Curso de Métodos Numéricos para ¿ingenieros?

Pedro Fortuny Ayuso

Curso 2022/23, EPIG, Gijón. Universidad de Oviedo
Email address: fortunypedro@uniovi.es

Índice general

Introducción

Estas notas cubren *esencialmente* los contenidos de las clases de Teoría de la asignatura de Métodos Numéricos de la EPI de Gijón, al estilo del autor. Como toda colección informal de *notas*, pueden contener errores: errores de concepto, erratas, e incluso errores en los enunciados y demostraciones. Aun así, he ido tratando de revisarlas cada curso y corregir las faltas que encontraba. De todos modos,
Estas notas no necesariamente suplen un buen libro (o dos)

1. Algunos comentarios

El objetivo que pretendo cuando explico esta asignatura es triple:

- Introducir los problemas básicos que afronta el Cálculo Numérico, en su versión más sencilla.
- Acostumbrar a los alumnos a enunciar algoritmos con precisión y a estudiar su complejidad.
- Transmitir a los alumnos la importancia de *comprobar que la salida de un programa tiene sentido*. Esto es mucho más importante de lo que parece. Que "un ordenador diga algo" no significa nada, de por sí.

Durante las notas haré referencia a varias herramientas informáticas. De momento:

Matlab: Aunque procuraré que todo el código que aparezca pueda utilizarse con Octave, haré referencia casi exclusivamente a Matlab, que es el programa al que están acostumbrados los alumnos en la Universidad de Oviedo y que usarán presumiblemente en otras asignaturas.

Wolfram Alpha: Es necesario conocer esta herramienta, pues permite gran cantidad de operaciones (matemáticas y no) en una simple página web.

Excel: Bastantes de los algoritmos pueden implementarse en hojas de cálculo. Los alumnos deberían intentar hacerlo: así se darán cuenta de la potencia que tiene una herramienta tan común (y que ya saben manejar).

Quizás lo que más me interesa es que los alumnos

- Aprendan a enunciar algoritmos *con precisión*, con la conciencia de que un algoritmo debe ser *claro*, *finito* y *debe terminar.*
- Sean capaces de seguir un algoritmo y de entender su funcionamiento.

Por supuesto, habrá que entender los algoritmos que se expongan y ser capaces de seguirlos paso a paso (esto es un requisito que estimo imprescindible). Todos los que se explicarán serán de memorización sencilla porque o bien serán muy breves (la mayoría) o bien tendrán una expresión geométrica evidente (y por tanto, su expresión verbal será fácilmente memorizable). La precisión formal es mucho más importante en esta asignatura que las "ideas geométricas", porque precisamente esta asignatura estudia los métodos *formales* de resolver ciertos problemas, no su expresión geométrica o intuitiva.

Las secciones ó párrafos indicados como [Contenido no esencial] solo se incluyen como información extra: no es mi intención evaluar al alumno sobre su contenido pero pienso que son lo suficientemente interesantes como para que a alguien le sirvan para aprender un poco más.

CAPÍTULO 1

Aritmética finita y análisis del error

Se revisan muy rápidamente las notaciones exponenciales, la coma flotante de doble precisión, la noción de error y algunas fuentes comunes de errores en los cómputos.

1. Notación exponencial

Los numeros reales pueden tener un número finito o infinito de cifras. Cuando se trabaja con ellos, siempre se han de aproximar a una cantidad con un número finito (y habitualmente pequeño) de dígitos. Además, conviene expresarlos de una manera uniforme para que su magnitud y sus dígitos significativos sean fácilmente reconocibles a primera vista y para que el intercambio de datos entre máquinas sea determinista y eficiente. Esto ha dado lugar a la notación conocida como *científica* o *exponencial*. En estas notas se explica de una manera somera; se pretende más transmitir la idea que describir con precisión las reglas de estas notaciones (pues tienen demasiados casos particulares como para que compense una explicación). A lo largo de este curso, utilizaremos las siguientes expresiones:

Definición 1. Una *notación exponencial* de un número real es una expresión del tipo

$$\pm A.B \times 10^C$$
$$10^C \times \pm A.B$$
$$\pm A.BeC$$

donde A, B y C son números naturales (que pueden ser nulos), $\pm$ indica un signo (que puede omitirse si es $+$). Cualquiera de esas expresiones se refiere al número $(A + 0.B) \cdot 10^C$ (donde $0.B$ es el número "cero coma B...".

Por ejemplo:

- El número 3.123 es el propio 3.123.
- El número $0.01e-7$ es 0.000000001 (hay ocho ceros después de la coma y antes del 1).

- El número $10^3 \times -2.3$ es -2300.
- El número $-23.783e-1$ es -2.3783.

Pero por lo general, la notación científica asume que el número a la derecha del punto decimal es un solo dígito no nulo:

DEFINICIÓN 2. La notación exponencial estándar es la notación exponencial en la que A es un número entre 1 y 9.

Finalmente, las máquinas almacenan —generalmente— los números de una manera muy concreta, en *coma flotante*:

DEFINICIÓN 3. Un formato en *coma flotante* es una especificación de una notación exponencial en la que la longitud de A más la longitud de B es una cantidad fija y en la que los exponentes (el número C) varían en un rango determinado.

Por tanto, un formato en coma flotante solo puede expresar una cantidad finita de números (aquellos que puedan escribirse según la especificación del formato).

2. Un ejemplo detallado de coma flotante

Los ordenadores trabajan con números decimales utilzando (en binario, pero para la explicación es mejor utilizar base 10) dos formatos:

- Punto "fijo" (fixed point): los números se almacenan en una lista de k_e elementos para la parte entera y k_d para la parte decimal, donde ambos números k_e y k_d están fijos de antemano: Las operaciones con este tipo de aritmética tienen

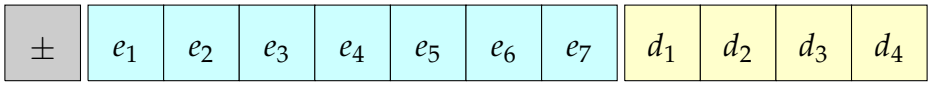

FIGURA 1. Punto fijo con 7 dígitos enteros y 4 decimales (y uno de signo).

siempre la misma precisión "absoluta" (cada operación se realiza exactamente con k_d decimales). Los valores extremos que pueden almacenarse son $\pm 10^{k_e}$ y los valores más pequeños significativos son $\pm 10^{-k_d}$. Cuando un valor "externo" se almacena en una variable de este tipo, los dígitos decimales más allá del k_d se truncan. Por tanto, se puede llegar a perder una precisión de $\simeq 10^{-k_d}$ como mucho. El problema de este tipo de aritmética es doble:

(1) Los extremos máximos por lo general son insuficientes (en binario, 32 dígitos solo cubren unos 4×10^9 valores, cuatro mil millones, que en seguida se queda corte).
(2) La resolución decimal puede también quedarse corta en problemas de muy alta precisión (que hoy en día son habituales en física e ingenierías).

- "Punto flotante" (usualmente llamado *coma* flotante, en castellano), (floating point): los números se almacenan en una pareja formada por k_m elementos de la llamada "mantisa" y k_e elementos del "exponente" (aparte del signo). En principio la estructura es similar al punto fijo: Un número en esta

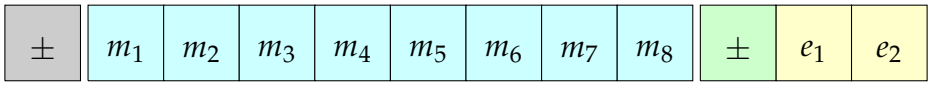

FIGURA 2. Punto flotante con mantisa de 8 dígitos enteros y exponente de 3 (dos más signo), y un signo.

expresión (la de la figura 2) es igual a

$$\pm 0.m_1 m_2 \cdots m_8 \times 10^{\pm e_1 e_2}$$

donde el signo en cada parte será el que esté especificado. Se exige que el primer dígito de la mantisa sea no nulo ($m_1 \neq 0$). En binario, el signo $+$ es un 0 y el singo $-$ es un 1. Por definición, el valor máximo que se puede almacenar en esta coma flotante es $\pm 0.99999999 \times 10^{99}$ (un valor enorme), mientras que en decimal, los valores ínfimos son $\pm 0.1 \times 10^{-99}$ (un cero seguido de 100 ceros y un 1 tras la coma). Como se ve, este tipo de almacenamiento permite cálculos en un rango mucho más amplio que el punto fijo. Esto hace que sea posible utilizar este tipo de aritmética en problemas en que los valores de las variables sean muy altos y en problemas en que sean muy bajos, sin perder *precisión relativa*. Cada vez que un número (real) se almacena en esa coma flotante, al ser de la forma

$$0.m_1 \cdots m_8 \times 10^e$$

el error que se comete es del orden de 0.00000001×10^e, es decir, 10^{e-8}: por tanto, si e es grande, el error *absoluto* es grande, y si e es pequeño, el error *absoluto* es pequeño. Lo que caracteriza la coma flotante es que *al almacenar un número en coma flotante, el error relativo que se comete es siempre del mismo orden*.

Pero para ello necesitamos dos definiciones. Supongamos que x_0 es un número que se quiere calcular (o almacenar) y que $\overline{x}_0$ es una aproximación.

DEFINICIÓN 4. El *error absoluto* cometido al utilizar $\overline{x}_0$ en lugar de x_0 es: $|x_0 - \overline{x}_0|$. El *error relativo* es

$$\frac{|x_0 - \overline{x}_0|}{|x_0|}$$

(que solo tiene sentido si $x_0 \neq 0$, claro).

El error absoluto mide la "distancia" que hay entre el valor y su aproximación. El error relativo mide el tamaño del error "comparado con" el valor exacto. Por lo general, el error relativo es más informativo que el absoluto, pero es importante tener en cuenta ambos, *siempre*.

Por tanto, en coma flotante, el *error relativo* cometido al truncar un valor y almacenarlo es *siempre* menor que 10^{-k_m} (en decimal, en binario será con un 2): viene dado por la longitud de la mantisa. El problema es que el error absoluto puede ser muy grande (del orden de $10^{99-8} = 10^{97}$, cuando se almacenan números de ese tamaño.

EJEMPLO 1 (Desbordamiento). Se tiene un controlador digital que almacena números en punto fijo (decimal) con 5 dígitos enteros y 4 decimales. El contador comienza en 0 y se aumenta en 0.001 cada vez (suma milésimas de segundo). ¿Cuánto tardará en ocurrir un desbordamiento?

Como se almacena en punto fijo con $5 + 4$ dígitos, el mayor valor posible es 99999.9999. El desbordamiento se produce cuando el contador llegue a 10^5, por tanto, cuando pasen

$$10^5 / 10^{-3} = 10^8$$

milésimas de segundo, es decir, 10^5 segundos, que son

$$\frac{10^5}{24 \cdot 60 \cdot 60} \simeq 1.157$$

días. Es decir, al cabo de 1 día y medio se producirá un error *a ciencia cierta*.

EJEMPLO 2 (Pérdida de precisión). El mismo contador que antes se almacena en un registro en coma flotante con 5 dígitos de mantisa y 4 de exponente (contando el signo). ¿Cuándo se perderá precisión?

El número 0.001 en nuestra coma flotante es 1×10^{-3}. Para que las milésimas sigan teniendo relevancia, el número mayor que se puede

considerar que tenga cinco cifras es:

$$99.999 = 0.99999 \times 10^2.$$

Por tanto, cuando el contador llegue a 99.999, se producirá una pérdida de precisión (en este caso total: ya no hay milésimas y el contador se queda parado en 100). Es decir, al cabo de 100 segundos el contador falla *totalmente*.

El ejemplo 2 es extremo (el formato de coma flotante utilizado es muy burdo porque la mantisa es muy pequeña) pero es lo que le ocurrió al sistema Patriot en la Guerra del Golfo, esencialmente.

La precisión simple *sigue existiendo*: por un lado, las entidades financieras *tienen obligación* de trabajar con un número específico de decimales. Por otro lado, los *microcontroladores* están diseñados para utilizar punto fijo porque la implementación de las operaciones básicas (y, sobre todo, de la suma y la resta) es mucho más sencilla (y, por tanto, más económica).

El formato estándar de coma flotante es el documento conocido como IEEE-754, que posee varias actualizaciones (el original es de 1987).

2.1. [Contenido no esencial] El formato binario de doble precisión IEEE-754. El formato de doble precisión es la especificación del IEEE sobre la representación de números reales en secuencias de 16, 32, 64, 128 bits (y más), y su representación en formato decimal. Se explican a continuación las propiedades principales de la doble precisión en binario.

Para expresar números en doble precisión se utilizan 64 bits, es decir, 64 dígitos *binarios*. El primero indica el signo (un 0 es signo positivo, un 1, signo negativo). Los 11 siguientes se utilizan para representar el exponente como se indicará, y los 52 restantes se utilizan para lo que se denomina la *mantisa*. De esta manera, un número en doble precisión tiene tres partes: s, el signo (que será 0 ó 1), e, el exponente (que variará entre 0 y 2047 (pues $2^{11} = 2048$), y m, un número de 52 bits. Dados tres datos s, e, m, el número real N que representan es:

- Si $e \neq 0$ y $e \neq 2047$ (si el exponente no es ningún valor extremo), entonces

  $$N = (-1)^s \times 2^{e-1023} \times 1.m,$$

  donde $1.m$ indica "uno-coma-m" en binario. Nótese, y esto es lo importante, que el exponente *no es el número representado por los* 11 *bits de* e, sino que "se desplaza hacia la derecha".

Un $e = 01010101011$, que en decimal es 683 representa realmente la potencia de 2 $2^{683-1023} = 2^{-340}$. Los e cuyo primer bit es cero corresponden a potencias negativas de 2 y los que tienen primer bit 1 a potencias positivas ($2^{10} = 1024$).

- Si $e = 0$ entonces, si $m \neq 0$ (si hay mantisa):

$$N = (-1)^s \times 2^{-1023} \times 0.m,$$

donde $0.m$ indica "cero-coma-m" en binario.
- Los ceros con signo: si $e = 0$ y $m = 0$, el número es $+0$ o -0, dependiendo de s. (es decir, el 0 tiene signo).
- El caso de $e = 2047$ (es decir, los 11 digitos del exponente son 1) se reserva para codificar $\pm\infty$ y otros objetos que se denominan NaN (Not-a-Number, que indica que una operación es ilegítima, como $1/0$ o $\log(-2)$ o $\text{acos}(3)$, en una operación con números reales).

En realidad, el estándar es mucho más largo y completo, como es natural, e incluye una gran colección de requisitos para los sistemas electrónicos que realicen cálculos en coma flotante (por ejemplo, especifica cómo se han de truncar los resultados de las operaciones fundamentales para asegurar que si se puede obtener un resultado exacto, *se obtenga*).

Las ventajas de la coma flotante (y en concreto de la doble precisión) son, aparte de su estandarización, que permite a la vez operar con números muy pequeños (el número más pequeño que puede almacenar es $2^{-1023} \simeq 10^{-300}$) y números muy grandes (el mayor es alrededor de $2^{1023} \simeq 10^{300}$). La contrapartida es que, si se trabaja simultáneamente con ambos tipos de datos, los pequeños pierden precisión y *desaparecen* (se produce un error de cancelación o de truncación). Pero *si se tiene cuidado*, es un formato enormemente útil y versátil.

2.2. [Contenido no esencial] Conversión de base decimal a base dos y vuelta. Explico a continuación, para quien no sepa, cómo se trasnforma un número en base decimal a base dos (binario) y al revés.

En el pseudocódigo de estas notas, se utilizará la siguiente notación: la expresión $x \leftarrow a$ significa que x es una variable, a representa un valor (así que puede ser un número u otra variable) y a x se le asigna el valor de a. La expresión $u = c$ es la expresión *condicional* ¿es el valor designado por u igual al valor designado por c? Además, $m//n$ indica el cociente de dividir el número entero $m \geq 0$ entre el

número entero $n > 0$ y $m\%n$ es el *resto* de dicha divisón. Es decir,

$$m = (m//n) \times n + (m\%n).$$

Finalmente, si x es un número real $x = A.B$, la expresión $\{x\}$ indica la *parte fraccionaria de* x, es decir, $0.B$.

Algoritmo 1 (Paso de decimal a binario, sin signo)

Input: $A.B$ un número en decimal, con B finito, k un entero positivo (que indica el número de digitos que se desea tras la coma en binario)
Output: $a.b$ (el número x en binario, truncado hasta 2^{-k})
if $A = 0$ **then**
  $a \leftarrow 0$ y calular solo la Parte Decimal
end if
⋆Parte Entera
$i \leftarrow -1$, $n \leftarrow A$
while $n > 0$ **do**
  $i \leftarrow i + 1$
  $x_i \leftarrow n\%2$
  $n \leftarrow n//2$
end while
$a \leftarrow x_i x_{i-1} \dots x_0$ [`la secuencia de restos en orden inverso`]
⋆Parte Decimal
if $B = 0$ **then**
  $b \leftarrow 0$
  return $a.b$
end if
$i \leftarrow 0$, $n \leftarrow 0.B$
while $n > 0$ **and** $i < k$ **do**
  $i \leftarrow i + 1$
  $m \leftarrow 2n$
  if $m \geq 1$ **then**
    $b_i \leftarrow 1$
  else
    $b_i \leftarrow 0$
  end if
  $n \leftarrow \{m\}$ [`la parte decimal de` m]
end while
$b \leftarrow b_1 b_2 \dots b_i$
return a.b

El Algoritmo 1 es una manera de pasar un número $A.B$ en decimal con un número finito de cifras a su forma binaria. El Algoritmo 2 se utiliza para realizar la operación inversa. Téngase en cuenta que, puesto que hay números con una cantidad finita de digitos decimales que no se pueden expresar con una cantidad finita de digitos en binario (el ejemplo más obvio es 0.1), se ha de especificar un número de cifras decimales para la salida (así que no se obtiene necesariamente el mismo número, sino una truncación).

Algoritmo 2 (Paso de binario a decimal, sin signo)

Input: $A.B$, un número positivo en binario, k un número entero no negativo (el número de decimales que se quiere utilizar en binario).
Ouput: $a.b$, la truncación hasta precisión 2^{-k} en decimal del número $A.B$

⋆PARTE MAYOR QUE 0
Se escribe $A = A_r A_{r-1} \dots A_0$ (los dígitos binarios)
$a \leftarrow 0, i \leftarrow 0$
while $i \leq r$ **do**
  $a \leftarrow a + A_i \times 2^i$
  $i \leftarrow i + 1$
end while
⋆PARTE DECIMAL
if $B = 0$ **then**
  return $a.0$
end if
$b \leftarrow 0, i \leftarrow 0$
while $i \leq k$ **do**
  $i \leftarrow i + 1$
  $b \leftarrow b + B_i \times 2^{-i}$
end while
return $a.b$

Pasar de binario a decimal es "más sencillo" pero requiere ir sumando: no obtenemos un dígito por paso, sino que se han de sumar potencias de 2. Se describe este proceso en el Algoritmo 2. Nótese que el número de decimales de la salida no está especificado (se podría, pero solo haría el algoritmo más complejo). Finalmente, en todos los pasos en que se suma $A_i \times 2^i$ o bien $B_i \times 2^{-i}$, tanto A_i como B_i son o bien 0 o bien 1, así que ese producto solo significa "sumar o no sumar" la potencia de 2 correspondiente (que es lo que expresa un dígito binario, al fin y al cabo).

3. El error, definiciones básicas

Siempre que se opera con números con una cantidad finita de cifras y siempre que se toman medidas en la realidad, hay que tener en cuenta que contendrán, casi con certeza, un error. Esto no es grave. Lo prioritario es *tener una idea de su tamaño* y saber que según vayan haciéndose operaciones puede ir propagándose. Al final, lo que importa es *acotar el error absoluto*, es decir, conocer un valor (la cota) que sea mayor que el error cometido, para saber *con certeza* cuánto, como mucho, dista el valor real del valor obtenido.

En lo que sigue, se parte de un valor exacto x (una constante, un dato, la solución de un problema...) y de una aproximación, $\tilde{x}$.

DEFINICIÓN 5. Se llama *error absoluto* cometido al utilizar $\tilde{x}$ en lugar de x al valor absoluto de la diferencia: $|x - \tilde{x}|$.

Pero, salvo que x sea 0, uno está habitualmente más interesado en *el orden de magnitud* del error, es decir, "cuánto se desvía $\tilde{x}$ de x en proporción a x":

DEFINICIÓN 6. Se llama *error relativo* cometido al utilizar $\tilde{x}$ en lugar de x, siempre que $x \neq 0$, al cociente

$$\frac{|\tilde{x} - x|}{|x|}$$

(que es siempre un número positivo).

No vamos a utilizar una notación especial para ninguno de los dos errores (hay autores que los llaman Δ y δ, respectivamente, pero cuanta menos notación innecesaria, mejor).

EJEMPLO 3. La constante π, que es la razón entre la longitud de la circunferencia y su diámetro, es, aproximadamente 3.1415926534+ (con el + final se indica que es mayor que el número escrito hasta la última cifra). Supongamos que se utiliza la aproximación $\tilde{\pi} = 3.14$. Se tiene:

- El error absoluto es $|\pi - \tilde{\pi}| = 0.0015926534+$.
- El error relativo es $\frac{|\pi - \tilde{\pi}|}{\pi} \simeq 10^{-4} \times 5.069573$.

Esto último significa que se produce un error de 5 diezmilésimas por unidad (que es más o menos 1/2000) cada vez que se usa 3.14 en lugar de π. Por tanto, si se suma 3.14 dos mil veces, el error cometido al usar esa cantidad en lugar de $2000 \times \pi$ es aproximadamente de 1. Este es el significado interesante del error relativo: su inverso es el número de veces que hay que sumar $\tilde{x}$ para que el error acumulado sea 1 (que, posiblemente, será el orden de magnitud del problema).

Antes de continuar analizando errores, conviene definir las dos maneras más comunes de escribir números utilizando una cantidad fija de dígitos: el *truncamiento* y el *redondeo*. No se van a dar las definiciones precisas porque en este caso la precisión parece irrelevante. Se parte de un número real (posiblemente con un número infinito de cifras):

$$x = a_1 a_2 \dots a_r . a_{r+1} \dots a_n \dots$$

Se definen:

DEFINICIÓN 7. El *truncamiento de* x a k cifras (significativas) es el número $a_1 a_2 \dots a_k 0 \dots 0$ (un número entero), si $k \leq r$ y si no, $a_1 \dots a_r . a_{r+1} \dots a_k$ si $k > r$. Se trata de *cortar* las cifras de x y poner ceros si aun no se ha llegado a la coma decimal.

DEFINICIÓN 8. El *redondeo de* x a k cifras (significativas) es el siguiente número:

- Si $a_{k+1} < 5$, entonces el redondeo es igual al truncamiento.
- Si $5 \leq a_{k+1} \leq 9$, entonces el redondeo es igual al truncamiento más 10^{r-k+1}.

Este redondeo se denomina *redondeo hacia más infinito*, porque siempre se obtiene un número *mayor o igual* que el truncamiento.

El problema con el redondeo es que pueden cambiar todas las cifras. La gran ventaja es que el error que se comete al redondear es menor que el que se comete al truncar (puede ser hasta de la mitad):

EJEMPLO 4. Si $x = 178.299$ y se van a usar 4 cifras, entonces el truncamiento es $x_1 = 178.2$, mientras que el redondeo es 178.3. El error absoluto cometido en el primier caso es 0.099, mientas que en el segundo es 0.001.

EJEMPLO 5. Si $x = 999.995$ y se van a usar 5 cifras, el truncamiento es $x_1 = 999.99$, mientras que el redondeo es 1000.0. Pese a que todas las cifras son diferentes, el error cometido es el mismo (en este caso, 0.005) y esto es lo importante, no que los digitos "coincidan".

¿Por qué se habla entonces de truncamiento? Porque cuando uno trabaja en coma flotante, es inevitable que se produzca truncamiento (pues el número de dígitos es finito) y se hace necesario tenerlo en cuenta. Actualmente (2012) la mayoría de programas que trabajan en doble precisión, realmente lo hacen con muchos más dígitos internamente y posiblemente al reducir a doble precisión redondeen. Aun así, los truncamientos se producen en algún momento (cuando se sobrepasa la capacidad de la máquina).

3.1. [Contenido no esencial] Fuentes del error. El error se origina de diversas maneras. Por una parte, cualquier medición está sujeta a él (por eso los aparatos de medida se venden con un margen estimado de precisión); esto es intrínseco a la naturaleza y lo único que puede hacerse es tenerlo en cuenta y saber su magnitud (conocer una buena cota). Por otro lado, las operaciones realizadas en aritmética finita dan lugar tanto a la propagación de los errores como a la aparición de nuevos, precisamente por la cantidad limitada de dígitos que se pueden usar.

Se pueden enumerar, por ejemplo, las siguientes fuentes de error:

- El error *en la medición*, del que ya se ha hablado. Es inevitable.
- El error de *truncamiento*: ocurre cuando un número (dato o resultado de una operación) tiene más dígitos que los utilizados en las operaciones y se "olvida" una parte.
- El error de *redondeo*: ocurre cuando, por la razón que sea, se redondea un número a una precisión determinada.
- El error de *cancelación*: se produce cuando una operación da lugar a errores relativos mucho más grandes que los absolutos. Habitualmente tiene lugar al sustraer de magnitud muy similar. El ejemplo canónico de esto aparece en la *inestabilidad de la ecuación cuadrática*.
- El error de *acumulación*: se produce al acumular (sumar, esencialmente) pequeños errores del mismo signo *muchas veces*. Es lo que ocurrió en el suceso de los Patriot en febrero de 1991, en la operación *Tormenta del Desierto*[1].

En la aritmética de precisión finita, todos estos errores ocurren. Conviene quizás conocer las siguientes reglas (que son los peores casos posibles):

- Al sumar *números del mismo signo*, el error absoluto puede ser la suma de los errores absolutos y el error relativo, lo mismo.
- Al sumar *números de signo contrario*, el error absoluto se comporta como en el caso anterior, pero *el error relativo puede aumentar de manera incontrolada*: $1000.2 - 1000.1$ solo tiene una cifra significativa (así que el error relativo puede ser de hasta un 10%, mientras que en los operandos, el error relativo era de 1×10^{-4}.

[1] Pueden consultarse el siguiente documento de la Univ. de Texas:
`http://www.cs.utexas.edu/~downing/papers/PatriotA1992.pdf`
y el informe oficial: `http://www.fas.org/spp/starwars/gao/im92026.htm`

- Al multiplicar, el error absoluto tiene la magnitud del mayor factor por el error en el otro factor. Si los factores son de la misma magnitud, puede ser el doble del mayor error absoluto por el mayor factor. El error relativo es de la misma magnitud que el mayor error relativo (y si son de la misma magnitud, su suma).
- Al dividir *por un número mayor o igual que* 1, el error absoluto es aproximadamente el error absoluto del numerador partido por el denominador y el error relativo es el error relativo del numerador (esencialmente como en la multiplicación). Al dividir por números cercanos a cero, lo que ocurre es que se perderá precisión absoluta y si luego se opera con números de magnitud similar, el error de cancelación puede ser importante. Compárense las siguientes cuentas:

$$26493 - \frac{33}{0.0012456} = -0.256 \text{ (el resultado buscado)} .$$

$$26493 - \frac{33}{0.001245} = -13.024$$

Si bien el error relativo de truncación es solo de 4×10^{-4} (media milésima), el error relativo del resultado es de 49.8 (es decir, el resultado obtenido es 50 veces más grande que el que debía ser). Este (esencialmente) es el problema fundamental de los métodos de resolución de sistemas que utilizan divisiones (como el de Gauss y por supuseto, el de Cramer). Cuando se explique el método de Gauss, se verá que es conveniente buscar la manera de hacer las divisiones con los divisores más grandes posibles (estrategias de *pivotaje*).

4. [Contenido no esencial] Acotar el error

Como se dijo antes, lo importante no es saber con precisión el error cometido en una medición o al resolver un problema, pues con casi certeza, esto será imposible, sino *tener una idea* y, sobre todo, *tener una buena cota*: saber que el error absoluto cometido es *menor* que una cantidad y que esta cantidad sea lo más ajustada posible (sería inútil, por ejemplo, decir que 2.71 es una aproximación de e con un error menor que 400).

