1 Vectores y Funciones

En esta primera Práctica, aprenderemos a utilizar las órdenes básicas de MATLAB para trabajar con escalares, vectores y matrices, evaluar funciones de una y dos variables y representarlas gráficamente. En prácticas posteriores usaremos habitualmente MATLAB para efectuar los cálculos.

MATLAB (MATrix LABoratory) es un programa orientado al cálculo con matrices, al que se reducen muchos de los algoritmos que resuelven problemas de Matemática Aplicada e Ingeniería.

MATLAB ofrece un entorno interactivo sencillo mediante una ventana (que llamaremos ventana de comandos) en la que podemos introducir ordenes en modo texto y en la que aparecen los resultados. Los gráficos se muestran en ventanas independientes. Cada ventana dispone de una barra de menús que controla su funcionalidad.

Aprenderemos a asignar, borrar, guardar y recuperar *variables*, utilizar las *funciones* incorporadas y, más adelante, a definir funciones nuevas.

En MATLAB todas las instrucciones tienen que estar escritas en minúsculas, de esa forma nos evitaremos errores de ejecución.

MATLAB opera directamente con números *complejos* y con números reales como caso particular.

Lo que distingue a MATLAB de otros sistemas de cálculo es su facilidad para trabajar con *vectores* y *matrices*. Las operaciones ordinarias, suma, producto, potencia, operan por defecto sobre matrices, sin más restricción que la compatibilidad de tamaños en cada caso.

1.1 Mandatos Básicos

1.1.1 Help, Dir, Pwd.

`help` nos da una lista de temas sobre los que hay información de ayuda. `helpwin` abre una ventana de ayuda que es útil para consultar información sobre órdenes de MATLAB sin interferir con la ventana principal.

`help` *`tema`* explica concisamente el tema elegido y añade información sobre temas relacionados.

Ejemplo 1

La instrucción `clc` borra la ventana de comandos, esa información la obtendrás al ejecutar: `help clc`.

Utiliza la ayuda que proporciona *help* siempre que no recuerdes bien el funcionamiento de alguna instrucción de MATLAB.

La instrucción `pwd` nos `informa del directorio actual de trabajo y` el mandato `dir` lista los ficheros de dicho directorio, también tenemos `what` que solo muestra los ficheros de función (es decir, los que tienen extensión '.m', que más adelante comentaremos).

1.1.2 Path, Diary.

Para cambiar de directorio tenemos que utilizar la instrucción `path`, que asigna el camino (directorios o subdirectorios) donde el programa buscará los archivos.

Ejemplo 2

`path` nos informa del orden de los subdirectorios donde MATLAB busca los archivos.

`path(path,'a:')` asigna el disco a: como primer subdirectorio de búsqueda, siempre y cuando nos hallamos asegurado de haber insertado un disco en la disquetera.

Otra forma más visual de utilizar la instrucción, es en la barra de menús, en el menú File/Archivo la instrucción set path abre una ventana para cambiar el `path`, y también podemos ejecutar el mandato `editpath` que hace lo mismo directamente.

Uno de los mandatos más útiles de MATLAB es `diary`, que permite guardar en un fichero todo el texto que aparece en la ventana de comandos. Escribe

```
diary a:practi01.txt
```

y todo lo que salga en pantalla se grabará en un fichero **practi01.txt** justo cuando vuelvas a introducir el mandato `diary`. Para añadir más texto en una misma sesión al diario creado usa `diary on` al principio de lo que quieras grabar y `diary off` al final (en este momento se graba realmente).

Cuando utilices esta instrucción espera a que el LED de la disquetera se apague, para dejar tiempo al programa a que guarde los datos en el disco. Ten en cuenta que esta instrucción es del tipo interruptor y solo graba cuando se ejecuta la opción off. Una mala ejecución puede hacerte perder la información, además como puede ocurrir que el ordenador falle alguna vez es aconsejable ir grabando la información cada cierto tiempo, acordándote de volver a ejecutar el diary para que se vaya almacenando la siguiente información.

La memoria de pantalla en MATLAB no es muy grande, y por tanto hay datos que van desapareciendo de la ventana, pero estos no se pierden si estas utilizando el diary.

El mandato `%` que convierte en comentario lo que se escribe a continuación.

```
% Esto es un comentario
```

Con la tecla [↑] recuperas los mandatos antes escritos, evitando así tener que reescribir órdenes iguales o parecidas. También puedes 'copiar' con el ratón texto de cualquier sitio y 'pegar' en la (única) línea de mandatos activa, eligiendo estas opciones en el menú de edición. Vale usar [Ctrl]+C y [Ctrl]+V con el mismo fin.

1.2 Variables

En MATLAB las variables se asignan de modo natural. Basta escribir un nombre de variable, a continuación el signo igual y luego el valor que toma esa variable. Para aceptar, como siempre, hay que pulsar [Intro]. Escribiendo sólo el nombre de una variable previamente asignada, MATLAB devuelve su valor.
Los signos +, –, *, / y ^ denotan las operaciones aritméticas de suma, resta, multiplicación, división y elevación a una potencia (de modo que resultan válidas para matrices, como veremos más adelante). Si el resultado de una operación no es asignado a ninguna variable, MATLAB lo asigna a la variable del sistema `ans`.

Al poner 'punto y coma', no se muestra el resultado por pantalla. Naturalmente, la asignación de la variable no resulta afectada.

1.2.1 Who, Whos.

La orden `who` lista las variables definidas y con la orden `whos` obtenemos además el tipo de variable y su tamaño.

Ejemplo 3

```
a = 3, b = 4, a
a + b
c = ans;
who
whos
```

1.2.2 Clear, Save, Load, Workspace.

`clear` sin argumentos elimina todas las variables. La misma instrucción seguida de nombres de variables, elimina a estas.

La instrucción `save`, guarda las variables definidas en el archivo que indiquemos para poderlas utilizar en otra sesión y lo hace con la extensión '.mat', lo mismo podemos hacer desde el Menú File con la instrucción save workspace.

Para recuperar del disco un espacio de trabajo donde tengamos almacenados datos se utiliza la instrucción `load` junto con el nombre del fichero, también desde el Menú File con la instrucción load workspace o import data según versiones.

La instrucción show workspace del Menú File, o la instrucción `workspace` muestran todas las variables de trabajo creadas en una ventana.

1.2.3 Variables especiales, format.

MATLAB utiliza ciertos nombres de variable para fines especiales, como `i` o `j`, que designan ambas a la unidad imaginaria ($\mathbf{i}^2 = \mathbf{j}^2 = -\mathbf{1}$) o `pi`, para el número π. El número **e**, base de los logaritmos neperianos, no está preasignado, pero se obtiene fácilmente como `exp(1)`.

La *precisión relativa* en operaciones de coma flotante se llama **eps**. El resultado de 1/0 en MATLAB es **Inf** y el de 0/0, **NaN**.

```
1/0          % Infinito
0/0          % Indeterminado
```

Podemos utilizar estos nombres de variable para almacenar otros valores, prevaleciendo nuestra asignación sobre el valor por defecto de MATLAB. Por ejemplo, si no utilizamos números complejos, no hay inconveniente en representar por **i** y **j** los índices de fila y columna de una matriz. Igualmente podríamos llamar **eps** a una cantidad a utilizar como criterio de convergencia, pero en general conviene evitar equívocos empleando otros nombres de variable.

Internamente MATLAB trabaja con mucha precisión, aunque por defecto muestra los resultados con cuatro decimales. La *apariencia de los resultados* se modifica por menú o con la orden `format`. `format long` aumenta el número de decimales visibles.

`format short` vuelve al estado inicial. `format rat` aproxima el resultado por un cociente de enteros pequeños. Explora otras opciones con `help format` .

```
pi,format long, pi
format rat, pi
```

1.2.4 Cadenas De Caracteres.

Podemos usar también *cadenas de caracteres* para manejar texto en funciones de MATLAB. Para introducir una cadena, basta escribir el texto entre comillas.

Un texto sin comillas produce error porque MATLAB lo interpreta como un nombre de variable o función.

El mandato `ischar` nos dice si una expresión es o no un carácter (responde 1 si es verdadero y 0 si es falso).

Ejemplo 4

```
a='Esto es una cadena'
b=Esto no
c=3
ischar(a)
ischar(c)
```

1.3 Vectores

1.3.1 Edición de Vectores

Los vectores se utilizan, entre otras cosas, para representar:

- Puntos del plano y del espacio.
- Puntos de un espacio n-dimensional.
- Magnitudes físicas.
- Filas o columnas de una matriz (recuerda la discusión de sistemas de ecuaciones lineales).

Para introducir un vector en MATLAB, escribimos sus componentes entre corchetes. Separando las componentes con comas o espacios obtenemos un vector fila. Separándolas por punto y coma o por [Intro], obtenemos un vector columna.

Ejemplo 5

```
u = [1 2 3], v = [1,2,3]     % Vectores fila
w = [1;2;3]
» z=[1
» 2
» 3]            % Vectores columna
```

Para calcular la longitud de un vector se utiliza el mandato `length`, ahora bien como MATLAB trabaja siempre todas las variables como matrices, tanto si son matrices como si son escalares como si son vectores, para obtener la dimensión de cualquier variable podemos utilizar la función `size` que devuelve un vector de dos componentes que son el número de filas y el número de columnas de la matriz.

Ejemplo 6

```
length(u)
length(w)
[f,c]=size(u)
dimension=size(w)
```

1.3.2 Vectores Progresivos.

Es muy frecuente tener que editar vectores con componentes equiespaciadas, por ejemplo, para crear una tabla de valores de una función.

Con `a:h:b` creamos un vector de componentes que van de **a** hasta **b** y distan **h** cada una de la siguiente.

La orden `linspace(a,b,n)` crea **n** términos en progresión aritmética, desde **a** hasta **b**.

Ejemplo 7

```
x=0:0.1:1
y=linspace(0,1,11)
```

1.3.3 Suma y producto por un escalar.

La *suma* de dos vectores del mismo tamaño se efectúa componente a componente y se obtiene con MATLAB escribiendo directamente

```
» u + v
```

La orden `sum(u)` proporciona la suma de todas las componentes del vector **u**.

Para *multiplicar* un vector **u** *por un escalar* **a**, basta escribir

```
» a*u
```

En ocasiones hay que multiplicar dos vectores *elemento a elemento* y eso MATLAB lo hace con la versión 'punto' del producto.

```
u.*v        % Producto elemento a elemento
```

Si intentamos multiplicar por las buenas dos vectores del mismo tamaño, nos da un error, pues MATLAB aplica el *producto matricial* y los tamaños no son coherentes.

El producto matricial requiere que el primer factor tenga tantas columnas como filas tiene el segundo. Por ejemplo, podemos multiplicar una fila por una columna del mismo número de elementos o viceversa.

```
» u*w       % Fila × Columna = Escalar
» w*u       % Columna × Fila = Matriz de rango 1
```

Finalmente, el *producto* de todas las *componentes* de un vector se obtiene con la función `prod`.

1.3.4 Producto escalar y vectorial de dos vectores.

El *producto escalar* de dos vectores de la misma dimensión se efectúa con `dot` y el producto vectorial de dos vectores de longitud 3 con `cross`.

Ejemplo 8

```
a = [1 2 3];
b = [4 5 6];
c = dot(a,b)
d = cross(a,b)
e = prod(a)
f = prod(b)
```

1.3.5 ', .', flipud, fliplr

La transpuesta (conjugada) de un vector (complejo) **v** es **v',** su instrucción equivalente es `ctranspose`, la transpuesta (no conjugada) de un vector (complejo) **v** es **v.',** su instrucción equivalente es `transpose`.

Mediante `fliplr` volteamos un vector fila de izquierda a derecha y con `flipud` ponemos cabeza abajo un vector columna.

Ejemplo 9

```
z=[1 i 2-i];
v=z'
w=z.'
```

1.3.6 Diferencias, sumas y productos acumulados

La instrucción `diff` aplicada a un vector X=[X(1), X(2), ..., X(n)], realiza la diferencia entre sus componentes de la siguiente forma [X(2)-X(1), X(3)-X(2), ..., X(n)-X(n-1)].

Las instrucciones `cumsum` y `cumprod` efectuan respectivamente las suma y los productos acumulados de un vector, son interesantes para el estudio de sumatorios y productorios finitos.

Ejemplo 10

```
a = 1:6
b = cumsum(a)
c = cumprod(a)
```

1.4 Matrices

1.4.1 Edición de Matrices

Por defecto, MATLAB trabaja con matrices. Esto supone la ventaja substancial de no tener que declarar tipos de variable ni tamaños de fila o columnas para trabajar tanto con matrices de números reales o complejos como con vectores o escalares, que se consideran casos particulares de matrices.

Las matrices se escriben por filas. Los elementos de una fila se separan por 'comas' y las distintas filas por 'puntos y comas'.

```
A = [1,2;3,4]
```

Lo mismo que con vectores, podemos también separar los elementos de una fila con espacios y las filas pulsando la tecla [Intro].

```
» B = [-1 -2
 -3 -4]
```

El elemento en la fila **i** y la columna **j** de la matriz **A** se denota por **A(i,j)**. Modifica, por ejemplo, el elemento 2,1 de A:

```
» A(2,1) = 0
```

A(i,:) denota la fila **i** de la matriz **A**. Análogamente, **A(:,j)** es la columna **j** de **A**.

```
» A(2,:), A(:,1)
```

En ocasiones resulta cómodo construir una matriz a partir de bloques. Con tal de que sus tamaños sean coherentes, basta escribir los bloques por filas, como si se tratase de elementos individuales.

```
» M = [A,B;B,A]
```

Para extraer una submatriz, indicaremos las filas y columnas de que se compone.

```
» M41 = M(1:3,2:4)
```

Las filas o columnas no tienen por que ser consecutivas

```
» fil = [1,2,4],  col = [1,3,4]
» M32 = M(fil,col)
```

1.4.2 Matrices usuales

Ya hemos visto cómo se escriben las matrices. MATLAB tiene varias funciones que facilitan la edición de matrices de uso frecuente:

- `eye(n)` proporciona la matriz identidad de orden **n**.
- `zeros(n,m)` inicializa una matriz **m** por **n** con todos los elementos nulos.
- `ones` hace lo mismo con elementos de valor 1.
- `rand` crea matrices con elementos aleatorios uniformemente distribuidos en el intervalo [0,1].

Ejemplo 11

```
M = eye(4)
N = zeros(3)
O = ones(2)
P = rand(3,2)
```

1.4.3 Operaciones con Matrices

Para sumar dos matrices del mismo tamaño, se suma cada elemento de una con el elemento correspondiente de la otra.

```
» A + B
```

El producto de matrices se hace multiplicando fila por columna y sumando los resultados. Comprueba que el elemento (1,1) de la matriz producto **A*B** es

```
» A(1,1)*B(1,1)+A(1,2)*B(2,1)
```

Observa que el producto de matrices NO es conmutativo

```
» A*B - B*A
```

MATLAB interpreta **A/B** como el producto de **A** por la inversa de **B**. Prueba:

```
» A/B, A*inv(B), A*inv(A)
```

La no conmutatividad del producto justifica que **inv(A)*B** se abrevie a **A\B**. Asimismo, la solución del sistema **Ax=b**, que formalmente es $\mathbf{x}=\mathbf{A}^{-1}\mathbf{b}$, se obtiene en MATLAB con **A\b**. Bien entendido que la solución no se obtiene calculando la inversa de **A**, sino aplicando métodos numéricamente más eficientes (ver `help slash`).

Para multiplicar dos matrices elemento a elemento, en lugar de filas por columnas, usamos las variantes 'punto' de las operaciones correspondientes. Comprueba la diferencia entre

```
» A*B,  A.*B,  A^-1,  A.^-1
```

Si **A** es una matriz real, **A'** es la *transpuesta* de **A**. En el caso complejo, **A'** es la transpuesta conjugada. La transpuesta sin conjugar se obtiene con **A.'**.

1.5 Funciones

MATLAB conoce las funciones matemáticas elementales:

- Trigonométricas: seno, coseno, tangente, cosecante, secante y cotangente.
- Trigonométricas inversas: arco seno, arco coseno, arco tangente, ...
- Exponencial y logaritmos neperiano, decimal y en base 2.
- Hiperbólicas: seno hiperbólico, coseno hiperbólico, tangente hiperbólica, ...
- Hiperbólicas inversas: argumento seno hiperbólico, coseno, tangente, ...
- Raíz cuadrada, parte entera, valor absoluto, ...

1.5.1 sqrt, abs.

La función `sqrt` realiza la raíz cuadrada de un número, si este es negativo da como resultado un complejo.

La orden `abs` calcula el valor absoluto de un número, si este es complejo devuelve su módulo.

Ejemplo 12

```
sqrt(4),sqrt(-4)
abs(4),abs(-4),abs(3+4*i)
```

1.5.2 exp,log,log10

Los mandatos `exp`, `log` y `log10` realizan respectivamente la exponencial, el logaritmo neperiano y el logaritmo decimal de un número.

Ejemplo 13

```
exp(1)
log(ans)
log10(10)
```

1.5.3 sin, cos, tan, atan, atan2

Las funciones trigonométricas tienen el argumento en radianes, aparte de estar el nombre en inglés.

Ejemplo 14

```
sin(pi/2)

sin(90)
```

Cuidado, en este ejemplo MATLAB dará un resultado para ambas ordenes, pero solo la primera es correcta, pues su argumento de entrada es correcto, ya que el ángulo esta en radianes.

La función atan devuelve la arcotangente de un número, sin embargo la función atan2 que en este caso tiene dos argumentos de entrada devuelve la arcotangente teniendo en cuenta el cuadrante.

Ejemplo 15

Si intentamos determinar el ángulo de un punto (x,y) en el plano este viene dado por arcotangente(y/x), teniendo en cuenta el cuadrante para dar la respuesta correcta,

Los puntos (1,1) y (-1,-1) se obtendrian con el mismo valor de la arcotangente, que si expresamos el resultado en grados seria

```
atan(1)*180/pi

ans =    45
```

luego considerando el cuadrante obtendriamos que los ángulos son respectivamente 45 y 225 (o –135) grados. Ahora bien si utilizamos la instrucción atan2(y,x) el resultado se obtiene directamente

```
atan2(1,1)*180/pi,atan2(-1,-1)*180/pi

ans =    45 ans =  -135
```

1.5.4 sinh, cosh, tanh

Estas órdenes representan a las funciones hiperbólicas, recuerda que para números reales se definen como:

$$sinh(x) = \frac{\exp(x) - \exp(-x)}{2}$$

$$\cosh(x) = \frac{epx(x) + \exp(-x)}{2}$$

Ejemplo 16

```
sinh(1),(exp(1)-exp(-1))/2
cosh(1),(exp(1)+exp(-1))/2
tanh(1),sinh(1)/cosh(1)
```

Una característica destacable de MATLAB es que evalúa una función sobre todas las componentes de un vector simultáneamente, lo cual es muy práctico.

Ejemplo 17

```
x = -1:0.01:1;
y = tanh(x);
plot(x,y)
```

Aparte de estas funciones MATLAB tiene definidas otras tantas menos conocidas pero no menos útiles, tanto funciones elementales que puedes consultar su sintaxis con `help elfun` . Y otras funciones especiales disponibles con `help specfun` .

1.6 Gráficas

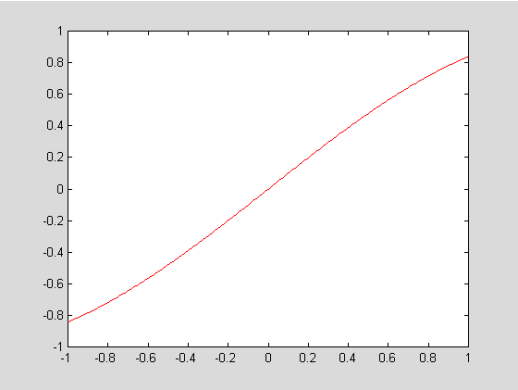
1.6.1 Tipos de linea, colores y marcadores.

En la siguiente tabla se muestran los tipos de color, marcadores y tipos de línea que admite la instrucción gráfica `plot`. Así como el código que los genera en el gráfico.

color	código	marcador	código	tipo de línea	código
Amarillo	y	punto	.	sólida	-
Magenta	m	circulo	o	punteada	:
Cyan	c	x-marca	x	punto y raya	-.
Rojo	r	más	+	rayada	--
Verde	g	estrella	*		
Azul	b	cuadrado	s		
Negro	k	diamante	d		
Blanco	w	triángulo (abajo)	v		
		triángulo (arriba)	^		
		triángulo (izquierda)	<		
		triángulo (derecha)	>		
		pentagrama	p		
		hexagrama	h		

Ejemplo 18

```
x = -1:0.01:1;
y = sin(x);
plot(x,y,'r')
```



1.7 Ficheros de función

1.7.1 Función y=f(x)

La programación en MATLAB de efectúa mediante *ficheros.m*. Son simplemente ficheros de texto que contienen órdenes de MATLAB. Su utilización requiere

- Editar el fichero con el editor de MATLAB o con un editor ASCII.
- Guardarlo con extensión .m.
- Indicar a MATLAB dónde está el archivo con: `path(path,'dirección').`
- Ejecutarlo escribiendo en la línea de órdenes el nombre de fichero y los parámetros de entrada necesarios.

Como ejemplo, creamos un fichero para evaluar la llamada función de Runge, que utilizaremos más adelante en la teoría de interpolaciones:

$$f(x) = \frac{1}{1 + 25 \cdot x^2}$$

Debemos escribir en el editor:

```
function y=runge(x)

% Función de Runge

y=1./(1+25.*x.^2);
```

Después de grabarlo con el nombre runge.m, e indicar el path a Matlab, podremos ejecutar llamadas a la función.

Nótese que se han utilizado las operaciones con el punto delante, esto es por si la entrada es un vector o matriz efectuará las operaciones indicadas para cada uno de los elementos.
A su vez hay que fijarse que el nombre asignado a la función es el mismo que el del archivo que se graba en el disco, también es aconsejable que este no contenga signos ni acentos y si es posible no supere 8 caracteres.

Ejemplo 19

```
runge(0)
runge([-1 0 1])
```

El fichero **runge.m** es un archivo de *función,* al incluir la palabra clave `function`. A continuación es obligatorio poner el *nombre de función.*

Notar que las primeras líneas de comentario que escribimos inmediatamente después del nombre de la función son las que se obtienen con el `help` de matlab

```
help runge
```

El nombre de función suele ir precedido de los *argumentos de salida*, entre corchetes, y seguido de los nombres de los *argumentos de entrada,* entre paréntesis. Entre estos y el nombre de función aparece el signo 'igual'.

Consideremos la función que da las coordenadas cartesianas de un punto, a partir de su radio vector y su ángulo respecto al eje OX:

$$\begin{cases} x = r \cdot \cos(\theta) \\ y = r \cdot sin(\theta) \end{cases}$$

Que llamaremos por ejemplo *cartpol*, en referencia al paso de polares a cartesianas, la definiríamos de la siguiente forma:

```
function [x,y]=cartpol(r,z)

% Entradas 'r':radio vector
% y 'z' ángulo respecto al eje OX
% Salidas 'x,y': coordenadas cartesinas

x=r.*cos(z);
y=r.*sin(z);
```

Ejemplo 20

```
ro=ones(1,3);
zeta=[pi/2 pi 3*pi/2];

[x,y]=cartpol(ro,zeta)
help cartpol
```

EJERCICIOS

1. Dados dos números $a < b$, obtén un vector x cuyas componentes sean los extremos de los intervalos obtenidos al dividir $[a,b]$ en n partes iguales. El vector x tiene $n + 1$ componentes a las que llamamos a $= x_0 < x_1 < x_2 < \ldots < x_n = b$. Elegir adecuadamente a, b y n para obtener representaciones gráficas de los valores en x de las siguientes funciones:

a) $f(x) = sen\ x$	b) $f(x) = cos\ x$	c) $f(x) = tg\ x$
d) $f(x) = sinh\ x$	e) $f(x) = cosh\ x$	f) $f(x) = tanh\ x$
g) $f(x) = log\ x$	h) $f(x) = exp\ x$	i) $f(x) = 1/x$
j) $f(x) = 1/x^2$	k) $f(x) = 1/(1+x^2)$	l) $f(x) = x/(1+x^2)$

2. Estudia de forma aproximada los siguientes límites de funciones sin resolverlos. Toma sucesiones de números que convergen al valor donde se evalúa el límite, evalúa estas sucesiones en la función para obtener su carácter convergente o divergente, y en el caso de convergencia obtener una aproximación al límite.

$$a)\ \lim_{x\to 0}\frac{e^x-1}{tan(2x)} \qquad b)\ \lim_{x\to 0}\frac{x-atan(x)}{x^3} \qquad c)\ \lim_{x\to 0}\frac{x\cos(x)-sin(x)}{x^3} \qquad d)\ \lim_{x\to 0}\frac{\log(sin(3x))}{\log(sin(2x))}$$

$$e)\ \lim_{x\to 0}\frac{e^x-sin(x)-1}{sin^2(x)} \qquad f)\ \lim_{x\to \frac{\pi}{2}}\frac{sin(x)-1}{2x-\pi} \qquad g)\ \lim_{x\to\infty}\frac{2^x}{x^5} \qquad h)\ \lim_{x\to 2}\frac{2^x-x^2}{x-2}$$

3. Define las siguientes matrices en MatLab:

$$A=\begin{pmatrix}2 & -1 & 1\\ 3 & 2 & 1\\ 5 & 1 & 4\end{pmatrix}\quad,\quad B=\begin{pmatrix}2 & 3 & 4\\ 4 & 3 & 2\\ 1 & 5 & -5\end{pmatrix}\quad,\quad C=\begin{pmatrix}\frac{1}{2} & \frac{1}{4} & \frac{1}{8}\\ \frac{1}{3} & \frac{2}{5} & 1\\ \frac{-1}{6} & 1 & 1\end{pmatrix}$$

y efectúa las siguientes operaciones:

a) Comprueba la no conmutatividad del producto de matrices calculando:
A*B y B*A.

b) Calcula C^3, utilizando las potencias de matrices y las potencias de matrices elemento a elemento.
c) Define la matriz identidad de orden 3 con la orden correspondiente y con ella calcula la inversa de A utilizando la división izquierda de matrices.
d) Calcula $(A*B)^{-1}$ de dos formas diferentes utilizando la teoría de matrices.

4. Utilizando las otras opciones de la instrucción `diff` de MatLab que puedes consultar con el Help, calcula las siguientes derivadas y derivadas parciales:

$a)\ \frac{d}{dt}\left(t^3 * sin^2(t)\right)$ $b)\ \frac{d^3}{dt^3}\left(\frac{1}{1+t^2}\right)$ $c)\ \frac{d^2}{dt^2}\left(tan\left(1/t\right)\right)$

$d)\ \frac{\partial}{\partial x}\left(x*sin(y)-y*\cos(x)\right)$ $e)\ \partial^2/\partial x\partial y\left(x*sin(y)-y*\cos(x)\right)$

5. Los primeros términos no nulos de la serie de MacLaurin para la función arcotangente son $x-\frac{x^3}{3}+\frac{x^5}{5}$. Calcula el error absoluto y relativo en las siguientes aproximaciones de π usando el polinomio en vez de la función arcotangente.

a. $4\cdot\left(atan\left(\frac{1}{2}\right)+atan\left(\frac{1}{3}\right)\right)$

b. $16\cdot atan\left(\frac{1}{5}\right)-4\cdot atan\left(\frac{1}{239}\right)$

Nota: Si p^* es una aproximación de p se definen error absoluto como $\left|p-p^*\right|$ y error relativo como $\frac{\left|p-p^*\right|}{p}$ si $p\neq 0$.

6. Calcula la norma euclídea del vector $(1,-1,2,3,-10,6)$. Hazlo de tres formas diferentes usando los siguientes seudocódigos que definen dos algoritmos para determinar la norma euclídea de un vector $x=(x_1,x_2,\ldots,x_n)$:

a) Nombre: norma1

Entrada : el vector x
Salida : norma de x

Paso1 : Determinar la longitud de x, que designaremos por 'n'
Paso 2 : Hacer suma=0
Paso 3: Para $i = 1,2,\ldots,n$ hacer
suma=suma + x^2_i

Paso 4: Hacer norma=suma^(1/2)

b) Nombre: norma2

Paso 1: Elevar al cuadrado elemento a elemento el vector x y nombrarlo x2 utilizando la instrucción de MatLab correspondiente.
Paso 2: Sumar los elementos de x2 utilizando la instrucción de MatLab correspondiente.
Paso 3: Hacer norma igual a la raíz cuadrada del último valor obtenido en el Paso 2 utilizando la instrucción de MatLab correspondiente.

c) Utilizar también la instrucción de MatLab que calcula directamente la norma euclídea.

7. El número π puede aproximarse teoricamente como la suma se la serie $4 \cdot \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$ aunque veremos que en la prática este método no es el mejor, hemos visto ya en el ejercicio 5 una forma más efectiva de cáculo de π.

a) Calcula las aproximaciones de π para n=100,1000,10000 utilizando la definición de vectores y la instrucción de MatLab `sum`, utilizando el valor de π de MatLab calcular la exactitud de las aproximaciones.
b) Crea un archivo de función que tenga argumento de entrada 'n' y que devuelva la aproximación de π para ese número de sumandos.
c) Fijándose en los datos obtenidos en el apartado a) da una estimación del 'n' necesario para aproximar π con una precisión del orden de 10^{-8}.

SOLUCIONES

4.a)
```
diff('t^3*sin(t)^2')

ans =
3*t^2*sin(t)^2+2*t^3*sin(t)*cos(t)
```

b)
```
diff('1/(1+t^2)',3)

ans =
-48/(1+t^2)^4*t^3+24/(1+t^2)^3*t
```

c)
```
diff('tan(1/t)',2)

ans =
2*tan(1/t)*(1+tan(1/t)^2)/t^4+2*(1+tan(1/t)^2)/t^3
```

d)
```
diff('x*sin(y)-y*cos(x)','x')

ans =
sin(y)+y*sin(x)
```

e)
```
diff(diff('x*sin(y)-y*cos(x)','x'),'y')

ans =
cos(y)+sin(x)
```

6.a)
```
a=[1,-1,2,3,-10,6];

function norma=norma1(x)

n=length(x);

suma=0;

for i=1:n
    suma=suma+x(i)^2;
end

norma=suma^(1/2);

norma1(a)

ans =
   12.2882
```

6.b)

```
function norma=norma2(x)

x2=x.^2;
suma=sum(x2);
norma=sqrt(suma);


norma2(a)

ans =
   12.2882
```

6.c)

```
norm(a)

ans =
   12.2882
```

7. a)

```
n=0:100;
```
% creamos el vector que va de 0 a 100 de 1 en 1.

```
x=(-1).^n./(2*n+1);
```
% a partir del vector anterior y evaluando elemento a elemento obtenemos el vector con los primeros sumandos de la serie hasta el número 100.

```
y=4*sum(x)
```
% evaluamos la suma y multiplicamos por 4 con lo que tendremos la aproximación.

```
y =   3.15149340107099
```

```
abs(y-pi)
```
% calculamos el error absoluto.

```
ans =   0.00990074748120
```

Si ponemos todas las instrucciones en una fila será más fácil ir calculando las aproximaciones, sin necesidad de escribir más, sólo hará falta ir modificando la fila:

```
n=0:1000;x=(-1).^n./(2*n+1);y=4*sum(x),abs(y-pi)

y =
   3.14259165433954 ans =    9.990007497511222e-004
n=0:10000;x=(-1).^n./(2*n+1);y=4*sum(x),abs(y-pi)

y =   3.14169264359053 ans =    9.999000074145670e-005
```

7.b)

```
function [s,e]=aproxpi(n)

% Efectúa una aproximación de pi utilizando la serie
% (-1)^(2n+1)/(2*n+1), hasta el sumando n
% Da también como salida el error absoluto cometido

i=0:n;
x=(-1).^i./(2*i+1);
s=4*sum(x);
e=abs(s-pi);
```

Comprobemos que funciona:

```
[s,e]=aproxpi(1000)

s =   3.14259165433954 e =    9.990007497511222e-004
```

7.c) Si creamos una tabla con los valores de n y los errores (redondeados) obtenidos tenemos:

n	error absoluto
10^2	10^{-2}
10^3	10^{-3}
10^4	10^{-4}

Con lo que aproximadamente necesitaremos un $n=10^8$ para obtener un error de 10^{-8}, si utilizamos esta serie para calcular π.

Se puede comprobar que en este caso nuestra función presenta problemas de memoria y de tiempo de cálculo a partir de cierto n grande, con lo que este método de aproximación de π no seria aplicable para aproximaciones con errores muy pequeños.
En temas posteriores veremos un caso general para acelerar la convergencia de sucesiones y minimizar los cálculos necesarios.

2. Complejos. Polinomios.Curvas en polares.

2.1 Números complejos

Los complejos tienen algo de real y algo de imaginario, pero hoy en día es imposible vivir sin ellos. Lo mejor es conocer sus mecanismos y acostumbrarse a sus repentinas apariciones. Vas tranquilamente por la calle resolviendo enigmas como ese de "la mayor toca el piano" y te tropiezas con un par de complejos conjugados. Puedes elegir entre echar a correr o seguir leyendo.
Como has optado por lo segundo, te vas a enterar de todo lo que siempre quisiste saber sobre los números complejos y nunca te atreviste a preguntar.

– La suma de las edades de dos hermanos es 13 y el producto 36.
– Fácil, tienen 4 y 9 años.
– Pues mi vecino tiene tres hijas cuyas edades también multiplican 36. El número de este portal es la suma de sus edades.
– Me falta un dato.
– ¡Ah, sí! La mayor toca el piano.
– Está claro. ¿Y si la suma es 6 y el producto 13?
– Imposible, te lo has inventado.
– ¿Cómo lo sabes?

Una ecuación de segundo grado con discriminante negativo no tiene solución real. Si insistimos en resolver todas estas ecuaciones, nos vemos en la necesidad de introducir unos entes que amplíen los números reales y contengan las soluciones de tales ecuaciones. Comenzando por la más elemental

$$x^2 + 1 = 0$$

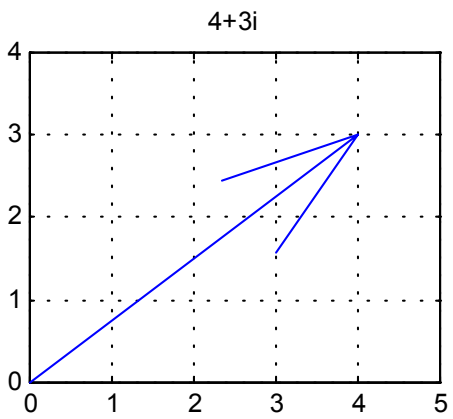
podemos llamar i a un "número imaginario" cuyo cuadrado sea –1 y añadirlo al sistema de números reales. Operando formalmente con $\mathrm{i} = \sqrt{-1}$, las raíces de cualquier ecuación de segundo grado con discriminante negativo se pueden expresar de la forma

$$z = a + b\,\mathrm{i}$$

con a y b números reales. Veremos que estas expresiones constituyen un cuerpo llamado *cuerpo C de los números complejos*. La expresión $z = a + b\,\mathrm{i}$ se denomina *forma binómica* del complejo z. Un número complejo de la forma $a + 0\,\mathrm{i}$ se identifica naturalmente con el número real a. Los complejos de la forma $0 + b\,\mathrm{i}$ se denominan *imaginarios puros*. En el cuerpo de los números complejos, no sólo las ecuaciones de segundo grado tienen solución, sino que toda ecuación polinómica de grado n tiene n soluciones en C. Este resultado se conoce como el Teorema Fundamental del Álgebra.

2.1.1 Representación cartesiana

Un complejo $z = a + b\,\mathrm{i}$ consta de una *parte real, a,* y de una *parte imaginaria, b*. Por ello, resulta natural representarlo en coordenadas cartesianas mediante un vector o un punto de abscisa a y ordenada b. De aquí el llamar *plano complejo* al cuerpo C.
MATLAB trabaja por defecto con números complejos, que se introducen tal y como se escriben en matemáticas, utilizando i o j para la unidad imaginaria. Escribe, por ejemplo `z = 4 + 3i`.



Observa que no es necesario poner el signo de multiplicación, *, entre el 3 y la i. Las funciones `real(z)` e `imag(z)` proporcionan, respectivamente, la parte real y la parte imaginaria del complejo *z*.
El complejo $z = a + b\,\mathrm{i}$ se representa gráficamente como el punto del plano de coordenadas cartesianas (*a*, *b*) o como un vector con que va del origen al mismo punto.
En MATLAB, el afijo representa con `plot(z,'*')`, mientras que para obtener el vector puede usarse la orden `compass(z)`. Esta última presenta el vector sobre uno ejes tipo radar (coordenadas polares). Podemos forzar el uso de los ejes cartesianos creándolos antes de representar el vector: `close, axis, hold on, compass(z)`.

2.1.2 Operaciones con complejos

2.1.2.1 Suma, resta, opuesto

Para *sumar* dos complejos basta sumar sus partes reales y sus partes imaginarias por separado.

$$(4 - \mathrm{i}) + (2 + 3\,\mathrm{i}) = (4 + 2) + (-1 + 3)\mathrm{i} = 6 + 2\,\mathrm{i}$$

La suma tiene las propiedades habituales que confieren a *C* la estructura de *grupo conmutativo*. El *neutro* es el complejo 0 + 0i. El *opuesto* de $z = a + b\,\mathrm{i}$ es $-z = -a - b\,\mathrm{i}$.

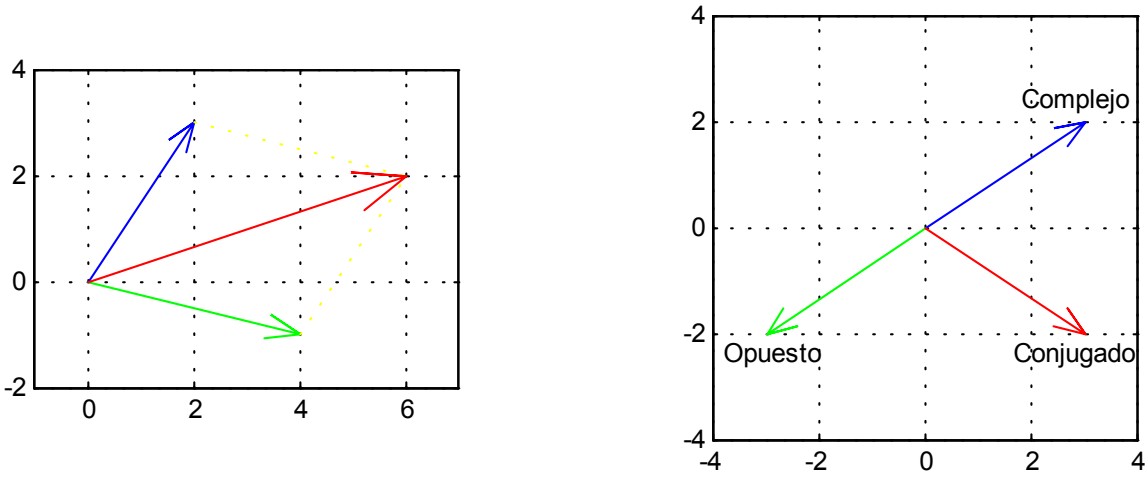


Figura 1 a) Suma de complejos; **b)** Opuesto y conjugado de un complejo

El concepto de conjugado juega un papel muy importante. Se define el *conjugado* de un complejo $z = a + b\,\mathrm{i}$ como el complejo $\bar{z} = a - b\,\mathrm{i}$. Gráficamente, tomar conjugados equivale a hacer una simetría respecto al *eje real*. La sintaxis en MATLAB es `conj(z)`.

2.1.2.2 Producto, inverso, cociente

La *multiplicación* de complejos se define de forma natural operando formalmente con las expresiones binómicas de los factores y aplicando la relación fundamental $\mathrm{i}^2 = -1$.

$$(a + b\,\mathrm{i})(c + d\,\mathrm{i}) = ac + ad\,\mathrm{i} + bc\,\mathrm{i} + bd\,\mathrm{i}^2 = (ac - bd) + (ad + bc)\mathrm{i}$$

Por ejemplo,

$$(4 - \mathrm{i}) \cdot (2 + 3\,\mathrm{i}) = (4 \cdot 2 + 3) + (4 \cdot 2 - 2)\mathrm{i} = 11 + 6\,\mathrm{i}$$

La operación así definida hereda las propiedades *conmutativa*, *asociativa* y *distributiva* del producto de números reales. El número 1 se comporta como *elemento unidad* del producto. Se comprueba que todo complejo no nulo tiene *inverso* operando

formalmente y usando el mismo truco que para eliminar una raíz cuadrada del denominador de una fracción.

$$\frac{1}{a+b\,\mathrm{i}}=\frac{a-b\,\mathrm{i}}{(a+b\,\mathrm{i})(a-b\,\mathrm{i})}=\frac{a-b\,\mathrm{i}}{a^2+b^2}=\frac{a}{a^2+b^2}-\frac{b\,\mathrm{i}}{a^2+b^2}$$

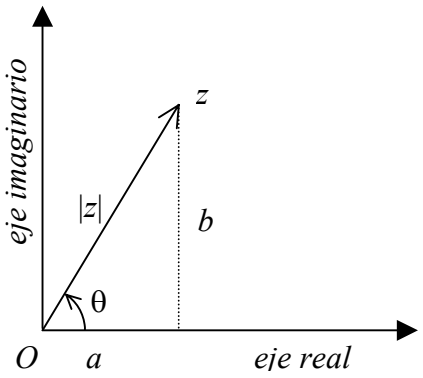
Todas estas propiedades se resumen diciendo que *C* tiene estructura de cuerpo.

MATLAB trabaja por defecto con números complejos, por lo que las operaciones anteriores se efectúan con el formalismo usual: si *u* y *v* son dos complejos, $u+v$, $u*v$ y u/v, proporcionan los resultados de las operaciones indicadas.

Las expresiones binómicas del producto y del cociente no son muy sugerentes. Las nociones de módulo y argumento que introduciremos a continuación, aparte de otras propiedades más importantes, permiten dar una interpretación geométrica intuitiva de dichas operaciones.

2.1.3 Módulo y Argumento

Si $z=a+b\,\mathrm{i}$ es un número complejo, a^2+b^2 es un número real, cuya raíz cuadrada se llama *módulo* del complejo *z* y se denota por $|z|$.



Geométricamente, el módulo representa la longitud del vector asociado a *z*. En MATLAB, el módulo se obtiene con `abs`.

El módulo de un complejo juega un papel análogo al del valor absoluto para números reales. Listemos sus propiedades fundamentales.

1. $|z|\geq 0;\quad |z|=0$ sólo si $z=0$.
2. $|zw|=|z||w|$
3. $|z+w|\leq |z|+|w|$

Estas propiedades definen una *norma* en *C*.

La segunda propiedad indica que el módulo del producto de dos complejos es igual al producto de sus módulos. Usaremos este hecho en la caracterización geométrica del producto.

La tercera propiedad es la llamada *desigualdad triangular*. por la interpretación geométrica de la suma de complejos.

Llamamos *argumento* de un número complejo no nulo $z=a+b\,\mathrm{i}$, al ángulo que forma el vector asociado con el semieje real positivo, medido en sentido contrario a las agujas del reloj. Al dividir *z* por su módulo, queda un complejo de módulo 1 cuyo afijo está sobre la circunferencia de centro el origen y radio 1. Si llamamos *x* e *y* a sus coordenadas, existe un único θ en [0, 2π[tal que

$$x=\cos\theta$$

$$y=\sin\theta$$

Decimos que θ es un argumento del complejo *z*. Cualquier otro valor que difiera de θ en un múltiplo entero de 2π es también argumento de *z*. En MATLAB, el argumento se obtiene con `angle`.

Un complejo *z* puede representarse en *forma modulo-argumental o trigonométrica* como

$$z=|z|(\cos\theta+i\sin\theta)$$

siendo $|z|$ el módulo *z* y θ es el argumento.

Aplicando relaciones de trigonometría elemental, se comprueba que el argumento del producto de dos complejos es la suma de los argumentos de los factores. Como el argumento de 1 es 0, complejos inversos tienen argumentos opuestos. Asimismo, el argumento del cociente es la diferencia de los argumentos. Siempre que hablamos de argumentos, hay que considerarlos definidos salvo adición de $2k\pi$, con k entero.
En consecuencia, podemos decir que para multiplicar complejos, se multiplican los módulos y se suman los argumentos. Para dividir complejos, se dividen los módulos y se restan los argumentos. El inverso de z tiene por módulo el inverso del módulo de z y por argumento el opuesto del argumento de z. Otra forma de caracterizar el inverso de z es como el conjugado dividido por el cuadrado del módulo. En general se tiene

$$z\,\bar{z} = |z|^2$$

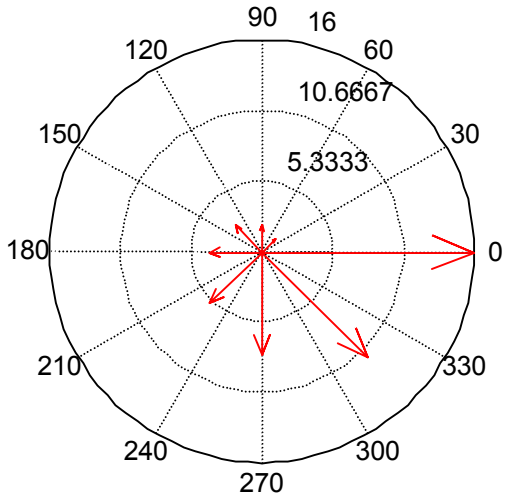
2.1.4 Potencias y raíces

De la interpretación geométrica del producto se deduce que la potencia n-ésima de un complejo z tendrá por módulo la potencia n-ésima del módulo de z y por argumento n veces el argumento de z. O sea,

$$|z^n| = |z|^n$$

$$\arg(z^n) = n \arg z.$$

Alternativamente, $z^n = (a + b\,\mathrm{i})^n$ puede desarrollarse por la fórmula de Newton de la potencia n-ésima de un binomio, obteniéndose, tras simplificar, la expresión binómica del resultado.



Ejemplo 1

El complejo $z = 1 + \mathrm{i}$ tiene módulo $\sqrt{2}$ y argumento $\pi/4$. Las sucesivas potencias de z, z^k, para $k = 0, 1, 2, \ldots, 7$ tienen de módulo $\sqrt{2}^k$ y de argumento $k\pi/4$. Podemos calcularlas y representarlas con MATLAB mediante:

```
z = 1 + i;
compass([z z^2 z^3 z^4 z^5 z^6 z^7 z^8],'r')
```

El cálculo de la raíz enésima de un complejo requiere algo más de cuidado. En el campo real, sólo los números positivos tienen raíz cuadrada y ésta puede tener dos signos. Por ejemplo, 3 y –3 son raíces cuadradas de 9, mientras que –9 no tiene raíz cuadrada. En cambio, todo real tiene exactamente una raíz cúbica real, por ejemplo, la raíz cúbica de –8 es –2 y la raíz cúbica de 8 es 2. Estos comportamientos se generalizan a raíces de índice par o impar, respectivamente.
Curiosamente, en el campo complejo, la situación es más sencilla. Todo complejo no nulo tiene exactamente n raíces de orden n. Esto constituye un caso particular del Teorema Fundamental del Álgebra que establece que todo polinomio de grado n tiene n raíces (contando la multiplicidad).
Las raíces n-simas de un complejo se obtienen fácilmente usando la expresión modulo-argumental.

Dado un complejo no nulo $z = |z|(\cos\theta + i\sin\theta)$, tratamos de hallar los complejos $w = |w|(\cos\varphi + i\sin\varphi)$ tales que $w^n = z$. Por las propiedades de las potencias tenemos que

$$|w^n| = |w|^n = |z|$$

de donde $|w|$ es la raíz n-ésima positiva (real) de $|z|$. Por otra parte,

$$\arg(w^n) = n \arg w = \arg z.$$

Como el argumento está definido salvo adición de múltiplos enteros de 2π, tenemos

$$n\varphi = \theta + 2k\pi,$$

de donde

$$\varphi = \frac{\theta}{n} + k\frac{2\pi}{n},$$

expresión que toma distintos valores para $k = 0, 1, 2, \ldots, n-1$ o para cualesquiera otros n valores de k consecutivos. En definitiva, tenemos n raíces n-simas distintas de un complejo z, que escribimos como:

$$w_k = |z|^{\frac{1}{n}}(\cos\varphi_k + i\sin\varphi_k), \text{ donde } \varphi_k = \frac{\theta}{n} + k\frac{2\pi}{n}, \text{ para } k = 0, 1, 2, \ldots, n-1.$$

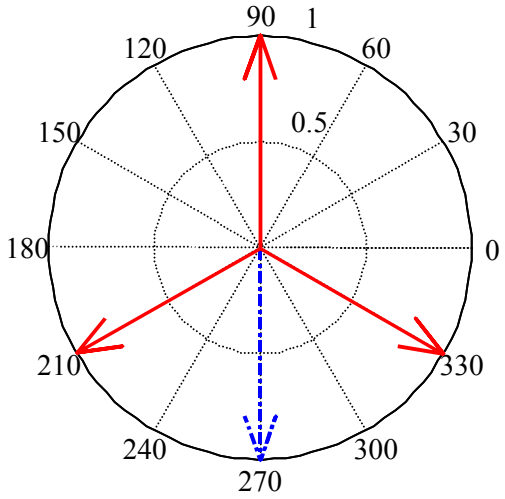
Ejemplo 2

El complejo $z = -\mathrm{i}$ tiene módulo 1 y argumento $3\pi/2$. Las raíces cúbicas de z tienen módulo 1 y argumento $\frac{\pi}{2} + k\frac{2\pi}{3}$, para $k = 0, 1, 2$ y sus expresiones son:

$$\cos\frac{\pi}{2} + i\sin\frac{\pi}{2} = \mathrm{i}$$

$$\cos\frac{7\pi}{6} + i\sin\frac{7\pi}{6} = -\frac{\sqrt{3}}{2} - \frac{1}{2}\mathrm{i}$$

$$\cos\frac{11\pi}{6} + i\sin\frac{11\pi}{6} = \frac{\sqrt{3}}{2} - \frac{1}{2}\mathrm{i}$$



Representémoslas con MATLAB:

```
z = -i;
compass(z,'-.')
hold
for k = 0:2
    fi = pi/2 + k*2*pi/3;
    compass(cos(fi)+i*sin(fi),'r')
end
```

2.1.5 Fórmula de Euler

Las funciones reales seno, coseno y exponencial pueden aproximarse indefinidamente por su desarrollo de Taylor en cualquier intervalo acotado. Si aplicamos el desarrollo de la exponencial al número imaginario ib, podemos escribir

$$
\begin{aligned}
e^{i\theta} &= 1 + i\theta + \frac{(i\theta)^2}{2!} + \frac{(i\theta)^3}{3!} + \frac{(i\theta)^4}{4!} + \frac{(i\theta)^5}{5!} + \cdots \\
&= 1 + i\theta - \frac{\theta^2}{2!} - i\frac{\theta^3}{3!} + \frac{\theta^4}{4!} + i\frac{\theta^5}{5!} + \cdots \\
&= 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \cdots \quad i\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \cdots\right)
\end{aligned}
$$

y recordando los desarrollos del seno y del coseno ponemos

$$e^{i\theta} = \cos\theta + i\sin\theta$$

que constituye la fórmula de Euler. Podemos rescribir la forma trigonométrica de un complejo como

$$z = |z|(\cos\theta + i\sin\theta) = |z|\,e^{i\theta}$$

La interpretación geométrica de producto y cociente se deriva ahora formalmente de las propiedades de la exponencial.

Definimos formalmente también la *exponencial compleja* mediante

$$e^{z} = e^{x+iy} = e^{x}(\cos y + i\sin y)$$

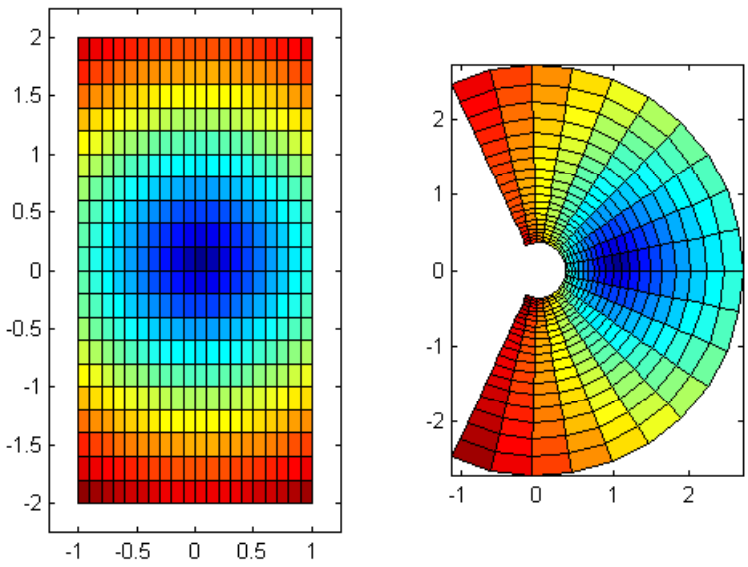


Figura 2: Original e imagen de un rectángulo por la exponencial compleja

La exponencial de z es un complejo w cuyo módulo es la exponencial de la parte real de z y cuyo argumento es la parte imaginaria de z, o sea,

$$|w| = e^{x}, \quad \arg(w) = y$$

Tomando logaritmos

$$x = \log|w|$$

luego

$$z = x + iy = \log|w| + i\arg w$$

Como w es la exponencial de z, parece lógico decir que z es el *logaritmo de w*. Más precisamente deberíamos decir que z es un logaritmo de w, pues cualquier complejo que difiera de z en un múltiplo entero de $2\pi i$, también lo será. En resumen,

$$\log w = \log|w| + i(\arg w + 2k\pi) \quad \text{para cualquier } k \text{ entero.}$$

2.2 Polinomios

Los polinomios constituyen una clase de funciones de gran importancia, debido a que son fáciles de evaluar y, sobre todo, a que toda función continua en un intervalo cerrado puede aproximarse mediante polinomios.
Los polinomios surgen a menudo en problemas de resolución de ecuaciones. Curiosamente, a pesar de la sencillez y las buenas propiedades de los polinomios, solamente se pueden resolver mediante fórmulas exactas las ecuaciones polinómicas de grado no superior a 4. De hecho, las fórmulas para ecuaciones de grado superior a 2 son raramente utilizadas; en su lugar se utilizan métodos numéricos.
En esta sección recordamos las operaciones y propiedades básicas de los polinomios y las funciones de MATLAB que actúan sobre ellos.

Ejemplo 3

Cada mes ahorras parte de tu paga para comprar un regalo a un/a amigo/a. A primeros de mes, haces un ingreso en una cuenta "remunerada", hasta que logras reunir la cantidad deseada. Esta cantidad se expresa como un polinomio que depende de las imposiciones y del tipo de interés mensual.
El tipo de interés mensual en tanto por uno, r, es lo que produce 1 euro ingresado en la cuenta al cabo de un mes. En general, un capital C se convierte en un mes en $C(1+r)$. El interés compuesto consiste en acumular los intereses al capital al final de cada periodo, de modo que produzcan a su vez intereses durante el periodo siguiente. En este caso, la cantidad acumulada al final del primer mes, $C(1+r)$, produce intereses durante el segundo mes, convirtiéndose al final de este periodo en $C(1+r)^2$ y así sucesivamente.
Si ahorras durante un año, imponiendo a_1 euros el primero de enero, a fin de año tendrás $a_1(1+r)^{12} = a_1 x^{12}$ euros, siendo $x = 1+r$. Un ingreso de a_2 euros el 1 de febrero se convierte en $a_2 x^{11}$ euros a final de año y así sucesivamente. El dinero acumulado a fin de año con este plan de ahorro viene dado por el polinomio

$$C = a_1 x^{12} + a_2 x^{11} + \cdots + a_{12} x + a_{13}$$

donde a_i es la cantidad impuesta al principio del i-ésimo mes. La cantidad a_{13} corresponde a una cantidad final que se añade el último día del año y que no produce intereses.

2.2.1 Representación mediante vectores

El *polinomio de grado n*

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

se representa en MATLAB mediante el vector de longitud $n+1$

```
p = [a1 a2  . . . an an+1]
```

cuyos elementos son los *coeficientes* del polinomio ordenados de mayor a menor grado. Incluimos los coeficientes nulos para conservar el grado del polinomio. Por ejemplo, el polinomio x^2 se representa mediante el vector `[1 0 0]`, mientras que `[1]` representa al *polinomio constante* 1.

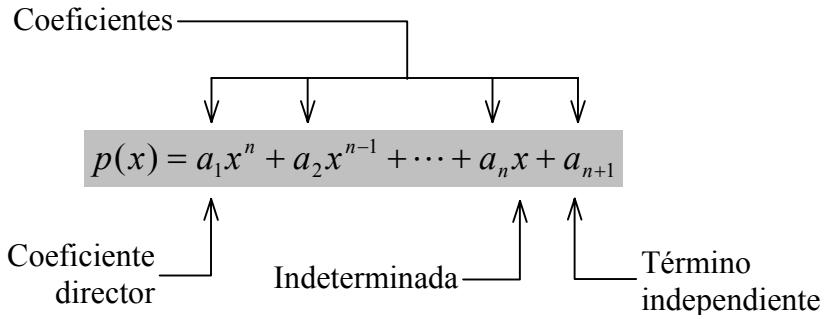


Figura 3 Nomenclatura de polinomios

2.2.2 Operaciones con polinomios

Para operar con polinomios en MATLAB, hay que traducir a operaciones con vectores las correspondientes operaciones con polinomios. Como la *suma*, resta y *producto por un escalar* se efectúan término a término, basta operar con los coeficientes de cada término. Para sumar o restar en MATLAB polinomios de distinto grado, hay que representarlos como vectores de la misma longitud.

Ejemplo 4

Para sumar en MATLAB los polinomios x^2 y 3 hacemos

```
p = [1 0 0]
q = [0 0 3]
p+q
```

Multiplicar polinomios es tarea fácil con MATLAB, usando la función `conv`.

Ejemplo 5

Con MATLAB $(x-1)^6$ se puede obtener mediante

```
p = [1 -1];
q = conv(p,p)      % (x-1)^2
r = conv(q,q)      % (x-1)^4
conv(q,r)
```

Recordemos el resultado fundamental de la *división* de polinomios:
Dados dos polinomios con coeficientes reales (complejos) $p(x)$ y $q(x)$, con $q(x)$ no nulo, pueden determinarse de forma única dos polinomios $c(x)$ y $r(x)$ tales que

(i) $p(x) = q(x)c(x) + r(x)$
(ii) $r(x)$ es idénticamente nulo o de menor grado que $q(x)$

Los polinomios $c(x)$ y $r(x)$ se llaman el *cociente* y el *resto* de la división. Estos polinomios pueden obtenerse a mano por el método de Euclides. Afortunadamente, si no recordamos este procedimiento, podemos recurrir la función `deconv` de MATLAB, cuya sintaxis es

```
[c,r] = deconv(p,q)
```

siendo `p`, `q`, `c` y `r` vectores que representan los polinomios dividendo, divisor, cociente y resto.

Ejemplo 6

Dividimos $p(x) = x^5 - 2x^4 + 2x^3 - x^2 + 3x + 7$ por $q(x) = x^2 + 2$, con `deconv`

```
p = [1 -2 2 -1 3 7];
q = [1 0 2];
```

```
[c,r] = deconv(p,q)
```

El cociente es $c(x) = x^3 - 2x^2 + 3$ y el resto $r(x) = 3x + 1$. Se cumplen las propiedades de la división porque el resto es de menor grado que el divisor y se verifica la relación (i), como se comprueba mediante

```
p - conv(q,c) - r
```

Los ceros iniciales que MATLAB añade a `r` sirven para que los sumandos de la expresión anterior tengan la misma longitud.

2.2.3 Valor de un polinomio. Teorema del resto

Para que la representación de un polinomio $p(x)$ mediante un vector `p` resulte práctica, se necesita una forma fácil de obtener el valor del polinomio al sustituir la *indeterminada* x por un número dado a. Esto se hace en MATLAB mediante `polyval(p,a)`.

