Apuntes de Métodos Numéricos
2º E.T.S.I. Telecomunicación
Universidad de Málaga

Carlos García Argos (garcia@ieee.org)
http://www.telecos-malaga.com

Curso 2002/2003

Índice general

Presentación

Estos apuntes de la asignatura de Métodos Numéricos, impartida en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Málaga se han escrito durante el actual curso 2002/2003, a partir de los apuntes anteriores, que contenían una serie de erratas e incorrecciones. Además, se han ampliado con código Matlab que implementa algunos de los métodos que se mencionan, así como algunos ejemplos adicionales, para facilitar la comprensión.

Asimismo, se ha incluido un resumen con lo más destacado de la asignatura para ayudar a memorizar lo más importante de cara al examen.

Cualquier comentario, corrección o sugerencia será bienvenido a la dirección `garcia@ieee.org`.

Apuntes escritos usando el programa LyX, front-end para el sistema de tipografía LaTeX 2_ε, ejecutado sobre un sistema Debian Linux.

Se advierte que los apuntes pueden contener errores, por supuesto no intencionados por parte del autor, pero que se subsanarán tan pronto como se encuentren.

Este documento se terminó el día 14 de Junio de 2003.

Sobre la notación

A lo largo de todos los apuntes se ha procurado mantener una notación uniforme, de forma que no haya que estar pensando constantemente en lo que significa cada cosa.

- $[a_{ji}]$ significa una matriz cuadrada formada por todos los elementos a_{ji}
- el exponente ^{t} denota la traspuesta de una matriz o un vector
- $\mathbb{R}$ representa el conjunto de los números reales
- $\mathbb{C}$ representa el conjunto de los números complejos
- ∇ es la divergencia
- ∇^2 es la Hessiana
- $[a, b]$ es un intervalo cerrado
- (a, b) es un intervalo abierto
- $[a, b)$ es un intervalo cerrado por la izquierda y abierto por la derecha
- $\aleph$ es el núcleo de una base o una matriz

TEMA 1

Errores

1.1 Introducción a los métodos numéricos computacionales

El *objetivo* de los Métodos Numéricos es el de obtener **soluciones aproximadas**. Se trabaja con números reales. Los Métodos Numéricos comprenden el desarrollo y la implementación y uso de algoritmos numéricos y de software para la resolución de un problema físico, a partir de un modelo matemático del sistema físico en cuestión.

Estos Métodos Numéricos se implementan con **algoritmos** en máquinas de cálculo como puede ser un ordenador personal con un software adecuado. Hay gran cantidad de software numérico disponible, y además de varios tipos, por ejemplo:

1. Paquetes software: algunos paquetes gratuitos son
   - ✏ Sistemas de ecuaciones lineales: LINPACK, LAPACK
   - ✏ Autovalores: EISPACK, ARPACK
2. Librerías de programas:
   - ✏ IMSL, NAG, HARWELL
3. Sistemas de software científico:
   - ✏ Análisis numérico: Matlab, octave
   - ✏ Análisis simbólico: Maple V

Estos algoritmos implican una serie de **simplificaciones** en cuanto al cálculo exacto de la solución del problema, como pueden ser las discretizaciones en los intervalos al calcular integrales numéricamente, evaluación de funciones trascendentes de forma aproximada (exponenciales por ejemplo), almacenamiento de números sin precisión infinita, etc. Esto genera **indeterminaciones** en los resultados, por lo que las soluciones obtenidas no serán exactas y tendremos que analizar hasta qué punto son válidas dichas soluciones.

1.2 Generación y propagación de errores

Hay varios tipos de errores que pueden aparecer en la resolución de problemas:

- ✏ **Errores experimentales o de medición**: son errores anteriores al cálculo y por tanto no aplicables a los Métodos Numéricos.
- ✏ **Errores en el cálculo numérico**:
  - ☞ Al operar en punto flotante (ordenadores). Son **errores de redondeo**.
  - ☞ De **discretización o truncamiento**: son errores propios de los métodos numéricos (no se hace el límite infinito para una serie, la integral con todos los valores del intervalo, etc.).
- ✏ **Implementación del método**: errores de programación.

A continuación únicamente vamos a considerar los errores de cálculo numérico.

Habrá que hacer las siguientes consideraciones:

- ✏ Compromiso entre error de truncamiento y error de redondeo
- ✏ Adecuabilidad del método: convergencia, velocidad, ...
- ✏ Coste computacional

1.2.1 Representación de números

Para representar números reales se usan las bases, que especifican el rango de valores que pueden tomar los dígitos de cada número y en cierta medida los dígitos que hacen falta para representar un número cualquiera.

- ✏ Representación de un número real x en el sistema decimal:

$$\begin{aligned} x &= \mp c_1 c_2 c_3 \cdots c_n . c_{n+1} \cdots \\ &= \mp \left(c_1 \cdot 10^{n-1} + c_2 \cdot 10^{n-1} + \cdots + c_n + \frac{c_{n+1}}{10} + \frac{c_{n+2}}{100} + \cdots \right) \\ & \quad c_j \in \{0, 1, \ldots 9\} \end{aligned}$$

- ✏ Representación de un número real x en base β:

$$\begin{aligned} x &= \mp d_1 d_2 \cdots d_k . d_{k+1} \cdots (\beta \\ &= \mp \left(d_1 \cdot \beta^{k-1} + d_2 \cdot \beta^{k-2} + \cdots + d_k + \frac{d_{k+1}}{\beta} + \frac{d_{k+2}}{\beta^2} + \cdots \right) \\ & \quad d_j \in \{0, 1 \ldots \beta - 1\} \end{aligned}$$

  En los sistemas computacionales suele ser $\beta = 2,\ 8,\ 16$.

- ✏ Representación normalizada:

$$x = \mp . d_1 d_2 \cdots d_k d_{k+1} \cdots \times \beta^e$$

  Donde: $d_1 \neq 0$; $e \in \mathbb{Z}$ y $d_1 d_2 \cdots d_k d_{k+1} \cdots$ es la **mantisa normalizada**, esto es, el número escrito de forma que su parte entera es cero y el primer decimal es no nulo.

El almacenamiento en una máquina se realiza como se ve en la tabla 1.1.

La **mantisa normalizada** consta de t dígitos, siendo el primero distinto de cero, y el exponente es tal que $l \leq e \leq L$. Normalmente una **mantisa** tiene la forma

$$\boxed{.d_1 d_2 d_3 \ldots \qquad \text{con } d_1 \neq 0} \tag{1.1}$$

Como se ha mencionado anteriormente.

Los números que existen para la máquina son los que se pueden representar mediante todas las combinaciones de esta forma de representación, los demás no son representables para ella.

signo	exponente(e)	mantisa normalizada

Tabla 1.1: Almacenamiento de un número normalizado

1.2.2 Truncamiento y redondeo

En un ordenador no se puede almacenar un número con infinitos decimales, ya que tiene una limitación en el tamaño de sus registros. Se usa la numeración en **punto flotante**: t dígitos en base β y exponente e entre l y L, limitándose por tanto la cantidad de números representables.

$$\begin{aligned} F(\beta,t,l,L) &= \left\{+.000\cdots 0\times\beta^0\right\}\cup \\ &\quad \{\mp.d_1d_2d_3\cdots d_t\times\beta^c,\ 0<d_1<\beta,\ 0\le d_i<\beta,\ i=2:t,\ l\le e\le L\} \end{aligned}$$

Ejemplo 1.1:

En el sistema de punto flotante $F(2,3,-1,2)$, los números representados tienen la forma

$$.1XY\times 2^e$$

Donde X e Y pueden tomar los valores 0 y 1, mientras que e puede ser -1, 0, 1 o 2. Por tanto, hay 16 números válidos en este sistema de punto flotante.

El número $.101\times 2^{-1}$ se puede poner en notación decimal de la forma

$$\begin{aligned} .101\times 2^{-1} &= \left(1\times 2^{-1}+0\times 2^{-2}+1\times 2^{-3}\right)\times 2^{-1} \\ &= \left(\frac{1}{2}+\frac{1}{8}\right)\times\frac{1}{2} \\ &= \frac{5}{16} \end{aligned}$$

■

Al representar los números de esta forma, se pierden infinitos valores de la recta real, ya que es imposible almacenar todos los valores.

$$\begin{aligned} x &= (\mp.d_1d_2\cdots d_t\times\beta^e)+\left(\mp.d_{t+1}\cdots\times\beta^{e-t}\right) \\ &= u\times\beta^c+v\times\beta^{e-t};\ \frac{1}{\beta}\le|u|<1;\ 0\le|v|<1 \end{aligned}$$

El primer paréntesis (u) es la parte del número que podemos representar con este tipo de almacenamiento, mientras que el segundo paréntesis (v) no nos cabe. Tenemos dos técnicas para representar con un número finito de decimales un número real:

✏ Quitar v sin más: **truncar** el número.

$$fl(x)=x_C=u\times\beta^e$$

Por ejemplo, la representación truncada de $\pi=3.14159265\ldots$ con $t=4$ decimales es $\pi^*=3.1415$.

✏ **Redondear** el número. Es lo más habitual en los ordenadores de hoy día. Con el truncamiento, si se desprecia una parte del número que sea pequeña no hay demasiado problema, pero si esa parte es grande (relativamente), podemos estar obteniendo un número aproximado especialmente malo. Por ejemplo, no es lo mismo truncar 1.9 a 1 que redondearlo a 2 (ejemplo radical).

$$fl\,(x) = x_R = \begin{cases} u \times \beta^e & |v| < \frac{1}{2} \\ u \times \beta^e + \beta^{c-t} & u > 0, |v| \geq \frac{1}{2} \\ u \times \beta^e - \beta^{c-t} & u < 0, |v| \geq \frac{1}{2} \end{cases}$$

Por ejemplo, para el número $\pi = 3.14159265\ldots$ con $t = 4$ su representación redondeada será $\pi^* = 3.1416$ y con $t = 5$, $\pi^* = 3.15159$.

Una vez aplicado el redondeo, si es necesario, se vuelve a normalizar el número.

Ejemplo 1.2:

Tenemos el número $0.999974 \cdot 10^4$ y vamos a redondearlo a 4 cifras:

$0.9999 \cdot 10^4 + 0.0001 \cdot 10^4 = 1.0000 \cdot 10^4$ número no normalizado:

$1.0000 \cdot 10^4 = 0.1000 \cdot 10^5$ ya está normalizado

■

Ejemplo 1.3:

Si tenemos un sistema que sólo representa números enteros, encontramos las siguientes representaciones de números para el truncamiento:

$$fl\,(30.4) = 30 \quad fl\,(30.6) = 30 \quad fl\,(30.9) = 30 \quad fl\,(31.1) = 31$$

Mientras que para la representación redondeada sería

$$fl\,(30.4) = 30 \quad fl\,(30.6) = 31 \quad fl\,(30.9) = 31 \quad fl\,(31.1) = 31$$

Si nos encontrásemos con el caso de $fl\,(30.5)$, habría que establecer un criterio, que por lo general será asignar el que sea par en la base correspondiente.

■

1.2.3 Errores en la representación de números en punto flotante

Si x^* es un valor aproximado de x: el error que resulta de sustituir x por su forma en punto flotante, x^*, se llama **error de redondeo** (tanto si se usa redondeo como truncado). Tenemos distintas formas de cuantificar el *grado de equivocación* que cometemos al hacer la representación de x por x^*. Estas son las siguientes:

✏ **Error** de x^*:

$$\boxed{E = x^* - x} \tag{1.2}$$

Esta magnitud tiene signo, el cual hay que interpretar de forma adecuada.

✏ **Error absoluto** de x^*:

$$\boxed{|E| = |x^* - x|} \tag{1.3}$$

Es simplemente el valor absoluto del error visto antes.

✏ **Error relativo** de x^*:

$$\boxed{\varepsilon = \frac{x^* - x}{x}} \tag{1.4}$$

Mide cuánto de grande es el error frente al valor real del número x. Se puede representar en forma porcentual como

$$\varepsilon = \frac{x^* - x}{x} \cdot 100(\,\%) \tag{1.5}$$

Para ver su utilidad, podemos ver que no es lo mismo equivocarse en 1 metro si la distancia real es de 10 metros que equivocarse en 1 metro midiendo la distancia de la Tierra a la Luna.

1.2.3.1 Cotas de error

Lo normal al calcular un número x^*, es no conocer el valor exacto x. Por tanto, es imposible conocer el error, el error absoluto o el error relativo de ese número aproximado. De todas formas sí que podríamos calcular algo que nos sirva para conocer el grado de error que tiene ese numero, sus **cotas de error**.

Un número M es una cota del error absoluto de x^* si

$$|x^* - x| \leq M \tag{1.6}$$

mientras que m es una cota del error relativo de x^* si

$$\frac{x^* - x}{x} \leq m \tag{1.7}$$

Lo normal cuando se dice "el error absoluto de x es y" es interpretarlo como que y es una cota del error absoluto de x. Podemos considerar equivalentes las siguientes expresiones

$$|x^* - x| \leq y \qquad x = x^* \pm y$$

También se suele calcular el error relativo usando el valor aproximado de x, x^* para dividir el error (en este caso la cota):

$$\varepsilon \simeq \frac{M}{x^*}$$

Las cotas de error en la representación de números en punto flotante truncada (si $x \neq 0$) son:

✏ $E_C = -v \times \beta^{e-t}$

✏ $|E_C| = |v| \times \beta^{e-t} < \beta^{e-t}$

✏ $|\varepsilon_C| = \frac{|fl(x)-x|}{|x|} < \frac{\beta^{e-t}}{|x|} \leq \frac{\beta^{e-t}}{\frac{1}{\beta}\beta^e} = \beta^{1-t}$

Cuantas más cifras se tomen, menor serán las cotas de error (y por tanto el error), y el error absoluto es mayor cuanto mayor es el número aproximado.

Y las cotas para la representación de números en punto flotante redondeada ($x \neq 0$):

✏ $E_R = \begin{cases} -v \times \beta^{e-t} & |v| < \frac{1}{2} \\ (1-v)\,\beta^{e-t} & u > 0;\ |v| \geq \frac{1}{2} \\ (-1-v)\,\beta^{e-t} & u < 0;\ |v| \geq \frac{1}{2} \end{cases}$

- $|E_R| \leq \frac{\beta^{e-t}}{2}$
- $|\varepsilon_R| \leq \frac{\beta^{1-t}}{2}$

La representación redondeada tiene la mitad de error que la truncada, por lo que es un sistema mejor para representar números, aunque es ligeramente más sofisticado.

Normalmente se nota la cota de error relativo con $\delta(x^*)$, mientras que la de error absoluto con $\Delta(x^*)$, así:

$$\Delta(x_c^*) = \beta^{e-t} \qquad \Delta(x_r^*) = \frac{1}{2}\beta^{e-t}$$

$$\delta(x_c^*) = \beta^{1-t} \qquad \delta(x_r^*) = \frac{1}{2}\beta^{1-t}$$

Donde x_c^* es el número aproximado con truncamiento y x_r^* el número aproximado con redondeo.

Se suele notar además, para los números

$$x = x^* \pm \Delta(x^*) \tag{1.8}$$

$$x^* = x(1 \pm \delta(x^*)) \tag{1.9}$$

Ejemplo 1.4:

Consideremos los errores absolutos y relativos:

- Si $p = 0.3000 \times 10^1$ y $p^* = 0.3100 \times 10^1$ el error absoluto es 0.1 y el relativo $0.333\hat{3} \times 10^{-1}$.
- Si $p = 0.3000 \times 10^{-3}$ y $p^* = 0.3100 \times 10^{-3}$ el absoluto es 0.1×10^{-4} y el relativo $0.333\hat{3} \times 10^{-1}$.
- Si $p = 0.3000 \times 10^4$ y $p^* = 0.3100 \times 10^4$ el error absoluto es 0.1×10^3 y el relativo $0.333\hat{3} \times 10^{-1}$.

Se observa que el error relativo es el mismo aún cambiando en varios órdenes de magnitud el número. El error relativo es maś significativo que el absoluto.

■

Veamos algunas definiciones:

- Se llama **unidad de error de redondeo** en base β (*unit roundoff* en inglés) al número $\mu = \frac{1}{2}\beta^{1-t}$.
- **Epsilon de máquina** es el menor número $\mu > 0$ tal que $fl(1+\mu) \neq 1$, el número que hay que sumarle a otro para que se obtenga el número siguiente en la numeración de la máquina. Es una forma de medir el número de decimales con el que opera una máquina. Por ejemplo, en Matlab, el epsilon de máquina *eps* es una constante que vale $2^{-52} \simeq 2 \cdot 10^{-16}$, y realmente es 2 veces la unidad de error de redondeo.
- Un valor x^* aproximado a x tiene k **dígitos fraccionarios exactos o correctos** en base β si cumple:

$$\boxed{|x^* - x| < \frac{1}{2}\beta^{-k}} \tag{1.10}$$

Aunque normalmente se suele hablar de dígitos correctos, en algunos libros se habla de dígitos exactos. Si por ejemplo aproximamos el número $\pi = 3.14159265\ldots$ por $\pi^* = 3.1416$, el dígito 6 será correcto pero no exacto si verifica la relación anterior.

✏ Un valor x^* aproximado a x tiene k **dígitos significativos** en base β si cumple:

$$\boxed{\left|\frac{x^* - x}{x}\right| < \frac{1}{2}\beta^{1-k}} \tag{1.11}$$

Visto de otra forma, sobre el número:

$$x^* = \underbrace{\underbrace{d_{n-1}\dots d_1 d_0}_{n \text{ digitos}} . \underbrace{d_{-1}d_{-2}\dots d_{-d}}_{d \text{ digitos fraccionarios}}}_{n+d=i \text{ digitos significativos}}$$

Las dos últimas definiciones son especialmente importantes.

Ejemplo 1.5:

Si consideramos el número exacto $x = 30.50028$ y su aproximación $x^* = 30.500$,

$$n - 1 = 1 \Rightarrow n = 2$$
$$-d = -3 \Rightarrow d = 3$$

Por tanto hay

$$i = n + d = 5$$

dígitos significativos.

Para otro número, por ejemplo $x = 0.00160428$ y $x^* = 0.001604$, tenemos 4 dígitos significativos ya que los ceros sólo sirven para indicar dónde está el punto decimal.

Si tomamos para el número exacto $x = 28294$ la aproximación $x^* = 28300$, entonces sólo hay 3 dígitos significativos, ya que los ceros nos indican también dónde está el punto decimal.

■

También se puede definir un dígito d_k como **dígito significativo correcto** (se entiende redondeado) si cumple

$$|x^* - x| \leq \Delta(x^*) \leq \frac{1}{2}10^k \qquad k \leq n - 1$$

Ejemplo 1.6:

Para el valor $x^* = 30.50028$ y el error $\Delta(x^*) = 0.0005$, quedándonos sólo con las cifras significativas correctas es

$$x^{*'} = 30.500$$

■

Ejemplo 1.7:

Si tenemos la siguiente cota de error para un número x dado

$$\frac{|x^* - x|}{|x|} \leq \frac{1}{2}10^{1-8} = tol$$

Donde tol representa la **tolerancia**, el error relativo del número. Si además encontramos que una cota inferior para el número x es $|x| \geq 3.75$, entonces podemos asegurar

$$x \in [x^* - 3.75tol, x^* + 3.75tol]$$

■

Habría que probar desde $n-1$ hasta d para encontrar el último dígito significativo correcto, pero es más práctico empezar buscando el d para el que se cumpla la condición.

Ahora consideremos la aritmética de punto flotante.

1.2.4 Errores generados en la aritmética de punto flotante:

Definimos una operación con el símbolo $\circ$, de forma que $\circ \in \{+, -, *, /\}$.

En una unidad aritmético-lógica se utilizan registros como unidades de almacenamiento de los datos que se van a operar. Estos tienen t posiciones para almacenar la mantisa, aunque en ocasiones se dispone de s posiciones adicionales, que nos pueden servir para almacenar los datos significativos de los datos en caso de que tengamos que desplazar los bits de la mantisa de alguno de ellos.

- ✏ Si $s = t$, se habla de **aritmética de doble precisión**
- ✏ Si $s = 0$, se dice que se trabaja con **aritmética simple** (o de simple precisión)
- ✏ Si $s = 1$, entonces tenemos **aritmética con centinela**

Si tenemos dos números en punto flotante:

$$fl\,(x) = x_i\,(m_x,\, e_x)$$

$$fl\,(y) = y_i\,(m_y,\, e_y)$$

Donde m_x, m_y son las mantisas de x e y y e_x, e_y los exponentes de x e y respectivamente, al hacer la suma, si tenemos que desplazar la mantisa (cuando $e_x \neq e_y$ por ejemplo), se pueden perder dígitos si no se dispone de posiciones adicionales para almacenarlos. Al multiplicar, las mantisas son de t dígitos, así que el resultado será de $2t$ dígitos, con lo cual podemos perder una cantidad considerable de precisión en el cálculo.

El caso más deseable es que $s = t$, pero esto implica un coste bastante elevado, por lo que se suele alcanzar un compromiso. A veces se usa $s = 1$, aunque soluciones radicalmente económicas no tienen dígitos adicionales.

El error relativo al hacer una operación $\circ$ es:

$$\left| \frac{fl\,(x^* \circ y^*) - (x^* \circ y^*)}{x^* \circ y^*} \right| \leq r\mu \tag{1.12}$$

Siendo $\mu = \frac{1}{2}\beta^{1-t}$ y $x^* = fl\,(x)\,;\, y^* = fl\,(y)$.

El valor de r depende del de s:

$$s = t \quad \Rightarrow \quad r = \begin{cases} 1 \text{ si el numero es redondeado} \\ 2 \text{ si el numero es truncado} \end{cases}$$

$$s = 1 \quad \Rightarrow \quad r = \begin{cases} 2 \text{ para multiplicacion y division} \\ 4 \text{ para sumas y restas} \end{cases} \tag{1.13}$$

$$s = 0 \quad \Rightarrow \quad r = \begin{cases} \text{puede ser muy grande para sumas y restas} \\ 4 \text{ para multiplicacion y division} \end{cases}$$

En el último caso, r depende de los operandos, y no tiene demasiada influencia para multiplicaciones y divisiones.

Por tanto: si hay posiciones adicionales, el error cometido no es realmente problemático, sin embargo, cuando no hay ninguna posición adicional, podemos tener un error muy grande. Esto ocurre, por ejemplo, al trabajar con números reales de doble precisión, que ocupan al almacenarlos todas las posiciones del registro, incluidas las adicionales, con lo que se comporta como si $s = 0$.

Conclusión: una máquina finita comete errores, tanto al almacenar datos como al operar con ellos. Por ello, en la aritmética de punto flotante se pierden, en general, las propiedades asociativa y distributiva, y habrá que especificar el orden en que se opera.

1.2.4.1 Propagación de errores:

Dadas dos aproximaciones de x e y, x^* e y^* respectivamente, el error total en el resultado de una operación aritmética $\circ$ es la suma del error generado y el propagado:

$$fl\left(x^* \circ y^*\right) - x \circ y = fl\left(x^* \circ y^*\right) - x^* \circ y^* + x^* \circ y^* - x \circ y \tag{1.14}$$

Donde:

- ✏ El error generado al hacer la operación se obtiene suponiendo los datos exactos y el proceso de operación inexacto:
$$fl\left(x^* \circ y^*\right) - x^* \circ y^*$$
- ✏ El error propagado en la operación se obtiene suponiendo que los datos son inexactos y que el proceso de operación se realiza de forma exacta:
$$x^* \circ y^* - x \circ y$$

Si sólo hay cotas $\Delta(\cdot)$ para los errores absolutos en los operandos, $|x^* - x| \leq \Delta(x^*)$ y $|y^* - y| \leq \Delta(y^*)$, es claro que, en el supuesto de $|x^*| \gg \Delta(x^*)$ y $|y^*| \gg \Delta(y^*)$, para el error absoluto se verifica:

$$\begin{array}{ll} \text{Suma} & \Delta\left(x^* + y^*\right) = \Delta\left(x^*\right) + \Delta\left(y^*\right) \\ \text{Resta} & \Delta\left(x^* - y^*\right) = \Delta\left(x^*\right) + \Delta\left(y^*\right) \\ \text{Multiplicacion} & \Delta\left(x^* \times y^*\right) \simeq \left|y^*\right| \Delta\left(x^*\right) + \left|x^*\right| \Delta\left(y^*\right) \\ \text{Division} & \Delta\left(\frac{x^*}{y^*}\right) \simeq \frac{|y^*|\Delta(x^*)+|x^*|\Delta(y^*)}{|y^*|^2} \Leftrightarrow y, y^* \neq 0 \end{array} \tag{1.15}$$

Veamos la demostración para la resta, la multiplicación y la división:

Para la resta,

$$\begin{aligned} \left|(x^* - y^*) - (x - y)\right| &= \left|(x^* - x) - (y^* - y)\right| \\ &\leq |x^* - x| + |y^* - y| \\ &\leq \Delta\left(x^*\right) + \Delta\left(y^*\right) \end{aligned}$$

Para la multiplicación, si ε, τ son los errores relativos de x^* e y^* respectivamente:

$$\begin{aligned} x^* &= x\left(1 + \varepsilon\right) \\ y^* &= y\left(1 + \tau\right) \end{aligned}$$

$$\begin{aligned} x^* y^* - xy &= x\left(1 + \varepsilon\right) y\left(1 + \tau\right) - xy \\ &= xy\varepsilon + xy\tau + xy\varepsilon\tau \\ &= y\left(x^* - x\right) + x\left(y^* - y\right) + \left(x^* - x\right)\left(y^* - y\right) \end{aligned}$$

Y tomando valores absolutos se obtiene la expresión antes vista. Los dos primeros paréntesis van multiplicados por y y x respectivamente, los cuales hay que sustituir por y^* y x^* respectivamente ya que el número que se conoce es la aproximación, el número almacenado en la máquina.

Para la división es:

$$\begin{aligned}\left|\frac{x^*}{y^*}-\frac{x}{y}\right| &= \left|\frac{x^*y-xy^*}{y^*y}\right| = \left|\frac{(x^*-x)\,y+x\,(y-y^*)}{y^*y}\right| \\ &\leq \frac{|x^*-x|\,|y|+|y^*-y|\,|x|}{|y^*|\,|y|}\end{aligned}$$

Se hace $|x^*-x|=\Delta(x^*)$, $|y^*-y|=\Delta(y^*)$, $x=x^*$ y $y=y^*$ y queda la expresión dada.

Volviendo sobre el grupo de ecuaciones (1.15), se puede observar que la división puede tener una cota de error grande, ya que y^* puede ser muy pequeño, con lo que su cuadrado lo es más y el cociente se hace muy grande.

Veamos ahora las cotas de error relativo:

Si ε y τ son los errores relativos de x^* e y^* respectivamente,

$$\begin{array}{ll}\text{Suma} & \frac{|(x^*+y^*)-(x+y)|}{|x+y|} \leq \frac{\delta(x^*)+\delta(y^*)}{|x+y|} \\ \text{Resta} & \frac{|(x^*-y^*)-(x-y)|}{|x-y|} \leq \frac{\delta(x^*)+\delta(y^*)}{|x-y|} \\ \text{Multiplicacion} & \frac{|(x^*\times y^*)-(x\times y)|}{|x\times y|} \lesssim |\varepsilon|+|\tau| \\ \text{Division} & \frac{\left|\left(\frac{x^*}{y^*}\right)-\left(\frac{x}{y}\right)\right|}{\left|\frac{x}{y}\right|} = \left|\frac{\varepsilon-\tau}{1+\tau}\right| \lesssim |\varepsilon|+|\tau|\end{array} \tag{1.16}$$

Desde el punto de vista del error relativo, las operaciones peligrosas son las sumas y las restas, ya que se divide y si los números son pequeños, la cota de error puede ser muy grande.

Aparecen las **diferencias cancelativas**, operaciones de resta que pueden anular o casi anular términos que realmente no se anulan, introduciendo un error más grande todavía. Se dan en sumas con operandos de signo contrario y casi del mismo módulo, y en diferencias de números con el mismo signo y módulos también aproximados.

Ejemplo 1.8:

$\sqrt{1+x}-\sqrt{1-x}$ si $x \simeq 0$ nos va a salir un resultado erróneo al operar directamente. Para obtener un resultado mejor, factorizamos la suma:

$$\sqrt{1+x}-\sqrt{1-x} = \frac{1+x-(1-x)}{\sqrt{1+x}+\sqrt{1-x}} = \frac{2x}{\sqrt{1+x}+\sqrt{1-x}}$$

Este resultado es preferible, ya que es un número muy pequeño multiplicado por 2 dividido entre un número que es aproximadamente 2. De la otra forma, al hacer la raíz de $1+x$ siendo x un número muy pequeño queda prácticamente la raíz de 1, con lo que el resultado es mucho menos preciso.

■

Veamos lo que ocurre con *funciones* (sucesión de operaciones, una forma de generalizar el error para *n* operaciones):

Tenemos $y = f(x)$ donde y es el resultado y x el dato, y $y^* = f(x^*)$ siendo x^* e y^* las aproximaciones de x e y respectivamente. Si f es derivable, podemos desarrollar la función en series de Taylor:

$$f(x) = f(x^*) + f'(x^*)(x - x^*) + \frac{f''(x^*)}{2!}(x - x^*)^2 + \dots$$

Si $f'(x^*) \neq 0$:

$$|y^* - y| \simeq \left|f'(x^*)\right| |x^* - x|$$

Si $f'(x^*) = 0$ hay que considerar la siguiente derivada.

Si ahora tenemos más de un dato: $y = f(x_1, x_2, \dots x_n)$; $y^* = f(x_1^*, x_2^*, \dots, x_n^*)$

Haciendo de nuevo el desarrollo de Taylor y cogiendo la primera derivada de cada término:

$$|y^* - y| \simeq \sum_{i=1}^{n} \left(\left| \frac{\partial f}{\partial x_i}(x_1^*, x_2^*, \dots, x_n^*) \right| |x_i - x_i^*| \right) \leq \|\nabla f(x^*)\|_2 \|x - x^*\|_2 \tag{1.17}$$

Donde la desigualdad viene de la desigualdad de Cauchy-Schwartz y el subíndice 2 indica que es la "norma 2", es decir, la norma de un vector tal y como se la conoce (raíz cuadrada de las componentes al cuadrado).

$$\nabla f(x^*) = \begin{vmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{vmatrix} \qquad x = \begin{vmatrix} x_1 \\ \vdots \\ x_n \end{vmatrix}$$

$$\vec{y} = \vec{f}(\vec{x}) \Rightarrow \|\vec{y}^* - \vec{y}\| \leq \|J_f(\vec{x}^*)\| \|\vec{x} - \vec{x}^*\|$$

Donde $J_f(\vec{x}^*)$ es el jacobiano de *f*.

La norma de una matriz A, $\|A\|$ se calcula haciendo la raíz cuadrada del radio espectral de la matriz por su traspuesta, que es el módulo mayor de los autovalores de la matriz:

$\lambda_1, \lambda_2, \dots, \lambda_n$ autovalores de la matriz A:

$\rho = \rho(A) = \lambda_i / \forall i, j \ \ |\lambda_i| > |\lambda_j|\,;\ i \neq j$ es el **radio espectral** de A.

$$\|A\| = \sqrt{\rho(A^t \cdot A)} \tag{1.18}$$

1.3 Inestabilidad de problemas y métodos numéricos

Vamos a empezar con algunos ejemplos:

Ejemplo 1.9:

Calcular la integral $I_{20} = \int_0^1 \frac{x^{20}}{x+6} dx$ usando la fórmula recurrente $I_k = \frac{1}{k} - 6I_{k-1}$ teniendo $I_0 = \ln\left(\frac{7}{6}\right)$.

Si usamos una calculadora científica normal, obtenemos (tras mucho teclear) el valor $I_{20} = -22875.75$. Sin embargo, hemos de notar que estamos evaluando una integral positiva en todo el intervalo de integración, con lo que el resultado debería ser positivo. Por otro lado, aunque no se vea a simple vista, este resultado en módulo es demasiado grande. El mismo cálculo, realizado con Matlab,

usando 16 dígitos de precisión, devuelve $I_{20} = 0.1690451452\ldots$ Este resultado puede parecer exacto, pero aún así, sigue sin serlo, ya que acotando la integral, por la de una función mayor:

$$I_{20} < \int_0^1 \frac{x^{20}}{6} dx = \frac{1}{21 \cdot 6} = \frac{1}{126} = 0.0079365$$

un valor menor que el obtenido, y si tiene que ser mayor que el valor real de la integral, está claro que ninguno de los dos resultados obtenidos es válido.

Este ejemplo refleja la incertidumbre en el resultado cuando manejamos números reales que al almacenarlos podemos hacer que se pierdan dígitos significativos.

■

Ejemplo 1.10:

Resolver el polinomio de segundo grado $x^2 + 1000x + 6.2521308 = 0$ usando 8 dígitos de resolución.

Podemos emplear la fórmula que siempre hemos usado para la ocasión: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, y obtenemos los resultados siguientes con una calculadora:

$$x_1 = -0.00625500$$

$$x_2 = -999.99375$$

Teóricamente, al operar con las raíces se obtiene:

$$\begin{aligned} x_1 + x_2 &= -\frac{b}{a} \\ x_1 x_2 &= \frac{c}{a} \end{aligned}$$

$$x_1 + x_2 = -1000.000005$$

$$x_1 x_2 = 6.2549609$$

En los cálculos realizados con Matlab (función roots):

$$x_1 = -0.00625216989$$

$$x_2 = -999.99374783$$

■

Ejemplo 1.11:

Se tienen los siguientes sistemas de ecuaciones:

$$\begin{pmatrix} 0.9999 & 1 \\ 2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 0.002 \\ 0.006 \end{pmatrix}$$

y

$$\begin{pmatrix} 0.99995 & 1 \\ 2 & 2 \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0.002 \\ 0.006 \end{pmatrix}$$

Se obtienen las siguientes soluciones:

$$x_1 = 10 \quad y_1 = -9.997$$

$$x_2 = 20 \quad y_2 = -19.997$$

Se observa que a pesar de variar muy poco un dato las soluciones cambian enormemente (en comparación con el cambio del dato).

■

A partir de los ejemplos podemos ver que hay diferentes comportamientos para diferentes tipos de problemas, ya sea por el método de resolución empleado o por otras causas.

Se dice que un **método** es **inestable** si para la mayoría de los casos da buenos resultados, pero en algunos casos, ya sea por la forma de los datos o por la forma de operarlos, se obtienen grandes diferencias (ejemplos 1.9 y 1.10).

Decimos que tenemos un **problema mal condicionado** si para pequeñas variaciones de los datos del problema, se obtienen grandes diferencias en los resultados independientemente del método o métodos empleados (ejemplo 3).

Se puede analizar el condicionamiento de un problema con un **número de condición relativo**, definido por

$$\boxed{N_{C_r} = \frac{|\text{Cambio relativo en la solucion del problema}|}{|\text{Cambio relativo en el dato de entrada}|}} \tag{1.19}$$

En base a este número de condición:

- ✏ Si es muy grande ($\gg 1$), entonces el problema es mal condicionado: la solución es muy sensible a pequeñas variaciones en los datos
- ✏ Si es próximo a 1, entonces el problema está bien condicionado: la solución no sufre grandes cambios con cambios pequeños en los datos

Si cuando resolvemos por un método un problema obtenemos un resultado diferente que la solución exacta, cabe plantearse si la diferencia viene de un método inestable o de un problema mal condicionado. Para ello, se realiza un análisis, que puede ser de cuatro tipos:

1. **Análisis progresivo de error**: partiendo de los errores en los datos, analizar, avanzando según los cálculos, los errores acumulados.
2. **Análisis regresivo de error**: una vez obtenido el resultado, ver de qué problema es solución exacta (en el ejemplo 1, cuando hacemos $x_1 + x_2$ y $x_1 x_2$ lo que hacemos es ver qué coeficientes son los de la ecuación que realmente se ha resuelto). Esto es, considerar la solución calculada como la solución exacta de otro problema perturbado. Es la forma más exacta de verificar la estabilidad de un método.
3. **Análisis estadístico de error**: se trata a los errores como variables aleatorias, con lo que se obtiene una función de distribución, a partir de la cual se calcula la varianza, la media, etc.
4. **Análisis intervalar de error o por aritmética de intervalos**: como no se conoce el dato preciso, se trabaja con un intervalo en el que sabemos que está ese dato, y operamos con los extremos izquierdo y derecho del mismo, de forma que sabemos que el resultado está dentro de un intervalo que es el que obtenemos como resultado.

Si tenemos el error regresivo (ε_r), el error progresivo (ε_p) se puede obtener como

$$\boxed{\varepsilon_p \lesssim N_{C_r} \cdot \varepsilon_r} \tag{1.20}$$

TEMA 2

Álgebra lineal numérica I

2.1 Introducción

En la mayoría de los problemas hay que resolver sistemas de ecuaciones lineales, aquí vamos a ver algunos métodos para resolverlas.

Empecemos por ver algunos conceptos básicos:

- ✏ Representación matricial de sistemas de ecuaciones:

$$\begin{array}{rcl} a_{11}x_1 + \ldots + a_{1n}x_n & = & b_1 \\ \vdots \qquad\qquad\quad & \vdots & \vdots \\ a_{n1}x_1 + \ldots + a_{nn}x_n & = & b_n \end{array}$$

Dado este sistema, hay que encontrar los valores de x_i para los que se cumplen las relaciones.

$$A = \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \ldots & a_{nn} \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Donde A se llama matriz del sistema, x vector de incógnitas y b vector de términos independientes.

En notación matricial, el sistema se escribe: $Ax = b$.

A la matriz $A|b$ se la llama **matriz ampliada**, y resulta de añadir el vector de términos independientes como la última columna de la matriz del sistema:

$$\begin{pmatrix} a_{11} & \ldots & a_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \ldots & a_{nn} & b_n \end{pmatrix}$$

2.1.1 Teorema de Rouché-Frobenius

Para la existencia de soluciones de un sistema de ecuaciones lineales se usa la siguiente regla:

$r(A) \neq r([A\|b])$	Incompatible (no existe solución)
$r(A) = r([A\|b]) = n$	Compatible determinado (una única solución)
$r(A) = r([A\|b]) < n$	Compatible indeterminado (infinitas soluciones)

Donde $r(A)$ representa el rango de la matriz A.

Como introducción vemos dos métodos de resolución que no se recomiendan por su poca eficiencia.

2.1.2 Regla de Cramer

Dado un sistema representado por su matriz del sistema A y por el vector de términos independientes b, las componentes de su vector de incógnitas x se obtienen de la siguiente forma:

$$x_j = \frac{det\left(A^j\right)}{det\left(A\right)}$$

Donde A^j es la matriz resultante de sustituir la columna j de la matriz A por el vector de términos independientes, y $det\left(A\right)$ representa el determinante de la matriz A.

Para hacer un determinante se puede usar el algoritmo

$$det\left(A\right) = \sum_{i=1}^{n} A_{ij} a_{ij} \qquad \text{para cada } j$$

Lo que requiere n determinantes de orden $n-1$, n productos y $n-1$ sumas. Por tanto en total son $(n!-1)$ sumas y

$$n! \sum_{i=1}^{n-1} \frac{1}{i!}$$

productos. Usando la definición de la función exponencial,

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} \Rightarrow e^1 = 1 + \sum_{i=1}^{\infty} \frac{1}{i!}$$

Por tanto, el cálculo de un determinante tiene un orden de

$$n! - 1 + n!\left(e-1\right) = en! - 1 = O\left(n!\right)$$

operaciones.

Para el método completo habrá que considerar el cálculo del otro determinante y el cociente entre ambos, que resulta en un orden de $O\left((n+1)!\right)$ operaciones.

Debido a esta eficiencia tan pobre no se recomienda su uso como método para resolver sistemas de ecuaciones.

2.1.3 Cálculo de la inversa

Dado que un sistema de ecuaciones se define matricialmente como

$$Ax = b$$

cabe suponer que la solución será

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

en caso de que exista A^{-1}. Definida como

$$A^{-1} = \frac{\left(adjunta\left(A\right)\right)^H}{det\left(A\right)}$$

Donde se define

$$A^H = A^* \equiv \text{traspuesta de la conjugada de } A \left(\text{para matrices complejas}\right)$$

$$adjunta\,(A) \equiv \text{sustituir los elementos por sus adjuntos}$$

Para que la matriz sea invertible debe ser $det\,(A) \neq 0$. En la bibliografía veremos esta condición como que A es una matriz "no singular". Nosotros la llamaremos "regular".

Es claro que computacionalmente tampoco sirve para resolver los sistemas.

2.2 Métodos directos para sistemas lineales

Fundamentos de Álgebra matricial en los apuntes de Álgebra de 1º de Luis Gimilio Barboza.

2.2.1 Introducción: métodos directos e indirectos

Se dice que un método para resolver un problema es directo si tras un número finito de operaciones con los datos del problema se obtiene una solución al problema, que, en general, será aproximada.

En los métodos iterativos se calcula una sucesión de vectores que son aproximaciones a la solución: $x^{(s)}$, y en el límite, tiende a la solución del problema: $\bar{x}$.

$$\bar{x} = \lim_{s\to\infty} x^{(s)}$$

Cuando se usa un ordenador para resolver un problema por un método de este tipo, no queda más remedio que truncar la solución, es decir, realizar un número finito de iteraciones para obtener una solución aproximada.

Normalmente los métodos iterativos se emplean sólo con problemas de gran tamaño, en los que las matrices están esparcidas o dispersas (*sparse* en inglés), es decir, que tienen la mayoría de sus elementos iguales a cero.

Un ejemplo en telecomunicaciones sería el flujo de una red de ordenadores a nivel internacional (Internet).

2.2.2 Métodos directos

El fundamento básico de estos métodos es que existen sistemas fáciles de resolver, así que trataremos de transformar el problema que tenemos a uno fácil de resolver.

Tenemos un sistema $Ax = b$ siendo A una matriz cuadrada. Tenemos varios casos:

- ✏ *A* es diagonal. En este caso, las soluciones son:

$$x_i = \frac{b_i}{a_{ii}} \quad i = 1 \ldots n \tag{2.1}$$

- ✏ *A* es ortogonal ($A^{-1} = A^t$):

$$x = A^{-1}b = A^t b \tag{2.2}$$

- ✏ *A* es triangular superior (todos los elementos por debajo de la diagonal principal son cero):

$$x_n = \frac{b_n}{a_{nn}} \quad x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=i+1}^{n} a_{ij}x_j\right) \qquad i = n-1 : -1 : 1 \text{ (Sustitución regresiva)} \tag{2.3}$$

✏ *A* es triangular inferior:

$$x_1 = \frac{b_1}{a_{11}} \quad x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j \right) \qquad i = 2 : n \text{ (Sustitución progresiva)} \tag{2.4}$$

Tanto la sustitución regresiva como la progresiva requieren un orden de n^2 operaciones.

Los códigos en Matlab para la sustitución regresiva y la sustitución progresiva lo vemos en los algoritmos siguientes.

Algoritmo 2.1: Sustitución regresiva

```
% Metodo de sustitucion regresiva para
% sistemas de ecuaciones lineales
function [x] = sustreg(A, b)
% ¿Es válido el sistema?
[M,N] = size(A);
tam = length(b);
if (M ~= N)|(tam ~= M)
   error('Las dimensiones del sistema no son válidas');
end;
% Vector de soluciones
x = zeros(tam,1);
x(tam) = b(tam)/A(tam,tam);
for k = tam-1:-1:1,
   suma = x(k+1:tam)'*A(k,k+1:tam)';
   x(k) = (b(k)-suma)/A(k,k);
end;
```

♦

Algoritmo 2.2: Sustitución progresiva

```
% Metodo de sustitucion progresiva para
% sistemas de ecuaciones lineales
function [x] = sustprog(A, b)
% ¿Es válido el sistema?
[M,N] = size(A);
tam = length(b);
if (M ~= N)|(tam ~= M)
   error('Las dimensiones del sistema no son válidas');
end;
% Vector de soluciones
x = zeros(tam,1);
x(1) = b(1)/A(1,1);
for k = 2:1:tam,
   suma = x(1:k-1)'*A(k,1:k-1)';
   x(k) = (b(k)-suma)/A(k,k);
end;
```

♦

2.2.3 Teorema de equivalencia de sistemas

Sea M una matriz invertible (regular). Entonces los sistemas $Ax = b$ y $MAx = Mb$ son equivalentes, es decir, tienen el mismo conjunto de soluciones.

Es necesario que M sea regular para que

$$M^{-1}MAx = M^{-1}Mb \Rightarrow Ax = b$$

Así pues, vamos a basar los métodos en pasar de $Ax = b$ a $\hat{A}x = \hat{b}$ equivalentes donde el último sistema sea fácil de resolver (alguno de la sección anterior).

$$\hat{A} = MA,\ \hat{b} = Mb$$

En los métodos que vamos a emplear no vamos a llegar a calcular una matriz M, simplemente vamos a transformar A en $\hat{A}$ mediante operaciones elementales.

Esto se puede ver de otra forma:

Si existe tal matriz M, entonces $A = M^{-1}\hat{A}$. De este modo, podemos plantearnos una factorización $A = N\hat{A}$ de forma que:

$$N\hat{A}x = b : \begin{cases} Ny = b & \text{es la solución intermedia} \\ \hat{A}x = y & \text{es la solución final} \end{cases}$$

2.2.4 Método de eliminación de Gauss

Trabajamos con la matriz ampliada del sistema y hacemos la transformación:

$$[A|b] \sim \left[\hat{A}|\hat{b}\right]$$

de forma que $\hat{A}$ sea triangular o diagonal.

$$I_n = \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix} = \begin{pmatrix} e_1 & e_2 & \dots & e_n \end{pmatrix}$$

I_n es lo que se conoce como **matriz identidad**, y e_i son los vectores unitarios de la base canónica (dispuestos como columnas).

Definimos la **matriz elemental de eliminación gaussiana** como:

$$E_{ji}(\alpha) = I + \alpha e_j e_i^t = \begin{array}{c} \\ j \rightarrow \end{array} \begin{array}{c} \begin{matrix} & & i & & \end{matrix} \\ \begin{matrix} & & \downarrow & & \end{matrix} \\ \begin{pmatrix} 1 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & \alpha & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 1 \end{pmatrix} \end{array} \tag{2.5}$$

Donde

$$det\left(E_{ij}\left(\alpha\right)\right) = \alpha$$

Entonces, al hacer $E_{ji}\left(\alpha\right)A = A + \alpha e_j e_i^t A$ nos quedaría la matriz de sistema siguiente (que además es equivalente a la original):

$$\begin{pmatrix} & \text{Igual que } A & \\ a_{j1} + \alpha a_{i1} & \dots & a_{jn} + \alpha a_{in} \\ & \text{Igual que } A & \end{pmatrix}$$

De forma que, si queremos anular el elemento $a_{j1} + \alpha a_{i1}$, tiene que ser $\alpha = -a_{j1}/a_{i1}$. Esta operación se llama **transformación de filaequivalencia "sumar a la fila j la fija i multiplicada por α"**.

Se define una **permutación de una matriz**, P_{ij} como el intercambio de la fila i con la fila j:

$$\begin{aligned} P_{ij} &= I_n + e_i\left(e_j^t - e_i^t\right) + e_j\left(e_i^t - e_j^t\right) \\ &= \begin{pmatrix} e_1 & e_2 & \dots & e_j & e_{i+1} & \dots & e_{j-1} & e_i & e_{j+1} & \dots & e_n \end{pmatrix} \\ &= P_{ji} \end{aligned}$$

$$det\left(P_{ij}\right) = -1$$

El determinante de la matriz resultante es el mismo que la original pero cambiado de signo.

$$P_{ij}A = \begin{matrix} \\ i \\ \\ j \\ \\ \end{matrix} \begin{pmatrix} & & \text{Igual} & & \\ a_{j1} & \dots & \dots & \dots & a_{jn} \\ & & \text{Igual} & & \\ a_{i1} & \dots & \dots & \dots & a_{in} \\ & & \text{Igual} & & \end{pmatrix}$$

Con esta permutación se puede mejorar la exactitud de los resultados escogiendo los pivotes que resulten en menos errores a la hora de dividir por ellos. Esto es porque cuando se divide por a_{ii} al calcular α conviene que a_{ii} en módulo no sea pequeño, por estabilidad numérica.

2.2.4.1 Triangulización

Primer paso:

$$\begin{array}{ll} P_{j_1 1} & j_1 \geq 1 \\ E_{n1}\left(\alpha_{n1}\right) \cdots E_{21}\left(\alpha_{21}\right) P_{j_1 1} & \alpha_{j1} = -\frac{a_{j1}^{(1)}}{a_{11}^{(1)}} \end{array}$$

$E_{ji}\left(\alpha_{ji}\right)$ representa una operación de suma de la fila i multiplicada por α_{ji} con la fila j.

La matriz resultante de esta operación:

$$\begin{pmatrix} a_{11} & \dots & a_{1n} & b_1 \\ a_{21} & \dots & a_{2n} & b_2 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \dots & a_{nn} & b_n \end{pmatrix} \sim \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} & b_2^{(1)} \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} & b_n^{(1)} \end{pmatrix}$$

Donde los superíndices (1) indican que son los resultados de una operación elemental con la primera fila de la matriz.

El siguiente paso sería:

$$\begin{aligned} &P_{j_2 2} \qquad j_2 \geq 2 \text{ (para no deshacer la operación anterior)} \\ &E_{n2}\left(\alpha_{n2}\right) \cdots E_{32}\left(\alpha_{32}\right) P_{j_2 2} \qquad\qquad \alpha_{j2} = -\frac{a_{j2}^{(2)}}{a_{22}^{(2)}} \end{aligned}$$

La matriz que nos queda:

$$\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \dots & b_2^{(2)} \\ \vdots & 0 & a_{33}^{(2)} & b_3^{(2)} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

El paso k-ésimo del método:

$$P_{j_k k} \;\rightarrow\; E_{nk}\left(\alpha_{nk}\right) \cdots E_{k+1,k}\left(\alpha_{k+1,k}\right) P_{j_k k} \quad j_k \geq k$$

Y al terminar se obtiene la siguiente matriz:

$$\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \dots & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & 0 & \ddots & \vdots & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn}^{(n)} & b_n^{(n)} \end{pmatrix}$$

El superíndice de la última fila es realmente $n-1$, así que hacemos una operación adicional que no sirve para nada y lo dejamos como n.

A lo largo de todo el proceso hemos multiplicado la matriz ampliada por matrices invertibles, por lo que el sistema que hemos obtenido al final tiene la misma solución que el original. La solución se obtiene por sustitución regresiva, como hemos visto antes.

Llamamos **pivotes** a los elementos que se quedan en la diagonal principal, $a_{ii}^{(i)}$.

El orden de este algoritmo es de $O\left(\frac{2n^3}{3}\right)$ operaciones.

2.2.4.2 Estrategias de pivotaje o formas de escoger los pivotes

Se usan para elegir los elementos que irán en la diagonal principal de forma que los errores de redondeo influyan lo menos posible en el error del resultado.

Se busca la estabilidad numérica del resultado.

Hay dos formas: pivotaje parcial y pivotaje total.

1. **Pivotaje parcial**:
   En el paso k: se coge de la misma columna el elemento cuyo módulo sea mayor de los que hay a partir de la fila k.
   $$\max_{j\geq k}\left|a_{jk}^{(k-1)}\right| = |a_{j_k k}|$$
   Se intercambian entonces las filas j y j_k.
   Este método no es numéricamente estable, ya que hay matrices para las cuales el crecimiento del error invalida el resultado. Son casos muy raros en la práctica, pero teóricamente es inestable.

2. **Pivotaje total**:
   En lugar de escoger el elemento mayor de la misma columna se escoge el mayor de la matriz que queda por triangulizar.
   $$\max_{n\geq i,j\geq k}\left|a_{ij}^{(k-1)}\right| = \left|a_{rs}^{(k-1)}\right|$$
   Se intercambian las filas r y j y las columnas s y k.
   Al intercambiar columnas se intercambian también las incógnitas, por lo que hay que mantener un vector de índices que controle el orden de las incógnitas.
   Este método es el más estable, numéricamente.

Ejemplo 2.1:

Vamos a ver el desarrollo por Gauss de un sistema:

$$\left[A^{(1)}|b^{(1)}\right] = \begin{pmatrix} 6 & -2 & 2 & 4 & 12 \\ 12 & -8 & 6 & -10 & 34 \\ 3 & -13 & 9 & 3 & 27 \\ -6 & 4 & 1 & -18 & -38 \end{pmatrix}$$

Haciendo pivotaje parcial y operando con la primera fila (que ahora es la que era antes la segunda):

$$\left[A^{(2)}|b^{(2)}\right] = \begin{pmatrix} 12.0000 & -8.0000 & 6.0000 & 10.0000 & 34.0000 \\ 0.0000 & 2.0000 & -1.0000 & -1.0000 & -5.0000 \\ 0.0000 & -11.0000 & 7.5000 & 0.5000 & 18.5000 \\ 0.0000 & 0.0000 & 4.0000 & -13.0000 & -21.0000 \end{pmatrix}$$

Pivotando de nuevo la segunda por la tercera fila:

$$\left[A^{(3)}|b^{(3)}\right] = \begin{pmatrix} 12.0000 & -8.0000 & 6.0000 & 10.0000 & 34.0000 \\ 0.0000 & -11.0000 & 7.5000 & 0.5000 & 18.5000 \\ 0.0000 & 0.0000 & 0.3636 & -0.9091 & -1.6364 \\ 0.0000 & 0.0000 & 4.0000 & -13.0000 & -21.0000 \end{pmatrix}$$

$$\left[A^{(4)}|b^{(4)}\right] = \begin{pmatrix} 12.0000 & -8.0000 & 6.0000 & 10.0000 & 34.0000 \\ 0.0000 & -11.0000 & 7.5000 & 0.5000 & 18.5000 \\ 0.0000 & 0.0000 & 4.0000 & -13.0000 & -21.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.2727 & 0.2727 \end{pmatrix}$$

Y ya tenemos la matriz del sistema triangulizada.

Si se hubiese hecho pivotaje total, en el primer paso, el 18 habría pasado a ser el primer elemento, y habría que llevar el orden de las incógnitas.

■

A continuación podemos ver el código en Matlab para el método de Gauss, con la posibilidad de usar pivotaje parcial o total.

Algoritmo 2.3: Método de Gauss para sistemas lineales

```
% Método de Gauss para Sistemas de Ecuaciones Lineales
function [x] = gauss(A,b,pivotaje)

if nargin<3
pivotaje = 'no';
   if nargin<2
      error('Faltan argumentos de entrada');
   end;
end;

% ¿Es válido el sistema?
[M,N] = size(A);
tam = length(b);
if (M ~= N)|(tam ~= M)
   error('Las dimensiones del sistema no son válidas');
end;

% Orden de las soluciones (puede cambiar con pivotaje total)
ordensolucion = 1:length(b);
for k = 1:tam,
   j = k;
   if (A(k,k) == 0)&(strcmp(pivotaje,'no') == 1)
     % Si el término a(k,k) es cero, pivotamos
      pivote = pivotemax(A,j,'parcial');
   end;
   pivote = pivotemax(A,j,pivotaje);
   if (pivote(1) ~= j)|(pivote(2) ~= k)
      coltemp = A(:,k); % Guardamos la columna a intercambiar
      A(:,k) = A(:,pivote(2));% Y hacemos el intercambio
      A(:,pivote(2)) = coltemp;
      filatemp = A(j,:);% Y ahora las filas
      A(j,:) = A(pivote(1),:);
      A(pivote(1),:) = filatemp;
      btemp = b(j); % Y el término independiente
      b(j) = b(pivote(1));
      b(pivote(1)) = btemp;
      soltemp = k; % Intercambiar las soluciones también
      ordensolucion(k) = ordensolucion(pivote(2));
      ordensolucion(pivote(2)) = soltemp;
   end;
   for j = k+1:tam,
      if (A(j,k) ~= 0)
         alfa = -A(j,k)/A(k,k);
         filatemp = A(k,:)*alfa;
         A(j,:) = A(j,:) + filatemp;
         btemp = b(k)*alfa;
        b(j) = btemp + b(j);
```

```
end;
end;
end;
% Matriz A triangular asi que hacemos
% sustitucion regresiva para obtener la solucion
xtemp = sustreg(A, b);
% Ahora se ordenan las soluciones
x = zeros(tam, 1);
for k = 1:tam,
x(ordensolucion(k)) = xtemp(k);
end;
```

♦

2.2.4.3 Diagonalización

Ahora la matriz del sistema junto con el vector de términos independientes se transforma en una matriz diagonal, para ello, se hacen ceros por encima del pivote.

$$\prod_{i \neq k} E_{ik}\left(\alpha_{ik}\right) P_{j_k k}$$

Las matrices quedan:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & & \ddots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix} \sim \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} & b_2^{(1)} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & a_{nn}^{(1)} & b_n^{(1)} \end{pmatrix} \sim \dots \sim \begin{pmatrix} a_{11}^{(n)} & 0 & \dots & 0 & b_1^{(n)} \\ 0 & a_{22}^{(n)} & \dots & 0 & b_2^{(n)} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & a_{nn}^{(n)} & b_n^{(n)} \end{pmatrix}$$

Esta vez sí que son *n* pasos para conseguir la matriz diagonal.

A priori no hay ninguna ventaja con respecto a la triangulización, ya que hay que realizar el doble de operaciones. Se usa cuando se quiere calcular la inversa de la matriz *A*:

$$Ax = I;\ A \begin{pmatrix} x^{(1)} & x^{(2)} & \dots & x^{(n)} \end{pmatrix}^t = \begin{pmatrix} e_1 & e_2 & \dots & e_n \end{pmatrix}^t$$

$$Ax^{(i)} = e_i \quad i = 1 : n \ (n \text{ sistemas})$$

$$[A|I] \overset{Gauss-Jordan}{\rightarrow} \left[D|I^{(n)}\right] \overset{Dividiendo\,por\,pivotes}{\rightarrow} \left[I|A^{-1}\right]$$

Este método también se conoce con el nombre de método de Gauss-Jordan.

2.2.4.4 Estrategias de escalado

Para evitar trabajar con números muy pequeños, se puede hacer un escalado de las ecuaciones o de las incógnitas:

$$D_1^{-1} A D_2 y = D_1^{-1} b$$

$$x = D_2^{-1} y$$

Siendo D_1 y D_2 matrices diagonales.

Estrategia de equilibrado: que los elementos de la matriz $D_1^{-1} A D_2$ estén en un mismo rango de valores.

2.2.5 Métodos de Crout y Doolittle

Se basan en la factorización LU de la matriz del sistema, de forma que quede el producto de una matriz triangular superior (U) y una triangular inferior (L):

$$A = LU = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{pmatrix} \tag{2.6}$$

En principio, al operarlas saldrían n^2 ecuaciones con $n^2 + 1$ incógnitas, lo cual no es nada práctico. Para evitarlo, se igualan los elementos de la diagonal principal de una de las dos matrices triangulares a uno ($l_{ii} = 1$ o $u_{ii} = 1$).

El **método de Crout** es el que iguala la diagonal principal de la matriz U a 1, mientras que el **método de Doolittle** es el que iguala la diagonal principal de la matriz L a 1.

Primero veamos algunos conceptos:

✏ ***Definición:*** dada una matriz $A\, n \times n$, se denominan **submatrices principales sucesivas** a las submatrices:

$$A_i = \begin{pmatrix} a_{11} & \dots & a_{1i} \\ \vdots & \ddots & \vdots \\ a_{i1} & \dots & a_{ii} \end{pmatrix} \qquad i = 1 : n$$

y **menores principales sucesivos** a los determinantes $det\,(A_i)$ para $i = 1 : n$.

Teorema 2.1. Dada una matriz cuadrada A, si $det\,(A_i) \neq 0 \quad \forall i = 1 : n-1$, entonces la matriz admite factorización LU (L matriz triangular inferior y U triangular superior) con $l_{ii} = 1$ o $u_{ii} = 1$. Es condición suficiente de matriz factorizable, no necesaria. Si la matriz es invertible y existe su factorización LU, los menores principales sucesivos de la matriz hasta el de orden $n-1$ son no nulos y la factorización es única. Es condición necesaria y suficiente.

Por tanto, si $det\,(A_i) = 0$ para algún i, entonces $det\,(A) = 0$ y/o A no es factorizable LU. Esto es, si encontramos algún menor principal sucesivo nulo y $det\,(A) \neq 0$, la matriz no es factorizable LU, mientras que si $det\,(A) = 0$, tendríamos que tratar de hacer la factorización para saber si es posible o no.

✏ ***Definición:*** una matriz P cuadrada de orden n es una **matriz de permutación** si se puede expresar como producto de matrices elementales P_{ij}.

Teorema 2.2. Sea A matriz cuadrada invertible, entonces existe una matriz de permutación P tal que la matriz PA admite factorización LU:

$$PA = LU$$

Ahora veamos los dos métodos antes mencionados:

2.2.5.1 Método de Crout

$$\begin{array}{llll} l_{i1} = a_{i1} & i = 1:n & u_{11} = 1 & u_{1i} = \frac{a_{1i}}{l_{11}} \quad i = 2:n \\ \\ j = 2:n \begin{cases} l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} & i = j:n \\ \\ u_{jj} = 1; \text{ Si } j < n: & u_{ji} = \left(a_{ji} - \sum_{k=1}^{j-1} l_{jk} u_{ki} \right) / l_{jj} \quad i = j+1:n \end{cases} \end{array} \tag{2.7}$$

2.2.5.2 Método de Doolittle

$$\begin{array}{llll} u_{1i} = a_{1i} & i = 1:n & l_{11} = 1 & l_{i1} = \frac{a_{i1}}{u_{11}} \quad i = 2:n \\ \\ j = 2:n \begin{cases} u_{ji} = a_{ji} - \sum_{k=1}^{j-1} l_{jk} u_{ki} & i = j:n \\ \\ l_{jj} = 1; \text{ Si } j < n: & l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) / u_{jj} \quad i = j+1:n \end{cases} \end{array} \tag{2.8}$$

✏ El **orden** de estos métodos es de $O\left(\frac{2n^3}{3}\right)$ operaciones. Por tanto, cabría preguntarse para qué nos va a servir un método que tiene el mismo número de operaciones que el método de Gauss. La razón para hacer la factorización LU es resolver diferentes sistemas con la misma matriz A y distintos vectores de términos independientes b, ya que con una sola factorización tenemos la de todos los sistemas.

A continuación vemos los códigos para realizar en Matlab los métodos de Crout y Doolittle.

Algoritmo 2.4: Método de Crout

```
% Factorización LU por el método de Crout
function [L,U] = crout(A)
if nargin<1
   error('No ha introducido la matriz de entrada');
else
   [cols filas]=size(A);
   if cols~=filas
      error('La matriz no es cuadrada');
   else
      L=zeros(cols);
      U=zeros(cols);
      U(1,1)=1;
      L(1,1)=A(1,1);
      for i=2:cols
         U(i,i)=1;
         L(i,1)=A(i,1);
```

```
            U(1,i)=A(1,i)/L(1,1);
         end;
         for j=2:cols
            for i=j:cols
               sumal=0;
               sumau=0;
               for k=1:j-1
                  if (L(i,k)~=0)&(U(k,j)~=0)
                     sumal=sumal+L(i,k)*U(k,j);
                  end;
                  if (L(j,k)~=0)&(U(k,i)~=0)
                     sumau=sumau+L(j,k)*U(k,i);
                  end;
               end;
               L(i,j)=A(i,j)-sumal;
               if (j<cols)&(i>j)
                  U(j,i)=(A(j,i)-sumau)/L(j,j);
               end;
            end;
         end;
      end;
   end;
```

♦

Algoritmo 2.5: Método de Doolittle

```
% Factorización LU por el método de Doolittle
function [L,U] = doolittle(A)
if nargin<1
   error('No ha introducido la matriz de entrada');
else
   [cols filas]=size(A);
   if cols~=filas
      error('La matriz no es cuadrada');
   else
      L=zeros(cols);
      U=zeros(cols);
      L(1,1)=1;
      U(1,1)=A(1,1);
      for i=2:cols
         L(i,i)=1;
         U(1,i)=A(1,i);
         L(i,1)=A(i,1)/U(1,1);
      end;
      for j=2:cols
         for i=j:cols
            sumal=0;
            sumau=0;
            for k=1:j-1
               if (U(k,i)~=0)&(L(j,k)~=0)
                  sumal=sumal+U(k,i)*L(j,k);
               end;
```

```
                    if (U(k,j)~=0)&(L(i,k)~=0)&(i~=j)
                        sumau=sumau+U(k,j)*L(i,k);
                    end;
                end;
                U(j,i)=A(j,i)-sumal;
                if (j<cols)&(i>j)
                    L(i,j)=(A(i,j)-sumau)/U(j,j);
                end;
            end;
        end;
    end;
end;
```

♦

2.2.6 Método de Cholesky

Se utiliza cuando la matriz del sistema es simétrica: $A = A^t$.

$$A = LL^t$$

$$A = \begin{pmatrix} l_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ l_{n1} & \dots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & \dots & l_{n1} \\ \vdots & \ddots & \vdots \\ 0 & \dots & l_{nn} \end{pmatrix}$$

$$\boxed{\begin{array}{l} l_{11} = \sqrt{a_{11}} \qquad l_{i1} = \frac{a_{i1}}{l_{11}} \qquad i = 2:n \\ \\ j = 2:n \quad \left\{ \begin{array}{l} l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2} \\ \\ \text{Si } j < n; \quad l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) / l_{jj} \qquad i = j+1:n \end{array} \right. \end{array}} \tag{2.9}$$

El orden del método de Cholesky es de $O\left(\frac{n^3}{3}\right)$ operaciones, la mitad que los anteriores.

Teorema 2.3. Una matriz A admite factorización Cholesky si y sólo si es definida positiva.

✏ ***Definición:*** una matriz A simétrica es definida positiva si $x^t Ax > 0 \quad \forall x \neq \overline{0}$. O bien, es definida positiva si, y sólo si, sus menores principales sucesivos son positivos: $det\,(A_i) > 0 \quad i = 1:n$. Es lo mismo que decir que sus autovalores son todos positivos.

Teorema 2.4. Una matriz A semidefinida positiva admite factorización Cholesky para L triangular inferior con todos sus elementos $l_{ij} \in \mathbb{R}$.

Es condición por tanto para que sea factorizable con elementos reales

$$l_{ii}^2 = \frac{det\,(A_i)}{det\,(A_{i-1})} \geq 0 \qquad l_{11}^2 = a_{11} \geq 0$$

✏ Si la matriz *A* es simétrica e indefinida, se puede usar una factorización LDL^t:

$$A = LDL^t$$

$$A = LU = LD\hat{U} \overset{A = A^t}{=} \hat{U}^t DL^t \Rightarrow \hat{U} = L^t$$

Con D diagonal y $\hat{U}$ tal que $\hat{u}_{ii} = 1$, si los menores principales sucesivos son no nulos.

El algoritmo es:

$$d_{11} = a_{11}$$

$$j = 2 : n \quad \begin{cases} v_i = a_{ji} - \sum_{k=1}^{i-1} v_k l_{ik} \quad l_{ji} = \frac{v_i}{d_{ii}},\ i = 1 : j-1 \\ d_{jj} = a_{jj} - \sum_{k=1}^{j-1} v_k l_{jk} \end{cases} \tag{2.10}$$

Siendo v un vector auxiliar.

No toda matriz simétrica indefinida puede factorizarse de esta forma, por ejemplo:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Sin embargo, para toda matriz simétrica indefinida existe una matriz de permutación P tal que:

$$PAP^t = LBL^t$$

Para B simétrica y diagonal por bloques 1×1 o 2×2. Se hace por los métodos de Aasen y Bunch-Kaufman, pero no vamos a desarrollarlos.

Ejemplo 2.2:

Dada la matriz

$$A = \begin{pmatrix} \alpha & 2 & 6 & 4 \\ 2 & \alpha & 1 & 2 \\ 0 & 1 & \alpha & 2 \\ 0 & 0 & 1 & \alpha \end{pmatrix}$$

determinar $\alpha \in \mathbb{R}$ para que *A* admita factorización LU.

Solución:

$$det\,(A_i) \neq 0 \qquad i = 1 : 3$$

$$det\,(A_1) = \alpha$$

$$det\,(A_2) = \alpha^2 - 4$$

$$det\,(A_3) = \alpha^3 - 5\alpha + 12$$

Igualando a cero los resultados, nos quedan los siguientes valores para los que se anulan los menores:

$$\begin{aligned}\alpha &= 0\\ \alpha &= \pm 2\\ \alpha &= -3\end{aligned}$$

Vemos que *A* es factorizable LU $\forall \alpha \in \mathbb{R} - \{0, -2, 2, -3\}$

Ahora cabe preguntarse si es factorizable para alguno de esos valores. Para ello, utilizamos la segunda parte del teorema 1:

$$det\,(A) = \alpha\left(\alpha^3 - 7\alpha + 14\right)$$

Y vemos que, para los valores encontrados:

$$\begin{array}{lll}\alpha = 0 & \Rightarrow & det\,(A) = 0\\ \alpha = 2 & \Rightarrow & det\,(A) \neq 0\\ \alpha = -2 & \Rightarrow & det\,(A) \neq 0\\ \alpha = -3 & \Rightarrow & det\,(A) \neq 0\end{array}$$

Por tanto, para los tres últimos valores no es factorizable, ya que sus menores son nulos. ■

Ejemplo 2.3:

Dada la matriz

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

hacer factorización Cholesky en función de α y concretar los valores de α para los que es posible la factorización en aritmética real.

Se ve que la matriz es simétrica, así que podemos expresarla:

$$A = \begin{pmatrix} l_{11} & 0 & 0 & 0\\ l_{21} & l_{22} & 0 & 0\\ l_{31} & l_{32} & l_{33} & 0\\ l_{41} & l_{42} & l_{43} & l_{44}\end{pmatrix} L^t$$

Aplicando el algoritmo de la factorización, llegamos a los siguientes valores:

$$l_{11} = 2$$

$$l_{21} = \tfrac{\alpha}{2} \quad l_{22} = \tfrac{\sqrt{12-\alpha^2}}{2}$$

$$l_{31} = 2 \qquad l_{32} = 0 \qquad l_{33} = 1$$

$$l_{41} = 0 \quad l_{42} = \tfrac{2}{\sqrt{12-\alpha^2}} \quad l_{43} = -1 \quad l_{44} = \sqrt{\tfrac{20-2\alpha^2}{12-\alpha^2}}$$

Para que sea factorizable tiene que ser definida positiva, los determinantes son fáciles de calcular, teniendo en cuenta:

$$det\left(A_i\right) = \left(det\left(L_i\right)\right)^2 = l_{11}^2 l_{22}^2 \cdots l_{ii}^2$$

Por tanto,

$$\begin{gathered} det\left(A_1\right) = 4 > 0 \\ det\left(A_2\right) = 12 - \alpha^2 \\ det\left(A_3\right) = det\left(A_2\right) \end{gathered}$$

Y el determinante de la matriz del sistema:

$$det\left(A\right) = 20 - 2\alpha^2$$

Para que sea factorizable:

$$\begin{gathered} 20 - 2\alpha^2 > 0 \\ 12 - \alpha^2 > 0 \end{gathered}$$

Que tiene dos soluciones, $-\sqrt{10} < \alpha < \sqrt{10}$ y $-\sqrt{12} < \alpha < \sqrt{12}$ por lo que cogemos la más restrictiva. Además, si $\alpha = \pm\sqrt{10}$ también es factorizable porque los menores sucesivos son no nulos.

Así pues, la matriz es factorizable por Cholesky si y sólo si:

$$-\sqrt{10} \leq \alpha \leq \sqrt{10}$$

■

Ejemplo 2.4:

Determinar los valores de α para los que *A* es factorizable LU y resolver el sistema $A\left(\alpha\right) \cdot x = b$ para $\alpha = 2$, siendo *A* y *b*:

$$A\left(\alpha\right) = \begin{pmatrix} \alpha & 3 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & \alpha \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 3 \\ 5 \\ -1 \end{pmatrix}$$

Empecemos por calcular los menores sucesivos de *A*:

$$\begin{gathered} det\left(A_1\right) = \alpha \\ det\left(A_2\right) = 2\alpha - 3 \\ det\left(A_3\right) = 2\alpha - 4 \end{gathered}$$

Con lo que se obtiene que es factorizable (en condición suficiente) para $\alpha \in \mathbb{R} - \left\{0, \frac{3}{2}, 2\right\}$.

Veamos si para alguno de esos valores es factorizable.

$$det\left(A\right) = \left(2\alpha - 1\right)\left(\alpha - 2\right)$$

Así, para cada uno de los valores de α:

$$\begin{array}{rcc} \alpha = 0 & \Rightarrow & \text{Invertible, no existe factorización} \\ \alpha = \frac{3}{2} & \Rightarrow & \text{Invertible y tampoco es factorizable} \\ \alpha = 2 & \Rightarrow & \text{No es invertible} \end{array}$$

En el último caso, como la matriz no es invertible, el teorema no nos asegura nada, por lo que tenemos que intentar factorizar:

$$A(2) = \begin{pmatrix} 2 & 3 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix} Crout = \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ l_{31} & l_{32} & l_{33} & \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} \begin{pmatrix} 1 & u_{12} & u_{13} & u_{14} \\ & 1 & u_{23} & u_{24} \\ & & 1 & u_{34} \\ & & & 1 \end{pmatrix}$$

Y ahora hallamos los valores de los elementos:

Para la primera columna de *L*:

$$\begin{aligned} l_{11} &= 2 \\ l_{21} &= 1 \\ l_{31} &= 0 \\ l_{41} &= 0 \end{aligned}$$

La primera fila de *U*:

$$\begin{aligned} l_{11}u_{12} = 3; &\quad u_{12} = \tfrac{3}{2} \\ l_{11}u_{13} = 2; &\quad u_{13} = 1 \\ l_{11}u_{14} = 1; &\quad u_{14} = \tfrac{1}{2} \end{aligned}$$

Segunda columna de *L*:

$$\begin{aligned} l_{21}u_{12} + l_{22} = 2; &\quad l_{22} = \tfrac{1}{2} \\ l_{31}u_{12} + l_{32} = 1; &\quad l_{32} = 1 \\ l_{41}u_{12} + l_{42} = 0; &\quad l_{42} = 0 \end{aligned}$$

Segunda fila de *U*:

$$\begin{aligned} l_{21}u_{13} + l_{22}u_{23} = 2; &\quad u_{23} = 2 \\ l_{21}u_{14} + l_{22}u_{24} = 1; &\quad u_{24} = 1 \end{aligned}$$

Tercera columna de *L*:

$$\begin{aligned} l_{31}u_{13} + l_{32}u_{23} + l_{33} = 2; &\quad l_{33} = 0 \\ l_{41}u_{13} + l_{42}u_{23} + l_{43} = 1; &\quad l_{43} = 1 \end{aligned}$$

Tercera fila de *U*:

$$l_{31}u_{14} + l_{32}u_{24} + l_{33}u_{34} = 1; \quad 0 \cdot u_{34} = 0$$

El resultado de esta última ecuación es que u_{34} puede valer lo que uno quiera, el problema vendría al intentar resolverlo con un ordenador, ya que no soporta la división por cero y no trabaja con parámetros. Hacemos $\beta = u_{34}$.

Por último, la última columna de *L*:

$$l_{41}u_{14} + l_{42}u_{24} + l_{43}\beta + l_{44} = 2; \quad l_{44} = 2 - \beta$$

Con este resultado, lo que ocurre es que tenemos infinitas factorizaciones posibles, por lo que la matriz *A* es factorizable si, y sólo si:

$$\alpha \in \mathbb{R} - \left\{0, \frac{3}{2}\right\}$$

Veamos ahora la segunda parte del problema: resolver el sistema para $\alpha = 2$:

$$A(2) = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 1 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2-\beta \end{pmatrix} \begin{pmatrix} 1 & \frac{3}{2} & 1 & \frac{1}{2} \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & \beta \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

La solución se obtiene al resolver:

$$\begin{aligned} Ly &= b \\ Ux &= y \end{aligned}$$

Habiendo operado, se obtiene:

$$\begin{array}{c} y = \begin{pmatrix} \frac{1}{2} & 5 & s & t \end{pmatrix} \\ x = \begin{pmatrix} -9-3t & 7+3t & -1-2t & t \end{pmatrix} \end{array} \qquad s + (2-\beta)\, t = -1$$

■

2.2.7 Métodos de ortogonalización

✏ ***Definición:*** Se dice que una matriz Q es **ortogonal** si $Q^t Q = Q Q^t = I$.

$$\|Qx\|_2^2 = x^t Q^t Q x = x^t x = \|x\|_2^2$$

La ventaja desde el punto de vista numérico de las transformaciones ortogonales es que son normas isométricas (se mantiene dicha norma), por tanto, estas transformaciones son estables numéricamente. Por contra, los métodos basados en semejanza o equivalencia de sistemas empeoran el condicionamiento del problema en el proceso.

Pregunta: ¿Y entonces porqué no hacemos todo con transformaciones ortogonales y nos olvidamos de Gauss, Cholesky y los demás?

Respuesta: Porque las transformaciones ortogonales son menos eficientes que el resto de los métodos, si tenemos una matriz A $m \times n$ a ortogonalizar, el número de operaciones es $O\left(3n^2\left(m - \frac{n}{3}\right)\right)$, por lo que en el caso de matrices cuadradas, hay que hacer 3 veces más operaciones que por el método de triangulización de Gauss.

Este método se suele usar cuando la estabilidad numérica es prioritaria.

Hay dos métodos que vamos a estudiar: el método de Givens y el método de Householder.

2.2.7.1 Método de Givens

Se define una **matriz de Givens**:

$$G_{pq} = \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & \cos\theta & \dots & \operatorname{sen}\theta & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -\operatorname{sen}\theta & \dots & \cos\theta & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix} \begin{matrix} \\ \\ p \\ \\ q \\ \\ \\ \end{matrix} \\ \begin{matrix} & & p & & q & & \end{matrix} \tag{2.11}$$

En adelante, consideraremos $c = \cos\theta$ y $s = \operatorname{sen}\theta$.

La premultiplicación por G_{pq}^t equivale a una rotación de θ radianes en el plano de coordenadas (p, q) en sentido contrario al de las agujas del reloj. Únicamente se cambian las filas *p* y *q*.

La diferencia de esta matriz con la que se usa en el método de Gauss es que la de Givens es ortogonal, mientras que la de Gauss no lo es.

$$G_{pq}^t x = \begin{pmatrix} x_1 & \dots & x_{p-1} & c \cdot x_p - s \cdot x_p & x_{p+1} & \dots & x_{q-1} & s \cdot x_p + c \cdot x_q & \dots & x_n \end{pmatrix}^t$$

Si queremos anular el q-ésimo elemento:

$$sx_p + cx_q = 0$$

Entonces, usando

$$\frac{s}{c} = -\frac{x_q}{x_p}$$

$$c^2 + s^2 = 1$$

Llegamos a que los valores de s y c son

$$\boxed{s = \frac{-x_q}{\sqrt{x_p^2 + x_q^2}} \qquad c = \frac{x_p}{\sqrt{x_p^2 + x_q^2}}} \tag{2.12}$$

El **método de Givens** consiste en sucesivas transformaciones de rotación usando matrices de Givens hasta reducir la matriz *A* rectangular $m \times n$ a una triangular. Si $m \geq n$:

$$G_n^t \cdots G_1^t A = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \quad A \in \mathbb{R}^{m \times n} \quad R_1 \in \mathbb{R}^{n \times n}$$

Siendo R_1 triangular superior.