Así pues, lo único que se podrá hacer realmente será estimar un número mayor que el error cometido y *razonablemente* pequeño. Eso es *acotar*.

4.1. Algunas cotas. Lo primero que ha de hacerse es acotar el error si se conoce una cantidad con una variación aproximada. Es decir, si se sabe que $x = a \pm \epsilon$, donde $\epsilon > 0$, el error que se comete al utilizar a en lugar de x es desconocido (esto es importante) pero es *a lo sumo*, ϵ. Por lo tanto, el error relativo que se comete es, *a lo sumo* el error absoluto dividido *entre el menor de los valores posibles en valor absoluto*: téngase en cuenta que esto es delicado, pues si $a - \epsilon < 0$ pero $a + \epsilon > 0$ entonces *no se puede acotar el error relativo* porque x podría ser 0 (y, como ya se explicó, el error relativo solo tiene sentido para cantidades no nulas).

Ejemplo 6. Si se sabe que $\pi = 3.14 \pm 0.01$, se sabe que el error absoluto máximo cometido es 0.01, mientras que el error relativo es como mucho

$$\frac{0.01}{|3.13|} \simeq .003194$$

(más o menos $1/300$).

Téngase en cuenta que, si lo que se busca es una cota superior y se ha de realizar una división, ha de tomarse el divisor *lo más pequeño posible* (cuanto menor es el divisor, mayor es el resultado). Con esto se ha de ser muy cuidadoso.

Las reglas de la sección 3.1 son esenciales si se quiere acotar el error de una serie de operaciones aritméticas. Como se dice allí, ha de tenerse un cuidado muy especial cuando se realizan divisiones con números menores que uno, pues posiblemente se llegue a resultados inútiles (como que "el resultado es 7 pero el error absoluto es 23").

CAPÍTULO 2

Ecuaciones no lineales: soluciones numéricas

Se estudia en este capítulo el problema de resolver una ecuación no lineal de la forma $f(x) = 0$, dada la función f y ciertas condiciones inciales.

Cada uno de los algoritmos que se estudia tiene sus ventajas e inconvenientes; no hay que desechar ninguno a priori simplemente porque "sea muy lento" —esta es la tentación fácil al estudiar la convergencia de la bisección. Como se verá, el "mejor" algoritmo de los que se estudiarán —el de Newton-Raphson— tiene dos pegas importantes: puede converger (o incluso no hacerlo) lejos de la condición inicial (y por tanto volverse inútil si lo que se busca es una raíz "cerca" de un punto) y además, requiere computar el valor de la derivada de f en cada iteración, lo cual puede tener un coste computacional excesivo.

Antes de proseguir, nótese que todos los algoritmos se describen con condiciones de parada *en función del valor de* f, no con condiciones de parada tipo *Cauchy*. Esto se hace para simplificar la exposición. Si se quiere garantizar un número de decimales de precisión, es necesario o bien evaluar cambios de signo cerca de cada punto o bien hacer un estudio muy detallado de la derivada, que está más allá del alcance de este curso a mi entender.

1. Introducción

Calcular raíces de funciones —y especialmente de polinomios— es uno de los problemas clásicos de las Matemáticas. Hasta hace unos doscientos años se pensaba que cualquier polinomio podía "resolverse algebraicamente", es decir: dada una ecuación polinómica $a_n x^n + \cdots + a_1 x + a_0 = 0$ habría una fórmula "con radicales" que daría todos los ceros de ese polinomio, como la fórmula de la ecuación de segundo grado pero para grado cualquiera. La Teoría de Galois mostró que esta idea no era más que un bonito sueño: el polinomio general de quinto grado no admite una solución en función de radicales.

De todos modos, buscar una *fórmula cerrada* para resolver una ecuaión no es más que una manera de postponer el problema de la aproximación, pues al fin y al cabo (y esto es importante):

Las únicas operaciones que se pueden hacer con precisión total siempre son la suma, la resta y la multiplicación.

Ni siquiera es posible dividir y obtener resultados exactos[1].

En fin, es lógico buscar maneras lo más precisas y rápidas posibles de resolver ecuaciones no lineales y que no requieran operaciones mucho más complejas que la propia ecuación (claro).

Se distinguiran en este capítulo dos tipos de algoritmos: los "geométricos", por lo que se entienden aquellos que se basan en una idea geométrica simple y el "de punto fijo", que requiere desarrollar la idea de *contractividad*.

Antes de proseguir, hay dos requerimientos esenciales en cualquier algoritmo de búsqueda de raíces:

- Especificar una *precisión*, es decir, un $\epsilon > 0$ tal que si $|f(c)| < \epsilon$ entonces se asume que c es una raíz. Esto se debe a que, al utilizar un número finito de cifras, es posible que *nunca* ocurra que $f(c) = 0$, siendo c el valor que el algoritmo obtiene en cada paso.
- Incluir una *condición de parada*. Por la misma razón o por fallos del algoritmo o porque la función no tiene raíces, podría ocurrir que nunca se diera que $|f(c)| < \epsilon$. Es *imprescindible* especificar un momento en que el algoritmo se detiene, *de manera determinista*. En estas notas la condición de parada será siempre un número de repeticiones del algoritmo, $N > 0$: si se sobrepasa, se devuelve un "error".

2. Algoritmos "geométricos"

Si suponemos que la función f cuya raíz se quiere calcular es continua en un intervalo compacto $[a, b]$ (lo cual es una suposición razonable), el Teorema de Bolzano puede ser de utilidad, si es que sus hipótesis se cumplen. Recordemos:

TEOREMA (Bolzano). *Sea $f : [a, b] \to \mathbb{R}$ una función continua en $[a, b]$ y tal que $f(a)f(b) < 0$ (es decir, cambia de signo entre a y b). Entonces existe $c \in (a, b)$ tal que $f(c) = 0$.*

[1] Podría argumentarse que si se dividen números racionales se puede escribir el resultado como decimales periódicos, pero esto sigue siendo "postponer el problema".

El enunciado afirma que si f cambia de signo en un intervalo cerrado, entonces al menos hay una raíz. Se podría pensar en muestrear el intervalo (por ejemplo en anchuras $(b-a)/10^i$) e ir buscando subintervalos más pequeños en los que sigue cambiando el signo, hasta llegar a un punto en que la precisión hace que hayamos localizado una raíz "o casi". En realidad, es más simple ir subdividiendo en mitades. Esto es el algoritmo de bisección.

3. El algoritmo de bisección

Se parte de una función f, continua en un intervalo $[a, b]$, y de los valores a y b. Como siempre, se ha de especificar una precisión $\epsilon > 0$ (si $|f(c)| < \epsilon$ entonces c es una "raíz aproximada") y un número máximo de ejecuciones del bucle principal $N > 0$. Con todos estos datos, se procede utilizando el Teorema de Bolzano:

Si $f(a)f(b) < 0$, puesto que f es continua en $[a, b]$, tendrá alguna raíz. Tómese c como el punto medio de $[a, b]$, es decir, $c = \dfrac{a+b}{2}$. Hay tres posibilidades:

- O bien $|f(c)| < \epsilon$, en cuyo caso se termina y se devuelve el valor c como raíz aproximada.
- O bien $f(a)f(c) < 0$, en cuyo caso se sustituye b por c y se repite todo el proceso.
- O bien $f(b)f(c) < 0$, en cuyo caso se sustituye a por c y se repite todo el proceso.

La iteración se realiza como mucho N veces (para lo cual hay que llevar la cuenta, obviamente). Si al cabo de N veces no se ha obtenido una raíz, se termina con un mensaje de error.

El enunciado formal es el Algoritmo 3. Nótese que se utiliza un signo nuevo: $a \leftrightarrow b$, que indica que los valores de a y b se intercambian.

4. El algoritmo de Newton-Raphson

Una idea geométrica clásica (y de teoría clásica de la aproximación) es, en lugar de calcular una solución de una ecuación $f(x) = 0$ directamente, utilizar la *mejor aproximación lineal a* f, que es la recta tangente en un punto. Es decir, en lugar de calcular una raíz de f, utilizar un valor $(x, f(x))$ para trazar la recta tangente a la gráfica de f en ese punto y resolver la ecuación dada por el corte de esta tangente con el eje OX, en lugar de $f(x) = 0$. Obviamente, el resultado no será una raíz de f, pero *en condiciones generales*, uno espera que se aproxime algo (la solución de una ecuación aproximada *debería ser*

Algoritmo 3 (Algoritmo de Bisección)

Input: una función $f(x)$, un par de números reales, a, b con $f(a)f(b) < 0$, una tolerancia $\epsilon > 0$ y un límite de iteraciones $N > 0$

Output o bien un mensaje de error o bien un número real r entre a y b tal que $|f(c)| < \epsilon$ (una raíz aproximada)

⋆INICIO
  $r \leftarrow \frac{a+b}{2}$
  $i \leftarrow 0$
  while $|f(r)| \geq \epsilon$ **and** $i < N$ **do**
    if $f(r)f(b) < 0$ **then**
      $a \leftarrow r$ [`intervalo` $[r, b]$]
    else
      $b \leftarrow r$ [`intervalo` $[a, r]$]
    end if
    $i \leftarrow i + 1$
    $r \leftarrow \frac{a+b}{2}$
  end while
  if $i \geq N$ **then**
    return ERROR
  end if
  return r

una solución aproximada). Si se repite el proceso de aproximación mediante la tangente, uno espera ir acercándose a una raíz de f. Esta es la idea del algoritmo de Newton-Raphson.

Recuérdese que la ecuación de la recta que pasa por el punto (x_0, y_0) y tiene pendiente b es:

$$Y = b(X - x_0) + y_0$$

así que la ecuación de la recta tangente a la gráfica de f en el punto $(x_0, f(x_0))$ es (suponiendo que f es derivable en x_0):

$$Y = f'(x_0)(X - x_0) + f(x_0).$$

El punto de corte de esta recta con el eje OX es, obviamente,

$$(x_1, y_1) = \left(x_0 - \frac{f(x_0)}{f'(x_0)}, 0 \right)$$

suponiendo que existe (es decir, que $f'(x_0) \neq 0$).

La iteración, si se supone que se ha calculado x_n es, por tanto:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Así que se tiene el Algoritmo 4. En el enunciado solo se especifica un posible lugar de error para no cargarlo, pero hay que tener en cuenta que cada vez que se evalúa f o cada vez que se realiza una operación (cualquiera) podría ocurrir un error de coma flotante. Aunque en los enunciados que aparezcan de ahora en adelante no se mencionen todos, el alumno debe ser consciente de que cualquier implementación ha de emitir una excepción (*raise an exception*) en caso de que haya un fallo de coma flotante.

Algoritmo 4 (Algoritmo de Newton-Raphson)

Input: una función $f(x)$ *derivable*, una semilla $x_0 \in \mathbb{R}$, una tolerancia $\epsilon > 0$ y un límite de iteraciones $N > 0$
Ouput: o bien un mensaje de error o bien un número real c tal que $|f(c)| < \epsilon$ (es decir, una raíz aproximada)

⋆INICIO
$i \leftarrow 0$
while $|f(x_i)| \geq \epsilon$ **and** $i \leq N$ **do**
  $x_{i+1} \leftarrow x_i - \frac{f(x_i)}{f'(x_i)}$ `[posibles NaN e ∞]`
  $i \leftarrow i + 1$
end while
if $i > N$ **then**
  return ERROR
end if
return $c \leftarrow x_i$

5. El algoritmo de la secante

El algoritmo de Newton-Raphson contiene la evaluación $\frac{f(x_n)}{f'(x_n)}$, para la que hay que calcular el valor de dos expresiones: $f(x_n)$ y $f'(x_n)$, lo cual puede ser exageradamente costoso. Además, hay muchos casos en los que ni siquiera se tiene información real sobre $f'(x)$, así que puede ser hasta utópico pensar que el algoritmo es utilizable.

Una solución a esta pega es aproximar el cálculo de la derivada utilizando la idea geométrica de que *la tangente es el límite de las secantes*; en lugar de calcular la recta tangente se utiliza una aproximación

con dos puntos que se suponen "cercanos". Como todo el mundo sabe, la derivada de $f(x)$ en el punto c es, si existe, el límite

$$\lim_{h\to 0} \frac{f(c+h)-f(c)}{h}.$$

En la situación del algoritmo de Newton-Raphson, si en lugar de utilizar un solo dato x_n, se recurre a los dos anteriores, x_n y x_{n-1}, se puede pensar que se está aproximando la derivada $f'(x_n)$ por medio del cociente

$$f'(x_n) \simeq \frac{f(x_n)-f(x_{n-1})}{x_n - x_{n-1}},$$

y por tanto, la fórmula para calcular el término x_{n+1} quedaría

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

y con esto se puede ya enunciar el algoritmo correspondiente. Téngase en cuenta que, para comenzarlo, hacen falta *dos semillas*, no basta con una. El factor de $f(x_n)$ en la iteración es justamente el inverso de la aproximación de la derivada en x_n utilizando el punto x_{n-1} como $x_n + h$.

Algoritmo 5 (Algoritmo de la secante)

Input: Una función $f(x)$, una tolerancia $\epsilon > 0$, un límite de iteraciones $N > 0$ y *dos* semillas $x_{-1}, x_0 \in \mathbb{R}$
Output: O bien un número $c \in \mathbb{R}$ tal que $|f(c)| < \epsilon$ o bien un mensaje de error

⋆INICIO
$i \leftarrow 0$
while $|f(x_i)| \geq \epsilon$ **and** $i \leq N$ **do**
  $x_{i+1} \leftarrow x_i - f(x_i)\dfrac{x_i - x_{i-1}}{f(x_i)-f(x_{i-1})}$ [posibles NaN e ∞]
  $i \leftarrow i+1$
end while
if $i > N$ **then**
  return ERROR
end if
return $c \leftarrow x_i$

Es conveniente, al implementar este algoritmo, conservar en memoria no solo x_n y x_{n-1}, sino los valores calculados (que se han utilizado ya) $f(x_n)$ y $f(x_{n-1})$, para no computarlos más de una vez.

6. [Contenido no esencial] Puntos fijos

Los algoritmos de punto fijo (que, como veremos, engloban a los anteriores de una manera indirecta) se basan en la noción de *contractividad*, que no refleja más que la idea de que una función puede hacer que las imágenes de dos puntos cualesquiera estén más cerca que los puntos originales (es decir, la función *contrae* el conjunto inicial). Esta noción, ligada directamente a la derivada (si es que la función es derivable), lleva de manera directa a la de *punto fijo* de una iteración y, mediante un pequeño artificio, a poder resolver ecuaciones generales utilizando iteraciones de una misma función.

6.1. Contractividad: las ecuaciones $g(x) = x$. Sea g una función real de una variable real derivable en un punto c. Esto significa que, dado cualquier infinitésimo o, existe otro o_1 tal que

$$g(c + o) = g(c) + g'(c)o + oo_1,$$

es decir, que *cerca de c, la función g se aproxima muy bien mediante una función lineal*.

Supongamos ahora que o es la anchura de un intervalo "pequeño" centrado en c. Si olvidamos el *error supralineal* (es decir, el término oo_1), pues es "infinitamente más pequeño" que todo lo demás, se puede pensar que el intervalo $(c - o, c + o)$ se transforma en el intervalo $(g(c) - g'(c)o, g(c) + g'(c)o)$: esto es, un intervalo de radio o se transforma aproximadamente, por la aplicación g en uno de anchura $g'(c)o$ (se dilata o se contrae un factor $g'(c)$). Esta es la noción equivalente de derivada que se utiliza en la fórmula de integración por cambio de variable (el Teorema del Jacobiano): la derivada (y en varias variables el *determinante jacobiano*) mide la dilatación o contracción que sufre la recta real (un abierto de $\mathbb{R}^n$) justo en un entorno infinitesimal de un punto.

Por tanto, si $|g'(c)| < 1$, la recta se contrae cerca de c.

Supóngase que eso ocurre en todo un intervalo $[a, b]$. Para no cargar la notación, supongamos que g' es continua en $[a, b]$ y que se tiene $|g'(x)| < 1$ para todo $x \in [a, b]$ —es decir, g siempre *contrae* la recta. Para empezar, hay un cierto $\lambda < 1$ tal que $|g(x)| < \lambda$, por el Teorema de Weierstrass (hemos supuesto g' continua). Por el Teorema del Valor Medio (en la forma de desigualdad, pero importa poco), resulta que, para cualesquiera $x_1, x_2 \in [a, b]$, se tiene que

$$|g(x_1) - g(x_2)| \leq \lambda |x_1 - x_2|,$$

en castellano: *la distancia entre las imágenes de dos puntos es menor que λ por la distancia enre los puntos*, pero como λ es menor que 1, resulta

que *la distancia entre las imágenes es siempre menor que entre los puntos iniciales*. Es decir: la aplicación g está *contrayendo* el intervalo $[a, b]$ *por todas partes*.

Hay muchos énfasis en el párrafo anterior, pero son necesarios porque son la clave para el resultado central: la existencia de un punto fijo.

Se tiene, por tanto, que la anchura del conjunto $g([a, b])$ es menor o igual que $\lambda(b - a)$. Si ocurriera además que $g([a, b]) \subset [a, b]$, es decir, si g transformara el segmento $[a, b]$ en una parte suya, entonces se podría calcular también $g(g([a, b]))$, que sería una parte propia de $g([a, b])$ y que tendría anchura menor o igual que $\lambda\lambda(b - a) = \lambda^2(b - a)$. Ahora podría iterarse la composición con g y, al cabo de n iteraciones se tendría que la imagen estaría contenida en un intervalo de anchura $\lambda^n(b - a)$. Como $\lambda < 1$, resulta que estas anchuras van haciéndose cada vez más pequeñas. Una sencilla aplicación del Principio de los Intervalos Encajados probaría que en el límite, los conjuntos $g([a, b])$, $g(g([a, b]))$, ..., $g^n([a, b])$, ..., terminan cortándose en un solo punto α. Además, este punto, por construcción, debe cumplir que $g(\alpha) = \alpha$, es decir, es un *punto fijo*. Hemos mostrado (no demostrado) el siguiente

Teorema 1. *Sea $g : [a, b] \to [a, b]$ una aplicación de un intervalo cerrado en sí mismo, continua y derivable en $[a, b]$. Si existe $\lambda < 1$ positivo tal que para todo $x \in [a, b]$ se tiene que $|g'(x)| \leq \lambda$, entonces existe un único punto $\alpha \in [a, b]$ tal que $g(\alpha) = \alpha$.*

Así que, si se tiene una función de un intervalo en sí mismo cuya deriva se acota por una constante menor que uno, la ecuación $g(x) = x$ tiene una única solución en dicho intervalo. Además, la explicación previa al enunicado muestra que para calcular α basta con tomar cualquier x del intervalo e ir calculando $g(x)$, $g(g(x))$, ...: el límite es α, independientemente de x.

Por tanto, *resolver ecuaciones de la forma $g(x) = x$ para funciones contractivas es tremendamente simple*: basta con iterar g.

6.2. Aplicación a ecuaciones cualesquiera $f(x) = 0$. Pero por lo general, nadie se encuentra una ecuación del tipo $g(x) = x$; lo que se busca es resolver ecuaciones como $f(x) = 0$.

Esto no presenta ningún problema, pues:

$$f(x) = 0 \Leftrightarrow x - f(x) = x$$

de manera que *buscar un cero de $f(x)$ equivale a buscar un punto fijo de $g(x) = x - f(x)$.*

En realidad, si $\phi(x)$ es una función que no se anula nunca, *buscar un cero de $f(x)$ equivale a buscar un punto fijo de $g(x) = x - \phi(x)f(x)$*. Esto permite, por ejemplo, *escalar* f para que su derivada sea cercana a 1 en la raíz y así que g' sea bastante pequeño, para acelerar el proceso de convergencia. O se puede simplemente tomar $g(x) = x - cf(x)$ para cierto c que haga que la derivada de g sea relativamente pequeña en valor absoluto.

6.3. El algoritmo. La implementación de un algoritmo de búsqueda de punto fijo es muy sencilla. Como todos, requiere una tolerancia ϵ y un número máximo de iteraciones N. La pega es que el algoritmo es inútil si la función g no envía un intervalo $[a, b]$ en sí mismo. Esto es algo que hay que comprobar a priori:

NOTA 1. Sea $g : [a, b] \to \mathbb{R}$ una aplicación. Si se quiere buscar un punto fijo de g en $[a, b]$ utilizando la propiedad de contractividad, es necesario:

- Comprobar que $g([a, b]) \subset [a, b]$.
- Si g es derivable[2] en todo $[a, b]$, comprobar que existe $\lambda \in \mathbb{R}$ tal que $0 < \lambda < 1$ y para el que $|g'(x)| \leq \lambda$ para todo $x \in [a, b]$.

Supuesta esta comprobación, el algoritmo para buscar el punto fijo de $g : [a, b] \to [a, b]$ es el siguiente:

6.4. Utilización para cómputo de raíces. El algoritmo de punto fijo se puede utilizar para computar raíces, como se explicó en la Sección 6.2; para ello, si la ecuación tiene la forma $f(x) = 0$, puede tomarse cualquier función $g(x)$ de la forma

$$g(x) = x - kf(x)$$

donde $k \in \mathbb{R}$ es un número conveniente. Se hace esta operación para conseguir que g tenga derivada cercana a 0 cerca de la raíz, y para que, si χ es la raíz (desconocida), conseguir así que g defina una función contractiva de un intervalo de la forma $[\chi - \rho, \chi + \rho]$ (que será el $[a, b]$ que se utilice).

Lo difícil, como antes, es probar que g es contractiva y envía un intervalo en un intervalo.

[2] Si g no es derivable, se requiere una condición mucho más complicada de verificar: que $|g(x) - g(x')| \leq \lambda|g(x) - g(x')|$ para cierto $0 < \lambda < 1$ y para todo par $x, x' \in [a, b]$. Esto es la *condición de Lipschitz con constante menor que* 1.

Algoritmo 6 (Búsqueda de punto fijo)

Input: una función g (que ya se supone contractiva, etc...), una semilla $x_0 \in [a, b]$, una tolerancia $\epsilon > 0$ y un número máximo de iteraciones $N > 0$
Output: o bien un número $c \in [a, b]$ tal que $|c - g(c)| < \epsilon$ o bien un mensaje de error

⋆INICIO
$i \leftarrow 0, c \leftarrow x_0$
while $|c - g(c)| \geq \epsilon$ **and** $i \leq N$ **do**
  $c \leftarrow g(c)$
  $i \leftarrow i + 1$
end while
if $i > N$ **then**
  return ERROR
end if
return c

6.5. Velocidad de convergencia. Si se tiene la derivada acotada en valor absoluto por una constante $\lambda < 1$, es relativamente sencillo acotar la velocidad de convergencia del algoritmo del punto fijo, pues como se dijo antes, la imagen del intervalo $[a, b]$ es un intervalo de anchura $\lambda(b - a)$. Tras iterar i veces, la imagen de la composición $g^i([a, b]) = g(g(\ldots(g([a, b]))))$ (una composición repetida i veces) está incluida en un intervalo de longitud $\lambda^i(b - a)$, de manera que se puede asegurar que la distancia entre la composición $g^i(x)$ y el punto fijo c es como mucho $\lambda^i(b - a)$. Si λ es suficientemente pequeño, la convergencia puede ser muy rápida.

EJEMPLO 7. Si $[a, b] = [0, 1]$ y $|(|g'(x)) < .1$, se puede asegurar que tras cada iteración, hay un decimal exacto en el punto calculado, sea cual sea la semilla.

7. Velocidad de convergencia de Newton-Raphson

Obsérvese que la expresión para calcular el punto siguiente en el algoritmo de Newton-Raphson es:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

que corresponde a buscar un punto fijo de la función

$$g(x) = x - \frac{1}{f'(x)} f(x).$$

que, como se explica arriba, es una manera de convertir una ecuación $f(x) = 0$ en un problema de punto fijo. La derivada de g es, en este caso:

$$g'(x) = 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2}$$

que, en el punto χ en que $f(\chi) = 0$ vale $g'(\chi) = 0$. Es decir, en el punto fijo, la derivada de g es 0. Esto hace que la convergencia (una vez que uno está "cerca" del punto fijo) sea muy rápida (se dice *de segundo orden*).

De hecho, se puede probar el siguiente resultado:

TEOREMA 2. *Supongamos que f es una función derivable dos veces en un intervalo $[r - \epsilon, r + \epsilon]$, que r es una raíz de f y que se sabe que*

- *La derivada segunda está acotada superiormente: $|f''(x)| < K$ para $x \in [r - \epsilon, r + \epsilon]$,*
- *La derivada primera está acotada inferiormente: $|f'(x)| > L$ para $x \in [r - \epsilon, r + \epsilon]$*

Entonces, si $x_k \in [r - \epsilon, r + \epsilon]$, el siguiente elemento de la iteración de Newton-Raphson está también en el intervalo y

$$|r - x_{k+1}| < |\frac{K}{2L}||r - x_k|^2.$$

Como corolario, se tiene que:

COROLARIO 1. *Si las condiciones del Teorema 2 se cumplen y $\epsilon < 0.1$ y $K < 2L$, entonces a partir de k, cada iteración de Newton-Raphson obtiene una aproximación con el doble de cifras exactas que la aproximación anterior.*

PRUEBA. Esto es porque, si suponemos que $k = 0$, entonces x_0 tiene al menos una cifra exacta. Por el Teorema, x_1 difiere de r en menos que $0.1^2 = .01$. Y a partir de ahora el número de ceros en la expresión se va duplicando. □

Para utilizar el Teorema 2 o su corolario, es preciso:

- Saber que se está cerca de una raíz. La manera común de verificarlo es computando f cerca de la aproximación (hacia la derecha o la izquierda): si el signo cambia, ha de haber una raíz (pues f es continua, al ser derivable).
- Acotar la anchura del intervalo (saber que se está a distancia menor de $1/10$, por ejemplo.
- Acotar $f''(x)$ superiormente y $f'(x)$ inferiormente (esto puede ser sencillo o no).

8. Anexo: Código en Matlab/Octave

Se incluyen a continuación algunos listados con implementaciones "correctas" de los algoritmos descritos en este capítulo, utilizables tanto en Matlab como en Octave. Téngase en cuenta, sin embargo, que, puesto que ambos programas tienen cuidado de que las operaciones en coma flotante no terminen en un error irrecuperable, las implementaciones que se presentan *no incluyen ninguna revisión de posibles errores*. Por ejemplo, en cualquiera de estos algoritmos, si se utiliza la función $\log(x)$ y se evalúa en un número negativo, el algoritmo posiblemente continuará y quizás incluso alcance una raíz (real o compleja). No se han incluido estos tests para no complicar la exposición.

8.1. Implementación del algoritmo de bisección. El siguiente código implementa el algoritmo de bisección en Octave/Matlab. Los parámetros de entrada son:

`f`: una función anónima,
`a`: el extremo izquierdo del intervalo,
`b`: el extremo derecho del intervalo,
`epsilon`: una tolerancia (por defecto, `eps`),
`n`: un número máximo de iteraciones (por defecto, 50).

La salida puede ser nula (si no hay cambio de signo, con un mensaje) o una raíz aproximada en la tolerancia o bien un mensaje (warning) junto con el último valor calculado por el algoritmo (que no será una raíz aproximada en la tolerancia). El formato de la salida, para facilitar el estudio es un par `[z, N]` donde `z` es la raíz aproximada (o el valor más cercano calculado) y `N` es el número de iteraciones hasta calcularla.

8.2. Implementación del algoritmo de Newton-Raphson. El algoritmo de Newton-Raphson es más sencillo de escribir (siempre sin tener en cuenta las posibles *excepciones* de coma flotante), pero requiere un dato más complejo en el input: la derivada de f, que debe ser otra función anónima.

Como se ha de utilizar la derivada de f y no se quiere suponer que el usuario tiene un programa con cálculo simbólico, se requiere que uno de los parámetros sea explícitamente la función $f'(x)$. En un entorno con cálculo simbólico esto no sería necesario.

