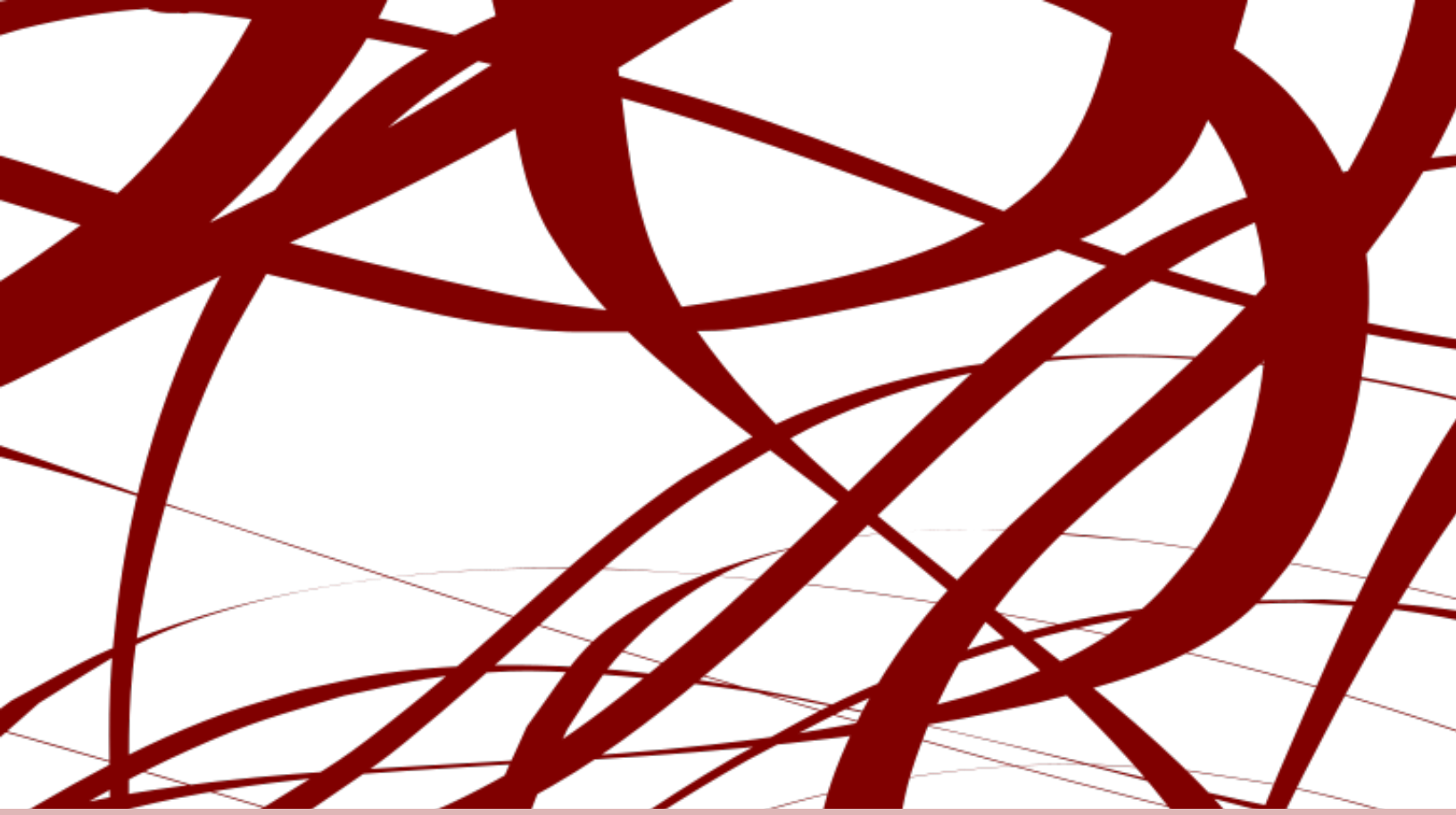
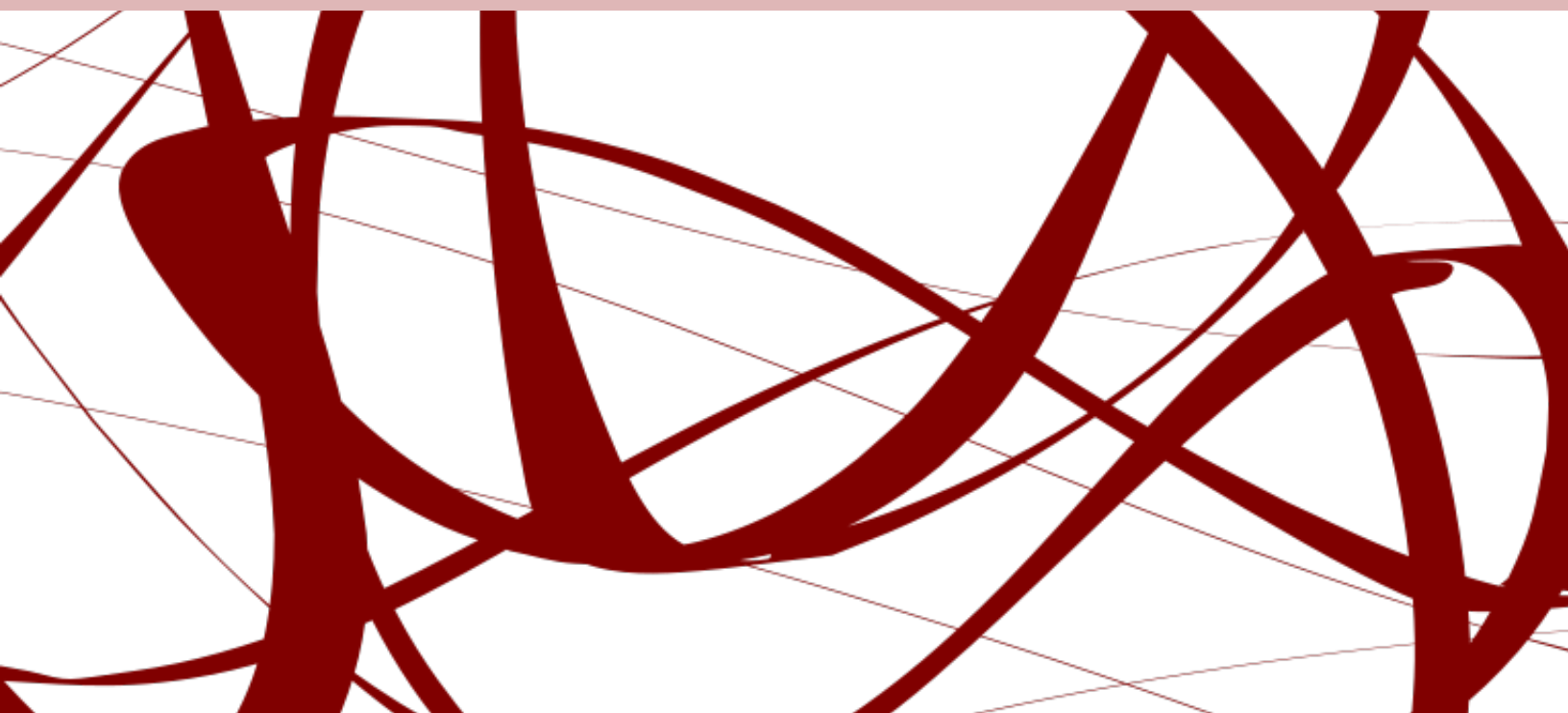


Análisis Numérico

Notas del curso

Dr. Jonathan Montalvo-Urquizo



HTTP://WWW.CIMAT.MX/~JONATHAN.MONTALVO/

Última actualización: 22.5.2017

Contenido

1. Introducción

1.1 Motivación e historia del cómputo

Muchos problemas científicos y tecnológicos que deben ser resueltos actualmente en áreas diversas del conocimiento como los procesos de producción, el entendimiento de los sistemas biológicos, o la exploración de fenómenos astronómicos o meteorológicos tiene un nivel alto de complejidad. La resolución de muchos de estos problemas requiere un proceso detallado de estudio que puede dividirse en las siguientes etapas:

1. Formulación del problema
2. Acotamiento del problema en un área científica
3. Abstracción conceptual
4. Modelo matemático del problema
5. Solución analítica o numérica
6. Interpretación de la solución

La obtención de una solución analítica (matemáticamente) depende no sólo de la existencia de una solución sino también de la posibilidad técnica de calcularla. Además, en muchos de los casos, a pesar de que las soluciones a los problemas existen, es imposible calcularlas.

La matemática numérica se encarga de encontrar soluciones aproximadas a problemas cuya complejidad supera las capacidades de calcular soluciones de manera analítica. Actualmente, el concepto de "Solución Numérica" se refiere a soluciones a problemas matemáticos obtenidas a través de cálculos realizados en una computadora. Este concepto lleva implícita la noción de que las soluciones obtenidas de esta manera

son, por su naturaleza, aproximaciones a las soluciones exactas del problema planteado.

Dentro de las funciones actuales que cumple la matemática numérica dentro del proceso científica de resolución de problemas aplicados son:

- el desarrollo de métodos para la construcción de soluciones numéricas,
- la implementación computacional de los métodos de manera eficiente,
- la selección de los parámetros adecuados para el uso de los métodos implementados,
- el análisis acerca de la calidad de las soluciones obtenidas en términos de nivel de exactitud, tiempo de cómputo y estabilidad de los resultados.

Aunque en la actualidad los métodos numéricos son comúnmente diseñados para su uso en computadoras digitales, las primeras metodologías de la matemática numérica pueden ser atribuídas a las prácticas de cálculo de poblaciones de la antigüedad.

Si solamente consideramos la era de computadoras basadas en algún tipo de maquinaria capaz de realizar cálculos de manera automatizada, algunos dispositivos de cómputo mejor documentados son:

1623 Máquina de cálculos (Rechenuhr) diseñada (y construida?) por Wilhelm Schikarden en Tübingen, Alemania.

1645 Blaise Pascal presenta su máquina de cálculo "Pascaline".

1673 Gottfried Wilhelm Leibniz presenta su máquina de cálculo basada en la hoy llamada rueda/cilindro de Leibniz (Staffelwalze).

1938 Konrad Zuse completa su máquina Z1 (originalmente llamada V1 como abreviatura del nombre Versuchsmodell-1). Ésta es considerada la primer máquina de cálculo que utilizó la aritmética de punto flotante y que utilizaba sistema de numeración binario.

1941 Konrade Zuse presenta la Z3. Por primera vez los casos excepcionales son considerados (NaN, $+\infty$, $-\infty$). Mejora sustancial desde la Z1.

1945 En marzo, Konrad Zuse presenta la Z4, primera computadora programable a través del uso de cintas perforadas similares a la cinta fílmica.

1946 En febrero, J. Presper Eckert y John W. Mauchly presentaron la computadora ENIAC (Electronic Numerical Integrator and Computer). Ésta es conocida como la primer computadora totalmente electrónica.

1951 UNIVAC, construida por J. P. Eckert y J. W. Mauchly y fue la primer computadora disponible para la venta comercial. UNIVAC existió hasta 1981.

En particular, las máquinas diseñadas por Konrad Zuse están bien documentadas a través del portal`http://zuse.zib.de/`, donde se exponen las versiones digitales de

planos, fotografías e incluso simulaciones de reconstrucciones de las máquinas Z1 y Z3.

A partir de la década de los 1950's, el uso de computadoras se incrementó en todas las áreas de actividad humana. En términos de cálculo, las capacidades computacionales empezaron a crecer impulsadas por los desarrollos de procesadores cada vez más potentes. En una necesidad de predecir el crecimiento acelerado de la industria de la computación, algunos pioneros se aventuraron a definir el nivel de crecimiento de las capacidades de cómputo. La predicción hasta ahora más certera es la famosa Ley de Moore[1], postulada en 1965 y que afirma que el número de transistores (y por ende la capacidad de cómputo) se duplica aproximadamente cada 2 años.

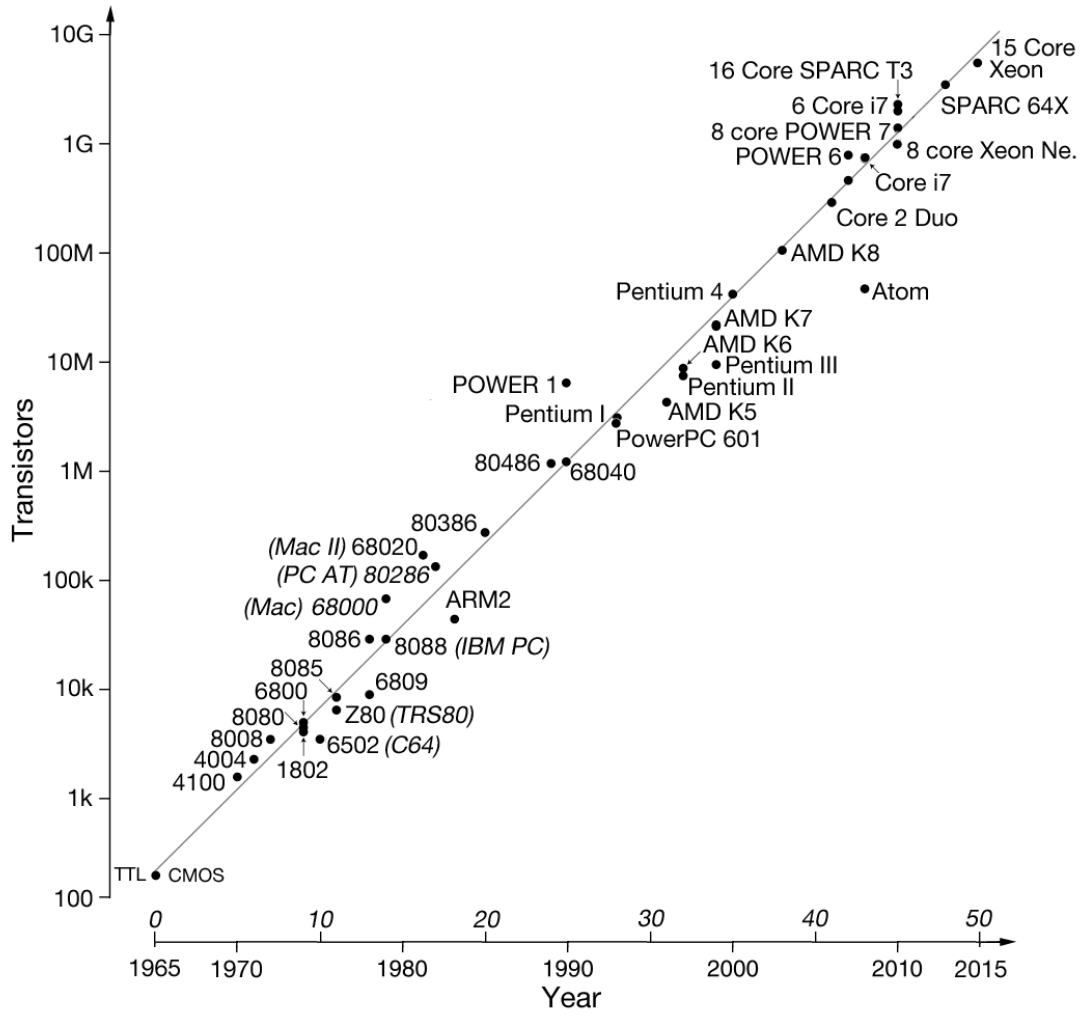


Figura 1.1: Cincuenta años de la Ley de Moore.

Como ya se dijo, la matemática numérica depende en gran medida de la capacidad de cómputo para calcular soluciones a modelos matemáticos, por lo que esta ley asegura

[1] Atribuída a Gondor E. Moore (1929-), cofundador del Intel Corporation.

que la eficiencia con que la matemátia numéric puede resolver problemas se incrementa en un 100 % cada dos años, como puede ser apreciado en la Figura 1.1.

Veamos ahora algunos ejemplos básicos de problemas que pueden ser resueltos mediante el uso de la matemática numérica.

■ **Ejemplo 1.1 — Corte de 3 planos.** Usado por ejemplo en la construcción de techos de dos aguas con cortes laterales

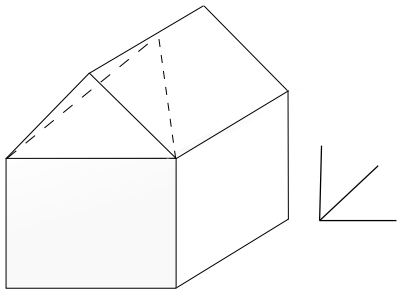


Figura 1.2: Corte de múltiples planos en $\mathbb{R}^3$.

Esto se traduce fácilmente en una tarea de resolver el sistema lineal

$$\begin{aligned} ax \quad\quad\quad +bz &= c, \\ -ax \quad\quad\quad +bz &= d, \\ ex \quad +fy \quad +gz &= h, \end{aligned} \tag{1.1}$$

y por ser pocas ecuaciones puede ser resuelto de manera analítica, pero esto no está automatizado directamente, a menos que se resuelva de manera numérica. Un simulador para resolver este tipo de problemas no podría considerar todas los posibles valores de las constantes $a,\ldots,h$, por lo que la resolución a través de una simulación numérica sería necesaria. ■

■ **Ejemplo 1.2 — Descripción de procesos dinámicos de primer orden.** La ecuación $y(t) = y'(t)$ cuya solución es $y(t) = e^t$ puede usarse como ejemplo de una dinámica temporal para describir la expansión bacteriana o el contagio de una enfermedad. Sin embargo, aunque la solución analítica sea conocida, saber exactamente cuántas bacterias se encuentran presentes en el modelo requiere de la evaluaión de una exponencial, lo cual solamente es asequible si se utiliza una cálculo numérico utilizando el tiempo deseado t. Evidentemente, este tipo de evaluaciones son imposibles de realizar utilizando cálculos hechos a mano o por medio de tablas precalculadas. ■

■ **Ejemplo 1.3 — Descripción de procesos dinámicos de segundo orden.** La ecuación $y(t) = -y''(t)$ es resuelta (entre otras) por la función $y(t) = \cos(t)$ y es usada para describir el movimiento ideal de un sistema formado por una masa colgada de un resorte (sin considerar los efectos de la gravedad). Esta función puede ser calculada

utilizando tablas preestablecidas hace muchos años, sin embargo, el nivel de precisión que estas tablas tienen no supera las 6 u 8 cifras decimales, por lo que la precisión del cálculo estaría limitada.

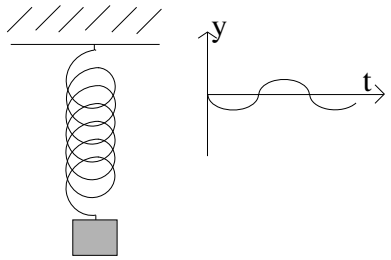


Figura 1.3: Oscilador armónico.

Por otro lado, los sistemas dinámicos de muchos problemas de interés, como son las enfermedades poblacionales están descritas por modelos dinámicos mucho más complejos (Dengue, Ébola, Zika, etc.). Otro ejemplo modelado por este tipo de osciladores son los sistemas de cuerpos articulados como el sistema de amortiguación de un automóvil, un brazo robótico, o incluso los planetarios. Todos estos sistemas son evidentemente mucho más complejos y requieren de una alta precisión para ser calculados. Por tanto, las ecuaciones diferenciales correspondientes a este tipo de problmas no pueden ser resueltas de manera analítica. ■

■ **Ejemplo 1.4 — Encontrar ceros de funciones.** El método más utilizado para funciones $f : \mathbb{R} \to \mathbb{R}$ consiste en buscar utilizando la información de la derivada para generar, a partir de una aproximación inicial x_0 una sucesión de parejas $(x_0, f(x_0)), (x_1, f(x_1)), \ldots$ a través del uso de la línea tangente a la curva, es decir

$$y(x) = f(x_0) + (x - x_0) f'(x_0), \tag{1.2}$$

cuyo corte con la línea del cero es

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{1.3}$$

En general, al construir las aproximaciones de la forma

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{1.4}$$

se tendría un proceso iterativo como el ilustrado en la Figura 1.4 en la que repetir el mismo procedimiento converge al punto donde la función vale cero. Al construir este proceso iterativo se espera que $f(x_i) \to 0$ y por lo tanto $|x_{i+1} - x_i| \to 0$.

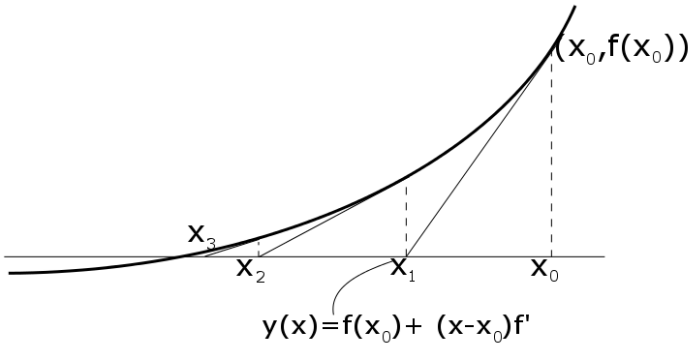


Figura 1.4: Encontrando el cero de una función a través de aproximaciones lineales.

Esto puede calcularse analíticamente algunas veces, por ejemplo cuando la función f es un polinomio o alguna otra función bien conocida analíticamente. Sin embargo, el caso general requerirá de un proceso iterativo cuya convergencia es desconocida y por lo tanto no se puede saber a priori cuántos cálculos requerirá. En este tipo de problemas, la única opción es considerar una solución numérica en la que puedan calcularse muchas iteraciones del proceso iterativo. ■

■ **Ejemplo 1.5 — Integración de funciones.** Es otro proceso que puede ser sencillamente aproximado de manera analítica a través de una partición del dominio $[a,b) = \cup_{i=1}^{n}[x_i, x_{i+1})$ y el uso de la idea original de la integral de Riemann, donde se consideraba un conjunto de barras de altura igual a la evaluación de la función y su respectiva área, tal como se ilustra en la Figura 1.5. Si se aproxima la integral de esta

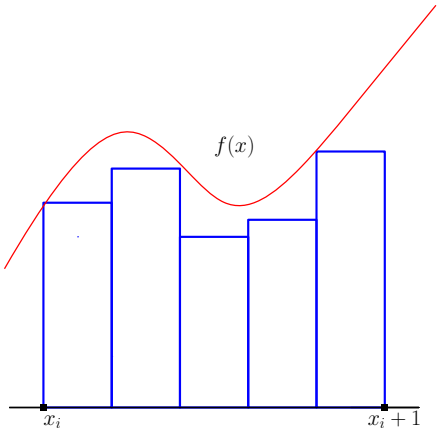


Figura 1.5: Aproximación de la integral de $f(x)$ en el intervalo $[x_i, x_{i+1}]$.

forma, pueden calcularse analíticamente una buena cantidad de áreas para anchos de las

barras relativamente pequeños. Otra opción de mejores resultados es considerar la regla del trapecio dada por

$$\int_{x_i}^{x_i+1} f(x) \approx \frac{f(x_i)+f(x_{i+1})}{2}(x_{i+1}-x_i) \tag{1.5}$$

que en general da buenas aproximaciones al valor de la integral (como veremos más adelante en el curso).

Sin embargo, el proceso de tomar barras cada vez más pequeñas solamente podrá ser realizado si se considera una gran cantidad de operaciones aritméticas, pues en cada barra considerada se tendrán que realizar evaluaciones de $f(x_i)$, las cuales pueden llegar a ser muy complicadas de caluar de forma analítica.

Existen ejemplos de aplicaciones en todas las áreas de la matemática donde el tratamiento de este tipo de problemas es necesario debido a la complejidad o a las dimensiones del espacio (euclidiano o no) en el que el modelo sea planteado. En particular, muchos métodos de solución numérica para resolver ecuaciones diferenciales están basados en calcular un gran número de integrales en varias dimensiones. Piense por ejemplo en el cálculo del volumen de almacenamiento de un tanque de gasolina moderno (3D, geometría rebuscada, asimetrías, etc.) ∎

1.2 Aritmética de punto flotante

El uso de cálculos numéricos tiene limitantes debido a que los números representables computacionalmente son finitos y, por muchos que parezcan, nunca será lo mismo que tener el continuo de números en $\mathbb{R}$.

Definición 1.1 Los números computacionales son llamados "de punto flotante" y forman, junto con el sistema aritmético para realizar operaciones básicas entre ellos, lo que se denomina "Aritmética de punto flotante" (APF).

Un número de punto flotante en base $b \in \mathbb{N} \setminus \{1\}$ es un número real de la forma

$$x = \pm m \cdot b^{\pm e} \tag{1.6}$$

donde los diferentes factores son

- El signo de x definido como $+$ ó $-$
- La mantisa, que es representable en términos de la base b y r números adicionales que forman la expresión

$$m = m_1 b^{-1} + m_2 b^{-2} + \cdots + m_r b^{-r}$$

- El exponente representable con s números

$$e = e_{s-1} b^{s-1} + \cdots + e_0 b^0$$

Tanto para la mantisa como el exponente, los números que actuan como coeficientes de la base cumplen con $m_i, e_i \in \{0, \cdots, b-1\}$.

Además, con el fin de normalizar la aritmética, se requiere que $m_1 \neq 0$, lo cual evita que los números puedan tener diversas representaciones, haciendo posible una única representación para cada número de la APF. Note que esta representación es cercana (aunque no exactamente igual) a la notación científica y permite considerar números de muy diversos tamaños como pudiera ser

- La velocidad de la luz $c = 0{,}299792458 \cdot 10^9 m/s$
- La masa de un electrón $M_o = 0{,}910938 \cdot 10^{-29} g$

Para guardar un número computacional hacen falta:

(a) r cifras para la mantisa más un signo;
(b) s cifras para el exponente más un signo;
(c) espacio para guardar los siguientes números:

- Infinito (positivo y negativo), que será cualquier número mayor al más grande representable $\rightarrow$ Región de *Overflow*. El infinito se representa usando $e = \text{máx}(posibles)$, $m = 0$
- NaN (not a number) que se obtiene al realizar operaciones no definidas en la aritmética. Este número se representa usando $e = \text{máx}(posibles)$, $m > 0$
- Números en la Región de *Underflow*, definida en $(0, \text{mín}_{x \neq 0}(|x|))$ y se representa por $e = \text{mín}(posibles)$, $m > 0$
- Cero, definido como el número en el centro de todos los reales y representado por $e = 0$, $m = 0$

Los números se guardan en formato

$$(\pm)[m_1 m_2 \cdots m_r](\pm)[e_{s-1} e_{s-2} \cdots e_0] \tag{1.7}$$

donde solamente es necesario guardar los coeficientes m_i y e_j. Debido a razones técnicas de construcción, las computadoras utilizan numeración binaria, es decir que las cifras pertenecen a $\{0,1\}$ y la base que se toma es $b = 2$. De esta forma, tiene sentido también declarar que $m_1 = 1$.

Otro recurso utilizado comúnmente es la representación del exponente como una traslación de un orden mayor, es decir en lugar de usar exponentes con signos y "desperdiciar" una cifra para el signo se puede usar un exponente siempre positivo y trasladarlo como

$$(\pm)[e_{s-1} e_{s-2} \cdots e_0] \longrightarrow [e_s e_{s-1} \cdots e_0] - T \tag{1.8}$$

donde T es el factor de traslación. Veamos porqué se hace esto en utilizando el ejemplo binario en que $s = 4$.

■ **Ejemplo 1.6 — APF con base $b = 2$, $s = 4$.** Con base en los valores de la base y el número de coeficientes del exponente se tiene que:

- En la primera representación $(\pm)e_3e_2e_1e_0$:
  Se tendrán exponentes mínimo absoluto de 0 (usando $e = \pm(0\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0)$) y máximo en valor absoluto de $e = 8+4+2+1 = 15$.
  Pero como el máximo exponente deberá ser usado para guardar NaN y los Infinitos, sólo los exponentes en $\{-15,-14,\cdots,0,1,2,\cdots,14\}$ serán de utilidad real, introduciendo una asimetría exponencial.
- En la segunda representación $e_4e_3e_2e_1e_0$:
  Se tendrán exponentes con mínimo absoluto de 0 y máximo absoluto de $e = 2^4+2^3+2^2+2^1+2^0 = 31$. Trasladando $\{0,\cdots,31\}$ con $T = 15$ se obtiene $\{-15,\cdots,0,1,\cdots,16\}$, lo que permite usar el exponente 16 para NaN/Infinito y mantener la simetría exponencial.

■

■ **Ejemplo 1.7 — Sistema binario.** Veamos el ejemplo concreto de un sistema binario, con límite para guardar números en 1 Byte=8 bits.

