escribir como el producto externo de dos vectores $\tilde{x}\tilde{y}^T$ con $\tilde{x}$ y $\tilde{y}$ próximos a $x$ y $y$, respectivamente. Ahora bien al hacer el cálculo de los elementos de la fila $i$ de $\tilde{A}$, debemos multiplicar en aritmética de punto flotante $\mathrm{fl}(x_i) \otimes \mathrm{fl}(\overline{y}_j)$, $j = 1, \ldots, n$, con los consiguientes redondeos. Aún suponiendo que $x_i$, $y_i$ ya están redondeados, tendríamos:

$$x_i \otimes y_j = x_i y_j (1 + \epsilon_{ij})$$

con, posiblemente, $\epsilon_{ij}$ diferentes para cada $i, j$. Es muy poco probable que, en estas circunstancias, $x_i \otimes y_j = \tilde{x}_i \tilde{y}_j$ para $i, j = 1, \ldots, n$. En otras palabras, muy probablemente la matriz $\tilde{A}$ tendrá rango mayor que 1. Esto imposibilita que se pueda escribir en la forma $\tilde{x}\tilde{y}^T$.

Podemos ver este hecho en el ejemplo anterior: Para

$$\tilde{f}(x, y) = \tilde{A} = \begin{bmatrix} 0{,}1111 & 0{,}6614 \\ 0{,}1569 & 0{,}9938 \end{bmatrix}$$

que es la matriz $\tilde{x}\tilde{y}^T$ redondeada con la aritmética de base decimal y 4 cifras en la mantisa, $\operatorname{rang} \tilde{A} = 2$. De hecho, sus valores singulares calculados con MATLAB son 1,16034185454010 y $2{,}454449082275455 \cdot 10^{-5}$. Por lo tanto, esta matriz nunca podrá escribirse como un producto externo de vectores, porque estas matrices tienen todas rango 1.

## 5.5. Estabilidad de la eliminación gaussiana

El problema de la estabilidad del algoritmo de la eliminación gaussiana ha sido objeto de estudio durante muchos años; y, aunque hoy en día se comprende bastante bien y su uso es casi siempre altamente fiable, hay todavía algunas cuestiones difíciles que merecen estudio y que inducen a utilizarlo con cuidado. Su estudio detallado requeriría mucho tiempo, así que aquí vamos ver lo justo para hacernos una idea de donde pueden estar las dificultades. Conviene en este punto repasar el Ejemplo 5.7 en el que se estudió la estabilidad hacia atrás de la eliminación gaussiana para un sistema de orden 2 pero que sirve como guía de lo que viene a continuación.

Como en casi todos los casos en los que se estudia la estabilidad de un algoritmo, la parte más tediosa del análisis es la de llevar la cuenta de los errores de redondeo en cada etapa del algoritmo. Haremos un análisis un poco detallado del primer paso del algoritmo y daremos directamente el resultado. Recordemos que este primer paso consiste en seleccionar el mayor elemento, en módulo, de la primera columna (el pivote) y permutar filas para ponerlo en la posición $(1,1)$. Esto no produce ningún tipo de error (el posible error en el redondedo de los datos ya se ha producido al almacenar la matriz en la memoria del ordenador). Así pues analizaremos lo que viene a continuación: se resta a cada fila la primera multiplicada por el multiplicador:

$$m_{i1} = a_{i1}/a_{11},$$

pero hecho en la aritmética de punto flotante (supondremos, para simplificar la notación y no hacer más tedioso el análisis, que los números de la matriz ya están redondeados al correspondiente número en punto flotante; i.e., $a_{ij} = \mathrm{fl}(a_{ij})$):

$$m_{i1} = \mathrm{fl}(a_{i1}/a_{11}) = \frac{a_{i1}}{a_{11}}(1+\epsilon_{i1})$$

con $|\epsilon_{i1}| \leqslant \epsilon_M$. Si ponemos

$$\tilde{a}_{i1} = a_{i1}(1+\epsilon_{i1}) \tag{5.10}$$

entonces, en aritmética exacta

$$m_{i1} = \tilde{a}_{i1}/a_{11}. \tag{5.11}$$

Procedemos a eliminar los elementos de la primera columna por debajo del elemento en la posición $(1,1)$. Para $i = 2,\ldots,n$ y $j = 1,\ldots,n$

$$\begin{aligned} a'_{ij} &= \mathrm{fl}(a_{ij} - m_{i1}a_{ij}) = [a_{ij} - m_{i1}a_{1j}(1+\epsilon_{ij})](1+\eta_{ij}) = \\ &= a_{ij} - m_{i1}a_{1j} + a_{ij}\eta_{ij} - m_{i1}a_{1j}(\epsilon_{ij} + \eta_{ij} + \epsilon_{ij}\eta_{ij}). \end{aligned}$$

