Facultad

de

Ciencias

¿Es la composición de dos algoritmos estables, un algoritmo estable?

Does the composition of two stable algorithms result in a stable algorithm?

Trabajo de Fin de Grado

para acceder al

GRADO EN MATEMÁTICAS

Autora: Ariadna Álvarez Navarro

Director: Carlos Beltrán Álvarez

Septiembre - 2022

AGRADECIMIENTOS:

Primero quiero agradecerle a Carlos toda la ayuda, el apoyo y el ánimo que me ha proporcionado durante este año. Gracias por ayudarme a avanzar y aconsejarme como lo has hecho.

También quiero agradecerle a mi madre todo el apoyo incondicional que me ha dado durante todos estos años. Sin su ánimo no hubiese podido llegar hasta aquí. A mi padre, gracias por entenderme y por sacar la vena científica que había en mí. Y a mi hermano, gracias por creer en mí. A toda mi familia, gracias por creer siempre en mí.

Y por último, gracias a todos mis amigos, que me han acompañado en todos los momentos durante la carrera y me han enseñado a no tirar nunca la toalla.

Y no me olvido, gracias Chito.

Abstract

An important mathematical issue, which is in constant evolution is the use of algorithms in numerical analysis. More than three thousands years ago, Babylonians achieved a numerical aproximation of $\sqrt{2}$ in sexagesimal system. And this manner has only grown, especially since the arrival of computer science. But then, when could we say that the calculation of an algorithm is accurate? And, what level of accuracy does it have? These questions are going to be the main topic of this document. We will learn that some problems result in a more precise solution just because of its nature. We will also define the concept of *stability* and we will work with several valid deffinitions of it. And all to get to prove that, under some reasonable hypotheses the composition of two stable algorithms is also another stable algorithm.

Key words: *algorithm, stability, condition number, distance, composition*

Resumen

El uso de algoritmos en el análisis numérico es una cuestión matemática que podemos ver en constante evolución. Hace ya más de 3.000 años, los babilonios consiguieron una aproximación numérica sexagesimal al valor de $\sqrt{2}$ y, desde entonces, (sobre todo a partir de la llegada de los ordenadores) este campo no ha hecho más que crecer. Pero entonces, ¿cuándo podemos decir que el resultado obtenido por el cómputo de un algoritmo es correcto? ¿Y en qué medida? Estas preguntas van a ser el tema principal de este trabajo. Veremos que, por su propia naturaleza, algunos problemas resultan en soluciones más acertadas. También le daremos sentido a la palabra *estabilidad* y trabajaremos con varias definiciones válidas. Todo para conseguir demostrar que la composición de dos algoritmos estables resulta en otro algoritmo que también está dotado de esta característica.

Palabras clave: *algoritmo, estabilidad, número de condición, distancia, composición*

Índice

Capítulo 1

Introducción

¿Qué significa que un algoritmo para el cálculo de una función sea estable? Para contestar esta pregunta primero analicemos, a grandes rasgos, el contexto histórico que induce al concepto de *estabilidad*.

1.1. Historia

A principios del siglo XX se hizo notoria la insuficiencia de los métodos algebraicos existentes hasta el momento para la resolución de problemas, dada la dificultad que estos comenzaban a presentar. Es por este motivo que a mediados de los años 30 surgió el concepto de ordenador de *programa almacenado*. Tiempo atrás, los pocos ordenadores existentes no eran reprogramables sino que ejecutaban un único programa ya cableado. No necesitaban ningún tipo de almacenamiento ya que no se podían introducir nuevas instrucciones ni guardar datos, lo que hacía muy limitado su uso. Las calculadoras de sobremesa o las computadoras usadas en la segunda guerra mundial para encriptar y desencriptar mensajes son ejemplos de este tipo de equipos con programas fijos.

La nueva idea de "programa almacenado" aparece a partir de la concepción de la *máquina universal de Turing* [7] que permite guardar en una memoria RAM un conjunto de instrucciones (un programa) y datos; y además dispone de mecanismos de entrada y salida. A finales de la década de 1940, gracias al trabajo de Turing [23], von Neumann y de Goldstine [24], la confianza en los resultados aportados por estos nuevos ordenadores aumentó exponencialmente y fue cuando la *estabilidad* de los algoritmos y el *condicionamiento* de los problemas se convirtió en una cuestión de suma importancia.

Más tarde, durante las décadas de 1950 y 1960, el estudio de la *estabilidad* fue más indagado por Wilkinson quien, en su libro [25], estudia la *estabilidad hacia atrás* (la cual está explicada en el capítulo 4). Décadas después, Jong y Bunch establecieron los cimientos de la definición de estabilidad, que fue mayormente descrita en el libro de Higham [12] en la última década del siglo XX y sobre la cual aún se apoyan muchos estudios a día de hoy.

Por último, en cuanto al *número de condición* ya introducido por Turing [23], la teoría fue dotada de mayor base en 1966 en el trabajo de Rice [21]; aunque también podemos encontrarnos con definiciones con una base geométrica en estudios como el de Bürgisser y Cucker [6]. Actualmente, nos referimos al trabajo de investigación más reciente de Blum, Shub, Cucker y Smale [4] en cuanto al número de condición.

1.2. Concepto

Para ayudar a la comprensión de las ideas de *condición* y *estabilidad* nombradas en el punto anterior, vamos a considerar una versión más sencilla del problema estudiado por Turing: dada una matriz cuadrada con determinante no nulo, $\mathbf{A}$, y un vector $\mathbf{b}$ queremos resolver el problema: $\mathbf{Ax} = \mathbf{b}$ para $\mathbf{x}$. Supongamos que los coeficientes de $\mathbf{A}$ son datos experimentales, tales que su precisión se ve limitada por los errores de los utensilios o programas utilizados para su medición. Suponiendo que los posteriores cálculos computacionales son exactos, podemos comparar la solución del sistema $\mathbf{Ax} = \mathbf{b}$ obtenido de la $\mathbf{A}$ experimental con la solución verdadera obtenida de los valores reales de la matriz $\mathbf{A}$. Con esto, ahora nos preguntamos en qué grado afecta el error de la medida de los valores de $\mathbf{A}$ a la solución obtenida $\mathbf{x}$. Para contestar esta cuestión se introdujeron los dos conceptos nombrados al inicio.

Condición

Dado un problema, su condición es una medida de la sensibilidad de la solución (valor de salida o *output*) al realizar pequeñas alteraciones en los datos de entrada (*input*). Este valor es conocido como el "número de condición" o *condition number*.

Hay varias formas de calcular la condición de una función, dependiendo de las características de esta. Por ejemplo, el número de condición o el condicionamiento de una matriz invertible, $\mathbf{A}$, fue dada por Turing y es $\kappa(\mathbf{A}) = ||\mathbf{A}|| \cdot ||\mathbf{A^{-1}}||$, con $|| \cdot ||$ una norma matricial. $\kappa(\mathbf{A}) = Cond(\mathbf{A})$ cuantifica la variación del error relativo de la

solución $\mathbf{x}$ con respecto al error relativo de la matriz de entrada $\mathbf{A}$. Si la solución de $\mathbf{Ax} = \mathbf{b}$ ha sido programada y calculada con una precisión fija, entonces $\kappa(\mathbf{A})$ debe ser comparado con dicha precisión. Cuanto más alto sea el número de condición de una matriz, menos precisa será la solución obtenida; es decir, menos cifras exactamente idénticas a la solución real tendrá. Por tanto, si el número de condición está próximo a 1 se dice que el problema está bien condicionado, mientras que si está muy alejado se dice que está mal condicionado. Esto último se traduce en que pequeñas modificaciones en los datos de entrada desencadenan grandes diferencias en los datos de salida. Algo similar sucede también si hacemos cambios en el vector b, como se muestra en el siguiente ejemplo.

Cabe destacar que la condición de un problema es una propiedad que no depende en absoluto del algoritmo utilizado en su resolución, ni siquiera de la existencia de uno.

Ejemplo 1.2.1.

Dado el sistema $\mathbf{Ax} = \mathbf{b}$ *donde*

$$A = \begin{pmatrix} 8 & 14 & 6 & 9 \\ 17 & 5 & 12 & 5 \\ 9 & 6 & 14 & 8 \\ 8 & 15 & 7 & 10 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Y el error asociado a los valores de b es

$$\epsilon_b = \begin{pmatrix} 0,1 \\ -0,1 \\ 0,1 \\ -0,1 \end{pmatrix}$$

Se calcula el número de condición de la matriz $\mathbf{A}$ *con ayuda de MATLAB:* $\kappa(\mathbf{A}) = norm(\mathbf{A}) \cdot norm(\mathbf{A^{-1}}) = 5496,2$

La solución real del problema es

$$x = \begin{pmatrix} 1,32 \\ -3,74 \\ -3,05 \\ 6,79 \end{pmatrix}$$

Mientras que la solución del sistema $\mathbf{Ax} = \mathbf{b} + \epsilon_b$ *es*

$$\hat{x} = \begin{pmatrix} 4,71 \\ -13,66 \\ -11,17 \\ 24,64 \end{pmatrix}$$

Como vemos, el resultado obtenido difiere en gran medida del resultado real. Esto era esperable ya que el número de condición de la matriz es muy grande (está muy alejado de 1).

Estabilidad

A diferencia del término anterior, la *estabilidad numérica* es una característica propia de los algoritmos.

Se dice que es la *fiabilidad* que tiene un algoritmo al usarse en la resolución de un problema. Describe cómo se propagan los errores desde los datos de entrada hasta la solución final a través del algoritmo. En un algoritmo *estable*, los errores implícitos en los datos de entrada se mantienen en un nivel moderado a medida que se corre el algoritmo; mientras que si es *inestable* los errores irán aumentando a medida que se computa el algoritmo.

Para un algoritmo $f(x)$, se calcula la estabilidad (respecto del error relativo) comparando los valores $\dfrac{x-(x+\epsilon)}{x}$ y $\dfrac{f(x)-(f(x-\epsilon))}{f(x)}$, donde x es el valor de entrada y ϵ su error asociado. Si dichas cantidades difieren en un grado muy pequeño, entonces decimos que el algoritmo es estable; y si existe una gran diferencia entre ellas, entonces el algoritmo es inestable.

Con la estabilidad, se busca medir el nivel de confianza en cuanto a la veracidad o exactitud del resultado obtenido tras aplicar el algoritmo en cálculos de aritmética finita.

A diferencia con el *número de condición*, existen varias definiciones de *estabilidad* y todas ellas válidas. En este trabajo haremos alusión a 3 tipos de estabilidad. Vienen descritas formalmente en el Capítulo 4, pero ahora vamos a ver una idea intuitiva de a lo que nos referimos con *estabilidad* de un algoritmo utilizando el sistema $\mathbf{Ax} = \mathbf{b}$ anterior para ejemplificarlo.

1. Un algoritmo **estable hacia adelante**, también denominada como *estabilidad débil* en [5], proporciona una solución próxima a la real para los datos dados. Es decir, para $\mathbf{Ax} = \mathbf{b}$ dicho algoritmo encuentra $\hat{\mathbf{x}} \sim \mathbf{x}$.

2. Un algoritmo con **estabilidad mixta**, también llamado *numéricamente estable* [13], suministra una solución casi exacta a un problema muy parecido al original. Es decir, para $\mathbf{Ax} = \mathbf{b}$ un algoritmo con estabilidad mixta encuentra una solución $\hat{\mathbf{x}} \sim \mathbf{y}$ donde $\mathbf{y}$ es la solución exacta del problema $\hat{\mathbf{A}}\mathbf{y} = \mathbf{b}$ para un $\hat{\mathbf{A}} \sim \mathbf{A}$. Esta aproximación es independiente del número de condición de $\mathbf{A}$.

3. Un algoritmo **estable hacia atrás**, también referida como *estabilidad fuerte* en [5], genera la respuesta exacta de un problema próximo al dado. Es decir, para $\mathbf{Ax} = \mathbf{b}$ este algoritmo encuentra $\hat{\mathbf{x}}$ de $\hat{\mathbf{A}}\hat{\mathbf{x}} = \mathbf{b}$ para un $\hat{\mathbf{A}} \sim \mathbf{A}$. Esta aproximación también es independiente de $\kappa(\mathbf{A})$.

Observación 1.2.2. *Cuando nos referimos a "grado muy pequeño", "gran diferencia" o que un valor sea "alto" o "bajo" queremos decir que dependiendo del problema o algoritmo serán adecuados unos valores para los errores u otros. Pero dentro de cada valor apropiado, debemos distinguir si este es más pequeño, más alto, etc. También, con " $\sim$ " nos referimos a un valor próximo al original. Y "próximo", de nuevo, dependerá de los datos y de las condiciones del problema y del algoritmo a implementar. Aunque todo esto será clarificado en el capitulo 4 4.*

Estas definiciones han asistido al desarrollo del análisis de algoritmos numéricos durante años, aún cuando los lenguajes informáticos existentes tenían una única precisión en coma flotante predefinida (simple, 32 bits o doble, 64 bits). Como sabemos, actualmente estos lenguajes están cambiando y van apareciendo muchos nuevos que nos permiten tener varias precisiones accesibles en un mismo programa que conceden al usuario la posibilidad de decidir la exactitud con la que desea operar. Si necesita un resultado con un grado alto de exactitud utilizará una precisión mayor, en contraposición a si precisa que los cálculos se hagan rápidamente que utilizará una precisión menor, renunciando a un cierto grado de exactitud. Un ejemplo en el que vemos este sistema de precisiones es en la resolución al problema 14 de los problemas de Smale [18] propuesto por Warwick Tucker [22]. Este problema trata, a grandes rasgos, sobre si las soluciones al sistema dinámico de ecuaciones diferenciales de Lorenz [20] coincide con las soluciones del atractor geométrico de Lorenz desccrito por Williams, Guckenheimer y Yorke [11]. W. Tucker respondió afirmativamente a esta pregunta, utilizando la *aritmética de intervalos* para demostrar su teoría. Dicha aritmética es un procedimiento computacional utilizado para establecer límites a los errores de redondeo y de medición en cálculos matemáticos para así obtener resultados más precisos.