Ejemplo 7

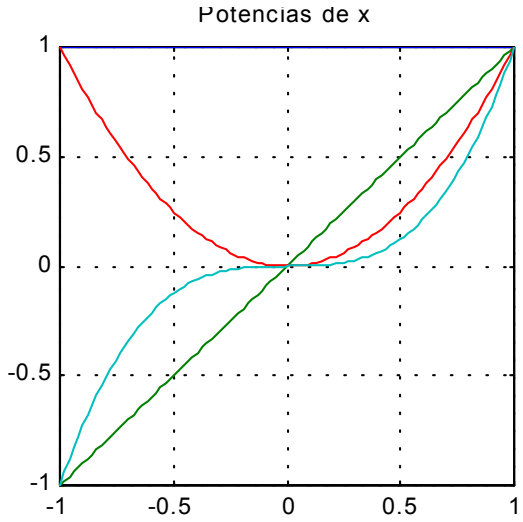
Si ingresas 10€ el 1 de cada mes, desde el 1 de enero, al 0.3% mensual, ¿cuánto habrás acumulado hasta el 1 de enero del año siguiente? (Incluyendo 10€ de ese último día).
Siguiendo la idea del Ejemplo 3, construimos un polinomio cuyos coeficientes son las imposiciones y lo evaluamos en $1 + r$, siendo r el tipo de interés mensual.

```
p = 10*ones(1,13);
r = 0.3/100;
Capital = polyval(p,1+r)
```

Ejemplo 8

Para representar gráficamente los polinomios 1, x, x^2 y x^3 en el intervalo $[-1, 1]$, es más fácil evaluarlos directamente que usar polyval.

```
x = linspace(-1,1)'
plot(x, [x.^0 x x.^2 x.^3])
title('Potencias de x'), grid
```



Para evaluar a mano un polinomio, en lugar de sustituir directamente la indeterminada en la expresión del polinomio, es más cómodo aplicar la regla de Ruffini, que equivale a dividir el polinomio por $x - a$, siendo a el punto en que queremos evaluar el polinomio. La relación entre evaluación y división se establece en el siguiente teorema.

Teorema del Resto

El valor del polinomio $p(x)$ en a es el resto de la división de $p(x)$ por $x - a$.

Ejemplo 9

Aplicamos el teorema del resto para el polinomio $p(x) = x^2 - 1$ con $a = 2$, comprobando que el resto de la división por $x - 2$ coincide con el valor del polinomio en 2.

```
p = [1 0 -1]; q = [1 -2]
[c, r] = deconv(p,q)    % El resto es la última componente de r
polyval(p, 2)           % Valor del polinomio en 2
```

2.2.4 Raíces de un polinomio. Divisibilidad.

Decimos que un número *r* es *raíz* del polinomio *p*(*x*), si el valor del polinomio en dicho número es 0, o sea, *p*(*r*) = 0. Por ejemplo, 1 y –1 son raíces del polinomio $x^2 - 1$.
La existencia y propiedades de las raíces están estrechamente relacionadas con las propiedades de divisibilidad de polinomios. Como consecuencia del antes mencionado teorema del resto, se obtiene un resultado que establece esta relación:

Teorema del Factor
r es raíz de *p*(*x*) si y sólo si *p*(*x*) es divisible por *x* – *r*.

Si *p*(*x*) admite varios factores de la forma *x* – *r*, entonces *r* es una *raíz múltiple* del polinomio. Si sólo admite un factor, es *raíz simple*. La multiplicidad *m* de una raíz *r* del polinomio *p(x)* es el máximo entero *m* tal que $(x-r)^m$ divide a *p(x)*. Por ejemplo, 0 es una raíz de multiplicidad *n* del polinomio x^n.
Los métodos de cálculo de raíces que se aprenden en la escuela consisten en buscar factores de la forma *x* – *r* del polinomio en cuestión. Naturalmente, estos métodos sólo son prácticos para hallar raíces enteras y poco más en casos preparados.
Teóricamente se demuestra (Teorema fundamental del álgebra) que un polinomio de grado *n* tiene *n* raíces en el cuerpo de los números complejos, contando sus multiplicidades.
La necesidad de trabajar con números complejos se manifiesta con la simple ecuación $x^2 + 1 = 0$, que no tiene raíces reales.
El cálculo de todas las raíces de un polinomio es un asunto complicado que MATLAB resuelve de un plumazo con `roots`.

Ejemplo 10

`roots([1 0 1])` proporciona las raíces de $x^2 + 1$.

Al aplicar `roots` a un polinomio se obtiene un vector con sus raíces. Recíprocamente, se puede obtener un polinomio a partir de sus raíces mediante la función `poly`.

Ejemplo 11

`poly([-i, i])` da como resultado `[1 0 1]`.

2.2.5 Descomposición en factores irreducibles.

Combinando los teoremas del resto y fundamental del álgebra se prueba que todo polinomio complejo de grado *n* > 0 se puede escribir como

$$p(x) = a_1 (x - r_1)^{m_1} (x - r_2)^{m_2} \quad \cdots\cdots (x - r_k)^{m_k}$$

donde $r_1, r_2, \ldots, r_k$ son todas las raíces distintas del polinomio $p(x)$. Esta descomposición es única, salvo el orden de los factores. El exponente m_j se denomina orden de multiplicidad de la raíz r_j.
En el caso real, la descomposición contiene también factores de grado 2 correspondientes a polinomios de segundo grado que no tienen raíces reales. Las raíces complejas de un polinomio real aparecen por pares conjugados, es decir, un complejo *r* es raíz de un polinomio $p(x)$ de coeficientes reales si y sólo si su conjugado $\bar{r}$ es raíz del polinomio. El producto $(x - r)(x - \bar{r})$ es un polinomio real de segundo grado sin raíces reales que aparece en la factorización real de $p(x)$.

descompone en producto de factores de la forma (de grado 1, corespondientes a las raíces reales o de grado 2, correspondientes a pares de raíces complejas conjugada)

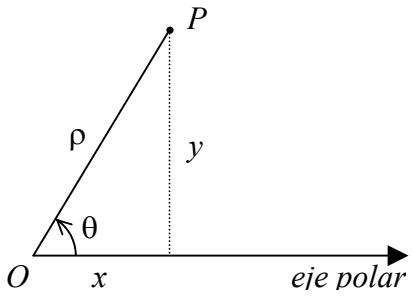
Ejemplo 12

El polinomio x^3-1 tiene una raíz real $x=1$ y dos raíces complejas conjugadas $x=-\frac{1}{2}\pm\frac{\sqrt{3}}{2}\mathrm{i}$. Sus factorizaciones compleja y real son , respectivamente,

$$x^3-1=(x-1)\left(x+\frac{1}{2}-\frac{\sqrt{3}}{2}\mathrm{i}\right)\left(x+\frac{1}{2}+\frac{\sqrt{3}}{2}\mathrm{i}\right)=(x-1)(x^2+x+1)$$

2.3 Curvas en coordenadas polares

El radar permite localizar un objeto en el plano conociendo su distancia a la antena y su orientación. Matemáticamente hablando, se fija un punto *O* del plano como origen de distancias o *polo* y un semieje que parte del polo a partir del cual se miden los ángulos en sentido contrario a las agujas del reloj. Las coordenadas polares de un punto *P* son la distancia ρ al polo y el ángulo θ que forma OP con el *eje polar*.



Tomando el polo como origen de coordenadas cartesianas y el eje polar como semieje de abscisas positivo, la relación entre las coordenadas polares (θ, ρ) y cartesianas (*x*, *y*) del punto P viene dada por las ecuaciones

$$x=\rho\cos\theta$$
$$y=\rho\sin\theta$$

La relación inversa debe tratarse con más cuidado, pues, si el punto no es el origen, el ángulo está definido salvo adición de múltiplos enteros de 2π. En todo caso, la distancia se determina mediante

$$\rho=\sqrt{x^2+y^2}$$

y para obtener el ángulo, se elige un valor de θ tal que

$$\cos\theta=\frac{x}{\rho}\quad\text{y}\quad\sin\theta=\frac{y}{\rho}$$

Este valor es único en el intervalo [0, 2π[o en qualquier trasladado del mismo. Alternativamente, se suele determinar θ usando juiciosamente la fórmula

$$\theta=\arctan\frac{y}{x}$$

Con MATLAB, la distancia del punto (*x*, *y*) al origen se obtiene mediante `norm([x,y])` y el ángulo mediante la función `atan2(y,x)`. Observa que `abs` actúa sobre el vector, mientras que `atan2` lo hace sobre las coordenadas en orden inverso.

Ejemplo 13

Las coordenadas polares de (1, −1) son $\theta=-\frac{\pi}{4}$ y $\rho=\sqrt{2}$. Hacemos el cálculo con MATLAB.

```
x = 1; y = -1;          % Abscisa y ordenada
```

```
r = norm([x,y])        % Distancia
z = atan2(y,x)         % Ángulo
```

La orden `polar(z,r,'*')` representa gráficamente los puntos del plano cuyas coordenadas polares están almacenadas en los vectores z (ángulo) y r (distancia).
También se puede pasar de coordenadas cartesianas a polares trabajando con complejos. Dado el punto (x, y), basta tomar el complejo $x + yi$ y calcular su módulo y su argumento.

Ejemplo 14

```
u = 1 - i;        % Pasa a complejos
r = abs(u)        % Módulo
z = angle(u)      % Argumento
```

2.3.1 Circunferencias, cardioides

Las coordenadas polares son útiles para describir algunas trayectorias, poniendo el módulo en función del ángulo, $\rho = f(\theta)$, y variando θ en determinado intervalo. Por ejemplo, $\rho = 2$ es la ecuación de una circunferencia de centro el origen y radio 2 en coordenadas polares. Para representarla gráficamente escribimos

```
z = linspace(0, 2*pi);
r = 2*ones(size(z));
polar(z,r)
```

Las ecuaciones polares $\rho = \sin\theta$ y $\rho = \cos\theta$ corresponden a circunferencias que pasan por el polo. La primera es tangente al eje polar y la segunda lo tiene por diámetro.

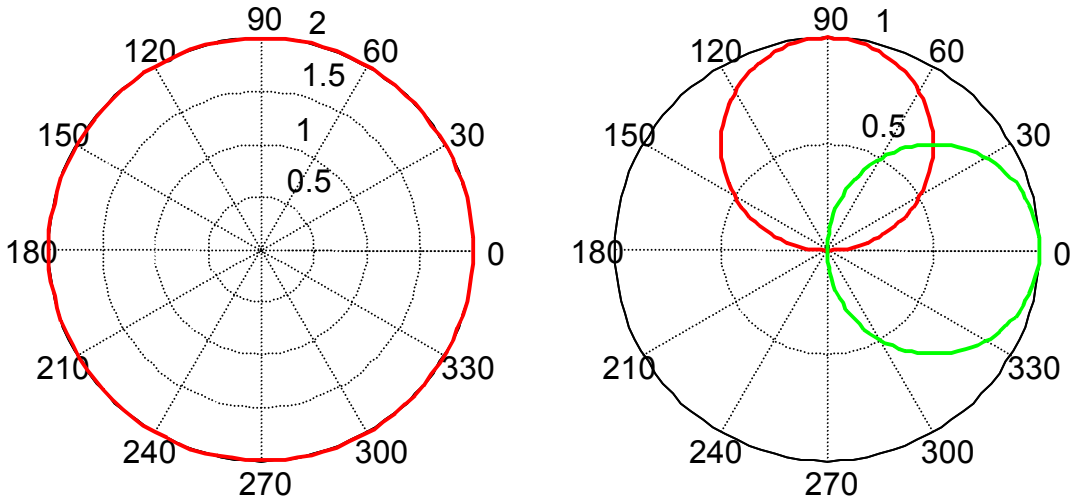


Figura 4: Circunferencias $\rho = 2$, $\rho = \sin\theta$ **y** $\rho = \cos\theta$

```
z = linspace(0, pi,101);
r = sin(z);
polar(z,r)
```

```
 z = linspace(0, pi);
r = cos(z);
polar(z,r)
```

La gráfica de ecuación polar $\rho = \sin(\theta/2)$ se denomina cardioide. Veamos por qué.

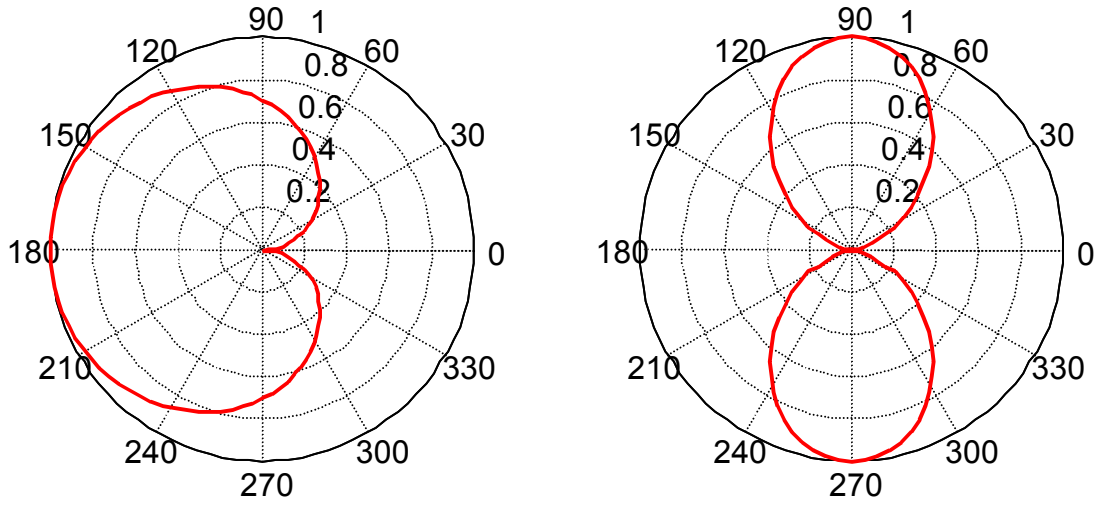


Figura 5: Cardioide $\rho = \sin(\theta/2)$ y diagrama de radiación de antena $P = \sin^2(\theta)$

2.3.2 Diagramas de radiación de antenas (I)

La densidad de potencia radiada por una antena elemental depende del ángulo respecto al eje de la antena según $P = P_{max}\sin^2(\theta)$.

```
z = linspace(0, 2*pi, 101);
r = sin(z/2);
polar(z,r)
```

En el caso de un dipolo de longitud L la expresión de la potencia radiada depende de

$$\frac{\left[\cos\left(\frac{kL}{2}\cos\theta\right) - \cos\left(\frac{kL}{2}\right)\right]^2}{\sin^2\theta}$$

El caso $kL = \pi$ se denomina dipolo de media onda y el de $kL = 2\pi$, dipolo de onda completa.

2.3.3 Espirales arquimediana y logarítmica

La función lineal en polares, $\rho = a\theta$, se representa como una espiral de paso fijo, llamada espiral de Arquímedes. En la naturaleza aparece frecuentemente la espiral logarítmica $\rho = e^{a\theta}$.

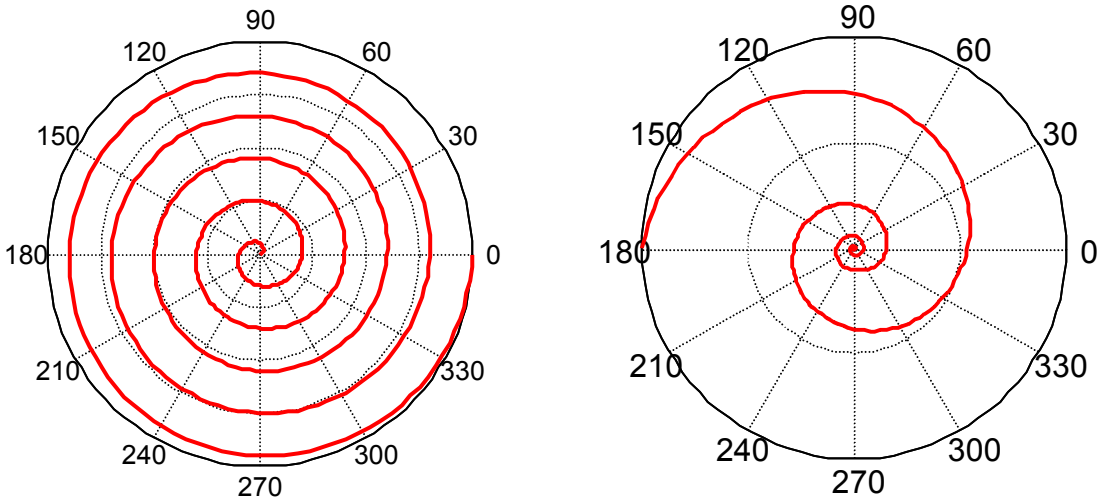


Figura 6: Espirales arquimediana y logarítmica

2.4 Problemas

1. Deduce las expresiones siguientes a partir de la fórmula de Euler.
$$\cos x = \frac{e^{ix} + e^{-ix}}{2}; \quad \sin x = \frac{e^{ix} - e^{-ix}}{2i}.$$
2. Deduce de la expresión $e^{ai} \cdot e^{bi} = e^{(a+b)i}$ las fórmulas del seno y coseno del ángulo suma $a + b$.
3. Prueba que la expresión $A\cos t + B\sin t$ puede ponerse como $\sqrt{A^2 + B^2}\,\sin(t + \varphi)$, para determinado valor de φ.
4. Un ahorrista ingresa una cantidad a_j al comienzo del j-ésimo periodo, j = 1, 2, ...n, a interés compuesto. El tipo de interés por periodo en tanto por uno es *r*. Escribe una función que proporcione el saldo de la cuenta al principio de cada periodo. Representa gráficamente la situación de una cuenta que ofrece un interés del 1% por periodo, en la que las imposiciones (disposiciones) al inicio de cada periodo son 100, 25, 80, –45, 125, 30, 70.
5. Divide el polinomio $x^3 - 2x - 5$ por $x - 2$
   a) manualmente, utilizando el algoritmo de división
   b) por la regla de Ruffini
   c) utilizando la orden `deconv`

   Evalúa el mismo polinomio en $x = 2$
   d) por sustitución directa
   e) usando `polyval`
6. Halla $(x-1)^6$ a partir de sus raíces.
7. Halla los coeficientes del polinomio $(2x-3)^5$ y represéntalo gráficamente en el intervalo [1, 2].
8. Expresa el polinomio $p(x) = 5x^3 - x^2 + 3x + 1$ como combinación lineal de potencias de $(x-1)$,

$$p(x) = a(x-1)^3 + b(x-1)^2 + c(x-1) + d$$

a) desarrollando las potencias de $x-1$ e identificando coeficientes de las potencias de x del mismo grado en ambas expresiones.

b) observando que $p(1) = d$, $p'(1) = c$, $p''(1) = 2b$ y $p'''(1) = 3 \cdot 2a$.

c) escribiendo $p(x)$ en forma anidada

$$p(x) = ((a(x-1) + b)(x-1) + c)(x-1) + d\,,$$

y observando que d es el resto de la división

$$p(x) = (x-1)p_1(x) + d\,,$$

que c es el resto de la división de $p_1(x)$ entre $x-1$ y así sucesivamente.

d) Escribe una función de MATLAB, `q = shiftpol(p,a)` que obtenga los coeficientes `q = [q1, q2, ..., qn]` de la expresión del polinomio `p = [p1, p2, ..., pn]` como combinación de potencias de $x - a$.

9. Las cifras de un número entero en notación decimal son los coeficientes de un polinomio cuyo valor en 10 es el número dado. Los dispositivos electrónicos almacenan y operan con números en base 2, porque basta usar dos cifras, 0 y 1, para representar todos los números. Para pasar de base 2 a base 10, basta evaluar el correspondiente polinomio en 2. Por ejemplo, $1101_2 = 2^3 + 2^2 + 0 \cdot 2 + 1 = 13$. Recíprocamente, las cifras de la expresión en base 2 un número decimal son los restos de las divisiones sucesivas por 2 de dicho número y sus cocientes. Expresa el número 53 en base 2.

10. La función e^x se aproxima en un entorno del origen mediante el polinomio de Taylor

$$p_n(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

Representa la función exponencial y sus sucesivas aproximaciones en un intervalo de la forma $[-a, a]$.

11. La función $\log(1+x)$ se aproxima en un entorno del origen mediante el polinomio de Taylor

$$p_n(x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + (-1)^{n+1}\frac{x^n}{n}$$

Representa la función $\log(1+x)$ y sus sucesivas aproximaciones en el intervalo $]-1, 1[$. Observa la diferencia con el problema anterior.

12. Escribe un fichero de MATLAB que genere dos gráficos en la misma ventana (usa subplot), uno que represente en el plano complejo las raíces de un polinomio real, y otro que represente en el plano cartesiano los valores del mismo polinomio en un intervalo que contenga las raíces reales.

13. Demuestra que las raíces complejas de un polinomio real aparecen en pares conjugados, es decir, que si $a + b\mathrm{i}$ es raíz de un polinomio con coeficientes reales, $a - b\mathrm{i}$ también lo es. Halla un polinomio real de segundo grado que admita como raíz. ¿Cuál es su discriminante?

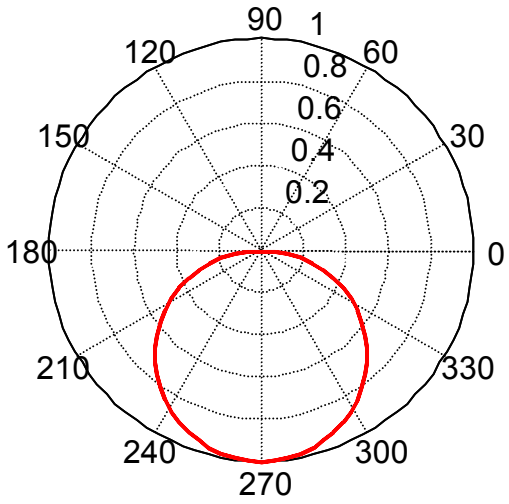
14. Halla la descomposición en factores irreducibles reales del polinomio $x^5 + x^4 + x^3 + x^2 + x + 1$.

15. Representa gráficamente las curvas

a) $\rho = \sin((2k+1)\theta)$, $\theta \in [0, \pi]$ para $k = 0, \pm 1, \pm 2, \ldots$

b) $\rho = \sin(2k\theta)$, $\theta \in [-\pi, \pi]$ para $k = 0, \pm 1, \pm 2, \ldots$

c) Averigua la ecuación polar de la curva de la figura.



16. Elige un intervalo adecuado y representa las siguientes curvas dadas por su ecuación polar:

a) $\rho = \frac{1}{\theta}$

b) $\rho = \frac{0.1}{1+\theta^2}$

c) $\rho = \frac{\theta}{1+\theta^2}$

d) $\rho = \log\theta$

e) $\rho = \frac{\log\theta}{\theta}$

f) $\rho = \frac{1}{1+e\cdot\cos\theta}$, para distintos valores de e.

2.5 Soluciones

4. Llamamos c_j al saldo a principio del periodo j.

Obviamente, $c_1 = a_1$, porque al principio no se han producido intereses.

Al cabo del primer periodo, se acumula a c_1 el interés producido, $c_1 r$, y la segunda imposición, a_2. El total es

$$c_2 = c_1(1+r) + a_2$$

Durante el segundo periodo se producen intereses por un importe de $c_2 r$ y, al final, se produce la tercera imposición, a_3, dando un total de

$$c_3 = c_2(1+r) + a_3$$

y así sucesivamente hasta la última imposición, a_n, con lo que al inicio del periodo n habrá acumulado la cantidad

$$c_n = c_{n-1}(1+r) + a_n$$

Los cálculos en MATLAB pueden efectuarse mediante la función siguiente:

```
function c = ahorro(a,r)

%AHORRO   Formación de capital a interés compuesto.
%
%  c = ahorro(a,r)
%
%  a: vector de cantidades ingresadas al inicio de cada
periodo
%  r: tipo de interés en tanto por uno por periodo
%  c: vector de capitales acumulados al inicio de cada periodo

n = length(a);
c(1) = a(1);
for k = 2:n
    c(k) = c(k-1)*(1+r) + a(k);
end
```

Para hallar la gráfica pedida, aplicamos esta función a los datos del problema y usamos la orden bar.

```
a = [100, 25, 80, -45, 125, 30, 70];
r = 0.01;
c = ahorro(a,r);
bar(c)
title('Interés compuesto')
xlabel('Periodo')
ylabel('Capital')
```

Nota: Este algoritmo obtiene el cociente y el resto de la división del polinomio cuyos coeficientes son los elementos del vector a entre $x-(1+r)$. Los primeros elementos de c contienen el cociente y el último, c(n) es el resto. Constituye la llamada Regla de Ruffini o algoritmo de Horner. Con MATLAB, este resultado se puede obtener mediante la expresión

```
[cociente, resto] = deconv(a, [1, -(1+r)])
```

.

5. a) En el algoritmo clásico, los cálculos se disponen de la siguiente forma

$$\begin{array}{rl|l} x^3 & -2x-5 & x-2 \\ \cline{3-3} -x^3+2x^2 & & x^2+2x+2 \\ \cline{1-1} \quad 2x^2-2x & & \\ -2x^2+4x & & \\ \cline{1-1} \quad 2x-5 & & \\ -2x+4 & & \\ \cline{1-1} \quad -1 & & \end{array}$$

El cociente es x^2+2x+2 y el resto -1.

b) Al ser el denominador de la forma $x-a$, se obtiene el mismo resultado por Ruffini, trabajando sólo con los coeficientes.

$$\begin{array}{r|cccc} & 1 & 0 & -2 & -5 \\ 2 & & 2 & 4 & 4 \\ \hline & 1 & 2 & 2 & -1 \end{array}$$

Los coeficientes del cociente son 1, 2, 2 y el resto es -1.

c)
```
p = [1 0 -2 -5];
q = [1 -2]];
[c,r] = deconv(p,q)
```

d) $p(2)=2^3-2\cdot 2-5=8-4-5=-1.$

e)
```
polyval(p,2)
```

6. El polinomio tiene a 1 como raíz de orden 6. Por tanto será (salvo factor constante que en este caso es 1)

```
poly([1 1 1 1 1 1])
```

8.c) El camino más corto es aplicar Ruffini reiteradamente

$$\begin{array}{r|cccc} & 5 & -1 & 3 & 1 \\ 1 & & 5 & 4 & 7 \\ \hline & 5 & 4 & 7 & 8 \\ 1 & & 5 & 9 & \\ \hline & 5 & 9 & 16 & \\ 1 & & 5 & & \\ \hline & 5 & 14 & & \end{array}$$

Los restos son los coeficientes de las sucesivas potencias de $x-1$:

$$5x^3 - x^2 + 3x + 1 = 5(x-1)^3 + 14(x-1)^2 + 16(x-1) + 8.$$

9. Dividimos por 2 sucesivamente y tomamos los restos en orden inverso:

$$\begin{array}{r|l} 53 & 2 \\ \hline \underline{1} \quad 26 & 2 \\ \hline \underline{0} \quad 13 & 2 \\ \hline \underline{1} \quad 6 & 2 \\ \hline \underline{0} \quad 3 & 2 \\ \hline 1 \quad 1 & \end{array}$$

Por tanto, $53 = 110101_2$.

10. La función siguiente genera vectores con los sumandos del desarrollo y los representa sucesivamente en el mismo gráfico:

```
function prob08(n)
x = linspace(-1,1);
y = exp(x);
plot(x,y,'r'), hold on,pause
yk = ones(size(x));
ck = 1;
plot(x,yk),pause
for k = 1:n
    ck = ck*k;
    yk = yk + x.^k/ck;
    plot(x,yk), pause
end
```

14. a) Con MATLAB. Calculamos en primer lugar las raíces del polinomio:

```
r = roots([1 1 1 1 1 1])
```

Obtenemos:

```
r =
   0.5000 + 0.8660i
   0.5000 - 0.8660i
  -1.0000
  -0.5000 + 0.8660i
  -0.5000 - 0.8660i
```

Observamos que las dos primeras son complejas conjugadas, por lo que dan lugar a un factor irreducible de segundo grado que podemos obtener con

```
p1 = conv([1 -r(1)], [1 -r(2)])

p1 = 1.0000   -1.0000    1.0000
```

O sea $p_1(x) = x^2 - x + 1$. La tercera raíz, -1, corresponde al factor $p_2(x) = x + 1$.

Las dos últimas raíces forman otro par conjugado y, operando como antes tenemos

```
p3 = conv([1 -r(4)], [1 -r(5)])

p3 = 1.0000    1.0000    1.0000
```

Es decir, $p_3(x) = x^2 + x + 1$. En conclusión, la descomposición pedida es:

$$x^5 + x^4 + x^3 + x^2 + x + 1 = (x^2 - x + 1)(x + 1)(x^2 + x + 1).$$

b) Sin MATLAB. Podemos recordar la fórmula de la suma de términos de una progresión geométrica o tener la feliz idea de multiplicar el polinomio dado por $x - 1$. De las dos maneras se llega a que dicho polinomio tiene las mismas raíces que $x^6 - 1$, salvo la raíz $x = 1$.

Tales raíces son $x_k = \cos\frac{k\pi}{3} + \mathrm{i}\sin\frac{k\pi}{3}$, para $k = 0, 1, \ldots, 5$. Eliminando $x_0 = 1$, quedan las raíces halladas en el apartado anterior y se puede operar como antes, pero haciendo a mano los cálculos.

3 Representación gráfica de funciones (II)

Muchos procesos de la vida real pueden expresarse mediante funciones elementales y combinaciones de estas. Los científicos, economistas y otros investigadores estudian relaciones entre cantidades. Por ejemplo, un bioquímico investigará el crecimiento de una población sometida a determinadas condiciones. Un economista estará interesado en estudiar los beneficios por la fabricación de un producto. Un ingeniero necesita saber la relación entre el peso que puede soportar un objeto y la deformación que en él se puede producir. Todas estas relaciones generalmente se formulan matemáticamente mediante una expresión algebraica que se conoce con el nombre de función.
Vamos a estudiar en este tema la representación gráfica de varios tipos de funciones, ya que disponer de la representación gráfica de una función proporciona información, que puede no ser evidente en los tratamientos algebraicos, sobre el fenómeno o proceso que se desea analizar.

Hemos utilizado ya algunas órdenes gráficas de MATLAB para representar expresiones algebraicas, `plot` para representar polinomios, `compass` para representar números complejos y `polar` para representar funciones a partir de su expresión en coordenadas polares.
Veremos algún comando más para representación de funciones en general, curvas en dos y tres dimensiones, coordenadas cartesianas, ecuaciones paramétricas, coordenadas cilíndricas y esféricas.

3.1 Funciones reales de una variable

Una función *f* real de variable real es una aplicación de *D* en $\mathbb{R}$ que asocia a cada elemento $x \in D \subseteq \mathbb{R}$ un numero real *y*, *D* es el dominio de la función.
Para representar geométricamente una función $y = f(x)$ como una gráfica, es tradicional usar el sistema de coordenadas cartesianas, con las unidades para la variable independiente *x* en el eje horizontal y las de la variable dependiente *y* en el eje vertical.
Gráfica de una función:
La gráfica de una función *f* es el conjunto de todos los puntos del plano cuyas coordenadas (x, y) verifican $y = f(x)$ siendo *x* del dominio de *f*.

En general en **MATLAB** utilizaremos `plot` para representar pares de valores de una función.
La orden `plot(x,y)` representa los pares $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, siendo el vector $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ del dominio y el vector $\boldsymbol{y} = (y_1, y_2, ..., y_n)$ las imágenes correspondientes. Por defecto, plot une puntos consecutivos mediante un segmento, por lo que la representación será más perfecta cuantos más puntos pongamos.

Ejemplo 1

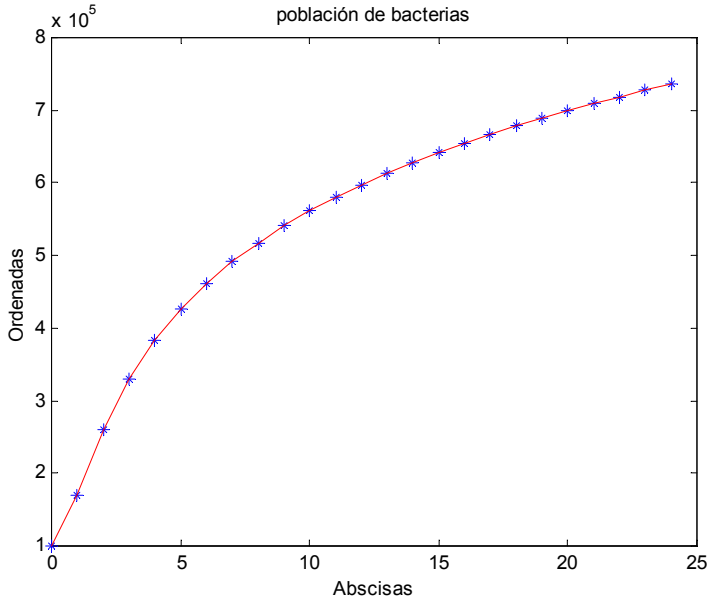
Una población de 100.000 bacterias se introduce en un cultivo, siendo su número al cabo de x horas:

$$f(x) = 10^5(1 + \log(x^2 + 1))$$

para estudiar la influencia de este cultivo en el crecimiento de la población conviene representar la función $f(x)$.

```
x=0:24;
```

```
y=1e5*(1+log(x.^2+1));
plot(x,y)
```
Observamos que la población a lo largo del día sigue multiplicándose pero cada vez de forma menos acusada Añadiendo un tercer argumento a `plot`, modificamos el color y el estilo del trazo. Así, con
```
plot(x,y,'*')
```
visualizamos los puntos que determinan la función del ejemplo anterior. Para trazar la gráfica de la función de color rojo, por ejemplo, en el mismo gráfico hacemos
```
plot(x,y,'*',x,y,'r')
```
Con `help plot` vemos las distintas combinaciones de caracteres válidas como tercer argumento de esa función.
La orden `title` sirve, como su nombre indica, para poner un título al gráfico. Igualmente podemos poner títulos en los ejes con `xlabel` e `ylabel`. El argumento de estas funciones ha de ser una cadena o variable de este tipo.
```
title('población de bacterias'
xlabel('Abscisas'), ylabel('Ordenadas')
```



Ejemplo 2

Representa la función de beneficios de una empresa si al vender un producto hasta un tope de *5000* unidades estos viene dados por $B(x)=x^3 - 9x^2 + 24x$, donde x son miles de unidades.
```
x=0:.1:5;y=x.^3-9*x.^2+24*x; plot(x,y )
```
Podemos añadir cualquier texto sobre el gráfico sin más que indicar la posición en que debe aparecer, bien mediante coordenadas o con el ratón. Para escribir la palabra Máximo en el punto de coordenadas *(2, 20)* utilizamos
```
text(2,20,'Máximo')
```
También podemos situar el texto con el ratón
```
gtext('Mínimo')
```

Las instrucciones interactivas, como pause, gtext, etc. para ser ejecutadas las tienes que probar en MATLAB, no se pueden ejecutar desde Word.

3.2 Curvas en 2D

Una aplicación continua *f* definida en un intervalo de la recta real $\mathbb{R}$ y con valores en el plano $\mathbb{R}^2$ se llama *curva* en dos dimensiones.

Ecuaciones paramétricas
Supongamos que al número real *t* corresponde el punto *f(t)*, cuyas coordenadas respecto a un sistema de referencia son *(x(t), y(t))*. Entonces la curva f se puede describir como la aplicación que a cada t de IR hace corresponder el punto *(x, y) = (x(t), y(t))*. Se dice que:

$$x = x(t)$$
$$y = y(t) \qquad t \in \mathbb{R}$$

son las ecuaciones paramétricas de la curva, *t* es el parámetro, en muchas ocasiones representa el tiempo y así tenemos la posición de un objeto a lo largo del tiempo.

Ejemplo 3

La curva definida por la aplicación que a cada número real *t* le hace corresponder el punto *(x, y)* dado por:

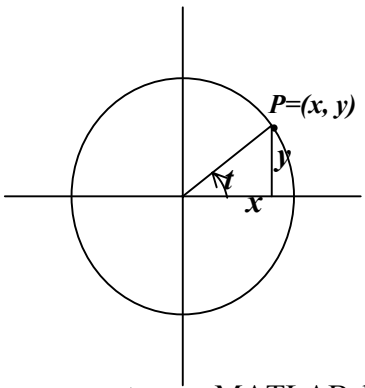
$$x = x_0 + t\, v_1$$
$$y = y_0 + t\, v_2$$

es una recta en el plano que pasa por el punto (x_0, y_0) y es paralela al vector (v_1, v_2).
Para representar la recta que pasa por el punto *(2, -1)* y tiene la dirección del vector *(3, -2)* hacemos:

```
t=0:3;v=[3,-2]; x=2+t*v(1); y=-1+t*v(2); plot(x,y)
```

realmente representamos el segmento de recta comprendido entre (2, -1) y (11, -7).

3.2.1 **Circunferencia**
La circunferencia representada por ecuación cartesiana por $x^2 + y^2 = r^2$ se expresa en ecuaciones paramétricas de la siguiente forma:



$x = r \cos t$
$y = r \sin t$

Con lo que para representar en MATLAB la circunferencia de centro (0,0) y radio 1 basta hacer:

```
t = 0:.01:2*pi; r=1; x=r*cos(t);y=r*sin(t); plot(x,y),axis equal
```

La orden `hold on` hará que la próxima gráfica se inserte en la misma figura, para dibujar un circunferencia concéntrica con la anterior pero de menor radio, hacemos:

```
hold on, r=0.5; x=r*cos(t);y=r*sin(t); plot(x,y),hold off
```

Recuerda que debes hacer hold off para desactivar el comando hold

En general la circunferencia de centro *(a, b)* y radio *r* , que en coordenadas cartesianas será

$$(x-a)^2+(y-b)^2=r^2$$

tendrá por ecuaciones paramétricas:

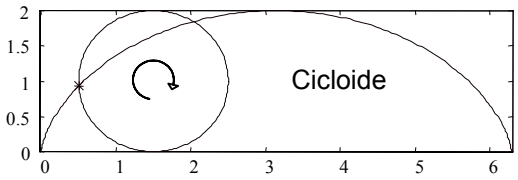
$$x = a + r\cos t$$
$$y = b + r\sin t$$

Prueba a representar por ejemplo la circunferencia de centro (-2, 3) y r = 2;

3.2.2 **Cicloide**

Las *ecuaciones paramétricas* de una curva aparecen frecuentemente en la descripción de trayectorias de móviles, cuya posición en el plano viene dada por dos funciones ***(x(t), y(t))*** del tiempo, ***t***, que es el parámetro. Por ejemplo, se llama **cicloide** a la trayectoria de un punto de una circunferencia que rueda sin deslizarse sobre una recta, se obtiene con:

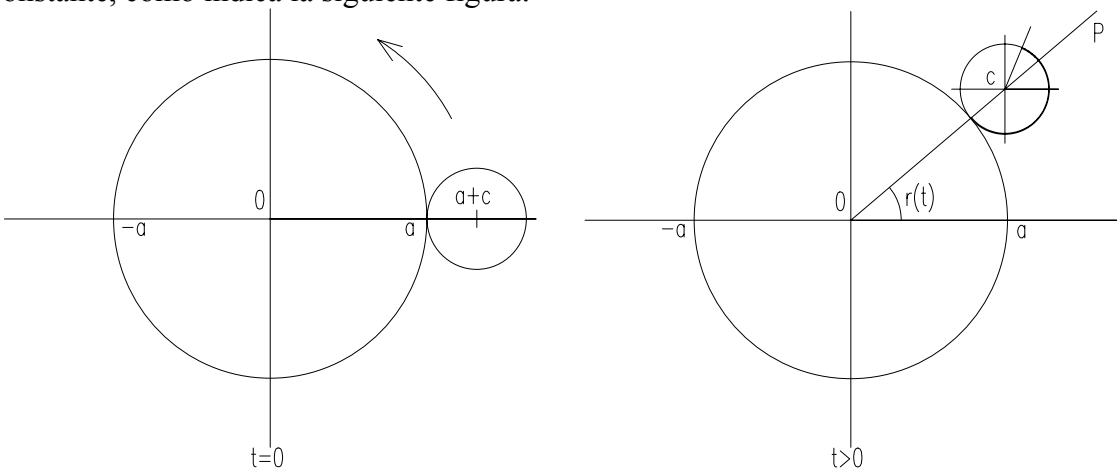
```
t = 0:pi/100:2*pi;
y = 1-cos(t);
x = t-sin(t);
plot(x,y)
axis equal
```



3.2.3 **Ruletas**

Las ruletas son una variedad de cicloides, se obtienen por el desplazamiento de una circunferencia sobre otra.

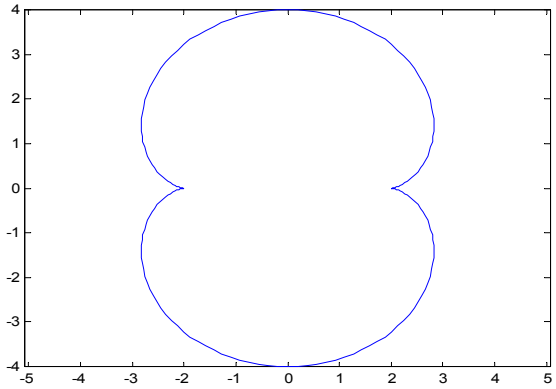
Se trata de obtener la trayectoria de un punto fijo *P* de una circunferencia de radio *c*, que es tangente exterior a otra circunferencia fija de radio *a* y rueda sobre ella con velocidad constante, como indica la siguiente figura:



Si en el momento inicial el punto de contacto de ambas circunferencias es *(a, 0)* y al cabo de un tiempo *t* suponemos que la circunferencia que se desliza recorre un ángulo la ecuación de la trayectoria es:

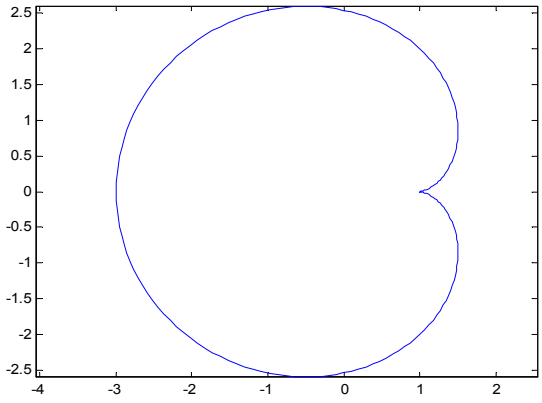
```
t = 0:pi/100:2*pi; a=2; c=1;
x=(a+c)*cos(t)-c*cos((a+c)*t/c);
y=(a+c)*sin(t)-c*sin((a+c)*t/c);
```

```
plot(x,y)
axis equal
```

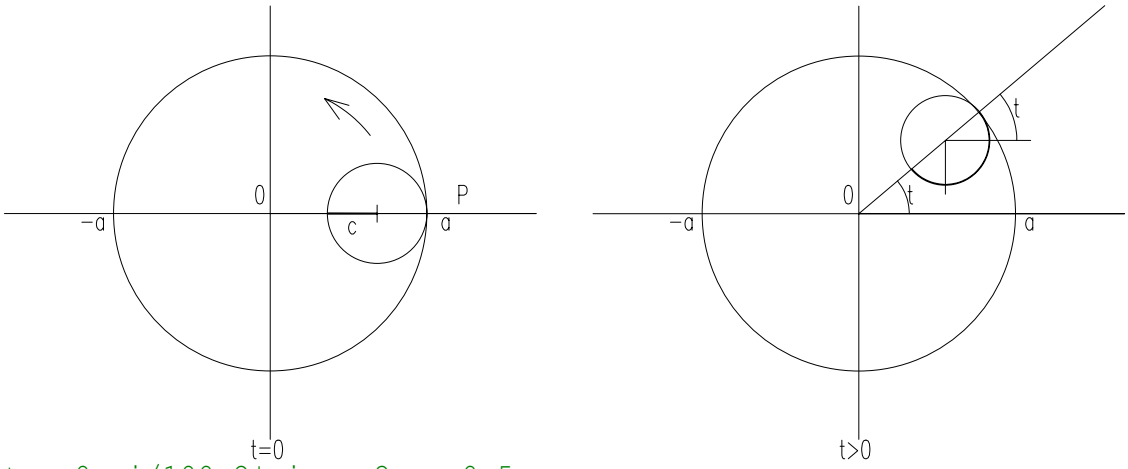


La curva obtenida se llama epicicloide.
Represéntala para valores de los parámetros $a = 2$; $c = 0.5$, y comprueba que para $a = c$ la curva obtenida es una cardioide, curva que ya vimos en coordenadas polares.

```
t = 0:pi/100:2*pi; a=1; c=1;
x=(a+c)*cos(t)-c*cos((a+c)*t/c);
y=(a+c)*sin(t)-c*sin((a+c)*t/c);
plot(x,y)
axis equal
```



Si suponemos que la circunferencia que se desliza es tangente interior a la primera tenemos una ruleta distinta que se denomina **hipocicloide**.




```
t = 0:pi/100:2*pi; a=2; c=0.5;
x=(a-c)*cos(t)+c*cos((a-c)*t/c);
y=(a-c)*sin(t)-sin((a-c)*t/c);
plot(x,y)
axis equal
```

En el caso de *c=a/2* la curva que se obtiene es un astroide. Dibújala.

3.3 Funciones reales de dos variables

Una función *f* real de dos variables es una aplicación de *D* en $\mathbb{R}^2$ que asocia a cada elemento $(x, y) \in D \subseteq \mathbb{R}^2$ un número real *z*; *D* es el dominio de la función.

Para representar geométricamente una función *z = f(x,y)* como una gráfica, es tradicional usar el sistema de coordenadas cartesianas de $\mathbb{R}^3$ con las unidades para la variables independientes *x* e *y* en el plano horizontal *XY* y las de la variable dependiente *z* en el eje vertical.

Gráfica de una función:
La gráfica de una función *f* es el conjunto de todos los puntos del espacio cuyas coordenadas *(x, y, z)* verifican *z = f(x,y)* siendo *(x, y)* del dominio de *f.*

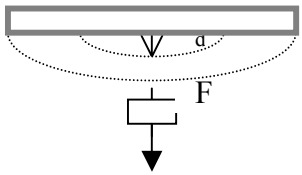
La idea básica para representar funciones de dos variables es evaluarlas en un conjunto de puntos del plano *XY* formando una malla rectangular, almacenar estos valores en una matriz *z* y utilizar órdenes de MATLAB que representan estos valores como alturas sobre el plano *XY*.

Ejemplo 4

La elasticidad por flexión es el fenómeno de deformación de un cuerpo, por efecto de una fuerza proporcional a su dimensión mayor, y el sólido se deforma de tal modo que el sistema de laminas planas paralelas se encorvan formando un haz de superficies curvas.
La deformación *d* para una barra de longitud *l* sometida a una fuerza transversal F aplicada en su centro tiene por valor:

$$d = \frac{F}{4E}\frac{l^3}{ah^3}$$

siendo *F* el peso o fuerza que se aplica, *l, a, h*: longitud, anchura y altura de la barra y *E* el módulo de elasticidad de Young.

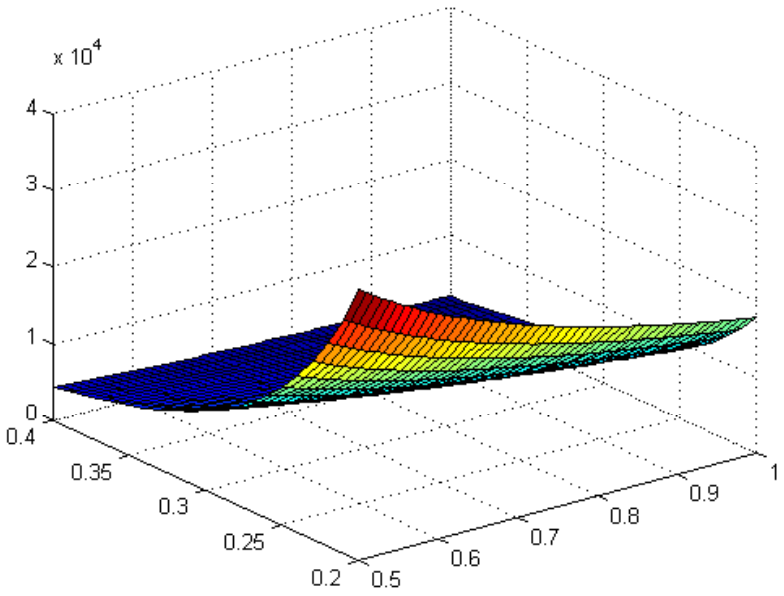


Consideremos láminas de cobre de longitud constante 10 cm y de anchura a y grosor h variables, sobre las que se aplica una fuerza de $5.1x10^8$ dinas. El modulo de Young del cobre es de $9.0x\ 10^{11}$ dinas/cm^2. La deformación queda como una función de dos variables, la anchura y grosor de las láminas.

Representación de d en función de a y h.

Damos a la anchura a valores entre 5 y 10 mm y a la altura h entre 2 y 4 mm. observamos que si calculamos directamente d obtenemos un error de dimensiones, MATLAB nos ayuda con la orden meshgrid, que a partir de un conjunto de valores de a y otro de h, crea dos matrices, *A* y *H*, con las coordenadas de los puntos de la malla. La matriz d se obtiene evaluando la función a representar sobre el par de matrices obtenidas. Hay que tener cuidado en emplear las versiones 'punto' de las operaciones producto, cociente y potencia, pues la función actúa elemento a elemento.

```
a = 0.5:0.01:1; h = 0.2:.01:0.4;
F =5.1e8;
 E=9e11;
 [A,H]=meshgrid(a,h);% ¡Truco!
 d=F*100^3./(4*E.*A.*H.^3);
 surf(a,h,d)
```



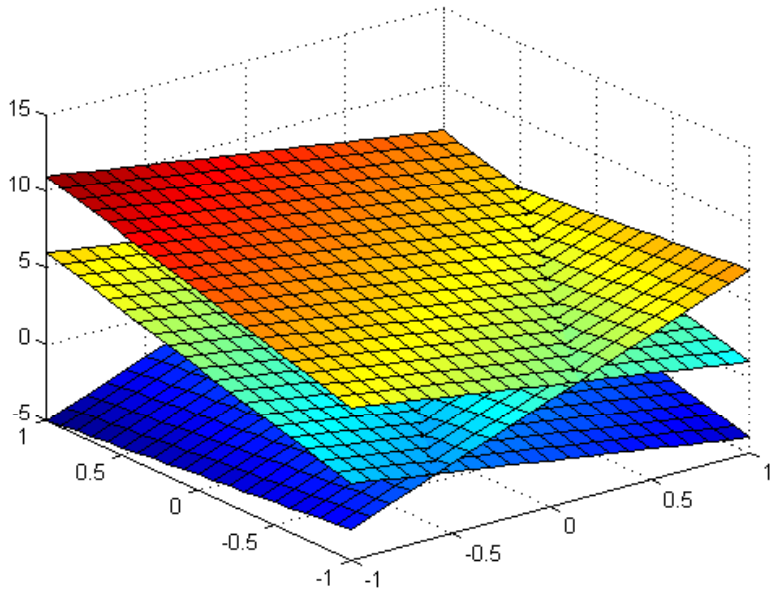
Ejemplo 5

Sabemos que una ecuación lineal en tres dimensiones representa un plano en el espacio $\mathbb{R}^3$, representa el plano $2x-3y+z=1$

```
 x = -1:0.1:1; y = -1:.1:1;
[X,Y] = meshgrid(x,y);
z=1+3*Y-2*X;
surf(x,y,z)
```

Representemos en la misma figura un plano paralelo a éste y otro que lo corte.

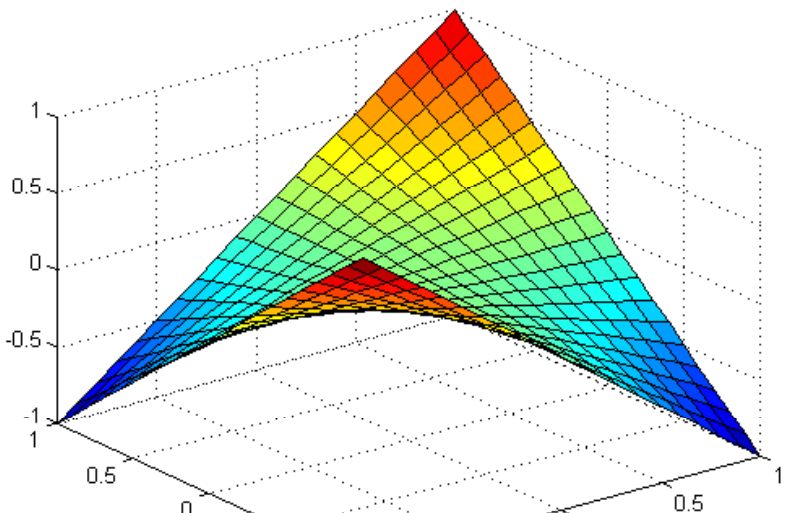
```
hold on
z=6+3*Y-2*X; %Plano paralelo, con el mismo vector característico
surf(x,y,z)
hold on
z=1-Y+5*X;
surf(x,y,z)
hold off
```



3.3.1 **Paraboloide hiperbólico**

La función $z = xy$ es la ecuación de una superficie llamada *paraboloide hiperbólico*. La representamos en el rectángulo $[-1,1]\times[-1,1]$, en este caso es más fácil disponer los valores de la función en una matriz **z** de modo que la posición en la matriz corresponda con las coordenadas del punto *(x,y)*, y podemos prescindir de la instrucción meshgrid.

```
x = -1:0.1:1; y = -1:.1:1;
z = y'*x;  % Columna×Fila=Matriz cuadrada
surf(x,y,z)      % Superficie de placas
```



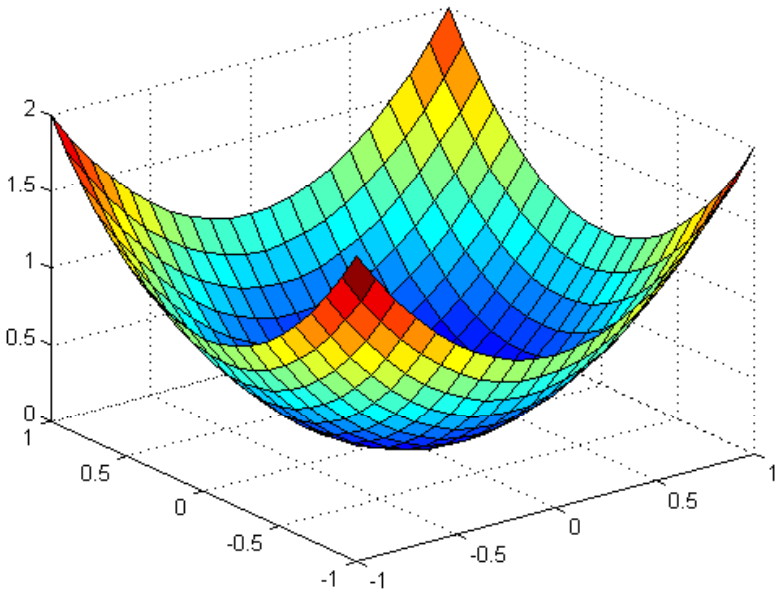
Puedes utilizar también la instrucción mesh para obtener el armazón de la superficie

```
mesh(x,y,z)      % Armazón de la superficie
```

3.3.2 **Paraboloide elíptico**

Representemos el paraboloide elíptico $z = x^2 + y^2$ *en* $[-1,1]\times[-1,1]$

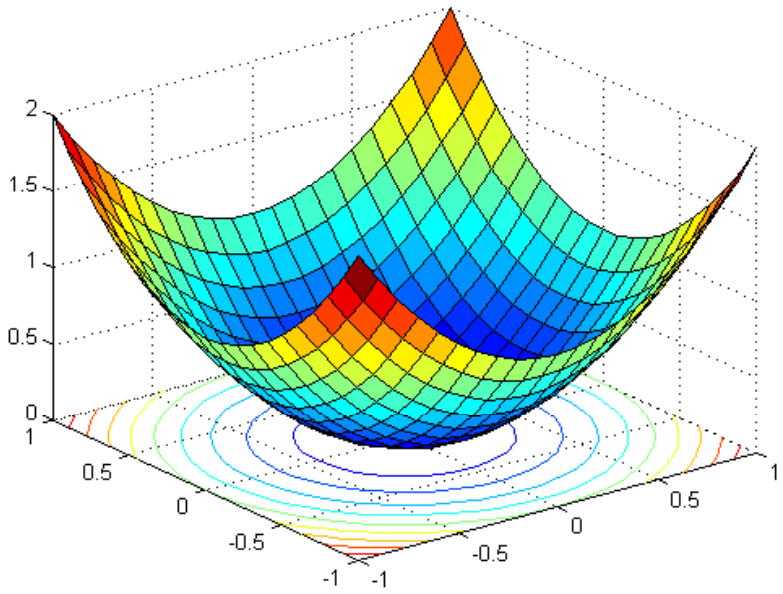
```
x = -1:0.1:1; y = -1:.1:1;
[X,Y] = meshgrid(x,y);           % ¡Truco!
z = X.^2 + Y.^2;            % !Ojo al punto!
surf(x,y,z)
```



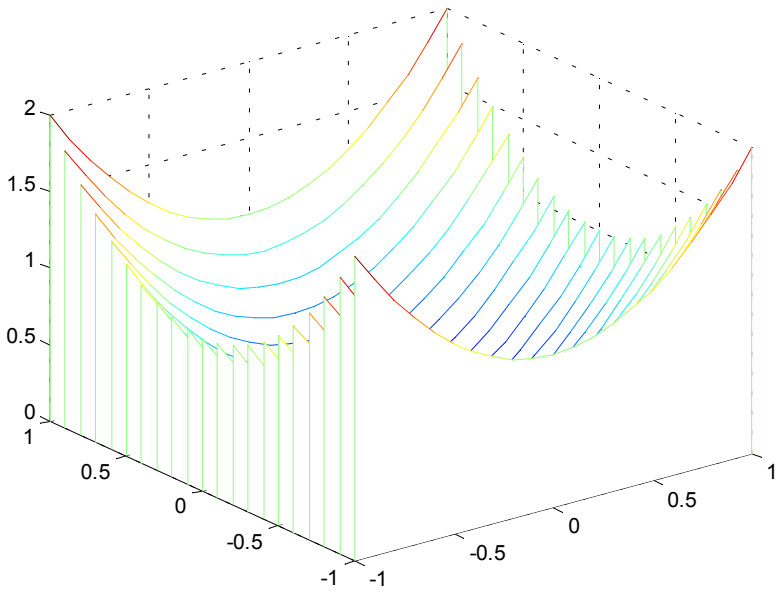
3.4 Curvas y tonos de nivel

Se llaman curvas de nivel de una función $z = f(x,y)$ a la curva del plano XY para la que la función toma un valor constante $f(x,y) = c$ ó $z = c$;
Por ejemplo las curvas de nivel del paraboloide elíptico son $x^2 + y^2 = c$, representan pues circunferencias en el plano XY, se pueden obtener en MATLAB con:

```
contour(x,y,z)   % Curvas de nivel
surfc(x,y,z) % Superficie y curvas de nivel
```




```
waterfall(x,y,z) % Cortes verticales
```




```
contourf(>
```

El aspecto de la superficie puede modificarse de varias maneras.
con surfl se ilumina la superficie desde una fuente de luz. Prueba las siguientes opciones

```
surfl(x,y,z,[0.5,0.5,0])%foco de luz en la posición [0.5,0.5,0])
surfl(x,y,z,[0,0,1])
surfl(x,y,z,[10,10,10])
surfl(x,y,z,[0.5,0.5,1])
```

Para crear una malla transparente hacemos

```
mesh(x,y,z), hiden off
```

Con hidden on volvemos al estado normal en el que se ocultan las líneas posteriores.
Las opciones de sombreado muestran u ocultan la malla sobre la superficie.

```
surf(x,y,z), shading flat  % sin malla
shading interp             % color degradado
shading faceted            % losetas con borde
```

El color se cambia con colormap que tiene algunas opciones predefinidas (ver la ayuda de graph3)

```
colormap pink
```

3.4.1 Sombrero

la función $z= 3 \cos((x^2+y^2)/3)/(3+x^2+y^2)$ se denomina *sombrero*, vamos a representarla:

```
x = -5:0.1:5; y = -5:.1:5;
[X,Y] = meshgrid(x,y);
z = 3*cos((X.^2 + Y.^2)/3)./(3+X.^2 + Y.^2);, % !Ojo al punto!
surf(x,y,z), hold on
ccontour3(x,y,z,'r')
```

Podemos ver distintas posiciones del sombrero, ya que el punto de vista de un gráfico en 3D puede modificarse interactivamente con el ratón, haciendo

```
rotate3d %
```

Recuerda que las instrucciones interactivas las tienes que probar directamente en Matlab, no a través de Word.