De los 8 lugares podemos usar 1 lugar para el signo, 4 para la mantisa y 3 para el exponente. Los números $m_i, e_i \in \{0,1\}$, $m_1 = 1$ tal que

$$m = 2^{-1} + m_2 2^{-2} + m_3 2^{-3} + m_4 2^{-4} + m_5 2^{-5} \quad \text{(4 cifras)} \tag{1.9}$$

$$e = e_0 2^2 + e_1 2^1 + e_2 2^0 \quad \text{(3 cifras)} \tag{1.10}$$

Trasladando e con $T = 3$ se obtiene $e \in \{-3,-2-,1,0,1,2,3\}$. De esta manera, el mayor número en el sistema, será dado por la mantisa con $m = \underline{1}1111$ y el exponente 3, es decir

$$\begin{aligned} x_{max} &= (2^{-1}+2^{-2}+2^{-3}+2^{-4}+2^{-5})\cdot 2^3 \\ &= 2^2+2^1+2^0+2^{-1}+2^{-2} \\ &= 7+\frac{1}{2}+\frac{1}{4} \\ &= 7\frac{3}{4}. \end{aligned} \tag{1.11}$$

El número positivo más pequeño es dado por la menor mantisa $m = 10000$ y el menor exponente posible (-3), y es entonces

$$\begin{aligned} x_{min} &= 2^{-1}\cdot 2^{-3} = 2^{-4} \\ &= \frac{1}{16} \end{aligned} \tag{1.12}$$

Esto quiere decir que la región de Underflow es

$$\left(-\frac{1}{16},0\right) \cup \left(0,\frac{1}{16}\right) \tag{1.13}$$

y la de Overflow es

$$\left(-\infty,-7\frac{3}{4}\right)\cup\left(7\frac{3}{4},\infty\right). \tag{1.14}$$

En este sistema hay un total de 16 posibles mantisas y 7 exponentes, es decir que sólo pueden representarse 112 números positivos y 112 negativos. ∎

∎ **Ejemplo 1.8 — IEEE-754.** Esta APF es definida por el Institute of Electrical and Electronic Engineers y es la norma más utilizada para los cálculos en los sistemas de hoy en día. Es un sistema binario con mantisa de 52 cifras y exponente de 11 cifras (sin signo). El bit restante de los 64 que usa es el signo del número. Trasladando los exponentes con $T = 1023$ se obtiene un rango de exponentes $e \in \{-1023,\cdots,1023\}$.

Nuevamente las cifras $m_i, c_i \in \{0,1\}$ y análogamente

$$m = 2^{-1} + m_2 2^{-2} + m_3 2^{-3} + \cdots + m_{53} 2^{-53} \tag{1.15}$$

$$e = e_0 2^1 0 + e_1 2^9 + \cdots + e_{10} 2^0 \tag{1.16}$$

El mayor número representable es entonces dado por el mayor exponente (1023) y la mantisa

$$m = \underline{1}\underbrace{11\cdots1}_{52 \text{ veces}} \tag{1.17}$$

$$\begin{aligned} \Rightarrow\ x_{max} &= (2^{-1}+2^{-2}+\cdots+2^{-53})\cdot 2^{1023} \\ &= 2^{1022}+2^{1021}+\cdots+2^{970} \\ &\approx 1{,}8\times 10^{308} \end{aligned} \tag{1.18}$$

El menor número positivo tiene el menor exponente (-1023) y la mantisa
$m = 1\underbrace{00\cdots0}_{52veces} \Rightarrow 2^{-1}\cdot 2^{-1023} = 2^{-1024} \approx 5{,}556\times 10^{-309}$

Se pueden guardar 2^{52} mantisas diferentes y 2046 ($\approx 2^{11}$) exponentes, es decir que el sistema es capaz de representar un total de 2^{63} números positivos y un total de 2^{64} números (recuerde que se usan 64 bits). ∎

Redondeo

Para todos los números computacionales, las operaciones entre ellos pueden resultar en números que no pertenecen al sistema de números representables. En estos casos es necesario realizar un redondeo en el que se pide la condición natural

$$|x - rd(x)| = \min_{y\in A} |x-y| \quad \forall x \in D$$

donde $D = [x_{min}, x_{negmaw}] \cup \{0\} \cup [x_{posmin}, x_{max}]$ y A es el conjunto finito de números representables.

Observe que la idea coincide con el redondeo justo hacia arriba y abajo al utilizar un sistema de base decimal.

Para el sistema **IEEE-754** esto se hace simplemente como

$$rd(x) = sign(x) \cdot \begin{cases} 0 \cdot m_1 \dots m_{53} \cdot 2^e & , m_{54} = 0 \\ (0 \cdot m_1 \dots m_{53} + 2^{-53}) \cdot 2^e & , m_{54} = 1 \end{cases} \tag{1.19}$$

Por supuesto que se introducirán errores de redondeo, las cuales se pueden acotar en términos absolutos como

$$|x - rd(x)| \leq \frac{1}{2} b^{-r} b^e \tag{1.20}$$

que no es muy útil dado su dependencia del exponente e, por lo que comúnmente se revisa el error relativo de redondeo dado por

$$\left\| \frac{x - rd(x)}{x} \right\| \leq \frac{1}{2} \frac{b^{-r} b^e}{|m| b^e} \leq \frac{1}{2} b^{-r+1} \tag{1.21}$$

para todo $x \in D, x \neq 0$.

Este error relativo está entonces acotado por la precisión de cómputo (machine precision) $eps := \frac{1}{2} b^{-r+1}$ para la cual vale siempre

$$rd(x) = x(1+\varepsilon) \text{ para algún } |\varepsilon| \leq eps \tag{1.22}$$

En el sistema **IEEE-754** esta cota es

$$eps = \frac{1}{2} 2^{-53+1} = 2^{-53} \approx 10^{-16} \tag{1.23}$$

Operaciones aritméticas básicas

Las operaciones en los números reales $* \in \{+, -, \cdot, /\}$ son reemplazadas por operaciones computacionales $\otimes \in \{\oplus, \ominus, \odot, \oslash\}$. Estas operaciones deben generar resultados dentro de la APF (los números computacionalmente representables). Algunas consecuencias es que la ley distributiva y asociativa de las operaciones $\oplus$ y $\odot$ se cumplen solamente de manera aproximada. En general para $x, y, z \in APF$

$$((x \oplus y) \oplus z) \neq x \oplus (y \oplus z) \tag{1.24}$$

$$(x \oplus y) \odot z \neq (x \odot z) \oplus (y \odot z) \tag{1.25}$$

Problemas Numéricos en este curso

Un problema numérico puede escribirse como el problema de encontrar una cantidad finita de valores $y_i \quad (i = 1, \dots, n)$ mediante el uso de otra cantidad finita de valores conocidos $x_j \quad (j = 1, \dots, m)$ a través de un funcional $y_i = f(x_1, \dots, x_m)$.

En este curso nos ocuparemos principalmente de problemas en los que x_j y y_i son números reales. Problemas en $\mathbb{C}$ pueden ser tratados muy análogamente y quedan fuera de los contenidos de este curso.

Errores absoluto y relativo

Definición 1.2 — Errores absoluto y relativo. Mediante la existencia de errores en los datos conocidos (p.ej. a través de redondeo) $x_j + \Delta x_j$ se producen errores en los valores calculados $y_i + \Delta y_i$. El error absoluto es definido como $|\Delta y_i|$ mientras que para $y_i \neq 0$ el error relativo es definido como $\left\| \frac{\Delta y_i}{y_i} \right\|$

Por la naturaleza de la definición de los errores absoluto y relativo, los errores de interés en el análisis numérico son los de tipo relativo.

Veamos ahora en análisis diferencial al considerar pequeños errores en los datos , i.e. $|\Delta x_j| \ll |x_j|$. Suponiendo que para el cálculo de los valores y_i se tienen funcionales f_i tal que $y_i = f_i(x_1, \ldots, x_m)$ donde f_i es parcialmente diferenciable con respecto a $x_j \quad (\forall i, \forall j)$ entonces de acuerdo al Teorema de Taylor y usando $x = (x_1, \ldots, x_m)$

$$\Delta y_i = f_i(x + \Delta x) - f_i(x) = \sum_{j=1}^{m} \frac{\partial f_i}{\partial x_j}(x) \Delta x_j + \mathscr{R}_i^f(x, \Delta x) \tag{1.26}$$

$i = 1, \ldots, m$, donde $\mathscr{R}_i^f(x, \Delta x)$ denota el residual.

Además puede demostrarse que $\mathscr{R}_i^f(x, \Delta x)$ decae más rápido hacia 0 que el valor $|\Delta x| = \text{máx}_{j=1,\ldots,m} |\Delta x_j|$ dado que se está suponiendo que todos los Δx_i son pequeños en valor absoluto.

Esto se denota normalmente como $\mathscr{R}_i^f(x, \Delta x) = \mathscr{o}(|\Delta x|)$ usando la conocida Notación de Landau. La notación de Londau[2] para describir procesos asintóticos cuantitativos utiliza los símbolos $\mathscr{O}(\cdot)$, $\mathscr{o}(\cdot)$ como sigue:

Definición 1.3 — Orden de una función. Sean $g(t)$ y $h(t)$ funciones para $t \in R_+$, entonces la notación $g(t) = \mathscr{O}(h(t)), (t \to 0)$ significa que para valores pequeños de $t \in [0, t_0]$, existe $c \geq 0$ tal que se cumple $|g(t)| \leq c \cdot |h(t)|$. De modo similar, si $g(t) = \mathscr{o}(h(t)) \quad (t \to 0)$ para valores de $t \in [0, t_0]$ significa que existe una función $c(t)_{t \to 0} \to 0$ tal que $|g(t)| \leq c(t) \cdot |h(t)|$

■ **Ejemplo 1.9** Sea $g(t) \in C^2$ con expansión de Taylor

$$g(t + \Delta t) = g(t) + \Delta t g'(t) + \frac{\Delta t^2}{2} g''(\tau), \tau \in (t, t + \Delta t)$$

entonces

$$\frac{g(t + \Delta t) - g(t)}{\Delta t} = g'(t) + \mathscr{O}(\Delta t)$$

■

[2] Debido a Edmund Geary Hermann Londau (1877 – 1938). Profesor en Götingen de nacionalidad alemana. Obligado a pensionarse en 1934 por ser de origen judío. Trabajos en Teoría de números. Teoría de funciones complejas, entre otras.

Veamos ahora que pasa en una *APF* cuando los datos contienen pequeños errores y se realizan operaciones con ellos.

Definición 1.4 — Número de condición de la suma. La función de suma $f : \mathbb{R}^2 \to \mathbb{R}, y : f(x_1,x_2) = x_1 + x_2$ puede analizarse a tráves de su "número de condición", definido como

$$k_1 = \frac{\partial f}{\partial x_1}\frac{x_1}{f} = 1 \cdot \frac{x_1}{x_1+x_2} = \frac{1}{1+\frac{x_2}{x_1}}$$
$$k_2 = \frac{\partial f}{\partial x_2}\frac{x_2}{f} = \frac{1}{1+\frac{x_1}{x_2}}$$

Puede observarse fácilmente que los números k_1 y k_2 presentan problemas cuando $1 + \frac{x_i}{x_j} \approx 0$ por lo que la suma es problemática cuando $x_1 \approx -x_2$, es decir cuando ambos sumandos son aproximadamente iguales pero de diferente signo.

Otro problema conocido al sumar números es el siguiente:

Definición 1.5 — Cancelación catastrófica. Se llama "Cancelación catastrófica" a la pérdida de cifras en un número de punto flotante a través de realizar una resta de dos números del mismo signo. Esto es un problema cuando alguno de los números no es exactamente representable en la aritmética computacional.

■ **Ejemplo 1.10 — Cancelación catastrófica en aritmética deicmal.** Considere la *APF* con base decimal y 4 dígitos significativos en contraste con la aritmetica exacta en $\mathbb{R}$

Aritmética en $\mathbb{R}$	APF ($b = 10, 4$ dígitos)
$x = 0{,}51232x10^1$	$fl(x) = 0{,}5123x10^1$
$y = 0{,}11230x10^1$	$fl(y) = 0{,}1123x10^1$
$(x-y) = 0{,}40002x10^1$	$fl(z) = -0{,}4x10^1$
$z = -0{,}4x10^1$	
$(x-y)-z = 0{,}40002x10^1 - 0{,}4x10^1$	$fl(x-y) = 0{,}4x10^1$
$= ,2x10^-4$	$fl((x-y)-z) = 0{,}0x10^1$

■

En el caso de la operación de multiplicación, el "número de condición" para el cálculo de $y = f(x_1,x_2) = x_1 \cdot x_2$ es

$$k_1 = \frac{\partial f}{\partial x_1}\frac{x_1}{f} = x_2 \cdot \frac{x_1}{x_1 \cdot x_2} = 1$$
$$k_2 = 1$$

por lo que al ser siempre bien condicionada, la multiplicación no representa problemas en función de qué valores tengan x_1, x_2.

Nota. Al realizar múltiplos cálculos computacionales es inevitable la aparición de pequeños errores debido a redondeo o errores al evaluar funciones aritméticas básicas.

Además, estos errores se acumulan a lo largo del proceso de cálculo. El interés de Análisis Numérico radica en generar algoritmos en los que esta acumulación de errores no domine a la buena aproximación numérica.

Un ejemplo del ahorro en errores al usar menos operaciones puede verse en la evaluación de un polinomio cuadrático, es decir en el cálculo de valores $y = a_0 + a_1 x + a_2 x^2$.

(a) Evaluado directamente, este valor requiere 2 sumas y 3 multiplicaciones, es decir 5 operaciones de punto flotante.
(b) Evaluado en la forma equivalente $y = a_0 + x(a_1 + a_2 x)$ requiere 2 sumas y 2 multiplicaciones $\Rightarrow$ 4 operaciones dep f .

El mismo ejemplo visto para polinomios de grado 3 lleva en evaluación directa a 3 sumas y 6 multiplicaciones, es decir a 9 operaciones de punto flotante en el estilo de evaluación directa. Para grado 4 esto es 4 sumas y 10 multiplicaciones $\Rightarrow$ 14 operaciones de punto flotante. Usando la evaluación anidada $y = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + xa_n)\ldots))$ esto lleva en grado 3 y 4 respectivamente a

$$\begin{aligned} y &= a_0 + x(a_1 + x(a_2 + xa_3)) && \Rightarrow 3 \text{ sumas, } 3 \text{ multiplicaciones} \\ y &= a_0 + x(a_1 + x(a_2 + x(a_3 + xa_4))) && \Rightarrow 4 \text{ sumas, } 4 \text{ multiplicaciones} \end{aligned}$$

y en general, esta forma anidada requiere n sumas y n multiplicaciones para evaluar un polinomio de grado n, es decir $2n$ operaciones de punto flotante. A la evaluación polinomial en esta forma se le conoce como el **Algoritmo de Hörner**[3]

1.3 Ejercicios

Ejercicio 1.1. Software y programación (0 Puntos)
Practique los comandos básicos de MATLAB con el fin de poder introducir vectores, matrices, manipular elementos de los mismos, graficar información en el plano, etc. Existe una basta oferta de tutoriales y recursos en línea para esto. Si lo desea, realice lo equivalente en otro software o leguaje (R, java, python, etc.) ■

Ejercicio 1.2. Sistema IEEE-754 (25 Puntos)
Para el sistema IEEE-754 visto en clase, determine el valor de los siguientes números:
1. el penúltimo número representable, y
2. el segundo número positivo más pequeño representable.

¿Qué consecuencias buenas o malas habrá en la diferencia de espaciamiento entre los números del sistema? ■

[3]Debido a William George Hörner (1786 – 1837). Matemático irlandés. Otras fuentes atribuyen este algoritmo a fuentes mucho mas antiguos como Zhu Shije en China del siglo XIII.

Ejercicio 1.3. Leyes asociativa y distributiva (35 Puntos)
Muestre un ejemplo en el que las leyes asociativa y distributiva no se cumplen, acorde a las expresiones (1.24) y (1.25).
Encuentre además las cotas para los números de punto flotante y tales que se cumpla $x \oplus y = x$. ■

Ejercicio 1.4. Generando una APF (20 Puntos)
Invente su propio sistema de números de punto flotante utilizando base binaria y espacio de 2 Bytes (16 bits). Defina una longitud de mantisa y encuentre los números positivos de mayor y menor tamaño para este sistema. ¿Cuántos números ditintos puede representar su sistema? ■

Ejercicio 1.5. Sistema decimal (20 Puntos)
Escriba ahora un sistema decimal ($b = 10$) con una mantisa de 4 cifras y un exponente de 3 cifras. Esto aunado al signo en la representación da un total de 8 'trozos' de información a ser guardados para los elementos de la aritmética. De esta manera, la construcción es más intuitiva que en los sistemas binarios. ¿Por qué cree entonces que no se usa el sistema decimal en los sistemas de cómputo? ■

Ejercicio 1.6. Número de condición (20 Puntos)
Use nuevamente el número de condición $k_i = \frac{\partial f}{\partial x_i}\frac{x_i}{f}$ para mostrar que la división x_1/x_2 está bien condicionada para $i = 1, 2$. ■

Ejercicio 1.7. Precisión de una computadora (20 Puntos)
El nivel de precisión de una computadora está definido como el número

$$\text{eps} = \min_{x \in D,\, x \neq 0} \left| \frac{\text{rd}(x) - x}{x} \right|.$$

en donde $D = [x_{min}, x_{negmax}] \cup \{0\} \cup [x_{posmin}, x_{max}]$ y $\text{rd}(\cdot)$ denota la función de redondeo. ¿Cómo cree que podría determinarse esta constante experimentalmente? Defina un algoritmo para ello, impleméntelo y determine así el número eps de su computadora. ■

Ejercicio 1.8. Aproximación de Taylor (30 Puntos)
Considere la aproximación a la función exponencial dada por la suma de Taylor

$$T_n = \sum_{k=0}^{n} \frac{x^k}{k!}.$$

Escriba un pequeño programa que calcule esta suma utilizando una cantidad diferente de términos en la aproximación para $n \in \{1,2,3,\ldots,20\}$ y considere los cálculos para valores de $x \in \{-10,-1,1,10\}$.

(a) Compare en una tabla la calidad de los resultados con el resultado de la función 'exp(x)' de MATLAB para los 20 niveles de aproximación.

(b) Explique los malos resultados para valores negativos de x y modifique su algoritmo de cálculo para que la calidad de los resultados no dependa del signo de x. Repita la comparación del inciso (a) con el algoritmo modificado.

■

Ejercicio 1.9. Evaluación de polinomios (30 Puntos)

Considere un polinomio expresado en las dos formas equivalentes

$$p(x) = a_0 + xa_1 + a_2x^2 + a_3x^3 + \cdots + a_nx^n$$

$$p(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n))$$

1. Suponga que los coeficientes a_i estan dados y determine el número de operaciones de punto flotante para evaluar ambas representaciones del polinomio en un punto x.
2. Defina 16 números enteros para ser usados como coeficientes de un polinomio de grado 15. Implemente ambas representaciones de la evaluación y realice un número grande de evauaciones para graficar su polinomio para $x \in [-2,2]$. Observe y comente sobre la cantidad de evaluaciones necesarias para que las diferencias se hagan notar en los tiempos de cálculo al usar una u otra representación.
   Algunas funciones que pueden ser útiles: `linspace`, `tic`, `toc`, `plot`, `for`, `function`.

■

2. Interpolación y Splines

Un problema fundamental en la práctica es la representación numérica de objetos matemáticos simples como las funciones reales, o bien de evaluaciones puntuales de éstas. Es evidente que muchos de los modelos que se utilizan para representar fenómenos reales se basan en funciones que, en su forma más simple, corresponden a funciones en los números reales. En este sentido, hay dos puntos importantes a considerar:

(a) Una función $f(x)$ solo puede conocerse en un número finito de puntos $x_0, \cdots, x_n$ y deberá ser reconstruida usando solamente esta información. Un ejemplo de reconstrucción es la representación gráfica de la función, efectuada comúnmente a traves de conocer algunos puntos de la función y conectarlos con una línea para lograr el símil gráfico de tener una función graficada.

(b) Una función $f(x)$ deberá ser representada numéricamente de modo que la evaluación en un valor dado x sea fácil de calcular en cualquier momento. Esto no es nada sencillo si la función es complicada, contiene múltiples funciones trigonométricas, etc. pues entre más variaciones existan en la función de interés, será mas difífcil calcular las evaluaciones en un punto dado.

En ambos casos se tendrá una dependencia funcional $y = f(x)$ con una cantidad de grados de libertad que puede llegar a ser muy grande e incluso ser infinita. Para esto, las funciones son comúnmente aproximadas a tráves de familias o clases de funciones mejor conocidas, como pueden ser:

- Polinomios $p(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Funciones racionales $r(x) = \dfrac{a_0 + a_1 x + \cdots + a_n x^n}{b_0 + b_1 x + \cdots + b_m x^m}$

- Polinomios trigonométricas $t(x) = \frac{1}{2}a_0 + \sum_{k=1}^{n} [a_k \cos(kx) + b_k \sin(bx)]$
- Sumas exponenciales $e(x) = \sum_{k=1}^{n} a_k \exp(b_k x)$

Por razones obvias, habrá una diferencia al aproximar una función utilizando estas clases de funciones. Además existe una diferencia terminológica cuando la aproximación consiste en mera "cercanía" o cuando se tienen puntos en los que la función original y la función proveniente de las clases de funciones conocidas comparten valores en ciertos puntos del dominio.

Definición 2.1 — Interpolación y Aproximación. Sea P una clase de funciones como las antes mencionadas. La asociación de un elemento $g \in P$ a la función f a través de una relación puntual en un número determinado de puntos

$$g(x_i) = y_i = f(x_i) \quad i = 1, \ldots, n$$

se denomina Interpolación. Si la relación de cercanía entre g y f esta dada en un sentido de "mejor" representación como pudiera ser:

$$\max_{a \leq x \leq b} |f(x) - g(x)| \text{ es mínimo para } g \in P,$$

o bien

$$\left(\int_a^b |f(x) - g(x)| dx \right)^{\frac{1}{2}} \text{ es mínimo para } g \in P$$

entonces se habla de Aproximación. Es evidente que la interpolación es un tipo de aproximación en la que la función de representación g satisface que

$$\max_{i=1,\ldots,n} |f(x_i) - g(x_i)| \text{ es mínimo para } g \in P.$$

2.1 Interpolación Polinomial

En esta sección consideraremos a P_{n-1} como el espacio vectorial de los polinomios reales de grado menor o igual a $n-1$

$$P_{n-1} = \{ p(x) = a_1 + a_2 x + \ldots + a_n x^{n-1} | a_i \in R, i = 1, \ldots, n \} \tag{2.1}$$

para considerar el problema de interpolación de Lagrange[1] como sigue:

[1] Debido a Joseph Louis de Lagrange (1736 – 1813). Matemático francés que fue director de la Mathematische Klasse der Berliner Akademie y después se convitió en Profesor en París. Sus cotribuciones incluyen obras fundamentales en Cálculo Varacional, Mecánica, Mecánica Celeste, entre otras áreas.

Definición 2.2 — Problema de interpolación de Lagrange. El problema de interpolación de Lagrange consiste en determinar un polinomio $p \in P_{n-1}$ tal que dados n pares de puntos $(x_1, y_1), \ldots, (x_n, y_n)$ con $y_i = f(x_i)$ para una función $f : R \to R$ se cumpla que $p(x_i) = y_i$ para todo $i = 1, \ldots, n$.

Teorema 2.1 — Solución del problema de interpolación de Lagrange. El problema de interpolación de Lagrange tiene solución y su solución es única.

Demostración (draft)

- Unicidad: Considere 2 soluciones al problema de interpolación p_1, p_2 y construya $p = p_1 - p_2$. El nuevo polinomio satisface que $p(x_i) = 0, \quad i = 1, \ldots, n$, es decir que tiene n raíces, además de que $p \in P_{n-1}$. Por conseuencia, deberá tenerse que $p(x) = 0 \quad \forall x, p = \overline{0} \in P_{n-1}$.
- Existencia Considere el polinomio genérico $p \in P_{n-1}$ expresado en la base de monomios $\{1, x, x^2, \ldots, x^{n-1}\}$ y construya el sistema lineal correspondiente a evaluar la ecuación $a_1 + a_2 x + \ldots + a_n x^{n-1} = b$ para cada par (x_i, y_i) a interpolar. Si los x_i's son distintos se obtendrá un sistema lineal con soluciones únicas.

Como es bien sabido, pueden construirse diferentes bases para el espacio vectorial de los polinomios. De acuerdo a la construcción tomada por Lagrange para la interpolación polinomial, la base de los polinomios a tomar es la formada por los polinomios en P_{n-1} de la forma

$$L_i^{(n)}(x) = \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \tag{2.2}$$

que cuentan con la propiedad de que $L_i^{(n)}(x) = 1$ en el caso de que $x = x_i$ y son iguales a cero en el resto de los nodos. De hecho, esta propiedad es fundamental para demostrar que el conjunto $\left\{ L_i^{(n)}, \quad i = 1, \ldots, n \right\}$ es una base de P_{n-1}.

Definición 2.3 — Interpolante de Lagrange. Dados los n pares $(x_1, y_1), \ldots, (x_n, y_n)$ el polinomio de la forma

$$p(x) := \sum_{i=1}^{n} y_i L_i^{(n)}(x) \quad \in P_{n-1} \tag{2.3}$$

cumple con la propiedad de que $p(x_i) = y_i$ y es llamado Interpolante de Lagrange del conjunto de (x_i, y_i)'s.

Este es un buen ejemplo para fabricar un programa en el que diferentes tareas se le asignen a diferentes funciones. Además es una tarea que comúnmente se requiere realizar cuando se tienen datos reales provenientes de mediciones en algunos puntos pero se requiere conocer más valores de los que se tiene.

■ **Ejemplo 2.1 — Funciones para interpolar puntos.** Supongamos que se desea construir un polinomio interpolante de Lagrange para un conjunto de datos (X_i, Y_i) con

$i \in \{1,\ldots,n\}$. Y además evaluarlo para un conjunto de puntos diferentes guardados en un vector x de longitud N, por ejemplo $N = 1000$. Este problema puede resolverse definiendo la siguiente estructura o algoritmo:

Función Principal

```
%-- Dados los datos X, Y , x,  realizar un ciclo
N = length(x);
 for s=1:N
   y(s)=Eval_pLagrange(X,Y,x(s))
 end

%-- Post-processing , por ejemplo
plot(x,y, X,Y, '*')
```

Función auxiliar para evaluar un punto en $p(x)$

```
function y= Eval_pLagrange(X, Y, evalx)
   n=length(X);
   p=0;
   for i=1:n
      p=p+ Y(i)*oneLagrange_pol(X,i,evalx);
   end
end
```

Función auxiliar para evaluar un término de la sumatori en $p(x)$

```
function L=oneLagrange_pol(X,k,evalx)
   L=1;
   n=length(X)
   for i=1:n
      if(i~=k)
         L=L*((evalx-X(i))/(X(k)-X(i)))
      end
   end
end
```

■

Los interpolantes de Lagrange tienen la ventaja de ser intuitivos en cuanto a que por construcción $L_i^{(n)}(x)$ es igual a 1 en x_i y es igual a 0 en los demás x_j's. Sin embargo presentan el inconveniente de que cada elemento de la base depende de todos los puntos x_j, por lo que al intentar incluir un nuevo punto x_j en la construcción se tendrían que recalcular todos los términos $L_i^{(n)}(x)$.

Una alternativa a este problema es considerar la base de polinomios como lo hizo Newton [2] definida como $\{N_i, \quad i = 1,\ldots,n\}$ que se construye como sigue:

$$N_1(x) = 1, \tag{2.4}$$

$$N_i(x) = \prod_{j=1}^{i-1}(x-x_j), \quad i \in \{2,\ldots,n\}, \tag{2.5}$$

y que puede ser usada para definir un polinomio

$$p(x) = \sum_{i=1}^{n} a_i N_i(x). \tag{2.6}$$

Con este Ansatz para la forma del polinomio $p(x)$ puede encontrarse por recursión sucesiva que

$$y_1 = p(x_1) = a_1 \tag{2.7}$$

$$y_2 = p(x_2) = a_1 + a_2(x_2-x_1) \tag{2.8}$$

$$y_3 = p(x_3) = a_1 + a_2(x_2-x_1) + a_3(x_3-x_1)(x_3-x_2) \tag{2.9}$$

$$\vdots$$

$$y_n = p(x_n) = a_1 + a_2(x_2-x_1) + \cdots + a_n(x_n-x_1)\cdots(x_n-x_{n-1}) \tag{2.10}$$

que pude ser visto como un sistema de ecuaciones lineales. Estas ecuaciones pueden ser modificadas fácilmente para formar un sistema de ecuaciones lineales como:

$$M_n \bar{a} = \bar{y} \tag{2.11}$$

con $\bar{a}$ y $\bar{y}$ los vectores con los componentes a_i, y_i y la matriz del sistema

$$M_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & (x_2-x_1) & 0 & \cdots & 0 \\ 1 & (x_3-x_1) & (x_3-x_1)(x_3-x_2) & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & (x_n-x_1) & (x_n-x_1)(x_n-x_2) & \cdots & (x_n-x_1)\cdots(x_n-x_{n-1}) \end{pmatrix} \tag{2.12}$$

de tamaño $n \times n$.

En caso de que un nuevo punto (x_{n+1}, y_{n+1}) deba ser considerado, bastaría con construir la matriz M_{n+1} como

$$M_{n+1} = \left[\begin{array}{ccc|c} & M_n & & 0_{nx1} \\ \hline 1 & (x_{n+1}-x_1) & \cdots & \prod_{j=1}^{n}(x_{n+1}-x_j) \end{array}\right] \tag{2.13}$$

[2] Isaac Newton $(1643-1727)$. Físico y Matemático inglés. Profesor en Cambridge que es ampliamente reconocido por haber realizado desarrollos fundamentales en Mecánica Clásica y Cálculo Diferencial.

y considerar ahora el sistema de ecuaciones

$$M_{n+1}\begin{pmatrix} a_1 \\ \vdots \\ a_{n+1} \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_{n+1} \end{pmatrix} \tag{2.14}$$

Además el sistema es de forma triangular inferior, lo que hace que resolver el problema de encontrar los coeficientes a_i sea simple de resolver numéricamente.