A continuación se muestra un ejemplo con el fin de ayudar a la comprensión de lo que es la *estabilidad*.

Ejemplo 1.2.3. *Dado el sistema anterior* $\mathbf{Ax} = \mathbf{b}$ *vamos a resolverlo mediante tres algoritmos distintos: el método QR, un método interno de MATLAB y el método de la inversa; para luego comparar los resultados obtenidos por cada uno así como sus errores y el número de condición asociados.*

- *El método QR: consiste en dado un sistema* $\mathbf{Ax} = \mathbf{b}$ *obtener la factorización* QR *de* A *y operar:* $QR \cdot x = b \;\Rightarrow\; R \cdot x = Q^T \cdot b \;\Rightarrow\; x = R^{-1} \cdot Q^T \cdot b$.
- *El método interno de MATLAB* [1]*: se va a utilizar la función* **mldivide()**[2] *programada en dicho software.*
- *El método de la inversa: consiste en obtener la matriz inversa de* A *y calcular* $A \cdot x = b \;\Rightarrow\; x = A^{-1} \cdot b$.

A partir de la función **randn()** *proporcionada por MATLAB, hemos creado un método que genera de manera aleatoria la matriz* A *y el vector* b *en las condiciones establecidas, para luego ser resuelto por los 3 métodos.*

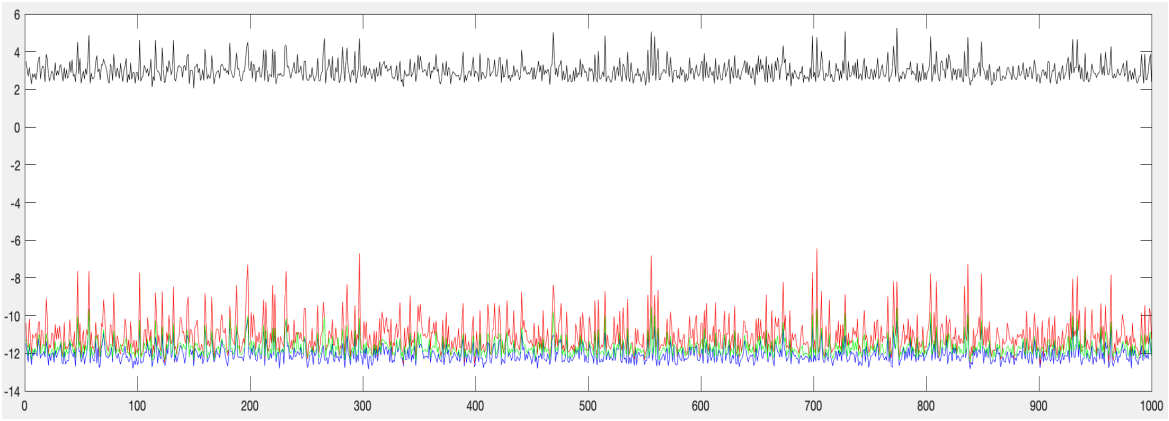


Gráfico 1

[1] Versión R2021a

[2] https://es.mathworks.com/help/matlab/ref/mldivide.html

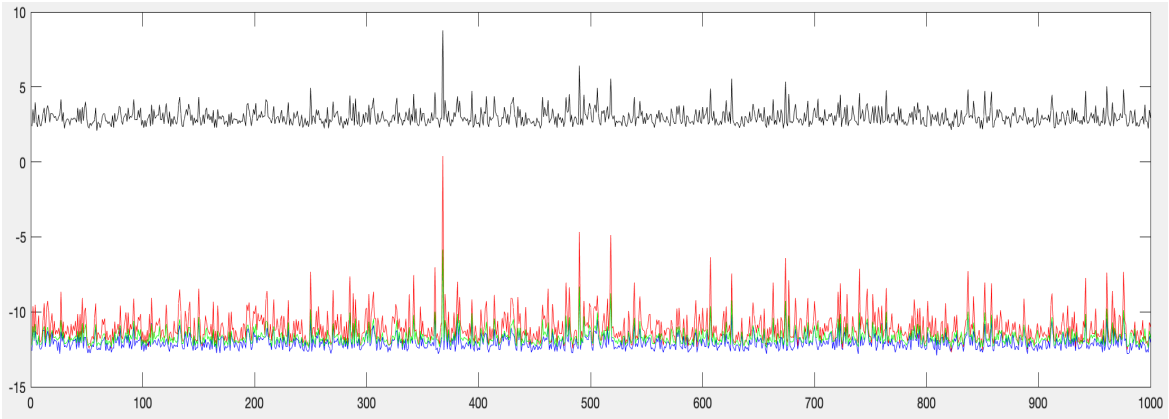


Gráfico 2

Figura 1.1: Gráficos de 2 ejecuciones del programa en MATLAB.

En cada uno de los gráficos vienen representadas mil iteraciones del programa (eje x); es decir, se han generado y resuelto mil problemas distintos (1.2.4). En el eje y viene representado el orden, en base 10*, del error asociado a cada resultado de cada uno de los métodos, representados en 3 colores. En rojo se ha obtenido el error asociado al método QR, en azul el asociado al método* mldivide() *y en verde el asociado al método de la inversa. Por último, la línea negra representa el número de condición de la matriz A asociado a cada problema.*

Al observar ambas figuras, se puede deducir que el método QR al tener un mayor rango de error (alcanzando incluso un pico de aproximadamente 10^1 *al rededor de la iteración 370 en el Gráfico 2) es un método menos estable que los otros dos. Se observa también que el método más regular es el interno de MATLAB, con lo que es el más estable de los tres.*

Cabe destacar también que en los picos en los que todos los métodos tienen un error mayor, el número de condición de A asociado a ese problema es también muy alto. Por ello, este valor es una buena herramienta para conocer de antemano si, sea cual sea el algoritmo que se vaya a utilizar, se obtendrá una mayor o menor precisión en el resultado del problema.

Observación 1.2.4.

1. *Al estar generándose los números de manera aleatoria, es posible que algún problema esté repetido, aunque es poco probable debido a las dimensiones escogidas para A y b:* $dim(A) = 200 \times 200$, $dim(b) = 200 \times 1$.

2. *Al comienzo de este capítulo se ha indicado que la matriz A es cuadrada e invertible; es decir, que aunque el programa podría generar una matriz cuyos*

elementos son todos cero, este caso es muy poco probable y además no tendría sentido resolver dicho problema.

Capítulo 2

El número de condición en el error relativo

2.1. El número de condición como error relativo

A partir de este momento, nos restringimos al estudio del número de condición y de la estabilidad a problemas definidos por una función f que va de un conjunto real a otro, ambos de dimensión uno. Comprobamos también la existencia de una función distancia tanto en el dominio como en el codominio de f, definida como sigue:

Definición 2.1.1. *(Función distancia). Dado un subconjunto $D \subseteq \mathbb{R}$, la aplicación $dist_D : D \times D \longrightarrow [0, \infty]$ es una* **función distancia** *si:*

(i) $dist(a, b) = 0 \Longleftrightarrow a = b$,

(ii) *es simétrica:* $dist(a, b) = dist(b, a) \quad \forall a, b \in D$,

(iii) *cumple la desigualdad triangular:* $dist(a, c) \leq dist(a, b) + dist(b, c) \quad \forall a, b, c \in D$.

Ahora podemos definir el número de condición:

Definición 2.1.2. *(Número de condición). Dada una función $f : S \longrightarrow T$, donde $S \subseteq \mathbb{R}$ y $T \subseteq \mathbb{R}$ y dos funciones distancia $dist_S$ y $dist_T$ definidas sobre S y T respectivamente,* **el número de condición** *$\kappa(f, x)$ de f en $x \in S$ es:*

$$\kappa(f, x) = \lim_{\epsilon \to 0} \sup_{y \in S,\ 0 < dist_S(x,y) \leq \epsilon} \frac{dist_T(f(x), f(y))}{dist_S(x, y)} \in [0, \infty]. \tag{2.1}$$

Si x es un punto aislado en la topología de S, entonces se tiene $\kappa(f, x) = 0$.

En la definición, el límite superior siempre estará definido, ya que ∞ es válido como resultado.

Con el número de condición medimos cuánto cambia el resultado de un problema, $f(x)$ comparado con $f(y)$, al hacer pequeñas variaciones en los datos de entrada, x e y. Por ello, cuanto más pequeño sea el número de condición, más exacto será el resultado del problema. Y viceversa, cuanto mayor sea el número de condición, mayor será la diferencia entre el dato obtenido y el real, y por tanto peor será el resultado que genere.

Se define también $\tilde{\kappa} = 1 + \kappa$ que será utilizado más adelante.

Una de las propiedades más importantes del número de condición es que se trata de un *invariante geométrico*; es decir, que depende únicamente de la función f, pero no del algoritmo usado para realizar los cálculos. Sí que se puede decir que hay una dependencia explícita en la elección de las funciones $dist_S$ y $dist_T$.

Los lemas y proposiciones de este capítulo están por lo general obtenidos, particularizando al caso unidimensional, los resultados de [2].

El número de condición también satisface:

Lema 2.1.3. *Si $f = g \circ h$ es una composición de funciones, entonces*

$$\tilde{\kappa}(f, x) \leq \tilde{\kappa}(g, h(x))\tilde{\kappa}(h, x) \tag{2.2}$$

Demostración. Sean $h : S \longrightarrow \mathbb{R}$, $g : T \longrightarrow \mathbb{R}$ tales que $h(S) \subseteq T \subseteq \mathbb{R}$. Tomamos un $x \in S$ tal que $\kappa(h, x)$, $\kappa(g, h(x)) < \infty$. Entonces, para todo $\delta \in (0, 1)$, existen ϵ', $\epsilon > 0$ tales que:

$$\sup_{z \in T,\ 0 < dist(h(x), z) \leq \epsilon} \frac{dist(g(h(x)), g(z))}{dist(h(x), z)} \leq \kappa(g, h(x)) + \delta, \text{ y}$$

$$\sup_{y \in S,\ 0 < dist(x,y) \leq \epsilon'} \frac{dist(h(x), h(y))}{dist(x, y)} \leq \kappa(h, x) + \epsilon$$

Notemos que $\epsilon', \epsilon > 0$ existen siempre por cómo están definidas las funciones $\kappa(h, x)$ y $\kappa(g, h(x))$, y además podemos elegir estos parámetros de manera que $\epsilon' < \epsilon < \delta < 1$. Por lo tanto, para $y \in S$, y $0 < dist(x, y) \leq \dfrac{\epsilon'}{2\tilde{\kappa}(h, x)}$ se tiene:

$$dist(h(x), h(y)) \leq (\kappa(h, x) + \epsilon)\ dist(x, y) \leq \frac{\kappa(h, x) + \epsilon}{2\tilde{\kappa}(h, x)}\epsilon' < \epsilon$$

Esto implica que

$$\begin{aligned} dist(g(h(x)), g(h(y))) &\leq\ (\kappa(g, h(x)) + \delta)\ dist(h(x), h(y)) \\ &\leq (\kappa(g, h(x)) + \delta)(\kappa(h, x) + \epsilon)\ dist(x, y) \end{aligned}$$

En consecuencia, hemos probado que

$$0 < dist(x, y) \leq \frac{\epsilon'}{2\tilde{\kappa}(h, x)} \Longrightarrow \frac{dist(f(x), f(y))}{dist(x, y)} \leq (\kappa(g, h(x)) + \delta)(\kappa(h, x) + \epsilon),$$

Que es lo mismo que: $\kappa(f,x) \leq (\kappa(g,h(x)) + \delta)(\kappa(h,x) + \epsilon)$. Como hemos definido al principio que era válido $\forall \delta \in (0,1)$ de manera directa se obtiene la desigualdad 2.2 que queríamos demostrar.
Por último, se añade que 2.2 es válida también para $\tilde{\kappa}(h,x) = \infty$ o para $\tilde{\kappa}(g,h(x)) = \infty$, ya que $\tilde{\kappa}(g,h(x)) \geq 1$ y $\tilde{\kappa}(h,x) \geq 1$. □

La elección de la función *dist* en la definición 2.1.2 depende del tipo de problema que queramos resolver. En la rama matemática de la aritmética real, se sustituyen las grandes operaciones entre números reales por operaciones en *coma flotante*, lo que deriva en pequeños errores relativos. Por esta razón, a partir de ahora se va a tomar una métrica que mida estos errores.

Lo que se suele conocer como *error relativo* en análisis numérico viene dado por: $\frac{|x-\tilde{x}|}{|x|}$, donde $\tilde{x} \in \mathbb{R}$ es una aproximación y $x \in \mathbb{R}$ es el valor exacto (real). Pero esta formulación no define una métrica en $\mathbb{R}$ ya que las propiedades de *simetría* y la *desigualdad triangular*, descritas al comienzo de este capítulo, no siempre se cumplen en este espacio. Por tanto, debemos utilizar otro tipo de métrica, y esta será la *métrica del error relativo* .

J.D. Pryce [19], a mediados de los años 80, fue quien presentó esta nueva métrica tras haberse basado en los estudios realizados unos años antes por Frank W.J. Olver [17]. Dicha métrica define una distancia en $\mathbb{R}$ determinada por la longitud de la curva más corta que une los puntos x e y, tomando como longitud de curva la dada por la métrica de Riemman, descrita en la Definición 2.1.5.