La matriz $Q = G_1 \cdots G_n$ no se suele calcular y sólo se almacena un número por rotación en la posición anulada. Se suele usar este método con matrices dispersas.

Ejemplo 2.5:

Dada la matriz

$$A = \begin{pmatrix} 3 & 5 & 3 \\ 4 & 0 & 1 \\ 0 & 3 & 1.35 \end{pmatrix}$$

La primera rotación tiene que anular el elemento $(2,1)$ con el $(1,1)$, por lo que

$$s = \frac{-4}{\sqrt{3^2+4^2}} = -0.8 \qquad c = \frac{3}{\sqrt{3^2+4^2}} = 0.6$$

Y la matriz de Givens es

$$G_1 = \begin{pmatrix} 0.6 & -0.8 & 0 \\ 0.8 & 0.6 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Así que, al premultiplicar A por la matriz de Givens traspuesta tenemos

$$G_1^t A = \begin{pmatrix} 5 & 3 & 2.6 \\ 0 & -4 & -1.8 \\ 0 & 3 & 1.35 \end{pmatrix}$$

Ahora la reducimos finalmente a triangular eliminando el elemento $(3,2)$ con el elemento $(2,2)$:

$$s = \frac{-3}{\sqrt{3^2+4^2}} = -0.6 \qquad c = \frac{-4}{\sqrt{3^2+4^2}} = -0.8$$

Así que la segunda matriz de Givens es

$$G_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.8 & -0.6 \\ 0 & 0.6 & -0.8 \end{pmatrix}$$

Y la premultiplicación nos deja

$$R = G_2^t G_1^t A = \begin{pmatrix} 5 & 3 & 2.6 \\ 0 & 5 & 2.25 \\ 0 & 0 & 0 \end{pmatrix}$$

Además, la matriz Q ortogonal es

$$Q = G_1 G_2 = \begin{pmatrix} 0.6 & 0.64 & 0.48 \\ 0.8 & -0.48 & -0.36 \\ 0 & 0.6 & -0.8 \end{pmatrix}$$

■

A continuación se incluye el código en Matlab para implementar el método de Givens.

Algoritmo 2.6: Método de Givens para sistemas lineales

```
% Metodo de Givens
% Uso: [Ab] = givens(A)
function [Q, R] = givens(A)
[m n] = size(A);
Q = eye(m);
R = A;
for i=1:n
   for k=i+1:m
```

```
            if (R(k,i) ~= 0)
               raiz = sqrt(R(k,i)^2 + R(i,i)^2);
               s = -R(k,i)/raiz;
               c = R(i,i)/raiz;
               G = eye(m); % Matriz de rotacion
               G(i,i) = c;
               G(k,k) = c;
               G(k,i) = -s;
               G(i,k) = s;
               Q = Q*G;    % Matriz ortogonal
               R = G'*R;   % Matriz triangular inferior
            end;
         end;
      end;
```

♦

2.2.7.2 Método de Householder

Matriz de Householder:

$$H = I - \frac{2}{v^t v} v v^t \tag{2.13}$$

Donde v es un vector no nulo, de forma que el producto vv^t es una matriz cuadrada (simétrica $n \times n$ y de rango 1) y $v^t v$ un número.

La matriz H es simétrica y ortogonal:

$$HH^t = H^2 = I + 4\frac{vv^t vv^t}{(v^t v)^2} - 4\frac{vv^t}{v^t v} = I$$

Fijado un vector $x \in \mathbb{R}^n$, la elección de v permite obtener Hx con varios elementos nulos:

$$\boxed{v^{(k)} = \begin{pmatrix} 0 & \dots & 0 & x_k + signo\,(x_k)\,\sigma & x_{k+1} & \dots & x_n \end{pmatrix}^t} \tag{2.14}$$

Donde

$$\boxed{\sigma = \sqrt{\sum_{j=k}^{n} x_j^2}} \tag{2.15}$$

$$Hx = x - \frac{2vv^t x}{v^t v} = x - \underbrace{\frac{1}{\frac{2v^t x}{v^t v}}}\, v = \begin{pmatrix} x_1 \\ \vdots \\ x_{k-1} \\ x_k \\ x_{k+1} \\ \vdots \\ x_n \end{pmatrix} - \begin{pmatrix} 0 \\ \vdots \\ 0 \\ x_k - signo\,(x_k)\,\sigma \\ x_{k+1} \\ \vdots \\ x_n \end{pmatrix}$$

Si demostramos que $\frac{2v^t x}{v^t v} = 1$, podemos hacer ceros a partir de donde queramos.

El multiplicar σ por el signo de x_k es para evitar diferencias cancelativas.

El **método de Householder** consiste en la sucesiva realización de transformaciones (un total de n) de reflexión utilizando matrices de Householder hasta reducir $A \in \mathbb{R}^{m\times n}$ a una matriz triangular. Si $m \geq n$:

$$H_n \cdots H_1 A = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \quad R_1 \in \mathbb{R}^{n\times n} \text{ triangular superior}$$

La diferencia con el método de Givens es que la matriz H, al premultiplicar a A, hace que se anulen todos los elementos de la columna que necesite, no elemento a elemento.

Podemos ilustrar los primeros pasos del método a continuación:

1. Obtención del primer vector $v^{(1)}$:

$$v^{(1)} = \begin{pmatrix} v_1^{(1)} & a_{21} & \dots & a_{n1} \end{pmatrix}^t$$

donde

$$v_1^{(1)} = a_{11} + signo\,(a_{11}) \sqrt{\sum_{j=1}^{n} a_{j1}^2} = a_{11} + signo\,(a_{11})\, \|A\,(:,1)\|_2$$

(siendo $\|A\,(:,1)\|_2$ la norma 2 de la primera columna de A por debajo de la diagonal principal)

2. Calcular la primera matriz de Householder, H_1:

$$H_1 = I - 2\frac{v^{(1)} v^{(1)^t}}{v^{(1)^t} v^{(1)}}$$

3. Con H_1 se hacen ceros por debajo de la diagonal en la primera columna

$$A_2 = H_1 A$$

4. Se obtiene $v^{(2)}$:

$$v^{(2)} = \begin{pmatrix} 0 & v_2^{(2)} & a_{32} & \dots & a_{n2} \end{pmatrix}^t$$

siendo esta vez

$$v_2^{(2)} = a_{22} + signo\,(a_{22}) \sqrt{\sum_{j=2}^{n} a_{j2}^2} = a_{22} + signo\,(a_{22})\, \|A\,(2:m,2)\|_2$$

5. Y de la misma forma que antes, H_2:

$$H_2 = I - 2\frac{v^{(2)} v^{(2)^t}}{v^{(2)^t} v^{(2)}}$$

6. Se vuelve a multiplicar, ahora A_2, para hacer ceros por debajo de la diagonal en la segunda columna

$$A_3 = H_2 A_2$$

7. Es decir, que en general el vector $v^{(i)}$ se obtiene como

$$\boxed{v^{(i)} = \begin{pmatrix} 0 & \dots & v_i^{(i)} & a_{i+1,i} & \dots & a_{ni} \end{pmatrix}^t} \tag{2.16}$$

y la componente i del vector

$$\boxed{v_i^{(i)} = a_{ii} + signo\,(a_{ii}) \sqrt{\sum_{j=i}^{n} a_{ji}^2} = a_{ii} + signo\,(a_{ii})\, \|A\,(i:m,i)\|_2} \tag{2.17}$$

8. Y posteriormente se calcula la matriz de Householder

$$\boxed{H_i = I - 2\frac{v^{(i)} \cdot v^{(i)^t}}{v^{(i)^t} \cdot v^{(i)}}} \tag{2.18}$$

El orden del método es de $O\left(2n^2\left(m - \frac{n}{3}\right)\right)$ operaciones, por lo que es preferible a Givens si la matriz tiene pocos ceros (es densa), pero si lo usamos para matrices dispersas, hacemos operaciones con ceros innecesarias.

Ejemplo 2.6:

Resolver el siguiente sistema usando el método de Householder:

$$[A|b] = \begin{pmatrix} 63 & 41 & -88 & 1 \\ 42 & 60 & 51 & 10 \\ 0 & -28 & 56 & 5 \\ 126 & 82 & -71 & 2 \end{pmatrix}$$

$$v^{(1)} = \begin{pmatrix} 63 + \sqrt{63^2 + 42^2 + 126^2} \\ 42 \\ 0 \\ 126 \end{pmatrix} = \begin{pmatrix} 210 \\ 42 \\ 0 \\ 126 \end{pmatrix}$$

$$v^{(2)} = \begin{pmatrix} 0 \\ 30.8 + \sqrt{30.8^2 + 28^2 + 5.6^2} \\ -28 \\ -5.6 \end{pmatrix}$$

■

Algoritmo 2.7: Método de Householder

```
% Metodo de Householder
% Uso: [Ab] = househ(A, b)
function [Ab] = househ(A, b)
[m n] = size(A);
Ab = [A b];
if m<=n,
   nl = m-1;
else
   nl = n;
end;
for i=1:nl
```

```
      v = Ab(i:m,i);
      nor = norm(v);
      if nor > 10*eps;
         signo = sign(v(1));
         if abs(v(1)) < 10*eps,
            signo = 1;
         end;
         v(1) = v(1) + signo*nor;
        % La siguiente línea está dividida en 2
         Ab(i:m,i:n+1)=Ab(i:m,i:n+1)-
           (2/(v'*v)*v)*(v'*Ab(i:m,i:n+1));
      end;
   end;
```

♦

2.3 Sistemas sobredeterminados y pseudoinversas

2.3.1 Sistemas sobredeterminados

Si tenemos un sistema de ecuaciones lineales de la forma

$$Ax = b, \qquad A \in \mathbb{R}^{m\times n}, \qquad m > n$$

Usualmente este tipo de sistemas no tiene solución, ¿qué hacemos pues?

Tratamos de calcular $\overline{x}$ tal que minimice la norma del vector residual:

$$\boxed{x/\min_x \|Ax-b\|_2^2} \tag{2.19}$$

Teorema 2.5. El vector $\overline{x}$ es solución de mínimos cuadrados

$$\min_{x\in\mathbb{R}^n} \|Ax-b\|_2^2 \qquad \text{con } A \in \mathbb{R}^{m\times n}, \quad b \in \mathbb{R}^m \text{ y } m \geq n$$

si, y sólo si, $\overline{x}$ es solución del sistema de ecuaciones normales:

$$A^tAx = A^tb$$

Si la matriz A^tA es invertible, es decir, que A tiene rango completo por columnas, la solución es única y es

$$\overline{x} = \left(A^tA\right)^{-1}A^tb$$

Ejemplo 2.7:

Hemos reducido una matriz por el método de Householder, de forma que se tiene el sistema

$$A = \begin{pmatrix} -4.3589 & -1.8353 & -5.7354 & -2.2942 \\ 0 & 3.2606 & 1.6787 & 4.8425 \\ 0 & 0 & 2.2994 & -2.2994 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad b = \begin{pmatrix} -5.5060 \\ 3.9547 \\ 2.0798 \\ 0.1002 \\ 3.0652 \\ -3.3636 \end{pmatrix}$$

Tomamos sólo la parte de la matriz A que no tiene filas nulas:

$$B^t = \begin{pmatrix} -4.3589 & -1.8353 & -5.7354 & -2.2942 \\ 0 & 3.2606 & 1.6787 & 4.8425 \\ 0 & 0 & 2.2994 & -2.2994 \end{pmatrix}$$

y la del vector b correspondiente a las filas que hemos tomado para B:

$$c = \begin{pmatrix} -5.5060 \\ 3.9547 \\ 2.0798 \end{pmatrix}$$

Ahora el sistema a resolver es

$$B^t x = c, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

Si hacemos $x_4 = 0$ (también podría ser $x_3 = 0$), tenemos la **solución básica de mínimos cuadrados**, que resulta ser

$$x_B = \begin{pmatrix} -0.2416 \\ 0.7472 \\ 0.9045 \\ 0 \end{pmatrix}$$

Su norma es

$$\|x_B\|_2 \simeq 1.1978$$

En caso de hacer $x_3 = 0$, tenemos

$$x_{B'} = \begin{pmatrix} 0.6629 \\ 2.5562 \\ 0 \\ -0.9045 \end{pmatrix}$$

Y la norma es

$$\|x_{B'}\|_2 \simeq 2.7913$$

Como las filas de B^t son linealmente independientes, el sistema $B^t x = c$ es incompatible indeterminado, y la solución será aquella que tenga norma 2 menor.

De todos los valores de x que verifiquen la ecuación quiero el que sea $\min \|x\|_2$ (x_{LS}). Para ello se usan las ecuaciones normales de 2ª clase:

$$x = By \Rightarrow B^t B y = c$$

Dado que $B^t B$ es definida positiva (B^t tiene rango completo por filas y B por columnas), entonces

$$y = \left(B^t B\right)^{-1} c \Rightarrow x_{LS} = B \left(B^t B\right)^{-1} c$$

Para nuestra matriz será

$$x_{LS} = \begin{pmatrix} -0.2913 \\ 06477 \\ 0.9543 \\ 0.0498 \end{pmatrix}$$

Y la norma de la solución de mínimos cuadrados es

$$\|x_{LS}\|_2 \simeq 1.19056$$

■

2.3.1.1 Método de las ecuaciones normales

Consiste en aplicar el método de Cholesky al sistema

$$A^tAx = A^tb$$

Sin embargo, este es un problema mal condicionado, así que normalmente se usa el método basado en transformaciones ortogonales (Givens, Householder).

Al hacer una transformación ortogonal, de forma que $QR = A$ y tal que

$$A = QR = \left(\underbrace{Q_1}_{r} \ \underbrace{Q_2}_{m-r} \right) \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = Q_1R_1$$

- Q_1 es base ortonormal de $\Re(A)$ (subespacio imagen formado por las columnas de A)
- Q_2 es base ortonormal de $\aleph\left(A^t\right)$ (núcleo de A^t)

Y se tiene que

$$A^tAx = A^tb \Rightarrow R_1^tQ_1^tQ_1R_1x = A^tb$$

2.3.1.2 Método Gram-Schmidt clásico/modificado

Con este método se trata de calcular directamente la factorización Q_1R_1 de A, sin tocar el vector b

$$A = Q_1R_1 \qquad \text{con} \quad A \in \mathbb{R}^{m\times n}, \quad Q_1 \in \mathbb{R}^{m\times n}, \quad R_1 \in \mathbb{R}^{n\times n}$$

Donde Q_1 es normalizada (no es ortogonal porque no es cuadrada): $Q_1^tQ_1 = 1$:

$$Ax = b$$

$$Q_1R_1x = b \ \Rightarrow \ R_1x = Q_1^tb$$

Necesita que las columnas sean linealmente independientes (que la matriz A tenga rango completo por columnas).

Cómo hacer la factorización: **Método de Gram-Schmidt**

Notamos A y Q como matriz de columnas y R triangular:

$$\begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix} = \begin{pmatrix} q_1 & q_2 & \dots & q_n \end{pmatrix} \begin{pmatrix} r_{11} & \dots & r_{1n} \\ \vdots & \ddots & \vdots \\ 0 & \dots & r_{nn} \end{pmatrix}$$

$$q_1^tq_1 = 1 \quad a_1 = q_1r_{11}$$

Se trata de escoger r_{11} de forma que q_1 esté normalizado:

$$q_k^t q_k = 1$$

$$q_k^t q_i = 0, \qquad i = 1 : k-1$$

$$q_i^t q_k = q_i^t a_n - r_{in} q_i^t q$$

Por tanto:

$$r_{ik} = \frac{q_i^t a_k}{q_i^t q_i} \tag{2.20}$$

Nos queda la columna k:

$$\begin{aligned} a_k &= r_{1k} q_1 + \ldots + r_{kk} q_k \\ q_k &= \tfrac{a_k - r_{1n} q_1 - \ldots - r_{1,k-1} q_{k-1}}{r_{kk}} \end{aligned} \tag{2.21}$$

Este método calcula las columnas k-ésima de Q_1 y R_1 en el paso k.

Podemos expresarlo de otra forma más simple. Suponiendo que existen Q y R para las que $A = QR$, entonces la k-ésima columna de la matriz A se puede expresar como

$$a_k = \sum_{i=1}^{k} r_{ik} q_i$$

Siendo q_i la i-ésima columna de la matriz Q. Suponiendo también que la matriz A tiene rango completo por columnas, de esta ecuación se obtiene

$$q_k = \left(a_k - \sum_{i=1}^{k-1} r_{ik} q_i \right) / r_{kk}$$

Si queremos que la columna q_k tenga norma 1 se toma

$$r_{kk} = \left\| a_k - \sum_{i=1}^{k-1} r_{ik} q_k \right\|$$

Asimismo, los elementos de la matriz R se pueden ir calculando como

$$r_{ik} = q_i' a_k \qquad \text{para } i = 1, 2, \ldots, k-1$$

A continuación vemos el algoritmo de Gram-Schmidt implementado en Matlab.

Algoritmo 2.8: Método de Gram-Schmidt

```
% Metodo de Gram-Schmidt para
% hacer la factorizacion QR reducida
% de una matriz A
function [Q, R] = gramsch(A)
[m,n] = size(A);
if m<n,
   error('Las dimensiones de la matriz A son incorrectas');
```

```
end;
Q = A;
R = zeros(n);
for k=1:n
   for i=1:k-1
      R(i,k) = Q(:,i)'*A(:,k);
      Q(:,k) = Q(:,k) - Q(:,i)*R(i,k);
   end;
   R(k,k) = norm(Q(:,k));
   Q(:,k) = Q(:,k)/R(k,k);
end;
```

♦

Sin embargo, este método es numéricamente inestable porque los q_k pierden ortogonalidad, ya que el numerador en el cálculo de las columnas q_k se puede anular usando precisión fija en la aritmética de punto flotante.

Por ello, se usa el **Método de Gram-Schmidt modificado**, que sólo cambia la forma de calcular las columnas q_k, y que es numéricamente estable y equivalente al clásico.

$$\left.\begin{array}{lll} r_{kk} = \|a_k\|_2 & q_k = \frac{a_k}{r_{kk}} & \\ r_{ki} = q_k^t a_i & a_i = a_i - r_{ki} q_k & i = k+1:n \end{array}\right\} \quad k = 1:n \tag{2.22}$$

Esto modifica la matriz A, aunque se puede escoger almacenar Q_1 en una matriz diferente. Se calcula la *k*-ésima fila de R_1 y la *k*-ésima columna de Q_1 en el paso *k*-ésimo.

En realidad lo único que hay que cambiar en el algoritmo es la línea

```
R(i,k) = Q(:,i)'*A(:,k);
```

Por la línea

```
R(i,k) = Q(:,i)'*Q(:,k);
```

Con lo que queda el algoritmo siguiente:

Algoritmo 2.9: Método de Gram-Schmidt modificado

```
% Metodo de Gram-Schmidt modificado
% para hacer la factorizacion
% QR reducida de una matriz A
function [Q, R] = gramschmod(A)
[m,n] = size(A);
if m<n,
   error('Las dimensiones de la matriz A son incorrectas');
end;
Q = A;
R = zeros(n);
for k=1:n
   for i=1:k-1
      R(i,k) = Q(:,i)'*Q(:,k);
      Q(:,k) = Q(:,k) - Q(:,i)*R(i,k);
```

```
        end;
        R(k,k) = norm(Q(:,k));
        Q(:,k) = Q(:,k)/R(k,k);
    end;
```

♦

Los dos métodos de Gram-Schmidt requieren del orden de $2mn^2$ operaciones.

2.3.2 Pseudo-inversas

Dada una matriz rectangular $A \in \mathbb{R}^{m\times n}$, la **pseudo-inversa de Moore-Penrose,** $A^+ \in \mathbb{R}^{n\times m}$, se define conforme a los siguientes axiomas:

1. $A^+AA^+ = A^+$
2. $AA^+A = A$
3. $(AA^+)^t = AA^+$
4. $(A^+A)^t = A^+A$
5. Al trabajar con matrices complejas:

$$\left(AA^+\right)^* = AA^+$$

Siendo $B^* = \overline{B}^t$ con $\overline{B}$ la conjugación de los elementos de la matriz.

Si A es rectangular y A^tA invertible,

$$\boxed{A^+ = \left(A^tA\right)^{-1}A^t} \tag{2.23}$$

✏ En general, no se cumple que la pseudoinversa del producto de dos matrices sea el producto de las pseudoinversas de ambas matrices

$$(AB)^+ \neq B^+A^+$$

✏ Por otro lado, se cumple que la traspuesta de la pseudoinversa es igual a la pseudoinversa de la traspuesta

$$\left(A^+\right)^t = \left(A^t\right)^+$$

Teorema 2.6. Cualquier vector x es solución mínimos cuadrados de un sistema $Ax = b$, y además $x = A^+b$ tiene mínima norma, si y sólo si se cumple:

$$Ax = AA^+b$$

Demostración.

$$\begin{aligned}\|Ax-b\|_2^2 &= \left\|Ax - AA^+b + AA^+b - b\right\|_2^2 \\ &= \left\|Ax - AA^+b\right\|_2^2 + \left\|AA^+b - b\right\|_2^2 + 2\left(Ax - AA^+b\right)^t\left(AA^+b - b\right)\end{aligned}$$

Siendo:

$$2x^tA^t\left(AA^+ - I\right)b = 2x^tA^t\left(A^tAA^t - A^t\right)b = 0$$

y:

$$-2b^t\left(AA^+\right)^t\left(AA^+ - I\right)b = -2b^t\left(AA^+AA^+ - AA^+\right)$$

como $A^+AA^+ = A^+$, lo de arriba vale cero, y queda:

$$\|Ax - b\|_2^2 = \left\|Ax - AA^+b\right\|_2^2 + \left\|AA^+b - b\right\|_2^2$$

2.3.2.1 Cómo calcular la pseudo-inversa

Aunque se puede calcular usando la ecuación que se vio antes, dicha ecuación sólo es válida si $\left|A^tA\right| \neq 0$, es decir, si A tiene rango completo por columnas. Por ello, computacionalmente se suele realizar el cálculo de la pseudoinversa haciendo la **descomposición en valores singulares** de la matriz A (primitiva `svd` de Matlab).

Toda matriz rectangular se puede descomponer en el producto de una ortogonal, una diagonal rectangular y otra ortogonal:

$$A = U\begin{pmatrix} \sigma_1 & 0 & \dots & \dots & 0 \\ 0 & \ddots & & & 0 \\ \vdots & & \sigma_r & & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & \dots & \dots & 0 \end{pmatrix}V^t \tag{2.24}$$

Donde $\sigma_i \geq 0$ se llaman **valores singulares** de la matriz y σ_i^2 son los autovalores de AA^t o A^tA. Para $rango\,(A) = r$ sólo hay r valores singulares no nulos.

$$A = USV^t$$

Dado que V es ortogonal, $V^tV = I$, así que postmultiplicando toda la ecuación por V:

$$AV = USV^tV \Rightarrow AV = US$$

Si notamos v_i y u_i a las columnas de V y U, respectivamente,

$$Av_i = \sigma_iu_i$$

O lo que es lo mismo:

$$AV = A\begin{pmatrix} v_1 & v_2 & \dots & v_n \end{pmatrix} = \begin{pmatrix} Av_1 & Av_2 & \dots & Av_n \end{pmatrix}$$
$$US = \begin{pmatrix} u_1 & u_2 & \dots & u_m \end{pmatrix} diag\,(\sigma_i) = \begin{pmatrix} \sigma_1u_1 & \sigma_2u_2 & \dots & \end{pmatrix}$$

La traspuesta de A es

$$A^t = VSU^t$$

Si postmultiplicamos por U, dado que U es ortogonal

$$A^tU = VSU^tU = VS \Rightarrow A^tu_i = \sigma_i v_i$$

Así que si multiplicamos $Av_i = \sigma_i^2 v_i$ por la traspuesta de A,

$$A^tAv_i = \sigma_i A^t u_i$$

Tenemos la definición de autovalor ($Bw = \lambda w$):

$$A^tAv_i = \sigma_i^2 v_i$$

Es decir, que σ_i^2 es autovalor de A^tA con autovector v_i.

$$AA^tu_i = \sigma_i^2 u_i$$

Se calculará la pseudoinversa, una vez obtenidos V, S y U como

$$\boxed{A^+ = VS^+U^t} \tag{2.25}$$

Donde

$$S^+ = \begin{pmatrix} \frac{1}{\sigma_1} & \dots & \dots & \dots & 0 \\ \vdots & \ddots & & & \vdots \\ 0 & & \frac{1}{\sigma_r} & & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & \dots & \dots & \dots & 0 \end{pmatrix} \tag{2.26}$$

Un ejemplo de aplicación de este método es la compresión de imágenes:

$$A = USV^t = \sum_{i=1}^{n} \sigma_i u_i v_i^t$$

Esto sirve para reconstruir la matriz A, y si tenemos una matriz original A muy grande (digamos 1000×1000), con 20 valores singulares, se queda en una mucho más pequeña (60×1000).

La solución de mínimos cuadrados usando la pseudoinversa es

$$\boxed{x_{LS} = A^+b = \sum_{i=1}^{r} \frac{1}{\sigma_i} v_i \left(u_i^t b\right)} \tag{2.27}$$

2.4 Estimaciones de error

Ahora vamos a ver lo próximos que están dos vectores, en lugar de números.

Definición: Se definen las **normas** de un vector $x \in \mathbb{R}^n$ de la siguiente forma:

$$\begin{aligned}\|x\|_1 &= \sum_{j=1}^{n} |x_j| \\ \|x\|_2 &= \sqrt{\sum_{j=1}^{n} |x_j|^2} \\ \|x\|_p &= \left(\sum_{j=1}^{n} |x_j|^p \right)^{\frac{1}{p}} \\ \|x\|_\infty &= \max_j |x_j|\end{aligned} \tag{2.28}$$

Definición: Para una matriz $A \in \mathbb{R}^{n \times n}$, se define la **norma matricial** como una aplicación $\|\cdot\| : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}$ tal que:

1. $\|A\| \geq 0$, $\|A\| = 0 \Leftrightarrow A = 0$
2. $\|\alpha A\| = |\alpha| \, \|A\|$
3. $\|A + B\| \leq \|A\| + \|B\|$
4. $\|AB\| \leq \|A\| \, \|B\|$

Definición: **norma matricial natural o inducida**:

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\| \tag{2.29}$$

Norma 1:

$$\|A\|_1 = \sup_{x \neq 0} \frac{\|Ax\|_1}{\|x\|_1} = \max_{i=1:n} \sum_{j=1}^{n} |a_{ji}| \tag{2.30}$$

Norma 2:

$$\|A\|_2 = \sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sqrt{\rho\left(A^t A\right)} = \max_i \sigma_i \tag{2.31}$$

Norma infinita:

$$\|A\|_\infty = \sup_{x \neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \max_{i=1:n} \sum_{j=1}^{n} |a_{ij}| \tag{2.32}$$

El ínfimo de todas las normas matriciales que pueden definirse de *A* es el radio espectral de la matriz:

$$\rho\left(A\right) = \inf_{\|\cdot\|} \|A\|$$

Definición: Si tenemos una matriz H invertible, se define la **norma transformada por una matriz** como:

$$\|A\|_H = \left\|H^{-1}AH\right\|$$

En el caso vectorial:

$$\|x\|_H = \left\|H^{-1}x\right\|$$

Definición: Se dice que una norma matricial es **consistente con una vectorial** si cumple:

$$\|Ax\| \leq \|A\| \|x\| \qquad \forall A, x$$

Definición: Una matriz consistente se dice **subordinada** si hay un sólo vector x que cumpla la igualdad, es decir:

$$\exists! x_0 = \overline{0} / \|Ax_0\| = \|A\| \|x_0\|$$

Si $\|\cdot\|$ es norma subordinada a una vectorial, se cumple que $\|I\| = 1$.

Un ejemplo de norma consistente no subordinada es la **norma de Frobenius**:

$$\|A\|_F = \sqrt{\sum_{j=1}^{n}\sum_{i=1}^{n}|a_{ij}|^2} = \sqrt{traza\left(AA^H\right)}$$

Ya que $\|I\|_F = \sqrt{n}$.

Definición: Dada una sucesión de vectores $\left\{x^{(s)}\right\}$ se define su límite de la siguiente forma:

$$\lim_{s\to\infty} x^{(s)} = x^* \text{ si } \lim_{s\to\infty} \left\|x^* - x^{(s)}\right\| = 0$$

En el caso matricial (dada la sucesión $\{A_s\}$):

$$\lim_{s\to\infty} A_s = A_* \text{ si } \lim_{s\to\infty} \|A_* - A_s\| = 0$$

Todo esto sin distinción de normas, ya que todas son equivalentes.

Teorema 2.7. (para una matriz convergente):

$$\lim_{s\to\infty} B^s = 0 \Leftrightarrow \rho(B) < 1$$

Teorema 2.8. Serie matricial geométrica:

$$I + B + \ldots + B^s + \ldots = \sum_{s=0}^{\infty} B^s$$

converge a$(I-B)^{-1}$ si y sólo si $\rho(B) < 1$.

2.4.1 Condicionamiento de una matriz invertible

Dado el sistema

$$Ax = b$$

Siendo A invertible, al calcular su solución computacionalmente, habrá que almacenar los datos, lo que originará un error de redondeo:

$$(A + e_a(A))(x + e_a(x)) = b + e_a(b) \longrightarrow \hat{A}\hat{x} = \hat{b}$$

Si ahora queremos determinar una cota del error relativo en la solución,

$$\|e_r(x)\| = \frac{\|\hat{x} - x\|}{\|x\|} = \frac{\|e_a(x)\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \varepsilon\kappa(A)} \left(\frac{\|e_a(A)\|}{\|A\|} + \frac{\|e_a(b)\|}{\|b\|} \right)$$

Siendo

$$\|e_a(A)\| \leq \varepsilon \|A\|, \quad \|e_a(b)\| \leq \varepsilon \|b\|, \quad \varepsilon \|A\| \left\|A^{-1}\right\| < 1$$

$$\kappa(A) = \|A\| \cdot \left\|A^{-1}\right\|$$

En caso de que A no sea invertible, $\kappa(A) = \infty$.

✏ A $\kappa(A)$ se le llama **número de condición**, y cuantifica la sensibilidad del problema. Dado el error en los datos, es un número que, multiplicando por él, da la cota de error en el resultado.

$$\boxed{\kappa(A) = \begin{cases} \|A\| \left\|A^{-1}\right\| & \text{si } A \text{ es invertible} \\ \infty & \text{si } A \text{ no es invertible} \end{cases}} \tag{2.33}$$

Se dice que la matriz es **perfectamente condicionada** si $\kappa(A) = 1$, **mal condicionada** si $\kappa(A)$ es un número grande y **bien condicionada** si es un número próximo a 1.

Ejemplo 2.8:

Almacenamos la siguiente matriz:

$$A = hilb(3) = \begin{pmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{pmatrix}$$

En un sistema con precisión fija que nos permite almacenar 4 decimales. La matriz del error absoluto es

$$e_a(A) = \begin{pmatrix} 0 & 0 & -t \\ 0 & -t & 0 \\ -t & 0 & 0 \end{pmatrix}, \qquad t = 0.0000333\ldots$$

La matriz almacenada es

$$\hat{A} = \begin{pmatrix} 1.0000 & 0.5000 & 0.3333 \\ 0.5000 & 0.3333 & 0.2500 \\ 0.3333 & 0.2500 & 0.2000 \end{pmatrix}$$

El número de condición de $\hat{A}$ es

$$\kappa(A) = 528.54$$

■

Normalmente se tiene

$$\varepsilon = \max\left\{\left\|e_r\left(A\right)\right\|, \left\|e_r\left(b\right)\right\|\right\} < \left(\kappa\left(A\right)\right)^{-1}$$

Para poder aplicar la definición del error relativo en la solución antes vista, se debe cumplir que si

$$\varepsilon = \left\|e_r\left(A\right)\right\| = \frac{\left\|e_a\left(A\right)\right\|}{\|A\|}$$

entonces sea

$$\left\|e_a\left(A\right)\right\| < \frac{1}{\left\|A^{-1}\right\|}$$

2.4.1.1 Caso particular: $e_a\left(A\right) = 0$

Si sólo tenemos errores en el vector b, entonces

$$A\hat{x} = \hat{b} = b + e_a\left(b\right) \rightarrow A\left(x + e_a\left(x\right)\right) = b + e_a\left(b\right)$$

Como $Ax = b$, tenemos el sistema

$$Ae_a\left(x\right) = e_a\left(b\right) \Rightarrow e_a\left(x\right) = A^{-1}e_a\left(b\right)$$

Tomando normas

$$\left\|e_a\left(x\right)\right\| = \left\|A^{-1}e_a\left(b\right)\right\| \leq \left\|A^{-1}\right\| \left\|e_a\left(b\right)\right\|$$

Se cumple además

$$\|b\| = \|Ax\| \leq \|A\| \|x\| \Rightarrow \frac{1}{\|x\|} \leq \frac{\|A\|}{\|b\|}$$

Así que podemos escribir el error relativo en la solución como

$$\left\|e_r\left(x\right)\right\| = \frac{\left\|e_a\left(x\right)\right\|}{\|x\|} \leq \left\|A^{-1}\right\| \|A\| \frac{\left\|e_a\left(b\right)\right\|}{\|b\|} = \kappa\left(A\right) \left\|e_r\left(b\right)\right\|$$

Ejemplo 2.9:

Dada una matriz A diagonal $n \times n$ con $a_{ii} = 10^{-2}$, el número de condición en norma ∞ es

$$\kappa_\infty\left(A\right) = \|A\|_\infty \left\|A^{-1}\right\|_\infty = 10^{-2}10^2 = 1$$

Es una matriz perfectamente condicionada a pesar de que $det\left(A\right) = 10^{-2n} \simeq 0$ (con n un relativamente grande).

■

Ejemplo 2.10:

Sea una matriz A ortogonal ($A^t = A^{-1}$, $\|A\|_2 = 1$). Su número de condición en norma 2 es

$$\kappa_2\left(A\right) = \|A\|_2 \left\|A^{-1}\right\|_2 = \|A\|_2 \|A\|_2 = 1$$

Por eso cuando no se quiere empeorar el condicionamiento de un problema se multiplica por matrices ortogonales.

■

Ejemplo 2.11:

Dados $\sigma_1 > \sigma_2 > \ldots > \sigma_n$ los valores singulares de la matriz A, la norma 2 de dicha matriz, rectangular $m \times n$, es

$$\|A\|_2 = \sqrt{\rho\left(A^t A\right)} = \sigma_1$$

Mientras que la norma de la pseudoinversa es

$$\left\|A^+\right\|_2 = \frac{1}{\sigma_n}$$

Por tanto, el número de condición de una matriz rectangular es

$$\kappa_2\left(A\right) = \frac{\sigma_1}{\sigma_n}$$

■

2.4.2 Condicionamiento de un problema de mínimos cuadrados

Para el problema de mínimos cuadrados se usa el número de condición en norma 2, dado por:

$$\kappa_2\left(A\right) = \frac{\sigma_1}{\sigma_r},\ A \in \mathbb{R}^{m\times n};\ m \geq n;\ rango\left(A\right) = n$$

Teorema 2.9. Dados x_{LS} y $\hat{x}$ soluciones respectivas de mínimos cuadrados de los sistemas:

$$\text{mín}\left\|Ax - b\right\|_2 \quad \text{mín}\left\|\left(A + \Delta A\right)x - \left(b + \Delta b\right)\right\|_2$$

con $A, \Delta A \in \mathbb{R}^{m\times n}$, $m \geq n$, $rango\left(A\right) = n$ y $b, \Delta b \in \mathbb{R}^m$. Si se cumple:

$$\varepsilon = \text{máx}\left\{\frac{\|\Delta A\|_2}{\|A\|_2}, \frac{\|\Delta b\|_2}{\|b\|_2}\right\} < \frac{\sigma_n(A)}{\sigma_1(A)} \quad \frac{\rho}{\|b\|_2} \neq 1$$

siendo $\rho = \left\|Ax_{LS} - b\right\|_2$, entonces

$$\frac{\|\hat{x} - x_{LS}\|_2}{\|x_{LS}\|_2} \leq \varepsilon\left(2\kappa_2\left(A\right)\frac{\|b\|_2}{\sqrt{\|b\|_2^2 - \rho^2}} + \kappa_2\left(A\right)^2\frac{\rho}{\sqrt{\|b\|_2^2 - \rho^2}}\right) + O\left(\varepsilon^2\right)$$

En este caso, al depender del cuadrado del número de condición, es más inestable para matrices que no sean perfectamente condicionadas.

2.4.3 El residual y el refinamiento iterativo

Se define el **residual** de x como:

$$r = b - Ax$$

Si tenemos una solución calculada $\hat{x}$, $A\hat{x} = b - r$, mientras que para una solución exacta $\overline{x}$, $A\overline{x} = b$.

$$A\left(\hat{x} - \overline{x}\right) = -r$$

Error absoluto de la solución calculada:

$$\|\hat{x} - \overline{x}\| \leq \left\|A^{-1}\right\| \|r\|$$

Error relativo:

$$\frac{\|\hat{x} - \overline{x}\|}{\|x\|} \leq \frac{\left\|A^{-1}\right\| \|r\|}{\|x\|}$$

El residuo no nos dice la diferencia entre la solución exacta y la calculada.

$$\|A\| \|x\| \geq \|b\| \Rightarrow \frac{\|\hat{x} - x\|}{\|x\|} \leq \kappa(A) \frac{\|r\|}{\|b\|}$$

Si resolvemos el sistema $A(\hat{x} - x) = -r$ (o $A(x - \hat{x}) = r$):

$$Ay = r$$

Una vez que tengamos calculada la y, $x - \hat{x} = y$; $x = \hat{x} + y$.

Esto es genial para obtener la solución exacta ¿no? Pues no, porque al resolver $Ay = r$ volvemos a cometer error, y sólo obtenemos una solución un poco más aproximada a la exacta. Esta es la base del **refinamiento iterativo**. Se suele hacer 2 o 3 veces.

2.5 Métodos iterativos para sistemas lineales

2.5.1 Definición

Dado un sistema

$$Ax = b$$

con A de gran tamaño y dispersa, un método apropiado para resolverlo es alguno iterativo, que básicamente consiste en construir una sucesión $x^{(s)}$ de aproximaciones a la solución o soluciones del sistema.

Un método iterativo es **consistente** con un problema si existe el límite $\lim_{s\to\infty} x^{(s)} = x^*$, donde x^* es la solución del problema.

2.5.2 Métodos de punto fijo

2.5.2.1 Definición, consistencia y convergencia

$$x^{(s+1)} = Bx^{(s)} + c \qquad B \in \mathbb{R}^{n\times n}, \quad c \in \mathbb{R}^n, \quad s = 0, 1, \ldots \tag{2.34}$$

Teorema de consistencia del método de punto fijo. El método de punto fijo (2.34) es consistente con un sistema de ecuaciones $Ax = b$ para una matriz A no singular si y sólo si $I - B$ es no singular y $c = (I - B) A^{-1}b$. Nos dice lo bien que modelan B y c al sistema $Ax = B$. La consistencia implica por otra parte

$$solucion\,(A, b) = solucion\,(I - B, c) \Rightarrow A^{-1}b = x = (I - B)^{-1} c$$

Lectura del teorema:

Si existe el límite $\lim x^{(s)} = x^*$ entonces $\lim x^{(s+1)} = \lim Bx^{(s)} + c$, y por tanto $x^* = Bx^* + c$.

Estamos resolviendo el sistema relativo al método. Como $(I - B) x^* = c$, dado que $I - B$ es invertible, existe una única solución y es la misma que la de $Ax = b$.

♠

La consistencia no asegura la convergencia.

Teorema de convergencia del método de punto fijo. El método de punto fijo (2.34), siendo consistente con un sistema, converge a $(I - B)^{-1} c$, con cualquier vector inicial $x^{(0)}$ si y sólo si $\rho(B) < 1$. Es condición necesaria y suficiente.

Dado que es para cualquier vector inicial, se trata de convergencia global.

Expresado de otra forma, como condición suficiente:

El método de punto fijo (2.34) converge a $x^* = (I - B)^{-1} c$ con cualquier vector inicial $x^{(0)}$ si $\|B\| < 1$ para alguna norma matricial subordinada. Si no lo cumple para alguna norma no se puede asegurar nada, ya que $\rho(B) \leq \|B\|$, y puede haber alguna otra norma que sí lo cumpla.

Recordemos que para números la serie

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1 - r}$$

converge al resultado si $|r| < 1$.

En el caso de matrices,

$$\sum_{i=0}^{\infty} B^i = (I - B)^{-1}$$

converge si $\rho(B) < 1$.

2.5.2.2 Cotas de error

Para un método de punto fijo con matriz B se satisfacen las siguientes cotas de error:

$$\left\|x^* - x^{(s)}\right\| \leq \frac{\|B\|^{s-k} \left\|x^{(k+1)} - x^{(k)}\right\|}{1 - \|B\|} \qquad \text{para } k = 0, 1, \ldots, s - 1 \tag{2.35}$$

$$\left\|x^* - x^{(s)}\right\| \leq \|B\|^s \left\|x^* - x^{(0)}\right\| \tag{2.36}$$

Hay otras cotas que se pueden usar, por ejemplo una para determinar el número de iteraciones para conseguir una cota dada, haciendo $k = 0$:

$$\left\|x^* - x^{(s)}\right\| \leq \frac{\|B\|^s}{1 - \|B\|} \left\|x^{(1)} - x^{(0)}\right\|$$

O también una **cota dinámica**, que nos puede servir para programar un método y acotar el número de iteraciones también por la cota de error (tolerancia):

$$\boxed{\left\|x^* - x^{(s)}\right\| \leq \frac{\|B\|}{1-\|B\|}\left\|x^{(s)} - x^{(s-1)}\right\|} \tag{2.37}$$

Además, en caso de que $\|B\| < 0.5$,

$$\frac{\|B\|}{1-\|B\|} < 1$$

y podemos usar sólo el término de

$$\left\|x^{(s)} - x^{(s-1)}\right\|$$

para programarlo y llamar a eso tolerancia.

Nota: es pregunta habitual de examen pedir el número de iteraciones mínimo para que el error cometido sea menor que un determinado valor ε. Para ello, cogemos la segunda parte de la desigualdad:

$$\frac{\|B\|^{s-k}\left\|x^{(k+1)} - x^{(k)}\right\|}{1-\|B\|} \leq \varepsilon$$

Se toman entonces logaritmos y se despeja s. Esta cota se llama **cota de error a priori**. Existe una **cota de error a posteriori**, que es da una mejor cota pero hay que calcular la iteración en la que se acota:

$$\left\|x^* - x^{(s)}\right\| \leq \frac{\|B\|^{s-k}\left\|x^{(s)} - x^{(s-1)}\right\|}{1-\|B\|}$$

$$\left\|x^* - x^{(s)}\right\|_\infty \leq \frac{\|B\|_\infty^{s}\left\|x^{(1)} - x^{(0)}\right\|}{1-\|B\|_\infty}$$

2.5.3 Comparación de métodos

Definición: **factor medio de reducción por iteración**:

$$\left(\frac{\left\|x^* - x^{(s)}\right\|}{\left\|x^* - x^{(0)}\right\|}\right)^{1/s} \leq \|B\| \tag{2.38}$$

Definición: **razón media por iteración**:

$$(\|B^s\|)^{1/s} \tag{2.39}$$

Se dice que un método con matriz B_1 es más rápido que otro con matriz B_2 si el radio espectral de B_1 es menor que el de B_2.

2.5.4 Métodos de descomposición o partición regular

Estos métodos se basan en dos afirmaciones:

- ✏ Si un método es de partición regular, entonces es consistente
- ✏ Si un método es convergente, entonces es de partición regular, aunque no necesariamente al revés

Si tenemos el sistema $Ax = b$ con A regular, y $A = M - N$ con M invertible, tenemos que $Mx = Nx + b$ es la ecuación relativa al método de partición regular, dado un vector inicial $x^{(0)}$. Es decir:

$$Mx^{(s+1)} = Nx^{(s)} + b \Rightarrow x^{(s+1)} = M^{-1}Nx^{(s)} + M^{-1}b$$

Los métodos clásicos de este tipo están definidos para matrices A con elementos de la diagonal principal no nulos y tienen la siguente partición:

$$A = D + L + R$$

Con D matriz diagonal, L triangular inferior y R triangular superior.

En caso de que haya elementos de la diagonal principal nulos, se reorganizan las filas y/o columnas de A.

Siendo:

$$D = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & \dots & a_{nn} \end{pmatrix} \quad L = \begin{pmatrix} 0 & 0 & \dots & 0 \\ a_{21} & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix} \quad R = \begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ 0 & 0 & \dots & a_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \tag{2.40}$$

Según la forma de la matriz B que se use, hay tres métodos clásicos: Jacobi, Gauss-Seidel y SOR.

2.5.4.1 Método de Jacobi

La descomposición es

$$M = D \qquad N = -(L + R) \tag{2.41}$$

Por lo que la matriz del método es

$$\boxed{B_J = -D^{-1}(L + R)} \tag{2.42}$$

El cálculo del vector en cada iteración es

$$\boxed{x^{(s+1)} = -D^{-1}(R + L)\,x^{(s)} + D^{-1}b} \tag{2.43}$$

Invierte la matriz diagonal, D.

El cálculo de cada componente del vector $x^{(s)}$ se realiza del siguiente modo:

$$\boxed{x_i^{(s+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(s)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(s)}}{a_{ii}} \qquad i = 1:n} \tag{2.44}$$

Es decir, que se opera con todas las componentes del vector anterior excepto con la que estamos calculando.

Para calcular $\rho(B_J)$,

$$det(B_J - \lambda I) = 0 \Rightarrow det\left(-D^{-1}(L + R) - \lambda I\right) = 0$$

Multiplicando por $det\,(-D)$ nos queda

$$det\,(-D)\,det\left(-D^{-1}\,(L+R)-\lambda I\right)=det\,(\lambda D+L+R)=0$$

Por lo que hay que multiplicar por λ sólo las componentes de la diagonal principal de A para calcular los autovalores a partir del polinomio característico.

A continuación podemos ver el código en Matlab para el método de Jacobi.

Algoritmo 2.10: Método iterativo de Jacobi para sistemas lineales

```
% Función que resuelve un sistema lineal por el metodo
% iterativo de Jacobi.
function [X,paso]=jacobi(A,b,x0,iter)
if nargin<4
   'Faltan argumentos de entrada'
   return;
else
   [cols filas]=size(A);
   if cols~=filas, 'La matriz no es cuadrada', return;
   else
      if (cols~=size(x0))|(cols~=size(b))
         error('El sistema no es valido');
      else
% Generamos las matrices D, L y R
         for k=1:cols
            for j=k+1:cols
               if k~=j
                  D(j,k)=0;
                  D(k,j)=0;
                  L(j,k)=A(j,k);
                  L(k,j)=0;
                  R(k,j)=A(k,j);
                  R(j,k)=0;
               end;
            end;
            D(k,k)=A(k,k);
            L(k,k)=0;
            R(k,k)=0;
         end;
         fin=0;
         paso=1;
         x(1,:)=x0';
% La condicion de parada adicional
% es que se llegue a la solución exacta
         while (fin==0)&(paso<=iter)
            for componente=1:cols
               vectant=0;
               vectsig=0;
               for k=1:componente-1
                  vectant=vectant+A(componente,k)*x(paso,k);
               end;
```

```
                for k=componente+1:cols
                   vectsig=vectsig+A(componente,k)*x(paso,k);
                end;
                x(paso+1,componente) = (b(componente)-vectant -
                  vectsig)/A(componente,componente);
             end;
% Si la solucion es igual a la anterior, salimos del bucle
             if x(paso,:)==x(paso+1,:)
                fin=1;
             end;
             paso=paso+1;
          end;
% Metemos la última solución en un vector
          X=x(paso,:);
       end;
    end;
end;
```

♦

2.5.4.2 Método de Gauss-Seidel

Las componentes ya calculadas se usan para calcular las siguientes en la misma iteración. Acelera la convergencia, sin embargo no se puede asegurar que en general sea mejor que Jacobi.

En este caso la descomposición es

$$M = D + L \qquad N = -R \tag{2.45}$$

Y la matriz del método

$$\boxed{B_{GS} = -\left(D + L\right)^{-1} R} \tag{2.46}$$

El cálculo del vector en cada iteración es

$$\boxed{x^{(s+1)} = -D^{-1}Lx^{(s+1)} - D^{-1}Rx^{(s)} + D^{-1}b} \tag{2.47}$$

Para el cálculo de cada componente se usa la fórmula:

$$\boxed{x_i^{(s+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(s+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(s)}}{a_{ii}} \qquad i = 1:n} \tag{2.48}$$

Esta vez, para calcular $\rho\left(B_{GS}\right)$ hay que hacer

$$det\left(\lambda D + \lambda L + R\right) = 0$$

Es decir, multiplicar las componentes de las matrices D y L por λ para obtener el polinomio característico y extraer los autovalores.

Ahora vemos el código en Matlab para implementar el método de Gauss-Seidel.

Algoritmo 2.11: Método iterativo de Gauss-Seidel para sistemas lineales

```
% Funcion que resuelve un sistema lineal por el método
% iterativo de Gauss-Seidel.
function [X,paso]=seidel(A,b,x0,iter)
if nargin<4
   error('Faltan argumentos de entrada');
else
   [cols filas]=size(A);
   if cols~=filas
      error('La matriz no es cuadrada');
   else
      if (cols~=size(x0))|(cols~=size(b))
         error('El sistema no es valido');
      else
% Generamos las matrices D, L y R
         for k=1:cols
            for j=k+1:cols
               if k~=j
                  D(j,k)=0;
                  D(k,j)=0;
                  L(j,k)=A(j,k);
                  L(k,j)=0;
                  R(k,j)=A(k,j);
                  R(j,k)=0;
               end;
            end;
            D(k,k)=A(k,k);
            L(k,k)=0;
            R(k,k)=0;
         end;
         fin=0;
         paso=1;
         x(1,:)=x0';
% La condicion de parada adicional
% es que se llegue a la solucion exacta
         while (fin==0)&(paso<iter)
            for componente=1:cols
               vectant=0;
               vectsig=0;
               for k=1:componente-1
                  vectant=vectant +
                    A(componente,k)*x(paso+1,k);
               end;
               for k=componente+1:cols
                  vectsig=vectsig+A(componente,k)*x(paso,k);
               end;
               x(paso+1,componente)=(b(componente)-vectant-
                 vectsig)/A(componente,componente);
            end;
% Si la solucion es igual a la anterior, salimos del bucle
            if x(paso,:)==x(paso+1,:)
```

```
                fin=1;
            end;
            paso=paso+1;
        end;
% Metemos la ultima solucion en un vector
        X=x(paso,:);
    end;
  end;
end;
```

♦

Ejemplo 2.12:

Dada la matriz A

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

y el vector

$$b = \begin{pmatrix} 2 \\ -1 \\ 3 \\ -1 \end{pmatrix}$$

Vamos a determinar si los métodos de Jacobi y Gauss-Seidel convergen a la solución.

Para el método de Jacobi, el polinomio característico es

$$p(\lambda) = \begin{vmatrix} 2\lambda & 1 & 0 & -1 \\ 2 & 2\lambda & 3 & 1 \\ 1 & 0 & 2\lambda & 5/3 \\ 2 & 0 & 0 & 4\lambda \end{vmatrix} = 32\lambda^4 - 8\lambda^2 + 16\lambda - 10 = 0$$

Si nos fijamos un poco, $p(-\infty) = \infty$ y $p(-1) = -2$, por lo que hay una raiz entre $-\infty$ y -1, es decir, de módulo mayor de la unidad, por tanto

$$\rho(B_J) > 1$$

y el método iterativo de Jacobi no converge.

Por otro lado, para Gauss-Seidel el polinomio característico es

$$p(\lambda) = \begin{vmatrix} 2\lambda & 1 & 0 & -1 \\ 2\lambda & 2\lambda & 3 & 1 \\ \lambda & 0 & 2\lambda & 5/3 \\ 2\lambda & 0 & 0 & 4\lambda \end{vmatrix} = \lambda \begin{vmatrix} 2 & 1 & 0 & -1 \\ 2-4\lambda & 0 & 3 & 1+2\lambda \\ 1 & 0 & 2\lambda & 5/3 \\ 2 & 0 & 0 & 4\lambda \end{vmatrix} = \lambda\left(32\lambda^3 - 8\lambda^2 + 16\lambda - 10\right) = 0$$

Se tiene $p(0) = 0$, $p(0.5) = 0$ y los otros dos autovalores son

$$\lambda_{3,4} = \frac{-1 \pm \sqrt{39}i}{8}$$

así que

$$\rho(B_{GS}) = \sqrt{\frac{5}{8}} < 1$$

por tanto, el método de Gauss-Seidel converge.

■

2.5.4.3 Método SOR (Sobrerrelajación)

La descomposición en este caso es

$$M = \frac{1}{\omega}D + L \qquad N = -\left(\frac{\omega - 1}{\omega}D + R\right) \tag{2.49}$$

Y por tanto la matriz del método

$$\boxed{B_{SOR} = (D + \omega L)^{-1}\left((1-\omega)\,D - \omega R\right)} \tag{2.50}$$

El cálculo del vector en cada iteración es

$$\boxed{x^{(s+1)} = (D + \omega L)^{-1}\left((1-\omega)\,D - \omega R\right)x^{(s)} + \omega\,(D + \omega L)^{-1}\,b} \tag{2.51}$$

Donde ω es el factor de relajación. Si $0 < \omega < 1$ el método se llama de **subrelajación**, mientras que si $\omega > 1$ es de **sobrerrelajación**, los cuales se pueden usar para acelerar la convergencia de sistemas que son convergentes por el método de Gauss-Seidel.

La fórmula para cada componente del vector en la iteración $s+1$ es la siguiente:

$$\boxed{x_i^{(s+1)} = (1-\omega)\,x_i^{(s)} + \frac{\omega\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(s+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(s)}\right)}{a_{ii}} \qquad i = 1:n} \tag{2.52}$$

Como complemento a los métodos, veamos los de **Richardson** y el **SOR simétrico**:

2.5.4.4 Método de Richardson

$$x^{(s+1)} = x^{(s)} + \gamma\left(b - Ax^{(s)}\right) \tag{2.53}$$

Con γ arbitrario no nulo.

2.5.4.5 Método SOR simétrico (SSOR)

$$\begin{aligned} x^{\left(s+\frac{1}{2}\right)} &= (1-\omega)\,x^{(s)} + \omega\left(-D^{-1}Lx^{(s+1/2)} - D^{-1}Rx^{(s)} + D^{-1}b\right) \\ x^{(s+1)} &= (1-\omega)\,x^{(s+1/2)} + \omega\left(-D^{-1}Lx^{(s+1/2)} - D^{-1}Rx^{(s+1)} + D^{-1}b\right) \end{aligned} \tag{2.54}$$

En el SOR se hace el cálculo de 1 a n, en el SSOR de n a 1.

2.5.4.6 Teoremas de convergencia

Definición: **Matriz de diagonal dominante***:*

Se dice que una matriz A es de diagonal dominante estrictamente (por filas) si cumple:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \qquad \forall i \tag{2.55}$$

Es decir, que el elemento de la diagonal principal en valor absoluto es mayor que la suma de los valores absolutos de los elementos restantes de la fila.

Teorema 2.10. Dado un sistema $Ax = b$ siendo A de diagonal dominante estrictamente, los métodos de Jacobi y Gauss-Seidel son convergentes a la solución del sistema.

Demostración. *Para el método de Jacobi.*

$$B_J = [b_{ij}]$$

Con $b_{ij} = -\frac{a_{ij}}{a_{ii}}$ *y* $b_{ii} = 0$*. Entonces:*

$$\|B_J\|_\infty = \max_i \sum_{j=1}^{n} |b_{ij}| = \max_i \frac{\sum_{j\neq i} a_{ij}}{|a_{ii}|} < 1$$

Por tanto, Jacobi converge si es matriz de diagonal dominante

Teorema 2.11. Sea un sistema $Ax = b$ siendo A simétrica definida positiva y $A = M - N$. Si M es invertible, entonces el método de descomposición regular

$$x^{(s+1)} = M^{-1}Nx^{(s)} + M^{-1}b$$

converge a la solución si la matriz $M^t + N$ es definida positiva.

Generalización:

Dada A simétrica tal que $M^t + N$ es definida positiva, el método es convergente si y sólo si A es definida positiva.

Hay dos casos particulares:

- ✏ Método de Jacobi:
$$M^t + N = D - L - R \neq A$$
- ✏ Método de Gauss-Seidel:
$$M^t + N = D + L^t - R = D + L^t - L^t = D$$
dado que A es simétrica, $R = L^t$

Teorema de Kahan. Dada una matriz A de elementos $a_{ii} \neq 0$ para todo i, el método SOR para esa matriz converge sólo si $0 < \omega < 2$.

Teorema de Ostrowski-Reich *(importante).* Dado un sistema $Ax = b$ con A simétrica definida positiva, el método SOR converge a la solución del sistema para $0 < \omega < 2$. Es condición suficiente. Como SOR es lo mismo que Gauss-Seidel cuando $\omega = 1$, todo lo que verifiquemos para SOR se cumple para Gauss-Seidel.

Teorema de Stein-Rosenberg *(convergencia de Jacobi y Gauss-Seidel).* Si se tiene un sistema $Ax = b$ con A tal que $a_{ii} > 0$ para $i = 1 : n$ y $a_{ij} \leq 0$ para $i \neq j$, entonces se cumple una y sólo una de las siguientes posibilidades:

1. $0 < \rho(B_{GS}) < \rho(B_J) < 1$, con lo que convergen ambos métodos.
2. $0 = \rho(B_{GS}) = \rho(B_J)$, con lo que también convergen ambos.
3. $1 < \rho(B_J) < \rho(B_{GS})$, con lo que no converge ninguno.
4. $1 = \rho(B_{GS}) = \rho(B_J)$, tampoco converge ninguno.

Teorema 2.12. Dada una matriz A tridiagonal con $a_{ii} \neq 0$ para $i = 1 : n$, si los autovalores de B_J son reales, tanto Jacobi como SOR para $0 < \omega < 2$ convergen o divergen simultáneamente. Si ambos convergen, entonces:

$$\rho(B_{SOR}(\omega)) = \begin{cases} 1 - \omega + \frac{1}{2}\omega^2\mu^2 + \omega\mu\sqrt{1 - \omega + \frac{1}{4}\omega^2\mu^2} & 0 < \omega \leq \omega_0 \\ \omega - 1 & \omega_0 \leq \omega < 2 \end{cases} \tag{2.56}$$

Siendo $\mu = \rho(B_J)$ y ω_0 la elección óptima de ω, hallándose de la forma siguiente:

$$\omega_0 = \frac{2}{1 + \sqrt{1 - \mu^2}} \tag{2.57}$$

Si además de tridiagonal, A es simétrica y definida positiva, SOR y Jacobi convergen (en virtud del teorema de Ostrowsky-Reich).

Ejemplo 2.13:

Dada la matriz A y el vector b:

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix} \quad b = \begin{pmatrix} 2 \\ 4 \\ 2 \end{pmatrix}$$

Estudiar si el método iterativo de Jacobi converge a la solución del sistema $Ax = b$ para A y b.

En caso afirmativo, determinar el número de iteraciones a realizar para que el error en la iteración m-ésima sea menor que $5 \cdot 10^{-5}$, con $x^{(0)} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}^t$.

Solución:

La matriz A no es de diagonal dominante, así que buscamos $\rho(B_J)$.

Como $B_J = -D^{-1}(L + R)$, para hallar los autovalores de B_J:

$$\left|-D^{-1}(L + R) - \lambda I\right| = 0 \overset{Multiplicando\,por\,-D}{\Longrightarrow} |L + R + \lambda D| = 0$$

Como D tiene inversa, $|L + R + \lambda D| = 0$:

$$\begin{vmatrix} 2\lambda & -1 & 0 \\ -1 & 2\lambda & 1 \\ 0 & 1 & 2\lambda \end{vmatrix} = 8\lambda^3 - 4\lambda = 0 \Rightarrow \begin{cases} \lambda_1 = 0 \\ \lambda_2 = \sqrt{\frac{1}{2}} \\ \lambda_3 = -\sqrt{\frac{1}{2}} \end{cases}$$

$\rho(B_J) = +\sqrt{\frac{1}{2}} < 1 \Rightarrow$ Jacobi converge

Ahora hallamos el número de iteraciones:

$$\left\| x^{(m)} - x^* \right\|_2 \leq \frac{\|B_J\|_2^m}{1 + \|B_J\|_2} \left\| x^{(1)} - x^{(0)} \right\|$$

Primero hallamos el vector $x^{(1)}$:

$$x_1^{(1)} = \frac{2 + x_2^{(0)}}{2} = 1$$

$$x_2^{(1)} = \frac{4 + x_1^{(0)} - x_3^{(0)}}{2} = 2$$

$$x_3^{(1)} = \frac{2 - x_2^{(0)}}{2} = 1$$

Por tanto:

$$\left\| x^{(1)} - x^{(0)} \right\|_2 = \left\| \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix} \right\|_2 = 2$$

$$\|B_J\|_2 = \sqrt{\rho\left(B_J^t \cdot B_J\right)} = \sqrt{\rho\left(B_J^2\right)} = \rho(B_J) = \frac{1}{\sqrt{2}}$$

Ya que:

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

$$\frac{\left(\frac{1}{\sqrt{2}}\right)^m}{1 - \frac{1}{\sqrt{2}}} \cdot 2 \leq 5 \cdot 10^{-5}$$

Tomando logaritmo en base $1/\sqrt{2}$ y despejando m:

$$m \geq 34'11$$

■

Ejemplo 2.14:

Dada la matriz $A(\beta)$, determinar si existen valores de β para los cuales es convergente el método de Jacobi pero no el de Gauss-Seidel.

$$A(\beta) = \begin{pmatrix} 2 & -1 & \beta \\ 1 & 2 & -1 \\ 0 & 1 & 2 \end{pmatrix}$$

Solución:

Necesitamos verificar la condición necesaria y suficiente.

Jacobi: $|D\lambda + L + R| = 0 = 8\lambda^3 + 4\lambda + \beta$

Gauss-Seidel: $|(D+L)\lambda + R| = 0 = \lambda^2(8\lambda + \beta + 4) \Rightarrow \begin{cases} \lambda_1 = 0 \text{ doble} \\ \lambda_2 = -\frac{\beta+4}{8} \end{cases}$

Gauss-Seidel converge si $\rho(B_{GS}) = \left|\frac{\beta+4}{8}\right| < 1$, es decir, si $-12 < \beta < 4$.

Si $\beta = 0$ convergen ambos métodos.

Jacobi: tenemos un polinomio de orden 3, con su derivada $p'(\lambda) = 24\lambda^2 + 4 > 0$, es decir, que $p(\lambda)$ es monótono creciente, con $p(-\infty) = -\infty$ y $p(\infty) = \infty$, lo que significa que sólo tiene una raiz real, al ser monótono creciente. Calcularemos las raíces complejas en función de una real que suponemos λ_1. Aplicando Ruffini, llegamos a:

$$8\lambda^2 + 8\lambda_1\lambda + \left(8\lambda_1^2 + 4\right) = 0$$

$$\lambda_2\lambda_3 = \frac{8\lambda_1^2 + 4}{8} = \lambda_1^2 + \frac{1}{2}$$

Al ser λ_2 y λ_3 raíces complejas conjugadas, tienen el mismo módulo:

$$|\lambda_2|^2 = |\lambda_3|^2 = \frac{8\lambda_1^2 + 4}{8} > 0$$

El radio espectral de B_J es:

$$\rho(B_J) = \sqrt{\lambda_1^2 + \frac{1}{2}}$$

Lo cual nos lleva a que $\lambda_1^2 < 0.5$.

Despejando β del polinomio sustituyendo λ_1:

$$\beta = -\left(8\lambda_1^3 + 4\lambda_1\right)$$

Con $\lambda_1 = \frac{1}{\sqrt{2}}$:

$$-4\sqrt{2} < \beta < 4\sqrt{2}$$

Es la condición de convergencia del método de Jacobi.

Intersecando las dos condiciones, para que converja Jacobi y no lo haga Gauss-Seidel:

$$4 \leq \beta < 4\sqrt{2}$$

Ya que para $\beta = 4$ Gauss-Seidel no converge.

■

2.5.5 Métodos iterativos basados en la optimización de una función

Se basa en pasar el problema de resolver un sistema de ecuaciones $Ax = b$ a un problema de optimización. Para ello se define una función a minimizar, la **función objetivo**:

$$\phi(x) = \frac{1}{2}x^t Ax - b^t x \qquad \phi : \mathbb{R}^n \to \mathbb{R} \tag{2.58}$$

Siendo A matriz simétrica definida positiva.

Por tanto, se trata de hallar $\min \phi(x)$. La condición necesaria de mínimo es:

$$x^* \text{ es minimo } \Leftrightarrow \nabla\phi(x^*) = 0$$

Por tanto:

$$\nabla\phi(x) = Ax - b$$

Que satisface:

$$Ax^* = b$$

La función que hemos definido tiene un único mínimo, global, y es estrictamente convexa.

$$\phi(x^* + v) = \phi(x^*) + v^t\nabla\phi(x^*) + \frac{1}{2}v^t Av = \phi(x^*) + \frac{1}{2}v^t Av > \phi(x^*)$$

Ahora se trata de construir la solución $x^{(s)}$ de forma que $\phi\left(x^{(s+1)}\right) < \phi\left(x^{(s)}\right)$.

De esa forma:

$$x^{(s+1)} = x^{(s)} + \alpha_s d^{(s)}$$

Donde $d^{(s)}$ es la **dirección de movimiento** y α_s es la **longitud de paso**, en la dirección $d^{(s)}$. Se trata de un proceso de búsqueda o movimiento hacia la solución, es decir, que sólo interesan las direcciones hacia las que ϕ decrece.

Fijada una dirección $d^{(s)}$, y desarrollando en series de Taylor:

$$\phi\left(x^{(s)} + \alpha d^{(s)}\right) = \varphi(\alpha) = \phi\left(x^{(s)}\right) + \alpha d^{(s)}\nabla\phi\left(x^{(s)}\right) + \frac{1}{2}\alpha^2 d^{(s)^t}\nabla^2\phi\left(x^{(s)}\right)d^{(s)} + \dots$$

Haciendo búsqueda en línea exacta:

$$\varphi'(\alpha) = 0 = d^{(s)^t}\nabla\phi\left(x^{(s)}\right) + \alpha d^{(s)^t}Ad^{(s)}$$

Donde hemos sustituido la Hessiana por A ya que es una forma cuadrática, por lo que es la propia matriz.

Despejando α:

$$\alpha = -\frac{d^{(s)^t}\nabla\phi\left(x^{(s)}\right)}{d^{(s)^t}Ad^{(s)}} = \frac{d^{(s)^t}r^{(s)}}{d^{(s)^t}Ad^{(s)}} \tag{2.59}$$

Donde $r^{(s)}$ es el residual,

$$r^{(s)} = b - Ax^{(s)}$$

2.5.5.1 Método del gradiente

La dirección es el residual, es decir, la de máximo decrecimiento, que cambiada de signo es la de máximo crecimiento, ya que $r = -\nabla\phi(x)$.

Mientras $r^{(s)} \neq 0$,

$$x^{(s+1)} = x^{(s)} + \alpha_s r^{(s)} \tag{2.60}$$

Con

$$\boxed{\alpha_s = \frac{r^{(s)^t} r^{(s)}}{r^{(s)^t} A r^{(s)}}} \tag{2.61}$$

Se puede demostrar que se cumple:

$$\phi\left(x^{(s+1)}\right) + \frac{b^t A^{-1} b}{2} \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1}\right)^2 \left(\phi\left(x^{(s)}\right) + \frac{b^t A^{-1} b}{2}\right)$$

La convergencia es lineal, salvo si $\kappa_2(A) = 1$, y es más lenta cuanto mayor sea $\kappa_2(A)$.

A continuación vemos el código en Matlab que lo implementa.

Algoritmo 2.12: Método del gradiente para sistemas lineales

```
% Metodo del gradiente para sistemas lineales
function [xs,iter]=gradsl(A,b,x0,maxiter)
if nargin<4
   disp('Faltan argumentos de entrada');
   return;
end;

r=b-A*x0;
iter=1;
x(1,:)=x0';
while (norm(r)~=0)&(iter<maxiter)
   alpha=(r'*r)/(r'*A*r);
   x(iter+1,:)=x(iter,:)+alpha*r';
   iter=iter+1;
   r=b-A*x(iter,:)';
end;

xs=x(iter,:);
```

♦

2.5.5.2 Método del gradiente conjugado

La dirección que se escoge es conjugada con las anteriores, es decir:

$$d^{(s)^t} A d^{(j)} = 0 \qquad j = 1 : s-1$$

El algoritmo es el siguiente:

$$
\begin{array}{ll}
s = 0 & r^{(0)} = b - Ax^{(0)} \\
\text{Mientras } r^{(s)} \neq 0 & \left\{ \begin{array}{ll}
d^{(0)} = r^{(0)} & \text{ó si } s > 0,\ d^{(s)} = r^{(s)} + \beta_s d^{(s-1)} \\
x^{(s+1)} = x^{(s)} + \alpha_s d^{(s)} & \beta_s = \frac{\left\| r^{(s)} \right\|_2^2}{\left\| r^{(s-1)} \right\|_2^2} \\
r^{(s+1)} = r^{(s)} - \alpha_s A d^{(s)} & \alpha_s = \frac{\left\| r^{(s)} \right\|_2^2}{d^{(s)^t} A d^{(s)}} \\
s = s + 1 &
\end{array} \right.
\end{array}
\tag{2.62}
$$

Con matrices vacías representa muy poco cálculo.

Ahora vemos el código en Matlab que implementa este método.

Algoritmo 2.13: Método del gradiente conjugado para sistemas lineales

```
% Metodo del gradiente conjugado para sistemas lineales
function [xs,iter]=gconjsl(A,b,x0,maxiter)

if nargin<4
   disp('Faltan argumentos de entrada');
   return;
end;
r(:,1) = b - A*x0; % Residuo inicial
iter = 1;          % Contador de iteraciones
x(:,1) = x0;       % Almacenar las soluciones sucesivas
                   % (habria que ahorrar almacenando
                   % solo las 2 ultimas)
d(:,1) = r(:,1);   % Direccion de descenso inicial

while (norm(r(:,iter)) ~= 0) & (iter < maxiter)
   alpha = norm(r(:,iter),2)^2/(d(:,iter)'*A*d(:,iter));
   r(:,iter+1) = r(:,iter) - alpha*A*d(:,iter);
   beta = (norm(r(:,iter+1),2)/norm(r(:,iter),2))^2;
   x(:,iter+1) = x(:,iter) + alpha*d(:,iter);
   iter = iter + 1;
   d(:,iter) = r(:,iter) + beta*d(:,iter-1);
end;

xs = x(:,iter);
```

♦

Teorema 2.13. Dado un sistema $Ax = b$ siendo A simétrica definida positiva, si $x^{(s)}$ es la s-ésima iteración calculada por el método del gradiente conjugado, se cumple:

$$
\phi\left(x^{(s)}\right) + \frac{b^t A^{-1} b}{2} \leq 4 \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^{2s} \left(\phi\left(x^{(0)}\right) + \frac{b^t A^{-1} b}{2} \right)
$$

En teoría es más rápido, ya que acaba a lo sumo en n iteraciones.

Ejemplo 2.15:

Si el número de condición de la matriz A en norma 2 es $\kappa_2(A) = 4$ y estamos en la iteración $s = 2$, por el método del gradiente conjugado tenemos

$$4\left(\frac{\sqrt{\kappa_2(A)}-1}{\sqrt{\kappa_2(A)}+1}\right)^{2s} = 4\left(\frac{1}{9}\right)^2 = \frac{4}{81} \simeq 0.05$$

Es decir, que reduce al $5\,\%$ el valor de la función objetivo en sólo 2 iteraciones. Por contra, con el método del gradiente

$$\left(\frac{\kappa_2(A)-1}{\kappa_2(A)+1}\right)^2 = \left(\frac{3}{5}\right)^2 = 0.36$$

Sólo lo reduce al $36\,\%$, por lo que la convergencia del método del gradiente conjugado es más rápida.

■

Si A es mal condicionada, se puede usar el **método del gradiente conjugado precondicionado**, en el que se elige una matriz C simétrica definida positiva y se aplica el método del gradiente conjugado al siguiente sistema transformado

$$\hat{A}\hat{x} = \hat{b}$$

Siendo

$$\hat{A} = C^{-1}AC^{-1} \qquad \hat{x} = Cx \qquad \hat{b} = C^{-1}b$$

Se puede evitar hacer referencia a C^{-1} en el algoritmo usando el precondicionador $M = C^2$. Para elegir un buen precondicionador hay diversas técnicas, que no vamos a explicar.

TEMA 3

Álgebra lineal numérica II

3.1 Cálculo de autovalores y autovectores

3.1.1 Introducción

Vamos a tratar de encontrar los valores $\lambda_i \in \mathbb{C}$ tales que $Ax = [\lambda_i]\, x$ para $x \neq 0$ y $A \in \mathbb{C}^{n \times n}$.

Esto es, tratamos de resolver el problema de calcular los autovalores λ y los autovectores v tales que

$$Av = \lambda v, \quad v \neq 0$$

Habrá que resolver

$$(A - \lambda I)\, v = 0$$

por lo que, para que $v \neq 0$, debe ser

$$det\,(A - \lambda I) = 0$$

Se llama **ecuación característica** de la matriz A a $det\,(A - \lambda I_n) = 0$, y polinomio característico a $p\,(\lambda) = det\,(A - \lambda I_n)$.

Si disponemos los autovectores v_i como columnas y llamamos a D la matriz diagonal con elementos los autovalores λ_i,

$$AV = VD, \qquad V = \begin{pmatrix} v_1 & \dots & v_n \end{pmatrix}, D = diag\,(\lambda_i)$$

Se define el **espectro de una matriz** como el conjunto de autovalores de dicha matriz. El **radio espectral** es el máximo en módulo de los autovalores.

El determinante de una matriz es el producto de todos sus autovalores.

Se define la **traza de una matriz** como:

$$traza\,(A) = \lambda_1 + \lambda_2 + \ldots + \lambda_n = \sum_{i=1}^{n} a_{ii}$$

Se dice que una matriz hermítica o simétrica es definida positiva si y sólo si sus autovalores son positivos, y además los autovalores de toda matriz hermítica o simétrica son reales.

Los autovalores de la inversa de una matriz son los inversos de los autovalores de la matriz sin invertir.

- ✏ Dos matrices semejantes tienen los mismos autovalores. Recordemos que dos matrices A y B son semejantes si existe una matriz P invertible tal que $P^{-1}AP = B$. Entonces,

$$Bv = \lambda v \Rightarrow PBv = \lambda Pv \Rightarrow A\left(Pv\right) = \lambda\left(Pv\right)$$

  Si λ es autovalor de B con autovector v, también es autovalor de A con autovector Pv

- ✏ El módulo de cualquier autovalor de una matriz ortogonal o unitaria es la unidad. Una matriz $A \in \mathbb{R}^{n\times n}$ es ortogonal si $A^{-1} = A^t$, mientras que una matriz $A \in \mathbb{C}^{n\times n}$ es unitaria si $A^{-1} = A^*$

- ✏ Si existe V^{-1}, entonces

$$V^{-1}AV = D$$

  y A es diagonalizable por semejanza

- ✏ Si V^{-1} es ortogonal, entonces A es diagonalizable por semejanza ortogonal o unitaria

- ✏ Relación entre el espectro de A y el de A^2:

$$Av = \lambda v \qquad A^2v = \lambda Av = \lambda^2 v$$

- ✏ Si v es autovector, ¿lo es también $-v$?

$$A\left(-v\right) = -Av = -\lambda v = \lambda\left(-v\right)$$

  Pues sí, resulta que es autovector con el mismo autovalor asociado

Teorema de Cayley-Hamilton. Toda matriz satisface su ecuación característica: $p\left(A\right) = 0$.

Teorema 3.1. Una matriz A es diagonalizable por semejanza unitaria si y sólo si es normal ($A^*A = AA^*$).

Por tanto, toda matriz compleja hermítica es diagonalizable por una transformación de semejanza unitaria y toda matriz real simétrica es diagonalizable por semejanza ortogonal.

Teorema de Rayleigh. Dada una matriz compleja y hermítica cuyos autovalores cumplen que $\lambda_n \leq \lambda_{n-1} \leq \ldots \leq \lambda_1$:

$$\lambda_n \leq \frac{x^*Ax}{x^*x} \leq \lambda_1 \quad x \in \mathbb{C}^n - \{0\}$$

y cada cota se alcanza cuando x es el autovector del autovalor correspondiente.

Esto no quiere decir otra cosa que si v es un autovector, se puede calcular su autovalor asociado como

$$\boxed{\lambda = \frac{v^tAv}{v^tv}} \tag{3.1}$$

Este cociente se conoce con el nombre de cociente de Rayleigh.

Teorema de Jordan. Toda matriz es semejante a una matriz de Jordan:

$$J = \begin{pmatrix} J_1\left(\lambda_1\right) & 0 & \ldots & 0 \\ 0 & J_2\left(\lambda_2\right) & \ldots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & J_k\left(\lambda_k\right) \end{pmatrix} \quad \text{con } J_i\left(\lambda_i\right) = \begin{pmatrix} \lambda_i & 1 & 0 & \ldots & 0 \\ 0 & \lambda_i & 1 & \ldots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ 0 & 0 & 0 & \ldots & \lambda_i \end{pmatrix} \tag{3.2}$$

Esto es, existe una matriz P invertible tal que

$$P^{-1}AP = J$$

En Matlab simbólico se puede calcular con la primitiva `jordan`.

El cálculo numérico de la forma de Jordan es muy inestable, por eso se usa la de Schur, que vemos en el siguiente teorema.

Teorema de Schur. Dada una matriz A, existe otra matriz unitaria U de forma que se cumple:

$$U^*AU = T$$

Siendo T triangular superior y los elementos de su diagonal son los autovalores de A.

Se trata de una transformación unitaria, ya que

$$U^{-1} = U^*$$

El **problema de autovalores generalizado**

$$Ax = \lambda Bx$$

puede ser resuelto, y en Matlab se usa para ello la función `eig(A,B)`.

Para resolverlo se puede hacer de diversas formas que vamos a ver posteriormente:

- ✏ resolver la ecuación característica obteniendo las raíces del polinomio característico
- ✏ con el método de las potencias obteniendo el autovalor de mayor módulo y luego usar deflación

3.1.2 Localización de los autovalores

Teorema de los círculos de Gerschgorin. Dada una matriz A real y cuadrada de orden n, y definiendo el dominio unión de los círculos de Gerschgorin:

$$D = \bigcup_{i=1}^{n} C_i \qquad C_i = \{z \in \mathbb{C} \,/\, |z - a_{ii}| \leq r_i\} \qquad r_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad i = 1:n \tag{3.3}$$

Si λ es un autovalor, entonces está dentro del dominio D.