La opción `view(x,y,z)` nos permite especificar el punto de vista de la superficie

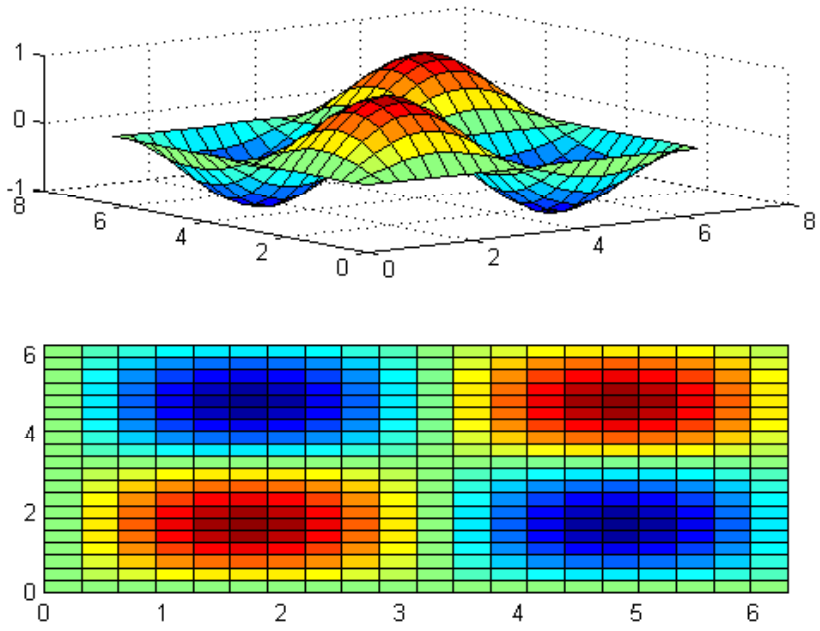
```
view([5,5,-5])%(x,y,z) posicion en coordenadas cartesianas.
view(0,0)   %(h,v) h rotación horizontal, v elevación vertical
```

La función `pcolor` muestra un rectángulo coloreado según el valor que se representa. Vamos a utilizar la orden `subplot` para crear varias gráficas en una misma figura. subplot (m,n,i) mxn gráficas distribuidas en m filas y n columnas, i gráfica que vamos a introducir.

Ejemplo 6

Representa la función $z = sen\ x\ cos\ y$ en $[0,2\pi]\times[0,2\pi]$

```
x = 0:pi/10:2*pi; y = x;
[X,Y] = meshgrid(x,y);
z = sin(X).*sin(Y);
subplot(2,1,1),surf(x,y,z)  % dos gráficas distribuidas en dos
filas y una columna,surf(x,y,z)% primera gráfica
subplot(2,1,2),pcolor(x,y,z),% segunda gráfica
```



Jugando con las opciones de sombreado y añadiendo curvas de nivel en negro, creamos un mapa físico

```
hold, shading interp, contour(x,y,z,'k'), hold
```

Si en cambio queremos mostrar las curvas de nivel sobre la superficie, hacemos lo mismo con surf y contour3

```
surf(x,y,z), shading interp, hold
 contour3(x,y,z,'k'), hold
```

3.5 Curvas en 3D

Al igual que en el plano, la trayectoria un móvil en el espacio puede definirse mediante sus ecuaciones paramétricas, ***(x(t), y(t), z(t))***, donde ***t*** varía en un intervalo y **x, *y*** y ***z*** son las coordenadas del móvil en función de ***t***.
En tres dimensiones la ecuación de las ecuaciones paramétricas de la recta serán:

$$x = x_0 + t\, v_1$$
$$y = y_0 + t\, v_2$$
$$z = z_0 + t\, v_3$$

Por ejemplo la que pasa por el punto *(0, -2,-1)* y tiene la dirección del vector *(3,2, -1*) será:

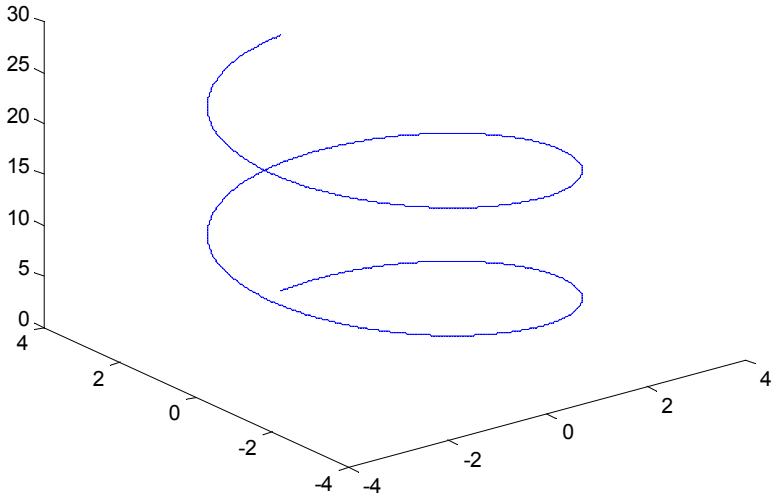
```
t=-1:.01:1;p=[0,-2,-1];v=[3,2,-1];x=p(1)+t*v(1);y=p(2)+t*v(2);
z=p(3)+t*v(3);plot3(x,y,z)
```

3.5.1 **Hélice**
Si ***x*** e ***y*** describen una circunferencia y ***z*** avanza linealmente, tenemos la trayectoria de una hélice, sus ecuaciones paramétricas serán:

$$x = r\cos t$$
$$y = r\sin t$$
$$z = k\, t$$

La representamos con:

```
t = 0:pi/500:4*pi;r=3; k=2;
x = r*sin(t); y =r* cos(t); z = k*t;
plot3(x,y,z)
```



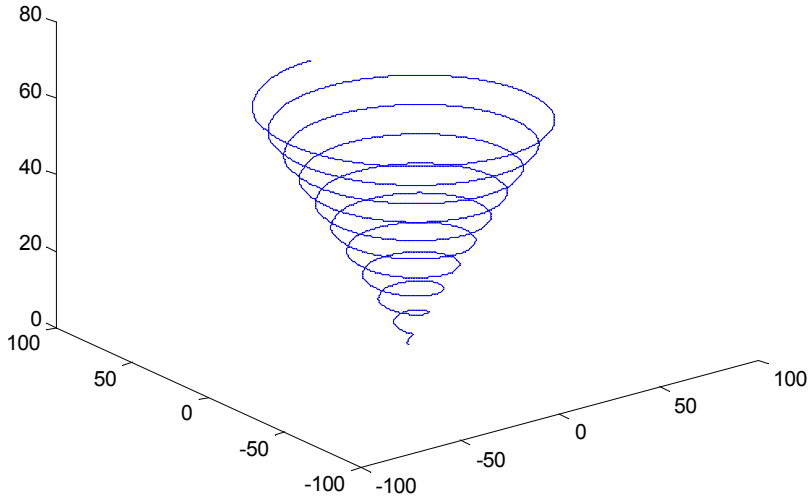
A los puntos t_1 y $t_1 + 2\pi$ corresponden los puntos $M_1 = (r\cos t_1, r\sin t_1, ht_1)$ y $M_2 = (r\cos t_1, r\sin t_1, ht_1 + 2\pi h)$, la distancia entre estos dos puntos $2\pi|h|$ se llama paso de la hélice.
Dibuja dos hélices entrelazadas con

```
t = 0:pi/500:4*pi;
x = sin(t); y = cos(t); z = t;
```

```
plot3(x,y,z,'r',-x,-y,-z,'g')
```

Para dibujar una trayectoria helicoidal sobre una superficie cónica haz:

```
t = 0:pi/400:20*pi;
x = t.*sin(t); y = t.*cos(t);
plot3(x,y,t)
```



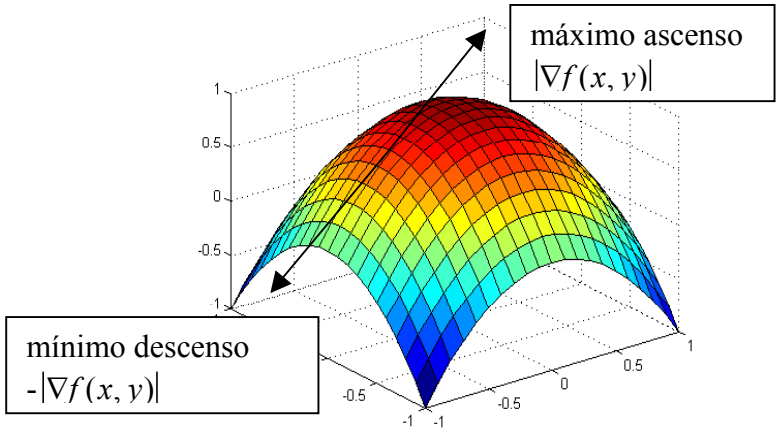
3.6 Líneas de máxima pendiente

Dada una función de dos variables $z = f(x,y)$ podemos obtener su vector gradiente $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$ evaluado en un conjunto de puntos y representarlo junto a las curvas de nivel. Notemos que el gradiente en cada punto tiene la dirección de máxima variación de la función y es perpendicular a las curvas de nivel de la superficie.

$$f(x,y) = 1 - (x^2 + y^2)$$

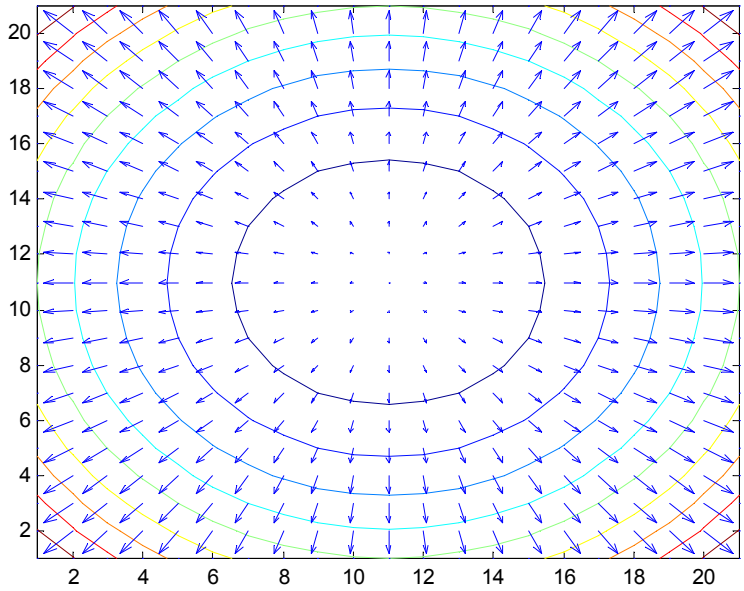
$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) = (-2x, -2y)$$

$$\nabla f(x_0, y_0) = (-2x_0, -2y_0)$$



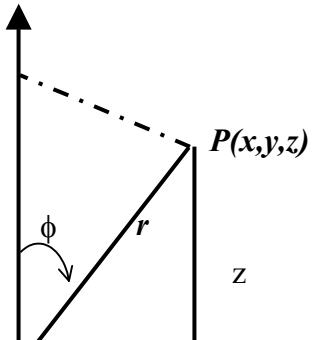
veámoslo en el caso del paraboloide elíptico:

```
x = -1:0.1:1; y = -1:.1:1;
[X,Y] = meshgrid(x,y);          % ¡Truco!
z = X.^2 + Y.^2;
[dx,dy] = gradient(z,.1,.1);%Obtiene el gradiente de z en los
puntos de la malla [X,Y].
contour(z),hold on,
quiver(dx,dy),%representa los vectores (dx,dy) mediante flechas;
hold off
```



3.7 Coo

Si las coordenadas x e y de un punto $P(x,y,z)$ del espacio se sustituyen por las coordenadas polares r y θ, la terna r, θ, z se denominan coordenadas cilíndricas del punto P. El número no negativo r representa ahora la distancia del punto P al eje z tal como se indica en la figura. Los puntos del espacio para los que r es constante equidistan del eje z y por tanto pertenecen a un cilindro circular (de aquí el nombre de coordenadas cilíndricas).



$y=r\ sin\ \theta$

$z=z$

Paso de c.cartesianas a c. cilindrícas

$r=\sqrt{x^2+y^2}$

$\text{tg}\,\theta=\frac{y}{x}$

$z=z$

Paso de c. cilindrícas a c.cartesianas

$x=r\ cos\ \theta$

Las coordenadas cilíndricas son muy convenientes para representar cilindros y superficies de revolución en general, para las cuales el eje z sea un eje de simetria.

Cilindro $x^2 + y^2 = a^2$ coordenadas cartesianas r = a coordenadas cilíndricas.

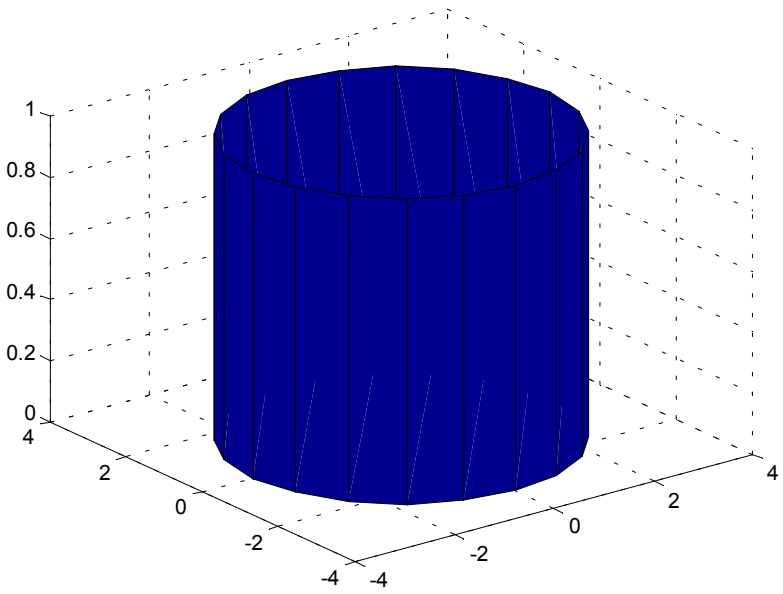
Cono $x^2 + y^2 = z^2$ coordenadas cartesianas r = z coordenadas cilíndricas

Generar superficies de revolución es muy fácil con MATLAB. Basta crear un vector con los radios de la superficie a intervalos regulares del eje de revolución y utilizar la función `cylinder`.

```
r=1; cylinder(r)%cilindro x^2+y^2=1, ó r=1 en c. cilíndricas
```

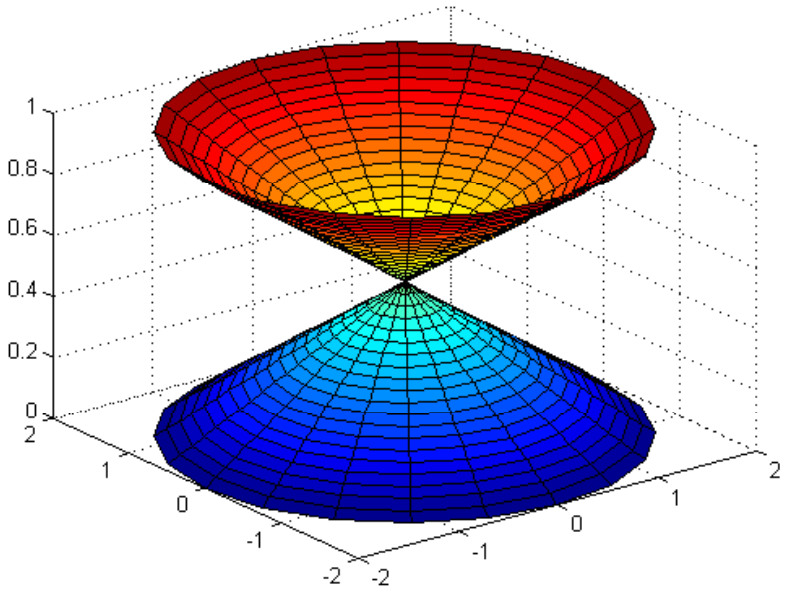
Por lo que un cilindro de radio 3 lo representaremos con:

```
 rr=3;,cylinder(rr);
```



Para el cono el vector de radios será variable:

```
r=-2:.1:2; cylinder(r)
```



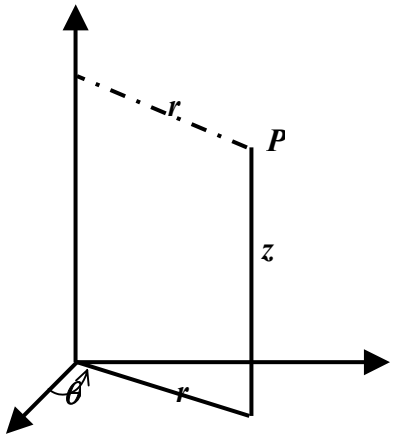
3.8 Coordenadas esféricas

Un punto *P(x,y,z)* del espacio se representa en coordenadas esféricas mediante la terna *(r, θ, ϕ)* de modo que:

r es la distancia del origen al punto P.

θ es el ángulo polar.

ϕ es ángulo, que forma el semieje positivo z con la semirrecta de origen el de coordenadas y que contiene a *P*. $0 \leq \phi \leq \pi$



Paso de c. esféricas a c. cartesianas

$x = r \sin\phi \cos\theta$

$y = r \sin\phi \sin\theta$

$z = r \cos\phi$

Paso de c.cartesianas a c. esféricas

$$r = \sqrt{x^2 + y^2 + z^2}$$

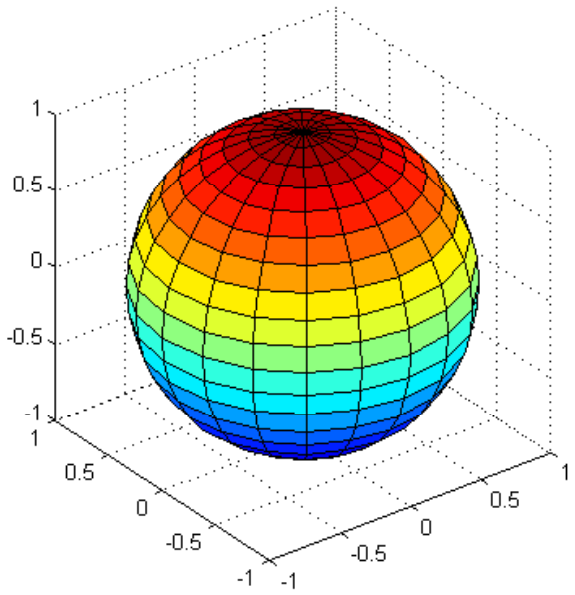
$$\cos\theta = \frac{y}{\sqrt{x^2 + y^2}}$$

$$\cos\phi = \frac{z}{\sqrt{x^2 + y^2 + z^2}}$$

La esfera $x^2 + y^2 + z^2 = a^2$ en coordenadas esféricas es $\phi = a$

En MATLAB puedes dibujar la esfera de radio 1 simplemente con:

```
sphere, axis equal
```



4 Problemas

1.-Las ecuaciones paramétricas de una elipse son:

$$x = a \cos t$$
$$y = b \sin t$$

donde *a* y *b* son los semiejes. Representa la elipse de ecuaciones cartesianas, $\frac{x^2}{9}+\frac{y^2}{16}=1$, pasándola previamente a ecuaciones paramétricas y obtén en la misma gráfica la hipérbola $x\,y = 1$ para obtener una aproximación de los punto de corte de ambas cónicas.

2.- La función $y(t) = 1\text{-}e^{-bt}$, donde t es el tiempo y $b>0$, describe muchos procesos de ingeniería, como la temperatura de un objeto que esta siendo calentado etc. Investiga el efecto del parámetro *b* en la función *y(t),* representando *y* para varios valores de *b* en la misma gráfica. ¿Al cabo de cuánto tiempo *y(t)* alcanzara el *98%* de su valor asintótico?.

3.- Un dipolo eléctrico consiste en dos cargas opuestas separadas por una distancia *d*. El potencial eléctrico en un punto es la suma de los potenciales de cada carga. Si una de las cargas *q* está en la posición *(a,b)* y la otra *–q* en *(-a,-b).* el potencial eléctrico es:

$$v(x,y)=\frac{q}{4\pi\varepsilon_0}\left(\frac{1}{r_+}-\frac{1}{r_-}\right)$$

siendo:

$$r_+=\sqrt{(x-a)^2+(y-b)^2}$$
$$r_-=\sqrt{(x+a)^2+(y+b)^2}$$

Sabiendo que el campo eléctrico y el potencial verifican la relación:

$$\vec{E}(x,y)=-\nabla v(x,y)$$

utiliza los comandos de MATLAB *gradient*, y *quiver* para calcular y representar el campo eléctrico y las curvas de nivel del potencial, producido por dos cargas opuestas de 2x10-6 C y $k = \frac{1}{4\pi\varepsilon_0} = 9\text{x}10^9$, en la posición (1.5,-1.5) y (-1.5,1.5).

4.- Representa gráficamente unos cuantos elementos de las sucesiones siguientes:

$$a_n = sen(n\pi/2)$$
$$b_n = (1+1/n)^n$$
$$c_n = (-1)^n$$

Elige un estilo de trazado que deje los puntos aislados .

5.- Dados dos números $a < b$, obtén un vector x cuyas componentes sean los extremos de los intervalos obtenidos al dividir $[a, b]$ en n partes iguales. El vector x tiene $n + 1$ componentes a las que llamamos a $= x_0 < x_1 < x_2 < \ . \ . \ . \ < x_n = b$. Elegir adecuadamente a, b y n para obtener representaciones gráficas de los valores en x de las siguientes funciones:

a) $f(x) = sen\ x$
b) $f(x) = sinh\ x$
c) $f(x) = log\ x$
d) $f(x) = 1/x$
e) $f(x) = 1/(1+x^2)$
f) $f(x) = 1/(1-x^2)$

6.- La ecuación de los gases perfectos nos da la relación entre la presión P, la temperatura absoluta T, la masa m y el volumen V de un gas:

$$pV = mRT$$

donde R es la constante del gas, para el aire *286.7 (Newton metro/ kilogramo grados Kelvin*) si estamos en una habitación a $20ºC = 293º K$.
Crea una gráfica que exprese la relación entre la presión y el volumen para tres masas de aire de *1Kg, 3, kg* y *7 kg* con $20 \leq V \leq 100, m^3$.

7.- Representa la función $(x(t),y(t))=(cosh\ t, sinh\ t)$. ¿Qué tipo de asíntotas presenta esta función?

8.- Representa las siguientes funciones en tres dimensiones junto a sus curvas de nivel:
a) $z = x^2 - y^2$ b) $z = (x + y)^2$ c) $z = x^2y^3$ d) $z = sen\ x\ sen\ y$

9.- Una placa cuadrada de metal se calienta a $80º\ C$ en la esquina correspondiente a $x = y = 1$. La temperatura se distribuye según la función:

$$T = 80e^{-(x-1)^2}e^{-3(y-1)^2}$$

Obtener la superficie y curvas de nivel para la temperatura. Pon títulos a los ejes ¿Cuál es la temperatura a la esquina correspondiente a $x = y = 0$?;

6 Soluciones

Problema 1

```
t=0:.1:2*pi;
x=3*sin(t);
y=4*cos(t);
```

```
xd=0.1:.1:5;xi=-5:.1:-0.1; %ara representar una gráfica hemos de
tener en cuenta los valores que nos interesa dar a la variable
independiente.
yd=1./xh;yi=1./xi;
plot(x,y,xd,yd,xi,yi)
```

Así tienes una aproximación de los puntos de corte que en este caso se pueden obtener fácilmente con la instrucción:

```
roots([16,0,-9*16,0,9])
```

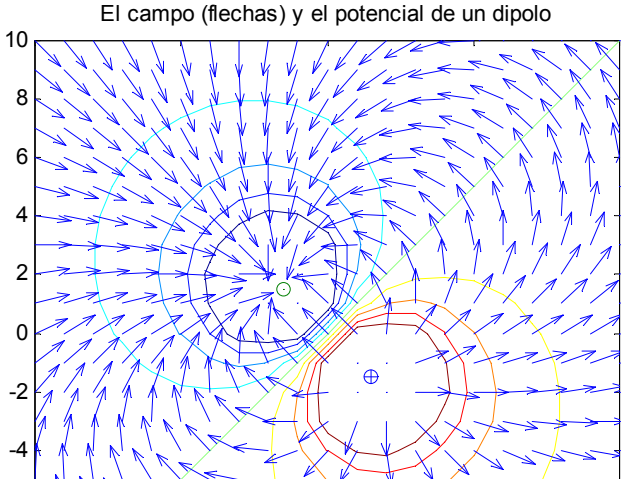
Problema 2

```
 for b=1:4
      t=0:.1:2;
      y=1-exp(-b*t);
      plot(t,y),text(1,y(11),['b =',num2str(b)])
       hold on
 end
title('Proceso de calentamiento de un cuerpo')

hold off
```

Al aumentar b, más rápido es el proceso. La función tiene un comportamiento asintótico en y = 1, por lo que para llegar al 98% tendremos:
$y = 1-e^{-bt} \cong 0.98$ de donde $t \cong 4/b$.

Problema 3

```
q=2e-6;    k=9e9;      a=1.5;    b=-1.5;
x=-10:1:10;      y=x;
[X,Y]=meshgrid(x,y);
rp=sqrt((X-a).^2+(Y-b).^2);
rn=sqrt((X+a).^2+(Y+b).^2);
V=q*k*(rp.^(-1)-rn.^(-1));
i=find(V>5000);V(i)=5000*ones(length(i),1);
i=find(V<-5000);V(i)=-5000*ones(length(i),1);
[Ex,Ey]=gradient(-V,1,1);
E=sqrt(Ex.^2+Ey.^2);
Ex=Ex./E;  Ey=Ey./E;%normalizamos  el  vector  E  para  que  las
%flechas que lo representan sean del mismo orden.
contour(X,Y,V)
title('El campo (flechas) y el potencial de un dipolo')
axis ('square')
hold on
plot(a,b,'o',-a,-b,'o');   plot(a,b,'+',-a,-b,'-');
quiver(X,Y,Ex,Ey)
hold off
```



Observa que la dirección del campo eléctrico es siempre perpendicular a las curvas de nivel del potencial.

ADVERTENCIA:

t=0 : 0.01 : 2*pi ; r=1 ; x=r*cos(t); y=r*sin(t); plot(x,y),axis equal
tiene un hueco a la derecha, porque no acaba en 2*pi
t=0 : pi/200 : 2*pi ;
seria lo correcto

PARA HACER EL GRADIENTE! :

```
x = -1: 0.25 : 1; y=x
[X,Y]=meshgrid(x,y)
Z = -X.^2 -Y.^2 %la función que sea
[dX,dY] = gradient(Z);

%con quiver se representa en cada punto un vector con las componentes
que se le diga.

quiver(X,Y,dX,dY)

%ahora se pueden superponer los contornos

hold

contour(X,Y,Z)
```

COORDENADAS CILINDRICAS:

ej:
r = -2:2;
con cylinder(r)

COORDENADAS ESFERICAS:

las polares en 3D

4 Aplicaciones de los sistemas lineales (I)

Muchos problemas en ciencias sociales, ingeniería, genética, etc..., se reducen finalmente a resolver un sistema de ecuaciones lineales. Este es uno de los problemas centrales del Álgebra Lineal, junto con el cálculo de valores y vectores propios. Ilustramos su importancia mediante tres ejemplos: un circuito eléctrico sencillo, el flujo de tráfico en una red de calles y la distribución de la temperatura en una barra en estado estacionario.

Los métodos directos para sistemas lineales consisten en transformar el sistema en otro equivalente más sencillo de resolver, por ejemplo un sistema triangular.

Esta transformación se interpreta matricialmente como una descomposición de la matriz del sistema en dos factores triangulares. Estos factores contienen toda la información relevante, de modo que cualquier sistema con la misma matriz, puede resolverse conociendo estos factores, sin más que resolver dos sistemas triangulares.

El proceso de eliminación, si funciona sin dificultades, produce además de la matriz triangular superior U otra matriz triangular inferior unidad L tal que $A = L \cdot U$**.**

El elemento *(i,k)* de la matriz L, con $i>k$, es el multiplicador usado para eliminar la variable x_k de la ecuación i-ésima, utilizando a_{kk} como pivote.

El conocimiento de la descomposición de una matriz A en factores triangulares LU facilita la resolución de posteriores sistemas de ecuaciones lineales con la misma matriz **A** y distintos términos independientes.

En lugar de resolver el sistema de matriz A, resolvemos dos sistemas triangulares

$$Ly = b \;\; ; \;\; Ux = y.$$

Así se ahorra trabajo, pues no hay que repetir los pasos de la fase de eliminación sobre toda la matriz, sino sólo sobre los nuevos términos independientes (mediante la sustitución progresiva en $Ly=b$).

Obtenido y, se resuelve el segundo sistema $Ux=y$, por sustitución regresiva. El coste de resolución de estos sistemas triangulares es del orden de n^2 operaciones, mientas que la eliminación sobre la matriz completa ha costado del orden de n^3 operaciones.

MATLAB proporciona también la descomposición LU de una matriz, mediante la función lu.

Si la eliminación tiene lugar sin intercambio de filas, utilizamos

```
[L,U]= lu(A)
```

En el caso general, hay que efectuar una permutación de filas de A para poder factorizar. La matriz de esta permutación se obtiene junto con los factores L y U de

```
[L,U,P]= lu(A)
```

verificándose la relación $PA = LU$.

Las operaciones elementales de la eliminación pueden aplicarse simultáneamente a varios términos independientes, con lo que resolvemos varios sistemas de una vez. Un caso particular interesante es el cálculo de la inversa de una matriz A, que puede interpretarse como la resolución de n sistemas lineales de matriz A y términos independientes, las columnas de la matriz identidad de orden n.

Ejemplo 1

Dada la matriz del calor

```
A=matcalor(3)
```

```
A =
     2    -1     0
    -1     2    -1
     0    -1     2
```

Con la función `lu` nos proporciona la descoposición *LU* de la matriz *A*

```
lu(A)

ans =
    2.0000   -1.0000         0
    0.5000    1.5000   -1.0000
         0    0.6667    1.3333
```

Los sistemas lineales surgen a menudo de la discretización de problemas de frontera de ecuaciones diferenciales lineales, ordinarias o en derivadas parciales. La discretización consiste en sustituir las derivadas por cocientes en diferencias en ciertos puntos del dominio. Así se obtienen sistemas de ecuaciones que se caracterizan por tener un elevado número de incógnitas, al tiempo que en cada ecuación sólo aparecen unas pocas. Las matrices de estos sistemas se denominan matrices dispersas y son de gran importancia en las aplicaciones.

4.1 Ecuación del calor

El *problema del calor* es un ejemplo de obtención de sistemas lineales por discretización de problemas continuos, normalmente formulados en términos de ecuaciones diferenciales o en derivadas parciales, aunque aquí daremos una justificación intuitiva de las ecuaciones planteadas.
Consideramos una barra conductora del calor cuyos extremos están a temperatura conocida T_a y T_b, respectivamente. Queremos hallar la temperatura a lo largo de la barra en estado estacionario.

Modelo Matemático **Matriz Asociada**

Para resolver numéricamente problemas de este tipo, consideramos un número finito de puntos a lo largo de la barra, por ejemplo, dividimos la longitud en $n+1$ partes iguales, tomando n puntos equiespaciados llamados nodos, $x_1, x_2, \ldots, x_n$ y estimaremos la temperatura de estos puntos, $T_1, T_2, \ldots, T_n$.

En el estado estacionario, suponemos que la temperatura en cada nodo es la media de las temperaturas de los nodos adyacentes. Al imponer esta condición a cada nodo obtenemos un sistema de ecuaciones lineales. Los valores de la temperatura en los extremos pasan al segundo miembro constituyendo los términos independientes.

$$
\begin{aligned}
T_1 &= (T_0 + T_2)/2 \\
T_2 &= (T_1 + T_3)/2 \\
T_3 &= (T_2 + T_4)/2 \\
&\vdots \\
T_n &= (T_{n\text{-}1} + T_{n+1})/2
\end{aligned}
$$

La matriz de este sistema tiene unas características especiales que aprovecharemos para su resolución. Se trata de una matriz tridiagonal simétrica; los elementos de la diagonal principal valen 2 y los de las diagonales situadas inmediatamente por encima y por debajo de la diagonal principal –1

$$
\begin{pmatrix}
2 & -1 & & & \\
-1 & 2 & -1 & & \\
 & -1 & 2 & \ddots & \\
 & & \ddots & \ddots & -1 \\
 & & & -1 & 2
\end{pmatrix}
$$

Ejemplo 2

Construimos esta matriz utilizando la orden `diag(v)` de MATLAB, que sitúa en una diagonal determinada los elementos del vector v. En nuestro caso hacemos:

```
n = 5;
v = ones(n-1,1);
calor = 2*eye(n) - diag(v,-1) - diag(v,1)
```

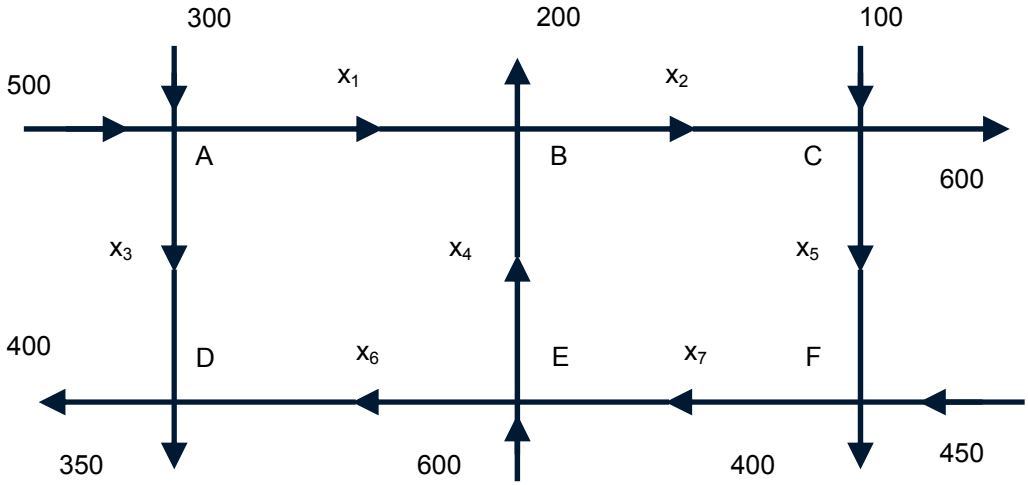
```
calor =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2
```

El segundo argumento de *diag* indica en que diagonal hay que colocar los elementos de v. Las diagonales se numeran a partir de la principal, que es la diagonal 0. La 1 esta situada justo encima de la principal y la –1 justo debajo. En general la diagonal k esta formada por los elementos a_{ij}

4.2 Red de calles

El flujo de tráfico en una red de calles se modeliza mediante un sistema de ecuaciones lineales. Supongamos una zona de la ciudad formada por calles que se cruzan

perpendicularmente en los puntos *A, B, ..., F*, como se muestra en la figura. Las calles son de sentido único de circulación indicado por las flechas. Denotamos el flujo de tráfico entre los cruces por las variables x_i, $i = 1, 2, ..., 7$. Junto a cada cruce aparece el número de vehículos que entran o salen por hora de la zona estudiada.



Tratamos de ver si se puede cortar la circulación de las calles 6 y 7 para realizar obras sin afectar el tráfico ajeno a la zona considerada. Formularemos el problema mediante un sistema de ecuaciones lineales. Suponemos que el tráfico no se acumula en las intersecciones. Entonces, el tráfico que entra en cada cruce debe ser igual al que sale.

Ejemplo 3

El tráfico que llega al cruce *E* es, según el mapa, $x_7 + 600$. El tráfico que sale es $x_4 + x_6$. Así obtenemos $x_4 + x_6 = 600 + x_7$, o bien
$x_4 + x_6 - x_7 = 600$**.**

El análisis de cada cruce proporciona el siguiente sistema:

$$\begin{aligned}
x_1 + x_3 &= 800\\
x_1 - x_2 + x_4 &= 200\\
x_2 - x_5 &= 500\\
x_3 + x_6 &= 750\\
x_4 + x_6 - x_7 &= 600\\
x_5 - x_7 &= -50
\end{aligned}$$

Es un sistema lineal de 6 ecuaciones, una por cruce, con 7 incógnitas, una por calle. Las hipótesis del modelo sugieren que la suma de los términos independientes ha de ser cero, pues en la zona no se crean ni se almacenan vehículos. Al haber menos ecuaciones que incógnitas, tal vez se puedan añadir condiciones adicionales como el cierre de las calles 6 y 7. La teoría de sistemas lineales responderá estas cuestiones. Comencemos analizando la estructura de la matriz del sistema.

4.3 Matriz de incidencia

		Calle						
		1	**2**	**3**	**4**	**5**	**6**	**7**
C	**A**	1	0	1	0	0	0	0
r	**B**	-1	1	0	-1	0	0	0
u	**C**	0	-1	0	0	1	0	0
c	**D**	0	0	-1	0	0	-1	0
e	**E**	0	0	0	1	0	1	-1
	F	0	0	0	0	-1	0	1

Las filas de la matriz corresponden a los cruces y las columnas a las calles que los conectan. En la fila correspondiente a un cruce, aparece un 1 por cada calle que 'sale' de ese cruce y un –1 por cada calle que 'entra' en el mismo.
Recíprocamente, en la columna de cada calle, hay un 1 en la posición del cruce donde 'empieza' la calle y un –1 en el lugar del cruce en que 'termina' la calle.
En cada columna hay 2 elementos no nulos, por lo que la matriz puede ser muy grande, pero tiene muchos ceros, lo cual debería ser aprovechado para simplificar los cálculos.

A(1,1) =1; A(2,1)=–1;
A(2,2) =1; A(3,2)=–1;
A(1,3) =1; A(4,3)=–1;
A(5,4) =1; A(2,4)=–1;
A(3,5)=1; A(6,5)=–1;
A(5,6)=1; A(4,6)=–1;
A(6,7)=1; A(5,7)=–1

Edita también los términos independientes.

b=[800;–200;–500;–750;600;50]

La matriz construida se denomina matriz de *incidencia. La* red de calles es un ejemplo de *grafo dirigido.* Un grafo dirigido está formado por un conjunto de vértices, los cruces, y un conjunto de arcos uniendo determinados pares de vértices. La matriz de incidencia de un grafo dirigido se construye como en el caso de la red de calles, poniendo una fila por vértice del grafo y una columna por arco. En cada columna hay un 1 en la fila del vértice inicial del arco y un 1 en la fila del vértice final.

Ejemplo 4

Editamos la matriz que denominanos *A* por calles, definiendo los elementos no nulos de cada columna en las filas correspondientes a los cruces.

```
A=[1,0,1,0,0,0,0;-1,1,0,-1,0,0,0;0,-1,0,0,1,0,0;0,0,-1,0,0,-1,0;0,0,0,1,0,1,-1;0,0,0,0,-1,0,1]
```

```
A =
     1     0     1     0     0     0     0
    -1     1     0    -1     0     0     0
     0    -1     0     0     1     0     0
     0     0    -1     0     0    -1     0
     0     0     0     1     0     1    -1
     0     0     0     0    -1     0     1
```

Editamos los términos independientes

```
b=[800 ;-200 ;-500 ;-750 ;600 ;50]
```

```
b =
   800
  -200
  -500
  -750
   600
    50
```

calculamos el rango de la matriz *A*

```
rank(A)
```

```
ans =
     5
```

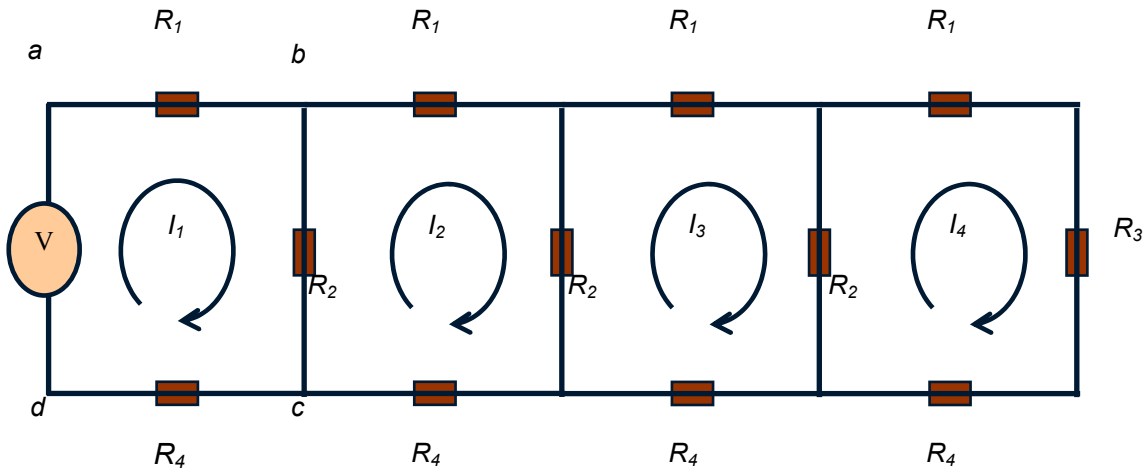
y ahora calculamos el valor de la matriz ampliada para ver si el sistema es compatible.

```
rank([A,b])
```

```
ans =
     5
```

Como el rango de la matriz *A* es igual al rango de la matriz ampliada $A|b$ entonces el sistema es compatible

4.4 Red eléctrica



Consideremos un circuito eléctrico con sólo resistencias y fuentes de potencial como el de la figura. Suponemos conocidos el voltaje V y los valores de las resistencias R_1, R_2, R_3, R_4. Determinaremos las intensidades en cada rama del circuito mediante un sistema de ecuaciones lineales que se obtiene aplicando las leyes de *Kirchhoff* y la ley de *Ohm*.
Ley de los nudos: la suma algebraica de las intensidades que concurren en un nudo es cero.
Ley de las mallas: la caída de potencial a lo largo de un camino cerrado del circuito es nula.

Estas leyes no dependen de los elementos físicos del circuito, sino sólo de las conexiones entre los mismos. En el caso de resistencias, su influencia queda establecida mediante la
Ley de Ohm: La caída de potencial en cada rama es el producto de la intensidad que circula por ella, por su resistencia:

$$V = I \cdot R$$

Aplicamos implícitamente la ley de los nudos al considerar que las intensidades están vinculadas a las mallas, en lugar de a los arcos. De este modo reducimos el número de incógnitas. Así, en la figura, la corriente I_1 circula sobre el circuito cerrado *abcd*. Como el cable *bc* es común a dos mallas, la corriente que circula por él es $I_1–I_2$.

Aplicando la ley de las mallas a *abcd* tenemos

$$V_{ab} + V_{bc} + V_{cd} = V$$

La ley de Ohm para el cable que conecta *b* y *c* establece que

$$V_{bc} = R_2(l_1-l_2)$$

donde R_2 es el valor de la resistencia que conecta *b* con *c*. Aplicando esta ley al resto de las ramas de la malla y sustituyendo en la ecuación anterior queda

$$R1I1+R2(l1-l2)\ R4I1 = V.$$

siendo V el voltaje de la fuente.

Repitiendo este proceso para cada malla, se obtiene un sistema de ecuaciones lineales cuya solución son las intensidades en cada malla del circuito. Las intensidades en cada rama se obtienen inmediatamente como suma algebraica de las intensidades que circulan por las mallas de las que forma parte.

Ejemplo 5

. Resolvamos el siguiente problema del circuito sí $V=10V$, $R_1 = R_3 = R_4 = 1\ k\Omega$ y $R_2 = 10\ k\Omega$.

El sistema de ecuaciones resultante es el siguiente:

$$\begin{aligned} 12I_1-10I_2 &= 10;\\ 10I_1-22I_2+10I_3 &= 0;\\ 10I_2-10I_3+10I_4 &= 0;\\ 10I_3-13I_4 &= 0; \end{aligned}$$

Este sera el sistema $A*x=b$, donde la matriz es

```
A=[12 -10 0 0;10 -22 10 0;0 10 -10 10;0 0 10 -13]
```

```
A =
    12   -10     0     0
    10   -22    10     0
     0    10   -10    10
     0     0    10   -13
```

y los términos independientes

```
b= [10 ; 0 ; 0 ; 0]
```

```
b =
    10
     0
     0
     0
```

luego encontramos los valores de la corriente I

```
I=A\b
```

```
I =
    0.5993
   -0.2809
   -1.2172
   -0.9363
```

4.5 Matrices Dispersas

Los sistemas lineales que aparecen en las aplicaciones suelen tener matriz *dispersa*, es decir, matriz con relativamente pocos elementos no nulos. Estas matrices son además de gran tamaño, por lo que almacenarlas y tratarlas como matrices normales (llenas, en oposición a dispersas) supone un despilfarro innecesario.
Por ello, MATLAB dispone de funciones específicas para estas matrices.
La función sparse declara dispersa la matriz A. Así, sólo se almacenan y se opera con sus elementos no nulos. Por ejemplo, la propia matriz identidad es dispersa, con **n** elementos no nulos sobre un total de n^2 elementos. En contraste con eye(n) que es la identidad de orden n considerada como matriz llena, n=3,A=speye(n) define la misma matriz, pero MATLAB la trata como matriz dispersa la orden . full(A) convierte en llena una matriz declarada previamente dispersa.
Al listar una matriz dispersa obtenemos una serie de filas con un par que indica la posición de un elemento no nulo seguido del valor de este elemento.
La función spy(A) muestra gráficamente las posiciones no nulas de la matriz A.

Ejemplo 4

Dada la matriz del calor del **Ejemplo 2** veremos algunas de las funciones de MATLAB.

```
n = 5;
v = ones(n-1,1);
A = 2*eye(n) - diag(v,-1) - diag(v,1)
```

```
A =
```

```
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2
```

A=matcalor(10) % Matriz ecuación del calor

```
A =
     2    -1     0     0     0     0     0     0     0     0
    -1     2    -1     0     0     0     0     0     0     0
     0    -1     2    -1     0     0     0     0     0     0
     0     0    -1     2    -1     0     0     0     0     0
     0     0     0    -1     2    -1     0     0     0     0
     0     0     0     0    -1     2    -1     0     0     0
     0     0     0     0     0    -1     2    -1     0     0
     0     0     0     0     0     0    -1     2    -1     0
     0     0     0     0     0     0     0    -1     2    -1
     0     0     0     0     0     0     0     0    -1     2
```

a=sparse(A) % La convierto en dispersa

```
a =
   (1,1)        2
   (2,1)       -1
   (1,2)       -1
   (2,2)        2
   (3,2)       -1
   (2,3)       -1
   (3,3)        2
   (4,3)       -1
   (3,4)       -1
   (4,4)        2
   (5,4)       -1
   (4,5)       -1
   (5,5)        2
```

full(a) % Vuelve a ser llena

```
ans =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2
```

issparse(A) % Para saber si es dispersa

```
ans =
     0
```

issparse(a)

```
ans =     1
```

```
spy(A)                    % Gráfico elementos no nulos

spy(a)                    % Lo mismo

[i,j,v]=find(A)           % Fila, columna y valor de    % los elementos no nulos de A

sparse(i,j,v)             % Lo mismo que sparse(A).
```

Ahora daremos algunos ejemplos:

1. Sea E la matriz de orden n

$$E = \frac{1}{n+1} C \;;\; c_{ij} = \begin{cases} i(n-i+1) & \text{si} \quad i = j \\ i(n-j+1) & \text{si} \quad i < j \\ c_{j,i} & \text{si} \quad i > j \end{cases}$$

a) Genera la matriz E con un fichero.m.

```
[E,C]=matrix(5)
```

b) Sea b el vector b = [1:n]'. Resuelve el sistema Ex = b

```
b=[1 2 3 4 5]'
x=E\b
```

2. Sea H(n)=hilb(n), la matriz de Hilbert de orden n. Halla la inversa de $H(n)^tH(n)$ para los valores de n=5, 10 ,15 de dos modos diferentes:

a) Directamente
b) Observando que $(H^tH)^{-1} = H^{-1}(H^{-1})^{-1}$
c) Compara con la solución exacta que es invhilb(n)*(invhilb(n))'.

Solución:

```
n=5
H=hilb(n)

H'
```

```
X=H.'*H

inv(X)

invhilb(n)*(invhilb(n))'
```

Analogamente hacemos lo mismo para n=10, 15.

Problemas

1. La segunda derivada de una función u(x) se puede aproximar por la fórmula

$$u''(x) \approx \frac{u(x+h) + u(x-h) - 2u(x)}{h^2}$$

para h pequeño.
Considera el siguiente problema de frontera definido en $x \in [0,1]$.

$$-u''(x) = 4\pi^2 \operatorname{sen}(2\pi x)$$
$$u(0) = 0$$
$$u(1) = 0$$

a) Tomando h = 0.1, $u_i = u(0.1*i)$, i = 0, 1, ..., 10 y aproximando la segunda derivada de u, obtén un sistema de ecuaciones 9×9 cuya solución aproxime la solución exacta del problema.
b) Resuelve el sistema anterior considerando las matrices como llenas y dispersas. Compara los resultados y el número de operaciones efectuadas.
c) Compara la solución con la solución exacta que es u(x) = sen(2πx).

2. Sea A la siguiente matriz de orden n:

$$\begin{pmatrix} a & b & 0 & \cdots & 0 & 0 \\ b & a & b & \cdots & 0 & 0 \\ 0 & b & a & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a & b \\ 0 & 0 & 0 & \cdots & b & a \end{pmatrix}$$

Se puede probar que $(I-A)^{-1} = I + A + A^2 + \cdots + A^k + \cdots$, converge si $|a|+2|b|<1$. Experimenta la fórmula descrita para varios valores de n, a y b.

3. Resuelve Ax = c siendo A la matriz del ejercicio 5, con a = b = 1/4 y c el vector formado por 1s, para los valores n = 20, 30, 40, 50. Considera las matrices como dispersas y como llenas, compara los resultados y el número de operaciones.

4. Escribe en MATLAB un fichero.m que permita encontrar la inversa de una matriz usando el método de Gauss-Jordan.

5. Considera la base en P_{10} $B=\{1, 1+x, 1+x+x^2, \cdots, 1+x+\cdots+x^{10}\}$ y sean los vectores x^5-6x^3+2,
$x^{10}-x^6+x^2+1$, x^9-1.
a) Halla las coordenadas de estos vectores en la base B resolviendo 3 sistemas de ecuaciones lineales.

b) Considera las matrices como dispersas y repite el apartado a).
c) Resuelve el apartado a) con la factorización LU (Observa que se trata de resolver 3 sistemas de ecuaciones simultáneamente).
d) Repite el apartado c) considerando las matrices como dispersas.
e) Compara el número de operaciones efectuadas.

5 Aplicaciones de los sistemas lineales (II)

Vamos a seguir mostrando en este tema diversas aplicaciones de los sistemas de ecuaciones lineales tales como la distribución de temperatura en una placa, el problema de interpolación numérica que da lugar a un sistema con matriz asociada la matriz de Van der Monde y un caso de integración numérica que se resuelve mediante un sistema lineal con la matriz de Hilbert como matriz asociada.

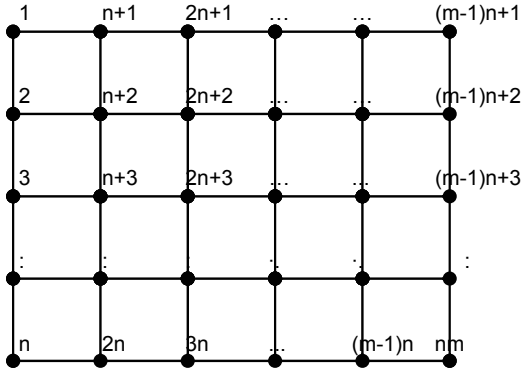
Analizaremos también el grado de precisión que se tiene al resolver resolver un sistema por un método directo ya que al trabajar en precisión finita, los errores de redondeo se van acumulando de un paso a otro. Aunque el error de cada operación es pequeño, como el número de operaciones puede ser muy grande (del orden de n^3, siendo n el tamaño del sistema), al final, el resultado puede verse muy afectado.
Aunque las operaciones se realicen en modo exacto, puede que los coeficientes del sistema sean imprecisos, por ejemplo si se han obtenido de medidas experimentales. Es importante conocer de qué modo afecta la incertidumbre de los datos al resultado final.
Para controlar este crecimiento se aplica la estrategia de pivotación parcial, evitando tomar pivotes muy pequeños durante la eliminación.

Veremos que hay sistemas que amplifican el error durante la eliminación, los *sistemas mal condicionados*, mientras que otros lo mantienen dentro de ciertos límites, los *bien condicionados*.
En todo caso, es de vital importancia para controlar el error, seguir alguna estrategia al elegir los pivotes de la eliminación, pues un pivote pequeño provoca un aumento del error. La *pivotación parcial* consiste en elegir como pivote el elemento de mayor valor absoluto en la columna a eliminar. Dado que a veces hay que buscar un pivote no nulo, no cuesta mucho más buscarlo siempre con la condición de tener valor absoluto máximo, y con ello se minimiza el error de la eliminación.

5.1 La ecuación del calor en una placa

El problema de la *distribución de temperatura en una placa*, es "continuo" en el sentido de que queremos conocer la temperatura en todos los (infinitos) puntos de su superficie. Este objetivo sólo es alcanzable si somos capaces de obtener una solución analítica, lo cual no es frecuente en las aplicaciones.
En cambio, si nos conformamos con conocer la temperatura en un número finito de puntos de la placa, podemos obtener por medios algebraicos sencillos una solución aproximada.
Consideramos pues $(n+2)\cdot(m+2)$ puntos regularmente distribuidos en la placa, en los nodos de una *malla* de celdas cuadradas.
Suponemos que en el estado estacionario, la temperatura en cada nodo es la media las temperaturas en los cuatro nodos adyacentes. Suponemos además, para que el problema esté determinado, que la temperatura de los nodos de los bordes de la placa es conocida.
Planteando estas condiciones para cada nodo del interior de la placa, se obtiene un sistema de $n\cdot m$ ecuaciones lineales con $n\cdot m$ incógnitas.

Numeramos los nodos por columnas, como indica el esquema adjunto, lo que establece una ordenación de las ecuaciones e incógnitas del sistema.



Con esta ordenación, el nodo *k* está relacionado con los nodos *k–1* y *k+1* de su columna, y con los nodos *k–n y k+n* de su fila. La ecuación correspondiente al nodo *k* se puede poner como:

$$-T_{k-n} - T_{k-1} + 4T_k - T_{k+1} - T_{k+n} = 0$$

En cada ecuación aparecen a lo sumo cinco coeficientes no nulos. Los nodos del borde tienen alrededor tres nodos incógnita y uno conocido, cuyo valor pasa al término independiente. Los nodos de las esquinas tienen dos vecinos incógnitos y dos conocidos.

La matriz del sistema es dispersa con elementos no nulos agrupados en cinco diagonales: la principal, las dos adyacentes y las dos situadas a distancia *n* de la principal, que corresponden en la malla a las columnas anterior y siguiente al nodo *k*. Los elementos de la diagonal principal valen 4, y los de las otras diagonales no nulas, –1, excepto algunos ceros correspondientes a los nodos de los bordes superior e inferior de la malla *(k = n, n+1, 2n, 2n+1).*

Para $n = m = 3$ si suponemos que en los bordes superior e inferior la temperatura es de 50 ºC, en el lado izquierdo 0 ºC y en el derecho 100 ºC la situación es la siguiente:

	50	50	50	
0	T_1	T_4	T_7	100
0	T_2	T_5	T_8	100
0	T_3	T_6	T_9	100
	50	50	50	

y el sistema de ecuaciones tiene la forma:

$$\begin{pmatrix} 4u_1 & -u_2 & & -u_4 & & & & & \\ -u_1 & 4u_2 & -u_3 & & -u_5 & & & & \\ & -u_2 & 4u_3 & & & -u_6 & & & \\ -u_1 & & & 4u_4 & -u_5 & & -u_7 & & \\ & -u_2 & & -u_4 & 4u_5 & -u_6 & & -u_8 & \\ & & -u_3 & & -u_5 & 4u_6 & & & -u_9 \\ & & & -u_4 & & & 4u_7 & -u_8 & \\ & & & & -u_5 & & & 4u_8 & -u_9 \\ & & & & & -u_6 & & -u_8 & 4u_9 \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \end{pmatrix} = \begin{pmatrix} 50 \\ 0 \\ 50 \\ 50 \\ 0 \\ 50 \\ 150 \\ 100 \\ 150 \end{pmatrix}$$

En MATLAB construimos la matriz por diagonales mediante un fichero.m

Programa para generar la matriz (calor2D)

Entrada: n,m dimensiones de la matriz.
Salida: A matriz del sistema.
Proceso:

```
% cálculo de las diagonales 1 y –1.
p = n*m;
v = ones(1,p-1);
for k = n:n:p-n, v(k) = 0; end

% cálculo de las diagonales n y –n.
w = ones(1,p-n);
% Matriz del sistema
A = 4*eye(p)- diag(v,1) - diag(v,-1)- diag(w,n) - diag(w,-n);
```

Ejemplo 1

Comprueba la estructura de la matriz del sistema en el caso de tomar 9 puntos en la placa.
Solución: basta ejecutar la función anterior, `A = calor2D(3,3)` .

Como hemos dicho la temperatura en los bordes de la placa es conocida, supongamos que los lados de la placa se mantienen a 0 ºC, salvo el inferior que se calienta a 100 ºC. Como los nodos vecinos al lado inferior ocupan los lugares *n*, *2n*, …, calculamos los términos independientes *b* del modo siguiente:

```
b = zeros(n*m,1)
b(n:n:n*m) = 100*ones(m,1)
```

Ejemplo 2

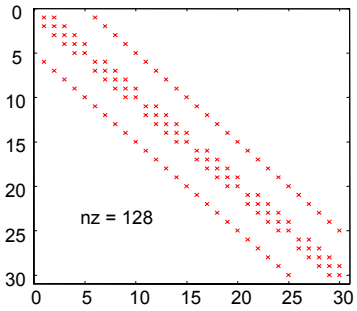
a) Halla la temperatura final en estado estacionario tomando 30 puntos en el interior de la placa, visualiza la estructura de la matriz del sistema.
b) Representa gráficamente los resultados.
c) Calcula el número de operaciones realizado obteniendo el ahorro que supone trabajar con matrices dispersas.

Solución: a) resolvemos un sistema de matriz *A* y término independiente *b* que incluye la condición de frontera dada.

```
n = 5; m = 6; A = calor2D(n,m); b = zeros(n*m,1);b(n:n:n*m) = 100*ones(m,1) ;
```

Visualizamos la estructura de la matriz con

```
spy(A)
```



Resolvemos el sistema y hallamos el coste:

```
flops(0), x = A\b, flops
```

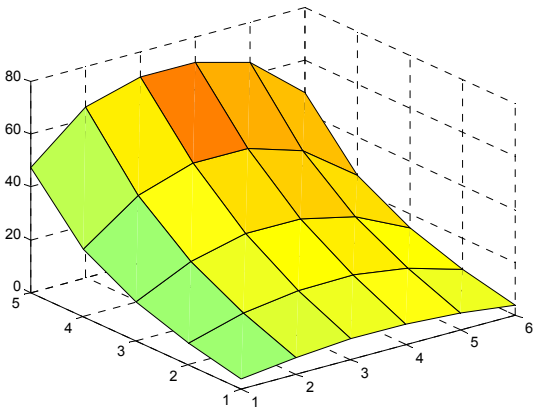
Reordenamos la columna de resultados en filas y columnas según la disposición rectangular de la malla inicial:

```
placa = reshape(x,n,m)
```

3.6120	6.3802	7.8437	7.8437	6.3802	3.6120
8.0679	14.0649	17.1510	17.1510	14.0649	8.0679
14.5947	24.6606	29.5443	29.5443	24.6606	14.5947
25.6503	40.4385	46.8213	46.8213	40.4385	25.6503
47.5681	64.6219	70.4811	70.4811	64.6219	47.5681

b) Representamos gráficamente los resultados con

```
surfl(placa)
```



c) El coste se reduce operando con matrices dispersas:

```
sA = sparse(A);
sb = sparse(b);
flops(0), sx = sA\sb, flops
```

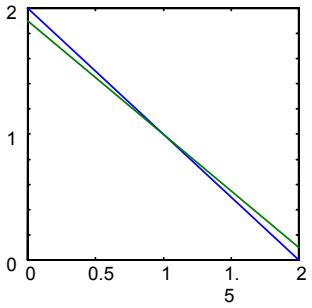
Obtenemos el mismo resultado a un coste diez veces menor. Observa que el resultado no es disperso. Para obtener la gráfica hay que ponerlo primero en forma normal:

```
x = full(sx)
surfl(reshape(x,n,m))
```

5.2 Condicionamiento de una matriz

Hay sistemas en los que un pequeño cambio en los coeficientes provoca un gran cambio en la solución. Estos sistemas se llaman *mal condicionados* . Por ejemplo, el sistema

$$\left.\begin{aligned} 0.0001\,x + \quad y &= \quad 1.0001 \\ x + \quad y &= \quad 2 \end{aligned}\right\}$$



tiene solución exacta $x = y = 1$. Si cambio el término independiente de la segunda ecuación a 2.0002,

$$\left.\begin{aligned} x + \quad\quad\quad y &= \quad 2 \\ x + \quad 1.0001\,y &= \quad 2.0002 \end{aligned}\right\}$$

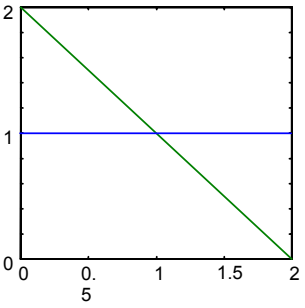
la solución cambia a $x = 0, y = 2$.