En un capítulo posterior de este curso nos ocuparemos de ver algoritmos para resolver problemas con este tipo de matrices. Por ahora basta con observar que el valor de a_1 esta ya resuelto, esto hace que a_2 sea muy fácil de calcular y este proceso puede continuarse hasta calcular todos los coeficientes a_i. En Matlab, puede usarse por ahora la resolución por medio del comando '\' (backslash). Una vez conocidos los coeficientes a_i es fácil evaluar el polinomio de acuerdo a las ecuaciones (2.5) y (2.6) como

$$p(x) = a_1 + a_2(x-x_1) + a_3(x-x_1)(x-x_2) + \cdots + a_n(x-x_1)\cdots(x-x_n) \tag{2.15}$$

Hasta ahora hemos asumido que el conjunto de puntos a aproximar se verá bien reflejado si lo hacemos a través de un polinomio. Sin embargo, los pares (x_i, y_i) a aproximar pueden provenir de una función arbitraria con gran cantidad de oscilaciones o con tendencias cambiantes. De esta reflexión resulta la pregunta inmediata sobre qué tan bien aproximará nuestro polinomio de interpolación (Lagrange o Newton) a la función original. A este respecto existe el siguiente teorema importante:

Teorema 2.2 — Calidad de la aproximación polinomial. Sea $f \in C^n[a,b]$ y p el polinomio de grado menor o igual a $n-1$ que interpola a los n pares de la forma $(x_i, f(x_i)), i \in \{1,\ldots,n\}, x_i \in [a,b]$. Entonces para todo $x \in [a,b]$ existe un $\varphi = \varphi(x) \in (a,b)$ tal que

$$f(x) - p(x) = \frac{1}{n!} f^{(n)}(\varphi(x)) \prod_{j=1}^{n} (x - x_j) \tag{2.16}$$

(Sin demostración en este curso)

Note que, a diferencia de una cota de error para una aproximación de Taylor alrededor de un punto $x*$, esta cota de error entre $p(x)$ y $f(x)$ incluye información acerca de todo el conjunto de puntos $x_1, \ldots, x_n$. Además es interesante ver que este resultado es válido para todas las funciones continuas, independientemente de que tengan una forma que provenga de un polinomio, una función trigonométrica o cualquier otra funcion que cumpla la condición de continuidad.

Veamos algunos ejemplos de la información que puede ser obtenida a través de este teorema:

■ **Ejemplo 2.2 — Calidad de la interpolacion de la funcion $\cos(x)$.** Considere la interpolación polinomial a la función $f(x) = \cos(x)$ a través de un polinomio de grado 6

que interpola utilizando 7 puntos en el intervalo $[0,1]$. De acuerdo al teorema 2.2 existe un $\varphi(x)$ tal que

$$\cos(x) - p(x) = \frac{1}{7!}\operatorname{sen}(\varphi(x)) \prod_{j=1}^{6}(x - x_j) \tag{2.17}$$

y sabemos que

$$|\operatorname{sen}(\varphi(x))| \leq 1 \tag{2.18}$$

y como $x \in [0,1]$ entonces se tiene que

$$|x - x_j| \leq 1 \Rightarrow \left| \prod_{j=1}^{6}(x - x_j) \right| \leq 1 \tag{2.19}$$

$$\Rightarrow |\cos(x) - p(x)| \leq \frac{1}{5040} \approx 0{,}198412x10^{-3} \tag{2.20}$$

■

■ **Ejemplo 2.3 — Calidad de la interpolacion de un polinomio.** Si $f \in P_m$ con $m \leq n-1 \Rightarrow f^{(n)} \equiv 0$, lo que indica según el Teorema 2.2 que $f(x) - p(x) = 0$. Es decir que cualquier función polinomial de grado $m \leq n-1$ será exactamente interpolada. Si $m \geq n \Rightarrow f^{(n)} \neq 0$ y el polinomio p solo será cercano a f. ■

■ **Ejemplo 2.4 — Un caso de mala calidad de la interpolacion.** Desafortunadamente, la dependencia de la cota en la derivada puede hacer también que las cosas no funcionen bien cuando la derivada no esta bien acotada.
Considere la función $f(x) = (1+x^2)^{-1}$, cuyas derivadas son

$$f^{(n)}: \quad n=1 \qquad -1(1+x^2)^{-2}(2x) \tag{2.21}$$

$$n=2 \qquad 2\cdot 1(1+x^2)^{-3}(4x^2) + \cdots \tag{2.22}$$

$$n=3 \qquad -3\cdot 2\cdot 1(1+x^2)^{-4}(8x^3) + \cdots \tag{2.23}$$

$$\vdots \qquad \vdots$$

$$n \qquad (-1)^n n!(1+x^2)^{-(n+1)}2^n x^n + \cdots \tag{2.24}$$

por lo que los valores de la función de derivada orden n pueden crecer enormemente. Este crecimiento puede explicarse como:

$$\left| f^{(n)} \right| \approx \mathcal{O}(2^n n!(1+x^2)^{-n-n}x) = 2^n n!\mathcal{O}(|x|^{-n-2}), \tag{2.25}$$

por lo que

$$\left| \frac{1}{n!} f^{(n)} \right| \approx 2^n \mathcal{O}(|x|^{-n-2}), \tag{2.26}$$

y el factor $2^n n!$ puede crecer fácilmente, independientemente de los valores que tome la variable x. ■

En este último ejemplo pareciera quedar a la vista que usar mayor cantidad de puntos provocará una aproximación mas pobre debido unicamente a la presencia de la derivada en la cota del teorema anterior. Esto podría resolverse restringiendo el polinomio de interpolación a través de sus derivadas, o intentando aproximar a la función original a trozos. Estos son precisamente las estrategias que abordaremos en las siguientes secciones de este capítulo.

2.2 Interpolación de Hermite

La interpolación de Hermite[3] utiliza no sólo las evaluaciones de la función original, sino también los valores de sus derivados, por ejemplo

$$\begin{array}{ccc} x_1 & x_2 & \cdots x_n \\ f(x_1) & f(x_2) & \cdots f(x_n) \\ f'(x_1) & f'(x_2) & \cdots f'(x_n) \end{array}$$

Al incluir los valores de las derivadas, se evitan comportamientos en que solo el punto es coincidente con el valor del interpolante y puede esperarse una mejor aproximación.

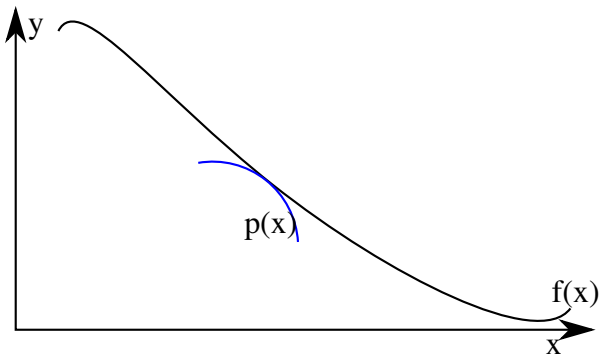


Figura 2.1: Aproximación por interpolación de Hermite

Como es de esperarse, el número de ecuaciones aumentará al incluir las derivadas para determinar el polinomio interpolante. Similar a lo que mostramos para la interpolación de Lagrange en el teorema 2.1, aquí es posible demostrar el siguiente:

Teorema 2.3 Dados n puntos distintos en $[a,b]$, $x_1,\cdots,x_n$ y dada una función $f \in C^1([a,b])$ existe un único polinomio p de grado mínimo tal que

$$\left.\begin{array}{l} p(x_i) = f(x_i) \\ p'(x_i) = f'(x_i) \end{array}\right\} \text{ para } i = 1,\ldots,n \tag{2.27}$$

[3]Charles Hermite (1822 – 1901). Matemático frances, Prof. en la École Polytechnique y la Sorbonne en París. Contribuyó en teoría de números y de las funciones elípticas.

(Sin demostración en este curso)

El polinomio p que satisface las condiciones de la interpolación de Hermite dado por la ecuación (2.27) es llamado "polinomio osculante" debido a su propiedad de tangencialidad a la función en los puntos $(x_i, f(x_i))$. De esta misma ecuación (2.27) se puede ver que el interpolante debe satisfacer $2n$ condiciones por lo que el grado del polinomio será a lo máximo $2n-1$. En la práctica muchas veces no contamos con $f'(x)$ sino solo con puntos discretos $(x_i, f(x_i))$. Para aproximar f' se puede usar el teorema del valor medio, que asegura la existencia de $\varphi \in (x_i, x_{i+1})$ tal que

$$f'(\varphi) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

siempre y cuando $f \in C^1([x_i, x_{i+1}])$, lo cual esta dado en nuestro caso.

Con esta idea en mente, se puede aproximar a una derivada en un punto a través de diferencias finitas utilizando alguna de las siguientes aproxmaciones:

- Diferencias hacia atrás:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \tag{2.28}$$

- Diferencias hacia adelante:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{2.29}$$

- Diferencias centrales:

$$f(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}} \tag{2.30}$$

Veamos como encontrar el polinomio de Hermite usando la aproximación por diferencias hacia adelante:

Buscamos un polinomio de la forma

$$p(x) = a_1 + a_2 x + \ldots + a_{n+1} x^n + a_{n+2} x^{n+1} + \ldots a_{2n} x^{2n-1} \tag{2.31}$$

con derivada

$$p'(x) = \quad a_2 + \ldots + n a_{n+1} x^{n-1} + (n+1) a_{n+2} x^n + \ldots (2n-1) a_{2n-1} x^{2n-2} \tag{2.32}$$

La ecuación (2.31) con el polinomio nos da un sistema lineal de ecuaciones

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{2n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{2n-1} \end{bmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{2n} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} \tag{2.33}$$

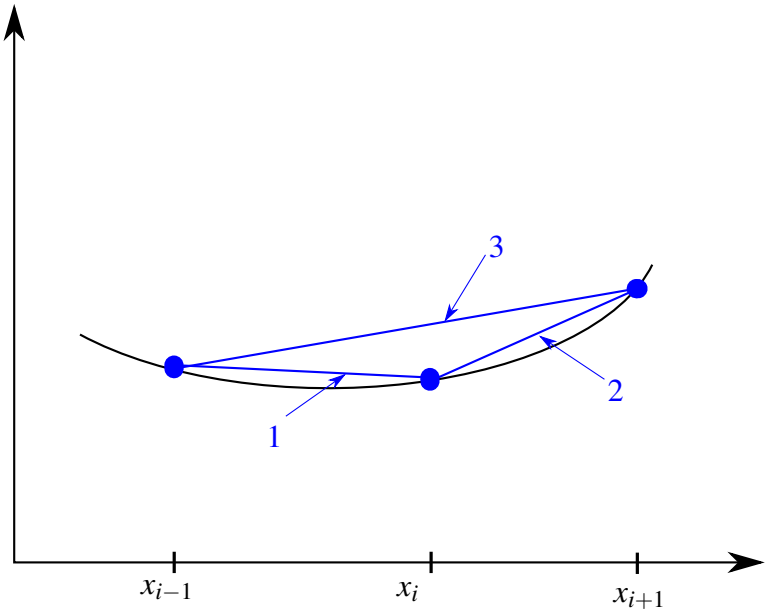


Figura 2.2: Distintas aproximaciones para la derivada: Diferencias hacia atrás (1), Diferencias hacia adelante (2) y Diferencias centrales (3).

y entonces lo que tenemos es un sistema lineal definido para $2n$ incógnitas y n ecuaciones. Las otras $n+1$ ecuaciones las obtendremos de la ecuación de la derivada. Para ello usaremos la aproximación por diferencias hacia adelante de acuerdo a la ecuación (2.28) para todos los x_i con $i = 1,\ldots,n-1$ y para x_n usaremos la aproximación por diferencias finitas hacia atrás de acuerdo con la ecuación (2.29). De esta manera, el sistema de x_1 hasta x_{n-1} quedaría como

$$\begin{bmatrix} 0 & 1 & 2x_1 & \cdots & (2n-1)x_1^{2n-2} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 1 & 2x_{n-1} & \cdots & (2n-1)x_{n-1}^{2n-2} \end{bmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{2n} \end{pmatrix} = \begin{pmatrix} \frac{f(x_2)-f(x_1)}{x_2-x_1} \\ \vdots \\ \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} \end{pmatrix} \tag{2.34}$$

y la última ecuación para la derivada en x_n sería tomada la aproximación hacia atrás de acuerdo a la ecuación (2.29) como

$$\begin{bmatrix} 0 & 1 & 2x_n & \cdots & (2n-1)x_n^{2n-2} \end{bmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{2n} \end{pmatrix} = \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} \tag{2.35}$$

El sistema a resolver puede formarse por bloques con las ecuaciones (2.36), (2.34) y

(2.35) y tendrá la forma

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{2n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{2n-1} \\ 0 & 1 & 2x_1 & \cdots & (2n-1)x_1^{2n-2} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 1 & 2x_{n-1} & \cdots & (2n-1)x_{n-1}^{2n-2} \\ 0 & 1 & 2x_n & \cdots & (2n-1)x_n^{2n-2} \end{bmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{2n} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \\ \frac{f(x_2)-f(x_1)}{x_2-x_1} \\ \vdots \\ \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} \\ \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} \end{pmatrix} \tag{2.36}$$

o bien

$$M\bar{a} = \bar{b} \tag{2.37}$$

con $M \in \mathbb{R}^{2nx2n}$, $\bar{a} = (a_1, \ldots, a_{2n})^T \in \mathbb{R}^{2n}$, $\bar{b} \in \mathbb{R}^{2n}$.

La construcción de M y $\bar{b}$ es muy sencilla de realizar en Matlab, R o cualquier lenguaje de programación. Con esta interpolación se espera que las problemáticas de polinomios que "exploten" en áreas donde no hay datos, sean disminuidos o eliminados. Sin embargo notemos dos posibles problemas:

- En un problema práctico, pudierá ser que la cantidad de datos que se tiene sea muy grande, haciendo que un sistema de tamaño $2n \times 2n$ deba ser mantenido en memoria computacional y quizá sobrepasando el límite de datos a ser manejado con sencillez. Esto se extiende a la capacidad y confiabilidad al resolver el sistema $M\bar{a} = \bar{b}$.
- La construcción de la matriz M implica cálculos de grandes potencias de x (hasta x^{2n-1}). En caso de que n sea medianamente grande y $x \gg 1$, estos valores pueden crecer e incluso llegar al área de overflow (obteniendo valores de $\pm Inf$ en nuestra aritmética de punto flotante). Del mismo modo si $x \ll 1$, puede llegarse fácilmente al underflow.

Como ya hemos visto y experimentado, utilizar polinomios para interpolar puntos se convierte en un problema con grandes oscilaciones en el resultado. Esto es conocido como el fenómeno de Runge[4], quien encontró esta problemática y la asoció a que, a pesar de que aunque la teoría dada por el teorema de Weierstrass [5] asegura que toda función continua en $[a,b]$ tiene un conjunto de polinomios en $P_1, P_2, \ldots$ que converge uniformemente a f, este teorema no da ninguna estrategia para encontrar dichos polinomios en la práctica.

[4]Carl David Tolmé Runge $(1856-1927)$. Matemático alemán, estudiante de Weierstrass que trabajo con Felix Klein desde su Lehrstuhl en Göttingen.

[5]Karl Weierstrass $(1815-1897)$. Matemático alemán llamado el "padre del análisi moderno". Formalizó conceptos del cálculo y el análisis como la definición de función continua. Teorema de Borel-Weierstrass.

Una alternativa es no intentar resolver el problema de manera global en el intervalo $[x_1, x_n]$, sino resolver pequeños problemas en subintervalos de este dominio, con la esperanza de que usando cada vez solo unos pocos puntos el grado polinomial se mantenga bajo.

2.3 Splines

Desde la década de los 1920's estaba claro que la interpolación polinomial contenía el problemático fenómeno de Runge. Sin embargo no fue sino hasta los trabajos de Schoenberg[6] en la década de los 1940's cuando se propuso una técnica que evitará este tipo de problema.

La idea básica es unir a la sucesión de puntos

$$a = x_1 < x_2 < \cdots < x_n = b \tag{2.38}$$

evaluados en la función $f : [a,b] \to \mathbb{R}$ de manera que en cada subintervalo $[x_i, x_{i+1}]$ se tenga un polinomio capaz de describir una trayectoria suave y cuyo grado polinomial sea de preferencia pequeño.

Esto puede verse que será resuelto con polinomios de un orden mayor o igual que tres, por lo que en la práctica son los de orden cúbico los que más se utilizan. Por esta razón nos concentraremos en este tipo de splines.

Definición 2.4 — Splines Cúbicos. Considere $f : [a,b] \to \mathbb{R}$ y un conjunto de nodos $a = x_1 < x_2 < \cdots < x_n = b$. Un spline cúbico para f es una función $S : [a,b] \to \mathbb{R}$ tal que se cumple:

(i) $S|_{[x_j, x_{j+1}]} = S_j$ donde $S_j \in P_3([x_j, x_{j+1}] \quad \forall j \in \{1,2,\ldots,n-1\}$
(ii) $S(x_j) = f(x_j) \quad \forall j \in \{1,\ldots,n\}$
(iii) $S_{j+1}(x_{j+1}) = S_j(x_{j+1}) \quad \forall j \in \{1,\ldots,n-2\}$
(iv) $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1}) \quad \forall j \in \{1,\ldots,n-2\}$
(v) $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1}) \quad \forall j \in \{1,\ldots,n-2\}$
(vi) Una de las siguientes condiciones de frontera se satisfacen
  (a) $S''(x_1) = S''(x_n) = 0$ (frontera libre)
  (b) $S'(x_1) = f'(x_1)$, $S'(x_n) = f'(x_n)$ (frontera sujeta)
  (c) Una mezcla de (a) y (b), por ejemplo $S''(x_1) = 0$, $S'(x_n) = f'(x_n)$

Con el fin de simplificar los desarrollos de construcción del spline cúbico denotaremos a los polinomios S_j con coeficientes a_j, b_j, c_j, d_j de la forma

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3. \tag{2.39}$$

[6] Matemático de origen Rumano. Estudió en Berlin y Göttingen con Edmund Landau y desde 1930 trabajo en los EEUU llegando a ser Profesor de la University of Pennsylvania. En 1943-1945 fue liberado de su puesto para unirse al grupo de científicos trabajando en el Aberdeen Proving Ground (el sitio de investigación más antiguo de la armada estadounidense) donde realizó su mayor contribución científica: los splines.

Este polinomio corresponde a la función S en $[x_j, x_{j+1}]$ y la completa determinación del spline consiste en encontrar los coeficientes para los trozos $S_1, S_2, \ldots, S_{n-1}$. El polinomio tiene las primeras dos derivadas dadas por

$$S'_j(x) = b_j + 2c_j(x-x_j) + 3d_j(x-x_j)^2 \tag{2.40}$$
$$S''_j(x) = 2c_j + 6d_j(x-x_j) \tag{2.41}$$

Ahora, considerando la condición de interpolación (ii) de la definición 2.4 debe cumplirse que $S_j(x_j) = f(x_j)$ por lo que al evaluar el polinomio de la ecuación (2.39) en el punto x_j tendremos que

$$\Rightarrow\ a_j = f(x_j) \tag{2.42}$$

Usando el hecho que S_j y S_{j+1} deben coincidir en el punto x_{j+1} según la condición (iii) de la definición 2.4 se tiene que

$$\begin{aligned} S_{j+1}(x_{j+1}) &= a_{j+1}, \\ &= a_j + b_j(x_{j+1}-x_j) + c_j(x_{j+1}-x_j)^2 + d_j(x_{j+1}-x_j)^3. \end{aligned} \tag{2.43}$$

En estos desarrollos estamos considerando un conjunto de puntos x_j que no necesariamente estan igualmente espaciados entre ellos. Si denotamos los espaciamientos entre x_j y x_{j+1} como h_j tendríamos que la ecuación anterior se convierte en

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3. \tag{2.44}$$

Usando ahora la condición (iv) en la definición 2.4 respecto a la primera derivada tenemos

$$b_{j+1} = b_j + 2c_j(x_{j+1}-x_j) + 3d_j(x_{j+1}-x_j)^2 \tag{2.45}$$

o bien

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2 \tag{2.46}$$

Análogamente, usando la condición (v) de la definición 2.4 correspondiente a la segunda derivada tendremos

$$2c_{j+1} = 2c_j + 6d_j(x_{j+1}-x_j), \tag{2.47}$$

o bien

$$2c_{j+1} = 2c_j + 6d_j h_j, \tag{2.48}$$

que al despejar d_j se convierte en

$$d_j = \frac{c_{j+1}-c_j}{3h_j}. \tag{2.49}$$

Reemplacemos ahora este valor de d_j en (2.49) en la ecuación (2.44) para obtener

$$0 = (a_{j+1} - a_j) - b_j h_j - c_j h_j^2 - \frac{h_j^2}{3}(c_{j+1} - c_j) \tag{2.50}$$

que nos permite ahora despejar el valor de b_j como

$$b_j h_j = (a_{j+1} - a_j) - \frac{h_j^2}{3}(c_{j+1} + 2c_j) \tag{2.51}$$

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(c_{j+1} + 2c_j) \tag{2.52}$$

y como el índice j es cualquiera de los índices posibles, podemos considerar esta misma expresión para un índice anterior $j-1$ de manera análoga como

$$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(c_j + 2c_{j-1}) \tag{2.53}$$

Reemplacemos ahora el valor de d_j que obtuvimos en (2.49) en la ecuación (2.46)

$$\begin{aligned} b_{j+1} &= b_j + 2c_j h_j + 3h_j^2 \frac{c_{j+1} - c_j}{3h_j} \\ &= b_j + 2c_j h_j + h_j(c_{j+1} - c_j) \\ &= b_j + h_j(c_j + c_{j+1}) \end{aligned} \tag{2.54}$$

Análogamente, esta ecuación puede escribirse para los índices j y $j-1$) como

$$b_{j-1} + h_{j-1}(c_{j-1} + c_j) = b_j \tag{2.55}$$

Reemplazando ahora la ecuación (2.53) en el lado izquierdo de esta ecuación y la ecuación (2.52) en el lado derecho de la misma se obtiene

$$\frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(c_j + 2c_{j-1}) + h_{j-1}(c_{j-1} + c_j) = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(c_{j+1} + 2c_j) \tag{2.56}$$

Reordenando para tener coeficientes c del lado izquierdo y coeficientes a del lado derecho, así como multiplicando todos lo términos por 3 se llega a

$$h_j(c_{j+1} + 2c_j) - h_{j-1}(c_j + 2c_{j-1}) + 3h_{j-1}(c_{j-1} + c_j) = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) \tag{2.57}$$

Ahora que el lado derecho de la ecuación (2.57) solo contiene coeficientes a_j's y el izquierdo solo c_j's podemos reescribir el lado izquierdo como

$$\begin{aligned} c_{j-1}(-2h_{j-1}+3h_{j-1})+c_j(2h_j-h_{j-1}+3h_{j-1})+c_{j+1}(h_j) \\ = h_{j-1}c_{j-1}+2(h_j+h_{j-1})c_j+h_jc_{j+1} \\ = \begin{bmatrix} h_{j-1} & 2(h_j+h_{j-1}) & h_j \end{bmatrix} \begin{bmatrix} c_{j-1} \\ c_j \\ c_{j+1} \end{bmatrix}, \end{aligned} \tag{2.58}$$

por lo que si ahora definimos las nuevas variables auxiliares

$$t_j = \frac{3}{h_j}(a_{j+1}-a_j) - \frac{3}{h_{j-1}}(a_j-a_{j-1}), \tag{2.59}$$

podemos usar la formulación en (2.58) para reescribir la ecuación (2.57) como

$$h_{j-1}c_{j-1}+2(h_j+h_{j-1})c_j+h_jc_{j+1} = t_j. \tag{2.60}$$

Esta ecuación puede reescribirse para todos los puntos interiores de nuestro problema de encontrar los polinomios cúbicos, es decir, que podemos reescribirla para $j \in \{2,\ldots,n-2\}$, resultando en el sistema lineal de ecuaciones

$$\begin{bmatrix} h_1 & 2(h_1+h_2) & h_2 & 0 & \cdots & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & \cdots & 0 \\ 0 & 0 & h_3 & 2(h_3+h_4) & \cdots & 0 \\ 0 & 0 & 0 & h_4 & \cdots & 0 \\ \vdots & \vdots & & & & \\ 0 & 0 & 0 & 0 & \cdots & h_{n-2} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} t_2 \\ t_3 \\ t_4 \\ t_5 \\ \vdots \\ t_{n-1} \end{bmatrix} \tag{2.61}$$

La matriz del sistema es de tamaño $(n-2)\times n$ y es tridiagonal por lo que se requieren dos ecuaciones más para poder cerrar el sistema. Estas ecuaciones faltantes corresponden a lo que se espera de los puntos frontera x_1 y x_n con respecto a la condición (vi) de la definición 2.4.

Para el caso de la condición de frontera libre en x_1 se tendría la condición de que $S_1''(x_1) = 0$, y basta revisar la forma de la segunda derivada en la ecuación (2.41) para ver que el primer trozo del Spline debería cumplir que

$$0 = 2c_1 + 6d_1(x_1 - x_1), \tag{2.62}$$

por lo que se tiene que $c_1 = 0$, o equivalentemente

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} = 0. \tag{2.63}$$

De manera similar, la condición de frontera libre en x_n corresponde a tener que la ecuación (2.41) en el último trozo del spline se convierte en la condición $c_{n-1} = 0$, o equivalentemente

$$\begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} = 0, \tag{2.64}$$

y el sistema completo de ecuaciones puede formarse haciendo uso del sistema en (2.61) y complementándolo con las ecuaciones (2.63) y (2.64) para formar un sistema para las $n-1$ variables c_j.

Con esto, el sistema completo para el caso de fronteras libres es de la forma $Mc = F$ con

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & \cdots & 0 \\ & h_2 & 2(h_2+h_3) & h_3 & \cdots & 0 \\ & & \ddots & \ddots & \ddots & \vdots \\ & & & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}_{n\times n} \tag{2.65}$$

$$\begin{aligned} c &= \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix}^T \in R^n \\ F &= \begin{bmatrix} 0 & t_2 & t_3 & \cdots & t_{n-1} & 0 \end{bmatrix}^T \in R^n \end{aligned} \tag{2.66}$$

Otra posibilidad en la frontera de x_1 es el caso de frontera fija o sujeta, en la que se asume que conocemos la primera derivada como un valor fijo $S_1'(x_1)$. Usando la ecuación (2.40) esto queda como

$$S_1'(x_1) = b_1 + 2c_1(x_1 - x_1) + 3d_j(x_1 - x_1)^2, \tag{2.67}$$

es decir que $b_1 = S_1'(x_1)$ y dado que ya sabemos que la ecuación (2.42) define claramente a los a_j's, podemos usar la ecuación (2.53) como

$$b_1 = \frac{1}{h_1}(a_2 - a_1) - \frac{h_1}{3}(2c_1 + c_2) \tag{2.68}$$

o bien

$$2h_1c_1 + h_1c_2 = \frac{3}{h_1}(a_2 - a_1) - 3S_1'(x_1) \tag{2.69}$$

De manera análoga, si se tiene una condición de frontera sujeta en x_n quiere decir que conocemos el valor de $S_{n-1}'(x_n)$, es decir

$$S_{n-1}'(x_n) = b_{n-1} + 2c_{n-1}h_{n-1} + 3d_{n-1}h_{n-1}^2 \tag{2.70}$$

y haciendo uso de (2.49) y (2.53) esto puede reescribirse como

$$-3S'_{n-1}(x_n) = -\frac{3}{h_{n-1}}(a_n - a_{n-1}) + h_{n-1}(2c_{n-1} + c_n) - 6c_{n-1}h_{n-1} - 3h_{n-1}^2\frac{c_n - c_{n-1}}{h_{n-1}}. \tag{2.71}$$

Reordenando términos se tiene que

$$\frac{3}{h_{n-1}}(a_n - a_{n-1}) - 3S'(x_n) = (2h_{n-1} - 6h_{n-1} + 3h_{n-1})c_{n-1} + (h_{n-1} - 3h_{n-1})c_n, \tag{2.72}$$

o bien

$$h_{n-1}c_{n-1} + 2h_{n-1}c_n = 3S'(x_n) - \frac{3}{h_{n-1}}(a_n - a_{n-1}). \tag{2.73}$$

Estas ecuaciones (2.69) y (2.73) servirán para ampliar el sistema y tener una matriz de tamaño $n \times n$ que, usando fronteras fijas puede formarse muy similar al sistema de las ecuaciones (2.65) y (2.74) pero intercambiando el primer y último renglón de M por

$$\begin{aligned} M_1 &= \begin{bmatrix} 2h_1 & h_1 & 0 & \cdots & 0 \end{bmatrix} \\ M_n &= \begin{bmatrix} 0 & \cdots & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix} \end{aligned} \tag{2.74}$$

de acuerdo a las ecuaciones (2.69) y (2.73) respectivamente. Los cambios correspondientes al lado derecho F también deben hacerse de la forma

$$\begin{aligned} F_1 &= -3f'(x_1) + \frac{3}{h_1}(a_2 - a_1) \\ F_2 &= 3f'(x_n) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \end{aligned} \tag{2.75}$$

En caso de que las condiciones de frontera sean distintas en x_1 y x_n, solamente el primer o último renglón del sistema deberán ser modificados.