Así, la entrada es:

`f`: una función anónima,
`fp`: otra función anónima, la derivada de `f`,

```
% Algoritmo de biseccion con error admitido y limite de parada
% Tengase en cuenta que en TODAS las evaluaciones f(...) podria
% ocurrir un error, esto no se tiene en cuenta en esta implementacion
% (en cualquier caso, se terimara)
function [z, N] = Bisec(f, x, y, epsilon = eps, n = 50)
  N = 0;
  if(f(x)*f(y)>0)
    warning('no hay cambio de signo')
    return
  end
  % guardar valores en memoria
  fx = f(x);
  fy = f(y);
  if(fx == 0)
    z = x;
    return
  end
  if(fy == 0)
    z = y;
    return
  end
  z = (x+y)/2;
  fz = f(z);
  while(abs(fz) >= epsilon & N < n)
    N = N + 1;
    % multiplicar SIGNOS, no valores
    if(sign(fz)*sign(fx) < 0)
      y = z;
      fy = fz;
    else
      x = z;
      fx = fz;
    end
    % podria haber un error
    z = (x+y)/2;
    fz = f(z);
  end
  if(N >= n)
    warning('No se ha alcanzado el error admitido antes del limite.')
  end
end
```

FIGURA 1. Código del algoritmo de Bisección

x0: la semilla,
epsilon: una tolerancia (por defecto `eps`),
N: el número máximo de iteraciones (por defecto 50).

El formato de la salida, para facilitar el estudio es un par `[xn, N]` donde `xn` es la raíz aproximada (o el valor más cercano calculado) y `N` es el número de iteraciones hasta calcularla.

```
% Implementacion del metodo de Newton-Raphson
function [z n] = NewtonF(f, fp, x0, epsilon = eps, N = 50)
  n = 0;
  xn = x0;
  % Se supone que f y fp son funciones
  fn = f(xn);
  while(abs(fn) >= epsilon & n <= N)
    n = n + 1;
    fn = f(xn); % evaluar una sola vez
    % siguiente punto
    xn = xn - fn/fp(xn); % podria haber una excepcion
  end
  z = xn;
  if(n == N)
    warning('No converje en MAX iteraciones');
  end
end
```

FIGURA 2. Implementación de Newton-Raphson

CAPÍTULO 3

Solución aproximada de sistemas lineales

Se estudia en este capítulo una colección de algoritmos clásicos para resolver de manera aproximada sistemas de ecuaciones lineales. Se comienza con el algoritmo de Gauss (y su interpretación como factorización LU) y se discute brevemente su complejidad. Se aprovecha esta sección para introducir la noción de *número de condición de una matriz* y su relación con el error relativo (de manera sencilla, sin contemplar posibles errores en la matriz). A continuación se introducen los algoritmos de tipo *punto fijo* (se estudian específicamente dos de ellos, el de Jacobi y el de Gauss-Seidel), y se compara su complejidad con la del algoritmo de Gauss.

Aunque los algoritmos son interesantes de por sí (y su enunciado preciso también), es igual de importante conocer las condiciones de utilidad de cada uno, de manera al menos teórica (por ejemplo, la necesidad de que la matriz de un algoritmo de punto fijo sea *convergente*, noción *análoga* en este contexto a la contractividad). No se discutirán en profundidad aspectos relacionados con el espectro de una aplicación lineal, aunque es la manera más simple de comprender la necesidad de la condición de convergencia.

En todo este tema, se supone que ha de resolverse un sistema de ecuaciones lineales de la forma

$$Ax = b \tag{1}$$

donde A es una matriz cuadrada de orden n y b es un vector de $\mathbb{R}^n$. Se supone, *siempre* que A no es singular (es decir, que el sistema es *compatible determinado*).

1. Algunos ejemplos importantes

Antes de proceder al estudio de cómo se resuelven los sistemas de ecuaciones lineales numéricamente, conviene tener algún ejemplo importante y conocido. E, incluso antes de plantearse "resolver" sistemas lineales, conviene conocer aplicaciones lineales útiles en la vida actual.

A día de hoy, una de las aplicaciones más importantes del Álgebra Lineal es el tratamiento de imágenes (en general, de señales). Lo primero que vamos a estudiar es cómo puede transformarse una imagen en un vector. El código necesario para las instrucciones de Matlab que se indican está en mi página web.

1.1. Imágenes en "escala de grises". Una imagen digital de $n \times m$ pixeles en color se compone (en el sistema RGB) de tres imágenes monocromas, una para el color rojo, otra para el verde (green) y otra para el azul (blue). En el modelo aditivo de color (que es el que utilizan las pantallas), con esos tres colores se pueden obtener (en teoría) todos los del espectro.

Para simplificar el estudio, nosotros trabajaremos siempre con imágenes monocromáticas, que entenderemos en "escala de grises".

La gran mayoría de las transformaciones de imágenes se traducen en realizar combinaciones lineales de los tonos de un punto con los de otros. Incluso girar una imagen puede entenderse así. Pero para trabajar cómodamente con estas transformaciones, es conveniente convertir la imagen original en un vector: transformar la matriz de tamaño $n \times m$ en un vector de nm coordenadas. Esto se realiza "concatenando" las filas de la imagen.

Llamemos I a la matriz que representa la imagen, un rectángulo de $n \times m$ entradas en cada una de las cuales se almacena un valor (que, de momento, supondremos que es un número entre 0 y 1). Así, pues

$$I = (I_{jk}), \;\; j = 0, \ldots, n-1, k = 0, \ldots, m-1.$$

El primer subíndice indica la fila (hay n de ellas) y, el segundo, la columna (hay m). Concatenemos las filas en un vector

$$v_I = (v_i), \;\; i = 1, \ldots, nm$$

de nm componentes. Concatenar las filas de I significa que: las primeros n componentes de v_I son los píxeles de la fila superior de I (se suelen enumerar así). Los siguientes n corresponden a la fila siguiente (la que tiene $j = 1$), etc. hasta terminar. Formalmente:

$$v_r = I_{jk} \Leftrightarrow r - 1 = jm + k.$$

Lo mejor para aclararse es hacer un ejemplo.

Ejemplo 8. Consideremos una imagen de 4 filas por 6 columnas como sigue (se han coloreado los primeros elementos de cada fila

para que después se vea dónde caen en el vector):

1	2	0	4	6	8
2	3	1	6	5	4
6	4	3	2	2	1
9	2	3	1	4	0

Compruébese cómo, la entrada $(1,2)$ tiene el valor 1 (recuérdese que se indexan los elementos desde 0). El vector correspondiente, de 24 componentes, es:

1	2	0	4	6	8	2	3	1	6	5	4	6	4	3	2	2	1	9	2	3	1	4	0

Como puede comprobarse, la componente $9 = 1 \cdot 6 + 3$ (es decir, v_9, contando como que v_1 es la primera) es 1, como corresponde al píxel de índices $j = 1$, $k = 2$, pues $9 - 1 = 1 \cdot 6 + 2$.

Una imagen de mayor tamaño solo da lugar a un vector más grande pero no hay complicación niguna.

La transformación es bidireccional: un vector de nm componentes se puede convertir en una matriz $n \times m$ haciendo. Para ello, si los elementos de la matriz son I_{jk}, entonces $I_{jk} = v_r$ para $r - 1 = jm + k$, donde k varía entre 0 y $m - 1$.

En todo este capítulo utilizaremos la fotografía que se muestra en la Figura 1, que es una imagen de 512 filas y 331 columnas en escala de grises de un ave (obtenida de `wallhere.com/es/wallpaper/853988`). Para cargarla en Matlab, basta ejecutar

FIGURA 1. Fotografía utilizada en este capítulo, de tamaño 512×331.

```
>> load('ave.dat','-mat');
```

que crea una variable `bird` en la que están almacenados los valores de la escala de grises de la imagen. Para visualizarla,

```
>> image(bird);
>> colormap(gray);
```

1.1.1. *Convoluciones.* Una de las técnicas más utilizadas en procesamiento de la imagen es la *convolución*. Este término tiene un significado matemático preciso que no vamos a explicar, pues simplemente lo aplicaremos a este problema gráfico.

En procesamiento de imágenes, se entiende por *convlución* la aplicación de un "núcleo", sobre todos los puntos de la imagen. Un núcleo es una matriz cuadrada "pequeña" N de número impar de filas (habitualmente de tamaño 3×3 ó 5×5, aunque puede ser algo mayor) que se "desplaza" sobre toda la imagen. Esta matriz N actúa como un filtro y transforma la entrada I_{ij} en la suma de los valores de las entradas que rodean al I_{ij} ponderadas por los valores de N_{kl}, como en la figura 2 (en realidad es un vídeo: en algún dispositivo puede verse como tal).

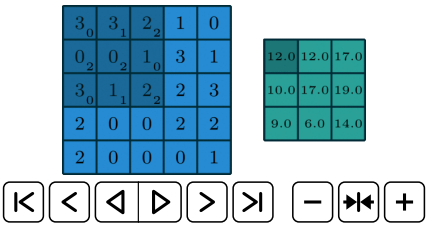

FIGURA 2. Un núcleo sobre una imagen.

En dicha figura, la imagen I corresponde a la matriz "azul", grande, de tamaño 5×5, el núcleo es la matriz que aparece como "subíndices" y se desplaza, de tamaño 3×3; y la imagen obtenida J es más pequeña que la inicial, de tamaño 3×3, en verde. Es decir:

$$I = \begin{pmatrix} 3 & 3 & 2 & 1 & 0 \\ 0 & 0 & 1 & 3 & 1 \\ 3 & 1 & 2 & 2 & 3 \\ 2 & 0 & 0 & 2 & 2 \\ 2 & 0 & 0 & 0 & 1 \end{pmatrix}, K = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}, J = \begin{pmatrix} 12 & 12 & 17 \\ 10 & 17 & 19 \\ 9 & 6 & 14 \end{pmatrix}$$

Por supuesto, un núcleo como el K de la imagen no tiene mucho sentido porque puede convertir números entre 0 y 1 en números mayores que 1. Por lo general, los núcleo que se utilizan en procesamiento de imagen tienen como peculiaridad que la suma de todas sus entradas es como mucho 1: es decir, transforman el valor de un punto de la imagen en una "media" ponderada de dicho punto y los que lo rodean (los 8 de alrededor, en este caso).

```
function [J] = nucleo1(I, K)
    [f, c] = size(I);
    J = zeros(size(I));
    Ks = size(K,1)*size(K,2); % filas, columnas
    Kv = reshape(K, 1, Ks);
    f2 = (size(K,1)-1)/2; % filas
    c2 = (size(K,2)-1)/2; % columnas
    for k = f2+1:f-f2
        for l = c2+1:c-c2
            J(k,l) = Kv * reshape(I(k-1:k+1,l-1:l+1),Ks,1);
        end
    end
    J;
end
```

FIGURA 3. Una función para aplicar un núcleo a una imagen (convolución).

1.1.2. *Desenfoque.* El ejemplo más sencillo de convolución es desenfocar una imagen. Puede entenderse que "desenfocar" consiste en hacer que la intensidad de color de cada punto se redistribuya entre él y los de alrededor. Por ejemplo, un núcleo como

$$K = \begin{pmatrix} 0.025 & 0.05 & 0.025 \\ 0.05 & 0.7 & 0.05 \\ 0.025 & 0.05 & 0.025 \end{pmatrix}$$

haría que cada punto de la imagen perdiera el 30% de su intensidad y la redistribuyera entre los de alrededor, dando el 5% a los de su norte, sur, este y oeste, y el 2.5% a los de las esquinas. Comprobemos cómo afecta este núcleo a la imagen de la Figura 1. Como vamos a iterar el desenfoque, llamamos J a la imagen de partida y de final. Utilizamos el programa `nucleo1.m` de la Figura 1.1.2 para aplicar el núcleo K a la imagen J.

```
>> J = double(bird); % para poder multiplicar
>> K = [0.025 0.05 0.025; 0.05 0.7 0.05; 0.025 0.05 0.025];
>> J = nucleo1(J, K);
>> image(J);
```

Si solo se realiza una vez el desenfoque, casi no se percibe. Si se repite el bucle unas cuantas veces, va observándose cómo se pierde claridad en los contornos (que es, justamente, en lo que consiste desenfocar). Este desenfoque es conocido como "gaussiano", pues se parece a una distribución "campana de Gauss" alrededor de cada punto.

```
function [J] = nucleo(I, K)
    [f, c] = size(I); % filas y columnas de una vez
    [fn, cn] = size(K);
    fn2 = (fn-1)/2;
    cn2 = (cn-1)/2;
    I1 = [zeros(fn2,c+cn-1); zeros(f,cn2) I zeros(f,cn2); zeros(fn2,c+cn
        -1)];
    J = zeros(size(I));
    Ks = size(K,1)*size(K,2);
    Kv = reshape(K, 1, Ks);
    for k = fn2+1:f+fn2;
        for l = cn2+1:c+cn2;
            J(k-fn2,l-fn2) = Kv * reshape(I1(k-fn2:k+fn2,l-cn2:l+cn2),Ks
                ,1);
      end
    end
    J;
end
```

FIGURA 4. Una función para aplicar un núcleo a una imagen (convolución) ampliando la original con 0.

Otro desenfoque distinto consiste en "redistribuir" la intensidad en un punto entre él y sus adyacentes homogéneamente:

$$K = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

1.1.3. *Los bordes de la imagen.* Quien haya experimentado con las operaciones anteriores y repetido varias veces la convolución, observará que la imagen modificada tiene bordes negros cada vez más anchos. Esto se debe a que, en la función `nucleo1`, los bucles no cubren *toda la fila y columna* de J, sino solo la parte en la que el núcleo cabe completamente. En la Figura 2 ya podía observarse este fenómeno y cómo la imagen obtenida, tenía tamaño menor que la original.

Este problema se "resuelve" expandiendo la matriz original para que el núcleo quepa en todos los puntos (por tanto, hay que añadir columnas a la izquierda y a la derecha y filas arriba y abajo, tantas como la mitad del tamaño del núcleo menos uno). Nosotros haremos la expansión más simple: por medio de ceros. Se implementa en el programa `nucleo.m` de la Figura 4. Como se ve en el listado, hace falta ajustar bien el tamaño de la matriz sobre la que se aplica la convolución y, al final, devolver solo la parte del resultado que interesa. Este ajuste es realmente importante en todos los métodos de convolución. Nuestra solución es la más elemental (no necesariamente la peor).

1.1.4. *Expresión matricial general de las convoluciones.* Lo interesante de estas transformaciones es que el valor de cada elemento del destino (cada punto de la imagen final) *es una combinación lineal* de valores de elementos del origen. Por este motivo, la transformación de convolución es *una aplicación lineal*: envía un vector (en este caso, de tamaño nm) en otro del mismo tamaño (o menor si no se hace el ajuste).

Ejemplo 9. En nuestro caso, la imagen es demasiado grande para escribir explícitamente la aplicación con una matriz. Tomemos un caso más pequeño: una imagen de 5×4 píxeles. El espacio vectorial al que pertenece tiene dimensión 20, así que la matriz de transformación será una matriz 20×20. Lo que vamos a ver es que *pese a ser muy grande, es muy simple*.

Si realizamos la convolución con un núcleo 3×3:

$$K = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix}$$

para los elementos del interior de la imagen, I_{ij}, se tiene, como se ve en el listado de la Figura 4:

$$J_{ij} = (k_{11}\ k_{12}\ \cdots\ k_{32}\ k_{33}) \begin{pmatrix} I_{i-1,j-1} \\ I_{i-1,j} \\ \vdots \\ I_{i+1,j} \\ I_{i+1,j+1} \end{pmatrix}$$

donde el vector fila son los elementos de K en filas y el vector columna son los 9 (en este caso) puntos que circundan el I_{ij}. Como se ve, el elemento J_{ij} solo está afectado por sí mismo y los otros 8 elementos de I que "lo rodean": es decir, en la matriz M de la transformación, si v_I es el vector correspondiente a la imagen I, cada fila solo tendrá 9 entradas: distribuidas según corresponde. Recuérdese que, si $v_I = (v_i)$, entonces

$$v_r = I_{ij} \Leftrightarrow r - 1 = im + j$$

de donde la submatriz 3×3:

$$\begin{pmatrix} I_{i-1,j-1} & I_{i-1,j} & I_{i-1,j+1} \\ I_{i,j-1} & I_{ij} & I_{i,j+1} \\ I_{i+1,j-1} & I_{i+1,j} & I_{i+1,j+1} \end{pmatrix}$$

se corresponde con las componentes siguientes del vector v_I:

$$\begin{pmatrix} I_{i-1,j-1} & I_{i-1,j} & I_{i-1,j+1} \\ I_{i,j-1} & I_{ij} & I_{i,j+1} \\ I_{i+1,j-1} & I_{i+1,j} & I_{i+1,j+1} \end{pmatrix} \longrightarrow \begin{pmatrix} v_{(i-1)m+j} & v_{(i-1)m+j+1} & v_{(i-1)m+j+2} \\ v_{im+j} & v_{im+j+1} & v_{im+j+2} \\ v_{(i+1)m+j} & v_{(i+1)m+j+1} & v_{(i+1)m+j+2} \end{pmatrix}$$

Obsérvese que las componentes de v_I que aparecen para I_{ij} se agrupan de 3 en 3 (en este caso, si el núcleo tiene tamaño 3×3): cada grupo de 3 está separado una distancia m del grupo siguiente y el grupo central corresponde a

$$(v_{im+j}, v_{im+j+1}, v_{im+j+2}).$$

Por tanto, la componente k del vector imagen $u_J = Mv_I^T$ tiene la siguiente expresión, si $k - 1 = lm + s$

$$\begin{aligned} u_k =& K_{11}v_{k-m-1} + K_{12}v_{k-m} + K_{13}v_{k-m+1} \\ &+K_{21}v_{k-1} + K_{22}v_k + K_{23}v_{k+1} \\ &+K_{31}v_{k+m-1} + K_{32}v_{k+m} + K_{33}v_{k+m+1} \end{aligned}$$

así que en la fila k de la matriz K todos los elementos son 0 excepto los 9 que corresponden a los índices $k-m-1, k-m, k-m+1$, $k-1, k, k+1$ y $k+m-1, k+m, k+m+1$. Como es lógico, esto solo tiene sentido para $k > m$ y $k < mn - m$. El resto son 0.

Como estamos suponiendo una imagen de 5×4, la dimensión del espacio inicial es 20. La matriz que se obtiene, está en la Figura 5. Se explica a continuación. Téngase en cuenta que el ejemplo es de una imagen *muy pequeña* (5×4), lo cual hace que todos los grupos de entradas no nulas estén muy cerca. Como se ve, se ha multiplicado todo por $1/40$ para no poner decimales. Se observa lo siguiente:

- Todas las filas no nulas suman 1: esto es normal en transformaciones digitales que "distribuyen la información" sin reducirla.
- Las 5 primeras filas son todas de 0: esto se debe a que estamos considerando el caso más fácil, en que el borde de la imagen se descarta (lo que corresponde a la función `nucleo1` de la Figura 2. Estas cinco filas son: los píxeles del borde superior (hay 4) y el píxel del borde izquierdo de la fila 2 (estos píxeles van seguidos en el vector v_I).
- Cada fila no nula contiene tres grupos de tres elementos separados: estos grupos, 1, 2, 1 (que en realidad es 0.025, 0.05, 0.025),

$$\frac{1}{40}\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&2&1&0&2&28&2&0&1&2&1&0&0&0&0&0&0&0&0&0\\
0&1&2&1&0&2&28&2&0&1&2&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&2&1&0&2&28&2&0&1&2&1&0&0&0&0&0\\
0&0&0&0&0&1&2&1&0&2&28&2&0&1&2&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&2&1&0&2&28&2&0&1&2&1&0\\
0&0&0&0&0&0&0&0&0&1&2&1&0&2&28&2&0&1&2&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

FIGURA 5. Matrix 20×20 para el ejemplo 8.

$2, 28, 2$ y $1, 2, 1$ son las filas de la matriz de la convolución K:

$$K = 1/40 \begin{pmatrix} 1 & 2 & 1 \\ 2 & 28 & 2 \\ 1 & 2 & 1 \end{pmatrix}.$$

- Cada dos píxeles con entradas no nulas, hay otros dos con entradas nulas: corresponden a los bordes derecho e izquierdo de cada fila y la siguiente.
- Como en la parte superior, en la inferior también hay 5 filas llenas de 0.

Finalmente, si lo que hiciéramos fuera incluir los bordes exteriores como "ceros", la matriz que se obtendría sería la de la Figura 7 (se deja como un ejercicio para el lector avezado). Esta transformación es más útil que la anterior porque los bordes de la imagen no desaparecen sin más: se supone que la imagen inicial tiene un borde externo de 0 y solo se pierde una pequeña parte de la información (como se ve, ahora las filas correspondientes a puntos del borde *no suman* 1). Es *importante* observar cómo la diagonal principal tiene un valor mayor que la suma del resto de los valores de la fila (y/o columna) correspondiente (a este tipo de matrices se les llama dominantes por

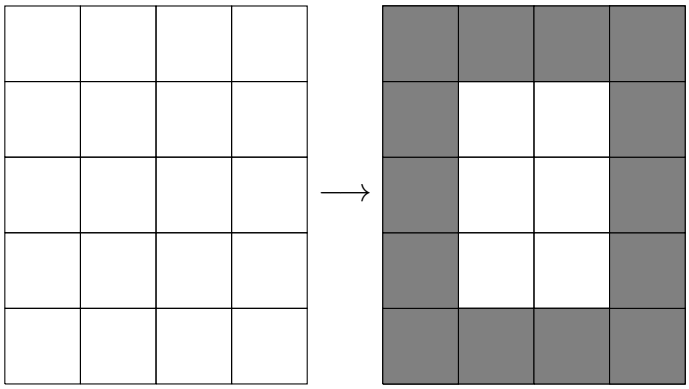

FIGURA 6. La imagen 5×4 convertida en una 3×2 con borde nulo al usar un núcleo de tamaño 3×3 y omitir los bordes.

$$\frac{1}{40}\begin{pmatrix}
28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 1 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 28
\end{pmatrix}$$

FIGURA 7. Matriz de cambio si se incluyen "bordes externos nulos".

filas/columnas). Para ejemplificarlo gráficamente, la Figura 8 muestra cómo se convertiría una imagen 5×4 en escala de grises (64 tonos) utilizando el filtro de este ejemplo. Las imágenes en sus valores

numéricos son:

$$\begin{pmatrix} 35 & 36 & 41 & 36 \\ 39 & 42 & 38 & 33 \\ 45 & 45 & 44 & 34 \\ 50 & 47 & 47 & 30 \\ 53 & 51 & 48 & 33 \end{pmatrix} \longrightarrow \begin{pmatrix} 31.25 & 38.65 & 41.75 & 33.27 \\ 35.47 & 42.20 & 39.15 & 30.92 \\ 40.07 & 44.92 & 43.20 & 31.60 \\ 44.37 & 46.82 & 45.52 & 28.82 \\ 46.15 & 48.92 & 42.02 & 28.42 \end{pmatrix}$$

y el núcleo que se utiliza, como ya se ha dicho, es

$$\frac{1}{40}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 28 & 2 \\ 1 & 2 & 1 \end{pmatrix}.$$

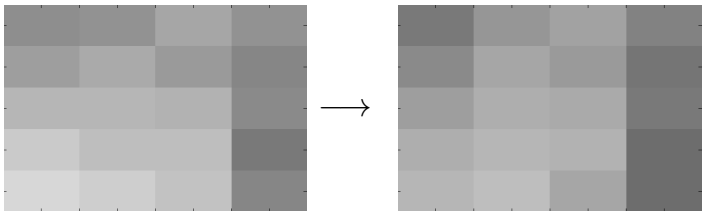

FIGURA 8. Conversión de una imagen 5×4 con el filtro del ejemplo (un desenfoque incluyendo bordes nulos).

1.2. Análisis de tráfico. Un barrio de una ciudad puede presentar, esquemáticamente, la estructura de calles que se muestra en la Figura 9. Los números indican el flujo de coches por hora durante

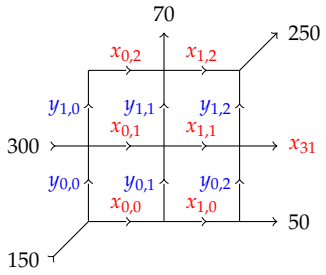

FIGURA 9. Tráfico esquemático de un barrio. Las variables x_{ij} son para el tráfico horizontal y las y_{ij} para el vertical.

la hora punta del mediodía. Se quiere tener una idea del comportamiento del tráfico y de si se puede gestionar de alguna manera para que en algún punto sea más fluido. Las variables x_{ij} indican, como se explica en la figura, el tráfico horizontal por cada calle y las y_{ij} el

vertical. Se supone que el sentido de las calles es el que indican las flechas (quizás poco realista), así que cada calle del modelo es de sentido único (si hubiera calles de doble sentido, bastaría añadir un eje en el grafo con una variable más).

Solo hay una ley que regula el flujo de tráfico: la de Kirchoff para nodos: en cada nodo del grafo, el tráfico entrante ha de ser igual al saliente. Por tanto, en el ejemplo, hay 9 requerimientos, cada uno de los cuales impone una condición lineal (una suma igual a otra). En concreto:

$$\begin{aligned}
150 &= x_{00} + y_{00} \\
300 + y_{00} &= x_{01} + y_{10} \\
y_{10} &= x_{02} \\
x_{00} &= x_{10} + y_{01} \\
x_{01} + y_{01} &= x_{11} + y_{11} \\
x_{02} + y_{11} &= x_{12} + 70 \\
x_{10} &= 50 + y_{02} \\
x_{11} + y_{02} &= x_{31} + y_{12} \\
x_{12} + y_{12} &= 250
\end{aligned}$$

La matriz aumentada asociada a este sistema en el orden de variables indicado, es:

$$\left(\begin{array}{ccccccc|cccccc|c}
x_{00} & x_{01} & x_{02} & x_{10} & x_{11} & x_{12} & x_{31} & y_{00} & y_{01} & y_{02} & y_{10} & y_{11} & y_{12} & \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 150 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 300 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
-1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -70 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -50 \\
0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -250
\end{array}\right)$$

Incluso para un problema tan simple (piénsese lo complicado que puede ser el tráfico de una ciudad entera), la dimensión del sistema es de 9 ecuaciones y 13 variables. Este sistema resulta ser compatible *indeterminado* pero incluso así, ya da algo de información. Por ejemplo (y este es un mero ejemplo de los muchos análisis que pueden hacerse): las variables y_{12}, x_{00}y x_{11} pueden tomar cualquier valor. Por tanto, *en principio*, cerrar esas tres calles podría seguir permitiendo el flujo de coches que exigen las entradas y salidas impuestas. Sin embargo, si se exige que las tres sean 0 (i.e. se cierran las tres calles al tráfico), resulta que la solución del sistema da $x_{00} = 0, y_{01} = -130$, lo cual significa que *habría que cerrar la calle de x_{00} y cambiar de sentido*

la de y_{01} (esto, para un ayuntamiento, es *peccata minuta*, como todos sabemos.

Se observa (como es natural) que la matriz del sistema es muy *dispersa*: la mayor parte de las entradas son 0. Esto es consecuencia de que las ecuaciones modelan un sistema real en que las relaciones son *locales* (i.e. las condiciones en un nodo son independientes de las de los nodos lejanos). Esta propiedad es bastante común en muchas familias de problemas reales.

Se ha indicado ya que este ejemplo es un caso particular de las Leyes de Kirchoff generales (hay una para nodos y otra para ciclos). En el caso del flujo de tráfico, solo aplica la de los nodos. En circuitos eléctricos aplican las dos.

1.3. Estática de un sistema de fuerzas: un puente. Supongamos que se quiere construir un puente con la estructura de la Figura 10. La distribución de fuerzas es la dada por las variables x_i, y_i, z_i y los

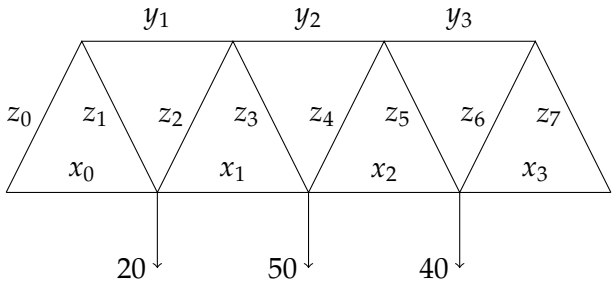

FIGURA 10. Esquema de un puente.

apoyos se sabe que están sujetos a las fuerzas indicadas (los dos extremos inferiores son fijos e inmóviles). Si se quiere que el sistema sea estático, la suma de fuerzas en cada nodo ha de ser 0 (i.e. los nodos *no se mueven*). Los ángulos se suponen de $\pi/3$ (60 grados). Se quiere saber cuál es la distribución de esfuerzos (un esfuerzo negativo es una compresión, uno positivo es una tensión) en los elementos.