Cambiando el coeficiente de x en la segunda ecuación a 1.0001, el primer sistema resulta incompatible y el segundo indeterminado.

Esto indica que el sistema es muy sensible a los errores de redondeo y su solución poco fiable. Si representamos gráficamente las ecuaciones del sistema, obtenemos dos rectas que se cortan formado un ángulo muy pequeño. El punto de intersección cambia mucho al mover poco una de las rectas.

En cambio, el sistema

$$\left.\begin{aligned} x + \quad\quad\quad y &= \quad 2 \\ x + \quad 1.0001\,y &= \quad 2.0001 \end{aligned}\right\}$$



está bien condicionado, pues la solución es poco sensible a los errores de los datos. En la representación gráfica vemos dos rectas cuya intersección está claramente situada.

Nota: para realizar todos los ejemplos y ejercicios de este apartado te puedes ahorrar introducir las matrices cargándolas de la carpeta practica04 con la instrucción load.

```
load E:\asig\1\matemati\laborato.rio\practica.04\practi04
who
```

Para sistemas de más de dos ecuaciones no es fácil analizar gráficamente su condicionamiento. Algorítmicamente, el mal condicionamiento se puede observar por el pequeño tamaño de los pivotes.
MATLAB dispone de medidas numéricas del condicionamiento.

Las funciones de MATLAB `cond` y `rcond` miden la sensibilidad al error al invertir una matriz o resolver un sistema con dicha matriz.
La función cond toma valores de 1 a infinito. Cuanto mayor es cond, peor es el comportamiento de la matriz.
Inversamente, la función rcond toma valores de 0 a 1, correspondiendo los próximos a cero a matrices mal condicionadas y los valores próximos a 1, a matrices bien condicionadas.

Ejemplo 3

Resuelve el siguiente sistema de ecuaciones lineales y compara la solución con la que resulta al variar ligeramente a_{22} a *4.2750*, obtén el numero de condición de la matriz inicial para establecer conclusiones al respecto.

$$\left.\begin{array}{l} 3.0210\,x_1 + 2.7140\,x_2 + 6.9130\,x_3 = 12.6480 \\ 1.0310\,x_1 - 4.2730\,x_2 + 1.1210\,x_3 = -2.1210 \\ 5.0840\,x_1 - 5.8320\,x_2 + 9.1550\,x_3 = 8.4070 \end{array}\right\}$$

Solución: Llama A a la matriz del sistema `A=malcon1` y b al vector de términos independientes `b=ti1` ejecuta en Matlab `X=A\b` y se tiene la solución exacta $\boldsymbol{x} = \boldsymbol{y} = \boldsymbol{z} = \boldsymbol{1}$, para obtener el sistema

$$\left.\begin{array}{l} 3.0210\,x_1 + 2.7140\,x_2 + 6.9130\,x_3 = 12.6480 \\ 1.0310\,x_1 - 4.2750\,x_2 + 1.1210\,x_3 = -2.1210 \\ 5.0840\,x_1 - 5.8320\,x_2 + 9.1550\,x_3 = 8.4070 \end{array}\right\}$$

sólo tienes que hacer `A(2,2)=-4.2750` ya que el resto de elementos son iguales, buscamos de nuevo la orden `X=A\b` y observaras que se tienen soluciones muy distintas.
Para conocer la fiabilidad de la solución, hallamos la condición de la matriz del sistema: `cond(A)` y vemos que es muy elevada, como era de esperar. La solución, por tanto no es muy fiable, si quieres probar con rcond obtendrás un valor muy pequeño ya que se trata de una matriz casi singular.

Podemos decir que `rcond` mide lo cerca que está una matriz de ser singular. El criterio teórico de singularidad, la anulación del determinante, no es válido para estudiar la casi singularidad, pues, entre otras cosas depende de la escala. En efecto, aunque en general una matriz casi singular tendrá determinante pequeño, el recíproco no es cierto, como puedes comprobar con el siguiente ejercicio.

Ejemplo 4

Considera el siguiente sistema de 10 ecuaciones con 10 incógnitas

$$0.1\, x_i = 1 \quad i=1, \ldots, 10.$$

Comprueba que aunque el determinante de la matriz es muy pequeño el sistema está bien condicionado.

Solución: La matriz de los coeficientes es *10x10* con todos los elementos nulos excepto la diagonal principal que son 0.1 por lo que su determinante es 10^{-10} , se trata de una matriz casi singular pero muy bien condicionada ya que su número de condición es 1.

```
M=0.1*eye(10), d=det(M), c=cond(M)
```

El mal condicionamiento de un sistema es difícil, si no imposible de resolver, pero hay que tener cuidado para no arruinar un sistema bien condicionado aplicándole un algoritmo de eliminación deficiente.

Por ejemplo, hemos visto que el sistema

$$\left.\begin{aligned} 0.0001\,x + \; y &= 1.0001 \\ x + \; y &= 2 \end{aligned}\right\}$$ está bien condicionado. Sin embargo, si elimino

normalmente x en la segunda ecuación, obtengo un sistema casi singular.

$$\left.\begin{aligned} 0.0001\,x + \; y &= 1.0001 \\ -9999\,y &= -9999 \end{aligned}\right\}$$

El empeoramiento es debido a que el pivote utilizado, a_{11}, que es diez mil veces menor que el coeficiente a eliminar, a_{21}. El remedio obvio será intercambiar las dos ecuaciones antes de eliminar. Al tomar 1 como pivote, queda el sistema

$$\left.\begin{aligned} x + \; y &= 2 \\ 0.9999\,y &= 0.9999 \end{aligned}\right\}$$

que sigue siendo bien condicionado.

La función *A\b* que usamos para resolver sistemas con MATLAB aplica pivotación parcial. En el caso de sistemas incompatibles, obtiene la solución más aproximada en sentido mínimo cuadrático.

Ejemplo 5

Piensa en un sistema de ecuaciones incompatible que no sea cuadrado e intenta resolverlo con MATLAB. ¿Qué ocurre?.

Solución: Es muy fácil obtener un sistema incompatible basta pensar por ejemplo en dos planos paralelos:

$$2x - y + z = 4$$
$$2x - y + z = 1$$

El sistema no tiene solución pues mientras la matriz del sistema es de rango 1 la ampliada es de rango 2, sin embargo al hacer *x=A\b* se obtiene solución al problema, aunque MATLAB muestra una advertencia.

Prueba ahora con la resolución de un sistema formado por dos rectas paralelas y otra secante a ellas, se trata también de un sistema incompatible en el que MATLAB nos da la solución sin mostrar en este caso ninguna advertencia.

Lo que esta ocurriendo es que cuando el sistema es incompatible se nos da la solución "menos mala" como veremos en el tema 10.

5.3 Interpolación polinómica. Matriz de van der Monde

En general, la *interpolación polinómica*, consiste en determinar un polinomio de grado *n* que pase por *n+1* puntos dados. Se eligen los polinomios porque son fáciles de evaluar y por el hecho fundamental de que por *n+1* puntos de abscisa distinta,
(x_1, y_1),..., (x_n, y_n), (x_{n+1}, y_{n+1}), pasa un único polinomio $P_n(x)$ de grado no superior a *n*.

En la *interpolación lineal*, la función se sustituye por la recta que pasa por dos puntos. Tres datos se interpolan con un polinomio de segundo grado, gráficamente una parábola que pasa por esos tres puntos; es la llamada *interpolación cuadrática*.

Ejemplo 6 :

Para hallar el polinomio de grado 3

$$p(x) = a_1x^3 + a_2x^2 + a_3x + a_4$$

que pasa por los puntos *(1,0), (2,1), (3,0) y (4,1),* sustituimos *(x, p(x))* por los pares dados, obteniendo un sistema de ecuaciones lineales

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 \\ 27 & 9 & 3 & 1 \\ 64 & 16 & 4 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

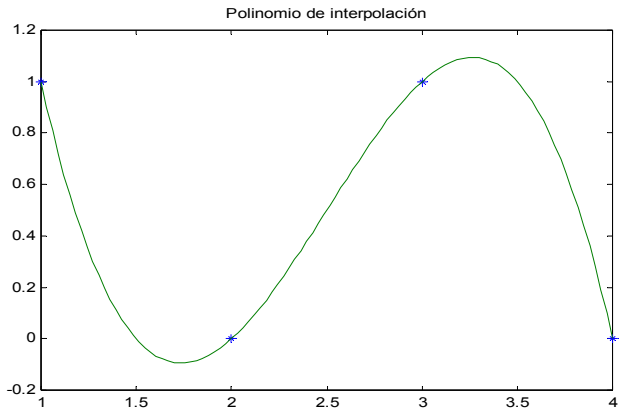
$$V\,p = y$$

donde *p* denota la columna de las incógnitas, a_1,..., a_4 ; *y* es la columna de ordenadas de los puntos dados y *V* es la matriz del sistema. Decimos que *V* es la matriz de van der Monde asociada al vector de abscisas, *x*.
Ejecuta en MATLAB `x = [1 2 3 4], V = vander(x)`
La solución del sistema se obtiene con `y = [1;0;1;0], p = V\y`
Analicemos la estabilidad del sistema:

```
cond(V)          % da 1.1710e+003
```

el mal condicionamiento nos llevará como veremos en el capítulo 10 a buscar distintas expresiones para los polinomios de interpolación.

```
xx=linspace(x(1),x(length(x)));
yy=polyval(p,xx);
plot(x,y,'*', xx, yy)
title('Polinomio de interpolación')
```



Ejemplo 7

Dado un vector $v=(v_1,v_2, ..., v_n)$ la matriz de Van der Monde de orden n puede definirse formalmente del siguiente modo:

$$V_n=\begin{pmatrix} 1 & v_1 & v_1^2 & . & . & v_1^{n-1} \\ 1 & v_2 & v_2^2 & & & v_2^{n-1} \\ . & . & & & & . \\ . & . & & & & . \\ . & . & & & & . \\ 1 & v_n & v_n^2 & . & . & v_n^{n-1} \end{pmatrix}$$

a) Edita una función en MATLAB para generarla. (Sin utilizar la orden vander).
b) ¿Cómo la obtendrías en MATLAB a partir de la instrucción Vander?
c) Comprueba los apartados a) y b) para $v=(1,2,3,4)$. Observa que la matriz de Van der Monde es otro ejemplo de mal condicionamiento.
d) Fácilmente se comprueba que *el* $det(V_n)=\prod_{i<j}(v_j-v_i)$, realiza una función en MATLAB que calcule el determinante a partir de esta expresión y comprueba que obtienes el mismo resultado que directamente en MATLAB con $det(V_n)$.

Solución: a)
```
function V=mivander(v)

for i=1:length(v)
   V(:,i)=v.^(i-1)';
end
```

b) Con *V=vander (v)* se obtiene la matriz volteada basta pues hacer V=fliplr(V).

c)
```
v=1:4;,V1=mivander(v),V2=vander(v),
V2=fliplr(vander(v)),C=cond(V1)

V1 =
     1     1     1     1
     1     2     4     8
     1     3     9    27
```

```
     1     4    16    64
V2 =
     1     1     1     1
     1     2     4     8
     1     3     9    27
     1     4    16    64
ans =
  1.1710e+003
```

d)
```
function d=detvander(v)
   % nota que el parámetro de entrada es el vector v
   %a partir del que se genera la matriz de Van der Monde.

   d=1;
   for j=1:n-1
      for i=j+1:n
       d=d*(v(i)-v(j));
      end
   end

det(V1), detvander(v)
```

5.4 Integración numérica. Matriz de Hilbert

Imagina ahora que queremos determinar los coeficientes del polinomio *p(x)* de modo que cumplan

$$\int_0^1 x^k\, p(x)\,dx = \int_0^1 x^k\, f(x)\,dx, \quad k = 0, 1, 2, 3$$

siendo *f(x)* una función dada. Imponiendo las condiciones obtenemos otro sistema lineal cuya matriz asociada es:

$$\begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{pmatrix}$$

denominada matriz de Hilbert y se obtiene en MATLAB con `hilb(4)`, esta matriz es un caso clásico de mal condicionamiento.

Eemplo 8

La matriz de Hilbert de orden n puede definirse formalmente del siguiente modo:

$$H_n = H(i,j) = \frac{1}{i+j-1} \qquad i, j = 1, \ldots, n.$$

e) Edita una función en MATLAB para generarla.
f) Escribe la matriz de Hilbert de orden 9, H_9. ¿Cuál es su número de condición?
g) Resuelve $H_9\, x = [1,1,1,1,1,1,1,1,1]^T$. Luego incrementa la primera componente del lado derecho en un 1%. ¿qué componente del vector solución cambia más?

Con formato

Solución: a) Teniendo en cuenta que la matriz de Hilbert es simétrica evitamos recorrer todos los elementos:

```
function H=mihilb(n)
for i=1:n
   for j=i:n
      H(i,j)=1/(i+j-1);
      H(j,i)=H(i,j);
   end
end
```

b) `format rat,  H=mihilb(9), cond(H)`

c) `b= ones(9,1); x=H\b, b(1)=1.01; xn= H\b` , observamos que la solución varía mucho. Ello es debido al mal condicionamiento de la matriz de hilbert.
Calculando `abs(x-xn)` obtenemos que la componente x_7 es la que más se altera.

6 Problemas

1.- En el ejemplo de la distribución de temperaturas en una placa hemos supuesto que la temperatura de los nodos de los bordes de la placa es conocida. Vamos a estudiar el caso más general generando una función que obtenga los términos independientes según las condiciones de contorno en los cuatro lados. Halla las temperaturas de la placa para n=5 y m=6 en el caso de que la temperatura en los bordes superior e inferior sea de 50 ºC, en el lateral derecho 100 ºC y en el izquierdo 0ºC.

2.- Los métodos que vimos en el tema 4 para resolver sistemas de ecuaciones lineales se llaman directos. Existe otra alternativa para hallar la solución del sistema, son los llamados métodos iterativos, vamos a ver en que consisten:

Dado el sistema
$$\begin{aligned} 5x_1 - 2x_2 - x_3 &= 1 \\ x_1 - 3x_2 - x_3 &= -2 \\ 2x_1 - x_2 + 4x_3 &= 1 \end{aligned}$$

basta despejar de la primera ecuación x_1, de la segunda x_2 y de la tercera x_3, con ello tendríamos:

$$x_1 = \frac{1}{5}(1 + 2x_2 - 3x_3)$$
$$x_2 = \frac{-1}{3}(-2 - x_1 + x_3)$$
$$x_3 = \frac{1}{4}(1 - 2x_1 + x_2)$$

tomamos ahora una estimación inicial para x_1, x_2 y x_3, por ejemplo (0,0,0), sustituimos estos valores en las expresiones anteriores y obtenemos los nuevos valores de nuestras incógnitas:

$$x_1 = \frac{1}{5},\ x_2 = \frac{2}{3},\ x_3 = \frac{1}{4}$$

Repetimos este proceso las veces que consideremos oportuno y vamos obteniendo distintos valores de las incógnitas, cuando veamos que la diferencia entre una terna de valores y la siguiente es muy pequeña pararemos, y diremos que hemos llegado a la solución aproximada del sistema por el **método de Jacobi**.
Realiza los cálculos en MATLAB.

3.- a)La matriz de Hilbert es bien conocida por estar mal condicionada. Si la matriz de Hilbert de orden n tiene la solución [1,1, ..., 1] , ¿cuál debe ser el vector de términos independientes?.
b)Resuelve el sistema del apartado a) para valores crecientes de n. Registra la magnitud del mayor error como una función de n. ¿El elemento del vector solución que tiene el mayor error es siempre el mismo?.

4.- Utiliza el comando `spdiags` para definir directamente como matriz dispersa la matriz del problema del calor en un rectángulo con las condiciones dadas en el problema resuelto 1, estudia el ahorro que ello supone en el número de operaciones.

5.- En el problema resuelto 2 se puede hacer una mejora, consiste en que una vez has obtenido x1n el nuevo valor de x1, puede utilizarse utilizarlo para calcular x2n en la misma iteración, y a su vez utiliza x1n y x2n para obtener x3n, este nuevo **método iterativo** es el de **Gauss-Seidel**. Escribe las fórmulas para realizar nuevas iteraciones con este método aplicadas al mismo ejemplo y comprueba que se llega a la solución con menos iteraciones.

6.- Por un tubo de secciones interior y exterior cuadradas circula un líquido a temperatura de 200º C. El tubo está parcialmente sumergido en hielo, de manera que la mitad inferior del exterior del tubo está a una temperatura de 0º C. La superficie superior del tubo se mantiene a 100º C. Se supone que la temperatura en los laterales varía linealmente entre los 0º de la parte superior de hielo y los 100º de la cara superior. La sección interior mide 4 cm. de lado y la exterior 10 cm. Estudiar la distribución de la temperatura en una sección transversal del tubo. Utiliza una malla con nudos cada 10 cm.

7.- Sea H(n) = `hilb(n)`, la matriz de Hilbert de orden n. Halla la inversa de $H(n)^tH(n)$ para los valores de n = 5, 10, 15 de dos modos diferentes:

a) Directamente.
b) Observando que $(A^tA)^{-1} = A^{-1}(A^{-1})^t$.
c) Compara con la solución exacta que es `invhilb(n)*(invhilb(n))'`. (La matriz `invhilb(n)` proporciona la inversa exacta de la matriz de Hilbert). Comprueba el número de operaciones efectuadas en los apartados a) y b).

Calcula el número de condición de A^tA, siendo A la matriz de Hilbert de orden n, n = 5, 10, 15, 20, 25, 30. Describe la relación con el ejercicio 3.

8.- Sea $J(n,\lambda)$ la matriz cuadrada de orden n con λ en la diagonal principal y 1s en la diagonal por encima de la diagonal principal. (A dichas matrices se les llama *bloques de Jordan*).

a) Edita una función cuyas entradas sean n y λ y la salida sea $J(n,\lambda)$.

Sean las matrices por bloques $A = \begin{pmatrix} J(5,5) & O \\ O & J(5,1) \end{pmatrix}$ y $B = \begin{pmatrix} J(5,3) & O \\ O & J(5,2) \end{pmatrix}$. Sea X la matriz (única) tal que $AX = B$

b) Halla X mediante $A^{-1}B$. Cuenta el número de operaciones.

c) Sean *U, V, W, Z* cuatro matrices de orden 5 de modo que $X = \begin{pmatrix} U & V \\ W & Z \end{pmatrix}$. Escribe las ecuaciones que verifican *U, V, W y Z.* Halla estas matrices contando el número total de operaciones necesarias.

d) Si A_i es el vector i-ésimo columna de *A,* B_i es el vector i-ésimo columna de *B,* X_i es el vector i-ésimo columna de *X,* entonces $AX_i=B_i$. Halla X_i utilizando \. Cuenta el número de operaciones.

e) Halla los vectores columna de *U, V, W, Z* resolviendo los sistemas correspondientes.

9.- Discretiza el problema de frontera

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = \operatorname{sen}(2x),\ x \in [0,1]$$
$$u(0) = 0$$
$$u(1) = 1$$

sustituyendo la derivada por el cociente de diferencias centrales

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2}(x_i) \cong \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{h^2}$$

Toma $h = 0.05$.

Resuelve la ecuación en derivadas parciales

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0\ ,\ x \in [0,1]\ ,\ y \in [0,1]$$
$$u(x, 0) = u(x,1) = 0$$
$$u(0,y) = u(1,y) = y(1-y)$$

por discretización con h = 0.1.

7 Soluciones

1.-

```
function b=tin(n,m,ts,ti,td,tiz)
% n*m nodos distribuidos en n filas y m columnas
%ts temperatura en el borde superior
%ti temperatura en el borde inferior
%td temperatura en el lado derecho
%temperatura en el lado izquierdo
%En las cuatro esquinas entra calor por dos lados,hemos de sumar las
temperaturas
b(1)=tiz+ts;
b(n)=tiz+ti;
b((m-1)*n+1)=ts+td;
b(n*m)=td+ti;
% temperatura en el lado izquierdo
```

```
b(2:n-1)=tiz;
% temperatura en el borde inferior
b(2*n:n:(m-1)*n)=ti;
% temperatura en el borde superior
b(n+1:n:(m-2)*n+1)=ts;
% temperatura en el borde derecho
b((m-1)*n+2:n*m-1)=td;
b=b';

n = 5; m = 6; A = calor2D(n,m);

xn=A\b; placan = reshape(xn,n,m)
```

placan =

27.4495	39.7386	46.9957	53.0043	60.2614	72.5505
20.0595	34.5091	45.2401	54.7599	65.4909	79.9405
18.2793	32.9983	44.6957	55.3043	67.0017	81.7207
20.0595	34.5091	45.2401	54.7599	65.4909	79.9405
27.4495	39.7386	46.9957	53.0043	60.2614	72.5505

```
surfl(placan)
```

Podemos observar las diferencias de la distribución de temperaturas con los resultados obtenidos en el ejemplo 1, donde se veía que la placa recibía calor por el borde inferior y se iba calentando hacia arriba.

2.- Inicialmente tomamos `x1=0, x2=0,x3=0` y ponemos `iter =0` , para contar el número de iteraciones.
Calculamos los nuevos valores de las incógnitas con las expresiones:

```
x1n=1/5*(1+2*x2-3*x3),
x2n=-1/3*(-2-x1+x3),
x3n=1/4*(1-2*x1+x2)
```

hacemos `iter=iter+1` para contar el número de iteraciones.
Actualizamos los valores de las incógnitas:

```
x1=x1n; x2=x2n;x3=x3n;
```

y repetimos el proceso calculando una nueva iteración. Basta que ejecutes estas tres instrucciones las veces que consideres conveniente, quizá hasta que obtengas una solución similar a la que te proporciona el comando \.
Este es llamado **método iterativo de Jacobi**.

3

Los términos independientes serán $H_n * [1,1,...,1]' = b$ con $b(i) = \frac{1}{i} + \frac{1}{i+1} + ... + \frac{1}{n+(i-1)}$ i=1,..,n. mejor todo con fórmulas

a) Crearemos una función .m que resuelva estos sistemas y devuelva la posición de la componente donde se alcanza el error máximo, observa que el elemento de la solución que tiene mayor error va cambiando.

```
function pos=er3t5(n)
%problema resuelto 3
for i=2:n
   H=hilb(i);   %matriz del sistema
   for j=1:i
    b(j)=sum(H(j,:)); % términos independientes
```

```
     end
        x=H\b'; %solución del sistema, observa que no siempre es la
     esperada [1,1,..,1]
               %observa la advertencia sobre le mal
     condicionamiento de la matriz
        [e,pos(i)]=max(abs(x-ones(size(x)))); %error máximo y
     posición que este ocupa
end
```

6 Valores y vectores propios

Los problemas de valores y vectores propios surgen en diversas ramas de la ciencia, tales como aplicaciones de ingeniería que estudian las vibraciones en determinados sistemas mecánicos o circuitos eléctricos, procesos estadísticos representados mediante cadenas de Markov, problemas clásicos de geometría como la clasificación de cónicas y cuádricas, en el estudio de la ley de gravitación y movimiento de los planetas.
Desde el punto de vista teórico,los problemas de valores y vectores propios, constituyen la técnica usual de análisis y resolución de ecuaciones en diferencias y de ecuaciones diferenciales lineales.
El objetivo de este tema es el cálculo aproximado de valores y vectores propios de una matriz. Los métodos teóricos no suelen ser aplicables a ejemplos reales, por lo que , los métodos numéricos se hacen imprescindibles, como en muchos otros campos de las matemáticas. En este sentido, el método de las potencias, que analizamos aquí, es un método elemental que permite obtener valores y vectores propios con gran precisión.

6.1 Conceptos básicos:

Los vectores sobre los que una matriz actúa de forma especialmente sencilla se llaman vectores propios. Un vector propio de una matriz se transforma en un múltiplo escalar de sí mismo al multiplicarse por la matriz. El factor de escala se denomina valor propio. Si conocemos una base formada por vectores propios de la matriz, podemos describir su acción sobre cualquier vector del espacio de forma relativamente sencilla. Formalicemos primero estos conceptos.
Si A es una matriz nxn diremos que un vector no nulo v es un *vector propio* asociado al *valor propio* λ si verifica:

$$Av = \lambda v$$

que podemos escribir como

$$(A - \lambda I)\, v = 0$$

de modo que para hallar los valores y vectores propios hemos obtener soluciones no nulas de un sistema homogéneo de ecuaciones lineales con un parámetro indeterminado λ.
Para que un sistema homogéneo tenga solución distinta de la trivial, su matriz debe ser singular (no invertible), por lo que se debe verificar

$$\det((A - \lambda I) = 0$$

El primer miembro de esta ecuación puede escribirse como un polinomio en la indeterminada λ, llamado *polinomio característico* de la matriz A

$$p(x) = det(A - xI)$$

y la ecuación

$$p(x) = 0$$

es la llamada *ecuación característica*.
Por lo tanto hallar los valores propios de la matriz A equivale a resolver su ecuación característica, o sea, a hallar las raíces del polinomio característico.
Si alguna raíz λ tiene multiplicidad k, se dirá también que es un valor propio con multiplicidad k.

Ejemplo 1

Halla los valores y vectores propios para la siguiente matriz diagonal.

$$A = \begin{pmatrix} 2 & 0 \\ 0 & -3 \end{pmatrix}$$

La ecuación característica de este sencillo caso será: $(2-\lambda)(-3-\lambda)=0$ con lo que los valores propios son claramente los elementos de la diagonal. Para calcular los vectores propios resolveremos el sistema:

$$(A-2I)v=0;\ \begin{pmatrix} 0 & 0 \\ 0 & -5 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}=\begin{pmatrix} 0 \\ 0 \end{pmatrix};(v_1,v_2)=(\alpha,0),\ \alpha\in\mathbf{R}$$

Análogamente el otro vector propio será cualquier múltiplo del vector de la base canónica $e_2=(0,1)$.

Las matrices que admiten una base de vectores propios se denominan *matrices diagonalizables.*

Ejemplo 2

La matriz

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

tiene un valor propio $\lambda=1,$ que es raíz doble de la ecuación característica, $(1-\lambda)^2=0.$ Los únicos vectores propios son de la forma $(\alpha, 0),\ \alpha\in\mathbf{R}$, por lo que no hay una base de vectores propios. La matriz *A* no es diagonalizable.

Los siguientes resultados de álgebra concernientes a valores y vectores propios nos van a ser muy útiles:

Teorema

Si una Matriz A_{nxn} tiene n valores propios distintos, entonces es diagonalizable, o sea admite una base de vectores propios.

La matriz S cuyas columnas son tales vectores, es invertible y verifica $A = S\,D\,S^{-1}$, siendo D la matriz diagonal con los valores propios de A en la diagonal.

Ejemplo 3

Una máquina se inspecciona cada día para comprobar si está funcionando (F), estropeada (E) o está siendo reparada (R). Cuando está funcionando, hay una probabilidad del 70% de que al día siguiente siga funcionando, del 20% de que esté estropeada y 10% de que esté siendo reparada. Las reparaciones son laboriosas de manera que, si está estropeada, al día siguiente está en reparación en el 80% de los casos y sólo en un 20% está en funcionamiento. Si la máquina se encuentra en reparación, al día siguiente sigue estándolo en el 90 % de los casos, mientras que en el 10% está ya en funcionamiento.

Vamos a modelizar el problema de forma que se pueda predecir el estado de la máquina en un futuro.

La máquina puede estar F funcionando, R en reparación y E estropeada. Las probabilidades de pasar de un estado a otro vienen dadas por:

		Estado actual		
		F	E	R
Estado futuro	F	0.7	0.2	0.1
	E	0.2	0	0
	R	0.1	0.8	0.9

y la *matriz de transición de estados* de un día al siguiente es:

$$A = \begin{pmatrix} 0.7 & 0.2 & 0.1 \\ 0.2 & 0 & 0 \\ 0.1 & 0.8 & 0.9 \end{pmatrix}$$

con lo que el estado de un día $D_k = (F_k, R_k, E_k)^t$ en función del estado el día anterior $D_{k-1} = (T_{k-1}, R_{k-1}, E_{k-1})^t$ puede expresarse de la forma:

$$\begin{pmatrix} F_k \\ R_k \\ E_k \end{pmatrix} = \begin{pmatrix} 0.7 & 0.2 & 0.2 \\ 0.2 & 0.7 & 0 \\ 0.1 & 0.1 & 0.8 \end{pmatrix} \begin{pmatrix} F_{k-1} \\ R_{k-1} \\ E_{k-1} \end{pmatrix}$$

Por recurrencia podemos expresar el estado de un día cualquiera del mes en función del primer día del mes ...

$$D_2 = AD_1$$
$$D_3 = AD_2 = A^2 D_1$$
$$D_4 = AD_3 = A^3 D_1$$
$$\ldots$$
$$D_k = AD_{k-1} = A^{k-1} D_1$$

Por tanto el problema se reduce a calcular potencias de la matriz A, si k es elevado calcular A^{k-1} requiere un gran número de operaciones. Para el caso particular en que A sea diagonalizable el resultado se simplifica de forma notable. .

Por lo que hallar el estado del proceso en este caso en el que la matriz de transición sea diagonalizable queda del siguiente modo:

$$A = SDS^{-1}$$
$$A^{k-1} = SD^{k-1}S^{-1}$$
$$D_k = A^{k-1}D_1 = SD^{k-1}S^{-1}D_1$$

Con lo que denotando $S = (v_1, v_2, v_3)$ los vectores propios, la solución a nuestro problema será

$$D_k = SD^{k-1}S^{-1}D_1 = S\begin{pmatrix} \lambda_1^{k-1} & 0 & 0 \\ 0 & \lambda_2^{k-1} & 0 \\ 0 & 0 & \lambda_3^{k-1} \end{pmatrix} S^{-1}D_1$$

como $S^{-1} D_1$ es un vector constante, lo denotamos $c = S^{-1}D_1 = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$

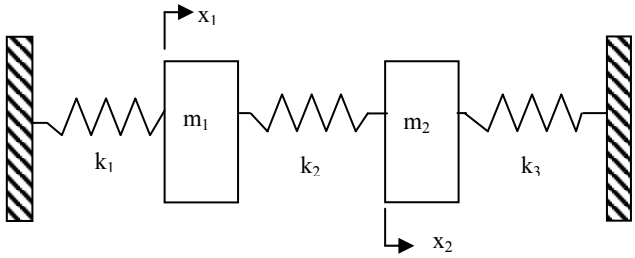
tenemos

$$D_k = (\lambda_1^{k-1}v_1, \lambda_2^{k-1}v_2, \lambda_3^{k-1})\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = c_1\lambda_1^{k-1}v_1 + c_2\lambda_2^{k-1}v_2 + c_3\lambda_3^{k-1}v_3$$

Así pues el estado de la máquina en un día cualquiera *k* se ha obtenido como una combinación lineal de los vectores propios asociados a la matriz de transición de estados, donde los coeficientes dependen de las potencias *k-1* de los valores propios.

Ejemplo 4

La siguiente figura corresponde a un sistema mecánico de masas y resortes.



La ecuación de movimiento *F=m a* junto a la ley de Hooke, que expresa que la fuerza de reacción elástica es proporcional al desplazamiento y de sentido contrario a él, *F = -K x*, nos proporcionan las ecuaciones de movimiento para el sistema:

$$m_1\ddot{x}_1 + (k_1 + k_2)x_1 - k_2x_2 = 0$$
$$m_2\ddot{x}_2 - k_2x_1 + (k_2 + k_3)x_2 = 0$$

donde m_1 *y,* m_2*,* son las masas en las posiciones x_1 *y* x_2 *y* k_i *i = 1, 2, 3* las constantes recuperadoras.

Supongamos una solución armónica cada coordenada

$$x_i(t) = -A_i \sin(wt + \phi_i) \qquad i=1,2.$$

donde *w* es una constante que corresponde a la frecuencia de la oscilación, A_i la amplitud y ϕ_i la fase de la oscilación. De esta forma derivando dos veces tenemos

$$\ddot{x}_i = -w^2 A_i \sin(wt + \phi_i) = -w^2 x_i$$

y sustituyendo en el sistema ya con los valores de $m_1=m_2=1$ *kg*, $k_1=5$ *kN/m*, $k_2=10$ *kN/m* y $k_3= 15$ *kN/m* quedará

$$-w^2x_1 + 15\,x_1 - 10\,x_2 = 0$$
$$-w^2x_2 - 10\,x_1 + 25\,x_2 = 0$$

que expresado en notación matricial nos proporciona un problema de valores propios:

$$\begin{pmatrix} 15 & -10 \\ -10 & 25 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = w^2\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$
$$Kx = \lambda x$$

con $\mathrm{K} = \begin{pmatrix} 15 & -10 \\ -10 & 25 \end{pmatrix}$ kN/m y $\lambda = \mathrm{w}^2$

6.2 Método de las potencias

Teóricamente, los valores propios de una matriz se pueden obtener resolviendo la ecuación característica $p(x) = 0$, y posteriormente los vectores propios resolviendo el sistema lineal asociado. En la práctica $p(x)$ es difícil de obtener y, además, si su grado es elevado, la determinación analítica de las raíces es difícil o imposible, pues existen sólo fórmulas para casos con $n < 5$.
Las técnicas numéricas de búsqueda de raíces tampoco son de gran ayuda en este caso, porque la localización de las n raíces (reales o complejas) de un polinomio es muy sensible a las variaciones de los coeficientes del polinomio, sobre todo en el caso de raíces múltiples. Esto hace que los errores de redondeo afecten mucho a la estimación de los valores propios. El problema se complica aún más al intentar resolver el sistema lineal homogéneo para un valor de λ que no llega a hacerlo "singular".
Recurrimos pues a métodos numéricos alternativos que eviten tener que determinar y resolver explícitamente el polinomio característico. La idea es calcular primero los vectores propios y, a partir de ellos, los valores propios. Los métodos no son sencillos, pero son tan eficaces que, por ejemplo, MATLAB obtiene las raíces de un polinomio hallando los valores propios de una matriz de la que es polinomio característico.
El método de las potencias es una técnica iterativa para obtener de forma aproximada valores y vectores propios de una matriz, que ilustramos con la matriz asociada al proceso de Markov del **Ejemplo 2**.

Comenzamos por elegir un vector inicial u = [1, 1, 1] y repetir los siguientes pasos:
1. Multiplica A*u.
2. Normaliza el vector resultante dividiendo cada componente entre la de mayor valor absoluto.
3. Repite los pasos 1 y 2 hasta que el cambio en el factor de normalización sea despreciable. En este momento, el factor de normalización es una aproximación a un valor propio y el vector resultante un vector propio, ya que se verifica A*u = λ *u.

$$A = \begin{pmatrix} 0.7 & 0.2 & 0.1 \\ 0.2 & 0 & 0 \\ 0.1 & 0.8 & 0.9 \end{pmatrix}$$

u = [1 1 1]'	$A*u$ = [1, 0.2, 1.8]' = 1.8 * [0.556, 0.1111,1]'	**λ = 1.8**
u = [0.556, 0.1111,1]'	$A*u$ = [0.5111, 0.1111, 1.0444]' = 1.0444 * [0.4894, 0.1064, 1]'	**λ = 1.0444**
u = [0.4894, 0.1064, 1]'	$A*u$ = [0.4638, 0.0979, 1.0340]' = 1.0340 * [0.4486, 0.0947, 1]'	**λ = 1.0340**

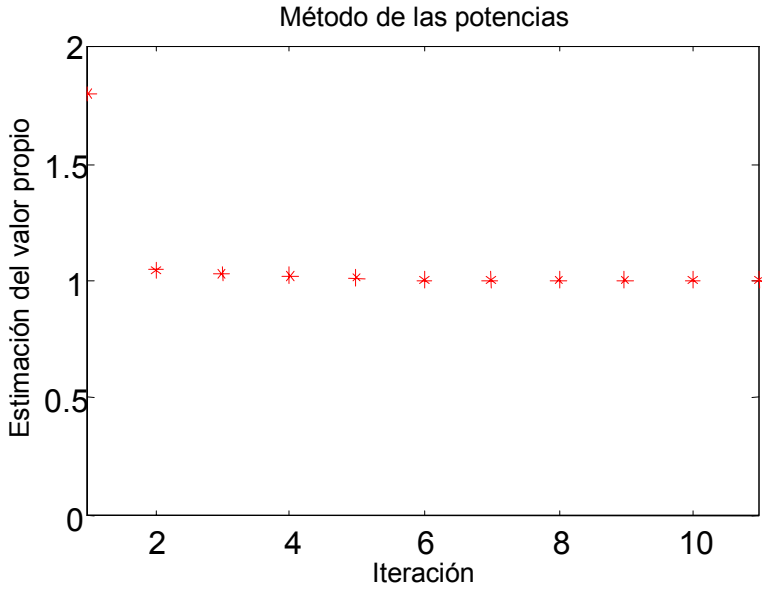
vemos que el valor de λ se va estabilizando, en las iteraciones 9 y 10 tenemos
λ = 1.0020 y **λ = 1.0012.** Por tanto λ es una estimación de un valor propio ya que ***A*u = λ*u***, no obstante si queremos mayor precisión podemos seguir iterando.

En MATLAB puedes realizar este proceso iterativo de una manera muy simple

Inicializamos

```
A=[.7 .2 .1;.2 0 0;.1 .8 .9];
```

```
u=[1, 1, 1]'; lambda = [],k=1;
```
paso 1:
```
v=A*u;
```
paso 2:
```
[mx,i] =max(abs(v));%Localizar componente de mayor valor
absoluto
lambda = [lambda v(i)]   % v(i) es a nueva estimación del valor
propio


% Normalizar v
u = v/v(i)  % Nueva estimación del vector propio k=k+1 % vamos a
realizar una nueva iteración
```
paso 3:
Actualiza las instrucciones del paso 1 y paso 2 para ir obteniendo nuevos valores de v, n y u a partir de los anteriores, hasta llegar al valor propio n y su correspondiente vector propio con la precisión que desees.
Si ejecutas paso 2 y 3 hasta 10 veces obtendrás
```
k =
    10
lambda =
  Columns 1 through 7
    1.0000    1.8000    1.0444    1.0340    1.0206    1.0127    1.0079
  Columns 8 through 10
    1.0050    1.0031    1.0020
```



Este método, aunque no es muy rápido, es muy fácil de programar ya que se realiza un proceso muy sencillo.

6.4 Métodos iterativos

La técnica general para resolver un problema por un método iterativo es la siguiente:

1. Se obtiene una *estimación inicial* x_1 de la solución. En ella se resume la información previa de que disponemos.

En el caso de cálculo de valores propios posiblemente no tengamos información inicial, normalmente se toma el vector columna de tantos unos como columnas haya en la matriz A.

2. Refinamos iterativamente la aproximación, obteniendo valores nuevos a partir de los anteriores, $x_2, x_3, x_4, \ldots$, que converjan a la solución x_*.
3. Establecemos un criterio de parada.

Dependiendo del problema que se resuelva, en nuestro caso, exigiremos que las iteraciones varíen poco.

A x_k lo denominaremos estimación k-ésima y el error de esta aproximación viene determinado por

$$\varepsilon_k = |x_k - x_*|, o \text{ bien } \|x_k - x_*\| \text{ si } x \text{ es un vector}$$

aunque generalmente, como es obvio, no se conocerá el limite x_* y no se podrá calcular ε_k, por lo que se define el error en cada iteración como el incremento entre dos iteraciones consecutivas

$$e_k = |x_{k+1} - x_k|$$

Si esta cantidad es suficientemente pequeña (menor que un argumento de entrada al que llamaremos **tol**), detenemos el algoritmo suponiendo que hemos alcanzado la precisión requerida. Tendremos en cuenta que si el método converge lentamente, estamos subestimando el error; al contrario, si converge rápidamente, el error será seguramente inferior a **tol**.

Conviene también limitar el número máximo de iteraciones, en previsión de que el algoritmo diverja, oscile indefinidamente o converja muy lentamente. Llamaremos **maxiter** al máximo número de pasos que permitimos realizar a un algoritmo iterativo. Si este número se supera, detenemos las iteraciones aunque no se haya alcanzado la convergencia y emitimos un mensaje advirtiendo de este hecho.

Tipos de convergencia

Para determinar la rapidez con que el método iterativo alcanza la solución se estudia la sucesión de incrementos $\{e_k\}$ k = 1, 2, 3, ...

Se dice que la **convergencia es lineal** si los cocientes $\frac{e_{k+1}}{e_k}$ tienden a estabilizarse en un valor constante y $\left|\frac{e_{k+1}}{e_k}\right| < 1$, y la **convergencia es cuadrática** si los cocientes $\frac{e_{k+1}}{e_k^2}$ se aproximan a un valor constante y $\left|\frac{e_{k+1}}{e_k^2}\right| < 1$, en ambos casos cuanto más próximo a 1 es el cociente mas lenta es la convergencia, y cuanto más próxima a cero más rápida. Cuando la tasa de convergencia $\left|\frac{e_{k+1}}{e_k}\right|$ tiende a cero se dice que la **convergencia superlineal.**

Para obtener el tipo de convergencia en el **Ejemplo 2,** procedemos del siguiente modo

```
e = abs(diff(lambda));          % ek = |xk+1 - xk|
m = length(e);
```

```
r = e(2:m)./e(1:m-1)  % r_k = e_{k+1}/e_k
```

Los cocientes incrementales tienden a 0.62 luego tenemos convergencia lineal.

Convergencia del método de las potencias

Analicemos en primer lugar la base teórica del método de las potencias, para ver que si el proceso es convergente, se encuentra el valor propio mayor en valor absoluto y su vector propio correspondiente.
Recordemos los siguientes resultados de Álgebra.

Una matriz A_{nxn} real se dice que es ortogonal si verifica $A^{-1} = A^T$.
Una matriz A_{nxn} compleja se dice que es unitaria si $A^{-1} = \overline{A}^T$.

Teorema espectral
Una matriz A_{nxn} real es simétrica si y sólo si A es diagonalizable mediante una matriz ortogonal, es decir existe una matriz ortogonal P tal que $P^T AP$ es diagonal.
La hipótesis de la simetría puede parecer muy restrictiva. Sin embargo, en la realidad, en muchas aplicaciones prácticas, aparece asociada una matriz simétrica, como puede verse en el **Ejemplo 3.**
Si una matriz A_{nxn}, tiene un valor propio dominante y es diagonalizable existen vectores propios $v_1, v_2, v_3, ..., v_n$ que forman una base de $R^{n\text{ñ}}$, tendremos que cualquier vector x_1 se expresará como combinación lineal de ellos

$$x_1 = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n$$

si multiplicamos por la matriz A, como v_i son vectores propios con valores propios asociados λ_i verifican $A v_i = \lambda_i v_i$, tendremos

$$x_2 = A x_1 = c_1 A v_1 + c_2 A v_2 + \cdots + c_n A v_n$$
$$= c_1 \lambda_1 v_1 + c_2 \lambda_2 v_2 + \cdots + c_n \lambda_n v_n$$

al multiplicar m veces por A se obtiene

$$x_{m+1} = A^m x_1 = c_1 \lambda_1^m v_1 + c_2 \lambda_2^m v_2 + \cdots + c_n \lambda_n^m v_n$$

Por lo tanto si el valor propio λ_i es mayor en valor absoluto que todo el resto, al hacer m suficientemente grande todos los demás sumandos serán despreciables

$$A^m x_1 \to c_i \lambda_i^m v_i$$

que es algún múltiplo del vector propio v_i con factor de normalización λ_i si $c_1 \neq 0$.
Por lo que la convergencia del método depende de los siguientes factores:

- Elegir el vector x_1 inicial de forma que $c_1 \neq 0$, basta con que x_1 no sea perpendicular al vector propio asociado a λ_i, del mismo modo si el vector inicial es casi paralelo al vector asociado al valor propio dominante λ_i, todos los demás coeficientes de la combinación lineal son muy pequeños en relación con λ_i y la convergencia será muy rápida.
- Si otro valor propio tiene el mismo valor absoluto que λ_i o un valor parecido no habrá convergencia o será muy lenta.

Por tanto el método de las potencias tiene la desventaja de que sólo es efectivo si la matriz posee un valor propio dominante y elegimos adecuadamente el vector inicial, cosas que en principio generalmente se desconocen.

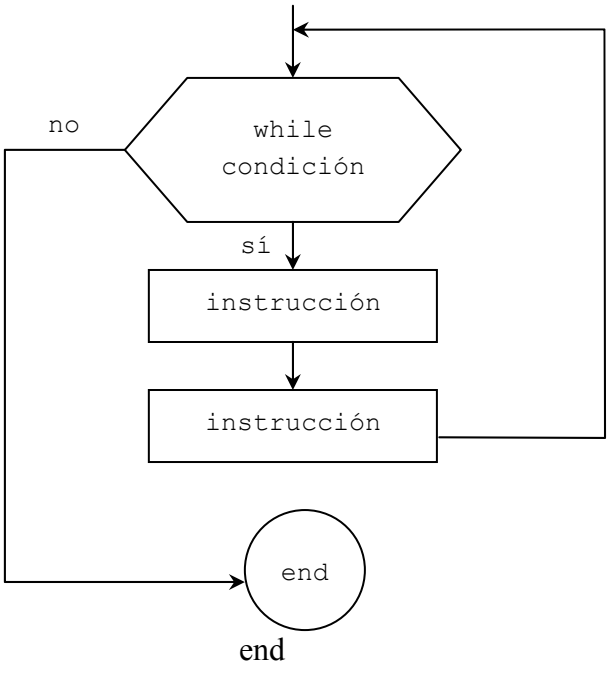
Programación del método de las potencias

Tratamos ahora de programar el método de las potencias en MATLAB. Crearemos un fichero **potencias.m** que, a partir de una estimación inicial de un vector propio, $\mathbf{u_1}$ calcule el valor propio asociado construyendo una sucesión de iterados (en realidad, un vector finito) hasta detectar si hay convergencia o no.
Los argumentos de entrada de **potencias** serán la matriz de la que queremos obtener el valor propio mayor en valor absoluto, la estimación inicial $\mathbf{u_1}$, la precisión con la que deseamos obtener la solución, **tol** y el número máximo de iteraciones, **maxiter.**
Como resultado, obtendremos la estimación final tanto del vector propio como del valor porpio asociado. También interesa conocer la distancia entre los últimos iterados y el número de iteraciones realizadas. Si el programa termina por haber superado el máximo de iteraciones sin alcanzar la convergencia, lo indicará mediante un mensaje.
Las iteraciones **u = A u** y posterior normalización del vector para hallar el nuevo valor de **u y lambda** constituyen el núcleo del algoritmo. En esta parte del programa controlamos la variación de un iterado al siguiente, $incr = \|u_{k+1} - u_k\|$ y contamos el número de iteraciones, **k**. Repetimos el proceso mientras se cumplan dos condiciones:

- que la variación sea mayor que la tolerancia: **incr > tol**
- que no se supere el máximo de iteraciones: **k < maxiter**

La instrucción de MATLAB que veremos a continuación, *while* e *if* permiten programar las iteraciones mencionadas.
while crea un bucle que se repite mientras se cumpla determinada condición. Su sintaxis es

```
while   condición
        instrucciones
```




```
end
```

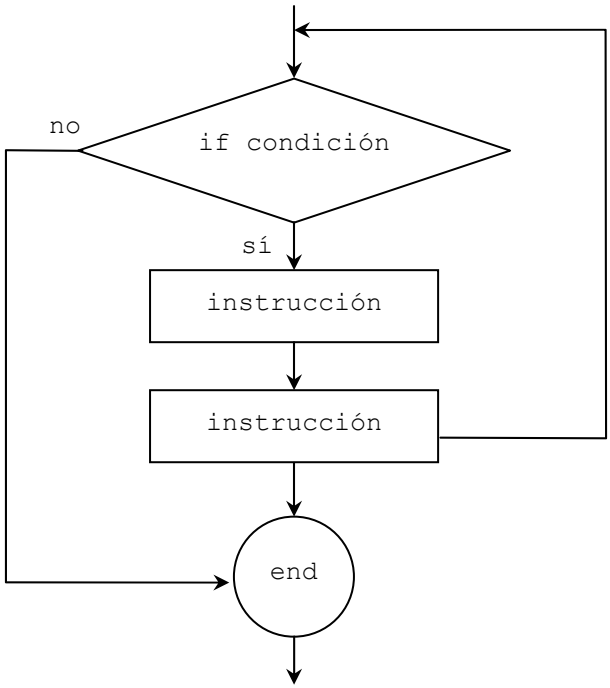
Las instrucciones que contiene el `while` se repiten mientras se verifica la `condición`. Si esta no se verifica, se produce un salto hasta la orden que sigue al `end`.

Para que el bucle no sea indefinido, las instrucciones del `while` modifican los términos de la condición, de modo que en algún momento deje de verificarse y entonces el bucle se termina.

La orden *if* proporciona bifurcaciones condicionales, su sintaxis es

```
if condición
    instrucciones
end
```

las instrucciones se realizan si la condición se verifica.



Otra estructura de control útil es la *bifurcación condicional*. Las instrucciones que contiene el `if` se ejecutan si se verifica la `condición`. Si ésta no se verifica, se produce un salto hasta la orden que sigue al `end`.

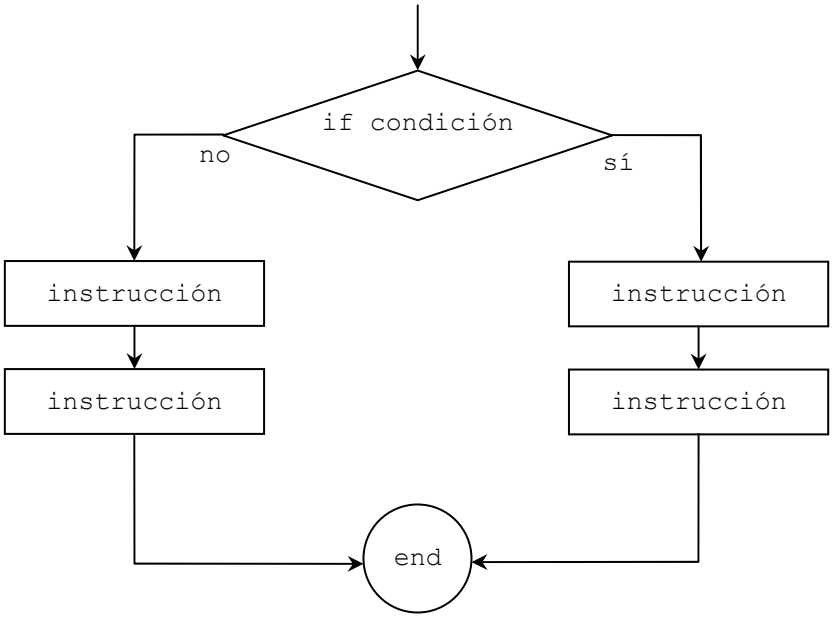
La instrucción `else` permite introducir instrucciones que se ejecutan si la condición es falsa. La sintaxis en este caso es

```
if condición
      instrucciones cierta
else
      instrucciones falsa
end
```

Se pueden anidar varios `if`. La instrucción `elseif` es una abreviatura de `else if`.

En el programa **potencias.m**, utilizaremos una bifurcación condicional para avisar que no ha habido convergencia si, al finalizar el bucle, la distancia entre los últimos iterados es mayor que la tolerancia.

Veamos algunos detalles sintácticos útiles para formular condiciones que aparecen en la definición de un bucle o una bifurcación condicional. MATLAB reconoce expresiones lógicas, asociándoles el valor **1** si son verdaderas y el valor **0** si son falsas. No debe sorprendernos a estas alturas que lo haga vectorialmente.



Las comparaciones entre dos variables se codifican de forma natural mediante los signos de desigualdad e igualdad. Prueba

```
a = [-1 0 1],
a > 0            % devuelve 0 0 1
a >= 0                % devuelve 0 1 1
a == 0                % devuelve 0 1 0
a ~= 0                % devuelve 1 0 1
```

Notamos que la igualdad se denota con dos signos 'igual', para distinguirla de la asignación, que usa un solo 'igual'.
La desigualdad se denota con el signo 'igual' precedido de la 'tilde', que se obtiene con la combinación de teclas [Alt]+126.
Para combinar varias condiciones utilizamos los operadores lógicos:

- Conjunción: A&B Es cierta cuando **A** y **B** lo son.
- Disyunción: A|B Es cierta cuando alguna de las dos lo es.
- O exclusiva: xor(A,B) Es cierta cuando ambas o ninguna lo son.
- Negación: ~A Es cierta cuando **A** es falsa.

A y **B** pueden ser valores cualesquiera, no necesariamente 0 ó 1. En este caso, todo valor no nulo se considera verdadero y viceversa.
Utilizamos comparaciones en los métodos iterativos para comprobar que la distancia entre iterados es grande, **incr>tol**, y que no se ha alcanzado aún el tope de iteraciones, **k<maxiter**. Efectuamos iteraciones mientras se cumplen simultáneamente los dos requisitos, por lo que la condición que controla el bucle será la conjunción:
(incr > tol) & (k < maxiter)

Algoritmo potencias

Entrada: **A** la matriz, **u** el vector inicial, **tol** la tolerancia, **maxiter** nº máximo de iteraciones.

Salida: `lambda` sucesión de iterados, el último valor será la aproximación al valor propio de mayor valor absoluto, **u** vector propio asociado, **k** nº de iteraciones realizadas.

Proceso:

Inicializar
lambda = []; k=1;
incr= tol+1; %hacemos el error mayor que la tolerancia para que entre en el bucle
Mientras no converja o no hayamos llegado a maxiter
    Multiplicar A*u.
    Normalizar el vector resultante dividiendo cada componente entre el mayor en magnitud.
    Calcular el error o incremento
    incrementar el contador de iteraciones
fin mientras
Si al salir del bucle no se ha alcanzado la tolerancia deseada
    escribir un mensaje indicando que se necesitan más iteraciones
fin del condicional

Ejecuta la función potencias para hallar un valor propio de la matriz del **Ejemplo 2**.tomando el vector u = [1,1,1]' , tol de una milésima y un máximo de 30 iteraciones.

```
A=[.7 .2 .1;.2 0 0;.1 .8 .9];
u=[1 1 1]'; tol =0.001; maxiter=30;
[lambda,u,k]=potencias(A,u,tol,maxiter)
```

Se tiene el valor propio λ=1.0012 y el vector propio asociado u = [0.3871, 0.0776, 1] en 11 iteraciones.

Ejemplo 5

Aplica el método de las potencias a la matriz B, inténtalo primero con un número pequeño de iteraciones de forma que no se alcance la convergencia, luego lo aumentas y estudias el tipo de convergencia que se produce.

$$B = \begin{pmatrix} -4 & 3 & 2 \\ 4 & 1 & 0 \\ 0 & 3 & -1 \end{pmatrix}$$

```
B=[-4 3 2; 4 1 0; 0 3 -1];
u=[1 1 1]'; tol =0.001; maxiter=4;,
[lambda,u,k]=potencias(B,u,tol,maxiter)
```

Observas que el programa da la solución que ha obtenido hasta ese momento advirtiendo que no se alcanza la tolerancia deseada ya que muestra el mensaje de que se necesitan más iteraciones. Fíjate que el contador de operaciones esta a 5, ¿cómo explicas esto si maxiter es 4?

Probamos ahora

```
u=[1 1 1]';  maxiter=20;
[lambda,u,k]=potencias(B,u,tol,maxiter)
```

Todavía no se alcanza la solución, probamos otra vez

```
u=[1 1 1]';  maxiter=40;
```

```
[lambda,u,k]=potencias(B,u,tol,maxiter)
```

Observa que se necesitan 30 iteraciones, para tolerancia de una milésima, esto ratifica que la convergencia es lenta, la razón de convergencia esta cerca de 0.7.

```
e = abs(diff(lambda));
m = length(e);
r = e(2:m)./e(1:m-1)
```

Método de las potencias inverso

Como hemos visto en los ejemplos previos el método de las potencias obtiene el valor propio mayor en valor absoluto, vamos a profundizar un poco más para poder obtener cualquier valor propio, para ello vamos a hacer uso de la siguiente propiedad.

Si A tiene un valor propio λ_i y es una matriz no singular entonces la matriz A^{-1} tiene el valor propio $1/\lambda_i$.

Por lo que si en el método de las potencias trabajamos con A^{-1} obtendremos el valor propio mayor en valor absoluto λ_i y por tanto $1/\lambda_i$ .será el menor valor propio en valor absoluto para la matriz A, el vector propio correspondiente es el mismo, no requiere cambios ya que

$$A^{-1}v = \lambda_i v \text{ multiplicando por } A$$
$$v = \lambda_i A \text{ v multiplicando por } 1/\lambda_i$$
$$\frac{1}{\lambda_i} v = A\, v$$

es decir v es también vector propio de A asociado al valor propio $1/\lambda_i$

Ejemplo 6

Vamos a obtener otro valor propio de la matriz A del **Ejemplo 2**, concretamente será el menor en valor absoluto.

```
A=[.7 .2 .1;.2 0 0;.1 .8 .9];
u=[1 1 1]'; tol =0.001; maxiter=30;
[lambda,u,k]=potencias(inv(A),u,tol,maxiter)
```

Se obtiene -31.5831 en la iteración 6 como valor propio de A^{-1}, el de A será -1/31.5831 que puedes obtener

```
1/lambda(6)
```

valor propio -0.0317 y vector u = [-0.158, 1.000, -0.8417].

Desplazamiento con el método de las potencias

Ya sabemos como obtener los valores propios de mayor y menor valor absoluto, veamos ahora como obtener los valores propios con un valor absoluto intermedio entre estos.

Utilizamos ahora la siguiente propiedad

Si A tiene un valor propio λ_i entonces la matriz $A+kI$ tiene el valor propio $\lambda_i + k$

Con lo cual si suponemos que se desea obtener un valor propio muy cercano a un valor k, tomamos la matriz desplazada $A-kI$ que tendrá un valor singular muy cerca de cero *y* por tanto la inversa de esta matriz desplazada tendrá un valor propio mucho mayor en

magnitud que cualquier otro lo que hará que la convergencia sea mucho más rápida, y deshaciendo los cambios tendremos el valor propio λ_i que buscábamos.

Ejemplo 7

En la matriz del **Ejemplo 2** hemos obtenido ya los valores propios 1.012 y -0.0317, para obtener el valor propio que nos falta que será un valor intermedio podemos hacer un desplazamiento de la matriz restando el punto medio de estos valores y proceder como hemos indicado.
(Notemos que podríamos obtener este valor propio de forma inmediata ya que la suma de los tres valores propios coincide con la traza de la matriz)

```
d= (1.012 + -0.0317)/2 %punto medio de los valores propios ya obtenidos
Ad=A-d*eye(3); % matriz des plazada
u=[1 1 1]'; tol =0.001; maxiter=30;
[lambda,u,k]=potencias(inv(Ad),u,tol,maxiter)
```

Obtenemos el valor propio 7.0669 en la iteración 11 con lo que el correspondiente valor propio de A será

```
1/lambda(11)+d
```

Así pues el valor propio de A que nos faltaba es 0.6317 y el vector propio asociado
u = [-0.7595, -0.2405, 1]

Como hemos hecho notar, esta técnica de desplazamiento del método de las potencias también puede utilizarse para acelerar la convergencia del proceso, pero para ello necesitamos tener información sobre en torno a que números se encuentran lo valores propios, para poder realizar el desplazamiento hacia ellos. Esta información nos la proporciona el siguiente resultado que establece que los valores propios se encuentran en círculos cuyos centros están en los elementos de la diagonal a_{ii} con radio igual a la suma de los valores absolutos de los otros elementos de la fila i.

Teorema de Gerschgorin

a) *Todo valor propio λ de A_{nxn} satisface al menos una de las siguientes desigualdades:*

$$|\lambda - a_{ii}| \le r_i \text{ en donde } r_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad \text{i=1,...,n}$$

esto es, cada valor propio está, por lo menos en uno de los discos con centro a_{ij} y con radio r_i.

b) *Si la unión de cualesquiera k de estos círculos no intersecta a los n-k restantes, entonces habrá precisamente k valores propios en esta unión.*

Es obvio que este teorema será útil si la matriz presenta fuerte dominancia en la diagonal.