Con esto, sabemos como evaluar los coeficientes c_j's de los polinomios a trazos S_j. Sin embargo, el algoritmo completo debe calcular todos los coeficientes. Usando todos los desarrollos anteriores puede escribirse un algoritmo completo como sigue: Dado los puntos $x_1, \ldots, x_n$ y sus respectivos valores $f(x_1), \ldots, f(x_n)$. El spline cúbico que pasa por ellos se calcula como

- Defina $a_j = f(x_j)$, $j = 1, \ldots, n-1$
- Obtenga $c_1, \ldots, c_n$ resolviendo el sistema lineal definido por la matriz y lado derecho en las ecuaciones (2.65), (2.66) . En caso de condiciones de frontera fija, modifique este sistema de acuerdo a las condiciones en (2.74) y (2.75).
- Calcule los coeficientes b_j de acuerdo a la ecuación (2.53) como

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(c_{j+1} + 2c_j)$$

- Calcule los coeficientes d_j de acuerdo a la ecuación (2.49) como

$$d_j = \frac{1}{3h_j}(c_{j+1} - c_j)$$

2.4 Ejercicios

En estos ejercicios, se intenta realizar observaciones sobre los métodos de interpolación. Para ello, deberán considerarse los tiempos de cálculo y la complejidad de implementación inherentes a los métodos, sin considerar los tiempos referentes al pre-procesamiento y post-procesamiento de datos. Ejemplos de éstos tiempos son la selección de datos a interpolar o los comandos utilizados para realizar gráficos ilustrativos del problema.
Para esto, es recomendable que las funciones generales de los métodos tengan argumentos de salida, entre los cuales se encuentre el tiempo usado para realizar los cálculos.

Ejercicio 2.1. Interpolación de Lagrange (35 Puntos)
Defina una función en MATLAB para interpolar por el método de Lagrange y úsela para aproximar a la función $f : [0,1] \to \mathbb{R},\ f(x) = \text{sen}(x)$ utilizando n puntos igualmente espaciados. Repita este ejercicio para $n = 2,4,7,11,16$ y haga observaciones sobre la calidad en las aproximaciones. ∎

Ejercicio 2.2. Interpolación de Newton y función de Runge (35 Puntos)
Considere la función de Runge

$$f(x) = \frac{1}{1+25x^2}, \qquad x \in [-1,1].$$

Use una implementación de la interpolación de Newton en MATLAB para aproximar a esta función en $[-1,1]$ utilizando 5, 9, 13, 17 y 21 puntos igualmente espaciados. Observe como el grado polinomial afecta la calidad de la aproximación. Realice una aproximación similar usando el programa de interpolación de Lagrange del ejercicio anterior. ¿Son los resultados mejores/peores/equivalentes? ∎

Ejercicio 2.3. Comparando interpolaciones (30 Puntos)
Defina dos siluetas simples que puedan representarse como un conjunto de 2 a 5 funciones en $\mathbb{R}^2$, use al menos una silueta con solo 3 puntos y otra con al menos 10 puntos. Implemente un programa que utilice interpolación de Lagrange y de Newton y compare los resultados de ambas aproximaciones en términos de 'calidad visual' (subjetivo), eficiencia en tiempo de calculo y complejidad de la implementación. ∎

Los modelos de predicción climática se basan en la inclusión de información sobre corrientes de presión, archivos históricos, estado actual de la temperatura, humedad, y una serie amplia de factores con influencia climática. La compleja relación de todos estos factores resulta en modelos basados en una gran cantidad de datos que no es posible actualizar de manera constante.
Por esta razón, en ocasiones se toman predicciones en periodos largos de tiempo (del orden de 12 o 24 horas) para predecir el estado del tiempo y con base en estos valores se calcula indirectamente una predicción para cada hora del día, basada principalmente en

Figura 2.3: Temperatura pronosticada para Monterrey el día 13.02.2015

una interpolación de los valores predichos para el periodo largo de tiempo.

Ejercicio 2.4. Interpolando Temperaturas (30 Puntos)

Asuma que las predicciones diarias de temperatura son conocidas para dos horas representativas (por ejemplo al amanecer y después del mediodía) en un conjunto de varios días. Diseñe un pseudo-código por escrito en el que describa cómo pueden ser calculadas las predicciones en intervalos de 1 hora, incluyendo la información sobre los datos de entrada y salida de cada parte del pseudo-código.

En su diseño no es necesario que describa el pseudo-código para calcular la interpolación. Asuma que existe la función

```
[evaly,execTime] = Interpolate(dataX,dataY,evalx)
```

que acepta los datos a interpolar y un vector de puntos a evaluar, y retorna un vector (*evaly*) de valores evaluados y la cantidad de segundos utilizada para calcular la interpolación (*execTime*). ■

Ejercicio 2.5. Interpolación de Hermite (50 Puntos)

Implemente rutinas en MATLAB para una función de interpolación utilizando el método de Hermite. Utilice la aproximación a las derivadas por diferencias hacia adelante $\frac{dy}{dx} \approx \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$ para todos los puntos x_i, $i = 1,\ldots,n-1$.

Consulte las predicciones diarias de temperatura en un servicio de Internet para los siguientes días y a dos horas representativas. Pruebe su código de inerpolación de Hermite realizando las predicciones de la temperatura en intervalos de 1 hora. Construya los resultados y represéntelos de forma gráfica en el espacio tiempo-temperatura. Use comandos de MATLAB (`xtick,grid on,xlabel,ylabel,plot,...`) para lograr un gráfico que muestre los bloques de 24 horas, y que incluya la descripción y unidades en los ejes. Además, use las opciones que ofrece la función `plot` para graficar como asteriscos los datos originales tomados de Internet. ■

Ejercicio 2.6. Implementación de splines cúbicos (40 Puntos)
Implemente un código para el cálculo del spline cúbico pasando por una lista de n puntos de la forma (X,Y) usando condiciones de frontera libre en ambos lados. Use preferentemente versiones vectoriales para las evaluaciones de las operaciones que así lo permitan. ■

Ejercicio 2.7. Splines cúbicos con frontera fija (15 Puntos)
Modifique el código del ejercicio anterior para generar un programa que permita el uso de condiciones de frontera fija. ■

Ejercicio 2.8. Spline para la fución de Runge (15 Puntos)
Use alguno de los códigos implementados para e cálculo de splines y construya una aproximación a la función de Runge

$$f(x) = \frac{1}{1+25x^2}, \qquad x \in [-1,1].$$

Realice esta aproximación usando puntos igualmente espaciados y comente sobre la cantidad de puntos que, según su criterio, son suficientes para lograr buenas aproximaciones. ■

Ejercicio 2.9. Utilizando los Splines cúbicos (15 Puntos)
Diseñe alguna silueta utilizando splines cúbicos. Trate de combinar ambas versiones para las condiciones de frontera. ■

3. Integración Numérica

Los usos prácticos de las integrales (cálculo de áreas, longitudes de curvas, volúmenes de sólidos, cálculos de FEM, etc) hacen que la necesidad de resolver integrales de manera numérica sea una tarea imprescindible. En muchos casos, es imposible integrar funciones de manera analítica dada su complejidad o incluso debido a que no existen funciones primitivas básicas que puedan ser utilizadas para ello.

En este capítulo nos concentraremos en resolver el problema de aproximación numérica a la integral de una función real, es decir $f : [a,b] \subset \mathbb{R} \to \mathbb{R}$. Las consideraciones para problemas en mayores dimensiones (por ejemplo en $\mathbb{R}^2$, $\mathbb{R}^3$) pueden generalizarse fácilmente de los contenidos presentados en este capítulo.

3.1 Reglas básicas de integración numérica

Empezaremos por considerar el problema de integrar una función $f : \mathbb{R} \to \mathbb{R}$, más precisamente $f : [a,b] \to \mathbb{R}$ donde $a,b \in \mathbb{R}$, $a < b$. Por otro lado, es suficiente con resolver la integración para una función definida en el $[0,1]$, pues el paso entre una función $\tilde{f} : [a,b] \to \mathbb{R}$ a otra de la forma $f : [0,1] \to \mathbb{R}$ puede realizarse fácilmente con un reescalamiento

$$x = \frac{1}{b-a}(\tilde{x} - a). \tag{3.1}$$

Desde los primeros conocimientos de integración se habla de que la integral $\int_0^1 f(x)dx$ corresponde al valor del área debajo de la curva $f(x)$ y se presenta la idea de que el área puede ser aproximada por el área de una gráfica de barras por debajo de la gráfica de

f. Incluso hay una conexión estrecha con la Integral de Riemann que se enseña en los cursos de cálculo integral. Como veremos a continuación, las estrategias de integración numérica más básicas retoman este concepto, usando luego algunas generalizaciones para obtener buenas aproximaciones al valor exacto de la integral.

Lo más simple que se puede hacer es tomar algún valor de entre todos los valores de $f(x)$ para algún $x \in [0,1]$ y tomar el área bajo la barra de esa altura tal como se muestra en la Figura 3.1, donde se tomó un punto $x_0 = 0{,}5$ y la barra de altura $f(x_0)$ y ancho 1 para aproximar al área bajo la curva de f usando el área de este rectángulo.

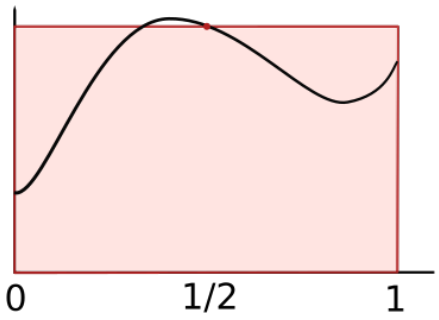


Figura 3.1: Área bajo la función f aproximada por el área del rectángulo de ancho 1 y altura $f(x_0)$ para algún $x_0 \in [0,1]$.

Claramente, el tomar un $x_0 \in [0,1]$ y usar el área bajo la línea con altura $f(x_0)$ dará un valor aproximado $\int_a^b f(x)dx = 1 \cdot f(x_0) = f(x_0)$ y esto será una aproximación muy sujeta a grandes errores para una amplia diversidad de funciones. Las mejoras posibles a esta estrategia son:

- Considerar el área bajo una curva que represente a una función más compleja que una constante.
- Considerar más de una barra de altura constante (histograma).

La segunda opción la analizaremos más adelante. Para la primera opción resulta natural tomar algo más complejo que una función constante, es decir una aproximación polinomial de grado mayor que cero (la constante).

Usando una aproximación en P_1, basta con tomar $x_1 = a$, $x_2 = b$ y construir la línea que pasa por los puntos $(a, f(a))$, $(b, f(b))$. De esta manera, la aproximación ilustrada en la Figura 3.2 estará dada como

$$\int_0^1 f(x)dx = (f(1) + f(2))\frac{1}{2}. \tag{3.2}$$

Por otro lado conocemos la representación de este elemento de P en forma de Lagrange como

$$p_1(x) = \sum_{j=1}^{2} f(x_j)\ell_j(x), \tag{3.3}$$

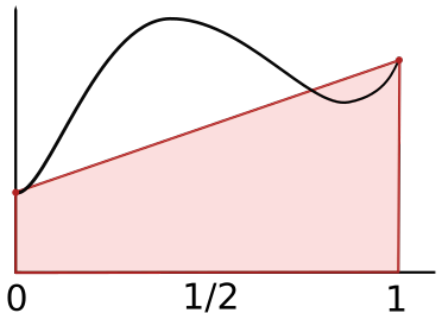


Figura 3.2: Aproximación usando un polinomio lineal a través del área de un trapecio.

con los polinomios

$$\ell_1(x) = \prod_{i=1,i\neq 1}^{2} \frac{x-x_i}{x_1-x_i} = \frac{x-x_2}{x_1-x_2} = \frac{x-1}{0-1} = 1-x, \tag{3.4}$$

$$\ell_2(x) = \prod_{i=1,i\neq 2}^{2} \frac{x-x_i}{x_2-x_i} = \frac{x-x_1}{x_2-x_1} = \frac{x-0}{1-0} = x. \tag{3.5}$$

Así que tomando el polinomio $p(x)$ como aproximación se tiene que

$$\begin{aligned}
\int_0^1 f(x)dx &\approx \int_0^1 P_1(x)dx, \\
&= \int_0^1 \sum_{j=1}^{2} f(x_j)\ell_j(x)dx, \\
&= f(0)\int_0^1 xdx + f(1)\int_0^1 (1-x)dx, \\
&= f(0)\left[\frac{x^2}{2}\right]_0^1 + f(1)\left[x-\frac{x^2}{2}\right]_0^1, \\
&= f(0)\frac{1}{2} + f(1)\frac{1}{2}, \\
&= \frac{1}{2}\left(f(0)+f(1)\right).
\end{aligned} \tag{3.6}$$

A esta fórmula de integración usando un polinomio de primer grado se le conoce como la regla del trapecio.

De manera similar, podemos usar una construcción similar para obtener una aproximación utilizando un polinomio cuadrático ($p \in P_2$) con la base de Lagrange usando los nodos $\{0,\ 1/2,\ 1\}$. Esta base puede verse fácilmente que está formada por los

polinomios

$$\ell_0^{(2)}(x) = \frac{x-1/2}{0-1/2}\cdot\frac{x-1}{0-1} = (-2x+1)(1-x) = 2x^2-3x+1 \tag{3.7}$$

$$\ell_1^{(2)}(x) = \frac{x-0}{1/2-0}\cdot\frac{x-1}{1/2-1} = (2x)(-2x+2) = -4x^2+4x \tag{3.8}$$

$$\ell_2^{(2)}(x) = \frac{x-0}{1-0}\cdot\frac{x-1/2}{1-1/2} = (x)(2x-1) = 2x^2-x \tag{3.9}$$

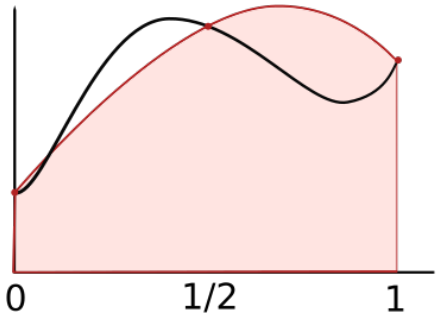


Figura 3.3: Aproximación mediante la regla de Simpson

Y la aproximación a la integral de $f(x)$ haciendo uso del mejor polinomio en P_2 que aproxima a esta función es ta como se ilustra en la Figura 3.3:

$$\begin{aligned}
\int_0^1 f(x)dx &\approx \int_0^1 p_2(x) \\
&= f(0)\int_0^1 (2x^2-3x+1)dx + f\left(\frac{1}{2}\right)\int_0^1 (-4x^2+4x)dx + \cdots \\
&\quad \cdots + f(1)\int_0^1 (2x^2-x)dx \\
&= f(0)\left[\frac{2}{3}x^3-\frac{3}{2}x^2+x\right]_0^1 + f\left(\frac{1}{2}\right)\left[-\frac{4}{3}x^3+2x^2\right]_0^1 + \cdots \\
&\quad \cdots + f(1)\left[\frac{2}{3}x^3-\frac{x^2}{2}\right]_0^1 \\
&= f(0)\frac{1}{6} + f\left(\frac{1}{2}\right)\frac{2}{3} + f(1)\frac{1}{6} \\
&= \frac{1}{6}\left[f(0)+4f\left(\frac{1}{2}\right)+f(1)\right]
\end{aligned} \tag{3.10}$$

Esta aproximación es conocida como la Regla de Simpson [1] para la integración y corresponde a integrar la función cuadrática que pasa por los puntos $(0, f(0))$, $\left(\frac{1}{2}, f\left(\frac{1}{2}\right)\right)$, $(1, f(1))$.

[1] Thomas Simpson (1710 – 1761). Matemático Británico a quien se le atribuye este desarrollo. Sin embargo, Johames Kepler ya lo había usado 100 años antes que Simpson.

De manera similar a las ecuaciones (3.6) y (3.10), el proceso de tomar un polinomio en P_n para algún $n > 2$ puede realizarse para obtener una fórmula similar. Para el caso $n = 3$ existe la regla conocida como Regla de Simpson de $3/8$ y para $n = 4$ se conoce como Regla de Milne. La siguiente definición contiene un resumen de las reglas de integración para polinomios de orden 1 hasta 4.

Definición 3.1 — Reglas de integración para $0 \leq x \leq 1$. La integral en el intervalo unitario $\int_0^1 f(x)dx$ puede ser aproximada utilizando las reglas de integración para polinomios. Estas reglas pueden resumirse como

Grado	Regla	Nodos / Fórmula
P_0	Punto medio	$\{\frac{1}{2}\}$
		$f\left(\frac{1}{2}\right)$
P_1	Trapecio	$\{0,1\}$
		$\frac{1}{2}(f(0)+f(1))$
P_2	Simpson	$\{0,\frac{1}{2},1\}$
		$\frac{1}{6}(f(0)+4f\left(\frac{1}{2}\right)+f(1))$
P_3	Simpson $\frac{3}{8}$	$\{0,\frac{1}{3},\frac{2}{3},1\}$
		$\frac{1}{8}(f(0)+3f\left(\frac{1}{3}\right)+3f\left(\frac{2}{3}\right)+f(1))$
P_4	Milne	$\{0,\frac{1}{4},\frac{1}{2},\frac{3}{4},1\}$
		$\frac{1}{90}\left(7f(0)+32f\left(\frac{1}{4}\right)+12f\left(\frac{1}{2}\right)+32f\left(\frac{3}{4}\right)+7f(1)\right)$

En caso de que la integral a calcular se requiera en un intervalo más general $[a,b]$, basta con considerar el reescalamiento de los nodos para obtener las versiones modificadas de las reglas contenidas en la siguiente definición:

Definición 3.2 — Reglas de integración para $a \leq x \leq b$. La integral en el intervalo unitario $\int_0^1 f(x)dx$ puede ser aproximada utilizando las reglas de integración para polinomios. Estas reglas pueden resumirse como
Regla del punto medio

$$(b-a)f\left(\frac{a+b}{2}\right) \tag{3.11}$$

Regla del trapecio

$$\left(\frac{b-a}{2}\right)[f(a)+f(b)] \tag{3.12}$$

Regla de Simpson

$$\left(\frac{b-a}{6}\right)\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right] \tag{3.13}$$

Regla de Simpson de 3/8

$$+3f\left(a+2\frac{b-a}{3}\right)+f(b)\Bigg] \tag{3.14}$$

Regla de Milne

$$\left(\frac{b-a}{90}\right)\left[7f(a)+32f\left(a+\frac{b-a}{4}\right)+12f\left(a+\frac{b-a}{2}\right)+32f\left(a+3\frac{b-a}{4}\right)+7f(b)\right] \tag{3.15}$$

De manera más general, en muchas ocasiones se presentan estas reglas de integración utilizando la fórmula que las engloba y generaliza y que es conocida como la fórmula de Newton-Cotes[2] dada por

$$\int_a^b f(x)dx \approx (b-a)\sum_{j=0}^{n}\alpha_j^{(n)} f\left(a+j\frac{b-a}{n}\right) \tag{3.16}$$

y los valores de los $\alpha_j^{(n)}$ son precisamente los que se presentan en las ecuaciones (3.11)–(3.15) y que pueden generalizarse para $n > 4$.

Como es natural, las fórmulas se complican cuando n crece y, además para $n \geq 8$ aparecen signos positivos y negativos en los factores $\alpha_j^{(n)}$, lo que hace más probable la aparición de cancelaciones numéricas no deseadas. Es por esto que en la práctica se utilizan valores de n que comúnmente son las de las ecuaciones (3.11)-(3.15) o quizá la correspondiente a $n = 5$.

Para funciones más complejas, puede utilizarse la segunda idea que presentamos al inicio del capítulo y que corresponde a dividir el intervalo en varios trozos y realizar varias integrales. Veamos como podría hacerse esto usando la regla del punto medio:

Primero que nada, partiremos el intervalo $[a,b]$ en m subintervalos entre los nodos $a = x_1 < \ldots < x_{m+1} = b$ de longitud $h = \frac{b-a}{m}$ y usaremos la propoedad de las integrales que asegura que

$$\int_a^b f(x)dx = \sum_{j=1}^{m}\int_{a+(j-1)h}^{a+jh} f(x)dx. \tag{3.17}$$

[2]Roger Cotes (1682 – 1716). Matemático ingles que colaboró con Isaac Newton y es particularmente reconocido como uno de los principales revisores de la afamada obra Principia Mathematica de Isaac Newton.

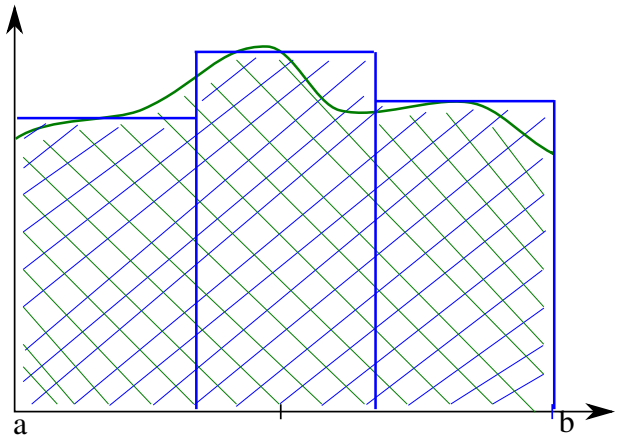


Figura 3.4: Integración aproximada mediante la regla de punto medio compuesta

Con esto no es difícil ver que se obtendrá una aproximación dada por la siguiente Regla del punto medio compuesta

$$\int_a^b f(x)dx \approx h \sum_{j=1}^{m} f\left((a+jh) - \frac{h}{2}\right). \tag{3.18}$$

Esta aproximación corresponde a tomar el histograma con m barras y alturas iguales a la evaluación de f en el punto medio de cada barra. La ilustración en la Figura 3.4 muestra un ejemplo de la aproximación que esta regla calcula utilizando tres particiones del intervalo ($m = 3$).

De manera análoga puede construirse la Regla del Trapecio Compuesta

$$\int_a^b f(x)dx \approx h \left[\frac{1}{2}f(a) + \sum_{j=1}^{m-1} f(a+jh) + \frac{1}{2}f(b)\right], \tag{3.19}$$

o también la regla usando polinomios cuadráticos presentada en la ecuación (3.10) que genera la Regla de Simpson compuesta

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_1) + 4\sum_{j=1}^{\frac{m}{2}} f(x_{2j}) + 2\sum_{j=1}^{\frac{m}{2}-1} f(x_{2j+1}) + f(x_{m+1})\right] \tag{3.20}$$

donde se ha modificado un poco el significado de la variable m, denotando nuevamente $x_1 = a$, $x_m = b$, $x_i = a + jh$ pero ahora la aproximación se realiza usando un polinomio cuadrático en cada intervalo de tamaño $2h$.

3.2 Cuadraturas de Gauss y errores de integración

Realizar integración numérica debe estar sujeto a cierto nivel de comprobación de que los resultados son buenos. Consideremos la integral $\int_0^1 f(x)dx$ y una cierta fórmula de integración en los nodos $0 = x_1 < x_2 < \cdots < x_n = 1$ con pesos $w_1, w_2, \cdots, w_n \in \mathbb{R}$ de la forma

$$Q(f) = \sum_{i=1}^{n} w_i f(x_i). \tag{3.21}$$

Estamos ahora interesados en la cantidad de error que se comete al calcular esta aproximación, para lo cual usaremos la siguiente definición:

Definición 3.3 — Orden de Integración. Dada una función en $[0,1]$ y una aproximación de $\int_0^1 f(x)dx$ a través de una cuadratura como la de la ecuación (3.21), el error de integración se define como

$$E(f) = \int_0^1 f(x) - Q(f) \equiv \int_0^1 f(x)dx - \sum_{i=1}^{n} w_i f(x_i). \tag{3.22}$$

Con esto, el orden de integración de la regla de cuadratura Q se define como el número m tal que se cumple

$$\begin{aligned} &E(p) = 0 \qquad \forall p \in P_{m-1}, \\ &E(p) \neq 0 \quad \text{para algún } p \in P_m. \end{aligned} \tag{3.23}$$

Esto significa que para una cuadratura de orden m, todas las funciones polinomiales de grado menor a m pueden ser aproximados de manera exacta (salvo errores menores de redondeo).

■ **Ejemplo 3.1 — La regla del trapecio.** $Q(f)$ está definida por los pesos $w_1 = w_2 = \frac{1}{2}$ y los nodos $\{0,1\}$. Veamos cual es el error de integración para esta fórmula de $Q(f)$:

Sea $p \in P_0$ (constante), entonces

$$\int_0^1 p(x)dx = c\,[x]\Big|_0^1 = c, \tag{3.24}$$

$$Q(p) = \sum_{i=1}^{2} w_i p(x_i) = \frac{1}{2}p(0) + \frac{1}{2}p(1) = c, \tag{3.25}$$

y por lo tanto $E(p) = 0$ para polinomios de grado cero. Continuando con un polinomio $p \in P_1$ se tiene que $p(x) = ax + b$ y entonces

$$\int_0^1 p(x)dx = a\left[\frac{x^2}{2}\right]_0^1 + b\,[x]_0^1 = \frac{1}{2}a + b, \tag{3.26}$$

$$Q(p) = \frac{1}{2}p(0) + \frac{1}{2}p(1) = \frac{1}{2}b + \frac{1}{2}(a+b) = \frac{1}{2}a + b, \tag{3.27}$$

y como ests dos expresiones son iguales, entonces $E(p) = 0$ también para polinomios de primer grado.

Consideremos ahora un polinomio $p \in P_2$, $p(x) = ax^2 + bx + c$, entonces tenemos para el cálculo de error de integración que

$$\begin{aligned} \int_0^1 p(x)dx &= \left[\frac{ax^3}{3} + \frac{bx^2}{2} + c\right]\Big|_0^1 = \frac{a}{3} + \frac{b}{2} + c, && (3.28) \\ Q(p) &= \frac{1}{2}p(0) + \frac{1}{2}p(1) = \frac{1}{2}c + \frac{1}{2}(a+b+c) = \frac{1}{2}a + \frac{b}{2} + c, && (3.29) \\ &\Rightarrow \quad E(p) = -\frac{1}{6}. \end{aligned}$$

Como conclusión, la regla del trapecio tiene orden 2. ∎

No es difícil ver que se puede utilizar la linealidad de la integración para descomponer este proceso en uno más simple en el que solo se verifique para los polinomios $x^j \in P_j$, $j \in \mathbb{N}_0$.

$$\begin{aligned} E(1) &= \int_0^1 1dx - \left(\frac{1}{2}(1) + \frac{1}{2}(1)\right) = 0 && (3.30) \\ E(x) &= \frac{1}{2}\left[x^2\right]\Big|_0^1 - \left(\frac{1}{2}(0) + \frac{1}{2}(1)\right) = \frac{1}{2} - \frac{1}{2} = 0 && (3.31) \\ E(x^2) &= \left[\frac{x^3}{3}\right]\Big|_0^1 - \left(\frac{1}{2}(0) + \frac{1}{2}(1)\right) = \frac{1}{3} - \frac{1}{2} = -\frac{1}{6} \end{aligned}$$

Es decir, que se tiene orden 2 para la regla del trapecio.

∎ **Ejemplo 3.2 — Regla de Simpson.** Es de esperarse que sea mejor que la regla del trapecio, por eso iniciamos con un polinomio de orden 2, para ver si la aproximación nos lleva a un nivel de error cero

$$E(x^2) = \int_0^1 x^2dx - \left[\frac{1}{6}(0)^2 + \frac{4}{6}\left(\frac{1}{2}\right)^2 + \frac{1}{6}(1)^2\right] = \frac{1}{3} - \left[\frac{1}{6} + \frac{1}{6}\right] = 0 \qquad (3.32)$$

es decir que el orden es al menos 3, veamos el siguiente grado polinomial

$$E(x^3) = \int_0^1 x^3dx - \left[\frac{1}{6}(0)^3 + \frac{4}{6}\left(\frac{1}{2}\right)^3 + \frac{1}{6}(1)^3\right] = \frac{1}{4} - \left[\frac{1}{12} + \frac{1}{6}\right] = \frac{1}{4} - \frac{3}{12} = 0 \quad (3.33)$$

entonces el orden es incluso mayor a 3 dado que para polinomios de tercer grado el resultado del error sigue siendo igual a cero. Considerando ahora el siguiente polinomio de grado 4 tenemos que

$$\begin{aligned} E(x^4) &= \int_0^1 x^4dx - \left[\frac{1}{6}(0)^4 + \frac{4}{6}\left(\frac{1}{2}\right)^4 + \frac{1}{6}(1)^4\right] = \frac{1}{5} - \left[\frac{1}{24} + \frac{1}{6}\right] \\ &= \frac{1}{5} - \frac{5}{24} \neq 0 && (3.34) \end{aligned}$$

por lo que podemos concluir que la regla de Simpson tiene orden 4. ∎

Observe el interesante resultado para la regla de Simpson en contraste con la regla de Simpson $\frac{3}{8}$, que utiliza en su construcción 4 nodos (uno más) pero que solo tiene orden 4. Este resultado sorprende un poco pues al agregar un nodo y usar 3 nodos (Simpson) se obtienen mejoras de 2 órdenes de aproximación, pero al agregar otro nodo y llegar a 4 (Simpson $\frac{3}{8}$) no hay ningún avance en la calidad de aproximación. Una explicación a esto podría ser que las aproximaciones tienen la limitante de nodos igualmente espaciados.

De acuerdo a los desarrollos de Gauss[3], basta ver que la cuadratura del punto medio puede derivarse de manera inversa, primero pidiendo que aproxime bien hasta orden 2 y después viendo si existe un peso w_1 y nodo x_1 tal que los polinomios en P_0 y P_1 son exactamente aproximados. Esto es

$$\int_0^1 x^0 dx = 1 \stackrel{!}{=} w_1 x_1^0, \tag{3.35}$$

$$\int_0^1 x^1 dx = \frac{1}{2} \stackrel{!}{=} w_1 x_1^1. \tag{3.36}$$

De la primera ecuación resulta que $w_1 = 1$ pues $x_1^0 = 1$. De la segunda ecuación resulta entonces $\frac{1}{2} = x_1$. Lo que es precisamente la regla del punto medio.