Lo que quiere decir la definición es que el centro de los círculos de Gerschgorin son las componentes de la diagonal principal en cada fila, y los radios la suma de los elementos de la fila exceptuando el de la diagonal principal.

Cualquier punto del plano complejo que caiga fuera de los círculos de Gerschgorin no será autovalor, mientras que en un círculo de Gerschgorin aislado habrá un único autovalor.

Ejemplo 3.1:

Dada la matriz

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

Se tiene que:

	Centro	Radio
C_1	-1	6
C_2	1	5
C_3	3	2

■

✏ Si A es de diagonal dominante estrictamente, es invertible. Por otro lado, $A-\lambda I$ no puede ser de diagonal dominante.

Teorema de Brauer. Dada la matriz A real y cuadrada de orden n, y un entero m tal que $1 \leq m \leq n$ y

$$|a_{mm} - a_{ii}| > \sigma_m + \sigma_i \quad i = 1:n,\, i \neq m$$

En ese caso, el círculo de Gerschgorin $C_m = \{z \in \mathbb{C} \,/\, |z - a_{mm}| \leq \sigma_m\}$ es disjunto de los demás círculos de Gerschgorin, y contiene un único autovalor de la matriz A.

3.1.3 Métodos de cálculo del polinomio característico

3.1.3.1 Introducción

Denotamos de forma general el polinomio característico de la matriz A:

$$p(\lambda) = |A - \lambda I| = (-1)^n \left(\lambda^n + p_1\lambda^{n-1} + \ldots + p_n\right)$$

Como se ve, el coeficiente del término de mayor grado es $(-1)^n$.

Se puede escribir también como su **matriz de compañía**, definida de la forma

$$A = \begin{pmatrix} -p_1 & \ldots & -p_{n-1} & -p_n \\ & I_{n-1} & & 0 \end{pmatrix} = \begin{pmatrix} -p_1 & -p_2 & -p_3 & \ldots & -p_n \\ 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Esta matriz a veces se define con los coeficientes del polinomio dispuestos en la última columna.

Si el término de mayor grado del poliniomio no fuera ± 1, la matriz se define, siendo a el coeficiente de mayor grado:

$$A = \begin{pmatrix} -p_1/a & -p_2/a & -p_3/a & \ldots & -p_n/a \\ 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Esto se puede usar para calcular las raíces de un polinomio, ya que los autovalores de esa matriz son las raíces del polinomio. Es lo que hace la función `roots` de Matlab.

Sólo nos falta poder calcular los p_i.

Ejemplo 3.2:

Dadas las matrices A_1 y A_2

$$A_1 = \begin{pmatrix} 0 & I_{99} \\ 0 & 0 \end{pmatrix} \qquad A_2 = \begin{pmatrix} 0 & I_{99} \\ \varepsilon & 0 \end{pmatrix}$$

Con $\varepsilon = 10^{-100}$ y siendo I_{99} la matriz identidad de dimensiones 99×99. Si calculamos los autovalores:

$$det\,(A_1 - \lambda I) = \lambda^{100} = 0 \Rightarrow \lambda = 0, |\lambda| = 0$$
$$det\,(A_2 - \lambda I) = \lambda^{100} - \varepsilon = 0 \Rightarrow |\lambda| = \left|10^{-1}\right| = 0.1$$

Se trata de un problema inestable, ya que un pequeño cambio en los datos provoca un gran cambio en el resultado.

■

3.1.3.2 Número de condición para el cálculo de autovalores

Sólo para matrices diagonalizables:

$$V^{-1}AV = D$$

Si $espectro\,(A) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, y $\lambda^* \in espectro\,(A + e_a\,(A))$,

$$d = \text{mín}\,\{|\lambda^* - \lambda_i|\} \leq r = \|V\| \left\|V^{-1}\right\| \|e_a\,(A)\|$$

$$\boxed{N_C = \|V\| \left\|V^{-1}\right\|} \tag{3.4}$$

Las matrices simétricas son bien condicionadas, ya que hemos dicho antes que toda matriz real simétrica (o compleja hermítica) es diagonalizable por semejanza ortogonal, por lo que

$$\|V\| \left\|V^{-1}\right\| = 1$$

3.1.3.3 Método de Le-Verrier

Utiliza la relación entre las raíces de una ecuación y sus coeficientes (relaciones de Newton):

$$a_0x^n + a_1x^{n-1} + \ldots + a_n \;\rightarrow\; \begin{cases} a_0s_1 + a_1 = 0 \\ a_0s_k + a_1s_{k-1} + \ldots + a_{k-1}s_1 + ka_k = 0 \qquad k = 2:n \\ s_k = \sum_{i=1}^{n} x_i^k \end{cases}$$

Que, trasladado a nuestro polinomio característico:

$$\boxed{\begin{array}{l} p_1 = -s_1 \\ p_k = -\frac{1}{k}\left(s_k + p_1s_{k-1} + \ldots + p_{k-1}s_1\right) \quad k = 2:n \\ s_k = \sum_{i=1}^{n} \lambda_i^k = traza\left(A^k\right) \end{array}} \tag{3.5}$$

Los s_k se obtienen como suma de los autovalores de potencias de la matriz A.

El orden de este método es de $O\left(n^4\right)$ operaciones.

El gran problema de este método es calcular las potencias de la matriz A.

Se puede seguir el siguiente esquema:

1. Determinar las potencias de la matriz A

$$A^0, A^1, A^2, \ldots, A^n$$

2. Calcular las trazas como la suma de los elementos de la diagonal principal de A

$$s_1 = traza\left(A\right), s_2 = traza\left(A^2\right), \ldots, s_n = traza\left(A^n\right)$$

3. A continuación se obtienen los coeficientes del polinomio característico de la forma

$$p_1 = -s_1$$

$$p_2 = -\left(s_2 + p_1 s_1\right)/2$$

$$\vdots$$

$$p_n = -\left(s_n + p_1 s_{n-1} + \ldots + p_{n-1} s_1\right)/n$$

Seguidamente se incluye el código en Matlab que implementa este método.

Algoritmo 3.1: Método de Le-Verrier para calcular el polinomio característico

```
% Metodo de Le-Verrier para el calculo del polinomio
% caracteristico de una matriz para hallar sus
% autovalores.
function y=leverrie(A)
[m,n]=size(A);
for k=1:n
   s(k)=trace(A^k);
end;
p(1)=-s(1);
a=zeros(1,n);
for k=2:n
   for l=1:k-1
      a(k)=a(k)+s(k-l)*p(l);
   end;
   p(k)=-(1/k)*(s(k)+a(k));
end;
y=(-1)^n*[1 p];
```

♦

3.1.3.4 Método de Faddeev-Souriau

Es una modificación del método anterior que reorganiza el cálculo. Partiendo de $B_0 = I_n$:

$$\begin{array}{|ccc|}\hline A_1 = AB_0 & p_1 = -traza\,(A_1) & B_1 = A_1 + p_1 I_n \\ A_2 = AB_1 & p_2 = -\frac{traza(A_2)}{2} & B_2 = A_2 + p_2 I_n \\ \vdots & \vdots & \vdots \\ A_k = AB_{k-1} & p_k = -\frac{traza(A_k)}{k} & B_k = A_k + p_k I_n \\ \hline \end{array} \tag{3.6}$$

Al final del proceso, en virtud del teorema de Cayley-Hamilton, $B_k = 0$.

A continuación vemos el algoritmo en Matlab para el método de Faddeev-Souriau.

Algoritmo 3.2: Método de Faddeev-Souriau para calcular el polinomio característico

```
% Metodo de Faddeev-Souriau para calculo de
% polinomio caracteristico
function [y,inv]=faddeev(A)
[m n]=size(A);
AA=A;
for k=1:n
   q(k)=trace(AA)/k;
   B=AA-q(k)*eye(n);
   if k==(n-1)
      casinv=B;
   end;
   AA=A*B;
   p(k)=-q(k);
end;
inv=casinv/q(n);
y=[1 p];
```

♦

Ejemplo 3.3:

Dada la matriz A

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

Calculamos el polinomio característico con el método de Faddeev-Souriau:

1.

$$A_1 = A$$
$$p_1 = -traza\,(A_1) = -1$$
$$B_1 = A_1 + p_1 I = \begin{pmatrix} 0 & -1 & -1 & 2 \\ 2 & 2 & 0 & -4 \\ 1 & 1 & -3 & -2 \\ 1 & 1 & 0 & -2 \end{pmatrix}$$

2.

$$A_2 = AB_1 = \begin{pmatrix} -1 & -2 & 2 & 4 \\ 2 & 0 & -2 & 0 \\ -2 & -3 & 5 & 6 \\ 1 & 0 & -1 & 0 \end{pmatrix}$$

$$p_2 = -\frac{1}{2} traza\left(A_2\right) = -2$$

$$B_2 = A_2 + p_2 I = \begin{pmatrix} -3 & -2 & 2 & 4 \\ 2 & -2 & -2 & 0 \\ -2 & -3 & 3 & 6 \\ 1 & 0 & -1 & -2 \end{pmatrix}$$

3.

$$A_3 = AB_2 = \begin{pmatrix} -1 & 3 & -1 & -6 \\ -4 & -10 & 2 & 16 \\ 1 & 2 & -4 & -4 \\ -2 & -4 & 1 & 6 \end{pmatrix}$$

$$p_3 = -\frac{1}{3} traza\left(A_3\right) = 3$$

$$B_3 = A_3 + p_3 I = \begin{pmatrix} 2 & 3 & -1 & -6 \\ -4 & -7 & 2 & 16 \\ 1 & 2 & -1 & -4 \\ 2 & -4 & 1 & 9 \end{pmatrix}$$

4.

$$A_4 = AB_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$p_4 = -\frac{1}{4} traza\left(A_4\right) = -1$$

5.

$$p\left(\lambda\right) = \lambda^4 - \lambda^3 - 2\lambda^2 + 3\lambda - 1$$

■

3.1.3.5 Método de Danilevski

Sólo como mención. Consiste en transformar la matriz A en una equivalente (por semejanza) cuyo polinomio característico sea fácil de determinar. Se transforma a una matriz de Frobenius:

$$A \rightarrow \begin{pmatrix} 0 & \dots & \dots & -p_n \\ 1 & 0 & \dots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 1 & -p_1 \end{pmatrix}$$

$$|F - \lambda I| = p\left(\lambda\right)$$

3.2 Casos sencillos para calcular el polinomio característico

3.2.1 Si A es tridiagonal

$$A = \begin{pmatrix} a_1 & c_2 & \dots & 0 \\ b_2 & a_2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & b_n & a_n \end{pmatrix}$$

$$\begin{aligned} |A-\lambda I| &= \begin{vmatrix} a_1-\lambda & c_2 & \dots & 0 \\ b_2 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \dots & b_n & a_n-\lambda \end{vmatrix} = \\ &= (a_n-\lambda)\,|A_{n-1}-\lambda I| - b_n \begin{vmatrix} a_1-\lambda & & & \\ & \ddots & & \\ & & \ddots & \\ & & b_{n-1} & c_n \end{vmatrix} = \\ &= (a_n-\lambda)\,p_{n-1}(\lambda) - b_n c_n p_{n-2}(\lambda) \end{aligned}$$

Entonces, queda, para los polinomios sucesivos:

$$\begin{aligned} &p_0(\lambda) = 1 \\ &p_1(\lambda) = a_1-\lambda \\ &p_k(\lambda) = (a_k-\lambda)\,p_{k-1}(\lambda) - b_k c_k p_{k-2}(\lambda) \qquad k = 2:n \end{aligned} \tag{3.7}$$

Ejemplo 3.4:

Para una matriz A 3×3 genérica,

$$A = \begin{pmatrix} a_1 & c_2 & 0 \\ b_2 & a_2 & c_3 \\ 0 & b_3 & a_3 \end{pmatrix}$$

Se tiene

$$\begin{aligned} det\,(A-\lambda I) &= \begin{vmatrix} a_1-\lambda & c_2 & 0 \\ b_2 & a_2-\lambda & c_3 \\ 0 & b_3 & a_3-\lambda \end{vmatrix} \\ &= (a_3-\lambda)\begin{vmatrix} a_1-\lambda & c_2 \\ b_2 & a_2-\lambda \end{vmatrix} - b_3 \begin{vmatrix} a_1-\lambda & 0 \\ b_2 & c_3 \end{vmatrix} \\ &= (a_n-\lambda)\,p_{n-1}(\lambda) - b_n c_n p_{n-2}(\lambda) \end{aligned}$$

■

Para que funcione el método, deben ser todos los $b_i \neq 0$. Si hay algún $b_k = 0$ se hace por bloques:

$$A = \begin{pmatrix} \boxed{B} & \\ & \boxed{C} \end{pmatrix} \Rightarrow det\,(A - \lambda I) = det\,(B - \lambda I)\,det\,(C - \lambda I)$$

Ejemplo 3.5:

$$A = \begin{pmatrix} \boxed{\begin{matrix} a_1 & c_2 \\ b_2 & a_2 \end{matrix}} & \\ & \boxed{a_3} \end{pmatrix} \Rightarrow det\,(A - \lambda I) = \begin{vmatrix} a_1 - \lambda & c_2 \\ b_2 & a_2 - \lambda \end{vmatrix} |a_3 - \lambda|$$

■

3.2.2 Si A es Hessenberg superior

Una matriz Hessenberg superior es una matriz triangular superior que también tiene elementos en la subdiagonal principal:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ b_2 & a_{22} & \dots & a_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & b_n & a_{nn} \end{pmatrix}$$

Con todos los $b_i \neq 0$ para $i = 2 : n$.

Al resolver el problema de autovalores (para $n = 4$):

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ b_2 & a_{22} & a_{23} & a_{24} \\ 0 & b_3 & a_{33} & a_{34} \\ 0 & 0 & b_4 & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \lambda \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

Se tiene un sistema compatible indeterminado. Por tanto, lo que se hace es darle un valor a x_4, por ejemplo

$$x_4 = 1 \equiv s_4\,(\lambda)$$

Seguimos resolviendo y tenemos

$$b_4 x_3 + a_{44} = \lambda \Rightarrow \underline{x_3 = \frac{1}{b_4}\,(\lambda - a_{44}) \equiv s_3\,(\lambda)}$$

$$b_3 x_2 + a_{33} s_3\,(\lambda) + a_{34} s_4\,(\lambda) = \lambda s_3\,(\lambda)$$

$$\underline{x_2 = \frac{1}{b_3}\,((\lambda - a_{33})\,s_3\,(\lambda) - a_{34} s_4\,(\lambda)) \equiv s_2\,(\lambda)}$$

$$b_2 x_1 + a_{22} s_2\,(\lambda) + a_{23} s_3\,(\lambda) + a_{24} s_4\,(\lambda) = \lambda s_2\,(\lambda)$$

$$\underline{x_1 = \frac{1}{b_2}\,((\lambda - a_{22})\,s_2\,(\lambda) - a_{23} s_3\,(\lambda) - a_{24} s_4\,(\lambda)) \equiv s_1\,(\lambda)}$$

$$a_{11} s_1\,(\lambda) + a_{12} s_2\,(\lambda) + a_{13} a_3\,(\lambda) + a_{14} s_4\,(\lambda) = \lambda s_1\,(\lambda)$$

$$c\,((\lambda - a_{11})\,s_{11}\,(\lambda) - a_{12} s_2\,(\lambda) - a_{13} s_3\,(\lambda) - a_{14} s_4\,(\lambda)) = 0$$

Siendo c una constante por determinar. El coeficiente de λ^n que nos queda es

$$\frac{c}{b_2 b_3 b_4}$$

y como queremos que sea 1, entonces

$$c = b_2 b_3 b_4$$

Si alguno de los b_i es nulo, se puede hacer como en el caso de tridiagonal, por bloques.

Lo que hemos hecho se llama **método de Hyman**, que, para el caso general, queda la siguiente recurrencia

$$\boxed{\begin{array}{l} s_n(\lambda) = 1 \\ \\ s_{k-1}(\lambda) = -\dfrac{(a_{kk} - \lambda)\, s_k(\lambda) + \displaystyle\sum_{j=k+1}^{n} a_{kj} s_j(\lambda)}{b_k} \qquad k = n : -1 : 2 \end{array}} \tag{3.8}$$

Si $b_k \neq 0$.

De esta forma, se obtiene:

$$det\,(A - \lambda I_n) = (-1)^{n+1} b_2 \cdots b_n \left((a_{11} - \lambda)\, s_1(\lambda) + \sum_{j=2}^{n} a_{1j} s_j(\lambda) \right)$$

Si se tiene $b_i = 0$ para algún i, se pueden hacer las recurrencias de las submatrices cuadradas principales.

3.3 Métodos de reducción a matriz tridiagonal y a matriz Hessenberg

3.3.1 Reducción por semejanza con el método de Givens

La reducción por semejanza de una matriz a tridiagonal (si es simétrica) o Hessenberg superior (en otro caso) se puede hacer por los métodos de Givens y Householder:

$$G_p^t \cdots G_1^t A G_1 \cdots G_p$$
$$H_{n-1} \cdots H_2 A H_2 \cdots H_{n-1}$$

Ahora vamos a ver con algo más de detalle el método de Givens, y posteriormente el de Householder.

Se hacen rotaciones para anular elementos por debajo de la subdiagonal principal. Ilustramos el método con una matriz 4×4 genérica:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Usando el índice i para las columnas y el q para las filas, para $i = 1$ usamos la matriz G_{23} para anular el elemento $(q, i) = (3, 1)$, y se hace

$$G_{23}^t A G_{23}$$

Recordemos que la matriz G_{23} es

$$G_{23} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & s & 0 \\ 0 & -s & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Hay que premultiplicar y postmultiplicar para que sean matrices semejantes.

Siguiendo con $i = 1$, para anular el elemento $(4, 1)$ se usa la matriz G_{24}

$$G_{24} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & 0 & s \\ 0 & -s & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ahora tendremos ceros en la primera columna, filas 3 y 4:

$$\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & a_{34}^{(1)} \\ 0 & a_{42}^{(1)} & a_{43}^{(1)} & a_{44}^{(1)} \end{pmatrix}$$

Se pasa a la segunda columna, $i = 2$, y se anula el elemento $(4, 2)$ con G_{34}

$$G_{34} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c & s \\ 0 & 0 & -s & c \end{pmatrix}$$

Por tanto, el método será:

- ✏ Anular columnas $i = 1 : n-2$ haciendo rotaciones $G_{i+1,q}$ para anular el elemento (q, i) con $q = i+2 : n$

Si la matriz es simétrica, entonces al anular el elemento (q, i) se anula también el (i, q), por lo que la matriz se reduce a tridiagonal.

3.3.2 Reducción por semejanza con el método de Householder

Cuando aplicamos el método de Householder a una matriz, hacíamos ceros por debajo de la diagonal principal. Ahora nos interesa hacerlos por debajo de la subdiagonal principal, por lo que hay que modificar ligeramente el método:

- ✏ La matriz queda igual que antes

$$H_i = I_n - \frac{2}{w^{(i)^t} w^{(i)}} w^{(i)} w^{(i)^t}$$

- ✏ El vector $w^{(i)}$ es ahora

$$w^{(i)} = \begin{pmatrix} \left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} i \text{ ceros} \\ v^{(i)} \end{pmatrix}$$

mientras que el vector $v^{(i)}$ se construye de forma similar al método de Householder tradicional:

$$v^{(i)} = \begin{pmatrix} a_{i+1,i} + signo\left(a_{i+1,i}\right) \|A\left(i+1:n,i\right)\|_2 \\ a_{i+2,i} \\ \vdots \\ a_{n,i} \end{pmatrix}$$

Esto es, en lugar de empezar en el elemento de la diagonal principal como se hacía con los sistemas lineales, se empieza a construir el vector v con los elementos de la subdiagonal principal. Así se eliminan los elementos de la columna $i-1$ en las filas $i+1$ hasta la n.

De nuevo, habrá que postmultiplicar y premultiplicar, de forma que en general se tiene

$$A_k = H_k \cdots H_2 H_1 A H_1 H_2 \cdots H_k$$

Y si la matriz A es simétrica se obtendrá una matriz tridiagonal.

A continuación vemos la implementación en Matlab de este método, que sólo representa unas pequeñas variaciones con respecto al método de Householder para sistemas lineales.

Algoritmo 3.3: Método de Householder para reducir una matriz a Hessenberg superior o tridiagonal

```
% Metodo de Householder para reducir
% A a tridiagonal o Hessenberg superior
% usado en el cálculo de autovalores
function [A] = houseig(A)
[m n] = size(A);
for i=1:n-1,
   w = zeros(n,1);
   v = A(i+1:n,i);
   nor = norm(v);
   if nor > 10*eps;
      signo = sign(v(1));
      if abs(v(1)) < 10*eps,
         signo = 1;
      end;
      v(1) = v(1) + signo*nor;
      w(i+1:n) = v;
      cociente = 2*(w*w')/(w'*w);
      A = A-cociente*A-cociente*A+cociente*A*cociente;
   end;
end;
```

♦

Ejemplo 3.6:

Dada la matriz

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

Al hacer ceros debajo de la subdiagonal principal en la primera columna, tenemos que usar

$$v^{(1)} = \begin{pmatrix} 3 + signo\,(3)\sqrt{3^2+1^2+2^2} \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 6.7417 \\ 1 \\ 2 \end{pmatrix}$$

$$w^{(1)} = \begin{pmatrix} 0 \\ 6.7417 \\ 1 \\ 2 \end{pmatrix}$$

Por tanto, la primera matriz de Householder es

$$H_1 = I - 2\frac{w^{(1)}w^{(1)^t}}{w^{(1)^t}w^{(1)}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -0.80178 & -0.26726 & -0.53542 \\ 0 & -0.26726 & 0.96036 & -0.07929 \\ 0 & -0.53542 & -0.07929 & 0.84143 \end{pmatrix}$$

Aplicando el algoritmo visto arriba se obtiene finalmente la siguiente matriz reducida a Hessenberg superior:

$$A_H = \begin{pmatrix} 2 & 2.138 & -2.33 & 0 \\ -3.742 & 3.286 & 4.392 & 0.92 \\ 0 & 1.868 & -2.18 & 4.29 \\ 0 & 0 & -0.788 & 0.895 \end{pmatrix}$$

■

3.4 Método de la potencia, de la potencia inversa y deflación

3.4.1 Método de la potencia

Este método calcula el autovalor de máximo módulo de la matriz A y el autovector asociado. Considerando λ_1 como el autovalor de máximo módulo, la forma de calcularlo es aplicando la siguiente recurrencia:

$$y^{(k)} = Ay^{(k-1)} = A^{(k)}y^{(0)} \qquad k = 1, 2, \ldots \qquad \frac{y_p^{(k+1)}}{y_p^{(k)}} \to \lambda_1 \tag{3.9}$$

Para que se pueda aplicar es necesario que:

- ✏ λ_1 es tal que domina a todos los demás autovalores (estrictamente mayor):

  $$|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$$

  Por tanto, si λ_1 es complejo no nos sirve, ya que existe otro autovalor con el mismo módulo, su conjugado

- ✏ La matriz A sea diagonalizable. Una matriz A es diagonalizable si y solo si dado un conjunto de vectores $\left\{v^{(1)}, v^{(2)}, \ldots, v^{(n)}\right\}$, estos son base de $\mathbb{R}^n$

Se construye por tanto la sucesión:

$$y^{(0)} = \sum_{i=1}^{n} \alpha_i v^{(i)}$$

$$y^{(1)} = \sum_{i=1}^{n} \alpha_i A v^{(i)} = \sum_{i=1}^{n} \alpha_i \lambda_i v^{(i)}$$

$$y^{(2)} = \sum_{i=1}^{n} \lambda_i \alpha_i A v^{(i)} = \sum_{i=1}^{n} \lambda_i^2 \alpha_i v^{(i)}$$

$$\vdots$$

$$y^{(m)} = \sum_{i=1}^{n} \lambda_i^m \alpha_i v^{(i)}$$

$$\begin{aligned} y^{(m)} &= \lambda_1^m \alpha_1 v^{(1)} + \sum_{i=2}^{n} \lambda_i^m \alpha_i v^{(i)} \\ &= \lambda_1^m \left(\alpha_1 v^{(1)} + \sum_{i=2}^{n} \left(\frac{\lambda_i}{\lambda_1} \right)^m \alpha_i v^{(i)} \right) \end{aligned}$$

Al hacer el límite,

$$\lim_{m\to\infty} y^{(m)} = C v^{(1)} \qquad C = \lambda_1^m \alpha_1$$

Y ahora se construye otra sucesión

$$\mu_{m+1} = \frac{y_k^{(m+1)}}{y_k^{(m)}} = \frac{\lambda_1^{m+1} \left(\alpha_1 v_k^{(1)} + \sum_{i=2}^{n} \left(\frac{\lambda_i}{\lambda_1} \right)^{m+1} \alpha_i v_k^{(i)} \right)}{\lambda_1^m \left(\alpha_1 v_k^{(1)} + \sum_{i=2}^{n} \left(\frac{\lambda_i}{\lambda_1} \right)^m \alpha_i v_k^{(i)} \right)}$$

donde v_k es la k-ésima componente del vector v y y_k la del vector y.

Para esta sucesión, si $\alpha_1 \neq 0$ y $v_k^{(1)} \neq 0$,

$$\lim_{m\to\infty} \mu_{m+1} = \lambda_1 \tag{3.10}$$

Si $\lambda_1 > 1$, entonces $C \to \infty$, mientras que si $\lambda_1 < 1$, $C \to 0$, por lo que suele ser recomendable normalizar la sucesión, por ejemplo con la norma infinito:

$$p/ \left| y_p^{(k)} \right| = \left\| y^{(k)} \right\|_\infty \to z^{(k)} = \frac{y^{(k)}}{y_p^{(k)}} \qquad \text{entonces } z_p^{(k)} = \pm 1$$

Ahora la recurrencia nos queda

$$z^{(k)} = \frac{y^{(k)}}{y_p^{(k)}} \qquad y^{(k+1)} = A z^{(k)} \qquad \mu_k = y_p^{(k+1)} \tag{3.11}$$

Usando el cociente de Rayleigh

$$\begin{aligned}\sigma_{k+1} &= \frac{y^{(k)^t}Ay^{(k)}}{y^{(k)^t}y^{(k)}} = \frac{\left\|y^{(k)}\right\|_2}{\left\|y^{(k)}\right\|_2}\frac{y^{(k)^t}Ay^{(k)}}{y^{(k)^t}y^{(k)}} \\ &= \frac{z^{(k)^t}Az^{(k)}}{z^{(k)^t}z^{(k)}} \\ &= \frac{z^{(k)^t}y^{(k+1)}}{\left\|z^{(k)}\right\|_2^2}\end{aligned}$$

Como $\left\|z^{(k)}\right\|_2^2 = 1$, entonces nos queda que el autovalor σ_{k+1} con autovector asociado $y^{(k)}$ es

$$\boxed{\sigma_{k+1} = z^{(k)^t}y^{(k+1)}} \tag{3.12}$$

Vimos que de la forma anterior, el autovalor es

$$\mu_m = \lambda_1 + o\left(\left|\frac{\lambda_2}{\lambda_1}\right|^m\right)$$

Mientras que, usando el coeficiente de Rayleigh tenemos

$$\sigma_m = \lambda_1 + o\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2m}\right)$$

Por tanto, usando el coeficiente de Rayleigh tiende más rápido al autovalor de mayor módulo.

El problema pues, es escoger un $y^{(0)}$ tal que $\alpha_1 \neq 0$. Aunque se escoja un $y^{(0)}$ para el que no se cumpla, los errores de redondeo pueden ocasionar a la larga que $\alpha_1 \neq 0$. Es prudente elegir varios $y^{(0)}$. En cualquier caso, el método funciona porque hay errores de redondeo, ya que puede ser matemáticamente $\alpha = 0$ y/o $v_i^{(j)} = 0$, pero ser 10^{-8} o 10^{-12} numéricamente, con lo que se puede proceder.

Por otro lado, si no se cumple que λ_1 es estrictamente dominante (que sea múltiple) o λ_1 es complejo (por lo que existe otro de módulo máximo, su conjugado), hay adaptaciones que no vamos a considerar.

Ahora la pregunta del millón: ¿va rápido o lento este método?

Pues eso depende de lo rápido que $\left|\frac{\lambda_i}{\lambda_1}\right|$ tienda a cero, es decir, que cuanto más dominante sea λ_1 más rápido será el método.

Hay una **variante** del método que consiste en desplazar el autovalor un cierto valor q, de forma que calculamos el autovalor $\lambda_1 + q$, aplicando el método de las potencias a $A - qI$.

Con ello, el cociente $\left|\frac{\lambda_i+q}{\lambda_1-q}\right|$ tiende a 1, siendo más rápido.

A continuación se muestra la implementación en Matlab del método de la potencia.

Algoritmo 3.4: Método de la potencia para calcular el autovalor de máximo módulo

```
% Algoritmo para el metodo de la potencia
function [vl,vc]=potencia(a,tol);
if nargin<2
   tol=1e-5;
end;
[m n]=size(a);
```

```
p=zeros(m,1);
y=ones(m,1);
z=a*y;
A=[y];% aquí voy a mostrar la secuencia de
      % autovectores para ver si hay
      % cambio de signo
while abs(norm(z,inf)-norm(p,inf))>tol
   p=z;
   y=z/norm(z,inf);
   z=a*y;
   A=[A y];
end;
vl=norm(z,inf);
vc=y;
```

♦

3.4.2 Método de la potencia inversa

Este método resuelve el problema de hallar el autovalor de menor módulo, ya que invirtiendo los λ_i el de menor módulo se convierte en el de mayor módulo. Por tanto, las inversas de los autovalores son los autovalores de la matriz inversa. No hay que invertir la matriz A, sólo hay que resolver el sistema de ecuaciones siguiente:

$$Ay^{(k+1)} = y^{(k)}$$

Por algún método de factorización. En este caso, la recurrencia tiende a

$$\frac{y_p^{(k+1)}}{y_p^{(k)}} \rightarrow \frac{1}{\lambda_n}$$

Siendo λ_n el autovalor de la matriz A de menor módulo.

Veamos que esto funciona. Si aplicamos el método de las potencias a A^{-1}, siendo μ un autovalor de A^{-1} con autovector asociado v,

$$A^{-1}v = \mu v \Rightarrow AA^{-1}v = A\mu v \Rightarrow v = \mu Av$$

Por tanto,

$$\frac{1}{\mu}v = Av$$

así que $1/\mu$ es autovalor de A.

Si se utiliza desplazamiento, siendo μ autovalor de $(A - \lambda I)^{-1}$, entonces

$$(A - \lambda I)^{-1} v = \mu v \Rightarrow v = \mu (A - \lambda I) v$$

$$\frac{1}{\mu}v = Av - \lambda v \Rightarrow \underbrace{\left(d + \frac{1}{\mu}\right)}_{\lambda_i} v = Av$$

$$\mu = \frac{1}{\lambda_i - d}$$

Esto nos sirve para calcular el autovalor auténtico más cercano a la aproximación λ, obtenida por otro método, por lo que es un refinamiento del cálculo de autovalores.

Interesa mucho factorizar la matriz A (o $A - qI$) para resolver rápidamente el sistema varias veces.

Todo lo visto anteriormente no sirve para una matriz ortogonal, ya que sus autovalores son todos de módulo la unidad.

Incluimos a continuación el algoritmo en Matlab para implementar el método de la potencia inversa.

Algoritmo 3.5: Método de la potencia inversa para calcular el autovalor de mínimo módulo

```
% Metodo de la potencia inversa para
% el calculo de autovalores
function [s,x]=potinv(A,x,q,n)
% s= autovalor de inv(A-q*I)
% x= autovector de inv(A-q*I)
[m h]=size(A);
k=1;
p=ones(m,1);
T=[p];
x=x/norm(x,inf);
z=inv(A-q*eye(size(A)));
while(k<n)
   y=z*x;
   s=norm(y,inf)
   x=y/s
   T=[T x];
   k=k+1;
end
```

♦

3.4.3 Método de deflación

Este método lo que hace es "desinflar" el problema. Se utiliza un autovalor λ_1 y un autovector $v^{(1)}$ para transformar A y obtener otro autovalor como autovalor de mayor módulo de una matriz de orden inferior.

Hay varias técnicas, alguna de las cuales vamos a comentar.

La deflación consiste en construir la matriz B transformando A de la siguiente forma:

$$B = A - \lambda_1 v^{(1)} x^t$$

Conociendo λ_1 y $v^{(1)}$ (autovalor de mayor módulo de A y su autovector asociado).

Se quiere que $espectro\,(B) = \{0, \lambda_2, \ldots, \lambda_n\}$ siendo λ_i para $i = 2 : n$ los autovalores de la matriz A que nos faltan por calcular. Para ello,

$$Bv^{(1)} = \underbrace{Av^{(1)}}_{\lambda_1 v^{(1)}} - \lambda_1 v^{(1)} x^t v^{(1)} = 0v^{(1)} \Rightarrow x^t v^{(1)} = 1$$

Por tanto, se busca

$$\boxed{x^t v^{(1)} = 1} \tag{3.13}$$

Vamos a ver dos de las técnicas de deflación: la deflación de Hotelling y la de Wielandt. Hay otra que es la deflación por semejanza, pero no la vamos a ver.

✏ Deflación de Hotelling: se usa para matriz A simétrica

$$\boxed{x^t = \frac{v^{(1)^t}}{\left\|v^{(1)}\right\|_2}} \tag{3.14}$$

✏ Deflación de Wielandt:

$$Av^{(1)} = \lambda_1 v^{(1)}$$

$$a_i v^{(1)} = \lambda_1 v_i^{(1)} \quad \text{con } v_i^{(1)} \neq 0$$

Siendo a_i la fila i de la matriz A y $v_i^{(1)}$ la componente i del autovector asociado a λ_1.

$$\underbrace{\frac{1}{\lambda_1 v_i^{(1)}} a_i}_{x^t} v^{(1)} = 1 \Rightarrow b_i = 0 \Rightarrow \hat{B} \in \mathbb{R}^{(n-1)\times(n-1)}$$

Así que hay que quitar la fila i y columna i de la matriz B.

Ejemplo 3.7:

Dada la matriz A y $\lambda_1 = 6$ el autovalor de módulo máximo, transformar A en B mediante la deflación de Wielandt teniendo el autovector $v^{(1)}$.

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

$$v^{(1)} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

Hallamos el vector x^t:

$$x^t = \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix} \frac{1}{6}$$

Por tanto, aplicando $B = A - \lambda_1 v^{(1)} x^t$:

$$B = \begin{pmatrix} 0 & 0 & 0 \\ 3 & 2 & -1 \\ -3 & -1 & 2 \end{pmatrix}$$

Y los autovalores de la submatriz

$$\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$$

Son los restantes autovalores de A.

■

3.5 Método de Jacobi

Sólo se puede aplicar para matrices A simétricas. Se transforma en diagonal por semejanza mediante rotaciones. Se aprovecha que las matrices simétricas son diagonalizables por semejanza ortogonal.

$$A_{k+1} = G_{p_k q_k}^t A_k G_{p_k q_k} \qquad k = 1, 2, 3, \ldots$$

Donde p_k y q_k dependen del paso en que nos encontremos. Sin considerar el subíndice k, la submatriz 2×2 clave de la transformación que tenemos que usar es:

$$\begin{pmatrix} b_{pp} & 0 \\ 0 & b_{qq} \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

Eligiendo s y c de forma que $|s/c| \leq 1$, por estabilidad numérica. No son los mismos que en el caso del método de Givens para sistemas lineales. Se escogen de forma que $b_{pq} = 0$.

Es decir, que se resuelve la ecuación $b_{pq} = 0$, dividiendo por $c^2 a_{pq}$ y nos quedamos con la raiz de menor módulo de la cuadrática

$$-t^2 + \frac{a_{pp} - a_{qq}}{a_{pq}} t + 1 = 0$$

$$\boxed{c = \frac{signo\,(x_p)}{\sqrt{1+t^2}} \qquad s = -\,|ct|\; signo\,(x_q)} \tag{3.15}$$

Las filas p y q no tienen porqué estar juntas, sólo lo marcamos así para ver cómo escoger s y c.

Se lleva el control de la suma de los cuadrados de los elementos no diagonales:

$$off\,(A) = \sqrt{\|A\|_F^2 - \sum_{i=1}^{n} a_{ii}^2}$$

Como

$$off\,(A_{k+1})^2 = off\,(A_k)^2 - 2\left(a_{p_k q_k}^{(k)}\right)^2$$

entonces $off\,(A_k)$ tiende a cero, y por tanto, A_k tiende a ser diagonal.

Después de hacer un cero, no se asegura que al hacer otro cero el anterior no deje de ser cero, por eso es iterativo, ya que hay que repetirlo.

Para elegir p_k y q_k:

1. **Criterio clásico**: p_k y q_k tales que $\left|a_{p_k q_k}^{(k)}\right| = \text{máx}_{i \neq j} \left|a_{ij}^{(k)}\right|$
2. **Jacobi cíclico**: (p_k, q_k) siguen un orden prefijado
3. **Jacobi cíclico con filtro**: se salta en el cíclico los pares (p_k, q_k) para los que $a_{p_k q_k}^{(k)}$ no supere un cierto umbral

3.6 Método QR

Cuando la matriz no es simétrica no se puede aplicar el método de Jacobi, por lo que se usa este método.

Con este método se consigue una matriz triangular o cuasitriangular realizando factorizaciones QR y multiplicando con el orden cambiado Q y R. Se construye la siguiente sucesión matricial:

$$A_1 = A \rightarrow \text{factorizacion QR}: \quad Q_1R_1 = A_1 \rightarrow A_2 = R_1Q_1$$
$$A_k = Q_kR_k \rightarrow Q_k^tA_k = R_k, \qquad A_{k+1} = R_kQ_k$$

Como $A_{k+1} = R_kQ_k = Q_k^tA_kQ_k$, entonces A_{k+1} es semejante a A_k y tiene sus mismos autovalores, y además es ortogonal.

Aquí se explota el teorema de Schur, que decía que

$$U^tAU = T \quad \text{(triangular)}$$

El límite de la sucesión es una matriz triangular, aunque no siempre se consigue la convergencia a una matriz triangular.

¿Qué matrices llegan a triangulares y cuáles a cuasitriangulares? La respuesta la tiene el siguiente teorema:

Teorema 3.2. El método QR sobre una matriz Hessenberg H irreducible de forma sucesiva, acaba siendo triangular si y sólo si no existen dos autovalores distintos con el mismo módulo y multiplicidades de la misma paridad.

Es decir, que se llegará a una matriz triangular partiendo de una Hessenberg cuando no haya dos autovalores del mismo módulo, distintos y con multiplicidades algebraicas de la misma paridad.

Por ejemplo, si una matriz Hessenberg tiene como autovalores 3 y -3, no acabará siendo triangular.

Se dice que una matriz H es irreducible si $h_{i+1,i} \neq 0$ para $i = 1 : n-1$.

Los autovalores deben estar "en la misma caja", es decir, que si al ir reduciendo por QR una matriz se obtiene una cuasitriangular que va tendiendo a triangular, esto se aplica a las partes de la matriz diferentes que se "resisten" a ser triangulares.

- ✏ En la práctica, se reduce la matriz a Hessenberg o tridiagonal (si es simétrica) haciendo $A_1 = Q_0^tAQ_0$, para anular los elementos de la primera subdiagonal por Givens, lo cual sólo requiere $n-1$ rotaciones, lo que representa un coste muy pequeño.
- ✏ Si $A_k = G_kR_k$, Q_k es Hessenberg superior.
- ✏ $A_{k+1} = R_kQ_k$ es Hessenberg superior o tridiagonal simétrica.
- ✏ También se aplican desplazamientos para acelerar el proceso:
  Hallar un μ_k para que:

$$H_k - \mu_kI_n = Q_kR_k$$
$$H_{k+1} = R_kQ_k + \mu_kI_n$$

  eligiéndose cada μ_k de forma que:

1. $\mu_k = h_{nn}^{(k)}$ mientras $h_{n,n-1}^{(k)} \neq 0$. Si es cero, se pasa a la fila anterior empezando por abajo.

2. El autovalor de $\begin{pmatrix} h_{n-1,n-1}^{(k)} & h_{n-1,n}^{(k)} \\ h_{n,n-1}^{(k)} & h_{nn}^{(k)} \end{pmatrix}$ es más próximo a $h_{nn}^{(k)}$.

Con esta aceleración se puede llegar en unas 10 iteraciones, mientras que de la otra forma son necesarias alrededor de 100.

TEMA 4

Interpolación y aproximación

4.1 Introducción

Se tiene información de una función f para determinados valores de las variables independiente y dependiente. Se trata de determinar un elemento p de una clase de funciones a elegir tal que p **interpole** o **aproxime** a f en los puntos en los que se dispone de la información.

Por tanto, p es la que mejor aproxima f siguiendo algún criterio.

Esto nos puede servir para varias cosas:

1. Averiguar cuánto vale $f(\alpha)$ si es difícil de evaluar o bien si sólo conocemos algunos valores en ciertos puntos en lugar de su expresión analítica. Vale $p(\alpha)$, considerando el error de interpolación o aproximación.
2. Hallar el valor de $f'(\alpha)$. Éste es $p'(\alpha)$.
3. Calcular $\int_a^b f(x)\,dx \rightarrow \int_a^b p(x)\,dx$.
4. Hallar un valor de α tal que $f(\alpha) = y$ dado el valor de y. Esto también se puede hacer mediante **interpolación inversa**, que consiste en tomar como nodos los valores de la función (sólo si son todos distintos) y como valores de la función los puntos x_i.

4.2 Interpolación: introducción

En la interpolación buscamos una función $\varphi(x)$ fácil de evaluar y tal que

$$f(x_i) = \varphi(x_i)$$

Para algunos x_i dados, que llamaremos **nube de puntos**, conocidos los valores de la función en esos puntos.

Con esta función $\varphi(x)$ podremos aproximar

$$f(\alpha) \simeq \varphi(\alpha)$$

Y dado que no es una igualdad, habrá un **error de interpolación**

$$E(\alpha) = f(\alpha) - \varphi(\alpha)$$

- En la interpolación se buscan los valores de la función para puntos α que pertenecen al intervalo $[x_0, x_n]$, de los puntos que se tienen de la función
- En la extrapolación se buscan los valores de la función para puntos α que están fuera de dicho intervalo
- En la interpolación inversa, se tiene el polinomio de interpolación $\varphi(x)$ y un valor de la función conocido, ρ, y se quiere encontrar el punto α en el que la función tiene ese valor: α tal que $\varphi(\alpha) = \rho$. Hay varias opciones:
  - Si existe la inversa de φ, entonces $\alpha = \varphi^{-1}(\rho)$
    Otra posibilidad es resolver la ecuación no lineal
    $$\varphi(\alpha) - \rho = 0$$
  - Si los valores de la función a interpolar, $f(x_i)$, son todos distintos, podemos considerar como nodos dichos $f(x_i)$ y como valores de la función los puntos x_i, de forma que el problema es ahora interpolar en los nodos y_i los valores $f^{-1}(y_i)$, y el valor de α buscado será
    $$\alpha \simeq \varphi^{-1}(\rho)$$

4.3 Interpolación polinomial

4.3.1 Introducción

Suponemos que tenemos los valores $\{x_i, y_i = f(x_i)\}$ $i = 0 : n$ con x_i distintos entre sí. Queremos determinar el polinomio $p \in P_n$ de grado máximo n tal que $p(x_i) = f(x_i)$ para $i = 0 : n$. Si no hay igualdad, no podemos llamarlo interpolación.

En primer lugar, veamos que la solución es única:

Suponemos que hay dos: $p_1 \in P_n$ y $p_2 \in P_n$.

Entonces $p_1(x_i) = f(x_i) \ \ \forall i$ y $p_2(x_i) = f(x_i) \ \forall i$.

Se cumple además que $p_1 - p_2 \in P_n$.

$p_1(x_i) - p_2(x_i) = 0$ constituye un polinomio de grado n con $n+1$ soluciones (ya que tenemos $n+1$ valores), por tanto:

$$p_1(x) = p_2(x)$$

Y la solución es única.

4.3.2 Interpolación lineal

En este caso, lo que se tiene para la función $\varphi(x)$ es una combinación lineal de una serie de funciones $\varphi_i(x)$ que llamaremos **funciones base**,

$$\varphi(x) = \sum_{i=0}^{n} a_i \varphi_i(x)$$

Por ejemplo, para $\varphi(x) \in P_n(\mathbb{R}^n)$ se puede tomar la base canónica

$$\varphi_i(x) \in \left\{x^0, x^1, x^2, \ldots, x^n\right\} \qquad i = 0 : n$$

Para $n = 2$ se tiene

$$\varphi(x) = a_0\varphi_0(x) + a_1\varphi_1(x) + a_2\varphi_2(x)$$

Se conocen los $\varphi_i(x)$ y hay que calcular los coeficientes a_i, imponiendo las condiciones de interpolación:

$$\begin{aligned}
\varphi(x_0) = &\quad a_0\varphi_0(x_0) + a_1\varphi_1(x_0) + a_2\varphi_2(x_0) = \quad f(x_0) \\
\varphi(x_1) = &\quad a_0\varphi_0(x_1) + a_1\varphi_1(x_1) + a_2\varphi_2(x_1) = \quad f(x_1) \\
\varphi(x_2) = &\quad a_0\varphi_0(x_2) + a_1\varphi_1(x_2) + a_2\varphi_2(x_2) = \quad f(x_2)
\end{aligned}$$

Que se pueden escribir matricialmente como un sistema de ecuaciones lineales

$$\underbrace{\begin{pmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \varphi_2(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \varphi_2(x_1) \\ \varphi_0(x_2) & \varphi_1(x_2) & \varphi_2(x_2) \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}}_{\overline{a}} = \underbrace{\begin{pmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \end{pmatrix}}_{\overline{y}}$$

Para la base canónica tendríamos una matriz de Wandermonde

$$A = \begin{pmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{pmatrix}$$

Esta matriz cumple

$$det(A) \neq 0 \qquad \text{si } x_0 \neq x_1 \neq x_2$$
$$det(A) = (x_0 - x_1)(x_0 - x_2)(x_1 - x_2)$$

- ✏ El problema de interpolación tiene solución única, que se puede expresar en distintas bases: canónica, de Lagrange, de Newton, ...
- ✏ La base canónica, sin embargo, tiene un inconveniente: si tenemos el polinomio para x_1, x_2 y x_3 y nos llega una nueva información para el punto x_4, lo que se ha calculado anteriormente no sirve para nada, por lo que no es adaptable a la llegada de nueva información. Veremos que la base de Lagrange tampoco lo es.

4.3.3 Método de interpolación de Lagrange

Usamos una base $\{l_0(x), l_1(x), \ldots, l_n(x)\}$ para el espacio P_n, de forma que:

$$p(x) = \sum_{k=0}^{n} y_k l_k(x) \tag{4.1}$$

Esta es la **fórmula de Lagrange**, y se tiene, para cada valor que tenemos:

$$p(x_i) = \sum_{k=0}^{n} y_k l_k(x_i) = y_i$$

Es decir, que se tiene que determinar la base para la cual

$$a_i = f(x_i) = y_i$$

La forma fácil de hacer esto es que sólo quede un sumando para cada x_i:

$$y_i l_i(x_i) = y_i$$

Para $n = 2$ tenemos

$$p_2(x) = y_0 l_0(x) + y_1 l_1(x) + y_2 l_2(x)$$

Y las condiciones de interpolación son, para este caso, las mismas

$$\begin{aligned} p_2(x_0) = &\quad y_0 l_0(x_0) + y_1 l_1(x_0) + y_2 l_2(x_0) = \quad y_0 \\ p_2(x_1) = &\quad y_0 l_0(x_1) + y_1 l_1(x_1) + y_2 l_2(x_1) = \quad y_1 \\ p_2(x_2) = &\quad y_0 l_0(x_2) + y_1 l_1(x_2) + y_2 l_2(x_2) = \quad y_2 \end{aligned}$$

De la primera, debe salir

$$l_0(x_0) = 1 \qquad l_1(x_0) = l_2(x_0) = 0$$

En general será

$$\boxed{l_i(x_i) = 1 \qquad l_i(x_k) = 0 \qquad \forall i \neq k} \tag{4.2}$$

Se tendrá para cada elemento de la base

$$l_i(x) = c_i (x - x_0)(x - x_1) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)$$

$$l_i(x_i) = 1 = c_i (x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)$$

Por tanto la constante c_i para que sea $l_i(x_i) = 1$ debe ser

$$\boxed{c_i = \frac{1}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)}} \tag{4.3}$$

Así, se escriben los $l_i(x)$ de la siguiente forma:

$$\boxed{l_i(x) = \frac{(x - x_0) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)}} \tag{4.4}$$

Y si escribimos $\theta_{n+1}(x) = (x - x_0) \cdots (x - x_n)$, nos queda:

$$l_i(x) = \frac{\theta_{n+1}(x)}{(x - x_i)\,\theta'_{n+1}(x_i)}$$

Ejemplo 4.1:

Si se tiene la siguiente tabla de valores:

x_i	y_i
-1	4
0	2
4	3
1	-2

El polinomio de Lagrange nos queda:

$$\begin{aligned} p(x) &= 4\frac{x(x-4)(x-1)}{(-1-0)(-1-4)(-1-1)} \\ &+ 2\frac{(x+1)(x-4)(x-1)}{(0+1)(0+4)(0-1)} \\ &+ 3\frac{(x+1)(x-0)(x-1)}{(4+1)(4-0)(4-1)} \\ &- 2\frac{(x+1)(x-0)(x-4)}{(1+1)(1-0)(1-4)} \end{aligned}$$

■

Ejemplo 4.2:

Dados los siguientes valores:

x_i	y_i
0	1
1	2
2	5
3	1

Calculamos el polinomio de grado 3 de Lagrange

$$p_3(x) = 1 \cdot l_0(x) + 2 \cdot l_1(x) + 5 \cdot l_2(x) + 1 \cdot l_3(x)$$

$$\begin{aligned} l_0(x) &= \frac{(x-1)(x-2)(x-3)}{(0-1)(0-2)(0-3)} \\ l_1(x) &= \frac{(x-0)(x-2)(x-3)}{(1-0)(1-2)(1-3)} \\ l_2(x) &= \frac{(x-0)(x-1)(x-3)}{(2-0)(2-1)(2-3)} \\ l_3(x) &= \frac{(x-0)(x-1)(x-2)}{(3-0)(3-1)(3-2)} \end{aligned}$$

El coeficiente de x^n será

$$\sum_{i=0}^{n} \frac{f(x_i)}{\prod_{\substack{k=0 \\ k \neq i}}^{n} (x_i - x_k)}$$

■

El inconveniente de este método es que el polinomio cuesta más evaluarlo que con el algoritmo de Horner (Ruffini):

$$p(x) = a_0 x^n + a_1 x^{n-1} + \cdots a_n$$

$$y = a_0$$

$$y = yx + a_i \quad i = 1 : n$$

Además, si se añade otro dato, hay que volver a escribir el polinomio entero desde el principio, por lo que no es adaptable.

A continuación vemos el código en Matlab para obtener el polinomio interpolante de Lagrange.

Algoritmo 4.1: Interpolación polinómica de Lagrange

```
% Esta funcion realiza la interpolacion
% polinomica a traves del uso
% de la formula de Lagrange
function [Px] = lagrange (x,fx)
long=length(x);
Px=0;
for i=1:long
   pol=1;
   for k=1:long
      if k~=i
         pol=conv(pol,[1 -x(k)]);
      end
   end
   Px=Px + fx(i)*(pol./polyval(pol,x(i)));
  % -> Px=Px + fx(k)*Lx(k) (Sumatoria)
end
```

♦

4.3.4 Polinomio de interpolación de Newton

4.3.4.1 Definición

Se pretende con este método que al añadir un punto, sólo se añada un sumando al polinomio, sin alterar el resto.

Es decir, si tenemos el polinomio $p_{k-1}(x)$ interpolante para $\{x_0, x_1, \ldots, x_{k-1}\}$ y queremos el polinomio $p_k(x)$ que sea interpolante para $\{x_0, x_1, \ldots, x_k\}$, pretendemos que podamos obtenerlo como

$$p_k(x) = p_{k-1}(x) + Q_k(x)$$

Y también que $p_k(x)$ sea fácilmente evaluable.

Para ello, debe cumplirse:

1. $p_k(x_i) = f(x_i)$ para $i = 0 : k-1$, para que sea interpolante. Para ello,

$$f(x_i) = p_k(x_i) = p_{k-1}(x_i) + Q_k(x_i)$$

como $p_{k-1}(x)$ interpolaba para $i = 0 : k-1$, debe ser

$$Q_k(x_i) = 0 \qquad i = 0 : k-1$$

$$Q_k(x) = a_k \theta_k(x)$$

2. $p_k(x_k) = f(x_k)$ para que interpole en el nuevo punto, y por tanto

$$f(x_k) = p_k(x_k) = p_{k-1}(x_k) + a_k \theta_k(x_k)$$

Es decir

$$\boxed{a_k = \frac{f(x_k) - p_{k-1}(x_k)}{\theta_k(x_k)}} \tag{4.5}$$

A a_k lo llamamos **diferencia dividida de orden** k, y se nota como

$$f[x_0, x_1, \ldots, x_k]$$

Así pues, para los polinomios sucesivos tendremos

$$p_0(x) = f[x_0]$$

$$p_1(x) = f[x_0] + f[x_0, x_1] \overbrace{(x - x_0)}^{\theta_1(x)}$$

$$p_2(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2] \overbrace{(x - x_0)(x - x_1)}^{\theta_2(x)}$$

$$\vdots$$

Definiendo $\theta_0(x) = 1$, tenemos el polinomio de interpolación en la base $\{\theta_0(x), \theta_1(x), \ldots, \theta_n(x)\}$, que llamamos base de Newton.

La ventaja principal es que se aprovecha la información conforme va llegando, por lo que **es adaptable**.

El coeficiente de x^n es

$$f[x_0, x_1, \ldots, x_n] = \sum_{i=0}^{n} \frac{f(x_i)}{\prod\limits_{\substack{k=0 \\ k \neq i}}^{n} (x_i - x_k)}$$

Es el mismo que en el polinomio de Lagrange, ya que hemos dicho que el polinomio de interpolación es único.

La derivada n-ésima de $p_n(x)$ es

$$p_n^{n)}(x) = f[x_0, x_1, \ldots, x_n]\, n!$$

En cuanto a la **evaluación**, dado un α en el que queremos conocer el valor de la función, se usa el esquema de Horner generalizado, que consiste en sacar factor común con un esquema anidado, por ejemplo

$$p_2(x) = f[x_0] + (\alpha - x_0)(f[x_0, x_1] + (\alpha - x_1) f[x_0, x_1, x_2])$$

Y en un ordenador operaremos desde dentro hacia afuera.

4.3.4.2 Desarrollo para obtener el polinomio

Vamos a ver el desarrollo paso a paso, empezando con un punto:

Se tiene (x_0, y_0):

$$p_0(x) = y_0$$

Si se añade un punto (x_1, y_1):

$$p_1(x) = p_0(x) + a_1(x - x_0) \rightarrow \begin{cases} p_1(x_0) = p_0(x_0) = y_0 \\ p_1(x_1) = y_1 = y_0 + a_1(x_1 - x_0) \end{cases}$$

$$a_1 = \frac{y_1 - y_0}{x_1 - x_0} = f[x_0, x_1]$$

Donde a $f[x_0, x_1]$ se le llama **diferencia dividida**.

Añadimos un tercer punto (x_2, y_2):

$$p_2(x) = p_1(x) + a_2(x - x_0)(x - x_1) \rightarrow \begin{cases} p_2(x_i) = p_1(x_i) \qquad i = 0:1 \\ y_2 = p_2(x_2) = p_1(x_2) + a_2(x_2 - x_0)(x_2 - x_1) \end{cases}$$

$$a_2 = \frac{y_2 - p_1(x_2)}{(x_2 - x_0)(x_2 - x_1)} = \frac{y_2 - (y_0 + f[x_0, x_1](x_2 - x_0))}{(x_2 - x_0)(x_2 - x_1)} = \frac{f[x_0, x_2] - f[x_0, x_1]}{x_2 - x_1}$$

Por inducción, llegamos a:

$$f[x_{i_0}, x_{i_1}, \ldots x_{i_k}] = \frac{\overbrace{f[x_{i_1}, \ldots, x_{i_k}]}^{(1)} - \overbrace{f[x_{i_0}, x_{i_1}, \ldots, x_{i_{k-1}}]}^{(2)}}{x_{i_k} - x_{i_0}} \tag{4.6}$$

El que está en (1) y no está en (2), tiene que estar en el denominador, lo mismo que el que está en (2) y no en (1).

Por tanto, nos queda el polinomio de interpolación siguiente:

$$p_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + \ldots + f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\cdots(x - x_{n-1}) \tag{4.7}$$

En la tabla (4.1) se puede ver un ejemplo de diferencias divididas sucesivas con 6 puntos. En esa tabla faltarían las diferencias con 5 y 6 componentes, $f[x_{i-4}, x_{i-3}, x_{i-2}, x_{i-1}, x_i]$ y $f[x_{i-5}, x_{i-4}, x_{i-3}, x_{i-2}, x_{i-1}, x_i]$.

Ejemplo 4.3:

Teniendo la siguiente tabla de valores, hallar el polinomio de interpolación de Newton:

x_i	y_i
-1	4
0	2
4	3
1	-2

x	$f(x)$	$f[x_{i-1}, x_i]$	$f[x_{i-2}, x_{i-1}, x_i]$	$f[x_{i-3}, x_{i-2}, x_{i-1}, x_i]$
x_0	$f[x_0]$			
x_1	$f[x_1]$	$\frac{f[x_1]-f[x_0]}{x_1-x_0}$		
x_2	$f[x_2]$	$\frac{f[x_2]-f[x_1]}{x_2-x_1}$	$\frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$	
x_3	$f[x_3]$	$\frac{f[x_3]-f[x_2]}{x_3-x_2}$	$\frac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1}$	$\frac{f[x_1,x_2,x_3]-f[x_0,x_1,x_2]}{x_3-x_0}$
x_4	$f[x_4]$	$\frac{f[x_4]-f[x_3]}{x_4-x_3}$	$\frac{f[x_3,x_4]-f[x_2,x_3]}{x_4-x_2}$	$\frac{f[x_2,x_3,x_4]-f[x_1,x_2,x_3]}{x_4-x_1}$
x_5	$f[x_5]$	$\frac{f[x_5]-f[x_4]}{x_5-x_4}$	$\frac{f[x_4,x_5]-f[x_3,x_4]}{x_5-x_3}$	$\frac{f[x_3,x_4,x_5]-f[x_2,x_3,x_4]}{x_5-x_2}$

Tabla 4.1: Diferencias divididas para 6 puntos

Hallamos las diferencias divididas sucesivas:

$$a_1 = \tfrac{2-4}{0-(-1)} = f[x_0, x_1] = -2$$

$$a_2 = \tfrac{3-4}{4-(-1)} = f[x_0, x_2] = -\tfrac{1}{5} \rightarrow \quad \tfrac{-\frac{1}{5}-(-2)}{4-0} = f[x_0, x_1, x_2] = \tfrac{9}{20}$$

$$a_3 = \tfrac{-2-4}{1+1} = f[x_0, x_3] = -3 \rightarrow \quad \tfrac{-3-(-2)}{1-0} = f[x_0, x_1, x_3] = -1 \rightarrow \quad \tfrac{-1-\frac{9}{20}}{1-4} = \tfrac{29}{60}$$

Tenemos, por tanto, el siguiente polinomio de interpolación:

$$p(x) = 4 - 2(x+1) + \frac{9}{20}(x+1)(x-0) + \frac{29}{60}(x+1)x(x-4)$$

■

4.3.4.3 Esquemas para calcular las diferencias divididas

Hay dos formas de calcular las diferencias divididas:

Neville

En este caso, si queremos calcular la diferencia dividida

$$f\left[x_1, x_2, x_3\right]$$

Tomamos $x_j = x_1$ y $x_i = x_3$ (x_j el primero y x_i el último). Nos quedará

$$f\left[x_1, x_2, x_3\right] = \frac{f\left[x_2, x_3\right] - f\left[x_1, x_2\right]}{x_3 - x_1}$$

Se usan diferentes conjuntos de puntos consecutivos, según un orden.

Vemos el esquema de cómo quedarían las diferencias divididas sucesivas:

x_0	$f\left[x_0\right]$			
x_1	$f\left[x_1\right]$	$f\left[x_0, x_1\right]$		
x_2	$f\left[x_2\right]$	$f\left[x_1, x_2\right]$	$f\left[x_0, x_1, x_2\right]$	
x_3	$f\left[x_3\right]$	$f\left[x_2, x_3\right]$	$f\left[x_1, x_2, x_3\right]$	$f\left[x_0, x_1, x_2, x_3\right]$

Aitken

Ahora, para calcular

$$f\left[x_0, x_1, x_3\right]$$

Se toma $x_j = x_1$ y $x_i = x_3$ (x_i el último y x_j el penúltimo). Ahora se tiene

$$f\left[x_0, x_1, x_3\right] = \frac{f\left[x_0, x_3\right] - f\left[x_0, x_1\right]}{x_3 - x_1}$$

Aquí se usa para calcularlas el mismo conjunto de puntos junto a otro punto que los diferencia.

En este caso, las diferencias divididas son

x_0	$f\left[x_0\right]$			
x_1	$f\left[x_1\right]$	$f\left[x_0, x_1\right]$		
x_2	$f\left[x_2\right]$	$f\left[x_0, x_2\right]$	$f\left[x_0, x_1, x_2\right]$	
x_3	$f\left[x_3\right]$	$f\left[x_0, x_3\right]$	$f\left[x_0, x_1, x_3\right]$	$f\left[x_0, x_1, x_2, x_3\right]$

La única diferencia que hay entre ambos es al implementarlo, ya que en Neville no se podría sobreescribir $f\left[x_1, x_2\right]$ con $f\left[x_0, x_1, x_2\right]$ porque $f\left[x_1, x_2, x_3\right]$ necesita el valor de $f\left[x_1, x_2\right]$, mientras que en el esquema de Aitken sí que se podría sobreescribir, ya que al calcular $f\left[x_0, x_2, x_3\right]$ se usa $f\left[x_0, x_1\right]$ y $f\left[x_0, x_3\right]$, por lo que se puede ahorrar en memoria sobreescribiendo valores.

A continuación vemos el código en Matlab para implementar el método de Newton para calcular el polinomio de interpolación.

Algoritmo 4.2: Interpolación de Newton

```
% Algoritmo de interpolacion de Newton
% en diferencias divididas
% function [Px]=inewton(x,fx)
%
%  x ---> puntos donde se conoce la funcion
% fx ---> valores de la funcion en dichos puntos
% Px ---> polinomio de interpolacion resultante
function [Px]=inewton(x,fx)
long=length(x);% Grado maximo del polinomio
               % de interpolacion
dif=zeros(long);% Matriz de diferencias divididas
dif(:,1)=fx';
cont=2;
for col=2:long
   for row=cont:long
      inicf=dif(cont-1,col-1);
      inicx=x(col-1);
      dif(row,col)=(dif(row,col-1)-inicf)/(x(row)-inicx);
   end
   cont=cont+1;
end
dif
coef=diag(dif)';% la diagonal principal de la matriz son los
                % coeficientes que hay que multiplicar.
Px=0;
for i=0:long-1
   pol=1;
   for k=1:i
      pol=conv(pol,[1 -x(k)]);
   end
   if i<0
      Px=polisum(Px,coef(1)*pol);
   else
      Px=polisum(Px,coef(i+1)*pol);
   end
end
```

Donde se ha definido la función polisum como

```
% Suma de polinomios
%
% [suma]=polisum(a,b);
function[suma]=polisum(a,b)
na=length(a);
nb=length(b);
suma=[zeros(1,nb-na) a]+[zeros(1,na-nb) b];
```

♦

4.3.4.4 Fórmulas en diferencias finitas

Esto es sólo como mención. Se usa para puntos equiespaciados, es decir, para $x_i = x_o + ih$

Se definen las siguientes **diferencias**:

1. **Progresiva**: $\Delta f(x) = f(x+h) - f(x)$; $\Delta^k = \Delta\left(\Delta^{k-1}\right)$
   Se usa cuando el valor α en el que se desea interpolar está cerca del principio de la tabla.
   Con esto, la fórmula de Newton queda

   $$f[x_0, \ldots, x_k] = \frac{\Delta^k f(x_0)}{h^k \cdot k!}$$

   Y el polinomio de Newton tiene la siguiente expresión

   $$p_n(x_0 + th) = \sum_{k=0}^{n} \binom{t}{k} \Delta^k f(x_0) \qquad t \in \mathbb{R}, t > 0$$

2. **Regresiva**: $\nabla f(x) = f(x) - f(x-h)$; $\nabla^k = \nabla\left(\nabla^{k-1}\right)$
   Se suelen usar cuando el valor α en el que se quiere interpolar está cerca del final de la tabla.
   El polinomio de Newton es

   $$p_n(x_n + th) = \sum_{k=0}^{n} \binom{t+k-1}{k} \nabla^k f(x_n) \qquad t \in \mathbb{R}, t < 0$$

   "El punto en el que estoy menos el anterior".

3. **Central**: $\delta f(x) = f(x + h/2) - f(x - h/2)$; $\delta^k = \delta\left(\delta^{k-1}\right)$

4.3.5 Error de interpolación

Supongamos que tenemos un polinomio interpolante $p_n(x)$ en $\{x_0, x_1, \ldots, x_n\}$ y llega un nodo nuevo (genérico) x. Quiero hacer

$$p_{n+1}(x) = p_n(x) + Q_{n+1}(x)$$

El valor del polinomio en el nuevo nodo x tiene que coincidir con el de la función $f(x)$. Por tanto,

$$p_{n+1}(x) = p_n(x) + f[x_0, \ldots, x_n, x]\,\theta_{n+1}(x)$$

Llamamos **error de interpolación** a la diferencia entre la función y el polinomio de orden n:

$$\boxed{E(x) = f(x) - p_n(x) = f[x_0, \ldots, x_n, x]\,\theta_{n+1}(x)} \tag{4.8}$$

Si la función f es de clase $n+1$ en $[a, b]$ (continua y con derivadas hasta el orden $n+1$ continuas) siendo

$$a = \text{mín}\{x_0, x_1, \ldots, x_n, x\} \qquad \text{y} \qquad b = \text{máx}\{x_0, x_1, \ldots, x_n, x\}$$

entonces $E_n(x) = f(x) - p_n(x)$ es de clase n en $[a, b]$, y además tiene al menos $n+1$ ceros en $[a, b]$. Así, por el teorema de Rolle:

- $E_n'(x)$ tiene al menos n ceros en $[a, b]$
- $E_n''(x)$ tiene al menos $n-1$ ceros en $[a, b]$

- ...
- $E_n^{n)}(x)$ tiene al menos 1 cero en $[a,b]$

Sea ξ tal que $E_n^{n)}(\xi)=0$, entonces

$$0=E_n^{n)}(\xi)=f^{n)}(\xi)-p_n^{n)}(\xi)=f^{n)}(\xi)-n!f[x_0,x_1,\ldots,x_n]$$

$$\Rightarrow f[x_0,x_1,\ldots,x_n]=\frac{f^{n)}(\xi)}{n!}$$

Al añadir un punto sólo hay que derivar una vez más y dividir por $n+1$:

$$f[x_0,x_1,\ldots,x_n,x]=\frac{f^{n+1)}(\xi)}{(n+1)!}$$

Por tanto, la expresión del error de interpolación queda

$$\boxed{E_n(x)=\frac{f^{n+1)}(\xi)}{(n+1)!}\theta_{n+1}(x)} \tag{4.9}$$

Siendo $\xi=\xi(x)$.

La expresión es similar a la del residuo de Taylor.

Comentarios:

- Por tanto, si evaluamos en un punto que esté cerca de alguno de los puntos de interpolación obtenemos mayor precisión por decrecer la función $\theta_{n+1}(x)$ para dichos puntos.
- No se debe extrapolar salvo en puntos próximos a los extremos.
- No usar muchos puntos de interpolación (derivadas de orden pequeño e interpolación de bajo grado).

Si se aumenta el número de puntos ¿el polinomio se parece más a la función?

No es lógico pensarlo, y en general no ocurre. Tampoco suele servir tomar puntos concretos. De hecho, cuando se aumenta el número de puntos, y con ello el grado del polinomio, se pone de manifiesto el comportamiento oscilatorio de los polinomios de grado alto.

Teorema de Weierstrass. Una función continua y definida en un intervalo $[a,b]$ se puede aproximar tanto como se quiera por un polinomio definido en ese intervalo.

4.4 Interpolación osculatoria

4.4.1 Introducción

Si las funciones que queremos interpolar tienen muchos cambios bruscos, el polinomio interpolante tendrá por lo general una evolución más suave, y el error de interpolación será muy grande. Por eso, se imponen también condiciones a los nodos en cuanto a las derivadas de la función.

Es decir, para $i=0:m$ se tienen en los nodos x_i los siguientes datos:

- ✏ los valores de la función en todos los puntos: $f(x_i)$
- ✏ las derivadas hasta el orden k_0 de la función en x_0: $f'(x_0)$, $f''(x_0)$, ..., $f^{k_0)}(x_0)$
- ✏ las derivadas hasta el orden k_1 de la función en x_1: $f'(x_1)$, $f''(x_1)$, ..., $f^{k_1)}(x_1)$
- ✏ ...
- ✏ las derivadas hasta el orden k_m de la función en x_m: $f'(x_m)$, $f''(x_m)$, ..., $f^{k_m)}(x_m)$

Por tanto, las $n+1$ condiciones que hacen falta para el polinomio de grado n serán que sea interpolante ($m+1$ condiciones, los valores de la función) y la suma de las k_i derivadas para cada punto:

$$n+1 = m+1+\sum_{i=0}^{m} k_i$$

Ejemplo 4.4:

El ejemplo más claro de interpolación osculatoria es el caso de $k_0 = n$, $m = 0$, es decir, el desarrollo en series de Taylor:

$$p_n(x) = f(x_0) + \frac{f'(x_0)}{1!}(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \ldots + \frac{f^{n)}(x_0)}{n!}(x-x_0)^n$$

■

4.4.2 Interpolación de Hermite

Los **polinomios de Hermite** se dan cuando $k_i = 1 \quad i = 0:n$, es decir, cuando se conocen la función y la primera derivada en cada punto. Este es un caso frecuente.

El grado del polinomio será

$$n = 2m+1$$

Al resolver estos polinomios se dan diferencias divididas con puntos coincidentes, ya que se conoce más de un dato por cada punto: $f\left[\underset{\ldots}{x^{k+1}}x\right]$

Para construir el polinomio se crean unos nodos ficticios duplicados:

$$z_{2i} = z_{2i+1} = x_i$$

tantas veces como indique k_i (en este caso, 1).

Ejemplo 4.5:

Si se tienen los datos $(x_i, f(x_i), f'(x_i))$ y $(x_{i+1}, f(x_{i+1}), f'(x_{i+1}))$, para obtener el polinomio de interpolación osculatoria de Hermite por diferencias divididas de Neville tenemos:

i	z_i	$f(z_i)$	$f[z_{i-1}, z_i]$	$f[z_{i-2}, z_{i-1}, z_i]$	$f[z_{i-3}, z_{i-2}, z_{i-1}, z_i]$
0	x_i	$f(x_i)$			
1	x_i	$f(x_i)$	$\frac{f'(x_i)}{1!}$		
2	x_{i+1}	$f(x_{i+1})$	$\frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}$	$\frac{f[x_i,x_{i+1}]-f'(x_i)}{x_{i+1}-x_i}$	
3	x_{i+1}	$f(x_{i+1})$	$\frac{f'(x_{i+1})}{1!}$	$\frac{f'(x_{i+1})-f[x_i,x_{i+1}]}{x_{i+1}-x_i}$	$\frac{f[z_1,z_2,z_3]-f[z_0,z_1,z_2]}{x_{i+1}-x_i}$

Donde hemos hecho:

$$f[z_0, z_1] = \frac{\text{algo}}{z_1 - z_0} = \frac{\text{algo}}{0}$$

como no se podría resolver, usamos

$$f[x_0, \ldots, x_n] = \frac{f^{n)}(\xi)}{n!} \qquad \xi \in [x_0, x_n]$$

$$f[z_0, z_1] = \frac{f'(\xi)}{1!} = \frac{f'(x_i)}{1!} \qquad x_i \leq \xi \leq x_i \Rightarrow \xi = x_i$$

2Y también para

$$f[z_2, z_3] = \frac{f'(x_{i+1})}{1!}$$

Por tanto, el polinomio de interpolación quedará de la siguiente forma:

$$\begin{aligned} p_3 &= f(x_i) + f'(x_i)(x - x_i) + \frac{f[x_i, x_{i+1}] f'(x_i)}{x_{i+1} - x_i}(x - x_i)^2 \\ &+ \frac{f[z_1, z_2, z_3] - f[z_0, z_1, z_2]}{x_{i+1} - x_i}(x - x_i)^2 (x - x_{i+1}) \end{aligned}$$

■

Si escribimos $x_{i+1} = x_i + h_i$ entonces se obtiene la expresión de **interpolación de Hermite cúbica**, que es una interpolación por segmentos, y su polinomio en cada subintervalo $[x_i, x_{i+1}]$ es

$$\begin{aligned} p_c(x) &= f(x_i) + f'(x_i)(x - x_i) + \frac{3f[x_i, x_{i+1}] - 2f'(x_i) - f'(x_{i+1})}{h_i}(x - x_i)^2 \\ &+ \frac{f'(x_i) - 2f[x_i, x_{i+1}] + f'(x_{i+1})}{h_i^2}(x - x_i)^3 \end{aligned}$$

En Matlab hay un tipo de interpolación que se llama "a trozos", y que se realiza con la función `interp1`, que tiene dos casos interesantes:

- `interp1(x,y,`α`,'linear')`: interpola mediante una función lineal (una recta) una función usando los nodos contenidos en x y los valores de la función en y. Devuelve los valores de la función interpolada para los nodos de α.
- `interp1(x,y,`α`,'cubic')`: en la versión 6.0 de Matlab interpola con una función cúbica y aproximando la derivada con diferencias finitas. En la versión 5.3 hace otro tipo de interpolación, con funciones *spline*.

4.4.2.1 Expresión de Newton para el polinomio de interpolación de Hermite

En la tabla (4.2) se pueden ver las diferencias divididas aplicables al caso de que haya puntos coincidentes.

El polinomio de interpolación sale, por tanto, de grado $2n + 1$:

z_i	$f[z_i]$	$f[z_{i-1}, z_i]$
x_0	$f(x_0)$	
x_0	$f(x_0)$	$f'(x_0)$
x_1	$f(x_1)$	$f[x_0, x_1]$
x_1	$f(x_1)$	$f'(x_1)$
$\vdots$	$\vdots$	$\vdots$
x_n	$f(x_n)$	$f[x_{n-1}, x_n]$
x_n	$f(x_n)$	$f'(x_n)$

Tabla 4.2: Diferencias divididas indefinidas

$$\begin{aligned} P_{2n+1}(x) &= f[x_0] + f[x_0, x_0](x - x_0) \\ &+ f[x_0, x_0, x_1](x - x_0)^2 \\ &+ f[x_0, x_0, x_1, x_1](x - x_0)^2 (x - x_1) \\ &+ \dots \\ &+ f[x_0, x_0, x_1, x_1, \dots, x_n, x_n](x - x_0)^2 \cdots (x - x_{n-1})^2 (x - x_n) \end{aligned}$$

A continuación se muestra el código en Matlab para implementar el método de Hermite con diferencias divididas de Newton.

Algoritmo 4.3: Método de Hermite para interpolación con diferencias divididas de Newton

```
% Metodo de interpolacion de Hermite con
% la expresion de Newton
%
% Uso: Px=hermnew(x,fx,fpx)
%
%   x ---> puntos donde se conoce la funcion
%   fx ---> valores de la funcion en dichos puntos
%   fx ---> valores de la derivada de la funcion en
%           dichos puntos
%   Px ---> polinomio de interpolacion resultante
function [Px]=hermnew(x,fx,fpx)
long=length(x);% Grado maximo del polinomio de interpolacion
dif=zeros(2*long);% Matriz de las diferencias divididas
colx=zeros(2*long,1);
% Creamos el vector con los puntos multiples
for k=1:2:2*long
   colx(k)=x((k+1)/2);
end;
for k=2:2:2*long
```

```
   colx(k)=x(k/2);
end;
% Introducimos los valores de la funcion
% por partida doble por cada punto
for k=1:2:2*long-1
   dif(k,1)=fx((k+1)/2);
end;
for k=2:2:2*long
   dif(k,1)=fx(k/2);
end;
% Montamos la columna de segundas diferencias divididas
for k=2:2*long
   inicf=dif(k-1,1);
   xin=colx(k-1);
   xfi=colx(k);
   if xin~=xfi
      dif(k,2)=(dif(k,2-1)-inicf)/(xfi-xin);
   else
      dif(k,2)=fpx((k)/2);
   end;
end;
% Iteracion
cont=3;
for col=cont:2*long
   for row=cont:2*long
      inicf=dif(row-1,col-1);
      xin=colx(row-cont+1);
      xfi=colx(row);
      dif(row,col)=(dif(row,col-1)-inicf)/(xfi-xin);
   end;
   cont=cont+1;
end;
coef=diag(dif)';% la diagonal principal de la matriz
                % son los coeficientes
                % que hay que multiplicar.
Px=0;
for i=0:2*long-1
   pol=1;
   for k=1:i
      pol=conv(pol,[1 -colx(k)]);
   end;
   if i<0
      Px=polisum(Px,coef(1)*pol);
   else
      Px=polisum(Px,coef(i+1)*pol);
   end;
end;
```

♦

4.4.2.2 Forma de Lagrange

$$P_{2n+1}(x) = \sum_{i=0}^{n} f(x_i) H_i(x) + \sum_{i=0}^{n} f'(x_i) \hat{H}_i(x) \tag{4.10}$$

Con H_i y $\hat{H}_i$ a determinar:

$$H_j(x) = \left(1 - 2(x - x_j) L'_{n,j}(x_j)\right) L^2_{n,j}(x) \tag{4.11}$$

y

$$\hat{H}_j(x) = (x - x_j) L^2_{n,j}(x) \tag{4.12}$$

Donde $L_{n,j}(x)$ denota el j-ésimo coeficiente del polinomio de Lagrange de grado n.