La métrica Riemannniana [9] funciona del siguiente modo: sea f una función derivable y continua en el intervalo $[x,y]$ se calcula la distancia entre x e y como la longitud del arco que une $f(x)$ con $f(y)$ mediante la integral $\int_x^y |\alpha'(t)|dt$, siendo α una parametrización de la curva que une x e y. Como trabajamos todo el tiempo en el espacio de los números reales, toda curva que tomemos será regular y simple, lo que permite afirmar que la parametrización que escojamos no depende en absoluto de la curva.

Observación 2.1.4.

1. *Se hace notar que existe un caso general en la teoría de Riemann sobre el estudio de la geometría en variedades diferenciales de dimensión n. Sin embargo, excede de los conocimientos requeridos para la comprensión de esta tarea y por ello no vamos a profundizar más.*

2. *De hecho, la métrica coordenada a coordenada se puede definir en* $\mathbb{R}^d$ *para*

cualquier $d \geq 1$. Pero se ha restringido el estudio de este trabajo a $\mathbb{R}$ por ser, de otra manera, demasiado extenso.

La métrica del error relativo induce una topología algo peculiar, como vemos en la siguiente definición:

Definición 2.1.5. *(Métrica del error relativo). La topología asociada a la* **métrica del error relativo** *en $\mathbb{R}$ está definida por 3 componentes conexas U_j en $\mathbb{R}$. La componente U_0 contiene un único punto: $\{0\}$, mientras que U_1 y U_2 contendrán a los intervalos $(-\infty, 0)$, $(0, +\infty)$ respectivamente, dependiendo del signo del número real al que estén asociados.*
La norma en $\mathbb{R}$ viene definida por $|x|$, $x \in \mathbb{R}$ y la distancia entre dos puntos x e y es:

- $dist(x, y) = 0 \quad si \quad x = y,$
- $dist(x, y) = \infty \quad si \quad sign(x) \neq sign(y),$
- $dist(x, y) = \displaystyle\int_0^1 |x-y|_{tx+(1-t)y}\, dt = \int_0^1 \frac{|x-y|}{|tx+(1-t)y|} dt = \left|\log \frac{x}{y}\right| \quad$ *en otro caso.*

Así, esta definición de función *distancia* satisface todas las condiciones expuestas al comienzo de este capítulo, incluyendo la desigualdad triangular 2.1.1.

Al tener definida dicha métrica en $\mathbb{R}$, esta definición de distancia no es más que la medida de la recta que une dos puntos, que es un caso muy sencillo de lo que en Teoría global de superficies se llama *geodésica*; es decir, la curva de menor longitud que une dichos puntos. Así que el siguiente lema nos puede servir de ayuda.

Lema 2.1.6. *Sea $f : S \longrightarrow \mathbb{R}$ la función de la definición 2.1.2. Sean $x, y \in S$ y sea $\gamma(t) \subseteq S$ el segmento que une x e y, y supongamos que cumple $\kappa(f, t) < C \quad \forall t \in \gamma$, para algún $C \in \mathbb{R}$. Entonces:*

$$dist(f(x), f(y)) \leq C\ dist(x, y)$$

Observación 2.1.7. *La función f va a ser siempre continua en las condiciones del Lema 2.1.6, ya que al pedir que $\kappa(f, t) < C \quad \forall t \in \gamma$, $C \in \mathbb{R}$ no tomamos en ningún caso $\kappa(f, t) = \infty$ y; por tanto, si $x \longrightarrow y$ implica que $f(x) \longrightarrow f(y)$.*

Demostración Lema 2.1.6. Sea $\gamma : [0, d] \longrightarrow \mathbb{R}$ el segmento anterior que cumple $dist(\gamma(a), \gamma(b)) = b - a$ para $0 \leq a \leq b \leq d$ y sea $\phi(t) = dist(f(x), f(\gamma(t)))$ que es continua porque $\kappa(f, t) < \infty$. Dado $\delta > 0$, construimos el conjunto:

$$S = \{t : \phi(t) \leq (C + \delta)t\} \subseteq [0, d]$$

e identificamos $\sigma = sup\ S$, que existe por el axioma del supremo y además $\sigma \in S$ por ser S un conjunto cerrado.

Supongamos que $\sigma < d$, entonces $\forall \epsilon > 0$ suficientemente pequeño se cumple

$$\phi(\sigma + \epsilon) \leq \phi(\sigma) + dist(f(\gamma(\sigma)), f(\gamma(\sigma + \epsilon)))$$

y de la definición de S se obtiene directamente, $\phi(\sigma) \leq (C + \delta)\sigma$.
Para la segunda parte de la desigualdad tomamos $\epsilon > 0$, teniendo

$$dist(f(\gamma(\sigma)), f(\gamma(\sigma+\epsilon))) \leq \Big(C+\frac{\delta}{2}\Big) dist(\gamma(\sigma), \gamma(\sigma+\epsilon)) = \Big(C+\frac{\delta}{2}\Big)(\sigma+\epsilon-\sigma) < (C+\delta)\epsilon$$

En consecuencia, para un ϵ suficientemente pequeño, $\phi(\sigma + \epsilon) < (C + \delta)(\sigma + \epsilon)$ y esto implicaría que $t = \sigma + \epsilon \in S$ lo que contradice la definición de σ como supremo. Por lo tanto, se concluye que $\sigma = d$, lo que prueba el lema. □

Teniendo en cuenta la definición anterior, se va a concretar una fórmula sencilla para el *número de condición* en caso de que la función sea derivable.

Proposición 2.1.8. *(El número de condición del error relativo). Sea $f : \mathbb{R} \to \mathbb{R}$ una aplicación derivable y sean U_j los componentes de la definición 2.1.5. Sea $\kappa(f, x)$ el número de condición descrito en la definición 2.1.2 con la métrica del error relativo en el dominio y el codominio. Entonces:*

(*i*) *si existe un entorno abierto N_x tal que $f(N_x)$ está contenido en una componente conexa de $\mathbb{R}$, entonces*

$$\kappa(f, x) = |x| \cdot \frac{|f'(x)|}{|f(x)|}. \tag{2.3}$$

(*ii*) *en otro caso* $\kappa(f, x) = \infty$

Demostración. Sea f la aplicación de la proposición y sean $x, y \in \mathbb{R} \setminus \{0\}$. Calculamos:

$$\kappa(f, x) = \lim_{y \to x} \frac{dist(f(y), f(x))}{dist(y, x)} = \lim_{y \to x} \frac{\log \frac{f(y)}{f(x)}}{\log \frac{y}{x}} = \frac{0}{0} \quad \text{(Indeterminación)}$$

Aplicamos L'Hôpital respecto de y

$$\lim_{y \to x} \frac{\frac{f'(y)}{f(y)}}{\frac{1}{y}} = \frac{x \cdot f'(x)}{f(x)}$$

y añadimos valores absolutos

$$\lim_{y \to x} \frac{\left|\frac{f'(y)}{f(y)}\right|}{|\frac{1}{y}|} = \left|\frac{x \cdot f'(x)}{f(x)}\right|$$

Pero, que y tienda a x es lo mismo que decir que la distancia entre x e y es nula. Por tanto se puede escribir también como

$$\lim_{dist(x,y)\to 0} \frac{\left|\frac{f'(y)}{f(y)}\right|}{\frac{1}{|y|}} = \left|\frac{x\cdot f'(x)}{f(x)}\right|$$

Luego, aplicando la definición de límite, dado un $\delta > 0$, $\exists \epsilon > 0$ tal que $dist(x,y) \leq \epsilon$

$$\left|\frac{|\log\frac{f(y)}{f(x)}|}{|\log\frac{y}{x}|} - \left|\frac{x\cdot f'(x)}{f(x)}\right|\right| < \delta$$

Es decir,

$$\lim_{\epsilon\to 0} \sup_{dist(x,y)\leq\epsilon} \left|\frac{|\log\frac{f(y)}{f(x)}|}{|\log\frac{y}{x}|} - \left|\frac{x\cdot f'(x)}{f(x)}\right|\right| = 0$$

quedando

$$\kappa(f,x) = \lim_{\epsilon\to 0} \sup_{dist(x,y)\leq\epsilon} \frac{|\log\frac{f(y)}{f(x)}|}{|\log\frac{y}{x}|} = \left|\frac{x\cdot f'(x)}{f(x)}\right|$$

como queríamos demostrar. □

A continuación, se destacan algunos casos particulares derivados de la fórmula 2.3:

Observación 2.1.9.

- *Si* $x = 0 \implies \kappa(f,x) = 0$ *sin importar el valor de* $f'(x)$ *ni de* $f(x)$.
- *Si* $x \neq 0$ *y* $f'(x) = 0$ *en una bola abierta de* $x \implies \kappa(f,x) = 0$.
- *Si* $x \neq 0$, $f(x) = 0$, *pero* $f'(x)$ *no es constantemente igual a* 0 *en una bola abierta de centro* $x \implies \kappa(f,x) = \infty$ *incluso si* $f'(x) = 0$.

Para finalizar, se calculan los números de condición de algunas funciones utilizando la Proposición 2.1.8.

Ejemplos 2.1.10.

1. $\mathbf{f(x) = x^2}$
   $$\kappa(f,x) = \frac{|x|\cdot|2x|}{|x^2|} = \frac{2x^2}{x^2} = 2 \quad \forall x$$
2. $\mathbf{f(x) = x^3 + 1}$
   $$\kappa(f,x) = \frac{|x|\cdot|3x^2|}{|x^3+1|} = \frac{3|x^3|}{|x^3+1|} = \begin{cases} a \in [0,3) & si \quad x \in \mathbb{R}\setminus\{\infty\} \\ 3 & si \quad x \longrightarrow \infty \end{cases}$$

3. $\mathbf{f(x) = e^x}$

$\kappa(f,x) = \dfrac{|x| \cdot |e^x|}{|e^x|} = |x| \ \in \ [0, \infty]$

4. $\mathbf{f(x) = cos(x)}$

$\kappa(f,x) = \dfrac{|x| \cdot |-sen(x)|}{|cos(x)|} = \dfrac{|x \cdot sen(x)|}{|cos(x)|}$ *Vamos a ver los diferentes casos:*

- *Si* $x = 2k\pi$ *para* $k \in \mathbb{Z}$ *entonces* $\kappa(f,x) = 0$
- *Si* $x = \dfrac{\pi}{2} + k\pi$ *para* $k \in \mathbb{Z}$ *entonces* $\kappa(f,x) = \dfrac{\pi/2 + k\pi}{0} = \infty$
- *Si* x *es distinto a cualquiera de los valores anteriores, entonces* $\kappa(f,x) \in (0,x]$

Capítulo 3

Modelo computacional de algoritmos numéricos

De manera informal, se conoce un algoritmo como una secuencia de instrucciones (operaciones definidas, ordenadas y finitas) programables en cualquier lenguaje de programación, como *C, Matlab* o *Python*. Sin embargo, la definición formal que se va a tomar en este trabajo es la del modelo *BSS* propuesto por *Leonore Blum, Michael Shub* y *Stephen Smale* [3] y más tarde desarrollado por esos académicos junto a *Felipe Cucker* [8] en 1998 [4]. En su libro, los autores tienen como objetivo desarrollar una base más sólida para los modelos de computación en matemática discreta. Declaran que los modelos clásicos, como por ejemplo *la máquina de Turing*, no son del todo adecuados para la realización de cálculos en problemas de algunas áreas de la computación. Esto se debe a que el arquetipo utilizado hasta entonces tenía una base binaria con un límite muy ajustado de bits que podían ser guardados para cada cálculo [1]. En cambio, el modelo Blum-Shub-Smale, basado en el modelo de la máquina de Turing, permitía la computación exacta de números reales. Es por esto que la máquina *BSS* es también conocida como "*la máquina de Turing de números reales* ".

3.1. Aritmética en coma flotante estándar

Un sistema numérico en coma flotante $\mathbb{F} \subseteq \mathbb{R}$ con base 2, es un subconjunto de números reales de la forma

$$\pm m \cdot 2^{e-t}$$

[1] Naturalmente, este límite sigue existiendo en la actualidad, pero es mucho más grande.

donde $t \in \mathbb{Z},\ t > 2$ es la *precisión* y $e \in \mathbb{Z},\ e_{min} \leq e \leq e_{max}$ está acotado. Es más, la mantisa (la parte decimal) $m \in \mathbb{Z}$ puede ser 0 o satisfacer $2^{t-1} \leq m \leq 2^t - 1$, asegurando una representación única de un valor real. Dicho con otras palabras, $x \in \mathbb{F}$ si $x = 0$ o si x es de la forma $x = \pm 2^e \cdot [0.a_1 \dots a_t]_2$ para e en un cierto rango de valores, y $a_1, \dots, a_t \in \{0, 1\},\ a_1 \neq 0$. La *unidad de redondeo* es $u = 2^{-t}$.

Para conseguir una teoría que sea lo suficientemente versátil como para poder probar teoremas con ella, lo primero que se debe hacer es ampliar las cotas de e. En consecuencia, desde ahora tomaremos $e_{min} = -\infty$ y $e_{max} = +\infty$, definiendo un sistema de coma flotante cuya unidad de redondeo puede ser u o la precisión t. A este nuevo sistema lo denotaremos por $\mathbb{F}_u$.