Similarmente, tenemos una integración con dos nodos (Trapecio) pero quizá podemos encontrar otra cuyo orden de aproximación sea mayor a la obtenida hasta ahora que es 2. Se necesita encontrar los valores de w_1,w_2 como pesos y de x_1, x_2 como nodos y esperamos que se obtenga un orden al menos igual a 3 para así mejorar el método conocido. Entonces debemos resolver el siguiente problema

$$\int_0^1 x^0 dx = 1 \stackrel{!}{=} w_1 x_1^0 + w_2 x_2^0, \tag{3.37}$$

$$\int_0^1 x^1 dx = \frac{1}{2} \stackrel{!}{=} w_1 x_1^1 + w_2 x_2^1, \tag{3.38}$$

$$\int_0^1 x^2 dx = \frac{1}{3} \stackrel{!}{=} w_1 x_1^2 + w_2 x_2^2, \tag{3.39}$$

$$\int_0^1 x^3 dx = \frac{1}{4} \stackrel{!}{=} w_1 x_1^3 + w_2 x_2^3, \tag{3.40}$$

el cual tiene 4 ecuaciones y 4 incógnitas. No es sencillo de resolver pero después de

[3] Atribuidos a Karl Friedrich Gauss (1777-1855). Matemático Alemán conocido como el "príncipe de las matemáticas" y reconocido por muchos como el matemático más prolífico después de los precursores de la antiguedad griega.

diversos desarrollos algebráicos puede verse que la solución está dada por

$$w_1 = \tfrac{1}{2}, \tag{3.41}$$
$$w_2 = \tfrac{1}{2}, \tag{3.42}$$
$$x_1 = \tfrac{1}{2}\left(1 - \tfrac{1}{\sqrt{3}}\right), \tag{3.43}$$
$$x_2 = \tfrac{1}{2}\left(1 + \tfrac{1}{\sqrt{3}}\right). \tag{3.44}$$

Con estos valores puede verse también que

$$\begin{aligned}\int_0^1 x^4 dx = \frac{1}{5} \neq w_1 x_1^4 + w_2 x_2^4 &= \frac{1}{2}\frac{28 - 4\sqrt{3}}{16 \cdot 9} + \frac{1}{2}\frac{28 + 4\sqrt{3}}{16 \cdot 9} \\ &= \frac{56}{32 \cdot 9} = \frac{7}{36}\end{aligned}$$

por lo que el orden de la cuadratura (3.41)–(3.44) es exactamente 4.

En general puede demostrarse que usando cuadraturas de Gauss puede obtenerse el mejor orden del error, siendo éste igual al doble del número de nodos usados para la integración. Los desarrollos generales hacen uso de la base de polinomios de Legendre[4] normalizados para obtener los nodos y lo pesos adecuados para la integración.

La siguiente tabla compara los métodos de integración, incluyendo la cuadratura de Gauβ con 3 puntos y con los nodos/pesos dados por

$$w_1 = \frac{5}{18} \qquad x_1 = \frac{1}{2}\left(1 - \sqrt{\frac{3}{5}}\right), \tag{3.45}$$
$$w_2 = \frac{8}{18} \qquad x_2 = \frac{1}{2}, \tag{3.46}$$
$$w_3 = \frac{5}{18} \qquad x_3 = \frac{1}{2}\left(1 + \sqrt{\frac{3}{5}}\right). \tag{3.47}$$

# Nodos	Nombre	Orden
1	Punto Medio / Gauβ-1	2
2	Trapecio	2
2	Gauβ-2	4
3	Simpson	4
3	Gauβ-3	6
4	Simpson $\frac{3}{8}$	4
4	Gauβ- 4	8

[4]Adrien- Marie Legendre (1752 – 1833) . Matemático francés conocido por sus contribuciones en funciones elípticas y álgebra realizados principalmente en París.

Dado que la teoría detrás de las cuadraturas de Gauβ es desarrollada para mayor simplicidad en el intervalo $[-1,1]$, los pesos w_i y nodos x_i son comunmente definidos en este intervalo. Para 1, 2 y 3 nodos los valores son

$$n=1 \qquad w_1=2 \qquad x_1=0 \tag{3.48}$$

$$\begin{aligned} n=2 \qquad w_1&=1 \qquad x_1=-\sqrt{\frac{1}{3}} \\ w_2&=1 \qquad x_2=+\sqrt{\frac{1}{3}} \end{aligned} \tag{3.49}$$

$$\begin{aligned} n=3 \qquad w_1&=\tfrac{5}{9} \qquad x_1=-\sqrt{\frac{3}{5}} \\ w_2&=\tfrac{8}{9} \qquad x_2=0 \\ w_3&=\tfrac{5}{9} \qquad x_3=+\sqrt{\frac{3}{5}} \end{aligned} \tag{3.50}$$

En la práctica, debe transformarse una integral $\int_a^b f(x)dx$ mediante el cambio de variable

$$x = \frac{b-a}{2}z+\frac{a+b}{2} \tag{3.51}$$

$$dx = \frac{b-a}{2}dz \tag{3.52}$$

que transforma $[a,b]$ en $[-1,1]$ y luego realizar la integral

$$\begin{aligned} \int_a^b f(x)dx &= \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}z+\frac{a+b}{2}\right)dz \\ &\approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}z_i+\frac{a+b}{2}\right) \end{aligned}$$

Donde los z_i son los nodos derivados para las cuadraturas de Gauβ correspondientes.

3.3 Ejercicios

Ejercicio 3.1. Regla de Simpson 3/8 (10 Puntos)
Realice los desarrollos para encontrar la fórmula de la Regla de Simpson 3/8 para aproximaciones a la integral con polinomios en $\mathbb{P}_3$. ∎

Ejercicio 3.2. Regla de Milne (10 Puntos)
Realice los desarrollos para encontrar la fórmula de la Regla de Milne para aproxi-

maciones a la integral con polinomios en $\mathbb{P}_4$. ■

Ejercicio 3.3. Implementación de aproximaciones de orden 0 y 1 (15 Puntos)
Implemente un programa que aproxime $\int_a^b f(x)dx$ utitlizando las aproximaciones polinomiales en $\mathbb{P}_0$, y $\mathbb{P}_1$ a través de las reglas del punto medio y del trapecio. ■

Ejercicio 3.4. Implementación de aproximaciones de orden 2, 3 y 4 (20 Puntos)
Similar al programa del ejercicio anterior, implemente un programa que aproxime $\int_a^b f(x)dx$ utitlizando las aproximaciones polinomiales en $\mathbb{P}_2$, $\mathbb{P}_3$ y $\mathbb{P}_4$ a través de las reglas de Simpson, de Simpson 3/8 y de Milne. ■

Ejercicio 3.5. Cálculo de las integrales numéricas (25 Puntos)
Use el programa del ejercicio anterior para calcular las siguientes integrales:

(a) $\int_1^5 (3x-4)dx$

(b) $\int_{-2}^1 (x^2-2x+3)dx$

(c) $\int_7^9 (x^3-5x^2+2x)dx$

(d) $\int_0^2 (2x^4-x^2+7)dx$

(e) $\int_0^3 \sin(\pi x)dx$

(f) $\int_0^{15} 6(x+1)e^{-3(x+1)^2}dx$

(g) $\int_0^1 [x+\sin(6x)/8]\,dx$

Calcule las integrales exactas analíticamente y compare los errores relativos en una tabla. Para la función en el último inciso, grafique la función junto con las aproximaciones en $\mathbb{P}_1$, $\mathbb{P}_2$, $\mathbb{P}_3$ y $\mathbb{P}_4$. ■

Ejercicio 3.6. Reglas compuestas (20 Puntos)
Implemente las reglas compuestas del trapecio y de Simpson y evalúe las integrales correspondientes a los ejercicios 21(f) y 21(g) utilizando $2,3,4\ldots$ subintervalos. Analice los resultados. ■

Ejercicio 3.7. Cuadratura de Gauss (25 Puntos)
Implemente la integración numérica por medio de cuadraturas de Gauss y compare

los resultados de integrar con tres nodos usando puntos igualmente espaciados (Regla de Simpson) y puntos espaciados según los cálculos de Gauss.
Considere las integrales de los incisos 21(c), 21(d) y 21(f) y compare los errores relativos[a] en el cálculo de cada una de estas integrales. ■

[a] Igual al valor absoluto de la resta entre el valor exacto y el aproximado, dividido por el valor exacto.

4. Solución de sistemas lineales

4.1 Sistemas de ecuaciones lineales

La resolución de sistemas lineales es quizá el problema de análisis numérico más frecuente en el cómputo científico ya sea para problemas teóricos o aplicados. Desde una categorización matemática, las áreas en las que se requiere resolver sistemas lineales son optimización, problemas inversos, ecuaciones diferenciales parciales y ordinarias, estadística, control, etc.

La amplia popularidad de los sistemas lineales se debe a que resultan de muchas aproximaciones a sistemas de ecuaciones de todos los órdenes (incluso no lineales). Por ejemplo, algunos métodos de optimización se basan en aproximar un funcional arbitrario mediante una función cuadrática de la forma $f(x) = \frac{1}{2}x^T Ax - x^T b$ que se asemeja al funcional en una cierta vecindad. La solución de minimizar el funcional requiere resolver $f'(x) = 0$ o bien $Ax - b = 0$.

Otro ejemplo son los métodos en diferencias finitas donde una malla para resolver una ecuación diferencial genera un sistema lineal de ecuaciones al considerar aproximaciones $\frac{\partial f}{\partial x} \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$ y que se ve traducido en una serie de ecuaciones donde los $f(x_i)$ son las incógnitas.

En muchas aplicaciones, los sistemas lineales que se obtienen pueden tener una estructura bien definida, o pueden también ser sistemas con "pocos" elementos distintos de cero en la matriz del sistema. Las matrices con pocos elementos distintos de cero son llamadas ralas, dispersas o simplemente *sparse*. La definición de pocos se refiere a que es una cantidad que permite usar técnicas de almacenamiento y manipulación de la matriz que sean más eficientes que las tradicionales. Típicamente, una matriz *sparse*

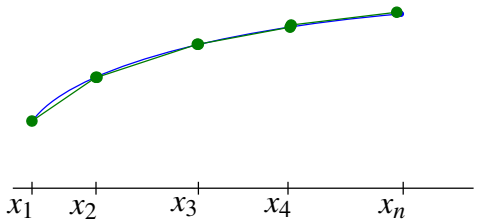


Figura 4.1: Aproximaciones por medio de funciones

contiene a lo mucho un 1 % de elementos distintos de cero.

Por otra parte, existen matrices con una estructura adecuada para ser tratadas de manera más sencilla. Los ejemplos más comunes de estos son:

- Diagonal : Representada con el símbolo $\begin{pmatrix} \diagdown \end{pmatrix}$ indicando que solo hay elementos $a_{ij} \neq 0$ en A y que a_{ij} para $i \neq j$ usualmente se denota como D.
- Bidiagonal superior : Símbolo $\begin{pmatrix} \diagdown \\ \diagdown \end{pmatrix}$
- Bidiagonal inferior : Símbolo $\begin{pmatrix} \diagdown \\ \diagdown \end{pmatrix}$
- Tridiagonal : Símbolo $\begin{pmatrix} \diagdown \\ \diagdown \\ \diagdown \end{pmatrix}$
- Triangular inferior $\begin{pmatrix} \diagdown \\ \diagdown \end{pmatrix}$, usualmente denotada L (lower)
- Triangular superior $\begin{pmatrix} \diagdown \\ \diagdown \end{pmatrix}$, usualmente denotada U (upper)
- Hessenberg $\begin{pmatrix} \diagdown \\ \diagdown \end{pmatrix}$

Esta claro que un sistema que se quiere resolver con una matriz diagonal , i.e, $Dx = b$ es un problema trivial de resolver pues cada ecuación es de la forma

$$a_{i,i}x_i = d_i x_i = b_i \qquad \Rightarrow x_i = \frac{b_i}{d_i} \qquad \forall i = 1,\ldots,n. \tag{4.1}$$

Para una matriz bidiagonal superior puede verse que la última ecuación tendrá la forma

$$a_{n,n}x_n = b_n \qquad \Rightarrow x_n = \frac{b_n}{a_{n,n}}, \tag{4.2}$$

y a partir de ahí todas las ecuaciones (ordenadas hacia atrás) pueden ser resueltas como

$$a_{i,i}x_i + a_{i,i+1}x_{i+1} = b_i \qquad \Rightarrow x_i = \frac{b_i - a_{i,i+1}x_{i+1}}{a_{i,i}} \qquad \forall i = n-1,\ldots,1. \tag{4.3}$$

De manera análoga, para una matriz bidiagonal inferior se ve que

$$a_{1,1}x_1 = b_1 \qquad \Rightarrow x_1 = \frac{b_1}{a_{1,1}} \tag{4.4}$$

y el resto de las ecuaciones se resuelve como

$$a_{i-1,i}x_{i-1} + a_{i,i}x_i = b_i \qquad \Rightarrow x_i = \frac{b_i - a_{i-1,i}x_{i-1}}{a_{i,i}} \qquad \forall i = 2,\cdots,n, \tag{4.5}$$

Para matrices tridiagonales existe un algoritmo simple conocido como Algoritmo de Thomas (ver ejercicio 4.1).

4.2 Factorización LU y su uso para resolver sistemas lineales

En general, estamos interesados en resolver un sistema lineal

$$Ax = b \tag{4.6}$$

con $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $x \in \mathbb{R}^n$ sin ninguna información adicional acerca de la estructura de A.

Para diseñar nuestro primer método de solución de (4.6) nos basaremos en el método para resolver sistemas triangulares superiores e inferiores dado que si pudiéramos expresar a la matriz A como la multiplicación de una matriz L y una U se tendría que (4.6) puede resolverse en dos pasos, resolviendo

$$Ly = b, \quad L \in \mathbb{R}^{nxn}, \left(\begin{array}{ll} \diagdown & \\ & \diagdown \end{array}\right), \quad y \in \mathbb{R}^n \tag{4.7}$$

$$Ux = y, \quad U \in \mathbb{R}^{nxn}, \left(\begin{array}{ll} \diagdown & \\ & \diagdown \end{array}\right). \tag{4.8}$$

Consideremos el sistema (4.7) . La i-ésima ecuación del mismo es

$$\ell_{i,1}y_1 + \ell_{i,2}y_2 + \cdots + \ell_{i,i}y_i = b_i, \tag{4.9}$$

o equivalentemente

$$y_i = \frac{1}{\ell_{i,i}}\left(b_i - \sum_{j=1}^{i-1} \ell_{i,j}y_j\right) \tag{4.10}$$

y en el caso de la primer ecuación se tiene que

$$y_1 = \frac{b_1}{\ell_{1,1}} \tag{4.11}$$

Usando (4.10) y (4.11) un algoritmo para resolver el sistema triangular inferior (4.7) es:

```
y(1)= b(1)/ L(1,1);
for i = 2:n
   y(i) = (b(i)-L(i,1:(i-1)) * y(1:(i-1)) / L(i,i):
end
```

El cálculo de la i-ésima componente de y requiere $2i-1$ operaciones de punto flotante, por lo que el total de operaciones requeridas son

$$\sum_{i=1}^{n}(2i-1) = 2\sum_{i=1}^{n} i - n \tag{4.12}$$

$$= 2\frac{n(n+1)}{2} - n \tag{4.13}$$

$$= n^2 + n - n = n^2 \tag{4.14}$$

Análogamente, para resolver el sistema (4.8) puede hacerse una sustitución similar haciendo uso de la i-ésima ecuación como

$$u_{i,i}x_i + u_{i,i+1}x_{i+1} + \cdots + u_{i,n}x_n = y_i, \tag{4.15}$$

o equivalentemente

$$x_i = \frac{1}{u_{i,i}}\left(y_i - \sum_{j=i+1}^{n} u_{i,j}x_j\right), \tag{4.16}$$

y la n-ésima ecuación como

$$x_n = \frac{y_n}{u_{n,n}}. \tag{4.17}$$

El algoritmo correspondiente tiene también n^2 operaciones y se ve como

```
x(n) = y(n)/u(n,n);
for i= n-1:-1:1
   x(i)= (y(i)-u(i,(i+1):n) * x((i+1):n))/u(i,i);
end
```

Lo que nos falta es ver cómo convertir A en su factorización LU como

$$A = LU \tag{4.18}$$

Por esto, consideraremos el proceso de llevar la matriz por bloques formada por la identidad, y a la matriz A a una forma LU, iniciando con la matriz IA y llevándola a LU a través de operaciones elementales de fila:

Usando la operación $Fila_2 - \frac{a_{21}}{a_{11}} Fila_1$:

$$\left(\begin{array}{ccc|cccc} 1 & & & a_{11} & a_{12} & \cdots & a_{1n} \\ & 1 & & a_{21} & a_{22} & \cdots & a_{2n} \\ & & \ddots & \vdots & \vdots & \ddots & \\ & & 1 & a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}\right) \longrightarrow \left(\begin{array}{ccc|cccc} 1 & 0 & & a_{11} & a_{12} & \cdots & a_{1n} \\ -\frac{a_{21}}{a_{11}} & 1 & & 0 & \times & \cdots & \times \\ & & \ddots & \vdots & \vdots & \ddots & \\ & & 1 & a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}\right)$$

Continuando de esta forma hasta completar todos los ceros debajo de a_{11} se obtiene algo de la forma (usando $Fila_i - \frac{a_{i1}}{a_{11}} Fila_{i-1}$)

$$\left(\begin{array}{ccc|cccc} 1 & 0 & & a_{11} & a_{12} & \cdots & a_{1n} \\ -\frac{a_{21}}{a_{11}} & 1 & & 0 & \times & \cdots & \times \\ \vdots & & \ddots & \vdots & \vdots & \ddots & \\ -\frac{a_{n1}}{a_{11}} & & 1 & 0 & \times & \cdots & \times \end{array}\right)$$

Esta parte podría ponerse en un ciclo de un algoritmo para ir reemplazando A hasta convertirse en U y utilizando solamente la memoria donde se guardan L y A como sigue:

```
k=1 ; L=eye(n);
for i=(k+1):n
   L(i,k)= - A(i,k) / A(k,k);
   A(i,k)= 0;
   A(i,(k+1):n)= A(i,(k+1):n) + L(i,k) * A(k,(k+1):n);
end
```

En MATLAB podemos incluso eliminar este ciclo y hacer todas las filas a la vez a través de los comandos

```
k=1 ; L=eye(n);
L((k+1):n,k)= - A((k+1):n)/A(k,k);
A((k+1):n,k)=zeros(n-k,1);
A((k+1):n, (k+1):n)= A((k+1):n,(k+1):n)+ L((k+1):n,k) * A(k,(k+1):n);
```

Con esto, la primeras columna de L y la primera columna de U pueden calcularse utilizando las siguientes operaciones de punto flotante:

$$\underbrace{(n-k)}_{\text{sumas}} + \underbrace{(n-k)}_{\text{diferentes } i} \underbrace{[(n-k)+(n-k)]}_{\text{cálculo fila } i} = (n-k)[1+2(n-k)] \tag{4.19}$$

y la factorización completa requiere realizar estas operaciones también para las columnas $2, 3, \cdots, (n-1)$

```
L= eye(n);
for k=1:(n-1)
   L((k+1):n,k)= - A((k+1):n)/A(k,k);
   A((k+1):n,k)=zeros(n-k,1);
   A((k+1):n, (k+1):n)= A((k+1):n,(k+1):n)+ L((k+1):n,k) * A(k,(k+1):n);
end
U=A;
```

Es posible mostrar que el número de operaciones en este algoritmo esta definido por el factor $\frac{2}{3}n^3$.

El único problema potencial que puede presentar este algoritmo es que la división realizada para obtener L podría resultar en números muy grandes si hay diferencias grandes de orden entre los elementos de $A(k+1:n,k)$ y el elemento $A(k,k)$. Para solucionar esto, podemos intercambiar (pivotear) para tomar la fila en la que se obtenga una mejor relación a través de usar en la k-ésima fila la que cuente con un $A(k,k)$ mayor, de manera que al dividir, los factores siempre sean menores a 1.

■ **Ejemplo 4.1 — Pivoteo.** En el 2^{do} paso:

$$\left(\begin{array}{ccccc|ccccc} 1 & 0 & & & & \times & \times & \cdots & \times \\ \times & 1 & & & & 0 & 7x10^{-3} & \cdots & \times \\ \times & 0 & 1 & & & 0 & 9 & \cdots & \times \\ \vdots & \vdots & & \ddots & & \vdots & \vdots & \ddots & \\ \times & 0 & & & 1 & 0 & 2 & \cdots & \times \end{array}\right)$$

Observe que las Las operaciones que se hacen cuando $k = 2$ solo afectan una parte de la matrix. Si se combinan fila de la matriz no afectaría, mientras se tenga un modo de "recordar" qué filas fueron cambiadas. ■

Si las filas de A son intercambiadas y x resuelve el sistema $Ax = b$, entonces la factorización de A en realidad factoriza a la matriz permutada como $PA = LU$, pero adicionalmente si $Ax = b$, entonces $PAx = Pb$, por lo que las matrices de la factorización pueden usarse para resolver $LUx = Pb$ y de este modo resolver en dos pasos como se hizo ya en (4.7) y (4.8) definiendo las ecuaciones

$$Ly = Pb, \tag{4.20}$$
$$UX = y. \tag{4.21}$$

Una manera fácil de guardar la información de filas permutadas, es definir un vector $p = [1,\cdots,n]$ y realizar las permutaciones en sus elementos para usarlos como índices de Pb. Por ejemplo si las únicas filas intercambiadas fueran la fila 2 y la fila n, el vector $p = [1,n,3,4,\cdots,n-1,2]$ y usar $b(p)$ equivale a tomar Pb.

Implementando esta idea en nuestro algoritmo nos lleva a:

```
[m,n]=size(A);
if m~=n
   error('La matriz A no es cuadrada');
end
piv=(1:n)';
L=eye(n);
for k=1:n-1
   [maxv,r]=max(abs(A(k:n,k)));
   q = r+k-1;
   p((k,q))= p([q,k]);
   A([k,q],:)=A([q,k],:);
   if abs(A(k,k))<1e-16 %(tolerancia)
      error('Matriz de rango deficiente');
   else
      L((k+1):n,k)=A((k+1):n,k)/A(k,k);
      A((k+1):n,(k+1):n)=A((k+1):n,(k+1):n)-L((k+1):n,k)*A(k,(k+1):n);
      A((k+1):n,k)=zeros(n-k,1);
   end (else)
end (for)
U=A;
```

Como ya se mencionó en la ecuación (4.14), la solución de las ecuaciones para los sistemas triangulares con L o con U requieren n^2 operaciones cada uno, por lo que queda claro que la parte dominante en la solución del sistema $PAx = Pb$ a través de (4.20) y (4.21) corresponde a las $\frac{2}{3}n^3$ de la factorización LU.

La ventaja de resolver usando la factorización LU radica en que la solución de múltiples problemas requiere que la factorización se realice una sola vez en problemas múltiples. Por ejemplo, si se tienen que resolver los w sistemas de ecuaciones lineales de la forma

$$\begin{aligned} Ax_1 &= b_1, && (4.22)\\ Ax_2 &= b_2, && (4.23)\\ &\vdots \\ Ax_w &= b_w, \qquad A \in \mathbb{R}^{nxn},\ x_1,\ldots,x_w,b_1,\ldots,b_w \in \mathbb{R}^n,\ w \in \mathbb{N} && (4.24) \end{aligned}$$

puede calcularse realizndo:

- una factorización $LU = A$, es decir $\frac{2}{3}n^3$ operaciones de punto flotante;
- w soluciones de sistemas de la forma $Ly = Pb$, es decir wn^2; y
- w soluciones de sistemas de la forma $Ux = y$, es decir wn^2.

Esto hace un total de $\frac{2}{3}n^3 + 2wn^2$ operaciones de punto flotante.

Con la finalidad de comparar, observe que el proceso Gram-Schmidt en una matriz $n \times n$ requiere $2n^3$ operaciones que deben ser tomadas para cada sistema de las w posibilidades dadas, es decir que se usaría n total de $2wn^3$. Claramente, realizar la

solución usando la factorización es más eficiente para sistemas donde el valor de n es grande como es el caso de muchos probelmas aplicados.

4.3 Factorización QR

La solución de sistemas sobredeterminados donde la matriz contiene más filas que columnas tiene amplios usos en aplicaciones (problemas de mínimos cuadrados, calculo de rangos de matrices, problemas de valores propios, etc). Es de esperarse que para resolver $Ax = b$ con $A \in \mathbb{R}^{m\times n}$ y $m > n$ no sea posible encontrar un vector $x \in \mathbb{R}^n$ que solucione el sistema lineal, por lo que deberemos aceptar una solución en la que Ax se aproxime suficientemente a b. Tal es el caso de los problemas de mínimos cuadrados en los que se intenta minimizar la distancia entre estos dos vectores usando la norma 2 en $\mathbb{R}^m$. Por ejemplo, suponga que se quiere resolver el siguiente sistema:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \tag{4.25}$$

Para que este sistema tenga solución es necesario que el vector b sea combinación lineal de la forma

$$b = x_1 \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} + x_2 \begin{pmatrix} a_{12} \\ a_{22} \\ a_{32} \end{pmatrix}. \tag{4.26}$$

Es decir que $Ax = b$ solo puede resolverse para $b \in span\{\overline{a_1}, \overline{a_2}\}$. Por esta razón, los sistemas sobredeterminados se resuelven bajo la idea de minimizar los errores inherentes al problema resolviendo

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|_2. \tag{4.27}$$

Resolver el problema de minimización en (4.27) no es sencillo y por eso se busca una manera de simplificar la estructura de la matriz del sistema para intercambiarla por una que tenga una forma más fácil de resolverse.

Si podemos encontrar una matriz M tal que el sistema $(MA)x = (Mb)$ sea fácil de resolver entonces será mejor utilizar esta segunda forma, tal como lo hicimos al simplificar una resolución con A llena en dos sistemas con L y U en la sección anterior. La única restricción que debemos cuidar es que toda modificación a las filas de A implicará una modificación a las filas de b.

La manera más efectiva de realizar esto es utilizando transformaciones ortogonales a través de una matriz Q ortogonal, es decir que cumple que $QQ^T = Q^TQ = I$. Dicho de otra forma, las columnas de Q son ortogonales entre sí y además tienen norma unitaria.

Un buen ejemplo de este tipo de matrices en $\mathbb{R}^{2x2}$ son las rotaciones

$$Q = \begin{pmatrix} cos\alpha & sin\alpha \\ -sin\alpha & cos\alpha \end{pmatrix} \tag{4.28}$$

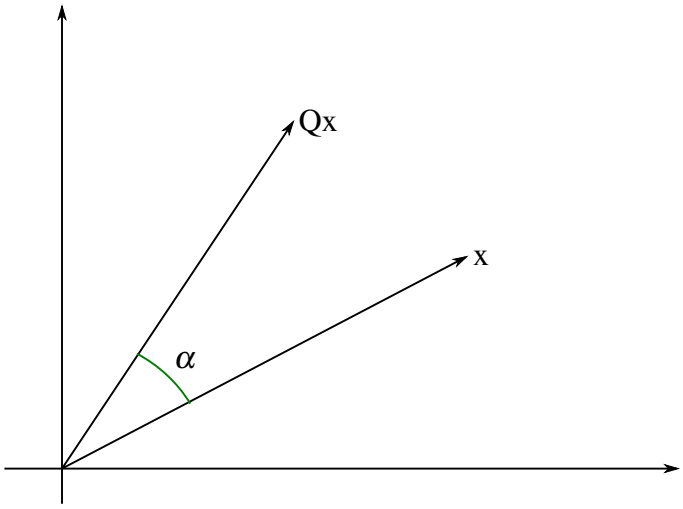


Figura 4.2: Rotación del vector x con un ángulo α

que al ser aplicadas a cualquier punto (vector) en $\mathbb{R}^2$ genera un vector rotado en un ángulo α y que mantiene su norma, es decir $\|x\|_2 = \|Qx\|_2$.

El problema (4.27) puede ser modificado si se pre-multiplica Ax y b por una matriz $Q \in \mathbb{R}^{mxm}$ que cumpla precisamente las características de preservar la norma. Lo que usaremos es una generalización de la matriz de rotación en (4.28) que podrá ser formada a través de una serie de rotaciones como

$$Q^T = G_t \cdots G_1 \tag{4.29}$$

tal que al aplicar la rotación a A se obtenga una matriz triangular superior R como

$$Q^T A = R \tag{4.30}$$

o bien

$$A = IA = (QQ^T)A = Q(Q^T A) = QR \tag{4.31}$$

y las matrices G_j de (4.29) serán tomadas como rotaciones planas (de Givens) para producir ceros en la matriz A de manera que $Q^T A$ sea triangular superior. Una rotación

plana está definida en $\mathbb{R}^m$ como la matriz

$$G(i,k,\alpha) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & cos\alpha & \cdots & sin\alpha & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -sin\alpha & \cdots & cos\alpha & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ \leftarrow \text{Fila } i \\ \\ \leftarrow \text{Fila } j \\ \\ \\ \end{matrix} \quad (4.32)$$

$$\uparrow \text{col } i \qquad \uparrow \text{col } j$$

y rota el espacio $\mathbb{R}^n$ en un ángulo α con respecto al plano cartesiano definido por los ejes i y j (es decir, con eje de rotación perpendicular al plano $i-j$). En particular, para

generar la matriz R a partir de A hará falta "producirçeros en los elementos
$\underline{A_{m,1}, A_{m-1,1}, \cdots, A_{2,1},}$
$\underline{A_{m,2},, \cdots, A_{3,2},}$
$\underline{A_{m,3}, \cdots, A_{4,3},}$
$\vdots$
$\underline{A_{m,n}, A_{m-1,n}, \cdots, A_{n+1,n},}$

Supongamos para ilustrar el proceso que $m = 4$, $n = 3$, entonces la construcción de ceros se vería como

$$\begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \overset{G_1}{\longrightarrow} \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \\ 0 & x & x \end{bmatrix} \overset{G_2}{\longrightarrow} \begin{bmatrix} x & x & x \\ x & x & x \\ 0 & x & x \\ 0 & x & x \end{bmatrix} \overset{G_3}{\longrightarrow} \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & x & x \\ 0 & x & x \end{bmatrix}$$

$$\overset{G_4}{\longrightarrow} \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & x & x \\ 0 & 0 & x \end{bmatrix} \overset{G_5}{\longrightarrow} \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & 0 & x \\ 0 & 0 & x \end{bmatrix} \overset{G_6}{\longrightarrow} \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & 0 & x \\ 0 & 0 & 0 \end{bmatrix} \quad (4.33)$$

de manera que $Q^T = G_6 G_5 G_4 G_3 G_2 G_1$ y la labor de G_1 será convertir a la primera columna de A en un vector con el último elemento igual a cero.