4.5 Interpolación recurrente

Vamos a ver el algoritmo de Neville para generar recursivamente aproximaciones polinómicas de Lagrange:

Dados S y T dos subconjuntos de $\{x_0, x_1, \ldots, x_n\}$ con $S - T = \{x_S\}$ y $T - S = \{x_T\}$, $P_{C;k}(x)$ es el polinomio de interpolación de grado k que utiliza los puntos de un conjunto $C \subseteq \{x_0, x_1, \ldots, x_n\}$ siendo $k+1$ el cardinal de C y los puntos x_i todos distintos:

$$P_{S\cup T;k}(x) = \frac{P_{S,k-1}(x)(x - x_T) - P_{T,k-1}(x)(x - x_S)}{x_S - x_T} \tag{4.13}$$

Denotándose $P_{0,\ldots n;k}$ el polinomio en los puntos $\{0, \ldots n\}$ de grado k, se pueden ver en la tabla (4.3) las aproximaciones para polinomios de Lagrange usando el método de Neville.

x_i	$f(x_i)$	1 punto	2 puntos	3 puntos
x_0	$f(x_0)$	$P_{0;0}(\alpha)$		
x_1	$f(x_1)$	$P_{1;0}(\alpha)$	$P_{0,1;1}(\alpha)$	
x_2	$f(x_2)$	$P_{2;0}(\alpha)$	$P_{1,2;1}(\alpha)$	$P_{0,1,2;2}(\alpha)$
$\vdots$				$\vdots$
x_n	$f(x_n)$	$P_{n;0}(\alpha)$	$P_{n-1,n;1}(\alpha)$	$P_{n-2,n-1,n;2}(\alpha)$

Tabla 4.3: Aproximaciones para el método de Neville

4.6 Interpolación trigonométrica

Pasa el problema de reales a complejos. Se toman puntos equiespaciados en $[0, 2\pi)$, conociendo valores de la función en esos puntos y considerando dicha función periódica de periodo 2π. Consiste entonces en expresar el polinomio de interpolación en función de senos y cosenos.

Si hay un número N impar de puntos ($N = 2M + 1$):

$$\Phi(x) = \frac{a_0}{2} + \sum_{k=1}^{M} (a_k \cos(kx) + b_k \operatorname{sen}(kx)) \tag{4.14}$$

y si es par ($N = 2M$):

$$\Phi(x) = \frac{a_0}{2} + \sum_{k=1}^{M-1} (a_k \cos(kx) + b_k \operatorname{sen}(kx)) + \frac{a_M}{2} \cos(Mx) \tag{4.15}$$

Los coeficientes a_k son la transformada **cosenoidal discreta de Fourier**, mientras que los b_k son la **transformada senoidal discreta de Fourier**.

Recordamos las fórmulas de Moivre:

$$\cos x = \frac{e^{ix} + e^{-ix}}{2}$$

$$\operatorname{sen} x = \frac{e^{ix} - e^{-ix}}{2i}$$

Ya que $e^{ix} = \cos x + i \operatorname{sen} x$ y $e^{-ix} = \cos x - i \operatorname{sen} x$.

Usando esta fórmula de Moivre, el problema se queda como:

$$\begin{aligned}
\Phi(x) &= \frac{a_0}{2} + \sum_{k=1}^{M} a_k \frac{e^{ikx} + e^{-ikx}}{2} + \sum_{k=1}^{M} b_k \frac{e^{ikx} - e^{-ikx}}{2i} \\
&= \alpha_0 + \sum_{k=1}^{M} \left(\frac{a_k - ib_k}{2} \right) e^{ikx} + \sum_{k=1}^{M} \left(\frac{a_k + ib_k}{2} \right) e^{-ikx} \\
&= \alpha_0 + \sum_{k=1}^{M} \alpha_k e^{ikx} + \sum_{k=1}^{M} \alpha_{N-k} e^{-ikx}
\end{aligned}$$

La periodicidad se refleja en la última expresión, ya que $e^{ikx} = e^{i(N-k)x}$.

Expresión compleja del polinomio trigonométrico:

$$t(x) = \alpha_0 + \alpha_1 e^{ix} + \alpha_2 e^{2ix} + \ldots + \alpha_{N-1} e^{(N-1)ix} \tag{4.16}$$

Con los α_k complejos, que se pueden calcular de la siguiente forma:

$$\boxed{\alpha_k = \frac{1}{N} \sum_{n=0}^{N-1} y_n \omega_n^{-k} = \frac{1}{N} \sum_{n=0}^{N-1} y_n e^{-\frac{2\pi i k n}{N}}} \tag{4.17}$$

Siendo $t(x_k) = y_k$ para $k = 0 : N-1$.

Tenemos por tanto un problema de interpolación compleja, que también tiene solución única. El problema es

$$A\alpha = y \longrightarrow A = \begin{pmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \ldots & \varphi_{N-1}(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \ldots & \varphi_{N-1}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_0(x_{N-1}) & \varphi_1(x_{N-1}) & \ldots & \varphi_{N-1}(x_{N-1}) \end{pmatrix}$$

Donde los $\varphi_k(x)$ son las funciones básicas, que en el problema de interpolación trigonométrica son

$$\boxed{\varphi_k(x) = \left(e^{ix}\right)^k} \tag{4.18}$$

Y los puntos x_k están equiespaciados entre 0 y 2π:

$$x_k = \frac{2\pi k}{N} \qquad k = 0 : N-1$$

Por tanto podemos expresar los elementos de la matriz A como

$$a_{kl} = \varphi_l(x_k) = e^{\frac{2\pi i l k}{N}} = \omega^{-lk} \qquad \text{definiendo } \omega = e^{-\frac{2\pi i}{N}}$$

La matriz A queda entonces una matriz de Wandermonde compleja. Para $N = 4$ será

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} \end{pmatrix} = F_N^H$$

$$F_N = \left(F_N^H\right)^H = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{1} & \omega^{2} & \omega^{3} \\ 1 & \omega^{2} & \omega^{4} & \omega^{6} \\ 1 & \omega^{3} & \omega^{6} & \omega^{9} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

Aunque no lo demostramos, se cumple

$$F_N F_N^H = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix} = 4I_N = NI_N$$

Por tanto, escribimos

$$A\alpha = y \rightarrow A^H A\alpha = A^H y \Rightarrow A^H A = F_N F_N^H = NI_N \Rightarrow NI\alpha = F_N y$$

Por tanto, se puede despejar α obteniéndose

$$\boxed{\alpha = \frac{1}{N} F_N I \triangleq \frac{1}{N} C} \tag{4.19}$$

la solución del problema de interpolación en el campo complejo. El vector C es la **transformada de Fourier discreta** de los valores y de la función:

$$\boxed{c_k = \sum_{n=0}^{N-1} y_n \omega^{kn} \qquad k = 0 : N-1} \tag{4.20}$$

El cálculo de los valores y_k en función de los c_k es la transformada inversa de Fourier discreta:

$$y_k = \frac{1}{N} \sum_{n=0}^{N-1} c_n \omega^{-kn} \qquad k = 0 : N-1$$

Pues $F_N y = C$ y $y = F_N^H z$, así que

$$F_N F_N^H z = C^* \Rightarrow NIz = C \Rightarrow z = \frac{1}{N} C$$

Para los coeficientes a_k y b_k con N impar:

$$\begin{array}{ccccc} \alpha_0 = \frac{a_0}{2} & \alpha_k = \frac{a_k - ib_k}{2} & \alpha_{N-k} = \frac{a_k + ib_k}{2} & k = 1 : M \\ a_0 = 2\alpha_0 & a_k = \alpha_k + \alpha_{N-k} & b_k = i\left(\alpha_k - \alpha_{N-k}\right) & k = 1 : M \end{array} \tag{4.21}$$

Y para N par:

$$\begin{array}{ccccccc} \alpha_0 = \frac{a_0}{2} & \alpha_k = \frac{a_k - ib_k}{2} & \alpha_{N-k} = \frac{a_k + ib_k}{2} & k = 1 : M-1 & \alpha_M = \frac{a_M}{2} \\ a_0 = 2\alpha_0 & a_k = \alpha_k + \alpha_{N-k} & b_k = i\left(\alpha_k - \alpha_{N-k}\right) & k = 1 : M-1 & a_M = 2\alpha_M \end{array} \tag{4.22}$$

Lo que se puede escribir de otra forma:

$$a_k = \alpha_k + \alpha_{N-k} = \frac{1}{N}\left(c_k + c_{N-k}\right)$$

$$b_k = i\left(\alpha_k - \alpha_{N-k}\right) = \frac{i}{N}\left(c_k - c_{N-k}\right)$$

Y para el caso par

$$\frac{a_M}{2} = \alpha_M = \frac{1}{N} c_M$$

Cuando los puntos x_i no los tenemos para el intervalo $[0, 2\pi)$,

$$t \to x_0, x_0 + h, x_0 + 2h, \ldots, x_0 + (N-1)\,h$$

Se hace un cambio de variable

$$x \to \frac{t - x_0}{h} \frac{2\pi}{N}$$

Ejemplo 4.6:

Dados los puntos

$$x = \left[\begin{array}{ccccccc} 1 & 1.25 & 1.5 & 1.75 & 2 & 2.25 & 2.5 \end{array}\right]$$

Y los valores de la función

$$y = \left[\begin{array}{ccccccc} -1 & 2 & 4 & 4 & 3 & 4 & 6 \end{array}\right]$$

Hay $N = 7$ puntos.

Para obtener el intervalo entre 0 y 2π hay que hacer el cambio

$$z = \frac{8\pi}{7}\left(x - 1\right)$$

El polinomio de interpolación tendrá la siguiente expresión:

$$\begin{aligned} \phi\left(x\right) &= a_0/2 + a_1 \cos\left(\frac{8\pi}{7}\left(x-1\right)\right) + a_2 \cos\left(\frac{16\pi}{7}\left(x-1\right)\right) + a_3 \cos\left(\frac{24\pi}{7}\left(x-1\right)\right) \\ &= b_1 \operatorname{sen}\left(\frac{8\pi}{7}\left(x-1\right)\right) + b_2 \operatorname{sen}\left(\frac{16\pi}{7}\left(x-1\right)\right) + b_3 \operatorname{sen}\left(\frac{24\pi}{7}\left(x-1\right)\right) \end{aligned}$$

Los coeficientes los obtendremos con las expresiones

$$a_k = \alpha_k + \alpha_{N-k} \qquad b_k = i\left(\alpha_k - \alpha_{N-k}\right)$$

Y los valores de α_k los podemos obtener usando la función `fft` de Matlab:

$$\alpha = \texttt{fft(y)}/7 = \begin{pmatrix} 3.14286 \\ -0.58558 + i0.38478 \\ -0.80336 + i0.66879 \\ -0.6825 + i0.10866 \\ -0.6825 + i - 0.10866 \\ -0.80336 - i0.66879 \\ -0.58558 - i0.38478 \end{pmatrix}$$

Por tanto

$$a_0 = 2 \cdot 3.14286 \quad a_1 = -1.1712 \quad a_2 = -1.6067 \quad a_3 = -1.365$$

$$b_1 = -0.76956 \quad b_2 = -1.33758 \quad b_3 = -0.21732$$

■

Podemos encontrar expresiones cerradas para calcular los coeficientes a_k y b_k, usando

$$c_k = \sum_{n=0}^{N-1} y_n \omega^{kn} \qquad y_k = \frac{1}{N}\sum_{n=0}^{N-1} c_k \omega^{-nk} \qquad \omega^{(N-l)k} = \omega^{-lk}$$

Para a_k:

$$\begin{aligned} a_k &= \frac{1}{N}\sum_{n=0}^{N-1} y_n \left(\omega^{kn} + \omega^{(N-k)n}\right) \\ &= \frac{1}{N}\sum_{n=0}^{N-1} y_n \left(\omega^{nk} + \omega^{-nk}\right) \\ &= \frac{1}{N}\sum_{n=0}^{N-1} y_n \left(e^{-2\pi ikn/N} + e^{2\pi ikn/N}\right) \end{aligned}$$

$$\boxed{a_k = \frac{2}{N}\sum_{n=0}^{N-1} y_n \cos\frac{2\pi kn}{N}} \tag{4.23}$$

Por eso se llama transformada cosenoidal discreta de Fourier, por que es función de los cosenos.

Para b_k:

$$\begin{aligned} b_k &= \frac{1}{N} i \sum_{n=0}^{N-1} y_n \left(\omega^{kn} - \omega^{-kn}\right) \\ &= \dots \\ &= \frac{i}{N} \sum_{n=0}^{N-1} y_n \left(-2i\right) \operatorname{sen} \frac{2\pi kn}{N} \end{aligned}$$

$$\boxed{b_k = \frac{2}{N} \sum_{n=0}^{N-1} y_n \operatorname{sen} \frac{2\pi kn}{N}} \tag{4.24}$$

Casos particulares:

- ✏ $k = 0$:

$$a_0/2 = \alpha_0 = \frac{1}{N} c_0 = \frac{1}{N} \sum_{n=0}^{N-1} y_n \omega^{0n} = \frac{1}{N} \sum_{n=0}^{N-1} y_n$$

  El coeficiente es la media de los valores de la función

- ✏ Si N es par:

$$a_M/2 = \alpha_M = \frac{1}{N} c_M = \frac{1}{N} \sum_{n=0}^{N-1} y_n \omega^{Mn} = \frac{1}{N} \sum_{n=0}^{N-1} y_n \left(-1\right)^n$$

Estas fórmulas son de interés para aproximar los coeficientes del desarrollo de Fourier: para $f\left(x\right)$ periódica en (a, b), el desarrollo en series de Fourier es

$$f\left(x\right) = \frac{A_0}{2} + \sum_{k=1}^{\infty} A_k \cos\left(\frac{2\pi}{b-a} kx\right) + \sum_{k=1}^{\infty} B_k \operatorname{sen}\left(\frac{2\pi}{b-a} kx\right)$$

Donde los coeficientes A_k y B_k son

$$A_k = \frac{2}{b-a} \int_a^b f\left(x\right) \cos\left(\frac{2\pi}{b-a} kx\right) dx \qquad B_k = \frac{2}{b-a} \int_a^b f\left(x\right) \operatorname{sen}\left(\frac{2\pi}{b-a} kx\right) dx$$

Si en vez de hacer el desarrollo con infinitos términos se aproxima con M términos,

$$\begin{aligned} T_M\left(x\right) &= \frac{A_0}{2} + \sum_{k=1}^{M} A_k \cos\left(\frac{2\pi}{b-a} kx\right) + \sum_{k=1}^{M} B_k \operatorname{sen}\left(\frac{2\pi}{b-a} kx\right) \simeq \\ &\simeq \frac{\hat{A}_0}{2} + \sum_{k=1}^{M} \hat{A}_k \cos\left(\frac{2\pi}{b-a} kx\right) + \sum_{k=1}^{M} \hat{B}_k \operatorname{sen}\left(\frac{2\pi}{b-a} kx\right) \triangleq t_M\left(x\right) \end{aligned}$$

Donde hemos aproximado

$$\hat{A}_k \simeq A_k \qquad \hat{B}_k \simeq B_k$$

Y siendo

$$\hat{A}_k = a_k \cos\left(\frac{2\pi}{b-a} ka\right) - b_k \operatorname{sen}\left(\frac{2\pi}{b-a} ka\right)$$

$$\hat{B}_k = a_k \operatorname{sen}\left(\frac{2\pi}{b-a}ka\right) + b_k \cos\left(\frac{2\pi}{b-a}ka\right)$$

Para que el desarrollo en series de Fourier se pueda hacer de esta forma, hay que considerar al menos $2M+2$ puntos en $[a,b]$, es decir, $N \geq 2M+1$. Esto es debido a que si se toman menos puntos, aparece *aliasing*.

Si se toman más puntos, se aproximan mejor los primeros A_k y B_k.

4.7 Teoría de la aproximación en espacios vectoriales normados

4.7.1 Introducción

Un espacio normado no es más que un espacio vectorial con una norma, por lo que se pueden definir distancias.

Ahora tratamos de aproximar un elemento f de un conjunto $\mathbb{E}$ mediante elementos de un subconjunto $\mathbb{V}$ de $\mathbb{E}$, siendo esta la mejor aproximación posible.

Hay que elegir un criterio de aproximación. Una vez fijado el criterio que seguimos tenemos que preguntarnos:

- ¿Bajo qué condiciones existe alguna mejor aproximación? ¿Es esta única?
- ¿Cómo se caracterizan las mejores aproximaciones?
- ¿Cómo se calculan?

Vamos a trabajar con espacios normados y subespacios de dimensión finita, y consideraremos aproximaciones mínimo-cuadráticas.

¿Y por qué queremos aproximar teniendo la interpolación? La respuesta es que cuando se hacen mediciones con un polímetro, un osciloscopio, etc., tenemos valores con error, por lo que tiene más sentido aproximar la nube de puntos que interpolarla, y con ello se obtiene un polinomio de menor grado.

El menor error posible en la aproximación es en el caso de que sea interpolación, es decir, que el grado del polinomio coincida con el número de puntos.

Este error será:

- Caso discreto (nube de puntos): es un vector de errores de $\mathbb{R}^k$ con k puntos
- Caso continuo (aproximando una función en un intervalo): es una función

  $$e(x) = f(x) - p_n(x)$$

  Para ver qué polinomio aproxima mejor tendremos que definir las normas funcionales

Veamos las normas que podemos definir, tanto para el caso discreto como para el continuo.

Caso	Producto escalar	Norma 2	Norma ∞
Discreto	$\langle x, y\rangle = \sum x_i y_i$	$\|e\|_2 = \sqrt{\langle e, e\rangle} = \sqrt{\sum e_i^2}$	$\|e\|_\infty = \max\limits_{1\leq i\leq m} w_i \lvert e_i\rvert$
Continuo	$\langle f, g\rangle = \int_a^b f(x)\, g(x)\, dx$	$\|e(x)\|_2 = \sqrt{\langle f, f\rangle} = \sqrt{\int_a^b e^2(x)\, dx}$	$\|e(x)\|_\infty = \max\limits_{a\leq x\leq b} w(x) \lvert e(x)\rvert$

Donde w_i y $w(x)$ son los pesos, ya que se pueden tener más en cuenta unas componentes que otras, es decir, dar más importancia a la información de unos puntos que a la de otros.

Ejemplo 4.7:

Sea la función

$$e(x) = \frac{1}{1+x}$$

en el intervalo $[0,1]$ con función de peso $w(x) = 1$. La norma 2 es

$$\|e(x)\|_2 = \sqrt{\int_0^1 \frac{dx}{(1+x)^2}} = \sqrt{\left[-\frac{1}{1+x}\right]_0^1} = \sqrt{-\frac{1}{2}+1} = \frac{\sqrt{2}}{2}$$

Mientras que la norma infinito

$$\|e(x)\|_\infty = \max_{0\leq x\leq 1}\left|\frac{1}{1+x}\right| = 1$$

para $x = 0$. Como la norma infinito es la unidad, se dice que está normalizada en norma infinito.

■

4.7.2 Teoremas y definiciones

Teorema 4.1. Dado un subespacio $\mathbb{V}$ del espacio normado $\mathbb{E}$, de dimensión finita, para cada $f \in \mathbb{E}$ existe alguna mejor aproximación en $\mathbb{V}$, y el conjunto de las mejores aproximaciones de f es un subconjunto convexo de $\mathbb{V}$.

Este teorema quiere decir que para que haya al menos una mejor aproximación, debe ser $\mathbb{V}$ un subespacio de dimensión finita.

Definición:

Un subconjunto de un espacio $\mathbb{V}$ es convexo si cumple:

$$\lambda f + (1-\lambda) g \in \mathbb{V}$$

$$\text{Si } f, g \in V \qquad 0 \leq \lambda \leq 1$$

Teorema 4.2. Si $\mathbb{V}$ es un espacio estrictamente convexo, para cada $f \in \mathbb{E}$ existe una y sólo una aproximación en un subespacio $\mathbb{V}$ de dimensión finita.

Definición:

Un espacio vectorial normado $\mathbb{E}$ es estrictamente convexo si:

$$\forall f, g \in \mathbb{E} / \begin{cases} f \neq g \\ \\ \|f\| = \|g\| = 1 \end{cases} \Rightarrow \begin{cases} \|\lambda f + (1-\lambda) g\| < 1 \\ \\ 0 < \lambda < 1 \end{cases}$$

O, de forma equivalente,

$$\left\|\frac{f+g}{2}\right\| < 1$$

Ejemplo 4.8:

El conjunto $\mathbb{R}^2$ con la norma infinito no es estrictamente convexo, ya que hay segmentos que unen puntos de norma 1 que a su vez son también de norma 1, en cambio, con la norma 2 sí es estrictamente convexo.

■

Definición:

Producto interior o escalar:

1. $\langle f, g\rangle \geq 0$ y $\langle f, f\rangle = 0 \Leftrightarrow f = 0$
2. $\langle f, g\rangle = \langle g, f\rangle$
3. $\langle \alpha f + \beta g, h\rangle = \alpha \langle f, h\rangle + \beta \langle g, h\rangle$

✏ E es normado con la norma $\|f\| = \sqrt{\langle f, f\rangle}$ (un vector no tiene norma cero si no es el vector nulo).

✏ E es estrictamente convexo.

Teorema 4.3. (Para caracterizar la mejor aproximación):

Un elemento $v^* \in \mathbb{V}$ (subespacio con producto escalar) es la mejor aproximación de $f \in \mathbb{E}$ en $\mathbb{V}$ si y sólo si se cumple la condición de ortogonalidad:

$$\langle f - v^*, v\rangle = 0 \qquad \forall v \in \mathbb{V}$$

4.7.3 Método de las ecuaciones normales

Dada una base de $\mathbb{V}$ (de dimensión $n+1$), $\{\phi_i(x), i = 0:n\}$, según la condición de ortogonalidad anterior, las coordenadas de la mejor aproximación v^*, a_i^* en esa base son solución del siguiente sistema de ecuaciones normales:

$$\sum_{i=0}^{n} \langle \phi_i(x), \phi_j(x)\rangle a_i = \langle f, \phi_j(x)\rangle \qquad j = 0:n \tag{4.25}$$

x

Con una base ortogonal el sistema de ecuaciones es sencillo de resolver.

Dada una base:

$$\begin{pmatrix} \langle \phi_0, \phi_0\rangle & \dots & \langle \phi_0, \phi_n\rangle \\ \vdots & & \vdots \\ \langle \phi_n, \phi_0\rangle & \dots & \langle \phi_n, \phi_n\rangle \end{pmatrix}$$

Que se conoce como **Matriz de Gram**, se dice que la base es ortogonal si cumple

$$\langle \phi_i, \phi_k\rangle = 0$$

Para todo $i \neq k$. Entonces, para el sistema anterior se tiene:

$$a_k^* = \frac{\langle \phi_k, f \rangle}{\langle \phi_k, \phi_k \rangle}$$

Esto es, dadas unas funciones básicas $\phi_i(x)$, podemos escribir

$$\langle f - v^*, v \rangle = \left\langle f - \sum c_i^* \phi_i, \phi_j \right\rangle = 0 \qquad \forall j \in 1:n$$

Obteniéndose el **sistema de ecuaciones normales** siguiente

$$j = 1 \quad \left\langle f - \sum c_i^* \phi_i, \phi_1 \right\rangle = 0 \quad \rightarrow \quad \langle f, \phi_1 \rangle = \sum c_i^* \langle \phi_i, \phi_1 \rangle$$

$$j = 2 \quad \left\langle f - \sum c_i^* \phi_i, \phi_2 \right\rangle = 0 \quad \rightarrow \quad \langle f, \phi_2 \rangle = \sum c_i^* \langle \phi_i, \phi_2 \rangle$$

$$\vdots$$

$$j = n \quad \left\langle f - \sum c_i^* \phi_i, \phi_n \right\rangle = 0 \quad \rightarrow \quad \langle f, \phi_n \rangle = \sum c_i^* \langle \phi_i, \phi_n \rangle$$

Escrito matricialmente

$$\begin{pmatrix} \langle \phi_1, \phi_1 \rangle & \langle \phi_2, \phi_1 \rangle & \dots & \langle \phi_n, \phi_1 \rangle \\ \langle \phi_1, \phi_2 \rangle & \langle \phi_2, \phi_2 \rangle & \dots & \langle \phi_n, \phi_2 \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \phi_1, \phi_n \rangle & \langle \phi_2, \phi_n \rangle & \dots & \langle \phi_n, \phi_n \rangle \end{pmatrix} \begin{pmatrix} c_1^* \\ c_2^* \\ \vdots \\ c_n^* \end{pmatrix} = \begin{pmatrix} \langle f, \phi_1 \rangle \\ \langle f, \phi_2 \rangle \\ \vdots \\ \langle f, \phi_n \rangle \end{pmatrix}$$

Lo mejor es que la matriz sea diagonal, para calcular los c_i^* más fácilmente. Esos c_i^* serán los que nos den la mejor aproximación.

Cuando el conjunto de funciones básicas de como resultado una matriz diagonal, será una base ortogonal y

$$\boxed{c_i^* = \frac{\langle f, \phi_i \rangle}{\langle \phi_i, \phi_i \rangle} \qquad i = 1:n} \tag{4.26}$$

4.8 Aproximación por mínimos cuadrados

4.8.1 Introducción

En la aproximación por mínimos cuadrados se minimiza la norma 2, mientras que si se minimiza la norma infinito se llama aproximación minimax, pero no la vamos a ver.

4.8.2 Aproximación polinomial por mínimos cuadrados continua

En este caso lo que tenemos es información de la función en todos los puntos de un intervalo $[a, b]$.

Si tenemos un espacio de funciones continuas en un intervalo $[a, b]$ con el producto escalar:

$$\langle f, g\rangle = \int_a^b w\left(x\right) f\left(x\right) g\left(x\right) dx$$

con $w\left(x\right) \geq 0$ en $[a, b]$ (función peso), nula en un número finito de puntos, y de forma que para toda función continua en el intervalo exista

$$\int_a^b w\left(x\right) f\left(x\right) dx$$

tratamos de hallar el polinomio de grado a lo sumo n, p_n que mejor aproxime a f. Esto implica minimizar

$$E = \int_a^b \left(f\left(x\right) - p_n\left(x\right)\right)^2 dx$$

Siendo $p_n\left(x\right) = \sum_{k=0}^{n} a_k x^k$:

$$E = \int_a^b \left(f\left(x\right)\right)^2 dx - 2\sum_{k=0}^{n} a_k \int_a^b x^k f\left(x\right) dx + \int_a^b \left(\sum_{k=0}^{n} a_k x^k\right)^2 dx$$

Por tanto, al minimizar para los a_j:

$$\frac{\partial E}{\partial a_j} = -2\int_a^b x^j f\left(x\right) dx + 2\sum_{k=0}^{n} a_k \int_a^b x^{j+k} dx = 0$$

Se obtiene el siguiente sistema de ecuaciones normales:

$$\sum_{k=0}^{n} a_k \int_a^b x^{j+k} dx = \int_a^b x^j f\left(x\right) dx \qquad j = 0 : n \tag{4.27}$$

El sistema tiene solución única dados $f \in C\left[a, b\right]$ y $a \neq b$. La resolución del sistema es inmediata si se usan polinomios ortogonales sobre el intervalo con respecto a la función peso $w\left(x\right)$, generados con la recurrencia de la sección anterior. Además, el uso de polinomios ortogonales permite aprovechar el esfuerzo de obtener el polinomio de grado k para obtener el de grado $k + 1$.

Por el contrario, la evaluación de las integrales

$$\int_a^b w\left(x\right) f\left(x\right) p_j\left(x\right) dx$$

normalmente requiere algún tipo de discretización si f es muy complicada (integración numérica, que se verá en el tema siguiente).

4.8.2.1 Base polinómica estándar

Este caso es el que las funciones básicas son

$$\boxed{\varphi_i\left(x\right) = x^i \qquad i = 0 : n} \tag{4.28}$$

Por lo que el polinomio de aproximación, de grado n, será

$$\boxed{p^*(x) = c_0^* + c_1^* x + c_2^* x^2 + \ldots + c_n^* x^n} \tag{4.29}$$

Esta es la mejor aproximación, cumpliendo los c_i^*

$$\left[\int_a^b w(x)\, x^{i+j} dx\right] C^* = \left[\int_a^b w(x)\, f(x)\, x^j dx\right] \qquad C^* = \left[\begin{array}{cccc} c_0^* & c_1^* & \ldots & c_n^* \end{array}\right]^t$$

Siendo la matriz del sistema de tamaño $(n+1) \times (n+1)$ simétrica y definida positiva, y sus filas son las integrales de la ecuación anterior.

4.8.2.2 Base polinómica ortogonal

Si la base es ortogonal,

$$\boxed{\varphi_i(x) = p_i(x) \qquad i = 0:n} \tag{4.30}$$

Se escribe el polinomio de aproximación como

$$\boxed{p^*(x) = \sum_{i=0}^{n} c_i^* p_i(x)} \tag{4.31}$$

Donde ahora los coeficientes c_i^* se pueden escribir como

$$\boxed{c_i^* = \frac{\displaystyle\int_a^b w(x)\, f(x)\, p_i(x)\, dx}{\displaystyle\int_a^b w(x)\, p_i(x)\, p_i(x)\, dx} \qquad i = 0:n} \tag{4.32}$$

La existencia de polinomios ortogonales en un intervalo $[a,b]$ para cualquier función peso $w(x)$ se puede demostrar:

Teorema 4.4. Existe una sucesión de polinomios $p_0, p_1, \ldots, p_k, \ldots$ que cumple $grado(p_i) = i$ y $\langle q, p_k \rangle = 0$ para todo polinomio q de grado menor que k. Esto es, el de grado k es ortogonal a todos los de grado menor. Definido el producto escalar como vimos antes,

$$\langle f, g \rangle = \int_a^b w(x)\, f(x)\, g(x)\, dx$$

La sucesión de polinomios cumple la siguiente recurrencia

$$\boxed{p_{j+1}(x) = \alpha_j (x - \beta_j)\, p_j(x) - \gamma_j p_{j-1}(x)} \tag{4.33}$$

Siendo

$$p_{-1}(x) = 0 \qquad p_0(x) = C_0 \qquad \alpha_j = \frac{C_{j+1}}{C_j} \qquad \beta_j = \frac{\langle p_j, x p_j \rangle}{\langle p_j, p_j \rangle} \qquad \gamma_j = \frac{\alpha_j \langle p_j, p_j \rangle}{\alpha_{j-1} \langle p_{j-1}, p_{j-1} \rangle}$$

donde $C_j \neq 0$ es el coeficiente principal del polinomio p_j.

Una vez fijada la sucesión de coeficientes principales C_j, la sucesión de polinomios ortogonales en un intervalo $[a,b]$ y para una función peso $w(x)$ dada es única.

A continuación veremos algunas sucesiones de polinomios ortogonales:

Nombre	Intervalo	Peso	Polinomio
Legendre	$[-1,1]$	$w(x)=1$	$\langle P_j,P_j\rangle=\frac{2}{2j+1}$ $(j+1)P_{j+1}(x)=(2j+1)xP_j(x)-jP_{j-1}(x)$
Chebyshev	$[-1,1]$	$w(x)=\frac{1}{\sqrt{1-x^2}}$	$\langle T_j,T_j\rangle=\begin{cases}\pi & j=0\\ \pi/2 & j>0\end{cases}$ $T_{j+1}(x)=2xT_j(x)-T_{j-1}(x)$
Laguerre	$[0,\infty]$	$\omega(x)=e^{-x}$	$\langle L_j,L_j\rangle=(j!)^2$ $L_{j+1}(x)=(2j+1-x)L_j(x)-j^2L_{j-1}(x)$
Hermite	$[-\infty,\infty]$	$w(x)=e^{-x^2}$	$\langle H_j,H_j\rangle=j!2^j\sqrt{\pi}$ $H_{j+1}(x)=2xH_j(x)-2jH_{j-1}(x)$

Siendo $P_{-1}(x)=0$ y $P_0(x)=1$, llevado a todos los polinomios, T, L y H.

Para los polinomios de Legendre:

$$P_n^*(x)=a_o^*P_0(x)+\ldots+a_n^*P_n(x)$$

$$a_i^*=\frac{\langle f,P_i\rangle}{\langle P_i,P_i\rangle}$$

Siendo $\langle f,P_i\rangle=\int_{-1}^{1}f(x)P_i(x)\,dx$.

Pudiéndose llevar esto a los otros polinomios teniendo en cuenta los intervalos y la función peso (que multiplica a f y P_i en la integral del producto escalar).

Los polinomios ortogonales de Chebyshev cumplen además que sus raíces son los mejores nodos para la aproximación minimax.

Dado un conjunto ortogonal de polinomios, $\{p_n\}_{n\in\mathbb{N}}$, entonces:

- ✏ Las raíces de $p_n(x)$ son reales, distintas, y están en $[a,b]$, siendo este el intervalo en el que dichos polinomios son ortogonales
- ✏ Las raíces de $p_{n+1}(x)$ se entrelazan con las de $p_n(x)$

4.8.3 Aproximación trigonométrica por mínimos cuadrados continua

Ahora el conjunto de funciones es continua en un intervalo $[a,b]$ de longitud 2π. Sea el espacio de dimensión infinita $\mathbb{E}$ de funciones $C[-\pi,\pi]$ (devuelven valores reales) o $L_2[-\pi,\pi]$ (devuelven valores complejos) que tiene el producto escalar

$$\langle f,g\rangle=\int_{(2\pi)}f(x)\,\overline{g(x)}dx$$

y el subespacio de $\mathbb{E}$, de dimensión finita, $2n+1$ y engendrado por las siguientes $2n+1$ funciones básicas:

$$\boxed{1,\cos(x),\operatorname{sen}(x),\cos(2x),\operatorname{sen}(2x),\ldots,\cos(nx),\operatorname{sen}(nx)}$$

Queremos encontrar la mejor aproximación a $f \in \mathbb{E}$ que tenga la siguiente forma

$$s_n(x) = \frac{a_0}{2} + \sum_{k=1}^{n} \left(a_k \cos(kx) + b_k \operatorname{sen}(kx)\right) \tag{4.34}$$

Esta mejor aproximación existe y es única. Además, la base es ortogonal, ya que

$$\langle \varphi_k, \varphi_l \rangle = \int_{-\pi}^{\pi} \varphi_k(x)\, \varphi_l(x)\, dx = \begin{cases} 0 & k \neq l \\ 2\pi & k = l = 0 \\ \pi & k = l > 0 \end{cases}$$

Los coeficientes de s_n^* son

$$\begin{aligned} a_k^* &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(kx)\, dx \qquad k = 0 : n \\ b_k^* &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \operatorname{sen}(kx)\, dx \qquad k = 1 : n \end{aligned}$$

En este desarrollo, cuando $n \to \infty$, se tiende al desarrollo en series de Fourier, que podemos escribir como

$$f(x) \simeq \sum_{k=0}^{\infty} \frac{\langle f, \varphi_k \rangle}{\langle \varphi_k, \varphi_k \rangle} \varphi_k(x)$$

4.8.4 Aproximación polinomial por mínimos cuadrados discreta

Dados m valores reales y_i, para $i = 1 : m$, de una función f en los puntos x_i, $i = 1 : m$, el problema es determinar un polinomio p_n $(n < m)$ de forma que minimice

$$\|e\|_2^2 = \sum_{i=1}^{m} w_i \left(y_i - p_n(x_i)\right)^2$$

Siendo los w_i pesos positivos. Se tendrá por tanto un polinomio de la forma

$$p_n(x_i) = \sum_{k=0}^{n} c_k \varphi_k(x_i) = Ac$$

Donde

$$A = \begin{pmatrix} \varphi_0(x) & \varphi_1(x) & \dots & \varphi_n(x) \end{pmatrix} \qquad c = \begin{pmatrix} c_0 & c_1 & \dots & c_n \end{pmatrix}^t$$

Donde las funciones $\varphi_k(x)$ están dispuestas como columnas, siendo cada fila un valor de x_k.

La aproximación del vector y es $Ac^* \in \mathbb{V} = \Re(A)$.

Si los valores en los nodos son todos distintos, entonces A tiene rango completo por columnas, por tanto

$$A^t A y = A^t c$$

Donde $A^t A$ es una matriz cuadrada invertible. Si los nodos no son distintos, no se asegura que A tenga rango completo por columnas.

La aplicación de esto es aproximar una información que se transmite y llega con ruido, dando mayor o menor peso según resulte la forma de onda esperada.

Se puede utilizar la base polinómica canónica:

$$\phi_0(x) = 1, \quad \phi_1(x) = x, \ldots, \quad \phi_n(x) = x^n$$

$$\langle \phi_k, \phi_j \rangle = \left\langle x^k, x^j \right\rangle = \sum_{i=1}^{m} w_i x_i^k x_i^j$$

$$\langle \phi_k, f \rangle = \sum_{i=1}^{m} w_i f(x_i) x_i^k$$

El sistema de ecuaciones normales queda:

$$A^t W A a = A^t W y$$

Sin pesos se obtiene, para $n = 3$,

$$A^t A = \begin{pmatrix} \sum 1 & \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \sum x_i^5 \\ \sum x_i^3 & \sum x_i^4 & \sum x_i^5 & \sum x_i^6 \end{pmatrix}$$

Si m crece, aparecen integrales y en las ecuaciones normales queda la matriz de Hilbert.

W es diagonal con los pesos w_i y el vector $y = (y_1, y_2, \ldots y_m)^t$

Este sistema es mal condicionado, por lo que se soluciona el siguiente sistema (solución de mínimos cuadrados) por alguno de los métodos de Householder, Givens o Gram-Schmidt:

$$Ma = b$$

$$M = \sqrt{W} a, \qquad b = \sqrt{W} y$$

$$\begin{pmatrix} \sqrt{w_1} & \ldots & 0 \\ \vdots & & \vdots \\ 0 & \ldots & \sqrt{w_n} \end{pmatrix} \begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & & & & \vdots \\ 1 & x_m & x_m^2 & \ldots & x_m^n \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(x_1)\sqrt{w_1} \\ \vdots \\ f(x_n)\sqrt{w_n} \end{pmatrix} \tag{4.35}$$

Si formar las ecuaciones normales es un problema mal condicionado, con los pesos se agrava aún más. Por tanto, es mejor resolver los sistemas sobredeterminados sin formar las ecuaciones normales.

Usando una base de polinomios ortogonales en un conjunto de puntos x_i, p_k, se reduce el sistema de ecuaciones normales a diagonal. Estos polinomios se pueden obtener aplicando la relación de recurrencia tres términos del caso continuo, pero interpretando los productos escalares de forma discreta, es decir

$$\langle f, g \rangle = \sum_{i=1}^{m} w_i f(x_i) g(x_i)$$

Con los mismos coeficientes que se obtenían en el caso anterior:

$$\alpha_j = \frac{C_{j+1}}{C_j} \qquad \beta_j = \frac{\langle p_j, xp_j\rangle}{\langle p_j, p_j\rangle} \qquad \gamma_j = \frac{\alpha_j \langle p_j, p_j\rangle}{\alpha_{j-1}\langle p_{j-1}, p_{j-1}\rangle}$$

Donde

$$xp_j = \begin{pmatrix} x_1 p_j(x_1) \\ x_2 p_j(x_2) \\ \vdots \\ x_m p_j(x_m) \end{pmatrix}$$

Siendo por tanto el producto componente a componente.

Y la base resultante es un vector de $\mathbb{R}^m$.

Entonces, dado que la base es ortogonal,

$$\langle p_k, p_j\rangle \sum_{i=1}^{m} w_i p_k(x_i) p_j(x_i) = 0 \qquad k \neq j$$

Y las coordenadas de la mejor aproximación vienen dadas por

$$\boxed{c_k^* = \frac{\sum_{i=1}^{m} w_i y_i p_k(x_i)}{\sum_{i=1}^{m} w_i p_k^2(x_i)} \qquad k = 0:n} \tag{4.36}$$

La base ortogonal para el caso discreto no se forma tomando los polinomios ortogonales del caso continuo y muestreándolos en los puntos de interés, ya que el resultado no es una base ortogonal.

4.8.5 Aproximación trigonométrica por mínimos cuadrados discreta

Se tienen los puntos $x_k = \frac{2\pi k}{N}$ equiespaciados, siendo $k = 0 : N-1, N = 2M+1$, en el intervalo $[0, 2\pi]$ y los valores de la función f en esos puntos $y_k, k = 0 : N-1$.

Queremos obtener la función trigonométrica:

$$\phi(x) = \frac{a_0}{2} + \sum_{k=1}^{q} (a_k \cos kx + b_k \operatorname{sen} kx) \qquad q \leq M \tag{4.37}$$

de forma que minimice

$$\sum_{k=0}^{N-1} (y_k - \phi(x_k))^2$$

Tendrá menos de M sumandos, ya que es aproximación y la fórmula interpolatoria tiene M sumandos.

Como en el caso de interpolación trigonométrica, el problema se transforma en uno complejo en el que se determina el polinomio

$$t_n(x) = \alpha_0 + \alpha_1 e^{ix} + \alpha_2 e^{2ix} + \ldots + \alpha_n e^{nix}$$

Siendo $\alpha_j \in \mathbb{C}$, $j = 0 : n$ y $n = 2q \leq N - 1$. Minimizando

$$\sum_{k=0}^{N-1} \left(y_k - t_n\left(x_k\right)\right)^2$$

En el espacio vectorial $\mathbb{C}^N$ cuyo producto escalar es $\langle x, y\rangle = \overline{x}^t v$ y siendo

$$w^{(k)} = \left(1, w_1^k, w_2^k, \ldots, w_{N-1}^k\right)^t$$

con $k = 0 : n$ y $w_k = e^{ix_k} = e^{\frac{2\pi ik}{N}}$, los $w^{(k)}$ son ortogonales y de norma 2 $\left\|w^{(k)}\right\|_2 = \sqrt{N}$.

Dicho de otra forma,

$$A = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w_1 & w_1^2 & \ldots & w_1^n \\ 1 & w_2 & w_2^2 & \ldots & w_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_{N-1} & w_{N-1}^2 & \ldots & w_{N-1}^n \end{pmatrix}$$

Si definimos

$$\omega = e^{-\frac{2\pi i}{N}}$$

Entonces el elemento k-ésimo del vector $w^{(j)}$ es

$$w_k^{(j)} = w_k^j = e^{ijx_k} = e^{ij\frac{2\pi k}{N}} = \omega^{-jk}$$

Con lo que podemos notar la matriz A también en función de los ω, quedando la misma matriz de Wandermonde compleja que en el caso de interpolación trigonométrica. Para $N = 4$ y $n + 1 = 2$ sería

$$A^H = F_N\left(:, 0 : n\right) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \end{pmatrix}$$

Entonces

$$F_N\left(:, 0 : 1\right) F_N^H\left(:, 0 : 1\right) = \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}$$

Es decir, que el producto queda

$$F_N\left(:, 0 : n\right) F_N^H\left(:, 0 : n\right) = NI_{n+1}$$

Esto nos sirve para obtener los coeficientes de la mejor aproximación $t_n^*\left(x\right)$, ya que

$$NI_{n+1}\alpha = A^H y = \begin{pmatrix} {w^{(0)}}^H y \\ \vdots \\ {w^{(n)}}^H y \end{pmatrix} = \begin{pmatrix} \left\langle y, w^{(0)}\right\rangle \\ \vdots \\ \left\langle y.w^{(n)}\right\rangle \end{pmatrix}$$

Por tanto,

$$\boxed{\alpha_k = \frac{\left\langle y, w^{(k)}\right\rangle}{\left\|w^{(k)}\right\|_2^2} = \frac{1}{N}\sum_{k=0}^{N-1} y_k w_k^{(j)} \qquad j = 0 : n}$$

Ya que

$$\left\langle w^{(k)}, w^{(k)}\right\rangle = N$$

La ortogonalidad de los $w^{(k)}$ permite asegurar que incrementando n, la nueva mejor aproximación se obtiene añadiendo términos a la anterior.

Ejemplo 4.9:

Para $N = 4$,

$$c = Y = F_4 y$$

Con

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & 1 & \omega^2 \\ 1 & \omega^3 & \omega^2 & \omega \end{pmatrix}$$

Si cambiamos el orden de las columnas de la siguiente forma:

$$\begin{aligned} F_4\left(:, \begin{bmatrix} 0 & 2 & 1 & 3 \end{bmatrix}\right) &= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & \omega & \omega^3 \\ 1 & 1 & \omega^2 & \omega^2 \\ 1 & \omega^2 & \omega^3 & \omega \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & \omega & \omega^3 \\ 1 & 1 & -1 & -1 \\ 1 & \omega^2 & -\omega & -\omega^3 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 & \begin{pmatrix} 1 & 0 \\ 0 & \omega \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & \omega^2 \end{pmatrix} \\ 1 & \omega^2 & -\begin{pmatrix} 1 & 0 \\ 0 & \omega \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & \omega^2 \end{pmatrix} \end{pmatrix} \end{aligned}$$

Vemos que ω^2 para N coincide con el ω para $N/2$. Por tanto, podemos escribir la matriz F_4 en función de la matriz F_2:

$$F_4\left(:, \begin{bmatrix} 0 & 2 & 1 & 3 \end{bmatrix}\right) = \begin{pmatrix} F_2 & \Omega F_2 \\ F_2 & -\Omega F_2 \end{pmatrix}$$

Siendo

$$\Omega = \begin{pmatrix} 1 & 0 \\ 0 & \omega \end{pmatrix}$$

Entonces, el vector de coeficientes de Fourier se puede escribir como

$$Y = F_4 y = F_4\left(:, \begin{bmatrix} 0 & 2 & 1 & 3 \end{bmatrix}\right) y\left(\begin{bmatrix} 0 & 2 & 1 & 3 \end{bmatrix}\right) = \begin{pmatrix} I & \Omega \\ I & -\Omega \end{pmatrix} F_2 y\left(\begin{bmatrix} 0 & 2 & 1 & 3 \end{bmatrix}\right)$$

$$Y = \begin{pmatrix} I & \Omega \\ I & -\Omega \end{pmatrix} \begin{pmatrix} F_2 y\left(\begin{bmatrix} 0 & 2 \end{bmatrix}\right) \\ F_2 y\left(\begin{bmatrix} 1 & 3 \end{bmatrix}\right) \end{pmatrix}$$

Esta forma de obtener la transformada discreta de Fourier es la que vamos a emplear en el siguiente apartado, donde vemos el algoritmo de la Transformada Rápida de Fourier (FFT), siendo

$$Y_T = F_2 y\left(\begin{bmatrix} 0 & 2 \end{bmatrix}\right)$$

$$Y_B = F_2 y\left(\begin{bmatrix} 1 & 3 \end{bmatrix}\right)$$

■

4.9 La transformada discreta de Fourier y el algoritmo FFT

La **transformada discreta de Fourier** de N valores y_k, $k = 0 : N-1$ son los N números siguientes:

$$Y_j = \sum_{k=0}^{N-1} y_k w_k^{-j} \tag{4.38}$$

Para $j = 0 : N-1$.

El algoritmo FFT o **Transformada Rápida de Fourier** permite evaluar los valores Y_j con un coste de operaciones del orden $O\left(N\log_2\left(N\right)\right)$. A continuación se incluye el código en Matlab que implementaría este algoritmo, válido sólo para el caso de $N = 2^t$. En cualquier caso, el algoritmo FFT está implementado en Matlab internamente.

Algoritmo 4.4: Algoritmo de la Transformada Rápida de Fourier

```
% Implementacion en Matlab del algoritmo FFT
% como curiosidad
%
function Y = fftr(y,N)
if N == 1
   Y = y;
else
   m = N/2;
   w = exp(-2*pi*i/N);
   Yt = fftr(y(1:2:N),m);
   Yb = fftr(y(2:2:N),m);
   d = ones(m,1);
   for k = 1:m-1,
      d(k) = w^k;
   end;
   z = d.*Yb;
   Y = [Yt+z;Yt-z];
end;
```

♦

TEMA 5

Derivación e integración numéricas

En este tema vamos a tratar de estimar la derivada o la integral de una función en un punto o intervalo, de forma numérica, bien porque la función sea muy complicada para hacerlo de forma analítica o porque sólo dispongamos de una tabla de datos de esa función, con la que tengamos que trabajar para obtener su derivada o integral.

5.1 Métodos de derivación numérica

5.1.1 Introducción

Dados $k \geq 1$ y α, vamos a tratar de aproximar o calcular $f^{(k)}(\alpha)$, de forma que tienda a un sólo número, utilizando fórmulas lineales del tipo

$$\sum_{i=0}^{n} A_i f(x_i) \simeq f^{(k)}(\alpha)$$

Donde $x_i, i = 0 : n$ son los puntos donde se evalúa f y A_i los coeficientes que nos encontraremos en la fórmula de derivación numérica.

Tenemos varias opciones a la hora de elegir un método para resolver el problema:

1. Usar una **función que interpole o aproxime a una tabla de datos**:
   Teniendo los valores $(x_i, f(x_i))$ con $i = 0 : n$, si usamos la fórmula de interpolación, obtenemos $f^{(k)}(\alpha) \simeq P_n^{(k)}(\alpha)$, siendo:

   $$f^{(k)}(\alpha) = P_n^{(k)}(\alpha) + D^k E(x)\Big|_{x=\alpha} = \sum_{i=0}^{n} A_i f(x_i) + E^{(k)}(\alpha) \tag{5.1}$$

   Con $A_i = l_i^{(k)}(\alpha)$, si usamos la interpolación de Lagrange.

2. **Método de los coeficientes indeterminados**: imponiendo que la fórmula sea exacta para x^j con $j = 0 : l$, siendo l el grado más alto posible. De esta forma, se obtiene el siguiente sistema:

   $$D^k x^j\Big|_{x=\alpha} = \sum_{i=0}^{n} A_i x_i^j \qquad j = 0, 1, 2, \ldots$$

$$\sum_{i=0}^{n} A_i x_i^j = \begin{cases} 0 & j \leq k-1 \\ k! & j = k \\ \frac{(k+1)!}{1!}\alpha & j = k+1 \\ \frac{(k+2)!}{2!}\alpha^2 & j = k+2 \\ \vdots & \vdots \end{cases} \tag{5.2}$$

El sistema es lineal en los A_i si los x_i están fijados, no lineal en otro caso.

3. **Desarrollo de Taylor**:

$$\begin{aligned} f^{(k)}(\alpha) \simeq \sum_{i=0}^{n} A_i f(x_i) &= \sum_{i=0}^{n} A_i \left(\sum_{j=0}^{m} \frac{f^{(j)}(\alpha)}{j!}(x_i-\alpha)^j + \frac{f^{(m+1)}(c)}{(m+1)!}(x_i+\alpha)^{m+1} \right) \\ &= \sum_{j=0}^{m} f^{(j)}(\alpha) \left(\frac{1}{j!} \sum_{i=0}^{n} A_i (x_i-\alpha)^j \right) + \sum_{i=0}^{n} \frac{f^{(m+1)}(c)}{(m+1)!}(x_i-\alpha)^{m+1} \end{aligned}$$

haciendo cero todos los coeficientes menos el de la derivada k-ésima:

$$\sum_{i=0}^{n} A_i (x_i-\alpha)^j = \begin{cases} 0 & j = 0:m, j \neq k \\ k! & j = k \end{cases}$$

Con m hasta donde se pueda llegar, lo suficientemente grande. Este método no se usa normalmente.

La derivación es una operación local, por lo que interesa elegir los puntos x_i próximos al punto donde se quiere obtener la derivada.

5.1.2 Grado de exactitud de una fórmula de derivación

Una fórmula se dice de grado de exactitud q si q es el menor entero positivo para el que la fórmula es exacta para todo polinomio de grado no mayor que q y no exacta para algún polinomio de grado $q+1$.

Teorema 5.1. Si f es continuamente diferenciable hasta el orden necesario, una fórmula $\sum_{i=0}^{n} A_i f(x_i)$ se dice que es interpolatoria si y sólo si su grado de exactitud es mayor o igual que n.

Una fórmula interpolatoria tiene grado de exactitud mayor o igual que n.

Ejemplo 5.1:

Deducir una fórmula que estime la derivada segunda de una función como

$$A_0 f(\alpha-h) + A_1 f(\alpha) + A_2 f(\alpha+h)$$

Usando el método de coeficientes indeterminados, imponemos que sea exacta para $1, x, x^2$:

$$f(x) = 1 \rightarrow 0 = A_0 + A_1 + A_2$$

$$f(x) = x \rightarrow 0 = A_0(\alpha-h) + A_1\alpha + A_2(\alpha+h)$$

$$f(x) = x^2 \rightarrow 2 = A_0(\alpha-h)^2 + A_1\alpha^2 + A_2(\alpha+h)^2$$

Montamos el sistema:

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ \alpha - h & \alpha & \alpha + h & 0 \\ (\alpha - h)^2 & \alpha^2 & (\alpha + h)^2 & 2 \end{pmatrix}$$

Reduciendo por Gauss, queda:

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & h & 2h & 0 \\ 0 & 0 & 2h^2 & 2 \end{pmatrix}$$

Las soluciones, por tanto son:

$$A_0 = \frac{1}{h^2} \quad A_1 = -\frac{2}{h^2} \quad A_2 = \frac{1}{h^2}$$

Y la fórmula queda:

$$f''(\alpha) = \frac{f(\alpha - h) - 2f(\alpha) + f(\alpha + h)}{h^2}$$

■

Ejemplo 5.2:

Dada la siguiente tabla de valores

t_i	1	1.01	1.02	1.03	1.04
$I(t_i)$	3.1	3.12	3.14	3.18	3.24

Que corresponde a la corriente medida que circula por un circuito RL, con valores $R = 0.142\Omega$ y $L = 0.98H$. Para calcular la tensión impuesta al circuito en función de esa corriente, con la primera Ley de Kirchhoff se obtiene

$$E = LI' + RI$$

t está dado en segundos y I en amperios. Se nos pide estimar E en el instante de tiempo 1.02.

Usando la fórmula centrada para $t_i = 1.02$, estimamos la derivada de la corriente

$$I'(1.02) \simeq \frac{-3.12 + 3.18}{2 \cdot 0.01} = \frac{0.06}{0.02} = 3$$

Y con ella el valor de la tensión pedida es

$$E(1.02) = 0.98 \cdot 3 + 0.142 \cdot 3.14 \simeq 3.386$$

■

5.1.3 Error en las fórmulas de derivación numérica

Para fórmulas interpolatorias podemos estimar el error de discretización E_d ya que se dispone del error de interpolación

$$E_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n)$$

Y entonces se puede estimar el error de discretización como

$$E_d = E_n^{(k)}(\alpha)$$

Si la fórmula de derivación se ha obtenido con las condiciones de exactitud, el error de discretización se puede deducir como

$$E_d = \frac{f^{(q+1)}(\xi)}{(q+1)!}C$$

donde $q \geq n$ es el grado de exactitud de la fórmula de derivación, y siendo $q+1$ el grado de derivación de f para el que $C \neq 0$, obteniendo C como

$$f(x) = x^{n+1} \Rightarrow D^k f|_{x=\alpha} = \sum A_i x_i^{n+1} + C$$

Se supone f derivable hasta el grado que sea necesario.

Para fórmulas de derivación deducidas mediante el desarrollo en series de Taylor, la expresión para el error de discretización es

$$E_d = -\sum_{i=0}^{n} A_i \frac{f^{(m+1)}(\xi_i)}{(m+1)!}(x_i-\alpha)^{m+1}$$

Y en esta expresión se aprecia la conveniencia de que los puntos x_i que usemos para obtener la fórmula de derivación estén próximos al deseado, α.

5.1.4 Fórmulas de derivación numérica

En la tabla (5.1) vemos algunas de las fórmulas de derivación numérica más usuales, para las derivadas de orden 1, 2 y 3. El punto c es interior al intervalo de evaluación de la función en cada una de las fórmulas (por ejemplo, $[\alpha, \alpha+2h]$ en la fórmula progresiva de 3 puntos).

Estas fórmulas de derivación son todas interpolatorias. Es más recomendable saber deducirlas que sabérselas de memoria.

Estas fórmulas de derivación son inestables, debido a los errores de discretizacion (E_d) y a los que se producen al evaluar $f(x_i)$ (ε_i).

$$f(x_i) = \hat{f}_i + \varepsilon_i \rightarrow \sum_{i=0}^{n} A_i \varepsilon_i = E_r$$

Siendo E_r el error de redondeo.

Por ello, aunque en principio parecería lógico pensar que tomar un valor de h muy pequeño daría un buen resultado, debido a que

$$\lim_{h\to 0} |E_d| = 0$$

El error de redondeo de la fórmula es

$$\lim_{h\to 0} |E_r| = \infty$$

$f'(\alpha) = \frac{f(\alpha+h)-f(\alpha)}{h} - \frac{h}{2}f^{(2)}(c)$	
$f'(\alpha) = \frac{-3f(\alpha)+4f(\alpha+h)-f(\alpha+2h)}{2h} + \frac{h^2}{3}f^{(3)}(c)$	Fórmula progresiva 3 puntos
$f'(\alpha) = \frac{f(\alpha-2h)-4f(\alpha-h)+3f(\alpha)}{2h} + \frac{h^2}{3}f^{(3)}(c)$	Fórmula regresiva 3 puntos
$f'(\alpha) = \frac{-f(\alpha-h)+f(\alpha+h)}{2h} - \frac{h^2}{6}f^{(3)}(c)$	Fórmula centrada
$f'(\alpha) = \frac{f(\alpha-2h)-8f(\alpha-h)+8f(\alpha+h)-f(\alpha+2h)}{12h} + \frac{h^4}{30}f^{(5)}(c)$	Fórmula centrada de 5 puntos
$f'(\alpha) = \frac{-25f(\alpha)+48f(\alpha+h)-36f(\alpha+2h)+16f(\alpha+3h)-3f(\alpha+4h)}{12h} + \frac{h^4}{5}f^{(5)}(c)$	Fórmula progresiva 5 puntos
$f''(\alpha) = \frac{f(\alpha-h)-2f(\alpha)+f(\alpha+h)}{h^2} - \frac{h^2}{12}f^{(4)}(c)$	
$f''(\alpha) = \frac{f(\alpha)-2f(\alpha+h)+f(\alpha+2h)}{h^2} - hf^{(3)}(c_1) + \frac{h^2}{6}f^{(4)}(c_2)$	
$f''(\alpha) = \frac{-f(\alpha-2h)+16f(\alpha-h)-30f(\alpha)+16f(\alpha+h)-f(\alpha+2h)}{12h^2} + \frac{h^4}{90}f^{(6)}(c)$	
$f'''(\alpha) = \frac{-f(\alpha-2h)+2f(\alpha-h)-2f(\alpha+h)+f(\alpha+2h)}{2h^3} - \frac{h^2}{4}f^{(5)}(c)$	

Tabla 5.1: Fórmulas de derivación numérica

Por tanto, para un h demasiado pequeño se pueden obtener resultados muy malos.

Para obtener un valor de h óptimo, hay que minimizar $|E_r| + |E_d|$ o $|E_r| = |E_d|$ (dos criterios). En general, lo segundo no da el mismo punto que lo primero, pero es más fácil de resolver, y la diferencia no es demasiado grande.

Una problemática de la derivación numérica es que **no existen fórmulas de derivación estables**.

5.1.5 Algoritmos Matlab para derivación numérica

Vamos a ver en este apartado los códigos fuente en Matlab para realizar algunos de los métodos vistos de derivación numérica, para calcular las derivadas primeras, segundas y terceras en un punto α.

5.1.5.1 Algoritmos para la derivada primera

Algoritmo 5.1: Fórmula dos puntos para obtener la derivada primera en un punto

```
% Formula de derivacion de dos puntos
% Aproxima la derivada primera de la funcion en un punto
% Uso: sol=deriv2p(funcion,alfa,h)
function [fp]=deriv2p(f,alfa,h)
  if nargin<2
     disp('Error: faltan argumentos de entrada');
     return;
  else
     if nargin<3
```

```
        h=0.5;
      end;
    end;
    fp=(feval(f,alfa+h)-feval(f,alfa))/h;
```

♦

Algoritmo 5.2: Fórmula progresiva tres puntos para obtener la derivada primera en un punto

```
% Formula de derivacion progresiva de tres puntos
% Aproxima la derivada primera de la funcion en un punto
% Uso: sol=derivp3p(funcion,alfa,h)
function [fp]=derivp3p(f,alfa,h)
  if nargin<2
     disp('Error: faltan argumentos de entrada');
     return;
  else
     if nargin<3
        h=0.5;
     end;
  end;
fp=(-3*feval(f,alfa)+4*feval(f,alfa+h)-feval(f,alfa+2*h))/(2*h);
```

♦

Algoritmo 5.3: Fórmula regresiva tres puntos para obtener la derivada primera en un punto

```
% Formula de derivacion regresiva de tres puntos
% Aproxima la derivada primera de la funcion en un punto
% Uso: sol=derivr3p(funcion,alfa,h)
function [fp]=derivr3p(f,alfa,h)
  if nargin<2
     disp('Error: faltan argumentos de entrada');
     return;
  else
     if nargin<3
        h=0.5;
     end;
  end;
  fp=(feval(f,alfa-2*h)-4*feval(f,alfa-h)+3*feval(f,alfa))/(2*h);
```

♦

Algoritmo 5.4: Fórmula progresiva cinco puntos para obtener la derivada primera en un punto

```
% Formula de derivacion progresiva de cinco puntos
% Aproxima la derivada primera de la funcion en un punto
% Uso: sol=derivp5p(funcion,alfa,h)
function [fp]=derivp5p(f,alfa,h)
  if nargin<2
     disp('Error: faltan argumentos de entrada');
     return;
```

```
  else
     if nargin<3
        h=0.5;
     end;
  end;
  fp=(-25*feval(f,alfa)+48*feval(f,alfa+h)-36*feval(f,alfa+2*h)
      +16*feval(f,alfa+3*h)-3*feval(f,alfa+4*h))/(12*h);
```

♦

Algoritmo 5.5: Fórmula centrada dos puntos para obtener la derivada primera en un punto

```
% Formula de derivacion centrada de dos puntos
% Aproxima la derivada primera de la funcion en un punto
% Uso: sol=dercen2p(funcion,alfa,h)
function [fp]=dercen2p(f,alfa,h)
  if nargin<2
     disp('Error: faltan argumentos de entrada');
     return;
  else
     if nargin<3
        h=0.5;
     end;
  end;
  fp=(-feval(f,alfa-h)+feval(f,alfa+h))/(2*h);
```

♦

Algoritmo 5.6: Fórmula centrada cinco puntos para obtener la primera derivada en un punto

```
% Formula de derivacion centrada de cinco puntos
% Aproxima la derivada primera de la funcion en un punto
% Uso: sol=dercen5p(funcion,alfa,h)
function [fp]=dercen5p(f,alfa,h)
  if nargin<2
     disp('Error: faltan argumentos de entrada');
     return;
  else
     if nargin<3
        h=0.5;
     end;
  end;
  fp=(feval(f,alfa-2*h)-8*feval(f,alfa-h)+8*feval(f,alfa+h)
      -feval(f,alfa+2*h))/(12*h);
```

♦

5.1.5.2 Algoritmos para la derivada segunda

Algoritmo 5.7: Fórmula centrada tres puntos para obtener la derivada segunda en un punto

```
% Formula de derivacion centrada de tres puntos
% Aproxima la derivada segunda de la funcion en un punto
% Uso: sol=der2c3p(funcion,alfa,h)
function [fp]=der2c3p(f,alfa,h)
  if nargin<2
     disp('Error: faltan argumentos de entrada');
     return;
  else
     if nargin<3
        h=0.01;
     end;
  end;
  fp=(feval(f,alfa-h)-2*feval(f,alfa)+feval(f,alfa+h))/h^2;
```

♦

Algoritmo 5.8: Fórmula progresiva tres puntos para obtener la derivada segunda en un punto

```
% Formula de derivacion progresiva de tres puntos
% Aproxima la derivada segunda de la funcion en un punto
% Uso: sol=der2p3p(funcion,alfa,h)
function [fp]=der2p3p(f,alfa,h)
  if nargin<2
     disp('Error: faltan argumentos de entrada');
     return;
  else
     if nargin<3
        h=0.01;
     end;
  end;
  fp=(feval(f,alfa)-2*feval(f,alfa+h)+feval(f,alfa+2*h))/h^2;
```

♦

Algoritmo 5.9: Fórmula centrada cinco puntos para obtener la derivada segunda en un punto

```
% Formula de derivacion centrada de cinco puntos
% Aproxima la derivada segunda de la funcion en un punto
% Uso: sol=der2c5p(funcion,alfa,h)
function [fp]=der2c5p(f,alfa,h)
  if nargin<2
     disp('Error: faltan argumentos de entrada');
     return;
  else
     if nargin<3
        h=0.5;
     end;
  end;
  fp=(-feval(f,alfa-2*h)+16*feval(f,alfa-h)-30*feval(f,alfa)
      +16*feval(f,alfa+h)-feval(f,alfa+2*h))/(12*h^2);
```

♦

5.1.5.3 Algoritmos para la derivada tercera

Algoritmo 5.10: Fórmula centrada cuatro puntos para obtener la derivada tercera en un punto

```
% Formula de derivacion centrada de cuatro puntos
% Aproxima la derivada tercera de la funcion en un punto
% Uso: sol=der3c4p(funcion,alfa,h)
function [fp]=der3c4p(f,alfa,h)
  if nargin<2
     disp('Error: faltan argumentos de entrada');
     return;
  else
     if nargin<3
        h=0.5;
     end;
  end;
  fp=(-feval(f,alfa-2*h)+2*feval(f,alfa-h)-2*feval(f,alfa+h)
      +feval(f,alfa+2*h))/(2*h^3);
```

♦

5.2 Métodos de integración numérica de Newton-Côtes y de Gauss

5.2.1 Introducción al método

Ahora tratamos de calcular o aproximar el valor de la integral $\int_a^b f(x)\,dx$ por $\sum_{i=0}^{n} A_i f(x_i)$. Con función peso $w(x) \geq 0$ (nula sólo en un número finito de puntos), podemos calcular por la misma fórmula la integral $\int_a^b w(x) f(x)\,dx$.

Si la primitiva es difícil de determinar o de evaluar, podemos usar alguno de estos métodos.

1. **Fórmulas interpolatorias**: se usa el polinomio de interpolación para la función f con los puntos x_i

$$\int_a^b w(x) f(x)\,dx \simeq \int_a^b w(x) s(x)\,dx$$

Con $s(x)$ el polinomio interpolante o aproximatorio. Si $f(x) = P_n(x) + E(x)$:

$$\int_a^b w(x) f(x)\,dx = \int_a^b w(x) P_n(x)\,dx + \int_a^b w(x) E(x)\,dx$$

Al usar la interpolación de Lagrange, tenemos que:

$$\sum_{i=0}^{n} \left(\int_a^b \underbrace{w(x) l_i(x)}_{A_i} dx \right) f(x_i) + \int_a^b \frac{w(x) f^{(n+1)}(c)}{(n+1)!} \theta_{n+1}(x) \tag{5.3}$$

con $\theta_{n+1}(x) = (x - x_0) \cdots (x - x_n)$.

2. **Método de coeficientes indeterminados**: de nuevo, se obtiene una fórmula exacta para un conjunto de funciones básicas (polinomios de bajo grado, por ejemplo $f(x) = x^i, i = 0, 1, \ldots$, hasta el grado más alto posible. Queda el sistema:

$$\int_a^b w(x)\, x^i dx = \sum_{k=0}^{n} A_k x_k^i \qquad i = 0, 1, 2, \ldots \tag{5.4}$$

3. **Desarrollo de Taylor**:

$$\sum_{j=0} \frac{f^{(j)}(\alpha)}{j!} \int_a^b w(x)\,(x-\alpha)^j\, dx \simeq \sum_{j=0} \frac{f^{(j)}(\alpha)}{j!} \sum_{i=0}^{n} A_i\,(x_i-\alpha)^j \tag{5.5}$$

Se obtiene, básicamente, el mismo sistema que en el método de los coeficientes indeterminados, con la única diferencia de que es exacta para $x - \alpha$ en lugar de x. Sin embargo, este método no se usa.

5.2.2 Grado de exactitud de una fórmula de integración

Se define el grado de exactitud de una fórmula de cuadratura de la misma forma que para las fórmulas de derivación numérica: Una fórmula de cuadratura se dice de grado de exactitud q si dicho q es el menor entero positivo para el que la fórmula es exacta para todo polinomio de grado no mayor que q y no exacta para algún polinomio de grado $q + 1$.

Se puede ver también el mismo teorema que en el apartado de derivación numérica, sin ninguna variación.

5.2.3 Error en las fórmulas de integración numérica

Para las fórmulas que se obtengan por alguno de los métodos citados para integración numérica, se obtienen expresiones equivalentes del error al caso de derivación.

Si la fórmula es interpolatoria se puede usar la misma fórmula del error de interpolación, para obtener el siguiente error de discretización del método de integración

$$E_d = \int_a^b w(x)\, \frac{f^{(n+1)}(\xi(x))}{(n+1)!}\,(x-x_0)\,(x-x_1)\cdots(x-x_n)\,dx$$

Si se aplica el teorema del valor medio integral, entonces se puede demostrar (no lo haremos) que se tiene, para K_1 y K_2 constantes con el mismo signo,

$$E_d = K_1 f^{(n+1)}(\zeta_1) - K_2 f^{(n+1)}(\zeta_2)$$

En cambio, si la fórmula de integración se ha obtenido usando el método de los coeficientes indeterminados para conseguir cierto grado de exactitud q, entonces el error será

$$E_d = \frac{f^{(q+1)}(\xi)}{(q+1)!} C$$

Siendo C, de la misma forma que antes, determinada para $f(x) = x^{q+1}$ y no nulo.

5.2.4 Fórmulas de integración de Newton-Côtes

En la tabla (5.2) se pueden ver las fórmulas simples de Newton-Côtes cerradas, que son las que se usan cuando los dos extremos del intervalo están contenidos en los puntos x_i que se evalúan, y en la tabla (5.3) las fórmulas simples de Newton-Côtes abiertas, que se usan para intervalos en los que ninguno de los extremos está en el conjunto de los x_i. Para aquellos intervalos en los que uno de los extremos está en el conjunto de puntos, la fórmula se llama semiabierta, pero no vamos a verlas. En ambas tablas, c es un punto interior al intervalo de integración.

Para estas fórmulas de Newton-Côtes, no se distingue peso en el integrando, es decir: $w(x) = 1$, y se consideran los puntos x_i equiespaciados.

$$\int_{x_0}^{x_1} f(x)\,dx = \frac{h}{2}\left(f(x_0) + f(x_1)\right) - \frac{h^3}{12}f^{(2)}(c) \qquad \text{Trapecios}$$

$$\int_{x_0}^{x_2} f(x)\,dx = \frac{h}{3}\left(f(x_0) + 4f(x_1) + f(x_2)\right) - \frac{h^5}{90}f^{(4)}(c) \qquad \text{Simpson}$$

$$\int_{x_0}^{x_3} f(x)\,dx = \frac{3h}{8}\left(f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right) - \frac{3h^5}{80}f^{(4)}(c) \qquad \text{Simpson } 3/8$$

$$\int_{x_0}^{x_4} f(x)\,dx = \frac{2h}{45}\left(7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)\right) - \frac{8h^7}{954}f^{(6)}(c) \qquad \text{Boole}$$

Tabla 5.2: Fórmulas de Newton-Côtes cerradas simples

$$\int_{x_{-1}}^{x_1} f(x)\,dx = 2hf(x_0) + \frac{h^3}{3}f^{(2)}(c) \qquad \text{Punto medio}$$

$$\int_{x_{-1}}^{x_2} f(x)\,dx = \frac{3h}{2}\left(f(x_0) + f(x_1)\right) + \frac{3h^3}{4}f^{(2)}(c)$$

$$\int_{x_{-1}}^{x_3} f(x)\,dx = \frac{4h}{3}\left(2f(x_0) - f(x_1) + 2f(x_2)\right) + \frac{14h^5}{45}f^{(4)}(c)$$

$$\int_{x_{-1}}^{x_4} f(x)\,dx = \frac{5h}{24}\left(11f(x_0) + f(x_1) + f(x_2) + 11f(x_3)\right) + \frac{95h^5}{144}f^{(4)}(c)$$

Tabla 5.3: Fórmulas de Newton-Côtes abiertas simples

Al contrario de lo que ocurre con las fórmulas de derivación numérica, los errores de redondeo y discretización en la integración numérica van multiplicadas por h, con lo que tienden a cero cuando h tiende a cero. Por tanto, **las fórmulas de integración numérica son estables**.

5.2.5 Error de las fórmulas de Newton-Côtes

Como fórmulas interpolatorias que son, las fórmulas integrales de Newton-Côtes tienen un error de discretización, debido al error de interpolación:

$$E_d = \int_a^b w(x) E(x) dx = \int_a^b w(x) \frac{f^{(n+1)}(\xi(x))}{(n+1)!} (x - x_0)(x - x_1) \cdots (x - x_n) dx$$

Y como vimos antes, nos queda

$$E_d = K_1 f^{(n+1)}(\zeta_1) - K_2 f^{(n+1)}(\zeta_2)$$

Siendo K_1, K_2 constantes del mismo signo.

Las fórmulas integrales de Newton-Côtes usan $N = n + 1$ puntos, y su grado de exactitud es

- ✏ Si N es impar, entonces el grado de exactitud es N
- ✏ Si N es par, el grado de exactitud es $N - 1$

5.2.6 Fórmulas compuestas

En este tipo de fórmulas lo único que se hace es romper el intervalo de integración en varios subintervalos, de forma que se aplican las fórmulas simples que se deseen a cada subintervalo. Esto nos sirve para reducir el valor de h sin tener que recurrir a usar más puntos. Por contra, aumenta el número de operaciones a realizar.

Por ejemplo, para aplicar el método de Simpson dividiendo en m subintervalos:

$$\int_a^b f(x) dx = \int_{x_0}^{x_2} f(x) dx + \int_{x_2}^{x_4} f(x) dx + \ldots + \int_{x_{2m-2}}^{x_{2m}} f(x) dx$$

Siendo $x_0 = a$, $x_{2m} = b$ y $h = \frac{b-a}{2m}$.

Ejemplo 5.3:

Determinar los coeficientes de la siguiente fórmula integral:

$$\int_{-1}^{1} f(x) dx \simeq A_0 f(-1) + A_1 f(\alpha) + A_2 f(1)$$

Siendo α fijo e incluido en el intervalo $[-1, 1]$, de forma que la fórmula sea lo más exacta posible. Razonadamente, ¿es posible una mejor elección de α?

Usar la fórmula anterior para calcular $\int_0^1 \frac{2}{\sqrt{5x+4}} dx$ acotando el error cometido, y usando la mejor elección.

Empezamos por determinar los A_i:

$$f(x) = 1 \quad \int_{-1}^{1} dx = A_0 + A_1 + A_2 = 2$$

$$f(x) = x \quad \int_{-1}^{1} x dx = -A_0 + \alpha A_1 + A_2 = 0$$

$$f(x) = x^2 \quad \int_{-1}^{1} x^2 dx = A_0 + \alpha^2 A_1 + A_2 = \frac{2}{3}$$

Montamos el sistema:

$$\begin{pmatrix} 1 & 1 & 1 & 2 \\ -1 & \alpha & 1 & 0 \\ 1 & \alpha^2 & 1 & 2/3 \end{pmatrix}$$

Y resolviendo el sistema, obtenemos los coeficientes siguientes:

$$A_0 = \frac{1+3\alpha}{3(1+\alpha)} \quad A_1 = \frac{4}{3(1-\alpha)^2} \quad A_2 = \frac{1-3\alpha}{3(1-\alpha)}$$

El grado de exactitud de la fórmula es $q \geq 2$, ya que es exacta para cualquier polinomio de grado menor o igual que 2 (como hemos exigido al crear el sistema de ecuaciones). Veamos si es exacta para x^3:

$$\begin{aligned} \int_{-1}^{1} x^3 dx = 0 \simeq -A_0 + \alpha^3 A_1 + A_2 &= -\frac{1+3\alpha}{3\left(1+\alpha\right)} + \frac{4\alpha^3}{3\left(1-\alpha\right)^2} + \frac{1-3\alpha}{3\left(1-\alpha\right)} \\ &= \frac{-4}{3}\alpha \end{aligned}$$

Por tanto, si $\alpha \neq 0$, el grado de exactitud es 2, y si es $\alpha = 0$, $q \geq 3$.

Pero si $\alpha = 0$, nos quedan como coeficientes $A_0 = 1/3$, $A_1 = 4/3$ y $A_2 = 1/3$, de forma que la fórmula que resulta es la de Simpson, cuyo grado de exactitud es 3.

A continuación, pasamos a resolver la integral propuesta:

El intervalo de integracion es distinto al de la fórmula que tenemos que aplicar, con lo que necesitamos hacer un cambio de variable para que los intervalos coincidan:

$$x^{'} = 2x - 1 \Rightarrow x = \frac{1}{2}\left(x^{'} + 1\right)$$

Con el que la integral se transforma en:

$$\int_{-1}^{1} \frac{2}{\sqrt{\frac{5}{2}\left(x^{'}+1\right)+4}} \frac{1}{2} dx^{'}$$

Aplicando la fórmula, nos queda:

$$\int_{-1}^{1} \frac{1}{\sqrt{\frac{5}{2}\left(x^{'}+1\right)+4}} dx^{'} \simeq \frac{1}{3}\frac{1}{2} + \frac{4}{3}\frac{1}{\sqrt{\frac{13}{2}}} + \frac{1}{3}\frac{1}{3} = \frac{1}{3}\left(\frac{5}{6} + \frac{4\sqrt{2}}{\sqrt{13}}\right)$$

■

5.2.7 Algoritmos Matlab para integración numérica

De la misma forma que lo hicimos para las fórmulas de derivación numérica, vamos a incluir a continuación los algoritmos para ejecutar en Matlab las fórmulas de integración numérica que hemos visto.

Veremos tanto las fórmulas cerradas como las abiertas simples de Newton-Côtes, así como un algoritmo que nos permite efectuar integración compuesta a partir de esas fórmulas simples.