Definimos la aplicación $fl_u : \mathbb{R} \longrightarrow \mathbb{F}_u$ que lleva $x \in \mathbb{R}$ al valor de $\mathbb{F}_u$ que más se le aproxima en valor absoluto. Dicha aplicación estará siempre definida para $0 < u < \frac{1}{4}$. Con esta definición, podemos afirmar que se cumplen:

- $\forall\ x \in \mathbb{R},\ fl_u(x) \in \mathbb{F}_u$.
- $\forall\ x \in \mathbb{R},\ fl_u(x) = x(1 + \delta)$ para algún real $|\delta| \leq u$.
- $\forall\ x \in \mathbb{F}_u,\ fl_u(x) = x$.
- $\forall\ x \in \mathbb{R},\ fl_u(-x) = -fl_u(x)$.
- si $u' < u$ y $fl_u(x) = x,\ \Longrightarrow x = fl_{u'}(x)$.
- si $x, y \in \mathbb{R}$ y $x \leq y,\ \Longrightarrow fl_u(x) \leq fl_u(y)$.
- Toda operación $* \in \{+_{\mathbb{R}}, -_{\mathbb{R}}, \cdot_{\mathbb{R}}, \div_{\mathbb{R}}\}$, tiene su operación correspondiente en coma flotante definida como: $\hat{*} : \mathbb{F}_u \times \mathbb{F}_u \longrightarrow \mathbb{F}_u$ tal que $\forall x, y \in \mathbb{F}_u$ se verifica:

  $(x\ \hat{*}\ y) = (x * y)(1 + \delta)$ para algún número real $|\delta| \leq u$,

  con la única excepción de que una división entre cero genera un resultado de NaN o $\pm\infty$.

3.2. El modelo de computación *BSS*

Una máquina BSS $M(\bar{I}, \bar{O}, \bar{S}, G)$ [10], en base canónica, se compone por un espacio de entrada $\bar{I}$, un espacio de salida $\bar{O}$, un espacio de estado $\bar{S}$ y un grafo dirigido, conexo y finito $G = (V, E)$ donde V son los nodos y E las aristas. El espacio de estado, $\bar{S}$, se puede entender como una "memoria interna" que puede crecer a placer: un elemento

$\bar{s} \in \bar{S}$ es una secuencia bi-infinita de números reales $(..., s_{-2}, s_{-1}, s_0, s_1, s_2, ...)$ donde todos son igual a 0 excepto un número finito de ellos.

Los nodos $\bar{v} \in V$ están enumerados de 0 a N y cada uno está ligado a una cierta instrucción del programa; es decir, existen $N + 1$ nodos o instrucciones. Estos nodos pueden ser de 5 tipos, que a su vez describen el funcionamiento de la máquina:

1. *Entrada o Input* : Este nodo es único y se encarga de guardar la información de entrada de la máquina. No tiene ninguna arista de llegada y solamente una arista de salida. Su función es coger los datos de entrada $\bar{i} \in \bar{I}$ y mandarlos a un nodo de estado $\bar{s} \in \bar{S}$.

2. *Salida u Output* : Es el único nodo sin aristas de salida, únicamente tiene aristas de entrada. Cuando este nodo es alcanzado, el programa termina y la información de estado $\bar{s} \in \bar{S}$ que ha llegado del nodo anterior, pasa a ser la información de salida $\bar{o} \in \bar{O}$.

3. *Computación* : Este tipo de nodo tiene exactamente una arista de salida y se ocupa de ejecutar una operación en el estado $\bar{s} \in \bar{S}$ en el que se encuentra en ese momento transformándolo en otro distinto $\hat{s} \in \bar{S}$. Con *operación* nos podemos referir a una de las siguientes acciones: ejecutar una operación básica $\{+, -, \times, /\}$ a dos elementos de $\bar{s}$, copiar un número en otro lugar de la memoria o añadir una constante interna de la máquina. Todas las operaciones producen resultados exactos.

4. *Ramificación o Branch* : Estos tienen dos aristas de salida. Si $s_0 > 0$ del estado en el que se encuentra, entonces la información toma la primera arista de salida; en otro caso, toma la segunda.

5. *"El quinto nodo"* : Se ocupan de trasladar los componentes del espacio de estado $\bar{S}$. Todos los elementos de $\bar{s} = (..., s_{-2}, s_{-1}, s_0, s_1, s_2, ...)$ se desplazan un lugar hacia la izquierda o hacia la derecha.

Veamos un ejemplo de cómo funcionaría una máquina BSS.

Ejemplo 3.2.1. *Como se ha comentado anteriormente, se numeran los nodos de* 0 *a* N*, siendo* $v_0 \in V$ *el nodo de entrada. Sea* $N' \in \mathbb{N}$*, los nodos de salida serán:* $v_{N+1}, v_{N+2}, v_{N+3}, ..., v_{N+N'}$*. El resto de nodos son numerados de* 1 *a* N*. A la secuencia de estados* $(0, v_0, s(0)), (1, v_{i_1}, s(1)), ...$ *se le llama "secuencia de estados de computación exacta" de entrada* $\bar{i}$ *si los datos de entrada son* $s(0) = \bar{i}$*; y cada* $(j, v_{i_j}, s(j)) \in (\mathbb{N} \times V \times \bar{S})$ *tal que* $(v_0, v_{i_1}) \in E$*,* $(v_{1_j}, v_{1_{j+1}}) \in E \ \forall j$*. Entonces se escoge la ramificación correcta. En los nodos de ramificación es posible tomar uno*

de los dos caminos de salida, lo cual se puede entender como una instrucción de "si no se cumple cierta condición, ve a la línea k". Esto permite que haya bucles y condicionales en el grafo. El programa termina para los datos de entrada $\bar{i}$ si su secuencia de estado de computación exacta asociada es finita. El quinto nodo es útil para acceder a una posición arbitraria de la secuencia; es decir, pensando en $\bar{S}$ como una memoria interna de la máquina y en sus elementos $\bar{s}$ como una colección finita de variables con algún valor asignado, el quinto nodo nos permite disponer de cualquier variable de la memoria.

Podemos describir el funcionamiento de una máquina BSS o bien de una forma gráfica, representando el grafo de computación G del programa o bien mediante un pseudocódigo adecuado que permita, por ejemplo, el cómputo exacto de funciones racionales con números reales. Otras funciones como $\sqrt{x}$, $\log x$ o $\sin x$ no pueden describirse en un solo pseudocódigo, sino que necesitan subalgoritmos implementados aparte.

Aún así, no se puede asegurar que una máquina BSS pueda finalizar sus cómputos con cualquier información de entrada que le suministremos; así como tampoco se puede garantizar que el tiempo de ejecución vaya a estar acotado por alguna constante independiente del input.

Observación 3.2.2. *De hecho, son en estas dos cuestiones anteriores en las que se ha invertido mucho tiempo y esfuerzo a la hora de estudiar las máquinas BSS. No van a ser tratadas en este trabajo, pero hay que tener presente que una máquina de computación no es una función definida en $\bar{I}$ porque puede que no se obtenga ningún resultado de salida para algún input; es decir, que el programa siga ejecutándose de manera infinita. Por ello, al conjunto de puntos $\Omega \subseteq \bar{I}$ los cuales van a generar un output cuando se introduzcan en la máquina se le conoce como* halting set *(en español* conjunto de parada*) porque va a interrumpir el programa en un número finito de cálculos obteniendo un resultado.*

También debemos tener en cuenta que el resultado obtenido en un cálculo puede no estar definido para algunos valores del input debido a la aparición de una división entre 0. Para que esto no ocurra, antes de un nodo de computación, se incluye un nodo de ramificación que revisa si hay algún valor con denominador nulo. Si encuentra alguno, la máquina repite el cómputo de ese nodo infinitamente para no obtener ningún output para ese input.

Una de las principales características innovadoras del modelo Blum-Shub-Smale a diferencia del antiguo modelo utilizado en las máquinas de Turing, es que en el primero

se realizan cálculos con números reales exactos almacenados (de manera arbitraria) en una memoria RAM con precisión "infinita", mientras que en la segunda se utiliza un lenguaje binario (y por tanto, finito). Es por este motivo que algunos científicos critican este modelo computacional, al no ser del todo correcto en cuanto a la "precisión" en los resultados [14]. Aún así, está claro que ambos modelos tienen límites en cuanto al número de operaciones y representaciones de números reales.

En concreto, pongamos el caso mencionado anteriormente en el que obtenemos un dato igual a 0. En coma flotante, dicho número podría estar representado, si es por ejemplo el resultado de un cálculo, como 1×2^{-52}, mientras que en el modelo BSS tendríamos un cero redondo. Esto lleva a que en sucesivas operaciones, con el segundo de los valores tengamos un bucle infinito que no produce ningún dato de salida, mientras que con el otro sí obtendríamos un output el cual no sería acertado. Así, el conjunto de parada de una máquina puede cambiar si hablamos de una máquina BSS o de su versión en coma flotante.

3.2.1. Cálculos aproximados

El análisis numérico estudia el comportamiento de los algoritmos cuando se dan errores de redondeo causados, entre otras cosas, por la representación aritmética en coma flotante. Recientemente [15] [8], el modelo BSS ha sido utilizado para formalizar conceptos intuitivos que surgen en el estudio de algoritmos desde un enfoque de aproximaciones computacionales. Lo que se hace es definir un *algoritmo numérico* como el código de una máquina BSS cuyos cálculos son ejecutados de forma exacta, añadiéndole su parte en coma flotante.

Definición 3.2.3. *(Computación aproximada fuerte). [15] Sea $M(\bar{I}, \bar{O}, \bar{S}, G)$ una máquina BSS y sea $0 < u < \frac{1}{4}$ una unidad de redondeo o, también llamado, epsilon máquina. La $u-$***computación fuerte** *de M en $\bar{i} \in \bar{I}$ es exactamente la secuencia de computación $(j, v_{i_j}, \hat{s}(j))$ que se obtiene de la máquina BSS $M^u(\bar{I}, \bar{O}, \bar{S}, \hat{G})$, con $\hat{G}$ el grafo obtenido a partir de G sustituyendo:*

- *operaciones de números reales $x * y$, con $* \in \{+_{\mathbb{R}}, -_{\mathbb{R}}, \cdot_{\mathbb{R}}, \div_{\mathbb{R}}\}$,*

  por su parte en coma flotante $x \hat{*} y$

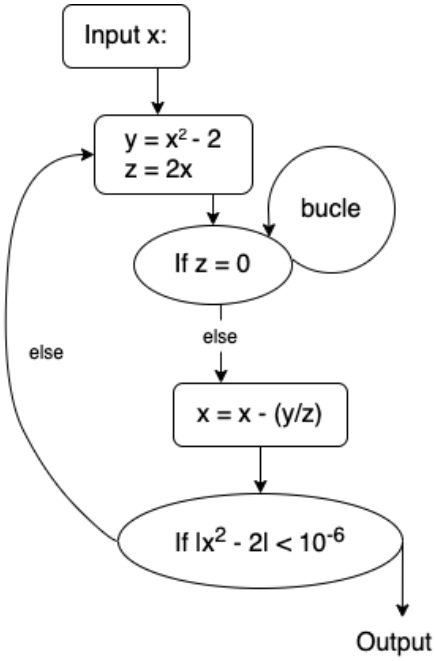
- *constantes $c \in \mathbb{R}$ por $fl_u(c)$, y*

- *datos de entrada o de salida $x \in \mathbb{R}$ por $fl_u(x)$*

Para cualquier valor de precisión $t > 2$ (equivalentemente $u < \frac{1}{4}$) la máquina M^u tiene

acceso a u y a t como constantes internas, lo que significa que los datos de entrada serán $(\bar{i}, u, t)$. El conjunto de parada Ω^u está formado por todos los inputs para los que la $u-$computación fuerte termina. Si $\bar{i} \in \Omega^u$, el valor de salida es $M^u(\bar{i})$ de la $u-$computación fuerte. Así, podemos describir M^u como una función $M^u : \Omega^u \longrightarrow \bar{O}$; y M^u es un algoritmo numérico.

Ejemplo 3.2.4. *Vamos a ilustrar el método de Newton para $f(x) = x^2 - 2 = 0$ con el modelo BSS.*
El método de Newton es un algoritmo iterativo que sirve para encontrar aproximaciones de los ceros de una función a partir de un valor conocido "lo suficientemente cercano" a estos. $\hat{f}(x)$: $x_{n+1} = x_n - \dfrac{f(x)}{f'(x)}$ es el algoritmo que sigue, donde $f'(x)$ denota la derivada de $f(x)$.



(a) Esquema método de Newton

```
ALGORITMO Newton
Variables: dato_inicial (x_0),
           tolerancia (10^-6),
           cant máx de iteraciones (N)
   PASO 1: tomamos i=1
   PASO 2: mientras i<=N seguimos los PASOS 3 a 5
   PASO 3: hacer x = x_0 - (y(x_0)/z(x_0))
   PASO 4: si |x-x_0| < 10^-6 entonces mostrar
           "Se ha encontrado solución: x"
           PARAR
   PASO 5: si no x = x_0 y i = i+1
           FIN Si
           FIN Mientras
   PASO 6: (i > N) "No se ha encontrado solución"
           PARAR
FIN ALGORITMO
```

(b) Pseudocódigo método de Newton

Figura 3.1: Funcionamiento del método de Newton en una máquina BSS, representado de forma gráfica y acompañado del pseudocódigo del mismo.

Como se puede ver en las Figuras 3.1a y 3.1b, el método toma un valor aproximado inicial x_0, que es suministrado por el usuario, y calcula el valor tanto de la función $f(x_0) = y$ como de la derivada $f'(x_0) = z$. A continuación aplica el algoritmo y comprueba si es lo "suficientemente bueno" como para mostrarlo como solución. Esto lo hace calculando la distancia entre el dato obtenido: x y la "solución exacta": 0. No obstante, aceptamos valores muy cercanos a cero, que es lo que representa la tolerancia 10^{-6}. Si esta operación ofrece un valor menor que dicha tolerancia, entonces

el algoritmo se detiene y se obtiene la solución. Sin embargo, si considera que aún no es lo suficientemente bueno como para tomarla como solución, el algoritmo vuelve a comenzar utilizando el valor obtenido x como nuevo dato de entrada x_0. Esto lo repite el número de veces que uno decida introducir en el programa, dependiendo del nivel de precisión que se quiera obtener o, en el caso en el que el algoritmo entre en bucle causado por una división entre 0*, pueda finalizar ofreciendo una respuesta de* NaN.