Si el sistema es incompatible, el puente no se puede construir con esas especificaciones. Si el sistema de ecuaciones es compatible *indeterminado*, el sistema (real) se denomina "hiperestático". Si el sistema de ecuaciones es compatible determinado, el sistema (real) se denomina "isostático".

Ha de tenerse en cuenta que, en cada nodo, hay que estudiar las componentes horizontal y vertical de las fuerzas. Si los nodos superiores son I_1, I_2, I_3, I_4 y los inferiores (sin contar los extremos) J_1, J_2, J_3, entonces:

(1) La estabilidad del nodo I_1 da

$$y_1 - \frac{1}{2}z_0 + \frac{1}{2}z_1 = 0 \ \text{(horizontal)}$$
$$\frac{\sqrt{3}}{2}z_0 + \frac{\sqrt{3}}{2}z_1 = 0 \ \text{(vertical)}$$

Como hay 7 nodos "efectivos" (los extremos, al estar sujetos al terreno, no añaden ni quitan esfuerzos), en total se obtienen 14 ecuaciones (que no vamos a escribir). Igual que en las de I_1, hay una condición vertical y una horizontal en cada una, y cada ecuación (en la estructura de la Figura 10) contiene a lo sumo 3 variables, por lo que el sistema de ecuaciones que se obtiene es también muy disperso (cada fila contiene 14 entradas, de la cuales por lo menos 11 son nulas).

Claramente, cuantos más elementos posee una estructura, mayor es el número de variables y, usualmente, de ecuaciones, con lo que es fácil obtener sistemas de decenas de ecuaciones e incógnitas.

Si, además, hubiera más fuerzas actuando fuera de los nodos, habría que utilizar también el principio de "anulación de momentos" (que no aplica en este ejemplo).

2. El algoritmo de Gauss y la factorizacón LU

Partiendo de la Ecuación (1), el algoritmo de Gauss consiste en transformar A y b mediante operaciones "sencillas" en una matriz $\tilde{A}$ triangular superior y un vector $\tilde{b}$, para que el nuevo sistema $\tilde{A}x = \tilde{b}$ sea directamente soluble mediante *sustitución regresiva* (es decir, se puede calcular la variable x_n despejando directamente y, sustituyendo "hacia arriba" en la ecuación $n-1$, se puede calcular x_{n-1}, etc.) Es obvio que se requiere que la solución del sistema nuevo sea la misma que la del original. Para ello, se permite realizar la siguiente operación:

- Se puede sustituir una ecuación E_i (la fila $i-$ésima de A) por una combinación lineal de la forma $E_i + \lambda E_k$ donde $k < i$ y $\lambda \in \mathbb{R}$. Si se hace esto, también se sustituye b_i por $b_i + \lambda b_k$.

El hecho de que la combinación tenga coeficiente 1 en E_i es lo que obliga a que las soluciones del sistema modificado sean las mismas que las del original.

Lema 1. *Para transformar una matriz A en una matriz $\tilde{A}$ según la operación anterior, basta multiplicar A por la izquierda por la matriz $L_{ik}(\lambda)$ cuyos elementos son:*

- *Si $m = n$, entonces $(L_{ik}(\lambda))_{mn} = 1$ (diagonal de 1).*

- *Si* $m = i, n = k$, *entonces* $(L_{ik}(\lambda))_{mn} = \lambda$ (*el elemento* (i,k) *es* λ).
- *Cualquier otro elemento es* 0.

Ejemplo 10. Si se parte de la matriz A

$$A = \begin{pmatrix} 3 & 2 & -1 & 4 \\ 0 & 1 & 4 & 2 \\ 6 & -1 & 2 & 5 \\ 1 & 4 & 3 & -2 \end{pmatrix}$$

y se combina la fila 3 con la fila 1 multiplicada por -2 (para "hacer un cero" en el 6), entonces se ha de multiplicar A por $L_{31}(-2)$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 2 & -1 & 4 \\ 0 & 1 & 4 & 2 \\ 6 & -1 & 2 & 5 \\ 1 & 4 & 3 & -2 \end{pmatrix} = \begin{pmatrix} 3 & 2 & -1 & 4 \\ 0 & 1 & 4 & 2 \\ 0 & -5 & 4 & -3 \\ 1 & 4 & 3 & -2 \end{pmatrix}.$$

De manera simplificada, el algoritmo de Gauss puede enunciarse como indica el Algoritmo 7.

La línea marcada con el asterisco en el Algoritmo 7 es exactamente la multiplicación de $\tilde{A}$ por la izquierda por la matriz $L_{ji}(m_{ji})$, donde el multiplicador m_{ji} es justamente

$$m_{ji} = -A_{ji}/A_{ii}.$$

Así que al final, la matriz $\tilde{A}$, triangular superior, es un producto de esas matrices:

$$\tilde{A} = L_{n,n-1}(m_{n,n-1})L_{n,n-2}(m_{n,n-2}) \cdots L_{2,1}(m_{2,1})A = \tilde{L}A \tag{2}$$

donde $\tilde{L}$ es una matriz triangular inferior con 1 en la diagonal (esto es un sencillo ejercicio). Resulta también sencillo comprobar que (y esto realmente parece *magia*) que

Lema. *La matriz inversa del producto de todas las matrices de* (2) *es la matriz triangular inferior que en cada componente* (j,i) *contiene el valor* $-m_{ji}$.

Así que, sin más complicación, se ha demostrado el siguiente resultado:

Teorema 3. *Si en el proceso de reducción de Gauss no hay ningún elemento de la diagonal igual a* 0, *entonces existe una matriz* L *triangular inferior cuyos elementos son los sucesivos multiplicadores* cambiados de signo *en su posición correspondiente y una matriz diagonal superior* U *tal que*

$$A = LU$$

Algoritmo 7 (Algoritmo de Gauss para sistemas lineales)

Input: Una matriz A y un vector b, ambos de orden n
Output: O bien un mensaje de error o bien una matriz $\tilde{A}$ y un vector $\tilde{b}$ tales que $\tilde{A}$ es triangular superior y que el sistema $\tilde{A}x = \tilde{b}$ tiene las mismas soluciones que el $Ax = b$

⋆INICIO
$\tilde{A} \leftarrow A, \tilde{b} \leftarrow b, i \leftarrow 1$
while $i < n$ **do**
  if $\tilde{A}_{ii} = 0$ **then**
    return ERROR [`división por cero`]
  end if
  [`combinar cada fila bajo la` i `con la` i]
  $j \leftarrow i + 1$
  while $j \leq n$ **do**
    $m_{ji} \leftarrow -\tilde{A}_{ji}/\tilde{A}_{ii}$
    [`La siguiente linea es un bucle, cuidado`]
    $\tilde{A}_j \leftarrow \tilde{A}_j + m_{ji}\tilde{A}_i$ [⋆]
    $\tilde{b}_j \leftarrow \tilde{b}_j + m_{ji}\tilde{b}_i$
    $j \leftarrow j + 1$
  end while
  $i \leftarrow i + 1$
end while
return $\tilde{A}, \tilde{b}$

y tal que el sistema $Ux = \tilde{b} = L^{-1}b$ *es equivalente al sistema incial* $Ax = b$.

Con este resultado, se obtiene una factorización de A que simplifica la resolución del sistema original, pues se puede reescribir $Ax = b$ como $LUx = b$; yendo por partes, se hace:

- Primero se resuelve el sistema $Ly = b$, por *sustitución directa* —es decir, de arriba abajo, sin siquiera dividir.
- Luego se resuelve el sistema $Ux = y$, por *sustitución regresiva* —es decir, de abajo arriba.

Esta manera de resolver solo requiere conservar en memoria las matrices L y U y es muy rápida por comparación con el método de Cramer, por ejemplo, o como se explica a continuación, con el uso de la inversa.

2.1. [Contenido no esencial] Complejidad del algoritmo de Gauss. Clásicamente, la complejidad de los algoritmos que utilizan operaciones aritméticas se medía calculando el número de multiplicaciones necesarias (pues la multiplicación era una operación mucho más compleja que la adición). Hoy día esta medida es menos representativa, pues las multiplicaciones en coma flotante se realizan en un tiempo prácticamente equivalente al de las adiciones (con ciertos matices, pero este es uno de los avances importantes en velocidad de procesamiento).

De esta manera, si *se supone que las divisiones pueden hacerse con exactitud*[1], se tiene el siguiente resultado:

LEMA 2. *Si en el proceso de simplificación de Gauss no aparece ningún cero en la diagonal, tras una cantidad de a lo sumo* $(n-1)+(n-2)+\cdots+1$ *combinaciones de filas, se obtiene un sistema* $\tilde{A}x=\tilde{b}$ *donde* $\tilde{A}$ *es triangular superior.*

Cada combinación de filas en el paso k, tal como se hace el algoritmo requiere una multiplicación por cada elemento de la fila (descontando el que está justo debajo del pivote, que se sabe que hay que sustituir por 0), que en este paso da $n-k$ productos, más una división (para calcular el multiplicador de la fila) y otra multiplicación para hacer la combinación del vector. Es decir, en el paso k hacen falta

$$(n-k)^2+(n-k)+(n-k)$$

operaciones "complejas" (multiplicaciones, esencialmente) y hay que sumar desde $k=1$ hasta $k=n-1$, es decir, hay que sumar

$$\sum_{i=1}^{n-1} i^2+2\sum_{i=1}^{n-1} i.$$

La suma de los primeros r cuadrados es (clásica) $r(r+1)(2r+1)/6$, mientras que la suma de los primeros r números es $r(r+1)/2$. Así pues, en total, se tienen

$$\frac{(n-1)n(2(n-1)+1)}{6}+(n-1)n=\frac{n^3}{3}+\frac{n^2}{2}-\frac{5n}{6}.$$

Para ahora resolver el sistema triangular superior $\tilde{A}x\tilde{b}$, hace falta una división por cada línea y $k-1$ multiplicaciones en la fila $n-k$ (con k desde 0 hasta n), así que hay que sumar $n+1+\cdots+(n-1)=$

[1]Lo cual es, como ya se ha dicho muchas veces, demasiado suponer.

$\frac{n(n+1)}{2}$. Por tanto, para resolver un sistema con el método de Gauss, hacen falta en total

$$\frac{n^3}{3} + n^2 - \frac{n}{3} \text{ operaciones para } Ax = b.$$

Si el algoritmo se utiliza para resolver m sistemas de la forma $Ax = b_i$ (para diferentes b_i), todas las operaciones son iguales para triangular A y simplemente hay que recomputar $\tilde{b}_i$ y resolver, Esto requiere $(n-1) + \cdots + 1 = n(n-1)/2$ multiplicaciones (las del proceso de triangulación) y $1 + 2 + \cdots + n$ para "despejar". Descontando las que se hicieron en la resolución de b, resulta que para resolver los m sistemas, hacen falta:

$$\frac{n^3}{3} + mn^2 - \frac{n}{3} \text{ operaciones para } m \text{ sistemas.} \tag{3}$$

2.1.1. *Comparación con utilizar A^{-1}.* Es sencillo comprobar que el cálculo general de la inversa de una matriz requiere, utilizando el algoritmo de Gauss-Jordan (o por ejemplo, resolviendo los sitemas $Ax = e_i$ para la base estándar) al menos n^3 operaciones. Hay mejores algoritmos (pero se están intentando comparar métodos análogos). Una vez calculada la inversa, resolver un sistema $Ax = b$ consiste en multiplicar b a la izquierda por A^{-1}, que requiere (obviamente) n^2 productos. Por tanto, la resolución de m sistemas requiere

$$n^3 + mn^2 \text{ operaciones complejas.}$$

Que siempre es más grande que (3). Así que, si se utiliza el método de Gauss, es mejor conservar la factorización LU y utilizarla para "la sustitución" que calcular la inversa y utilizarla para multiplicar por ella.

Claro está que *esta comparación es entre métodos análogos*: hay maneras de computar la inversa de una matriz en menos (bastante menos) de n^3 operaciones (aunque siempre más que un orden de $n^2 \log(n)$).

2.2. Estrategias de Pivotaje, el algoritmo LUP. Como ya se dijo, si en el proceso de reducción gaussiana aparece un pivote (el elemento que determina el multiplicador) con valor 0 (o incluso con denominador pequeño), o bien puede no continuarse de manera normal o bien puede que aparezcan errores muy grandes debido al redondeo. Esto puede aliviarse utilizando estrategias de pivotaje: cambiando el orden de las filas o de las columnas. Si solo se cambia el orden de las filas, se dice que se realiza un *pivotaje parcial*. Si se hacen ambas operaciones, se dice que se realiza un *pivotaje total*. En estas notas solo se estudiará el pivotaje parcial.

Definición 9. Una *matriz de permutación* es una matriz cuadrada formada por ceros salvo que en cada fila y en cada columna hay exactamente un 1.

(Para que una matriz sea de permutación basta con que se construya a partir de la matriz identidad, *permutando* filas).

Es obvio que el determinante de una matriz de permutación es distinto de 0 (de hecho, es bien 1 bien -1). No es tan sencillo comprobar que la inversa de una matriz de permutación P es también una matriz de permutación y, de hecho, es P^T (su traspuesta).

Lema 3. *Si A es una matriz $n \times m$ y P es una matriz cuadrada de orden n, de permutación, que solo tiene* 2 *elementos fuera de la diagonal no nulos, digamos los (i, j) y (j, i) (pues tiene que ser simétrica), entonces PA es la matriz obtenida a partir de A intercambiando las* filas *i y j. Para intercambiar por columnas se ha de hacer a la derecha (pero no se explica aquí).*

Definición 10. Se dice que en el proceso de reducción de Gauss se sigue una estrategia de *pivotaje parcial* si el pivote en el paso i es siempre el elemento de la columna i de mayor valor absoluto.

Para realizar el algoritmo de Gauss con pivotaje parcial, basta realizarlo paso a paso salvo que, antes de fijar el pivote, se ha de buscar, bajo la fila i el elemento de mayor valor absoluto de la columna i. Si este está en la fila j (y será siempre $j \geq i$, al buscar por debajo de i), entonces se han de intercambiar las filas i y j de la matriz.

La estrategia de pivotaje da lugar a una factorización que no es *exactamente* la LU, sino con una matriz de permutación añadida:

Lema 4. *Dado el sistema de ecuaciones $Ax = b$ con A no singular, siempre existe una matriz P de permutación y dos matrices L, U, la primera triangular inferior, la segunda triangular superior, con*

$$PA = LU.$$

La prueba de este resultado no es directa, hay que hacer un razonamiento por recuerrencia (sencillo pero no inmediato).

Tanto Octave como Matlab incluyen la función `lu` que, dada una matriz A, devuelve tres valores: L, U y P.

Por ejemplo, si

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ -1 & -2 & 5 & 6 \\ -1 & -2 & -3 & 7 \\ 0 & 12 & 7 & 8 \end{pmatrix}$$

entonces

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \quad U = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 12 & 7 & 8 \\ 0 & 0 & 8 & 10 \\ 0 & 0 & 0 & 11 \end{pmatrix}, \quad P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Para calcular L, U y P no hay más que realizar el algoritmo ordinario de Gauss salvo que *cuando se realice un intercambio entre la fila i y la fila j* se ha de realizar el mismo en la L hasta entonces calculada (cuidado: solo en las primeras $i-1$ columnas, la zona de la diagonal con "1" no se ha de tocar) y multiplicar la P ya calculada (comenzando con $P = \mathrm{Id_n}$) por la izquierda por P_{ij} (la matriz de permutación de las filas i y j).

En fin, puede expresarse el algoritmo LUP como en el Algoritmo 8.

2.3. El número de condición: comportamiento del error relativo. ¿Cómo es de "estable" la resolución de un sistema de ecuaciones lineales? Una manera *muy simple* de acercarse a este problema es comparar los "errores relativos" en una solución si se cambia el vector b por uno *modificado un poco*. Supóngase que en lugar del sistema (1) original, cuya solución es x, se tiene uno modificado

$$Ay = b + \delta b$$

donde δb es un *vector pequeño*. La solución será de la forma $x + \delta x$, para cierto δx (que uno *espera* que sea pequeño).

La manera de medir tamaños de vectores es mediante *una norma* (la más común es la *longitud*, en el espacio eculídeo, pero no es la que se utilizará aquí). Como x es una solución, se tiene que

$$A(x + \delta x) = b + \delta b,$$

así que

$$A\delta x = \delta b,$$

pero como se parte de una modificación de b y estudiar cómo se modifica x, despejando y queda

$$\delta x = A^{-1}\delta b$$

y midiendo *tamaños* (es decir, *normas*, que se denotan $\|\|$), quedaría

$$\|\delta x\| = \|A^{-1}\delta b\|.$$

Se está estudiando el *desplazamiento relativo*, que es más relevante que el absoluto. Ahora se ha incluir $\|x\|$ en el primer miembro. La información que se tiene es que $Ax = b$, de donde $\|Ax\| = \|b\|$. Así que

Algoritmo 8 (Factorización LUP para una matriz A)

Input: Una matriz A de orden n
Output: O bien un mensaje de error o bien tres matrices: L triangular inferior con 1 en la diagonal, U triangular superior y P matriz de permutación tales que $LU = PA$

⋆INICIO
$L \leftarrow \mathrm{Id_n}, \mathrm{U} \leftarrow \mathrm{A}, \mathrm{P} \leftarrow \mathrm{Id_n}, i \leftarrow 1$
while $i < n$ **do**
  $p \leftarrow$ fila tal que $|U_{pi}|$ es máximo, con $p \geq i$
  if $U_{pi} = 0$ **then**
    return ERROR [`división por cero`]
  end if
  [`intercambiar filas` i `y` p]
  $P \leftarrow P_{ip}P$
  $U \leftarrow P_{ip}U$
  [`en` L `solo se intercambian las filas` i `y` p `de la submatriz` $n \times (i-1)$ `de la izquierda, ver texto`]
  $L \leftarrow \tilde{L}$
  [`combinar filas en` U `y llevar cuenta en` L]
  $j \leftarrow i + 1$
  while $j <= n$ **do**
    $m_{ji} \leftarrow U_{ji}/U_{ii}$
    $U_j \leftarrow U_j - m_{ij}U_i$
    $L_{ji} \leftarrow m_{ji}$
    $j \leftarrow j + 1$
  end while
  $i \leftarrow i + 1$
end while
return L, U, P

queda

$$\frac{\|\delta x\|}{\|Ax\|} = \frac{\|A^{-1}\delta b\|}{\|b\|},$$

pero esto no dice mucho (pues es una identidad obvia, se querría acotar el *desplazamiento* relativo de la solución si se desplaza un poco el término independiente).

Supóngase que existe determinado objeto llamado *norma de una matriz*, que se denotaría $\|A\|$ que cumple que $\|Ax\| \leq \|A\|\|x\|$. Entonces el miembro de la izquierda la ecuación anterior, leída de derecha a izquierda quedaría

$$\frac{\|A^{-1}\delta b\|}{\|b\|} = \frac{\|\delta x\|}{\|Ax\|} \geq \frac{\|\delta x\|}{\|A\|\|x\|}$$

y, por otro lado, aplicando el mismo razonamiento a la parte "de las b", se tendría que

$$\frac{\|A^{-1}\|\|\delta b\|}{\|b\|} \geq \frac{\|A^{-1}\delta b\|}{\|b\|},$$

y combinando todo,

$$\frac{\|A^{-1}\|\|\delta b\|}{\|b\|} \geq \frac{\|\delta x\|}{\|A\|\|x\|}$$

de donde, finalmente, se obtiene una cota superior para el *desplazamiento relativo de x*:

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|\delta b\|}{\|b\|}$$

El caso es que tal objeto, llamado *norma de una matriz* (*o más bien de una aplicación lineal*) existe. De hecho, existen muchos. En estas notas se va a utilizar el siguiente:

Definición 11. Se denomina *norma infinito* de una matriz cuadrada $A = (a_{ij})$ al número

$$\|A\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}|,$$

es decir, el máximo de las sumas de los valores absolutos *por filas*.

En realidad, se podrían utilizar muchas otras definiciones, pero esta es la que se usará en estas notas.

Lema 5. *La* norma infinito *cumple que, para cualquier vector x, $\|Ax\|_\infty \leq \|A\|_\infty\|x\|_\infty$, donde $\|x\|_\infty$ es la norma dada por el máximo de los valores absolutos de las coordenadas de x.*

Es decir, si se toma como *medida del tamaño de un vector* el máximo de sus coordenadas en valor absoluto, y lo denominamos $\|x\|_\infty$ se tiene que

$$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \leq \|A\|_\infty\|A^{-1}\|_\infty\frac{\|\delta b\|_\infty}{\|b\|_\infty}.$$

Al producto $\|A\|_\infty \|A^{-1}\|_\infty$ se le denomina *número de condición de la matriz A para la norma infinito*, se denota $\kappa(A)$ y es una medida del máximo desplazamiento posible de la solución si se desplaza un poco el vector. Así que, cuanto más grande sea el número de condición peor se comportará en principio el sistema respecto de pequeños cambios en el término independiente.

El número de condición también sirve para acotar *inferiormente* el error relativo cometido:

LEMA 6. *Sea A una matriz no singular de orden n y x una solución del sistema $Ax = b$. Sea δb un "desplazamiento" de las condiciones iniciales y δx el "desplazamiento" correspondiente en la solución. Entonces:*

$$\frac{1}{\kappa(A)} \frac{\|\delta b\|}{\|b\|} \leq \frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\delta b\|}{\|b\|}.$$

Así que se puede acotar el error relativo con el residuo relativo (*el número* $\|\delta b\| / \|b\|$).

EJEMPLO 11. Considérese el sistema

$$\begin{aligned} 0.853x + 0.667y &= 0.169 \\ 0.333x + 0.266y &= 0.067 \end{aligned}$$

Su número de condición para la norma infinito es 376.59, así que un cambio de una milésima relativo en las condiciones iniciales (el vector b) puede dar lugar a un cambio relativo de más del 37% en la solución. La de dicho sistema es $x = 0.055+, y = 0.182+$. Pero el número de condición tan grande avisa de que una pequeña perturbación originará grandes modificaciones en la solución. Si se pone, en lugar de $b = (0.169, 0.067)$ el vector $b = (0.167, 0.067)$ (un desplazamiento relativo del 1.1% en la primera coordenada), la solución es $x = -0.0557+, y = 0.321+$. Por un lado, la x no tiene siquiera el mismo signo, por otro, el desplazamiento relativo en la y es del 76%, totalmente inaceptable. Imagínese que el problema describe un sistema estático de fuerzas y que las medidas se han realizado con instrumentos de precisión de $\pm 10^{-3}$. Para $b = (0.168, 0.067)$, la diferencia es menor (aunque aun inaceptable en y), pero el cambio de signo persiste...

2.4. Un ejemplo explícito de Gauss con pivotaje. Se muestra a continuación el desarrollo completo del método de Gauss con pivotaje parcial aplicado a un sistema (matriz de coeficientes) 4×4. Sea

$$A = \begin{pmatrix} 1 & 6 & 8 & 3 \\ 2 & 4 & 6 & 0 \\ -1 & 3 & 6 & 1 \\ 0 & 4 & 2 & 8 \end{pmatrix}, b = \begin{pmatrix} 2 \\ 1 \\ 2 \\ 3 \end{pmatrix}$$

Se quieren calcular las matrices L (triangular inferior), U (triangular superior) y P (matriz de permutación) que satisfagan:

$$PA = LU$$

pero vamos a llevar cuenta de b, también, así que en todo el desarrollo trabajaremos con la matriz ampliada. Comenzamos, por tanto, haciendo

$$U = \begin{pmatrix} 1 & 6 & 8 & 3 & 2 \\ 2 & 4 & 6 & 0 & 1 \\ -1 & 3 & 6 & 1 & 2 \\ 0 & 4 & 2 & 8 & 3 \end{pmatrix},$$

que es la matriz que vamos a ir transformando hasta que la parte de los coeficientes sea triangular superior. Procedemos a realizar el algoritmo, paso a paso.

(1) Se han de intercambiar las filas 1 y 2 para que el pivote tenga el valor 2. Para esto, se usa la matriz P_{12}, y se obtiene

$$U_1 \leftarrow \begin{pmatrix} 2 & 4 & 6 & 0 & 1 \\ 1 & 6 & 8 & 3 & 2 \\ -1 & 3 & 6 & 1 & 2 \\ 0 & 4 & 2 & 8 & 3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 6 & 8 & 3 & 2 \\ 2 & 4 & 6 & 0 & 1 \\ -1 & 3 & 6 & 1 & 2 \\ 0 & 4 & 2 & 8 & 3 \end{pmatrix},$$

(2) Utilizando las matrices $L_{12}(-1/2)$ y $L_{13}(1/2)$ se eliminan los términos 1 y -1, respectivamente, de debajo del pivote de la primera columna. Recuérdese que

$$L_{12}(-1/2) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, L_{13}(1/2) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

y, haciendo todos los cálculos, queda

$$U_2 \leftarrow L_{13}(1/2) \cdot L_{12}(-1/2)U_1 = \begin{pmatrix} 2 & 4 & 6 & 0 & 1 \\ 0 & 4 & 5 & 3 & \frac{3}{2} \\ 0 & 5 & 9 & 1 & \frac{5}{2} \\ 0 & 4 & 2 & 8 & 3 \end{pmatrix}$$

(3) Ahora hay que intercambiar las filas 2 y 3 para que el 5 quede como pivote. Para esto se utiliza P_{23}:

$$U_3 \leftarrow P_{23}U_2 = \begin{pmatrix} 2 & 4 & 6 & 0 & 1 \\ 0 & 5 & 9 & 1 & \frac{5}{2} \\ 0 & 4 & 5 & 3 & \frac{3}{2} \\ 0 & 4 & 2 & 8 & 3 \end{pmatrix}.$$

(4) Se eliminan los dos 4 que hay debajo del pivote, utilizando $L_{23}(-4/5)$ y $L_{24}(-4/5)$:

$$U_4 \leftarrow \begin{pmatrix} 2 & 4 & 6 & 0 & 1 \\ 0 & 5 & 9 & 1 & \frac{5}{2} \\ 0 & 0 & -\frac{11}{5} & \frac{11}{5} & -\frac{1}{2} \\ 0 & 0 & -\frac{26}{5} & \frac{36}{5} & 1 \end{pmatrix} = L_{24}(-4/5)L_{23}(-4/5)U_3.$$

(5) Se intercambian las filas 3 y 4 para que el $-26/5$ quede como pivote, utilizando P_{34}:

$$U_5 \leftarrow \begin{pmatrix} 2 & 4 & 6 & 0 & 1 \\ 0 & 5 & 9 & 1 & \frac{5}{2} \\ 0 & 0 & -\frac{26}{5} & \frac{36}{5} & 1 \\ 0 & 0 & -\frac{11}{5} & \frac{11}{5} & -\frac{1}{2} \end{pmatrix}.$$

(6) Y, finalmente, se elimina el término $-11/5$ utilizando $L_{34}(-11/26)$:

$$U_6 \leftarrow \begin{pmatrix} 2 & 4 & 6 & 0 & 1 \\ 0 & 5 & 9 & 1 & \frac{5}{2} \\ 0 & 0 & -\frac{26}{5} & \frac{36}{5} & 1 \\ 0 & 0 & 0 & -\frac{11}{13} & -\frac{12}{13} \end{pmatrix}.$$

Así pues, se puede transformar el sistema original en

$$U_6 = \begin{pmatrix} 2 & 4 & 6 & 0 & 1 \\ 0 & 5 & 9 & 1 & \frac{5}{2} \\ 0 & 0 & -\frac{26}{5} & \frac{36}{5} & 1 \\ 0 & 0 & 0 & -\frac{11}{13} & -\frac{12}{13} \end{pmatrix},$$

que es triangular superior, con la siguiente secuencia de matrices:

$$U_6 = L_{34}(-11/26)P_{34}L_{24}(-4/5)L_{23}(-4/5)P_{23}L_{13}(1/2)L_{12}(-1/2)P_{12}A.$$

Para construir L, hay que utilizar los multiplicadores cambiados de signo y hacer los intercambios parciales de filas según indiquen los P_{ij}, sin contar el primero:

(1) Se construye la matriz de multiplicadores con el signo cambiado para la primera columna:

$$L_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(2) Se intercambia *la parte anterior a la columna* 2 según indique el segundo cambio de filas, P_{23} (es decir, filas 2 y 3):

$$L_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

como se ve, *la diagonal de* 1 *sigue estando ahí*.