5.2.7.1 Fórmulas de Newton-Côtes cerradas simples

Algoritmo 5.11: Fórmula de los trapecios

```
% Metodo de los trapecios para integracion numerica
% de funciones
% Uso: sol=trapecio(funcion,a,b)
%
function [ys]=trapecio(f,a,b)
   if nargin<3
      disp('Faltan argumentos de entrada');
      return;
   end;
   h=b-a;
   ys=(feval(f,a)+feval(f,b))*(h/2);
```

♦

Algoritmo 5.12: Fórmula de Simpson

```
% Metodo de Simpson para integracion numerica
% de funciones
% Uso: sol=simpson(funcion,a,b)
%
function [ys]=simpson(f,a,b)
   if nargin<3
      disp('Faltan argumentos de entrada');
      return;
   end;
   h=(b-a)/2;
   c=(b-a)/2;
   ys=(feval(f,a)+4*feval(f,a+c)+feval(f,b))*(h/3);
```

♦

Algoritmo 5.13: Fórmula de Simpson 3/8

```
% Metodo de Simpson 3/8 para integracion numerica
% de funciones
% Uso: sol=simpso38(funcion,a,b)
%
function [ys]=simpson(f,a,b)
   if nargin<3
      disp('Faltan argumentos de entrada');
      return;
   end;
   h=(b-a)/3;
   c=a+h;
   d=c+h;
   ys=(feval(f,a)+3*feval(f,c)+3*feval(f,d)+feval(f,b))*(3*h/8);
```

♦

Algoritmo 5.14: Fórmula de Boole

```
% Metodo de Boole para integracion numerica
% de funciones
% Uso: sol=boole(funcion,a,b)
%
function [ys]=boole(f,a,b)
   if nargin<3
      disp('Faltan argumentos de entrada');
      return;
   end;
   h=(b-a)/4;
   c=a+h;
   d=c+h;
   e=d+h;
   ys=(7*feval(f,a)+32*feval(f,c)+12*feval(f,d)
       +32*feval(f,e)+7* feval(f,b))*(2*h/45);
```

♦

5.2.7.2 Fórmulas de Newton-Côtes abiertas simples

Algoritmo 5.15: Fórmula del punto medio

```
% Metodo del punto medio para integracion numerica
% de funciones
% Uso: sol=pmedio(funcion,a,b)
%
function [ys]=pmedio(f,a,b)
   if nargin<3
      disp('Faltan argumentos de entrada');
      return;
   end;
   h=b-a;
   ys=h*feval(f,(b+a)/2);
```

♦

Algoritmo 5.16: Fórmula abierta con dos puntos

```
% Metodo de integracion numerica de Newton-Cotes
% Formula abierta con 2 puntos
% Uso: sol=intab2(funcion,a,b)
%
function [ys]=intab2(f,a,b)
   if nargin<3
      disp('Faltan argumentos de entrada');
      return;
   end;
   h=(b-a)/3;
   ys=(3*h/2)*(feval(f,a+h)+feval(f,a+2*h));
```

♦

Algoritmo 5.17: Fórmula abierta con tres puntos

```
% Metodo de integracion numerica de Newton-Cotes
% Formula abierta con 3 puntos
% Uso: sol=intab3(funcion,a,b)
%
function [ys]=intab3(f,a,b)
   if nargin<3
      disp('Faltan argumentos de entrada');
      return;
   end;
   h=(b-a)/4;
   ys=(4*h/3)*(2*feval(f,a+h)-feval(f,a+2*h)+2*feval(f,a+3*h));
```

♦

Algoritmo 5.18: Fórmula abierta con cuatro puntos

```
% Metodo de integracion numerica de Newton-Cotes
% Formula abierta con 4 puntos
% Uso: sol=intab4(funcion,a,b)
%
function [ys]=intab4(f,a,b)
   if nargin<3
      disp('Faltan argumentos de entrada');
      return;
   end;
   h=(b-a)/5;
   ys=(5*h/24)*(11*feval(f,a+h)+feval(f,a+2*h)
       +feval(f,a+3*h)+11*feval(f,a+4*h));
```

♦

5.2.7.3 Integración compuesta

A continuación lo que hay es un algoritmo que, recibiendo la función, el intervalo, el número de subintervalos y el método a utilizar, descompone el intervalo y aplica el método indicado a cada subintervalo. Se puede usar con cualquiera de los algoritmos anteriores.

Algoritmo 5.19: Algoritmo para incluir integración compuesta con cualquiera de los algoritmos anteriores

```
% Metodo de integracion compuesta
% Escojer el metodo de integracion en los argumentos
% de entrada. El metodo por defecto es Simpson
% Opciones: Simpson, Trapecio, Simpson38, Boole, Pmedio,
% Abierta2, Abierta3, Abierta4
% Uso: sol=intcomp(funcion,a,b,intervalos,metodo)
%
function ys=intcomp(f,a,b,m,metodo)
   if nargin<4
      disp('Error: faltan argumentos de entrada');
```

```
   return;
else
   if nargin<5
      metodo='Simpson';
   end;
end;
% Subdivision de los intervalos
h=(b-a)/m;
ys=0;
if strcmp(metodo,'Simpson')
   for k=1:m
      ys=ys+simpson(f,a+(k-1)*h,a+k*h);
   end;
elseif strcmp(metodo,'Simpson38')
   for k=1:m
      ys=ys+simpso38(f,a+(k-1)*h,a+k*h);
   end;
elseif strcmp(metodo,'Trapecio')
   for k=1:m
      ys=ys+trapecio(f,a+(k-1)*h,a+k*h);
   end;
elseif strcmp(metodo,'Boole')
   for k=1:m
      ys=ys+boole(f,a+(k-1)*h,a+k*h);
   end;
elseif strcmp(metodo,'Pmedio')
   for k=1:m
      ys=ys+pmedio(f,a+(k-1)*h,a+k*h);
   end;
elseif strcmp(metodo,'Abierta2')
   for k=1:m
      ys=ys+intab2(f,a+(k-1)*h,a+k*h);
   end;
elseif strcmp(metodo,'Abierta3')
   for k=1:m
      ys=ys+intab3(f,a+(k-1)*h,a+k*h);
   end;
elseif strcmp(metodo,'Abierta4')
   for k=1:m
      ys=ys+intab4(f,a+(k-1)*h,a+k*h);
   end;
else
   disp('Metodo incorrecto');
   return;
end;
```

♦

5.3 Técnicas avanzadas de cuadratura o integración numérica

5.3.1 Método de extrapolación de Richardson

Supuesto que se desea calcular un valor v_f dependiente de una función utilizando valores de f en puntos que dependen de un valor h y que es posible establecer:

$$v_f = \underbrace{F(h)}_{Método\,o\,fórmula} + \underbrace{c_1 h^p + O(h^q)}_{E_d} \tag{5.6}$$

Donde $q > p$, c_1 no depende de h y $O(h^p)$ es el orden de error del método.

Aplicándole el método con $\frac{h}{2}$, tenemos un nuevo valor estimado:

$$v_f = F(h/2) + c_1 (h/2)^p + O(h^q)$$

Restando la primera de la segunda:

$$2^p v_f - v_f = 2^p F(h/2) - F(h) + O(h^q)$$

Despejando v_f:

$$\boxed{v_f = \frac{2^p F(h/2) - F(h)}{2^p - 1} + O(h^q)} \tag{5.7}$$

En esto es lo que consiste el método de Richardson, que se puede aplicar de forma sucesiva, y también con la derivación numérica y problemas diferenciales.

El error que se produce al estimar el valor por este método es menor, ya que si por ejemplo $h = 0.01$, para el caso sin extrapolación con $h/2$ el error de discretización es $E_d \propto 10^{-4}$, mientras que tras aplicarle la extrapolación queda $E_d \propto 10^{-6}$ si $q = 3$.

Para la aplicación recurrente simplemente se define

$$F_1(h) = \frac{2^p F(h/2) - F(h)}{2^p - 1} \rightarrow F_{k+1}(h) = \frac{2^p F_k(h/2) - F_k(h)}{2^p - 1} \qquad k = 1 : m$$

Ejemplo 5.4:

Calcular la integral

$$I = \int_{-1}^{1} \left(1 - x^2\right) \left(\cos x + x^2 \ln(2 - x)\right) dx$$

Usando $h = 0.5, h = 0.25, h = 0.125$.

Se tiene para la fórmula de Simpson:

$$I = S(h) + C_1 h^4 + C_2 h^6 + \ldots$$

Calculando las integrales para cada uno de los valores de h usando el método de Simpson:

$$S(0.5) = \frac{0.5}{3}\left(f(-1) + 4f(-0.5) + 2f(0) + 4f(0.5) + f(1)\right) = 1.37613537522150$$
$$S(0.25) = \ldots = 1.37440280250285$$
$$S(0.125) = \ldots = 1.37413646808619$$

La primera aplicación del método de Richardson nos lleva a

$$S_1(h = 0.5) = \frac{2^4 S(0.25) - S(0.5)}{2^4 - 1} = 1.37428729765484$$
$$S_1(h = 0.25) = \frac{2^4 S(0.125) - S(0.25)}{2^4 - 1} = 1.374118712458$$

Y aplicandolo una segunda vez

$$S_2(0.5) = \frac{2^6 S_1(0.25) - S_1(0.5)}{63} = 1.37411603650$$

El valor obtenido mediante la fórmula de Barrow es $I = 1.374114920$.

Se observa como se aproxima cada vez más el valor a medida que se profundiza más.

■

5.3.2 Método de Romberg

Este método consiste en aplicar la extrapolación de Richardson a la fórmula de los trapecios compuesta de forma sistemática. Para dicha fórmula, se puede establecer el siguiente resultado:

$$\int_a^b f(x)\,dx = \underbrace{\frac{h}{2}\left(f(x_0) + 2f(x_1) + \ldots + 2f(x_{n-1}) + f(x_n)\right)}_{T(h)} + \underbrace{c_1 h^2 + c_2 h^4 + \ldots c_m h^{2m} + \ldots}_{E_d}$$

Donde los c_i son independientes de h. $T(h)$ es lo que en el apartado anterior llamamos $F(h)$. Así pues, aplicando sucesivamente la extrapolación de Richardson:

$$T_1(h) = \frac{2^2 T(h/2) - T(h)}{2^2 - 1} + O\left(h^4\right)$$

$$T_2(h) = \frac{2^4 T_1(h/2) - T_1(h)}{2^4 - 1} + O\left(h^6\right)$$

$$T_i(h) = \frac{4^i T_{i-1}(h/2) - T_{i-1}(h)}{4^i - 1} = T_{i-1}(h/2) + \frac{T_{i-1}(h/2) - T_{i-1}(h)}{4^i - 1} \tag{5.8}$$

La fórmula (5.8) se conoce como esquema de Romberg, y se aplica cogiendo un h, aplicando trapecios a una integral, luego se aplica sucesivamente con $\frac{h}{2}$, $\frac{h}{4}$, ...

$$\begin{array}{ccccc}
T(h) & & & & \\
 & T_1(h) & & & \\
T(h/2) & & T_2(h) & & \\
 & T_1(h/2) & & T_3(h) & \\
T(h/4) & & T_2(h/2) & & \\
 & T_1(h/4) & & & \\
T(h/8) & & & &
\end{array}$$

El valor de $T_3\,(h)$ es el mejor resultado de todos.

Este método converge al resultado y es numéricamente estable, lo que se puede demostrar, aunque no vamos a hacerlo.

5.3.3 Algoritmos recursivos con estimación de error

En los algoritmos vistos hasta ahora, como el de extrapolación de Richardson, para hacer la recursión había que repetir las evaluaciones de la función en algunos puntos, lo que es realmente innecesario, por lo que vamos a ver cómo podemos evitar tener que hacerlo.

En los algoritmos recursivos con estimación del error se aplican recursivamente las fórmulas compuestas evitando evaluaciones de la función innecesarias (porque ya se han hecho), y estimando el error de discretización.

Supongamos que hemos estimado una integral con una fórmula de cuadratura numérica y un h determinado. Si queremos hacerlo con $h/2$, no volvemos a evaluar la función en los puntos en los que ya lo hemos hecho, lo que conseguiremos expresando los resultados de la fórmula para $h/2$ en función de los resultados para la fórmula con h.

Por ejemplo, si tenemos $n+1$ puntos con un h y queremos calcular una integral por Simpson:

$$S_{n+1} = \frac{h}{3}\left(f\left(x_0\right) + 4f\left(x_1\right) + 2f\left(x_2\right) + \ldots + 2f\left(x_{n-2}\right) + 4f\left(x_{n-1}\right) + f\left(x_n\right)\right)$$

Al usar $h/2$, se subdivide el intervalo en $2n+1$ puntos, y las evaluaciones de la función que ya hemos hecho no las repetimos. Notando $S_{n+1}^{(k)}$ a los sumandos que tienen el número k como factor ($kf\left(x_i\right)$):

$$S_{n+1} = S_{n+1}^{(1)} + S_{n+1}^{(2)} + S_{n+1}^{(4)}$$

Para $2n+1$ tenemos:

$$\begin{aligned} S_{2n+1} &= \frac{h}{6}\left(f\left(x_0\right) + 4f\left(x_0 + \frac{h}{2}\right) + 2f\left(x_1\right) + 4f\left(x_1 + \frac{h}{2}\right) + \ldots + \right. \\ &\quad \left. + 2f\left(x_{n-1}\right) + 4f\left(x_{n-1} + \frac{h}{2}\right) + f\left(x_n\right)\right) \\ &= S_{2n+1}^{(1)} + S_{2n+1}^{(2)} + S_{2n+1}^{(4)} \end{aligned}$$

$$S_{2n+1}^{(1)} = \frac{S_{n+1}^{(4)}}{2}$$

$$S_{2n+1}^{(2)} = \frac{2S_{n+1}^{(2)} + S_{n+1}^{(4)}}{4}$$

$$S_{2n+1}^{(4)} = \frac{4h}{6}\left(f\left(x_0 + \frac{h}{2}\right) + f\left(x_1 + \frac{h}{2}\right) + \ldots + f\left(x_{n-1} + \frac{h}{2}\right)\right)$$

Los únicos puntos en que se evalúa f son aquellos que nos permiten calcular $S_{2n+1}^{(4)}$.

5.3.3.1 Estimación del error:

$$I = S_{n+1} + E_d \Rightarrow E_d = \frac{-h^4}{180}(b-1) f^{(4)}(c)$$

$$I \simeq S_{2n+1} + \frac{1}{16}E_d \Rightarrow E_d = \frac{h^4}{16 \cdot 180}(b-a) f^{(4)}(\eta)$$

Suponiendo que $f^{(4)}(c) \simeq f^{(4)}(\eta)$.

Si tomamos S_{2n+1} con E_d su estimación del error, estamos siendo muy conservadores, ya que su estimación de error suele ser $\frac{1}{16}E_d$. Podemos repetir el proceso hasta que tengamos la precisión requerida.

$$|E_d| \simeq \frac{16}{15}|S_{2n+1} - S_{n+1}| \tag{5.9}$$

Normalmente al estimar el error en un algoritmo, se usa esta expresión para dar un margen de confianza, por la aproximación anterior de $f^{(4)}(c) \simeq f^{(4)}(\eta)$.

5.3.4 Métodos adaptativos

La idea base de nuevo es no utilizar más puntos de evaluación que los necesarios para obtener la precisión requerida, de forma que no se realizan evaluaciones innecesarias o inútiles.

Se divide sucesivamente el intervalo de integración, aplicando la estimación de error y continuando la subdivisión en aquellos intervalos en los que no se obtenga la precisión buscada. Son necesarias además técnicas para detectar o prevenir invalidaciones del método como subintervalos muy pequeños o errores de redondeo grandes.

Esta técnica nos sirve con funciones que tengan regiones de grandes variaciones de su valor y otras regiones con pequeñas variaciones, ya que una fórmula compuesta obtendrá buenos resultados en las regiones de grandes variaciones si h es muy pequeño, pero con un coste computacional elevado para las regiones con pequeñas variaciones, mientras que con un h grande, el valor del resultado en las regiones con grandes variaciones será malo.

Así pues, se aplica la estimación de error en subintervalos para diferenciar regiones con grandes variaciones de regiones con pequeñas variaciones, profundizando más en las regiones con más variaciones en el valor de la función.

Supongamos que queremos aproximar $\int_a^b f(x)\,dx$ con una precisión especificada ε. Aplicamos Simpson con una longitud de paso $h = (b-a)/2$, y obtenemos la siguiente fórmula:

$$\int_a^b f(x)\,dx = S(a,b) - \frac{h^5}{90}f^{(4)}(\mu) \tag{5.10}$$

Para algún $\mu \in (a,b)$. Ahora estimamos la precisión de nuestra aproximación, pero de forma que no tengamos que calcular $f^{(4)}(\mu)$, para lo cual aplicamos la fórmula compuesta de Simpson con $n = 4$ y longitud de paso $(b-a)/4 = h/2$:

$$\begin{aligned}\int_a^b f(x)\,dx &= \frac{h}{6}\left(f(a) + 4f\left(a+\frac{h}{2}\right) + 2f(a+h) + 4f\left(a+\frac{3h}{2}\right) + f(b)\right) \\ &\quad - \left(\frac{h}{2}\right)^4 \frac{b-a}{180} f^{(4)}(\mu')\end{aligned} \tag{5.11}$$

Simplificando la notación:

$$S\left(a,\tfrac{a+b}{2}\right)=\tfrac{h}{6}\left(f\left(a\right)+4f\left(a+\tfrac{h}{2}\right)+f\left(a+h\right)\right)$$

$$S\left(\tfrac{a+b}{2},b\right)=\tfrac{h}{6}\left(f\left(a+h\right)+4f\left(a+\tfrac{3h}{2}\right)+f\left(b\right)\right)$$

Si podemos suponer que $\mu \simeq \mu'$ sin riesgo a equivocarnos mucho (o mejor $f^{(4)}\left(\mu\right) \simeq f^{(4)}\left(\mu'\right)$), igualando (5.10) a (5.11) y usando las simplificaciones de notación:

$$S\left(a,\frac{a+b}{2}\right)+S\left(\frac{a+b}{2},b\right)-\frac{1}{16}\frac{h^5}{90}f^{(4)}\left(\mu\right)\simeq S\left(a,b\right)-\frac{h^5}{90}f^{(4)}\left(\mu\right)$$

Por tanto:

$$\frac{h^5}{90}f^{(4)}\left(\mu\right)\simeq\frac{16}{15}\left(S\left(a,b\right)-S\left(a,\frac{a+b}{2}\right)-S\left(\frac{a+b}{2},b\right)\right)$$

Aplicando esta estimación de error, podemos escribir:

$$\left|\int_a^b f\left(x\right)dx-S\left(a,\frac{a+b}{2}\right)-S\left(\frac{a+b}{2},b\right)\right|\simeq\frac{1}{15}\left|S\left(a,b\right)-S\left(a,\frac{a+b}{2}\right)-S\left(\frac{a+b}{2},b\right)\right|$$

Lo que implica que el uso de la fórmula compuesta aproxima el valor de la integral unas 15 veces mejor que la fórmula simple. Por tanto, si:

$$\left|S\left(a,b\right)-S\left(a,\frac{a+b}{2}\right)-S\left(\frac{a+b}{2},b\right)\right|<15\varepsilon$$

entonces

$$\left|\int_a^b f\left(x\right)-S\left(a,\frac{a+b}{2}\right)-S\left(\frac{a+b}{2},b\right)\right|<\varepsilon$$

Si no ocurriese así, en lugar de aplicar de nuevo la regla compuesta para todo el intervalo, miraríamos si es suficientemente precisa para uno de los subintervalos y usando la mitad del error, en lugar de ε.

Esta técnica es la que se usa hoy en día para estimar valores de integrales con un error determinado y haciendo el menor número de cálculos posible. Matlab lo utiliza para aproximar una integral por Simpson al llamar a la función `quad8`.

Veamos a continuación un algoritmo que nos permite implementar la integración adaptativa en Matlab, usando los algoritmos anteriormente vistos para integración simple y compuesta.

Algoritmo 5.20: Algoritmo para integración adaptativa

```
% Algoritmo para implementar la integración adaptativa
% con cualquier método de integración (compuesta+simple)
%
% Uso:
%  sol = intadapt(funcion,a,b,tolerancia,metodo,maxiter)
%
%  metodo puede ser: Simpson, Trapecio, Simpson38, Boole,
```

```
%  Pmedio, Abierta2, Abierta3, Abierta4
% Si no se especifica metodo, aplica Simpson por defecto
% Si no se indica tolerancia se usa 1e-5
% Con maxiter se evita hacer demasiadas recursiones
function sol = intadapt(f,a,b,tol,metodo,maxiter)
if nargin < 6, maxiter = 10000; end;
if nargin < 5, metodo = 'Simpson'; end;
if nargin < 4, tol = 1e-5; end;
if nargin < 3, error('Faltan argumentos de entrada'); end;
maxiter = maxiter - 1;
if (maxiter = 0), error('Maximo de iteraciones alcanzado'); end;
% Usando el algoritmo de integración compuesta podemos
% hacer las dos integrales
solsimple = intcomp(f,a,b,1,metodo);
solcomp = intcomp(f,a,b,2,metodo);
err = abs((solsimple - solcomp)/solcomp);% Error relativo
if (err < tol),
   sol = solcomp;
else
   c = (b+a)/2;
   sol = intadapt(f,a,c,tol,metodo,maxiter)
         + intadapt(f,c,b,tol,metodo,maxiter);
end;
```

♦

5.3.5 Fórmulas de Gauss o de cuadratura superexactas

Se puede hacer una elección especial de los puntos x_i y los coeficientes A_i de forma que el grado de exactitud de la fórmula $\sum_{k=0}^{n} A_k f(x_k)$ sea mayor que n. Los puntos elegidos para que se cumpla suelen ser puntos "rarillos".

Teorema 5.2. Para que una fórmula $\sum_{k=0}^{n} A_k f(x_k)$ sea exacta para todo polinomio de grado a lo sumo $n+k$ con $k \geq 1$ (condición necesaria y suficiente), debe ser interpolatoria y además se cumple:

$$\int_a^b w(x)\, x^j \theta_{n+1}(x)\, dx = 0 \qquad j = 0 : k-1 \qquad \theta_{n+1}(x) = (x-x_0)\cdots(x-x_n)$$

Como tiene que ser exacta para $x^j\theta_{n+1}(x)$, que es un polinomo de grado $n+1+j$, el grado de exactitud es, efectivamente, $n+k$.

Teorema 5.3. No existe ninguna fórmula del tipo $\sum_{i=0}^{n} A_i f(x_i)$ exacta para todos los polinomios de grado $2n+2$.

Demostración. *Supongamos que una fórmula es de grado de exactitud* $2n+2$*:*

θ_{n+1}^2 *es de grado* $2n+2$.

$$\int_a^b w(x)\, \theta_{n+1}^2(x)\, dx = \sum_{i=0}^{n} A_i \theta_{n+1}^2(x_i) = 0$$

Sin embargo, $w(x)\,\theta_{n+1}^2(x)$ es un número positivo, sólo nulo en un número finito de puntos, por lo que la integral no puede ser nula.

Se parte de un supuesto incorrecto, por tanto, no existe ninguna fórmula de grado de exactitud $2n+2$.

Existen, sin embargo, fórmulas de grado de exactitud $2n+1$, que es el máximo posible, son las fórmulas de Gauss.

El grado de exactitud es $n+k=2n+1$. Los coeficientes a_i del polinomio

$$\theta_{n+1}(x) = x^{n+1} + a_1 x^n + \ldots + a_{n+1}$$

se pueden calcular con el siguiente sistema lineal

$$\sum_{i=1}^{n+1} a_i \int_a^b w(x)\, x^{n+1-i} x^j dx = -\int_a^b w(x)\, x^{n+1} x^j dx \qquad j=0:n \Rightarrow k=n+1 \tag{5.12}$$

Los A_i para la fórmula interpolatoria se calculan con la siguiente fórmula:

$$A_i = \int_a^b w(x)\, l_i(x)\, dx$$

O por el método de los coeficientes indeterminados.

El polinomio $\theta_{n+1}(x)$ se puede obtener también utilizando la recurrencia del siguiente teorema:

Teorema 5.4. Existe una sucesión de polinomios $p_i \quad i=0:n$ que verifica:

$$\begin{gathered} grado(p_n) = n \\ \langle q, p_n \rangle = 0 \end{gathered}$$

para todo polinomio q de grado menor que n.

Definiendo el producto escalar de la siguiente forma:

$$\langle f, g \rangle = \int_a^b w(x)\, f(x)\, g(x)\, dx$$

Dicha sucesión de polinomios satisface la siguiente relación de recurrencia de tres términos:

$$p_{j+1}(x) = \alpha_j (x - \beta_j)\, p_j(x) - \gamma_j p_{j-1}(x) \tag{5.13}$$

siendo $p_{-1}(x) = 0$, $p_0(x) = C_0$ y:

$$\alpha_j = \frac{C_{j+1}}{C_j} \qquad \beta_j = \frac{\langle p_j, x p_j \rangle}{\langle p_j, p_j \rangle} \qquad \gamma_j = \frac{\alpha_j \langle p_j, p_j \rangle}{\alpha_{j-1} \langle p_{j-1}, p_{j-1} \rangle} \tag{5.14}$$

con $C_j \neq 0$ el coeficiente principal del polinomio p_j.

$$\theta_{n+1}(x) = \frac{p_{n+1}(x)}{C_{n+1}} \tag{5.15}$$

Teorema 5.5. Si p_n es un polinomio ortogonal a todos los de grado inferior sobre un intervalo $[a,b]$ y para la función peso w, todas sus raíces son reales, simples e interiores al intervalo (a,b).

Demostración. *Supongamos que todas las raíces $t_i \in [a,b]$ del polinomio son k raíces de multiplicidad impar, todas distintas e incluidas en el intervalo. Considerando el polinomio*

$$\prod_{i=1}^{k}(x-t_i)\,p_n(x)$$

e integrándolo

$$I=\int_a^b w(x)\prod_{i=1}^{k}(x-t_i)\,p_n(x)$$

Las k raíces se convierten en raíces de multiplicidad par ya que van multiplicadas por $(x-t_i)$.

Si $k<n$, $I=0$ ya que p_n es ortogonal a todos los polinomios de grado inferior al suyo. En cambio, si $k=n$, $I\neq 0$, de forma que todas las raíces son diferentes, y por tanto, simples y reales.

✏ El grado de exactitud $2n+1$ se obtiene al calcular los coeficientes A_i mediante el método de los coeficientes indeterminados u otro método que asegure que la fórmula es interpolatoria.

5.3.5.1 Procedimiento para deducir la fórmula superexacta

1. En primer lugar tenemos el polinomio

$$\theta_{n+1}(x)=x^{n+1}+a_1x^n+\ldots+a_{n+1}$$

2. Y se obtienen los coeficientes del siguiente sistema de ecuaciones lineales para $j=0:n$

$$a_1\int_a^b w(x)\,x^n x^j dx+a_2\int_a^b w(x)\,x^{n-1}x^j dx+\ldots+a_{n+1}\int_a^b w(x)\,x^j dx=-\int_a^b w(x)\,x^{n+1}x^j dx$$

   Se puede demostrar que este sistema tiene solución y es única. Se puede obtener también $\theta_{n+1}(x)$ con la recurrencia antes mencionada.

3. Se resuelve el sistema determinando el polinomio $\theta_{n+1}(x)$, ortogonal a todos los x^j
4. Se obtienen los x_i como raíces de $\theta_{n+1}(x)=0$
5. Se calculan los A_i por el método de los coeficientes indeterminados (por ejemplo), para obtener la fórmula interpolatoria

Si el polinomio $\theta_{n+1}(x)$ es ortogonal a todos los de grado inferior en $[a,b]$, sus raíces son reales y pertenecientes al intervalo (a,b). Esta es una propiedad de todos los polinomios ortogonales.

Los coeficientes A_i de las fórmulas gaussianas son todos positivos.

Dado que los polinomios son "rarillos", sus raíces también lo son, y con ello los coeficientes A_i, razón por la cual las fórmulas superexactas se suelen dar con tablas.

Existe otra alternativa para calcular los coeficientes de las fórmulas superexactas, partiendo de los polinomios ortogonales $p_j(x)$, y siendo C_j el coeficiente principal del polinomio $p_j(x)$, con la expresión

$$A_i=-\frac{C_{n+2}\,\langle p_{n+1},p_{n+1}\rangle}{C_{n+1}p'_{n+1}(x_i)\,p_{n+2}(x_i)}$$

5.3.5.2 Error de discretización

El error en la fórmula de cuadratura superexacta se obtiene por la fórmula:

$$E_{n+1}(f) = \frac{f^{(2n+2)}(c)}{(2n+2)!}\int_a^b w(x)\left(\theta_{n+1}(x)\right)^2 dx \tag{5.16}$$

5.3.5.3 Fórmulas de Gauss

En la tabla (5.4) se pueden ver las fórmulas superexactas de Gauss más usadas, con las funciones peso usadas y el intervalo sobre el que se aplican. Se han omitido los términos de error.

	Peso	Intervalo	Integral
Gauss-Legendre	$w(x) = 1$	$[-1, 1]$	$\int_{-1}^{1} f(x)\,dx \simeq \sum_{k=1}^{N} \frac{2}{\left(1-x_k^2\right)\left(P'_N(x_k)\right)^2} f(x_k)$
Gauss-Chebyshev	$w(x) = \frac{1}{\sqrt{1-x^2}}$	$[-1, 1]$	$\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}dx \simeq \frac{\pi}{N}\sum_{k=1}^{N} f\left(\cos\left(\frac{2k-1}{2N}\pi\right)\right)$
Gauss-Laguerre	$w(x) = e^{-x}$	$[0, \infty)$	$\int_0^{\infty} e^{-x} f(x)\,dx \simeq \sum_{k=1}^{N} \frac{(N!)^2}{x_k\left(L'_N(x_k)\right)^2} f(x_k)$
Gauss-Hermite	$w(x) = e^{-x}$	$(-\infty, \infty)$	$\int_{-\infty}^{\infty} e^{-x^2} f(x)\,dx \simeq \sum_{k=1}^{N} \frac{2^{N+1}N!\sqrt{\pi}}{\left(H'_N(x_k)\right)^2} f(x_k)$

Tabla 5.4: Fórmulas de Gauss

5.3.5.4 Características de las fórmulas de Gauss

1. Son interpolatorias y abiertas
2. Sus puntos y coeficientes de cuadratura son, en general, números irracionales
3. Sus coeficientes son positivos
4. Su grado de exactitud es $2n + 1$ con $n + 1$ evaluaciones de la función a integrar. Por consiguiente, en general (no siempre), una fórmula superexacta de Gaus será mucho mejor que cualquier otra fórmula con el mismo número de puntos.
5. Su convergencia al valor real de la integral puede demostrarse si el integrando es una función continua, como se ve en el siguiente teorema

Teorema 5.6. Si tenemos una función f continua en un intervalo $[a, b]$ y fijando tanto el intervalo como la función peso y usando sucesivas fórmulas de Gauss para $n \in 0, 1, 2, 3, \ldots$, tal que

$$\int_a^b w(x) f(x) dx \simeq \sum_{i=0}^{n} A_{in} f(x_{in})$$

para $n \geq 0$, entonces las aproximaciones de las fórmulas de Gauss convergen al valor de la integral conforme $n \to \infty$.

Esto que acabamos de ver no es, en general, cierto para otras fórmulas de cuadratura que no sean las superexactas de Gauss.

Existen unas fórmulas llamadas **pseudogaussianas** de dos tipos, que se pueden deducir de manera similar a las gaussianas:

- ✏ De Radau: se hace $x_0 = a$ o bien $x_n = b$, y es de grado de exactitud $2n$ como máximo.
- ✏ De Lobatto: se hace $x_0 = a$ y $x_n = b$. Su grado de exactitud es $2n - 1$ como máximo.

Ejemplo 5.5:

Queremos la fórmula de Lobatto para estimar

$$\int_{-1}^{1} f(x) dx \simeq A_0 f(-1) + A_1 f(x_1) + A_2 f(x_2) + A_3 f(1)$$

Entonces es

$$\theta_4(x) = (x+1)(x-1)\left(x^2 + ax + b\right)$$

Se fuerzan las raíces 1 y -1 para $\theta_4(x)$.

■

Hay otros métodos adaptativos que utilizan fórmulas de Gauss, como el método de Gauss-Kronrod, donde la estimación del error se basa en la evaluación en puntos especiales, denominados "puntos de Kronrod".

5.3.5.5 Tablas para las fórmulas de Gauss

A continuación vamos a incluir las tablas de algunas de las fórmulas de Gauss, concretamente la de Hermite, Laguerre y Legendre para $N = 2$ hasta $N = 5$, siendo N el número de puntos a considerar. No se incluyen las tablas para la fórmula de Chebyshev porque sus puntos y coeficientes se pueden obtener analíticamente como

$$A_i = \frac{\pi}{N} \qquad x_i = \cos\left(\frac{2i-1}{2N}\pi\right)$$

Recordemos que las fórmulas de Legendre y Hermite se aplican para funciones en el intervalo $[-1, 1]$ y $(-\infty, \infty)$, por lo que se presentan en las tablas sólo la parte positiva de los intervalos, que habrá que reproducir para la parte negativa, conservándose el signo de los coeficientes A_i.

Ejemplo 5.6:

Determinar la fórmula de Gauss de tres puntos para

N	x_i	A_i
2	0.707106781186547244	0.88622692545275801365
3	1.224744871391589049 1	0.29540897515091933788
	0	1.1816359000551283651
4	1.6506801238857845559	0.081312835447245177143
	0.5246476232752903178 8	0.80491409000551283651
5	2.0201828704560856329	0.019953242059045913208
	0.9585724646138185071 1	0.39361932315224115983
	0	0.94530882048294418812 3

Tabla 5.5: Puntos y coeficientes de las fórmulas de Gauss-Hermite hasta $N = 5$

N	x_i	A_i
2	0.58578643762690495120	0.85355339059327376220
	3.41421356237309504 88	0.14644660940672623780
3	0.41577455678347908331	0.71109300992917301545
	2.29428036027904171 98	0.27851773356924084880
	6.28994508293747919 69	0.01038925650158613574 9
4	0.32254768961939231180	0.60315410434163360164
	1.74576110115834657 57	0.35741869241779968664
	4.53662029692112798 33	0.03888790851500538427 2
	9.39507091230113312 92	0.00053929470556132745010
5	0.26356031971814091020	0.52175561058288065248
	1.41340305910651679 22	0.39866811083175927 45
	3.59642577104072208 12	0.07594244968170759538 8
	7.08581000585883755 69	0.00361175867992204845 45
	12.6408008442757826 59	0.00002336997238577622789 1

Tabla 5.6: Puntos y coeficientes de las fórmulas de Gauss-Laguerre hasta $N = 5$

$$\int_{-1}^{1} \frac{f(x)}{1+x^2} dx \simeq A_0 f(x_0) + A_1 f(x_1) + A_2 f(x_2)$$

y utilizarla para calcular la integral

$$I = \int_1^2 \frac{x^4}{(x-1.5)^2 + 0.25} dx$$

estimando el error cometido.

Solución:

$$\theta_3(x) = (x-x_0)(x-x_1)(x-x_2) = x^3 + a_1 x^2 + a_2 x + a_3$$

Planteando el sistema de ecuaciones para hallar los a_i:

$$\int_{-1}^{1} \frac{x^j}{1+x^2} \theta_3(x)\, dx = 0 \qquad j = 0, 1, 2$$

$$a_1 \int_a^b \frac{x^{j+2}}{1+x^2} dx + a_2 \int_a^b \frac{x^{j+1}}{1+x^2} dx + a_3 \int_a^b \frac{x^j}{1+x^2} dx = \int_{-1}^{1} \frac{x^{j+3}}{1+x^2} dx$$

N	x_i	A_i
2	0.57735026918962576451	1
3	0.77459666924148337704	0.55555555555555555556
	0	0.88888888888888888889
4	0.86113631159405257523	0.34785484513745385737
	0.33998104358485626480	0.65214515486254614263
5	0.90617984593866399280	0.23692688505618908751
	0.53846931010568309104	0.47862867049936646804
	0	0.56888888888888888889

Tabla 5.7: Puntos y coeficientes de las fórmulas de Gauss-Legendre hasta $N = 5$

Nos queda el siguiente sistema:

$$\begin{pmatrix} 2-\frac{\pi}{2} & 0 & \frac{\pi}{2} \\ 0 & 2-\frac{\pi}{2} & 0 \\ \frac{2}{3}-\left(2-\frac{\pi}{2}\right) & 0 & 2-\frac{\pi}{2} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 0 \\ -\frac{2}{3}+\left(2-\frac{\pi}{2}\right) \\ 0 \end{pmatrix}$$

Resolviendo el sistema se obtienen las siguientes soluciones:

$$\begin{gathered} a_1 = a_3 = 0 \\ a_2 = \frac{-\frac{2}{3}+2-\frac{\pi}{2}}{2-\frac{\pi}{2}} \end{gathered}$$

El polinomio queda por tanto:

$$\theta_3(x) = x^3 + a_2 x$$

Y sus raíces:

$$\begin{gathered} x_1 = 0 \\ x_2 = \sqrt{-a_2} \\ x_0 = -\sqrt{-a_2} \end{gathered}$$

Usando el método de los coeficientes indeterminados para hallar los A_i:

$$\int_{-1}^{1} \frac{1}{1+x^2} dx = A_0 + A_1 + A_2 = \frac{\pi}{2}$$

$$\int_{-1}^{1} \frac{x}{1+x^2} dx = A_0 x_0 + A_1 \cdot 0 - A_2 x_0 = 0$$

$$\int_{-1}^{1} \frac{x^2}{1+x^2} dx = A_0 x_0^2 + A_1 \cdot 0 + A_2 x_0^2 = 2 - \frac{\pi}{2}$$

$$\begin{gathered} A_0 = A_2 = \frac{4-\pi}{4x_0^2} \\ A_1 = \frac{\pi}{2} - 2A_0 \end{gathered}$$

Para aplicar la fórmula a la integral pedida, primero hay que cambiar el intervalo de integración, lo cual lo hacemos mediante el cambio de variable $t = 2x - 3$:

$$\begin{aligned} I = \int_1^2 \frac{x^4}{(x-1.5)^2 + 0.25} dx &= \frac{1}{8} \int_{-1}^1 \frac{(t+3)^4}{t^2+1} dt \\ &\simeq \frac{1}{8} \left(A_0 (x_0+3)^4 + (-x_0+3)^4 + A_1 3^4 \right) \end{aligned}$$

Si en el denominador no hubiésemos tenido lo mismo que en la fórmula de integración (que es el peso), basta con multiplicar y dividir por ese peso.

Para estimar el error, tendríamos que usar la fórmula (5.16), pero observamos que la función $f(x)$ que está en el numerador es un polinomio de orden 4, y la fórmula es de grado de exactitud 5, por lo que no hay error para esta función (la fórmula es exacta).

■

Ejemplo 5.7:

Veamos un ejemplo del uso de las tablas para las fórmulas de Gauss, concretamente para Gauss-Legendre con $N = 5$.

$$\begin{aligned} \int_{-1}^1 f(x)\, dx \simeq\ & 0.23692\ldots \cdot f(-0.90617\ldots) + \\ & +0.4786\ldots \cdot f(-0.5384\ldots) + \\ & +0.5688\ldots \cdot f(0) + \\ & +0.4786\ldots \cdot f(0.5384\ldots) + \\ & +0.23692\ldots \cdot f(0.90617\ldots) \end{aligned}$$

se observa que son simétricas, razón por la cual en las tablas sólo se muestran los valores absolutos de los puntos.

■

Ejemplo 5.8:

A partir de una fórmula de Gauss en un intervalo estándar, obtener la fórmula para otro intervalo cualquiera. Tenemos la siguiente fórmula

$$\int_a^b w(x) f(x)\, dx = \sum_{i=0}^n A_i f(x_i) + E_{n+1}$$

Haciendo el cambio $x = \alpha t + \beta$ se obtiene

$$\int_c^d w(\alpha t + \beta) f(\alpha t + \beta) \alpha dt = \alpha \int_c^d w^*(t) g(t)\, dt = \alpha \left(\sum_{i=0}^n A_i^* g(t_i) + E_{n+1}^* \right)$$

Por tanto, tendríamos que aplicar la fórmula para

$$x_i = \alpha t_i + \beta \qquad A_i = \alpha A_i^*$$

Esta forma de hacerlo es rápida si se usan las tablas.

■

5.4 Integración impropia

Vamos a plantear las dificultades en el cálculo que plantea la resolución de este tipo de problemas, suponiendo de antemano que la integral existe.

✏ Función discontinua de salto finito: si la función es discontinua en un punto $c \in [a, b]$, se puede dividir el intervalo de integración en dos:

$$\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx$$

Si usamos una fórmula cerrada, no tenemos forma de coger un valor concreto para $f(c)$ al evaluar la integral, así que hacemos $f(c) = f(c^-)$ para el primer subintervalo y $f(c) = f(c^+)$ para el segundo. Si usamos fórmulas abiertas no hay problema, de forma que es una opción más adecuada.

✏ $\lim_{x\to a} f(x) = \infty$

$$\int_a^b f(x)\,dx \simeq \int_{a+\varepsilon}^b f(x)\,dx$$

con un ε suficientemente pequeño y empleando una fórmula cerrada.
También se pueden emplear fórmulas abiertas que no tengan en cuenta el extremo afectado, pero habrá que usar un h pequeño para que no se cometa demasiado error.

✏ Intervalos infinitos

$$\int_a^\infty f(x)\,dx$$

Hay varias opciones:

1. Usar Gauss-Laguerre, cambiando el extremo inferior del intervalo de integración: $x = t + a$.
2. Aproximar el valor mediante $\int_a^M$ con un M suficientemente grande, de forma que $\left|\int_M^\infty f(x)\,dx\right| < \varepsilon$.
3. Hacer un cambio de variable $x = \frac{1}{t}$, con lo que queda:

$$\int_{\frac{1}{a}}^0 \left(-\frac{1}{t^2}\right) f\left(\frac{1}{t}\right) dt = \int_0^{1/a} \frac{f(1/t)}{t^2} dt$$

5.5 Integración múltiple

$$\int_a^b \int_c^d f(x, y)\,dy dx$$

El recinto de integración es un rectángulo, por lo que se construye una malla de puntos en los que evaluar la función.

$$\begin{array}{ll} x_i = a + ih & h = \frac{b-a}{n} \\ y_j = c + jk & k = \frac{d-c}{m} \end{array}$$

Aplicando Simpson:

$$\begin{aligned}\int_a^b \int_c^d f(x,y)\,dydx &\simeq \int_a^b \frac{k}{3}\left(f(x,y_0)+4f(x,y_1)+\ldots+4f(x,y_{n-1})+f(x,y_n)\right)dx \\ &\simeq \frac{k}{3}\left(\frac{h}{3}\left(f(x_0,y_0)+4f(x_1,y_0)+\ldots+f(x_n,y_0)\right)+\ldots\right) \\ &\simeq \frac{hk}{9}\sum_i\sum_j A_{ij}f(x_i,y_j)\end{aligned}$$

Habría que aplicar Simpson a cada una de las $\int_a^b f(x,y_i)\,dx$.

La cosa se complica cuando el recinto no es rectangular, sino que tenemos dos funciones dependientes de la otra variable independiente:

$$\int_a^b \int_{c(x)}^{d(x)} f(x,y)\,dydx$$

Ya que ahora es:

$$h = \tfrac{b-a}{n}$$

$$k(x) = \tfrac{d(x)-c(x)}{m}$$

De forma que queda:

$$\simeq \int_a^b \frac{k(x)}{3}\left(f(x,c(x))+4f(x,c(x)+k(x))+2f(x,c(x)+2k(x))+\ldots+f(x,d(x))\right)dx$$

TEMA 6

Métodos numéricos para ecuaciones diferenciales ordinarias

A lo largo de este tema vamos a considerar que los problemas a resolver tienen solución, por tanto no nos vamos a plantear su existencia. Notaremos y_{i+1} como la solución aproximada en un punto t_{i+1} y $y\left(t_{i+1}\right)$ a la solución exacta en ese mismo punto.

Resolveremos tanto problemas de valor inicial como problemas de valores de contorno.

6.1 Problema de valor inicial

En este apartado vamos a tratar de resolver de forma numérica y computacionalmente un problema de ecuaciones diferenciales con un valor inicial. Por ejemplo, una EDO (Ecuación Diferencial Ordinaria) de primer orden:

$$y' = f\left(t, y\right) \qquad y\left(a\right) = y_0$$

Y obtener la solución para $a \leq t \leq b$. Podemos tener igualmente $y\left(b\right) = y_0$ como valor inicial y aplicar los métodos hacia atrás.

También vamos a ver cómo resolver EDOs de orden n:

$$y^{(n)} = f\left(t, y, y', \ldots, y^{(n-1)}\right)$$

$$\begin{gathered} y\left(a\right) = y_{00} \\ y'\left(a\right) = y_{10} \\ \vdots \\ y^{(n-1)}\left(a\right) = y_{n-1,0} \end{gathered}$$

Esto lo podemos aplicar también para sistemas de EDOs de primer orden:

$$\begin{gathered} y_1' = f\left(t, y_1, y_2, \ldots, y_n\right) \\ \vdots \\ y_n' = f\left(t, y_1, y_2, \ldots, y_n\right) \end{gathered}$$

Que se puede expresar también de forma vectorial:

$$\vec{y}' = \vec{f}(t, \vec{y})$$

De esta forma, las EDOs de orden superior, podemos resolverlas como sistemas de EDOs de un orden menor:

$$\begin{array}{cc} u_1' = u_2 & u_1 = y \\ u_2' = u_3 & u_2 = y' \\ \vdots & \vdots \\ u_n' = u_{n+1} = f(t, u_1, \ldots, u_n) & \end{array}$$

Cuando calculemos los valores numéricos de la solución, notaremos y_i como la aproximación de la solución, $y(t_i)$.

Los métodos se basan en crear una malla de puntos dentro del intervalo $[a, b]$ de forma que aproximemos las soluciones en los puntos de esa malla.

6.2 Métodos unipaso de Taylor y de Runge-Kutta

Se da un problema de valor inicial:

$$\begin{array}{c} y' = f(t, y) \\ y(t_0) = y_0 \\ a \leq t \leq b \end{array}$$

6.2.1 Introducción a los métodos unipaso

Los **métodos de discretización** fijan una malla de puntos entre a y b y calculan una aproximación de la solución en cada uno de los puntos.

Los **métodos unipaso** "propagan" la solución de un extremo a otro del intervalo, usando únicamente la solución inmediatamente anterior. Obviamente, empiezan con el valor inicial dado por el problema. Si se propaga la solución a través de una malla de puntos $t_i \in [a, b]$, se dice que son métodos de discretización unipaso.

Se escriben de forma general:

$$y_{i+1} = y_i + h\phi(t_i, y_i, h) \tag{6.1}$$

para $i = 0, 1, 2 \ldots$ y siendo ϕ una función que depende de la función (f) del problema. Cuando se tiene la función ϕ, se puede utilizar la recurrencia para propagar la solución a partir de la condición inicial. La determinación de ϕ depende del método unipaso utilizado.

6.2.2 Error local de truncamiento

La solución exacta del problema de valor inicial, $y(t)$, no tiene porqué cumplir la recurrencia anterior, así que se define un **error local de truncamiento** en t_{i+1}, T_{i+1} con la siguiente fórmula:

$$\frac{y(t_i + h) - y(t_i)}{h} - \phi(t_i, y(t_i), h) = T_{i+1} \tag{6.2}$$

- ✏ el método es **consistente** con el problema si cuando $h \to 0$, el error en t_i tiende a cero: $\lim_{h \to 0} T_i \to 0$
- ✏ El método es **consistente de orden** p si $T_i = O(h^p)$ con $p \geq 1$

6.2.3 Método de Taylor

Supongamos que queremos aproximar el valor de $y(t_{i+1})$ por y_{i+1}. Una forma de hacerlo, sería desarrollando en series de Taylor (las derivadas son absolutas[1]):

$$y(t_{i+1}) = y(t_i + h) = y(t_i) + y'(t_i)\,h + \frac{y''(t_i)}{2!}h^2 + \ldots + \frac{y^{(p)}(t_i)}{p!}h^p + \frac{y^{(p+1)}(c_i)}{(p+1)!}h^{p+1}$$

Para algún $c_i = t_i + h \in (t_i, t_{i+1})$.

$$y(t_{i+1}) \simeq y_{i+1} = y_i + f(t_i, y_i)\,h + \frac{f'(t_i, y_i)}{2!}h^2 + \ldots + \frac{f^{(p-1)}(t_i, y_i)}{p!}h^p$$

Despreciando el resto. Vamos a usar la notación $f_i^{(k)} = f^{(k)}(t_i, y_i)$, por simplificación. Por tanto, el método de Taylor utiliza la siguiente función:

$$\phi(t_i, y_i, h) = f_i + \frac{h}{2}f_i' + \ldots + \frac{h^{p-1}}{p!}f_i^{(p-1)} \tag{6.3}$$

Y el error de truncamiento es

$$T_{i+1} = \frac{h^p}{(p+1)!}y^{(p+1)}(c_i) \tag{6.4}$$

El inconveniente más grande de este método es tener que hacer todas las derivadas de f, ya que como vemos en la nota al pie, cada derivada es más larga y penosa.

Otros métodos son:

- ✏ **Método de Euler**: es el caso más sencillo de método de Taylor, ya que únicamente usa la evaluación de la función $f(t, y)$, sin hacer ninguna derivada

$$\boxed{y_{i+1} = y_i + hf(t_i, y_i)} \tag{6.5}$$

- ✏ **Método de segundo orden**:

$$\boxed{y_{i+1} = y_i + hf(t_i, y_i) + \frac{h^2}{2}f'(t_i, y_i)} \tag{6.6}$$

6.2.4 Métodos de Runge-Kutta

En estos métodos se trata de integrar (numéricamente). Dado el problema $y' = f(t, y)$, integrando la expresión:

$$\int_{t_i}^{t_{i+1}} y'(t)\,dt = \int_{t_i}^{t_{i+1}} f(t, y)\,dt$$

Nos queda:

[1] Si la función es de dos variables:
$f' = f_x + f_y y'$
$f'' = f_{xx} + f_{xy}y' + f_y y'' + f_{yy}(y')^2$

$$y\left(t_{i+1}\right)=\underbrace{y\left(t_i\right)}_{(1)}+\underbrace{\int_{t_i}^{t_{i+1}} f\left(t,y\right)dt}_{(2)}$$

Donde (1) no lo tenemos, aunque sí una aproximación, mientras que (2) lo podemos integrar numéricamente usando $h\sum_{r=1}^{q} c_k k_r$.

$$y_{i+1}=y_i+h\sum_{r=1}^{q} c_r k_r$$

Siendo

- $k_r=f\left(\theta_r,y_r\right)$ para y_r un valor intermedio entre y_i e y_{i+1},

$$y_r=y_i+h\sum_{s=1}^{r-1} b_{r,s}k_s$$

- $\theta_r\in[t_i,t_{i+1}]$,

$$\theta_r=t_i+a_r h$$

con $a_1=0$, $0\leq a_r\leq 1$ para $r=2:q$

La dificultad ahora está en que no conocemos el valor de y en los puntos intermedios (y_r). Para obtenerlos, aplicamos la fórmula a esos puntos:

$$y_r=y_i+h\sum_{s=1}^{r-1} b_{rs}k_s$$

De forma que tenemos:

$$k_r=f\left(t_i+a_r h,y_i+h\sum_{s=1}^{r-1} b_{rs}k_s\right)\qquad r=2:q$$

y $k_1=f\left(t_i,y_i\right)$.

Los valores de a_r, c_r y b_{rs} se escogen de forma que el método sea consistente del mayor orden posible y difieren según el método. Normalmente se obtienen por anulación de los coeficientes de las potencias de h en el desarrollo en series de Taylor de T_{i+1}. En el apartado (6.2.4.1) tenemos algunos de los métodos de Runge-Kutta, notados como RK_{pq} siendo p el orden máximo posible del método y q el número de evaluaciones de la función. Con 6 evaluaciones sólo se pueden conseguir métodos de orden 5, de hecho, para $q=5,6,7$ el máximo orden es $p=q-1$, mientras que para $q\geq 8$, $p=q-2$.

6.2.4.1 Algunos métodos de Runge-Kutta

Métodos RK_{22}

✏ Método del punto medio:

$$\begin{aligned} y_{i+1} &= y_i + hk_2 \\ k_1 &= f_i \\ k_2 &= f\left(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_1\right) \end{aligned}$$

✏ Método de Euler modificado:

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{2}\left(k_1 + k_2\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + h, y_i + hk_1\right) \end{aligned}$$

✏ Método de Heun:

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{4}\left(k_1 + 3k_2\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + \tfrac{2h}{3}, y_i + \tfrac{2h}{3}k_1\right) \end{aligned}$$

Métodos RK_{33}

✏ Método de Kutta:

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{6}\left(k_1 + 4k_2 + k_3\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_1\right) \\ k_3 &= f\left(x_i + h, y_i + h\left(-k_1 + 2k_2\right)\right) \end{aligned}$$

✏ Método de Nÿstrom:

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{8}\left(2k_1 + 3k_2 + 3k_3\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + \tfrac{2h}{3}, y_i + \tfrac{2h}{3}k_1\right) \\ k_3 &= f\left(x_i + \tfrac{2h}{3}, y_i + \tfrac{2h}{3}k_2\right) \end{aligned}$$

Métodos RK_{44}

✏ Método de Runge-Kutta:

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_1\right) \\ k_3 &= f\left(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_2\right) \\ k_4 &= f\left(x_i + h, y_i + hk_3\right) \end{aligned}$$

✏ Método de Kutta:

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{8}\left(k_1 + 3k_2 + 3k_3 + k_4\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + \tfrac{h}{3}, y_i + \tfrac{h}{3}k_1\right) \\ k_3 &= f\left(x_i + \tfrac{2h}{3}, y_i + \tfrac{h}{3}\left(-k_1 + 3k_2\right)\right) \\ k_4 &= f\left(x_i + h, y_i + h\left(k_1 - k_2 + k_3\right)\right) \end{aligned}$$

Métodos RK_{56}

✏ A:

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{90}\left(7k_1 + 7k_3 + 32k_4 + 12k_5 + 32k_6\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + h, y_i + hk_1\right) \\ k_3 &= f\left(x_i + h, y_i + \tfrac{h}{2}\left(k_1 + k_2\right)\right) \\ k_4 &= f\left(x_i + \tfrac{h}{4}, y_i + \tfrac{h}{64}\left(14k_1 + 5k_2 - 3k_3\right)\right) \\ k_5 &= f\left(x_i + \tfrac{h}{3}, y_i + \tfrac{h}{96}\left(-12k_1 - 12k_2 + 8k_3 + 64k_4\right)\right) \\ k_6 &= f\left(x_i + \tfrac{3h}{4}, y_i + \tfrac{h}{64}\left(-9k_2 + 5k_3 + 16k_4 + 36k_5\right)\right) \end{aligned}$$

✏ B:

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{90}\left(7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_1\right) \\ k_3 &= f\left(x_i + \tfrac{h}{4}, y_i + \tfrac{h}{8}\left(k_1 + k_2\right)\right) \\ k_4 &= f\left(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_3\right) \\ k_5 &= f\left(x_i + \tfrac{3h}{4}, y_i + \tfrac{h}{16}\left(-3k_2 + 6k_3 + 9k_4\right)\right) \\ k_6 &= f\left(x_i + h, y_i + \tfrac{h}{7}\left(k_1 + 4k_2 + 6k_3 - 12k_4 + 8k_5\right)\right) \end{aligned}$$

Para el cálculo computacional se suele hacer

$$\begin{aligned} K_1 &= hk_1 \\ K_2 &= hk_2 \end{aligned}$$

Así, el método de Heun, por ejemplo, quedaría:

$$y_{i+1} = y_i + \frac{1}{4}\left(K_1 + 3K_2\right)$$

6.2.4.2 Error de discretización

$$T_{i+1} = \frac{y\left(t_{i+1}\right) - y\left(t_i\right)}{h} - \sum_{r=1}^{q} c_r f\left(\theta_r, y\left(\theta_r\right)\right), \qquad \theta_r = t_i + a_r h$$

Desarrollando por Taylor en t_i:

$$\underbrace{0 \cdot y\left(t_i\right)}_{y(t_i)-y(t_i)} + \underbrace{0 \cdot y'\left(t_i\right)}_{y'(t_i)-y'(t_i)} + 0 \cdot y''\left(t_i\right) + \ldots \sum_{r=1}^{q} c_r$$

De ello se extrae que para que un metodo sea consistente de orden $p \geq 1$:

$$\sum_{r=1}^{q} c_r = 1 \tag{6.7}$$

6.2.4.3 Métodos implícitos

Los métodos de Runge-Kutta que hemos visto hasta ahora se llaman explícitos, pero existen unos métodos, que no vamos a desarrollar, llamados implícitos y que implican resolver un sistema no lineal para cada punto de la malla (si f es no lineal). Estos aplican la siguiente fórmula al calcular los k_r:

$$k_r = f\left(t_i + a_r h, y_i + h\sum_{s=1}^{q} b_{rs}k_s\right)$$

Es decir, que al calcular k_r, tenemos que evaluar f en un punto que depende de k_r. Por ello, si la función f es no lineal, hay que resolver una ecuación no lineal.

Ejemplo 6.1:

Resolver el siguiente problema de valor inicial por el método de Heun:

$$xy'' + yy' + \operatorname{sen}\frac{\pi x}{6} = 0$$

Con valores iniciales $y(1) = 1$ y $y'(1) = 2$, siendo $h = 0.5$.

Solución:

Convertimos la ecuación de segundo orden en un sistema de orden inferior:

$$\begin{array}{cc} u_1 = y & u_1' = u_2 \\ u_2' = \frac{\left(-u_1u_2 - \operatorname{sen}\frac{\pi}{6}x\right)}{x} & \end{array}$$

$$\vec{u}(1) = \begin{pmatrix} \vec{u}_1(1) \\ \vec{u}_2(1) \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$y_{i+1} = y_i + \frac{h}{4}(k_1 + 3k_2)$$

$$k_1 = f_i$$

$$k_2 = f\left(x_i + \frac{2h}{3}, y_i + \frac{2h}{3}k_1\right)$$

Usando K_r en lugar de k_r:

$$\begin{pmatrix} y(1.5) \\ y'(1.5) \end{pmatrix} \simeq \vec{u}(x_1) = \begin{pmatrix} u_1(x_1) \\ u_2(x_1) \end{pmatrix} = \vec{u}(x_0 = a) + \frac{1}{4}(K_1 + 3K_2)$$

Calculando los K_r:

$$K_1 = 0.5\begin{pmatrix} 2 \\ \frac{-2-\operatorname{sen}\frac{\pi}{6}}{1} \end{pmatrix} = \begin{pmatrix} 1 \\ -1.25 \end{pmatrix}$$

Para el K_2 tenemos que usar $f\left(1 + \frac{2\cdot 0.5}{3}, \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \frac{2}{3}\begin{pmatrix} 1 \\ -1.25 \end{pmatrix}\right)$:

$$K_2 = 0.5\begin{pmatrix} 2 + \frac{2}{3}(-1.25) \\ \frac{\left(1+\frac{2}{3}\right)\left(2+\frac{2}{3}(-1.25)\right)+\operatorname{sen}\frac{4\pi}{3\cdot 6}}{\frac{4}{3}} \end{pmatrix} = \begin{pmatrix} 0.5833\ldots \\ -0.97021202029\ldots \end{pmatrix}$$

Obtenemos para el primer punto:

$$\vec{u}(x_1) = \begin{pmatrix} 1.68750 \\ 0.959840 \end{pmatrix}$$

Para el resto de puntos se procede de la misma forma.

■

6.2.5 Algoritmos en Matlab para métodos de Runge-Kutta

En este apartado incluimos los códigos en Matlab para implementar los métodos de Runge-Kutta que hemos visto antes.

Algoritmo 6.1: Método Runge-Kutta del Punto Medio para EDOs

```
% Implementacion del Metodo de Runge-Kutta de segundo orden
% Metodo del punto medio
% Uso: [t,y,dy]=edpmedio(funcion,inicio,fin,alfa,paso)
function [ts,ys,fs]=edpmedio(f,inicio,fin,alfa,h)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
   for i=1:n
      k1=feval(f,ts(i),ys(:,i));
      k2=feval(f,ts(i)+h/2,ys(:,i)+(h*k1)/2);
      fs(:,i)=k1;
      ys(:,i+1)=ys(:,i)+k2*h;
   end;
end;
```

♦

Algoritmo 6.2: Método Runge-Kutta de Euler modificado para EDOs

```
% Implementacion del Metodo de Runge-Kutta de segundo orden
% Metodo de Euler modificado
% Uso: [t,y,dy]=eulermod(funcion,inicio,fin,alfa,paso)
function [ts,ys,fs]=eulermod(f,inicio,fin,alfa,h)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
   for i=1:n
      k1=feval(f,ts(i),ys(:,i));
      k2=feval(f,ts(i)+h*(2/3),ys(:,i)+h*(2/3)*k1);
      fs(:,i)=k1;
      ys(:,i+1)=ys(:,i)+(k1+k2)*h/4;
   end;
end;
```

♦

Algoritmo 6.3: Método Runge-Kutta de Heun

```
% Implementacion del Metodo de Runge-Kutta de segundo orden
% Metodo de Heun
% Uso: [t,y,dy]=heun(funcion,inicio,fin,alfa,paso)
function [ts,ys,fs]=heun(f,inicio,fin,alfa,h)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
   for i=1:n
      k1=feval(f,ts(i),ys(:,i));
      k2=feval(f,ts(i)+h/2,ys(:,i)+(h*k1)/2);
      fs(:,i)=k1;
      ys(:,i+1)=ys(:,i)+k2*h;
   end;
end;
```

♦

Algoritmo 6.4: Método Runge-Kutta 33 (de Kutta)

```
% Implementacion del Metodo de Runge-Kutta de tercer orden
% Metodo de Kutta de tercer orden
% Uso: [t,y,dy]=kutta3(funcion,inicio,fin,alfa,paso)
function [ts,ys,fs]=kutta3(f,inicio,fin,alfa,h)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
   for i=1:n
      k1=feval(f,ts(i),ys(:,i));
      k2=feval(f,ts(i)+h/2,ys(:,i)+h/2*k1);
      k3=feval(f,ts(i)+h,ys(:,i)+h*(-k1+2*k2));
      fs(:,i)=k1;
      ys(:,i+1)=ys(:,i)+(k1+4*k2+k3)*h/6;
   end;
end;
```

♦

Algoritmo 6.5: Método Runge-Kutta 33 (de Nÿstrom)

```
% Implementacion del Metodo de Runge-Kutta de tercer orden
```

```
% Metodo de Nystrom
% Uso: [t,y,dy]=nystrom(funcion,inicio,fin,alfa,paso)
function [ts,ys,fs]=nystrom(f,inicio,fin,alfa,h)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
   for i=1:n
      k1=feval(f,ts(i),ys(:,i));
      k2=feval(f,ts(i)+h*(2/3),ys(:,i)+h*(2/3)*k1);
      k3=feval(f,ts(i)+h*(2/3),ys(:,i)+h*(2/3)*k2);
      fs(:,i)=k1;
      ys(:,i+1)=ys(:,i)+(2*k1+3*k2+3*k3)*h/8;
   end;
end;
```

♦

Algoritmo 6.6: Método Runge-Kutta 44 (de Runge-Kutta)

```
% Implementacion del Metodo de Runge-Kutta de cuarto orden
% Uso: [t,y,dy]=rkc4(funcion,inicio,fin,alfa,paso)
function [ts,ys,fs]=rkc4(f,inicio,fin,alfa,h)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
   for i=1:n
      k1=feval(f,ts(i),ys(:,i));
      k2=feval(f,ts(i)+h/2,ys(:,i)+(h*k1)/2);
      k3=feval(f,ts(i)+h/2,ys(:,i)+(h*k2)/2);
      k4=feval(f,ts(i)+h,ys(:,i)+h*k3);
      fs(:,i)=k1;
      ys(:,i+1)=ys(:,i)+(k1+2*k2+2*k3+k4)*h/6;
   end;
end;
```

♦

Algoritmo 6.7: Método Runge-Kutta 44 (de Kutta)

```
% Implementacion del Metodo de Runge-Kutta de cuarto orden
% Metodo de Kutta de orden 4
% Uso: [t,y,dy]=kutta4(funcion,inicio,fin,alfa,paso)
function [ts,ys,fs]=kutta4(f,inicio,fin,alfa,h)
```

```
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
   for i=1:n
      k1=feval(f,ts(i),ys(:,i));
      k2=feval(f,ts(i)+h/3,ys(:,i)+(h*k1)/3);
      k3=feval(f,ts(i)+h*(2/3),ys(:,i)+(-k1+3*k2)*h/3);
      k4=feval(f,ts(i)+h,ys(:,i)+h*(k1-k2+k3));
      fs(:,i)=k1;
      ys(:,i+1)=ys(:,i)+(k1+3*k2+3*k3+k4)*h/8;
   end;
end;
```

♦

Algoritmo 6.8: Método Runge-Kutta 56 (A)

```
% Implementacion del Metodo de Runge-Kutta de quinto orden
% Metodo A de orden 5 con 6 evaluaciones
% Uso: [t,y,dy]=rk56a(funcion,inicio,fin,alfa,paso)
function [ts,ys,fs]=rk56a(f,inicio,fin,alfa,h)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
   for i=1:n
      k1=feval(f,ts(i),ys(:,i));
      k2=feval(f,ts(i)+h,ys(:,i)+h*k1);
      k3=feval(f,ts(i)+h,ys(:,i)+(k1+k2)*h/2);
      k4=feval(f,ts(i)+h/4,ys(:,i)+h*(14*k1+5*k2-3*k3)/64);
      k5=feval(f,ts(i)+h/3,ys(:,i)+h/96*
               (-12*k1-12*k2+8*k3+64*k4));
      k6=feval(f,ts(i)+h*(3/4),ys(:,i)+h/64*
               (-9*k2+5*k3+16*k4+36*k5));
      fs(:,i)=k1;
      ys(:,i+1)=ys(:,i)+(7*k1+7*k3+32*k4+12*k5+32*k6)*h/90;
   end;
end;
```

♦

Algoritmo 6.9: Método Runge-Kutta 56 (B)

```
% Implementacion del Metodo de Runge-Kutta de quinto orden
```

```
% Metodo B de orden 5 con 6 evaluaciones
% Uso: [t,y,dy]=rk56b(funcion,inicio,fin,alfa,paso)
function [ts,ys,fs]=rk56b(f,inicio,fin,alfa,h)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
   for i=1:n
      k1=feval(f,ts(i),ys(:,i));
      k2=feval(f,ts(i)+h/2,ys(:,i)+h*k1/2);
      k3=feval(f,ts(i)+h/4,ys(:,i)+(k1+k2)*h/8);
      k4=feval(f,ts(i)+h/2,ys(:,i)+h*k3/2);
      k5=feval(f,ts(i)+h*(3/4),ys(:,i)+h/16*(-3*k2+6*k3+9*k4));
      k6=feval(f,ts(i)+h,ys(:,i)+h/7*(k1+4*k2+6*k3-12*k4+8*k5));
      fs(:,i)=k1;
      ys(:,i+1)=ys(:,i)+(7*k1+32*k3+12*k4+32*k5+7*k6)*h/90;
   end;
end;
```

♦

6.3 Métodos multipaso

6.3.1 Introducción a los métodos multipaso

En este tipo de métodos, para calcular un valor en la malla de puntos se usan varios de los valores previamente calculados, y no sólo el valor anterior.

$$\int_{t_{i+1-m}}^{t_{i+1}} y'(t) = \int_{t_{i+1-m}}^{t_{i+1}} f(t,y)\,dt$$

usando $y_{i+1-m} \dots y_{i-1}, y_i$,

$$\begin{aligned} \alpha_0 y_{i+1} + \alpha_1 y_i + \ldots + \alpha_m y_{i+1-m} &= \int_{t_{i+1-m}}^{t_{i+1}} f(t,y)\,dt \\ &= h\left(\beta_0 f_{i+1} + \beta_1 f_i + \ldots + \beta_m f_{i+1-m}\right) \end{aligned}$$

Donde hemos usado integración numérica para la integral de $f(t,y)$. Al despejar y_{i+1} se tiene

$$y_{i+1} = \sum_{r=1}^{m} a_r y_{i+1-r} + h\sum_{r=0}^{m} b_r f\left(t_{i+1-r}, y_{i+1-r}\right) \tag{6.8}$$

Si $b_0 \neq 0$, lo que queremos calcular (y_{i+1}) está en ambos lados, entonces el método se llama **implícito**, en otro caso, **explícito**. Ahora no se toman puntos intermedios θ_r, sino puntos anteriores en los cuales tenemos la información que necesitamos.

Cuando empezamos, en un PVI de primer orden, sólo tenemos un valor, así que para **preparar el principio de tabla** hasta los m valores que necesita el método multipaso, se usan métodos unipaso.

Veamos el caso en que $b_0 \neq 0$:

$$y_{i+1} = v_{1,i} + hb_0 f\left(t_{i+1}, y_{i+1}\right) + \underbrace{h\sum_{r=1}^{m} b_r f\left(t_{i+1-r}, y_{i+1-r}\right)}_{v_{2,i}}$$

$$\underline{y_{i+1}} = c_i + hb_0 f\left(t_{i+1}, \underline{y_{i+1}}\right)$$

Si f no es lineal, tenemos una ecuación no lineal en y_{i+1}, lo cual podemos resolver mediante aproximaciones sucesivas con una iteración. Una alternativa a resolver la ecuación no lineal es usar lo que se conoce como método **predictor-corrector**, que básicamente consiste en "predecir" un y_{i+1} usando un método explícito que nos da y_{i+1}^p (valor predicho). Luego "corregimos" el valor usando

$$y_{i+1}^c = c_i + hb_0 f\left(t_{i+1}, y_{i+1}^p\right)$$

Siendo y_{i+1}^c el valor corregido. En general funciona bien, si h es suficientemente pequeña. También se puede hacer recursivamente, de forma que con suficientes repeticiones, converja a la solución.

Los métodos multipaso se pueden deducir por integración de la ecuación diferencial y usando interpolación polinomial o utilizando el método de los coeficientes indeterminados, imponiendo que la ecuación en diferencia sea exacta para todo problema de solución un polinomio de bajo grado.

Ejemplo 6.2:

Resolver el mismo ejemplo que en el apartado (6.2.4.3) utilizando los métodos de Adams del mismo orden en la forma predictor-corrector, suponiendo calculada por un método unipaso una estimación de $\overline{u}_1$.

$$ty'' + yy' + \operatorname{sen}\frac{\pi t}{6} = 0$$

$$y\left(1\right) = 1,\ y'\left(1\right) = 2$$

Solución:

Teniendo el valor aproximado de $\overline{u}_1$ del ejemplo mencionado, y usando los métodos explícito como predictor y el implícito como corrector, tenemos que usar:

$$y_{i+1} = y_i + \frac{h}{2}\left(3f_i - f_{i-1}\right)$$

$$y_{i+1} = y_i + \frac{h}{2}\left(f_{i+1} - f_i\right)$$

Por ejemplo, para un método de segundo orden. Predecimos el valor de $\overline{u}_2$:

$$\begin{pmatrix} y\left(2\right) \\ y'\left(2\right) \end{pmatrix} \simeq \overline{u}_2^P = \overline{u}_1 + \frac{h}{2}\left(3f\left(1.5, \overline{u}_1\right) - f\left(1, \overline{u}_0\right)\right) = \begin{pmatrix} 1.90738 \\ 0.421421 \end{pmatrix}$$

Vemos que el valor de la función crece, lo cual es un reflejo de que $y^{'}(1.5) > 0$, tal y como se ve en el ejemplo anterior.

Ahora corregimos el valor,

$$\begin{pmatrix} y(2) \\ y'(2) \end{pmatrix} \simeq \overline{u}_2^C = \overline{u}_1 + \frac{0.5}{2}\left(f\left(2, \overline{u}_2^P\right) + f\left(1.5, \overline{u}_1\right)\right) = \begin{pmatrix} 2.03281 \\ 0.3633 \end{pmatrix}$$

Si lo hacemos con el siguiente punto,

$$\begin{pmatrix} y(2.5) \\ y'(2.5) \end{pmatrix} \simeq \overline{u}_3^P = \overline{u}_2^C + \frac{0.5}{2}\left(3f\left(2, \overline{u}_2^C\right) - f\left(1.5, \overline{u}_1\right)\right)$$

$$\begin{pmatrix} y(2.5) \\ y'(2.5) \end{pmatrix} \simeq \overline{u}_3^C = \overline{u}_2^C + \frac{0.5}{2}\left(f\left(2.5, \overline{u}_3^P\right) + f\left(2, \overline{u}_2^C\right)\right)$$

Con lo que aplicándolo repetidamente, tenemos la solución, aunque es más cómodo hacerlo con un programa que no en papel.

■

6.3.2 Error de truncamiento

$$T_{i+1} = \frac{y(t_i + h) - \sum_{r=1}^{m} a_r y(t_{i+1-r})}{h} - \sum_{r=0}^{m} b_r \underbrace{f\left(t_{i+1-r}, y(t_{i+1-r})\right)}_{y'(t_{i+1-r})} \tag{6.9}$$

Haciendo el desarrollo en serie:

$$T_{i+1} = \frac{1 - \sum_{r=1}^{m} a_r}{h} y(t_i) + \left(h\frac{1 + \sum_{r=1}^{m}(r-1)a_r}{h} - \sum_{r=0}^{m} b_r \right) y'(t_i) + O(h)$$

Si $h \to 0$, la *consistencia* viene dada por

✏ $y' = 0 \Rightarrow y = c$, por lo que la solución nos lleva al siguiente error de truncamiento

$$T_{i+1} = \frac{c\left(1 - \sum_{r=1}^{m} a_r\right)}{h}$$

entonces, para que $\lim_{h\to 0} T_{i+1} = 0$ debe ser

$$\boxed{1 - \sum_{r=1}^{m} a_r = 0} \tag{6.10}$$

es la condición necesaria para la consistencia

✏ para $y' = 1 \Rightarrow y = t$ (con la condición inicial adecuada para que así sea), el error de truncamiento es

$$T_{i+1} = \frac{t_{i+1} - \sum_{r=1}^{m} a_r t_{i+1-r}}{h} - \sum_{r=0}^{m} b_r$$

Escribiendo

$$t_{i+1-r} = t_{i+1-r+m-m} = t_{i+1-m} + (m-r)\,h$$

nos queda

$$T_{i+1} = \frac{t_{i+1-m} + mh - \sum_{r=1}^{m} a_r \left(t_{i+1-m} + (m-r)\,h\right)}{h} - \sum_{r=0}^{m} b_r$$

Suponiendo que se cumple la condición anterior, resulta

$$\boxed{m - \sum_{r=1}^{m} (m-r)\,a_r = \sum_{r=0}^{m} b_r} \tag{6.11}$$

Si definimos unos **polinomios característicos** del método, a partir de las condiciones anteriores:

$$\rho(\lambda) = \lambda^m - a_1\lambda^{m-1} - \ldots - a_m$$

es el polinomio primero, y

$$\sigma(\lambda) = b_0\lambda^m + b_1\lambda^{m-1} + \ldots + b_m$$

el segundo.

Entonces, las condiciones de consistencia son:

$$\boxed{\begin{array}{c}\rho(1) = 0 \\ \\ \rho'(1) = \sigma(1)\end{array}} \tag{6.12}$$

Que se corresponden con las anteriores. Veamos que es así para la segunda:

$$\sigma(1) = 1 + \sum_{r=1}^{m} (r-1)\,a_r = 1 + a_2 + 2a_3 + \ldots + (m-1)\,a_m$$

Sumándole y restándole $(m-1)\,\rho(1)$:

$$\begin{aligned}\sigma(1) &= \underbrace{m - (m-1)\,a_1 - (m-2)\,a_2 - \ldots - a_{m-1}}_{\rho'(1)} - (m-1)\,\rho(1) \\ &= \rho'(1) - (m-1)\underbrace{\rho(1)}_{0} \\ &= \rho'(1)\end{aligned}$$

6.3.3 Algunos métodos multipaso

Métodos de Adams-Bashforth

Tienen el mismo orden que número de pasos, y son explícitos.

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{2}\left(3f_i - f_{i-1}\right) \\ y_{i+1} &= y_i + \tfrac{h}{12}\left(23f_i - 16f_{i-1} + 5f_{i-2}\right) \\ y_{i+1} &= y_i + \tfrac{h}{24}\left(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}\right) \\ y_{i+1} &= y_i + \tfrac{h}{720}\left(1901f_i - 2774f_{i-1} + 2616f_{i-2} - 1274f_{i-3} + 251f_{i-4}\right) \end{aligned}$$

Métodos de Adams-Moulton

Tienen de orden una unidad mayor que el número de pasos, y son implícitos.

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{2}\left(f_{i+1} + f_i\right) \\ y_{i+1} &= y_i + \tfrac{h}{12}\left(5f_{i+1} + 8f_i - f_{i-1}\right) \\ y_{i+1} &= y_i + \tfrac{h}{24}\left(9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}\right) \\ y_{i+1} &= y_i + \tfrac{h}{720}\left(251f_{i+1} + 646f_i - 264f_{i-1} + 106f_{i-2} - 19f_{i-3}\right) \end{aligned}$$

Métodos multipaso de Nÿstrom

$$\begin{aligned} y_{i+1} &= y_{i-1} + 2hf_i \\ y_{i+1} &= y_{i-1} + \tfrac{h}{3}\left(7f_i - 2f_{i-1} + f_{i-2}\right) \\ y_{i+1} &= y_{i-1} + \tfrac{h}{3}\left(8f_i - 5f_{i-1} + 4f_{i-2} - f_{i-3}\right) \end{aligned}$$

Veamos para el último (4 pasos):

$$\rho\left(\lambda\right) = \lambda^4 - \lambda^2$$

$$\sigma\left(\lambda\right) = \left(8\lambda^3 - 5\lambda^2 + 4\lambda - 1\right)\frac{1}{3}$$

$$\rho\left(1\right) = 0,\ \rho'\left(\lambda\right) = 4\lambda^3 - 2\lambda$$

$$\rho'\left(1\right) = 2 = \sigma\left(1\right)$$

Por tanto, es consistente.

Métodos multipaso de Milne *(Milne como predictor y Simpson como corrector)*

$$\begin{aligned} y_{i+1} &= y_{i-3} + \tfrac{h}{3}\left(8f_i - 4f_{i-1} + 8f_{i-2}\right) \\ y_{i+1} &= y_{i-1} + \tfrac{h}{3}\left(f_{i+1} + 4f_i + f_{i-1}\right) \end{aligned}$$

Métodos multipaso de Hermite

$$\begin{aligned} y_{i+1} &= -4y_i + 5y_{i-1} + 2h\left(2f_i + f_{i-1}\right) \\ y_{i+1} &= 2y_i - y_{i-1} + \tfrac{h}{2}\left(f_{i+1} - f_{i-1}\right) \end{aligned}$$

Predictor-corrector de Hamming

Son de orden 4.

$$\begin{aligned} y^p_{i+1} &= y_{i-3} + \tfrac{h}{3}\left(8f_i - 4f_{i-1} + 8f_{i-2}\right) \\ y^m_{i+1} &= y^p_{i+1} - \tfrac{112}{121}\left(y^p_i - y^c_i\right) \\ y^c_{i+1} &= \tfrac{9}{8}y_i - \tfrac{1}{8}y_{i-2} + \tfrac{3h}{8}\left(f^m_{i+1} + 2f_i - f_{i-1}\right) \\ y_{i+1} &= y^c_{i+1} + \tfrac{9}{121}\left(y^p_{i+1} - y^c_{i+1}\right) \end{aligned}$$

6.3.4 Algoritmos en Matlab para métodos multipaso

Igual que vimos en el apartado de métodos unipaso, vamos a incluir aquí los algoritmos en Matlab para los métodos multipaso vistos en el tema.