Capítulo 4

Un modelo formal para la estabilidad numérica

En este capítulo, siguiendo [2], vamos a explicar qué quiere decir que *un algoritmo calcule una función matemática*.

Es bien conocido por todos el *Algoritmo de Euclídes*, el cual permite calcular el máximo común divisor de dos números enteros en un número finito de pasos. En este caso, al igual que muchos otros, el resultado obtenido al aplicar dicho algoritmo es la solución correcta (exacta). Sin embargo, existen otros algoritmos que, debido a la naturaleza de los datos introducidos, no nos proporcionan la respuesta exacta, pero sí una que es lo "suficientemente buena" como para tomarla como solución. Tenemos por ejemplo el algoritmo que calcula la raíz cuadrada de un número real cualquiera: $\sqrt{x}$. Para algunos valores de x, esta función obtiene el resultado exacto. Ahora bien, si introducimos por ejemplo $x = 2$ obtenemos un resultado aproximado a lo que sería la solución real (que sabemos es un número irracional). Pues bien, lo que vamos a estudiar son las condiciones bajo las cuáles podemos decir que un algoritmo nos va a proporcionar un resultado "aceptable".

Se va a tomar una máquina BSS que tratará de implementar una función f mediante una equivalente $\hat{f}$. Al output, de haberlo, lo llamaremos $\hat{f}^u(x)$ siendo x el input. Lo que va a tratar de hacer este programa es aproximar una función f realizando los cálculos en $\mathbb{F}_u$.

El concepto de *estabilidad* se introdujo con el fin de clasificar los algoritmos numéricos $\hat{f}^u$ según si consiguen implementar y con qué grado de exactitud una función f.

4.1. Estabilidad hacia atrás

Decimos que un algoritmo es *estable hacia atrás* si dado un error de redondeo, se consigue la solución exacta para un valor de entrada lo suficientemente cerca del valor real.

Definición 4.1.1. *Sea $f : \mathbb{R} \longrightarrow \mathbb{R}$ una función y $\hat{f}$ una máquina BSS. Entonces, $\hat{f}$ es un algoritmo* **estable hacia atrás** *de f si existen constantes $a, b \in \mathbb{R} \setminus \{0\}$ tales que $\forall x \in \mathbb{R}$,*

$$0 < u \leq \frac{1}{a} \Longrightarrow \exists\ y \text{ tal que } \hat{f}^u(x) = f(y)\ \wedge\ \text{dist}(x, y) \leq bu \tag{4.1}$$

En particular, los datos de salida deben poder ser generados para cualquier valor de x y de u en ese rango.

4.2. Estabilidad mixta

Decimos que un algoritmo es *estable mixto* si con un error de redondeo dado, se consigue un resultado lo suficientemente cerca del resultado real para un valor de entrada lo suficientemente cerca del valor original.

Definición 4.2.1. *Sea $f : \mathbb{R} \longrightarrow \mathbb{R}$ una función y $\hat{f}$ una máquina BSS. Entonces, $\hat{f}$ es un algoritmo* **de estabilidad mixta** *de f si existen constantes $a, b, c \in \mathbb{R} \setminus \{0\}$ tales que $\forall x \in \mathbb{R}$,*

$$0 < u \leq \frac{1}{a} \Longrightarrow \exists\ y \text{ tal que } \text{dist}(\hat{f}^u(x), f(y))\ \leq bu\ \wedge\ \text{dist}(x, y)\ \leq cu \tag{4.2}$$

En particular, los datos de salida deben poder ser generados para cualquier valor de x y de u en ese rango.

4.3. Estabilidad hacia adelante

Decimos que un algoritmo es *estable hacia adelante* si con un error de redondeo dado, se consigue un resultado lo suficientemente cerca del resultado real para los valores de entrada exactos.

Definición 4.3.1. *Sea $f : \mathbb{R} \longrightarrow \mathbb{R}$ una función y $\hat{f}$ una máquina BSS. Entonces, $\hat{f}$ es un algoritmo* **estable hacia adelante** *de f si existe una constante $a \in \mathbb{R} \setminus \{0\}$ tal*

que o bien $\forall x \in \mathbb{R}$,

$$0 < u \leq \frac{1}{a\tilde{\kappa}(f,x)} \Longrightarrow \text{dist}(\hat{f}^u(x), f(x)) \leq \ a\tilde{\kappa}(f,x)u \tag{4.3}$$

o, equivalentemente, se tiene $\forall x \in \mathbb{R}$ *y* $\forall \epsilon \in [0,1]$,

$$0 < u \leq \frac{\epsilon}{a\tilde{\kappa}(f,x)} \Longrightarrow \ \text{dist}(\hat{f}^u(x), f(x)) \leq \ \epsilon \tag{4.4}$$

En particular, los datos de salida deben poder ser generados para cualquier valor de x *y de* u *en ese rango. Si* $\tilde{\kappa}(f,x) = \infty$*, entonces todas las implicaciones anteriores se cumplen automáticamente y dicho algoritmo estable hacia adelante puede generar un output cualquiera o no computarse y no generar ninguno, para* $u \neq 0$.

Observación 4.3.2. *Como se ha señalado, las dos condiciones de la Definición 4.3.1, son equivalentes. El motivo de mencionar ambas es que más adelante nos será útil utilizar una u otra, dependiendo de lo que queramos demostrar. La condición 4.3 nos dice que dada una unidad de redondeo* u *lo suficientemente pequeña, un algoritmo* estable hacia adelante *nos garantiza una cierta precisión; mientras que 4.4 se refiere a que para una cierta precisión requerida* ϵ*, se puede encontrar una unidad de redondeo tal que un algoritmo* estable hacia adelante *alcance la precisión exigida.*

Observación 4.3.3. *Se recalca también que el motivo de utilizar* $\tilde{\kappa}$ *en la definición 4.3.1 es porque en algunos problemas, el número de condición* κ *puede ser igual a* 0 *o a un número muy pequeño, así pues, pedirle al error que se ajuste a estos valores es algo poco realista.*

Ejemplos 4.3.4.

- $\mathbf{f(x) = x^2}$

  Sea $\hat{f}(x) = x \cdot x$ *el algoritmo que vamos a computar. Vamos a comprobar si es estable hacia adelante.*

  Sea $fl_u(x) = (1+\delta)x$ *la representación de* x *en coma flotante para un* $|\delta| < u$*, donde* u *es la unidad de redondeo descrita en el capítulo 3.*

  Escribimos $\hat{f}(x) = fl_u(x) \hat{*} fl_u(x) = (1+\delta)x \cdot (1+\delta)x \cdot (1+\tilde{\delta}) = (1+\delta)^2 \cdot x^2 \cdot (1+\tilde{\delta})$ *para un* $|\tilde{\delta}| < u \implies \hat{f}(x) = (1+\tilde{\tilde{\delta}})^3 \cdot x^2$ *con* $|\tilde{\tilde{\delta}}| < u$.

  Ahora calculamos el condicionamiento: $\tilde{\kappa}(f) = 1 + |x| \cdot \dfrac{|f'(x)|}{|f(x)|} = 1 + |x| \cdot \left|\dfrac{2x}{x^2}\right| = 3$

  Y vemos si se cumple 4.3; es decir, si existe un a *tal que* $0 < u \leq \dfrac{1}{a \cdot 3} \implies$ $\left|\log \dfrac{(1+\tilde{\tilde{\delta}})^3 \cdot x^2}{x^2}\right| < a \cdot u \cdot 3 \equiv |\log(1+\tilde{\tilde{\delta}})| < a \cdot u$, *lo que se cumple si* $2u < u \cdot a$.

  Por ejemplo, para $a = 3$ *las desigualdades se cumplen y por tanto se concluye que* $\hat{f}$ *es un algoritmo estable hacia adelante para* f.

  En el último cálculo se ha utilizado la propiedad: $|\log(1+\delta)| < 2|\delta|, \ |\delta| < \dfrac{1}{2}$.

- $\mathbf{f(x) = x^3 + 1}$

  Sea $\hat{f}(x) = x \cdot x \cdot x + 1$ *el algoritmo que se va a computar. Vamos a comprobar si es estable hacia adelante. La notación es exactamente la misma que en el ejemplo anterior.*

  Sea $fl_u(x) = (1+\delta)x$ *para un* $|\delta| < u$.

  Escribimos $\hat{f}(x) = fl_u(x) \hat{*} fl_u(x) \hat{*} fl_u(x) \hat{+} 1 = \big((1+\delta)x \cdot (1+\delta)x \cdot (1+\delta)x + 1\big) \cdot (1+\tilde{\delta})$ *para un* $|\tilde{\delta}| < u$. *Entonces,* $\hat{f}(x) = \big((1+\delta)^3 x^3 + 1\big)(1+\tilde{\delta}) = (1+\tilde{\tilde{\delta}})^4 x^3 + (1+\tilde{\delta})$, $|\tilde{\tilde{\delta}}| < u$. *Que podemos poner como* $\hat{f}(x) = (1+\hat{\delta})^4(x^3+1)$, *con* $|\hat{\delta}| < u$.

  Ahora calculamos el condicionamiento de $f(x)$: $\tilde{\kappa}(f) = 1 + |x| \cdot \dfrac{|3x^2|}{|x^3+1|} = \dfrac{|4x^3+1|}{|x^3+1|}$. *Y vemos si* $\exists a : 0 < u \leq \dfrac{|x^3+1|}{a|4x^3+1|} \implies \left|\log \dfrac{(1+\hat{\delta})^4(x^3+1)}{x^3+1}\right| < a \cdot u \cdot \dfrac{|4x^3+1|}{|x^3+1|} \overset{x\to\infty}{\longrightarrow} 4 \cdot a \cdot u \equiv |\log(1+\hat{\delta})^4| < 4 \cdot a \cdot u$ *lo que viene implicado por* $4 \cdot 2 \cdot u < 4 \cdot a \cdot u$.

  Y por ejemplo, para $a = 3$ *las desigualdades se cumplen; pudiendo concluir que* $\hat{f}$ *es un algoritmo estable hacia adelante para* f.

Para finalizar este capítulo, vamos a mostrar un ejemplo de un algoritmo a simple vista muy sencillo que matemáticamente funciona, pero que no cumple la definición de estabilidad.

Ejemplo 4.3.5.

- $\mathbf{f(x) = x}$

  Sea $\hat{f}(x) = (x+1) - 1$ *el algoritmo que se va a computar. Utilizando la misma notación que en los ejemplos anteriores vamos a probar que este algoritmo no es estable.*

  Sea $fl_u(x) = (1+\delta)x$ *para un* $|\delta| < u$.

  Escribimos $\hat{f}(x) = (fl_u(x) \hat{+} 1) \hat{-} 1 = \Big(\big(x(1+\delta_1)+1\big)(1+\delta_2) - 1\Big)(1+\delta_3)$ *para* $|\delta_1|, |\delta_2|, |\delta_3| \leq u$. *Entonces* $\big(x + x\delta_1 + 1 + x\delta_2 + x\delta_1\delta_2 + \delta_2 - 1\big)(1+\delta_3) = x + x\delta_1 + x\delta_2 + x\delta_1\delta_2 + \delta_2 + x\delta_3 + x\delta_1\delta_3 + x\delta_2\delta_3 + x\delta_1\delta_2\delta_3 + \delta_2\delta_3$ *y si sacamos factor común a la* x:

  $\hat{f}(x) = x \cdot \big(1 + \delta_1 + \delta_2 + \delta_3 + \delta_1\delta_2 + \delta_1\delta_3 + \delta_2\delta_3 + \delta_1\delta_2\delta_3 + \dfrac{\delta_2 + \delta_2\delta_3}{x}\big)$. *Calculamos el condicionamiento de* $f(x)$: $\tilde{\kappa}(f) = 1 + |x| \cdot \dfrac{1}{|x|} = 2$. *Y nos preguntamos si* $\exists a : 0 < u \leq \dfrac{1}{a \cdot 2} \implies \left|\log \dfrac{\hat{f}(x)}{x}\right| = \log\Big|(1 + \delta_1 + \delta_2 + \delta_3 + \delta_1\delta_2 + \delta_1\delta_3 + \delta_2\delta_3 + \delta_1\delta_2\delta_3 + \dfrac{\delta_2 + \delta_2\delta_3}{x})\Big| \leq 2 \cdot a \cdot u$. *Sin embargo, se puede ver que dado cualquier "a", siempre vamos a poder encontrar un* $x > 0$ *lo suficientemente pequeño como para*

que la desigualdad no se cumpla. Es por esto que este algoritmo no es estable. De hecho, hemos calculado los errores de este algoritmo con ayuda de MATLAB obteniendo la siguiente gráfica.