Ejemplo 8

Vamos a intentar acelerar la convergencia en el **Ejemplo 3,** aunque la matriz no es diagonal dominante**,** los valores propios estarán en los círculos -4 ± 5, 1 ± 4 y -1 ± 3, lo hacemos desplazando primero por -4

```
B=[-4 3 2; 4 1 0; 0 3 -1];
Bd= B+4*eye(3);
u=[1 1 1]'; tol =0.001; maxiter=40;,
[lambda,u,k]=potencias(inv(Bd),u,tol,maxiter)
```

Para encontrar el valor propio de B

```
1/lambda(14)-4
```

hemos reducido el número de iteraciones a la mitad, pues inicialmente había necesitado 30 iteraciones y ahora lo obtiene en 14.

Valores y vectores propios en MATLAB

Existen funciones incorporadas a MATLAB que nos permiten calcular tanto el polinomio característico de una matriz A, como sus valores y vectores propios.

Si A es una matriz nxn,

P = poly (A) proporciona los coeficientes del polinomio característico de A.

r=roots (P) nos daría las raíces de P y por tanto los valores propios de A.

r=eig(A) nos da directamente los valores propios de A.

[S, D] = eig(A), los parámetros de salida son dos matrices, la primera tiene por columnas los vectores propios de A y la segunda es una matriz diagonal con los valores propios de forma que $A = S\ D\ S^{-1}$.

En el **Ejemplo 1** para tener los valores propios basta hacer

```
A=[2 0;0 -3]; lambda=eig(A)
```

y los vectores propios

```
[S, D] = eig(A)
```

Comprobemos los resultados que hemos obtenido en el **Ejemplo 2** con el método de las potencias utilizando ahora los comandos de MATLAB.

```
A=[.7 .2 .1;.2 0 0;.1 .8 .9];
[V,D]=eig(A)
V*D*inv(V) %comprobamos que obtenemos A
```

Notemos que las matrices V y S no coinciden pues los vectores propios no son únicos, en cada caso se han obtenido de formas diferentes, ahora bien , puedes comprobar que los vectores propios obtenidos asociados al mismo valor propio son proporcionales, es decir generan el mismo subespacio propio.

La primera columna de S correspondía al valor propio cercano a 1 que en V se encuentra en al tercera columna, puedes ver la proporcionalidad haciendo

```
S(:,1)./V(:,3)
```

Apéndice: Descomposición en valores singulares

La descomposición de una matriz empleando matrices unitarias u ortogonales es muy útil para diversas aplicaciones, en otros temas trataremos la descomposición QR de una matriz y la descomposición de Shur, vamos a ver ahora la descomposición de una matriz en valores singulares, especialmente útil para la compresión de datos.

Teorema:

a) Toda matriz real A, *pxq* puede descomponerse como $A = U \sum V^T$, en donde U es *pxp* unitaria y ortogonal, V es *qxq* unitaria, y ortogonal y $\sum$ es *pxq* y "diagonal" ya que

$\sum_{ij} = 0$ para i $\neq$ j, y $\sum_{ii} = \sigma_i \geq 0$ siendo $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_s$ donde s=min $\{p,q\}$

b) Los números σ_i se llaman los valores singulares de *A*, y las columnas de *U* y *V* se denominan vectores singulares derecha e izquierda de A, se verifica que σ_i^2 son los valores propios de $A^H A$ (quizá agregando algunos ceros) y los vectores propios asociados son las columnas u_i de *U*, es decir $Av_i = \sigma_i u_i$ para $1 \leq i \leq s$.

En la práctica, el cálculo de una descomposición en valores singulares se hace con técnicas de programación, esta descomposición contiene gran información sobre la matriz *A*.

Valores singulares y rango

La descomposición en valores singulares se puede usar para determinar el rango de una matriz. Más importante aún es que se puede usar para analizar los distintos rangos que puede tener la matriz si sus elementos estuvieran sometidos a errores (por ejemplo de medición o modelado) que podrían ocasionar creer que el rango es mayor de lo que en realidad puede ser.

Teorema:

a) El rango k de A es igual al número de valores singulares de *A* diferentes de cero.

b) Si A es una matriz real *pxq*, de rango *k*, para un $r < k$, la matriz A_r de rango r que hace mínimo a $\|A - A'\|_2$ entre todas las matrices pxq, A', de rango r viene dada por

$$A_r = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + ... + \sigma_r u_r v_r^T$$ y el mínimo es $\|A\text{-}A_r\|_2 = \sigma_{r+1}$

La capacidad de la descomposición en valores singulares para obtener aproximaciones de bajo rango de una matriz dada, puede ser útil en la compresión de datos, por ejemplo al enviar fotografías desde el espacio. Una cámara de un satélite fuera de la Tierra, puede enviarnos miles de fotos. Supongamos que para realizar el envío se discretiza o digitaliza la foto dividiendo la imagen en muchos cuadros pequeños y asignando un tono de un mismo color a cada uno de ellos. Por ejemplo si hacemos un sobre la foto un cuadriculado de 1000×1000 tendremos que mandar a la Tierra 1.000.000 de enteros para cada foto, y así reconstruir la foto en la Tierra.

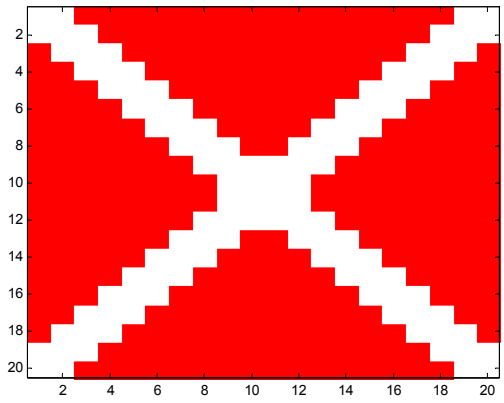
El problema es que esto son muchos datos si tenemos a su vez gran cantidad de fotos. Necesitamos pues un método para comprimir datos con el objetivo de transmitir menor cantidad. Un método es utilizar la descomposición en valores singulares de la matriz 1000×1000 A y ver si una aproximación de bajo rango

$$A_r = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + ... + \sigma_r u_r v_r^T$$

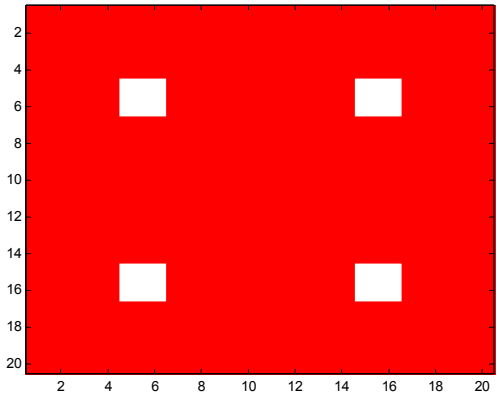
representa adecuadamente la imagen; ésta se transmitiría a la Tierra en la forma de los *2r* vectores $\mathbf{u_i}$, y $\mathbf{v_i}$ y los **r** valores singulares. Fijémonos que por ejemplo si *r = 5* para reconstruir la foto es suficiente con el envío de $2 \times 5 \times 1000 + 5 = 10005$ datos por foto e vez del 1.000.000 inicial, lo que supone un ahorro de más del 99%.

Ejemplo 9

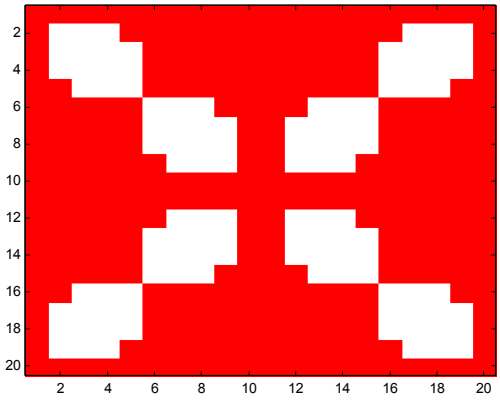
Vamos a practicar el método de comprensión de datos que acabamos de exponer con un ejemplo sencillo en el a partir de la siguiente imagen discretizada de la foto de una X se obtienen las aproximaciones de grado 1,2 y 3.



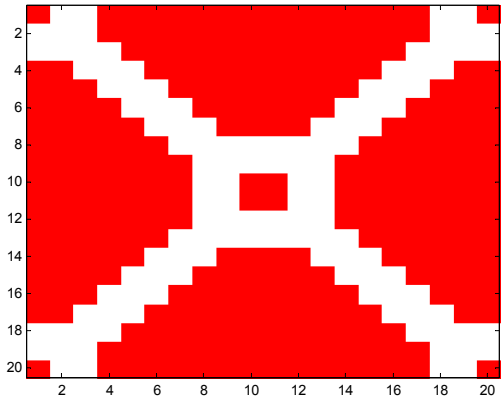
a) Imagen discretizada



b) Aproximación de grado 1



c) Aproximación de grado 2



d) Aproximación de grado 3

La imagen discretizada se crea con el siguiente fichero.m, con el que se obtiene la matriz de datos A de *20x20.*

```
function A=fotox
dp=ones(1,20);
d1=ones(1,19);
a=diag(dp)+diag(d1,-1)+diag(d1,1);
b=fliplr(a);
A=max(a,b);
Af=A+1;
image(Af)
colormap flag
```

```
A=fotox;
```

Calculamos la descomposición en valores singulares de la matriz A.

```
[V,S,U]=svd(A);
S=diag(S)
```

Observamos que A es de rango 10 ya que los valores singulares son

$$\sigma_1 = 5.8, \quad \sigma_2 = 5.3 \quad \sigma_3 = 4.6 \quad \sigma_4 = 3.6 \quad \sigma_5 = 2.5 \quad \sigma_6 = 1.8 \quad \sigma_7 = 1.4$$
$$\sigma_8 = 1.3 \quad \sigma_9 = 0.6 \quad \sigma_{10} = 0.3 \quad \text{y los otros } \sigma_i \approx 0$$

La aproximación de rango 1 será $A_1 = \sigma_1 u_1 v_1^T$ y $\|\text{A-A}_1\|_2 = \sigma_2 = 5.3$

```
A1=S(1)*U(:,1)*V(:,1)';
image(round(A1)+1)
colormap flag
```

L aproximación de rango 2 será $A_2 = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T$ y $\|\text{A-A}_2\|_2 = \sigma_3 = 4.6$

```
A2=A1+S(2)*U(:,2)*V(:,2)';
image(round(A2)+1)
colormap flag
```

L aproximación de rango 3 será $A_3 = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \sigma_3 u_3 v_3^T$ y $\|\text{A-A}_3\|_2 = \sigma_4 = 3.6$

```
A3=A2+S(3)*U(:,3)*V(:,3)';
image(round(A3)+1)
colormap flag
```

Observamos que la aproximación de grado 1 no es satisfactoria, la de grado 2 ya nos proporciona una idea muy próxima de la foto inicial con sólo 82 datos y la de rango 3 se puede decir que es muy buena ya que con 123 datos se aprecia la foto inicial, que contenía información de 400 datos.

7 Problemas

1.- a)Aplica el método de las potencias para calcular el valor propio dominante de la matriz *A, a*sí como un vector propio asociado.

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

b) Obtén con el método inverso el valor propio de menor valor absoluto y un vector propio asociado.
c) Utiliza los círculos de Gerschgorin y la técnica de desplazamiento para acelerar la convergencia del método.
d) Obtén la descomposición de A de la forma $A = S\ D\ S^{-1}$, siendo D la matriz diagonal con los valores propios de A en la diagonal
e) Compara los resultados obtenidos utilizando los comandos de MATLAB

2.- Utiliza el comando **eig** de MATLAB para resolver el **Ejemplo 2**, suponiendo que inicialmente la máquina está en perfecto estado, D_1 = [1,0,0] averigua el estado al cabo de una semana y al finalizar el mes. ¿Crees qué se puede determinar el estado a largo plazo?.

3.- Una red de comunicaciones transmite dígitos binarios, 0 ó 1. Al pasar por la red, existe una probabilidad q de que el dígito que se recibe en la siguiente estación sea incorrecto. Si X_1 denota el dígito binario que entra al sistema, X_2 el dígito binario registrado después de la primera transición y así sucesivamente. Plantea el problema como una sucesión de 4 estados, obtén la matriz de transición de un paso a otro como en el ejemplo 2, para analizar la validez de la transmisión a largo plazo.

4.- Resuelve el **Ejemplo 3** correspondiente a un sistema mecánico de masas y resortes con los datos que en el mismo figuran.

5-. Una matriz **persimétrica** es una matriz que es simétrica alrededor de ambas diagonales; es decir

$$A=(a_{ij})_{nxn} \text{ es persimétrica si } a_{ij}=a_{ji}=a_{n+1-i\ \text{n+1-j}}\ \forall\ i=1,...,n \text{ y } j=1,...,n.$$

Muchos problemas de la teoría de la comunicación tienen soluciones que involucran a los valores y vectores propios de matrices persimétricas. Por ejemplo el vector característico correspondiente al valor característico menor de la matriz persimétrica de 4×4

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

da la respuesta impulsiva de un canal de energía-unidad para una secuencia de error dada de longitud 2 y subsecuentemente, el peso mínimo de cualquier secuencia de error posible.

Utiliza el método de las potencias y sus variantes para obtener el menor valor característico de la matriz A y su vector propio correspondiente.

6.- Un sistema dinámico lineal puede presentarse por las ecuaciones

$$\frac{dx}{dt}=A(t)x(t)+B(t)u(t),\quad y(t)=C(t)x(t)+D(t)u(t)$$

donde *A* es una matriz variable de, B es una matriz variable de, C es una matriz variable de, una matriz variable de, **x** es un vector variable n-dimensional, **y** es un vector variable m-dimensional y **u** es un vector variable r-dimensional. Para que el sistema sea estable, todos los valores característicos de la matriz *A* deben tener una parte real no positiva para toda *t*.

Determina la estabilidad del sistema en los siguientes casos:

a) $$A(t)=\begin{pmatrix} -1 & 2 & 0 \\ -2.5 & -7 & 4 \\ 0 & 0 & -5 \end{pmatrix}$$

b) $$A(t)=\begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -2 & 1 & 0 \\ 0 & 0 & -5 & 1 \\ -1 & -1 & -2 & -3 \end{pmatrix}$$

7.- Elige una letra del abecedario, la inicial de tu nombre o de alguno de tus apellidos, y realiza el proceso seguido en el apéndice del tema para comprimir datos. Para ello debes obtener una foto discretizada de la letra mediante una matriz de 20×20 y realizar la descomposición en valores singulares de esta matriz para obtener las aproximaciones de rango mínimo que permitan identificar la foto inicial.

8 Soluciones

2.-

```
A=[.7 .2 .1;.2 0 0;.1 .8 .9];
[S,D]=eig(A)
```

supongamos que el primer día del mes la máquina esta en perfecto estado, al cabo de 15 días tendremos

$$D_{15} = (\lambda_1^{15-1}v_1, \lambda_2^{15-1}v_2, \lambda_3^{15-1}) \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = c_1\lambda_1^{15-1}v_1 + c_2\lambda_2^{15-1}v_2 + c_3\lambda_3^{15-1}v_3$$

con $c = S^{-1}D_1 = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$ y λ_i los valores propios de A

```
D1=[1 0 0]';
c=inv(S)*D1;
D7=c(1)*D(1,1)^6*S(:,1)+c(2)*D(2,2)^6*S(:,2)+c(3)*D(3,3)^6*S(:,3
)
```

y al finalizar el mes

```
D30=c(1)*D(1,1)^29*S(:,1)+c(2)*D(2,2)^29*S(:,2)+c(3)*D(3,3)^29*S
(:,3)
```

A largo plazo podemos simplificar el resultado ya que existe un valor propio dominante

```
c(3)*D(3,3)^29*S(:,3)
```

y los otros dos sumandos son prácticamente nulos.

```
c(2)*D(2,2)^29*S(:,2)
c(1)*D(1,1)^29*S(:,1)
```

Por lo que a largo plazo el proceso alcanza un estado estacionario sin importar el estado inicial del que partimos, que coincide con un vector propio del valor propio $\lambda = 1$, en este caso :dicho estado estacionario será:

```
c(3)*S(:,3)
```

```
ans =
    0.2632
    0.0526
    0.6842
```

Es decir, independiente de cómo se encuentre la máquina un determinado día al cabo de un tiempo la probabilidad de que este funcionando es aproximadamente del 26%, la probabilidad de que este estropeada es sólo del 5% y en el 68% de los casos la máquina está siendo reparada.

4.-Para resolver el problema de valor propio $\begin{pmatrix} 15 & -10 \\ -10 & 25 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = w^2 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ hallaremos los valores propios de La matriz K;

```
K=[15 -10;-10 25]
```

```
K =
    15   -10
```

```
   -10    25
```

```
lambda=eig(K)
```

```
lambda =
    8.8197
   31.1803
```

y finalmente w = $\sqrt{\lambda}$

```
w=sqrt(lambda)
```

```
w =
    2.9698
    5.5839
```

A partir de estos valores e imponiendo las condiciones iniciales $x_1(0)=0$ y $x_1'(0)=1$ obtenemos las ecuaciones de movimiento:

$$x_1(t) = -A_1 \sin(wt+\phi_1)$$

$x_1(0) = -A_1 \sin(\phi_1) = 0$ de donde $A_1 = 0$ o bien $\phi_1 = 0$, supongamos este caso. Para determinar A_1 utilizaremos $x_1'(0)=1$

$$x_1'(t) = -wA_1 \cos(wt+\phi_1)$$

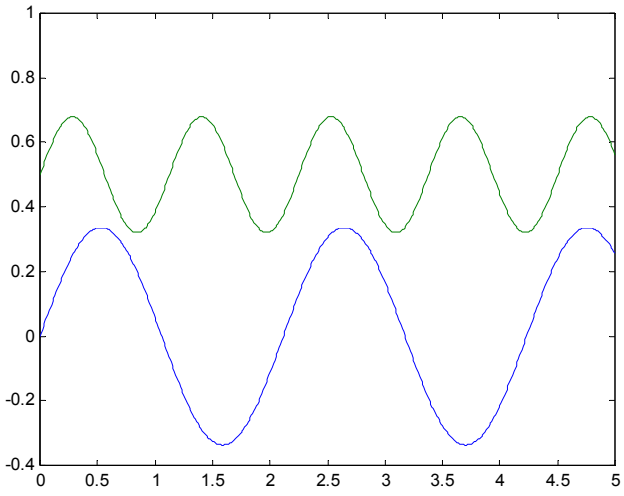
Para $t = 0$ tendremos 1= - 2.9698 A_1 por lo que $A_1 = -1/w_1$.
Análogamente se obtiene A_2. .

```
A=-1./w
```

```
A =
   -0.3367
   -0.1791
```

Y por tanto queda determinado el movimiento que representamos a continuación:

```
t=0:0.01:5;
x1=-A(1).*sin(w(1)*t);
x2=-A(2).*sin(w(2)*t);
plot(t,[x1;x2+0.5])
```



```
diary on

% ------------------------------------------------------
%  VALORES PROPIOS Y VECTORES PROPIOS
% ------------------------------------------------------

% Se tiene una aplicacion lineal representable mediante una matriz
%   ej: giro en el plano, proyeccion en un plano, ...

% la matriz actua anteponiendose al punto sobre el cual actua

%ej: proyeccion en plano XY

A = [1 0 0; 0 1 0; 0 0 0];
x1 = [2; 3; 4];
x2 = A*x1

x2 =

     2
     3
     0



clear

%lo ideal seria que la matriz fuera diagonal, reduciendo enormemente la dificultad de aplicar la matriz a un punto
%si la aplicacion lineal se representa mediante una diagonal, ejerce el cambio independientemente a cada componente del punto.
% Obviamente e suna forma mas sencilla de representar la matriz

%PAra llegar a esta forma se utilizan los valores y vectores propios

%los valores propios coinciden precisamente con los elementos de la diagonal

%Definicion:
%    Sea A (n*n)
%        v (!= 0) es vector propio de A si: A*v = hv   (h un numero real)
%                          normalmente se escribe como lambda, pero como el teclado no esta ne griego, usare la h, que 'se parece' :)

% si tenemos una base del espacio formada por vectores propios, entonces la matriz de la A. L. referida a esa base tiene forma diagonal
%   ademas, dicha matriz (A) viene dada por A = S * D * inv(S)
%   donde S es una matriz formada por los vectores propios

%la forma numerica de calcular dicha matriz no tiene nada que ver con la teoria

% basicamente hay 2 procedimientos
```

% - uno consiste en transformar la matriz en diagonal mediante una serie de transformaciones, y el registrro de dichas transformaciones da lugar a la matriz S de los vectores propios

```
>> % El procedimiento analitico para obtener valores y vectores propios es totalmente opuesto
>> % se plantea el problema como un sistema de ecuaciones lineales
>> %        A X = h X
>>
>> %que queda como...
>>
>> %   [a11-h  a12  a13;        [x1;           [0;
>> %   a21    a22-h a23;         x2;    =  h .   0;
>> %   a31    a32   a33-h]       x3]            0]
```

se busca que el sistema trenga infinitas soluciones
por lo tanto se busca que el determinante sea cero
al calcular el determinante se obtiene un polinomio con h
se calculan las raices de dicho polinomio y se obtienen las h1, h2, .. hi que van en la matriz diagonal D
luego se buscan los vectores para cada h
y de ahi se puede escribir la matriz S, con los vectores propios en las columnas

A nosotros este sistema no nos sirve de mucho

la construccion del polinomio e sun poco incomoda
y sobre todo la obtencion de las raices esta sujeta a error
y encima si quiero resolver el sistema no homogeneo con solucion no trivial

Analiticamente esta todo perfecto, pero numericamente el proceso tiene dificultades

Numericamente lo que se hace es obtener D mediante usa serie de procedimientos a partir de la matriz A, durante los cuales se pueden obtener los vectores propios

¿Que hace matlab para calcular las raices de un polinomio? no os lo podeis ni imaginar
construye una matriz cuyo polinomio caracteristica sea el polinomio
obtiene los vectores y valores propios de esa matriz
y el resultado de los valores propios se los asigna a las raices del polinomio

--- Aplicacion interesante de valores propios que ademas nos da la idea del metodo nuemrico que vamos a utilizar es:

:::: El calculo de potencias de una matriz ::::

---

que ventaja tiene trabajr con valores propios para hallar las potencias?
pues muy sencillo

para elevar al cuadrado una matriz hay que determinar n^2 elementos, cada uno de ellos requeire n operaciones
el total es n^3 operaciones cada vez que se multiplica una matriz por otra
ahora bien, si la matriz fuese diagonal, para multiplicarla bastarian solo n operaciones (cada elementos de la diagonal)

Procedimiento:
- Diagonalizar la matriz
- Elevar a la potencia que se requeira

si se tiene A = S D inv(S)
A^2 = (S D inv(S) S D inv(S) = S D^2 inv(S)

luego ya solo queda deshacer el cambio para obntener el resultado en los terminos iniciales

sirve para cualquier potencia

A^k = S D^k inv(S)

La potencia de matrices es muy util para ver lo que ocurre en diversas situaciones a la larga (procesos de MARKOV)

Ejercicio: Aplicar lo que veamos hoy al primer problema que hicimos

---
Ejemplo:
---
ver ejemplo en las transparencias de la practica 6
estado de una maquina en uan fabrica al cabo de k dias
D2 = A D1
D3 = A D2 = A^2 D1

Dk = A Dk-1 = A^(k-1) D1

EN MATLAB

```
A = [0.7 0.2 0.2; 0.2 0.7 0; 0.1 0.1 0.8];
u = [1 1 1]' %hace la traspuesta para ser mas rapido...
lamda = []; %valores propios
k = 1; %contador de iteraciones


%PROCESO
v = A*u

v =

   1.1000
   0.9000
   1.0000

u = v/v(1) %para conseguir el primer elemente igual a 1
```

```
u =

    1.0000
    0.8182
    0.9091



plot(u)
hold on

%volvemos a repetir los pasos
v = A*u

v =

    1.0455
    0.7727
    0.9091

u = v/v(1)

u =

    1.0000
    0.7391
    0.8696

plot(u,'g')

%vamos a repetir este procedimientoi hasta cansarse

v = A*u
u = v/v(1);
plot(u,'r');

v = A*u;
u = v/v(1);
plot(u,'r');

for i = 1:10
        v = A*u;
        u = v/v(1);
        plot(u,'r');
        hold on
end
```

```
%llega un momento en el cual aunque siga se queda igual
% hemos obtenido UN VECTOR PROPIO de valor propio = 1
% ya que la imagen de si mismo da si mismo!

%con muy poco mas tendremos le metodo iterativo: el poco mas es automatizar los pasos y dar un criterio de convergencia

%el criterio de convergencia:  |xk+1 - xk| < tol    o     k >= maxiter

% si xk+1 y xk fueran vectores, loq ue se haria es calcular la 'norma' de la diferencia
%como las cosas pueden ir mal y nunca se llegara a esa tolerancia se utiliza un maximo de iteraciones

% mas o menos se tiene como hacer el algoritmo en las transparencias

%procedemos con el fichero

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

function [lambda, u, k]=potencias(A,u,tol,maxiter)

% POTENCIAS para calcular valor propio de A

% Inicializar el grafico
plot(u)
pause
hold on

% Inicializar
k = 1;
lambda(1) = 1; %para que no ocurra nada al comprobar lambda(k-1) la primera vez
incr = tol+1;

% Iteraciones

while incr>tol & k<maxiter

    %contar las iteraciones
    k = k+1;

    v = A*u;

    %normalizamos el vector (dividir por la mayor componente)
    [mx,i]=max(abs(v)); %devuelve cual es el maximo y en que posicion se encuentra
    u = v/v(i);

    %medimos el incremento
    lambda(k) = v(i); %guardamos en un vector los coeficientes de normalizacion
```

```
    incr = abs(lambda(k)-lambda(k-1));

    plot(u)
    pause

end

hold off
close

% Aviso no convergencia
if incr>tol
       disp('No ha convergido')
end
```

```
 [lambda, u, k]=potencias(A,u,tol,maxiter)

u =

    1.0000
    0.6667
    0.8333


k =

    16


plot(lambda, '*')
```

PROBLEMAS QUE PUEDEN SURGIR

```
>> [lambda, u, k]=potencias([1 2 ; 0 1],[1 1]',1e-5,40)
No ha convergido

% el vector propio es el 1 0, la convergencia es MUUUUUY lenta.


>> [lambda, u, k]=potencias([1 -1 ; 1 1],[1 1]',1e-5,20)
No ha convergido
```

%se puede ver porque si se quita el hold on
%lo que ocurre es que los vectores van ciclando y no converge nunca

---
MEJORAS DEL ALGORITMO

Rapidez de convergencia
Tipos de convergencia

Error del paso k (Diferencia entre sucesivas aproximaciones)
ek = | xk+1 - xk|

Convergencia lineal cuando
ek+1 / ek --> cte < 1 (si tiende a cero, converge muy rapidamente)
(el peor de los casos seria que la cte tiende a 1, en cuyo caso
el error se reduce muy lentamente)
Convergencia cuadratica cuando
ek+1/ek^2 --> cte < 1
(es mejor que una convergencia lineal, mas rapida
el error entre pasos es cada vez menor)

incluyamos la velocidad de convergencia

sabiendo que diff(lambda) da u n vector con las diferencias entre elementos sucesivos

%Analisis de la convergencia
e = diff(lambda); %vector con lambda 2-lambda1, lambda3 - lambda2, etc...
r = e(2:end)./e(1:end-1); %Tasa de convergencia

La funcion queda como:

---
function [lambda, u, k,e,r]=potencias(A,u,tol,maxiter)

% POTENCIAS para calcular valor propio de A

% Inicializar el grafico
plot(u)
pause
hold on

% Inicializar
k = 1;
lambda(1) = 1; %para que no ocurra nada al comprobar lambda(k-1) la primera vez
incr = tol+1;

```
% Iteraciones

while incr>tol & k<maxiter

   %contar las iteraciones
   k = k+1;

   v = A*u;

   %normalizamos el vector (dividir por la mayor componente)
   [mx,i]=max(abs(v)); %devuelve cual es el maximo y en que posicion se encuentra
   u = v/v(i);

   %medimos el incremento
   lambda(k) = v(i); %guardamos en un vector los coeficientes de normalizacion
   incr = abs(lambda(k)-lambda(k-1));

   plot(u)
   pause

end

%Analisis de la convergencia
e = diff(lambda); %vector con lambda 2-lambda1, lambda3 - lambda2, etc...
r = e(2:end)./e(1:end-1); %Tasa de convergencia

hold off
close

% Aviso no convergencia
if incr>tol
      disp('No ha convergido')
end
---------------------------------------------------------------------
---------------------------------------------------------------------


>> [lambda, u, k,e, r]=potencias(hilb(10),ones(10,1),1e-5,40)

lambda =

  Columns 1 through 5
    1.0000    2.9290    1.9164    1.7819    1.7577
  Columns 6 through 10
    1.7530    1.7521    1.7520    1.7519    1.7519
```

```
u =
    1.0000
    0.6090
    0.4531
    0.3653
    0.3078
    0.2667
    0.2358
    0.2116
    0.1920
    0.1759


k =
    10


e =
  Columns 1 through 5
    1.9290   -1.0125   -0.1345   -0.0242   -0.0047
  Columns 6 through 9
   -0.0009   -0.0002   -0.0000   -0.0000


r =
  Columns 1 through 5
   -0.5249    0.1329    0.1799    0.1923    0.1950
  Columns 6 through 8
    0.1956    0.1957    0.1957
```

Nuestro metodo lo que obtiene es lamda, cuyo limite es el valor propio dominante
y un vector u, tal que

```
>> hilb(10)*u-lambda(end)*u  %da cero dentro de la tolerancia elegida

ans =

  1.0e-005 *

   -0.1331
   -0.1044
   -0.0869
   -0.0748
   -0.0659
   -0.0589
   -0.0534
   -0.0488
   -0.0450
```

-0.0418

>> eig(hilb(10))

ans =

0.0000
0.0000
0.0000
0.0000
0.0000
0.0001
0.0025
0.0357
0.3429
1.7519

---

nuestro metodo de las potencias lo que ha calculado es el mayor valor propio (llamado dominante) (el de mayor valor absoluto)

si no hay tal valor propio dominante, funciona peor o no funciona

---
METODO DE LAS POTENCIAS INVERSAS

Calcula el menor valor propio (el mas proximo a cero)
Es generalmente mas rapido que el anterior

Permite calcular el valor propio mas proximo a donde se quiera

El algoritmo de potencias converge al valor propio máximo de una matriz.

¿Cómo nos acercamos a OTROS valores propios?

ej: los valores propios d euna matriz son (en la recta real):

```
|----------x-------x---------------x----
0                          ^
```

el ultimo es el que se encuentra mediante el metodo de las potencias.

Si tenemos:

A*v = h*v
v = h * inv(A) * v
inv(A) * v = v/h = (1/h) * v

para la matriz inv(A)
el vector propio es el mismo, con un valor propio (1/h)

haciendo el método de las potencias a inv(A) proporciona el mayor
valor propio de inv(A), es decir, el mayor (1/h), que es igual al
menor valor propio (h) de A.

Basta con aplicar el metodo de las potencias a la matriz inversa.

(AL aplicar el metodo de las potenicas, el elemento que interesa es lamda(end)

```
>> [lambda, u, k,e,r]=potencias2(hilb(3),[1 1 1]',1e-5,10);
>> lambda(end) %el valor propio
ans =
   1.4083

>> u %el vector propio
u =
   1.0000
   0.5560
   0.3909
```

Que comparando con los obtenidos mediante eig.

```
>> [S,D] = eig(hilb(3))

S =

   -0.1277    0.5474    0.8270
    0.7137   -0.5283    0.4599
   -0.6887   -0.6490    0.3233


D =

    0.0027         0         0
         0    0.1223         0
         0         0    1.4083
```

---

corresponde al ultimo de lso valores propios con su vector propio.

Si lo aplicamos a la inversa

```
>> [lambda, u, k,e,r]=potencias2(inv(hilb(3)),[1 1 1]',1e-5,10);

>> lambda(end)
ans =
  372.1151

>> 1/lambda(end)
ans =
    0.0027

>> u
u =
   -0.1789
    1.0000
   -0.9649
```

vemos que el valor es igual.
El vector tb es igual (hay que normalizarlo)

```
>> S/S(2,1)
ans =
   -0.1789    0.7670    1.1587
```

```
    1.0000   -0.7402    0.6443
   -0.9649   -0.9093    0.4530
```

Como se evita el calculo de la inversa?
---
en lugar de pedirle que haga

v = inv(A)*u

podemos pedirle

v = A\u

para ser mas estricto aun se podria utilizar la factorizacion LU
ya que va a resolver el mismo sistema repetidas veces.

---
potinv.m

editamos potencias.m y cambiamos
    v = A\u;
y añadimos
    lambda = 1./lambda

Veamos como funciona

```
>> [lambda, u, k,e,r]=potinv(hilb(3),[1 1 1]',1e-5,10)

>> lambda(end)
ans =
    0.0027

>> u
u =
   -0.1789
    1.0000
   -0.9649
```


Como encontramos los valores propios intermedios?

A*u = h*u

si desplazamos la matriz A con la identidad: dI
da el desplazado del valor propio

(A+dI)*u = (h+d)*u

al desplazar la matriz, el valor mas procimo a cero resulta ser
el valor mas procimo a d

```
|----------x-------x---------------x----
0          ^
```

```
------------x-----|-x---------------x----
              0 ^
```

Vamos a añadir al algoritmo de las potencias un parametro que sea d

y le añadimos el comando:

A = A - d*eye(size(A));

y a lambda le quitamos el desplazamiento que hemos hecho:

que no te lie dçlo de restar y sumar, simplemente como el
desplazamiento siempre conviene que sea negativo, lo ponemos
ya como supuesto.

```
>> [lambda, u, k,e,r]=potinv(hilb(3),[1 1 1]',1e-5,20,0.1);
```

--> Valor propio mas proximo a d:

ans =

0.1223

--> Vector propio asociado:

u =

```
   -0.8435
    0.8140
    1.0000
```

Que es en efecto el valor propio intermedio junto con su vector propio

>> S./S(3,2)

ans =

```
    0.1967   -0.8435   -1.2743
   -1.0998    0.8140   -0.7086
    1.0611    1.0000   -0.4981
```

La situacion es mas complicada que esto, puesto que los valores propios
pueden estar en C (complejos)

Problemas que pueden haber

Cuando los valores propios se amontonan

El peor caso es cuando hay un valor propio doble.
Para hallar los vectores propios independientes.
Puede solo existir un vector propio.

Puedes verlo en la aplicacion colgada en la web;

Hay otra forma de calcular valores propios que numericamente tiene mejores
propiedades.

A = S * D * inv(S)

Descomposicion en valores singulares

A(mxn) = U(mxm) S(mxn) V*(nxn)

U y V unitarias (V* significa traspuesta conjugada)
S 'diagonal' (puede no ser cuadrada)

Las columnas de U son una base en el espacio de llegada
Las sigmas, que forman la diagonal de S, son unos escalares
Las filas (porq esta traspuesta) de V* son elemntos de una base en el espacio de partida

|Rn ----> |Rn

ver .doc

Vamos a discretizar la X.

>> A = fotox;

7 Integración Numérica (I)

La *integral definida* de una función de una variable aparece en muchas aplicaciones geométricas y físicas, como el cálculo de longitudes, áreas y volúmenes, centros de gravedad y momentos de inercia. Para resolver analíticamente estos problemas necesitamos una expresión explícita del integrando y que ésta admita una primitiva sencilla, lo que limita la aplicabilidad a recintos de geometría simple. En la práctica además, sólo disponemos generalmente de un conjunto discreto de valores del integrando, por lo que hay que buscar una alternativa numérica a la evaluación de la integral.

Las fórmulas de cuadratura numérica se reducen finalmente a una suma ponderada de valores de la función en determinados puntos del intervalo de integración. La forma de elegir estos puntos, llamados *nodos*, y los valores asignados a los *pesos* caracterizan las distintas fórmulas utilizadas. Los métodos que tratamos aquí toman nodos equiespaciados, lo que facilita su aplicación a datos experimentales.

Una regla de integración numérica sencilla es la de *trapecios*. Ésta integra exactamente polinomios de primer grado.

Un aspecto clave de la integración numérica es la estimación del error de las fórmulas de cuadratura. Damos fórmulas teóricas que acotan este error, aunque en la práctica lo estimamos comparando valores aproximados de la integral obtenidos con distinto número de intervalos. El *método iterativo de trapecios* evalúa eficientemente la integral al tiempo que proporciona una estimación del error.

Exigiendo que la fórmula integre exactamente polinomios de segundo grado se obtiene la *regla de Simpson*. Aquí obtenemos esta fórmula mediante el análisis del error de la fórmula de los trapecios, según el llamado método de extrapolación de Richardson.

La aplicación reiterada de esta técnica da lugar a la integración de Romberg que, bajo ciertas condiciones, proporciona fórmulas de precisión teóricamente tan alta como se quiera. MATLAB incorpora métodos de cuadratura adaptativa, que toman más o menos nodos según el comportamiento de la función en cada parte del intervalo de trabajo.

7.1 Integral definida: Cálculo

Los *métodos numéricos de integración* se basan en la interpretación de la integral definida como medida del área entre la gráfica de la función, el eje de abscisas y las ordenadas de los extremos del intervalo.

Constituyen la alternativa a la *Regla de Barrow* cuando no se dispone de la primitiva, bien porque es difícil de calcular o bien porque el integrando sólo se conoce en un número finito de puntos, al proceder de datos experimentales.

Hay funciones relativamente sencillas, como *sen(x)/x,* cuya primitiva no tiene una expresión analítica finita. Muchas funciones de gran importancia en las aplicaciones se definen mediante integrales de este tipo, por ejemplo, la *función Gamma de Euler*

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt, \qquad x > 0$$

o la *función de error*,

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\ dt$$

que está relacionada con la distribución de probabilidad normal, o *distribución de Gauss*. Se supone generalmente que los errores producidos al efectuar una medida experimental siguen la distribución normal.

Numéricamente, aproximaremos la integral mediante una suma ponderada de valores de la función en ciertos puntos del intervalo de integración llamados *nodos*. Los coeficientes de la fórmula se llaman *pesos*.

$$\int_a^b f(x)\,dx \cong p_1 f(x_1) + p_2 f(x_2) + \ldots + p_n f(x_n)$$

La suma de los pesos debe ser igual a la longitud del intervalo.

$$p_1 + p_2 + \cdots + p_n = b - a$$

En el caso de integrandos procedentes de datos experimentales, además de no disponer, obviamente, de la primitiva, conocemos los valores de la función sólo en un número finito de nodos. Esto limita los métodos aplicables, pues muchos de ellos requieren nodos en cantidad y posición fijas, lo que es difícil de conseguir experimentalmente.

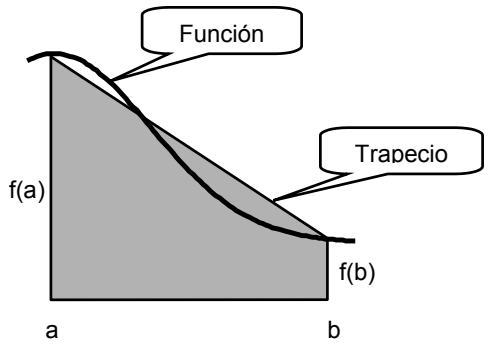
7.2 Formula de los Trapecios

La *fórmula de los trapecios simple* se obtiene al aproximar el integrando *f* en [*a,b*] por la recta que une los puntos *(a, f(a))* y *(b, f(b))*. La integral de esta función de grado 1 es el área del trapecio rectángulo de bases *f(a)* y *f(b)* y altura *b-a*. De aquí el nombre del método. Dicha área es

$$I_T = (b-a)\frac{f(a)+f(b)}{2}$$

El error de la fórmula depende de lo curvada que sea la función entre *a* y *b*. Como la derivada segunda mide la curvatura, es razonable que esta derivada aparezca en la expresión del error. En consecuencia, la fórmula será exacta para funciones con derivada segunda nula, o sea, polinómicas de primer grado.

$$E_T = -\frac{(b-a)^3}{12} f''(\xi), \qquad \text{para cierto } \xi \in [a,b]$$



Fórmula de los Trapecios Simple

Para disminuir el error, dividimos el intervalo en subintervalos y aplicar la fórmula de trapecios en cada subintervalo. Así obtenemos la fórmula de trapecios compuesta.

Si dividimos [*a,b*] en *n* subintervalos iguales de longitud *h=(b-a)/n*, aplicamos la fórmula de trapecios simple a cada subintervalo y sumamos los resultados, obtenemos la *fórmula de trapecios compuesta*. Se trata de una fórmula cerrada (es decir, que los extremos del intervalo son nodos de integración) en la que los nodos interiores tienen peso *h* y los extremos peso *h/2* cada uno.

$$I_T[h] = \frac{h}{2}(y_0 + 2y_1 + 2y_2 + \cdots + 2y_{n-1} + y_n)$$

El error de la fórmula de los trapecios se expresa en función de la derivada segunda de *f* en un punto del intervalo.

$$E_T = -\frac{h^2}{12}(b-a)f''(\xi), \qquad \text{para cierto } \xi \in [a,b]$$

Si la derivada segunda no varía mucho. Se puede suponer que el error es proporcional a $\mathbf{h^2}$, lo que significa que al duplicar el número de subintervalos, esperamos que el error se divida por 4. Esta suposición no es válida si la derivada segunda no está definida en algún punto del intervalo.

En MATLAB construimos fichero.m con el siguiente algoritmo.

Algoritmo de Trapecios

- Entrada: *a, b:* intervalo
  n: número de subintervalos
- Salida: ***I*:** estimación de la integral por la fórmula de los trapecios
- Proceso: Calcular el paso, *h=(b-a)/n*
  Crear el vector de nodos**,** *x=a:h:b*

Calcular el vector de ordenadas, *y=f(x)*
Construir un vector con los pesos**,** p=h/2*[1 2 2 . . . 2 1**]**
Evaluar la formula de los trapecios: *I=p*y'*

Ejemplo 1

Considera la integral $\int_0^{\pi/2} \frac{\operatorname{sen} x}{x} dx$

Evalúa la integral por el Método de los Trapecios con 2, 4 y 8 intervalos. Define la función en x=0 para evitar la indeterminación.

damos los valores del intervalo y calculamos para el primer valor de *n y* obtenemos *h.*

```
n=2,a=0, b=pi/2, h=(b-a)/n
```

```
n =
     2
a =
     0
b =
    1.5708
h =
    0.7854
```

Definimos una función.m senxx y calculamos el valor de la integral

```
i=trapecio('senxx',a,b,2)
```

```
i =
    1.3498
```

Caslculamos la integral para evitar que se indetermine en x=0.

```
i=trapecio('f',a+eps,b,2)
```

```
y =
    1.0000    0.9003    0.6366
i =
    1.3498
```

Lo mismo para n=4, 8.

La acotación del error de la fórmula de los trapecios no es práctica pues requiere el cálculo de la derivada segunda. En su lugar, aplicamos la fórmula con distinto número de subintervalos, n_1 y n_2 y tomamos la diferencia entre los resultados como estimación del error.

Si el n_2 es múltiplo de n_1, se pueden reutilizar los nodos ya calculados. Duplicando cada vez el número de intervalos, introducimos la mínima cantidad posible de nodos nuevos.

Con el mismo trabajo que costaría considerar un nodo más, obtenemos una precisión cuatro veces mayor.

Veamos la relación entre dos evaluaciones de la fórmula de los trapecios, una con doble número de subintervalos que la otra.

Denotamos por $I_T[h]$ la integral de f en $[a,b]$, por la fórmula de los trapecios con paso $h = (b - a)/n$ y nodos

$$a = x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n = b.$$

Introducimos ahora nodos intermedios,

$$a + h/2 = x_{1/2} < x_{3/2} < \ldots < x_{(2n-1)/2} = b - h/2$$

quedando $2n$ subintervalos de longitud mitad, $h/2$.

Evaluamos la fórmula de los trapecios con paso $h/2$ y, agrupando los términos correspondientes a la integración de paso h, obtenemos la nueva integral en términos de la anterior y de los valores de f en los nuevos nodos. De este modo sólo realizamos las operaciones estrictamente necesarias y obtenemos de propina la estimación del error.

$$\begin{aligned} I_T[h/2] &= h/4\,(y_0 + 2y_{1/2} + 2y_1 + 2y_{3/2} + 2y_2 + \ldots + 2y_{(2n-1)/2} + y_n) = \\ &= h/4\,(y_0 + 2y_1 + 2y_2 + \ldots + 2y_{n-1} + y_n) + \\ &+ h/2\,(y_{1/2} + y_{3/2} + y_{5/2} + \ldots + y_{(2n-1)/2}) = \\ &= I_T[h]/2 + h/2(y_1 + y_3 + \ldots + y_{n-1}) \end{aligned}$$

Mediante esta relación, construiremos un *algoritmo de trapecios iterativo* que va duplicando el número de subintervalos hasta que la variación entre dos evaluaciones consecutivas sea menor que la tolerancia exigida.

$$E[h/2] \cong I_T[h/2] - I_T[h]$$

Algoritmo de iterativo de Trapecios

- Entrada: *a, b*: intervalo
  n: número de subintervalos
  tol: precisión
  maxiter: tope de iteraciones
- Salida: *I*: estimación de la integral por la fórmula de los trapecios
  incr: estimación del error
  iter: número de iteraciones efectuadas
- Proceso: Calcular la estimación inicial de $I_T[h]$ por trapecios con paso h
  Inicializar *incr* y el contador de iteraciones, *iter*
  Mientras *incr* >*tol* y *iter* < *maxiter*
  Hallar los nosdos nuevos, $x=[x_{1/2}, x_{3/2}, \ldots, x_{n-1/2}]$
  Calcular el vector de ordenadas $y=f(x)$
  Refinar la estimación $I_T[h/2]=I_T[h] + h/2*sum(y)$
  Calcular *incr,* incrementar *iter*
  Dividir *h* por *2*
  Actualizar $I_T[h]$.

Ejemplo 2

Definimos una función .m llamada *trapiter* y resolvemos el siguiente aopartado del ejemplo anterior.

¿Cuántos intervalos serían necesarios para obtener una precisión del orden de 10^{-4}?

Para obtener una precisión de ese orden *tol* vale 10^-3 porque si ponemos nos sale una precisión de 10^-5, 10^-6, ...

Aplicamos la función .m trapiter

```
[sol,iter,incr]=trapiter('senxx',0,pi/2,4,1e-4,100)
```

```
sol =
    1.3707
iter =
     5
incr =
  6.1037e-005
```

Sol: n=4*2^(iter-1)=64

7.3 Regla de Simpson

La fórmula de los trapecios integra exactamente los polinomios de primer grado utilizando dos nodos, x_0 y x_1 en el intervalo $[a,b]$. Añadiendo un nodo más, esperamos que para ciertos pesos a determinar, la fórmula

$$I = p_0 f(x_1) + p_1 f(x_2) + p_2 f(x_3)$$

sea exacta para polinomios de segundo grado. Los pesos se determinan sustituyendo *f* por un polinomio de grado 2 y resolviendo el sistema lineal resultante.

En el caso particular de $x_1 = a;\ x_2 = (a+b)/2;\ x_3 = b,$ queda la *regla de Simpson simple*.

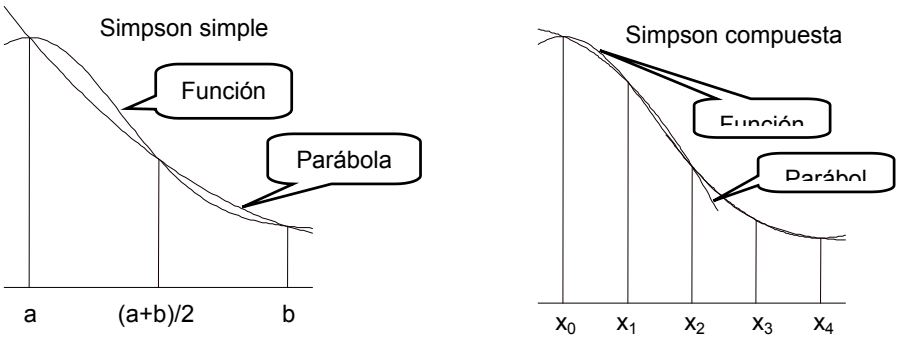
$$\int_a^b f(x)dx \cong I_S = \frac{b-a}{6}\left(f(a) + 4f\left(\tfrac{a+b}{2}\right) + f(b)\right)$$

La característica más destacable de esta fórmula es que resulta exacta para polinomios de tercer grado, mientras que en su construcción sólo se exigía la exactitud para grado 2.

El error cometido al aproximar la integral de una función cualquiera con derivada cuarta continua viene dado por la fórmula

$$E_S = I - I_S = -\frac{(b-a)^5}{2880} f^{IV}(\xi), \qquad \xi \in [a,b]$$

Para reducir el error, subdividimos el intervalo [*a,b*] en partes iguales antes de aplicar la regla de Simpson. Así obtenemos la regla de Simpson compuesta.



Algoritmo de la regla de Simpson Compuesta

• Entrada: *a, b***:** intervalo
*n***:** número de subintervalos (ha de ser par)
• Salida: *I:* estimación de la integral por la regla de Simpson
• Proceso: Calcular el paso, *h=(b-a)/n*
Crear el vector de nodos**,** *x=a:h:b*
Calcular el vector de ordenadas, *y=f(x***)**
Construir un vector con los pesos**,** *p=h/3*[1 4 2 4 2 . . . 2 4 1]*
Evaluar la formula de los trapecios: *I=p*y'*

La fórmula de Simpson es de orden 4, lo que significa que al duplicar el número de intervalos, el error se divide por 2^4. En la práctica esto supone que cada vez que el paso se divide por 2, el error se divide por $2^4 = 16$, con lo que se obtiene al menos una cifra decimal exacta más en el resultado.

Vamos a obtener la regla de Simpson por un procedimiento alternativo que proporciona directamente estimaciones del error. Además, este procedimiento se generaliza fácilmente, dando lugar a fórmulas de integración de precisión tan alta como queramos, al menos en teoría.

El error de la fórmula de los trapecios es aproximadamente proporcional a h^2, si la derivada segunda del integrando no varía mucho en el intervalo de integración.

Evaluando esta fórmula para dos valores de *h* (como hemos visto, lo más fácil es tomar *h* y *h/2,* y aplicar trapecios iterativo), eliminamos el término de orden 2 del error y obtenemos una estimación más precisa de la integral. Este truco se denomina *extrapolación de Richardson*.

Desarrollando las integrales que aparecen en la fórmula de Richardson, se comprueba que esta estimación es precisamente la Regla de Simpson con paso *h/2,* cuyo error es de orden h^4.

Algoritmo de Simpson-Richardson

- Entrada: *a, b*: intervalo
  n: número de subintervalos dobles
- Salida: *I*: estimación de la integral por la regla de Simpson
- Proceso: Calcular el paso, $h=(b-a)/n$
  Calcular $I_T[h]$ por trapecios
  Calcular $I_T[h/2]$ como en trapecios iterativos
  Calcular I_S por la formula $I_S=(4*I_T[h/2]-I_T[h])/3$

La idea de Richardson que proporciona la integral de Simpson (de orden 4) a partir de dos estimaciones de trapecios (de orden 2) se puede aplicar también a dos estimaciones de Simpson para obtener una estimación de mayor orden (6).

$$I \cong \frac{4^2 I_S[h/2] - I_S[h]}{4^2 - 1} \underset{def}{=} I_R[h/2]$$

Análogamente, dos estimaciones de orden 6 proporcionan una estimación de orden 8, y así sucesivamente.

Ejemplo 3

Definimos una función .m *simpson* la cual aplicamos para calcular la integral del **Ejemplo 1** por la Regla de Simpson con 2, 4 y 8 intervalos.

```
i=simpson('senxx',0,pi/2,2)

n =
     2
i =
    1.3713
```

7.4 Integración de Romberg

La aplicación sistemática de la extrapolación de Richardson se conoce con el nombre de *Integración de Romberg*. La fórmula de los trapecios para cierto *h* proporciona la primera estimación de la integral, $I_T[h]$. Obtenemos una segunda estimación con paso *h/2*, por trapecios iterativos, y de ambas, una estimación de Simpson $I_S[h/2]$. El tercer paso comienza con la estimación $I_T[h/4]$, que junto con $I_T[h/2]$, da lugar a otra estimación de Simpson, $I_S[h/4]$. De las dos estimaciones disponibles de Simpson se obtiene la primera de Romberg: $I_R[h/4]$. El proceso se puede continuar hasta alcanzar la precisión requerida.

Diseñamos un procedimiento de MATLAB que devuelva una matriz con las estimaciones calculadas por el método de Romberg. El programa contiene dos bucles while anidados. El while interno recorre una fila y el externo crea una fila nueva.

El primer elemento de cada fila se obtiene por la fórmula de los trapecios, directa en la primer fila y por refinamiento iterativo en las siguientes.

Los restantes elementos de la fila, hasta la diagonal, se obtienen por la fórmula de Romberg adecuada.

- Entrada: *a, b:* intervalo
  n: número de subintervalos
  tol: precisión
  maxiter: tope de iteraciones
- Salida: *I*: Tabla de Romberg
  incr: estimación del error
  k: número de iteraciones efectuadas
- Proceso: Calcular I_{11} por trapecios con *n* intervlos
  Inicializar *incr, k*
  Mientras *incr* >*tol* y *k* < *maxiter*
  Evaluar I_{k1} refinado $I_{k-1,1}$ % Primer elemento
  Inicializar *j*
  Mientras *Incr* > *tol y j* < *k* % Resto de la fila k
  Calcular I_{kj} por la fórmula de Romberg
  Incrementar *j*
  Incrementar *k*

Si *incr* > *tol* mensaje de error: no logragada convergencia

Aplicamos ahora el Método de Romberg

Ejemplo 4

Definimos una función .m romberg. Evalúamos la integral del **Ejemplo 1** por el método de Romberg, partiendo de trapecios simple con un máximo de 8 subintervalos.

```
 [sol,i,incr,k]=romberg('senxx',0,pi/2,8,1e-4,100)

sol =
    1.3708
i =
    1.3695         0         0
    1.3704    1.3708         0
    1.3707    1.3708    1.3708
incr =
```

```
  7.4353e-009
k =
     2
```

7.5 Problemas

1. La longitud de una curva de ecuación y=f(x), para x variando en [*a,b*], viene dada por la integral

$$\int_a^b \sqrt{1+y'^2}\,dx\,.$$

Halla la longitud de una elipse de semiejes 2 y 1.

2. La integral elíptica completa de primera especie es

$$K(\theta)=\int_0^{\pi/2}\frac{dx}{\sqrt{1-\sin^2\theta\,\sin^2 x}}$$

Calcula K(π/6) utilizando la regla de Simpson con cuatro intervalos. El resultado exacto con cuatro decimales es 1'6858. Halla K(17π/36), cuyo valor aproximado es 3'832. ¿Por qué tiene tanto error este resultado, mientras que el primero era bastante preciso?

3. En teoría de campos eléctricos se prueba que el campo magnético inducido por una corriente en una espira circular de alambre es

$$H(x)=\int_0^{\pi/2}\sqrt{I-(\tfrac{x}{r})^2\sin^2\theta}\;\,d\theta$$

donde I es la intensidad de la corriente, r el radio de la circunferencia y x la distancia del centro al punto donde se calcula el campo. Si I=15.3, r=120 y x=84, halla el campo en x por integración de Romberg con tolerancia 10^{-6}.

4. La función de Bessel de orden 0 está definida por la ecuación

$$J_0(x)=\frac{1}{x}\int_0^{\pi}\cos(x\sin\theta)\,d\theta$$

Halla $J_0(1)$ por la fórmula de Simpson 3/8.

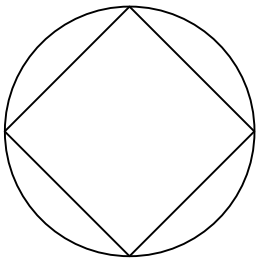
4. (Examen de Septiembre de 1998). El área de un círculo de radio 1 se obtiene integrando su ecuación explícita en [-1,1]:

$$\pi=2\int_{-1}^{1}\sqrt{1-x^2}\,dx$$

a) Aproxima esta integral por el método de Romberg, a partir de valores de trapecios con 2, 4, 8, 16 y 32 intervalos.

Explica por qué la estimación obtenida por trapecios no mejora sustancialmente al aplicar Romberg.

El área del círculo puede aproximarse mediante el área de polígonos inscritos. Utilizaremos para ello el cuadrado (de lado $\sqrt{2}$) y los polígonos regulares de 8, 16, 32 y 64 lados. Es fácil comprobar que el área del polígono regular de 2n lados es $A_{2n} = nx_n/2$, siendo x_n el lado del polígono regular de n lados (la mitad).



También es fácil obtener una relación algebraica entre los lados de estos polígonos:

$$x_{2n}^2 = 2 - \sqrt{4 - x_n^2}$$

b) Halla las áreas de los polígonos mencionados. Escribe los comandos o archivos.m elaborados para ello.

c) Comprueba numéricamente que la diferencia entre el área del círculo y la del polígono es proporcional al cuadrado del lado. ¿Cuál es el coeficiente de proporcionalidad?

d) Suponiendo que al tomar el doble de lados, la longitud del lado se divide aproximadamente por 2, aplica el algoritmo de Romberg a las áreas halladas para obtener una mejor estimación de π. Compara los resultados con los del apartado a). (Indicación: modifica el algoritmo de Romberg para que en lugar de calcular la primera columna por trapecios, la admita como argumento de entrada)

7.6 Soluciones

Métodos para calcular la primitiva

FORMULA DE LOS TRAPECIOS

Aproximar el area bajo una curva mediante una figura mas sencilla pero similar. El area del trapecio (que se puede convertir en un cuadrado cortando un trozo y poniendola arriba) viene dada como

$$(b-a)\frac{f(a)+f(b)}{2}$$

El error en esa aproximacion viene dada por:

$$E = -\frac{(b-a)^3}{12} f''(e) \text{ para cierto } e \in [a,b]$$

Nota: f"(e) = 0 para polinomios de primer grado, en los cuales el error es cero
Nota: f"(e) indica si la funcion es concava o convexa, indicando si el error es positivo o negativo (sobreestimado o subestimado)
Nota: Si la funcion no es ni fu ni fa, algunos trozos por encima, otros por debajo para saber si el error es negativo o positivo depende del tamaño de las zonas por encima o por debajo.
Nota: (b-a)^3 puede ser un numero muy grande. Conviene dividir el trapecio en trapecios mas pequeños (ver apuntes a mano)

```
>> [I,incr,k] = trapiter('cos',0,pi/2,5,1e-8,20)

I =

  Columns 1 through 2

   0.99176176875473   0.99794298635436
        ....


incr =

    5.882744735785650e-009


k =

    12

>> I'

ans =

   0.99176176875473
   0.99794298635436
   0.99948590524853
   0.99987148622292
   0.99996787217507
   0.99999196808247
   0.99999799202304
   0.99999949800591
   0.99999987450149
   0.99999996862537
   0.99999999215634
   0.99999999803909

>> (4*I(2)-I(1))/3

ans =

   1.00000339222090
```

8. Integración numérica (III)

8.1. Métodos de Newton-Cotes

Los métodos numéricos de integración vistos hasta ahora se basan en dividir el intervalo de integración en subintervalos iguales a los que se aplica una fórmula simple que satisface ciertas condiciones y sumar los resultados de cada subintervalo. En el caso de la fórmula de los trapecios, la condición es integrar exactamente funciones de primer grado y en la de Simpson funciones de segundo grado, aunque en este caso resulta exacta para polinomios de tercer grado incluso.

El aumento de precisión de estos métodos se consigue aumentando el número de subintervalos. La alternativa que examinamos en esta sección consiste en lograr fórmulas simples más precisas, a base de exigir que integren exactamente polinomios de mayor grado. Se obtienen así las llamadas fórmulas de Newton-Cotes, que engloban las anteriores como caso particular.

En la sección siguiente abordamos una estrategia complementaria más sofisticada, consistente en aplicar estas fórmulas a subintervalos de tamaño variable que se determina en función del error estimado en cada subintervalo. Es lo que se llama integración adaptativa.

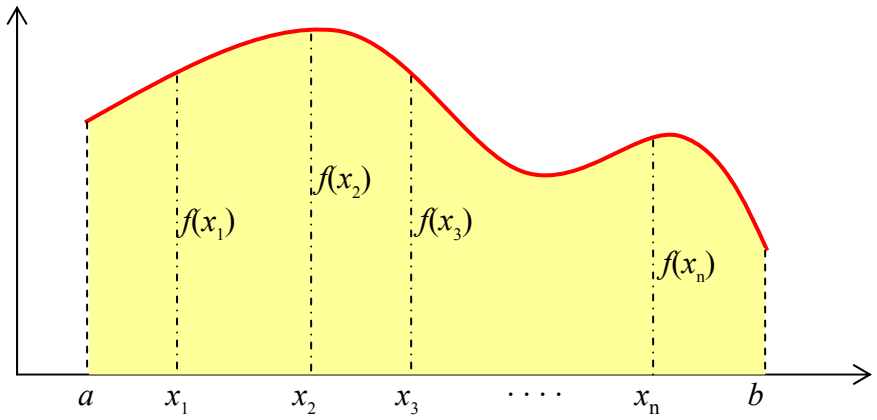


Figura 1: Nodos y valores de la función en el intervalo de integración.