Algoritmo 6.10: Método multipaso de Adams-Bashforth de orden 2

```
% Implementacion de metodos multipaso para
% Ecuaciones Diferenciales Ordinarias
% Metodo de Adams-Bashforth de segundo orden
%
% Uso: [ts,ys]=adbash2(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
function [ts,ys]=adbash2(f,inicio,fin,alfa,h,metodo)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
```

```
    if nargin<6
       metodo='Runge-Kutta4';
    end;
    ts=inicio:h:fin;
    ys(:,1)=alfa;
    n=(fin-inicio)/h;
   % Calculo de los valores iniciales
    if strcmp(metodo,'Runge-Kutta4')
       [tinic,yinic]=rkc4(f,inicio,inicio+h,alfa,h);
    elseif strcmp(metodo,'Pmedio')
       [tinic,yinic]=edpmedio(f,inicio,inicio+h,alfa,h);
    elseif strcmp(metodo,'Euler-modificado')
       [tinic,yinic]=eulermod(f,inicio,inicio+h,alfa,h);
    elseif strcmp(metodo,'Heun')
       [tinic,yinic]=heun(f,inicio,inicio+h,alfa,h);
    elseif strcmp(metodo,'Kutta3')
       [tinic,yinic]=kutta3(f,inicio,inicio+h,alfa,h);
    elseif strcmp(metodo,'Nystrom')
       [tinic,yinic]=nystrom(f,inicio,inicio+h,alfa,h);
    elseif strcmp(metodo,'Kutta4')
       [tinic,yinic]=kutta4(f,inicio,inicio+h,alfa,h);
    elseif strcmp(metodo,'RK56A')
       [tinic,yinic]=rk56a(f,inicio,inicio+h,alfa,h);
    elseif strcmp(metodo,'RK56B')
       [tinic,yinic]=rk56b(f,inicio,inicio+h,alfa,h);
    else
       disp('Metodo incorrecto');
       return;
    end;
    ys(:,2)=yinic(:,2);
   % Calculo de los restantes valores
    for i=2:n
       ys(:,i+1)=ys(:,i)+h*(3*feval(f,ts(i),ys(:,i))-
                 feval(f,ts(i-1),ys(:,i-1)))/2;
    end;
end;
```

♦

Algoritmo 6.11: Método multipaso de Adams-Bashforth de orden 3

```
% Implementacion de metodos multipaso para
% Ecuaciones Diferenciales Ordinarias
% Metodo de Adams-Bashforth de tercer orden
%
% Uso: [ts,ys]=adbash3(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
```

```
function [ts,ys]=adbash3(f,inicio,fin,alfa,h,metodo)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   if nargin<6
      metodo='Runge-Kutta4';
   end;
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
  % Calculo de los valores iniciales
   if strcmp(metodo,'Runge-Kutta4')
      [tinic,yinic]=rkc4(f,inicio,inicio+2*h,alfa,h);
   elseif strcmp(metodo,'Pmedio')
      [tinic,yinic]=edpmedio(f,inicio,inicio+2*h,alfa,h);
   elseif strcmp(metodo,'Euler-modificado')
      [tinic,yinic]=eulermod(f,inicio,inicio+2*h,alfa,h);
   elseif strcmp(metodo,'Heun')
      [tinic,yinic]=heun(f,inicio,inicio+2*h,alfa,h);
   elseif strcmp(metodo,'Kutta3')
      [tinic,yinic]=kutta3(f,inicio,inicio+2*h,alfa,h);
   elseif strcmp(metodo,'Nystrom')
      [tinic,yinic]=nystrom(f,inicio,inicio+2*h,alfa,h);
   elseif strcmp(metodo,'Kutta4')
      [tinic,yinic]=kutta4(f,inicio,inicio+2*h,alfa,h);
   elseif strcmp(metodo,'RK56A')
      [tinic,yinic]=rk56a(f,inicio,inicio+2*h,alfa,h);
   elseif strcmp(metodo,'RK56B')
      [tinic,yinic]=rk56b(f,inicio,inicio+2*h,alfa,h);
   else
      disp('Metodo incorrecto');
      return;
   end;
   ys(:,2)=yinic(:,2);
   ys(:,3)=yinic(:,3);
  % Calculo de los restantes valores
   for i=3:n
      ys(:,i+1)=ys(:,i)+h*(23*feval(f,ts(i),ys(:,i))-
                16*feval(f,ts(i-1),ys(:,i-1))+
                5*feval(f,ts(i-2),ys(:,i-2)))/12;
   end;
end;
```

♦

Algoritmo 6.12: Método multipaso de Adams-Bashforth de orden 4

```
% Implementacion de metodos multipaso para
% Ecuaciones Diferenciales Ordinarias
% Metodo de Adams-Bashforth de cuarto orden
%
```

```
% Uso: [ts,ys]=adbash4(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
function [ts,ys]=adbash4(f,inicio,fin,alfa,h,metodo)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   if nargin<6
      metodo='Runge-Kutta4';
   end;
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
  % Calculo de los valores iniciales
   if strcmp(metodo,'Runge-Kutta4')
      [tinic,yinic]=rkc4(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Pmedio')
      [tinic,yinic]=edpmedio(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Euler-modificado')
      [tinic,yinic]=eulermod(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Heun')
      [tinic,yinic]=heun(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Kutta3')
      [tinic,yinic]=kutta3(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Nystrom')
      [tinic,yinic]=nystrom(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Kutta4')
      [tinic,yinic]=kutta4(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'RK56A')
      [tinic,yinic]=rk56a(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'RK56B')
      [tinic,yinic]=rk56b(f,inicio,inicio+3*h,alfa,h);
   else
      disp('Metodo incorrecto');
      return;
   end;
   ys(:,2)=yinic(:,2);
   ys(:,3)=yinic(:,3);
   ys(:,4)=yinic(:,4);
  % Calculo de los restantes valores
   for i=4:n
      ys(:,i+1)=ys(:,i)+h*(55*feval(f,ts(i),ys(:,i))-
                59*feval(f,ts(i-1),ys(:,i-1))+
                37*feval(f,ts(i-2),ys(:,i-2))-
                9*feval(f,ts(i-3),ys(:,i-3)))/24;
   end;
end;
```

♦

Algoritmo 6.13: Método multipaso de Adams-Bashforth de orden 5

```
% Implementacion de metodos multipaso para
% Ecuaciones Diferenciales Ordinarias
% Metodo de Adams-Bashforth de quinto orden
%
% Uso: [ts,ys]=adbash5(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
function [ts,ys]=adbash5(f,inicio,fin,alfa,h,metodo)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   if nargin<6
      metodo='Runge-Kutta4';
   end;
   ts=inicio:h:fin;
   ys(1)=alfa;
   n=(fin-inicio)/h;
  % Calculo de los valores iniciales
   if strcmp(metodo,'Runge-Kutta4')
      [tinic,yinic]=rkc4(f,inicio,inicio+4*h,alfa,h);
   elseif strcmp(metodo,'Pmedio')
      [tinic,yinic]=edpmedio(f,inicio,inicio+4*h,alfa,h);
   elseif strcmp(metodo,'Euler-modificado')
      [tinic,yinic]=eulermod(f,inicio,inicio+4*h,alfa,h);
   elseif strcmp(metodo,'Heun')
      [tinic,yinic]=heun(f,inicio,inicio+4*h,alfa,h);
   elseif strcmp(metodo,'Kutta3')
      [tinic,yinic]=kutta3(f,inicio,inicio+4*h,alfa,h);
   elseif strcmp(metodo,'Nystrom')
      [tinic,yinic]=nystrom(f,inicio,inicio+4*h,alfa,h);
   elseif strcmp(metodo,'Kutta4')
      [tinic,yinic]=kutta4(f,inicio,inicio+4*h,alfa,h);
   elseif strcmp(metodo,'RK56A')
      [tinic,yinic]=rk56a(f,inicio,inicio+4*h,alfa,h);
   elseif strcmp(metodo,'RK56B')
      [tinic,yinic]=rk56b(f,inicio,inicio+4*h,alfa,h);
   else
      disp('Metodo incorrecto');
      return;
   end;
   ys(2)=yinic(2);
   ys(3)=yinic(3);
   ys(4)=yinic(4);
```

```
      ys(5)=yinic(5);
      % Calculo de los restantes valores
      for i=5:n
         ys(i+1)=ys(i)+h*(1901*feval(f,ts(i),ys(i))-
                 2774*feval(f,ts(i-1),ys(i-1))+
                 2616*feval(f,ts(i-2),ys(i-2))-
                 1274*feval(f,ts(i-3),ys(i-3))+
                 251*feval(f,ts(i-4),ys(i-4)))/720;
      end;
   end;
```

♦

Algoritmo 6.14: Método multipaso de Adams-Moulton de orden 2

```
% Implementacion de metodos multipaso para
% Ecuaciones Diferenciales Ordinarias
% Metodo de Adams-Moulton de segundo orden
%
% Uso: [ts,ys]=admoul2(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales que ademas se usan como valores
% predecidos. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
function [ts,ys]=admoul2(f,inicio,fin,alfa,h,metodo)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   if nargin<6
      metodo='Runge-Kutta4';
   end;
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
   % Calculo de los valores iniciales
   if strcmp(metodo,'Runge-Kutta4')
      [tpred,ypred]=rkc4(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Pmedio')
      [tpred,ypred]=edpmedio(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Euler-modificado')
      [tpred,ypred]=eulermod(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Heun')
      [tpred,ypred]=heun(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Kutta3')
      [tpred,ypred]=kutta3(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Nystrom')
      [tpred,ypred]=nystrom(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Kutta4')
```

```
      [tpred,ypred]=kutta4(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'RK56A')
      [tpred,ypred]=rk56a(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'RK56B')
      [tpred,ypred]=rk56b(f,inicio,fin,alfa,h);
   else
      disp('Metodo incorrecto');
      return;
   end;
   ys(:,2)=ypred(:,2);
  % Calculo de los restantes valores
   for i=1:n
      ys(:,i+1)=ys(:,i)+h*(feval(f,tpred(i+1),ypred(:,i+1))+
                feval(f,ts(i),ys(:,i)))/2;
   end;
end;
```

♦

Algoritmo 6.15: Método multipaso de Adams-Moulton de orden 3

```
% Implementacion de metodos multipaso para
% Ecuaciones Diferenciales Ordinarias
% Metodo de Adams-Moulton de tercer orden
%
% Uso: [ts,ys]=admoul3(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales que ademas se usan como valores
% predecidos. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
function [ts,ys]=admoul3(f,inicio,fin,alfa,h,metodo)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   if nargin<6
      metodo='Runge-Kutta4';
   end;
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
  % Calculo de los valores iniciales
   if strcmp(metodo,'Runge-Kutta4')
      [tpred,ypred]=rkc4(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Pmedio')
      [tpred,ypred]=edpmedio(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Euler-modificado')
      [tpred,ypred]=eulermod(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Heun')
```

```
      [tpred,ypred]=heun(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Kutta3')
      [tpred,ypred]=kutta3(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Nystrom')
      [tpred,ypred]=nystrom(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Kutta4')
      [tpred,ypred]=kutta4(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'RK56A')
      [tpred,ypred]=rk56a(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'RK56B')
      [tpred,ypred]=rk56b(f,inicio,fin,alfa,h);
   else
      disp('Metodo incorrecto');
      return;
   end;
   ys(:,2)=ypred(:,2);
   ys(:,3)=ypred(:,3);
  % Calculo de los restantes valores
   for i=2:n
      ys(:,i+1)=ys(:,i)+h*(5*feval(f,tpred(i+1),ypred(:,i+1))+
                8*feval(f,ts(i),ys(:,i))-
                feval(f,ts(i-1),ys(:,i-1)))/12;
   end;
end;
```

♦

Algoritmo 6.16: Método multipaso de Adams-Moulton de orden 4

```
% Implementacion de metodos multipaso para
% Ecuaciones Diferenciales Ordinarias
% Metodo de Adams-Moulton de cuarto orden
%
% Uso: [ts,ys]=admoul4(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales que ademas se usan como valores
% predecidos. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
function [ts,ys]=admoul4(f,inicio,fin,alfa,h,metodo)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   if nargin<6
      metodo='Runge-Kutta4';
   end;
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
```

```
    % Calculo de los valores iniciales
     if strcmp(metodo,'Runge-Kutta4')
        [tpred,ypred]=rkc4(f,inicio,fin,alfa,h);
     elseif strcmp(metodo,'Pmedio')
        [tpred,ypred]=edpmedio(f,inicio,fin,alfa,h);
     elseif strcmp(metodo,'Euler-modificado')
        [tpred,ypred]=eulermod(f,inicio,fin,alfa,h);
     elseif strcmp(metodo,'Heun')
        [tpred,ypred]=heun(f,inicio,fin,alfa,h);
     elseif strcmp(metodo,'Kutta3')
        [tpred,ypred]=kutta3(f,inicio,fin,alfa,h);
     elseif strcmp(metodo,'Nystrom')
        [tpred,ypred]=nystrom(f,inicio,fin,alfa,h);
     elseif strcmp(metodo,'Kutta4')
        [tpred,ypred]=kutta4(f,inicio,fin,alfa,h);
     elseif strcmp(metodo,'RK56A')
        [tpred,ypred]=rk56a(f,inicio,fin,alfa,h);
     elseif strcmp(metodo,'RK56B')
        [tpred,ypred]=rk56b(f,inicio,fin,alfa,h);
     else
        disp('Metodo incorrecto');
        return;
     end;
     ys(:,2)=ypred(:,2);
     ys(:,3)=ypred(:,3);
     ys(:,4)=ypred(:,4);
    % Calculo de los restantes valores
     for i=3:n
        ys(:,i+1)=ys(:,i)+h*(9*feval(f,tpred(i+1),ypred(:,i+1))+
                  19*feval(f,ts(i),ys(:,i))-
                  5*feval(f,ts(i-1),ys(:,i-1))+
                  feval(f,ts(i-2),ys(:,i-2)))/24;
     end;
end;
```

♦

Algoritmo 6.17: Método multipaso de Adams-Moulton de orden 5

```
% Implementacion de metodos multipaso para
% Ecuaciones Diferenciales Ordinarias
% Metodo de Adams-Moulton de quinto orden
%
% Uso: [ts,ys]=admoul5(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales que ademas se usan como valores
% predecidos. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
function [ts,ys]=admoul5(f,inicio,fin,alfa,h,metodo)
```

```
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   if nargin<6
      metodo='Runge-Kutta4';
   end;
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
  % Calculo de los valores iniciales
   if strcmp(metodo,'Runge-Kutta4')
      [tpred,ypred]=rkc4(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Pmedio')
      [tpred,ypred]=edpmedio(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Euler-modificado')
      [tpred,ypred]=eulermod(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Heun')
      [tpred,ypred]=heun(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Kutta3')
      [tpred,ypred]=kutta3(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Nystrom')
      [tpred,ypred]=nystrom(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'Kutta4')
      [tpred,ypred]=kutta4(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'RK56A')
      [tpred,ypred]=rk56a(f,inicio,fin,alfa,h);
   elseif strcmp(metodo,'RK56B')
      [tpred,ypred]=rk56b(f,inicio,fin,alfa,h);
   else
      disp('Metodo incorrecto');
      return;
   end;
   ys(:,2)=ypred(:,2);
   ys(:,3)=ypred(:,3);
   ys(:,4)=ypred(:,4);
   ys(:,5)=ypred(:,5);
  % Calculo de los restantes valores
   for i=4:n
      ys(:,i+1)=ys(:,i)+h*(251*feval(f,tpred(i+1),ypred(:,i+1))+
                646*feval(f,ts(i),ys(:,i))-
                264*feval(f,ts(i-1),ys(:,i-1))+
                106*feval(f,ts(i-2),ys(:,i-2))-
                19*feval(f,ts(i-3),ys(:,i-3)))/720;
   end;
end;
```

♦

Algoritmo 6.18: Método multipaso de Milne

```
% Implementacion de metodos multipaso para
```

```
% Ecuaciones Diferenciales Ordinarias
% Metodo de Milne de 4 pasos con prediccion-correccion
%
% Uso: [ts,ys]=milne(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
function [ts,ys]=milne(f,inicio,fin,alfa,h,metodo)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   if nargin<6
      metodo='Runge-Kutta4';
   end;
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
  % Calculo de los valores iniciales
   if strcmp(metodo,'Runge-Kutta4')
      [tinic,yinic]=rkc4(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Pmedio')
      [tinic,yinic]=edpmedio(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Euler-modificado')
      [tinic,yinic]=eulermod(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Heun')
      [tinic,yinic]=heun(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Kutta3')
      [tinic,yinic]=kutta3(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Nystrom')
      [tinic,yinic]=nystrom(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Kutta4')
      [tinic,yinic]=kutta4(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'RK56A')
      [tinic,yinic]=rk56a(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'RK56B')
      [tinic,yinic]=rk56b(f,inicio,inicio+3*h,alfa,h);
   else
      disp('Metodo incorrecto');
      return;
   end;
   ys(:,2)=yinic(:,2);
   ys(:,3)=yinic(:,3);
   ys(:,4)=yinic(:,4);
  % Calculo de los restantes valores
   for i=4:n
     % Prediccion
      yp(:,i+1)=ys(:,i-3)+h*(8*feval(f,ts(i),ys(:,i))-
```

```
                4*feval(f,ts(i-1),ys(:,i-1))+
                8*feval(f,ts(i-2),ys(:,i-2)))/3;
      % Correccion
       ys(:,i+1)=ys(:,i-1)+h*(feval(f,ts(i+1),yp(:,i+1))+
                4*feval(f,ts(i),ys(:,i))+
                feval(f,ts(i-1),ys(:,i-1)))/3;
   end;
end;
```

♦

Algoritmo 6.19: Método multipaso de Hermite

```
% Implementacion de metodos multipaso para
% Ecuaciones Diferenciales Ordinarias
% Metodo de Hermite de 2 pasos con prediccion-correccion
%
% Uso: [ts,ys]=edohermi(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
function [ts,ys]=edohermi(f,inicio,fin,alfa,h,metodo)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   if nargin<6
      metodo='Runge-Kutta4';
   end;
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
  % Calculo de los valores iniciales
   if strcmp(metodo,'Runge-Kutta4')
      [tinic,yinic]=rkc4(f,inicio,inicio+h,alfa,h);
   elseif strcmp(metodo,'Pmedio')
      [tinic,yinic]=edpmedio(f,inicio,inicio+h,alfa,h);
   elseif strcmp(metodo,'Euler-modificado')
      [tinic,yinic]=eulermod(f,inicio,inicio+h,alfa,h);
   elseif strcmp(metodo,'Heun')
      [tinic,yinic]=heun(f,inicio,inicio+h,alfa,h);
   elseif strcmp(metodo,'Kutta3')
      [tinic,yinic]=kutta3(f,inicio,inicio+h,alfa,h);
   elseif strcmp(metodo,'Nystrom')
      [tinic,yinic]=nystrom(f,inicio,inicio+h,alfa,h);
   elseif strcmp(metodo,'Kutta4')
      [tinic,yinic]=kutta4(f,inicio,inicio+h,alfa,h);
   elseif strcmp(metodo,'RK56A')
      [tinic,yinic]=rk56a(f,inicio,inicio+h,alfa,h);
```

```
   elseif strcmp(metodo,'RK56B')
      [tinic,yinic]=rk56b(f,inicio,inicio+h,alfa,h);
   else
      disp('Metodo incorrecto');
      return;
   end;
   ys(:,2)=yinic(:,2);
  % Calculo de los restantes valores
   for i=2:n
     % Prediccion
      yp(:,i+1)=-4*ys(:,i)+5*ys(:,i-1)+
                2*h*(2*feval(f,ts(i),ys(:,i))+
                feval(f,ts(i-1),ys(:,i-1)));
     % Correccion
      ys(:,i+1)=2*ys(:,i)-ys(:,i-1)+h*(feval(f,ts(i+1),yp(:,i+1))-
                feval(f,ts(i-1),ys(:,i-1)))/2;
   end;
end;
```

♦

Algoritmo 6.20: Método multipaso de Hamming

```
% Implementacion de metodos multipaso para
% Ecuaciones Diferenciales Ordinarias
% Metodo de Hamming de cuarto orden con prediccion-correccion
%
% Uso: [ts,ys]=hamming(funcion,inicio,fin,alfa,paso,metodo)
%
% 'metodo' indica el metodo unipaso a emplear para el calculo
% de los valores iniciales. Por defecto es 'Runge-Kutta4'.
% Opciones: 'Runge-Kutta4', 'Pmedio', 'Euler-modificado',
% 'Heun', 'Kutta3', 'Nystrom', 'Kutta4', 'RK56A' y 'RK56B'
function [ts,ys]=hamming(f,inicio,fin,alfa,h,metodo)
if nargin<5
   disp('Faltan argumentos de entrada');
   return;
else
   if nargin<6
      metodo='Runge-Kutta4';
   end;
   ts=inicio:h:fin;
   ys(:,1)=alfa;
   n=(fin-inicio)/h;
  % Calculo de los valores iniciales
   if strcmp(metodo,'Runge-Kutta4')
      [tinic,yinic]=rkc4(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Pmedio')
      [tinic,yinic]=edpmedio(f,inicio,inicio+3*h,alfa,h);
   elseif strcmp(metodo,'Euler-modificado')
      [tinic,yinic]=eulermod(f,inicio,inicio+3*h,alfa,h);
```

```
    elseif strcmp(metodo,'Heun')
       [tinic,yinic]=heun(f,inicio,inicio+3*h,alfa,h);
    elseif strcmp(metodo,'Kutta3')
       [tinic,yinic]=kutta3(f,inicio,inicio+3*h,alfa,h);
    elseif strcmp(metodo,'Nystrom')
       [tinic,yinic]=nystrom(f,inicio,inicio+3*h,alfa,h);
    elseif strcmp(metodo,'Kutta4')
       [tinic,yinic]=kutta4(f,inicio,inicio+3*h,alfa,h);
    elseif strcmp(metodo,'RK56A')
       [tinic,yinic]=rk56a(f,inicio,inicio+3*h,alfa,h);
    elseif strcmp(metodo,'RK56B')
       [tinic,yinic]=rk56b(f,inicio,inicio+3*h,alfa,h);
    else
       disp('Metodo incorrecto');
       return;
    end;
    ys(:,2)=yinic(:,2);
    ys(:,3)=yinic(:,3);
    ys(:,4)=yinic(:,4);
    yc=ys;
   % Calculo de los restantes valores
    for i=4:n
      % Prediccion
       yp(:,i+1)=ys(:,i-3)+h*(8*feval(f,ts(i),ys(:,i))-
                 4*feval(f,ts(i-1),ys(:,i-1))+
                 8*feval(f,ts(i-2),ys(:,i-2)))/3;
       ym(:,i+1)=yp(:,i+1)-112*(yp(:,i)-yc(:,i))/121;
      % Correccion
       yc(:,i+1)=9/8*ys(:,i)-1/8*ys(:,i-2)+
                 3*h*(feval(f,ts(i+1),ym(:,i+1))+
                 2*feval(f,ts(i),ys(:,i))-
                 feval(f,ts(i-1),ys(:,i-1)))/8;
      % Solucion
       ys(:,i+1)=yc(:,i+1)+9/121*(yp(:,i+1)-yc(:,i+1));
    end;
end;
```

♦

6.4 Convergencia y estabilidad de los métodos

6.4.1 Métodos unipaso

Se tiene un método unipaso tal que $\phi = 0$ si $f = 0$

$$y_{i+1} = y_i + h\phi\left(x_i, y_i, h\right)$$

El error de truncamiento es

$$0 = \lim_{h\to 0} T_{i+1} = \lim_{h\to 0}\left(\frac{y(t_i+h)-y(t_i)}{h} - \phi(t_i, y(t_i), h)\right) = y'(t_i) - \lim_{h\to 0}\phi(t_i, y(t_i), h) \qquad \forall i$$

Un método unipaso es **consistente** si y sólo si:

$$\boxed{\lim_{h\to 0}\phi(t,y,h) = f(t,y)}$$

La consistencia de orden $p \geq 1$ se consigue con una elección adecuada de ϕ.

La **convergencia** de un método unipaso viene determinada por el **error de discretización global**, definido

$$e_i = y_i - y(t_i)$$

Y el método se dice convergente si

$$\lim_{h\to 0}|e_i| = 0 \qquad i = 0:N$$

siendo $N = \frac{b-a}{h}$.

La consistencia no es suficiente para asegurar la convergencia, aunque sí es necesaria. Veamos que además, necesitamos la **estabilidad** del método.

Si simplificamos el problema a $y' = 0, y(a) = y_0$, el método unipaso nos queda:

$$y_{i+1} - y_i = 0$$

el cual es estable ya que su solución general es

$$y_i = c_1 1^i = y_0$$

Se dice, por tanto, que un método unipaso es (cero)estable. Todos los métodos unipaso vistos son estables.

Además, si cometemos un error al evaluar y_0, el error no se incrementa al propagarse, en todo caso, se mantiene.

Se obtiene el siguiente teorema como resultado:

Teorema 6.1. Dado un método unipaso $y_{i+1} = y_i + h\phi(t_i, y_i, h)$ para resolver un PVI y si existen $h_0 > 0, L > 0$ tales que

$$|\phi(t,u,h) - \phi(t,v,h)| \leq L|u-v|$$

siendo

$$a \leq t \leq b \qquad -\infty < u, v < \infty \qquad 0 \leq h \leq h_0$$

entonces el método es convergente si y sólo si es consistente y estable.

Además, si existe una función τ tal que $|T_i(h)| \leq \tau(h)$ para todo i con $0 \leq h \leq h_0$, entonces

$$|y_i - y(t_i)| \leq \frac{\tau(h)\, e^{L(t_i-a)}}{L}$$

La aplicación de los métodos vistos a ciertos problemas pueden dar resultados malos. Los problemas de este tipo se llaman **rígidos** (*stiff* en inglés), y un ejemplo es

$$y' = \mu y$$

$$y\left(0\right) = y_0$$

siendo $\Re e\left(\mu\right) < 0$.

Para este tipo de problemas existen una solución transitoria (varía mucho con t y decae a cero) y una estacionaria (varía poco con t), y puede ocurrir que si h no es suficientemente pequeño, el transitorio no decaiga a cero y se vaya a infinito.

Veamos la solución al sistema:

$$\frac{dy}{y} = \mu dt \Rightarrow \ln y = \mu t$$

$$y = y_0\, e^{\mu t}$$

La cual tiende a cero cuando t tiende a ∞.

Si usamos el método de Euler para resolver el problema:

$$y_{i+1} = y_i + hf\left(x_i, y_i\right) \rightarrow y_{i+1} = y_i + h\mu y_i = \left(1 + h\mu\right) y_i$$

$$y_i = \left(1 + h\mu\right)^i y_o$$

Dicha solución debe tender a cero cuando $i \rightarrow \infty$. Esto sólo ocurre si $|1 + h\mu| < 1$.

$$Q\left(h\mu\right) = 1 + h\mu \qquad \left|Q\left(h\mu\right)\right| < 1$$

Definición:

La **región de estabilidad absoluta** de un método unipaso es el conjunto

$$R = \left\{h\mu \in \mathbb{C} \,/\, \left|Q\left(h\mu\right)\right| < 1\right\}$$

Definición:

Se dice que un método unipaso es **A-estable** si

$$\left\{z \in \mathbb{C} \,/ \Re e\left(z\right) < 0\right\} \subseteq R$$

Es decir, que su región de estabilidad absoluta comprende todo el semiplano izquierdo.

Ejemplo 6.3:

Dado el problema siguiente

$$y' = -30y \qquad y\left(0\right) = 2$$

y longitud de paso $h = 0.1$, verificar si el método de Euler es absolutamente estable para este problema

$$y_{i+1} = y_i + hf\left(t_i, y_i\right)$$

Sustituyendo $f(t_i, y_i) = -30y_i$,

$$y_{i+1} = y_i + h(-30y_i) = y_i(1 - 30h)$$

$$Q(\mu, h) = 1 - 30h$$

Hay que verificar si

$$|Q(\mu, h)| < 1$$

que en este caso, dado que $Q(-30, 0.1) = -2$, por lo que no es absolutamente estable.

Si usamos el método de Euler regresivo

$$y_{i+1} = y_i + hf(t_{i+1}, y_{i+1})$$

tenemos

$$y_{i+1} = y_i + h(-30y_{i+1}) \Rightarrow y_{i+1}(1 + 30h) = y_i \Rightarrow y_{i+1} = \frac{1}{1 + 30h} y_i$$

En este caso

$$\left|\frac{1}{1 + 30h}\right| = \left|\frac{1}{4}\right| < 1$$

por lo que es absolutamente estable. De hecho, este método es A-estable, ya que para todo h y $\mu < 0$ es absolutamente estable.

■

6.4.2 Métodos multipaso

Dado un método multipaso

$$y_{i+1} = a_1 y_i + \ldots + a_m y_{i+1-m} + h \sum_{r=0}^{m} b_r f_{i+1-r}$$

Hemos visto que es *consistente* si y sólo si

$$\rho(1) = 0$$

$$\rho'(1) = \sigma(1)$$

Se puede conseguir una consistencia de mayor orden con condiciones adicionales. Por ejemplo, a partir del error de truncamiento

$$T_{i+1} = \frac{y(t_{i+1}) - a_i y(t_i) - \ldots - a_m y(t_{i+1-m})}{h} - \sum_{r=0}^{m} b_r f(t_{i+1-r}, y(t_{i+1-r}))$$

Desarrollando en series de Taylor la expresión, se llega al siguiente resultado ($y' = t^k$):

$$T_{i+1}(h) = Ch^p y^{(p+1)}(c)$$

Si el método es de orden p.

$$\frac{t_{i+1}^{k+1} - a_1 t_1^{k+1} - \ldots - a_m t_{i+1-m}^{k+1}}{h} - \sum_{r=0}^{m} b_r t_{i+1-r}^{k} = 0 \qquad \forall i$$

Usando que $t_{i+1-r} = t_{i+1-m+m-r} = t_{i+1-m} + (m-r)\,h$ y si $t_{i+1-m} = 0$

$$\boxed{m^{k+1} - \sum_{r=1}^{m} a_r\,(m-r)^{k+1} = (k+1)\sum_{r=0}^{m} b_r\,(m-r)^{k} \qquad k = 1:p} \tag{6.13}$$

Es la condición de **método de orden** p.

En cuanto a la **estabilidad**, podemos hacer lo mismo que antes, que el problema sea $y' = 0, y(a) = y_0$, con lo que nos queda

$$y_{i+1} - a_i y_i - \ldots - a_m y_{i+1-m} = 0$$

La solución es de la forma $y_i = \sum c_k \lambda_k^i$ siendo λ_k una raiz de $\rho(\lambda) = 0$.

Si hay un pequeño error en las condiciones iniciales, las constantes pueden sufrir pequeños cambios, pero el mayor problema viene del exponente i que tienen los λ_k, ya que si i es suficientemente grande, puede dispararse a menos que $|\lambda_k| \leq 1$. Además, si uno de los λ_k es una raiz múltiple, aparece el término $i^j \lambda_k^i$ siendo j un número menor que la multiplicidad, i^j supone un considerable incremento del error. Por tanto, se tienen los siguientes requisitos para que el método sea **estable** (**condición de raiz**):

$$\boxed{\begin{array}{c} |\lambda_k| \leq 1 \\ \text{Si } |\lambda_k| = 1, \text{ la raiz es simple} \end{array}}$$

Se dice además que un método es **estable fuerte** si la única raíz característica de módulo unidad es $\lambda_1 = 1$.

Ejemplo 6.4:

Determinar α, b_1 y b_2 para que el método multipaso

$$y_{i+1} = 9\alpha y_i + (1 - 8\alpha)\,y_{i-1} - \alpha y_{i-2} + h\,(b_1 f_i + b_2 f_{i-1})$$

sea consistente, estable y del orden más alto posible.

Solución:

Veamos primero las condiciones de consistencia, para ello, escribimos los polinomios característicos:

$$\rho(\lambda) = \lambda^3 - 9\alpha\lambda^2 - (1 - 8\alpha)\,\lambda + \alpha$$
$$\sigma(\lambda) = b_1\lambda^2 + b_2\lambda$$

La condición $\rho(1) = 0$ se cumple para todo α y la otra:

$$\rho'(1) = 2 - 10\alpha$$

$$\sigma(1) = b_1 + b_2$$

por tanto, se cumple si

$$2 - 10\alpha = b_1 + b_2$$

Para la estabilidad, veamos las raíces de $\rho(\lambda) = 0$:

$$\begin{aligned}\lambda_1 &= 1\\ \lambda_2 &= \frac{9\alpha - 1 + \sqrt{(9\alpha-1)^2 + 4\alpha}}{2}\\ \lambda_3 &= \frac{9\alpha - 1 - \sqrt{(9\alpha-1)^2 + 4\alpha}}{2}\end{aligned}$$

Es fácil comprobar que las tres raíces son reales para todo valor de α. Para obtener el intervalo de α en el que el método es estable, tenemos que obtener a lo sumo una raíz que sea -1 y ninguna que sea 1 de λ_2 y λ_3 ya que $\lambda_1 = 1$ y para que sea estable, esa raiz tiene que tener multiplicidad la unidad. En ese caso:

$$\lambda_3 = -1 \quad \Rightarrow \quad 9\alpha - 1 - \sqrt{(9\alpha-1)^2 + 4\alpha} = -2$$

$$9\alpha + 1 = \sqrt{(9\alpha-1)^2 + 4\alpha}$$

$$81\alpha^2 + 18\alpha + 1 = 81\alpha^2 - 18\alpha + 1 + 4\alpha$$

$$\alpha = 0$$

$$\lambda_2 = 1 \quad \Rightarrow \quad \alpha = 0.2$$

Por tanto, es estable si y sólo si

$$0 \leq \alpha < 0.2$$

Veamos ahora cuál es el orden del método.

$$m = 3, k = 1$$

$$\begin{aligned}3^2 - 9\alpha 2^2 - (1 - 8\alpha)\,1^2 &= 2\left(2^1 b_1 + 1^1 b_2\right)\\ 4 - 14\alpha &= 2b_1 + b_2\end{aligned}$$

Esto nos asegura que el orden no es inferior a 2 si

$$\begin{cases} b_2 = -6\alpha \\ b_1 = 2 - 4\alpha \\ 0 \leq \alpha < 0.2 \end{cases}$$

Si queremos que sea de orden 3, debe satisfacer

$$k = 2$$

$$\begin{aligned}3^3 - 9\alpha 2^3 - (1 - 8\alpha)\,1^3 &= 3\left(2^2 b_1 + 1^2 b_2\right)\\ 26 - 64\alpha &= 3\left(4b_1 + b_2\right)\end{aligned}$$

Nos lleva a

$$26 - 64\alpha = 3\left(8 - 16\alpha - 6\alpha\right)$$

Por tanto

$$\alpha = -1$$

Que está fuera del intervalo de estabilidad para α, por lo que no puede ser de orden 3.

■

Veamos a continuación la estabilidad absoluta para métodos multipaso. Tenemos el mismo tipo de problema rígido que en el caso de métodos unipaso

$$y' = \mu y$$

$$y\left(0\right) = y_0$$

con $\Re e\left(\mu\right) < 0$.

$$\begin{aligned} y_{i+1} - a_1 y_i - a_2 y_{i-1} - \ldots - a_m y_{i+1-m} &= h\sum_{r=0}^{m} b_r f_{i+1-r} \\ &= h\sum_{r=0}^{m} b_r \mu y_{i+1-r} \end{aligned}$$

Por tanto

$$\left(1 - b_0 h\mu\right) y_{i+1} - \left(a_1 + h\mu b_1\right) y_i - \ldots - \left(a_m + b_m h\mu\right) y_{i+1-m} = 0$$

La solución tiene la forma

$$y_i = \sum c_k \lambda_k^i \tag{6.14}$$

La ecuación característica queda

$$Q\left(\lambda, h\mu\right) = \left(1 - b_0 h\mu\right)\lambda^m - \left(a_1 + b_1 h\mu\right)\lambda^{m-1} - \ldots - \left(a_m + b_m h\mu\right)$$

Para que (6.14)$\rightarrow 0$ con $i \rightarrow \infty$, las raíces de $Q\left(\lambda, h\mu\right)$ tienen que cumplir

$$\left|\lambda_k\right| < 1 \qquad \forall k$$

Definición:

La **región de estabilidad absoluta** de un método multipaso lineal es el conjunto

$$R = \left\{h\mu \in \mathbb{C} \,/\, \left|\lambda_k\right| < 1 / Q\left(\lambda_k, h\mu\right) = 0\right\}$$

Definición:

Se dice que un método multipaso es **A-estable** si

$$\left\{z \in \mathbb{C} \,/ \Re e\left(z\right) < 0\right\} \subseteq R$$

El único método multipaso A-estable es el método de los trapecios.

6.5 Estimación de error de truncamiento y cambio de paso

El error de truncamiento local de un método unipaso de orden p se puede estimar a partir de otro método de orden q siendo $q > p$.

$$y\left(t_{i+1}\right) \simeq y_{i+1} + hT_{i+1}\left(h\right)$$

$$y\left(t_{i+1}\right) \simeq \hat{y}_{i+1} + h\hat{T}_{i+1}\left(h\right)$$

Se tiene

$$T_{i+1}\left(h\right) \simeq \hat{T}_{i+1}\left(h\right) + \frac{\hat{y}_{i+1} - y_{i+1}}{h}$$

$$T_{i+1}\left(h\right) \simeq \frac{\hat{y}_{i+1} - y_{i+1}}{h}$$

Esto en principio requiere un esfuerzo doble, ya que hay que resolver el problema dos veces, aunque se obtiene la ventaja de que se puede estimar el error.

Además, se pueden usar métodos **incrustados** (embedded), de forma que los dos métodos usados utilicen valores comunes de f. Ejemplos de estos métodos son los de Kunge-Kutta-Fehlberg y Runge-Kutta-Merson.

Si tenemos $T_{i+1}\left(h\right) = Kh^p$ y queremos cambiar el paso de h a ch, entonces para que se cumpla la cota de error

$$\left|Kc^p h^p\right| = c^p \left|\frac{\hat{y}_{i+1} - y_{i+1}}{h}\right| < \varepsilon$$

la elección de c debe ser tal que

$$c \leq \sqrt[p]{\frac{\varepsilon h}{\left|\hat{y}_{i+1} - y_{i+1}\right|}}$$

Una elección más conservadora de c es

$$c \leq \sqrt[p]{\frac{\varepsilon h}{2\left|\hat{y}_{i+1} - y_{i+1}\right|}}$$

además de usar medidas que eviten grandes modificaciones del tamaño del paso. El usar este tipo de medidas más conservadoras es porque antes hemos hecho aproximaciones.

El error de truncamiento local para un método predictor-corrector que utilice métodos multipaso del mismo orden p se puede estimar según los valores predicho y corregido. Teniendo

$$y\left(t_{i+1}\right) \simeq y_{i+1}^P + K_1 h^{p+1} y^{(p+1)}\left(c_1\right)$$

$$y\left(t_{i+1}\right) \simeq y_{i+1}^C + K_2 h^{p+1} y^{(p+1)}\left(c_2\right)$$

Si h es suficientemente pequeño, $y^{(p+1)}\left(c_1\right) \simeq y^{(p+1)}\left(c_2\right)$, y si restamos las dos expresiones

$$y^{(p+1)}\left(c_1\right) \simeq y^{(p+1)}\left(c_2\right) \simeq \frac{y_{i+1} - y_{i+1}^P}{\left(K_1 - K_2\right) h^{p+1}}$$

Y se queda la estimación del error de truncamiento

$$y\left(t_{i+1}\right) - y_{i+1} \simeq \frac{K_2}{K_1 - K_2}\left(y_{i+1} - y_{i+1}^P\right) \tag{6.15}$$

De esta forma, puede plantearse un cambio de h según el error relativo (6.16) supere la tolerancia, en cuyo caso se reduce a la mitad el tamaño de paso, o si es mucho menor que dicha tolerancia, con lo que se duplicaría el tamaño de paso, reduciéndose el número de evaluaciones de la función para resolver el problema con cierta precisión.

$$\frac{\left|y\left(t_{i+1}\right) - y_{i+1}\right|}{\left|y_{i+1}\right|} < \varepsilon \tag{6.16}$$

De una forma práctica, la estimación de error puede usarse para modificar el valor predicho, en algunas ocasiones también el corregido (ejemplo: método de Hamming).

6.5.1 Algunos métodos con estimación de error

6.5.1.1 Runge-Kutta-Fehlberg (orden 4)

$$\begin{aligned}
y_{i+1} &= y_i + h\left(\tfrac{25}{216}k_1 + \tfrac{1408}{2565}k_3 + \tfrac{2197}{4104}k_4 - \tfrac{1}{5}k_5\right)\\
k_1 &= f_i\\
k_2 &= f\left(x_i + \tfrac{h}{4}, y_i + \tfrac{h}{4}k_1\right)\\
k_3 &= f\left(x_i + \tfrac{3}{8}h, y_i + \tfrac{3h}{32}\left(k_1 + 3k_2\right)\right)\\
k_4 &= f\left(x_i + \tfrac{12}{13}h, y_i + \tfrac{h}{2197}\left(1932k_1 - 7200k_2 + 7296k_3\right)\right)\\
k_5 &= f\left(x_i + h, y_i + h\left(\tfrac{439}{216}k_1 - 8k_2 + \tfrac{3680}{513}k_3 - \tfrac{845}{4104}k_4\right)\right)\\
k_6 &= f\left(x_i + \tfrac{h}{2}, y_i + h\left(-\tfrac{8}{27}k_1 + 2k_2 - \tfrac{3544}{2565}k_3 + \tfrac{1859}{4104}k_4 - \tfrac{11}{40}k_5\right)\right)
\end{aligned}$$

✏ Error local de truncamiento

$$T_{i+1} \simeq \frac{\hat{y}_{i+1} - y_{i+1}}{h} = \frac{1}{360}k_1 - \frac{128}{4275}k_3 - \frac{2197}{75240}k_4 + \frac{1}{50}k_5 + \frac{2}{55}k_6$$

$$\hat{y}_{i+1} = y_i + h\left(\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6\right)$$

6.5.1.2 Runge-Kutta-Merson (orden 4)

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{6}\left(k_1 + 4k_4 + k_5\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + \tfrac{h}{3}, y_i + \tfrac{h}{3}k_1\right) \\ k_3 &= f\left(x_i + \tfrac{h}{3}, y_i + \tfrac{h}{6}\left(k_1 + k_2\right)\right) \\ k_4 &= f\left(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{8}\left(k_1 + 3k_3\right)\right) \\ k_5 &= f\left(x_i + h, y_i + h\left(\tfrac{k_1}{2} - \tfrac{3k_3}{2} + 2k_4\right)\right) \end{aligned}$$

✏ Error local de truncamiento

$$T_{i+1} \simeq \frac{\hat{y}_{i+1} - y_{i+1}}{h} = \frac{2k_1 - 9k_3 + 8k_4 - k_5}{30}$$

6.5.1.3 Métodos implícitos de Runge-Kutta

✏ Gauss (RK_{21}):

$$\begin{aligned} y_{i+1} &= y_i + hk_1 \\ k_1 &= f\left(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_1\right) \end{aligned}$$

✏ Radau (RK_{32}):

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{4}\left(k_1 + 3k_2\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + \tfrac{2h}{3}, y_i + \tfrac{h}{3}\left(k_1 + k_2\right)\right) \end{aligned}$$

✏ Lobatto (RK_{43}):

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{h}{6}\left(k_1 + 4k_2 + k_3\right) \\ k_1 &= f_i \\ k_2 &= f\left(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{4}\left(k_1 + k_2\right)\right) \\ k_3 &= f\left(x_i + h, y_i + hk_2\right) \end{aligned}$$

6.6 Problema diferencial de valores de contorno

Vamos ahora a tratar de resolver numéricamente el siguiente problema de contorno:

$$y'' = f\left(t, y, y'\right)$$

$$y\left(a\right) = \alpha,\ y\left(b\right) = \beta$$

(la condición de contorno podría ser cualquier otra, pero usamos esta)

Para este tipo de problemas en principio, al tener valores al principio y al final, no se puede propagar la solución desde un extremo al otro del intervalo, sino que hay que resolverlo todo de golpe.

Los métodos que vamos a ver aquí nos servirán también para ver la forma de poder abordar la resolución numérica de ecuaciones en derivadas parciales.

6.6.1 Método del disparo

Se basa en el mismo principio que el tiro al blanco: **Apunto, disparo, corrijo**.

De una manera más formal, se puede decir que el método sigue el planteamiento:

- ✏ Resuelve en función de z el problema de valor inicial

  $$y'' = f\left(t, y, y'\right) \qquad y\left(a\right) = \alpha \qquad y'\left(a\right) = z \qquad a \leq t \leq b$$

  calculando la solución $y\left(t, z\right)$. Como se ve, z es la pendiente en el punto a (pendiente de disparo).
- ✏ Determinar el z^* que hace $y\left(b, z^*\right) = \beta$.
- ✏ La solución del problema de contorno es $y\left(t, z^*\right)$.

Por tanto, lo que entra en juego al resolver un problema de valor de contorno es resolver varios problemas de valor inicial.

En la práctica, nos dan un valor de z, z_j, y con ese valor resolvemos el problema de valor inicial, y mientras el valor de pendiente que usemos no resulte en $y\left(b, z_k\right) = \beta$, tenemos que modificar el z_j con algún criterio de forma que tienda a z^*.

Un método que puede usarse es el **método de la secante**, que sirve para resolver ecuaciones no lineales

$$z_{j+1} = z_j - \frac{\left(z_j - z_{j-1}\right)\phi\left(z_j\right)}{\phi\left(z_j\right) - \phi\left(z_{j-1}\right)}$$

siendo $\phi\left(z\right) = y\left(b, z\right) - \beta$. El método requiere de dos valores iniciales z_0 y z_1, y para el caso que nos ocupa, puede aplicarse de la siguiente forma

$$z_{j+1} = z_j - \frac{\left(z_j - z_{j-1}\right)\left(y_{N,z_j} - \beta\right)}{y_{N,z_j} - y_{N,z_{j-1}}}$$

La forma de usarlo es realizar 2 disparos para obtener $\phi\left(z_{i-1}\right)$ y $\phi\left(z_i\right)$, y si ninguno es tal que $\phi\left(z\right) = 0$, se aplica el método de la secante, volviéndose a disparar.

En la práctica puede que no haya solución, e incluso que el resultado obtenido con la herramienta correspondiente se desmadre (en Matlab: `NaN`). Sin embargo, el método **converge localmente**, para valores de z cercanos a la solución.

Hemos visto que se dispara hacia adelante (avanzando del valor inicial hacia el final), pero también se puede disparar hacia atrás, disparando desde el valor final hacia el inicial. De hecho, algunos problemas se resuelven mejor disparando hacia atrás.

Si lo que tenemos es un **problema lineal** de la forma

$$y'' = f\left(t, y, y'\right) = p\left(t\right) y' + q\left(t\right) y + r\left(t\right) \qquad y\left(a\right) = \alpha, y\left(b\right) = \beta$$

se puede demostrar que tiene una solución, que se puede calcular con sólo dos disparos con z_1 y z_2 cualesquiera (en principio, para algunos casos puede ser necesario elegirlos por algún criterio para evitar salidas de rango en la representación de los números) aplicando la siguiente fórmula:

$$y\left(t\right) = \lambda y_1\left(t\right) + \left(1 - \lambda\right) y_2\left(t\right) \tag{6.17}$$

siendo

$$\lambda = \frac{\beta - y_2\left(b\right)}{y_1\left(b\right) - y_2\left(b\right)} \tag{6.18}$$

Veamos que sólo hacen falta 2 disparos:

- ✏ En primer lugar, aplicando el método a $y\left(a\right)$ se tiene

$$y\left(a\right) = \lambda y_1\left(a\right) + \left(1 - \lambda\right) y_2\left(a\right) = \alpha\left(\lambda + 1 - \lambda\right) = \alpha$$

- ✏ Y luego, para $y\left(b\right)$

$$\begin{aligned} y\left(b\right) &= \lambda y_1\left(b\right) + \left(1 - \lambda\right) y_2\left(b\right) \\ &= \frac{\beta - y_2\left(b\right)}{y_1\left(b\right) - y_2\left(b\right)} y_1\left(b\right) + \left(1 - \frac{\beta - y_2\left(b\right)}{y_1\left(b\right) - y_2\left(b\right)}\right) y_2\left(b\right) \\ &= \beta \frac{y_1\left(b\right) - y_2\left(b\right)}{y_1\left(b\right) - y_2\left(b\right)} = \beta \end{aligned}$$

Por tanto, para resolver un problema de valores de contorno lineal sólo hay que resolver dos problemas de valor inicial.

6.6.2 Método de diferencias finitas

Estos métodos sustituyen las derivadas del problema diferencial por aproximaciones de las mismas en base a fórmulas de derivación numérica en los puntos de una malla. De esta forma, para obtener los valores $y_i \simeq y\left(t_i\right)$ se resuelven las ecuaciones en diferencia. Con ello, se obtienen sistemas de ecuaciones, lineales en el caso de que el problema sea lineal, y no lineales cuando el problema sea no lineal.

6.6.2.1 Problema de contorno lineal

Se tiene el problema

$$\begin{gathered} y'' = p\left(t\right) y' + q\left(t\right) y + r\left(t\right) \\ y\left(a\right) = \alpha,\ y\left(b\right) = \beta \end{gathered}$$

con $a \leq t \leq b$, $a \neq b$. También se pueden tener las condiciones de contorno más generales:

$$c_1 y(a) + c_2 y'(a) = \alpha \quad |c_1| + |c_2| \neq 0$$

$$d_1 y(b) + d_2 y'(b) = \beta \quad |d_1| + |d_2| \neq 0$$

Si aplicamos la fórmula centrada de tres puntos de derivación numérica, nos queda

$$y(a) = \alpha$$

$$\frac{y_{i+1}-2y_i+y_{i-1}}{h^2} = p(t_i)\frac{y_{i+1}-y_{i-1}}{2h} + q(t_i)y_i + r(t_i) \qquad i = 1 : N-1$$

$$y(b) = \beta$$

siendo $t_i = a + ih$, $Nh = b - a$.

En el caso que nos ocupa, el sistema de ecuaciones que nos resulta es lineal y se puede escribir

$$\begin{pmatrix} a_1 & b_1 & \dots & 0 & 0 \\ c_1 & a_2 & & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & & a_{N-2} & b_{N-2} \\ 0 & 0 & \dots & c_{N-2} & a_{N-1} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{pmatrix} = \begin{pmatrix} d_1 - c_0\alpha \\ d_2 \\ \vdots \\ d_{N-2} \\ d_{N-1} - b_{N-1}\beta \end{pmatrix}$$

donde

$$a_i = 2 + h^2 q(t_i)$$

$$b_i = -1 + \frac{h}{2}p(t_i)$$

$$c_i = -1 - \frac{h}{2}p(t_{i+1})$$

$$d_i = -h^2 r(t_i)$$

A efectos de trabajo con un ordenador, con el fin de ahorrar memoria, lo recomendable es guardar únicamente las diagonales, ya que la matriz de sistema es tridiagonal y el sistema puede ser muy grande. Será además más grande cuantos más puntos se usen en las fórmulas de derivación numérica.

6.6.2.2 Problema de contorno no lineal

Ahora ocupamos el caso en que f no sea lineal. Aplicando de nuevo la fórmula de derivación centrada de tres puntos,

$$y(a) = \alpha$$

$$\frac{y_{i+1}-2y_i+y_{i-1}}{h^2} = f\left(t_i, y_i, \frac{y_{i+1}-y_{i-1}}{2h}\right); \quad i = 1 : N-1$$

$$y(b) = \beta$$

De esta forma, el sistema resultante es no lineal y se puede escribir

$$\begin{pmatrix} 2 & -1 & & & \\ & \ddots & & & \\ & -1 & 2 & -1 & \\ & & & \ddots & \\ & & & -1 & 2 \end{pmatrix} [y_i] = \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \\ \beta \end{pmatrix} - h^2 \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{N-1} \end{pmatrix}$$

donde

$$g_1 = f\left(t_1, y_1, \tfrac{y_2-\alpha}{2h}\right)$$

$$g_i = f\left(t_i, y_i, \tfrac{y_{i+1}-y_i}{2h}\right)$$

$$\vdots$$

$$g_{N-1} = f\left(t_{N-1}, y_{N-1}, \tfrac{\beta-y_{N-2}}{2h}\right)$$

La solución de este sistema puede obtenerse por algún método iterativo para sistemas no lineales que se verán en el siguiente tema, o bien con los métodos de Piccard y Gauss-Seidel o SOR adaptados a sistemas no lineales.

6.6.3 Métodos de elementos finitos o de proyección

En estos métodos se trata de aproximar la solución de un problema de contorno con una **combinación lineal finita de funciones conocidas**, que se llaman funciones básicas o de ensayo. Son relativamente simples: polinomios, trigonométricas o "splines". Se busca por tanto la mejor aproximación a la solución.

Conceptualmente, se intenta obtener la solución aproximada como la proyección de la solución (que se supone incluida en un espacio de dimensión infinita de funciones) en el espacio finito determinado por las funciones básicas.

La diferencia entre los métodos está en el criterio empleado para determinar los coeficientes c_k en la fórmula que representa dicha proyección:

$$u_n(t) = \sum_{k=1}^{n} c_k \phi_k(t) \tag{6.19}$$

supuesto que las funciones básicas (ϕ_k) cumplen las condiciones de contorno.

Esta es la forma en que se resuelven hoy en día la mayoría de problemas en derivadas parciales.

6.6.3.1 Método de colocación

Resulta al imponer a u_n que satisfaga la ecuación diferencial en los puntos $t_i \in [a, b]$ de una malla.

En general, transformamos el problema en el siguiente sistema

$$\sum_{k=1}^{n} c_k \phi_k''(t_i) = f\left(t_i, \sum_{k=1}^{n} c_k \phi_k(t_i), \sum_{k=1}^{n} c_k \phi_k'(t_i)\right) \qquad i = 1:n$$

Por tanto, dado el problema lineal

$$y'' = p(t)\, y'(t) + q(t)\, y(t) + r(t) \qquad y(a) = \alpha, y(b) = \beta$$

nos queda el sistema lineal (de incógnitas c_k)

$$\sum_{k=1}^{n} c_k \phi_k''(t_i) = p(t_i) \sum_{k=1}^{n} c_k \phi_k'(t_i) + q(t_i) \sum_{k=1}^{n} c_k \phi(t_i) + r(t_i) \qquad i = 1:n$$

Si no se satisfacen las condiciones de contorno, se añaden al sistema

$$\sum_{k=1}^{n} c_k \phi_k(a) = \alpha \qquad \sum_{k=1}^{n} c_k \phi_k(b) = \beta$$

6.6.3.2 Método de Galerkin

Se conoce también como método de los momentos.

Para resolver el problema, se impone que la **función residual**

$$r(t) = u_n''(t) - f\left(t, u_n, u_n'\right)$$

sea ortogonal, con respecto a un producto escalar, a todas las funciones básicas:

$$\langle r, \phi_k \rangle = 0 \quad k = 1:n$$

Para el mismo problema lineal que en el apartado anterior, y usando como producto escalar

$$\langle f, g \rangle = \int_a^b w(t)\, f(t)\, g(t)\, dt$$

Tratamos de minimizar $\langle r, r \rangle$, que será

$$\langle r, r \rangle = \min_{c_1, c_2, \ldots, c_n} \int_a^b w(t) \left(u''(t) - f\left(t, u, u'\right)\right)^2 dt$$

Para el residuo ortogonal a todas las funciones básicas se obtiene el sistema de ecuaciones normales

$$\langle r, \phi_j \rangle = 0 \qquad \forall j$$

Haciendo $w(t) = 1$ queda el sistema lineal siguiente

$$\int_a^b \left(\sum_{k=1}^{n} c_k \phi_k''(t) - p(t) \sum_{k=1}^{n} c_k \phi_k'(t) - q(t) \sum_{k=1}^{n} c_k \phi_k(t) \right) \phi_i(t)\, dt = \int_a^b r(t)\, \phi_i(t)\, dt \qquad i = 1:n$$

6.6.3.3 Método de Rayleigh-Ritz

Resulta de aplicar el principio variacional a problemas con ciertas propiedades determinando los coeficientes en base a un problema de optimización.

Aplicado al siguiente problema lineal

$$-\left(p(t)\, y'\right)' + q(t)\, y = r(t) \qquad y(0) = 0, y(1) = 0 \quad 0 \le t \le 1$$

$$\min_{v\in V}\int_0^1 \left(p\left(t\right)\left(v'\left(t\right)\right)^2+q\left(t\right)\left(v\left(t\right)\right)^2-2r\left(t\right)v\left(t\right)\right)dt$$

Siendo V el conjunto de funciones de clase 2 en $[0,1]$ que se anulan en 0 y en 1, por lo que se determinan los c_k^* tales que resuelven el sistema

$$\min_{c_1,c_2,..,c_k}\int_0^1 \left(p\left(t\right)\left(\sum_{k=1}^n c_k\phi_k'\left(t\right)\right)^2+q\left(t\right)\left(\sum_{k=1}^n c_k\phi_k\left(t\right)\right)^2-2r\left(t\right)\sum_{k=1}^n c_k\phi_k\left(t\right)\right)dt$$

que resulta en un sistema lineal.

Ejemplo 6.5:

Encontrar la solución al problema

$$y''=6t,\ y\left(0\right)=0,y\left(1\right)=1$$

que sea de la forma

$$u_3\left(t\right)=c_1+c_2t+c_3t^2$$

por el método de colocación.

Solución:

$$u_3''\left(t\right)=2c_3$$

tenemos un problema: el resultado es una constante, pero realmente debería depender de t. El método no nos valdría para este problema ya que no satisface la ecuación diferencial. Vamos a hacerlo de todas formas, imponiendo que se cumpla en un punto intermedio:

$$2c_3=6\cdot 0.5$$

$$c_3=1.5$$

Para imponer las condiciones de contorno hacemos

$$c_1=0$$
$$c_2+c_3=1$$

Resolviendo

$$c_2=-0.5$$
$$u_3^*\left(t\right)=-0.5t+1.5t^2$$

■

TEMA 7

Resolución de ecuaciones no lineales

7.1 Introducción

En este tema se trata de determinar los valores $\overline{x}$ (valores concretos) tales que $f(\overline{x}) = 0$. A dichos valores concretos se les denomina raíces o ceros de la ecuación $f(x) = 0$.

Esto es válido también para n variables y n funciones:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

También notado como

$$\vec{f}(\vec{x}) = 0$$

siendo 0 el vector cero y

$$\vec{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \quad \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Para resolverlos se usan, en general, métodos iterativos. Para ello, se construye una sucesión de valores $\{x_s\}$ que se pretende converja a una de las raíces que se busca.

$$s \to \infty \Rightarrow x_s \to \overline{x} / f(\overline{x}) = 0$$

Distinguimos además dos conceptos:

- **localizar raíces**: determinar los intervalos $[a_i, b_i]$ donde existe al menos una raiz
- **separar raíces**: lo mismo, pero asegurando que en el intervalo hay una y sólo una raiz. Por ejemplo, asegurando que la primera derivada es de un solo signo en el intervalo, por lo que es monótona

Es conveniente separar dos partes de la función que se puedan dibujar fácilmente por separado, y donde se corten habrá una raiz.

Para resolver ecuaciones no lineales se usan **métodos iterativos**, que consisten en construir una sucesión $\{x_s\}$ tal que

$$\lim_{s\to\infty} x_s \to \overline{x}$$

Los métodos básicos son:

- ✏ Métodos de intervalos encajados ("bracketing"), en los que se trabaja con un intervalo en el cual está contenida una raiz, determinando un subintervalo del anterior en el que siga estando la raiz, haciendo que la longitud del intervalo tienda a cero. Siempre que la longitud del intervalo tienda a cero, convergen, aunque sea de forma muy lenta.
- ✏ Métodos de punto fijo o aproximaciones sucesivas. Se tratan en la sección 7.3.1.

7.2 Métodos backeting

Se construye una sucesión de intervalos de forma que cada uno esté contenido en el anterior y que haya al menos una raiz en ellos:

$$[a_0, b_0] \supseteq [a_1, b_1] \supseteq \ldots \supseteq [a_n, b_n]$$
$$\overline{x} \in [a_i, b_i] \qquad \forall i$$

Converge si

$$\lim l\left([a_n, b_n]\right) = b_n - a_n = 0$$

siendo $l\left([a_n, b_n]\right)$ la longitud del intervalo $[a_n, b_n]$.

7.2.1 Método de bipartición

Dada una función f continua en un intervalo cerrado $[a, b]$ de forma que $f(a) f(b) < 0$, entonces en virtud del **Teorema de Bolzano**, existe un punto h contenido en el intervalo abierto (a, b) tal que $f(h) = 0$. Se supone por tanto que hay al menos una raiz de la función en el intervalo $[a, b]$ y que se producen cambios de signo.

Por tanto, la iteración consiste en, tras determinar el intervalo en el cual se encuentra la solución, $[a, b]$, escoger el punto medio $c = \frac{a+b}{2}$ y evaluar f en c:

- ✏ Si $f(a) f(c) < 0$ entonces el nuevo intervalo en el que está la raiz es $[a, c]$; es decir, hacemos $b = c$.
- ✏ En otro caso, el intervalo es $[c, b]$ ($a = c$).

El método es **convergente**, pero lento.

A continuación presentamos el código en Matlab para implementar este método.

Algoritmo 7.1: Método de bipartición para resolver ecuaciones no lineales

```
% Metodo de biparticion para resolucion de
% ecuaciones no lineales de una sola variable
% Uso: sol=bipart(funcion,extremoder,extremoizq,maxiter,tol)
function xs=bipart(f,a,b,iter,tol)
if nargin<5
```

```
   tol=0;
   if nargin<4
      disp('Error: faltan argumentos de entrada');
      return;
   end;
end;
% Dividir sucesivamente el intervalo para buscar la solucion
fin=0;
iteracion=0;
fizq=feval(f,a);
fder=feval(f,b);
if fizq*fder<0
   while (iteracion<iter) & (abs(b-a)>tol) & (fin==0)
      c=(a+b)/2;
      fcent=feval(f,c);
      if (fizq~=0)&(fder~=0)
         if fizq*fcent<0
            b=c;
            fder=fcent;
         elseif (fcent*fder<0)
            a=c;
            fizq=fcent;
         elseif (fder==0)|(fizq==0)
            if fizq==0
               xs=a;
            else
               xs=b;
            end;
            fin=1;
         end;
      end;
      iteracion=iteracion+1;
   end;
   if iteracion>=iter
      disp('Se ha llegado al numero maximo de iteraciones');
      xs=[a b];
   else
      if (fcent==0)|(a==b)
         disp('Se ha encontrado una solucion exacta');
         xs=c;
      else
         disp('Se ha encontrado un intervalo que cumple la
              tolerancia');
         xs=[a b];
      end;
   end;
else
   disp('Error: no existe ninguna solucion en el intervalo');
end;
```

♦

7.2.2 Método de la "regula falsi" (regla falsa)

Parte del mismo supuesto que el método de bipartición, pero determinando la recta que une los puntos $(b, f(b))$ y $(a, f(a))$, siendo c el punto de corte de la recta con el eje X:

$$c = \frac{bf(a) - af(b)}{f(a) - f(b)}$$

La iteración es la misma.

Puede ocurrir sin embargo que la longitud del intervalo no tienda a cero, así que para evitarlo puede escogerse en una iteración, en lugar del valor de la función en el extremo correspondiente, por ejemplo la mitad de dicho valor. Este sería el **método de la regla falsa modificada**. Se suele aplicar también cuando al hacer dos pasos uno de los extremos del intervalo se queda estancado (es el mismo en cada iteración), que puede ser señal de una convergencia muy lenta.

Vemos ahora el algoritmo implementado en Matlab.

Algoritmo 7.2: Método de la regula falsi para ecuaciones no lineales

```
% Metodo de la regula falsi para resolucion de
% ecuaciones no lineales de una sola variable
% Uso: sol=rfalsi(funcion,extizq,extder,maxiter,tol,error)
%
% El error es el valor maximo de diferencia entre el valor de la
% funcion en el punto y 0.
function xs=rfalsi(f,a,b,iter,tol,error)
if nargin<6
   error=0;
elseif nargin<5
   tol=0;
   if nargin<4
      disp('Error: faltan argumentos de entrada');
      return;
   end;
end;
% Dividir sucesivamente el intervalo para buscar la solucion
fin=0;
iteracion=0;
fizq=feval(f,a);
fder=feval(f,b);
if fizq*fder<0
   while (iteracion<iter) & (abs(b-a)>tol) & (fin==0)
      c=(b*fizq-a*fder)/(fizq-fder);
      fcent=feval(f,c);
      if (abs(fizq)>error)&(abs(fder)>error)
         if fizq*fcent<0
            b=c;
            fder=fcent;
         elseif (fcent*fder<0)
            a=c;
            fizq=fcent;
         end;
```

```
            else
               if abs(fizq)<=error
                  xs=a;
               else
                  xs=b;
               end;
               fin=1;
            end;
            iteracion=iteracion+1;
         end;
         if iteracion>=iter
            disp('Se ha llegado al numero maximo de iteraciones');
            xs=[a b];
         else
            if (abs(fcent)<error)|(a==b)
               disp('Se ha encontrado una solucion exacta');
               xs=c;
            else
               disp('Se ha encontrado un intervalo que cumple la
                    tolerancia');
               xs=[a b];
            end;
         end;
      else
         if fder==0
            xs=b;
         elseif fizq==0
            xs=a;
         else
            disp('Error: no existe ninguna solucion en el intervalo');
         end;
      end;
```

♦

7.2.3 Método de Pegasus

Consiste en aplicar el método de la regula falsi modificada multiplicando la evaluación de la función por α en la iteración $k+1$ siendo

$$\alpha = \frac{f(x_k)}{f(x_k) + f(x_{k+1})} \tag{7.1}$$

7.2.4 Método de Illinois

De nuevo se aplica el método de la regula falsi modificada con

$$\alpha = 0.5 \tag{7.2}$$

Ahora se incluye el código en Matlab para implementar la regula falsi modificada, pudiendo escoger entre Pegasus e Illinois.

Algoritmo 7.3: Método de la regula falsi modificada para resolver ecuaciones no lineales

```
% Metodo de la regula falsi modificada para resolucion de
% ecuaciones no lineales de una sola variable
% Se selecciona método de Pegasus o Illinois
% Uso:
%  sol=rfalsi(funcion,extizq,extder,maxiter,metodo,tol,error)
%
% Metodo puede ser 'Pegasus' o 'Illinois'
%
% El error es el valor maximo de diferencia entre el valor de la
% funcion en el punto y 0.
function xs=rfalsimod(f,a,b,iter,metodo,tol,error)
if nargin<7
   error=0;
elseif nargin<6
   tol=0;
   if nargin<5
      metodo = 'Pegasus';
   end;
   if nargin<4
      disp('Error: faltan argumentos de entrada');
      return;
   end;
end;
% Dividir sucesivamente el intervalo para buscar la solucion
fin=0;
iteracion=0;
fizq=feval(f,a);
fder=feval(f,b);
fant=fizq;
if fizq*fder<0
   while (iteracion<iter) & (abs(b-a)>tol) & (fin==0)
      c=(b*fizq-a*fder)/(fizq-fder);
      fcent=feval(f,c);
      if strcmp(metodo,'Pegasus')==1,
         alfa = fant/(fant+fcent);
      else
         if strcmp(metodo,'Illinois')==1,
            alfa = 0.5;
         else
            error('El metodo seleccionado es desconocido');
         end;
      end;
      if (abs(fizq)>error)&(abs(fder)>error)
         if fizq*fcent<0
            b=c;
            fder=fcent;
            if fcent*fant>0,
               fizq = alfa*fizq;
            end;
```

```
            elseif (fcent*fder<0)
               a=c;
               fizq=fcent;
               if fcent*fant>0,
                  fder = alfa*fder;
               end;
            end;
         else
            if abs(fizq)<=error
               xs=a;
            else
               xs=b;
            end;
            fin=1;
         end;
         iteracion=iteracion+1;
      end;
      if iteracion>=iter
         disp('Se ha llegado al numero maximo de iteraciones');
         xs=[a b];
      else
         if (abs(fcent)<error)|(a==b)
            disp('Se ha encontrado una solucion exacta');
            xs=c;
         else
            disp('Se ha encontrado un intervalo que cumple la
                 tolerancia');
            xs=[a b];
         end;
      end;
   else
      if fder==0
         xs=b;
      elseif fizq==0
         xs=a;
      else
         disp('Error: no existe ninguna solucion en el intervalo');
      end;
   end;
```

♦

7.3 Métodos de aproximaciones sucesivas

En estos métodos se calcula cada iteración como función de las iteraciones anteriores, obteniéndose dos tipos de métodos:

- ✏ De punto fijo

$$x_{s+1} = g\left(x_s\right)$$

✏ Multipunto

$$x_{s+1} = g\left(x_s, x_{s-1}, \ldots, x_{s-m}\right)$$

7.3.1 Métodos de iteración de punto fijo

Este tipo de métodos son los que, para una ecuación $f\left(x\right) = 0$ con f definida real, establecen una sucesión de aproximaciones del valor de x para el que se cumple la ecuación, x^* tal que

$$x_{s+1} = g\left(x_s\right), \qquad s = 0, 1, \ldots$$

Dado x_0 valor inicial.

Estos son los llamados métodos unipunto, mientras que los multipunto son aquellos en los que

$$x_{s+1} = g\left(x_s, x_{s-1}, \ldots, x_{s+1-m}\right), \qquad s = m, m+1, \ldots$$

Dados $x_0, x_1, \ldots, x_{m-1}$ valores iniciales.

Y si existe el límite

$$x^* = \lim_{s\to\infty} x_s$$

si g es continua, entonces se tiene que x^* es un punto fijo:

$$x^* = \lim_{s\to\infty} g\left(x_s\right) = g\left(\lim_{s\to\infty} x_s\right) = g\left(x^*\right)$$

Un punto fijo es aquel en el que la función es el mismo punto ($x = y$). Por tanto, será el punto de corte de la función $y_1 = g\left(x\right)$ y la función $y_2 = x$.

Estos métodos se llaman también de aproximaciones sucesivas.

Los métodos de punto fijo han de ser **consistentes** con el problema que se quiere resolver, es decir, si queremos calcular una raiz de la ecuación $f\left(x\right) = 0$, la raiz debe ser punto fijo de la función g que escojamos.

Por ejemplo, si tenemos

$$x_{s+1} = x_s + f\left(x_s\right)$$

El método es consistente, ya que en x^*, la función es nula y se tiene $x^* = x^*$, o lo que es lo mismo, $x^* = g\left(x^*\right)$.

Ejemplo 7.1:

$$x^3 + x \operatorname{sen} x + 1 = 0$$

Se pueden generar los siguientes métodos:

$$x_{s+1} = \frac{-1}{x_s^2 + \operatorname{sen} x_s}$$

$$x_{s+1} = -\frac{x_s^3 + 1}{\operatorname{sen} x_s}$$

$$x_{s+1} = \operatorname{arc\,sen}\left(-\frac{x_s^3 + 1}{x_s}\right)$$

■

7.3.2 Método de la tangente o de Newton

Este es uno de los métodos más utilizados.

Se basa en calcular la recta tangente a la curva de la función f en el punto $(x_s, f(x_s))$ y el punto x_{s+1} es el punto en que dicha recta tangente corta al eje X. Es decir, teniendo la recta tangente a una función f en un punto x_s

$$y = f(x_s) + f'(x_s)(x - x_s)$$

si se iguala dicha recta a cero, despejando el punto x, se tiene

$$\boxed{x_{s+1} = x_s - \frac{f(x_s)}{f'(x_s)}} \tag{7.3}$$

El inconveniente que tiene es tener que evaluar la derivada de la función. Una simplificación posible es sustituir la derivada por el cociente incremental, con lo que se obtiene el **método de la secante**:

$$\boxed{x_{s+1} = x_s - \frac{f(x_s)(x_s - x_{s-1})}{f(x_s) - f(x_{s-1})}} \tag{7.4}$$

El cual es un método multipunto y además se corre el peligro que uno de los puntos en la iteración se nos vaya a infinito, o muy lejos.

Vemos a continuación la implementación en Matlab de los métodos de Newton y de la secante para ecuaciones no lineales.

Algoritmo 7.4: Método de Newton para ecuaciones no lineales

```
% Metodo de Newton para resolucion de
% ecuaciones no lineales
%
% Uso: [sol,error]=enewton(funcion,xinicial,maxiter,tol,metodo)
%
% metodo es el metodo de derivacion numerica que se usara.
% Las opciones son: deriv2p, derivp3p, derivr3p, derivp3p,
% derivp5p
% Por defecto es deriv2p, pero se puede especificar una funcion
% que contenga la derivada funcional.
function [xs,errs]=enewton(f,x0,maxiter,tol,metodo)
if nargin<5
   metodo='deriv2p';
   if nargin<4
      tol=1e-5;
   end;
   if nargin<3
      disp('Error: faltan argumentos de entrada');
      return;
   end;
end;
h=0.01;
```

```
iteraciones=0;
fin=0;
x(1)=x0;
while (iteraciones<maxiter)&(fin==0)
   iteraciones=iteraciones+1;
   fx=feval(f,x(iteraciones));
   fp=feval(metodo,f,x(iteraciones),h);
   x(iteraciones+1)=x(iteraciones)-(fx/fp);
   errs=abs(x(iteraciones+1)-x(iteraciones));
   if errs<tol
     fin=1;
   end;
end;
if fin==0
   disp('Se ha llegado al numero maximo de iteraciones');
else
   disp('Se ha encontrado solucion que cumple la tolerancia');
end;
xs=x(iteraciones+1);
```

♦

Algoritmo 7.5: Método de la secante para resolver ecuaciones no lineales

```
% Metodo de la secante para resolucion de
% ecuaciones no lineales de una sola variable
%
% Uso: sol=secante(funcion,xinicial,maxiter,tol)
function [xs,errs]=secante(f,x0,maxiter,tol)
if nargin<4
   tol=1e-5;
   if nargin<3
      disp('Error: faltan argumentos de entrada');
      return;
   end;
end;
% Obtener el primer punto con el metodo de Newton
iteraciones=0;
fin=0;
fx=feval(f,x0);
fp=feval('deriv2p',f,x0,0.01);
x(1)=x0;
x(2)=x(1)-(fx/fp);
iteraciones=iteraciones+1;
while (iteraciones<maxiter)&(fin==0)
   iteraciones=iteraciones+1;
   fx=feval(f,x(iteraciones));
   fp=fx-feval(f,x(iteraciones-1));
   x(iteraciones+1)=x(iteraciones)-eval(f,x(iteraciones));
   errs=abs(x(iteraciones+1)-x(iteraciones));
   if errs<tol
     fin=1;
```

```
   end;
end;
if fin==0
   disp('Se ha llegado al numero maximo de iteraciones');
else
   disp('Se ha encontrado solucion que cumple la tolerancia');
end;
xs=x(iteraciones+1);
```

♦

7.3.3 Método de Steffensen

En este caso se sustituye la derivada por un cociente incremental "rarito", que es

$$f'(x_s) = \frac{f(x_s + f(x_s)) - f(x_s)}{x_s + f(x_s) - x_s} = \frac{f(x_s + f(x_s)) - f(x_s)}{f(x_s)}$$

con lo que resulta

$$\boxed{x_{s+1} = x_s - \frac{(f(x_s))^2}{f(x_s + f(x_s)) - f(x_s)}} \tag{7.5}$$

Y el código en Matlab que implementa este método se ve a continuación.

Algoritmo 7.6: Método de Steffensen para resolver ecuaciones no lineales

```
% Metodo de Steffensen para resolucion de
% ecuaciones no lineales de una sola variable
%
% Uso: sol=steffensen(funcion,xinicial,maxiter,tol)
function [xs,errs]=steffensen(f,x0,maxiter,tol)
if nargin<4
   tol=1e-5;
   if nargin<3
      disp('Error: faltan argumentos de entrada');
      return;
   end;
end;
% Obtener el primer punto con el metodo de Newton
iteraciones=0;
fin=0;
fx=feval(f,x0);
fp=feval('deriv2p',f,x0,0.01);
x(1)=x0;
x(2)=x(1)-(fx/fp);
iteraciones=iteraciones+1;
while (iteraciones<maxiter)&(fin==0)
   iteraciones=iteraciones+1;
   fx=feval(f,x(iteraciones));
   fp=feval(f,x(iteraciones-1)+fx)-fx;
   x(iteraciones+1)=x(iteraciones)-(fx^2/fp);
```

```
        errs=abs(x(iteraciones+1)-x(iteraciones));
        if errs<tol
          fin=1;
        end;
      end;
      if fin==0
        disp('Se ha llegado al numero maximo de iteraciones');
      else
        disp('Se ha encontrado solucion que cumple la tolerancia');
      end;
      xs=x(iteraciones+1);
```

♦

7.3.4 Método de Halley

Este método no es tan importante, pero lo mencionamos. En este caso, la iteración es

$$\boxed{x_{s+1} = x_s - \frac{2f(x_s) f'(x_s)}{2(f'(x_s))^2 - f(x_s) f''(x_s)}} \tag{7.6}$$

Como se ve, hay que hacer la derivada y la derivada segunda, lo que tiene bastante coste numérico.

7.4 Condiciones de salida

Para los algoritmos computacionales se pueden dar unas condiciones de salida:

- En los bracketing, siendo $l([a_k, b_k])$ la longitud del intervalo:

$$l([a_k, b_k]) < \varepsilon$$

- En los de aproximaciones sucesivas

$$|x_s - x_{s-1}| < \varepsilon$$

- También para los de aproximaciones sucesivas

$$|f(x_s)| < \varepsilon$$

  El cual hay que usar con cuidado, ya que puede ocurrir que la función llegue a estar muy próxima al eje y el cero alejado del punto donde encontramos que se cumple la condición, o incluso que no se llegue a tener un cero.

- Se suele usar además una condición de que se alcance un número máximo de iteraciones

7.5 Teoremas de convergencia

Los teoremas que vamos a ver aquí son aplicaciones del teorema del punto fijo que vemos ahora:

Teorema del punto fijo:

Si g es una aplicación contractiva en un espacio normado y completo $\mathbb{X}$, entonces existe un punto fijo de g en $\mathbb{X}$.

✏ **Aplicación contractiva**: una aplicación f se dice contractiva si es tal que

$$\|f(x) - f(y)\| \leq L \|x - y\| \qquad \forall x, y \in \mathbb{X} \qquad \text{con } L < 1$$

es decir, si f es de Lipschitz

✏ En un **espacio normado completo** toda sucesión de Cauchy tiene límite.

Teorema 7.1. *(teorema de convergencia global)* Dada una función real definida en un intervalo $[a, b]$ tal que

1. $g([a, b]) \subseteq [a, b]$
2. g derivable en $[a, b]$ con $\left|g'(x)\right| \leq K < 1$ para todo $x \in [a, b]$

en ese caso, la iteración de punto fijo unipunto es convergente, para cualquier punto x_0 inicial del intervalo, al único punto fijo x^* de g en el intervalo, verificándose la estimación de error

$$|x_{s+1} - x^*| \leq \frac{K^{s-r}}{1-K} |x_{r+1} - x_r| \qquad r = 1 : s$$

Es decir, la unicidad del punto fijo se puede garantizar si la imagen no se sale del espacio (es aplicación contractiva). El teorema asegura que el método converge cualquiera que sea el x_0 que se elija, es una **condición suficiente**, no necesaria.