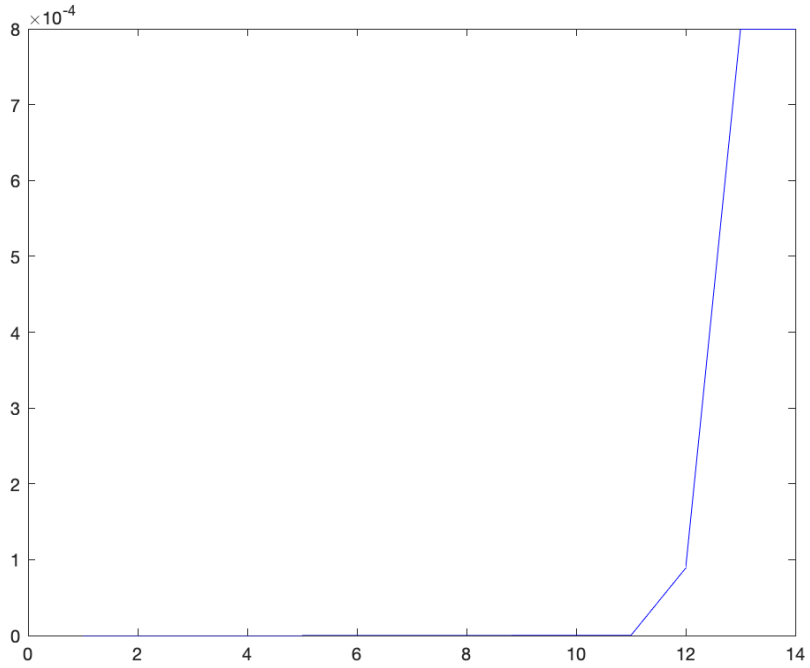


Figura 4.1: Gráfico de errores de $\hat{f}(x)$

En la Figura 4.1 hemos representado el error absoluto entre el valor real del $x = 10^{-i}$ *para* $1 \leq i \leq 14$ *y el valor del algoritmo* $\hat{f}(x) = (x+1) - 1$. *Como vemos, cuando la* x *no tiene un valor demasiado pequeño, el error es nulo; es decir, el algoritmo calcula bien la solución para esos valores de* x. *Sin embargo, llega a un punto en el que en la práctica, el valor de* $\hat{f}(x)$ *alcanza el* 0, *lo que provoca una división que hace que el error tienda a infinito (en la gráfica sucede cuando tenemos valores inferiores a* 10^{-11}*).*

Observación 4.3.6. *En el ejemplo anterior, podría suceder que* δ_2, δ_3 *fuesen 0, con lo que sí se cumpliría la definición de estabilidad para todo* x, *pero es una posibilidad ínfima que no es realista en la práctica.*

Capítulo 5

Estabilidad de algoritmos numéricos para problemas manejables

En este capítulo, se van a tratar varios conceptos nuevos. El primero es el de problema computacional *manejable*. Esto es, un tipo de función que admite un crecimiento no uniforme y acotado de su número de condición y que a su vez tiene un buen comportamiento cerca de los límites de su dominio. Segundo, se va a definir una *condición de compatibilidad* que permite asegurar que la composición de dos funciones manejables g y h, $g \circ h$ es manejable. Por último, probaremos el teorema principal de esta sección: *El Teorema de Composición*.

Este capítulo reproduce los resultados principales de [2] vistos para una variable.

5.1. Problemas manejables

El objetivo de este trabajo es propiciar que la composición de dos algoritmos numéricos estables resulte en un nuevo algoritmo también estable. Vamos a comenzar tomando dos funciones g, h que puedan ser compuestas. A la hora de llevar a cabo esta composición de funciones estables hacia adelante podemos toparnos con 2 obstáculos:

- Que el número de condición de g o de h crezca incontrolablemente.
- Que el máximo entre los números de condición de $\kappa(g, h(x))$ y de $\kappa(h, x)$ sea considerablemente mayor que el de $\kappa(g \circ h, x)$.

La idea principal de la "manejabilidad" de funciones es la de prevenir el primer

inconveniente. Este supone que la información proporcionada por el número de condición no sea útil; es decir, no aporte límites razonables para cambios en la entrada de datos muy pequeños. El segundo obstáculo trata de la inestabilidad del método de "mínimos cuadrados" para resolver el problema $Ax = b$ expuesto en el capítulo 1. Este punto fue incluso aprovechado en [16] para probar la inestabilidad hacia adelante de algunos métodos para resolver sistemas de ecuaciones polinomiales y en [1] para demostrar la inestabilidad hacia adelante en algunos métodos para el cálculo de descomposiciones de tensores según su rango.

Definición 5.1.1. *(Función manejable). Una función $f : \mathbb{R} \longrightarrow \mathbb{R}$ se dice que es* **manejable** *si existe una constante $a \in \mathbb{R}$ que cumple:*

D.1. Para todo $x \in S \subset \mathbb{R}$, la bola

$$B_x = \left\{ y \in \mathbb{R} : dist(x, y) \leq \frac{1}{a \cdot \tilde{\kappa}(f, x)} \right\}$$

está contenida en S.

D.2. Para todo $y \in B_x$ se tiene $\tilde{\kappa}(f, y) \leq a \cdot \tilde{\kappa}(f, x)$.

Para todos los puntos $x \in S$ tales que $\tilde{\kappa}(f, x) = \infty$, las condiciones de la definición se cumplen directamente. Por lo tanto, es necesario probar la manejabilidad únicamente de los puntos x en los que $\tilde{\kappa}(f, x)$ es finito.

El siguiente lema nos puede ser de ayuda a la hora de identificar si una función es o no manejable.

Lema 5.1.2. *Dada una función manejable $f : S \subset \mathbb{R} \longrightarrow \mathbb{R}$, supongamos que existe una constante $a \in \mathbb{R}$ que cumple las siguientes condiciones:*

(i) Sea $(x_0, x_1, x_2, ...) \subseteq S$ una secuencia tal que

$$dist(x_j, \partial S \cup \Gamma) \longrightarrow 0$$

donde ∂S es el borde de S y $\Gamma = \{x \in S : \kappa(f, x) = \infty\}$. Entonces $\kappa(f, x_j) \to \infty$.

(ii) Sea $V = S \setminus \Gamma$, el conjunto formado por los puntos cuyo número de condición es finito, y sea $\tilde{\kappa}_f : V \to \mathbb{R}$ definida como $\tilde{\kappa}_f(x) = \tilde{\kappa}(f, x)$. El número de condición de $\tilde{\kappa}_f$ cumple:

$$\kappa(\tilde{\kappa}_f, x) \leq \frac{a}{4} \tilde{\kappa}(f, x), \quad \forall x \in S \tag{5.1}$$

Entonces f es manejable.

Demostración. Sea $x \in S \subset \mathbb{R}$ cualquiera, $\tilde{\kappa}(f, x) < \infty$ y sea $y \in B_x$. Sea $\gamma(t) : [0, d] \longrightarrow \mathbb{R}$ la recta que une x e y, donde $d = dist(x, y) \leq \frac{1}{a \cdot \tilde{\kappa}(f, x)}$.

Primero se demuestra que $\gamma \subset V$ en las condiciones del lema. Asumamos que esto es falso.

Por (i) se tiene que $\gamma(0) = x \in V \subseteq S \setminus \partial S$ entonces existe un supremo $t \in [0, d]$ tal que $\gamma(s) \in V \ \forall \ 0 \leq s < t$ pero que $\gamma(t) \notin V$. Aplicando (ii) del Lema 5.1.2 y que $\tilde{\kappa}(f, \gamma(s)) < \infty \ \forall \ s \in [0, t)$ porque $\gamma([0, t)) \subset V$, se tiene que la función

$$\begin{aligned} H : [0, t) &\longrightarrow [0, \infty] \\ s &\longmapsto \tilde{\kappa}(f, \gamma(s)) \end{aligned}$$

es continua. Sea $C_s = \sup_{\alpha \in [0,s)} \tilde{\kappa}(f, \gamma(\alpha))$, una función continua en $0 \leq s < t$.

Del Lema 2.1.6 y de (ii) tenemos que para todo $s \in [0, t)$,

$$\log \left| \frac{\tilde{\kappa}_f(\gamma(s))}{\tilde{\kappa}_f(x)} \right| = dist(\tilde{\kappa}_f(\gamma(s)), \tilde{\kappa}_f(x)) \ \leq \ \sup_{\alpha \in [0,s)} \kappa(\tilde{\kappa}_f, \gamma(\alpha))s$$

$$\leq \frac{a}{4} \sup_{\alpha \in [0,s)} \tilde{\kappa}(f, \gamma(\alpha))s \ \leq \ \frac{a}{4} C_s s$$

En particular, como $0 < t \leq d \leq \dfrac{1}{a \cdot \tilde{\kappa}(f, x)}$ entonces

$$\log \left| \frac{C_s}{\tilde{\kappa}_f(x)} \right| \leq \frac{a}{4} C_s s \leq \frac{C_s}{4\tilde{\kappa}(f, x)}, \ s \in [0, t).$$

Lo que significa que el conjunto $R = \{\dfrac{C_s}{\tilde{\kappa}(f, x)} : s \in [0, t)\}$ está contenido en el conjunto $\{\alpha \in (0, \infty) : \log(\alpha) \leq \dfrac{\alpha}{4}\}$ el cual está a su vez contenido en $(0, 2] \cup [8, \infty)$.

Como C_s es continua implica que R es conexo y este contiene al punto 1, ya que $C_0 = C(0) = \tilde{\kappa}(f, \gamma(0))$ donde $\gamma(0) = x \in S \setminus \partial S$ es un punto aislado y por tanto $C_0 = \tilde{\kappa}(f, x) = 1 + \kappa(f, x)$. Esto implica que $R \subseteq (0, 2]$. Es decir,

$$C_s \leq 2\tilde{\kappa}(f, x) \ \forall \ s \in [0, t)$$

Pero en la hipótesis habíamos supuesto que $\tilde{\kappa}(f, x) \longrightarrow \infty$ cuando $\gamma(s) \longrightarrow \partial S$, sin embargo, ahora vemos que $\sup_{\alpha \in [0,s]} \tilde{\kappa}(f, \gamma(\alpha)) \not\longrightarrow \infty$, luego $\gamma(s) \not\longrightarrow \partial S$. Esto contradice la existencia de un $t \in [0, d]$ tal que $\gamma(t) \notin V$. Por todo ello, se concluye que $\gamma \subset V$ y, en consecuencia, la bola B_x de radio $\dfrac{1}{a \cdot \tilde{\kappa}(f, x)}$ también está contenida en $V \subseteq S$ lo que prueba el primer apartado del lema.

Puesto que todo el razonamiento anterior es válido para todo $t \in [0, d]$, se puede afirmar que

$$\tilde{\kappa}(f, y) \leq C_d \leq 2\tilde{\kappa}(f, x)$$

quedando probado así también el segundo apartado del lema. □

Del Lema 5.1.2 se sigue directamente que V es un subconjunto abierto de $\mathbb{R}$ y que la restricción $\tilde{\kappa}_f|_V$ es continua.

Observación 5.1.3. *Si $\kappa(f,x)$ viene dada por una función de tipo $\mathcal{C}^\infty$, entonces la fórmula 5.1 se satisface si:*

$$x \cdot \frac{\partial \kappa(f,x)}{\partial x} \leq q \cdot \tilde{\kappa}(f,x)^2 \tag{5.2}$$

para alguna constante $q \in \mathbb{R}$.

Demostración. Sea f una función en las condiciones del Lema 5.1 y sea $\kappa(f,x) \in \mathcal{C}^\infty$. Se calcula,

$$\kappa(\tilde{\kappa}_f, x) \overset{(2,3)}{=} \frac{|x| \cdot |\tilde{\kappa}_f'(x)|}{|\tilde{\kappa}_f(x)|} \leq \frac{a}{4}\tilde{\kappa}_f(x) \Longrightarrow |x| \cdot |\tilde{\kappa}_f'(x)| \leq \frac{a}{4}\tilde{\kappa}_f^2(x)$$

Con $q = \dfrac{a}{4}$ ya tenemos la parte de la derecha. Si seguimos operando,

$$|x| \cdot |\tilde{\kappa}_f'(x)| = |x| \cdot \frac{\big(|f'(x)| + |x||f''(x)|\big)|f(x)| - |x||f'(x)|^2}{f^2(x)}$$

$$= |x| \cdot \frac{|f'(x)||f(x)| - |x|f'(x)^2}{f^2(x)} = |x| \cdot \kappa'(f,x)$$

Ya que $f''(x) = 0$ al ser f una función real. Y así, hemos terminado la demostración. □

Lema 5.1.4. *Si $f|_{S_n}$, con $S_n \subseteq \mathbb{R}$ es manejable para un conjunto finito de intervalos abiertos ordenados, entonces $f|_S$ es una función manejable con $S = \bigcup_n S_n$.*

Es más, si todos los $f|_{S_n}$ comparten la misma constante de la definición de manejabilidad, entonces $f|_S$ también es manejable para esa misma constante incluso si el número de conjuntos S_n es infinito.

Demostración. Sea f una función definida en un espacio métrico $\mathbb{R}$ con su topología. Sean S_n una sucesión finita de intervalos en $\mathbb{R}$: $S_1 = (a_1, b_1)$, $S_2 = (a_2, b_2)$, ... $S_n = (a_n, b_n)$ con $a_1 \leq a_2 \leq \ .... \ \leq a_n$; $b_1 \leq b_2 \leq \ .... \ \leq b_n$; $a_i < b_i$. Suponemos que f es manejable en cada S_i, $i \in \{1, ..., n\}$. Entonces f cumple la definición 5.1.1 en cada intervalo con una constante para cada uno; es decir,

$$f \text{ manejable en } S_1 \text{ para } \alpha_1; \quad ..... \ ; \ f \text{ manejable en } S_n \text{ para } \alpha_n$$

donde $\alpha_i \in S_i$.
Así, tenemos un conjunto $\{\alpha_1, ... \ , \alpha_n\}$ finito, discreto de números reales y que por tanto tiene un máximo $\gamma = \text{máx}\{\alpha_1, ... \ , \alpha_n \ \}$.
Pues bien, veamos si se cumple la definición de manejabilidad para $f|_S$:

- Sea $x \in S_i$ para algún i. Entonces $x \in S$, y por tanto sabemos que se cumple, $\forall x \in S:\ B_x^1 = \left\{y \in \mathbb{R}:\ dist(x,y) \leq \dfrac{1}{\gamma \cdot \tilde{\kappa}(f,x)}\right\} \subset B_x^2 = \left\{y \in \mathbb{R}:\ dist(x,y) \leq \dfrac{1}{\alpha_i \cdot \kappa(\tilde{f},x)}\right\} \subset S$. Teniendo así probada la primera condición 5.1.1 [D.1].
- Ahora, $\forall y \in B_x^1$, se cumple

$$\tilde{\kappa}(f,y) \leq \alpha_i \cdot \tilde{\kappa}(f,x) \leq \gamma \cdot \tilde{\kappa}(f,x)$$

Lo que prueba también la segunda condición 5.1.1 [D.1] y en consecuencia que f es manejable en $S = \cup_n S_n$.