Si ponemos

$$\tilde{a}_{ij} = a_{ij} + a_{ij}\eta_{ij} - m_{i1}a_{1j}(\epsilon_{ij} + \eta_{ij} + \epsilon_{ij}\eta_{ij}) \tag{5.12}$$

entonces

$$a'_{ij} = \tilde{a}_{ij} - m_{i1}a_{1j}. \tag{5.13}$$

De (5.13) y (5.11) se sigue que el resultado de realizar la eliminación gaussiana sobre la matriz original $A$ con redondeo es el mismo que el de realizarla en aritmética exacta sobre la matriz

$$\tilde{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \tilde{a}_{21} & \tilde{a}_{22} & \cdots & \tilde{a}_{2n} \\ \vdots & \vdots & & \vdots \\ \tilde{a}_{n1} & \tilde{a}_{n2} & \cdots & \tilde{a}_{nn} \end{bmatrix}.$$

Además, $A$ y $\tilde{A}$ están próximas la una de la otra. En efecto, de (5.12) se sigue que para $j > 1$

$$|\tilde{a}_{ij} - a_{ij}| \leqslant (a_{ij} + 3m_{i1}a_{1j})\epsilon_M.$$

Como la eliminación ha sido realizada con pivoteo, $|m_{i1}| \leqslant 1$. Y si $\alpha = \max_{i,j} |a_{ij}|$ entonces

$$|\tilde{a}_{ij} - a_{ij}| \leqslant 4\alpha\epsilon_M.$$

Por (5.10), esta acotación también se satisface para $j = 1$.

Esto nos da una idea del análisis de la estabilidad del algoritmo de eliminación gaussiana con pivoteo parcial por columnas. Una análisis completo requiere mucho más tiempo y detalle. Damos, a continuación, el resultado general sin demostración adicional y presentamos un ejemplo de una matriz $3 \times 3$

**Teorema 5.6** *Si se utiliza eliminación gaussiana con pivoteo parcial para resolver el sistema $Ax = b$, con $A \in \mathbb{F}^{n\times n}$, en un ordenador que cumple las condiciones (5.2) y (5.4), entonces la solución calculada, $\tilde{x}$, cumple*

$$(A + \delta A)\tilde{x} = b$$

*donde*

$$\frac{\|\delta A\|}{\|A\|} \leqslant \varphi(n)\gamma\epsilon_M, \tag{5.14}$$

*siendo $\varphi$ una función creciente de $n$ que depende de la norma, y $\gamma$ el cociente entre el mayor elemento en la matriz triangular superior resultante en el proceso de eliminación y el mayor elemento de $A$, todo ello en módulo.*

**Ejemplo 5.7** Supongamos que queremos reducir la matriz

$$A = \begin{bmatrix} 3{,}000 & 2{,}000 & 1{,}000 \\ 1{,}000 & 2{,}000 & 3{,}000 \\ 2{,}000 & 3{,}000 & 2{,}000 \end{bmatrix}$$

a una matriz triangular superior mediante eliminación gaussiana usando, como hasta ahora, aritmética decimal con 4 dígitos en la fracción.

En la primera columna el pivote es el elemento en la posición $(1,1)$ de modo que no hay que permutar filas. Los multiplicadores serán:

$$\begin{aligned} m_{21} &= \mathrm{fl}(1/3) = 0{,}3333 \\ m_{31}\, &\mathrm{fl}(2/3) = 0{,}6666 \end{aligned}$$

Para obtener ceros en las posiciones $(2,1)$ y $(3,1)$ con aritmética exacta y estos multiplicadores, ponemos

$$\tilde{a}_{21} = 3 \cdot 0{,}3333 = 0{,}9999, \quad \tilde{a}_{31} = 3 \cdot 0{,}6666 = 1{,}9998.$$

La submatriz formada por las dos últimas filas y columnas queda de la siguiente forma:

$$\begin{aligned} a'_{22} &= \mathrm{fl}(2 - 2 \cdot 0{,}3333) = 1{,}333, \quad a'_{23} = \mathrm{fl}(3 - 1 \cdot 0{,}3333) = 2{,}667, \\ a'_{32} &= \mathrm{fl}(3 - 2 \cdot 0{,}6666) = 1{,}667, \quad a'_{33} = \mathrm{fl}(2 - 1 \cdot 0{,}6666) = 1{,}333 \end{aligned}$$