Esto puede efectuarse mediante una rotación en el plano definido por la penúltima y la última coordenada en el que el vector $\begin{pmatrix} a_3 \\ a_4 \end{pmatrix}$ será transformado en un vector de la forma $\begin{pmatrix} x \\ 0 \end{pmatrix}$. En otras palabras, lo que necesítamos es una rotación de la forma

$$G_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & cos\alpha & sin\alpha \\ 0 & 0 & -sin\alpha & cos\alpha \end{bmatrix}$$

que dejará intactas las primeras 2 filas de A y modificará las últimas dos filas para generar un 0 en la posición $(4,3)$. Dado que A es una matriz fija, puede verse que

$$G_1 A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ x & x & x \\ 0 & x & x \end{bmatrix}$$

siempre y cuando se cumpla que

$$(-sin\alpha)a_{31} + (cos\alpha)a_{41} = 0 \tag{4.34}$$

Sabemos ademas que debe cumplirse que $(sin\alpha)^2 + (cos\alpha)^2 = 1$. De 4.34 podemos ver que $sin\alpha = cos\alpha \frac{a_{41}}{a_{31}}$

$$\Rightarrow (cos\alpha)^2 \left(\frac{a_{41}}{a_{31}}\right)^2 + (cos\alpha)^2 = 1$$
$$\Rightarrow (cos\alpha)^2 \left(\left(\frac{a_{41}}{a_{31}}\right)^2 + 1\right) = 1$$
$$\Rightarrow (cos\alpha)^2 = \frac{a_{31}^2}{a_{31}^2 + a_{41}^2}$$
$$\Rightarrow (cos\alpha) = \frac{a_{31}}{\sqrt{a_{31}^2 + a_{41}^2}}$$

y además

$$\Rightarrow (sin\alpha)^2 = 1 - \left(\frac{a_{31}^2}{a_{31}^2 + a_{41}^2}\right) = \frac{a_{41}}{\sqrt{a_{31}^2 + a_{41}^2}}$$
$$\Rightarrow sin\alpha = \frac{a_{41}}{\sqrt{a_{31}^2 + a_{41}^2}}$$

por lo que la primer rotación es

$$G_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{a_{31}}{\sqrt{}} & \frac{a_{41}}{\sqrt{}} \\ 0 & 0 & -\frac{a_{41}}{\sqrt{}} & \frac{a_{31}}{\sqrt{}} \end{bmatrix}$$

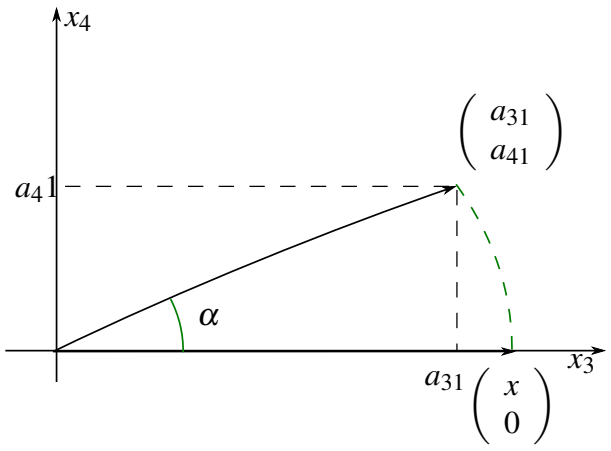


Figura 4.3: Rotación de ejes 3 y 4

Si dibujáramos la rotación en el plano de los ejes 3 y 4, la rotación del vector se vería como en el dibujo 4.3 y además podemos ver geométricamente que con los valores de a_{31} y a_{41} se puede definir directamente $tan\alpha$ o $cot\alpha$. Por esto, preferiremos una definición de G en términos de la cotangente, usando el siguiente resultado de equivalencia:

$$sin\alpha = \sqrt{\frac{(sin\alpha)^2}{1}} = \sqrt{\frac{(sin\alpha)^2}{sin^2\alpha + cos^2\alpha}} = \frac{1}{\sqrt{\frac{sin^2\alpha+cos^2\alpha}{sin^2\alpha}}}$$

$$= \frac{1}{\sqrt{1+\frac{cos^2\alpha}{sin^2\alpha}}} = \frac{1}{\sqrt{1+cot^2\alpha}} \ ,$$

$$cos\alpha = (cot\alpha)(sin\alpha) \tag{4.35}$$

o análogamente

$$cos\alpha = \frac{1}{\sqrt{1+tan^2\alpha}} \ , \ sin\alpha = (tan\alpha)(cos\alpha) \tag{4.36}$$

Para una rotación como la necesaria, basta un ángulo α en $[-90°, 90°]$ o $[-\frac{\pi}{2}, \frac{\pi}{2}]$. De hecho para valores de $\alpha = 0$ se debería tener que $a_{41} = 0$, por lo que la rotación será trivial con $cos\alpha = 1$, $sin\alpha = 0$ y $G = I$.

Las opciones para escoger entre la versión con tangente y cotangente se obtienen a partir de la observación de que la función tangente tiende a $\pm\infty$ cuando el ángulo se acerca a $\pm\frac{\pi}{2}$. Por otro lado, la función cotangente tiende a $\pm\infty$ cuando el ángulo se acerca a cero por la derecha o por la izquierda. (figura 4.4).

Por esta razón usaremos la rotación usando las fórmulas según (4.35) cuando $\alpha \in [-\frac{\pi}{2}, -\frac{\pi}{4}] \cup [\frac{\pi}{4}, \frac{\pi}{2}]$ y las fórmulas según (4.36) cuando $\alpha \in (-\frac{\pi}{4}, 0) \cup (0, \frac{\pi}{4})$. Obser-

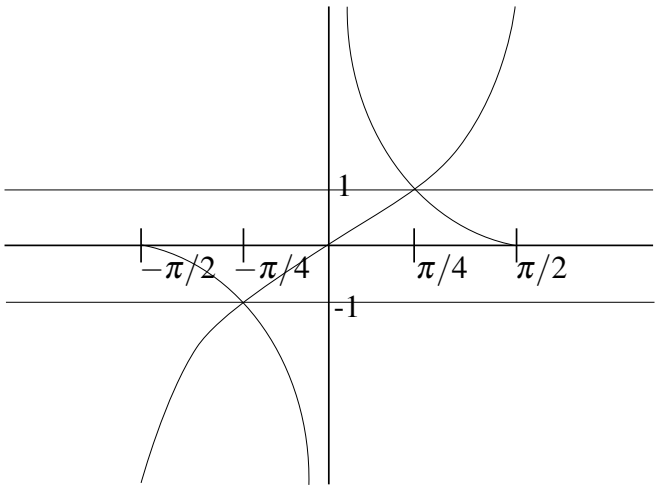


Figura 4.4: Función tangente y cotangente entre $[-\frac{\pi}{2}, \frac{\pi}{2}]$

vando que estos dos casos corresponden directamente a los casos en que $|a_{41}| \geq |a_{31}|$ y $|a_{41}| < |a_{31}|$ podemos definir nuestro cálculo utilizando esta comprobación para escoger la forma de calcular la rotación.

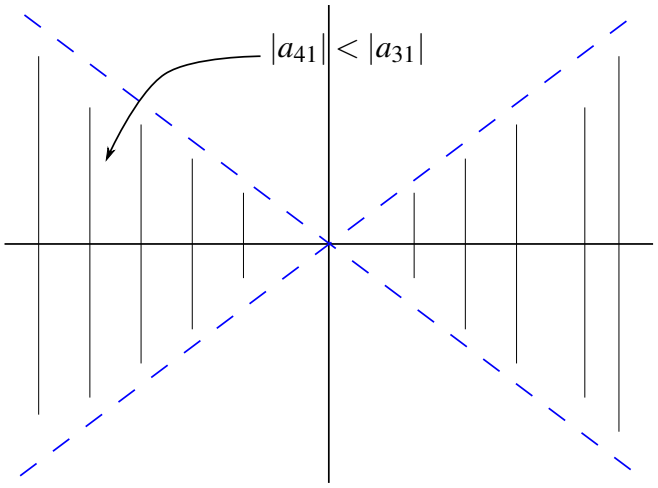


Figura 4.5: Rotación de ejes

Además es importante observar que las formulaciones anteriores aplican a cualquier par de ejes x_i , x_j rotando sobre el plano generado por éstos y por lo tanto, generar una rotación que produzca un cero en el componente j de un sistema al rotar en un plano i, j puede realizarse mediante una rotación como la formulada en (4.32) con los valores de

$cos\alpha$ y $sin\alpha$ tomados de las formulaciones (4.35) y (4.36) según sea el caso definido por los valores en las filas i, j.

El algoritmo para definir $cos\alpha$, $sin\alpha$ es:

```
function [c,s]=Rotate(p1, p2)
   Tol=1e-16;
   if abs(p2)<Tol % no rotation is needed, angle is cero
    c=1; s=0;
   else
      if abs(p2)>=abs(p1) % angle larger or equal to pi/4
         cotangent= p1/p2;
         s=1/sqrt(1+cotangent^2);
         c=s*cotangent;
      else   % angle in (-pi/4, pi/4)\ {0}
         tangent= p2/p1;
         c=1/sqrt(1+tangent^2);
         s=c*tangent;
      end
   end
end
```

Además es bastante práctico ver que la matriz de rotación resultante puede construirse como

$$G = eye(m);$$
$$G([i\ ,j],[i\ ,j]) = [c\ s; -s\ c];$$

o mejor aún, que la premultiplicación de una matriz A con G puede calcularse como

$$GA = A;$$
$$GA([i,\ j],:) = [c\ s; -s\ c] * A;$$

donde c y s son los valores de $cos\alpha$ y $sin\alpha$ obtenidos en el algoritmo anterior. Para

generar los ceros en A tal como lo propusimos en (4.33) es necesario formar rotaciones donde además $j = i+1$, pues siempre podemos generar el cero en la fila $i+1$ usando la fila i , por lo que el algoritmo que se obtiene es:

```
function [Q,R]= myQR(A)
%input: A of size mxn with  m>n
%output : Q, R with Q mxm and orthogonal , R upper triang mxn
   [m,n]=size(A);
   Q=eye(m);
   for j=1:n %loop over columns
      for i=m:-1:(j+1) %loop over rows (backwards)
         [c,s ]=Rotate(A(i-1,j), A(i,j));
```

```
            A([i-1,i], j:n)=[c s;-s c]*A([i-1,i], j:n);
            Q(:, [i-1 i])=Q(:,[i-1 i])*[c s; -s c ];
        end
        A(j+1:m,j)=zeros(m-j,j);
    end
end
```

Es posible mostrar que la factorización QR para $A \in \mathbb{R}^{mxn}$ requiere $(9m-4n)n^2$ operaciones de punto flotante. Esto quiere decir que si A fuera cuadrada (nxn) el número de operaciones sería $5n^3$, es decir 7,5 veces más costosa que la factorización LU.

Sin embargo, la información para matrices rectangulares no es fácil de obtener y una factorización QR transforma el sistema lineal en uno con una estructura mucho mas simple.

Para el caso de las ecuaciones normales que deben ser resueltas para problemas de mínimos cuadrados se tien que si ahora podemos asumir que ya tenemos Q y R tal que $A = QR$, entonces

$$\begin{aligned} x &= \left(A^T A\right)^{-1} A^T b && (4.37)\\ &= \left(R^T Q^T QR\right)^{-1} R^T Q^T b && (4.38)\\ &= \left(R^T R\right)^{-1} R^T Q^T b && (4.39)\\ &= R^{-1} R^{-T} R^T Q^T b && (4.40)\\ &= R^{-1} Q^T b && (4.41) \end{aligned}$$

Por lo que resolver las ecuaciones normales se reduce a resolver el sistema trangular superior

$$Rx = Q^T b \tag{4.42}$$

que sabemos que es muy sencillo de resolver.

4.4 Descomposición en valores singulares (SVD)

La factorización SVD es muy importante y puede ser utilizada para resolver sistemas lineales, pero también para otros usos menos convencionales. La razón para esto es que contiene una descomposición con mucha interpretación desde el punto de vista teórico que puede ser explotada desde el punto de vista práctico. Muchos de los usos se basan en una versión truncada de la misma descomposición, como veremos más adelante.

Esta factorización está definida para $A \in \mathbb{R}^{m\times n}$ con $m \geq n$ y tiene la forma

$$A = U\Sigma V^T \tag{4.43}$$

con

$$\begin{aligned} U &= [u_1, \cdots, u_m] \in \mathbb{R}^{m\times m}, && (4.44)\\ V &= [v_1, \cdots, v_n] \in \mathbb{R}^{n\times n}, && (4.45)\\ \Sigma &= diag(\sigma_1, \cdots, \sigma_p) \in \mathbb{R}^{p\times p} && (4.46) \end{aligned}$$

donde $p = \min(m,n)$ y las matrices U y V están formadas por vectores ortonormales ($u_i^T u_j = 0,\ v_k^T v_l = 0,\ \|u_i\| = 1,\ \|v_j\| = 1$).

Además los valores reales σ_i son los valores singulares ($\sigma_i = |\lambda_i|$) de la matriz y los vectores u_i, v_i son llamados vectores singulares izquierdos y derechos, respectivamente.

Esta factorización está definida como sigue:

Definición 4.1 — Dscomposición en valores singulares (SVD). Sea $A \in \mathbb{R}^{m\times n}$ con $m \geq n$. Entonces definimos la descomposición en valores singulares de la forma

$$A = U\Sigma V^T, \tag{4.47}$$

con

$$U = [u_1, \cdots, u_m] \in \mathbb{R}^{m\times m}, \tag{4.48}$$
$$V = [v_1, \cdots, v_n] \in \mathbb{R}^{n\times n}, \tag{4.49}$$
$$\Sigma = diag(\sigma_1, \cdots, \sigma_p) \in \mathbb{R}^{p\times p} \tag{4.50}$$

donde $p = \min(m,n)$ y las matrices U y V están formadas por vectores ortonormales ($u_i^T u_j = 0,\ v_k^T v_l = 0,\ \|u_i\| = 1,\ \|v_j\| = 1$).

Además los valores reales σ_i son los valores singulares ($\sigma_i = |\lambda_i|$) de la matriz y los vectores u_i, v_i son llamados vectores singulares izquierdos y derechos, respectivamente.

La *SVD* tiene un sin número de aplicaciones, pues representa una de las formas más ilustrativas de la estructura de la matriz A como operador lineal. Puede demostrarse (no se incluye en este curso) que esta facorización tiene las siguientes propiedades algebráicas y geométricas:

- Suponga que A es simétrica, con eigenvalores λ_i y eigenvectores ortogonales u_i, es decir que se tiene la eigen-descomposición $A = U\Lambda U^T$. Entonces la descomposición SVD de A puede escribirse como $A = U\Sigma V^T$ donde $\sigma_i = |\lambda_i|$ y $v_i = \text{sign}(\lambda_i)u_i$ usando $\text{sign}(0) = 1$.
- Los eigenvalores de la matriz simétrica $A^T A$ son iguales a los σ_i^2. Además, los vectores singulares derechos v_i son los eigenvectores orthonormales correspondientes.
- Los eigenvalores de la matriz simétrica AA^T son los σ_i^2 y $m-n$ ceros. Los vectores singulares izquierdo u_i son los eigenvectores correspondientes y uno puede tomar hasta $m-n$ vectores ortogonales adicionales como eigenvectores del eigenvalor igual a 0.
- Si consideramos la matriz

  $$H = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}, \tag{4.51}$$

  donde A es cuadrada con $A = U\Sigma V^T$, entonces los $2n$ eigenvalores de H son $\pm\sigma_i$ con los eigenvectores correspondientes dados por $\frac{1}{\sqrt{2}} \begin{bmatrix} v_i \\ \pm u_i \end{bmatrix}$.

- Si A tiene rango completo, la solución del problema de mínimos cuadrados

$$\min_x \|Ax - b\|$$

está dado por

$$x = V\Sigma^{-1}U^T b. \tag{4.52}$$

- La norma 2 de la matriz es $\|A\|_2 = \sigma_1$. Si adicionalmente la matriz es no-singular entonces $\|A^{-1}\|_2 = \frac{1}{\sigma_n}$ y $\|A\|_2 \cdot \|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n}$.
- Suponga que $\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0$. Entonces el rango de la matriz A es igual a r. El espacio nulo de A es $span\{v_{r+1}, \ldots, v_n\}$ y el rango de la matriz A es igual a $span\{u_1, \ldots, u_r\}$.
- Sea $S^{(n-1)}$ la esfera unitaria en $\mathbb{R}^n$ y sea $A \cdot S^{(n-1)}$ la imagen de esta esfera al aplicarle la transformacion A, es decir el conjunto $\{Ax : \ x \in \mathbb{R}^n, \ \|x\|_2 = 1\}$. Entonces este conjunto $A \cdot S^{(n-1)}$ es un elipsoide centrado en el origen de $\mathbb{R}^m$ con ejes principales igual a $\sigma_i u_i$.
- La descomposición $A = U\Sigma V^T$ permite escribir a la matriz como

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T, \tag{4.53}$$

donde cada sumando es una matriz de tamaño $m \times n$.

Puede demostrrse además que esta última propiedad asegura que la suma

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_n u_n v_n^T, \tag{4.54}$$

puede truncarse en algun momento, para obtener una matriz

$$A_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T, \qquad k < n, \tag{4.55}$$

donde ahora A_k puede verse como una nueva matriz en $\mathbb{R}^{m \times n}$ que aproxima a A.

Esta es la base de lo que se conoce como la descomposición en valores singulares truncada (TSVD) y que puede usarse para diversas aplicaciones. En caso de que la transformación lineal tenga algunos valores propios nulos (igual a cero) entonces querrá decir que la versión truncada de la aproximación generará una matriz A_k que genere la misma tansformación A, pero en términos de menos información.

En otro caso, piense en la posibilidad de un espectro que decae rápidamente. Esto sucede frecuentemente en operadores lineales que provienen de problemas 'mal planteados' de acuerdo a la definición de Hadamard.

Existen además versiones modificadas de la SVD que pueden ser aplicadas a casos específicos. Por ejemplo en el caso de simetría $(A = A^T)$ se tendrá que ambas U y V son la misma colección de vectores singulares por lo que

$$A = QSQ^T. \tag{4.56}$$

Como es de esperarse, el cálculo de la factorización SVD es muy alto. Los cálculos se basan en encontrar llos eigenvalores de la matriz simétrica A^TA para luego obtener los σ_i como las raices cuadradas de esto. Todo esto debe ser hecho sin calcular la matriz A^TA.

En la práctica se calcula frecuentemente una aproxmación debida a Golub[1], Kahan[2] y Reinsch, misma que involucra una gran cantidad de operaciones igual a $2m^2n + 4mn^2 + \frac{9}{2}n^3$.

Note que esta es una gran cantidad de operaciones, pues en el caso de una matriz cuadrada derivaria en $10{,}5n^3$ operaciones.

Esto es más del doble de operaciones que requeriría una factorización QR o 78.5 veces las que requeriría LU.

4.5 Otras Factorizaciones

De manera similar a las factorizaciones presentadas en las secciones anteriores, existen otras similares que pueden utilizarse y cuya derivación es en cierto modo similar a las ya presentadas.

Por ejemplo, existe la factorización $A = L_1 D L_2^T$ para A cuadrada y con L_1, L_2 triangulares inferiores y D diagonal. Esta factorización es fácil de derivar analíticamente a partir de la factorización LU. Además si A es simétrica $(A = A^T)$, entonces esta factorización se convierte en $A = LDL^T$.

Otra factorización relevante es la llamada factorización de Choleshy, [3] que se aplica a matrices cuadradas simétricas y positivas dedfinidas (s.p.d.), es decir, para $A \in \mathbb{R}^{nxn}$, $A = A^T$, con $x^TAx > 0 \; \forall x \neq 0$ y tiene la forma

$$A = GG^T, \qquad G = \left(\begin{array}{cc} \diagup & \\ \diagup & \diagup \end{array}\right) \tag{4.57}$$

4.6 Métodos iterativos básicos

Hasta ahora hemos hablado de métodos de solución que se basan en encontar una factorización matricial del sistema lineal en la que los factores permitan una descripción distinta del operador lineal que pueda dar la solución del sistema. Estos métodos funcionan bien para matrices de tamaño pequeño y mediano, donde el cálculo de la factorización requiere un número de operaciones que aún puede efectuarse.

[1] Gene H. Golub (1932–2007), Fletcher Jones Professor of Computer Science en la Universidad de Stanford. Reconocido como una de las eminencias en Métodos Numéricos de las últimas décadas

[2] William Kahan (1933–), matemático numérico y computólogo. Recibió el premio *Turing Award* in 1989 por sus contribuciones al Análisis Numérico, entre muchos otros reconocimientos. Es además el principal creador del estándar IEEE-754 (1985)

[3] Debida a André-Louis Cholesky $(1875 - 1918)$. Matemático Frances de la École poly technique que se unió a la armada y murió al final de la 1a Guerra Mundial. Obra de la fact publicada como obra postuma.

Sin embargo, como ya vimos en el caso de las factorizaciones QR y SVD, el cálculo de las matrices (Q,R) o (U,S,V) puede ser muy costoso en términos de operaciones de punto flotante, haciendo en ocasiones prohibitibo el uso de estos métodos. Por ejemplo, el cálculo de la factorización SVD de una matriz cuadrada de 5000×5000 puede llevar a un a cantidad de operaciones de alrededor de 1,312,500,000,000 operaciones de punto flotante.

De manera ilustrativa, piense en un problema de segmentación de imágenes en el que los pixeles de una imagen se convierten en variables de un sistema lineal. esto quiere decir que, por ejemplo una imagen en resolución baja, digamos de 80×80 generaría una matriz de tamaño 6400×6400.

Una alternativa a estas costosas factorizaciones es utilizar los llamados métodos iterativos, que consisten en generar una sucesión de soluciones aproximadas que convergen a la solución exacta de un sistema lineal sin la necesiad de realizar cálculos costosos de las matrices de una factorización exacta. Dada una aproximación inicial x_0, estos métodos generan una sucesión de aproximaciones x_m que converge a la solución de $Ax = b$, donde la construcción de x_{m+1} se calcula de manera sencilla y a un bajo costo computacional a través del uso de x_m.

Si consideramos que la matriz A puede representarse como la diferencia de dos matrices

$$A = M - K, \tag{4.58}$$

entonces tendremos el sistema como $Mx - Kx = b$, o bien $Mx = Kx + b$. Definiendo ahora la nueva matriz $F = M^{-1}K$ y el nuevo vector derecho como $c = M^{-1}b$ tendremos que el sistema lineal equivale a

$$x = Fx + c, \tag{4.59}$$

y puesto de esta forma podemos pensar en esta ecuación (4.59) como nuestro método iterativo.

Para determinar bajo que condiciones la sucesión generada por un método iterativo converge es necesaria la siguiente definición:

Definición 4.2 — Radio espectral. Para un operdor lineal F, definimos su radio espectral como

$$\rho(F) = \max_i(|\lambda_i|). \tag{4.60}$$

Y con esta definición puede demostrarse (no se incluye en este curso) el siguiente teorema de convergencia de una método iterativo:

Teorema 4.1 — Convergencia del método iterativo. Dada cualquier aproximación inicial x_0, la iteración $x_{m+1} = Fx_m + c$ converge a la solución del sistema $Ax = b$ para cualuier lado derecho b si y solo si $\rho(F) < 1$.

Nuestra meta ahora es tener un método en el que tengamos una partición de $A = M - K$ en la que cumplamos las siguientes metas:

- que las cantidades $Fx = M^{-1}Kx$ y $c = M^{-1}b$ sean fáciles de evaluar, y
- que el radio espectral $\rho(F)$ sa pequeño.

Estos dos requerimientos se contraponen y tenemos que encontrar un balance entre ellos. Por ejemplo, si consideramos a la matriz $M = I$ entonces la primer meta se obtendrá directamente, pero puede que eso no asegure un radio espectral pequeño. En caso de que consideremos $M = A$ y $K = 0$ entonces el radio espectral es el más pequeño posble (cero) pero el cálculo de $c = M^{-1}b$ vuelve a ser igual de complicado que nuestro problema inicial.

4.6.1 Métodos de Jacobi y de Gauss-Seidel

El método iterativo más sencillo es conocido como el método de Jacobi y se basa en reescribir a la matriz A como sigue

$$A = D - \tilde{L} - \tilde{U} = D(I - L - U) \tag{4.61}$$

donde D es la diagonal de A, $-\tilde{L}$ es la parte estrictamente triangular inferior de A con $DL = \tilde{L}$ y $\tilde{U}$ es la parte estrictamente triangular superior de A con $DU = \tilde{U}$.

La iteración del método de Jacobi se basa en considerar ahora a la matriz separada como

$$A = D - \tilde{L} - \tilde{U}. \tag{4.62}$$

Si ahora denotamos a la matriz y lado derecho de la iteración de Jacobi como

$$F_J = D^{-1}(\tilde{L} + \tilde{U}) = L + U \tag{4.63}$$

$$c_J = D^{-1}b \tag{4.64}$$

dandonos la iteración

$$x_{m+1} = F_J x_m + c_J. \tag{4.65}$$

Esta iteración puede pensarse como que el método va tomando la j-ésima ecuación y va resolviendola de manera exacta en cada iteración pues de la iteración puede verse que la j-ésima ecuación se convierte en

$$a_{jj}x_{m+1,j} + \sum_{k \neq j} a_{jk}x_{m,k} = b_j. \tag{4.66}$$

El algoritmo de cálculo de un paso de la iteración queda como

```
for j=1:n
   x_new(j) = b(j);
   for k=1:n
      if (k~=j)
```

```
            x_new(j) = x_new(j) - A(j,k)*x_old(k);
        end
    end
    x_new(j) = x_new(j)/A(j,j);
end
```

Observe que el número de operaciones para calcular cada componente de la nueva iteración es $2(n-1)+1$ or lo que el total de esfuerzo para calcular la nueva iteración es de tan solo $2n^2-n$ operaciones.

Una primera mejora que puede realizarse dal método de Jacobi es considerando que el cálculo de cada componente j en el ciclo de cálculo es independiente. Con esto, podemos considerar que si las primeras componentes de la nueva iteración ya fueron calculadas, entonces sus valores tendrán una mejor aproximación y por tanto deberiamos usarlas al realizar el cálculo de las componentes siguientes. Esto es conocido com el método de Gauss-Seidel y en términos de ecuaciones basta con separar la suma para el cálculo de la j-ésima componente de x_{m+1} como sigue:

$$x_{m+1,j} = \frac{1}{a_{jj}}\left(b_j - \sum_{k=1}^{j-1} a_{jk}x_{m+1,k} - \sum_{k=j+1}^{n} a_{jk}x_{m,k}\right), \tag{4.67}$$

donde presuponemos que las primeras $j-1$ componentes de x_{m+1} ya han sido pre-calculadas. El algoritmo correspondiente utiliza exactamente la misma cantidad de operaciones y se ve como

```
for j=1:n
    x_new(j) = b(j);
    for k=1:n
        if (k<j)
            x_new(j) = x_new(j) - A(j,k)*x_new(k);
        elseif (k>j)
            x_new(j) = x_new(j) - A(j,k)*x_old(k);
        end
    end
    x_new(j) = x_new(j)/A(j,j);
end
```

Note que la ecuación (4.67) que define la iteración de Gauss-Seidel puede reescribirse al multiplicar por a_{jj} y pasando la primer sumatoria al lado izquierdo como

$$\sum_{k=1}^{j} a_{jk}x_{m+1,k} = b_j - \sum_{k=j+1}^{n} a_{jk}x_{m,k}, \tag{4.68}$$

y utilizando la notación de la ecuación (4.61) esto es

$$(D-\tilde{L})x_{m+1} = \tilde{U}x_m + b, \tag{4.69}$$

o bien, poniendolo en la notación que se introdujo anteriormente para los métodos iterativos

$$\begin{aligned} x_{m+1} &= (D-\tilde{L})^{-1}\tilde{U}x_m + (D-\tilde{L})^{-1}b, && (4.70)\\ &= (I-L)^{-1}Ux_m + (I-L)^{-1}D^{-1}b, && (4.71)\\ &\equiv F_{GS}x_m + c_{GS}. && (4.72) \end{aligned}$$

4.6.2 Método de sobrerelajación sucesiva (SOR)

Una modificción adicional al método iterativo que presentamos en la subsección anterior consiste en pensar que tanto x_m como x_{m+1} son aproximaciones a la solución exacta del sistema $Ax = b$ y por lo tanto tiene sentido utilizar una suma ponderada para calcular cada componente de x_{m+1} como

$$x_{m+1,j} = (1-\omega)x_{m,j} + \omega x^{GS}_{m+1,j} \tag{4.73}$$

donde x^{GS} denota las iteraciones del método de Gauss-Seidel. Este método es conocido como el método de sobrerelajación sucesiva o simplemente SOR, por sus siglas en inglés.