(3) Se introducen los multiplicadores cambiados de signo en la segunda columna:

$$L_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{2} & \frac{4}{5} & 1 & 0 \\ 0 & \frac{4}{5} & 0 & 1 \end{pmatrix}.$$

(4) Se intercambia la parte izquierda de las filas correspondientes a la siguiente matriz P, que es P_{34}, es decir, las filas 3 y 4:

$$L_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ 0 & \frac{4}{5} & 1 & 0 \\ \frac{1}{2} & \frac{4}{5} & 0 & 1 \end{pmatrix}.$$

(5) Se termina introduciendo el último multiplicador, cambiado de signo:

$$L_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ 0 & \frac{4}{5} & 1 & 0 \\ \frac{1}{2} & \frac{4}{5} & \frac{11}{26} & 1 \end{pmatrix}.$$

Con esto, se obtiene que

$$P_{34}P_{23}P_{12}A = L_4U_6,$$

donde es fácil ver que

$$P = P_{34}P_{23}P_{12} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

3. Algoritmos aproximados (de punto fijo)

Tanto la complejidad computacional del algoritmo de Gauss como su inestabilidad respecto a la división por pequeños números (debida a la introducción inevitable de errores de redondeo) llevan a plantearse buscar otro tipo de algoritmos con mejores propiedades. En algunas situaciones sería incluso utópico plantearse resolver un sistema en un cantidad similar a n^3 multiplicaciones (pongamos por caso que $n \simeq 10^6$, entonces $n^3 \simeq 10^{18}$ y haría falta demasiado tiempo para realizar todas las operaciones).

Si se parte de un sistema como el original (1), de la forma $Ax = b$, puede tratar de convertirse en un problema de punto fijo mediante una *separación* de la matriz A en dos, $A = N - P$, donde N es invertible. De este modo, el sistema $Ax = b$ queda $(N - P)x = b$, es decir

$$(N - P)x = b \Rightarrow Nx = b + Px \Rightarrow x = N^{-1}b + N^{-1}Px,$$

si se llama $c = N^{-1}b$ y $M = N^{-1}P$, queda el siguiente problema de punto fijo

$$x = Mx + c$$

que, si puede resolverse, puede hacerse mediante iteraciones de la misma forma que en el Capítulo 2: se comienza con una semilla x_0 y se itera

$$x_n = Mx_{n-1} + c,$$

hasta alcanzar una precisión suficiente. Hacen falta los siguientes resultados.

TEOREMA 4. *Supongamos que M es una matriz de orden n y que $\|M\|_\infty < 1$. Entonces la ecuación $x = Mx + c$ tiene solución única para todo c y la iteración $x_n = Mx_{n-1} + c$ converge a ella para cualquier vector inicial x_0.*

TEOREMA 5. *Dada la matriz M con $\|M\|_\infty < 1$, y dado x_0 una semilla para el método iterativo del Teorema 4, si s es la solución del problema $x = Mx + c$, se tiene la siguiente cota:*

$$\|x_n - s\|_\infty \leq \frac{\|M\|_\infty^n}{1 - \|M\|_\infty} \|x_1 - x_0\|_\infty.$$

Recuérdese que, para vectores, la norma infinito $\|x\|_\infty$ viene dada por el mayor valor absoluto de las componentes de x.

Con estos resultados, se explican los dos métodos básicos iterativos para resolver sistemas de ecuaciones lineales: el de Jacobi, que corresponde a tomar N como la diagonal de A y el de Gauss-Seidel, que corresponde a tomar N como la parte triangular inferior de A incluyendo la diagonal.

3.1. El algortimo de Jacobi. Si en el sistema $Ax = b$ se "despeja" cada coordeanada x_i, *en función de las otras*, queda una expresión así:

$$x_i = \frac{1}{a_{ii}}(b_i - a_{i1}x_1 - \cdots - a_{ii-1}x_{i-1} - a_{ii+1}x_{i+1} - \cdots - a_{in}x_n),$$

en forma matricial,

$$x = D^{-1}(b - (A - D)x),$$

donde D es la matriz diagonal cuyos elementos no nulos son exactamente los de la diagonal de A. En fin, puede escribirse, por tanto,

$$x = D^{-1}b - D^{-1}(A - D)x,$$

que es una ecuación de tipo *punto fijo*. Si la matriz $D^{-1}(A - D)$ cumple las condiciones del Teorema 4, entonces la iteración de Jacobi converge para cualquier semilla x_0 y se tiene la cota del Teorema 5. Para comprobar las condiciones, hace falta calcular $D^{-1}(A - D)$, aunque puede comprobarse de otras maneras (véase el Lema 7).

3.2. El algoritmo de Gauss-Seidel. Si en lugar de utilizar la diagonal de la matriz A para descomponerla, se utiliza la parte triangular inferior (incluyendo la diagonal), se obtiene un sistema de la forma

$$x = T^{-1}b - T^{-1}(A - T)x,$$

que también es una ecuación de *punto fijo*. Si la matriz $T^{-1}(A - T)$ cumple las condiciones del Teorema 4, entonces la iteración de Gauss-Seidel converge para cualquier semilla x_0, y se tiene la cota del Teorema 5. Para comprobar las condiciones, haría falta calcular $T^{-1}(A - T)$, aunque puede comprobarse de otras maneras.

Definición 12. Se dice que una matriz $A = (a_{ij})$ es *estrictamente diagonal dominante por filas* si

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

para todo i entre 1 y n.

Para estas matrices, tanto el método de Jacobi como el de Gauss-Seidel son convergentes:

Lema 7. *Si A es una matriz* estrictamente diagonal dominante por filas*, entonces tanto el método de Jacobi como el de Gauss-Seidel convergen para cualquier sistema de la forma $Ax = b$.*

Para el de Gauss-Seidel, además, se tiene que

Lema 8. *Si A es una matriz simétrica definida positiva, entonces el método de Gauss-Seidel converge para cualquier sistema de la forma $Ax = b$.*

4. Anexo: Código en Matlab/Octave

Se incluye el código de varios de los algoritmos descritos, usable tanto en Matlab como en Octave.

4.1. El algoritmo de Gauss. El siguiente código implementa el algoritmo de reducción de Gauss para un sistema $Ax = b$ y devuelve la matriz L, la matriz U y el vector b transformado, *suponiendo que los multiplicadores por defecto nunca son* 0. Si alguno es cero, se abandona el procedimiento.

La entrada ha de ser:

- `A:` una matriz cuadrada (si no es cuadrada, la salida es la triangulación por la diagonal principal),
- `b:` un vector con tantas filas como columnas tiene `A`.

La salida es una terna `L, At, bt`, como sigue:

```
function [L, At, bt] = gauss(A,b)
  n = size(A);
  m = size(b);
  if(n(2) ~= m(1))
    warning('The sizes of A and b do not match');
    return;
  end
  At=A; bt=b; L=eye(n);
  k=1;
  while (k<n(1))
    l=k+1;
    if(At(k,k) == 0)
      warning('There is a 0 on the diagonal');
      return;
    end
    % careful with rows & columns:
    % L(l,k) means ROW l, COLUMN k
    while(l<=n)
      L(l,k)=At(l,k)/At(k,k);
      % Combining rows is easy in Matlab
      At(l,k:n) = [0 At(l,k+1:n) - L(l,k) * At(k,k+1:n)];
      bt(l)=bt(l)-bt(k)*L(l,k);
      l=l+1;
    end
    k=k+1;
  end
end
```

FIGURA 11. Código del algoritmo de reducción de Gauss

L: es la matriz triangular inferior (de los multiplicadores),
At: es la matriz A transformada (la reducida), que se denomina U en la factorización LU, y es triangular superior.
bt: el vector transformado.

De manera, que el sistema que se ha de resolver, equivalente al anterior, es $At \times x = bt$.

4.2. El algoritmo LUP. Como se vio arriba, el algoritmo de reducción de Gauss está sujeto a que no aparezca ningún cero como pivote y además puede dar lugar a errores importantes de redondeo si el pivote es pequeño. A continuación se muestra un código en Matalb/Octave que implementa el algoritmo LUP, por el cual se obtiene una factorización $LU = PA$, donde L y U son triangular inferior y superior, respectivamente, la diagonal de L está formada por 1 y P es una matriz de permutación. La entrada es:

A: una matriz cuadrada de orden n.
b: un vector de n filas.

Devuelve cuatro matrices, `L`, `At`, `P` y `bt`, que corresponded a L, U, P y al vector transformado según el algoritmo.

```
function [L, At, P, bt] = gauss_pivotaje(A,b)
  n = size(A);
  m = size(b);
  if(n(2) ~= m(1))
    warning('Dimensions of A and b do not match');
    return;
  end
  At=A;
  bt=b;
  L=eye(n);
  P=eye(n);
  i=1;
  while (i<n)
    j=i+1;
    % beware nomenclature:
    % L(j,i) is ROW j, COLUMN i
    % the pivot with greatest absolute value is sought
    p = abs(At(i,i));
    pos = i;
    for c=j:n
      u = abs(At(c,i));
      if(u>p)
        pos = c;
        p = u;
      end
    end
    if(u == 0)
      warning('Singular system');
      return;
    end
    % Swap rows i and p if i != p
    % in A and swap left part of  L
    % This is quite easy in Matlab, there is no need
    % for temporal storage
    P([i pos],:) = P([pos i], :);
    if(i ~= pos)
      At([i pos], :) = At([pos i], :);
      L([i pos], 1:i-1) = L([pos i], 1:i-1);
      b([i pos], :) = b([pos i], :);
    end
    while(j<=n)
      L(j,i)=At(j,i)/At(i,i);
      % Combining these rows is easy
      % They are 0 up to column i
      % And combining rows is easy as above
      At(j,i:n) = [0 At(j,i+1:n) - L(j,i)*At(i,i+1:n)];
      bt(j)=bt(j)-bt(i)*L(j,i);
      j=j+1;
    end
    i=i+1;
  end
end
```

FIGURA 12. Código del algoritmo *LUP*

CAPÍTULO 4

Interpolación

Dada una colección de datos, la *tentación* humana es utilizarlos como fuentes de conocimiento en lugares desconocidos. Específicamente, dada una lista de coordenadas ligadas a un tipo de suceso (un experimento, unas mediciones...) (x_i, y_i), lo "natural" es intentar utilizarla para *deducir* o *predecir* el valor que tomaría la y si la x fuera cualquier otra. Este es el *afán interpolador y extrapolador* del hombre. No se puede hacer nada para evitarlo. Lo que se puede hacer es calcular las maneras más razonables de llevar a cabo dicha interpolación.

En todo este tema se parte de una lista de datos

$$\begin{array}{c|c|c|c|c|c} \mathbf{x} & x_0 & x_1 & \dots & x_{n-1} & x_n \\ \hline \mathbf{y} & y_0 & y_1 & \cdots & y_{n-1} & y_n \end{array} \tag{4}$$

que se supone ordenada en las coordenadas $\mathbf{x}$, que además son diferentes: $x_i < x_{i+1}$. El objetivo es encontrar funciones que —de alguna manera— tengan relación (cierta *cercanía*) con dicha lista de datos o *nube de puntos*.

1. Interpolación lineal (a trozos)

La primera idea (sencilla pero funcional) es utilizar la función definida a trozos entre x_0 y x_n que consiste en "unir cada punto con el siguiente con una recta". Esto se denomina *interpolación lineal a trozos* ó *spline lineal* (se definirá *spline* más adelante con generalidad).

Definición 13. La *función de interpolación lineal a trozos* de la tabla (4) es la función $f : [x_0, x_n] \to \mathbb{R}$ definida de la siguiente manera:

$$f(x) = \frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x - x_{i-1}) + y_{i-1} \quad \text{si } x \in [x_{i-1}, x_i]$$

es decir, la función definida a trozos que consiste en los segmentos que unen (x_{i-1}, y_{i-1}) con (x_i, y_i), definida desde x_0 hasta x_n, para $i = 1, \dots, n$.

La interpolación lineal tiene varias características que la hacen interesante:

- Es *muy* sencilla de calcular.

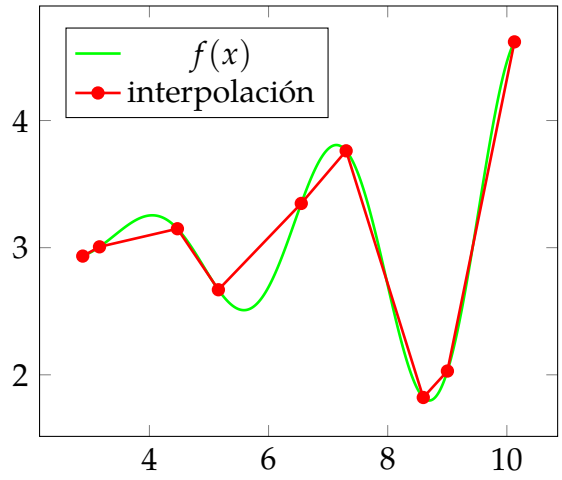

FIGURA 1. Interpolación lineal de una nube de 9 puntos. Compárese con la función original (en verde).

- Pasa por todos los puntos de la tabla de datos.
- Es continua.

Por eso se utiliza con frecuencia para representar funciones (es lo que hace Matlab por defecto), pues si la nube de puntos es densa, los segmentos serán pequeños y las esquinas se notarán poco.

La pega de la interpolación lineal son precisamente las esquinas que aparecen siempre que la tabla de datos no corresponda a puntos de una recta.

2. ¿Tendría sentido interpolar con parábolas?

Está claro que la interpolación lineal a trozos está abocada a generar esquinas siempre que la tabla de datos no corresponda a una función lineal. En general, la interpolación busca no solo una función que *pase por todos los puntos*, sino que sea *razonablemente suave* (por motivos no solo visuales sino de aproximación a la realidad). Se podría intentar mejorar la aproximación lineal con funciones de grado mayor, exigiendo que las tangentes en los puntos intermedios fueran iguales. Podría intentarse con segmentos parabólicos: puesto que tienen tres grados de libertad, al segmento i-ésimo se le puede exigir que pase por los puntos (x_{i-1}, y_{i-1}) y (x_i, y_i) y que tenga la misma derivada en x_i que el siguiente. Esto puede parecer razonable pero tiene cierta pega difícilmente salvable: genera una asimetría intrínseca al método (si uno plantea las ecuaciones de dicho sistema,

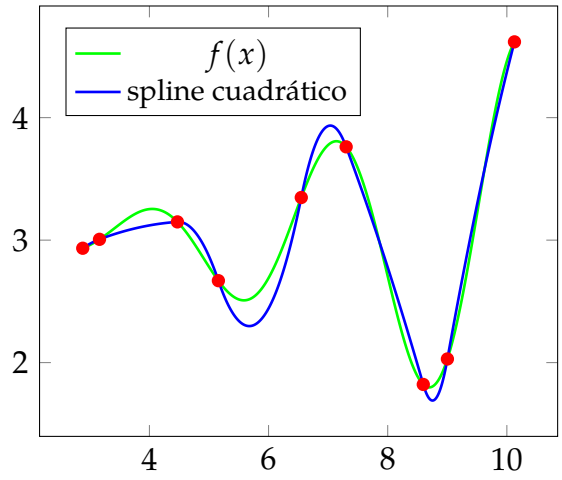

FIGURA 2. Interpolación cuadrática de una nube de 9 puntos. Compárese con la función original (en verde).

falta exactamente una por imponer para que sea compatible determinado; esto hace que sea diferente el caso de un número par de nodos y un número impar, o que la función de interpolación sea asimétrica para datos simétricos). Sin entrar en detalles, puede comprobarse que este *spline cuadrático* no es optimo, aunque aproxime la nube de puntos de manera que no haya esquinas.

Tiene más problemas (por ejemplo, la curva se desvía mucho de la nube de puntos si hay puntos cercanos en x pero lejanos en y en sentidos diferentes). No se utiliza prácticamente nunca, salvo en construcción, por ejemplo, pues los arcos muy tendidos se aproximan con parábolas.

El caso siguiente, el *spline cúbico* es el más utilizado: de hecho, es lo que los programas de dibujo vectorial utilizan para trazar arcos (aunque no con una tabla como (4), sino con dos tablas, pues las curvas son parametrizadas como $(x(t), y(t))$).

3. Splines cúbicos: curvatura continua

Para aproximar la nube de puntos con polinomios de grado tres, se utiliza la siguiente definición:

DEFINICIÓN 14. Un *spline cúbico* de la tabla (4) es una función $f : [x_0, x_n] \to \mathbb{R}$ tal que:

- La función f es un polinomio de grado 3 en cada segmento $[x_{i-1}, x_i]$, para $i = 1, \ldots, n$.

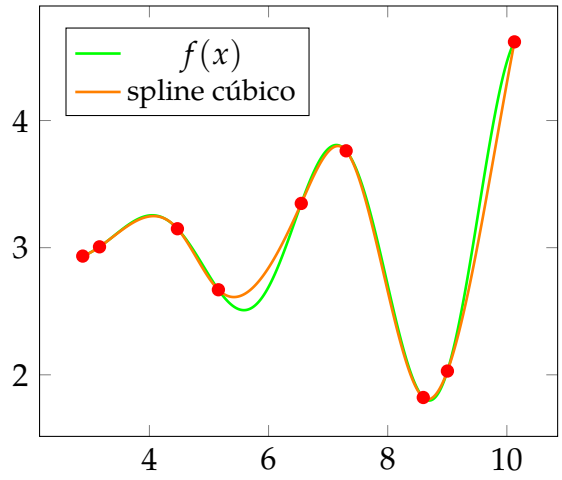

FIGURA 3. Spline cúbico de una nube de 9 puntos. Compárese con la función original (en verde).

- La función f es derivable dos veces con continuidad en todos los puntos x_i, para $i = 1, \ldots, n-1$.

De aquí se deduce que, si se llama P_i al polinomio de grado 3 que coincide con f en $[x_{i-1}, x_i]$, entonces $P_i'(x_i) = P_{i+1}'(x_i)$ y $P_i''(x_i) = P_{i+1}''(x_i)$; esto junto con el hecho de que $P_i(x_i) = y_i$ y $P_i(x_{i+1}) = y_{i+1}$ impone 4 condiciones para cada polinomio P_i, salvo para el primero y el último, para los que solo hay 3 (esta es la *simetría* que poseen los splines cúbicos y no poseen los cuadráticos). Por tanto, las condiciones de spline cúbico determinan *casi* unívocamente los polinomios P_i. Hace falta una condición en cada polinomio extremal (en P_1 y P_n) para determinarlo totalmente. Estas condiciones pueden ser, *por ejemplo*:

- Que la derivada segunda en los puntos extremos sea 0. A esto se le denomina el *spline cúbico natural*, pero no tiene por qué ser el mejor para un problema concreto. Las ecuaciones son $P_1''(x_0) = 0$ y $P_n''(x_n) = 0$.
- Que la derivada *tercera* coincida en los puntos casi-extremos: $P_1'''(x_1) = P_2'''(x_1)$ y $P_n'''(x_{n-1}) = P_{n-1}'''(x_{n-1})$. Se dice que este spline es *extrapolado*.
- Que haya cierta condición de periodicidad: $P_1'(x_0) = P_n'(x_n)$ y lo mismo con la derivada segunda: $P_1''(x_0) = P_n''(x_n)$. Esto tiene su interés, por ejemplo, si se está interpolando una función periódica.

3.1. El cálculo del spline cúbico: una matriz tridiagonal. Para calcular *efectivamente* el spline cúbico conviene normalizar la notación. Como arriba, se llamará P_i al polinomio correspondiente al segmento $[x_{i-1}, x_i]$ y se escribirá *relativo al punto* x_{i-1}:

$$P_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3,$$

de modo que se han de calcular los a_i, b_i, c_i y d_i a partir de la tabla de datos $(\mathbf{x}, \mathbf{y})$ que se supone dada como arriba (4). La siguiente normalización consiste en, en lugar de utilizar continuamente $x_i - x_{i-1}$, llamar

$$h_i = x_i - x_{i-1}, \text{ para } i = 1, \ldots, n$$

(es decir, utilizar la anchura de los n intervalos en lugar de las coordenadas x_i propiamente dichas).

Para aclarar todo el razonamiento, los datos conocidos se escribirán en negrita.

El razonamiento es como sigue:

- Los a_i se calculan directamente, pues $P_i(\mathbf{x_{i-1}}) = a_i$ y tiene que ser igual a $\mathbf{y_{i-1}}$. Por tanto,

$$a_i = \mathbf{y_{i-1}} \text{ para } i = 1, \ldots, n$$

- Se ha de cumplir que $P_i(\mathbf{x_i}) = \mathbf{y_i}$, así que, utilizando la condición anterior y llevando $\mathbf{y_{i-1}}$ al miembro de la derecha, queda

$$b_i\mathbf{h_i} + c_i\mathbf{h_i}^2 + d_i\mathbf{h_i}^3 = \mathbf{y_i} - \mathbf{y_{i-1}}. \tag{5}$$

  para $i = 1, \ldots, n$. Esto da n ecuaciones.
- La condición de derivada continua es $P_i'(\mathbf{x_i}) = P_{i+1}'(\mathbf{x_i})$, así que

$$b_i + 2c_i\mathbf{h_i} + 3d_i\mathbf{h_i}^2 = b_{i+1}, \tag{6}$$

  para $i = 1, \ldots n-1$. De aquí salen $n-1$ ecuaciones.
- Finalmente, las derivadas segundas han de coincidir en los puntos intermedios, así que

$$2c_i + 6d_i\mathbf{h_i} = c_{i+1}, \tag{7}$$

  para $i = 1, \ldots, n-1$. Esto da $n-1$ ecuaciones.

En total se obtienen (aparte de las a_i, que son directas), $3n-2$ ecuaciones para $3n$ incógnitas (las, b, c y d). Como ya se dijo, se imponen condiciones en los extremos, pero al finalizar todo el razonamiento.

Una vez enunciadas las ecuaciones, se realizan simplificaciones y sustituciones para obtener un sistema más *inteligible*. De hecho, se

despejan las d y las b en función de las c y queda un sistema lineal en las c. Se procede como sigue:

Primero, se utiliza la ecuación (7) para despejar las d_i:

$$d_i = \frac{c_{i+1} - c_i}{3\mathbf{h_i}} \tag{8}$$

y sustituyendo en (5), queda (hasta despejar b_i):

$$b_i = \frac{\mathbf{y_i} - \mathbf{y_{i-1}}}{\mathbf{h_i}} - \mathbf{h_i}\frac{c_{i+1} + 2c_i}{3}; \tag{9}$$

por otro lado, sustituyendo el d_i en (5) y operando hasta despejar b_i, queda

$$b_i = b_{i-1} + \mathbf{h_{i-1}}(c_i + c_{i-1}). \tag{10}$$

para $i = 2, \ldots, n$. Ahora no hay más que utilizar la ecuación (9) para i y para $i-1$ e *introducirla* en esta última, de manera que solo queden c. Tras una colección de operaciones simples, se obtiene que

$$\mathbf{h_{i-1}}c_{i-1} + (2\mathbf{h_{i-1}} + 2\mathbf{h_i})c_i + \mathbf{h_i}c_{i+1} = 3\left(\frac{\mathbf{y_i} - \mathbf{y_{i-1}}}{\mathbf{h_i}} - \frac{\mathbf{y_{i-1}} - \mathbf{y_{i-2}}}{\mathbf{h_{i-1}}}\right) \tag{11}$$

para $i = 2, \ldots, n$.

En forma matricial (ya sin utilizar negritas, para no recargar), queda un sistema de ecuaciones $Ac = \alpha$, donde A es

$$A = \begin{pmatrix} h_1 & 2(h_1+h_2) & h_2 & 0 & \ldots & 0 & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & h_{n-1} & 2(h_{n-1}+h_n) & h_n \end{pmatrix} \tag{12}$$

y c indica el vector columna de $c_1, \ldots, c_n$, mientras que α es el vector columna

$$\begin{pmatrix} \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_n \end{pmatrix}$$

con

$$\alpha_i = 3\left(\frac{y_i - y_{i-1}}{h_i} - \frac{y_{i-1} - y_{i-2}}{h_{i-1}}\right).$$

Como se ve, este sistema es compatible *indeterminado* (le faltan dos ecuaciones para tener solución única). Las ecuaciones se suelen imponer, como se explicó, como condiciones en los extremos. Por ejemplo, el *spline natural* significa que $c_1 = 0$ y que $c_n = 0$, así que A se

completaría por arriba con una fila $(1\ 0\ 0\ \ldots\ 0)$ y por abajo con otra $(0\ 0\ 0\ \ldots\ 1)$, mientras que al vector α se le añadiría una primera componente igual a 0 y una última también igual a 0. Con estas n ecuaciones se calcularían las c_i y a partir de ellas, utilizando las ecuaciones (8) y (9), se calculan los valores de las demás variables.

El sistema de ecuaciones de un spline cúbico, somo se ve por la ecuación (12), es *tridiagonal*: esto significa que solo tiene elementos diferentes de cero en la diagonal principal y en las dos diagonales adyacentes. (Esto es cierto para el spline natural y para cualquiera que imponga condiciones en c_1 y c_n directamente). Estos sistemas se resuelven muy fácilmente con una factorización LU —o bien, puede implementarse directamente la solución como un algoritmo en función de las α_i y las h_i. En cualquier caso, este tipo de sistemas tridiagonales es importantes reconocerlos y saber que su factorización LU es rapidísima (pues si se piensa en el algoritmo de Gauss, cada operación de *hacer ceros* solo requiere hacerlo en una fila por debajo de la diagonal).

3.2. El algoritmo: sencillo, resolviendo sistemas lineales. Tras todo lo dicho, puede enunciarse el algoritmo para calcular el spline cúbico que interpola una tabla de datos $\mathbf{x}, \mathbf{y}$ de longitud $n+1$, $\mathbf{x} = (x_0, \ldots, x_n)$, $\mathbf{y} = (y_0, \ldots, y_n)$, en la que $x_i < x_{i+1}$ (y por tanto todos los valores de $\mathbf{x}$ son diferentes), como indica el algoritmo 9.

3.3. Una cota sobre la aproximación. El hecho de que el spline cúbico sea *gráficamente* satisfactorio no significa que sea *técnicamente útil*. En realidad, lo es más de lo que parece. Si una función se "comporta bien" en la cuarta derivada, entonces el spline cúbico la aproxima razonablemente bien (y mejor cuanto más estrechos sean los intervalos entre nodos). En concreto:

Teorema 6. *Sea $f : [a,b] \to \mathbb{R}$ una función derivable 4 veces con continuidad y supóngase que $|f^{4)}(x)| < M$ para $x \in [a,b]$. Sea h el máximo de las anchuras $x_i - x_{i-1}$ para $i = 1, \ldots, n$. Si $s(x)$ es el spline cúbico para los puntos $(x_i, f(x_i))$ que satisface que $f'(x_0) = s'(x_0)$ y $f'(x_n) = s'(x_n)$, entonces*

$$|s(x) - f(x)| \leq \frac{5M}{384} h^4.$$

Este resultado puede ser de gran utilidad, por ejemplo, para calcular integrales o para acotar errores en soluciones de ecuaciones diferenciales (que viene a ser lo mismo que integrar, vaya).

Algoritmo 9 (Cálculo del spline cúbico.)