Ahora en la matriz

$$A' = \begin{bmatrix} 3 & 2 & 1 \\ 0 & 1{,}333 & 2{,}667 \\ 0 & 1{,}667 & 1{,}333 \end{bmatrix}$$

el pivote de la segunda columna es el elemento en la posición $(3,2)$. Permutamos las dos filas para obtener

$$A'_1 = \begin{bmatrix} 3 & 2 & 1 \\ 0 & 1{,}667 & 1{,}333 \\ 0 & 1{,}333 & 2{,}667 \end{bmatrix}$$

Procedemos a anular el elemento en la posición $(3,2)$ de esta matriz (antiguo elemento $a'_{22}$). El multiplicador es

$$m_{32} = \mathrm{fl}(1{,}333/1{,}667) = 0{,}7996.$$

Para que en aritmética exacta con este multiplicador obtuviéramos un cero en la posición $(3,2)$ debería ser

$$\tilde{a'}_{22} = 1{,}667 \cdot 0{,}7996 = 1{,}3329332.$$

Pero $a'_{22}$ provenía de hacer una operación elemental con el multiplicador $m_{21}$. Así

$$\tilde{a}_{22} = \tilde{a'}_{22} + m_{21} \cdot a_{12} = 1{,}9995332.$$

Finalmente, debemos calcular el elemento que queda en la posición $(3,3)$:

$$\overline{a}_{33} = \mathrm{fl}(2{,}667 - 1{,}333 \cdot 0{,}7996) = 1{,}601.$$

Así en aritmética exacta

$$\tilde{a'}_{23} = 1{,}601 + 1{,}333 \cdot 0{,}7996 = 2{,}6668668.$$

Y como este elemento provenía de la operación elemental con el multiplicador $m_{21}$:

$$\tilde{a}_{23} = \tilde{a'}_{23} + m_{21} \cdot a_{13} = 3{,}0001668.$$

En aritmética en punto flotante la matriz triangular superior que se obtiene es

$$U = \begin{bmatrix} 3 & 2 & 1 \\ 0 & 1{,}667 & 1{,}333 \\ 0 & 0 & 1{,}601 \end{bmatrix}$$

Y esta matriz se obtendría mediante aritmética exacta sobre la matriz

$$\tilde{A} = \begin{bmatrix} 3 & 2 & 1 \\ 0{,}9999 & 1{,}9995332 & 3{,}0001668 \\ 1{,}9998 & 3{,}0002 & 1{,}9996 \end{bmatrix}$$

En efecto, $A$ y $\tilde{A}$ son muy parecidas. ¿Cuánto? En la norma espectral:

$$\frac{\|A - \tilde{A}\|_2}{\|A\|_2} \leqslant 1{,}04 \cdot 10^{-4}.$$

Si queremos tener una apreciación de la cota (5.14) para este ejemplo, calculamos

$$\gamma = \frac{\max_{ij} |u_{ij}|}{\max_{ij} |a_{ij}|} = \frac{3}{3} = 1.$$

Así

$$\frac{\|\delta A\|}{\|A\|} \leqslant \varphi(n) \cdot 10^{-3},$$

una cantidad muy conservadora en relación a la cota realmente obtenida con $\tilde{A}$.

**Observaciones 5.8** .-

1. El análisis del error hacia atrás muestra que la solución calculada para $Ax = b$ por eliminación gaussiana es la solución exacta de una matriz obtenida al perturbar ligeramente $A$. Si la cota (5.14) es pequeña pero el resultado no es preciso entonces no es el algoritmo el que está fallando sino el condicionamiento del problema o los errores de los datos iniciales.

2. La función $\varphi$ en (5.14) es, por lo general, una pequeña potencia de $n$, digamos $n^2$. Sin embargo, cualquier $\varphi$ matemáticamente rigurosa resulta siempre una sobreestimación del error real en la solución, que habitualmente es del orden de $n$ o menor.

3. El número $\gamma$ es conocido con el nombre de *factor de crecimiento* porque mide el crecimiento de los números durante el proceso de eliminación. Mantener este factor de crecimiento lo más pequeño posible es la razón para usar pivoteo. En efecto, con pivoteo los multiplicadores se matienen siempre acotados por 1; sin él, podrían alcanzar (por ejemplo cuando el elemento en la diagonal de la columna que se está reduciendo es casi cero) valores muy grandes. Esto provocaría que en la submatriz afectada por la eliminación los elementos crecieran muchísimo y $\gamma$ sería enorme comparado con $\epsilon_M$.

4. Desafortunadamente, hay matrices para las que $\gamma$ es enorme aún cuando se use pivoteo parcial. Por ejemplo, la eliminación gaussiana aplicada a la matriz

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}$$

sería de la siguiente forma

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & -1 & 1 & 0 & 2 \\ 0 & -1 & -1 & 1 & 2 \\ 0 & -1 & -1 & -1 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & -1 & 1 & 4 \\ 0 & 0 & -1 & -1 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & -1 & 8 \end{bmatrix} \rightarrow$$

$$\rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 0 & 16 \end{bmatrix}$$

Y si se aplica la eliminación gaussiana sobre una matriz del tipo de $W$ pero de tamaño $n \times n$, el factor de crecimiento sería $2^{n-1}$.