Sustituyendo la ecuación (4.67) en (4.73) se tiene que

$$x_{m+1,j} = (1-\omega)x_{m,j} + \frac{\omega}{a_{jj}}\left(b_j - \sum_{k=1}^{j-1} a_{jk}x_{m+1,k} - \sum_{k=j+1}^{n} a_{jk}x_{m,k}\right), \tag{4.74}$$

y esto puede reescribirse como

$$a_{jj}x_{m+1,j} + \omega\sum_{k=1}^{j-1} a_{jk}x_{m+1,k} = (1-\omega)a_{jj}x_{m,j} - \omega\sum_{k=j+1}^{n} a_{jk}x_{m,k} + \omega b_j. \tag{4.75}$$

Y usando nuevamente la notación de la ecuación (4.61) se tiene que

$$(D-\omega\tilde{L})x_{m+1} = \left((1-\omega)D + \omega\tilde{U}\right)x_m + \omega b, \tag{4.76}$$

que equivale a tener la forma del método iterativo como

$$\begin{aligned} x_{m+1} &= (D-\omega\tilde{L})^{-1}\left((1-\omega)D + \omega\tilde{U}\right)x_m + \omega(D-\omega\tilde{L})^{-1}b, && (4.77)\\ &= (D(I-\omega L))^{-1}\left((1-\omega)D + \omega\tilde{U}\right)x_m + \omega(D(I-\omega L))^{-1}b, && (4.78)\\ &= (I-\omega L)^{-1}D^{-1}\left((1-\omega)D + \omega\tilde{U}\right)x_m + \omega(I-\omega L)^{-1}D^{-1}b, && (4.79)\\ &= (I-\omega L)^{-1}\left((1-\omega)I + \omega D^{-1}\tilde{U}\right)x_m + \omega(I-\omega L)^{-1}D^{-1}b, && (4.80)\\ &= (I-\omega L)^{-1}\left((1-\omega)I + \omega U\right)x_m + \omega(I-\omega L)^{-1}D^{-1}b, && (4.81)\\ &\equiv F_{SOR}x_m + c_{SOR}. && (4.82) \end{aligned}$$

En otras palabras, la partición de la matriz original del sistema A se toma para el método SOR como

$$M = D(I-\omega L), \qquad K = D\left((1-\omega)I + \omega U\right). \tag{4.83}$$

Para el método SOR, pueden verse tres casos particulares:

- tomando $\omega = 1$ se obtiene exactamente el método de Gauss-Seidel;
- para valores $\omega < 1$ se conoce como subrelajación; y
- usar un valor de $\omega > 1$ es llamado sobrerelajación.

Para el caso de sobrerelajación se usa como motivación la idea de que si la dirección de x_m hacia x_{m+1} es una buena dirección para corregir la iteración, entonces moverse más de una vez en esa direción debe resultar aún mejor.

Es posible demostrar que el método SOR requiere que $0 < \omega < 2$ para que el proceso iterativo obtenga convergencia. Además, en el caso de que la matriz A del sistema sea positiva definida, esta condición es suficiente para obtener la convergencia.

Al comparar los métodos de Jacobi, Gauss-Seidel y SOR vemos que los dos primeros no requieren ninguna información acerca de la matriz del sistema y que el último depende de un valor adicional para el parámetro ω. Para este último, en problemas en que se tiene información adicional acerca del espectro de la matriz original del sistema, es posible asegurar o incluso acelerar la convergencia utilizando versiones modificadas. Un ejemplo de esto es el uso de la llamada aceleración de Chebyshev para generar el método SSOR.

4.7 Métodos de Krylov

Una familia de métodos ampliamente reconocida por su efectividad en resolución de sistemas lineales o de problemas de valores propios es el conjunto del los así llamados Métodos de Krylov.

La base de los métodos de Krylov es que asume que la matriz A solamente es accesible a través de un sistema tipo 'caja negra' que proporciona el resultado de la multiplicación de A con cualquier vector. Es decir, lo único que se requiere de A es la existencia de una rutina o estrategia para calcular el producto $y = Az$, incluso sin necesidad de conocer explicitamente la matriz. En el caso no-simétrico algunos métodos tambien necesitan una rutina para calcular $y = A^T z$.

Estos métodos son de particular interés en problemas donde la estructura del problema genera matrices sparse, pues el número de operaciones para evaluar una multiplicación matriz vector es de $NNZ(A)$ multiplicaciones y a lo más $NNZ(A)$ sumas, donde $NNZ(A)$ denota el número total de elementos diferentes de cero en la matriz A. Otro tipo de problemas donde estos es interesante es donde A no se conoce de manera explícita sino que se recibe como salida de algun dispositivo de cálculo o incluso algún dispositivo experimental.

Existe una gra variedad de métodos de Krylov que funcionan para diversas formas de problemas y que pueden ser utilizados para sistemas lineales con matrices cuadradas/no cuadradas, simétricas/no simétricas, etc. Algunos de estos métodos de solución requieren solamente una forma de calcular productos de la forma Az mientras otros requieren además una forma para calcular productos de la forma $A^T z$.

El método más bien comprendido y que tiene una gran catidad de usos es el método conocido como Gradiente Conjugado, del cual se derivan toda una gama de métodos útiles para una variedad amplia de formas matriciales.

La idea pricipal de los métodos de Krylov se basa en considerar el espacio de Krylov definido precisamente a través de operaciones de la forma Ax. Por simplicidad, nos concentraremos en sistemas donde la matriz del problema $Ax = b$ es cuadrada y supondremos que conocemos una forma de evaluar el producto Az para cualquier vector z. Podemos considerar la serie de productos de la forma

$$\begin{aligned} y_1 &= b, && (4.84)\\ y_2 &= Ay_1 = Ab, && (4.85)\\ y_3 &= Ay_2 = A^2 b, && (4.86)\\ &\vdots && (4.87)\\ y_n &= Ay_{n-1} = A^{n-1} b. && (4.88) \end{aligned}$$

Si ahora consideramos la matriz

$$M = [y_1,\ y_2,\ \ldots,\ y_{n-1},\ y_n] \in \mathbb{R}^{n\times n}, \tag{4.89}$$

podemos ver que al premultiplicar por A se obtiene que

$$AM = [Ay_1,\ Ay_2,\ \ldots,\ Ay_{n-1},\ Ay_n] = [y_2,\ y_3,\ \ldots,\ y_n,\ A^n y_n] \in \mathbb{R}^{n\times n}. \tag{4.90}$$

Note ahora que las primeras $n-1$ columnas de AM son las mismas que las últimas $n-1$ columnas de M. Si ahora pudieramos suponer que M es no singular entonces podemos usar su inversa para ver que si definimos $c = -M^{-1}A^n y_1$ podemos escribir

$$AM = M \cdot [e_2,\ e_3,\ \ldots,\ e_n,\ -c], \tag{4.91}$$

donde los e_i's denotan los vectores canónicos en $\mathbb{R}^n$, por lo que si ahora re-multiplicaramos por la inversa de M tendriamos

$$\begin{aligned} M^{-1}AM &= M^{-1}M \cdot [e_2,\ e_3,\ \ldots,\ e_n,\ -c] \\ &= \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -c_1 \\ 1 & 0 & 0 & \cdots & 0 & -c_2 \\ 0 & 1 & 0 & \cdots & 0 & -c_3 \\ 0 & 0 & 1 & \cdots & 0 & -c_4 \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -c_n \end{bmatrix}, \end{aligned} \tag{4.92}$$

con lo que vemos que con simple pre y post multiplicación con la matriz M hemos reducido la matriz del sistema a una forma muy simple. Sin embargo esta forma solamente nos es útil para apreciar el potencial de los productos $A^j b$, porque en la práctica no podemos suponer que M es invertible y además es fácil ver que la form de las columnas

de M provocan que su número de condición crezca rápidamente, lo que hace en la práctica muy dificil considerar que podemos acceder a la inversa de alguna forma.

Una alternativa es coniderar que conocemos la factorización QR de la matriz M. Entonces la matriz resultante de la ecuación (4.92), a la cual denotaremos como C podemos expresarla como

$$C = (QR)^{-1}AQR = R^{-1}Q^TAQR, \tag{4.93}$$

lo cual implica que

$$Q^TAQ = RCR^{-1} \equiv H, \tag{4.94}$$

y como R es triangular superior, su inversa también es también es triangular superior.

Además es simple de demostrar que la nueva matriz H es Hessemberg superior y, en el caso particular cuando A es una matriz simétrica tendremos que esta nueva matriz $H = Q^TAQ$ también es simétrica, por lo que al ser Hessemberg tendrá que ser tridiagonal.

Nuestro único problema a resolver por ahora es el hecho de que no deseamos calcular toda la matriz Q sino requerimos un método para calcular una a ua sus columnas sin recurrir al método tradicional para el cálculo de Q visto en la sección anterior de este capítulo. Lo que haremos es pensar en la igualdad $AQ = QH$ e igualar a j-ésima columna de ambos lados de esta ecuación, es decir

$$Aq_j = Qh_j = \sum_{i=1}^{j+1} h_{ij}q_i, \tag{4.95}$$

y premultiplicando esta ecuación por cualquier columna de q_m de Q tendremos

$$q_m^TAq_j = \sum_{i=1}^{j+1} h_{ij}q_m^Tq_i = h_{mj}, \tag{4.96}$$

y esto es válido para cualquier $1 \leq m \leq j$.

La ecuación (4.95) puede reescribirse como

$$h_{j+1,j}q_{j+1} = Aq_j - \sum_{i=1}^{j} h_{ij}q_i, \tag{4.97}$$

y ésta nos da una forma de ir calculando iterativamnte las comonentes de H y las columnas de Q. Esto es conocido como el algortmo de Arnoldi[4] para calcular una reducción parcial a la forma de Hessemberg de la matriz A y puede calcularse usando el siguiente algoritmo:

[4]Walter Edwin Arnoldi (1917-1995). Ingniero estadounidense que trabajó para la empresa United Aircraft Corp. en temas relacionados al cálculo de vibraciones, acústica y aerodinámica.

```
function [Q,H] = Arnoldi(A,b,k)
% Funcion to compute the first k columns of H and Q
   [m,n] = size(A);
   if (m~=n)
      error('Algorithm works for square matrices')
   end
   Q = zeros(n,k);
   H = zeros(n,k);

   Q(:,1) = b/norm(b);
   for j=1:k
      z = A*Q(:,j);
      for i=1:j
         H(i,j) = Q(:,i)'*z;
         z = z - H(i,j)*Q(:,i);
      end
      H(j+1,j) = norm(z);
      if (H(j+1,j)=0)
         disp('Rank deficient found. Number of columns computed:')
         disp(j)
         break;
      end
      Q(j+1,:) = z/H(j+1,j);
   end
end
```

Los vectores q_j son muchas veces llamados vectores de Arnoldi y el costo de realizar el cálculo de k columna de Q y de H es de k multiplicaciones de la forma Az más un total de $\mathcal{O}(k^2 n)$ operaciones extra.

En el caso de que la matriz original A sea simétrica, nuestro algoritmo puede simplificarse pues la matriz H toma una forma tridiagonal simétrica T y en cada columna solo se tendrán a lo más tres componentes distintas de cero

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{bmatrix}. \tag{4.98}$$

De esta forma, al igualar la columna j de ambos lados de la ecuación $AQ = QT$ se tiene

$$Aq_j = \beta_{j-1} q_{j-1} + \alpha_j q_j + \beta_j q_{j+1}, \tag{4.99}$$

y nuevamente al considerar que las columnas de Q deben ser ortonormales y premultiplicando por q_j se tiene que

$$q_j^T A q_j = \alpha_j. \tag{4.100}$$

Esta idea es la base del conocido método de Lanczos[5] para la reducción parcial a la forma triangular que puede ser calculada como:

```
function [Q,alpha,beta] = Lanczos(A,b,k)
% Funcion to compute the first k columns of T and Q
   [m,n] = size(A);
   if ((m~=n) || (norm(A-A')>1e-16))
      error('Algorithm works for symmetric and square matrices')
   end
   Q = zeros(n,k);
   H = zeros(n,k);

   alpha = zeros(k,1);
   beta = zeros(k,1);
   Q(:,1) = b/norm(b);
   q_previus = zeros(n,1);
   for j=1:k
      z = A*Q(:,j);
      alpha(j) = Q(:,j)'*z;
      z = z - alpha(j)*Q(:,j) - beta(j)*q_previous;
      beta(j+1) = norm(z);
      if (beta(j+1)=0)
         disp('Rank deficient found. Number of columns computed:')
         disp(j)
         break;
      end
      Q(j+1,:) = z/beta(j+1);
      q_previous = Q(j+1,:);
   end
   beta = beta(2:k);
end
```

Note que la forma de funcionar de este algoritmo incluye un proceso en el que en cada iteración se calcula un vector a través de pre-multiplicar con la matriz A del sistema y después encontrar la parte ortogonal de este nuevo vector a los vecyores previamente

[5]Cornelius Lanczos (1893-1974). Matemático y Fisico de origen Húngaro que trabajó en Hungría, Estados Unidos e Irlanda. Durante muchos años trabajo para el *National Institute of Standards and Technology* donde desarrollo diversos métodos numéricos que hoy en día son usados para el cálculo de valores propios y para la solución de sistemas lineales de gran escala.

obtenidos. En otras palabras, lo que se esta haciendo es generar vectores que pertenecena los subespacios de Krylov anidados que se obtienen como el conjunto generado por los vectores $b, Ab, A^2b, A^3b, \ldots$ para luego buscar una base ortogonal que genere el mismo espacio que estos vectores, es decir $span\{b, Ab, A^2b, \ldots, A^kb\} \equiv span\{q_1, q_2, \ldots, q_k\}$ para cualquier valor de k.

El método iterativo más popular para resolver sistemas lineales es el Método de Gradiente Conjugado (GC), el cual funciona para el caso en el que la matriz del sistema es simétrica y positiva definida. Este método se deriva a partir de la idea de direcciónes conjugadas, A-conjugadas o A-ortogonales, que es una generalización del concepto de ortogonalidad definido como sigue:

Definición 4.3 — Direcciones conjugadas. Dada una matriz $A \in \mathbb{R}^{n\times n}$, se dice que dos vectores d_1 y d_2 en $\mathbb{R}^n$ son A-conjugados si cumplen que

$$d_1^T A d_2 = 0. \tag{4.101}$$

Una forma intuitiva de derivar este método es considerar que el problema de resolver el sistema $Ax = b$ puede verse también a través de considerar una función $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ dada por

$$f(x) = \frac{1}{2}x^T Ax - x^T b, \tag{4.102}$$

la cual al ser derivada da un valor del gradiente

$$\nabla f(x) = Ax - b, \tag{4.103}$$

y esto quiere decir que si $x \in \mathbb{R}^n$ minimiza la función f, entonces debe tenerese que su gradiente es cero y por tanto esa mismo x resuleve el sistema $Ax = b$.

Por otra parte, los conjuntos o curvas de nivel de la función f, es decir los conjuntos definidos por la ecuación

$$\frac{1}{2}x^T Ax - x^T b = c, \qquad c \in \mathbb{R}^+ \cup \{0\} \tag{4.104}$$

son elipsoides en $\mathbb{R}^n$ para un valor dado de c.

Supongamos que tenemos una primer aproximación x_0 a la solución exacta del sistema $x*$. La aproximación x_0 no presupone nada y puede incluso ser un vector muy lejano a la solución que se busca.

Algo que puede realizarse siempre es considerar el elipsoide al cual pertenece x_0 e intentar acceder a un elipsoide que se encuentre en el interior de este elipsoide inicial pues eso querrá decir que el nuevo elipsoide corresponde a una constante c más pequeña, y sabemos que la solución se encuentra en el elipsoide en el que $c = 0$.

Acceder a un nuevo elipsoide en el interior puede hacerse escogiendo un vector unitario p_1 que no sea tangente al elipsoide en el punto x_0 y considerar la recta

$$\ell_1(t) = x_0 + tp_1, \qquad t \in \mathbb{R}. \tag{4.105}$$

A lo largo de esta recta, puede observarse que laevaluación de f esta dada por

$$\begin{aligned} g(t) &= f(\ell_1(t)), && (4.106)\\ &= \frac{1}{2}(x_0+tp_1)^T A(x_0+tp_1)-(x_0+tp_1)^T b, && (4.107)\\ &= \frac{1}{2}x_0^T Ax_0+\frac{1}{2}t^2 p_1^T Ap_1+tp_1^T Ax_0-x_0^T b-tp_1^T b. && (4.108)\end{aligned}$$

Estas evaluaciones de f a lo largo de la linea recta con dirección p_1 pueden minimizarse encontrando el valor de t_{min} que hace que $g'(t_{min})=0$. Puede verse fácilmente que este valor esta dado por

$$t_{min}=\frac{p_1^T(b-Ax_0)}{p_1^T Ap_1}\equiv\frac{p_1^T r_0}{p_1^T Ap_1}, \tag{4.109}$$

donde usamos la definición del residuo para una aproximación x_j definido como

$$r_j=b-Ax_j. \tag{4.110}$$

Además, es posible demostrar que este valor t_{min} genera el punto $\ell(t_{min})$ de manera que el minimizador se encuentra justo a la mitad de la cuerda generada por la dirección p_1 a partir de x_0. Por lo tanto, a este punto se le da el nombre de x_1 y se le considera nuestra siguiente aproximación a $x*$. Lo que tenemos ahora es que

$$x_1=x_0+\frac{p_1^T r_0}{p_1^T Ap_1}p_1. \tag{4.111}$$

También es un simple cálculo algebraico revisar qu se obtiene un decremento de la función f al verificar que

$$f(x_0)-f(x_1)=\frac{1}{2}\frac{(p_1^T r_0)^2}{p_1^T Ap_1}, \tag{4.112}$$

y dado que A es s.p.d. el denominador es positivo y entonces el valor de f presenta un decremento en x_1 con respecto de su valor en x_0.

Los siguientes dos teoremas provienen de geometría y conectan la idea de las direcciones A conjugadas con nuestra búsqueda de la solución a traves de los elipsoides generados por la función f:

Teorema 4.2 — Hiperplano en el elipsoide. El conjunto de los puntos medios de todas las cuerdas paralelas en un elipsoide en $\mathbb{R}^n$ forma un hiperplano de dimensión $n-1$. En particular, el centro del elipsoide pertenece a este hiperplano que denotaremos P_{n-1}.

Teorema 4.3 — Hiperplano A-conjugado. Sea P_{n-1} el hiperplano generado por los puntos medios de cuerdas paralelas con dirección p_1 al cual pertenece x_1. Si se considera cualquier otro punto que pertenece a est hiperplano $y \in P_{n-1}$ entonces el vector que conecta a x_1 con y dado por $y - x_1$ es A-conjugado del vector p_1.

De manera recíproca, si cualquier vector v es A-conjugado de p_1, entonces el vector $x + \omega v$ también pertenece a P_{n-1}

El método de gradiente conjugado puede entonces construirse siguiendo la siguiente estrategia:

- transformar el problema en la búsqueda del centro de un elipsoide en $\mathbb{R}^n$ a través de la función f,
- definir una linea recta de búsqueda en dirección p_1 y minimizar f para encontrar el punto x_1,
- considerar la intersección del hiperplano P_{n-1} con el elipsoide en $\mathbb{R}^n$,
- observar que esta intersección es ahora un elipsoide en $\mathbb{R}^{n-1}$,
- determinar una dirección conjugada a la dirección inicial y resolver el mismo problema para encontrar ahora un x_2.

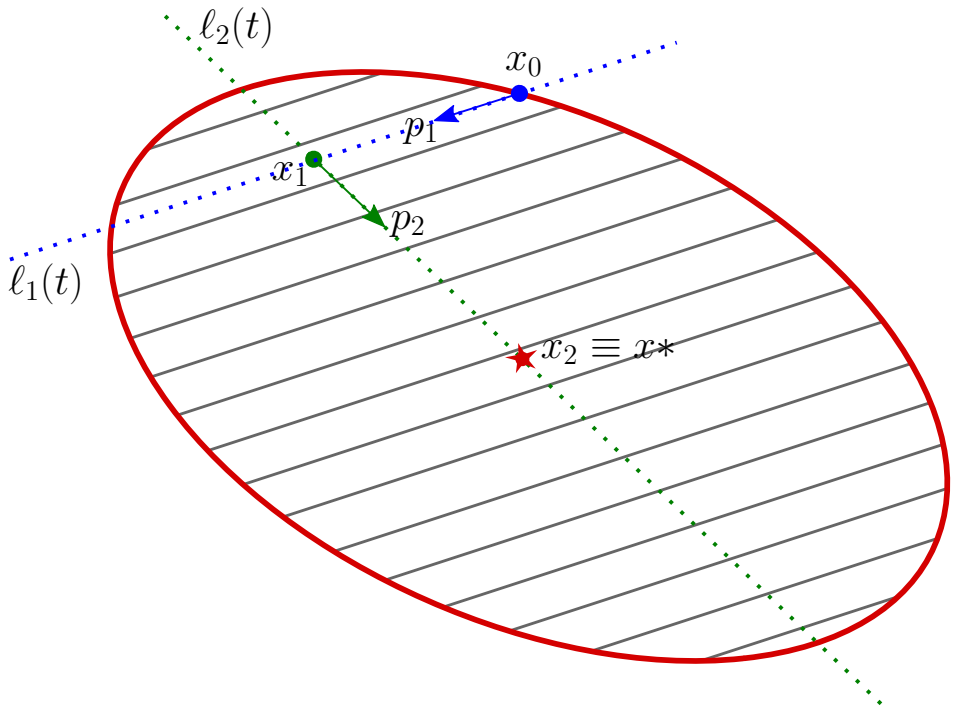


Figura 4.6: Esquemática del método de GC para el caso $n = 2$. Dado un punto x_0 y una dirección inicial p_1, puede calcularse la primer iteración x_1 y una nueva direción de busqueda que será la dirección A-conjugada p_2. En este caso, solamente dos iteraciones bastarían para resolver el sistema en aritmética exacta.

La única parte que habria que mostrar es el hecho de que las direcciones conjugadas pueden irse obteniendo de manera iterativa usando los residuales como

$$p_k = r_{k-1} + \beta_k p_{k-1}, \tag{4.113}$$

$$\beta_k = \frac{r_{k-1}^T r_{k-1}}{r_{k-2}^T r_{k-2}}. \tag{4.114}$$

Esto deja la versión general de la ecuacion (4.111) como

$$x_k = x_{k-1} + \alpha_k p_k \tag{4.115}$$

$$\alpha_k = \frac{r_{k-1}^T r_{k-1}}{p_k^T A p_k}. \tag{4.116}$$

Finalmente, el algoritmo que puede implementarse puede resumirse en el siguiente algoritmo simple:

Algoritmo de Gradiente Conjugado

$k = 0, \quad r_0 = b - Ax_0$

WHILE $\|r_k\| > TOL$

$k = k + 1$

IF $k = 1$

$p_1 = r_0$

ELSE

$\beta_k = \frac{r_{k-1}^T r_{k-1}}{r_{k-2}^T r_{k-2}}$

$p_k = r_{k-1} + \beta_k p_{k-1}$

END

$\alpha_k = \frac{r_{k-1}^T r_{k-1}}{p_k^T A p_k}$

$x_k = x_{k-1} + \alpha_k p_k$

$r_k = r_{k-1} - \alpha_k A p_k$

END

Una de las mayores bondades del método de GC es que, en teoría, encuentra la solución exacta $x*$ en solamente n iteraciones debido que existen a lo más n direcciones conjugadas en $\mathbb{R}^n$.

Por otro lado, en la práctica se observa que este método tiene lo que se llama en ocasiones *propiedades regularizantes* que se refieren a que realiza el cálculo de iteraciones $x_0, x_1, x_2, \ldots$ que aproximan primero a la parte bien condicionada de sistemas mal condicionados.

4.8 Ejercicios

Ejercicio 4.1. Algoritmo de Thomas. (25 Puntos)

Considere el sistema lineal tridiagonal $Ax = q$ con la matriz de tamaño $n \times n$ dada como

$$A = \begin{pmatrix} +b_1 & -c_1 & & & \\ -a_2 & +b_2 & -c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & -a_{n-1} & +b_{n-1} & -c_{n-1} \\ & & & -a_n & +b_n \end{pmatrix}$$

Muestre analíticamente que la matriz A puede ser convertida en una matriz bidiagonal superior utilizando operaciones elementales de fila para posteriormente resolver un sistema bidiagonal y obtener los valores de $x_n, x_{n-1}, \ldots, x_1$, lo cual puede ser escrito a través del siguiente algoritmo:

1. Start: $\alpha_1 = b_1, \quad \beta_1 = q_1$
2. Forward steps: $\alpha_j = b_j - \frac{c_{j-1}}{\alpha_{j-1}} a_j, \quad \beta_1 = q_j + \frac{\beta_{j-1}}{\alpha_{j-1}} a_j, \qquad j = 2, \ldots, n$
3. Backward steps: $x_n = \frac{\beta_n}{\alpha_n}$

$$x_j = \frac{\beta_j + c_j x_{j+1}}{\alpha_i}, \qquad j = n-1, \ldots, 1$$

Calcule además el número de operaciones y determine con ello el orden del algoritmo en función de n.

Realice también la implementación del Algoritmo de Thomas en una función de MATLAB con la forma *[x,exectime]=mysolveThomas(A,q)*. ■

Ejercicio 4.2. Resolución usando LU (30 Puntos)

Implemente los siguientes programas en archivos independientes

- *[L,U,exectime]=myLU(A)*:
  Cálculo de la factorizacion LU de una matriz cuadrada A, con salida de las dos matrices L, U y el tiempo de ejecución.
- *[y,exectime]=mysolveL(L,c)*:
  Cálculo de la solución al sistema triangular inferior $Ly = c$, con salida de la solución al sistema y y el tiempo de ejecución.
- *[x,exectime]=mysolveU(U,b)*:
  Cálculo de la solución al sistema triangular superior $Ux = b$, con salida de la solución al sistema x y el tiempo de ejecución.
- *[L,U,x,exectime]=mysolveLU(A,b)*:
  Cálculo de la solución al sistema $Ax = b$ con factorizacion LU de una matriz cuadrada A, con salida de las dos matrices L, U y el tiempo de ejecución. Use para esta función las funciones propias *myLU*, *mysolveL* y *mysolveU*.

■

Ejercicio 4.3. Codificando mensajes (30 Puntos)

Una opción bastante simplista de codificar un mensaje escrito es asignar a cada una de las letras del abecedario, el espacio vacío, los dígitos y algunos símbolos de puntuación los números enteros de la forma $A = 1$, $B = 2, \ldots$, $\tilde{N} = 15, \ldots$, $Z = 27, \ldots$.

Con esto, un mensaje de la forma 'MENSAJE 1' puede ser re-escrito en trozos de longitud k en formato numérico. Una vez en formato numérico, cada trozo del mensaje puede codificarse mediante una multiplicación matricial como sigue:

Considere la matriz de rango completo con $k = 3$ dada por

$$M = \begin{pmatrix} 2 & 3 & 5 \\ 7 & 11 & 13 \\ 17 & 19 & 23 \end{pmatrix}$$

y úsela para codificar el mensaje por trozos usando tres caracteres en cada trozo. Para hacer esto construya vectores de la forma $b_j = Mx_j$ donde x_j es un vector de longitud k que contiene tres caracteres del mensaje a codificar.

Asuma que la persona que debe recibir el mensaje conoce de antemano la matriz M y que usted le envía solamente los vectores b_j. Para decodificar el mensaje, el receptor deberá resolver una colección de problemas $Mx = b$, donde los diferentes vectores b irán siendo trozos de mensaje codificado y esto resultará en la version (numérica) del mensaje original.

Implemente esta codificación y decodificación en un programa de MATLAB. Use para esto la factorización LU de la matriz M.