□

Veamos algunos ejemplos de funciones manejables.

Ejemplo 5.1.5.

- $\mathbf{f(x) = \dfrac{1}{x}}$

$$\kappa(f,x) = \frac{|x| \cdot \left|\frac{-1}{x^2}\right|}{\left|\frac{1}{x}\right|} = 1 \implies \tilde{\kappa}(f,x) = \kappa(f,x) + 1 = 2$$

$$\kappa(\tilde{\kappa}(f,x),x) = \frac{|x| \cdot 0}{2} = 0 \ \leq\ \frac{1}{4} \cdot \tilde{\kappa}(f,x) = \frac{1}{2} \quad \forall\ x \in \mathbb{R} \setminus \{0\}$$

Por el Lema 5.1.2, $f(x)$ es manejable.

- $\mathbf{f(x) = x^2 + 1}$

$$\kappa(f,x) = \frac{|x| \cdot |2x|}{|x^2+1|} = \frac{2x^2}{x^2+1} \implies \tilde{\kappa}(f,x) = \frac{2x^2}{x^2+1} + 1 = \frac{3x^2+1}{x^2+1}$$

$$\kappa(\tilde{\kappa}(f,x),x) = \frac{|x| \cdot \dfrac{|6x(x^2+1) - 2x(3x^2+1)|}{(x^2+1)^2}}{\dfrac{(3x^2+1)}{x^2+1}} = \frac{2x^2(3x^2+3-3x^2-1)}{(x^2+1)(3x^2+1)}$$

$$= \frac{4x^2}{(x^2+1)(3x^2+1)}$$

Y comprobamos si $\exists\ a \in \mathbb{R}$ *tal que* $4x^2 \ \leq\ \dfrac{a}{4}(3x^2+1)^2 \ \ \forall x \in \mathbb{R}$

Fácilmente se comprueba que para $a = 4$, por ejemplo, se cumple y por el Lema 5.1.2 se concluye que $f(x)$ es manejable.

Un ejemplo de una función no manejable es:

Ejemplo 5.1.6.

- $\mathbf{f(x)} = \sin x,\ x \in \mathbb{R}.$

Dado el condicionamiento de esta función $\kappa(f,x) = \dfrac{|x|\cdot|cos(x)|}{|sen(x)|}$ *hay puntos en los que es igual a* 0*, que son los de la forma* $x = k\pi + \dfrac{\pi}{2}$*, para cualquier* $k \in \mathbb{Z}$*; y puntos en los que es infinito, que son todos los de la forma* $x = k\pi,\ \forall\ k \in \mathbb{Z}$*. Si intentamos aplicar el Lema 5.1.2, vemos que la primera condición ya falla: sea la sucesión* $x_0 = \dfrac{\pi}{2},\ x_1 = \pi + \dfrac{\pi}{2},\ x_2 = 2\pi + \dfrac{\pi}{2},\ \ldots\ ,\ x_n = n\pi + \dfrac{\pi}{2}$ *para* $n \in \mathbb{N}\cup\{0\}$*. Tomando el límite cuando* $n \longrightarrow \infty$ *se tiene que* $d(x_n,\Gamma) = \left|\log\dfrac{x_n + \frac{\pi}{2}}{x_n}\right| = \left|\log(1 + \dfrac{\pi}{2x_n})\right| \underset{n\to\infty}{\longrightarrow} 0$*; pero* $\kappa(x_n) = 0$ *funciona. Luego no se cumple la primera condición que exige el Lema 5.1.2. Así que vamos a demostrar que efectivamente* $\sin x$ *no cumple la definición de función manejable 5.1.1. Supongamos que existe un* $a > 0$ *tal que* $\forall\ x$ *se cumple que* $B_x = \{y : \left|\log\dfrac{x}{y}\right| \leq \dfrac{1}{a\cdot\tilde{\kappa}}\} \subseteq S$*. Esto es siempre cierto ya que* $S = \mathbb{R}$*. Y, además, supongamos que se satisface que* $\forall y \in B_x,\ \tilde{\kappa}(f,y) < a\cdot\tilde{\kappa}(f,x)$*. Entonces, se cumple que* $\forall n,\ \exists y \in B_{x_n} = \{y : \left|\log\dfrac{x_n}{y}\right| \leq \dfrac{1}{a}\} \implies \tilde{\kappa}(f,y) \leq a$*. Por lo tanto, si es cierto para todo* n*, también lo será para uno muy grande,* $y = x_n - \dfrac{\pi}{2} \in B_{x_n} \Longrightarrow \left|\log\dfrac{x_n}{y}\right| = \left|\log(1+\dfrac{1}{2n})\right| \leq \dfrac{1}{a}$*. Sin embargo, hemos visto que* $\tilde{\kappa}(f,y) = \infty \not\leq a$*. Entonces, no existe* a *con esa propiedad y, en consecuencia,* $f(x) = \sin x$ *no es manejable.*

5.2. Condición de compatibilidad

El segundo obstáculo con el que nos podemos encontrar a la hora de componer dos funciones, al que nos hemos referido al inicio del capítulo, trata sobre descomponer un problema bien condicionado f en la composición de otros dos $f = g\circ h$ donde o bien g o bien h está mucho peor condicionado que f; lo cual conllevaría que $\hat{g}\circ\hat{h}$ fuese un algoritmo inestable hacia adelante. Para evitar que esto ocurra, tenemos la condición de compatibilidad.

Definición 5.2.1. *(Condición de compatibilidad). Sean* g, h *dos funciones tales que su composición* $g\circ h$ *está bien definida.*

$$S \subseteq \mathbb{R} \xrightarrow{h} T \subseteq \mathbb{R} \xrightarrow{g} \mathbb{R}$$

$$x \quad\mapsto\quad h(x) \quad\mapsto\quad g(h(x))$$

Se dice que g *y* h *son* **compatibles** *si existe una constante* b *tal que* $\forall x \in S$ *se cumple*

$$\tilde{\kappa}(g,h(x))\cdot\tilde{\kappa}(h,x) \leq b\cdot\tilde{\kappa}(g\circ h,x). \tag{5.3}$$

Esta condición de compatibilidad garantiza que la composición de dos funciones manejables sea otra función manejable, como se muestra a continuación:

Lema 5.2.2. *Si g y h son dos funciones compatibles y manejables, entonces $g \circ h$ es también una función manejable.*

Demostración. Sean a_g y a_h las constantes para las cuales g y h son manejables respectivamente, y sea b la constante que aparece en la definición 5.2.1. Vamos a demostrar que $f = g \circ h$ es manejable y su constante asociada es $a = a_g \cdot a_h \cdot b$.
De la definición de manejabilidad (5.1.1) sabemos que solo necesitamos estudiar los casos en los que $\tilde{\kappa}(g \circ h, x)$ es finito. Y de la definición de compatibilidad (5.2.1) sabemos que tanto $\tilde{\kappa}(g, h(x))$ como $\tilde{\kappa}(h, x)$ son finitos individualmente. Ahora, sean $x \in S$ e $y \in \mathbb{R}$ tales que

$$dist(x, y) \leq \frac{1}{a \cdot \tilde{\kappa}(f, x)} \overset{(5.3)}{\leq} \frac{b}{a \cdot \tilde{\kappa}(h, x)} \leq \frac{1}{a_h \cdot \tilde{\kappa}(h, x)}.$$

Por lo tanto, usando 5.1.1 [D.1.] para h, se tiene que $y \in S$ lo que implica que y está en el dominio de h.

$$S \xrightarrow{h} T \subseteq \mathbb{R} \xrightarrow{g} \mathbb{R}$$

$$S \xrightarrow{f} \mathbb{R}$$

A continuación, queremos comprobar que $h(y) \in T$; es decir, que la imagen por h de y está en el dominio de g.

$$dist(h(x), h(y)) \leq dist(x, y) \cdot \max_{z \in B_x} \tilde{\kappa}(h, z) \leq dist(x, y) \cdot a_h \cdot \tilde{\kappa}(h, z)$$

$$\leq \frac{a_h \cdot \tilde{\kappa}(h, x)}{a \cdot \tilde{\kappa}(f, x)} \overset{(5.3)}{\leq} \frac{1}{a_g \cdot \tilde{\kappa}(g, h(x))}$$

Por lo que $\tilde{\kappa}(g, h(y)) \leq a_g \cdot \tilde{\kappa}(g, h(x))$ y efectivamente, con la definición 5.1.1 [D.1.] lo que queríamos probar es cierto. Ahora nos falta ver la segunda condición de la definición de manejabilidad (5.1.1 [D.1.]).

$$\tilde{\kappa}(f, y) \overset{(2.2)}{\leq} \tilde{\kappa}(h, y)\tilde{\kappa}(g, h(y)) \leq a_h \cdot a_g \cdot \tilde{\kappa}(g, h(x)) \overset{(5.2.1)}{\leq} a \cdot \tilde{\kappa}(f, x)$$

Que es lo que queríamos demostrar. □

Ejemplo 5.2.3. *Vamos a comprobar si la composición de las funciones del ejemplo 5.1.5, las cuales sabemos que son manejables, es también una función manejable.*
Sean $g(x) = x^2 + 1$ y $h(x) = \frac{1}{x}$. Entonces, $f = g \circ h = \frac{1}{x^2} + 1$:

$$f : \mathbb{R} \setminus \{0\} \xrightarrow{h} \mathbb{R} \xrightarrow{g} \mathbb{R}$$

Veamos si se cumple (5.3):

$$\tilde{\kappa}(g, h(x)) = 1 + \frac{|h(x)| \cdot |g'(x)|}{|g(x)|} = 1 + \frac{\frac{1}{x} \cdot 2x}{x^2+1} = \frac{x^2+3}{x^2+1}$$

$$\tilde{\kappa}(h, x) = 2 \quad \textit{por} \quad 5{,}1{,}5$$

$$\tilde{\kappa}(g \circ h, x) = \tilde{\kappa}(f, x) = 1 + \frac{|x| \cdot \left|\frac{-2}{x^3}\right|}{\left|\frac{1}{x^2} + 1\right|} = 1 + \frac{\frac{2}{x^2}}{\frac{1+x^2}{x^2}} = \frac{x^2+3}{x^2+1}$$

$$\Longrightarrow \frac{x^2+3}{x^2+1} \cdot 2 \leq b \cdot \frac{x^2+3}{x^2+1} \Longrightarrow \textit{para } b = 2 \textit{ se cumple la condición y se concluye que } f = g \circ h \textit{ es manejable.}$$

5.3. El Teorema de composición

Y llegamos al teorema más importante de este trabajo, para lo que se ha desarrollado todo lo anterior.

Teorema 5.3.1. *Sean g y h dos funciones compatibles y manejables. Para cualesquiera algoritmos estables hacia adelante $\hat{g}^u$, $\hat{h}^u$ que implementan g y h respectivamente, la composición $\hat{f}^u = \hat{g}^u \circ \hat{h}^u$ es un algoritmo estable hacia adelante que implementa $f = g \circ h$.*

Demostración. Sean $h : S \subseteq \mathbb{R} \longrightarrow \mathbb{R}$ y $g : T \subseteq \mathbb{R} \longrightarrow \mathbb{R}$ funciones compatibles y manejables con a_h y a_g las constantes de la definición 5.1.1 para las que son manejables respectivamente. Sean $\hat{a}_g$ y $\hat{a}_h$ las constantes de la definición 4.3.1 para $\hat{g}, \hat{h}$ respectivamente. Por último, sea b la constante de la definición 5.2.1 para g y h.

$$S \subset \mathbb{R} \xrightarrow{h,\ \hat{h}} T \subseteq \mathbb{R} \xrightarrow{g,\ \hat{g}} \mathbb{R}$$

$$S \xrightarrow{f=g\circ h,\ \hat{f}=\hat{g}\circ\hat{h}} \mathbb{R}$$

Vamos a demostrar que el teorema de composición se cumple con constante de estabilidad igual a

$$a = b \cdot a_g \cdot \left(\hat{a}_g + \hat{a}_h\right)$$

Igual que en la demostración del Lema 5.2.2, solo necesitamos estudiar los casos en los que $\tilde{\kappa}(g \circ h, x)$ sea finito. Por ello, podemos suponer que $\tilde{\kappa}(g, h(x))$ y $\tilde{\kappa}(h, x)$ son ambos finitos.