En realidad estos son ejemplos de laboratorio y rara vez aparecen en aplicaciones reales (aunque se han dado casos). La opinión general es que el algoritmo de eliminación gaussiana con pivoteo parcial es uno de los algoritmos más estables y eficientes para resolver sistemas lineales. A pesar de ello, hay que utilizarlo con un poco cuidado y estar alerta a las posibles anomalías que puedan presentar los resultados cuando son comparados con los esperados en la aplicación concreta que se tenga entre manos.

## 5.6. Precisión de un algoritmo estable hacia atrás

En la sección anterior al analizar la cota (5.14) del error en la eliminación gaussiana, dijimos que si ésta se mantiene razonablemente baja y los resultados son pobres, hay que echar la culpa a los errores en los datos iniciales (i.e., porque empezamos con una mala aproximación a $A$) o al condicionamiento del problema, que sabemos que es $\kappa(A)$. ¿Cómo influye el condicionamiento del problema en el error de los resultados proporcionados por los algoritmos estables? Esta es una pregunta natural que pretendemos contestar en esta última sección.

Supongamos que tenemos un algoritmo estable, $\tilde{f}$, para un problema $f : X \rightarrow Y$. ¿Cuán preciso será el resultado que devuelve? La respuesta depende del número de condición del problema. Si éste es pequeño, el resultado será preciso (en términos de

error relativo); pero si es grande, la falta de precisión será proporcional al número de condición:

**Teorema 5.9** *Dado un problema $f : X \to Y$ cuyo número de condición es $\kappa$, supongamos que un algoritmo estable hacia atrás es aplicado para resolverlo en un ordenador que cumple los axiomas (5.2) y (5.4). Entonces el error relativo cumple*

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O(\kappa(x)\epsilon_M).$$

**Demostración**.- Demostraremos este teorema en el caso en que $f$ es diferenciable. En el caso general la demostración es un poco más técnica. Si $f$ es diferenciable en $x$, para todo $\tilde{x}$ suficientemente próximo a $x$ tenemos que

$$f(x) - f(\tilde{x}) = f'(x)(x - \tilde{x}) + r(x - \tilde{x})$$

donde $\lim\limits_{\|x-\tilde{x}\|\to 0} \dfrac{\|r(x-\tilde{x})\|}{\|x-\tilde{x}\|} = 0$. Así

$$\begin{aligned} \frac{\|f(x) - f(\tilde{x})\|}{\|f(x)\|} &\leqslant \frac{\|x-\tilde{x}\|}{\|x\|}\frac{\|f'(x)\|\,\|x\|}{\|f(x)\|} + \frac{\|r(x-\tilde{x})\|}{\|f(x)\|} \\ &\leqslant \frac{\|x-\tilde{x}\|}{\|x\|}\kappa(x) + \frac{\|r(x-\tilde{x})\|}{\|f(x)\|}. \end{aligned}$$

Ahora bien si $\tilde{f}$ es un algoritmo estable hacia atrás para $x$, existe $\tilde{x}$ con

$$\frac{\|x-\tilde{x}\|}{\|x\|} = O(\epsilon_M) \tag{5.15}$$

tal que $f(\tilde{x}) = \tilde{f}(x)$. Entonces

$$\frac{\|f(x) - \tilde{f}(x)\|}{\|f(x)\|} \leqslant \left(\kappa(x) + \frac{\|r(x-\tilde{x})\|}{\|x-\tilde{x}\|}\frac{\|x\|}{\|f(x)\|}\right)\frac{\|x-\tilde{x})\|}{\|x\|}.$$

Dado que $\lim\limits_{\|x-\tilde{x}\|\to 0} \dfrac{\|r(x-\tilde{x})\|}{\|x-\tilde{x}\|} = 0$, para $\tilde{x}$ suficientemente próximo a $x$,

$$\frac{\|f(x) - \tilde{f}(x)\|}{\|f(x)\|} \leqslant \kappa(x)\frac{\|x-\tilde{x})\|}{\|x\|}.$$

Y como $\|x - \tilde{x}\| \to 0$ cuando $\epsilon_M \to 0$, concluímos que

$$\frac{\|f(x) - \tilde{f}(x)\|}{\|f(x)\|} = \kappa(x)O(\epsilon_M) = O(\kappa(x)\epsilon_M),$$

tal y como se quería demostrar. ■