La segunda condición puede ser sustituida por que la función g sea contractiva.

Demostración. *Demostremos que si la derivada es menor o igual que K, es una aplicación contractiva:*

En virtud del Teorema del Valor Medio,

$$g(x) - g(y) = g'(c)(x - y) \qquad x < c < y$$

$$\begin{aligned} |g(x) - g(y)| &\leq \left|g'(c)\right| |x - y| \\ &\leq K |x - y| \qquad K < 1 \end{aligned}$$

Demostrado.

Ahora demostremos la unicidad del punto fijo:

Suponemos que hay dos puntos fijos x_1^* y x_2^*. En ese caso suponemos que existen dos puntos fijos x_1^* y x_2^*, teniéndose

$$|g(x_1^*) - g(x_2^*)| \leq K |x_1^* - x_2^*| < |x_1^* - x_2^*|$$

Dado que $K < 1$.

Sin embargo, se llega a una contradicción ya que

$$|g(x_1^*) - g(x_2^*)| < |x_1^* - x_2^*|$$

y por la propia definición de punto fijo es

$$|g(x_1^*) - g(x_2^*)| = |x_1^* - x_2^*|$$

así que no puede haber más de un punto fijo.

Teorema 7.2. *(teorema de convergencia local)* Dados $g \in C^1[a,b]$ y un punto fijo de g, $x^* \in [a,b]$. Si $|g'(x^*)| < 1$, existe un valor $\varepsilon > 0$ tal que la iteración de punto fijo converge a x^* para cualquier x_0 tal que $|x_0 - x^*| < \varepsilon$.

Demostración. *Podemos escribir*

$$x_{s+1} = g(x_s) = g(x^*) + g'(c)(x_s - x^*)$$

con lo que las cotas de error nos quedan

$$|x_{s+1} - x^*| \leq \underbrace{|g'(c)|}_{<1} |x_s - x^*| < |x_s - x^*|$$

por tanto, el error en la iteración es menor que en la iteración anterior, lo que significa que el método convergen a la solución.

Un problema que nos encontramos al aplicar el teorema es que no conocemos el valor de x^* ya que es lo que buscamos. Por tanto, no podemos conocer $g'(x^*)$ ni verificar que $|g'(x^*)| < 1$. Sin embargo, si en el intervalo en el que está x^* la derivada en módulo es menor que 1, en x^* también lo será. A la inversa también: si $|g'(x^*)| < 1$, en un entorno de x^* la derivada de g es menor que 1. Por esto, a x^* se le suele llamar **punto de atracción**.

Por otro lado, salvo casualidad, el método diverge si $|g'(x^*)| > 1$, y si $|g'(x^*)| = 1$, entonces hay que acudir al siguiente término del desarrollo en series de Taylor, algo que no vamos a ver.

Teorema 7.3. Sea la función real g definida y contractiva en

$$B(x_0, \rho) = \{x \in \mathbb{R} \,/\, |x - x_0| \leq \rho\}$$

y tal que

$$|x_1 - x_0| \leq (1 - K)\rho$$

siendo $K < 1$ la constante de contractividad, la iteración de punto fijo $x_{s+1} = g(x_s)$ converge a un único punto fijo de g en $B(x_0, \rho)$.

Con este teorema, se evita la comprobación de $g([a,b]) \subseteq [a,b]$

Para determinar el número de iteraciones necesarias para obtener t dígitos significativos usamos la cota de error relativo a priori:

$$\boxed{\frac{|x_s - x^*|}{|x^*|} \leq \frac{K^s}{1-K} \frac{|x_1 - x_0|}{|x^*|} < \frac{1}{2} 10^{1-t}} \tag{7.7}$$

sustituyendo $|x^*|$ por una cota inferior y despejando s.

Si queremos t dígitos exactos, usamos la cota de error absoluto.

En caso de que queramos obtener la solución con un número de dígitos significativos o exactos, se usa la cota de error a posteriori

$$\boxed{|x_s - x^*| \leq \frac{K}{1-K} |x_s - x_{s-1}|} \tag{7.8}$$

Y se va aplicando conforme se hace la recursión, aunque es recomendable aplicarla por primera vez tras unas cuantas iteraciones, no al comenzar. Con esta cota se obtiene un número de iteraciones menor que con la de error a priori, por lo que es mejor si se quiere calcular la solución en lugar de una estimación del número de iteraciones.

Ejemplo 7.2:

Dado el método

$$g(x) = 0.5\, e^x \cos x$$

estudiar si converge para todo $x_0 \in [0,1]$, y si converge para $x_0 = 0$, estudiar el número de iteraciones necesarias para conseguir 6 decimales exactos.

Solución:

$$g'(x) = 0.5\, e^x (\cos x - \operatorname{sen} x)$$
$$g''(x) = -0.5\, e^x\, 2 \operatorname{sen} x$$

para $x \in [0,1]$

$$g''(x) \leq 0$$

por lo que g' es monótona decreciente en $(0,1]$. Además

$$g'(0) = 0.5$$
$$g'(1) = -0.5 \times 2.7182 \times 0.3012 > -0.5$$

es decir,

$$\left|g'(x)\right| \leq 0.5 < 1 \qquad \forall x \in [0,1]$$

por lo que $|g'(x^*)| < 1$

Veamos que g es contractiva:

$$g'(x) = 0 \Rightarrow x = \frac{\pi}{4}$$
$$g\left(\frac{\pi}{4}\right) = 0.5 \times e^{\frac{\pi}{4}} \times \frac{\sqrt{2}}{2} < 0.5 \times e \times \frac{\sqrt{2}}{2} = \frac{2.718}{2.828} < 1$$

así que como es contractiva, podemos asegurar la convergencia global para $[0,1]$.

Veamos ahora qué pasa con el error absoluto:

$$|x_s - x^*| \leq \frac{K^s}{1-K} |x_1 - x_0| \leq \frac{1}{2} 10^{-6}$$
$$x_1 = g(0) = 0.5 \qquad K = 0.5$$

llegamos a

$$\frac{\left(\frac{1}{2}\right)^s}{\frac{1}{2}} \left(\frac{1}{2} - 0\right) \leq \frac{1}{2} 10^{-6}$$

despejando s

$$s = 21$$

■

7.6 Orden de un método iterativo

El orden de un método, suponiendo que converge, sirve para comparar la rapidez en la convergencia de los métodos (o para medir la rapidez de uno).

Se dice que un método iterativo es convergente de orden p:

1. $p = 1$ si el $\lim_{s\to\infty} \dfrac{|x_{s+1} - \alpha|}{|x_s - \alpha|} = C$ siendo $0 < C < 1$ (el error en cada iteración es menor que en la anterior)
2. $p > 1$ si el $\lim_{s\to\infty} \dfrac{|x_{s+1} - \alpha|}{|x_s - \alpha|^p} = C$ con $0 < C < \infty$

La convergencia es

- **Lineal** si $p = 1$
- **Superlineal** si $1 < p < 2$
- **Cuadrática** si $p = 2$
- **Cúbica** si $p = 3$

El orden de un método da una estimación de lo eficiente del método y posibilita comparar los métodos entre ellos como ya hemos dicho, además de facilitar elecciones razonables del número de iteraciones para que un proceso iterativo termine.

Para el caso de las iteraciones de punto fijo, se dice que son de orden p:

1. $p = 1$ si g admite derivada continua en un entorno de un punto fijo α y $0 < |g'(\alpha)| < 1$, ya que

$$x_{s+1} = \alpha + g'(c)(x_s - \alpha)$$

$$\lim_{s\to\infty} \frac{|x_{s+1} - \alpha|}{|x_s + \alpha|} = \lim_{s\to\infty} |g'(c)| = |g'(\alpha)|$$

2. $p > 1$ si g admite derivadas continuas hasta la de orden $p > 1$ en un entorno de un punto fijo α y

$$g'(\alpha) = g''(\alpha) = g'''(\alpha) = \ldots = g^{(p-1)}(\alpha) = 0, \qquad g^{(p)}(\alpha) \neq 0$$

ya que

$$\lim_{s\to\infty} \frac{|x_{s+1} - \alpha|}{|x_s - \alpha|^p} = \frac{1}{p!} \left|g^{(p)}(\alpha)\right|$$

El **error en la iteración siguiente** es

$$|x_{s+1} - \alpha| \simeq \frac{\left|g^{(p)}(\alpha)\right|}{p!} |x_s - \alpha|^p$$

Por tanto, en cuanto al error, interesan métodos de orden elevado, aunque tienen mayor coste de cálculo por iteración.

Los tipos de los métodos más habituales son:

- ✏ **Lineales**: métodos de bipartición y de la regla falsa.
- ✏ **Superlineales**: métodos de la regla falsa modificada (Illinois ($p = 1.442$), Pegasus ($p = 1.648$)), de la secante ($p = \left(1 + \sqrt{5}\right)/2$) y Muller ($p = 1.839$).
- ✏ **Cuadrático**: Newton y Steffensen
- ✏ **De orden 3**: Halley

Ejemplo 7.3:

Comprobar que el método de Newton es cuadrático:

$$x_{s+1} = x_s - \frac{f(x_s)}{f'(x_s)}$$

Suponemos que $f \in C^2[a,b]$ y $\alpha / f(\alpha) = 0, f'(\alpha) \neq 0$

Calculamos la derivada de $g(x) = x - \frac{f(x)}{f'(x)}$:

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

Sustituimos x por α:

$$g'(\alpha) = 1 - 1 + \frac{f(\alpha) f''(\alpha)}{(f'(\alpha))^2} = \frac{f(\alpha) f''(\alpha)}{(f'(\alpha))^2} = 0$$

ya que $f(\alpha) = 0$.

Por tanto, en un entorno de α, la convergencia está asegurada, y es de orden 2 salvo para valores muy concretos de α en los que puede ser de orden superior. Es localmente convergente.

■

Hemos mencionado el **método de Muller** anteriormente, pero aún no lo hemos visto. Hagámoslo pues:

En este método se traza una parábola de interpolación que pasa por 3 puntos concretos x_s, x_{s-1} y x_{s-2}. La parábola queda

$$p_2(x) = A(x - x_s)^2 + B(x - x_s) + C$$

cumpliendo las condiciones

$$p_2(x_s) = f(x_s)$$

$$p_2(x_{s-1}) = f(x_{s-1})$$

$$p_2(x_{s-2}) = f(x_{s-2})$$

Se determinan A, B y C a partir de las tres condiciones y se resuelve la ecuación

$$p_2(x_{s+1}) = 0$$

que puede hacerse de forma más sencilla como

$$Ah^2 + Bh + C = 0$$

y teniendo en cuenta que $x_{s+1} = x_s + h$. Es decir, que el siguiente punto de la iteración es el punto de corte de la parábola con el eje X.

Una **recomendación** para resolver ecuaciones no lineales es estimar un punto de inicio x_0 mediante algún método de bipartición y continuar con un método de punto fijo.

7.6.1 Acelerar la convergencia

Dado un método de punto fijo que converge lentamente,

$$x_{s+1} = g\left(x_s\right)$$

se puede modificar de forma que su convergencia sea más rápida. Por ejemplo, sumando a cada término kx:

$$kx + x = kx + g\left(x\right)$$

$$kx_{s+1} + x_{s+1} = kx_s + g\left(x_s\right)$$

despejando x_{s+1}

$$x_{s+1} = \frac{kx_s + g\left(x_s\right)}{1+k} = h\left(x_s\right)$$

La pregunta del millón: ¿cuánto vale k?

La respuesta es que se escoge de forma que el método sea convergente con algún criterio. Por ejemplo, podemos escoger que sea de orden 2, con lo que la primera derivada de $h\left(x\right)$ tiene que anularse en un punto α:

$$h'\left(x\right) = \frac{k + g'\left(x\right)}{1+k}$$

y para que se anule en α:

$$-g'\left(\alpha\right) = k$$

Otra forma de acelerar la convergencia es con el **método Δ^2 de Aitken**,

$$\tilde{x}_s = x_s - \frac{\left(\Delta x_s\right)^2}{\Delta^2 x_s}$$

que, usando diferencias finitas

$$\tilde{x}_s = x_s - \frac{\left(x_{s+1} - x_s\right)^2}{x_{s+2} - 2x_{s+1} + x_s} = x_s - \frac{x_s x_{s+2} - x_{s+1}^2}{x_{s+2} - 2x_{s+1} + x_s} \tag{7.9}$$

Esto es, teniendo una sucesión $x_0, x_1, \ldots, x_s \rightarrow \alpha$, obtenemos una sucesión $\tilde{x}_0, \tilde{x}_1, \ldots, \tilde{x}_s \rightarrow \alpha$, tal que se puede demostrar que cumple

$$\lim_{s\rightarrow\infty} \frac{|\tilde{x}_s - \alpha|}{|x_s - \alpha|} = 0$$

lo que implica que $\tilde{x}_s$ tiende más rápido a α que x_s, y por tanto el método es más rápido.

La aplicación del método es la siguiente:

- ✏ Con un método típico de punto fijo, obtener una sucesión de soluciones hasta un x_s
- ✏ Aplicar el método de Aitken a los 3 últimos valores, x_{s-2}, x_{s-1} y x_s.
- ✏ Con el nuevo valor de $\tilde{x}_s$ se sigue aplicando el mismo método de punto fijo de antes.

Lo que hacemos es dar un "salto" para acelerar la convergencia en pasos intermedios.

Si lo hiciésemos al obtener los 3 primeros valores, y de forma continua, tenemos el caso límite, que coincide con el método de Steffensen visto en la sección 7.3.3.

7.6.2 Inestabilidad o mal condicionamiento del problema

Tenemos un problema de ecuación no lineal

$$f(x) = 0 \qquad \overline{x}/f(\overline{x}) = 0$$

Si perturbamos el problema

$$f_\mu(x) = f(x) + \mu g(x) = 0$$

la solución es

$$\overline{x}_\mu/f(\overline{x}_\mu) + \mu g(\overline{x}_\mu) = 0$$

y la diferencia entre la solución del problema original y el perturbado

$$|\overline{x}_\mu - \overline{x}| \simeq \mu \left| \frac{g(\overline{x})}{f'(\overline{x})} \right|$$

En el caso que $f'(\overline{x}) \to 0$, la diferencia entre ambas soluciones es muy grande, aunque μ sea pequeño. Esto ocurre con raíces múltiples.

Por tanto, la ecuación es mal condicionada para el cálculo de la raiz $\overline{x}$ si es raiz múltiple o tiene otras raíces próximas. El concepto de proximidad es muy relativo.

Por ejemplo, el método de Newton deja de ser cuadrático con raíces múltiples.

Para arreglarlo, se modifica el método:

$$\begin{gathered} f'(\overline{x}) = 0 \\ f''(\overline{x}) = 0 \\ \overline{x}/ \\ \vdots \\ f^{(m-1)}(\overline{x}) = 0 \end{gathered}$$

con lo que $\overline{x}$ es de multiplicidad m. El cambio en el método es

$$x_{s+1} = x_s - m\frac{f(x_s)}{f'(x_s)}$$

Entonces sí es cuadrático.

7.7 Sistemas de ecuaciones no lineales

Ahora el problema se amplía a calcular las soluciones para múltiples funciones y múltiples variables

$$\begin{aligned} f_1(x_1 \dots x_n) &= 0 \\ &\vdots \\ f_n(x_1 \dots x_n) &= 0 \end{aligned}$$

Tratamos por tanto de determinar el vector

$$\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

Tal que $\vec{f}(\vec{x}) = 0$, siendo $\vec{f} : \mathbb{R}^n \to \mathbb{R}^n$ y

$$\vec{f} = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}$$

7.7.1 Iteración de punto fijo para sistemas de ecuaciones no lineales

Con un método iterativo de punto fijo se calcula una sucesión de vectores que notaremos $x^{(s)}$, omitiendo los vectores por simplicidad. La sucesión es de la forma

$$x^{(s+1)} = g\left(x^{(s)}\right) \qquad s = 1, 2, \dots$$

dado un vector inicial $x^{(0)}$ y siendo $g : \mathbb{R}^n \to \mathbb{R}^n$.

Si existe

$$\lim_{s \to \infty} x^{(s)} = x^*$$

entonces $x^* = \vec{g}(x^*)$ es punto fijo de $\vec{g}$, y el método es consistente con el problema a resolver.

El vector de funciones $\vec{g}$ se puede montar despejando una incógnita de cada ecuación, por ejemplo.

Ejemplo 7.4:

$$x_1^2 + x_2^2 - 2 = 0$$

$$x_1 x_2 + x_2^3 + 3x_1^2 - 4 = 0$$

Despejando x_1 de la primera y x_2 de la segunda

$$x_1 = \sqrt{2 - x_2^2}$$

$$x_2 = \sqrt{\frac{4 - 3x_1^2}{x_1 + x_2^2}}$$

Otra forma podría ser sumando a cada lado de cada ecuación la incógnita que se quiere despejar:

$$x_1 = x_1^2 + x_2^2 + x_1 - 2$$

$$x_2 = x_2 + x_1 x_2 + x_2^3 + 3x_1^2 - 4$$

■

El espacio $\mathbb{R}^n$ es un espacio vectorial normado de dimensión finita y todo conjunto cerrado es completo.

El **teorema de punto fijo** da la existencia de un único punto fijo de $g : \mathbb{H} \to \mathbb{H}$ siendo $\mathbb{H}$ un subconjunto cerrado de $\mathbb{R}^n$ si g es contractiva en $\mathbb{H}$. En este caso, se puede demostrar el siguiente teorema:

Teorema 7.4. Dado $g : \mathbb{H} \to \mathbb{H}$ siendo $\mathbb{H}$ un subconjunto cerrado de $\mathbb{R}^n$. Si g es una aplicación contractiva en $\mathbb{H}$, la iteración de punto fijo es convergente para cualquier $x^{(0)} \in \mathbb{H}$ al único punto fijo x^* de g en $\mathbb{H}$. Además, se verifica la estimación de error:

$$\left\| x^{(s)} - x^* \right\| \leq \frac{K^{s-r}}{1-K} \left\| x^{(r+1)} - x^{(r)} \right\|$$

con $0 \leq r < s$.

El uso de cotas de error es similar al caso de una variable, solo que hay que usar normas, y estas pueden ser diferentes. Los conjuntos cerrados que se suelen escoger son intervalos n-dimensionales o entornos cerrados de un punto $\overline{x}$.

La contractividad para g en $\mathbb{H}$ se puede demostrar si g admite derivadas parciales primeras y si para la Jacobiana J_g podemos establecer

$$\|J_g(x)\| \leq K < 1 \qquad \forall x \in \mathbb{H}, \qquad J_g(x) = \left[\frac{\partial g_i(x)}{\partial x_j} \right]$$

en una norma matricial cualquiera que sea subordinada a la vectorial usada. Esta condición también se puede escribir de la siguiente forma si existe una constante $K < 1$ tal que se cumpla

$$\left| \frac{\partial g_i(x)}{\partial x_j} \right| \leq \frac{K}{n}$$

para todo $x \in \mathbb{H}; i, j = 1 : n$.

De esta última forma, se simplifica la verificación de contractividad, sobre todo si lo que tenemos son muchas ecuaciones, ya que es más sencillo verificarlas una por una que realizar la norma de una matriz muy grande.

Ejemplos típicos del conjunto $\mathbb{H}$ son

$$\mathbb{H} = \{x \in \mathbb{R}^n / a_i \leq x_i \leq b_i, \forall i\}$$

$$\mathbb{H} = \left\{x \in \mathbb{R}^n / \left\| x - x^{(0)} \right\| \leq \rho\right\}$$

7.7.2 Método de Newton para sistemas de ecuaciones no lineales

El método de Newton es fundamental en la resolución de sistemas no lineales. Tiene además un comportamiento fractal, lo que significa que su convergencia depende en gran medida del vector origen $x^{(0)}$.

Este método se define por la ecuación de recurrencia

$$x^{(s+1)} = x^{(s)} - J_f^{-1}\left(x^{(s)}\right) f\left(x^{(s)}\right) \qquad s = 0, 1, 2, \ldots$$

suponiendo que existe la inversa de la jacobiana J_f en los puntos en los que sea necesario. En la práctica, no se calcula la inversa de la matriz jacobiana, sino que se sigue el siguiente procedimiento:

1. Evaluar $b = -f\left(x^{(s)}\right)$ y $A = J_f\left(x^{(s)}\right)$
2. Resolver el sistema lineal $Ad^{(s)} = b$
3. La siguiente iteración es $x^{(s+1)} = x^{(s)} + d^{(s)}$

Por tanto, lo que se hace es resolver el sistema lineal, por alguno de los métodos ya estudiados en el tema 2.

Esto se puede obtener de

$$z_i = \left(\nabla f_i\left(x^{(s)}\right)\right)^t \left(x - x^{(s)}\right) + f_i\left(x^{(s)}\right)$$

que es la interpretación geométrica del método, en base a los hiperplanos tangentes en $x^{(s)}$.

Bajo los siguientes supuestos, el método es localmente convergente y cuadrático:

- ✏ la jacobiana es invertible en el punto solución del sistema
- ✏ la jacobiana es de Lipschitz en un entorno del punto solución del sistema

Esto es justamente lo que dice el siguiente teorema.

Teorema 7.5. Dado $x^* \in \mathbb{R}^n$ tal que $f(x^*) = 0$ siendo f continuamente diferenciable en una bola $B(x^*, \rho)$, y suponiendo que

- ✏ $J_f(x^*)$ es invertible y $\left\|J_f^{-1}(x^*)\right\| \leq \beta$
- ✏ $\|J_f(x) - J_f(y)\| \leq \gamma \|x - y\|$ para todo $x, y \in B(x^*, \rho)$

entonces existe $\varepsilon > 0$ de forma que el método de Newton converge a x^* para cualquier $x^{(0)} \in B(x^*, \varepsilon)$ y se cumple

$$\left\|x^{(s+1)} - x^*\right\| \leq \gamma\beta \left\|x^{(s)} - x^*\right\|^2$$

para $s = 0, 1, 2, \ldots$

Teorema 7.6. Dada f continuamente diferenciable en una bola $B\left(x^{(0)}, \rho\right)$, y suponiendo que existen unas constantes α, β y γ tales que

- ✏ $\left\|J_f^{-1}\left(x^{(0)}\right)\right\| \leq \alpha$, $\left\|J_f^{-1}\left(x^{(0)}\right) f\left(x^{(0)}\right)\right\| \leq \beta$
- ✏ $\|J_f(x) - J_f(y)\| \leq \gamma \|x - y\|$ para todo $x, y \in B\left(x^{(0)}, \rho\right)$
- ✏ $r = \alpha\beta\gamma < \frac{1}{2}$

en ese caso, el método de Newton converge a un único cero x^* de f y satisface

$$\left\|x^{(s)} - x^*\right\| \leq (2r)^{2^s} \frac{\beta}{r}$$

El método de Newton tiene desventajas por la falta de convergencia global para muchos problemas y por la necesidad de evaluar $J_f\left(x^{(s)}\right)$ y resolver un sistema lineal por cada iteración.

Sin embargo, su convergencia local es cuadrática.

7.7.2.1 Modificaciones al método de Newton

Existen unas modificaciones que se realizan al método original de Newton que intentan reducir algunas de sus desventajas, aunque su convergencia es sólo lineal. Los más importantes son:

- Método modificado con **única evaluación de la Jacobiana**

$$x^{(s+1)} = x^{(s)} - J_f^{-1}\left(x^{(0)}\right) f\left(x^{(s)}\right) \qquad s = 0, 1, 2, \ldots$$

  lo que permite reducir el número de operaciones, ya que una vez evaluada la Jacobiana en $x^{(0)}$, para resolver los sistemas que se van obteniendo es suficiente con factorizar la matriz una sola vez y resolver de forma directa los sistemas en cada iteración. También se llama **método de Newton simplificado**

- Método modificado que usa **fórmulas de derivación numérica** para estimar las derivadas parciales en $x^{(s)}$ y aproximar $J_f\left(x^{(s)}\right)$.

- Método modificado que introduce una **longitud de paso** α_s, de forma que

$$x^{(s+1)} = x^{(s)} - \alpha_s J_f^{-1}\left(x^{(s)}\right) f\left(x^{(s)}\right) \qquad s = 0, 1, 2, \ldots$$

  con lo que se busca la convergencia global con α, estimado según técnicas de optimización aplicadas al problema de mínimos cuadrados

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} f(x)^t f(x)$$

  que veremos en el siguiente tema.

En el siguiente algoritmo se ilustra el código en Matlab que implementa el método de Newton para sistemas de ecuaciones no lineales.

Algoritmo 7.7: Método de Newton para sistemas de ecuaciones no lineales

```
% Metodo Newton de resolucion de
% Sistemas No Lineales
%
% Uso: [sol,valf,error]=snewton(f,j,xs0,met,maxiter,p,tol)
%  f = matriz del sistema
%  j = jacobiana del sistema
%  xs0 = vector solución inicial
%  met = 'divizqda' -> división izquierda para resolver un sistema
%        lineal con la jacobiana. Otra cosa, evalúa el método con
%        las 3 diagonales principales
%  maxiter = número máximo de iteraciones (por defecto 10)
%  p = norma de la matriz para hallar el error (por defecto 2)
%  tol = tolerancia de las soluciones (por defecto 1e-5)
function [xxs,ffs,ers]=snewton(f,jacob,xsmenos1,met,maxiter,p,tol)
if nargin<7
   tol=1e-5;
   if nargin<3
      error('Faltan argumentos de entrada');
   end;
   if nargin<6
```

```
      p=2;
   end;
   if nargin<5
      maxiter=10;
   end;
   if nargin<4
      met='divizqda';
   end;
end;
xxs=xsmenos1';
ffs=[];
ers=NaN;
error=Inf;
iteraciones=0;
while (iteraciones<maxiter)&(error>tol)
   fxsmenos1=feval(f,xsmenos1);
   ffs=[ffs;fxsmenos1'];
   if strcmp(lower(met),'divizqda')
      Jfxsmenos1=feval(jacob,xsmenos1);
      xs=xsmenos1-Jfxsmenos1\fxsmenos1;
   else
      [dinf,dpri,dsup]=feval(jacob,xsmenos1);
      xs=xsmenos1-feval(met,dinf,dpri,dsup,fxsmenos1);
   end;
   xxs=[xxs;xs'];
   error=norm(xs-xsmenos1,p);
   ers=[ers;error];
   if error<tol
      ffs=[ffs;feval(f,xs)'];
   end;
   xsmenos1=xs;
   iteraciones=iteraciones+1;
end;
if iteraciones==maxiter
   disp('Se ha llegado al numero maximo de iteraciones');
end;
```

♦

7.7.3 Métodos cuasi-Newton

7.7.3.1 Introducción

También llamados de la secante, sustituyen la matriz Jacobiana por otra matriz A_s que se actualiza de forma que satisface la **ecuación cuasinewton o secante**, esto es, con A_{s+1} tal que

$$A_{s+1}\left(x^{(s+1)}-x^{(s)}\right)\simeq f\left(x^{(s+1)}\right)-f\left(x^{(s)}\right)$$

lo cual se puede notar también, si hacemos $y^{(s)}=f\left(x^{(s+1)}\right)-f\left(x^{(s)}\right)$:

$$A_{s+1}d^{(s)}\simeq y^{(s)}$$

Esto hace que para determinar A_{s+1}, las soluciones posibles sean muchas. Por supuesto, interesan las soluciones que impliquen una actualización de A_s lo más simple posible. Normalmente son actualizaciones de rango 1 o 2, es decir

$$A_{s+1} = A_s + a^{(s)}b^{(s)^t}$$

o

$$A_{s+1} = A_s + a^{(s)}b^{(s)^t} + u^{(s)}v^{(s)^t}$$

con a, b, u y v vectores columna.

El método de Broyden corresponde a una actualización de rango 1, mientras que BFGS y DFP son actualizaciones de rango 2 de la forma

$$A_{s+1} = A_s + \beta_s a^{(s)}b^{(s)^t} + \delta_s u^{(s)}v^{(s)^t}$$

El método en general puede escribirse de la forma

$$x^{(s+1)} = x^{(s)} - A_s^{-1} f\left(x^{(s)}\right)$$

y teniendo en cuenta que es más cómodo resolver el sistema

$$A_s \left(\underbrace{x^{(s+1)} - x^{(s)}}_{d^{(s)}} \right) = -f\left(x^{(s)}\right)$$

Una elección posible de A_0 es $J_f\left(x^{(0)}\right)$, o incluso I_n.

7.7.3.2 Métodos de Broyden, BFGS y DFP

Estos métodos, por ser métodos secantes, son superlineales, y no es necesario evaluar las derivadas.

El **método de Broyden** es el que utiliza la actualización

$$A_{s+1} = A_s + a^{(s)}b^{(s)^t} = A_s + \frac{\left(y^{(s)} - A_s d^{(s)}\right) d^{(s)^t}}{d^{(s)^t} d^{(s)}} \tag{7.10}$$

eligiendo los vectores $a^{(s)}$ y $b^{(s)}$ tales que se cumpla la ecuación cuasinewton y además con la condición

$$b^{(s)^t}\left(x - x^{(s)}\right) = 0$$

si $\left(x - x^{(s)}\right)^t \left(x^{(s+1)} - x^{(s)}\right) = 0$.

El método consta de las siguientes operaciones que se repiten:

- ✏ Obtener $d^{(s)}$ como solución del sistema lineal $A_s d^{(s)} = -f\left(x^{(s)}\right)$
- ✏ $x^{(s+1)} = x^{(s)} + d^{(s)}$ y $y^{(s)} = f\left(x^{(s+1)}\right) - f\left(x^{(s)}\right)$
- ✏ Hallar A_{s+1} actualizando A_s

Si usamos la inversa de A_s, $H_{s+1} = A_{s+1}^{-1}$, la actualización queda

$$H_{s+1} = H_s + \frac{\left(d^{(s)} - H_s y^{(s)}\right) d^{(s)^t} H_s}{d^{(s)^t} H_s y^{(s)}} \tag{7.11}$$

que se puede obtener aplicando la fórmula de inversión de **Sherman-Morrison-Woodbury**:

$$\left(A + uv^t\right)^{-1} = A^{-1} - \frac{1}{\sigma} A^{-1} u v^t A^{-1}$$

siendo

$$\sigma = 1 + v^t A^{-1} u$$

La convergencia local superlineal puede establecerse para el método de Broyden ya que se puede demostrar que

$$\lim_{s\to\infty} \frac{\left\| \left(A_s - J_f\left(x^{(s)}\right)\right)\left(x^{(s)} - x^*\right)\right\|}{\left\|x^{(s)} - x^*\right\|} = 0$$

lo cual no implica que A_s converja a $J_f\left(x^*\right)$ siendo x^* el punto límite de la sucesión.

Las fórmulas de los **métodos BFGS y DFP** son

- **BFGS**:

$$A_{s+1} = A_s + \frac{y^{(s)} y^{(s)^t}}{y^{(s)^t} d^{(s)}} - \frac{A_s d^{(s)} d^{(s)^t} A_s}{d^{(s)^t} A_s d^{(s)}} \tag{7.12}$$

  o

$$H_{s+1} = H_s + \left(1 + \frac{y^{(s)^t} H_s y^{(s)}}{d^{(s)^t} y^{(s)}}\right) \frac{d^{(s)} d^{(s)^t}}{d^{(s)^t} y^{(s)}} - \frac{d^{(s)} y^{(s)^t} H_s + H_s y^{(s)} d^{(s)^t}}{d^{(s)^t} y^{(s)}} \tag{7.13}$$

- **DFP**:

$$A_{s+1} = A_s + \frac{\left(y^{(s)} - A_s d^{(s)}\right) y^{(s)^t} + y^{(s)} \left(y^{(s)} - A_s d^{(s)}\right)^t}{y^{(s)^t} d^{(s)}} - \frac{\left(y^{(s)} - A_s d^{(s)}\right)^t d^{(s)} y^{(s)} y^{(s)^t}}{\left(y^{(s)^t} d^{(s)}\right)^2} \tag{7.14}$$

  o bien

$$H_{s+1} = H_s + \frac{d^{(s)} d^{(s)^t}}{d^{(s)^t} y^{(s)}} - \frac{H_s y^{(s)} y^{(s)^t} H_s}{y^{(s)^t} H_s y^{(s)}} \tag{7.15}$$

Estos dos métodos de rango 2 son interesantes para problemas cuya Jacobiana es simétrica, que es el caso de los problemas de optimización en los que se busca

$$\nabla f\left(x\right) = 0$$

ya que la Jacobiana de $\nabla f\left(x\right)$, que es la Hessiana de $f\left(x\right)$, es simétrica. Además, si en estos métodos se parte de una matriz definida positiva, generan una sucesión de matrices definidas positivas.

Terminamos el tema con los códigos en Matlab para implementar los métodos cuasi-Newton.

Algoritmo 7.8: Función de actualización de la matriz H mediante Broyden

```
% Metodo de Broyden para actualizar
% la matriz H en los metodos cuasi-Newton
function H=broyden(H,d,y)
    Hy=H*y;
    dp=d';
    H=H+(1/(dp*Hy))*(d-Hy)*(dp*H);
```

♦

Algoritmo 7.9: Función de actualización de la matriz H mediante BFGS

```
% Metodo BFGS para actualizar la matriz
% H en los metodos cuasi-Newton
function H=bfgs(H,d,y)
    Hy=H*y;
    dp=d';
    dpy=dp*y;
    H=H+((1+y'*Hy/dpy)/dpy)*d*dp-(1/dpy)*(d*y'*H+Hy*dp);
```

♦

Algoritmo 7.10: Función de actualización de la matriz H mediante DFP

```
% Metodo DFP para actualizar la matriz H
% en los metodos cuasi-Newton
function H=dfp(H,d,y)
    Hy=H*y;
    dp=d';
    H=H+(1/(dp*y))*d*dp-(1/(y'*Hy))*Hy*(y'*H);
```

♦

Algoritmo 7.11: Implementación de los métodos cuasi-Newton para sistemas de ecuaciones no lineales

```
% Metodos cuasi-Newton de resolucion de
% Sistemas No Lineales
% Se puede escoger el metodo cuasi-Newton.
% Opciones: broyden, dfp y bfgs
%
% Uso: [sol,ffs,error]=scnewton(f,metodo,xs0,Hs0,maxiter,norma,tol)
%  f = matriz que contiene el sistema
%  metodo = broyden, dfp, bfgs
%  xs0 = valor inicial de la solución
%  Hs0 = valor inicial de la jacobiana
%  maxiter = máximo número de iteraciones
%  norma = norma de la matriz para calcular el error
%  tol = tolerancia, por defecto 1e-5
%
% (C) - Carlos Garcia Argos, 8/VI/2000
function [xxs,ffs,ers]=scnewton(f,met,xsmenos1,Hsmenos1,iter,p,tol)
if nargin<7
```

```
   tol=1e-5;
   if nargin<3
      error('Faltan argumentos de entrada');
   end;
   if nargin<6
      p=2;
   end;
   if nargin<5
      iter=10;
   end;
   if nargin<4
       Hsmenos1=eye(length(xsmenos1));
   end;
end;
metok=strcmp(lower(met),'broyden')|strcmp(lower(met),'dfp')|
      strcmp(lower(met),'bfgs');
if metok
   iteraciones=0;
   fxsmenos1=feval(f,xsmenos1);
   xxs=xsmenos1';
   ffs=fxsmenos1';
   ers=NaN;
   while iteraciones < iter
      dsmenos1=-Hsmenos1*fxsmenos1;
      xs=xsmenos1+dsmenos1;
      xxs=[xxs;xs'];
      err=norm(dsmenos1,p);
      ers=[ers;err];
      if err<tol
         ffs=[ffs;feval(f,xs)'];
         return;
      end;
      xsmenos1=xs;
      iteraciones=iteraciones+1;
      ysmenos1=-fxsmenos1;
      fxsmenos1=feval(f,xs);
      ffs=[ffs;fxsmenos1'];
      ysmenos1=ysmenos1+fxsmenos1;
      Hsmenos1=feval(met,Hsmenos1,dsmenos1,ysmenos1);
   end;
   disp('Se ha llegado al numero maximo de iteraciones');
else
   disp('El metodo seleccionado no es correcto');
   return;
end;
```

♦

7.7.4 Test de terminación

En la resolución numérica de sistemas no lineales, cuando se hace por métodos iterativos, hay que dar alguna condición para finalizar las iteraciones.

Normalmente se dan dos condiciones, que suelen ser:

- ✏ Que la solución en la iteración actual no cambie mucho con respecto a la iteración anterior:

$$\left\| x^{(s+1)} - x^{(s)} \right\| < \varepsilon$$

- ✏ Que se haya llegado a un número máximo de iteraciones. En este caso, la solución obtenida no será válida en general, pero nos servirá para que acabe el método y no se quede el programa calculando indefinidamente

7.8 Métodos para ecuaciones algebraicas

7.8.1 Introducción

Las ecuaciones algebraicas son las bien conocidas por todos:

$$p(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_n = 0$$

Para ellas se puede aplicar toda la metodología vista en los apartados anteriores para ecuaciones no lineales, pero vamos a ver métodos específicos que se comportan mejor que los ya vistos.

Para evaluar la función se utiliza el **algoritmo de Horner**, usando la regla de Ruffini tan famosa que nos servía para hallar las raíces de los polinomios.

Si $z \in \mathbb{C}$ es raiz de $p(x) = 0$, se cumple que

- ✏

$$|z| \leq 1 + \frac{a_{max}}{|a_0|} = r$$

donde $a_{max} = \underset{1 \leq i \leq n}{\text{máx}} |a_i|$

Es decir, que en una circunferencia de radio r están contenidas todas las raíces complejas de p.

- ✏

$$|z| \geq \frac{1}{1 + \frac{a^*_{max}}{|a_n|}}$$

donde $a^*_{max} = \underset{0 \leq i \leq n-1}{\text{máx}} |a_i|$

Si ninguna raiz es nula ($a_n \neq 0$), todas las raíces están fuera de este círculo.

7.8.2 Acotación de las raíces reales

Teorema 7.7. (Teorema de Sturm)

Dados $a, b \in \mathbb{R}$ y $a < b$, tales que no son raíces de $p(x) = 0$, entonces el número de raíces reales distintas de la ecuación en el intervalo (a, b) es igual a

$$V(a) - V(b)$$

donde $V(\alpha)$ es el número de cambios de signo en la sucesión

$$p_0(\alpha), p_1(\alpha), \ldots, p_m(\alpha)$$

$$p_0(x) = p(x)$$

$$p_1(x) = p'(x)$$

$$p_2(x) = q_1(x)\, p_1(x) - p_0(x)$$

$$p_3(x) = q_2(x)\, p_2(x) - p_1(x)$$

$$\vdots$$

$$q_k(x) = \frac{p_{k-1}(x)}{p_k(x)}$$

El fin de la sucesión está cuando

$$p_{m-1}(x) = q_m(x)\, p_m(x)$$

Si el grado de $p_m(x)$ es mayor de cero, la ecuación tiene raíces múltiples.

✏ **Regla de Laguerre-Thibault**:

$$p(x) = (x - L)\left(b_0 x^{n-1} + b_1 x^{n-2} + \ldots + b_{n-1}\right) + p(L)$$

Sea $L > 0 / b_i \geq 0\ \forall i$ y $p(L) > 0$. Entonces si $x > L$ y $p(x) > 0$, L es **cota superior de las raíces reales positivas**.
Se pueden obtener cotas inferiores para las raíces negativas haciendo $x \rightsquigarrow -x$ y las cotas inferiores para las positivas con $x \rightsquigarrow 1/x$.
Otra forma de expresar esta regla es: dado $L > 0$, si los coeficientes del polinomio

$$\frac{p(x)}{x - L}$$

son positivos y $p(L) \geq 0$, L es cota superior de las raíces positivas de la ecuación $p(x) = 0$

Ejemplo 7.5:

$$x^6 - x^4 + 3x^3 + 2x - 1 = 0$$

$p(1) = 4$, con lo que $L = 1$ es cota superior de las raíces positivas. Haciendo $x \rightsquigarrow -x$, tenemos

$$x^6 - x^4 - 3x^3 - 2x - 1 = 0$$

y para $x = 3$ el polinomio es positivo, por lo que $L = -3$ es cota inferior de raíces negativas. Ahora, cambiando x por $1/x$ tenemos la cota inferior de las raíces positivas:

$$\frac{1 - x^2 + 3x^3 + 2x^5 - x^6}{x^6} = 0$$

pero con cuidado ya que $b_0 < 0$, así que multiplicamos por -1 la ecuación entera y queda

$$x^6 - 2x^5 - 3x^3 + x^2 - 1 = 0$$

$L = \frac{1}{3}$ es cota inferior de las raíces positivas.
De forma similar puede calcularse una cota superior para las raíces negativas.

■

Teorema 7.8. (De Du Gua)

Si todas las raíces de la ecuación $p(x) = 0$ son reales, entonces

$$a_k^2 > a_{k-1}a_{k+1} \qquad k = 1 : n-1$$

Es una condición necesaria, si no se cumple no se puede asegurar nada.

✏ **Regla de Descartes**:

$$N_+ = N_s - \dot{2}$$

donde N_+ es el número de raíces reales positivas, N_s el número de cambios de signo de la sucesión $a_0, a_1, \ldots, a_n$

Ejemplo 7.6:

$$x^6 - x^4 + 3x^3 + 2x - 1 = 0$$

tiene $N_s = 3$ y $N_+ \in \{1, 3\}$

■

7.8.3 Método de Lin

Es un método iterativo de punto fijo en el que se transforma el polinomio:

$$p(x) = 0 \rightarrow xQ(x) + a_n = 0$$

y la iteración queda

$$x_{s+1} = -\frac{a_n}{Q(x_s)} = \frac{a_n x_s}{a_n - P(x_s)} \tag{7.16}$$

No tiene teoremas específicos de convergencia y sólo es válido para ecuaciones algebraicas.

Algoritmo 7.12: Método de Lin para ecuaciones algebraicas

```
% Metodo de Lin para ecuaciones algebraicas
%
% Uso: [sol,error]=lin(pol,xinicial,maxiter,tol)
%
% La tolerancia por defecto es tol=1e-5
function [xs,errs]=lin(poli,x0,maxiter,tol)
if nargin<4
   tol=1e-5;
end;
if nargin<3
   disp('Error: faltan argumentos de entrada');
   return;
end;
iteraciones=0;
fin=0;
x(1)=x0;
```

```
while (iteraciones<maxiter)&(fin==0)
   iteraciones=iteraciones+1;
   an = poli(length(poli));
   x(iteraciones+1)=an*x(iteraciones)/
                   (an-polyval(poli,x(iteraciones)));
   errs=abs(x(iteraciones+1)-x(iteraciones));
   if errs<tol
     fin=1;
   end;
end;
if fin==0
   disp('Se ha llegado al numero maximo de iteraciones');
else
   disp('Se ha encontrado solucion que cumple la tolerancia');
end;
xs=x(iteraciones+1);
```

♦

7.8.4 Método de Laguerre

Este método que vamos a ver es de orden 3 y sirve incluso con raíces complejas. La iteración es la siguiente:

$$x_{s+1} = x_s - \frac{np\left(x_s\right)}{p'\left(x_s\right) \pm \sqrt{H\left(x_s\right)}} \tag{7.17}$$

siendo

$$H\left(x\right) = \left(n-1\right)\left(\left(n-1\right)\left(p'\left(x\right)\right)^2 - np\left(x\right)p''\left(x\right)\right)$$

y para n el grado del polinomio.

Para evitar la inestabilidad del método, se suele cambiar el denominador, quedando

$$x_{s+1} = x_s - \frac{np\left(x_s\right)}{p'\left(x_s\right) + signo\left(p'\left(x_s\right)\right)\sqrt{H\left(x_s\right)}}$$

De forma que no se produzcan diferencias cancelativas.

Está montado de forma que converja a la raiz más próxima al punto de salida.

El método suele utilizarse de la siguiente forma:

1. Se empieza con $x_s = 0$. De esta forma, la raiz que se calcula es la de menor módulo.
2. Se halla la primera raiz $\overline{x}_1$.
3. Se deflaciona (desinfla) el problema usando

$$Q\left(x\right) = \frac{p\left(x\right)}{x - \overline{x}_1} = 0$$

4. Se vuelve a 1 usando $Q\left(x\right)$.

El método de Newton también puede usarse con raíces complejas.

Algoritmo 7.13: Método de Laguerre para ecuaciones algebraicas

```
% Metodo de Laguerre para ecuaciones algebraicas
%
% Uso: [sol,error]=laguerre(pol,xinicial,maxiter,tol)
%
% La tolerancia por defecto es tol=1e-5
function [xs,errs]=laguerre(poli,x0,maxiter,tol)
if nargin<4
   tol=1e-5;
end;
if nargin<3
   disp('Error: faltan argumentos de entrada');
   return;
end;
iteraciones=0;
fin=0;
x(1)=x0;
n = length(poli) - 1;
while (iteraciones<maxiter)&(fin==0)
   iteraciones=iteraciones+1;
   xant = x(iteraciones);
   num = polyval(poli,xant)*n;
   ppri = polyder(poli);
   pseg = polyder(ppri);
   poliev = polyval(poli,xant);
   ppriev = polyval(ppri,xant);
   psegev = polyval(pseg,xant);
   H = (n-1)*((n-1)*ppriev^2-n*poliev*psegev);
   denom = ppriev+sign(ppriev)*sqrt(H);
   x(iteraciones+1)=xant-num/denom;
   errs=abs(x(iteraciones+1)-x(iteraciones));
   if errs<tol
     fin=1;
   end;
end;
if fin==0
   disp('Se ha llegado al numero maximo de iteraciones');
else
   disp('Se ha encontrado solucion que cumple la tolerancia');
end;
xs=x(iteraciones+1);
```

♦

7.8.5 Método de Bairstow

En este método se trabaja con aritmética real y se calculan las raíces a pares, determinando un factor cuadrático del polinomio.

$$p(x) = \underbrace{\left(x^2 + px + q\right)}_{\text{factor cuadrático}} \underbrace{\left(b_0 x^{n-2} + \ldots + b_{n-2}\right)}_{\text{con esto se sigue aplicando}}$$

Aunque se trabaja con aritmética real todo el tiempo, al final se pueden obtener las raíces complejas solucionando el factor cuadrático.

Normalmente tendremos un resto al dividir $p(x)$ por el factor cuadrático:

$$p(x) = \left(x^2 + px + q\right)\left(b_0 x^{n-2} + \ldots + b_{n-2}\right) + b_{n-1}(x + p) + b_n$$

De forma que interesa hacer

$$b_{n-1}(p, q) = 0$$

$$b_n(p, q) = 0$$

Lo cual lleva a un sistema de ecuaciones no lineal a resolver por el método de Newton.

Procedimiento para resolverlo:

Se hace un proceso iterativo en el que

$$p_{s+1} = p_s + \Delta p_s$$

$$q_{s+1} = q_s + \Delta q_s$$

Para calcular los incrementos, vamos a usar una operación conocida como **división sintética doble** que consiste en un Ruffini especial llamado "Ruffini doble". Disponiendo en la forma de Ruffini:

	a_0	a_1	a_2	$\ldots$	a_{n-2}	a_{n-1}	a_n
$-p)$	$\downarrow$	$-pb_0$	$-pb_1$	$\ldots$	$-pb_{n-3}$	$-pb_{n-2}$	$-pb_{n-1}$
$-q)$	$\downarrow$		$-qb_0$	$\ldots$	$-qb_{n-4}$	$-qb_{n-3}$	$-qb_{n-2}$
	b_0	b_1	b_2	$\ldots$	b_{n-2}	b_{n-1}	b_n

Se repite una vez más el proceso

	b_0	b_1	b_2	$\ldots$	b_{n-2}	b_{n-1}	b_n
$-p)$	$\downarrow$	$-pc_0$	$-pc_1$	$\ldots$	$-pc_{n-3}$	$-pc_{n-2}$	$-pc_{n-1}$
$-q)$	$\downarrow$		$-qc_0$	$\ldots$	$-qc_{n-4}$	$-qc_{n-3}$	$-qc_{n-2}$
	c_0	c_1	c_2	$\ldots$	c_{n-2}	c_{n-1}	c_n

Se pueden obtener los incrementos a partir de

$$\Delta p = \frac{\begin{vmatrix} b_{n-1} & b_n \\ c_{n-3} & c_{n-2} \end{vmatrix}}{\delta} \tag{7.18}$$

$$\Delta q = \frac{\begin{vmatrix} c_{n-2} & c_{n-1} \\ b_{n-1} & b_n \end{vmatrix}}{\delta} \tag{7.19}$$

siendo

$$\delta = \begin{vmatrix} c_{n-2} & c_{n-1} \\ c_{n-3} & c_{n-2} \end{vmatrix} \tag{7.20}$$

Se parte de un par p_0, q_0, se calculan los $\Delta p_0, \Delta q_0$ obteniendo así p_1, q_1 y se repite para este último par hasta que $b_n = b_{n-1} = 0$.

Cuando se obtengan los p_s, q_s, la ecuación

$$x^2 + p_s x + q_s = 0$$

es fácil de resolver y sólo hay que repetir el proceso para el resto de la ecuación.

La elección del par inicial p_0, q_0 es vital ya que el método de Newton es localmente convergente. Como vemos en el siguiente ejemplo, una buena elección es posible si se conocen x_1 y x_2 puntos cercanos a dos raíces, haciendo que el factor cuadrático sea

$$x^2 - (x_2 + x_1)\, x + x_1 x_2$$

$$p_0 = -(x_2 + x_1) \qquad q_0 = x_1 x_2$$

Ejemplo 7.7:

$$x^4 + 2x^3 - x^2 - 1 = 0$$

Si vemos su gráfica, encontramos que tiene un cero dentro del intervalo $[-2.5, -2.4]$ y otro en $[0.8, 0.9]$, por lo que, por las propiedades de las raíces, podemos encontrar el par inicial:

$$p_0 = -(-2.5 + 0.8) = 1.7$$

$$q_0 = -2.5 \cdot 0.8 = -2$$

Con los que podemos proceder a usar el método.

■

7.8.6 Método de Bernouilli

Tiene el mismo fundamento que el método de la potencia para el cálculo de autovalores. Se calcula la raíz de mayor módulo, supuesta única. Si fuera múltiple, habría que adaptar el método.

Dada la ecuación

$$a_0 x^n + a_1 x^{n-1} + \ldots + a_n = 0$$

hacemos la siguiente conversión

$$a_0 y_{n+k} + a_1 y_{n+k-1} + \ldots + a_n y_k = 0 \qquad k = 1, 2, \ldots$$

Tenemos una ecuación de recurrencia.

Ahora cabe preguntarse a qué tiende $\frac{y_{i+1}}{y_i}$. En primer lugar, suponemos que hay una raiz dominante:

$$y_i = \sum c_j x_j^i$$

siendo x_j una raiz cualquiera del polinomio.

Si x_1 es la raiz dominante,

$$y_i = x_1^i \sum c_j \left(\frac{x_j}{x_1}\right)^i = x_1^i \left(c_1 + 0\right)$$

$$y_{i+1} = x_1^{i+1} \sum c_j \left(\frac{x_j}{x_1}\right)^{i+1} = x_1^{i+1} \left(c_1 + 0\right)$$

con lo que al hacer tender i a ∞,

$$\frac{y_{i+1}}{y_i} \to x_1$$

Hacen falta n valores para empezar la recurrencia. Una buena elección de los valores se hace con ayuda de las relaciones de Newton:

$$a_0 y_1 + a_1 = 0$$

$$a_0 y_k + a_1 y_{k-1} + \ldots + a_{k-1} y_1 + k a_k = 0$$

despejando se obtienen las condiciones iniciales para evitar que se haga $c_1 = 0$

$$y_1 = -\tfrac{a_1}{a_0}$$

$$y_k = -\tfrac{1}{a_0}\left(k a_k + a_1 y_{k-1} + \ldots + a_{k-1} y_1\right) \qquad k = 2 : n$$

7.8.7 Deflación

En la práctica es conveniente aplicar técnicas de deflación, ya que al buscar nuevas raíces los métodos vistos, y en general todos los métodos iterativos, podrían converger a raíces que ya han sido calculadas.

De esta forma, se eliminan las raíces que ya se han obtenido, pero se corre el riesgo de que haya inestabilidad. En cualquier caso, se pueden aplicar técnicas de **purificación** de resultados, o incluso **deflación implícita**, como el siguiente **método de Newton-Maehly**

$$\boxed{x_{s+1} = x_s - \frac{p\left(x_s\right)}{p'\left(x_s\right) - \sum_{j=1}^{m} \frac{p\left(x_s\right)}{x_s - x_j^*}}} \tag{7.21}$$

siendo x_j, $j = 1 : m$ las raíces que ya están calculadas. Este método es el mismo método de Newton quitando de forma implícita las raíces ya calculadas.

TEMA 8

Métodos numéricos de optimización

Este tema trata de maximizar o minimizar (optimizar) una función, aunque vamos a considerar únicamente el segundo caso, ya que si sabemos cómo minimizar una función, es trivial que para maximizar:

$$\max_{x\in\mathbb{R}^n} f(x) = \min_{x\in\mathbb{R}^n} (-f(x))$$

También usaremos desarrollos de Taylor constantemente.

8.1 Optimización no restringida

Tenemos que minimizar $f(x) : \mathbb{R}^n \to \mathbb{R}^n$ con $f \in C^2(\mathbb{R}^n)$. Primero veamos algunas definiciones:

- Se llama **mínimo** al punto en que la función se hace mínima (algunos autores pueden hablar de mínimo como el valor mínimo de la función).
- Se dice que un punto $x^* \in \mathbb{R}^n$ es **mínimo local** si cumple

  $$\exists \varepsilon > 0 / f(x^*) \leq f(x) \qquad \forall x \in B(x^*, \varepsilon) = \{x / \|x - x^*\| \leq \varepsilon\}$$

  estricto si además

  $$\exists \varepsilon > 0 / f(x^*) < f(x) \qquad \forall x \in B(x^*, \varepsilon) - \{x^*\}$$

- Por otro lado, $x^* \in \mathbb{R}^n$ es **mínimo global** si

  $$\exists \varepsilon > 0 / f(x^*) \leq f(x) \qquad \forall x \in \mathbb{R}^n$$

  estricto si

  $$\exists \varepsilon > 0 / f(x^*) < f(x) \qquad \forall x \in \mathbb{R}^n - \{x^*\}$$

  El cálculo de mínimos globales suele interesar sólo en aplicaciones concretas, así que en general nos preocuparemos del cálculo de mínimos locales
- **Puntos críticos o estacionarios** son aquellos en los que $\nabla f(x) = 0$ (satisfacen la condición de primer orden).
- **Puntos de silla** son los puntos críticos que no son ni mínimos ni máximos locales.

Existen teoremas sobre existencia de mínimos, como por ejemplo que si la función es convexa en un conjunto convexo, tiene un mínimo global. Sin embargo, no vamos a considerar estos teoremas.

Si vemos un desarrollo de Taylor para $f\left(x+\alpha d\right)$, es decir, un desplazamiento del punto de la función,

$$f\left(x+\alpha d\right)=f\left(x\right)+\alpha d^{t}\nabla f\left(x\right)+\frac{1}{2}\alpha^{2}d^{t}\nabla^{2}f\left(x+\theta\alpha d\right)d$$

donde d es el **vector de dirección** perteneciente a $\mathbb{R}^n$ y dispuesto como columna, $\alpha\in\mathbb{R}$ y $0\leq\theta\leq 1$.

Mirando en el desarrollo de Taylor, vemos que para que el punto x sea un mínimo, un desplazamiento en cualquier dirección no debe resultar en un valor menor de f que aquel que se tiene para x, es decir, que el término de la derivada de primer orden no debe restar, y tampoco el de la derivada de segundo orden.

Para que el término de la primera derivada ($\nabla f\left(x\right)$) no reste, dado que α puede ser positivo o negativo (y ello no debe afectar a la condición de máximo o mínimo), debe ser $\nabla f\left(x\right)=0$. De esta forma no cuenta en el resultado.

Por otro lado, en la segunda derivada ($\nabla^2 f\left(x\right)$) α^2 siempre será positivo, y si $\nabla^2 f\left(x+\theta\alpha d\right)$ es distinto de cero, no debe ser negativo para no restar.

De aquí se extraen las

- **Condiciones necesarias de mínimo local**:

$$\nabla f\left(x^{*}\right)=0$$

$$\nabla^{2}f\left(x^{*}\right)\text{ semidefinida positiva}$$

- **Condiciones suficientes de mínimo local estricto**:

$$\nabla f\left(x^{*}\right)=0$$

$$\nabla^{2}f\left(x^{*}\right)\text{ definida positiva}$$

- En los puntos de silla, $\nabla^2 f\left(x\right)$ es indefinida.

8.2 Métodos iterativos de descenso

En los métodos más usuales no se suele hacer $\nabla f\left(x^{*}\right)=0$ y ver qué clase de punto crítico es, sino que se construye una sucesión $x^{(s)}$ de forma que la función en el punto siguiente a considerar valga menos que en el anterior, es decir

$$f\left(x^{(s+1)}\right)<f\left(x^{(s)}\right)\qquad\forall s \tag{8.1}$$

Por eso se llama a estos métodos "**métodos de descenso**".

En ellos, se fija una dirección de búsqueda de forma que se cumpla la condición (8.1), buscando un factor adecuado $\alpha_s>0$ (**longitud de paso)** para que se cumpla

$$f\left(x^{(s)}+\alpha_s d^{(s)}\right)<f\left(x^{(s)}\right)$$

con lo que la iteración es

$$x^{(s+1)} = x^{(s)} + \alpha_s d^{(s)}$$

y lo que hay que hacer es determinar la longitud de paso α_s. La forma de fijar la **dirección de búsqueda o descenso** d es la que distingue los métodos.

Condición necesaria para que la dirección sea de descenso:

$$\varphi(\alpha) = f\left(x^{(s)} + \alpha d^{(s)}\right)$$

donde d es función de α. Derivando

$$\frac{d\varphi(\alpha)}{d\alpha} = {d^{(s)}}^t \nabla f\left(x^{(s)} + \alpha d^{(s)}\right) < 0$$

la condición es cuando $\alpha = 0$ ($\varphi'(0)$)

$$\boxed{{d^{(s)}}^t \nabla f\left(x^{(s)}\right) < 0} \tag{8.2}$$

Si no lo cumple, existen alternativas para resolverlo, pero es conveniente que se cumpla.

$$\min_{\alpha} \varphi(\alpha) \rightarrow \alpha_s / \frac{d\varphi}{d\alpha} = 0$$

Los métodos de descenso se diferencian en la forma de elegir la dirección de búsqueda $d^{(s)}$ y la determinación de la longitud de paso α_s es el **subproblema de búsqueda en línea**. Se puede usar un mismo método con distintas formas de resolver este subproblema.

8.2.1 Búsqueda en línea exacta

$$\alpha_s \,/\, \frac{d\varphi_s(\alpha_s)}{d\alpha} = {d^{(s)}}^t \nabla f\left(x^{(s)} + \alpha_s d^{(s)}\right) = 0$$

Para

$$\varphi_s(\alpha) = f\left(x^{(s)} + \alpha d^{(s)}\right)$$

$$\alpha_s = -\frac{{d^{(s)}}^t \nabla f\left(x^{(s)}\right)}{{d^{(s)}}^t \nabla^2 f\left(x^{(s)}\right) d^{(s)}}$$

El problema que hay con este método es que la ecuación resultante puede ser muy complicada, por eso se buscan otros métodos.

8.2.2 Búsqueda en línea aproximada

Este método busca imponer que la función en el punto siguiente valga "algo suficientemente bajo".

Un teorema de Wolfe establece que para una función $f \in C^1$ acotada inferiormente, si $d^{(s)}$ es dirección de descenso, se pueden determinar valores de la longitud de paso α_s dentro de un intervalo $[\alpha_-, \alpha_+]$ para $\alpha_- < \alpha_+$ tales que satisfacen la siguiente desigualdad (**desigualdad de Armijo-Goldstein**):

$$f\left(x^{(s)}\right)+\mu_2\alpha_s {d^{(s)}}^t\nabla f\left(x^{(s)}\right)\leq f\left(x^{(s+1)}\right)\leq f\left(x^{(s)}\right)+\mu_1\alpha_s {d^{(s)}}^t\nabla f\left(x^{(s)}\right) \tag{8.3}$$

y la **desigualdad de Wolfe**

$${d^{(s)}}^t\nabla f\left(x^{(s)}+\alpha_s d^{(s)}\right)\geq \eta {d^{(s)}}^t\nabla f\left(x^{(s)}\right) \tag{8.4}$$

con $0<\mu_1\leq\eta<1$. Los valores suelen rondar $\mu_1=0.1$ y $\eta=0.9$. Valores de η menores exigen más esfuerzo para obtener un α_s válido.

Los algoritmos de cálculo de longitud de paso suelen hacer una búsqueda de la forma $\left\{w^j\alpha_0\right\}\ j=0,1,2,\ldots$ siendo α_0 un paso inicial que normalmente vale 1 y w un escalar que actúa de factor de reducción.

- ✏ **Condiciones de Armijo-Goldstein**: la longitud de paso α_s debe cumplir

$$0<-\mu_1\alpha_s {d^{(s)}}^t\nabla f\left(x^{(s)}\right)\leq f\left(x^{(s)}\right)-f\left(x^{(s+1)}\right)\leq-\mu_2\alpha_s {d^{(s)}}^t\nabla f\left(x^{(s)}\right) \tag{8.5}$$

  siendo

$$0<\mu_1\leq\mu_2<1$$

- ✏ **Condición de Wolfe**: la longitud de paso debe ser tal que la derivada direccional de f en $x^{(s)}+\alpha_s d^{(s)}$ se aproxime a cero:

$$\left|{d^{(s)}}^t\nabla f\left(x^{(s)}+\alpha_s d^{(s)}\right)\right|\leq\eta {d^{(s)}}^t\nabla f\left(x^{(s)}\right) \tag{8.6}$$

  con

$$0\leq\eta<1$$

El método de búsqueda en línea aproximada lo que hace es combinar una de las condiciones de Armijo con una de las de Wolfe.

Lo que se obtiene es un intervalo $[\alpha_-,\alpha_+]$ en el cual se encuentran los valores de α_s adecuados para la dirección de descenso. Con esto se asegura la convergencia saliendo desde cualquier punto.

Para hacer la búsqueda del α_s se puede proceder comenzando desde un α_0, comprobando si queda dentro de los límites, modificándolo por algún factor si no es así.

8.2.3 Convergencia global

La convergencia de la sucesión $\left\{x^{(s)}\right\}$ a un mínimo de la función a minimizar f no está asegurada si $f\left(x^{(s+1)}\right)<f\left(x^{(s)}\right)$ con cualquier vector de inicio $x^{(0)}$. La convergencia de un método de descenso para cualquier $x^{(0)}$ se llama **convergencia global**.

Teorema 8.1. (Teorema de Wolfe): dada f de clase 2 en $\mathbb{R}^n$, un método con dirección de descenso (nula si $\nabla f\left(x^{(s)}\right)=0$) de búsqueda en línea aproximada con las condiciones (8.3) y (8.4) genera una sucesión $\left\{x^{(s)}\right\}$ que cumple una de las siguientes:

1. $\nabla f\left(x^{(s)}\right)=0$ para algún $s\geq1$ (encontrar un punto crítico)
2. $\lim\limits_{s\to\infty} f\left(x^{(s)}\right)=-\infty$ (sin límite inferior)
3. $\lim\limits_{s\to\infty}\dfrac{{d^{(s)}}^t\nabla f\left(x^{(s)}\right)}{\left\|d^{(s)}\right\|_2^2}=0$ (dirección de descenso perpendicular al gradiente, que es la dirección de máximo descenso)

Cuando se quiere implementar un método de búsqueda en línea aproximada se asegura antes que 2 y 3 no se cumplan.

8.2.4 Método del gradiente

La dirección de descenso es la de máxima variación de f:

$$d^{(s)} = -\nabla f\left(x^{(s)}\right) \tag{8.7}$$

El punto siguiente es

$$x^{(s+1)} = x^{(s)} - \alpha_s \nabla f\left(x^{(s)}\right)$$

Vemos que es dirección de descenso:

$$d^{(s)^t}\left(-\nabla f\left(x^{(s)}\right)\right) = -\left\|\nabla f\left(x^{(s)}\right)\right\|_2^2 < 0$$

Este método converge lentamente, aunque es sencillo de implementar. Un método más rápido es el método del gradiente conjugado, que veremos después. Su convergencia es lenta porque "zigzaguea", al ser cada dirección ortogonal a la anterior.

Ejemplo 8.1:

Resolver

$$\text{mín}\, q\left(x\right) = \frac{1}{2}x^t G x + x^t c$$

siendo

$$G = \begin{pmatrix} 10 & 2 & -2 \\ 2 & 4 & 2 \\ -2 & 2 & 4 \end{pmatrix} \quad c = \begin{pmatrix} 0 \\ 0 \\ -6 \end{pmatrix}$$

$$x^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\nabla q = Gx + c$$

$$\nabla^2 q = G$$

$$d^{(0)} = -\nabla q\left(x^{(0)}\right) = \begin{pmatrix} 0 \\ 0 \\ -6 \end{pmatrix}$$

hallamos los vectores sucesivos con búsqueda en línea exacta:

$$x^{(1)} = x^{(0)} - \frac{\left\|d^{(s)}\right\|_2^2}{d^{(s)^t} G d^{(s)}} \nabla f\left(x^{(0)}\right) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} - \frac{1}{4}\begin{pmatrix} 0 \\ 0 \\ -6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 3/2 \end{pmatrix}$$

$$d^{(1)} = -\nabla q\left(x^{(1)}\right) = \begin{pmatrix} 3 \\ -3 \\ 3 \end{pmatrix}$$

$$x^{(2)} = \begin{pmatrix} 1 \\ -1 \\ -5/2 \end{pmatrix}$$

omitiendo el cálculo de $d^{(2)}$

$$x^{(3)} = \begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix}$$

$$\nabla q\left(x^{(3)}\right) = 0$$

se obtiene que el vector $x^{(3)}$ es un punto crítico y la matriz G es definida positiva, con lo que $x^{(3)}$ es un mínimo global.

■

Algoritmo 8.1: Método del gradiente para optimización

```
% Metodo del gradiente para optimizacion de funciones
% Optimiza una funcion del tipo
%  q(x)=1/2*x'*A*x+x'*c
%
% Uso: [sol,iter]=optgrad(A,c,x0,maxiter,tol)
function [xs,iter]=optgrad(A,c,x0,maxiter,tol)
if nargin<4
   error('Faltan argumentos de entrada');
   return;
end;
if nargin<5
   tol=1e-5;
end;
fin=0;
iter=1;
ds=-A*x0-c;
alpha=(ds'*ds)/(ds'*A*ds);
x(:,1)=x0;
while (fin==0)&(iter<maxiter)
   x(:,iter+1)=x(:,iter)+alpha*ds;
   iter=iter+1;
   ds=-A*x(:,iter)-c;
   if norm(ds,Inf)<tol
      fin=1;
   else
      alpha=norm(ds,2)^2/(ds'*A*ds);
   end;
end;
xs=x(:,iter);
if iter==maxiter
   disp('Se ha alcanzado el maximo de iteraciones');
else
   disp('Se ha encontrado un punto que cumple la tolerancia');
end;
```

♦

8.2.5 Método del gradiente conjugado

Como vimos al final del tema 2, podemos optimizar una función objetivo de la forma

$$\phi(x) = \frac{1}{2}x^t Ax - x^t b$$

Para el caso que nos ocupa:

$$A = \nabla^2 f(x)$$

$$-r^{(s)} = \nabla f(x)$$

Recordamos que el gradiente sea conjugado significa:

$$\left\langle d^{(s+1)}, d^{(s)} \right\rangle = {d^{(s+1)}}^t A d^{(s)} = 0$$

Sin embargo, como ahora usamos una matriz que no es fija (la Hessiana se evalúa en cada punto de la iteración), no conserva las propiedades que vimos en el tema 2, por ejemplo, no tiene porqué terminar en n pasos como lo hacía el método del gradiente conjugado para funciones cuadráticas definidas positivas.

El algoritmo es similar al visto anteriormente:

$$\begin{array}{l} s = 0 \qquad\qquad\qquad\qquad\qquad\qquad g^{(0)} = \nabla f\left(x^{(0)}\right) \\ g^{(s)} \neq 0 \left\{ \begin{array}{ll} d^{(0)} = -g^{(0)} & \text{si } s > 0 \rightarrow d^{(s)} = -g^{(s)} + \beta_s d^{(s-1)} \\ & \beta_s = \frac{{g^{(s)}}^t \nabla^2 f\left(x^{(s)}\right) d^{(s-1)}}{{d^{(s-1)}}^t \nabla^2 f\left(x^{(s)}\right) d^{(s-1)}} \\ x^{(s+1)} = x^{(s)} + \alpha_s d^{(s)} & \alpha_s = -\frac{{g^{(s)}}^t d^{(s)}}{{d^{(s)}}^t \nabla^2 f\left(x^{(s)}\right) d^{(s)}} \\ g^{(s+1)} = \nabla f\left(x^{(s+1)}\right) & \\ s = s + 1 & \end{array} \right. \end{array} \tag{8.8}$$

Como siempre que tenemos que evaluar matrices grandes, tenemos la pega de evaluar una Hessiana en todos los puntos de la iteración, con la problemática que esto conlleva si el sistema es muy grande y necesitamos muchas iteraciones. Por eso, hay algunas fórmulas alternativas al método ya visto.

Además, se sustituye el cálculo de α_s por una búsqueda en línea aproximada, para evitar evaluar la Hessiana al calcularlo.

Veamos algunas de las modificaciones más usuales:

8.2.5.1 Modificación de Hestenes-Stiefel

Esta modificación funciona mejor si se usa búsqueda en línea aproximada para calcular α_s.

$$\beta_s = \frac{{g^{(s)}}^t \left(g^{(s)} - g^{(s-1)}\right)}{\left(g^{(s)} - g^{(s-1)}\right)^t d^{(s-1)}} \tag{8.9}$$

8.2.5.2 Modificación de Fletcher-Reeves

$$\beta_s = \frac{\left\|g^{(s)}\right\|_2^2}{\left\|g^{(s-1)}\right\|_2^2} \tag{8.10}$$

8.2.5.3 Modificación de Polak-Ribiére

Empíricamente, este método es el mejor si se usa búsqueda en línea exacta para el cálculo de α_s.

$$\beta_s = \frac{g^{(s)^t}\left(g^{(s)} - g^{(s-1)}\right)}{\left\|g^{(s-1)}\right\|_2^2} \tag{8.11}$$

Vamos a ver ahora el código fuente en Matlab para implementar el método del gradiente conjugado para resolver problemas de optimización que permite las modificaciones vistas.

Algoritmo 8.2: Método del gradiente conjugado para resolver problemas de optimización

```
% Metodo del gradiente conjugado para optimizacion de funciones
% Optimiza una funcion del tipo:
%   q(x)=1/2*x'*A*x+x'*b
%
% Uso: [sol,iter]=gconjopt(A,b,xinicial,maxiter,modificacion,tol)
% xinicial debe ser un vector columna
%
% Admite las modificaciones de 'hestenes-stiefel',
% 'fletcher-reeves' o 'polak-ribiere'. Por defecto no
% hace modificacion.
function [xs,iter]=gconjopt(A,b,x0,maxiter,modif,tol)
if nargin<4
   error('Faltan argumentos de entrada');
   return;
end;
if nargin<6
   tol=1e-5;
end;
if nargin<5
   modif='ninguna';
end;
r(:,1)=-b-A*x0;
G=-r(:,1);
iter=1;
x(:,1)=x0;
d(:,1)=r(:,1);
beta=0;
H=A;
switch lower(modif)
case {'ninguna'}
   disp('Sin usar modificación')
case {'hestenes-stiefel'}
   disp('Modificación de Hestenes-Stiefel')
case {'fletcher-reeves'}
   disp('Modificación de Fletcher-Reeves')
case {'polak-ribiere'}
   disp('Modificación de Polak-Ribiére')
otherwise
```

```
   error('Método desconocido');
end;
while (norm(r(:,iter))>tol)&(iter<maxiter)&(norm(G,2)~=0)
   alpha=-(G'*d(:,iter))/(d(:,iter)'*H*d(:,iter));
   x(:,iter+1)=x(:,iter)+alpha*d(:,iter);
   iter=iter+1;
   G=A*x(:,iter)+b;
   r(:,iter)=-G;
   switch lower(modif)
   case {'ninguna'}
      beta=(G'*A*d(:,iter-1))/(d(:,iter-1)'*A*d(:,iter-1));
   case {'hestenes-stiefel'}
      beta=(-r(:,iter)'*(r(:,iter)-r(:,iter-1)))/
           ((r(:,iter)-r(:,iter-1))'*r(:,iter-1));
   case {'fletcher-reeves'}
      beta=norm(r(:,iter),2)^2/norm(r(:,iter-1),2)^2;
   case {'polak-ribiere'}
      beta=(r(:,iter)'*(r(:,iter)-r(:,iter-1)))/
            norm(r(:,iter-1),2)^2;
   end;
   d(:,iter)=r(:,iter)+beta*d(:,iter-1);
end;
xs=x(:,iter);
if iter==maxiter
   disp('Se ha alcanzado el número máximo de iteraciones');
else
   disp('Se ha encontrado una solución que cumple la tolerancia');
end;
```

♦

8.2.6 Método de Newton

La optimización es

$$\min_{d\in\mathbb{R}^n}\left(f\left(x^{(s)}\right)+d^t\nabla f\left(x^{(s)}\right)+\frac{1}{2}d^t\nabla^2 f\left(x^{(s)}\right)d\right)$$

Veamos la condición de que el gradiente sea cero:

$$\nabla f\left(x^{(s)}\right)+\nabla^2 f\left(x^{(s)}\right)d^{(s)}=0$$

$$\boxed{d^{(s)}=-\left(\nabla^2 f\left(x^{(s)}\right)\right)^{-1}\nabla f\left(x^{(s)}\right)} \tag{8.12}$$

¿Es dirección de descenso? Pues puede que si o puede que no... Veámoslo:

Si $x^{(s)}\simeq x^*$ es un mínimo local y $\nabla^2 f\left(x^{(s)}\right)$ es definida positiva

$$d^{(s)^t}\nabla f\left(x^{(s)}\right)=-d^{(s)^t}\nabla^2 f\left(x^{(s)}\right)d^{(s)}=-\nabla f\left(x^{(s)}\right)^t\left(\nabla^2 f\left(x^{(s)}\right)\right)^{-1}\nabla f\left(x^{(s)}\right)<0$$

entonces es dirección de descenso, ya que se cumple la condición (8.2).

Entonces, para encontrar la dirección de búsqueda tenemos que resolver el sistema

$$\nabla^2 f\left(x^{(s)}\right) d^{(s)} = -\nabla f\left(x^{(s)}\right) \tag{8.13}$$

y entonces el punto siguiente es

$$x^{(s+1)} = x^{(s)} - \alpha_s \left(\nabla^2 f\left(x^{(s)}\right)\right)^{-1} \nabla f\left(x^{(s)}\right) \tag{8.14}$$

donde normalmente $\alpha_s = 1$, aunque también se puede usar búsqueda en línea aproximada.

El método de Newton tiene convergencia cuadrática siempre y cuando el vector de inicio $x^{(0)}$ esté suficientemente cerca de la solución (ver los teoremas del tema anterior sobre la convergencia del método de Newton para ecuaciones no lineales).

Este método tiene algún que otro inconveniente, como es el tener que resolver sistemas que pueden ser arbitrariamente grandes y evaluar en cada punto la Hessiana. Además, la convergencia no es global. También es un inconveniente evidente que $\left(\nabla^2 f\left(x^{(s)}\right)\right)^{-1}$ no sea definida positiva.

Por eso, se suelen hacer algunas **modificaciones**:

- ✏ Usar durante varias iteraciones la misma Hessiana ya evaluada o incluso sólo usar la primera evaluación para todas las iteraciones.
- ✏ Estimar los elementos de $\nabla^2 f\left(x^{(s)}\right)$ por derivación numérica.
- ✏ Hacer búsqueda en línea aproximada.
- ✏ Si $\nabla^2 f\left(x^{(s)}\right)$ no es definida positiva se puede hacer alguna de las siguientes modificaciones:
  - ☞ Garantizando que la Hessiana es de diagonal dominante sumando una diagonal a la que se tiene,

    $$\nabla^2 f\left(x^{(s)}\right) \longrightarrow \nabla^2 f\left(x^{(s)}\right) + \mu I \tag{8.15}$$

    donde μ garantiza que la nueva Hessiana es de diagonal dominante, y por tanto definida positiva.
  - ☞ O la dirección de curvatura negativa:

    $$d^{(s)^t} \nabla^2 f\left(x^{(s)}\right) d^{(s)} < 0 \tag{8.16}$$

  - ☞ También se puede combinar el método de Newton con el del gradiente, usando éste cuando la Hessiana no sea definida positiva

Algoritmo 8.3: Método de Newton para optimización de funciones

```
% Metodo de Newton para optimizacion de funciones
%
% Uso: [sol,iter]=newtonop(G,J,x0,maxiter,tol,metodo)
%
% - G es la función del gradiente
% - J es la función de la Jacobiana
% - metodo es la forma de calcular el alfa (ninguno,
%   o lineaex)
function [xs,iter]=newtonop(G,J,x0,maxiter,tol,metodo)
if nargin<5
   metodo = 'ninguno';
```

```
end;
if nargin<4
   error('Faltan argumentos de entrada');
   return;
end;
if nargin<5
   tol=1e-5;
end;
fin=0;
iter=1;
jacob = feval(J,x0);
grad = feval(G,x0);
ds=-jacob\grad;
switch lower(metodo)
case {'ninguno'}
   disp('   alfa = 1');
   alpha = 1;
case {'lineaex'}
   disp('   Calcular alfa por linea exacta');
   alpha=(ds'*grad)/(ds'*jacob*ds);
otherwise
   error('Metodo para calcular alfa desconocido');
end;
x(:,1)=x0;
while (fin==0)&(iter<maxiter)
   x(:,iter+1)=x(:,iter)+alpha*ds;
   iter=iter+1;
   jacob = feval(J,x(:,iter));
   grad = feval(G,x(:,iter));
   ds=-jacob\grad;
   if norm(ds,Inf)<tol
      fin=1;
   else
      switch lower(metodo)
      case {'ninguno'}
         alpha = 1;
      case {'lineaex'}
         alpha=(ds'*grad)/(ds'*jacob*ds);
      end;
   end;
end;
xs=x(:,iter);
if iter==maxiter
   disp('Se ha alcanzado el maximo de iteraciones');
else
   disp('Se ha encontrado un punto que cumple la tolerancia');
end;
```

♦

8.2.7 Métodos cuasi-Newton

También se llaman métodos secante, y se trata de satisfacer la **condición secante**:

$$A_{s+1}\left(x^{(s+1)}-x^{(s)}\right)=\nabla f\left(x^{(s+1)}\right)-\nabla f\left(x^{(s)}\right) \tag{8.17}$$

de forma que A_{s+1} sea una aproximación a $\nabla^2 f\left(x^{(s)}\right)$. Además cada A_s debe ser simétrica definida positiva y tal que se pueda actualizar de forma sencilla (computacionalmente) y poco costosa. Como vimos en el tema anterior, también se puede plantear la actualización de H_s como aproximación a $\left(\nabla^2 f\left(x^{(s)}\right)\right)^{-1}$. Para las fórmulas que vamos a ver se toma la siguiente notación

$$y^{(s)}=\nabla f\left(x^{(s+1)}\right)-\nabla f\left(x^{(s)}\right)$$

$$p^{(s)}=-\alpha_s A_s^{-1}\nabla f\left(x^{(s)}\right)$$

8.2.7.1 Fórmula simétrica de rango uno

Aquí se actualiza la matriz $H_s=A_s^{-1}$. Tiene el inconveniente de que no mantiene la matriz definida positiva, por lo que puede generar una dirección que no sea de descenso.

$$H_{s+1}=H_s+\frac{\left(p^{(s)}-H_s y^{(s)}\right)\left(p^{(s)}-H_s y^{(s)}\right)^t}{{y^{(s)}}^t\left(p^{(s)}-H_s y^{(s)}\right)} \tag{8.18}$$

8.2.7.2 Fórmula Powell-Symmetric-Broyden (PSB)

$$\begin{aligned}A_{s+1} &= A_s+\frac{\left(y^{(s)}-A_s p^{(s)}\right){p^{(s)}}^t+p^{(s)}\left(y^{(s)}-A_s p^{(s)}\right)^t}{{p^{(s)}}^t p^{(s)}}- \\ &-\frac{\left(y^{(s)}-A_s p^{(s)}\right)^t p^{(s)}p^{(s)}{p^{(s)}}^t}{\left({p^{(s)}}^t p^{(s)}\right)^2}\end{aligned} \tag{8.19}$$

8.2.7.3 Fórmula BFGS

$$A_{s+1}=A_s+\frac{y^{(s)}{y^{(s)}}^t}{{y^{(s)}}^t p^{(s)}}-\frac{A_s p^{(s)}{p^{(s)}}^t A_s}{{p^{(s)}}^t A_s p^{(s)}} \tag{8.20}$$

8.2.7.4 Fórmula DFP

$$\begin{aligned}A_{s+1} &= A_s+\frac{\left(y^{(s)}-A_s p^{(s)}\right){y^{(s)}}^t+y^{(s)}\left(y^{(s)}-A_s p^{(s)}\right)^t}{{y^{(s)}}^t p^{(s)}}- \\ &-\frac{\left(y^{(s)}-A_s p^{(s)}\right)^t p^{(s)}y^{(s)}{y^{(s)}}^t}{\left({y^{(s)}}^t p^{(s)}\right)^2}\end{aligned} \tag{8.21}$$

Este método y el anterior son los dos mejores. Además, son simétricos, ya que

$$H_{BFGS} = A_{DFP} \qquad H_{DFP} = A_{BFGS}$$

Si, además, se intercambian $y^{(s)}$ y $p^{(s)}$.