La idea de los métodos de integración de tipo interpolatorio consiste en determinar los pesos $p_1, p_2, \ldots, p_n$, para que la fórmula

$$I_{n-1}(f) = p_1 f(x_1) + p_2 f(x_2) + \ldots + p_n f(x_n) \qquad (8.1)$$

integre exactamente polinomios de grado inferior a n en el intervalo [a, b], siendo $x_1, x_2, \ldots, x_n$, puntos dados de dicho intervalo llamados *nodos*.

Los pesos se determinan resolviendo el sistema lineal que se obtiene al imponer las condiciones requeridas. Para $k = 0, 1, 2, \ldots, n-1$, al exigir que la fórmula integre exactamente x^k en $[a, b]$ queda

$$I_{n-1}(x^k) = \int_a^b x^k dx$$

$$p_1 x_1^k + p_2 x_2^k + \ldots + p_n x_n^k = \frac{b^{k+1} - a^{k+1}}{k+1} \qquad (8.2)$$

La matriz del sistema resultante es de tipo Vandermonde, lo que, en teoría, garantiza la existencia de solución y, en la práctica, la inestabilidad del resultado.

La calidad de la fórmula resultante depende en gran manera de los nodos. Lo más habitual es considerar nodos uniformemente repartidos en el intervalo, dado que son aplicables a la integración de datos experimentales. Este es el caso de las fórmulas de Newton-Cotes. Las fórmulas de integración de Gauss, que no tratamos aquí, eligen los nodos de manera que se consiga la mayor precisión posible. Los nodos resultan no equiespaciados, por lo que el integrando debe poder evaluarse en ellos, lo cual es un inconveniente grave en las aplicaciones prácticas. Casi siempre se eligen los nodos dispuestos simétricamente en el intervalo, aunque a continuación vemos un ejemplo elemental dónde no es así.

Ejemplo 1

Si consideramos un solo nodo situado en el extremo izquierdo del intervalo y exigimos que la fórmula 8.1 integre exactamente polinomios de grado 0, o sea, constantes, se ha de verificar:

$$\int_a^b 1\,dx = b - a = p_1 1$$

de donde se obtiene la fórmula

$$\int_a^b f(x)\,dx \cong (b-a)\,f(a)$$

que se denomina fórmula del extremo izquierdo y que, por construcción resulta exacta para polinomios de grado 0.

Para obtener una fórmula más precisa, hemos de tomar más nodos.

Ejemplo 2

Supongamos que queremos integrar la función seno entre 0 y $\pi/2$ a partir de los valores de la tabla adjunta, utilizando una fórmula interpolatoria de máximo grado.

x	0	$\pi/6$	$\pi/4$	$\pi/3$	$\pi/2$
$\sin x$	0	$1/2$	$\sqrt{2}/2$	$\sqrt{3}/2$	1

En primer lugar creamos un vector para los nodos y otro para los valores del integrando.

```
x = [0 pi/6 pi/4 pi/3 pi/2]
y = [0 1/2 sqrt(2)/2 sqrt(3)/2 1]
```

Como hay 5 nodos, la fórmula debe integrar polinomios de grado 4. La matriz del sistema tiene en la fila k los valores de x^{k-1} en los nodos.

```
A = [x.^0; x; x.^2; x.^3; x.^4]
```

```
A =
    1.0000    1.0000    1.0000    1.0000    1.0000
         0    0.5236    0.7854    1.0472    1.5708
         0    0.2742    0.6169    1.0966    2.4674
         0    0.1435    0.4845    1.1484    3.8758
         0    0.0752    0.3805    1.2026    6.0881
```

El término independiente de la ecuación k-ésima es

$$\int_0^{\pi/2} x^{k-1}dx = \frac{(\pi/2)^k}{k}$$

o sea

```
q =pi/2; c = [q; q.^2/2; q.^3/3; q.^4/4; q.^5/5]
```

```
b =
    1.5708
    1.2337
    1.2919
    1.5220
    1.9126
```

Resolvemos el sistema para hallar los pesos

```
pesos = A\c
```

```
pesos =
    0.1440
    1.0603
   -0.8378
    1.0603
    0.1440
```

Multiplicando por los valores y sumando se obtiene la integral aproximada:

```
y*pesos
```

```
ans = 1.0000
```

Nota: La obtención de la matriz del sistema puede abreviarse con MATLAB usando `A = rot90(vander(x))`.

A menudo, y sobre todo cuando se trabaja con nodos equiespaciados, es conveniente normalizar el intervalo y utilizar los pesos calculados para el intervalo normalizado. Éste intervalo suele ser [0, 1] o [-1, 1]. Para aplicar una fórmula válida en un intervalo normalizado a una integral sobre un intervalo cualquiera [a, b], basta tomar los nodos adecuados en éste intervalo y multiplicar los pesos por el cociente entre la longitud del intervalo de integración y el normalizado.

Ejemplo 3

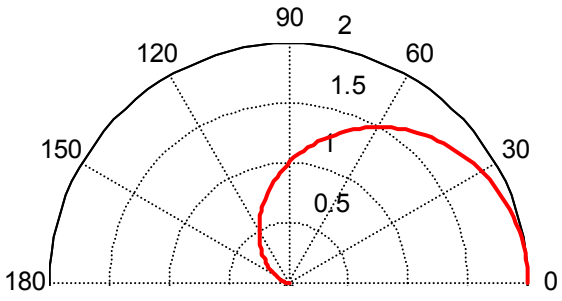
Vamos a calcular el área de la cardioide de ecuación polar $\rho = 1 + \cos\theta$ en el intervalo $[0, \pi]$. No es difícil obtener el resultado analíticamente

$$A = \frac{1}{2}\int_0^{\pi} \rho^2 d\theta = \frac{1}{2}\int_0^{\pi} (1+\cos\theta)^2 d\theta =$$

$$\frac{1}{2}\int_0^{\pi} \left(1 + 2\cos\theta + \cos^2\theta\right) d\theta =$$

$$\frac{1}{2}\int_0^{\pi} \left(1 + 2\cos\theta + \frac{1+\cos 2\theta}{2}\right) d\theta =$$

$$\frac{1}{2}\int_0^{\pi} \left(1 + \frac{1}{2}\right) d\theta = \frac{3\pi}{4} \cong 2.3562$$



Se trata de aproximar esta integral por la fórmula del Ejemplo 2. En este caso, basta multiplicar los pesos por 2 y escalar los nodos en la misma proporción.

```
nodos = 2*x;
y = (1+cos(nodos)).^2/2;
pesos2 = 2*pesos;
y*pesos2

ans = 2.3889
```

El error cometido es

```
3*pi/4-ans

ans = -0.0327
```

8.1.1.1. Transformación de intervalo

Si la adaptación no es obvia, se efectúa una transformación lineal que aplique un intervalo en otro. Supongamos que tenemos la fórmula normalizada

$$\int_{\alpha}^{\beta} f(x)\,dx \cong p_1 f(x_1) + p_2 f(x_2) + \ldots + p_n f(x_n)$$

y hemos de evaluar la integral

$$\int_a^b g(t)\,dt\,.$$

Buscamos un cambio de variable de la forma $t = mx + n$ que transforme el intervalo $[a, b]$ en $[\alpha, \beta]$. Ello se consigue para los valores de m y n siguientes

$$m = \frac{b-a}{\beta-\alpha}, \quad n = \frac{\beta a - \alpha b}{\beta - \alpha}$$

La fórmula de integración adaptada al nuevo intervalo es

$$\int_a^b g(t)\,dt \cong q_1 g(t_1) + q_2 g(t_2) + \ldots + q_n g(t_n)\,,$$

donde

$$q_k = m p_k \text{ y } t_k = m x_k + n\,, \text{ para } k = 1, 2, \ldots, n\,.$$

Los nodos de las fórmulas de Newton-Cotes son puntos equiespaciados en el intervalo de integración que se obtienen dividiéndolo en partes iguales y tomando los extremos de los subintervalos resultantes. Las fórmulas de Newton-Cotes se denominan fórmulas cerradas o fórmulas abiertas, según consideren o no como nodos los extremos del intervalo inicial.

8.1.1 Fórmulas de Newton-Cotes cerradas

La fórmula de Newton-Cotes cerrada de orden n se obtiene dividiendo el intervalo en n subintervalos iguales, tomando sus n+1 extremos como nodos de integración y determinando los pesos de la fórmula 8.1 para que integre exactamente polinomios de grado n.

Las fórmulas basadas en un número impar de nodos (n par) son exactas para polinomios de grado n+1, un grado más de lo esperado, por lo que suelen preferirse a las de n impar.

Ejemplo 4

La fórmula de los trapecios simple es la fórmula de Newton-Cotes cerrada de orden 1. Se obtiene al tomar como nodos los extremos del intervalo. El sistema 8.2 queda :

$$p_1 + p_2 = b - a$$

$$p_1 a + p_2 b = \frac{b^2 - a^2}{2}$$

y los pesos son

$$p_1 = p_2 = \frac{b-a}{2}$$

dando la fórmula de Trapecios simple:

$$I_T(f) = \frac{b-a}{2}\big(f(a) + f(b)\big).$$

Ejemplo 5

Vamos a obtener la fórmula de Newton-Cotes cerrada de orden 2. Para simplificar los cálculos elegimos [−1, 1] como intervalo de integración. Imponemos que la fórmula 8.1 integre exactamente los polinomios 1, x, y x^2. El sistema 8.2 queda entonces

$$\begin{aligned} p_1 \;+\; p_2 \;+\; p_3 &= \;\; 2 \\ -\,p_1 \;\;\;\;\;\; +\; p_3 &= \;\; 0 \\ p_1 \;\;\;\;\;\; +\; p_3 &= \frac{2}{3} \end{aligned}$$

de donde

$$p_1 = p_3 = \frac{1}{3}; \quad p_2 = \frac{4}{3}.$$

Para que los pesos sean aplicables a un intervalo cualquiera [a, b], realizamos el cambio de intervalo (3) con $\alpha = -1$ y $\beta = 1$:

$$m = \frac{b-a}{\beta - \alpha} = \frac{b-a}{2},$$

$$n = \frac{\beta a - \alpha b}{\beta - \alpha} = \frac{a+b}{2}.$$

Los pesos quedan

$$q_1 = q_3 = m\frac{1}{3} = \frac{b-a}{6}; \quad q_2 = m\frac{4}{3} = 4\frac{b-a}{6},$$

y los nodos

$$t_1 = a, \quad t_2 = \frac{a+b}{2}, \quad t_3 = b.$$

La fórmula obtenida es la regla de Simpson simple:

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

Ejemplo 6

En el Ejemplo 3 calculamos el área de media cardioide mediante una fórmula basada en 5 nodos no equiespaciados. Tomando los nodos equiespaciados, el cálculo corresponde a la fórmula de Newton-Cotes de orden 4.

```
x = 0:pi/4:pi
y = (1+cos(x)).^2/2
A = rot90(vander(x))
q = pi; c = [q; q.^2/2; q.^3/3; q.^4/4; q.^5/5]
pesos = A\c
```

```
y*pesos
```

El resultado obtenido, 2.3736, es más aproximado que el del citado ejemplo, pues el error es de –0.0175.

La determinación numérica de los pesos se ve afectada por el mal condicionamiento del sistema lineal resultante. Para *n* elevado, los errores de redondeo limitan la precisión de los pesos que se obtienen, por lo que hay que recurrir a métodos analíticos para su determinación.
Otro inconveniente de estas fórmulas al aumentar *n* es que aparecen pesos de gran valor absoluto y distinto signo, lo que se traduce al aplicar la fórmula en substracción de cantidades grandes, que pueden dar lugar a "cancelaciones catastróficas".

8.1.2 Fórmulas de Newton-Cotes abiertas

La fórmula de Newton-Cotes abierta de orden *n* se obtiene dividiendo el intervalo en *n*+2 partes iguales, tomando como nodos los *n*+1 puntos de división y determinando los pesos de la fórmula 8.1 para que integre exactamente polinomios de grado *n*.
Como en el caso de las fórmulas cerradas, las fórmulas con un número impar de nodos (*n* par) son exactas para polinomios de grado *n*+1, cuando lo exigido era de grado *n*.
El hecho de que en las fórmulas abiertas no intervengan los valores del integrando en los extremos del intervalo permite aplicarlas a integrales impropias en que presentan valores infinitos en los extremos. En este caso, conviene analizar previamente la convergencia de la integral y estimar de algún modo el error.
En contrapartida, las fórmulas abiertas presentan mayor error que las cerradas del mismo orden, por lo que éstas son más utilizadas en general.

Ejemplo 7

La fórmula de Newton-Cotes abierta correspondiente a *n*=0 es la llamada fórmula de punto medio, porque éste es el único nodo de integración.

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

Esta fórmula supera las expectativas de precisión al integrar exactamente polinomios de grado 1 y no sólo los de grado 0 para los que está diseñada.

Ejemplo 8

La fórmula de Newton-Cotes abierta de orden 2 se obtiene dividiendo el intervalo en 4 partes y considerando como nodos los extremos interiores de los subintervalos. Por ejemplo, tomando [–1, 1] como intervalo de integración, los nodos son –1/2, 0 y 1/2. Exigiendo que la fórmula 8.1 integre exactamente los polinomios 1, x, y x^2, se obtiene el sistema:

$$\begin{aligned} p_1 + p_2 + p_3 &= 2 \\ -\frac{1}{2}p_1 + \frac{1}{2}p_3 &= 0 \\ \frac{1}{4}p_1 + \frac{1}{4}p_3 &= \frac{2}{3} \end{aligned}$$

de donde

$$p_1 = p_3 = \frac{4}{3}; \quad p_2 = -\frac{2}{3}.$$

Para que los pesos sean aplicables a un intervalo cualquiera [*a*, *b*], realizamos el cambio de intervalo (3) con $\alpha = -1$ y $\beta = 1$:

$$m = \frac{b-a}{\beta-\alpha} = \frac{b-a}{2},$$

$$n = \frac{\beta a - \alpha b}{\beta-\alpha} = \frac{a+b}{2}.$$

Los pesos quedan

$$q_1 = q_3 = m\frac{4}{3} = 2\frac{b-a}{3}; \quad q_2 = m\left(-\frac{2}{3}\right) = -\frac{b-a}{3},$$

y los nodos

$$t_1 = \frac{3a+b}{4}, \quad t_2 = \frac{a+b}{2}, \quad t_3 = \frac{a+3b}{4}.$$

La fórmula obtenida es:

$$\int_a^b f(x)\,dx \cong \frac{b-a}{3}\left(2f\left(\frac{3a+b}{4}\right) - f\left(\frac{a+b}{2}\right) + 2f\left(\frac{a+3b}{4}\right)\right).$$

Ejemplo 9

Comparemos la precisión de las fórmulas abiertas y cerradas estimando el área de media cardioide por medio de la fórmula de Newton-Cotes abierta de orden 4.

```
h = pi/6
x = 0+h:h:pi-h
y = (1+cos(x)).^2/2
A = rot90(vander(x))
q = pi; c = [q; q.^2/2; q.^3/3; q.^4/4; q.^5/5]
pesos = A\c
y*pesos
```

El resultado obtenido es 2.3169 y el error de 0.0393, que es mayor que el de la fórmula cerrada.

8.1.3 Fórmulas de Newton-Cotes compuestas

El error de las fórmulas de Newton-Cotes depende de una potencia del tamaño del intervalo. Conviene pues que el intervalo de integración sea pequeño. Si no lo es, la solución obvia es dividir el intervalo de integración en pequeños subintervalos iguales y aplicar a cada subintervalo la correspondiente fórmula. De este modo se obtienen las fórmulas de Newton-Cotes compuestas.

Son conocidas las fórmulas los Trapecios y la de Simpson, pero el procedimiento es completamente general, dando lugar a fórmulas compuestas de orden cualquiera. Por regularidad y precisión, son más usuales las de tipo cerrado, siendo la de punto medio la única abierta que suele usarse en versión compuesta.

Ejemplo 10

La fórmula de punto medio compuesta en [*a*, *b*] se obtiene dividiendo el intervalo en *m* subintervalos y tomando como nodos los puntos medios de los mismos, $x_1, x_2, \ldots, x_m$.

Se evalúa la integral por punto medio en cada subintervalo, multiplicando su longitud por el valor de la función en el nodo correspondiente y se suman los resultados. La fórmula queda

$$I_M[h] = h\big(f(x_1) + f(x_2) + f(x_m)\big),$$

donde h es la longitud de los subintervalos:

$$h = \frac{b-a}{m}.$$

Ejemplo 11

Para estimar la integral

$$\int_0^{2\pi} \frac{\sin x}{x}\, dx$$

utilizando un método cerrado, hay que definir explícitamente el valor del integrando en 0, que es 1, porque si evaluamos directamente la función, da una indeterminación. Sin embargo, usando un método abierto, evitamos calcular la función en 0 y se puede aplicar directamente la fórmula de integración. Por ejemplo, estimemos esta integral por el método de punto medio con 25 subintervalos.

```
a = 0; b = 2*pi; m = 25;
h = (b-a)/m;
x = a+h/2 : h : b-h/2; % Nodos de integración
y = sin(x)./x;
I = h*sum(y)

I = 1.4177
```

Aplicando un método compuesto con distintos valores de h podemos estimar el error y su orden.

Ejemplo 12

Evaluamos la integral anterior por la fórmula de punto medio con 25 y 50 intervalos, obteniendo $I_1 = 1.41773$ e $I_2 = 1.41805$, respectivamente. La diferencia entre ambos, $3.1 \cdot 10^{-4}$ es una estimación aproximada del error.
El resultado de la fórmula con 100 intervalos es $I_3 = 1.41813$. El valor del cociente

$$\frac{I_3 - I_2}{I_2 - I_1} \cong 0.25$$

indica que el error estimado se divide por 4, al dividir el paso h por 2. Concluimos que el error es proporcional a h^2.
Podemos obtener una estimación mejor de la integral siguiendo la idea de Romberg:

$$I_R = \frac{4I_3 - I_2}{3} = 1.41815.$$

8.2. Integración adaptativa

Los métodos de integración vistos hasta ahora son independientes del comportamiento del integrando en el intervalo. La estrategia para reducir el error consiste en aumentar el número de subintervalos o el orden del método utilizado. Estos métodos pueden tener dificultades si el integrando varía bruscamente en unas zonas del intervalo y suavemente

en otras. En el mejor de los casos, se toma un paso muy pequeño para aproximar bien las variaciones bruscas, lo que trae consigo un elevado número de operaciones que no se necesitan en la zona de variación suave.
Para tratar estos casos, se definen los métodos adaptativos, cuya idea es trabajar más allí donde sea necesario y ahorrar operaciones donde el integrando se comporte mejor. Para ello, comienzan aplicando un método determinado, trapecios por ejemplo, primero con uno y luego con dos subintervalos. Si la diferencia entre las estimaciones es mayor que la tolerancia establecida, se repite el procedimiento con cada uno de los dos subintervalos considerados. El proceso de subdivisión prosigue hasta alcanzar la precisión requerida o un nivel máximo de recursión que se establece para que el algoritmo no cicle indefinidamente.
La adaptabilidad del método consiste en que en cada subintervalo, el número de subdivisiones realizadas puede ser distinto, dependiendo del comportamiento del integrando en el mismo.
Las funciones `quad` y `quad8` de MATLAB son métodos adaptativos basados en el método de Simpson y en método de Newton-Cotes cerrado con 8 subintervalos, respectivamente. Introduciendo un parámetro adicional en la llamada a estos métodos, se representan los valores usados en la estimación de la integral. Se observa la acumulación de los mismos en las zonas más críticas de la función.

Ejemplo 13

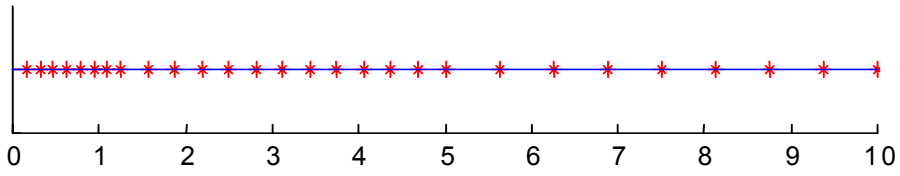
Para integrar la función de Runge en el intervalo [0, 10] mediante `quad`, hay que definir previamente un fichero, **runge.m**, con la función a integrar:

```
function y=runge(x)
y=1./(1+x.^2);
```

Utilizamos `quad` con tolerancia 10^{-2}:

```
quad('runge',0,10,1e-2,1)

ans = 1.47115451163704
```

Los argumentos son: el nombre del fichero con el integrando, los extremos del intervalo de integración, la tolerancia y un quinto parámetro que causa la representación gráfica de los valores del integrando utilizados en la evaluación. La figura adjunta muestra las abscisas de dichos valores. Los nodos se acumulan entorno al origen que es donde la función presenta la variación más acusada.

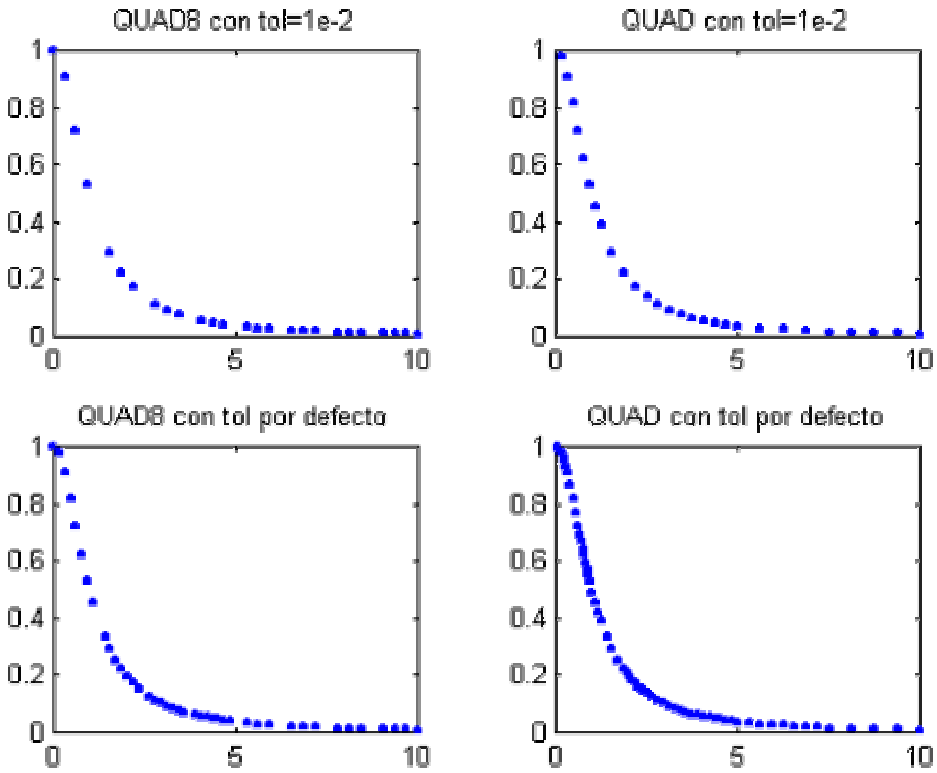


Si ejecutamos estas funciones omitiendo el parámetro de la tolerancia, se utiliza un error relativo de 10^{-3}. Con la misma tolerancia, ambas producen un error del mismo orden (en este caso, `quad8` tiene mayor error), aunque `quad` necesita más evaluaciones de la función.

Ejemplo 14

El gráfico adjunto compara los nodos utilizados para integrar la función de Runge con ambas funciones y distintas tolerancias. La figura se obtiene con el código siguiente:

```
subplot 221
quad8('runge',0,10,1e-2,1);
title('QUAD8 con tol=1e-2')
subplot 222
quad('runge',0,10,1e-2,1);
title('QUAD con tol=1e-2')
subplot 223
quad8('runge',0,10,[],1);
title('QUAD8 con tol por defecto')
subplot 224
quad('runge',0,10,[],1);
title('QUAD con tol por defecto')
```



Ejemplo 15

Evaluamos la integral del Ejemplo 11 mediante

```
quad('sinc',eps,2*pi)
ans = 1.41815007017436
```

donde `sinc` es un fichero.m con la expresión `y = sin(x)./x;`

8.3. Problemas

1 **a)** Comprueba que los pesos de la fórmula de Newton-Cotes cerrada de orden 3 en el intervalo [0, 1] son 1/8, 3/8, 3/8 y 1/8. Esta fórmula recibe el nombre de regla de Simpson 3/8.
b) Comprueba que fórmula de orden 2 integra exactamente los mismos polinomios que la de orden 3.
c) Compara la precisión de ambas fórmulas al integrar la función $1/x$ entre 1 y 2.

2 Considera la integral

$$k(\theta) = \int_0^{\pi/2} \frac{d\phi}{\sqrt{1 - \sin^2(\theta)\sin^2(\phi)}}$$

a) Trabajando siempre en radianes, calcula k(30º) por la fórmula de Newton-Cotes abierta de orden 3. (Valor exacto con 4 decimales: 1.6858).
b) Calcula k(85º) por el mismo procedimiento. (Valor exacto con 3 decimales: 3.832). Justifica la magnitud del error, en comparación con el del apartado anterior.

3 Halla el número de condición del sistema planteado para determinar los pesos de la fórmula cerrada de Newton-Cotes de orden 6 en el intervalo [0, 1].
Observa cómo cambia el número de condición al tomar el intervalo $[-1, 1]$.
Escribe un fichero.m para estimar la integral de una función en un intervalo cualquiera por la fórmula de Newton-Cotes de orden 6. El propio fichero debe calcular los pesos teniendo el cuenta lo anterior y aplicar la fórmula de cambio de intervalo.

4 Evalúa la integral

$$\int_0^{2\sqrt{\pi}} \sin x^2 \, dt$$

aplicando la fórmula de Newton-Cotes compuesta cerrada de orden 4 y estima el error cometido, comparando los resultados obtenidos para distinto número de subintervalos.

5 Se mide a intervalos regulares de 0.25 s la velocidad de un móvil en m/s obteniéndose los valores siguientes

t	v
0	0.9162
0.25	0.8109
0.5	0.6931
0.75	0.5596
1	0.4055

Se trata de hallar el espacio recorrido por el móvil, integrando la velocidad en [0, 1].
a) Indica todas las fórmulas de Newton-Cotes simples o compuestas, abiertas o cerradas que pueden utilizarse con los datos disponibles.
b) Evalúa dichas fórmulas para estimar el recorrido.

6 La función densidad de probabilidad de la distribución normal es

$$f(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2}$$

La probabilidad de que una variable aleatoria X con distribución normal tome valores entre $-x$ y x viene dada por la integral

$$p(|X| < x) = \frac{1}{\sqrt{2\pi}} \int_{-x}^{x} e^{-t^2/2} dt = \sqrt{\frac{2}{\pi}} \int_{0}^{x} e^{-t^2/2} dt = \operatorname{erf}(x)$$

Evalúa esta integral por Newton-Cotes de distintos tipos y órdenes, para un valor de x determinado. Analiza el error comparando el resultado con la función `erf` de MATLAB.

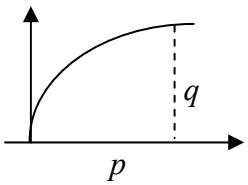
7 La longitud del arco de la curva de ecuación $y = f(x)$ entre los puntos de abscisa $x = a$ y $x = b$ viene dada por la integral

$$L = \int_{a}^{b} \sqrt{1 + f'(x)^2}\, dx$$

Calcula la longitud de una circunferencia de radio 1 aplicando una fórmula de integración abierta para evitar la singularidad del integrando. Indica el error respecto al valor exacto.

8 El área de la superficie de revolución engendrada por la curva de ecuación $y = f(x)$ entre los puntos de abscisa $x = a$ y $x = b$, al girar entorno al eje OX viene dado por la integral

$$A = 2\pi \int_{a}^{b} |y| \sqrt{1 + y'^2}\, dx$$



Halla el área de la superficie de revolución engendrada por la curva $y = q\sqrt{x/p}$, con x entre 0 y p, al girar entorno al eje OX (un plato de antena "parabólica").

8.4. Soluciones

1 **a)** Para obtener la fórmula de Newton-Cotes cerrada de orden 3 en [0, 1], dividimos el intervalo en 3 partes y consideramos como nodos los extremos de los subintervalos, 0, ⅓, ⅔ y 1. Exigiendo que la fórmula integre exactamente polinomios de grado 3, obtenemos el sistema lineal que proporciona los pesos:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & ⅓ & ⅔ & 1 \\ 0 & ⅑ & ⅘ & 1 \\ 0 & 1⁄27 & 8⁄27 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = \begin{pmatrix} 1 \\ ½ \\ ⅓ \\ ¼ \end{pmatrix}$$

Efectuando los cálculos con MATLAB se obtiene el resultado pedido:

```
h = 1/3;
x = 0:h:1;
A = rot90(vander(x));
b = [1; 1/2; 1/3; 1/4];
pesos = A\b

pesos =
    0.1250
    0.3750
    0.3750
    0.1250
```

La fórmula, llamada de Simpson 3/8, queda

$$\int_0^1 f(x)\,dx \cong \frac{1}{8}\left(f(0)+3f(⅓)+3f(⅔)+f(1)\right)$$

b) Por construcción, la fórmula de Newton-Cotes de orden 3 integra exactamente polinomios de tercer grado. La fórmula de orden 2 está diseñada para integrar polinomios de grado 2, pero también resulta exacta para los de grado 3. En efecto,

$$\int_0^1 x^3\,dx = \frac{1}{4} = \frac{1}{6}\left(0+4\frac{1}{8}+1\right)$$

c) El valor exacto de la integral es

$$\int_1^2 \frac{1}{x}\,dx = \log 2$$

Estimándola por Newton-Cotes de segundo orden obtenemos

$$I_2 = \frac{1}{6}\left(f(1)+4f\left(\frac{3}{2}\right)+f(2)\right) = \frac{1}{6}\left(1+4\frac{2}{3}+\frac{1}{2}\right) = \frac{25}{36}$$

y por Newton-Cotes de tercer orden

$$I_3 = \frac{1}{8}\left(f(1)+3f\left(\frac{4}{3}\right)+3f\left(\frac{5}{3}\right)+f(2)\right) = \frac{1}{8}\left(1+3\frac{3}{4}+3\frac{3}{5}+\frac{1}{2}\right) = \frac{111}{160}$$

Los errores respectivos son

$$\log 2 - I_2 \cong -0.0013$$

y

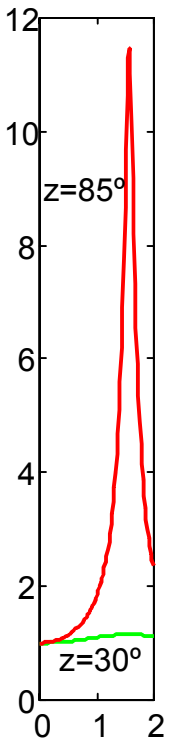
$$\log 2 - I_3 \cong -0.0006$$

La fórmula de tercer orden sólo reduce el error a la mitad, a costa de evaluar la función en un nodo más.

2 Dividimos el intervalo $[0, 2\pi]$ en 5 partes iguales. Los extremos interiores son los nodos de integración. Los pesos se obtienen resolviendo el sistema que resulta al exigir que la fórmula integre exactamente los polinomios de grado 4.

```
a = 0; b = pi/2;
n = 5; h = (b-a)/n;
x = a+h : h : b-h;     % Nodos
A = rot90(vander(x));
c = [b; b^2/2; b^3/3; b^4/4];
pesos = A\c

pesos =
    0.7199
    0.0654
    0.0654
    0.7199
```

a) Evaluamos el integrando en los nodos y aplicamos los pesos para estimar la integral




```
z = pi/6
y = 1./sqrt(1-(sin(z)*sin(x)).^2);
I = y*pesos

z = 0.5236
I = 1.6871
```

El error es

```
1.6858-I

ans = -0.0013
```

b) Repetimos los cálculos para el ángulo de 85º. El resultado es ahora

```
z = 1.4835
I = 3.1982
```

Y el error

```
3.832-I
ans = 0.6338
```

Del aspecto de las gráficas se deduce que el comportamiento de la segunda es mucho más variable que el de la primera y esto se traduce en un mayor error al calcular la integral.

3 Para calcular los pesos de la fórmula de Newton-Cotes cerrada de orden 6, tomamos 7 puntos equiespaciados de 0 a 1 y calculamos la matriz del sistema resultante al imponer que la fórmula integre exactamente polinomios de grado 6.

```
nodos = 0 : 1/6 : 1;
A = rot90(vander(nodos));
```

El número de condición de A es bastante elevado.

```
disp(cond(A))
  3.6061e+004
```

Repitiendo los cálculos en el intervalo [−1, 1] se observa que el número de condición se reduce notablemente.

```
nodos = -1 : 1/3 : 1;
A = rot90(vander(nodos));
disp(cond(A))
  189.8141
```

Para utilizar la fórmula de Newton-Cotes en un intervalo cualquiera [a, b], determinamos los pesos para un intervalo normalizado [−1, 1] y los multiplicamos por (b–a)/2. Los valores de la función se calculan en nodos equiespaciados en el intervalo [a, b].

4 Teniendo en cuenta las observaciones del problema 3, evaluamos primero los pesos de la fórmula de Newton-Cotes simple cerrada de orden 4 en el intervalo normalizado [−1, 1].

```
nodos = -1 : 0.5 : 1;
A = rot90(vander(nodos));
c = 2*[1; 0; 1/3; 0; 1/5];
pesos = A\c
pesos =
    0.1556
    0.7111
    0.2667
    0.7111
    0.1556
```

Para aplicar la fórmula compuesta, dividimos el intervalo de integración en m subintervalos iguales y estimamos la integral en cada uno de ellos mediante la fórmula simple multiplicada por el factor de escala, que es el cociente entre la longitud h de cada subintervalo y la del intervalo normalizado, que es 2. Finalmente, sumamos los resultados de cada trozo para obtener la estimación le la integral en el intervalo total.

```
a = 0; b = 2*sqrt(pi);
m = 10; h = (b-a)/m;
x = a : h : b;          % Extremos de los subintervalos
I = 0;                  % Inicialización de la integral
for k = 1:m
     nodos = x(k) : h/4 : x(k+1);
     y = sin(nodos.^2);
     p(k) = h/2*y*pesos;   % Integral en el subintervalo
end
I = sum(p)       % Integral en el intervalo total

I = 0.4863
```

Para estimar la precisión, repetimos los cálculos con $m = 20$, obteniendo

```
I2 = 0.48624731612146;
```

La diferencia entre ambos resultados es una estimación del error de la fórmula

```
I-I2 = 2.0972e-005
```

5 **a)** La tabla adjunta muestra las fórmulas aplicables con los pesos correspondientes a cada nodo y la estimación que resulta.

				Nodos					
				0	1/4	1/2	3/4	1	
Tipo	**Orden**	**Interv.**	**Nombre**	**Pesos**					**Valor**
A	0	1	Punto medio simple			1			0.6931
A	0	2	Punto medio compta.		1/2		1/2		0.68525
A	2	1	Newton-Cotes		2/3	−1/3	2/3		0.68263
C	1	1	Trapecios simple	1/2				1/2	0.66085
C	1	2	Trapecios compta.	1/4		1/2		1/4	0.67698
C	1	4	Trapecios compta.	1/8	1/4	1/4	1/4	1/8	0.68111
C	2	1	Simpson simple	1/6		2/3		1/6	0.68235
C	2	2	Simpson compta.	1/12	1/3	1/6	1/3	1/12	0.68249
C	4	1	Newton-Cotes	7/90	32/90	12/90	32/90	7/90	0.6825

NEWTON COTES
---

ejemplo del seno

```
>> y = [ 0 1/2 sqrt(2)/2 sqrt(3)/2 1] %se puede hacer y=sin(x)
>> x = [0 pi/6 pi/4 pi/3 pi/2];
>> A = [x.^0; x.^1; x.^2; x.^3; x.^4]
A =
   1.0000    1.0000    1.0000    1.0000    1.0000
        0    0.5236    0.7854    1.0472    1.5708
        0    0.2742    0.6169    1.0966    2.4674
        0    0.1435    0.4845    1.1484    3.8758
        0    0.0752    0.3805    1.2026    6.0881
>> q = pi/2; %Longitud del intervalo
>> b = [q; q^2/2; q^3/3; q^4/4; q^5/5]
b =
   1.5708
   1.2337
   1.2919
   1.5220
   1.9126
>> pesos = A\b % si haces sum pesos tiene que dar b
pesos =
  0.14398966328953
  1.06028752058655
 -0.83775804095728
  1.06028752058655
  0.14398966328953
>> format long
>> I = dot(pesos,y)
I =
  0.99998495997193
```

---
ejemplo 2

```
>> %area de la cardioide, adptando los nodos ya obtenidos
>> nodos = 2*x
>> pesos = 2 * pesos;
>> %introducimos la funcion:
>> y = (1+cos(nodos)).^2/2;
>> I = dot(pesos,y)
I =
   2.3889
>> (3*pi/4)-I
ans =
  -0.0327
```

---
ejemplo 3

```
>> % cambiar intervalo [0, pi] a [-1,1]
>> pesos = (2/pi)*pesos
pesos =
   0.1833
   1.3500
  -1.0667
```

```
    1.3500
    0.1833
>> nodos = (2/pi) * nodos - 1
nodos =
   -1.0000   -0.3333         0    0.3333    1.0000
>> y = 1./(1+nodos.^2)
y =
    0.5000    0.9000    1.0000    0.9000    0.5000
>> I = dot(pesos,y)
I =
    1.5467
>> pi/2 - I %error cometido
ans =
    0.0241
```

---
ejemplo con mas nodos

```
>> nodos = 0:pi/4:pi;
>> A = rot90(vander(x));
>> q = pi; b = [q; q.^2/2; q.^3/3; q.^4/4; q.^5/5]
b =
    3.1416
    4.9348
   10.3354
   24.3523
   61.2039
>> pesos = A\b
pesos =
    0.2443
    1.1170
    0.4189
    1.1170
    0.2443
>> %con estos pesos puedo integrar cualqueir funcion en el intervalo [0 pi] o generalizarlo a cualqueir intervalo mediante el itercambio de coordenadas
>> sum(pesos) %comprobacion rutinaria
ans =
    3.1416
>> y = (1+cos(nodos)).^2/2;
>> I = dot(pesos,y)
I =
    5.6430 (puede que el resultado este mal,he cambiado una cosa)
```

>> %N-C abierto

```
>> nodos = 1/6:1/6:(1-1/6);
>> A = rot90(vander(nodos));
>> q=1;
>> b = [q; q.^2/2; q.^3/3; q.^4/4; q.^5/5];
>> pesos = A\b
pesos =
    0.5500
   -0.7000
    1.3000
   -0.7000
    0.5500
>> sum(pesos) %la formula del algodon...
```

```
ans =
     1
>> y = 1./sqrt(nodos);
>> I = dot(pesos,y)
I =
    1.7184
>> %tiene mucho error (debiera ser 1) porque hemos elegido pocos nodos
>> %y la funcion se comporta mal (tiene asintita en x=0)!
```

---
otro ej que se comporte mejor
la cardioide

```
>> nodos = pi*nodos;
>> pesos = pi*pesos
pesos =
    1.7279
   -2.1991
    4.0841
   -2.1991
    1.7279
>> y = (1+cos(nodos)).^2/2;
>> I = dot(pesos,y)
I =
    2.3169
>> 3*pi/4 - I
ans =
    0.0393
    2.3169
```

Punto medio compuesta

```
>> h = 1/50;
>> x = h/2: h: 1-h/2;
>> y = 1./sqrt(x);
>> I = h*sum(y) %en este caso es facil, porque los pesos son todos iguales y son h.
I =
    1.9145


>> h = 1/500;
>> x = h/2: h: 1-h/2;
>> y = 1./sqrt(x);
>> I = h*sum(y)
I =
    1.9729
```

10 Interpolación Polinómica

10.1 Introducción

En temas anteriores hemos estudiado problemas en los que aparecía una función, dando directamente su expresión en la forma $y = f(x)$, o bien los datos necesarios para establecerla. Esto nos permitía realizar un estudio sobre las características y el comportamiento, y utilizarlas en la resolución del problema.

Ahora nos planteamos la situación inversa, es decir, a partir de una serie de puntos (x_i, y_i), $i = 1,...,n+1$, hemos de encontrar una función *f(x),* que verifique todos estos valores. La técnica que nos permite, a partir de unas observaciones, hallar los valores que toma la función desconocida *f(x)* se conoce como interpolación.

La necesidad de plantear un problema de este tipo deriva del siguiente hecho: En la mayoría de casos, las normas que rigen el comportamiento humano, la evolución de la naturaleza, etc. no son descritas por una fórmula concreta y conocida, normalmente, sólo se dispone de una serie más o menos amplia de valores, obtenidos de forma experimental, que nos dan la pauta de su comportamiento; a partir de éstos podremos deducir, con un error más o menos controlado, datos no considerados en el estudio.

Así, podemos estimar la altura de un individuo teniendo en cuenta la nuestra propia y la de nuestros conocidos, o el tiempo que tardará en llegar a cierto punto un vehículo sabiendo qué tiempo ha tardado en distancias y circunstancias similares.

Otra aplicación de la interpolación es encontrar valores aproximados de una función conocida pero difícil de estimar en determinados puntos. Para las funciones trigonométricas y logarítmicas ha sido tradicional el uso de tablas; el cálculo de los valores no señalados en las tablas había que obtenerlo por interpolación. Las calculadoras han arrinconado las tablas, pero los procedimientos siguen siendo válidos.

La importancia de la interpolación sigue vigente, siendo la base de algoritmos de diseño gráfico, de métodos de integración aproximada y de evaluación de funciones de matrices, entre otros.

10.2 Interpolación polinómica

Ante la diversidad de funciones interpoladoras que se pueden construir, es lógico elegir la de cálculo más simple y mejor comportamiento, selección que recae en la función polinómica.

Por tanto el objetivo es la obtención del polinomio de interpolación, que es el polinomio de grado menor o igual que n que pasa por n+1 puntos (x_i, y_i), $i = 1,...,n+1$, también llamados nodos de interpolación.

Planteando directamente las condiciones anteriores se obtiene un sistema de ecuaciones lineales con solución única, pero generalmente mal condicionado.

Los polinomios de Lagrange permiten obtener una expresión explícita del polinomio de interpolación cuyo interés es más bien teórico, pues es costosa de evaluar en puntos concretos.

Numéricamente es mucho más útil la forma de Newton del polinomio de interpolación. Aunque no tiene expresión explícita, su obtención es más estable que por los métodos anteriores, su evaluación no presenta los inconvenientes de los polinomios de Lagrange, y sobre todo, se puede actualizar fácilmente si se añaden nuevos nodos de interpolación.

Interpolación lineal

Si el polinomio es de grado 1 hablamos de interpolación lineal, tremendamente útil por su fácil cálculo y por su error reducido si los puntos están suficientemente próximos. La utilizaremos si los datos que poseemos son dos puntos (x_1, y_1) y (x_2, y_2), o se eligen sólo dos puntos entre varios para calcular la aproximación en un valor intermedio, a partir de la recta que pasa por dichos puntos.

Ejemplo 1

Sabiendo que $\log_{10} 10 = 1$ y $\log_{10} 100 = 2$. Calcula por interpolación lineal el logaritmo en base 10 de 90.

Los puntos de interpolación son (10,1) y (100,2) calculamos pues la ecuación de la recta que pasa por ellos.

$$p_1(x) = a_1 x + a_2$$
$$p_1(10) = 10a_1 + a_2 = 1$$
$$p_1(100) = 100a_1 + a_2 = 2$$

Resolviendo el sistema obtendremos los coeficientes del polinomio y podremos evaluarlo en puntos intermedios.

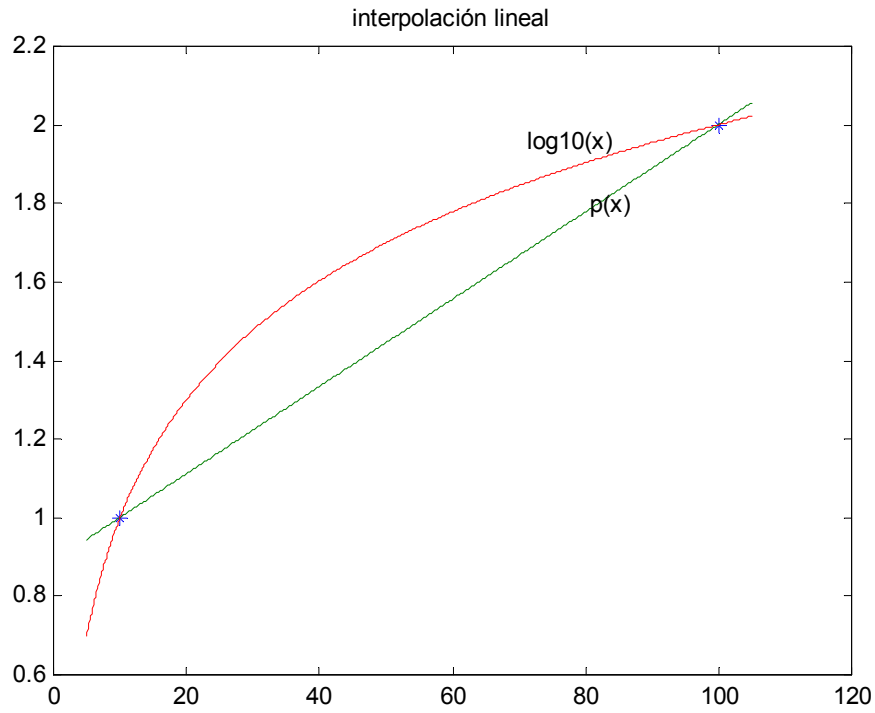
```
A=[10 1;100 1];
y=[1;2];
p=A\y;
p90=polyval(p,90), %valor aproximado del log10 90
%Representación de la función f(x) = log10(x) junto al polinomio
que la interpola, y los nodos de interpolación .
x=[10 100];y=[1 2];
xg=5:.1:105;
yg=log10(xg);
yp=polyval(p,xg);
plot(x,y,'*',xg,yp,xg,yg,'r')
Title('interpolación lineal')
```

```
p90 =
    1.8889
```

Si calculamos con MATLAB `log10(90)`

```
ans =
    1.9542
```

Observamos que el error cometido `ans-p90` es considerable ello es debido a que los puntos de interpolación son muy distantes. Si elegimos abscisas cercanas, el error es mucho menor, de forma que la gráfica de la función logarítmica y la recta que la interpola en un intervalo muy pequeño prácticamente coinciden. Esto justifica el uso de la interpolación lineal para calcular valores de una función tabulada



Interpolación cuadrática

Si disponemos de tres puntos que determinan una parábola tendremos un polinomio de grado 2, se trata de interpolación cuadrática.

El polinomio de grado dos $p_2(x) = a_1x^2 + a_2x + a_3$ que pasa por (x_1, y_1), (x_2, y_2) y (x_3, y_3) se determina análogamente resolviendo el sistema.

$$a_1x_1^2 + a_2x_1 + a_3 = y_1$$
$$a_1x_2^2 + a_2x_2 + a_3 = y_2$$
$$a_1x_3^2 + a_2x_3 + a_3 = y_3$$

Ejemplo 2

De un muelle con un extremo fijo colgamos distintas masas obteniendo la siguiente tabla:

Masa (kg)	5	10	15
Longitud (cm)	395	620	795

Utiliza la interpolación cuadrática para estimar la longitud del muelle para una masa de 7 Kg.

Imponiendo las condiciones de interpolación (5,395), (10,620) y (15,795) tenemos un sistema de ecuaciones lineales cuya expresión matricial es:

$$\begin{pmatrix} 25 & 5 & 1 \\ 100 & 10 & 1 \\ 225 & 15 & 1 \end{pmatrix}\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}=\begin{pmatrix} 395 \\ 620 \\ 795 \end{pmatrix}$$

Observemos que la matriz del sistema es la matriz de Van der Monde. Esta matriz está mal condicionada para tamaños relativamente pequeños. Esto hace desaconsejable la obtención del polinomio de interpolación por este método.
Creamos en MATLAB dos vectores con las abscisas y las ordenadas, respectivamente de los puntos a interpolar:

```
x=[5;10;15];
y=[395;620;795];
A=vander(x);
cond(A)

ans =
  1.1079e+003
```

Resolviendo el sistema se obtienen los coeficientes del polinomio buscado.

```
p=A\y

p =
   -1.0000
   60.0000
  120.0000
```

Por lo que el polinomio de interpolación se tiene la siguiente expresión:

$$p_2(x) = -x^2 + 60x + 120$$

Comprobamos que cumple las condiciones de interpolación, o sea, que pasa por los puntos dados.

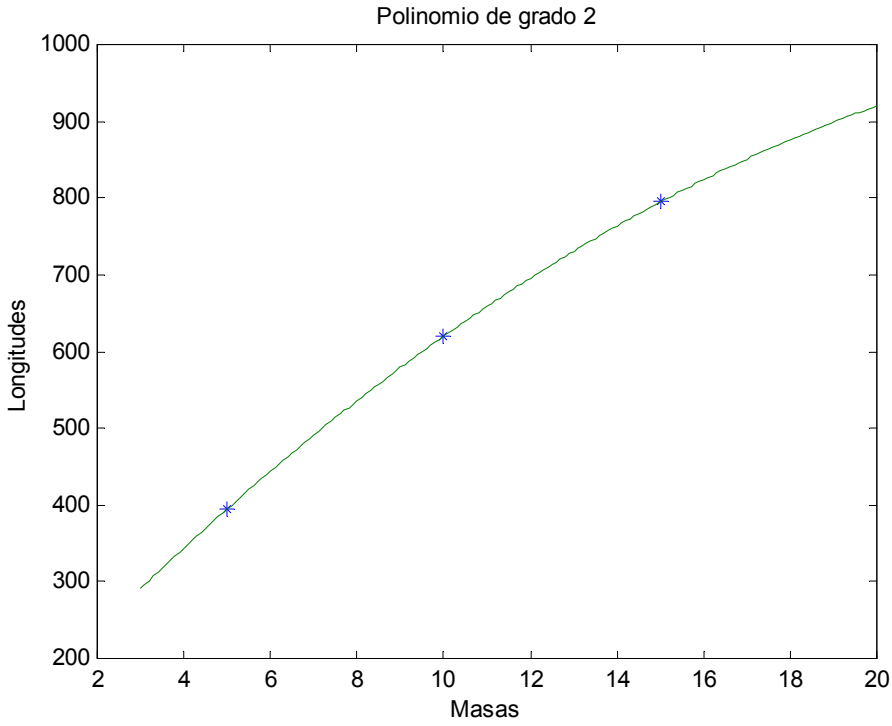
```
polyval(p,x)

ans =
  395.0000
  620.0000
  795.0000
```

Valor esperado en $m = 7$ kg

```
v=polyval(p,7)
v =
  491.0000
```

Representamos el polinomio junto a los puntos de interpolación.

```
xn=3:.1:20;
yn=polyval(p,xn);
plot(x,y,'*',xn,yn)
xlabel('Masas')
ylabel('Longitudes')
Title('Polinomio de grado 2')
```



Interpolación polinómica en general

El problema de interpolación es más general y se adapta mejor a los casos reales si se dispone de una serie de puntos, obtenidos de forma experimental, y analizamos si con una interpolación lineal o cuadrática se obtiene una buena estimación o necesitamos un polinomio de interpolación de mayor grado.
El hecho de aumentar el grado del polinomio de interpolación no implica que se obtenga mejor aproximación puesto que la coincidencia del polinomio con muchos puntos de interpolación se consigue a costa de grandes oscilaciones en los intervalos entre nodos o puntos de interpolación dados.
La obtención de este polinomio se basa en el siguiente resultado:.
Dados $n+1$ *puntos de abscisas distintas* (x_1, y_1), (x_2, y_2),..., (x_{n+1}, y_{n+1}), *existe un único polinomio de grado menor o igual que n, cumpliendo las condiciones de interpolación*

$$P_n(x_i) = y_i, \quad i = 1,2,...,n+1.$$

Para demostrar este resultado basta expresar el polinomio buscado en forma normal

$$P_n(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_{n+1}$$

e imponer las condiciones de interpolación, con lo que se obtiene el siguiente sistema de $n+1$ ecuaciones con $n+1$ incógnitas

$$\begin{pmatrix} x_1^n & x_1^{n-1} & x_1^{n-2} & \cdots & 1 \\ x_2^n & x_2^{n-1} & x_2^{n-2} & \cdots & 1 \\ x_3^n & x_3^{n-1} & x_3^{n-2} & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n+1}^n & x_{n+1}^{n-1} & x_{n+1}^{n-2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n+1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n+1} \end{pmatrix}$$

El determinante de la matriz del sistema es un determinante de Van der Monde de la forma:

$$V(x_1, x_2, x_3, \ldots, x_{n+1}) = \prod_{1 \le i < j \le n+1} (x_j - x_i)$$

y es no nulo al ser distintos por hipótesis, los n+1 valores de x. Por lo tanto, el sistema es compatible determinado, es decir, admite solución única.

Este resultado, aparte de asegurar la existencia y unicidad del polinomio de interpolación, proporciona también un método de cálculo que, en general, suele ser muy laborioso y presenta el problema del mal condicionamiento de la matriz del sistema.
El polinomio de interpolación obtenido mediante los valores (x_1, y_1), (x_2, y_2),..., (x_{n+1}, y_{n+1}) se utiliza para estimar valores en puntos intermedios a los que se han dado, es decir $x : x_1 \le x \le x_{n+1}$, si se utiliza para un valor de x que no pertenece al intervalo $[x_1, x_{n+1}]$ estaremos ante un problema de extrapolación, el método de operar es idéntico pero con riesgo de obtener resultados con errores considerables, ya que no se puede garantizar el comportamiento fuera del intervalo.
Vamos a crear una función en MATLAB que utilizando el método expuesto calcule el polinomio de interpolación que pasa por todos los puntos.

Algoritmo INTERPOL
Entrada: (x_i, y_i) $i = 1,.. ., n+1$, puntos de interpolación.
Salida: $P_n(x)$ polinomio de interpolación de grado n
Proceso:
n = longitud del vector de nodos –1; (grado del polinomio)
V= vander(n+1)
p=V\y;
xg=linspace(x(1),x(n+1));
yg=polyval(p,xg);
plot(xg,yg)
title([‘polinomio de interpolación de grado ’,num2str(n)])

Ejemplo 3

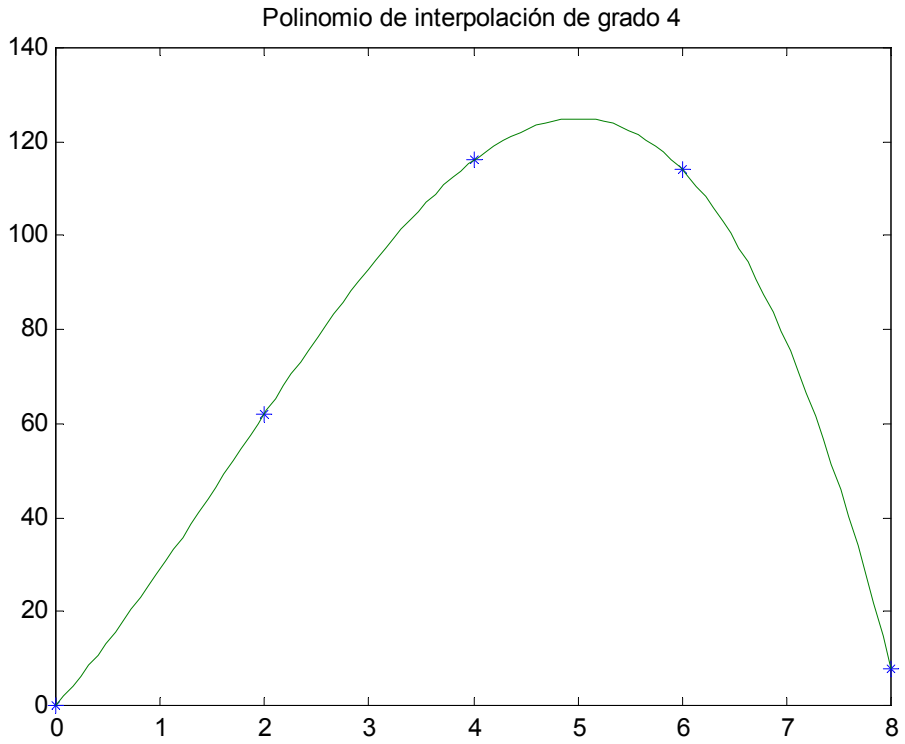
Un estudio sobre la eficiencia de los trabajadores en una factoría ha determinado que el promedio de piezas producidas por hora de sus trabajadores está relacionado con el número de horas transcurridas a partir del comienzo de la jornada. A partir de la siguiente tabla haz una estimación del número de piezas producidas después de 3, 5 y 7 horas de trabajo.

Horas transcurridas	0	2	4	6	8
Piezas producidas	0	62	116	114	8

En este caso si consideramos todos los nodos o puntos de interpolación podremos hallar el polinomio de grado 4 resolviendo el sistema de ecuaciones lineales de 5×5 vamos a utilizar nuestra función **interpol** ya preparada en MATLAB, a la que añadimos una instrucción para obtener el número de condición de la matriz del sistema:

```
x=0:2:8;
y=[0 62 116 114 8];
p=interpol(x,y);
```

```
v=polyval(p,[3,5,7])


ncon =

  1.4361e+004
v =
   93.0000  125.0000   77.0000
```



Podemos establecer como conclusión que el rendimiento de los empleados va aumentando a lo largo de la jornada hasta que alrededor de las cinco horas de trabajo se alcanza la producción máxima y a partir de ese momento la producción empieza a disminuir.
Si en este problema nos preguntáramos sobre la producción al cabo de 10 horas de trabajo estaríamos ante un problema de extrapolación que proporciona un resultado inadmisible al evaluar el polinomio de interpolación.

```
v=polyval(p,10)
v =
 -250.0000
```

Desplazamiento del origen

El problema del mal condicionamiento de la matriz de Van der Monde asociada al sistema que resulta al hallar el polinomio de interpolación puede resolverse mediante un desplazamiento del origen, es decir, eligiendo otra base para expresar este polinomio. Desplazar el origen a un nodo x_i implica expresar el polinomio como una combinación lineal de potencias de $x - x_i$:

$$P_n(x) = b_1(x - x_i)^n + b_2(x - x_i)^{n-1} + \ldots + b_{n+1}$$

La condición $P_n(x_i) = y_i$ proporciona directamente el valor de b_{n+1} y queda un sistema mejor condicionado que el anterior. Esta mejora no es definitiva, pues la matriz del nuevo sistema es parecida a la de Van der Monde y para mayor grado reaparecerá el mal condicionamiento.

Ejemplo 4

Vamos a resolver el Ejemplo 2 realizando un desplazamiento del origen a $x_1 = 5$. Comprueba la mejora del número de condición de la matriz del sistema, expresa el polinomio respecto de la nueva base matizando que se trata del mismo polinomio que el hallado en dicho ejemplo.

```
x=[5;10;15];
y=[395;620;795];
A=vander(x);
ncond1=cond(A)
p=A\y;
xd=x-5;
Ad=vander(xd);
ncond=cond(Ad)
pd=Ad\y

ncond1 =
  1.1079e+003
ncond =
  109.0659
pd =
    -1
    50
   395
```

Observamos que aunque el número de condición ha mejorado todavía sigue siendo alto. La expresión del polinomio será:

$$p_2(x) = -(x-5)^2 + 50(x-5) + 395$$

Como el polinomio de interpolación es único éste coincidirá con el anteriormente hallado:

$$p_2(x) = -x^2 + 60x + 120$$

Para comprobarlo evaluamos los dos en un mismo conjunto de puntos y vemos que los resultados coinciden;

```
xn=linspace(5,15,10);
vp=polyval(p,xn);
vpd=polyval(pd,xn-5);%observa como se evalúa el polinomio desplazado
E=max(abs(vp-vpd))

E =
  1.1369e-013
```

Método de Lagrange

Este método de interpolación consiste en obtener directamente el polinomio de interpolación expresado respecto de otra base de forma que no precisa realizar ningún cálculo, la notación utilizada se debe a Lagrange quien proporcionó las fórmulas.