Input: una tabla de datos **x**, **y** como se especifica arriba y **dos** condiciones sobre el primer nodo y el último (una en cada uno)
Output: un spline cúbico que interpola los puntos de dicha tabla. Específicamente, una lista de n cuaternas (a_i, b_i, c_i, d_i) tales que los polinomios $P_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3$ constituyen un spline cúbico para la tabla

⋆INICIO
$a_i \leftarrow y_{i-1}$ para i desde 1 hasta n
$h_i \leftarrow x_i - x_{i-1}$ para i desde 1 hasta n
$i \leftarrow 1$
while $i \leq n$ **do**
  if $i > 1$ **and** $i < n$ **then**
    $F_i \leftarrow (0 \ \cdots \ 0 \ h_{i-1} \ 2(h_{i-1} + h_i) \ h_i \ 0 \ \cdots \ 0)$
    $\alpha_i = 3(y_i - y_{i-1})/h_i - 3(y_{i-1} - y_{i-2})/h_{i-1}$
  else
    $F_i \leftarrow$ la fila correspondiente a la ecuación para P_i
    α_i el coeficiente correspondiente a la condición para P_i
  end if
  $i \leftarrow i + 1$
end while
$M \leftarrow$ la matriz cuyas filas son las F_i para $i = 1$ hasta n
$c \leftarrow M^{-1}\alpha$ (resolver el sistema $Mc = \alpha$)
$i \leftarrow 1$
while $i < n$ **do**
  $b_i \leftarrow (y_i - y_{i-1})/h_i - h_i(c_{i+1} + 2c_i)/3$
  $d_i \leftarrow (c_{i+1} - c_i)/(3h_i)$
  $i \leftarrow i + 1$
end while
$b_n \leftarrow b_{n-1} + h_{n-1}(c_n + c_{n-1})$
$d_n \leftarrow (y_n - y_{n-1} - b_n h_n - c_n h_n^2)/h_n^3$
return (a, b, c, d)

3.4. Definición de spline general. Se prometió incluir la deficinión general de spline:

DEFINICIÓN 15. Dada una nube de puntos como (4), un *spline de grado n*, para $n > 0$, que los interpola es una función $f : [x_0, x_n] \to \mathbb{R}$ tal que

- Pasa por todos los puntos: $f(x_i) = y_i$,
- Es derivable $n - 1$ veces,

- En cada intervalo $[x_i, x_{i+1}]$ coincide con un polinomio de grado n.

Es decir, es una función polinomial a trozos de grado n que pasa por todos los puntos y es derivable $n-1$ veces (se entiende que ser derivable 0 veces significa ser continua).

De hecho, los que se utilizan son los de grado 1 y 3.

4. El polinomio interpolador de Lagrange: un solo polinomio para todos los puntos

Hasta ahora se han estudiado *splines*, funciones que son polinómicas a trozos, para interpolar los datos de una tabla como (4), imponiendo la condición de que la función interpoladora pase por todos los puntos de la tabla.

Se puede plantear el problema "límite": buscar un polinomio que pase por todos los puntos. Naturalmente, se procurará que sea de grado mínimo (para, entre otras cosas, simplificar la búsqueda). Es relativamente sencillo comprobar que existe uno de grado n (hay, recuérdese, $n+1$ datos) y que es único con esta condición. Este es el *polinomio interpolador de Lagrange*:

Teorema 7 (del polinomio interpolador de Lagrange). *Dada una lista de puntos como* (4) (*recuérdese que* $x_i < x_{i+1}$), *existe un único polinomio de grado n que pasa por cada* (x_i, y_i) *para* $i = 0, \ldots, n$.

La prueba del resultado es relativamente elemental. Tómese un problema algo más "sencillo": dados $n+1$ valores $x_0 < \cdots < x_n$, ¿se puede construir un polinomio $p_i(x)$ de grado n que valga 1 en x_i y cero en x_j para $j \neq i$? Con estas condiciones, se busca un polinomio de grado n que tenga n ceros ya prefijados; es obvio que ha de ser múltiplo de $\phi_i(x) = (x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)$. Lo único que se ha de hacer es multiplicar a $\phi_i(x)$ por una constante para que valga 1 en x_i. El valor de $\phi_i(x)$ en x_i es

$$\phi_i(x_i) = (x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n) = \prod_{j\neq i}(x_i-x_j),$$

así que se ha de tomar

$$p_i(x) = \frac{\prod_{j\neq i}(x-x_j)}{\prod_{j\neq i}(x_i-x_j)}. \tag{13}$$

Estos polinomios, para $i = 0, \ldots, n$, se llaman *polinomios de base*. Como se ha visto, cada uno de ellos vale 1 en el x_i correspondiente y 0

en el resto, así que pueden pensarse como los vectores de la base estándar de $\mathbb{R}^{n+1}$. El vector $(y_0, y_1, \ldots, y_n)$ es el que se quiere expresar como combinación lineal de estos vectores, así que el polinomio que pasa por los todos puntos (x_i, y_i) para $i = 0, \ldots, n$ es:

$$\begin{aligned} P(x) &= y_0 p_0(x) + y_1 p_1(x) + \cdots + y_n p_n(x) = \sum_{i=0}^{n} y_i p_i(x) \\ &= \sum_{i=0}^{n} y_i \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}. \end{aligned} \tag{14}$$

Para comprobar que es único, basta tener en cuenta que, si hubiera otro, la diferencia sería un polinomio de grado n con $n + 1$ ceros, así que la diferencia sería el polinomio nulo (y por tanto los dos que pasan por (x_i, y_i) serían iguales).

El polinomio interpolador de Lagrange tiene algunas ventajas pero tiene un par de graves inconvenientes:

- Los denominadores que aparecen pueden ser muy pequeños cuando hay muchos puntos y dar lugar a errores de redondeo.
- Es demasiado *curvo*.

El primer problema es intrínseco, pues los denominadores que aparecen hay que calcularlos. El segundo depende esencialmente de la distribución de puntos. Un ejemplo famoso e importante se debe a Runge y ejemplifica el *fenómeno de Runge*: si se utilizan puntos equidistantes para interpolar una función con derivadas grandes en valor absoluto, el polinomio interpolador de Lagrange se desvía mucho (mucho, realmente) de la función, aunque pase por todos los puntos de interpolación. Esto no le ocurre a los splines cúbicos (un spline cúbico de once puntos de la función de Runge es indistinguible de ella, por ejemplo).

Para solventar el problema de la *curvatura* del polinomio interpolador de Lagrange cuando se trata de interpolar una función conocida, se utilizan métodos basados en la idea de minimizar el valor máximo que pueda tomar el polinomio

$$P(x) = (x - x_0)(x - x_1) \ldots (x - x_n),$$

es decir, se busca una distribución de los puntos x_i que resuelva un problema de tipo *minimax* (minimizar un máximo). Sin entrar en detalles técnicos, dado el intervalo $[-1, 1]$, se tiene que

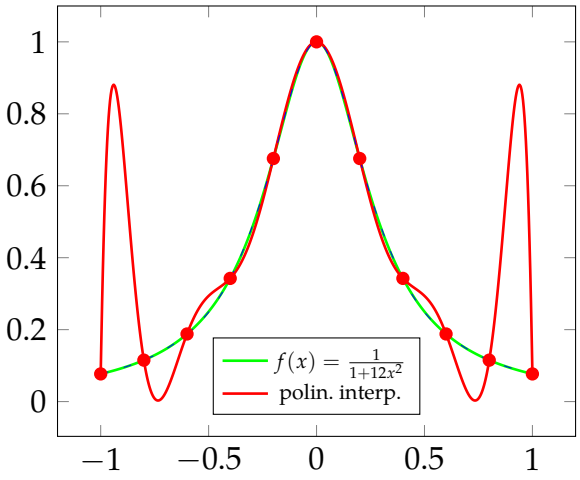

FIGURA 4. Fenómeno de Runge: el polinomio interpolador de Lagrange (rojo) se aleja ilimitadamente de la función si los nodos están equiespaciados. Los trazos negros siguen el spline cúbico (indistinguible de $f(x)$.)

LEMA 9. *Los puntos $x_0, \ldots, x_n$ que minimizan el valor absoluto máximo del polinomio $P(x)$ en el intervalo $[-1, 1]$ vienen dados por la fórmula*

$$x_i = \cos\left(\frac{2k+1}{n+1}\frac{\pi}{2}\right)$$

Por tanto, los puntos correspondientes para el intervalo $[a, b]$, donde $a, b \in \mathbb{R}$, son

$$\tilde{x_i} = \frac{(b-a)x_i + (a+b)}{2}.$$

A los puntos del lemma se les denomina *nodos de Chebychev* de un intervalo $[a, b]$. Son los que se han de utilizar si se quiere aproximar una función por medio del polinomio interpolador de Lagrange.

5. Interpolación aproximada

En problemas relacionados con estudios estadísticos o experimentales, los datos se suponen sujetos a errores, así que tratar de interpolar una nube de puntos por medio de funciones que pasan por todos ellos puede no tener ningún interes (de hecho, casi nunca lo tiene, en este contexto). Para delimitar el problema con precisión, hace falta indicar *qué significa interpolar*, pues cada problema puede tener una idea distinta de lo que significa que *una función se parezca a otra*. En el caso discreto (en el que se tiene una tabla de datos como (4), lo habitual es intentar minimizar la distancia cuadrática de $f(x_i)$ a y_i,

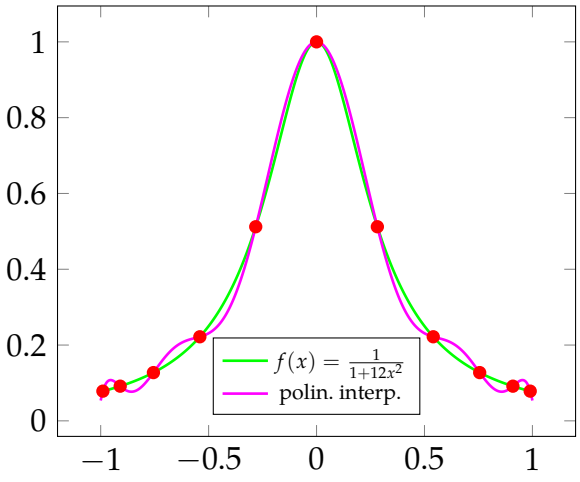

FIGURA 5. Aproximación de la función de Runge por el polinomio de Lagrange utilizando los nodos de Chebychev. El error máximo cometido es mucho menor.

donde f sería la función de interpolación. Pero podría interesar algo diferente (p.ej. minimizar la distancia de la gráfica de f a los puntos (x_i, y_i), un problema diferente del anterior), o cualquier otro criterio —el que mejor corresponda a los datos.

5.1. Interpolación por mínimos cuadrados. El método más utilizado de interpolación, sin duda alguna, es el de *mínimos cuadrados*. Se parte de una nube de puntos $\mathbf{x}, \mathbf{y}$, donde $\mathbf{x}$ e $\mathbf{y}$ son vectores de tamaño n arbitrarios (es decir, puede haber valores repetidos en las $\mathbf{x}$ y el orden es irrelevante). Dada una función $f : \mathbb{R} \to \mathbb{R}$, se define:

DEFINICIÓN. El *error cuadrático* de f en el punto x_i (una componente de $\mathbf{x}$) es el valor $(f(x) - y_i)^2$. El *error cuadrático total* de f en la nube dada por $\mathbf{x}$ e $\mathbf{y}$ es la suma

$$\sum_{i=0}^{n} (f(x_i) - y_i)^2.$$

El problema de la interpolación por mínimos cuadrados consiste en, dada la nube de puntos, encontrar, *en un conjunto determinado de funciones* una que minimice el error cuadrático total. El matiz de "en un conjunto de funciones" es crucial. De hecho, en estas notas se estudiará con precisión el problema en un espacio vectorial de funciones y se tratará de manera aproximada el de algunos conjuntos que no son espacios vectoriales.

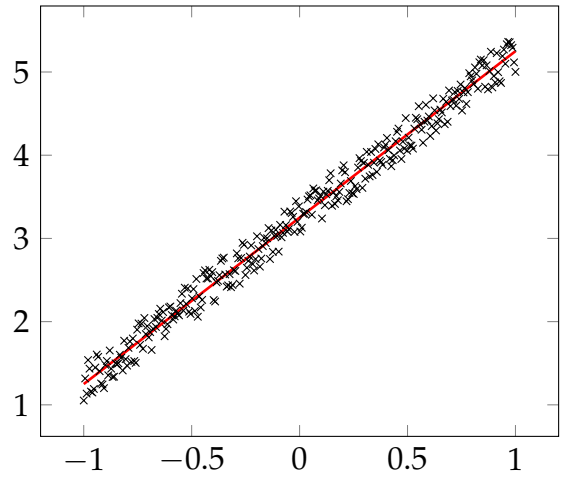

FIGURA 6. Interpolación de una nube de puntos por mínimos cuadrados lineales por una ecuación del tipo $y = ax + b$.

5.1.1. *Mínimos cuadrados lineales.* Supóngase que ha de aproximarse una nube de puntos de tamaño N mediante una función f que pertenece a un espacio vectorial V de funciones y que se conoce una base de V: $f_1, \ldots, f_n$. Se supone siempre que $N > n$ (y de hecho, habitualmente, N es mucho mayor que n). Es decir, se busca la función $f \in V$ tal que

$$\sum_{i=1}^{N} (f(x_i) - y_i)^2$$

es mínimo. Pero, puesto que V es un espacio vectorial, dicha función es una combinación lineal de los elementos de la base:

$$f = a_1 f_1 + a_2 f_2 + \cdots + a_n f_n.$$

Y, en realidad, se puede plantear el problema como la búsqueda de los coeficientes $a_1, \ldots, a_n$ que hacen mínima la función

$$F(a_1, \ldots, a_n) = \sum_{i=1}^{N} (a_1 f_1(x_i) + \cdots + a_n f_n(x_i) - y_i)^2,$$

es decir, dada $F(a_1, \ldots, a_n)$, una función de n variables, se ha de calcular un mínimo. La expresión de F indica claramente que es una función diferenciable; por tanto, el punto que dé el mínimo resuelve las ecuaciones correspondientes a hacer las parciales iguales a 0. Es

decir, hay que resolver el sistema

$$\frac{\partial F}{\partial a_1} = 0, \frac{\partial F}{\partial a_2} = 0, \ldots, \frac{\partial F}{\partial a_n} = 0.$$

Si se escribe la parcial de F con respecto a a_j, queda

$$\frac{\partial F}{\partial a_j} = \sum_{i=1}^{N} 2 f_j(x_i)(a_1 f_1(x_i) + \cdots + a_n f_n(x_i) - y_i) = 0.$$

Si, para simplificar, se denomina

$$\mathbf{y_j} = \sum_{i=1}^{N} f_j(x_i) y_i,$$

uniendo todas las ecuaciones y escribiendo en forma matricial, sale el sistema

$$\begin{pmatrix} f_1(x_1) & f_1(x_2) & \ldots & f_1(x_N) \\ f_2(x_1) & f_2(x_2) & \ldots & f_2(x_N) \\ \vdots & \vdots & \ddots & \vdots \\ f_n(x_1) & f_n(x_2) & \ldots & f_n(x_N) \end{pmatrix} \begin{pmatrix} f_1(x_1) & f_2(x_1) & \ldots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \ldots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_N) & f_2(x_N) & \ldots & f_n(x_N) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \tag{15}$$

$$= \begin{pmatrix} \mathbf{y_1} \\ \mathbf{y_2} \\ \vdots \\ \mathbf{y_n} \end{pmatrix},$$

que puede escribirse

$$XX^t a = Xy$$

donde se sobreentiende que la matriz X es la lista (por filas) de los valores de cada f_i en los puntos x_j, y la y es el vector columna $(y_0, y_1, \ldots, y_n)^T$, es decir:

$$X = \begin{pmatrix} f_1(x_1) & f_1(x_2) & \ldots & f_1(x_N) \\ f_2(x_1) & f_2(x_2) & \ldots & f_2(x_N) \\ \vdots & \vdots & \ddots & \vdots \\ f_n(x_1) & f_n(x_2) & \ldots & f_n(x_N) \end{pmatrix}, \quad y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Este sistema es compatible determinado si hay al menos tantos puntos como la dimensión de V y las funciones f_i generan filas independientes en la matriz X.

Es bastante probable que el sistema dado por (15) no esté muy bien acondicionado.

5.1.2. *Interpolación no lineal: aproximaciones.* En bastantes casos se requiere aproximar una nube de puntos por una función que pertenezca a una familia que no forme un espacio vectorial. Un ejemplo clásico es tratar de encontrar una función del tipo

$$f(x) = ae^{bx^2} \tag{16}$$

que aproxime una cierta nube de puntos. Las funciones de ese tipo dan lugar (para $a, b > 0$) a "campanas de Gauss". Está claro que dichas funciones no forman un espacio vectorial, así que no se pueden aplicar las técnicas del apartado anterior.

Sin embargo, puede intentar trasladarse el problema a una familia lineal, aproximar esta por mínimos cuadrados y calcular la función original equivalente.

Por seguir con el ejemplo, si se toman logaritmos en ambos lados de la ecuación (16), se obtiene la expresión

$$\log(f(x)) = \log(a) + bx^2 = a' + bx^2,$$

y, considerando las funciones 1 y x^2, se puede ahora aproximar la nube de puntos $\mathbf{x}, \log(\mathbf{y})$, en lugar de la original, utilizando mínimos cuadrados lineales (pues 1 y x^2 forman un espacio vectorial). Si se obtiene la solución $\log(a_0), \log(b_0)$, entonces puede pensarse que

$$g(x) = e^{a_0}e^{b_0x^2}$$

será una buena aproximación de la nube de puntos original $\mathbf{x}, \mathbf{y}$. Pero *es importante ser consciente de que* esta aproximación posiblemente no sea la mejor respecto del error cuadrático total. Piénsese que, al tomar logaritmos, los puntos de la nube cercanos a 0 estarán cerca de $-\infty$, con lo cual tendrán un peso mayor que los que al principio están cerca de 1 (que pasan a estar cerca de 0): al tomar logaritmos, los errores absolutos y relativos cambian de manera no lineal. Más aun, si uno de los valores de $\mathbf{x}$ es 0, el problema no puede convertirse con el logaritmo, y ha de eliminarse ese valor o cambiarse por otro aproximado...

6. Código de algunos algoritmos

6.1. El spline cúbico natural. Matlab, por defecto, calcula splines cúbicos con la condición *not-a-knot*, que es cierta relación entre las derivadas terceras en los puntos segundo y penúltimo. Solo se pueden calcular splines naturales con un *toolbox*. El código que sigue implementa el cálculo del spline cúbico natural de una colección de puntos. La entrada es:

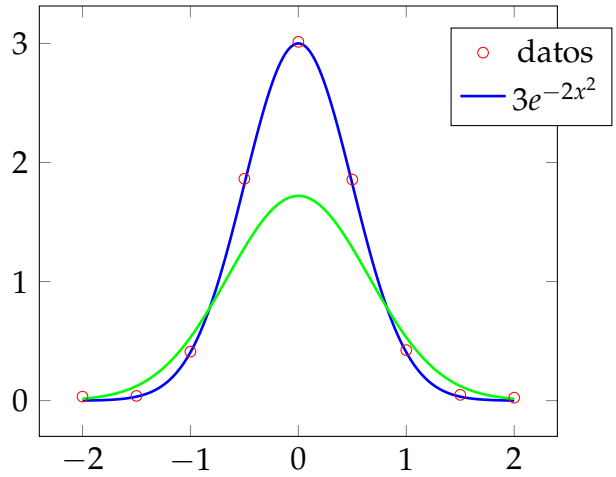

FIGURA 7. Interpolación no lineal tomando logaritmos: la nube de puntos se parece mucho a la función $f(x)$, pero la interpolación lineal tomando logaritmos y luego haciendo la exponencial es *muy mala* (verde). El problema reside en los valores de y muy cercanos a 0: al tomar logaritmos, adquieren una importancia desproporcionada.

x: la lista de las coordenadas x de los puntos
y: la lista de las coordenadas y de los puntos

Devuelve un "objeto" de Matlab que se denomina *polinomio a trozos*, que describe exactamente un objeto polinomial definido a trozos: los intervalos en los que está definido vienen dados por el vector `x` y su valor en un t (que hay que computar utilizando la función `ppval`) viene dado por el valor del polinomio correspondiente al intervalo de las `x` que contiene a t.

Un ejemplo de uso podría ser el siguiente, para comparar el spline cúbico con la gráfica de la función seno:

```
> x = linspace(-pi, pi, 10);
> y = sin(x);
> f = spline_cubico(x, y);
> u = linspace(-pi, pi, 400);
> plot(u, ppval(f, u)); hold on;
> plot(u, sin(u), 'r');
```

6.2. El polinomio interpolador de Lagrange. El cálculo del Polinomio interpolador de Lagrange en Matlab/Octave es extremadamente simple, como se puede ver mirando el Código 9.

La entrada es la siguiente:

```
% natural cubic spline: second derivative at both
% endpoints is 0. Input is a pair of lists describing
% the cloud of points.
function [f] = spline_cubico(x, y)
  % safety checks
  n = length(x)-1;
  if(n<=1 | length(x) ~= length(y))
    warning('Wrong data')
    f= [];
    return
  end

  % variables and coefficients for the linear system,
  % these are the ordinary names. Initialization
  a = y(1:n);
  h = diff(x);
  F = zeros(n);
  alpha = zeros(n, 1);

  % true coefficients (and independent terms) of the linear system
  for k=1:n
    if(k> 1& k < n)
      F(k,[k-1 k k+1]) = [h(k-1), 2*(h(k-1) + h(k)), h(k)] ;
      alpha(k) = 3*(y(k+1)-y(k))/h(k) - 3*(y(k) - y(k-1))/h(k-1);
    else
      % these two special cases are the 'natural' condition
      % (second derivatives at endpoints = 0)
      F(k,k) = 1;
      alpha(k) = 0;
    end
    k=k+1;
  end

  % These are the other coefficients of the polynomials
  % (a + bx + cx^2 + dx^3)... Initialization
  c = (F\alpha)';
  b = zeros(1,n);
  d = zeros(1,n);
  % unroll all the coefficients as in the theory
  k = 1;
  while(k<n)
    b(k) = (y(k+1)-y(k))/h(k) - h(k) *(c(k+1)+2*c(k))/3;
    k=k+1;
  end
  d(1:n-1) = diff(c)./(3*h(1:n-1));

  % the last b and d have explicit expressions:
  b(n) = b(n-1) + h(n-1)*(c(n)+c(n-1));
  d(n) = (y(n+1)-y(n)-b(n)*h(n)-c(n)*h(n)^2)/h(n)^3;

  % finally, build the piecewise polynomial (a Matlab function)
  % we might implement it by hand, though
  f = mkpp(x,[d; c; b ;a ]');
end
```

FIGURA 8. Código del algoritmo para calcular un spline cúbico

```
% Lagrange interpolation polynomial
% A single base polynomial is computed at
% each step and then added (multiplied by
% its coefficient) to the final result.
% input is a vector of x coordinates and
% a vector (of equal length) of y coordinates
% output is a polynomial in vector form (help poly).
function [l] = lagrange(x,y)
  n = length(x);
  l = 0;
  for m=1:n
    b = poly(x([1:m-1 m+1:n]));
    c = prod(x(m)-x([1:m-1 m+1:n]));
    l = l + y(m) * b/c;
  end
end
```

FIGURA 9. Código del algoritmo para calcular el polinomio interpolador de Lagrange

x: El vector de las coordenadas **x** de la nube de puntos que se quiere interpolar
y: El vector de las coordenadas **y** de la nube

La salida es un polinomio *en forma vectorial*, es decir, una lista de los coeficientes $a_n, a_{n-1}, \ldots, a_0$ tal que el polinomio $P(x) = a_n x^n + \cdots + a_1 x + a_0$ es el polinomio interpolador de Lagrange para la nube de puntos $(\mathbf{x}, \mathbf{y})$.

6.3. Interpolación lineal. Para la interpolación lineal en Matlab/Octave se ha de utilizar un objeto especial: un *array de celdas*: la lista de funciones que forman la base del espacio vectorial se ha de introducir *entre llaves*.

La entrada es la siguiente:

x: El vector de las coordenadas **x** de la nube de puntos que se quiere interpolar
y: El vector de las coordenadas **y** de la nube
F: Un array de celdas de funciones anónimas. Cada elemento es una función de la base del espacio vectorial que se utiliza para interpolar

La salida es un vector de coeficientes c, tal que $c_1 F_1 + \cdots + c_n F_n$ es la función de interpolación lineal de mínimos cuadrados de la nube $(\mathbf{x}, \mathbf{y})$.

Un ejemplo de uso podría ser el siguiente, para aproximar con funciones trigonométricas

```
% interpol.m
% Linear interpolation.
% Given a cloud (x,y), and a Cell Array of functions F,
% return the coefficients of the least squares linear
% interpolation of (x,y) with the base F.
#
% Input:
% x: vector of scalars
% y: vector of scalars
% F: Cell array of anonymous functions
#
% Outuput:
% c: coefficients such that
% c(1)*F{1,1} + c(2)*F{1,2} + ... + c(n)*F{1,n}
% is the LSI function in the linear space <F>.

function [c] = interpol(x, y, F)
  n = length(F);
  m = length(x);
  X = zeros(n, m);
  for k=1:n
    X(k,:) = F{1,k}(x);
  end
  A = X*X.';
  b = X*y.';
  c = (A\b)';
end
```

FIGURA 10. Código del algoritmo para calcular la interpolación lineal por mínimos cuadrados.

```
> f1=@(x) sin(x); f2=@(x) cos(x); f3=@(x) sin(2*x); f4=@(x) cos(2*x);
> f5=@(x)sin(3*x); f6=@(x)cos(3*x);
> F={f1, f2, f3, f4, f5, f6};
> u=[1,2,3,4,5,6];
> r=@(x) 2*sin(x)+3*cos(x)-4*sin(2*x)-5*cos(3*x);
> v=r(u)+rand(1,6)*.01;
> interpol(u,v,F)
ans =
   1.998522   2.987153  -4.013306  -0.014984  -0.052338  -5.030067
```

CAPÍTULO 5

Derivación e Integración Numéricas

Se expone con brevedad la aproximación de la derivada de una función mediante la fórmula simétrica de incrementos y cómo este tipo de fórmula puede generalizarse para derivadas de órdenes superiores. El problema de la inestabilidad de la derivación numérica se explica someramente.

La integración se trata con más detalle (pues el problema es mucho más estable que el anterior), aunque también con brevedad. Se definen las nociones de fórmulas de cuadratura de tipo interpolatorio y de grado de precisión, y se explican las más sencillas (trapecios y Simpson) en el caso simple y en el compuesto.

1. Derivación Numérica: un problema inestable

A veces se requiere calcular la derivada aproximada de una función mediante fórmulas que no impliquen la derivada, bien sea porque esta es difícil de computar —costosa—, bien porque realmente no se conoce la expresión simbólica de la función a derivar. En el primer caso se hace necesario sobre todo utilizar fórmulas de derivación numérica lo más precisas posibles, mientras que en el segundo caso lo que se requerirá será, casi con certeza, conocer una cota del error cometido. En estas notas solo se va a mostrar cómo la fórmula simétrica para calcular derivadas aproximadas es *mejor* que la aproximación naíf consistente en calcular el cociente del incremento asimétrico. Se expone la fórmula equivalente para la derivada segunda y se da una idea de la inestabilidad del método.

1.1. La derivada aproximada simétrica. Una idea simple para calcular la derivada de una función f en un punto x es, utilizando la noción de derivada como límite de las secantes, tomar la siguiente aproximación:

$$f'(x) \simeq \frac{f(x+h) - f(x)}{h}, \tag{17}$$

donde h será un *incremento pequeño* de la variable independiente.

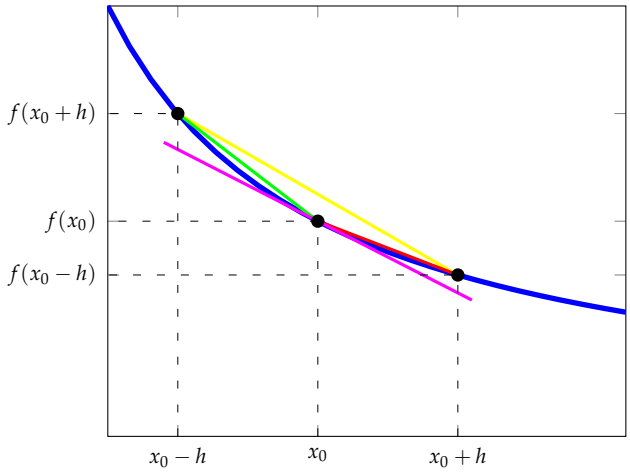

FIGURA 1. Derivada aproximada: por la derecha (rojo), por la izquierda (verde) y centrada (amarillo). La centrada es mucho más parecida a la tangente (magenta).