Una vez implementado pruebe su método para codificar y después decodificar el siguiente mensaje[a]:

No es el conocimiento, sino el acto de aprendizaje; y no la posesión, sino la obtención del mismo; no el estar ahí, sino el acto de llegar, lo que concede el mayor disfrute - Carl Friedrich Gauss

Observe que este método es eficiente y difícil de burlar si una tercera persona recibe solamente la información concatenada de los b_j (sin enviar el mensaje en trozos). Esto es posible porque el emisor y el receptor conocen la matriz M pero no es necesario enviar la información sobre cuántos caractéres deberán formar cada bloque. Una decodificación no es trivialmente realizable sin conocer M. Además si se utilizara una matriz M de mayor tamaño, las dificultades para 'espiar' el mensaje serán aún mayores. ■

[a] Frase original tomada de una carta de K. F. Gauss a Wolfgang Bolyai en Septiembre de 1808. 'Wahrlich es ist nicht das Wissen, sondern das Lernen, nicht das Besitzen, sondern das Erwerben, nicht das Da-sein, sondern das Hinkommen, was den grössten Genuss gewährt'

Ejercicio 4.4. Facorización QR (35 Puntos)

Implemente un programa para calcular la factorización QR de una matriz $A \in \mathbb{R}^{m\times n}$, $m \geq n$ y con $Q \in \mathbb{R}^{m\times m}$, $R \in \mathbb{R}^{m\times n}$. Úsela para calcular las factorizaciones de las siguientes matrices:

$$M_1 = \begin{pmatrix} 1 & 3 & 6 \\ 5 & 7 & 4 \\ 2 & 2 & 8 \\ 4 & 1 & 9 \\ 3 & 6 & 6 \\ 2 & 4 & 7 \end{pmatrix} \qquad M_2 = \begin{pmatrix} 1 & 3 & 7 & 8 \\ 0 & 5 & 9 & 6 \\ 0 & 0 & 2 & 4 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

$$M_3 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 1 \\ 6 & 9 \\ 7 & 8 \end{pmatrix} \qquad M_4 = \begin{pmatrix} 15 & 0 & 0 \\ 0 & 28 & 0 \\ 0 & 0 & 48 \end{pmatrix}$$

Escriba sus observaciones sobre la forma de las matrices originales M_i y sus matrices de factorización Q y R. ■

Ejercicio 4.5. Aproximación de terreno. (30 Puntos)

Considere el conjunto de puntos (x, y, z) donde x denota latitud, y denota longitud y z denota altitud en su archivo de datos GPS del Proyecto 1 (o uno similar) y el problema de encontrar el plano que mejor aproxime al terreno del cual fueron tomados estos datos. El problema consiste en encontrar los coeficientes $\xi = (\alpha, \beta, \gamma)^T$ del plano

$$z = \alpha x + \beta y + \gamma$$

en $\mathbb{R}^3$ que mejor aproxima al conjunto de puntos (x, y, z) y que por tanto resuelve el problema de mínimos cuadrados

$$\min_{\xi\in\mathbb{R}^3} \|A\xi - b\|_2$$

donde las filas de la matriz A tienen la forma $(x_i, y_i, 1)$ y el lado derecho se forma como $b_i = z_i$.

Use los algoritmos para la factorización QR y la solución de sistemas triangulares superiores para resolver este problema y grafique el plano con los coeficientes obtenidos en un dominio $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ que abarca a todos los puntos tomados por GPS. Sobre el gráfico, represente los puntos originales (x_i, y_i, z_i) en forma de puntos o asteriscos (comandos útiles para los gráficos: *meshgrid, surf, hold, plot3, rotate3d*). ■

Ejercicio 4.6. Matriz de Hessenberg (20 Puntos)
Modifique su función para calcular la factorización QR para ser aplicada al caso específico en que se tiene una matriz superior de Hessenberg que corresponde a una matriz triangular superior que además tiene una subdiagonal inferior de la forma

$$H = \begin{pmatrix} \times & \times & \times & \cdots & \times & \times \\ \times & \times & \times & \cdots & \times & \times \\ & \times & \times & \cdots & \times & \times \\ & & \times & \cdots & \times & \times \\ & & & \ddots & \vdots & \vdots \\ & & & & \times & \times \end{pmatrix}$$

■

Ejercicio 4.7. Aplicación real (Bonus[a], 15 Puntos)
Encuentre una aplicacion real en la que pueda modelar un fenómeno y reducirlo a un problema de mínimos cuadrados con más de 3 coeficientes. Resuélvalo utilizando los algoritmos ya progrmados, analice e interprete sus resultados a través de gráficos y comentarios. ■

[a] La función de estos ejercicios tipo *bonus* es que los estudiantes trabajen un poco extra y de manera autónoma para recuperar puntos si así lo desean. Son opcionales y no habrá asesoramiento para realizarlos.

Ejercicio 4.8. Su recorrido de deporte (Bonus, 10 Puntos)
Encuentre el plano que mejor aproxime a los datos tridimenionales de su medición GPS obtenida de un recorrido de deporte y registrada a través de un dispositivo móvil. Haga representaciones gráficas interesantes y describa sus hallazgos. ■

Ejercicio 4.9. Dibujando con la SVD (30 Puntos)
Considere la penúltima propiedad de la decomposición en valores singulares referente a la transformación de la esfera en un elipsoide. Considere una matriz de tamaño 2×2 y la visualización de sus valores y vectores propios. Implemente un programa que grafique dos subfiguras similares a las que se realizaron en la clase:

- Subfigura 1: Debe contener el gráfico del círculo unitario en $\mathbb{R}^2$ y un conjunto de varios puntos identificados por diferentes colores o símboloss.
- Subfigura 2: Debe contener los puntos correspondientes a las transformaciones de los puntos identificables de la subfigura 1, así como los puntos correspondientes a $\sigma_1 u_1$ y $\sigma_2 u_2$.

Asegúrese de que su implementación funciona para cualquier matriz de 2×2. Realice la visualización de algunas matrices diferentes, en particular use al menos una matriz en la que sepa que no tiene inversa y pruebe con alguna en la que

los valores propios sean de ordenes muy diferentes (e.g. Una matriz en la que $\sigma_1 = 100,\ \sigma_2 = 10^{-8}$).

Para este ejercicio puede usar la imlementación de la factorización precargada en el software (Matlab, R, etc.). Entregue un reporte con sus observaciones. ■

5. Solución de ecuaciones no lineales

5.1 Introducción

Desde un punto de vista analítico, siempre es importante saber encontrar las raíces de una función. Existen diversos métodos para resolver ecuaciones de tipos conocidos como polinomios, algunas funciones trigonométricas, etc.

Sin embargo, los métodos analíticos se complican o son inexistentes cuandos las funciones de las que se busca encontrar ceros son analíticamente difíciles.

Esto se ve agravado además para el caso de funciones multivariadas, en las que la dimensión de los puntos a evaluar o del espacio imagen no es 1 y por tanto no son subconjuntos de $\mathbb{R}$. Problemas de este tipo hay muchos. Pensemos por ejemplo en la concentración de un químico que se diluye a una razón de

$$f(t) = 25e^{-3t} + 6e^{-5t} - 17, \qquad t > 0$$

y en la que se está interesado en saber en qué momento del tiempo t se logra una concentración de la mitad de la cantidad inicial. Es decir, se busca la solución a la ecuación

$$f(t) - \frac{1}{2}f(0) = 0$$

Otro ejemplo mucho más complejo es ver si en un sistema planetario se puede encontrar un momento en que algunos planetas estén alineados a través de una función que incluya al seno del ángulo entre ellos y resolver para saber cuándo este ángulo es cero.

5.2 Bisección para problemas en una dimensión

Partiendo del supuesto de que se tiene una función continua $f : [a,b] \subset \mathbb{R} \to \mathbb{R}$ en la que $f(a) < 0$ y $f(b) > 0$ (o de forma equivalente $f(a) > 0$ y $f(b) < 0$) y que se puede evaluar la función f, es posible pensar que una búsqueda sencilla consiste en saber qué trozo del intervalo $[a,b]$ contiene al cero de f una vez que se considere una bisección del mismo en los dos subintervalos

$$\left[a,\ \frac{a+b}{2}\right], \qquad \left[\frac{a+b}{2},\ b\right]$$

Considerando el signo de $f\left(\frac{a+b}{2}\right)$ puede reiniciarse la búsqueda en uno de los subintervalos y continuar este proceso hasta encontrar el cero de la función con una buena calidad de aproximación numérica.

La buena calidad en la aproximación numérica consiste en llevar el proceso hasta tener un subintervalo tan pequeño que no pueda distinguirse entre los límites izquierdo y derecho del mismo. En forma de pseudo-código, la búsqueda puede hacerse como:

1: while $|a-b| > \delta + eps * \text{máx}(|a|,|b|)$
2: if $f(a)\cdot f(\frac{a+b}{2}) \leq 0$ Cambio de signo en $\left[a,\ \frac{a+b}{2}\right]$
3: if $f(a)\cdot f(\frac{a+b}{2}) \leq 0$ Cambio de signo en $\left[a,\ \frac{a+b}{2}\right]$
4: else Cambio de signo en $\left[\frac{a+b}{2},\ b\right]$
5: $a = \frac{a+b}{2}$
6: end
7: end
8: $root = \frac{a+b}{2}$

Algoritmo 1: Algoritmo

```
while (|a-b|) > tolerancia
   if (f(a) * f((a+b)/2) <= 0
(Cambio de signo en [a, (a+b)/2)] y b = f((a+b)/2))
   else
(Cambio de signo en [f((a+b)/2)), b] y a = \frac{a+b}{2}
   end
end
root = f((a+b)/2))
```

Una pequeña mejora consiste en pensar que un usuario de este algoritmo pudiera definir un δ igual a algo muy pequeño (e.g. $\delta = 10^{-12}$). Sin embargo, si el intervalo original $[a,b]$ es tal que el espaciamiento en la APF entre a y b es muy grande, cabe la posibilidad de que en algún momento $\frac{a+b}{2}$ no sea elemento de la APF, por lo que será redondeado hacia a o hacia b y el ciclo **while** podrá caer en una de estas opciones:

- los límites izquierdo y derecho vuelven a ser los mismos, por lo que el ciclo nunca terminará
- ambos límites obtienen el mismo valor, por lo que el ciclo terminará pero dará un valor equivocado pues tanto a como b pueden estar lejos de la raíz correcta.

Por esto, modificaremos la línea del código que mide la calidad y la pondremos como:

$$\text{while} |a-b| > \delta + eps * \text{máx}(|a|,|b|)$$

Algoritmo 2: Modificación del algoritmo

```
while (|a-b|) > tolerancia + epsilon
max(|a|,|b|)
```

Otra mejora que podríamos hacer es evitar evaluar dos veces la función f en cada paso del ciclo. Esto solo requiere declarar una variable adicional y el algoritmo de bisección queda como

```
function root = Bisection(fname, a, b, delta)
% Input:  fname --> string with function f(x)
%         a,b   --> search interval  with f(a)*f(b)<0
%         delta --> non-negative real as tolerance
% Output: root  --> midpoint of last interval


    fa = feval(fname,a);
    fb = feval(fname,b);
    while ( abs(a-b)>delta+eps*max(abs(a),abs(b)) )
        mid = (a+b)/2;
        fmid = feval(fname,mid);
        if (fa*fmid <= 0)  %root is in [a,mid]
            b = mid;
            fb = fmid;
        else               %root is in [mid,b]
            a = mid;
            fa = fmid;
        end
    end
    root = (a+b)/2;
end
```

Si consideramos que en el k-ésimo paso se tendrá el intervalo $[a_k, b_k]$ entonces

$$|a_k - b_k| = \frac{|a_0 - b_0|}{2^k}$$

donde a_0 y b_0 son los límites iniciales. Por lo tanto la convergencia de $[a_k, b_k]$ a un intervalo que cumpla con la condición de proximidad $\delta > |a_k - b_k|$ está garantizada. Además como $x_k = \frac{a_k+b_k}{2}$ podemos ver que si la raíz buscada es x_* entonces

$$|x_k - x_*| \leq \frac{|a_0 - b_0|}{2^{k+1}}$$

El error en esta aproximación se reduce a la mitad en cada iteración.

Esto es un ejemplo de lo que se denomina convergencia lineal en la que se dice que un sucesión $\{x_k\}_{k\in\mathbb{N}}$ converge a x_* de manera lineal si existe una constante $c \in [0,1)$ y un entero k_0 tal que

$$|x_{k+1} - x_*| \leq c|x_k - x_*| \qquad \forall k \geq k_0 \tag{5.1}$$

5.3 El Método de Newton para encontrar raíces de funciones

El Método de Newton[1] sirve para encontrar raíces de funciones. Supongamos que tenemos una función $f : [a,b] \subset \mathbb{R} \to \mathbb{R}$ con $f \in \mathscr{C}^1((a,b))$ y que podemos realizar evaluaciones tanto de f como de su derivada.

El método de Newton para encontrar los ceros de f en $[a,b]$ consiste en un proceso iterativo en el que cada paso resuelve el problema de encontrar un cero de una aproximación lineal a f.

Esta aproximación esta dada por la línea recta que pasa por $f(x_i)$ y tiene pendiente $f'(x_i)$, es decir que es la recta tangente a f en el punto x_i.

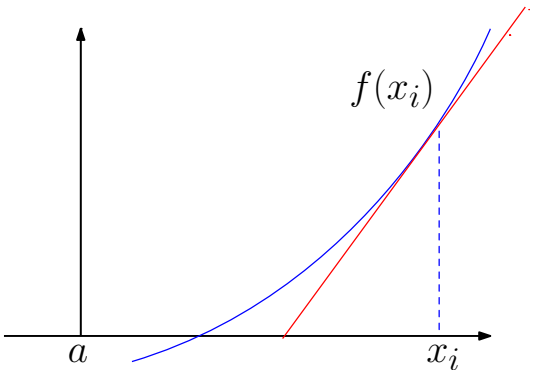


Figura 5.1: Aproximación por medio del método de Newton

[1] Isaac Newton (1643–1727), Científico reconocido en campos diversos como la física, matemática, filosofía, y Teología. Es autor de la obra *Philosophiae Naturalis Principia Mathematica* en la que desarrolló las leyes de gravitación y la gravitación universal que dieron origen a la mecánica clásica. También es reconocido como uno de los desarrolladores del cálculo infinitesimal y diversos trabajos en óptica.

La recta tiene la forma

$$\ell_i(x) = f(x_i) + (x - x_i) \cdot f'(x_i), \tag{5.2}$$

por lo que no es difícil ver que esta recta cruza al eje horizontal, es decir $\ell_i(x) = 0$, cuando

$$x = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{5.3}$$

Entonces, asignaremos a x_{i+1} este valor en el proceso de búsqueda iterativa de x_* tal que $f(x_*) = 0$. Suponiendo que f no contiene fuertes no-linealidades en el intervalo de búsqueda, es de esperarse que el resultado después de algunas iteraciones aproxime bien al cero de f. Un algoritmo básico para este proceso sería:

```
function root = Newton_basic(fname, fpname, x0, delta)
    xold = x0;
    fvalue = feval(fname,xold);
    fpvalue = feval(fpname,xold);
    xnew = xold - fvalue/fpvalue;
    while ( abs(xnew-xold)>delta+eps*max(abs(a),abs(b)) )
        xold = xnew;
        fvalue = feval(fname,xold);
        fpvalue = feval(fpname,xold);
        xnew = xold - fvalue/fpvalue;
    end
    root = xnew;
end
```

Es posible mostrar que este método presenta convergencia cuadrática, es decir que

$$|x_{i+1} - x_*| \leq c|x_i - x_*|^2$$

donde nuevamente x_* denota el cero buscado. Sin embargo, esto solamente es válido en un intervalo alrededor de x_* y la convergencia no está garantizada como en el método de bisección de la Sección anterior. El método de Newton puede no converger en casos en que f presente muchas no-linealidades o tenga valores muy chicos de derivadas.

El ejemplo clásico en el que el método de Newton falla es la función $f(x) = \arctan(x)$, $f : \mathbb{R} \to \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$ En este caso $f'(x) = \frac{1}{1+x^2}$ por lo que la iteración tiene la forma

Con estas funciones puede verificarse que si $x \notin [-1{,}3917,\ 1{,}3917]$ entonces $|x_{i+1}| > |x_i|$, haciendo que el proceso diverja y nunca encuentre una única posición en que $f(x) = 0$. El problema radica en que para valores fuera de $[-1{,}3917,\ 1{,}3917]$ la derivada posee valores muy pequeños. En los casos en que esto no sucede y que f no presenta grandes no-linealidades, la convergencia si está garantizada a través del siguiente teorema:

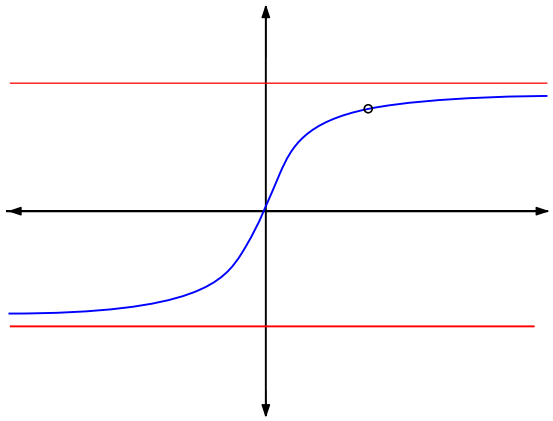


Figura 5.2: Función $f(x) = \arctan(x)$

Teorema 5.1 Sean las funciones f y su derivada f' definidas en un intervalo $I = [x_* - \mu, x_* + \mu]$ donde $f(x_*) = 0$, $\mu > 0$ y además existen contantes positivas ρ, δ tales que

1. $|f'(x)| \geq \rho, \quad \forall x \in I$
2. $|f'(x) - f'(y)| \leq \delta |x - y|, \quad \forall x, y \in I$
3. $\mu \leq \rho/\delta$

Si $x_i \in I$, entonces $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \in I$ y

$$|x_{i+1} - x_*| \leq \frac{\delta}{2\rho} |x_{i+1} - x_*|^2 \leq \frac{1}{2} |x_i - x_*|$$

Lo que quiere decir que un paso de Newton mejora la aproximación en una tasa mejor a $1/2$ (bisección) y que garantiza una convergencia que además es cuadrática.

Demostración: Usando el teorema fundamental del cálculo y álgebra puede verse que

$$-f(x_i) = f(x_*) - f(x_i) = \int_{x_i}^{x_*} f'(z) dz$$

$$\implies -f(x_i) - (x_* - x_i) f'(x_i) = \int_{x_i}^{x_*} f'(z) dz - \int_{x_i}^{x_*} f'(x_i) dz$$

Incorporando las dos integrales y dividiendo por $f'(x_i)$

$$\implies \left(x_i - \frac{f(x_i)}{f'(x_i)} \right) - x_* = \frac{1}{f'(x_i)} \int_{x_i}^{x_*} \left(f'(z) - f'(x_i) \right) dz$$

y tomando valores absolutos

$$\implies |(x_{i+1} - x_*| = \left| \frac{1}{f'(x_i)} \right| \cdot \left| \int_{x_i}^{x_*} \left(f'(z) - f'(x_i) \right) dz \right|$$

y usando la condición (i) como $\frac{1}{|f'(x)|} \leq \frac{1}{\rho}$

$$\begin{aligned} |x_{i+1} - x_*| &\leq \frac{1}{\rho} \left| \int_{x_i}^{x_*} \left(f'(z) - f'(x_i) \right) dz \right|, \\ &\leq \frac{1}{\rho} \int_{x_i}^{x_*} \left| f'(z) - f'(x_i) \right| dz. \end{aligned}$$

Si ahora usamos la condición (ii) e integramos se obtiene

$$\begin{aligned} |x_{i+1} - x_*| &\leq \frac{\delta}{\rho} \int_{x_i}^{x_*} |z - x_i| \, dz, \\ &\leq \frac{\delta}{\rho} \cdot \frac{1}{2} |x_* - x_i|^2, \end{aligned}$$

y esto prueba la primer desigualdad. Por último, es fácil ver que la segunda desigualdad puede demostrarse al considerar que $x_{i+1} \in I \;\Rightarrow\; |x_{i+1} - x_*| \leq \mu$ y después usar la condición (iii) para ver que

$$\frac{\delta}{2\rho} |x_{i+1} - x_*|^2 \leq \frac{1}{2} \frac{\delta \mu}{\rho} |x_i - x_*| \leq \frac{1}{2} \rho |x_i - x_*| \quad \blacksquare$$

Aunque el teorema podría servir para indicarnos cuándo podemos esperar convergencia del método de Newton, (en la práctica las condiciones del mismo son difíciles de cumplir) para muchas funciones. Otro punto importante es que para asegurar que el proceso iterativo de búsqueda no se pierda, podemos establecer un **Método Híbrido** en el que según convenga se utilice una iteración del método de Bisección o del de Newton. Para esto será necesario asumir que se tiene un intervalo de búsqueda $[a,b]$ y que la iteración $x_i \in [a,b]$. Procederemos como sigue:

- si x_{i+1} definido por el método de Newton como en la ecuación (5.3) pertenece a $[a,x_i]$ o $[x_i,b]$ (según corresponda) entonces usaremos el nuevo x_{i+1} de esta forma (Newton);
- en caso contrario, tomaremos un paso del método de bisección haciendo $x_{i+1} = \frac{a+b}{2}$.

Para revisar si $x_{i+1} \in [a,b]$ podemos usar este algoritmo:

```
function inside=StepIsIn(x, fx, fpx, a, b)
    % Input    x: current iterate
    %         fx: function evaluation at x
    %        fpx: derivative evaluation at x
    %        a,b: interval left and right limits
    %
    % Output  inside: flag for inside (1) or outside (0)
```

```
    if ( fpx>0 )
        inside = ( ((a-x)*fpx)<=-fx )&&( -fx<=(b-x)*fpx );
    elseif ( fpx<0 )
        inside = ( ((a-x)*fpx)>=-fx ) && ( -fx>=(b-x)*fpx );
    else
        inside = 0;
    end
end
```

Además, para garantizar una correcta terminación del algoritmo es necesario detener el proceso si alguna de los siguientes condiciones se cumple:

1. la longitud del intervalo de búsqueda es menor a una tolerancia *tolx*
2. el valor absoluto de $f(x_i)$ es menor que una tolerancia *tolf*
3. el número de evaluaciones de f excede un límite *nEvalsMax*

Note que esto no garantiza haber encontrado un cero, por ejemplo si $f(x) = (x-1)^{0,1}$ y $x_i = 1{,}0000000001$. Entonces $|x_i - x_*|$ es pequeño pero $f(x)$ no esta cercano a cero.

Note que eso no garantiza que $|x_i - x_*|$ sea pequeño para funciones muy planas cerca de la raíz, e.g. $f(x) = (x-1)^{10}$.

5.4 Método de Newton Multivariado

Encontrar las raíces de una función $f : \mathbb{R}^n \to \mathbb{R}^n$ puede hacerse de manera similar, siguiendo la misma estrategia pero considerando las generalizaciones correspondientes. Ahora f es una función que se evalúa en vectores de $\mathbb{R}^n$ y que le asigna un vector también en $\mathbb{R}^n$. La funcón puede ser escrita como

$$f(x) \equiv f(x_1,\ldots,x_n) = \begin{bmatrix} f_1(x_1,\ldots,x_n) \\ f_1(x_1,\ldots,x_n) \\ \vdots \\ f_n(x_1,\ldots,x_n) \end{bmatrix} \tag{5.4}$$

con $x \in \mathbb{R}^n$, $f(x) \in \mathbb{R}^n$. Para evitar confusiones denotaremos a los diferentes puntos de las iteraciones de Newton con un super-índice, es decir

$$x^{(j)} \equiv \left(x_1^{(j)}, x_2^{(j)}, \ldots, x_n^{(j)}\right) \tag{5.5}$$

será la j-ésima aproximación a una raíz y su k-ésima componente será $x_k^{(j)}$. Para las

derivadas necesarias requeriremos la generalización dada por el Jacobiano[2]

$$J(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_n} \end{bmatrix} \tag{5.6}$$

El Jacobiano será utilizado para obtener la dirección de búsqueda del método mediante la aproximación lineal con

$$\ell(x) = f\left(x^{(k)}\right) + J(x^{(k)})\left(x - x^{(k)}\right) \tag{5.7}$$

y la búsqueda sobre la aproximación lineal se refiere a encontrar el punto x tal que $\ell(x) = 0$, es decir

$$f\left(x^{(k)}\right) + J\left(x^{(k)}\right)x - J\left(x^{(k)}\right)x^{(k)} = 0,$$

o bien

$$x = x^{(k)} - \left[J\left(x^{(k)}\right)\right]^{-1} f\left(x^{(k)}\right),$$

que es precisamente como se define la siguiente iteración. Definiendo la siguiente iteración como

$$x^{(k+1)} = x^{(k)} - \left[J\left(x^{(k)}\right)\right]^{-1} f\left(x^{(k)}\right) \tag{5.8}$$

el algoritmo de cálculo es básicamente igual al definido para el caso univariado de la Sección anterior. Lo que si es mucho más complicado es la decisión sobre bisectar o usar un paso de Newton. Esto es una consecuencia indirecta de que no es tan simple determinar si la nueva iteración cae en el dominio de interés, pues ahora este dominio es

$$x^{(k)} \in [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$$

5.5 Ejercicios

Ejercicio 5.1. Método de Bisección (40 Puntos)

Implemente el algoritmo de bisección visto en clase y también un programa con una función que reciba una cadena de caracteres (string) y los límites del intervalo $[a,b]$ como entrada y aplique el método de bisección para encontrar una raiz de la función.

Su programa deberá verificar que los signos de la función evaluada sean distintos en ambos límites y encontrar una raiz con una tolerancia $\delta = 10^{-11}$.

[2] Atribuido a Carl Gustav Jacob Jacobi (1804–1851). Matemático Alemán que trabajó en funciones elípticas, ecuaciones diferenciales y teoría de números. Fue el primer judío nombrado Professor en Alemania.

La función también deberá desplegar en pantalla una tabla con las columnas correspondientes al número de iteración y los tres valores para a_k, b_k, $b_k - a_k$ y al final, el valor de la aproximación obtenida. Use el comando `format long` para desplegar la información en pantalla con más cifras decimales y los comandos `disp`, `sprintf` para generar los mensajes de texto en pantalla.

Use las funciones implementadas para aproximar al cero de la función

$$f(x) = \tan\left(\frac{x}{4}\right) - 1$$

en el intervalo $[1,5]$ que analíticamente se encuentra en $x = \pi$. Observe los resultados en su tabla y vea que la convergencia es lineal según lo visto en clase. ¿Qué constante de convergencia lineal c aparenta tener el algoritmo para esta función f? ■

Ejercicio 5.2. Método de Newton (40 Puntos)

Implemente el método de Newton para obtener nuevamente la raiz de la función del ejercicio anterior con $f'(x) = \sec^2(x/4)/4$ y el inicio de la busqueda con $x_0 = 1$. Observe el nuevo orden de convergencia y verifique que es cuadrático. ■

Ejercicio 5.3. Funciones no lineales con varias raíces (10 Puntos)

Reflexione sobre la posibilidad de que una función tenga varias raíces en el intervalo de búsqueda. Sin saber el tipo de función ni tampoco dónde ni cuántas raíces tendrá ¿qué podria hacerse para encontrar varias de ellas? ■

Ejercicio 5.4. Ejemplo de uso (30 Puntos)

Busque un problema aplicado en el que pueda utilizar el método de Newton para encontrar la raiz de una función que no pueda ser resuelta analíticamente. Use sus propios programas para resolverlo. ■

Ejercicio 5.5. Convergencia del método de Newton (20 Puntos)

Considere las siguientes funciones $f_i : [1,3] \subset \mathbb{R} \to \mathbb{R}$

$$f_1(x) = x-2, \quad f_2(x) = (x-2)^2, \quad f_3(x) = (x-2)^3,$$

$$f_4(x) = (x-2)^4, \quad f_5(x) = (x-2)^5, \quad f_6(x) = (x-2)^6,$$

$$f_7(x) = (x-2)^7, \quad f_8(x) = (x-2)^8, \quad f_9(x) = (x-2)^9$$

todas con una raiz en $x = 2$. Encuentre numéricamente la raiz de cada una de estas funciones iniciando el método de Newton en el punto $x_0 = 3$.

Utilice la información generada edurante el proceso iterativo para determinar si hay algo que pueda decirse sobre la similitud/diferencia en la rapidez de convergencia del método para funciones de diferente grado polinomial. ■

Ejercicio 5.6. Método de Newton multivariado (40 Puntos)

Implemente el algoritmo de Newton para funciones multivariadas asumiendo que no se requieren pasos intercalados de partición del dominio (como los pasos de bisección en el caso univariado) ni un control para verificar que las iteraciones se encuentren dentro del dominio deseado.

Use este algoritmo para encontrar la raíz a la función

$$f(x_1,x_2) = \begin{bmatrix} \frac{1}{2}(1-\cos(x_1-x_2)) \\ 1-\sin\left(\frac{x_1+x_2}{2}\right) \end{bmatrix}$$

en el dominio $[0,\pi]\times[0,\pi]\subset\mathbb{R}^2$. Es posible mostrar analíticamente que la raíz se encuentra en el punto $\tilde{x}=(\pi/2,\pi/2)$, por lo que puede comprobar la convergencia de sus resultados numéricos mediante la cantidad $\|x^{(i)}-\tilde{x}\|$. ■

Ejercicio 5.7. Jacobiano no invertible (20 Puntos)

Observe que el algoritmo requiere que el Jacobiano pueda ser invertido. Dada la forma del Jacobiano que se obtiene al sacar las derivadas parciales en este ejemplo de función, ¿siempre es posible calcular $J^{-1}(x)$? ¿Que opciones alternativas se tendrian? Argumente, discuta, y escriba uns pseudo-código con sus ideas de solución. ■

Ejercicio 5.8. Aproximación del Jacobiano por diferencias (30 Puntos)

Suponga ahora que es imposible calcular las derivadas analíticas de la función. Modifique su programa del ejercicio anterior para que calcule aproximaciones al Jacobiano a través de Diferencias Finitas. Asuma que la función multivariada tiene la forma $f:\mathbb{R}^2\to\mathbb{R}^2$ y asegurese de que el Jacobiano pueda ser calculado para cualquier punto de un dominio predefinido como $[a_1,b_1]\times[a_2,b_2]$.

Use un tamaño de paso $\Delta x_1=\Delta x_2=0{,}1$ y pruebe su implementación usando la misma función del ejercicio anterior pero ahora sin usar el Jacobiano analítico. Compare la eficiencia de los resultados. ■

Ejercicio 5.9. Paso adecuado de las Diferencias Finitas (20 Puntos)

Haga un análisis con respecto al tamaño de paso de las diferencias finitas para ver como cambia/empeora/mejora la aproximación entre la implementacion de Diferencias Finitas en este Ejercicio y la del Ejercicio anterior. Incluya argumentaciones concretas acerca de sus resultados. ■