En el caso de la interpolación cuadrática dados los nodos (x_1, y_1), (x_2, y_2) y (x_3, y_3) el polinomio será de la forma:

$$P(x) = y_1 L_1(x) + y_2 L_2(x) + y_3 L_3(x)$$

de forma que:

$$L_1(x_1) = 1; \quad \mathrm{L}_1(x_2) = 0; \quad \mathrm{L}_1(x_3) = 0;$$

Por lo que

$$L_1(x) = a_1(x - x_2)(x - x_3)$$

por tanto

$$L_1(x_1) = 1 = a_1(x_1 - x_2)(x_1 - x_3) \rightarrow a_1 = \frac{1}{(x_1 - x_2)(x_1 - x_3)}$$

y en consecuencia queda determinado este factor

$$L_1(x) = \frac{(x - x_1)(x - x_2)}{(x_1 - x_2)(x_1 - x_3)}$$

Análogamente tendremos

$$L_2(x_1) = 0; \quad \mathrm{L}_2(x_2) = 1; \quad \mathrm{L}_2(x_3) = 0;$$
$$L_3(x_1) = 0; \quad \mathrm{L}_3(x_2) = 0; \quad \mathrm{L}_2(x_3) = 1;$$

Estos polinomios serán:

$$L_2(x) = \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)}$$

$$L_3(x) = \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

Así pues la expresión completa del polinomio de Lagrange es:

$$P_2(x) = y_1 \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + y_2 \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + y_3 \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

y claramente se tiene

$$P_2(x_i) = y_i \quad i = 1, 2, 3.$$

En general dados n+1 nodos (x_1, y_1), (x_2, y_2),..., (x_{n+1}, y_{n+1}), consideramos $L_i(x)$ un polinomio de grado n, que se anule en todos los puntos $x_j, j = 1, ..., n+1$, salvo en el i-ésimo, donde vale 1; es decir, tal que

$$L_i(x_j) = 0 \ \ si\, j \neq i \ \ y \ L_i(x_i) = 1$$

Razonando de forma análoga al caso $n = 2$ se obtiene la siguiente expresión para estos polinomios:

$$\mathrm{L}_i(x) = \frac{(x - x_1)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_{n+1})}{(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_{n+1})} = \prod_{\substack{i=1 \\ j \neq i}}^{n+1} \frac{x - x_i}{x_i - x_j}$$

Es inmediato comprobar entonces que el polinomio

$P_n(x) = y_1 L_1(x) + y_2 L_2(x) + y_3 L_3(x) + \cdots + y_{n+1} L_{n+1}(x) = \prod_{i=1}^{n+1} y_i L_i(x)$ cumple las condiciones

$P_n(x_i) = y_i,\ i = 1,2...,n+1.$ lo que prueba directamente la existencia del polinomio de interpolación. La unicidad se puede garantizar utilizando el hecho de que un polinomio de grado *n* puede tener a lo sumo *n* raíces. Si dos polinomios de grado ≤ *n* interpolan *n*+1 puntos, su diferencia se anula en dichos puntos, por lo que sólo puede ser el polinomio idénticamente nulo.

Ejemplo 5

Expresa el polinomio de interpolación de Lagrange para los datos del Ejemplo 2:

Directamente sin realizar ningún cálculo a partir de los nodos (5,395), (10,620) y (15,795) podemos expresar el polinomio de grado 2:

$$P_2(x) = 395\frac{(x-10)(x-15)}{(5-10)(5-15)} + 620\frac{(x-5)(x-15)}{(10-5)(10-15)} + 795\frac{(x-5)(x-10)}{(15-5)(15-10)}$$

Las operaciones que nos hemos ahorrado en su determinación, hemos de pagarlas al evaluar el polinomio en un punto concreto (del orden de n^2 operaciones por cada evaluación). Además, los productos a efectuar pueden causar desbordamiento numérico y la fórmula no es estable numéricamente.

Método de Newton

El primer método estudiado para hallar el polinomio de interpolación, que resulta de forma natural al plantear las condiciones de interpolación, da lugar a un sistema de ecuaciones lineales mal condicionado por lo que su utilización no resulta aconsejable, aunque una vez obtenido resulta muy cómodo de evaluar para obtener estimaciones. Lo contrario sucede con el método de Lagrange que proporciona directamente la expresión del polinomio de interpolación con el inconveniente de la evaluación del mismo en un punto dado, por el elevado número de operaciones que se precisan realizar.
Vamos a ver ahora el método de Newton que salva los inconvenientes anteriores y resulta ser el más útil y práctico.
Este nuevo método consiste en expresar el polinomio de interpolación respecto de la base 1, $x - x_1$, $(x - x_1)(x - x_2)$, ..., $(x - x_1)(x - x_2) \cdots (x - x_n)$. Por lo que su expresión será:

$$P_n(x) = c_{n+1} + c_n(x - x_1) + c_{n-1}(x - x_1)(x - x_2) + \cdots + c_1(x - x_1)(x - x_2) \cdots (x - x_n)$$

Imponiendo las condiciones de interpolación, podemos determinar los coeficientes de este polinomio.

$$
\begin{aligned}
P_n(x_1) &= y_1 = c_{n+1} \\
P_n(x_2) &= y_2 = c_{n+1} + c_n(x_2 - x_1) \\
P_n(x_3) &= y_3 = c_{n+1} + c_n(x_3 - x_1) + c_{n-1}(x_3 - x_2)(x_3 - x_1) \\
&\cdots \\
P_n(x_{n+1}) &= y_{n+1} = c_{n+1} + c_n(x_{n+1} - x_1) + c_{n-1}(x_{n+1} - x_1)(x_{n+1} - x_2) + \cdots \\
&\quad + c_1(x_{n+1} - x_1)(x_{n+1} - x_2) \cdots (x_{n+1} - x_n)
\end{aligned}
$$

El sistema lineal obtenido tiene una matriz análoga a la de Van der Monde, pero con la ventaja de ser triangular inferior.

$$
\begin{pmatrix}
1 & 0 & 0 & \cdots & 0 \\
1 & x_2 - x_1 & 0 & \cdots & 0 \\
1 & x_3 - x_1 & (x_3 - x_1)(x_3 - x_2) & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & 0 \\
1 & x_{n+1} - x_1 & (x_{n+1} - x_1)(x_{n+1} - x_2) & \cdots & (x_{n+1} - x_1)(x_{n+1} - x_2)\cdots(x_{n+1} - x_n)
\end{pmatrix}
\begin{pmatrix} c_{n+1} \\ c_n \\ c_{n-1} \\ \vdots \\ c_1 \end{pmatrix} =
\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n+1} \end{pmatrix}
$$

Los coeficientes pueden determinarse con menos operaciones (del orden de n^2, en lugar de n^3), no obstante en la mayoría de casos va a persistir el mal condicionamiento del sistema.

Es importante notar que el hecho de añadir nuevos nodos no afecta a los coeficientes ya calculados, por lo que se pueden ir añadiendo nuevos puntos aumentando el grado del polinomio y conseguir la precisión deseada sin tener que calcular de nuevo todos los coeficientes. Es decir,

c_3	es el polinomio de grado 0 que pasa por (x_1, y_1),
$c_3 + c_2(x - x_1)$	es el polinomio de grado 1 que pasa por (x_1, y_1) y (x_2, y_2),
$c_3 + c_2(x - x_1) + c_1(x - x_1)(x - x_2)$	es el polinomio de grado 2 que pasa por (x_1, y_1), (x_2, y_2) y (x_3, y_3)

En general, cada polinomio se obtiene del anterior mediante

$$P_i(x) = P_{i-1}(x) + c_1(x - x_1)(x - x_2) \cdots (x - x_{i-1}))$$

Como consecuencia de este hecho se suelen tomar los nodos en cierto orden, considerando primero los más próximos al punto que nos interesa, aunque el polinomio obtenido finalmente es independiente del orden
considerado.

Ejemplo 6

Utiliza el método de Newton para obtener el polinomio de interpolación de grado 3 que evaluarías para estimar en el Ejemplo 3 el número de piezas producidas por los empleados al cabo de 5 horas trabajadas: .

Horas transcurridas	0	2	4	6	8
Piezas producidas	0	62	116	114	8

El polinomio de interpolación de Newton de grado 3 tendrá la forma:

$$p_3(x) = c_4 + c_3(x - x_1) + c_2(x - x_1)(x - x_2) + c_1(x - x_1)(x - x_2)(x - x_3)$$

Para determinarlo se necesitan 4 nodos, elegimos los más cercanos al punto que se quiere realizar la estimación, es decir (4,116), (6,114), (2,62), y (8,8). Sustituyendo estos nodos en la expresión anterior obtenemos el siguiente sistema triangular:

$$\begin{aligned} P_3(4) &= 116 = c_4 \\ P_3(6) &= 114 = c_4 + 2c_3 \\ P_3(2) &= 62 = c_4 - 2c_3 + 8c_2 \\ P_3(8) &= 8 = c_4 + 4c_3 + 8c_2 + 48c_1 \end{aligned}$$

que directamente proporciona la solución:

```
c4=116,c3=(114-c4)/2, c2=(62-c4+2*c3)/8,c1=(8-c4-4*c3-8*c2)/48 ,

c4 =
   116
c3 =
    -1
c2 =
    -7
c1 =
    -1
```

Por lo que el polinomio tendrá la siguiente expresión:

$$p_3(x) = 116 - (x - 4) - 7(x - 4)(x - 6) - (x - 4)(x - 6)(x - 8)$$

Notemos que

$p_0(x) = 116$	es el polinomio de grado 0 que pasa por (4,116),
$p_1(x) = 116 - (x - 4)$	es el polinomio de grado 1 que pasa por (4,116) y (6,114),
$p_2(x) = 116 - (x - 4) - 7(x - 4)(x - 6)$	es el polinomio de grado 2 que pasa por (4,116), (6,114) *y (*2,62*)*

Si resolvemos el sistema por el método de Gauss puede haber desbordamiento numérico debido al mal condicionamiento de la matriz, en particular para tamaños mayores:

```
A=[0 0 0 1;0 0 2 1;0 8 -2 1;48 8 4 1];
b=[116 114 62 8]';
cond(A)
p=A\b

ans =
   57.5941
p =
    -1
```

```
    -7
    -1
   116
```

Por lo que normalmente se recurre a otra técnica, la tabla de diferencias divididas, que proporciona los coeficientes del polinomio sin necesidad de resolver el sistema de ecuaciones lineales.

Tabla de diferencias divididas

Dado el polinomio de interpolación de Newton

$$P_n(x) = c_{n+1} + c_n(x - x_1) + c_{n-1}(x - x_1)(x - x_2) + \cdots + c_1(x - x_1)(x - x_2)\cdots(x - x_n)$$

Imponiendo las condiciones de interpolación, podemos determinar sus coeficientes:

$P_n(x_1) = y_1 = c_{n+1}$ que denotaremos $f[x_1]$, teniendo así:

$$c_{n+1} = f[x_1]$$

$P_n(x_2) = y_2 = c_{n+1} + c_n(x_2 - x_1)$ de donde

$$c_n = \frac{y_2 - f[x_1]}{x_2 - x_1} = \frac{f[x_2] - f[x_1]}{x_2 - x_1}$$

Esta expresión se denomina cociente de diferencias o diferencias divididas de primer orden, proporciona el valor de c_n en función de los puntos de interpolación, y se denota por $f[x_1x_2]$, es decir $c_n = f[x_1x_2]$ y el polinomio de Newton de grado 1 es de la forma:

$$p_1(x) = f[x_1] + f[x_1x_2](x - x_1)$$

Veamos que el polinomio de grado 2 tiene la forma:

$$p_2(x) = f[x_1] + f[x_1x_2](x - x_1) + f[x_1x_2x_3](x - x_1)(x - x_2)$$

siendo $f[x_1x_2x_3] = \frac{f[x_2x_3] - f[x_1x_2]}{x_3 - x_1}$ el cociente de diferencias divididas de orden 2.

Para ello veamos que este polinomio pasa por los puntos de interpolación (x_1, y_1), (x_2, y_2) y (x_3, y_3) y por unicidad tendremos la conclusión.

En efecto

$$p_2(x_1) = f[x_1] = y_1$$

$$p_2(x_2) = f[x_1] + \frac{f[x_2] - f[x_1]}{x_2 - x_1}(x_2 - x_1) = f[x_2] = y_2$$

$$p_2(x_3) = f[x_1] + \frac{f[x_2] - f[x_1]}{x_2 - x_1}(x_3 - x_1) + \frac{f[x_2x_3] - f[x_1x_2]}{x_3 - x_1}(x_3 - x_1)(x_3 - x_2) =$$

$$f[x_1] + \frac{f[x_2] - f[x_1]}{x_2 - x_1}(x_3 - x_1) + \frac{f[x_3] - f[x_2]}{x_3 - x_2}(x_3 - x_2) - \frac{f[x_2] - f[x_1]}{x_2 - x_1}(x_3 - x_2) =$$

$$f[x_1] + \frac{f[x_2] - f[x_1]}{x_2 - x_1}(x_2 - x_1) + f[x_3] - f[x_2] = f[x_3] = y_3$$

Una vez demostrado que $p_2(x)$ tiene la forma citada, por un proceso de inducción se puede probar que en general el coeficiente c_{n-k+1} de grado k tiene la forma

$$c_{n-k+1} = f[x_1 x_2 \ldots x_{k+1}] = \frac{f[x_2 x_3 \ldots x_{k+1}] - f[x_1 x_2 \ldots x_k]}{x_{k+1} - x_1}$$

que es el llamado cociente de diferencias divididas de orden k, y se expresa en función de las diferencias divididas de orden k-1.

En la práctica, los cálculos se disponen en una tabla de diferencias divididas, colocando en la primera columna los valores de la función o diferencias divididas de orden 0, en la segunda columna las diferencias divididas de primer orden, en la tercera columna las de orden 2, y así sucesivamente.

La tabla queda de la forma siguiente:

$$\begin{array}{llll}
y_1 = f[x_1] & & & \\
y_2 = f[x_2] & f[x_1, x_2] & & \\
y_3 = f[x_3] & f[x_2, x_3] & f[x_1, x_2, x_3] & \\
y_4 = f[x_4] & f[x_3, x_4] & f[x_2, x_3, x_4] & f[x_1, x_2, x_3, x_4] \\
\ldots & \ldots & \ldots & \ldots
\end{array}$$

La construcción de la tabla de diferencias divididas es relativamente sencilla teniendo en cuenta el esquema anterior. En la diagonal de la tabla aparecen los coeficientes c_{n+1}, c_n, c_{n-1},..., c_1 de los polinomios de interpolación.

Ejemplo 7

Utiliza MATLAB para resolver el Ejemplo 6 mediante la tabla de diferencias divididas

Para el polinomio de grado 3 tomamos 4 puntos y los ordenamos según su proximidad a las 5h transcurridas que es donde se quiere realizar la estimación

```
% Horas  transcurridas
t = [2 4 6 8]';
% Piezas producidas
T = [62 116 114 8]';
% Orden de las componentes por proximidad a las 5h horas
transcurridas
I = [2 3 1 4];
% Valores ordenados
X = t(I); Y = T(I);
% Inicialización de la Tabla de Newton
n = 3; A = zeros(n+1,n+1);
% La primera columna son las ordenadas
A(:,1) = Y;
% La segunda son los cocientes de diferencias primeras
A(2:n+1,2)=(A(2:n+1,1)-A(1:n,1))./(X(2:n+1)-X(1:n));
% Luego, las diferencias segundas
A(3:n+1,3)=(A(3:n+1,2)-A(2:n,2))./(X(3:n+1)-X(1:n-1));
% Y así sucesivamente
A(4:n+1,4)=(A(4:n+1,3)-A(3:n,3))./(X(4:n+1)-X(1:n-2))
p=diag(A)
```

```
A =
   116     0     0     0
   114    -1     0     0
    62    13    -7     0
     8    -9   -11    -1
p =
   116
    -1
    -7
    -1
```

Tenemos así el polinomio de Newton que ya habíamos calculado en el Ejemplo 6, nos falta ahora evaluarlo en el punto requerido.

Evaluación del polinomio de Newton

Para evaluar el polinomio de interpolación en un punto dado $x = a$ tendremos en cuenta la expresión anidada del polinomio:

$$P_n(x) = c_{n+1} + c_n(x - x_1) + c_{n-1}(x - x_1)(x - x_2) + \cdots + c_1(x - x_1)(x - x_2)\cdots(x - x_n) =$$
$$c_{n+1} + (x - x_1)(c_n + (x - x_2)(c_{n-1} + \ldots + (x - x_{n-2})(c_3 + (x - x_{n-1})(c_2 + (x - x_n)c_1))\ldots)$$
$$= (\cdots((c_1(x - x_n) + c_2)(x - x_{n-1}) + c_3)(x - x_{n-2}) + \cdots + c_n)(x - x_1) + c_n +$$

Las operaciones se efectúan teniendo en cuenta la precedencia establecida mediante los paréntesis, o sea, comenzando con los más interiores.

Ejemplo 8

Evalúa el polinomio de interpolación de grado 3 que has obtenido en el ejemplo anterior para estimar el número de piezas producidas por los empleados al cabo de 5 horas trabajadas:

Expresemos el polinomio de grado 3 en su forma anidada.

$$p_3(x) = c_4 + c_3(x - x_1) + c_2(x - x_1)(x - x_2) + c_1(x - x_1)(x - x_2)(x - x_3) =$$
$$= c_4 + (x - x_1)(c_3 + (x - x_2)(c_2 + (x - x_3)c_1)) =$$
$$= ((c_1(x - x_3) + c_2)(x - x_2) + c_3)(x - x_1) + c_4$$

De esta forma es fácil calcular con MATLAB el valor numérico del polinomio paso a paso.

```
 a=5;
v=p(4);
v=v*(a-X(3))+p(3);
v=v*(a-X(2))+p(2);
v=v*(a-X(1))+p(1)

v =
   125
```

Valor que coincide con el obtenido en el Ejemplo 3, aunque allí se evaluaba en un vector de puntos, vamos a hacer los cambios necesarios para conseguir que este proceso sea válido también para el caso que se quiera evaluar en más de un punto, basta con darle carácter vectorial a las instrucciones, y tenemos así la posibilidad de representar el polinomio.

```
a=[3 5 7];
v=p(4);
v=v.*(a-X(3))+p(3);
v=v.*(a-X(2))+p(2);
v=v.*(a-X(1))+p(1)

v =
    93   125    77
```

Representemos el polinomio:

```
a=linspace (t(1),t(4));
v=p(4);
v=v.*(a-X(3))+p(3);
v=v.*(a-X(2))+p(2);
v=v.*(a-X(1))+p(1);
plot(t,T,'*',a,v)
```

Proponemos en el problema 4 la generalización de este razonamiento para obtener el valor numérico mediante una función de MATLAB.

Error de interpolación

Teorema: Si $p_n(x)$ interpola a la función *f(x)* en los puntos distintos $x_1,\ldots,x_{n+1}$ y f tiene derivadas continuas hasta orden n en un intervalo I que contiene a los $x_i's$. Entonces para cualquier x en I existe un η entre x y los $xi's$ tal que:

$$f(x)-P_n(x)=\frac{f^{(n+1)}(\eta)}{(n+1)!}(x-x_1)(x-x_2)\cdots(x-x_{n+1})$$

Una consecuencia práctica de la forma del error es que hemos de tomar puntos próximos al punto x en que hemos de evaluar el polinomio. Normalmente, comenzamos con un polinomio de grado bajo, por ejemplo, la recta que pase por los dos puntos más próximos a x, y vamos añadiendo puntos por orden de proximidad y calculando polinomios de mayor grado, hasta alcanzar la precisión deseada.
La derivada que aparece en la expresión anterior puede aproximarse a su vez por un cociente en diferencias, pues se tiene que

$$f\left[x_1,\ \cdots,\ x_n,\ x_{n+1}\right]=\frac{f^{(n+1)}(\eta)}{(n+1)!}$$

para cierto η en el menor intervalo que contiene a $x_1, x_2, \ldots, x_{n+1}$.
Esta expresión sugiere una regla práctica para decidir qué polinomio interpola mejor $n+1$ puntos. Si en la tabla de diferencias divididas, los valores de la columna k, por ejemplo, son aproximadamente iguales y los de la columna $k+1$ son aproximadamente cero, el polinomio interpolador más adecuado es de grado k. La razón es que el error viene dado por diferencias divididas de la columna siguiente, $k+1$, que suponemos casi nulas.
Los productos que aparecen en la fórmula del error nos indican que éste puede ser muy grande si hay muchos puntos o si x no está muy próximo a ellos. Cuando x no está en el menor intervalo determinado por $x_1, x_2, \ldots, x_{n+1}$, estamos extrapolando, en lugar de interpolando.

Si *f(x)* es una función en $[a,b]$ y los $x_i's$ están uniformemente distribuidos en $[a,b]$, i.e.,

$$x_i = a + (i-1)\,h, \qquad i=1,\ldots,n+1.$$

entonces se puede demostrar que:

$$\left|f(x)-p_n(x)\right| \le \left(\frac{h^{n+1}}{(n+1)!}\right)\max_{x\in[a,b]}\left|f^{(n+1)}(x)\right|$$

De modo que si

$$\max_{x\in[a,b]}\left|f^{(n)}(x)\right| \le M$$

donde *M* es una constante independiente de *n*, entonces el error de interpolación tiende a cero según *h* se va acercando a cero, i.e., hay convergencia. Este es el caso, por ejemplo para las funciones trigonométricas *sin (x)* y *cos(x)*, y para la exponencial *exp(x)* en un intervalo finito.

Ejemplo 9

Halla la cota del error al aproximar la función *sin(x)* con nodos equiespaciados en $[0,\pi\ ]$ por el polinomio interpolador de tercer grado.

Según la expresión del error para este caso:

$$\left|f(x)-p_n(x)\right| \le \left(\frac{h^{n+1}}{(n+1)!}\right)\max_{x\in[a,b]}\left|f^{(n+1)}(x)\right|$$

como la derivada n-ésima de la función está acotada por la unidad, podemos generalizar $\max_{x\in[a,b]}\left|f^{(n+1)}(x)\right| \le 1$ y decir que para cualquier valor de n el error está acotado por

$$\left|f(x)-p_n(x)\right| \le \left(\frac{h^{n+1}}{(n+1)!}\right)$$

en particular para $n=3$ y $h=\frac{\pi}{4}$ será: $E \le \frac{1}{24}\left(\frac{\pi}{4}\right)^4 = 0.0159$

Esta propiedad (derivadas de todo orden acotadas uniformemente) no la poseen todas las funciones. El ejemplo clásico de este mal comportamiento en las derivadas es la función de Runge como vemos a continuación:

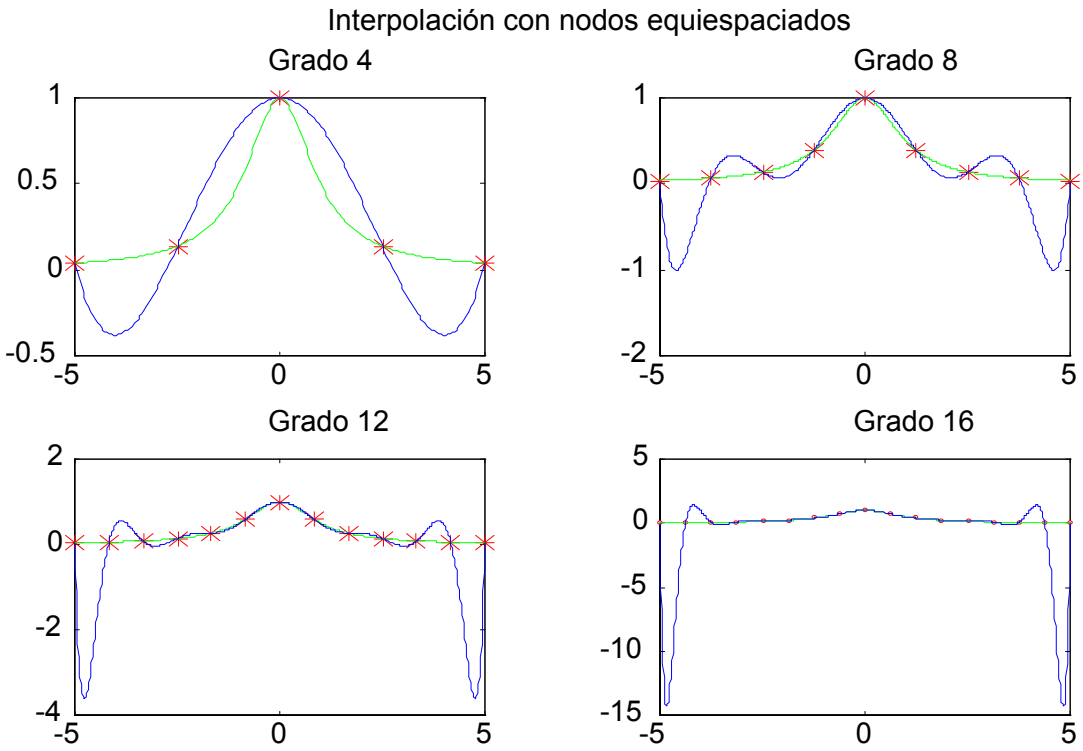
Nodos de Chebyshev

Cuando una función tiene características muy diferentes a los polinomios, la interpolación polinómica con nodos equiespaciados puede resultar inadecuada, el aumento del grado empeora el resultado en vez de mejorarlo.

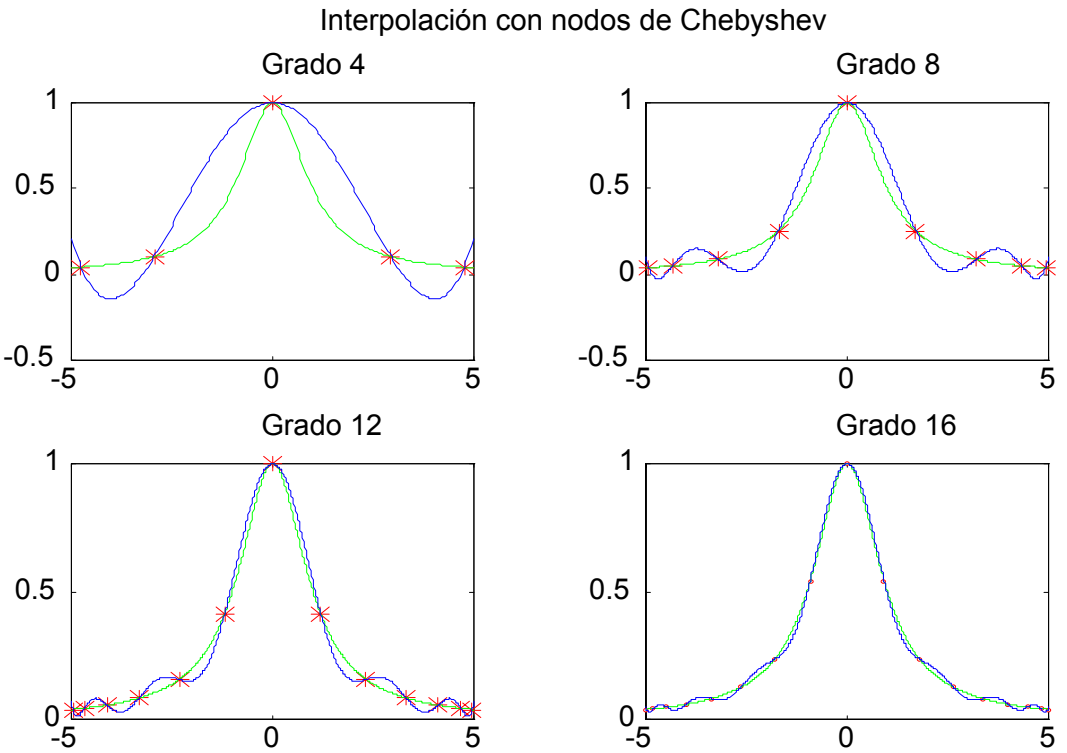
Consideremos la función de Runge en el intervalo [−5,5]:

$$y=\frac{1}{1+x^2}$$

Los polinomios que interpolan sus valores en puntos equiespaciados de este intervalo se desvían bastante de la función, sobre todo cerca de los extremos.



Observamos que el error máximo en el intervalo aumenta con el grado del polinomio interpolante. Para minimizar el error es conveniente tomar nodos de interpolación especiales, en lugar de los nodos equiespaciados considerados hasta ahora.



Los llamados nodos de Chebyshev hacen mínimo, en un intervalo dado, el valor máximo del polinomio $(x-x_1)(x-x_2)\cdots(x-x_{n+1})$ que aparece en la expresión del error. Para el caso particular del intervalo [−5,5], estos nodos son

$$x_i = 5\cos\left(\frac{2(n-(i-1))+1}{2n+2}\pi\right), \qquad i = 1,2,...,n+1.$$

En las gráficas se aprecia la reducción del error al interpolar la función de Runge en los nodos de Chebyshev.
Otra alternativa puede ser interpolar mediante funciones de otro tipo. En lugar de polinomios, podemos considerar funciones racionales, funciones definidas por intervalos, etc., como veremos en temas posteriores.

Problemas

1.- La solubilidad del cloruro amónico en el agua toma, a distintas temperaturas, los valores siguientes:

Temperatura	10	20	30	40	50	60
Solubilidad	33	37	42	46	51	55

Utiliza interpolación lineal para estimar la solubilidad del compuesto a una temperatura de 45º. Compara el resultado con el obtenido mediante interpolación cuadrática.

2.- Hace unos años, no tantos como imaginas, disponer de una calculadora no era nada fácil, para evaluar las funciones trigonométricas, por ejemplo *sin(x),* se disponía de tablas del siguiente tipo:

	0'	**10'**	**20'**	**30'**	**40'**	**50'**
45	7071	7092	7112	7133	7153	7173
46	7193	7214	7234	7254	7274	7294
47	7314	7333	7353	7373	7392	7412
48	7431	7451	7470	7490	7509	7528

Halla mediante interpolación lineal el seno de 47º 34', compara el resultado con el obtenido al utilizar todos los puntos para obtener el polinomio de interpolación y en ambos casos halla el error cometido utilizando el valor que proporciona MATLAB.

3.- Las diferentes contracciones de un resorte dependiendo de las cargas aplicadas vienen dadas en la siguiente tabla:

Carga (Kp)	5	10	15	20	25
Contracción (mm)	49	105	172	253	352

a) Escribe el polinomio de Lagrange de grado 2 para estimar la contracción producida por una carga de13 Kp.
b) Edita una función en MATLAB para evaluar el polinomio de interpolación de Lagrange en un punto dado y ejecútala para obtener el valor numérico del polinomio anterior en $x = 13$.

4.- La emisora de radio de la UPV ha determinado, por medio de encuestas, el porcentaje de ciudadanos que la sintonizan entre las 6 de la tarde y las 12 de la noche, lo mostramos en la siguiente tabla:

Horas	6	7	8	9	10	11	12
Oyentes (%)	0.2	0.25	0.1	0.12	0.15	0.3	0.11

Construye la tabla de diferencias divididas para estos valores y expresa de forma completa el polinomio de Newton que consideres más adecuado para estimar el porcentaje de oyentes a las 9:30 de la noche.

5.- Edita una función en MATLAB para que a partir de los puntos de interpolación construya la tabla de diferencias dividas y obtenga como parámetro de salida los coeficientes del polinomio de Newton, lo evalúe en una serie de puntos y lo represente junto a los puntos de interpolación dados.

6.-Utiliza la función de MATLAB que has programado en el problema 4 para obtener las gráficas de los polinomios de interpolación que aparecen en el apartado de teoría de **Nodos de Chebyshev** para aproximar a la función de Runge tanto con nodos equiespaciados como con nodos de Chebyshev.

7.-El número de personas afectadas por el virus contagioso que produce la gripe en una determinada población viene dado por la siguiente función, donde t indica el tiempo en días:

$$f(t) = \frac{1000}{2 + 999e^{-2.1t}}$$

a) Aproxima esta función en [0,7] por polinomios de interpolación de grados 4,6 y 8, tomando nodos equiespaciados. Representa la función junto al polinomio para comprobar que en los extremos del intervalo el error es considerable.
b) Calcula ahora los polinomios de interpolación tomando nodos de Chebyshev, para ello edita una función en MATLAB que vaya calculando los sucesivos polinomios de interpolación tomando nodos de Chebyshev hasta que el error global dado por la expresión:

$$E^2 = \frac{1}{101}\sum_{k=1}^{100} (f(t_k) - p(t_k))^2$$

sea menor que una décima.
Nota: Los nodos de Chebyshev se han dado en el intervalo [-5,5], hemos de obtenerlos ahora en [0,7], sabiendo que existe una correspondencia biyectiva entre los dos intervalos.

8.-Considera el polinomio $q(x) = (x - x_1)(x - x_2)\cdots(x - x_{n+1})$ para nodos igualmente equiespaciados una distancia h. Demuestra que

$$|q(x)| \le n!h^{n+1} \quad x_1 \le x \le x_{n+1}$$

Si $p_n(x)$ es el polinomio interpolador de $f(x) = e^x$ *en* [0,1] usa el resultado anterior para demostrar que:

$$\max_{0\le x\le 1} |e^x - p_n(x)| \to 0 \text{ cuando } n \to +\infty$$

9**.-.** Se desea aproximar la función *tg x* en el intervalo [–3/2,3/2].

a) Considera como nodos de interpolación los puntos $x_k = k \cdot \alpha$, *para* $k = 0, \pm 1, \pm 2, \pm 3$, precisamente en este orden. Construye la tabla de diferencias divididas y justificar el comportamiento de los coeficientes de interpolación.

b) Representa gráficamente la diferencia entre el polinomio de grado 5 y la función interpolada en [–3/2, 3/2], tomando 150 intervalos. ¿Cuál es el error máximo apreciado en la tabla de valores?

c) Halla un valor de α que minimice el error máximo. Explica el procedimiento seguido en su determinación.

d) Toma como nodos de interpolación los puntos $x_k = 3\,\alpha\, sen(k\pi/6)$, hallar el α óptimo y el error máximo. Compara con el error obtenido con nodos equiespaciados.

Soluciones

3.- a) Elegimos los tres nodos más cercanos a $x = 13$, que son (10, 105), (15,172) y (20, 253) el polinomio de Lagrange será de la forma:

$$P_2(x) = 172\frac{(x-10)(x-20)}{(15-10)(15-20)} + 105\frac{(x-15)(x-20)}{(10-15)(10-20)} + 253\frac{(x-15)(x-10)}{(20-15)(20-10)}$$

b) Para evaluarlo se requieren realizar numerosos cálculos y puede acumularse error de redondeo por lo que este tipo de polinomio no suele ser muy utilizado, no obstante como práctica de programación es interesante pensar el algoritmo que nos proporcionaría la evaluación de estos polinomios.

```
function v=lagrange(x,y,a)
n=length(x);%grado pol n-1
v=0;
for i=1:n
   xi=x(i);
   xn=[x(1:i-1) x(i+1:n)];
   l(i)=(prod(a-xn))./(prod(xi-xn));
   x=[x(1:i-1) xi x(i+1:n)];
   v=v+y(i)*l(i);
end

xn=[15 10 20];
yn=[172 105 253];
v=lagrange(xn,yn,13)

v =
  143.5200
```

4.- Editamos en primer lugar una función para crear la tabla de diferencias divididas y así obtener los coeficientes del polinomio de Newton de forma análoga a como lo hicimos en el Ejemplo 7, posteriormente en otra función procedemos a evaluar este polinomio utilizando su forma anidada como vimos en el Ejemplo 8, finalmente representamos el polinomio mediante una función que llama a estas dos.

```
function p =tabladd(x,y)
%parámetros de entrada:  x e y son los puntos de interpolación
%A es la matriz de diferencias divididas de Newton
%parámetro de salida: p son los coeficientes del polinomio de Newton

n=length(y);
A=zeros(n,n);
A(:,1)=y;
for  k=2:n
   A(k:n,k)=(A(k:n,k-1)- A(k-1:n-1,k-1))./ (x(k:n)-x(1:n-k+1));
end
p=diag(A);

function y=polynew(p,nodo,x)
%p es el vector de coeficientes del polinomio
%nodo son las abscisas de los puntos de interpolación
%evalúa el polinomio de Newton en el punto x

n=length(p)-1;
y=p(n+1);
for k=n:-1:1
        y=y.*(x-nodo(k))+p(k);
end

y=polynw(p,nodo,x)

function repnew(x,y)
p=tabladd(x,y);
xp=linspace(x(1),x(length(x));
yp=polynw(p,x,xp)
plot(x,y,'*',xp,yp)
```

5.-Editamos una función para evaluar la función de Runge.

```
function y=f(x)
y=1./(1+x.^2);
```

Mediante distintas llamadas a las funciones anteriores obtenemos los polinomios que aproximan a la función de Runge en el primer caso tomando nodos equiespaciados y en el segundo con nodos de Chebychef

```
xf=linspace(-5,5);
yf=feval('f',xf);
j=1;
for i=5:4:17
   x=linspace(-5,5,i);
   y=feval('f',x);
   subplot(2,2,j)
   j=j+1;
  p=tabladd(x',y');
   yp=polynew(p,x,xf);
   plot(x,y,'*',xf,yf,xf,yp)
end
```

Ahora con nodos de chebyshev

```
clear x
```

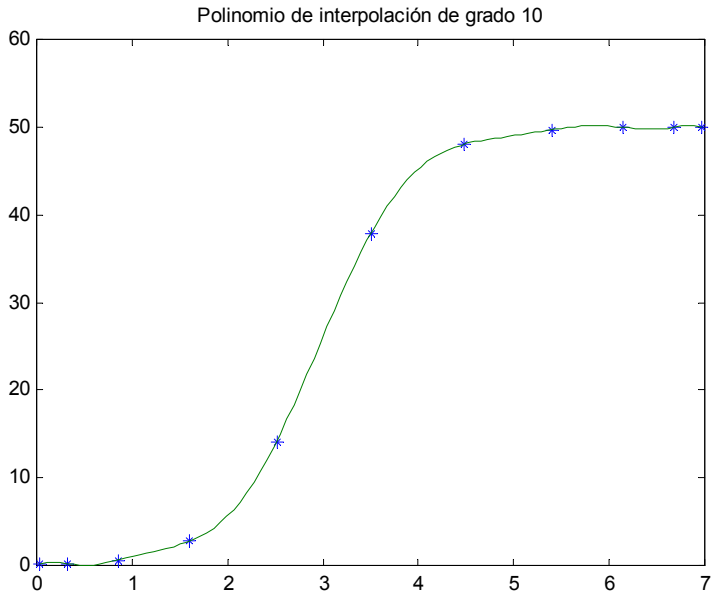
```
xf=linspace(-5,5);
yf=feval('f',xf);
j=1;
for i=4:4:16
   for k=1:i+1
      x(k)=5*cos(((2*(i-(k-1))+1)/(2*i+2))*pi);
   end
y=feval('f',x);
p=tabladd(x',y');
yp=polynew(p,x,xf);
subplot(2,2,j)
j=j+1;
plot(x,y,'*',xf,yf,xf,yp)
end
```

7.- b) Escribimos el código de la función:

```
function y=gr(t)
y=100./(2+999*exp(-2.1*t));
```

Interpolamos con nodos de Chebyshev:

```
clear x
xg=linspace(0,7);
yg=feval('gr',xg);
   E=1; n=1;
while E>0.1
   for i=1:n+1

     x(i)=cos(((2*(n-(i-1))+1)/(2*n+2))*pi);% nodos de Chebyshev en [-1,1]

   end
         xc=(x+1)/2*7;% nodos de Chebyshev en [0,7]
         yc=feval('gr',xc);
         p=interpol(xc',yc');
         yp=polyval(p,xg);
         E=sum((yg-yp).^2)/100;
         n=n+1;
end
E
E =
    0.0344
```



PRACTICA 10 - INTERPOLACION POLINOMICA

ejemplo del muelle:

```
>> x = [5 10 15]'; y = [395 620 795]';
>> A = [x.^2 x x.^0]
A =
    25     5     1
   100    10     1
   225    15     1
>> p = A\y
p =
   -1.0000
   60.0000
  120.0000
>>  %p(x) = -x.^2 + 60x + 120
>> xg = linspace(5,15);
>> yg = polyval(p,xg);
>> plot (x,y,'*',xg, yg)


>> %Desplazando el origen
>> clear
>> x = [5;10;15];
>> y = [395; 620; 795];
>> xd = x-10;
>> Ad = vander(xd);
>> cond(Ad)
ans =
   35.4120
>> pd = Ad\y;
>> xn = linspace(5,15,10);
>> vpd = polyval(pd, xn-10);
>> plot (x,y,'*',xn, vpd)
>> %la grafica resultante, es la misma
>> %pero se ha calculado con operaciones mas estables
```

METODO DE NEWTON

```
>> x = [4,6,2,8]'; y = [114 116 62 8]';
>> [p,v] = pnewton(x,y,2)
p =
  114.0000
    1.0000
   -6.2500
   -1.2500
v =
    62
```

Recuerda que Newton no devuelve los coeficientes del polinomio
Para evaluar el polinomio que devuelve newton se utiliza el parametro
z de la propia funcion

```
>> x = [5 10 15]'; y = [395 620 795]';
>> A = vander(x)
A =
    25     5     1
   100    10     1
   225    15     1
>> p = A\y
p =
   -1.0000
   60.0000
  120.0000
>>
>> polyval(p,1)
ans =
  179.0000


>>
>> [pn,v] = pnewton(x,y,1)
pn =
   395
    45
    -1
v =
   179
```

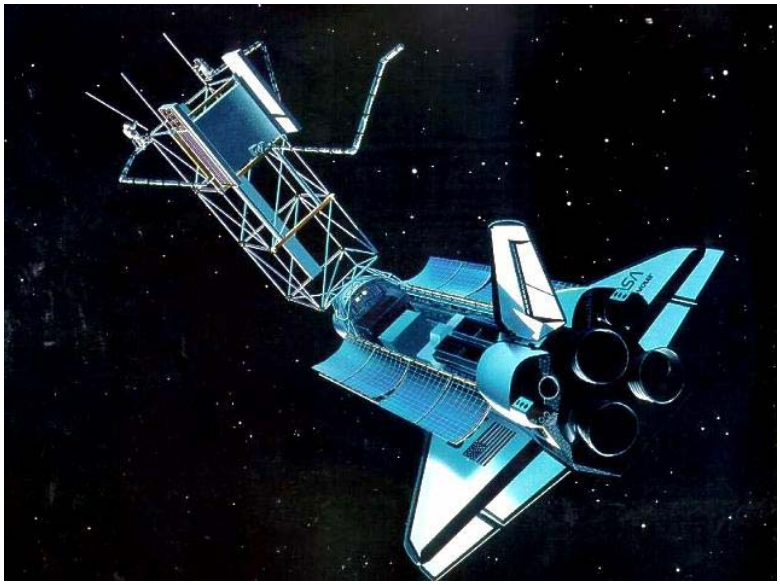
representacion gráfica

```
>> z = linspace(5,15);
>> [pn,v] = pnewton(x,y,z);
>> plot(z,v,x,y,'*')
```

ej de las transparencias

```
>> x = [2 4 6 8]'; y = [62 114 116 8]';
>> z = linspace(2,8);
>> [pn,v] = pnewton(x,y,z);
>> plot(z,v,x,y,'*')
```

11 Interpolación Polinómica Segmentaria



Podemos ver en la figura la puesta en órbita del primer elemento de la estación orbital internacional, que incluía el sistema de manipulación remota canadiense: Canadian Mobile Servicing Center. Los sistemas de manipulación remota como éste y otros muchos utilizan un avanzado sistema de control que guía un brazo manipulador hacia posiciones predeterminadas. Uno de los requisitos de dicho sistema de control es el diseño de una trayectoria para que el brazo se mueva de un sitio a otro de manera que se eviten movimientos bruscos que puedan causar que se desprendan los objetos transportados o incluso que se dañe el propio brazo robótico. Dicha trayectoria se diseña inicialmente en términos de un cierto número de puntos hacia los que debe moverse el brazo. Debe encontrarse una curva que pase por todos esos puntos y que además sea lo más suave posible, de manera que no se produzcan daños en el brazo.
La interpolación polinómica segmentaria nos permitirá definir una curva suficientemente suave (contínua, diferenciable) para este y otros propósitos. El caso lineal nos permitirá un ajuste contínuo en los puntos con pocas operaciones, pero sin "suavidad", es decir, sin derivabilidad en los nodos. Sin embargo, la interpolación polinómica segmentaria cúbica cumplirá nuestras espectativas definiendo una trayectoria suave que se ajuste a los puntos predefinidos y sin las grandes oscilaciones de los polinomios interpolantes de grado superior.

11.1Interpolación Polinómica Segmentaria Lineal

La interpolación polinómica ordinaria presenta inconvenientes cuando hay muchos nodos y la función no se parece mucho a un polinomio. En la práctica anterior hemos visto que al interpolar la función de Runge, el aumento del número de nodos (y consiguientemente, del grado del polinomio) en lugar de proporcionar más precisión, provoca mayores oscilaciones del interpolante e inestabilidad en el cálculo. Los polinomios no son adecuados para aproximar ciertos tipos de funciones, por ejemplo funciones racionales como la de Runge o funciones periódicas como el seno o el coseno.
La interpolación polinómica segmentaria evita estos inconvenientes limitando el grado del polinomio. En contrapartida, las condiciones de interpolación se satisfacen localmente, utilizando polinomios distintos en cada intervalo.

Dada una función f o un conjunto de pares de valores (x_k, y_k), $k = 1, 2, \ldots, n+1$, determinaremos una función s que pasa por los puntos dados y que en cada intervalo $[x_k, x_{k+1}]$ es un polinomio

$$s(x) = q_k(x), \quad x \in [x_k, x_{k+1}], \quad k = 1, 2, \ldots, n$$

La función interpolante s se denomina *spline*, aludiendo a unas reglas elásticas que se usan en delineación. Los splines son muy utilizados en diseño, porque se adaptan a los nodos prescritos sin oscilar demasiado en puntos intermedios. Los distintos trozos se ensamblan suavemente si se toman los polinomios de grado suficiente.

El polinomio de interpolación es muy sensible a pequeñas variaciones en los datos. En cambio, la obtención de los splines es numéricamente estable y puede hacerse utilizando matrices dispersas (véase el código de la función `spline` de MATLAB).

El caso más simple de interpolación polinómica segmentaria consiste en tomar, en cada subintervalo, el segmento de recta que une los valores de la función en los extremos. El resultado es una línea quebrada, poco agradable estéticamente, si la función oscila mucho.

Evitamos las esquinas considerando polinomios de mayor grado en cada subintervalo. Los splines cúbicos ofrecen un buen equilibrio entre dificultad de cálculo y apariencia suave del resultado, que es de grado 3 en cada subintervalo. Para determinar el spline cúbico, se necesitan un par de condiciones adicionales, que dan lugar a distintos tipos de spline. Analizaremos los llamados splines naturales, splines completos y splines no nodo.

La interpolación segmentaria lineal utiliza polinomios de primer grado para construir el spline. En cada subintervalo hemos de determinar los coeficientes a_k y b_k para que la recta de ecuación

$$q_k(x) = a_k + b_k(x - x_k), \quad k = 1, 2, \ldots, n$$

pase por los puntos (x_k , y_k) y (x_{k+1} , y_{k+1}).

La condición de interpolación en el extremo izquierdo de cada intervalo, $q_k(x_k) = y_k$ proporciona directamente a_k

$$a_k = y_k, \quad k = 1, 2, \ldots, n$$

Imponiendo la condición de interpolación en el extremo derecho, $q_k(x_{k+1}) = y_{k+1}$, obtenemos la fórmula para b_k

$$b_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}, \quad k = 1, 2, \ldots, n$$

Para aplicar estas fórmulas con MATLAB conviene considerar los coeficientes a_k como componentes de un vector *a*. Análogamente, los coeficientes b_k serán las componentes de un vector b.

Ejemplo 1

Obtengamos estos vectores con MATLAB para la función de Runge en el intervalo [-1,1]:

$$f(x) = \frac{1}{1 + 25x^2}.$$

```
n = 10; h = 2/n;% Tomamos 10 subintervalos
x = [-1:h:1];% Abscisas
y = 1./(1 + 25*x.^2);% Ordenadas de Runge
a = y(1:n)% y tiene n+1 componentes
b = diff(y)./diff(x)

a =
```

```
  Columns 1 through 7
    0.0385    0.0588    0.1000    0.2000    0.5000    1.0000    0.5000
  Columns 8 through 10
    0.2000    0.1000    0.0588
b =
  Columns 1 through 7
    0.1018    0.2059    0.5000    1.5000    2.5000   -2.5000   -1.5000
  Columns 8 through 10
   -0.5000   -0.2059   -0.1018
```

Una vez obtenidos los coeficientes del spline, su evaluación en un punto x requiere un poco de trabajo: determinar a qué intervalo nodal $[x_k, x_{k+1}]$ pertenece el punto x, elegir los coeficientes a_k, b_k del polinomio correspondiente y ya, por fin, evaluar $q_k(x)$ según su fórmula. Afortunadamente, MATLAB automatiza estas tareas mediante las órdenes que veremos a continuación.

11.2 Funciones de MATLAB para Splines

MATLAB dispone de varias funciones específicas para calcular splines y evaluarlos. Si hemos calculado ya los coeficientes, lo primero que hacemos es unirlos en una matriz.

```
s = [b' a']% De mayor a menor grado
```

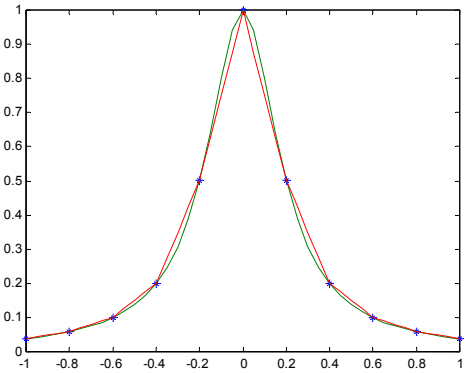
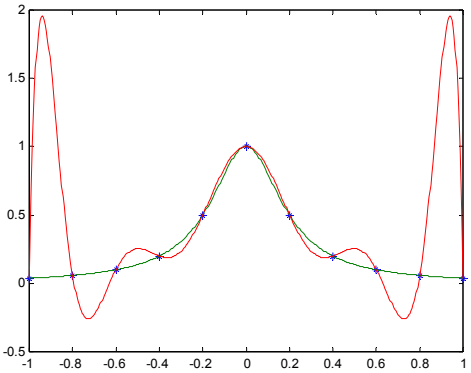
```
s =
    0.1018    0.0385
    0.2059    0.0588
    0.5000    0.1000
    1.5000    0.2000
    2.5000    0.5000
   -2.5000    1.0000
   -1.5000    0.5000
   -0.5000    0.2000
   -0.2059    0.1000
   -0.1018    0.0588
```

Para evaluar el spline, además de los coeficientes, se necesitan los nodos. Con ambos datos, la función `mkpp` (make piecewise polynomial) crea un vector con toda la información relativa al spline.

```
ps = mkpp(x,s)
```

La evaluación del spline en esta forma recuerda a la de un polinomio ordinario, usando `ppval` en lugar de `polyval`. Representemos gráficamente la aproximación segmentaria lineal de la función de Runge.

```
xg = (-1:.005:1)';
yg = 1./(1+25*xg.^2);
ys = ppval(ps,xg);
plot(x,y,'*',xg,yg,xg,ys)
```

Comparando el spline lineal con el polinomio de grado 10 que interpola los mismos nodos, observamos que este último presenta más oscilaciones, sobre todo en los extremos del intervalo, mientras que el spline se ajusta mejor, pero no es derivable en los nodos. Empleando polinomios de mayor grado se obtienen splines que sí lo son. Consideraremos el caso de grado 3.

11.3 Interpolación Polinómica Segmentaria Cúbica

El spline cúbico elimina dos problemas de la interpolación lineal: su lentitud de convergencia y el que presenta ángulos en los nodos, lo cual es inadecuado en ciertas aplicaciones. Un spline cúbico es una función polinómica segmentaria de grado 3 con derivada primera y segunda continuas.

Dados n + 1 puntos $(x_k\ ,\ y_k)$, $k = 1,\ 2,\ \ldots,\ n+1$, con $x_1 < x_2 < \ldots < x_{n+1}$, hallaremos splines cúbicos

$$s(x) = q_k(x), \qquad x \in [x_k, x_{k+1}], \quad k = 1, 2, \ldots, n$$

que verifiquen las siguientes condiciones de interpolación

$$q_k(x_k) = y_k, \qquad k = 1, 2, \ldots, n$$
$$q_k(x_{k+1}) = y_{k+1}, \quad k = 1, 2, \ldots, n$$

y también las condiciones de conexión

$$q_k'(x_{k+1}) = q_{k+1}'(x_{k+1}), \quad k = 1, 2, \ldots, n-1$$
$$q_k''(x_{k+1}) = q_{k+1}''(x_{k+1}), \quad k = 1, 2, \ldots, n-1$$

Las primeras condiciones establecen que s es continua e interpola los puntos $(x_k\ ,\ y_k)$. Las segundas implican que s tiene derivadas primera y segunda continuas.

Para determinar el spline, podemos escribir cada polinomio componente de forma adecuada, por ejemplo,

$$q_k(x) = a_k + b_k(x - x_k) + c_k(x - x_k)^2 + d_k(x - x_k)^3, \qquad k = 1, 2, \ldots, n$$

y sustituir esta expresión en las condiciones anteriores. Así obtenemos un sistema de ecuaciones lineales que deben verificar los coeficientes.

Este procedimiento tiene dos desventajas. Primero, produce un sistema con ancho de banda innecesariamente grande y cuyos coeficientes pueden oscilar en varios ordenes de magnitud. En segundo lugar, el sistema tiene 4n incógnitas, mientras que las condiciones anteriores sólo proporcionan 4n–2 ecuaciones. Obviamente, esta indeterminación es intrínseca al problema, pero el ataque directo da poca información sobre cómo resolverlo.

Por ello, procederemos de otro modo: obtendremos los coeficientes del spline en función de la derivada segunda del spline en los nodos, empleando las condiciones de interpolación. A continuación, las condiciones de conexión nos permitirán plantear un

sistema lineal de ecuaciones, cuya resolución proporcionará el valor de la derivada segunda en los nodos y, por tanto, el valor de los coeficientes del spline.

11.3.1 Determinación de las derivadas segundas del spline

Supongamos por un momento que conocemos los valores r_k de las derivadas segundas del spline en los nodos

$$s''(x_k) = r_k, \qquad k = 1, 2, ..., n+1$$

Entonces, para *k = 1, 2, ..., n*

$$q_k''(x_k) = r_k$$

$$q_k''(x_{k+1}) = r_{k+1}$$

Como la derivada segunda de q_k es de primer grado, su ecuación es la de la recta que pasa por los puntos (x_k, r_k) y (x_{k+1}, r_{k+1}):

$$q_k''(x) = r_k + \frac{r_{k+1} - r_k}{h_k}(x - x_k), \quad k = 1, 2, ..., n$$

donde $h_k = x_{k+1} - x_k$.

Integrando dos veces obtenemos

$$q_k(x) = a_k + b_k(x - x_k) + \frac{r_k}{2}(x - x_k)^2 + \frac{r_{k+1} - r_k}{6h_k}(x - x_k)^3, \quad k = 1, 2, ..., n$$

donde a_k y b_k son constantes a determinar. Esta expresión proporciona directamente los coeficientes c_k y d_k del polinomio q_k en función de las derivadas segundas r_k.

$$c_k = \frac{r_k}{2}$$

$$d_k = \frac{r_{k+1} - r_k}{6h_k}$$

Veamos cómo obtener los otros dos coeficientes.

La primera condición de interpolación proporciona directamente el valor de a_k.

$$q_k(x_k) = a_k = y_k$$

La condición de interpolación en el extremo derecho proporciona

$$q_k(x_{k+1}) = a_k + b_k h_k + \frac{r_k}{2}h_k^2 + \frac{r_{k+1} - r_k}{6}h_k^2 = y_{k+1}$$

De las dos expresiones precedentes obtenemos el valor de b_k:

$$b_k = \frac{y_{k+1} - y_k}{h_k} - \left(\frac{r_{k+1}}{6} + \frac{r_k}{3}\right)h_k = e_k - \left(\frac{r_{k+1}}{6} + \frac{r_k}{3}\right)h_k$$

Una vez expresados todos los coeficientes en función de las r_k, volvamos al cálculo de estos valores. Ya hemos utilizado las condiciones de interpolación. La ecuación de partida presupone la continuidad de la derivada segunda. Sólo falta imponer, *para k = 1, 2, ..., n–1*, la coincidencia de las derivadas primeras en los nodos interiores:

$$q_k'(x_{k+1}) = b_k + r_k h_k + \frac{r_{k+1} - r_k}{2}h_k$$

$$q_{k+1}'(x_{k+1}) = b_{k+1}$$

Igualando tenemos

$$\frac{r_k + r_{k+1}}{2}h_k = b_{k+1} - b_k$$

Sustituyendo las expresiones de los b_k y agrupando los coeficientes de r_k queda

$$\frac{h_k}{6} r_k + \frac{h_k + h_{k+1}}{3} r_{k+1} + \frac{h_{k+1}}{6} r_{k+2} = e_{k+1} - e_k, \quad k = 1, 2, ..., n-1$$

que es un sistema lineal de n–1 ecuaciones con n+1 incógnitas, los r_k. Veamos la estructura de la matriz del sistema y los términos independientes.

Las ecuaciones obtenidas se expresan matricialmente como

$$M\, r = g$$

donde r es la columna de incógnitas, la matriz del sistema es

$$M = \begin{bmatrix} \frac{h_1}{6} & \frac{h_1 + h_2}{3} & \frac{h_2}{6} & & & & & \\ & \frac{h_2}{6} & \frac{h_2 + h_3}{3} & \frac{h_3}{6} & & & & \\ & & \frac{h_3}{6} & \frac{h_3 + h_4}{3} & \frac{h_4}{6} & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \frac{h_{n-2}}{6} & \frac{h_{n-2} + h_{n-1}}{3} & \frac{h_{n-1}}{6} & \\ & & & & & \frac{h_{n-1}}{6} & \frac{h_{n-1} + h_n}{3} & \frac{h_n}{6} \end{bmatrix}$$

y el vector de términos independientes

$$g = \begin{bmatrix} e_2 - e_1 \\ e_3 - e_2 \\ e_{4-} - e_3 \\ \vdots \\ e_{n-1} - e_{n-2} \\ e_n - e_{n-1} \end{bmatrix}$$

El sistema tiene dos ecuaciones menos que incógnitas, por lo que hacen falta dos condiciones adicionales para que tenga solución única. La elección de estas condiciones puede hacerse de varias formas, dando lugar a distintos tipos de spline.

11.4 Splines naturales

Si en el anterior sistema de ecuaciones lineales eliminamos dos de las incógnitas, quedarán (n-2) ecuaciones y (n-2) valores de la derivada segunda por averiguar. Así, podemos definir un spline fijando de antemano el valor de la derivada segunda en el primer nodo y en el último.

11.4.1 Derivadas segundas del spline natural

Si las derivadas segundas en los extremos son conocidas, pasamos r_1 y r_{n+1} al segundo miembro de las primera y última ecuaciones, que quedarán como

$$\frac{h_1 + h_2}{3} r_2 + \frac{h_2}{6} r_3 = e_2 - e_1 - \frac{h_1}{6} r_1$$

$$\frac{h_{n-1}}{6} r_{n-1} + \frac{h_{n-1} + h_n}{3} r_n = e_n - e_{n-1} - \frac{h_n}{6} r_{n+1}$$

El sistema resultante es tridiagonal simétrico y se resuelve por eliminación gaussiana con O(n) operaciones sin necesidad de pivotación parcial.

$$\begin{bmatrix} \frac{h_1+h_2}{3} & \frac{h_2}{6} & & & & \\ \frac{h_2}{6} & \frac{h_2+h_3}{3} & \frac{h_3}{6} & & & \\ & \frac{h_3}{6} & \frac{h_3+h_4}{3} & \frac{h_4}{6} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ & & & & \frac{h_{n-1}}{6} & \frac{h_{n-1}+h_n}{3} \end{bmatrix} \begin{bmatrix} r_2 \\ r_3 \\ r_4 \\ \vdots \\ r_{n-1} \\ r_n \end{bmatrix} = \begin{bmatrix} e_2 - e_1 - \frac{h_1}{6} r_1 \\ e_3 - e_2 \\ e_4 - e_3 \\ \vdots \\ e_{n-1} - e_{n-2} \\ e_n - e_{n-1} - \frac{h_n}{6} r_{n+1} \end{bmatrix}$$

Calculados los restantes r_k, los coeficientes del spline se hallan aplicando las relaciones obtenidas anteriormente.