Sin embargo, la propia fórmula (17) muestra su debilidad: ¿se ha de tomar h positivo o negativo? Esto no es irrelevante. Supongamos que $f(x) = 1/x$ y se quiere calcular la derivada en $x = 1/2$. Utilicemos aproximaciones de $|h| = .01$. Sabemos que $f'(0.5) = 0.25$. Con la aproximación "natural", se tiene, por un lado,

$$\frac{f(x+.01)-f(x)}{.01} = \frac{\frac{1}{2.01}-\frac{1}{2}}{.01} = -0{,}248756+$$

mientras que, por otro,

$$\frac{f(x-.01)-f(x)}{-.01} = -\frac{\frac{1}{1.99}-\frac{1}{2}}{.01} = -0{,}251256+$$

que difieren en el segundo decimal. ¿Hay alguna de las dos que tenga preferencia *en un sentido abstracto*? No.

Cuando se tienen dos datos que aproximan un tercero y no hay ninguna razón abstracta para elegir uno por encima de otro, *parece razonable* utilizar la media aritmética. Probemos en este caso:

$$\frac{1}{2}\left(\frac{f(x+h)-f(x)}{h} + \frac{f(x-h)-f(x)}{-h}\right) = -0.2500062+$$

que aproxima la derivada real 0.25 con cuatro cifras significativas (cinco si se trunca).

Esta es, de hecho, la manera correcta de plantear el problema de la derivación numérica: utilizar la diferencia simétrica alrededor del

punto y dividir por dos veces el intervalo (es decir, la media entre la "derivada por la derecha" y la "derivada por la izquierda").

TEOREMA 8. *La aproximación naíf a la derivada tiene una precisión de orden* 1*, mientras que la fórmula simétrica tiene una precisión de orden* 2*.*

PRUEBA. Sea f una función con derivada tercera en el intervalo $[x-h, x+h]$. Utilizando el Polinomio de Taylor de grado uno, se tiene que, para cierto $\xi \in [x-h, x+h]$,

$$f(x+h) = f(x) + f'(x)h + \frac{f''(\xi)}{2}h^2,$$

así que, despejando $f'(x)$, queda

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{f''(\xi)}{2}h,$$

que es lo que significa que la aproximación tiene orden 2. Como se ve, el término de más a la derecha no puede desaparecer.

Sin embargo, para la fórmula simétrica se utiliza el Polinomio de Taylor *dos veces*, y de segundo grado:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f^{3)}(\xi)}{6}h^3,$$
$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f^{3)}(\zeta)}{6}h^3$$

para $\xi \in [x-h, x+h]$ y $\zeta \in [x-h, x+h]$. Restando ambas igualdades queda

$$f(x+h) - f(x-h) = 2f'(x)h + K(\xi, \zeta)h^3$$

donde K es una mera suma de los coeficientes de grado 3 de la ecuación anterior, así que

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + K(\xi, \zeta)h^2,$$

que es justo lo que significa que la fórmula simétrica es de segundo orden. □

2. Integración Numérica: fórmulas de cuadratura

La integración numérica, como corresponde a un problema *en el que los errores se acumulan* (un problema global, por así decir) es *más estable* que la derivación, por extraño que parezca (pese a que integrar simbólicamente es más difícil que derivar, la aproximación numérica se comporta mucho mejor).

En general, uno está interesado en *dar una fórmula* de integración numérica. Es decir, se busca una expresión algebraica general tal que, dada una función $f : [a, b] \to \mathbb{R}$ cuya integral se quiere aproximar, se pueda sustituir f en la expresión y obtener un valor —la *integral aproximada*.

DEFINICIÓN 16. Una *fórmula de cuadratura simple* para un intervalo $[a, b]$ es una colección de puntos $x_1 < x_2 < \cdots < x_{n+1} \in [a, b]$ y de valores $a_1, \ldots, a_{n+1}$. La *integral aproximada de una función f en $[a, b]$* utilizando dicha fórmula de cuadratura es la expresión

$$a_1 f(x_1) + a_2 f(x_2) + \cdots + a_{n+1} f(x_{n+1}).$$

Es decir, una fórmula de cuadratura no es más que "la aproximación de una integral utilizando valores intermedios y pesos". Se dice que la fórmula de cuadratura es *cerrada* si $x_1 = a$ y $x_{n+1} = b$; si ni a ni b están entre los x_i, se dice que la fórmula es *abierta*.

Si los puntos x_i están *equiespaciados*, la fórmula se dice que es *de Newton-Coates*.

Lógicamente, interesa encontrar las fórmulas de cuadratura que mejor aproximen integrales de funciones conocidas.

DEFINICIÓN 17. Se dice que una fórmula de cuadratura es de *orden n* si para cualquier polinomio $P(x)$ de grado n se tiene que

$$\int_a^b P(x)\, dx = a_1 P(x_1) + a_2 P(x_2) + \cdots + a_{n+1} P(x_{n+1}).$$

Es decir, si *integra con exactitud polinomios de grado n*.

Las fórmulas de cuadratura más sencillas son: la del punto medio (abierta, $n = 1$), la fórmula del trapecio (cerrada, $n = 2$) y la fórmula de Simpson (cerrada, $n = 3$). Se explican a continuación.

Se explican la fórmulas *simples* a continuación y luego se generalizan a sus versiones *compuestas*.

2.1. La fórmula del punto medio. Una manera burda pero natural de aproximar una integral es multiplicar el valor de la función en el punto medio por la anchura del intervalo. Esta es la fórmula del punto medio:

DEFINICIÓN 18. La fórmula de cuadratura del punto medio corresponde a $x_1 = (a + b)/2$ y $a_1 = (b - a)$. Es decir, se aproxima

$$\int_a^b f(x)\, dx \simeq (b - a) f\left(\frac{a + b}{2}\right).$$

por el área del rectángulo que tiene un lado horizontal a la altura del valor de f en el punto medio.

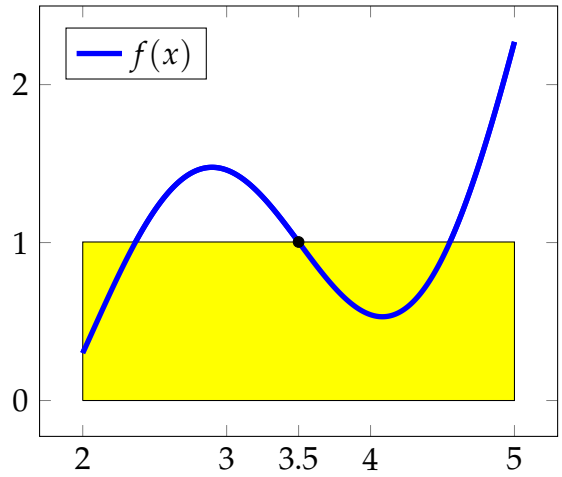

FIGURA 2. Fórmula del punto medio: se aproxima el área debajo de f por el rectángulo.

Es elemental comprobar que *la fórmula del punto medio es de orden* 1: integra con precisión funciones lineales pero no polinomios de segundo grado.

2.2. La fórmula del trapecio. La siguiente aproximación natural (no necesariamente óptima) es utilizar dos puntos. Puesto que ya hay dos dados, a y b, lo intuitivo es utilizarlos.

Dados dos puntos a y b, se tendrán los valores de f en cada uno de ellos. Se puede interpolar linealmente f con una recta y aproximar la integral por medio de esta o bien aproxmar la integral con dos rectángulos como en la regla del punto medio y hacer la media. Ambas operaciones producen el mismo resultado. En concreto:

DEFINICIÓN 19. La *fórmula del trapecio* para $[a, b]$ consiste en tomar $x_1 = a$, $x_2 = b$ y pesos $a_1 = a_2 = (b - a)/2$. Es decir, se aproxima

$$\int_a^b f(x)\,dx \simeq \frac{b-a}{2}\left(f(a) + f(b)\right)$$

la integral de f por el área del trapecio con base en OX, que une $(a, f(a))$ con $(b, f(b))$ y es paralelo al eje OY.

La fórmula del trapecio, pese a incorporar un punto más que la regla del punto medio, es también de orden 1.

2.3. La fórmula de Simpson. El siguiente paso *natural* (que no óptimo) es utilizar, en lugar de 2 puntos, 3 y en vez de aproximar mediante rectas, utilizar parábolas. Esta aproximación es singularmente eficaz, pues tiene orden 3. Se denomina *fórmula de Simpson*.

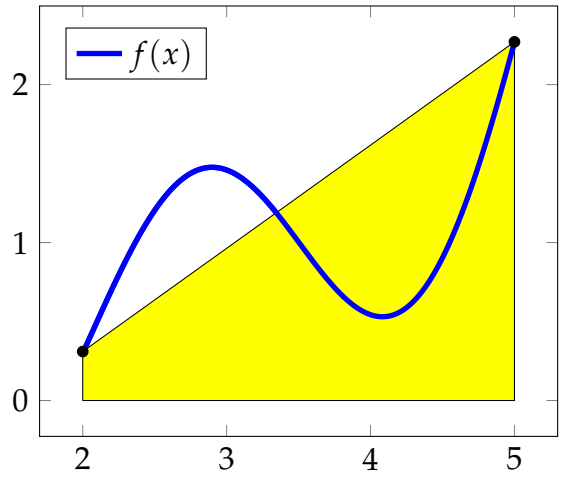

FIGURA 3. Fórmula del trapecio: se aproxima el área debajo de f por el trapecio.

DEFINICIÓN 20. La *fórmula de Simpson* es la fórmula de cuadratura que consiste en tomar $x_1 = a, x_2 = (a+b)/2$ y $x_3 = b$, y como pesos, los correspondientes a la interpolación correcta de un polinomio de grado 2. Es decir[1], $a_1 = \frac{b-a}{6}$, $a_2 = \frac{4(b-a)}{6}$ y $a_3 = \frac{b-a}{6}$. Así pues, consiste en aproximar

$$\int_a^b f(x)\,dx \simeq \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

la integral por una media ponderada de áreas de rectángulos intermedios. Esta fórmula es sorprendentemente precisa y *conviene sabérsela de memoria*: un sexto de la anchura por la ponderación $1, 4, 1$ de los valores en extremo, punto medio, extremo.

Lo singular de la fórmula de Simpson es que *es de tercer orden*: pese a *interpolar con parábolas*, se obtiene una fórmula que integra con precisión polinomios de tercer grado. Para terminar, téngase en cuenta que *la ecuación de la parábola que pasa por los tres puntos es irrelevante*: no hace falta calcular el polinomio interpolador, basta usar la fórmula de la Definición 20.

2.4. Las fórmulas compuestas. Las fórmulas de cuadratura compuestas consisten en aplicar una fórmula de cuadratura en subintervalos. En lugar de aproximar, por ejemplo, utilizando la fórmula del

[1]Esto es sencillo de calcular, usando un polinomio de grado 3, por ejemplo.

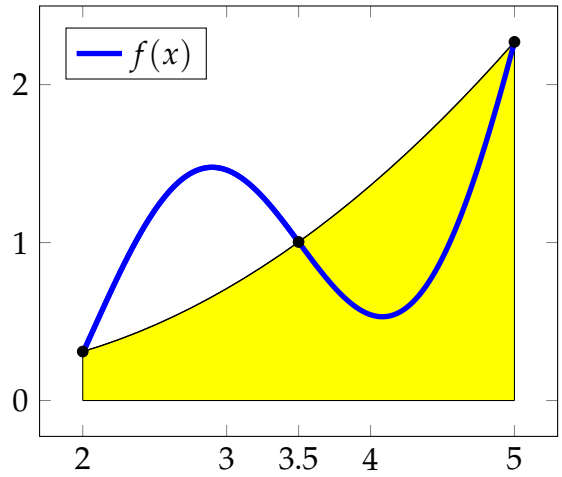

FIGURA 4. Fórmula de Simpson: se aproxima el área debajo de f por la parábola que pasa por los puntos extremos y el punto medio (en negro).

trapecio

$$\int_a^b f(x)\,dx \simeq \frac{b-a}{2}\left(f(a)+f(b)\right)$$

se *subdivide* el intervalo $[a,b]$ en subintervalos y se aplica la fórmula en cada uno de estos.

Podría hacerse sin más, mécanicamente, pero el caso es que, si la fórmula es cerrada, para las fórmulas de Newton-Coates, *siempre es más sencillo aplicar la fórmula general* que ir subintervalo a subintervalo, pues los valores en los extremos se ponderan.

Nota: aún así, al alumno se le aconseja no aprenderse las fórmulas siguientes de memoria, pues lo que suele ocurrir es que se haga un lío. Basta con aplicarlas subintervalo a subintervalo. Los siguientes apartados pueden omitirse, y de hecho es lo que se aconseja. De ahí que estén en gris claro.

2.4.1. *Fórmula compuesta de los trapecios.* Si se tienen dos intervalos consecutivos $[a,b]$ y $[b,c]$ *de igual longitud* (es decir, $b-a=c-b$) y se aplica la fórmula de los trapecios en $[a,b]$ y en $[b,c]$ para aproximar la integral total de f entre a y c, queda:

$$\int_a^c f(x)\,dx = \frac{b-a}{2}\left(f(b)+f(a)\right)+\frac{c-b}{2}\left(f(c)+f(b)\right) = \frac{h}{2}\left(f(a)+2f(b)+f(c)\right),$$

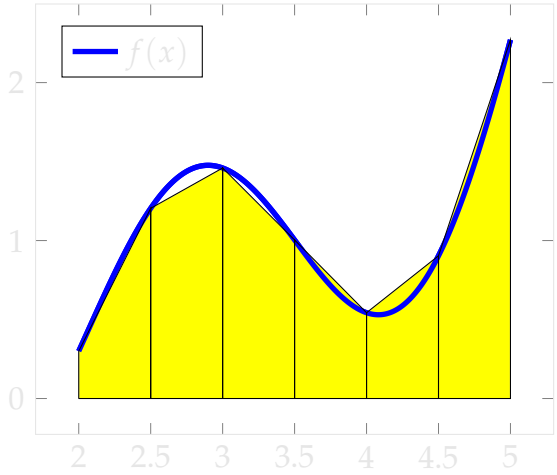

FIGURA 5. Fórmula del trapecio compuesta: simplemente repetir la fórmula del trapecio en cada subintervalo.

donde $h = b - a$ es la anchura *del subintervalo*, que es una ponderación de los valores en puntos extremos e intermedio). En el caso general, queda algo totalmente análogo:

DEFINICIÓN 21 (Fórmula de los trapecios compuesta). Dado un intervalo $[a, b]$ y un número de nodos $n + 1$, la *fórmula de los trapecios compuesta* para $[a, b]$ con n nodos viene dada por los nodos $x_0 = a, x_1 = a + h, \ldots, x_n = b$, con $h = (b - a)/(n - 1)$[2] y por la aproximación

$$\int_a^b f(x)\,dx \simeq \frac{h}{2}\left(f(a) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(b)\right).$$

Es decir, la mitad de la anchura de cada intervalo por la suma de los valores en los extremos y los dobles de los valores en los puntos *interiores*.

2.4.2. *Fórmula de Simpson compuesta.* De modo análogo, la fórmula de Simpson compuesta consiste en utilizar la fórmula de Simpson en varios intervalos que se unen para formar uno mayor. Como en la anterior, al ser iguales los extremos final e inicial de intervalos consecutivos, la fórmula final compuesta no es una simple *suma tonta* de todas las fórmulas intermedias: hay cierta combinación que simplifica el resultado final.

[2] Esto no es más que la anchura de cada subintervalo, no hay que liarse con la n.

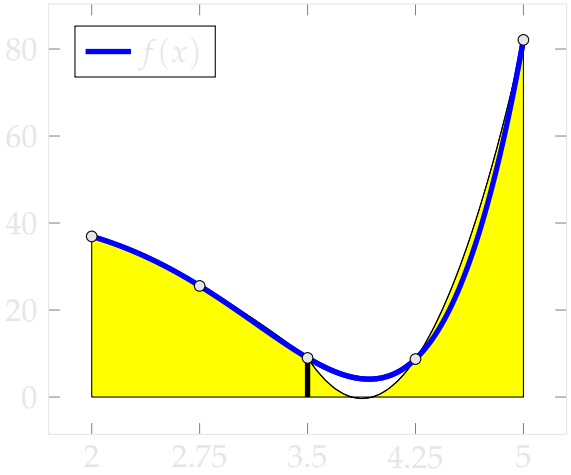

FIGURA 6. Fórmula de Simpson compuesta: conceptualmente, repetir la fórmula de Simpson en cada subintervalo.

DEFINICIÓN 22 (Fórmula de Simpson compuesta). Dado un intervalo $[a, b]$ dividido en n subintervalos (y por tanto, dado un número de nodos equiespaciados $2n + 1$), la *fórmula de Simpson compuesta* para $[a, b]$ con $2n + 1$ nodos viene dada por los nodos $x_0 = a, x_1 = a + h, \ldots, x_{2n} = b$, (así que $h = (b - a)/(2n)$) y por la aproximación

$$\int_a^b f(x)\,dx \simeq \frac{b-a}{6n}\left(f(a) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots \right.$$
$$\left. \cdots + 2f(x_{2n-2}) + 4f(x_{2n-1}) + f(b)\right).$$

Es decir, un tercio del *espacio entre nodos* multiplicado por: el valor en los extremos más el doble de la suma de los valores en los puntos extremos de los subintervalos más cuatro veces la suma de los valores en los puntos medios de los subintervalos.

Es decir, la fórmula de Simpson compuesta es la misma que la de Simpson en todos los aspectos *salvo en los puntos extremos de los intervalos*, donde hay que multiplicar por dos —obviamente, pues se están sumando. Es más fácil recordar la fórmula si simplemente se considera que hay que aplicar la fórmula simple en cada subintervalo y sumar todos los términos que salen.

CAPÍTULO 6

Ecuaciones diferenciales

Sin duda, la actividad primordial para la que se utilizan los métodos numéricos es *integrar ecuaciones diferenciales* (es la denominación clásica del proceso de resolución de una ecuación diferencial).

1. Introducción

Una ecuación diferencial es un tipo especial de ecuación funcional: una ecuación (es decir, una igualdad en la que alguno de los elementos es desconocido) en la que una de las incógnitas es una función. Ya se han resuelto ecuaciones diferenciales antes: cualquier integral es la solución de una ecuación diferencial. Por ejemplo,

$$y' = x$$

es una ecuación en la que se busca conocer una función $y(x)$ cuya derivada es x. Como se sabe, la solución *no es única*: hay una constante de integración, de manera que la solución general se escribe

$$y = \frac{x^2}{2} + C.$$

En realidad, la constante de integración tiene una manera más natural de entenderse. Cuando se calcula una primitiva, lo que se está buscando es una función cuya derivada es conocida. Una función, al fin y al cabo, no es más que una gráfica en el plano X, Y. Esta gráfica puede estar situada más arriba o más abajo (la derivada no cambia por dibujar una función paralelamente arriba o abajo). Sin embargo, si lo que se busca es resolver el problema

$$y' = x, \;\; y(3) = 28,$$

entonces se está buscando una función cuya derivada es x *con una condición puntual*: que su valor en 3 sea 28. Una vez que este valor está fijado, *solo hay una gráfica que tenga esa forma y pase por ese punto* (el $(3, 28)$): a esto se le llama una *condición inicial*: se busca una función cuya derivada cumpla algo, pero *imponiendo que pase por determinado*

punto: de este modo la constante ya no aparece, se puede determinar sin más sustituyendo:

$$28 = 3^2 + C \quad \Rightarrow \quad C = 19.$$

Esta es la idea de *condición inicial* para una ecuación diferencial.

Considérese la ecuación

$$y' = y$$

(cuya solución general debería ser obvia). Esta ecuación significa: la función $y(x)$ cuya derivada es igual a ella misma. Uno tiende siempre a pensar en $y(x) = e^x$, pero ¿es esta la única solución? Si el problema se piensa de otra manera, geométrica, la ecuación significa: la función $y(x)$ cuya derivada es igual a la altura (y) en cada punto de la gráfica. Pensada así, está claro que tiene que haber más de una función solución. De hecho, uno supone que por cada punto del plano (x, y) pasará al menos una solución (pues geométricamente no hay ninguna razón por la que los puntos (x, e^x) sean preferibles a otros para que pase por ellos una solución.) En efecto, la solución general es

$$y(x) = Ce^x,$$

donde C es una *constante de integración*. Ahora, si especificamos una condición inicial, que $y(x_0) = y_0$ para $x_0, y_0 \in \mathbb{R}$, entonces está claro que

$$y_0 = Ce^{x_0}$$

y por tanto $C = y_0/e^{x_0}$, que está siempre definido (pues el denominador no es 0.)

De este modo, dada una ecuación diferencial, hace falta especificar al menos una *condición inicial* para que la solución esté completamente determinada. Si se considera

$$y'' = -y,$$

no basta con especificar una sola condición inicial. ¿Qué soluciones tiene esta ecuación? En breve, son las de la forma $y(x) = a(x) + b\cos(x)$, para dos constantes $a, b \in \mathbb{R}$. Para que la solución esté totalmente determinada hay que fijar esas dos constantes. Esto, por ejemplo, puede hacerse imponiendo (es lo común) el valor en un punto, $y(0) = 1$ y el valor de la derivada en ese punto, $y'(0) = -1$. En cuyo caso, la solución es $y(x) = -(x) + \cos(x)$.

Así que este tema trata de la solución *aproximada* de ecuaciones diferenciales. De momento se estudiarán solo funciones de una variable y ecuaciones en las que solo aparece la primera derivada, pero esto puede variar en el futuro.

2. Lo más básico

Comencemos por el principio:

DEFINICIÓN 23. Una *ecuación diferencial ordinaria* es una igualdad $a = b$ en la que aparece la derivada (o una derivada de orden superior, pero no derivadas parciales) de una variable libre cuyo rango es el conjunto de funciones de una variable real.

Obviamente, así no se entiende nada. Se puede decir que una ecuación diferencial ordinaria es una ecuación en la que una de las incógnitas es una función de una variable $y(x)$, cuya derivada aparece explícitamente en dicha ecuación. Las *no ordinarias* son las ecuaciones *en derivadas parciales*, que no se tratan en este texto, de momento.

EJEMPLO 12. Arriba se han dado dos ejemplos, las ecuaciones diferenciales pueden ser de muchos tipos:

$$\begin{aligned} y' &= \sin(x) \\ xy &= y' - 1 \\ (y')^2 - 2y'' + x^2 y &= 0 \\ \frac{y'}{y} - xy &= 0 \end{aligned}$$

etc.

De ahora en adelante se supondrá que la ecuación diferencial de partida solo tiene una variable libre, que es la de la función que se busca, que se asume que es y.

DEFINICIÓN 24. Se dice que una ecuación diferencial tiene orden n si n es la mayor derivada de y que aparece. Si y y sus derivadas solo aparen como sumas y productos, se llama *grado* de la ecuación a la mayor potencia de una derivada de y que aparece.

En este capítulo estamos interesados exclusivamente en ecuaciones diferenciales *resueltas* (que no significa que ya estén resueltas, sino que tienen una estructura concreta):

$$y' = f(x, y).$$

DEFINICIÓN 25. Un *problema de condición inicial* es una ecuación diferencial junto con una condición inicial de la forma $y(x_0) = y_0$, donde $x_0, y_0 \in \mathbb{R}$.

DEFINICIÓN 26. La *solución general* de una ecuación diferencial E es una familia de funciones $f(x, c)$ donde c es una (o varias) constantes tal que:

- Cualquier solución de E tiene la forma $f(x, c)$ para algún c.
- Cualquier expresión $f(x, c)$ es una solución de E,

salvo para quizás un número finito de valores de c.

Lo primero que hay que tener en cuenta es que, si ya el problema de calcular una primitiva de una función es complejo, *integrar una ecuación diferencial* es, por lo general, imposible. Es decir, calcular la función que cumple una determinada ecuación diferencial y su condición inicial (o calcular la solución general) es un problema *que no se desea resolver en general*. Lo que se desea es (sobre todo en ingeniería), calcular una solución aproximada y conocer una buena cota del error cometido al utilizar dicha solución en lugar de la "real".

3. Discretización

En todo este capítulo, se supondrá que se tiene una función $f(x, y)$ de dos variables, definida en una región $x \in [x_0, x_n]$, $y \in [a, b]$, que cumple la siguiente condición:

Definición 27. Se dice que $f(x, y)$ definida en un conjunto $X \in \mathbb{R}^2$ *cumple la condición de Lipschitz* si existe una constante $K > 0$ tal que

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$$

para cualesquiera $x_1, x_2 \in X$, donde $|\ |$ denota el módulo de un vector.

La condición de Lipschitz es una forma de continuidad un poco fuerte (es decir, es más fácil ser continua que Lipschitz). Lo importante de esta condición es que asegura la existencia y unicidad de solución de las ecuaciones diferenciales "normales". Sea X un conjunto de la forma $[x_0, x_n] \times [a, b]$ (una banda vertical) y $f(x, y) : X \to \mathbb{R}$ una función de Lipschitz en X:

Teorema 9 (de Cauchy-Kovalevsky). *Si f cumple las condiciones anteriores, cualquier ecuación diferencial $y' = f(x, y)$ con una condición inicial $y(x_0) = y_0$ con $y_0 \in (a, b)$ tiene* una única solución $y = y(x)$ *definida en $[x_0, x_0 + t]$ para cierto $t \in \mathbb{R}$ mayor que 0.*

No es difícil que una función cumpla la condición de Lipschitz: basta con que sea, por ejemplo, diferenciable con continuidad en todos los puntos y tenga derivada acotada. De hecho, todos los polinomios y todas las funciones derivables que se han utilizado en este curso son de Lipschitz en conjuntos de la forma $[x_0, x_n] \times [a, b]$, salvo que tengan una discontinuidad o algún punto en el que no sean derivables. Por ejemplo, la función $f(x) = \sqrt{x}$ *no es de Lipschitz* si el

intervalo $[x_0, x_n]$ contiene al 0 (porque el crecimiento de f cerca de 0 es "vertical").

Ejemplo 13 (Bourbaki "Functions of a Real Variable", Ch. 4, §1). La ecuación diferencial $y' = 2\sqrt{|y|}$ con la condición inicial $y(0) = 0$ tiene infinitas soluciones. Por ejemplo, cualquier función definida así:

(1) $y(t) = 0$ para cualquier intervalo $(-b, a)$,
(2) $y(t) = -(t+\beta)^2$ para $t \leq -b$,
(3) $y(t) = (t-a)^2$ para $t \geq a$

es una solución de dicha ecuación diferencial. Esto se debe a que la función $\sqrt{|y|}$ *no es de Lipschitz* cerca de $y = 0$. (Compruébense ambas afirmaciones: que dichas funciones resuelven la ecuación diferencial y que la función $\sqrt{|y|}$ no es de Lipschitz).

Es decir, cualquier problema de condición inicial "normal" tiene una única solución. Lo difícil es calcularla con exactitud.

¿Y cómo puede calcularse de manera aproximada?

3.1. La derivada como flecha. Es costumbre pensar en la derivada de una función como *la pendiente* de la gráfica de dicha función en el punto correspondiente. Es más útil pensar que es la *coordenada Y del vector velocidad de la gráfica*.

Cuando se representa gráficamente una función, lo que se debe pensar es que se está trazando una curva con velocidad horizontal constante (el eje OX tiene la misma escala en todas partes, "se va de izquierda a derecha con velocidad uniforme"). De esta manera, una gráfica de una función $f(x)$ es una curva $(x, f(x))$. El vector tangente a esta curva es (derivando respecto del parámetro, que es x) el vector $(1, f'(x))$: la derivada de f indica la dirección vertical del vector tangente a la gráfica si la velocidad horizontal es constante.

De esta manera, se puede entender una ecuación diferencial de la forma $y' = f(x, y)$ como una ecuación que dice "se busca la curva $(x, y(x))$ tal que el vector velocidad en cada punto es $(1, f(x, y))$." Así que puede dibujarse el conjunto de vectores "velocidad" en el plano (x, y), que están dados por $(1, f(x, y))$ (la función f es conocida) y se trata de dibujar una curva que tenga como vector velocidad ese en cada punto y (la condición inicial) que pase por un punto concreto $(y(x_0) = y_0)$. Como se ve en la Figura 1, si se visualiza así, es sencillo hacerse una idea de cómo será la curva que se busca.