Sabemos que g y h con compatibles y $\hat{h}^u$ es estable hacia adelante por hipótesis, así

que se cumple

$$u \overset{(4.3)}{\leq} \frac{1}{a \cdot \tilde{\kappa}(f,x)} \overset{(5.3)}{\leq} \frac{b}{a \cdot \tilde{\kappa}(h,x)} \overset{(4.3)\ para\ h}{\leq} \frac{1}{\hat{a}_h \cdot \tilde{\kappa}(h,x)} \Longrightarrow$$

$$\text{por } (4.3): \ dist(\hat{h}^u(x), h(x)) \leq \hat{a}_h \cdot \tilde{\kappa}(h,x) \cdot u \overset{(5.3)}{\leq} \frac{\hat{a}_h \cdot b \cdot \tilde{\kappa}(g \circ h, x)}{a \cdot \tilde{\kappa}(g, h(x)) \cdot \tilde{\kappa}(f,x)}$$

$$\leq \frac{b \cdot \hat{a}_h}{a \cdot \tilde{\kappa}(g, h(x))} \overset{(4.3)\ para\ g}{\leq} \frac{1}{a_g \cdot \tilde{\kappa}(g, h(x))}$$

Esto nos dice que (por 5,1,1 [D.1]) la bola $B_{h(x)}$ de centro $h(x)$ y radio $\dfrac{1}{a_g \cdot \tilde{\kappa}(g, h(x))}$ está en el dominio de g; es decir, está contenida en T. Y por tanto $g(\hat{h}(x))$ está bien definida. Además, por 5,1,1 [D.2] en g se tiene,

$$\tilde{\kappa}(g, dist(\hat{h}(x), h(x))) \leq a_g \cdot \tilde{\kappa}(g, h(x)) \tag{5.4}$$

Ahora, a partir del Lema 2.1.6 comprobamos,

$$dist(g(\hat{h}(x)), g(h(x))) \leq C \cdot dist(\hat{h}(x), h(x))$$

donde $C = \text{máx } \tilde{\kappa}(g, dist(\hat{h}(x), h(x))) \Longrightarrow$

$$\begin{aligned} &\text{Por } (5.4) && \leq dist(\hat{h}(x), h(x)) \cdot a_g \cdot \tilde{\kappa}(g, h(x)) \\ &\text{Por } (5.3) && \leq a_g \cdot \hat{a}_h \cdot \tilde{\kappa}(g, h(x)) \cdot \tilde{\kappa}(h,x) \cdot u \\ &\text{Por 5,1,1 } [D,1] && \leq a_g \cdot \hat{a}_h \cdot b \cdot \tilde{\kappa}(f,x) \cdot u \end{aligned} \tag{5.5}$$

De (5.4) deducimos que se cumple $\tilde{\kappa}(g, dist(\hat{h}(x), h(x))) \leq \tilde{\kappa}(g, \hat{h}^u(x))$ y, por tanto, también se verifica la siguiente cadena de desigualdades:

$$u \leq \frac{1}{a \cdot \tilde{\kappa}(f,x)} \leq \frac{b}{a \cdot \tilde{\kappa}(g, h(x))} \leq \frac{a_g \cdot b}{a \cdot \tilde{\kappa}(g, \hat{h}^u(x))} \leq \frac{1}{\hat{a}_g \cdot \tilde{\kappa}(g, \hat{h}^u(x))}$$

Y como $\hat{g}^u$ es estable hacia adelante, lo anterior nos asegura que $\hat{g}^u(\hat{h}^u(x))$ está bien definida. Con lo cual,

$$dist\big(\hat{g}^u(\hat{h}^u(x)), g(\hat{h}^u(x))\big) \leq \hat{a}_g \cdot \tilde{\kappa}(g, \hat{h}^u(x)) \cdot u \leq a_g \cdot \hat{a}_g \cdot \tilde{\kappa}(g, h(x)) \cdot u \tag{5.6}$$

Por último, juntando (5.5) y (5.6) se comprueba que:

$$dist(\hat{f}^u(x), f(x)) \leq dist\big(\hat{g}^u(\hat{h}^u(x)), g(\hat{h}^u(x))\big) + dist(g(\hat{h}^u(x)), g(h(x)))$$

$$\leq a_g \cdot \hat{a}_g \cdot \tilde{\kappa}(g, h(x)) \cdot u + a_g \cdot \hat{a}_h \cdot b \cdot \tilde{\kappa}(f,x) \cdot u$$

$$= \Big(a_g \cdot \hat{a}_g \cdot b + a_g \cdot \hat{a}_h \cdot b\Big) \tilde{\kappa}(f,x) \cdot u$$

De donde obtenemos $a_g \cdot b \cdot (\hat{a}_g + \hat{a}_h) = a$ como queríamos. En consecuencia, se cumple la condición (4.3) quedando probado así el teorema. □

Observación 5.3.2. *El teorema anterior se puede extender para componer tres o más funciones. Por ejemplo, sean dadas las funciones f, g, h que puedan ser compuestas y que las tres sean manejables. Primero se comprueba que g y h ; f y $(g \circ h)$ son compatibles entre sí, para después dados los algoritmos estables $\hat{f}, \hat{g}, \hat{h}$ y con el Teorema 5.3.1, probar que $(\hat{g} \circ \hat{h})$ es un algoritmo estable para $(g \circ h)$ y que $\hat{f} \circ (\hat{g} \circ \hat{h})$ es un algoritmo estable para $f \circ (g \circ h)$.*

Ejemplo 5.3.3.

Tomemos las dos funciones anteriormente descritas: $g(x) = x^2 + 1$ y $h(x) = \dfrac{1}{x}$. Por el ejemplo 5.1.5 sabemos que ambas son manejables y por el ejemplo 5.2.3 sabemos que son compatibles y que su composición $f = g \circ h = \dfrac{1}{x^2} + 1$ es también manejable.

A continuación, veremos si $\hat{h}^u = \dfrac{1}{x}$, $\hat{g}^u = x \cdot x + 1$ son algoritmos estables hacia adelante. Siguiendo el mismo procedimiento que en el ejemplo 4.3.4, se concluye que:

- $\hat{g}^u = (1+\hat{\delta})^3(x^2+1) \Longrightarrow$ *necesitamos un* $a \in \mathbb{R}$ *tal que* $0 < u \leq \dfrac{|x^2+1|}{a|3x^2+1|} \Longrightarrow$ $\left|\log(1+\hat{\delta})^3\right| < a \cdot u \cdot \dfrac{|3x^2+1|}{|x^2+1|} \overset{x\to\infty}{\longrightarrow} 3 \cdot a \cdot u$. *Concluyendo que para* $a = 3$ *funciona, por lo que* $\hat{g}^u(x)$ *es un algoritmo estable hacia adelante para* $g(x)$.
- $\hat{h}^u = \dfrac{1}{x} \Longrightarrow 0 < u \leq \dfrac{1}{a \cdot 2} \Longrightarrow |\log 1| < a \cdot u \cdot 2$. *Para* $a = 1$ *se prueba que* $\hat{h}^u(x)$ *es un algoritmo estable para* $h(x)$.

Con esto, nos encontramos en las condiciones del Teorema de Composición; así que se tiene que cumplir que $\hat{f}^u = \hat{g}^u \circ \hat{h}^u$ es un algoritmo estable que implementa a $f = g \circ h$.

Comprobémoslo directamente:

$$\hat{f}^u = \frac{(1+\delta)(x^2+1)}{x^2} = (1+\delta) \cdot f(x)$$

$$\tilde{\kappa}(x) = 1 + |x| \frac{\left|\frac{2}{x^3}\right|}{\left|\frac{1}{x^2}+1\right|} = \frac{x^2+3}{x^2+1}$$

Necesitamos encontrar un $a \in \mathbb{R}$ *que cumpla:* $0 < u \leq \dfrac{x^2+1}{a(x^3+1)} \Longrightarrow$

$$|\log(1+\delta)| < a \cdot u \cdot \frac{x^2+1}{x^2+3} \overset{x\to\infty}{\longrightarrow} a \cdot u.$$

Por lo tanto, escogiendo por ejemplo $a = 2$ tenemos finalmente que se cumple la definición de estabilidad para $\hat{f}^u(x)$ como decía el Teorema de Composición 5.3.1.

Capítulo 6

Conclusiones

Como hemos visto, ya a los antiguos matemáticos les preocupaba la fiabilidad de sus cálculos y problemas. Pero fue con la llegada de los ordenadores cuando las cuestiones que comenzaban a presentarse resultaban ser de una dificultad notablemente mayor y fue por ello que la comunidad científica ahondó en el estudio del campo del análisis numérico.

Tanto el condicionamiento de un problema, como la estabilidad del algoritmo que usemos para resolverlo, son aspectos que hay que tener en cuenta en el análisis numérico. Su desarrollo está en constante evolución, al igual que sus definiciones. Ya sea de una manera u otra, necesitamos tener una idea de si los problemas que se nos presenten pueden ser resueltos con precisión satisfactoria o no y, claro está, cuál es el método más eficaz para poder conseguirlo. Puede que sea necesario obtener resultados con un grado mayor de precisión, a costa de generarlo en un periodo largo de tiempo. Y viceversa, obtener el resultado en un instante, perdiendo así algo de exactitud. Lo importante es conocer las características de los elementos que vamos a usar y necesitar para poder tomar decisiones eficientes a la hora de enfrentarnos a estos problemas.

Por ello es tan importante tener herramientas como el del cálculo del número de condición o teoremas y proposiciones que nos ayuden a identificar la estabilidad de un algoritmo.

La razón principal del cálculo del número de condición es conocer de antemano si es factible resolver un problema en estudio. En este trabajo hemos estudiado una manera asequible de obtener esta respuesta.

Primero se han presentado tres formas de clasificar un algoritmo en cuanto a su estabilidad. Entonces, conociendo la estabilidad de un algoritmo, hemos introducido las definiciones de *compatibilidad* y *manejabilidad*. Gracias a estos dos nuevos conceptos,

podemos asegurar que un nuevo algoritmo es estable para el cálculo de funciones si sabemos que proviene de dos algoritmos estables cuyas funciones asociadas son compatibles y manejables.

Todos los conceptos vistos en este trabajo han sido ilustrados con ejemplos que aunque aparenten ser sencillos esconden algunas sorpresas, como por ejemplo en el caso de la función $\text{sen}(x)$.

Bibliografía

[1] Carlos Beltrán, Paul Breiding y Nick Vannieuwenhoven. «Pencil-based algorithms for tensor rank decomposition are not stable». En: *SIAM Journal on Matrix Analysis and Applications* 40.2 (2019), págs. 739-773.

[2] Carlos Beltrán, Vanni Noferini y Nick Vannieuwenhoven. «When can forward stable algorithms be composed stably?» En: *arXiv preprint arXiv:2109.10610* (2022).

[3] Lenore Blum, Mike Shub y Steve Smale. «On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines». En: *The Collected Papers of Stephen Smale: Volume 3*. World Scientific, 2000, págs. 1293-1338.

[4] Lenore Blum y col. *Complexity and real computation*. Springer Science & Business Media, 1998.

[5] James R Bunch. «The weak and strong stability of algorithms in numerical linear algebra». En: *Linear Algebra and Its Applications* 88 (1987), págs. 49-66.

[6] Peter Bürgisser y Felipe Cucker. «The geometry of numerical algorithms». En: *Springer Sci. & Bus. Media* 349 (2013).

[7] Alonzo Church. «AM Turing. On computable numbers, with an application to the Entscheidungs problcm. Proceedings of the London Mathematical Society, 2 s. vol. 42 (1936–1937), pp. 230–265.» En: *The Journal of Symbolic Logic* 2.1 (1937), págs. 42-43.

[8] Felipe Cucker. «A theory of complexity, condition, and roundoff». En: *Forum of Mathematics, Sigma*. Vol. 3. Cambridge University Press. 2015.

[9] LP Eisenhart. *Riemannian geometry 2nd ed.* 1949.

[10] Christine Gaßner. «A General BSS Model over Arbitrary Structures». En: (2011).

[11] John Guckenheimer y Robert F Williams. «Structural stability of Lorenz attractors». En: *Publications Mathématiques de l'IHÉS* 50 (1979), págs. 59-72.

[12] Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[13] Lieuwe Sytse de Jong. «Towards a formal definition of numerical stability». En: *Numerische Mathematik* 28.2 (1977), págs. 211-219.

[14] Pascal Koiran. «A weak version of the Blum, Shub, and Smale model». En: *Journal of Computer and System Sciences* 54.1 (1997), págs. 177-189.

[15] Gregorio Malajovich y Mike Shub. «A Theory of NP-completeness and Ill-conditioning for Approximate Real Computations». En: *Journal of the ACM (JACM)* 66.4 (2019), págs. 1-38.

[16] Vanni Noferini y Alex Townsend. «Numerical instability of resultant methods for multidimensional rootfinding». En: *SIAM Journal on Numerical Analysis* 54.2 (2016), págs. 719-743.

[17] FWJ Olver. «A new approach to error arithmetic». En: *SIAM Journal on Numerical Analysis* 15.2 (1978), págs. 368-393.

[18] «Problemas matemáticos para el próximo siglo.» es. En: *Gaceta de la Real Sociedad Matemática Española* 3 (2000), págs. 413-434. URL: http://dml.mathdoc.fr/item/urn:eudml:doc:43264.

[19] JD Pryce. «A new measure of relative error for vectors». En: *SIAM journal on numerical analysis* 21.1 (1984), págs. 202-215.

[20] Soledad Represa. «Ecuaciones de Lorenz». En: ().

[21] John R Rice. «A theory of condition». En: *SIAM Journal on Numerical Analysis* 3.2 (1966), págs. 287-310.

[22] Warwick Tucker. «A rigorous ODE solver and Smale's 14th problem». En: *Foundations of Computational Mathematics* 2.1 (2002), págs. 53-117.

[23] Alan M Turing. «Rounding-off errors in matrix processes». En: *The Quarterly Journal of Mechanics and Applied Mathematics* 1.1 (1948), págs. 287-308.

[24] John Von Neumann y Herman H Goldstine. «Numerical inverting of matrices of high order». En: *Bulletin of the American Mathematical Society* 53.11 (1947), págs. 1021-1099.

[25] James Hardy Wilkinson. *Rounding errors in algebraic processes*. Courier Corporation, 1994.