Se puede hacer un conjunto de métodos cuasinewton conocido como **familia Broyden**, si se usa un parámetro $\phi \in \mathbb{R}$ que pueda tomar cualquier valor salvo el degenerado:

$$A(\phi) = (1-\phi) A_{DFP} + \phi A_{BFGS}$$

Podemos ver a continuación el algoritmo en Matlab que implementa los métodos cuasi-Newton para optimización de funciones.

Algoritmo 8.4: Métodos cuasi-Newton para optimización de funciones

```
% Metodos cuasi-Newton para optimizacion de funciones
%
% Uso: [sol,iter]=cnewtonop(G,x0,maxiter,modif,H0,tol,metodo)
%
% - G es la función del gradiente
% - modif es la modificacion cuasinewton (rango1, psb,
%   bfgs o dfp)
% - H0 es la aproximacion inicial a la Jacobiana (por defecto
%   la matriz identidad)
% - metodo es la forma de calcular el alfa (ninguno,
%   lineaex o lineaaprox)
function [xs,iter]=cnewtonop(G,x0,maxiter,modif,H0,tol,metodo)
if nargin<7
   metodo = 'ninguno';
end;
if nargin<4
   error('Faltan argumentos de entrada');
   return;
end;
if nargin<5
   H0 = eye(length(x0));
end;
if nargin<6
   tol=1e-5;
end;
switch lower(modif)
case {'rango1'}
   disp('   Formula de rango 1');
case {'psb'}
   disp('   Formula PSB');
case {'bfgs'}
   disp('   Formula BFGS');
case {'dfp'}
   disp('   Formula DFP');
otherwise
   error('Metodo para aproximar la Jacobiana desconocido');
end;
```

```
fin=0;
iter=1;
grad = feval(G,x0);
H = H0;
ds=-H*grad;
switch lower(metodo)
case {'ninguno'}
   disp('   alfa = 1');
   alpha = 1;
case {'lineaex'}
   disp('   Calcular alfa por linea exacta');
   alpha=(ds'*grad)/(ds'*H*ds);
otherwise
   error('Metodo para calcular alfa desconocido');
end;
x(:,1)=x0;
while (fin==0)&(iter<maxiter)
   x(:,iter+1)=x(:,iter)+alpha*ds;
   iter=iter+1;
   gradant = grad;
   grad = feval(G,x(:,iter));
   Hant = H;
   y = grad-gradant;
   p = -alpha*Hant*gradant;
   switch lower(modif)
   case {'rango1'}
      H = Hant+(p-Hant*y)*(p-Hant*y)'/(y'*(p-Hant*y));
   case {'psb'}
      Aant = Hant^-1;
      A = Aant + ((y-Aant*p)*p'+p*(y-Aant*p)')/(p'*p)-
          ((y-Aant*p)'*p*p*p')/(p'*p)^2;
      H = A^-1;
   case {'bfgs'}
      H = Hant + ((p-Hant*y)*p'+p*(p-Hant*y)')/(p'*y) -
          ((p-Hant*y)'*y*p*p')/(p'*y)^2;
   case {'dfp'}
      H = Hant + (p*p')/(p'*y) - (Hant*y*y'*Hant)/(y'*Hant*y);
   otherwise
      error('Metodo para aproximar la Jacobiana desconocido');
   end;
   ds=-H*grad;
   if norm(ds,Inf)<tol
      fin=1;
   else
      switch lower(metodo)
      case {'ninguno'}
         alpha = 1;
      case {'lineaex'}
         alpha=(ds'*grad)/(ds'*H*ds);
      end;
   end;
```

```
end;
xs=x(:,iter);
if iter==maxiter
   disp('Se ha alcanzado el maximo de iteraciones');
else
   disp('Se ha encontrado un punto que cumple la tolerancia');
end;
```

♦

8.3 Problema de mínimos cuadrados no lineal

Tenemos una colección de puntos

$$\{(t_i, y_i), i = 1 : m\}$$

y queremos optimizar la función

$$\min_{a_1,a_2,\ldots,a_n} \frac{1}{2}\sum_{i=1}^{m} \left(y\left(t_i, a_1, a_2, \ldots, a_n\right) - y_i\right)^2 = \frac{1}{2} R\left(x\right)^t R\left(x\right) = \frac{1}{2}\left\|R\left(x\right)\right\|_2^2$$

con una función modelo que nos dice cómo depende y de t, por ejemplo

$$y\left(t\right) = a_1\, e^{a_2 t} + a_3\, e^{a_4 t} \cos\left(a_5 t + a_6\right)$$

siendo los a_i las variables del vector x del problema, queremos minimizar los residuos al cuadrado

$$\min_{x\in\mathbb{R}^n} f\left(x\right) = \frac{1}{2}\sum_{i=1}^{m} r_i^2\left(x\right) = \frac{1}{2}\left\|R\left(x\right)\right\|_2^2 \qquad x = \begin{pmatrix} a_1 & a_2 & \ldots & a_n \end{pmatrix}^t$$

$R\left(x\right)$ es el vector de residuos $y\left(t_i, a_1, a_2, \ldots, a_n\right) - y_i$

$$R\left(x\right) = \begin{pmatrix} r_1\left(x\right) & r_2\left(x\right) & \ldots & r_m\left(x\right) \end{pmatrix}^t$$

Si definimos

$$J\left(x\right) = \left[\frac{\partial r_i}{\partial x_j}\right]$$

podemos escribir el problema de la siguiente forma

$$\nabla f\left(x\right) = J^t\left(x\right) R\left(x\right) \tag{8.22}$$

$$\nabla^2 f\left(x\right) = J^t\left(x\right) J\left(x\right) + \sum_{i=1}^{m} r_i\left(x\right) \nabla^2 r_i\left(x\right) \tag{8.23}$$

El gradiente no es difícil de obtener, pero la Hessiana sí. La forma habitual de resolverlo es Gauss-Newton, Newton, cuasi-Newton o gradiente.

8.3.1 Método de Gauss-Newton

En este método, se quita el término de la Hessiana, $\nabla^2 r_i\left(x\right)$, y haciendo $\alpha_s = 1$, nos queda

$$\boxed{x^{(s+1)} = x^{(s)} - \left(J^t\left(x^{(s)}\right) J\left(x^{(s)}\right)\right)^{-1} J^t\left(x^{(s)}\right) R\left(x^{(s)}\right)} \tag{8.24}$$

siempre que $J\left(x\right)$ tenga rango columna completo, entonces la dirección es de descenso y $J^t J$ es invertible.

Este método converge localmente, como todos los de Newton.

8.3.2 Método de Levenberg-Marquardt

También llamado **metodo de región de confianza** o de paso restringido, usa el mismo modelo cuadrático que el método de Gauss-Newton, pero añadiendo una restricción, que es la que marca la región de confianza.

La dirección d es tal que

$$\min_d q_s(d) = f\left(x^{(s)}\right) + d^t \nabla f\left(x^{(s)}\right) + \frac{1}{2} d^t \nabla^2 f\left(x^{(s)}\right) d \tag{8.25}$$

restringiendo d a

$$\|d\|_2 \leq h_s$$

(norma dos porque estamos haciendo mínimos cuadrados), siendo h_s tal que se restringe la longitud de paso a una región alrededor de $x^{(s)}$ de forma que se confía que $q_s(d)$ modela a f.

Teorema 8.2. La dirección $d^{(s)}$ es solución del problema (8.25) si y sólo si existe $\mu \geq 0$ tal que

$$\left(J^t\left(x^{(s)}\right) J\left(x^{(s)}\right) + \mu I\right) d^{(s)} = -J^t\left(x^{(s)}\right) R(x)$$

con

$$\mu\left(h_s - \left\|d^{(s)}\right\|_2\right) = 0$$

y

$$J^t\left(x^{(s)}\right) J\left(x^{(s)}\right) + \mu I$$

es semidefinida positiva. Si es definida positiva, entonces $d^{(s)}$ es solución única.

El sistema resultante es de $n+1$ componentes, las n de $d^{(s)}$ y μ.

Si $\mu = 0$ entonces la dirección calculada es la dirección de Gauss-Newton

$$d_N^{(s)} = -\left(J^t\left(x^{(s)}\right) J\left(x^{(s)}\right)\right)^{-1} J^t\left(x^{(s)}\right) R\left(x^{(s)}\right)$$

por lo que $x^{(s)} + d_N^{(s)}$ está en la región de confianza, y por tanto, la solución está en la frontera y debe satisfacer

$$\phi(\mu) = h_s - \|d(\mu)\|_2 = 0 \tag{8.26}$$

siendo $d(\mu) = -\left(J^t\left(x^{(s)}\right) J\left(x^{(s)}\right) + \mu I\right)^{-1} J^t\left(x^{(s)}\right) R\left(x^{(s)}\right)$.

La forma de resolver esta ecuación puede ser por dos métodos conocidos como **paso garfio *y* paso pata de perro**.

Teniendo el siguiente problema

$$\begin{cases} \phi(\mu) = \mu\left(h_s - \|d(\mu)\|_2\right) = 0 \\ d(\mu) = -\left(J^t J + \mu I\right)^{-1} J^t R \end{cases}$$

Aplicándole el método de Newton,

$$\begin{cases} \mu_{i+1} = \mu_i - \frac{\|d(\mu)\|_2}{h_s} \frac{\phi(\mu_i)}{\phi'(\mu_i)} \\ \phi'(\mu) = \frac{d^t(\mu)\left(J^t J + \mu I\right)^{-1} d(\mu)}{\|d(\mu)\|_2} \end{cases}$$

Esto es el **método de paso garfio**. Se van calculando los μ_i hasta que se cumpla $\mu\left(h_s - \left\|d^{(s)}\right\|_2\right) = 0$. Si la solución $\mu = 0$ (Gauss-Newton) nos deja en la región de confianza ($\|d\|_2 \leq h_s$), entonces va bien el método, si no, hay que buscar otra para $\mu \neq 0$.

En el **método paso pata de perro**, se toma

$$d^{(s)} = -\nabla f\left(x\right)$$

y el punto es aquel que da la búsqueda en línea exacta

$$x^{(s)} - \frac{\left\|\nabla f\left(x^{(s)}\right)\right\|^2}{\nabla f^t\left(x^{(s)}\right)\nabla^2 f\left(x^{(s)}\right)\nabla f\left(x^{(s)}\right)}\nabla f\left(x^{(s)}\right)$$

También llamado **punto de Cauchy**.

8.4 Programación no lineal

8.4.1 Definición y condiciones de mínimo

Tratamos ahora de minimizar una función con restricciones, siendo la función y/o las restricciones no lineales. Esto es,

$$\min_{x\in\mathbb{R}^n} f\left(x\right) \qquad x \text{ sujeto a } \begin{cases} h_i\left(x\right) = 0 & i = 1:t \\ \\ g_i\left(x\right) \geq 0 & i = 1:m \end{cases}$$

Donde f, g_i y h_i son de clase 2 en $\mathbb{R}^n$ y alguna no lineal como ya hemos dicho.

Llamamos $\mathcal{F}$ al conjunto de puntos que cumplen las restricciones. A todo punto $x \in \mathcal{F}$ se le llama **punto factible**, por lo que a $\mathcal{F}$ se le llama **región factible**.

- Definimos la **función Lagrangiana** asociada al problema de programación no lineal como

$$\boxed{L\left(x,\lambda\right) = f\left(x\right) - \sum_{i=1}^{t}\lambda_i h_i\left(x\right) - \sum_{i=1}^{m}\lambda_{i+i} g_i\left(x\right)} \tag{8.27}$$

  donde hemos llamado a

$$\lambda = \begin{pmatrix} \lambda_1 & \lambda_2 & \dots & \lambda_{t+m} \end{pmatrix}^t$$

  el vector de **multiplicadores de Lagrange**.

- Definición de **mínimo local restringido**

$$\boxed{f\left(x^*\right) \leq f\left(x\right) \qquad \forall x \in B\left(x^*,\varepsilon\right) + \mathcal{F}}$$

- **Restricción activa** en un punto: una restricción $h_i\left(x\right)$ (de igualdad) o $g_i\left(x\right)$ en un punto $x \in \mathcal{F}$ es activa si

$$h_i\left(x\right) = 0 \qquad \text{o} \qquad g_i\left(x\right) = 0$$

  Las restricciones de igualdad siempre deben ser activas

- **Restricción inactiva**: una restricción $g_i\left(x\right)$ es inactiva en $x \in \mathcal{F}$ si

$$g_i\left(x\right) > 0$$

✏ Definimos el **arco de curva diferenciable y factible** $x(\xi) \in \mathcal{F}$ como una aplicación $x(\xi) : \mathbb{R} \to \mathbb{R}^n$.

✏ Vamos a buscar las **condiciones necesarias de mínimo local restringido**:
Nuestro propósito es buscar el mínimo en la frontera de la región factible, por lo que habrá que verificar que la función f no decrece alrededor del punto x^* por el arco de curva diferenciable.

$$f(x) \to \frac{d}{d\xi} f(x(\xi)) = x'(\xi)^t \nabla f(x(\xi))$$

$$\frac{d^2}{d\xi^2} f(x(\xi)) = x'(\xi)^t \nabla^2 f(x(\xi)) x'(\xi) + x''(\xi)^t \nabla f(x(\xi))$$

Si queremos que $x^* = x(0)$ sea un punto crítico,

$$\frac{d}{d\xi} f(x(0)) = x'(0) \nabla f(x^*) = 0$$

Llamamos **direcciones factibles** a

$$p = x'(0)$$

Entonces, encontramos la primera condición necesaria para mínimo local:

$$\boxed{p^t \nabla f(x^*) = 0}$$

Para que además la función no decrezca al movernos por un entorno de x^*, debe ser

$$\frac{d^2}{d\xi^2} f(x(0)) \text{ no negativa}$$

Por tanto, la segunda condición necesaria de mínimo local es

$$\boxed{p^t \nabla^2 f(x^*) p + x''(0)^t \nabla f(x^*) \geq 0}$$

Como p es la dirección de la tangente en cada arco, serán direcciones factibles si $x(\xi) \in \mathcal{F}$, $x(0) = x^*$ y $p = x'(0)$.
Para las restricciones de igualdad,

$$h_i(x(\xi)) = 0$$

Es constante para todo ξ, así que

$$\frac{d}{d\xi} h_i(x(\xi)) = 0 \Rightarrow p \nabla h_i(x^*) = 0 \qquad i = 1 : t$$

Montamos la matriz

$$A = \begin{pmatrix} \nabla h_1(x^*) & \dots & \nabla h_t(x^*) \end{pmatrix}$$

Para todas las direcciones que pertenezcan al núcleo (ortogonales a $\nabla h_i(x^*)$), se cumple

$$A^t p = 0 \qquad p \in \aleph$$

y

$$p = Z$$

es una matriz cuyas columnas son base del núcleo de A^t.

$$p_Z^t Z^t \nabla f(x^*) = 0 \Rightarrow \boxed{Z^t \nabla f(x^*) = 0}$$

es condición necesaria. Se llama **gradiente reducido en** x^* a

$$Z^t \nabla f(x^*)$$

Donde la matriz Z también depende del punto.
$\nabla f(x^*)$ es ortogonal al núcleo de A^t, por lo que pertenece a la imagen de A, y por tanto

$$\nabla f(x^*) = A\lambda$$

siendo λ los multiplicadores de Lagrange, como ya hemos visto. Para la derivada segunda se tiene

$$\frac{d^2}{d\xi^2} h_i(x(\xi)) = 0 = p^t \nabla^2 h_i(x^*) p + x''(0)^t \nabla h_i(x^*)$$

que se puede escribir como

$$\sum_i \lambda_i \left(p^t \nabla^2 h_i(x^*) p + x''(0)^t \nabla h_i(x^*) \right) = 0$$

Entonces, podemos escribir

$$x''(\xi)^t \nabla f(x^*) = -\sum \lambda_i p^t \nabla^2 h_i p$$

$$\boxed{p^t \left(\nabla^2 f(x^*) - \sum_i \lambda_i \nabla^2 h_i(x^*) \right) p \geq 0}$$

Es decir, que la Hessiana de la Lagrangiana (considerando sólo las restricciones de igualdad) sea semidefinida positiva.
Por tanto, tenemos ya las **condiciones necesarias de mínimo** (si sólo tenemos en cuenta las restricciones de igualdad), y son que el gradiente reducido sea nulo y la Hessiana de la Lagrangiana sea semidefinida positiva para todas las direcciones factibles. Si fuera definida positiva, sería condición suficiente, ya que hay decrecimiento.
Podemos independizar la condición de que la Hessiana sea semidefinida positiva de las direcciones factibles, definiendo la **Hessiana reducida** como

$$Z^t \left(\nabla^2 f(x^*) - \sum_i \lambda_i \nabla^2 h_i(x^*) \right) Z$$

Ahora añadimos las restricciones de desigualdad. Nos importan las restricciones activas:

$$g_i(x(\xi)) = 0$$

Así, formamos la nueva matriz A añadiendo dichas restricciones activas

$$A_A = \begin{pmatrix} \nabla h_1(x^*) & \dots & \nabla h_t(x^*) & \dots & \nabla g_i(x^*) \end{pmatrix} \qquad i \text{ tal que } g_i \text{ es restricción activa}$$

Las condiciones serán ahora

$$Z_A^t \nabla f(x^*) = 0$$

$$Z_A^t \nabla_{xx}^2 L(x^*, \lambda) Z_A \qquad \text{semidefinida positiva (al menos)}$$

Si ahora $g_i(x(\xi)) \geq 0$ para algunos i, entonces debe ser $p^t \nabla g_i(x^*) \geq 0$ para que haya decrecimiento, así

$$p^t \nabla f(x^*) = p^t A_A \lambda \geq 0$$

Donde $p^t \nabla f(x^*)$ implica moverse por uno de los arcos de curva y alejarse de los otros. Nos queda entonces la **condición de optimalidad de primer orden**

$$\boxed{\nabla f(x^*) = \sum_i \lambda_i \nabla g_i(x) \qquad \lambda_i \geq 0} \tag{8.28}$$

Si se asigna un multiplicador λ_i, positivo, a cada restricción, sólo nos preocupamos de las activas: o es activa en el punto o el multiplicador es cero.
Ahora la **Hessiana reducida** es

$$Z_A^t \left(\nabla^2 f(x^*) - \sum_i \lambda_i \nabla^2 h_i(x^*) - \sum_i \lambda_{t+i} \nabla^2 g_i(x^*) \right) Z_A$$

Esto lo hemos hecho suponiendo que A_A tiene rango completo por columnas, así serán válidas las condiciones. Ahora vamos a resumir las condiciones.

✏ **Condiciones necesarias de mínimo local restringido**:

$$\boxed{\begin{array}{l} p^t \nabla f(x^*) = 0 \\ \\ Z_A^t \left(\nabla^2 f(x^*) - \sum_i \lambda_i \nabla^2 h_i(x^*) - \sum_i \lambda_{t+i} \nabla^2 g_i(x^*) \right) Z_A \geq 0 \end{array}} \tag{8.29}$$

✏ **Condiciones suficientes de mínimo local restringido**:

$$\boxed{\begin{array}{l} p^t \nabla f(x^*) = 0 \\ \\ Z_A^t \left(\nabla^2 f(x^*) - \sum_i \lambda_i \nabla^2 h_i(x^*) - \sum_i \lambda_{t+i} \nabla^2 g_i(x^*) \right) Z_A > 0 \end{array}} \tag{8.30}$$

✏ Un par (x^*, λ) es un **par Kuhn-Tucker** para el problema de programación no lineal dado si $x^* \in \mathcal{F}$ y además se satisface

$$\nabla_x L(x^*, \lambda) = 0 \qquad \lambda_{t+i} \geq 0 \text{ para } i = 1:m \qquad \lambda_{t+i} g_i(x^*) = 0 \text{ para } i = 1:m$$

8.4.2 Métodos de penalización

Lo habitual para resolver problemas restringidos es trasladarlos a problemas no restringidos, cambiando la función a minimizar. Esto es lo que se conoce como métodos de penalización.

Nos preocupan los mínimos que se puedan encontrar en la frontera, ya que son los difíciles de decidir si son o no mínimos. No nos afecta este problema en mínimos que estén dentro de la región factible.

Así pues, nuestra metodología a seguir es generar una sucesión de subproblemas no restringidos

$$\min_{x \in \mathbb{R}^n} F(x, \sigma)$$

de los cuales obtendremos las soluciones x_σ, las cuales, bajo ciertas condiciones, harán que la sucesión que generan dé una solución aproximada del problema restringido.

8.4.2.1 Función de penalización cuadrática

Se hacen penalizaciones sucesivas hasta que la solución cumpla las restricciones con la función

$$\boxed{F_Q(x, \sigma) = f(x) + \frac{\sigma}{2} \left(\sum_{i=1}^{t} h_i^2(x) + \sum_{i=1}^{m} \min\{0, g_i(x)\} \right)^2} \tag{8.31}$$

El segundo término de penalización, el asociado a $g_i(x)$, sólo penaliza si $g_i < 0$. El primero lo hace sólo si no se cumple la restricción de igualdad, $h_i(x) = 0$.

Se aplica tomando una sucesión de valores de σ_k crecientes, que normalmente es $\sigma_k = 10^{k-1}$ para $k = 1, 2, \ldots$

Se resuelven entonces los sucesivos subproblemas hasta que las restricciones y las condiciones de mínimo se cumplan. La solución de cada subproblema se puede usar como punto inicial para el siguiente.

Si hay inecuaciones de restricción, la función $F_Q(x, \sigma)$ tiene derivadas segundas discontinuas en los puntos en los cuales alguna de esas restricciones se activa.

8.4.2.2 Función de penalización valor absoluto

Se obtiene una función no diferenciable, lo que puede ser un problema. Sin embargo, tiene la ventaja de que basta pasar un umbral de σ para resolver el problema. Se trata por tanto de una **penalización exacta**.

$$\boxed{F_1(x, \sigma) = f(x) + \sigma \left(\sum_{i=1}^{t} |h_i(x)| + \sum_{i=1}^{m} \text{máx}\{0, -g_i(x)\} \right)} \tag{8.32}$$

8.4.2.3 Función de penalización barrera

Se usa cuando en el problema restringido sólo hay inecuaciones de restricción ($t = 0$). Hay dos tipos de funciones:

- ✏ logarítmica

$$\boxed{F_B(x, \rho) = f(x) - \rho \sum_{i=1}^{m} \ln(g_i(x))} \tag{8.33}$$

- ✏ inversa

$$\boxed{F_Q(x, \sigma) = f(x) + \rho \sum_{i=1}^{m} \frac{1}{g_i(x)}} \tag{8.34}$$

Para la logarítmica, el valor de la función crece mucho, lo que se compensa haciendo que los valores sucesivos de ρ sean decrecientes y tendiendo a cero, por lo que los subproblemas se resuelven manteniéndose en el interior del conjunto factible.

8.4.2.4 Función Lagrangiana aumentada

Es otra penalización exacta. La función es, ahora

$$\boxed{\begin{aligned} L(x, y, \lambda, \sigma) &= f(x) - \sum_{i=1}^{t} \lambda_i h_i(x) - \sum_{i=1}^{m} \lambda_{t+i} (g_i(x) - y_i) \\ &\quad + \tfrac{\sigma}{2} \left(\sum_{i=1}^{t} h_i^2(x) + \sum_{i=1}^{m} (g_i(x) - y_i)^2 \right) \end{aligned}} \tag{8.35}$$

Se han introducido variables "artificiales", $y_i\ i = 1 : m$ para que todas las restricciones sean por igualdad. Este método requiere una estimación y actualización de los multiplicadores λ_i.

Ejemplo 8.2:

Dado el siguiente problema de optimización restringida

$$\max_{x\in\mathbb{R}^2} x_1^2 + x_2^2 - 1.5x_1 + 1.5x_2 \qquad x \text{ sujeto a } \begin{cases} (1-x_1)^3 \geq x_2 \\ \\ x_1^2 + x_2^2 \leq 1 \end{cases}$$

1. Estudiar si hay máximos locales restringidos con ambas restricciones activas
2. Estudiar la solución óptima del problema

Solución:

1. Para la primera parte, primero tenemos que darnos cuenta que nos están pidiendo maximizar una función y toda la metodología vista es para minimizar, así que hay que transformar el problema en

$$\min_{x\in\mathbb{R}^2} -x_1^2 - x_2^2 + 1.5x_1 - 1.5x_2 \qquad x \text{ sujeto a } \begin{cases} (1-x_1)^3 - x_2 \geq 0 \\ \\ 1 - x_1^2 - x_2^2 \geq 0 \end{cases}$$

Definimos las zonas activas con el corte de las curvas:

$$\begin{cases} x_2 = (1-x_1)^3 \\ \\ x_1^2 + (1-x_1)^6 = 1 \end{cases} \Rightarrow \begin{array}{c} x_1^{(1)} = 0 \\ \\ x_1^{(2)} = 1 \end{array}$$

Por tanto, los puntos activos son

$$x^{(1)} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad x^{(2)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Y ahora verificamos las condiciones de mínimo, calculando previamente los gradientes

$$\nabla f(x) = \begin{pmatrix} -2x_1 + 1.5 \\ \\ -2x_2 - 1.5 \end{pmatrix} \qquad \nabla g_1(x) = \begin{pmatrix} -3(1-x_1)^2 \\ \\ -1 \end{pmatrix} \qquad \nabla g_2(x) = \begin{pmatrix} -2x_1 \\ \\ -2x_2 \end{pmatrix}$$

Y evaluamos en los puntos:

$$\nabla f\left(x^{(1)}\right) = \begin{pmatrix} 1.5 \\ -3.5 \end{pmatrix} \qquad \nabla g_1\left(x^{(1)}\right) = \begin{pmatrix} -3 \\ -1 \end{pmatrix} \qquad \nabla g_2\left(x^{(1)}\right) = \begin{pmatrix} 0 \\ -2 \end{pmatrix}$$

$$\nabla f\left(x^{(2)}\right) = \begin{pmatrix} -0.5 \\ -1.5 \end{pmatrix} \qquad \nabla g_1\left(x^{(2)}\right) = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \qquad \nabla g_2\left(x^{(2)}\right) = \begin{pmatrix} -2 \\ 0 \end{pmatrix}$$

Si imponemos la condición de optimalidad de primer orden, para el primer punto es

$$\nabla f\left(x^{(1)}\right) = \begin{pmatrix} \nabla g_1\left(x^{(1)}\right) & \nabla g_2\left(x^{(1)}\right) \end{pmatrix} \begin{pmatrix} \lambda_1^{(1)} \\ \\ \lambda_2^{(1)} \end{pmatrix} \Rightarrow \lambda^{(1)} = \begin{pmatrix} -0.5 \\ 2 \end{pmatrix}$$

Dado que $\lambda_1^{(1)} \leq 0$, puede asegurarse que $x^{(1)}$ no es mínimo local restringido, y máximo local en el problema original. Para el segundo punto es

$$\nabla f\left(x^{(2)}\right) = \begin{pmatrix} \nabla g_1\left(x^{(2)}\right) & \nabla g_2\left(x^{(2)}\right) \end{pmatrix} \begin{pmatrix} \lambda_1^{(2)} \\ \\ \lambda_2^{(2)} \end{pmatrix} \Rightarrow \lambda^{(2)} = \begin{pmatrix} 1.5 \\ 0.25 \end{pmatrix}$$

Como ahora ambos multiplicadores, $\lambda_1^{(2)}$ y $\lambda_2^{(2)}$, son positivos, entonces se puede asegurar que se cumplen las condiciones de primer orden en el punto $x^{(2)}$. Como además los gradientes de las funciones restricción en el punto forman una base, se satisfacen las de segundo orden. Por tanto, $x^{(2)}$ es mínimo local restringido, máximo local en el problema original.

2. La función objetivo crece al desplazarse hacia la parte superior izquierda de la región factible, por lo que probablemente existe una solución óptima x^* siendo sólo activa la segunda restricción en dicho punto. Con la condición de optimalidad de primer orden tenemos

$$\nabla f(x^*) = \nabla g_2(x^*)\lambda^* \Rightarrow \begin{pmatrix} -2x_1^* + 1.5 \\ -2x_2^* - 1.5 \end{pmatrix} = \begin{pmatrix} -2x_1^* \\ -2x_2^* \end{pmatrix} \lambda^*$$

de donde se tiene

$$x_1^* = \frac{1.5}{2(1-\lambda^*)} \qquad x_2^* = \frac{-1.5}{2(1-\lambda^*)}$$

Como la segunda restricción es activa en x^*, entonces

$$(x_1^*)^2 + (x_2^*)^2 = \frac{9}{8(1-\lambda^*)^2} = 1$$

permite el cálculo de un par Kuhn-Tucker (x^*, λ) para

$$x^* = \begin{pmatrix} -1/\sqrt{2} \\ \\ 1/\sqrt{2} \end{pmatrix} \qquad \lambda = 1 + \frac{3}{2\sqrt{2}}$$

Ahora comprobamos el cumplimiento de la condición de optimalidad de segundo orden, siendo la Hessiana de la Lagrangiana

$$\nabla^2 L(x^*, \lambda^*) = \begin{pmatrix} -2 & 0 \\ 0 & -2 \end{pmatrix} - \lambda^* \begin{pmatrix} -2 & 0 \\ 0 & -2 \end{pmatrix} = \begin{pmatrix} 3/\sqrt{2} & 0 \\ 0 & 3/\sqrt{2} \end{pmatrix}$$

En el punto, tenemos

$$(\nabla g_2(x^*))^t p = 0 \Rightarrow \begin{pmatrix} \sqrt{2} \\ -\sqrt{2} \end{pmatrix}^t p = 0$$

por lo que

$$p = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

es la base del núcleo de A^t y por tanto es la matriz que nos sirve para hacer la Hessiana reducida, que queda

$$Z^{*^t} \nabla^2 L(x^*, \lambda) Z^* = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} 3/\sqrt{2} & 0 \\ 0 & 3/\sqrt{2} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 3\sqrt{2}$$

Como es definida positiva, x^* es mínimo local restringido del problema, y máximo local del problema original, solución óptima con valor objetivo

$$-f(x^*) = 1 + 1.5\sqrt{2}$$

■

APÉNDICE A
Resumen

A.1 Tema 1: Errores

A.1.1 Generación y propagación de errores

Dados un número exacto, x, y su aproximación, x^* (redondeada: x_R; truncada: x_C), se definen

- **error de** x^*

$$\boxed{E = x^* - x}$$

- **error absoluto de** x^*

$$\boxed{|E| = |x^* - x|}$$

- **error relativo de** x^*

$$\boxed{\varepsilon = \frac{x^* - x}{x}}$$

Se pueden ver las **cotas de error** según el tipo de representación:

- Representación redondeada:

$$fl\,(x) = x_R = \begin{cases} u \times \beta^e & |v| < 1/2 \\ u \times \beta^e + \beta^{e-t} & u > 0, |v| \geq 1/2 \\ u \times \beta^e - \beta^{e-t} & u < 0, |v| \geq 1/2 \end{cases}$$

  - cota del error

$$E_R = \begin{cases} -v \times \beta^{e-t} & |v| < 1/2 \\ (1-v)\,\beta^{e-t} & u > 0;\ |v| \geq 1/2 \\ (-1-v)\,\beta^{e-t} & u < 0;\ |v| \geq 1/2 \end{cases}$$

  - cota del error absoluto

$$|E_R| \leq \frac{\beta^{e-t}}{2}$$

- ☞ cota del error relativo

$$|\varepsilon_R| \leq \frac{\beta^{1-t}}{2}$$

✏ Representación truncada:

$$fl\,(x) = x_C = u \times \beta^e$$

- ☞ cota del error

$$E_C = -v \times \beta^{e-t}$$

- ☞ cota del error absoluto

$$|E_C| = |v| \times \beta^{e-t} < \beta^{e-t}$$

- ☞ cota del error relativo

$$|\varepsilon_C| = \frac{|fl\,(x) - x|}{|x|} < \frac{\beta^{e-t}}{|x|} \leq \frac{\beta^{e-t}}{\frac{1}{\beta}\beta^e} = \beta^{1-t}$$

Otras definiciones de interés son:

✏ Un valor x^* aproximado a x tiene k **dígitos fraccionarios exactos o correctos** en base β si cumple:

$$\boxed{|x^* - x| < \frac{1}{2}\beta^{-k}}$$

Aunque normalmente se suele hablar de dígitos correctos, en algunos libros se habla de dígitos exactos. Si por ejemplo aproximamos el número $\pi = 3.14159265\ldots$ por $\pi^* = 3.1416$, el dígito 6 será correcto pero no exacto si verifica la relación anterior.

✏ Un valor x^* aproximado a x tiene k **dígitos significativos** en base β si cumple:

$$\boxed{\left|\frac{x^* - x}{x}\right| < \frac{1}{2}\beta^{1-k}}$$

Visto de otra forma, sobre el número:

$$x^* = \underbrace{\underbrace{d_{n-1}\ldots d_1 d_0}_{n \text{ digitos}} . \underbrace{d_{-1}d_{-2}\ldots d_{-d}}_{d \text{ digitos fraccionarios}}}_{n+d=i \text{ digitos significativos}}$$

✏ También se puede definir un dígito d_k como **dígito significativo correcto** (se entiende redondeado) si cumple

$$|x^* - x| \leq \Delta\,(x^*) \leq \frac{1}{2}10^k \qquad k \leq n-1$$

A.2 Tema 2: Álgebra lineal numérica I

A.2.1 Factorización LU

Antes de aplicar los métodos de factorización LU, es recomendable asegurar que la factorización existe:

✏ se llaman **submatrices principales sucesivas** a

$$A_i = \begin{pmatrix} a_{11} & \dots & a_{1i} \\ \vdots & \ddots & \vdots \\ a_{i1} & \dots & a_{ii} \end{pmatrix} \qquad i = 1:n$$

y **menores principales sucesivos** a los determinantes $det\,(A_i)$ para $i = 1:n$.

✏ dada una matriz A de tamaño $n \times n$, si $det\,(A_i) \neq 0$ para $i = 1:n-1$, entonces la matriz admite factorización LU con $l_{ii} = 1$ o $u_{ii} = 1$ (condición suficiente)

✏ si la matriz es invertible y su factorización LU existe, entonces los menores principales sucesivos de la matriz hasta el orden $n-1$ son no nulos y la factorización es única (condición necesaria y suficiente)

✏ Si $det\,(A_i) = 0$ para algún i, entonces $det\,(A) = 0$ y/o A no es factorizable LU

A.2.1.1 Método de Crout

Es un método de factorización LU, en el que se hacen los elementos de la diagonal principal de la matriz U iguales a 1

$$\begin{array}{llll} l_{i1} = a_{i1} & i = 1:n & u_{11} = 1 & u_{1i} = \frac{a_{1i}}{l_{11}} \qquad i = 2:n \end{array}$$

$$j = 2:n \begin{cases} l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj} \quad i = j:n \\ u_{jj} = 1; \ \text{Si } j < n: \quad u_{ji} = \left(a_{ji} - \sum_{k=1}^{j-1} l_{jk}u_{ki} \right) / l_{jj} \qquad i = j+1:n \end{cases}$$

A.2.1.2 Método de Doolittle

Es otro método de factorización LU, en el que se hacen los elementos de la diagonal principal de la matriz L iguales a 1

$$\begin{array}{llll} u_{1i} = a_{1i} & i = 1:n & l_{11} = 1 & l_{i1} = \frac{a_{i1}}{u_{11}} \qquad i = 2:n \end{array}$$

$$j = 2:n \begin{cases} u_{ji} = a_{ji} - \sum_{k=1}^{j-1} l_{jk}u_{ki} \qquad i = j:n \\ l_{jj} = 1; \ \text{Si } j < n: \quad l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj} \right) / u_{jj} \qquad i = j+1:n \end{cases}$$

A.2.1.3 Método de Cholesky

✏ Se utiliza cuando la matriz del sistema es simétrica: $A = A^t$

✏ Una matriz A admite factorización Cholesky si y sólo si es definida positiva (sus menores principales sucesivos son positivos: $det\,(A_i) > 0$)

$$A = LL^t$$

$$A = \begin{pmatrix} l_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ l_{n1} & \dots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & \dots & l_{n1} \\ \vdots & \ddots & \vdots \\ 0 & \dots & l_{nn} \end{pmatrix}$$

$$l_{11} = \sqrt{a_{11}} \qquad l_{i1} = \frac{a_{j1}}{l_{11}} \qquad i = 2:n$$

$$j = 2:n \quad \begin{cases} l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2} \\ \text{Si } j < n; \quad l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) / l_{jj} \qquad i = j+1:n \end{cases}$$

✏ La matriz admite factorización Cholesky con elementos reales si es semidefinida positiva, o si

$$l_{ii}^2 = \frac{det\,(A_i)}{det\,(A_{i-1})} \geq 0 \qquad l_{11}^2 = a_{11} \geq 0$$

A.2.2 Definiciones de normas matriciales y vectoriales

✏ Se definen las **normas** de un vector $x \in \mathbb{R}^n$ de la siguiente forma:

$$\|x\|_1 = \sum_{j=1}^{n} |x_j|$$

$$\|x\|_2 = \sqrt{\sum_{j=1}^{n} |x_j|^2}$$

$$\|x\|_p = \left(\sum_{j=1}^{n} |x_j|^p \right)^{\frac{1}{p}}$$

$$\|x\|_\infty = \max_j |x_j|$$

✏ **Norma matricial natural o inducida**:

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|$$

Norma 1:

$$\|A\|_1 = \sup_{x \neq 0} \frac{\|Ax\|_1}{\|x\|_1} = \max_{i=1:n} \sum_{j=1}^{n} |a_{ji}|$$

Norma 2:

$$\|A\|_2 = \sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sqrt{\rho\,(A^t A)} = \max_i \sigma_i$$

Norma infinito:

$$\|A\|_\infty = \sup_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \max_{i=1:n} \sum_{j=1}^{n} |a_{ij}|$$

El ínfimo de todas las normas matriciales que pueden definirse de *A* es el **radio espectral** de la matriz:

$$\rho(A) = \inf_{\|\cdot\|} \|A\|$$

A.2.3 Métodos iterativos

A.2.3.1 Métodos de punto fijo: definición, consistencia y convergencia

$$x^{(s+1)} = Bx^{(s)} + c \qquad B \in \mathbb{R}^{n\times n}, \quad c \in \mathbb{R}^n, \quad s = 0, 1, \ldots$$

Teorema de consistencia del método de punto fijo. El método de punto fijo es consistente con un sistema de ecuaciones $Ax = b$ para una matriz A no singular si y sólo si $I - B$ es no singular y $c = (I - B)\, A^{-1}b$. La consistencia implica

$$solucion\,(A, b) = solucion\,(I - B, c) \Rightarrow A^{-1}b = x = (I - B)^{-1}\, c$$

La consistencia no asegura la convergencia.

Teorema de convergencia del método de punto fijo. El método de punto fijo, siendo consistente con un sistema, converge a $(I - B)^{-1}\, c$, con cualquier vector inicial $x^{(0)}$ si y sólo si $\rho(B) < 1$. Es condición necesaria y suficiente.

A.2.3.2 Cotas de error

✏ Para un método de punto fijo con matriz B se satisfacen las siguientes cotas de error:

$$\left\|x^* - x^{(s)}\right\| \leq \frac{\|B\|^{s-k} \left\|x^{(k+1)} - x^{(k)}\right\|}{1 - \|B\|} \qquad \text{para } k = 0, 1, \ldots, s-1$$

$$\left\|x^* - x^{(s)}\right\| \leq \|B\|^s \left\|x^* - x^{(0)}\right\|$$

Hay otras cotas que se pueden usar, por ejemplo una para determinar el número de iteraciones para conseguir una cota dada, haciendo $k = 0$:

$$\boxed{\left\|x^* - x^{(s)}\right\| \leq \frac{\|B\|^s}{1 - \|B\|} \left\|x^{(1)} - x^{(0)}\right\|}$$

✏ O también una **cota dinámica**, que nos puede servir para programar un método y acotar el número de iteraciones también por la cota de error (tolerancia):

$$\boxed{\left\|x^* - x^{(s)}\right\| \leq \frac{\|B\|}{1 - \|B\|} \left\|x^{(s)} - x^{(s-1)}\right\|}$$

✏ **Cota de error a priori**

$$\frac{\|B\|^{s-k} \left\|x^{(k+1)} - x^{(k)}\right\|}{1 - \|B\|} \leq \varepsilon$$

✏ **Cota de error a posteriori**

$$\left\|x^* - x^{(s)}\right\| \leq \frac{\|B\|^{s-k} \cdot \left\|x^{(s)} - x^{(s-1)}\right\|}{1 - \|B\|}$$

$$\left\|x^* - x^{(s)}\right\|_\infty \leq \frac{\|B\|_\infty^s \cdot \left\|x^{(1)} - x^{(0)}\right\|}{1 - \|B\|_\infty}$$

A.2.3.3 Métodos de descomposición o partición regular

✏ Los métodos clásicos de este tipo están definidos para matrices A con elementos de la diagonal principal no nulos y tienen la siguente partición:

$$A = D + L + R$$

Con D matriz diagonal, L triangular inferior y R triangular superior.

$$D = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & \dots & a_{nn} \end{pmatrix} \quad L = \begin{pmatrix} 0 & 0 & \dots & 0 \\ a_{21} & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix} \quad R = \begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ 0 & 0 & \dots & a_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}$$

A.2.3.4 Método de Jacobi

✏ La descomposición es

$$M = D \qquad N = -(L + R)$$

✏ Por lo que la matriz del método es

$$\boxed{B_J = -D^{-1}(L + R)}$$

✏ El cálculo del vector en cada iteración es

$$\boxed{x^{(s+1)} = -D^{-1}(R + L)\, x^{(s)} + D^{-1}b}$$

✏ Cálculo de cada componente:

$$\boxed{x_i^{(s+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(s)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(s)}}{a_{ii}} \qquad i = 1:n}$$

$$det(-D)\, det\left(-D^{-1}(L + R) - \lambda I\right) = det(\lambda D + L + R) = 0$$

✏ Por lo que hay que multiplicar por λ sólo las componentes de la diagonal principal de A para calcular los autovalores a partir del polinomio característico.

A.2.3.5 Método de Gauss-Seidel

- Las componentes ya calculadas se usan para calcular las siguientes en la misma iteración. Acelera la convergencia, sin embargo no se puede asegurar que en general sea mejor que Jacobi.
- En este caso la descomposición es

$$M = D + L \qquad N = -R$$

- Y la matriz del método

$$\boxed{B_{GS} = -\left(D + L\right)^{-1} R}$$

- El cálculo del vector en cada iteración es

$$\boxed{x^{(s+1)} = -D^{-1}Lx^{(s+1)} - D^{-1}Rx^{(s)} + D^{-1}b}$$

- Para el cálculo de cada componente se usa la fórmula:

$$\boxed{x_i^{(s+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(s+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(s)}}{a_{ii}} \qquad i = 1:n}$$

$$det\left(\lambda D + \lambda L + R\right) = 0$$

Hay que multiplicar las componentes de las matrices D y L por λ para obtener el polinomio característico y extraer los autovalores.

A.2.3.6 Teoremas de convergencia

- **Matriz de diagonal dominante***:* Se dice que una matriz A es de diagonal dominante estrictamente (por filas) si cumple:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \qquad \forall i$$

El elemento de la diagonal principal en valor absoluto es mayor que la suma de los valores absolutos de los elementos restantes de la fila.

- Si en un sistema $Ax = b$ es la matriz A de diagonal dominante estrictamente, entonces los métodos de Jacobi y Gauss-Seidel convergen a la solución del sistema
- Sea un sistema $Ax = b$ siendo A simétrica definida positiva y $A = M - N$. Si M es invertible, entonces el método de descomposición regular

$$x^{(s+1)} = M^{-1}Nx^{(s)} + M^{-1}b$$

converge a la solución si la matriz $M^t + N$ es definida positiva.

- Dada A simétrica tal que $M^t + N$ es definida positiva, el método es convergente si y sólo si A es definida positiva.
- Casos particulares:
  - Método de Jacobi:

$$M^t + N = D - L - R \neq A$$

☞ Método de Gauss-Seidel:

$$M^t + N = D + L^t - R = D + L^t - L^t = D$$

dado que A es simétrica, $R = L^t$

Teorema de Stein-Rosenberg *(convergencia de Jacobi y Gauss-Seidel).* Si se tiene un sistema $Ax = b$ con A tal que $a_{ii} > 0$ para $i = 1 : n$ y $a_{ij} \leq 0$ para $i \neq j$, entonces se cumple una y sólo una de las siguientes posibilidades:

1. $0 < \rho(B_{GS}) < \rho(B_J) < 1$, con lo que convergen ambos métodos.
2. $0 = \rho(B_{GS}) = \rho(B_J)$, con lo que también convergen ambos.
3. $1 < \rho(B_J) < \rho(B_{GS})$, con lo que no converge ninguno.
4. $1 = \rho(B_{GS}) = \rho(B_J)$, tampoco converge ninguno.

A.3 Tema 3: Álgebra lineal numérica II

A.3.1 Localización de autovalores

Teorema de los círculos de Gerschgorin. Dada una matriz A real y cuadrada de orden n, y definiendo el dominio unión de los círculos de Gerschgorin:

$$D = \bigcup_{i=1}^{n} C_i \qquad C_i = \{z \in \mathbb{C} \,/\, |z - a_{ii}| \leq r_i\} \qquad r_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad i = 1 : n$$

Si λ es un autovalor, entonces está dentro del dominio D.

✏ El centro de los círculos de Gerschgorin son las componentes de la diagonal principal en cada fila, y los radios la suma de los elementos de la fila exceptuando el de la diagonal principal.

✏ Cualquier punto del plano complejo que caiga fuera de los círculos de Gerschgorin no será autovalor, mientras que en un círculo de Gerschgorin aislado habrá un único autovalor.

✏ Si A es de diagonal dominante estrictamente, es invertible. Por otro lado, $A - \lambda I$ no puede ser de diagonal dominante.

Teorema de Brauer. Dada la matriz A real y cuadrada de orden n, y un entero m tal que $1 \leq m \leq n$ y

$$|a_{mm} - a_{ii}| > \sigma_m + \sigma_i \quad i = 1 : n,\, i \neq m$$

En ese caso, el círculo de Gerschgorin $C_m = \{z \in \mathbb{C} \,/\, |z - a_{mm}| \leq \sigma_m\}$ es disjunto de los demás círculos de Gerschgorin, y contiene un único autovalor de la matriz A.

A.3.2 Cálculo del polinomio característico

✏ El polinomio característico es

$$p(\lambda) = |A - \lambda I| = (-1)^n \left(\lambda^n + p_1 \lambda^{n-1} + \ldots + p_n\right)$$

A.3.2.1 Número de condición para el cálculo de autovalores

✏ Sólo para matrices diagonalizables:

$$V^{-1}AV = D$$

✏ El número de condición es

$$\boxed{N_C = \|V\| \left\|V^{-1}\right\|}$$

✏ Las matrices simétricas son bien condicionadas, ya que hemos dicho antes que toda matriz real simétrica (o compleja hermítica) es diagonalizable por semejanza ortogonal, por lo que

$$\|V\| \left\|V^{-1}\right\| = 1$$

A.3.2.2 Método de Le-Verrier

✏ Utiliza la relación entre las raíces de una ecuación y sus coeficientes (relaciones de Newton):

$$\boxed{\begin{array}{l} p_1 = -s_1 \\ p_k = -\frac{1}{k}\left(s_k + p_1 s_{k-1} + \ldots + p_{k-1}s_1\right) \quad k = 2:n \\ \qquad\qquad\qquad\qquad\qquad\qquad s_k = \sum_{i=1}^{n} \lambda_i^k = traza\left(A^k\right) \end{array}}$$

✏ Los s_k se obtienen como suma de los autovalores de potencias de la matriz A (que es también la suma de los elementos de la diagonal principal de la matriz).

✏ Se puede seguir el siguiente esquema:

1. Determinar las potencias de la matriz A

$$A^0, A^1, A^2, \ldots, A^n$$

2. Calcular las trazas como la suma de los elementos de la diagonal principal de A

$$s_1 = traza\left(A\right), s_2 = traza\left(A^2\right), \ldots, s_n = traza\left(A^n\right)$$

3. A continuación se obtienen los coeficientes del polinomio característico de la forma

$$p_1 = -s_1$$

$$p_2 = -\left(s_2 + p_1 s_1\right)/2$$

$$\vdots$$

$$p_n = -\left(s_n + p_1 s_{n-1} + \ldots + p_{n-1}s_1\right)/n$$

A.3.2.3 Método de Faddeev-Souriau

✏ Es una modificación del método anterior que reorganiza el cálculo. Partiendo de $B_0 = I_n$:

$A_1 = AB_0$	$p_1 = -traza\,(A_1)$	$B_1 = A_1 + p_1 I_n$
$A_2 = AB_1$	$p_2 = -\frac{traza(A_2)}{2}$	$B_2 = A_2 + p_2 I_n$
$\vdots$	$\vdots$	$\vdots$
$A_k = AB_{k-1}$	$p_k = -\frac{traza(A_k)}{k}$	$B_k = A_k + p_k I_n$

Al final del proceso, en virtud del teorema de Cayley-Hamilton, $B_k = 0$.

A.3.2.4 Si A es tridiagonal

$$A = \begin{pmatrix} a_1 & c_2 & \dots & 0 \\ b_2 & a_2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & b_n & a_n \end{pmatrix}$$

$$\begin{aligned} |A - \lambda I| &= \begin{vmatrix} a_1 - \lambda & c_2 & \dots & 0 \\ b_2 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \dots & b_n & a_n - \lambda \end{vmatrix} = \\ &= (a_n - \lambda)\,|A_{n-1} - \lambda I| - b_n \begin{vmatrix} a_1 - \lambda & & & \\ & \ddots & & \\ & & \ddots & \\ & & b_{n-1} & c_n \end{vmatrix} = \\ &= (a_n - \lambda)\,p_{n-1}(\lambda) - b_n c_n p_{n-2}(\lambda) \end{aligned}$$

Entonces, queda, para los polinomios sucesivos:

$$p_0(\lambda) = 1$$

$$p_1(\lambda) = a_1 - \lambda$$

$$p_k(\lambda) = (a_k - \lambda)\,p_{k-1}(\lambda) - b_k c_k p_{k-2}(\lambda) \qquad k = 2 : n$$

Para que funcione el método, deben ser todos los $b_i \neq 0$. Si hay algún $b_k = 0$ se hace por bloques:

$$A = \begin{pmatrix} \boxed{B} & \\ & \boxed{C} \end{pmatrix} \Rightarrow det\,(A - \lambda I) = det\,(B - \lambda I)\,det\,(C - \lambda I)$$

A.3.2.5 Si A es Hessenberg superior

Una matriz Hessenberg superior es una matriz triangular superior que también tiene elementos en la subdiagonal principal:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ b_2 & a_{22} & \dots & a_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & b_n & a_{nn} \end{pmatrix}$$

Con todos los $b_i \neq 0$ para $i = 2 : n$.

Al resolver el problema de autovalores (para $n = 4$):

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ b_2 & a_{22} & a_{23} & a_{24} \\ 0 & b_3 & a_{33} & a_{34} \\ 0 & 0 & b_4 & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \lambda \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

Se tiene un sistema compatible indeterminado. Por tanto, lo que se hace es darle un valor a x_4, por ejemplo

$$x_4 = 1 \equiv s_4(\lambda)$$

Seguimos resolviendo y tenemos

$$b_4 x_3 + a_{44} = \lambda \Rightarrow \underline{x_3 = \frac{1}{b_4}(\lambda - a_{44}) \equiv s_3(\lambda)}$$

$$b_3 x_2 + a_{33} s_3(\lambda) + a_{34} s_4(\lambda) = \lambda s_3(\lambda)$$

$$\underline{x_2 = \frac{1}{b_3}\left((\lambda - a_{33}) s_3(\lambda) - a_{34} s_4(\lambda)\right) \equiv s_2(\lambda)}$$

$$b_2 x_1 + a_{22} s_2(\lambda) + a_{23} s_3(\lambda) + a_{24} s_4(\lambda) = \lambda s_2(\lambda)$$

$$\underline{x_1 = \frac{1}{b_2}\left((\lambda - a_{22}) s_2(\lambda) - a_{23} s_3(\lambda) - a_{24} s_4(\lambda)\right) \equiv s_1(\lambda)}$$

$$a_{11} s_1(\lambda) + a_{12} s_2(\lambda) + a_{13} a_3(\lambda) + a_{14} s_4(\lambda) = \lambda s_1(\lambda)$$

$$c\left((\lambda - a_{11}) s_{11}(\lambda) - a_{12} s_2(\lambda) - a_{13} s_3(\lambda) - a_{14} s_4(\lambda)\right) = 0$$

Siendo c una constante por determinar. El coeficiente de λ^n que nos queda es

$$\frac{c}{b_2 b_3 b_4}$$

y como queremos que sea 1, entonces

$$c = b_2 b_3 b_4$$

Si alguno de los b_i es nulo, se puede hacer como en el caso de tridiagonal, por bloques.

Lo que hemos hecho se llama **método de Hyman**, que, para el caso general, queda la siguiente recurrencia

$$\boxed{\begin{aligned} &s_n(\lambda) = 1 \\ &s_{k-1}(\lambda) = -\frac{(a_{kk} - \lambda) s_k(\lambda) + \sum_{j=k+1}^{n} a_{kj} s_j(\lambda)}{b_k} \qquad k = n : -1 : 2 \end{aligned}}$$

Si $b_k \neq 0$. Si se tiene $b_i = 0$ para algún i, se pueden hacer las recurrencias de las submatrices cuadradas principales.

A.4 Tema 4: Interpolación y aproximación

A.4.1 Interpolación polinomial

A.4.1.1 Interpolación de Lagrange

✏ Usamos una base $\{l_0(x), l_1(x), \ldots, l_n(x)\}$ para el espacio P_n, de forma que:

$$p(x) = \sum_{k=0}^{n} y_k l_k(x)$$

$$p(x_i) = \sum_{k=0}^{n} y_k l_k(x_i) = y_i$$

$$\boxed{l_i(x) = \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}}$$

✏ Y si escribimos $\theta_{n+1}(x) = (x-x_0)\cdots(x-x_n)$, nos queda:

$$l_i(x) = \frac{\theta_{n+1}(x)}{(x-x_i)\,\theta'_{n+1}(x_i)}$$

A.4.1.2 Interpolación de Newton

✏ **Diferencias divididas**:

$$f[x_{i_0}, x_{i_1}, \ldots x_{i_k}] = \frac{\overbrace{f[x_{i_1}, \ldots, x_{i_k}]}^{(1)} - \overbrace{f[x_{i_0}, x_{i_1}, \ldots, x_{i_{k-1}}]}^{(2)}}{x_{i_k} - x_{i_0}}$$

con

$$f[x_{i_j}] = f(x_{i_j}) \Rightarrow f[x_{i_j}, x_{i_m}] = \frac{f[x_{i_m}] - f[x_{i_j}]}{x_{i_m} - x_{i_j}} \Rightarrow \ldots$$

✏ **Polinomio de interpolación**:

$$p_n(x) = f[x_0] + f[x_0, x_1](x-x_0) + \ldots + f[x_0, x_1, \ldots, x_n](x-x_0)(x-x_1)\cdots(x-x_{n-1})$$

Neville

Esquema de las diferencias divididas sucesivas:

x_0	$f[x_0]$			
x_1	$f[x_1]$	$f[x_0, x_1]$		
x_2	$f[x_2]$	$f[x_1, x_2]$	$f[x_0, x_1, x_2]$	
x_3	$f[x_3]$	$f[x_2, x_3]$	$f[x_1, x_2, x_3]$	$f[x_0, x_1, x_2, x_3]$

Aitken

Esquema de las diferencias divididas

x_0	$f[x_0]$			
x_1	$f[x_1]$	$f[x_0,x_1]$		
x_2	$f[x_2]$	$f[x_0,x_2]$	$f[x_0,x_1,x_2]$	
x_3	$f[x_3]$	$f[x_0,x_3]$	$f[x_0,x_1,x_3]$	$f[x_0,x_1,x_2,x_3]$

La única diferencia que hay entre ambos es al implementarlo, ya que en Neville no se podría sobreescribir $f[x_1,x_2]$ con $f[x_0,x_1,x_2]$ porque $f[x_1,x_2,x_3]$ necesita el valor de $f[x_1,x_2]$, mientras que en el esquema de Aitken sí que se podría sobreescribir, ya que al calcular $f[x_0,x_2,x_3]$ se usa $f[x_0,x_1]$ y $f[x_0,x_3]$, por lo que se puede ahorrar en memoria sobreescribiendo valores.

Al fin y al cabo, los valores que interesan son los que están formando la diagonal del esquema.

A.4.1.3 Error de interpolación

✏ Llamamos **error de interpolación** a la diferencia entre la función y el polinomio de orden n:

$$\boxed{E(x) = f(x) - p_n(x) = f[x_0,\ldots,x_n,x]\,\theta_{n+1}(x)}$$

$$\boxed{E_n(x) = \frac{f^{n+1)}(\xi)}{(n+1)!}\theta_{n+1}(x)}$$

A.4.2 Interpolación osculatoria (Hermite)

✏ Los **polinomios de Hermite** se dan cuando se conocen la función y la primera derivada en cada punto. Este es un caso frecuente.

✏ El grado del polinomio será

$$n = 2m + 1$$

✏ Se dan diferencias divididas con puntos coincidentes, ya que se conoce más de un dato por cada punto: $f\left[x\overset{k+1}{\ldots}x\right]$

✏ Para construir el polinomio se crean unos nodos ficticios duplicados:

$$z_{2i} = z_{2i+1} = x_i$$

A.4.2.1 Expresión de Newton

$$\begin{aligned} P_{2n+1}(x) &= f[x_0] + f[x_0, x_0](x - x_0) \\ &+ f[x_0, x_0, x_1](x - x_0)^2 \\ &+ f[x_0, x_0, x_1, x_1](x - x_0)^2 (x - x_1) \\ &+ \dots \\ &+ f[x_0, x_0, x_1, x_1, \dots, x_n, x_n](x - x_0)^2 \cdots (x - x_{n-1})^2 (x - x_n) \end{aligned}$$

A.4.2.2 Expresión de Lagrange

$$P_{2n+1}(x) = \sum_{i=0}^{n} f(x_i) H_i(x) + \sum_{i=0}^{n} f'(x_i) \hat{H}_i(x)$$

Con H_i y $\hat{H}_i$ a determinar:

$$H_j(x) = \left(1 - 2(x - x_j) L'_{n,j}(x_j)\right) L^2_{n,j}(x)$$

$$\hat{H}_j(x) = (x - x_j) L^2_{n,j}(x)$$

A.5 Tema 5: Derivación e integración numéricas

A.5.1 Derivación

✏ Pretendemos aproximar la derivada de orden k de una función en un punto α con una fórmula lineal

$$f^{(k)}(\alpha) \simeq \sum_{i=0}^{n} A_i f(x_i)$$

✏ **Método de los coeficientes indeterminados**: consiste en hacer que la fórmula lineal sea exacta para los polinomios x^j siendo $j = 0 : l$, con l el grado más alto posible, obteniéndose un sistema lineal para calcular los coeficientes A_i

$$D^k x^j = \sum_{i=0}^{n} A_i x_i^j \qquad j = 0, 1, 2, \dots$$

✏ Una fórmula de derivación numérica es de **grado de exactitud** q si q es el menor entero positivo para el que la formula es exacta para todo polinomio de grado no mayor que q y no exacta para algún polinomio de grado $q + 1$

✏ **Fórmulas de derivación numérica**:

$$f'(\alpha) = \frac{f(\alpha+h)-f(\alpha)}{h} - \frac{h}{2}f^{(2)}(c)$$

$$f'(\alpha) = \frac{-3f(\alpha)+4f(\alpha+h)-f(\alpha+2h)}{2h} + \frac{h^2}{3}f^{(3)}(c) \quad \text{Fórmula progresiva 3 puntos}$$

$$f'(\alpha) = \frac{f(\alpha-2h)-4f(\alpha-h)+3f(\alpha)}{2h} + \frac{h^2}{3}f^{(3)}(c) \quad \text{Fórmula regresiva 3 puntos}$$

$$f'(\alpha) = \frac{-f(\alpha-h)+f(\alpha+h)}{2h} - \frac{h^2}{6}f^{(3)}(c) \quad \text{Fórmula centrada}$$

✏ Las fórmulas de derivación son inestables

✏ **Error de discretización** para una fórmula obtenida con condiciones de exactitud

$$E_d = \frac{f^{(q+1)}(\xi)}{(q+1)!}C$$

A.5.2 Integración

✏ Pretendemos ahora aproximar la integral de una función mediante una función lineal de la forma

$$\int_a^b f(x)\,dx \simeq \sum_{i=0}^{n} A_i f(x_i)$$

✏ **Método de los coeficientes indeterminados**:

$$\int_a^b w(x)\,x^i dx = \sum_{k=0}^{n} A_k x_k^i \qquad i = 0, 1, 2, \ldots$$

✏ **Error de discretización**:

$$E_d = \frac{f^{(q+1)}(\xi(x))}{(q+1)!}C$$

✏ **Fórmulas de Newton-Côtes**:

$$\int_{x_0}^{x_1} f(x)\,dx = \frac{h}{2}(f(x_0)+f(x_1)) - \frac{h^3}{12}f^{(2)}(c) \quad \text{Trapecios}$$

$$\int_{x_0}^{x_2} f(x)\,dx = \frac{h}{3}(f(x_0)+4f(x_1)+f(x_2)) - \frac{h^5}{90}f^{(4)}(c) \quad \text{Simpson}$$

$$\int_{x_0}^{x_3} f(x)\,dx = \frac{3h}{8}(f(x_0)+3f(x_1)+3f(x_2)+f(x_3)) - \frac{3h^5}{80}f^{(4)}(c) \quad \text{Simpson } 3/8$$

$$\int_{x_{-1}}^{x_1} f(x)\,dx = 2hf(x_0) + \frac{h^3}{3}f^{(2)}(c) \quad \text{Punto medio}$$

✏ Las fórmulas de integración son estables

✏ **Fórmulas de Gauss**: fórmulas de cuadratura superexactas, grado de exactitud máximo $2n+1$. Requiere una elección de los puntos x_i y de los coeficientes A_i especial. Procedimiento:

1. Dado el polinomio
$$\theta_{n+1}(x) = x^{n+1} + a_1 x^n + \ldots + a_{n+1}$$
2. Se monta el siguiente sistema de ecuaciones lineales para calcular los coeficientes, con $j = 0 : n$
$$a_1 \int_a^b w(x) x^n x^j dx + a_2 \int_a^b w(x) x^{n-1} x^j dx + \ldots + a_{n+1} \int_a^b w(x) x^j dx = -\int_a^b w(x) x^{n+1} x^j dx$$
3. Se resuelve el sistema determinando el polinomio $\theta_{n+1}(x)$, ortogonal a todos los x^j
4. Se obtienen los x_i como raíces de $\theta_{n+1}(x) = 0$
5. Se calculan los A_i por el método de los coeficientes indeterminados (por ejemplo), para obtener la fórmula interpolatoria

A.5.3 Extrapolación de Richardson

✏ Aplicable a métodos de derivación e integración

$$F_{k+1}(h) = \frac{2^p F_k(h/2) - F_k(h)}{2^p - 1} \qquad k = 1 : m$$

siendo $O(h^p)$ el orden del método $F_k(h)$

A.6 Tema 6: Métodos numéricos para ecuaciones diferenciales ordinarias

A.6.1 Métodos unipaso

✏ Son de la forma
$$y_{i+1} = y_i + h \sum_{r=1}^{q} c_r k_r = y_i + h\phi(t, y, h)$$
con
$$k_1 = f(t_i, y_i) \qquad k_r = f\left(t_i + a_r h, y_i + h \sum_{s=1}^{r-1} b_{rs} k_s\right) \qquad r = 2 : q$$
escogiéndose a_r, c_r y b_{rs} de forma que el método sea consistente del mayor orden posible

✏ Son métodos conocidos como Runge-Kutta, notados como RK_{pq} siendo p el orden máximo y q el número de evaluaciones de la función necesarias. Para $q < 5$, $p = q$, para $q = 5, 6, 7$ es $p = q - 1$ y para $q \geq 8$ es $p = q - 2$

✏ Para que el método sea consistente de orden $p \geq 1$ (válido para los Runge-Kutta)
$$\sum_{r=1}^{q} c_r = 1$$

✏ En general, el método es consistente si
$$\lim_{h \to 0} \phi(t, y, h) = f(t, y)$$

A.6.2 Métodos multipaso

✏ Necesitan m evaluaciones previas de la función, que se obtendrán con métodos unipaso (preparación del principio de tabla)

✏ Son de la forma

$$y_{i+1} = \sum_{r=1}^{m} a_r y_{i+1-r} + h \sum_{r=0}^{m} b_r f\left(t_{i+1-r}, y_{i+1-r}\right)$$

Si $b_0 = 0$ se llama explícito, mientras que si $b_0 \neq 0$ implícito (predictor-corrector)

✏ La **consistencia** del método viene dada por

$$1 - \sum_{r=1}^{m} a_r = 0 \qquad \text{condición necesaria}$$

$$m - \sum_{r=1}^{m} (m-r)\, a_r = \sum_{r=0}^{m} b_r \qquad \text{condición suficiente}$$

✏ La consistencia en función de los polinomios característicos se puede expresar como

$$\begin{array}{ll} \rho(1) = 0 & \rho(\lambda) = \lambda^m - a_1\lambda^{m-1} - a_2\lambda^{m-2} - \ldots - a_m \\ \text{para} & \\ \rho'(1) = \sigma(1) & \sigma(\lambda) = b_0\lambda^m + b_1\lambda^{m-1} + \ldots + b_m \end{array}$$

✏ El **método es de orden** p si

$$m^{k+1} - \sum_{r=1}^{m} a_r (m-r)^{k+1} = (k+1) \sum_{r=0}^{m} b_r (m-r)^k \qquad k = 1:p$$

A.7 Tema 7: Resolución de ecuaciones no lineales

A.7.1 Métodos bracketing

✏ En estos métodos se construye una sucesión de intervalos de forma que cada intervalo esté contenido en el anterior y que haya al menos una raíz en ellos

✏ **Método de bipartición**: si f es continua en $[a,b]$ y $f(a) f(b) < 0$, existe un punto $h \in (a,b)$ tal que $f(h) = 0$, por lo que se toma $c = (a+b)/2$ y se continúa con el intervalo $[a,c]$ o $[c,b]$, según sea $f(a) f(c) < 0$ o $f(c) f(b) < 0$ respectivamente

✏ **Método de la regula falsi**: mismo procedimiento que bipartición, pero

$$c = \frac{b f(a) - a f(b)}{f(a) - f(b)}$$

A.7.2 Métodos de aproximaciones sucesivas

✏ Punto fijo

$$x_{s+1} = g(x_s)$$

✏ **Método de Newton**:

$$x_{s+1} = x_s - \frac{f(x_s)}{f'(x_s)}$$

✏ **Método de la secante**:

$$x_{s+1} = x_s - \frac{f(x_s)}{f(x_s) - f(x_{s-1})}(x_s - x_{s-1})$$

✏ **Método de Steffensen**:

$$x_{s+1} = x_s - \frac{(f(x_s))^2}{f(x_s + f(x_s)) - f(x_s)}$$

✏ **Convergencia**: para que un método de punto fijo sea convergente (y lo será para cualquier punto inicial $x_0 \in [a,b]$), debe cumplir

1. $g([a,b]) \subseteq [a,b]$
2. g derivable en $[a,b]$ y $|g'(x)| \leq K < 1$ para todo $x \in [a,b]$

✏ Se verifica en caso de convergencia la siguiente estimación de error

$$|x_{s+1} - x^*| \leq \frac{K^{s-r}}{1-K}|x_{r+1} - x_r| \qquad r = 1:s$$

✏ Determinar número de iteraciones s para t decimales exactos:

$$|x_s - x^*| \leq \frac{K^s}{1-K}|x_1 - x_0| < \frac{1}{2}10^{-t}$$

✏ Número de iteraciones s para t dígitos significativos:

$$\frac{|x_s - x^*|}{|x^*|} \leq \frac{K^s}{1-K}\frac{|x_1 - x_0|}{|x^*|} < \frac{1}{2}10^{1-t}$$

A.7.3 Ecuaciones algebraicas

✏ Si $z \in \mathbb{C}$ es raiz de $p(x) = 0$, se cumple que

☞ En una circunferencia de radio r están contenidas todas las raíces complejas de p:

$$|z| \leq 1 + \frac{a_{max}}{|a_0|} = r$$

donde $a_{max} = \max\limits_{1 \leq i \leq n} |a_i|$

☞ Si ninguna raiz es nula ($a_n \neq 0$), todas las raíces están fuera del círculo

$$|z| \geq \frac{1}{1 + \frac{a^*_{max}}{|a_n|}}$$

donde $a^*_{max} = \max\limits_{0 \leq i \leq n-1} |a_i|$

✏ **Regla de Laguerre-Thibault**:

$$p(x) = (x - L)\left(b_0 x^{n-1} + b_1 x^{n-2} + \ldots + b_{n-1}\right) + p(L)$$

Sea $L > 0/b_i \geq 0\ \forall i$ y $p(L) > 0$. Entonces si $x > L$ y $p(x) > 0$, L es **cota superior de las raíces reales positivas**.
Se pueden obtener cotas inferiores para las raíces negativas haciendo $x \rightsquigarrow -x$ y las cotas inferiores para las positivas con $x \rightsquigarrow 1/x$.
Otra forma de expresar esta regla es: dado $L > 0$, si los coeficientes del polinomio

$$\frac{p(x)}{x - L}$$

son positivos y $p(L) \geq 0$, L es cota superior de las raíces positivas de la ecuación $p(x) = 0$

✏ **Regla de Descartes**:

$$N_+ = N_s - \dot{2}$$

donde N_+ es el número de raíces reales positivas, N_s el número de cambios de signo de la sucesión $a_0, a_1, \ldots, a_n$

A.8 Tema 8: Métodos numéricos de optimización

A.8.1 Optimización no restringida: métodos iterativos de descenso

✏ Se dice que un punto $x^* \in \mathbb{R}^n$ es **mínimo local** si cumple

$$\exists \varepsilon > 0 / f(x^*) \leq f(x) \qquad \forall x \in B(x^*, \varepsilon) = \{x / \|x - x^*\| \leq \varepsilon\}$$

estricto si además

$$\exists \varepsilon > 0 / f(x^*) < f(x) \qquad \forall x \in B(x^*, \varepsilon) - \{x^*\}$$

✏ Se llama **puntos críticos o estacionarios** a los que cumplen la condición de primer orden

$$\nabla f(x) = 0$$

✏ **Puntos de silla** son los puntos críticos que no son máximos ni mínimos

☞ **Condiciones necesarias de mínimo local**:

$$\nabla f(x^*) = 0$$

$$\nabla^2 f(x^*) \text{ semidefinida positiva}$$

☞ **Condiciones suficientes de mínimo local estricto**:

$$\nabla f(x^*) = 0$$

$$\nabla^2 f(x^*) \text{ definida positiva}$$

☞ En los puntos de silla, $\nabla^2 f(x)$ es indefinida.

✏ **Método de descenso**:

$$f\left(x^{(s+1)}\right) < f\left(x^{(s)}\right) \Rightarrow x^{(s+1)} = x^{(s)} + \alpha_s d^{(s)} \text{ siendo } {d^{(s)}}^t \nabla f\left(x^{(s)}\right) < 0$$

☞ **búsqueda en línea exacta**:

$$\alpha_s = -\frac{d^{(s)^t} \nabla f\left(x^{(s)}\right)}{d^{(s)^t} \nabla^2 f\left(x^{(s)}\right) d^{(s)}}$$

☞ **Método del gradiente**:

$$d^{(s)} = -\nabla f\left(x^{(s)}\right)$$

☞ **Método de Newton**:

$$d^{(s)} = -\left(\nabla^2 f\left(x^{(s)}\right)\right)^{-1} \nabla f\left(x^{(s)}\right) \qquad \alpha_s = 1 \text{ o con línea aproximada}$$

Convergencia cuadrática para $x^{(0)}$ suficientemente cerca de la solución

A.8.2 Programación no lineal

✏ Se minimiza con restricciones

$$h_i(x) = 0 \quad i = 1:t$$

$$g_i(x) \geq 0 \quad i = 1:m$$

✏ Llamamos $\mathcal{F}$ al conjunto de puntos que cumplen las restricciones. A todo punto $x \in \mathcal{F}$ se le llama **punto factible**, por lo que a $\mathcal{F}$ se le llama **región factible**.

✏ Definimos la **función Lagrangiana** asociada al problema de programación no lineal como

$$\boxed{L(x,\lambda) = f(x) - \sum_{i=1}^{t} \lambda_i h_i(x) - \sum_{i=1}^{m} \lambda_{i+i} g_i(x)}$$

siendo

$$\lambda = \begin{pmatrix} \lambda_1 & \lambda_2 & \dots & \lambda_{t+m} \end{pmatrix}^t$$

el vector de **multiplicadores de Lagrange**.

✏ Definición de **mínimo local restringido**

$$\boxed{f(x^*) \leq f(x) \qquad \forall x \in B(x^*, \varepsilon) + \mathcal{F}}$$

✏ **Restricción activa** en un punto: una restricción $h_i(x)$ (de igualdad) o $g_i(x)$ en un punto $x \in \mathcal{F}$ es activa si

$$h_i(x) = 0 \qquad \text{o} \qquad g_i(x) = 0$$

En el problema de programación no lineal se buscan sólo los puntos activos que minimizan la función, no los puntos de la región factible que lo hagan

✏ Definimos el **arco de curva diferenciable y factible** $x(\xi) \in \mathcal{F}$ como una aplicación $x(\xi) : \mathbb{R} \to \mathbb{R}^n$.

✏ **Condiciones necesarias de mínimo local restringido**:

$$\boxed{\begin{array}{l} p^t \nabla f(x^*) = 0 \\ \\ Z_A^t \left(\nabla^2 f(x^*) - \sum_i \lambda_i \nabla^2 h_i(x^*) - \sum_i \lambda_{t+i} \nabla^2 g_i(x^*)\right) Z_A \geq 0 \end{array}}$$

✏ Llamamos **direcciones factibles** a

$$p = x'(0)$$

✏ **Condición de optimalidad de primer orden**

$$\boxed{\nabla f(x^*) = \sum_i \lambda_i \nabla g_i(x) \qquad \lambda_i \geq 0}$$

Si algún λ_i es negativo, entonces no es mínimo local restringido

✏ **Condición de optimalidad de segundo orden**:

$$\boxed{Z_A^t \left(\nabla^2 f(x^*) - \sum_i \lambda_i \nabla^2 h_i(x^*) - \sum_i \lambda_{t+i} \nabla^2 g_i(x^*) \right) Z_A \geq 0}$$

Esto lo hemos hecho suponiendo que A_A tiene rango completo por columnas, así serán válidas las condiciones. Ahora vamos a resumir las condiciones.

✏ **Condiciones suficientes de mínimo local restringido**:

$$\boxed{\begin{array}{l} p^t \nabla f(x^*) = 0 \\ \\ Z_A^t \left(\nabla^2 f(x^*) - \sum_i \lambda_i \nabla^2 h_i(x^*) - \sum_i \lambda_{t+i} \nabla^2 g_i(x^*) \right) Z_A > 0 \end{array}}$$

✏ Un par (x^*, λ) es un **par Kuhn-Tucker** para el problema de programación no lineal dado si $x^* \in \mathcal{F}$ y además se satisface

$$\nabla_x L(x^*, \lambda) = 0 \qquad \lambda_{t+i} \geq 0 \text{ para } i = 1:m \qquad \lambda_{t+i} g_i(x^*) = 0 \text{ para } i = 1:m$$

Bibliografía

[1] Burden, R.L.; Faires, J.D. *Análisis Numérico*. Grupo Editorial Iberoamérica, 1985

[2] Burden, R.L.; Faires, J.D. *Numerical Analysis*. Pws-Kent Publishing Company, 1993

[3] Ángel Santos, *Transparencias de Métodos Numéricos*, Curso 2002/2003

[4] Kincaid, D.; Cheney, W. *Análisis Numérico*. Addison Wesley Iberoamericana, 1994