Si se tuvieran las flechas (los vectores $(1, f(x, y))$) dibujados en un plano, trazar una curva tangente a ellos no es necesariamente muy difícil. De hecho, si lo que se quiere es una aproximación, la idea más

intuitiva es "sustituir la curva por segmentos que van en la dirección de las flechas". Si el segmento se toma con longitud muy pequeña, uno piensa que se aproximará a la solución.

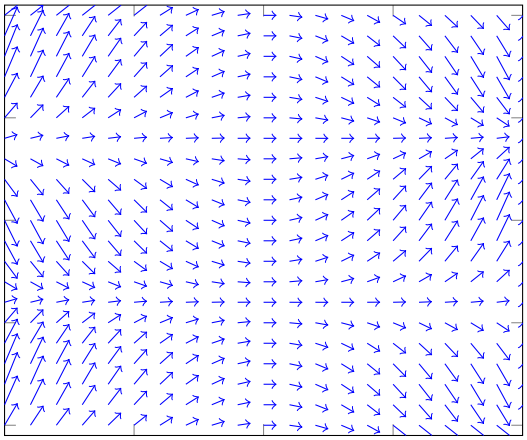

FIGURA 1. Flechas que representan vectores de una ecuación diferencial del tipo $y' = f(x, y)$. Obsérvese que todos los vectores tienen misma magnitud en horizontal.

Esta es precisamente la idea de Euler.

Ante un plano "lleno de flechas" que indican los vectores velocidad en cada punto, la idea más simple para trazar una curva que

- Pase por un punto especificado (condición inicial $y(x_0) = y_0$)
- Tenga el vector velocidad que indica $f(x, y)$ en cada punto

La idea más sencilla es *discretizar las coordenadas* x. Partiendo de x_0, se supone que el eje OX está cuantizado en intervalos de anchura h —de momento una constante fija— "lo suficientemente pequeña". Con esta hipótesis, en lugar de trazar una *curva suave*, se aproxima la solución *dando saltos de anchura* h en la variable x.

Como se exige que la solución pase por un punto concreto $y(x_0) = y_0$, no hay más que:

- Trazar el punto (x_0, y_0) (la condición inicial).
- Computar $f(x_0, y_0)$, que es el valor *vertical* del vector velocidad de la curva en cuestión en el punto inicial.
- Puesto que las coordenadas x están cuantizadas, el *siguiente* punto de la curva tendrá coordenada x igual a $x_0 + h$.

- Puesto que la velocidad en (x_0, y_0) es $(1, f(x_0, y_0))$, la aproximación más simple a *cuánto se desplaza la curva en un intervalo de anchura h en la x* es $(h, hf(x_0, y_0))$ (tomar la x como el tiempo y desplazarse justamente ese tiempo en línea recta). Llámense $x_1 = x_0 + h$ e $y_1 = y_0 + hf(x_0, y_0)$.
- Trácese el segmento que une (x_0, y_0) con (x_1, y_1): este es el primer "tramo aproximado" de la curva buscada.
- En este momento se tiene la misma situación que al principio pero con (x_1, y_1) en lugar de (x_0, y_0). Repítase el proceso con (x_1, y_1).

Y así hasta que se desee. Lo que se acaba de describir se conoce como el *algoritmo de Euler* para integrar numéricamente una ecuación diferencial.

4. [Contenido no esencial] Errores: runcamiento y redondeo

Obviamente, la solución de una ecuación diferencial $y' = f(x, y)$ no será siempre (ni casi nunca) una curva formada por segmentos rectilíneos: al aproximar la solución $y(x)$ por los segmentos dados por el método de Euler, se está cometiendo un error que puede analizarse con la fórmula de Taylor. Si se supone que $f(x, y)$ es suficientemente diferenciable, entonces $y(x)$ también lo será y se tendrá que

$$y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{h^2}{2}y''(\theta)$$

para cierto $\theta \in [x_0, x_0 + h]$. Como lo que se está haciendo es *olvidarse del último sumando*, el error que se produce es exactamente ese sumando: un término del tipo $O(h^2)$, de segundo orden *al hacer la primera iteración*. Este error, que aparece al aproximar una función por su desarrollo limitado de Taylor de un grado finito, se denomina *error de truncamiento* (pues se está truncando la expresión exacta de la función solución).

Por lo general, se supone siempre que el intervalo en que se integra la ecuación diferencial es del orden de magnitud de h^{-1}. De hecho, si el intervalo de cálculo es $[a, b]$, se fija un número de intervalos n y se toma $x_0 = a$, $x_n = b$ y $h = (b - a)/n$. Por tanto, realizar el cálculo desde $x_0 = a$ hasta $x_n = b$ requiere $n = 1/h$ iteraciones, de manera que el error de truncamiento *global* es del orden de $h^{-1}O(h^2) \simeq O(h)$: tiene magnitud similar a la anchura intervalo. Aunque, como se verá, esto hay que entenderlo bien, pues por ejemplo $(b - a)^{10}h$ es del orden de h, pero si $b \gg a$, entonces es mucho mayor que h. En estos

problemas en que se acumulan errores, es importante tener en cuenta que la acumulación puede hacer aparecer constantes muy grandes.

Pero el truncamiento no es la única parte de la que surgen errores. Como ya se ha insistido, calcular en coma flotante (que es lo que se hace en este curso y en todos los ordenadores, salvo raras excepciones) lleva implícito el *redondeo* de todos los datos a una precisión específica. En concreto, si se utiliza coma flotante de doble precisión como especifica el estándar IEEE-754, se considera que el número más pequeño *significativo* (el *épsilon* de dicho formato) es $\epsilon = 2^{-52} \simeq 2.2204 \times 10^{-16}$. Es decir, lo más que se puede suponer es que el error de redondeo es *menor que* ϵ en cada operación. Si se supone que "operación" significa "iteración del algoritmo", se tiene que cada paso de x_i a x_{i+1} genera un error de magnitud ϵ. Como hay que realizar $n = 1/h$ pasos desde x_0 hasta x_n, el *error de redondeo global* es del orden de ϵ/h. Por tanto, como ϵ es un dato *fijo* (no se puede mejorar en las circunstancias de trabajo), *cuanto más pequeño sea el intervalo entre x_i y x_{i+1}, mayor será el error de redondeo*.

Así pues, la suma del error de truncamiento y el de redondeo es de la forma (tomando infinitésimos equivalentes)

$$E(h) \simeq \frac{\epsilon}{h} + h,$$

que crece cuando h se hace grande y cuando h se hace pequeño. El mínimo de $E(h)$ se alcanza (esta es una fácil cuenta) cuando $h \simeq \sqrt{\epsilon}$: no tiene sentido utilizar intervalos de anchura menor que $\sqrt{\epsilon}$ porque el error de redondeo se magnificará. Por tanto: *tomar intervalos lo más pequeños posibles puede conducir a errores de gran magnitud.*

En el caso de doble precisión, la anchura más pequeña sensata es del orden de 10^{-8}: anchuras menores son inútiles, además de la sobrecarga de operaciones que exigen al ordenador. Si en algún momento parece necesario elegir una anchura menor, lo más probable es que lo que se requiera es elegir un algoritmo mejor.

Esta consideración sobre el error de redondeo y de truncamiento es independiente del método: cualquier algoritmo discreto incurrirá en un error de truncamiento inherente al algoritmo y en un error de redondeo. Lo que se ha de hacer es conocer cómo se comporta cada algoritmo en ambos casos.

4.1. Anchura variable. Hasta ahora se ha supuesto que el intervalo $[a, b]$ (o, lo que es lo mismo, $[x_0, x_n]$) se divide en subintervalos de la misma anchura h. Esto no es necesario en general. Pero simplifica la exposición lo suficiente como para que siempre supongamos

que los intervalos son de anchura constante, salvo que se indique explícitamente lo contrario (y esto es posible que no ocurra).

5. [Contenido no esencial] Solución de EDOs e integral

Supóngase que la ecuación diferencial $y' = f(x, y)$ es tal que la función f depende solo de la variable x. En este caso, la ecuación se escribirá

$$y' = f(x)$$

y significa *y es la función cuya derivada es* $f(x)$, es decir, es el problema de calcular una primitiva. Este problema ya se ha estudiado en su capítulo, pero está intrínsecamente ligado a casi todos los algoritmos de integración numérica y por ello es relevante.

Cuando la función $f(x, y)$ depende también de y, la relación es más difícil, pero también se sabe (por el Teorema Fundamental del Cálculo) que, si $y(x)$ es la solución de la ecuación $y' = f(x, y)$ con condición inicial $y(x_0) = y_0$, entonces, integrando ambos miembros, queda

$$y(x) = \int_{x_0}^{x} f(t, y(t))\, dt + y_0,$$

que no es el cálculo de una primitiva *pero se le parece bastante.* Lo que ocurre en este caso es que los puntos en los que hay que evaluar $f(x, y)$ no se conocen, pues justamente dependen de la gráfica de la soclución (mientras que al calcular una primitiva, el valor de $f(x, y)$ *no depende de* y). Lo que se hace es *intentar acertar lo más posible* con la información local que se tiene.

6. El método de Euler: integral usando el extremo izquierdo

Si se explicita el algoritmo para integrar una ecuación diferencial utilizando el método de Euler como en el Algoritmo 10, se observa que se aproxima cada valor siguiente y_{i+1} como el valor anterior al que se le suma el valor de f *en el punto anterior* multiplicado por la anchura del intervalo. Es decir, se está aproximando

$$\int_{x_i}^{x_{i+1}} f(t, y(t))\, dt \simeq (x_{i+1} - x_i) f(x_i, y_i) = h f(x_i, y_i).$$

Si la función $f(x, y)$ *no dependiera de* y, se estaría haciendo la aproximación

$$\int_a^b f(t)\, dt = (b - a) f(a),$$

que se puede denominar (por falta de otro nombre mejor) la *regla del extremo izquierdo*: se toma como área bajo $f(x)$ la anchura del intervalo por el valor en el extremo izquierdo de este.

Algoritmo 10 (Algortimo de Euler con aritmética exacta)

Input: Una función $f(x, y)$, una condición inicial (x_0, y_0), un intervalo $[x_0, x_n]$ y un paso $h = (x_n - x_0)/n$
Output: una colección de puntos $y_0, y_1, \ldots, y_n$ (que aproximan los valores de la solución de $y' = f(x, y)$ en la malla $x_0, \ldots, x_n$)

⋆INICIO
$i \leftarrow 0, n \leftarrow (x_n - x_0)/h$
while $i \leq n$ **do**
  $y_{i+1} \leftarrow y_i + hf(x_i, y_i)$
  $i \leftarrow i + 1, x_i \leftarrow x_{i-1} + h$
end while
return $(y_0, \ldots, y_n)$

Podría intentarse (se deja de momento como ejercicio) plantearse el problema "y si se quisiera utilizar el extremo derecho del intervalo, ¿qué habría que hacer?": esta pregunta da lugar al método implícito más simple, que todavía no estamos en condiciones de explicar.

7. [Contenido no esencial] Euler modificado: "punto medio"

En lugar de utilizar el extremo izquierdo del intervalo $[x_i, x_{i+1}]$ para "integrar" y calcular el siguiente y_{i+1}, sería preferible intentar utilizar la regla del punto medio como primera idea. El problema es que con los datos que se tienen, no se conoce cuál sea dicho punto medio (por la dependencia de y que tiene f). Aun así, se puede tratar de aproximar de la siguiente manera:

- Se utilizará un punto cercano a $P_i = (x_i, y_i)$ y cuya coordenada x esté mitad de la distancia entre x_i y x_{i+1}.
- Como no se conoce nada mejor, se hace la primera aproximación como en el algoritmo de Euler y se toma como extremo derecho $Q_i = (x_{i+1}, y_i + hf(x_i, y_i))$.
- Se calcula el punto medio del segmento $\overline{P_iQ_i}$, que es $(x_i + h/2, y_i + h/2f(x_i, y_i))$. Llámese k a su coordenada y.
- Se utiliza el valor de f en este punto para calcular el siguiente y_{i+1}: $y_{i+1} = y_i + hf(x_i + h/2, k)$.

Si $f(x, y)$ no depende de y, es sencillo comprobar que lo que se está haciendo con los pasos descritos es la aproximación

$$\int_a^b f(x)\,dx \simeq (b-a)f(\frac{a+b}{2}),$$

que es la *regla del punto medio* de integración numérica.

Este método se denomina *método de Euler modificado* y tiene un *orden* de precisión superior al de Euler; es decir, es de orden 2, lo que significa que el error acumulado en x_n es $O(h^2)$. Se describe en detalle en el Algoritmo 11.

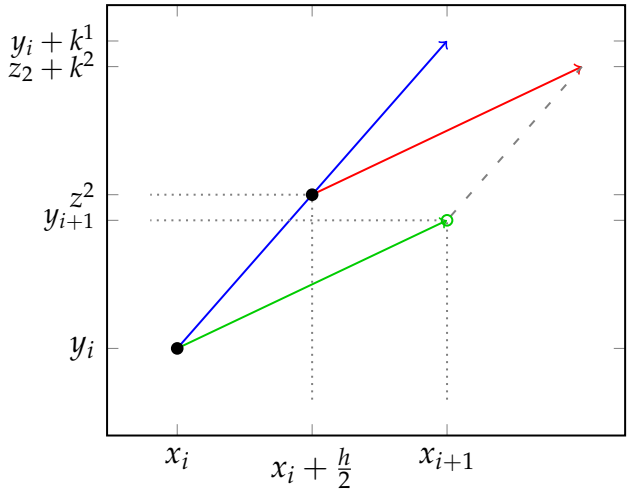

FIGURA 2. Ilustración de Algoritmo de Euler modificado. En lugar de sumar el vector $(h, hf(x_i, y_i))$ (azul), se suma en el punto (x_i, y_i) el vector verde, que corresponde al vector tangente dado en el punto medio (en rojo).

Como muestra la Figura 2, el algoritmo de Euler modificado consiste en utilizar, en lugar del vector tangente a la curva descrito por $(1, f(x_i, y_i))$, "llevar a (x_i, y_i)" el vector tangente en el punto medio del segmento que une (x_i, y_i) con el punto que correspondería a utilizar el algoritmo de Euler. Desde luego, se sigue cometiendo un error, pero al tomar la información *un poco más lejos*, se aproxima algo mejor el comportamiento de la solución real.

La siguiente idea es utilizar una "media" de vectores. En concreto, la media entre los dos vectores, el del "inicio del paso" y el del "final del paso de Euler."

Algoritmo 11 (Algortimo de Euler Modificado con aritmética exacta)

Input: Una función $f(x,y)$, una condición inicial (x_0,y_0), un intervalo $[x_0,x_n]$ y un paso $h=(x_n-x_0)/n$
Output: una colección de puntos $y_0,y_1,\ldots,y_n$ (que aproximan los valores de la solución de $y'=f(x,y)$ en la malla $x_0,\ldots,x_n$)

⋆INICIO
$i \leftarrow 0$
while $i \leq n$ **do**
$k^1 \leftarrow f(x_i,y_i)$
$z^2 \leftarrow y_i + \frac{h}{2}k^1$
$k^2 \leftarrow f(x_i+\frac{h}{2},z^2)$
$y_{i+1} \leftarrow y_i + hk^2$
$i \leftarrow i+1$
end while
return $(y_0,\ldots,y_n)$

8. Método de Heun (regla del trapecio)

En lugar de utilizar el punto medio del segmento que une el inicio y el final del método de Euler puede hacerse una operación parecida con los vectores: utilizar la media entre el vector al inicio del paso y el vector al final del paso de Euler. Este es el algoritmo de Euler *mejorado* ó algoritmo de Heun. Se describe en detalle en el Algoritmo 12.

Es decir, cada paso consiste en las siguientes operaciones:

- Se calcula el valor $k^1 = f(x_i,y_i)$.
- Se calcular el valor $z^2 = y_j + hk^1$. Esta es la coordenada y_{i+1} en el algoritmo de Euler.
- Se calcula el valor $k^2 = f(x_{i+1},z^2)$. Esta sería la pendiente en el punto (x_{i+1},y_{i+1}) en el algoritmo de Euler.
- Se calcula la media de k^1 y k^2: $\frac{k^1+k^2}{2}$ y se usa este valor como "paso", es decir: $y_{i+1} = y_i + \frac{h}{2}(k^1+k^2)$.

El método de Heun se describe gráficamente en la Figura 3.

9. El modelo Predador-Presa (o de Lotka-Volterra)

El primer sistema de ecuaciones diferenciales que modela un sistema biológico complejo es el de "Lotka-Volterra": se utiliza para describir la evolución de un entorno en el que dos especies conviven; una de ellas actúa como depredador y otra como presa (el ejemplo más habitual que se da es el de zorros y conejos).

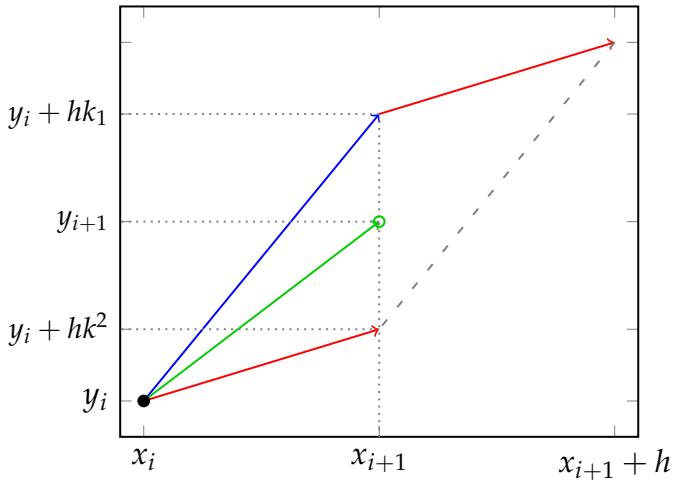

FIGURA 3. Ilustración de Algoritmo de Heun. En el punto (x_i, y_i) se añade la media (en verde) de los vectores correspondientes al punto (x_i, y_i) (azul) y al siguiente de Euler (en rojo).

Algoritmo 12 (Algortimo de Heun con aritmética exacta)

Input: Una función $f(x, y)$, una condición inicial (x_0, y_0), un intervalo $[x_0, x_n]$ y un paso $h = (x_n - x_0)/n$
Output: una colección de puntos $y_0, y_1, \ldots, y_n$ (que aproximan los valores de la solución de $y' = f(x, y)$ en la malla $x_0, \ldots, x_n$)
⋆INICIO
$i \leftarrow 0, n \leftarrow (x_n - x_0)/h$
while $i \leq n$ **do**
$m_e \leftarrow f(x_i, y_i)$
$y_e \leftarrow y_i + hk^1$
$m_f \leftarrow f(x_i + h, z^2)$
$y_{i+1} \leftarrow y_i + \frac{h}{2}(m_e + m_f)$
$i \leftarrow i + 1, x_i \leftarrow x_{i-1} + h$
end while
return $(y_0, \ldots, y_n)$

Supongamos que $x(t)$ denota la población de la especie "presa" en el momento t y que $y(t)$ denota la población de la especie "predador". La ecuación diferencial de Lotka-Volterra que describe la evolución conjunta de las dos especies se basa en las siguientes suposiciones (simplistas, claro está):

(1) La población de presas crece, sin mirar la interacción con los predadores, en proporción a su tamaño, pues tiene alimento suficiente (pongamos, yerba). En realidad, las muertes están incluidas en este apartado, pues basta reducir el incremento para incluirlas.
(2) Las presas, además, mueren como consecuencia de ser víctima de un depredador. Esto ocurre con una probabilidad constante.
(3) La población de depredadores muere, en ausencia de interacción con las presas, en proporción a su tamaño.
(4) Los predadores solo se multiplican en proporción a las presas que comen.

Estas cuatro reglas elementales (insistimos, muy simples), se trasladan a ecuaciones como se explica a continuación. Pese a lo elemental de la descripción, el modelo, como se verá presenta propiedades interesantes.

Del punto 1 se deduce que existe una constante $\alpha > 0$ tal que

$$\dot{x}(t) = \alpha x(t) + \dots$$

donde los puntos denotan otra expresión que habrá que descubrir. Si solo se tuviera esta ecuación, habría un crecimiento de presas exponencial (la ecuación $\dot{x}(t) = \alpha x(t)$ tiene como solución $x(t) = Ke^{\alpha t}$, donde K es el valor inicial).

Del punto 3 se deduce que existe un número $\gamma > 0$ tal que

$$\dot{y}(t) = -\gamma y(t) + \dots$$

igual que antes para las presas. Esto hace que, de por sí, los depredadores decrezcan según una ley exponencial negativa.

Es fácil convencerse de que la probabilidad de que un depredador se encuentre con una presa en algún lugar del terreno es proporcional al producto del número de predadores y de presas. Pero como no todo encuentro mutuo termina en una caza, se modela la probabilidad de que un predador coma a una presa como una constante β por dicho producto: $\beta x(t)y(t)$ (con $\beta > 0$). Por otro lado, el hecho de que un depredador se alimente no garantiza que se reproduzca; haremos que esto ocurra con un factor δ. Así, los puntos suspensivos que hemos dejado arriba han de sustituirse por $-\beta x(t)y(t)$ (en la parte de las presas) y por $\delta x(t)y(t)$ (en la parte de los depredadores). Se obtiene el sistema de ecuaciones diferenciales de Lotka-Volterra:

$$\begin{aligned} \dot{x}(t) &= \alpha x(t) - \beta x(t)y(t) \\ \dot{y}(t) &= -\gamma y(t) + \delta x(t)y(t) \end{aligned} \tag{18}$$

9.1. Cómo se aplica Heun al sistema Predador-Presa. En lugar de tratar de implementar el algoritmo de Euler y luego el de Heun para un sistema de ecuaciones, describimos directamente el segundo (así que el primero queda incluido en él). Como ya se explicó, el algoritmo de Euler es casi siempre inadecuado.

En el momento en que aparece más de una variable, los cálculos se multiplican pero la filosofía es siempre la misma: discretizar la variable independiente e ir *paso a paso* sin precipitarse.

El Problema de Condición Inicial es:

$$\begin{cases} \dot{x} = 0.8x - 0.4xy & x(0) = 18 \\ \dot{y} = -2y + 0.2xy & y(0) = 3 \end{cases}$$

(los valores iniciales en un sistema Predador-Presa siempre se conciben como *densidades de población*, no como números absolutos, pues si fuera esto llegaríamos a poblaciones de "medio zorro" o "doscientos conejos y cuarto").

La variable independiente es el tiempo, que denotamos con una t. Fijemos una discretización, por ejemplo $h = 0.25$ y calculemos dos pasos (más sería demasiado pesado).

La única diferencia con el caso de una variable es que, en cada paso, hay que hacer tantos cálculos como variables dependientes. Dos para este ejemplo.

La función que da la derivada de x con respecto a t es $f(t, x, y) = 0.8x - 0.4xy$. Para la variable y, la función correspondiente es $g(t, x, y) = -2y + 0.2xy$. Obsérvese —*esto es importante*— que tanto f como g podrían depender de t, aunque en este caso no ocurra así.

(1) **Primer paso,** $t_0 = 0$. Tenemos que $\tilde{x}_0 = x(0) = 18$ y que $\tilde{y}_0 = y(0) = 3$.
   (a) La pendiente de Euler de la x en el punto inicial es:
   $$m_{e,x} = f(t_0, \tilde{x}_0, \tilde{y}_0) = -7.2.$$
   (b) La pendiente de Euler de la y en el punto inicial es:
   $$m_{e,y} = g(t_0, \tilde{x}_0, \tilde{y}_0) = 4.8.$$
   (c) Calculamos la predicción que daría el algoritmo de Euler para la x:
   $$x_e = \tilde{x}_0 + h \cdot m_{e,x} = 18 + 0.25 \cdot (-7.2) = 16.2.$$
   (d) Calculamos la predicción que daría el algoritmo de Euler para la y:
   $$y_e = \tilde{y}_0 + h \cdot m_{e,y} = 3 + 0.25 \cdot 4.8 = 4.2.$$

(e) Calculamos la pendiente de la x en el punto final:

$$m_{f,x} = f(t_0, \tilde{x}_e, \tilde{y}_e) = -14.256.$$

(f) Calculamos la pendiente de la y en el punto final:

$$m_{f,y} = g(t_0, \tilde{x}_e, \tilde{y}_e) = 5.208.$$

(g) Las pendientes medias son, por tanto:

$$m_x = (-7.2 - 14.256)/2 = -10.728,$$
$$m_y = (4.8 + 5.208)/2 = 5.004.$$

(h) Así que, por fin:

$$x_1 = x_0 + h \cdot m_x = 15.318$$
$$y_1 = y_0 + h \cdot m_y = 4.251.$$

(2) **Segundo paso,** $t_1 = 0.25$. Tenemos que $\tilde{x}_1 = 15.318$ y que $\tilde{y}_1 = 4.251$.

(a) Pendientes de Euler (las dos):

$$m_{e,x} = f(t_1, x_1, y_1) = -13.79233$$
$$m_{e,y} = g(t_1, x_1, y_1) = 4.5214.$$

(b) Predicciones de Euler:

$$x_e = x_1 + h \cdot m_{e,x} = 11.87$$
$$y_e = y_1 + h \cdot m_{e,y} = 5.381.$$

(c) Pendientes en el punto final:

$$m_{f,x} = f(t_1, x_e, y_e) = -16.055$$
$$m_{f,y} = g(t_1, x_e, y_e) = 2.013.$$

(d) Pendientes medias:

$$m_x = (m_{e,x} + m_{f,x})/2 = -14.923$$
$$m_y = (m_{e,y} + m_{f,y})/2 = 3.267.$$

(e) Valores de $\tilde{x}_2, \tilde{y}_2$:

$$\tilde{x}_2 = \tilde{x}_1 + h \cdot m_x = 11.587$$
$$\tilde{y}_2 = \tilde{y}_1 + h\dot{m}_y = 5.068.$$

Los valores exactos son $x(0.5) = 11.7$ e $y(0.5) = 5.131$. Se aprecia una *muy buena aproximación* para lo grande que es el paso (0.25).

Pero lo más claro es lo laborioso del cálculo. Hoy día estas cuentas se realizan por ordenador. Durante el programa Apollo, la mayoría de estas operaciones se llevaban a cabo a mano. Quienes lo hacían

son los héroes no reconocidos de la Carrera Espacial (aunque ya se ha hecho alguna película al respecto).

9.2. Representación de una solución. En la Figura 4, se ha representado un ejemplo de solución (usando el algoritmo de Heun) del sistema de Lotka-Volterra de la sección anterior. Obsérvese (esto es quizás lo más importante de este sistema) que ambas poblaciones tienen un comportamiento creciente y decreciente con un desfase entre ellas.

Con algo de esfuerzo se puede verificar que los puntos críticos de la función $x(t)$ se alcanzan cuando la función $y(t)$ vale γ/δ y que los puntos críticos de $y(t)$ se alcanzan cuando la función $x(t)$ vale α/β (¿cómo se puede comprobar esto? es fácil pero requiere una explicación).

Sin embargo, se sabe (desde el punto de vista teórico) que el sistema de Lotka-Volterra *es periódico*. Las soluciones que se han dibujado no lo son (si uno calculara las soluciones para tiempos mucho más largos, vería cómo las funciones se vuelven cada vez más exageradamente "puntiagudas").

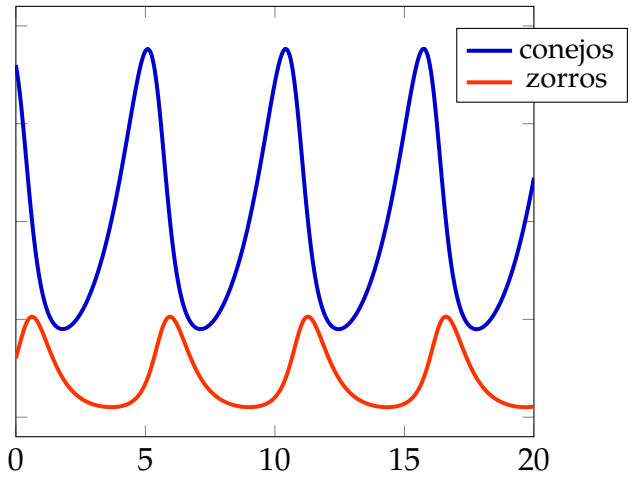

FIGURA 4. Ejemplo de sistema predador-presa. En este sistemas se supone siempre que los valores son "densidades de población".