Cuando las derivadas en los extremos son nulas el *spline* se llama *natural*. A pesar de su nombre, este spline no es especialmente recomendable, pues en general, la función a interpolar no tendrá derivada segunda nula en los extremos, e imponer esta condición empeorará la aproximación, al menos cerca de estos.

Ejemplo 2

Halla el spline cúbico que interpola la función $y = \sin(x) + \cos\left(\frac{x}{2}\right)$, tomando 7 nodos equiespaciados en el intervalo $[-2\pi, 2\pi]$, sabiendo que el valor de la derivada segunda en ambos extremos del intervalo es 0.25.

Determinemos en primer lugar los nodos y valores a interpolar

```
n = 6;% Número de subintervalos
x = linspace(-2*pi,2*pi,n+1)% Nodos equiespaciados
y = sin(x)+cos(x/2)% Valores a interpolar
```

```
x =
   -6.2832   -4.1888   -2.0944         0    2.0944    4.1888    6.2832
y =
   -1.0000    0.3660   -0.3660    1.0000    1.3660   -1.3660   -1.0000
```

Los valores de la derivada segunda en los extremos son las componentes primera y última del vector *r*.

```
r1 = 0.25; rn1 = 0.25;
```

Obtendremos las restantes componentes de r resolviendo el sistema lineal que satisfacen las derivadas segundas.

Al pasar al segundo miembro los términos correspondientes a r_1 y r_{n+1} queda un sistema cuadrado de orden n–1 con matriz tridiagonal simétrica.

Construimos su matriz por diagonales, operando adecuadamente con los elementos del vector h.

```
h = diff(x);% Vector de pasos
ds = h(2:n-1)/6;% Diagonales secundarias
dl = h(1:n-1)/3;
dr = h(2:n)/3;
```

```
dp = dl+dr;% Diagonal principal
M = diag(dp) + diag(ds,-1) + diag(ds,1)

M =
    1.3963    0.3491         0         0         0
    0.3491    1.3963    0.3491         0         0
         0    0.3491    1.3963    0.3491         0
         0         0    0.3491    1.3963    0.3491
         0         0         0    0.3491    1.3963
```

Los términos independientes se obtienen fácilmente con el comando `diff`

```
e = diff(y)./h
g = diff(e)

e =
    0.6522   -0.3495    0.6522    0.1748   -1.3045    0.1748
g =
   -1.0018    1.0018   -0.4775   -1.4792    1.4792
```

Si las derivadas segundas no son nulas, hay que ajustar el primer y el último elementos de g:

```
g(1) = g(1) - h(1)*r1/6
g(n-1) = g(n-1) - h(n)*rn1/6

g =
   -1.0890    1.0018   -0.4775   -1.4792    1.4792
g =
   -1.0890    1.0018   -0.4775   -1.4792    1.3920
```

Ya podemos resolver el sistema lineal para obtener los valores desconocidos de las segundas derivadas, r_k.

```
r = M\g'

r =
   -1.0415
    1.0460
   -0.2727
   -1.3232
    1.3277
```

```
r = [r1;r;rn1]

r =
    0.2500
   -1.0415
    1.0460
   -0.2727
   -1.3232
```

```
    1.3277
    0.2500
```

11.4.2 Coeficientes de los polinomios $q_k(x)$, $k = 1, 2, ..., n$

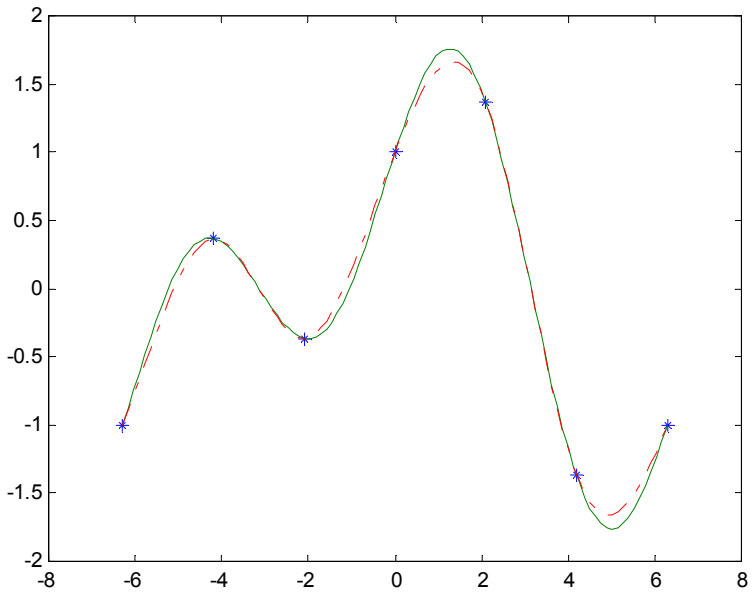
Una vez obtenidas las derivadas segundas, aplicamos las fórmulas de los coeficientes de los polinomios interpolantes.

```
rl = r(1:n);rr = r(2:n+1);
a = y(1:n)';
b = e' -(rr + 2*rl).*h'/6;
c = rl/2;
d = diff(r)./h'/6;
```

Ya sólo queda componer la matriz del spline y representarlo gráficamente, como en el caso lineal:

```
s = [d, c, b, a] % Coeficientes de los qk
ps = mkpp(x,s); % Codificación del spline
xg = linspace(-2*pi,2*pi); % Abscisas para gráfico
ys = ppval(ps,xg); % Evalúa el spline
yg = sin(xg)+cos(xg/2); % Evalúa la función
plot(x,y,'*',xg,yg,xg,ys,'-.')% Gráfico
```

```
s =
   -0.1028    0.1250    0.8412   -1.0000
    0.1661   -0.5207    0.0124    0.3660
   -0.1049    0.5230    0.0172   -0.3660
   -0.0836   -0.1363    0.8270    1.0000
    0.2110   -0.6616   -0.8442    1.3660
   -0.0858    0.6639   -0.8394   -1.3660
```



La gráfica muestra la excelente aproximación de la función mediante el spline calculado conociendo las derivadas segundas en los extremos.

11.5 Splines completos

Si tenemos información de la derivada de la función en los extremos del intervalo, podemos obtener un spline cuya derivada tome estos valores.

11.5.1 Derivadas segundas del spline completo

Sean y'_1 e y'_n estas derivadas. Entonces se ha de cumplir

$$q'_1(x_1) = y'_1 \qquad q'_n(x_{n+1}) = y'_{n+1}$$

Imponemos estas condiciones y expresamos los b_k en función de los r_k:

$$y'_1 = b_1 = e_1 - \left(\frac{r_2}{6} + \frac{r_1}{3}\right)h_1$$

$$y'_{n+1} = b_n + \left(\frac{r_n + r_{n+1}}{2}\right)h_n = e_n - \left(\frac{r_{n+1}}{6} + \frac{r_n}{3}\right)h_n + \left(\frac{r_n + r_{n+1}}{2}\right)h_n$$

Ordenando quedan dos ecuaciones en los coeficientes r_k

$$\frac{h_1}{3}r_1 + \frac{h_1}{6}r_2 = e_1 - y'_1$$

$$\frac{h_n}{6}r_n + \frac{h_n}{3}r_{n+1} = y'_{n+1} - e_n$$

que añadimos al sistema lineal para tener n+1 ecuaciones con *n + 1* incógnitas.

$$\begin{bmatrix} \frac{h_1}{3} & \frac{h_1}{6} & & & & & \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & & & & \\ & \frac{h_2}{6} & \frac{h_2+h_3}{3} & \frac{h_3}{6} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \frac{h_{n-1}}{6} & \frac{h_{n-1}+h_n}{3} & \frac{h_n}{6} & \\ & & & & \frac{h_n}{6} & \frac{h_n}{3} & \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_n \\ r_{n+1} \end{bmatrix} = \begin{bmatrix} e_1 - y'_0 \\ e_2 - e_1 \\ e_3 - e_2 \\ \vdots \\ e_n - e_{n-1} \\ y'_n - e_n \end{bmatrix}$$

El spline así obtenido se denomina *spline completo*.

Ejemplo 3

Hallaremos a continuación el spline completo de la función $y = \sin(x) + \cos\left(\frac{x}{2}\right)$, tomando 7 nodos equiespaciados en el intervalo $[-2\pi, 2\pi]$, sabiendo que el valor de la derivada primera en ambos extremos del intervalo es 1.

La matriz del sistema para el caso de condiciones sobre la derivada primera es fácil de construir:

```
ds = h/6;                    % Diagonal secundaria
```

```
dp = ([h';0]+[0;h'])/3;     % Diagonal principal
M = diag(dp) + diag(ds,-1) + diag(ds,1)

M =
    0.6981    0.3491         0         0         0         0         0
    0.3491    1.3963    0.3491         0         0         0         0
         0    0.3491    1.3963    0.3491         0         0         0
         0         0    0.3491    1.3963    0.3491         0         0
         0         0         0    0.3491    1.3963    0.3491         0
         0         0         0         0    0.3491    1.3963    0.3491
         0         0         0         0         0    0.3491    0.6981
```

De nuevo la matriz del sistema es tridiagonal simétrica y estrictamente diagonalmente dominante.
El término independiente se expresa sencillamente como:

```
e = diff(y')./h';
dy0=1;
dyn=1;
g = diff([dy0; e; dyn])

g =
   -0.3478
   -1.0018
    1.0018
   -0.4775
   -1.4792
    1.4792
    0.8252
```

siendo dy0 y dyn las derivadas en los nodos extremos.
El sistema lineal proporciona ahora los valores r_k de la derivada segunda en todos los nodos.

```
r = M\g

r =
   -0.0124
   -0.9715
    1.0287
   -0.2736
   -1.3023
    1.2451
    0.5595
```

Sustituyendo en las expresiones adecuadas calculamos los coeficientes del spline completo, a_k, b_k, c_k y d_k, exactamente igual que en el caso del spline natural.

```
rl = r(1:n); rr = r(2:n+1);
a = y(1:n)';
b = e - h'.*(2*rl + rr)/6;
c = rl/2;
d = diff(r)./h'/6;
```

Formamos la matriz de coeficientes del spline:

```
s = [d c b a]
s =
   -0.0763   -0.0062    1.0000   -1.0000
    0.1592   -0.4858   -0.0304    0.3660
   -0.1036    0.5144    0.0295   -0.3660
```

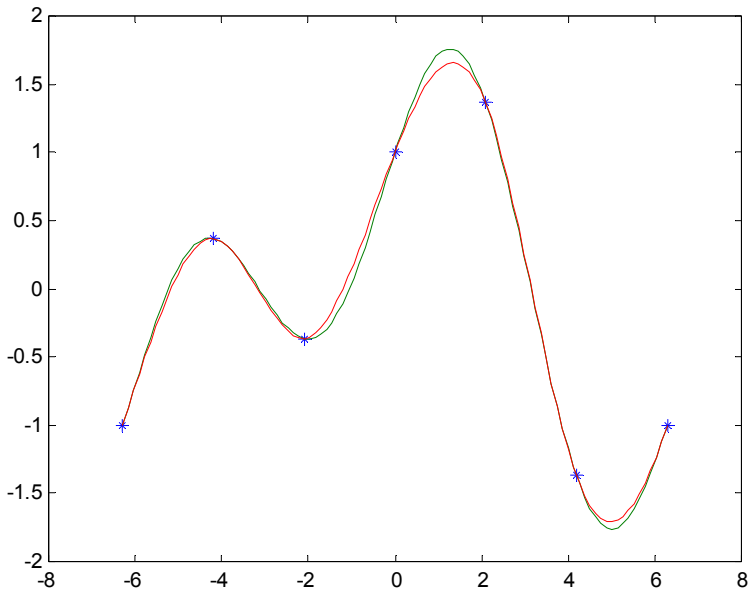
```
   -0.0819   -0.1368    0.8203    1.0000
    0.2027   -0.6512   -0.8299    1.3660
   -0.0546    0.6226   -0.8898   -1.3660
```

Una vez obtenidos los coeficientes, los pasamos al modo de polinomio segmentario:

```
ps = mkpp(x,s)
```

Evaluamos el spline y lo representamos:

```
ys = ppval(ps,xg);
plot(x,y,'*',xg,yg,xg,ys)
```



11.6 Splines no nodo

Si no disponemos de información sobre el comportamiento de la función en los nodos extremos, en lugar de asignar arbitrariamente las derivadas primeras o segundas, una alternativa mejor es suponer que los dos primeros polinomios, q_1 y q_2, coinciden y los dos últimos, q_{n-1} y q_n, también. Así obtenemos dos condiciones adicionales que hacen determinado el sistema lineal.

Como q_1 y q_2 son de tercer grado y coinciden en x_2, lo mismo que sus derivadas primera y segunda, sólo falta imponer que las derivadas terceras sean iguales:

$$q_1'''(x_2) = q'''(x_2)$$

La derivada segunda de q_k es de primer grado, por lo que su derivada es el cociente incremental

$$q_k''' = \frac{q_k''(x_{k+1}) - q_k''(x_k)}{x_{k+1} - x_k} = \frac{r_{k+1} - r_k}{h_k}$$

Tomando k = 1, 2 y sustituyendo en la igualdad anterior obtenemos una ecuación lineal

$$-h_2 r_1 + (h_1 + h_2) r_2 - h_1 r_3 = 0$$

que añadirmos al principio del sistema.

Análogamente, igualando las derivadas terceras en x_n obtenemos otra ecuación lineal

$$-h_n r_{n-1} + (h_{n-1} + h_n) r_n - h_{n-1} r_{n+1} = 0$$

que añadimos al final del sistema.

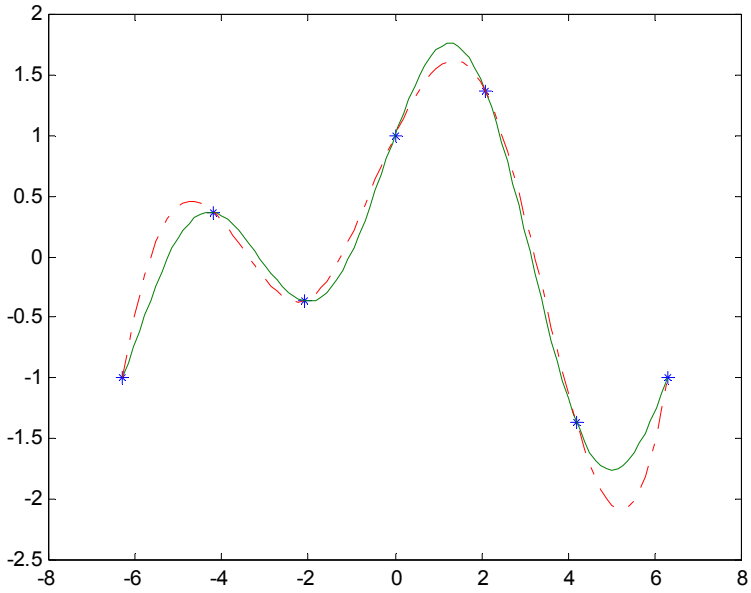
Estas condiciones determinan un spline, llamado *spline no nodo* porque el polinomio interpolador no cambia al pasar por x_2 y por x_n.

11.7 Splines en MATLAB

Recogiendo todas las operaciones efectuadas en los ejemplos de esta Práctica, no es difícil editar ficheros.m para obtener splines con condiciones naturales o condiciones sobre la derivada en los extremos.

La función `spline` incorporada en MATLAB obtiene el spline no nodo. Calcula internamente los coeficientes de los polinomios y da el resultado ya en forma de polinomio segmentario, que se evalúa con `ppval` para su representación gráfica.

```
ps = spline(x,y);
ys = ppval(ps,xg);
plot(x,y,'*',xg,yg,xg,ys,'-.')
```



Los coeficientes de los polinomios se obtienen con la función `unmkpp` (unmake piecewise polynomial), que separa la información en un vector de nodos y una matriz de coeficientes. El proceso inverso, obtener el polinomio segmentario a partir de los nodos y los coeficientes del spline, se realiza con la función `mkpp`, ya utilizada previamente en varias ocasiones.

```
[nodos, coefs] = unmkpp(ps)
```

Si damos tres argumentos de entrada a `spline`, el tercero es tomado como abscisas en las que se evalúa el spline, y el programa devuelve las ordenadas correspondientes, en lugar de los coeficientes del polinomio segmentario.

```
ys = spline(x,y,xg)
```

MATLAB dispone asimismo de una función genérica para la interpolación segmentaria de funciones de una variable, interp1, que engloba la interpolación lineal y los splines cúbicos. Poniendo

```
yl = interp1(x,y,xg)
```

se obtienen las ordenadas correspondientes a xg que interpolan linealmente (x,y). Añadiendo como cuarto argumento la cadena `'spline'`, da el mismo resultado que la orden `spline`.

```
ys = interp1(x,y,xg,'spline')
```

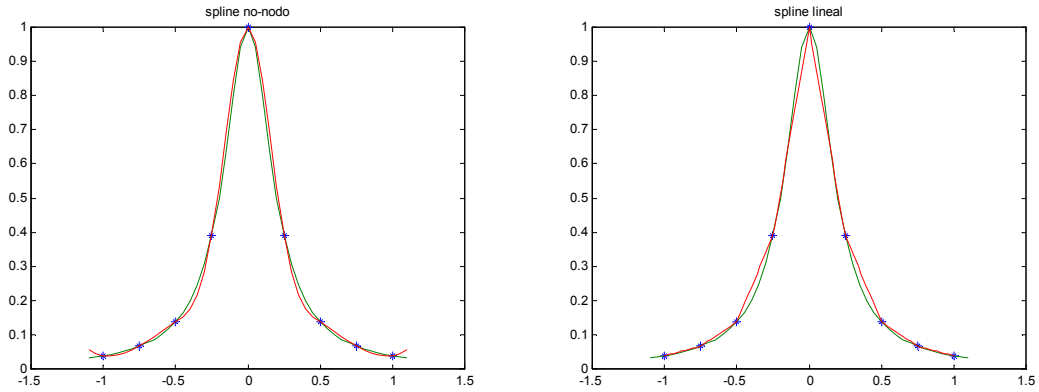
Ejemplo 4

Las gráficas muestran distintas aproximaciones de la función de Runge

$$y = \frac{1}{1+25x^2}$$

obtenidas interpolando sus valores en 9 nodos equiespaciados en [–1,1].
La primera se obtiene con la función spline de MATLAB.

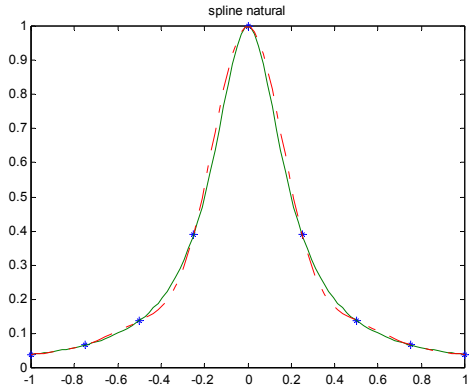
```
x = -1:.5:1;           % Nodos
y = 1./(1+25*x.^2);    % Valores
xg = -1.1:0.05:1.1;    % Abscisas para el gráfico
yg = 1./(1+25*xg.^2); % Ordenadas
ys = spline(x,y,xg);
plot(x,y,'*',xg,yg,xg,ys)
```

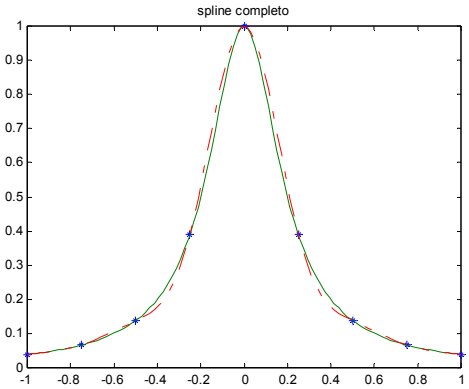


La interpolación segmentaria lineal de esta función, a la que corresponde la segunda gráfica, se ejecuta con las siguientes instrucciones:

```
xl = -1:0.02:1;
yl = interp1(x,y,xl);
plot(x,y,'*',xg,yg,xl,yl)
```

Los dos últimos gráficos muestran los splines obtenidos con condiciones naturales (considerando nulas las derivadas segundas en los extremos) y con derivadas primeras dadas en los extremos (aproximadamente 0.08 y - 0.08, para la función de Runge).





El peor ajuste del primer gráfico en los extremos se debe a que, en el `spline` de MATLAB, los dos primeros polinomios coinciden, al igual que los dos últimos, y en este ejemplo sólo hay ocho intervalos.

Ejemplo 5

Consideremos un brazo robótico como el que nos ha servido para introducir el tema. Tenemos almacenados en una tabla algunos puntos de la trayectoria del brazo; además, dichos puntos están en el orden necesario para que el brazo se mueva hasta una posición para coger un objeto y luego vuelva a la posición original. Supondremos también que los puntos intermedios incluidos en la trayectoria tratan de evitar choques del brazo o bien guiarlo hacia sensores que recogen información. Así, cada punto tendrá tres coordenadas: las coordenadas *x* e *y* relativas a la posición del brazo y una tercera componente, que denotaremos por *acción*, codificada del siguiente modo:

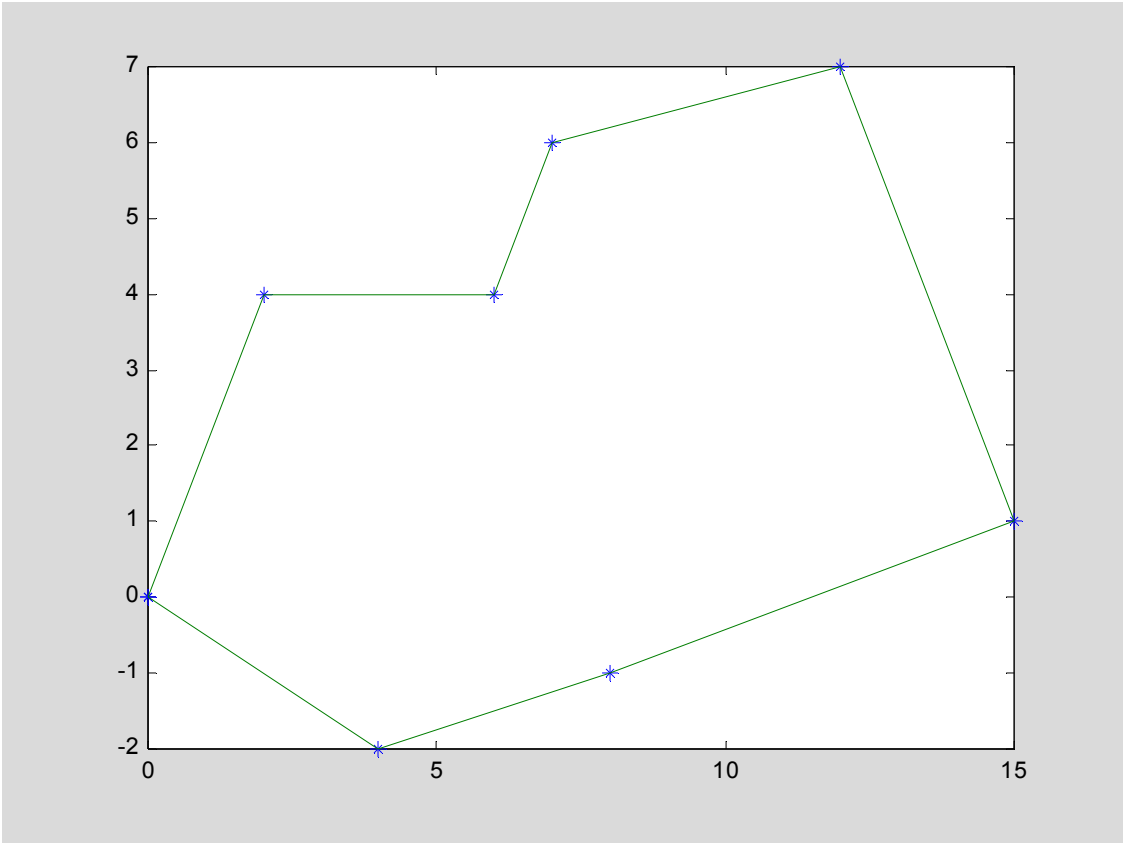
Acción	Significado
0	posición de partida
1	posición intermedia
2	posición de coger objeto
3	posición de soltar objeto

El problema consiste en diseñar una curva "suave" mediante interpolación cúbica segmentaria de manera que sirva de guía para que el brazo robótico, partiendo de un punto determinado, se mueva a una posición concreta para coger un objeto, lo deje en otra posición y vuelva al punto de partida. Estas posiciones, junto con otras intermedias, se ven reflejadas en la siguiente tabla:

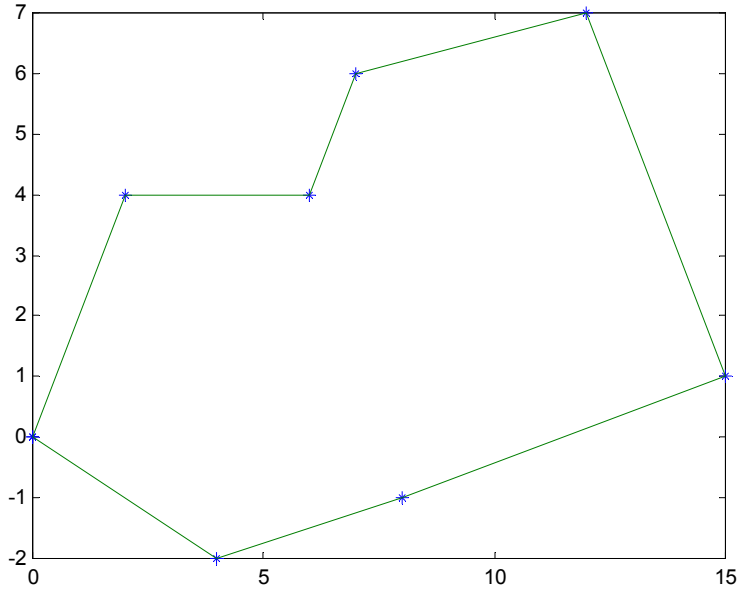
x	y	acción
0	0	0
2	4	1
6	4	1
7	6	2
12	7	1
15	1	3
8	-1	1
4	-2	1
0	0	0

En primer lugar representamos gráficamente dichos puntos conectándolos mediante rectas, lo que en la práctica supone una interpolación polinómica segmentaria lineal.

```
x=[0 2 6 7 12 15 8 4 0];
y=[0 4 4 6 7 1 -1 -2 0];
plot(x,y,'*',x,y)
```



Observamos cómo esta aproximación, además de ser muy pobre, se traduce en movimientos excesivamente bruscos para un comportamiento eficiente de un brazo robótico. Se hace, por tanto, necesaria una trayectoria que permita al brazo moverse con suavidad de una posición a otra sin poner en peligro a los objetos que transporta.



Para diseñar el spline cúbico dividiremos la trayectoria en tres trozos: desde la posición de partida hasta coger el objeto, desde ese intante hasta que soltamos el objeto y, por último, desde que se suelta el objeto hasta que se vuelve a la posición original. Esta división es lógica puesto que el brazo ha de detenerse al final de cada una de estas trayectorias.

```
accion=[0 1 1 2 1 3 1 1 0];
coger=find(accion==2);
soltar=find(accion==3);
fin=length(x);
x1=x(1:coger),    y1=y(1:coger)
x2=x(coger:soltar),    y2=y(coger:soltar)
x3=x(soltar:fin),    y3=y(soltar:fin)  ;

x1 =     0     2     6     7
y1 =     0     4     4     6
x2 =     7    12    15
y2 =     6     7     1
x3 =    15     8     4     0
y3 =     1    -1    -2     0
```

Para determinar el número de puntos que usaremos para representar los splines, calculamos la distancia mínima entre los puntos conocidos en toda la trayectoria. Así, el incremento en la variable x para definir el spline será la décima parte de esa distancia mínima.

```
incr=min(abs(x(2:fin)-x(1:fin-1)))/10;
```

De este modo, al menos habrá 10 puntos de representación entre cada par de puntos de la tabla anterior, distribuidos del siguiente modo:

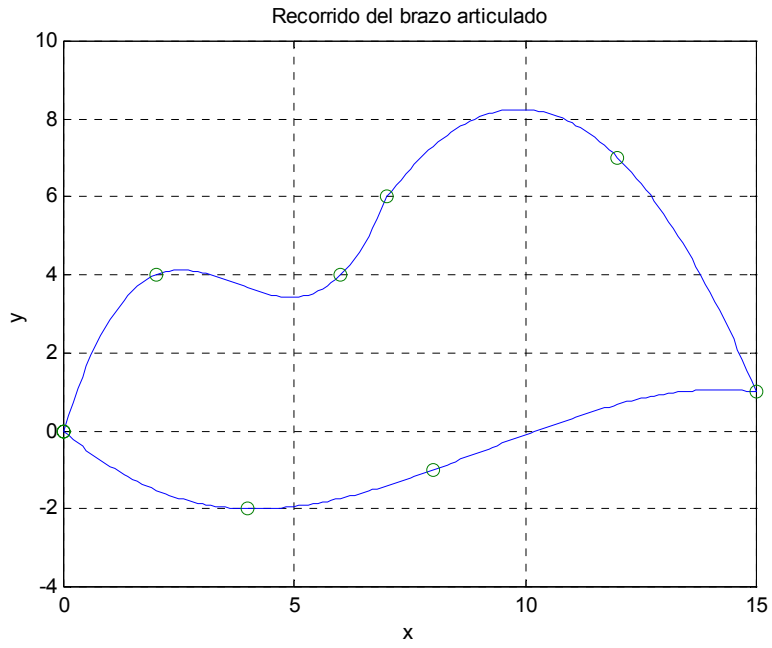
```
t1=x(1):incr*sign(x(coger)-x(1)):x(coger);
t2=x(coger):incr*sign(x(soltar)-x(coger)):x(soltar);
t3=x(soltar):incr*sign(x(fin)-x(soltar)):x(fin);
```

Calculamos los splines no-nodo mediante el comando de MATLAB:

```
s1=spline(x1,y1,t1);
s2=spline(x2,y2,t2);
s3=spline(x3,y3,t3);
```

y dibujamos la trayectoria resultante:

```
plot([t1 t2 t3],[s1 s2 s3],x,y,'o')
title('Recorrido del brazo articulado'),
xlabel('x'),ylabel('y'),grid
```



11.8 Problemas

1. Aproxima el valor de la función $f(x) = x - \cos(x^3)$ definida en el intervalo [-3,3], en el punto $x = 0.75$, empleando para ello:
   a) el polinomio interpolador de Newton de quinto grado, empleando nodos equiespaciados distribuidos en torno al punto a aproximar,
   b) un spline lineal, empleando 10 puntos equidistribuidos en el intervalo [-3,3],
   c) un spline no-nodo, utilizando los mismos puntos que en el apartado anterior.
   d) . ¿Cuál es el valor aproximado de la función en el punto $x = 0.75$ en cada una de las aproximaciones?. ¿Cuál es el error cometido en cada caso?

2. Siguiendo las instrucciones de la sección 11.4, calcula la aproximación a la función $f(x) = x - \cos(x^3)$ del problema anterior mediante un spline natural. Emplea para ello 10 puntos equidistribuidos en el intervalo [-3,3]. ¿Se observan mejoras respecto a los resultados obtenidos en el problema 1?.

3. Se ha plasmado en una tabla la evolución, cada dos horas, de la temperatura en Valencia durante un día invernal:

Hora	Grados	Hora	Grados	Hora	Grados	Hora	Grados
2	8	8	8	14	16	20	11
4	8	10	14	16	14	22	10
6	7	12	15	18	13	24	8

   a) Representa gráficamente los valores de la tabla.
   b) Mediante un polinomio interpolador de tercer grado, aproxima la temperatura en Valencia a las 8:45 de la fecha.
   c) Emplea un spline lineal para aproximar la temperatura a la misma hora.
   d) ¿Qué temperatura marcarían los termómetros en Valencia a las 8:45 si la aproximamos mediante un spline natural?¿Y con un spline no-nodo?

e) Representa gráficamente las curvas calculadas en los apartados anteriores.

4. Siguiendo las instrucciones de la sección 11.5, calcula la temperatura en Valencia a las 8:45 horas mediante un spline completo, suponiendo que la variación de la tempertura es nula en los instantes inicial y final del estudio. ¿Consideras esta suposición razonable a la vista de los resultados obtenidos en el ajuste?.
5. A continuacion motramos, tabulados, algunos valores de la función error, erf(x).

x	0.15	0.27	0.76	0.89	1.07	2.11
***f*(*x*)**	0.1680	0.2974	0.7175	0.7918	0.8698	0.9972

a) Utiliza los datos de la tabla para construir un spline cúbico (no-nodo) que aproxime la función error. Representa gráficamente dicho spline, junto con los datos tabulados.
b) Compara las aproximaciones obtenidas en $x = 0.33$, $x = 0.92$ y $x = 2.05$ con los valores que MATLAB proporciona empleando la instrucción `erf`.
c) ¿Mejoran los resultados si empleamos un spline natural?¿y con un spline completo?.

6. Las estrellas llamadas cefeidas se caracterizan por una variación en el tiempo suave y periódica de su brillo aparente, llamado magnitud visible. En la siguiente tabla se proporcionan datos experimentales acerca de la variación de la magnitud Δm de la estrella δ Cephei, que da nombre a este tipo de estrellas variables, en un periodo de tiempo t determinado.

t	-0.2	-0.1	0	0.1	0.2	0.3	0.4	0.5
Δm	-0.3	-0.5	-0.97	-0.94	-0.8	-0.7	-0.56	-0.45

t	0.6	0.7	0.8	0.9	1	1.1	1.2
Δm	-0.34	-0.25	-0.35	-0.6	-1	-0.97	-0.8

a) Representa gráficamente los puntos dados y el spline lineal calculado para aproximar la magnitud aparente de δ Cephei en los instantes $t = 0.43$, $t = 0.67$ y $t = 1.05$. ¿Cuál es la aproximación en cada caso.
b) Aproxima la magnitud aparente de δ Cephei en los instantes $t = 0.43$, $t = 0.67$ y $t = 1.05$ mediante un spline no-nodo, empleando los datos de la tabla adjunta. Representa gráficamente la aproximación obtenida.
c) ¿Varía sustancialmente la aproximación si se consideran únicamente los datos correspondientes a $t = -0.2, 0, 0.2, ..., 1.2$?. Razona la respuesta.

7. La medición del tiempo se basa en las observaciones de la rotación diurna de la bóveda celeste y del movimiento anual del Sol, es decir en la rotación de la Tierra sobre su eje y en la traslación de ésta alrededor del Sol. Para determinar la hora oficial de un lugar del que conocemos la posición del meridiano, resulta necesario conocer el valor de la llamada ecuación de tiempos (Eq), que proporciona la diferencia entre el tiempo solar medio y el tiempo solar verdadero en un mismo instante y lugar. En la siguiente tabla se proporciona el valor de la ecuación de tiempos (en minutos y segundos) el día 15 de cada mes para un año determinado (bisiesto). En aras de una mayor simplicidad, se han numerado los días del año de forma consecutiva.

a) Representa gráficamente los datos tabulados.
b) Aproxima el valor de la ecuación de tiempos el 3 de mayo (día 124) mediante un spline lineal. ¿Cuál es el error cometido teniendo en cuenta que el valor exacto es de -3m 8s? ¿Mejora un spline cúbico la aproximación?¿Cuál es el error cometido en este caso?
c) Utiliza un spline cúbico para aproximar el valor de la ecuación de tiempos el 30 de septiembre (día 274). ¿Cuál es el error cometido teniendo en cuenta que el valor exacto es de -9m 57s?
d) Representa gráficamente los splines obtenidos en los apartados anteriores.

Día	Eq	Día	Eq	Día	Eq
15	9m 3s	136	-3m 42s	259	-4m 45s
46	14m 13s	167	0m 23s	289	-14m 11s
75	9m 0s	197	5m 54s	320	-15m 25s
106	0m 5s	228	4m 29s	350	-4m 56s

11.9 Soluciones

Problema 1

Aproximaremos la función $f(x) = x - \cos(x^3)$ en el intervalo [-3,3] empleando distinto número de puntos, en función de lo pedido en cada apartado:

a) En primer lugar, partimos de 6 puntos equiespaciados, ya que el polinomio interpolador es de grado 5:

```
x = -3:6/5:3
y = x-cos(x.^3)
```

```
x =   -3.0000   -1.8000   -0.6000    0.6000    1.8000    3.0000
y =   -2.7079   -2.6999   -1.5768   -0.3768    0.9001    3.2921
```

Reordenamos ambos vectores en torno al punto a aproximar:

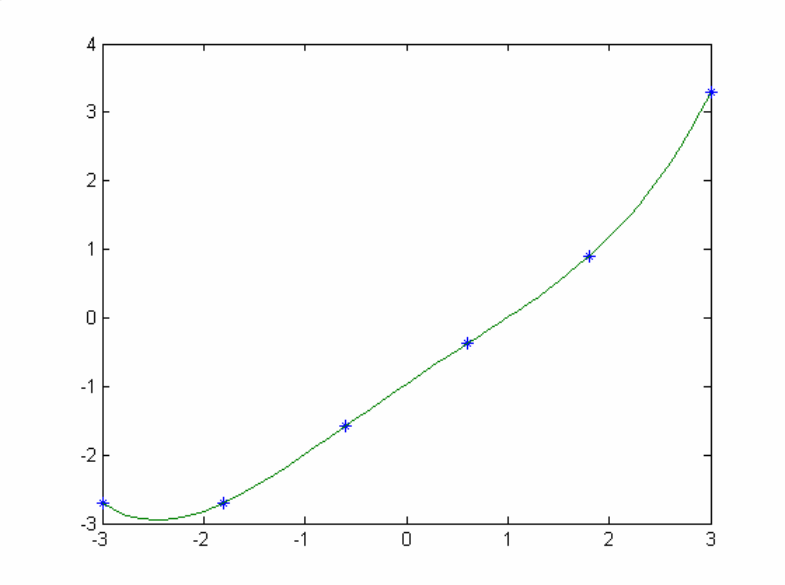
```
x1 = [0.6 -0.6 1.8 -1.8 3.0 -3.0];
y1 = [-0.3768 -1.5768 0.9001 -2.6999 3.2921 -2.7079];
```

Utilizaremos el vector

```
xg = -3:0.05:3;
```

para representar gráficamente el polinomio interpolador de Newton.

```
[yg,p]=polynew(x,y,x1,y1,xg,5);
plot(x,y,'*',xg,yg)
```



b) Para construir el spline lineal con diez puntos equiespaciados:

```
x = -3:2/3:3;
y = x-cos(x.^3);
n=length(x);
a = y(1:n-1);% y tiene n+1 componentes
b = diff(y)./diff(x);
s = [b' a']% De mayor a menor grado
```

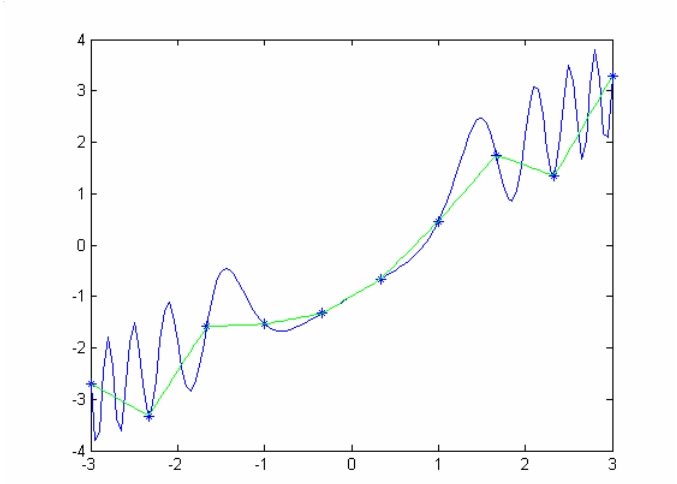
```
s =
   -0.9241   -2.7079
    2.6099   -3.3239
    0.0655   -1.5840
    0.3115   -1.5403
    1.0000   -1.3326
    1.6885   -0.6660
    1.9345    0.4597
   -0.6099    1.7493
    2.9241    1.3427
```

Para evaluar el spline, creamos el vector ps con toda la información relativa al spline.

```
ps = mkpp(x,s);
```

Evaluamos el spline y representemos gráficamente la aproximación segmentaria lineal de la función, junto con la función a aproximar:

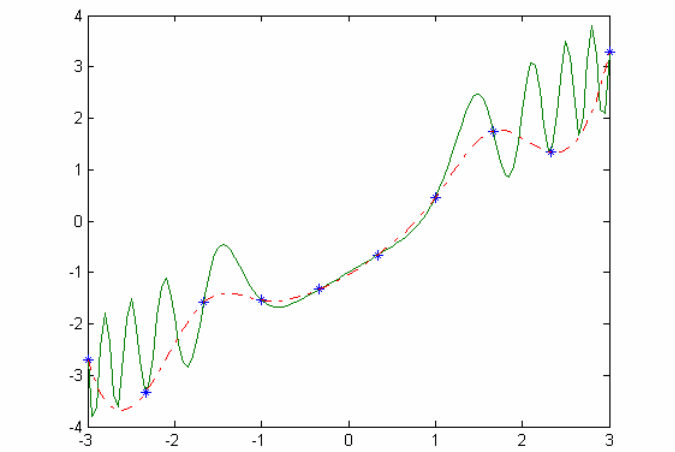
```
ys = ppval(ps,xg);
xg = -3:0.05:3;
yg = xg-cos(xg.^3);
plot(x,y,'*',xg,yg,'b',xg,ys,'g')
```



observando cómo el spline lineal es una aproximación sumamente pobre.

c) A continuación emplearemos un spline no-nodo, utilizando los mismos puntos que en el apartado anterior,

```
ps2 = spline(x,y);
ys2 = ppval(ps2,xg);
plot(x,y,'*',xg,yg,xg,ys2,'-.')
```



observando una cierta mejora en la aproximación de la función, aunque todavía está lejos de ser una buena aproximación, ya que el número de puntos empleados es muy pequeño en proporción a la complejidad de la función a aproximar.

d) Si evaluamos cada una de las aproximaciones en el punto x = 0.75 y calculamos el error exacto en cada caso, obtenemos los siguientes resultados:

- en el caso de la interpolación polinómica de grado 5:

```
x = -3:6/5:3;y = x-cos(x.^3);
x1 = [0.6 -0.6 1.8 -1.8 3.0 -3.0];
y1 = [-0.3768 -1.5768 0.9001 -2.6999 3.2921 -2.7079];
[ya,p]=polynew(x,y,x1,y1,0.75,5)
ya =   -0.2327
yex=0.75-cos(0.75^3);
errora=abs(ya-yex)
errora =    0.0704
```

- en el caso del spline lineal:

```
yb = ppval(ps,0.75);
errorb=abs(yb-yex)
```

```
errorb =    0.1999
```

- para el spline no-nodo:

```
yc = ppval(ps2,0.75);
errorb=abs(yc-yex)
errorb =    0.1173
```

Concluimos de los resultados obtenidos que, aunque en general el spline cúbico es la mejor aproximación, en el caso particular de la abscisa 0.75 la mejor aproximación la proporciona el polinomio interpolador de grado 5.

¡Error! No se encuentra el origen de la referencia.
Dado que la función a aproximar es la misma que en el ejercicio anterior:

```
x = -3:2/3:3;
y = x-cos(x.^3);
n=length(x)-1;  % número de subintervalos
```

Calculamos los valores en los extremos de la derivada segunda de la función a interpolar:

```
r1=9*x(1)^4*cos(x(1)^3)+6*x(1)*sin(x(1)^3)
rn1=9*x(n+1)^4*cos(x(n+1)^3)+6*x(n+1)*sin(x(n+1)^3)

r1 =
 -195.7544
rn1 =
 -195.7544
```

Para obtener las restantes componentes de r resolveremos el sistema lineal definido por la matriz:

```
h = diff(x);% Vector de pasos
ds = h(2:n-1)/6;% Diagonales secundarias
dl = h(1:n-1)/3;
dr = h(2:n)/3;
dp = dl+dr;% Diagonal principal
M = diag(dp) + diag(ds,-1) + diag(ds,1)

M =
0.4444   0.1111    0         0         0         0         0         0
0.1111   0.4444    0.1111    0         0         0         0         0
0        0.1111    0.4444    0.1111    0         0         0         0
0        0         0.1111    0.4444    0.1111    0         0         0
0        0         0         0.1111    0.4444    0.1111    0         0
0        0         0         0         0.1111    0.4444    0.1111    0
0        0         0         0         0         0.1111    0.4444    0.1111
0        0         0         0         0         0         0.1111    0.4444
```

y el vector de términos independientes

```
e = diff(y)./h
g = diff(e)

e =
   -0.9241    2.6099    0.0655    0.3115    1.0000    1.6885    1.9345
-0.6099    2.9241
g =
```

```
    3.5340   -2.5443    0.2459    0.6885    0.6885    0.2459   -2.5443
3.5340
```

Por último, ajustamos el primer y el último elemento de g:

```
g(1) = g(1) - h(1)*r1/6;
g(n-1) = g(n-1) - h(n)*rn1/6
g =
   47.0349   -2.5443    0.2459    0.6885    0.6885    0.2459   -
2.5443   47.0349
```

y resolvemos el sistema lineal:

```
r = M\g'

r =
  115.0765
  -36.9916
    9.9910
   -0.7589
   -0.7589
    9.9910
  -36.9916
  115.0765
```

Completamos el vector de las derivadas segundas:

```
r = [r1;r;rn1]
r =
 -195.7544
  115.0765
  -36.9916
    9.9910
   -0.7589
   -0.7589
    9.9910
  -36.9916
  115.0765
 -195.7544
```

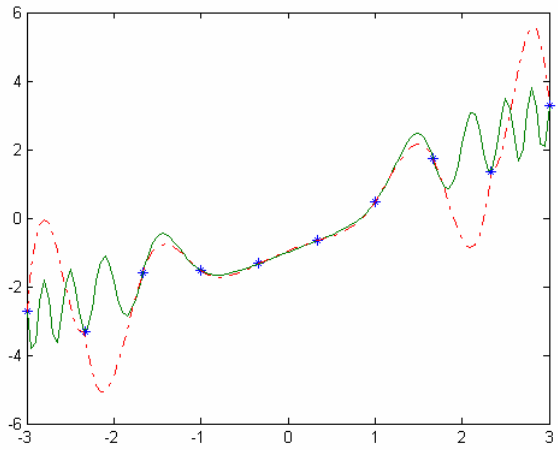
Una vez obtenidas las derivadas segundas, calculamos los coeficientes de los polinomios interpolantes.

```
rl = r(1:n);rr = r(2:n+1);
a = y(1:n)';
b = e' -(rr + 2*rl).*h'/6;
c = rl/2;
d = diff(r)./h'/6;
```

construimos el spline y lo representamos gráficamente:

```
s = [d, c, b, a] % Coeficientes de los qk
ps3 = mkpp(x,s); % Codificación del spline
xg = -3:0.05:3; yg = xg-cos(xg.^3);
ys3 = ppval(ps3,xg); % Evalúa el spline
plot(x,y,'*',xg,yg,xg,ys3,'-.')% Gráfico
s =
   77.7077  -97.8772   29.7906   -2.7079
  -38.0170   57.5383  -18.8525   -3.3239
   11.7456  -18.4958    7.1758   -1.5840
```

```
   -2.6875    4.9955   -1.8244   -1.5403
   -0.0000   -0.3794    1.2530   -1.3326
    2.6875   -0.3794    0.7470   -0.6660
  -11.7456    4.9955    3.8244    0.4597
   38.0170  -18.4958   -5.1758    1.7493
  -77.7077   57.5383   -0.8980    1.3427
```



Comprobemos el error exacto de este spline en el punto x = 0.75:

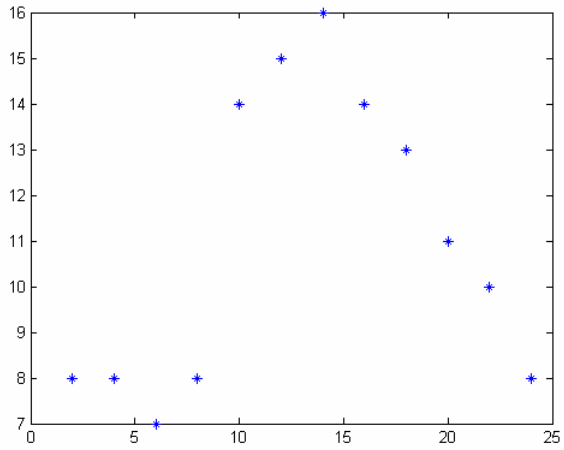
```
y3 = ppval(ps3,0.75);
errorb=abs(y3-yex)
errorb =    0.0639
```

y efectivamente, el error es el menor de los cometidos con los diferentes métodos estudiados.

Problema 3

a)Para representar gráficamente los datos de la tabla, generamos los vectores x (hora) e y (grados):

```
x=2:2:24;y=[8 8 7 8 14 15 16 14 13 11 10 8];
plot(x,y,'*')
```



b)Construiremos un polinomio interpolador de tercer grado, reordenando ambos vectores en torno al punto a aproximar y tomando los cuatro primeros elementos:

```
x1 = [8 10 6 12];
y1 = [8 14 7 15];
```

y la aproximación de la temperatura es:

```
[yb,p]=polynew(x,y,x1,y1,8.75,3)
```

```
yb =
 10.2012
```

c)Emplearemos a continuación un spline lineal con el mismo propósito; para ello, utilizaremos la instrucción de MATLAB `interp1`, simplemente introduciendo los vectores correspondientes a los datos de la tabla y el punto en el que queremos obtener la aproximación:

```
yc = interp1(x,y,8.75)

yc =
   10.2500
```

d)Para emplear un spline natural, generamos una función que incluya los pasos llevados a cabo en la sección 11.4 y la ejecutamos con los vectores x e y, considerando que la derivada de segundo orden de la temperatura es nula en los extremos. Obtendremos la matriz s en la que se encuentran los coeficientes de los sucesivos polinomios de tercer grado que forman el spline, junto con ps, que prepara el camino para la evaluación del trazador cúbico en un punto o conjunto de puntos.

```
[s,ps] = natural(x,y,0,0)

s =
   -0.0360         0    0.1442    8.0000
    0.0552   -0.2163   -0.2883    8.0000
    0.1902    0.1150   -0.4908    7.0000
   -0.4410    1.2562    2.2517    8.0000
    0.3239   -1.3899    1.9842   14.0000
   -0.2296    0.5536    0.3114   15.0000
    0.2196   -0.8243   -0.2300   16.0000
   -0.1489    0.4936   -0.8914   14.0000
    0.1261   -0.4001   -0.7044   13.0000
   -0.1056    0.3567   -0.7911   11.0000
    0.0461   -0.2767   -0.6311   10.0000
ps =
      form: 'pp'
    breaks: [2 4 6 8 10 12 14 16 18 20 22 24]
     coefs: [11x4 double]
    pieces: 11
     order: 4
       dim: 1
```

Evaluamos el spline mediante la instrucción `ppval`:

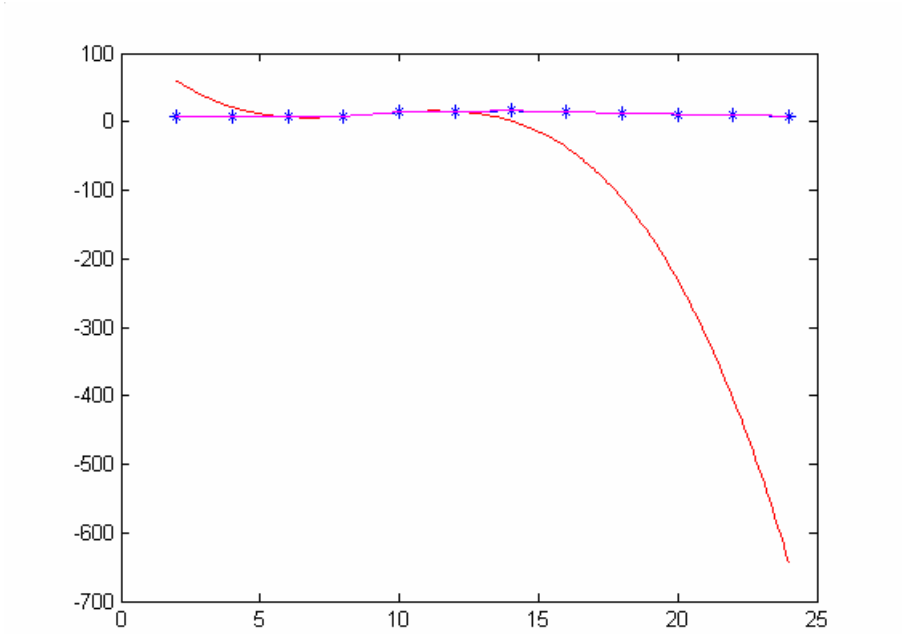
```
ys = ppval(ps,8.75)
ys =
   10.2093
```

La instrucción `spline` de MATLAB nos permite calcular la aproximación mediante un spline no-nodo:

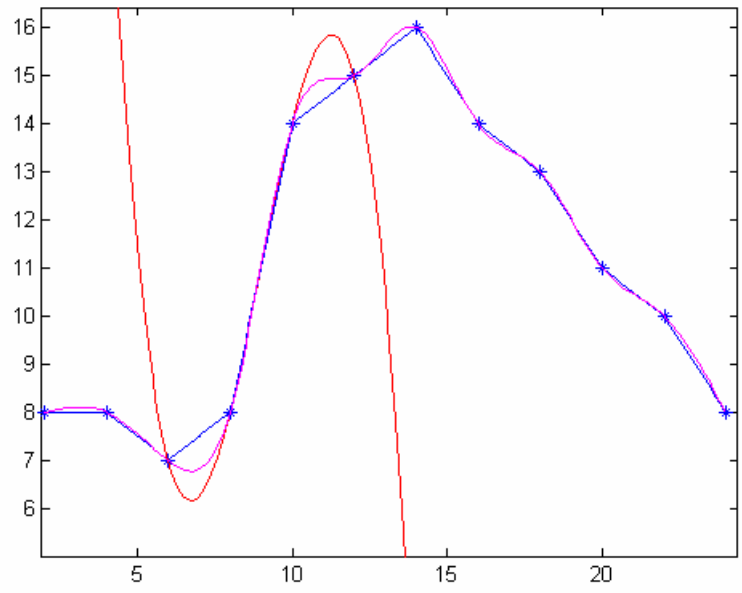
```
ps2 = spline(x,y);
ys2 = ppval(ps2,8.75)
ys2 =
 10.2067
```

e)Por último, representaremos gráficamente las distintas aproximaciones:

```
[yg,p]=polynew(x,y,x1,y1,xg,3);
ysn = ppval(ps,xg);
ysnn = ppval(ps,xg);
plot(x,y,'*',xg,yg,'r',x2,y2,'b',xg,ysn,'g',x2,ysnn,'m')
```



El polinomio interpolador decrece muy rápidamente y distorsiona la gráfica. Efectuamos un zoom sobre la región en la que se encuentran los puntos de interpolación y observamos que los resultados obtenidos por los splines natural y no-nodo se superponen, pudiendo considerarse ambos la mejor aproximación.



Spline Lineal

```
>> x = 0:pi/3:2*pi;
>> y = sin(x);
>> n = length(x) - 1;
>> a = y(1:n);
>> b = diff(y)./diff(x);
>> s = [b' a'];
>> ps = mkpp(x,s);
>>
>> %la grafica
>>
>> xg = linspace(0,2*pi);

>> yg = sin(xg);
>> ys = ppval(ps,xg);
>> plot (x, y, '*', xg, yg, xg, ys);
```

Spline Cubico
Natural

```
>> %Spline natural para el seno , sabiendo las derivadas segudnas en los extremos
>>
>> x = -2*pi:pi/3:2*pi;
>> y = sin(x);
>> xg = linspace(-2*pi,2*pi);
>> yg = sin(xg);
>> [ps,s]=natural(x,y,0,0)

ps =

      form: 'pp'
    breaks: [1x13 double]
     coefs: [12x4 double]
    pieces: 12
     order: 4
       dim: 1
```

s =

```
  -0.0754         0    0.9097    0.0000
  -0.0000   -0.2369    0.2481    0.8660
   0.0754   -0.2369   -0.6616    0.8660
   0.0754   -0.0000   -0.9097   -0.0000
  -0.0000    0.2369   -0.2481   -0.8660
  -0.0754    0.2369    0.6616   -0.8660
  -0.0754         0    0.9097         0
  -0.0000   -0.2369    0.2481    0.8660
   0.0754   -0.2369   -0.6616    0.8660
   0.0754    0.0000   -0.9097    0.0000
  -0.0000    0.2369   -0.2481   -0.8660
  -0.0754    0.2369    0.6616   -0.8660
```

```
>> ys = ppval(ps,xg);
>> plot (x, y, '*', xg, yg, xg, ys);
%xg, yg es la funcion original
% xg, ys es la funcion obtenida
```

%Aproximar cualquier dato

```
>> x = 1:10;
>> y = rand(10,1);
>> [ps,s]=natural(x,y,0,0);
>> xg = linspace(1,10);
>> ys = ppval(ps, xg);
>> plot (x, y, '*', xg, ys);
```

Otras posibles condiciones de derivadas segundas en los extremos
dan lugar a otros splines.

%SPLINE de MATLAB

```
>> ps = spline(x,y);
>> ysm = ppval(ps,xg);
```

```
>> plot (xg, ys);
>> plot (x, y, '*', xg, ys);
```

difiere de la nuestra sólo en los extremos

12. La función *f*(*x*) = 0

El problema inverso al de evaluar una función en un punto consiste en hallar un punto en el que la función tome un valor determinado. Normalmente, el problema inverso es más difícil, porque no disponemos de una expresión sencilla de la función inversa.
Por ejemplo, hallar el volumen de un cubo conociendo la arista es fácil, mientras que para obtener la arista a partir del volumen, necesitamos una calculadora capaz de extraer raíces cúbicas.
Matemáticamente, estos problemas se plantean como la resolución de una ecuación de la forma

$$f(x) = 0$$

donde *f* es una función continua conocida y *x* es la incógnita a obtener, con la condición de que verifique la ecuación dada.
Es posible obtener una idea aproximada de la solución empleando técnicas gráficas, que desarrollamos en la primera sección, aunque en este caso ni siquiera puede hablarse de precisión en los cálculos, ya que no hay tales.
Dispuestos a obtener una solución precisa de la ecuación $f(x) = 0$, nos planteamos el dilema entre resolución analítica (limitada sólo a algunos casos en los que se puede despejar la variable independiente, o la función *f* pertenece a una clase de funciones determinada) o numérica, en cuyo caso recurrimos a un *método iterativo*. La idea es una formalización del proceso de tanteo, en el que vamos probando valores de *x* cada vez más próximos a la solución, que hagan cada vez más pequeño el valor de *f(x)*.
En este capítulo resolvemos de forma aproximada ecuaciones no lineales de una variable empleando para ello diferentes métodos iterativos: distinguiremos en primer lugar aquellos que utilizan el teorema de Bolzano, aprovechando el cambio de signo de la función *f* antes y después de la raíz. Llamaremos a éstos *métodos de intervalos*, y concretamente estudiaremos los métodos de *Bisección* y *Regula Falsi*. Completaremos el estudio de los métodos elementales de resolución de ecuaciones de una variable en la siguiente sección, con los llamados métodos abiertos, entre los que destacan los métodos de Punto Fijo, Newton y de la Secante.
Recordemos que para resolver una ecuación por un método iterativo hemos de:

1. Obtener una *estimación inicial* x_1 de la solución. En ella se resume la información previa de que disponemos. En la siguiente sección aprenderemos a deducir la información necesaria de la representación gráfica de la función cuyo cero estamos buscando.
2. *Refinar iterativamente* la aproximación, obteniendo valores x_2, x_3, x_4, ..., que converjan a la solución x^*.
3. Establecer un *criterio de parada*.

Respecto a éste último punto, dependiendo del método numérico utilizado, exigiremos que las iteraciones varíen poco o que la ecuación se satisfaga con aproximación suficiente. Para determinar el error de la aproximación x_k, deberíamos calcular la diferencia, en valor absoluto, entre aproximación y solución exacta:

$$\left| x_k - x^* \right|$$

Sin embargo, desconocemos el valor de la solución exacta x^*, así que consideramos

$$\left| x_k - x_{k-1} \right|$$

como estimación del error. Si esta cantidad es suficientemente pequeña (menor que un argumento de entrada al que siempre llamaremos *tol*), detenemos el algoritmo suponiendo que hemos alcanzado la precisión requerida. Tendremos en cuenta que si el método converge lentamente, estamos subestimando el error; al contrario, si converge rápidamente, el error será seguramente inferior a *tol*.
A veces, no interesa tanto la determinación precisa de la solución, como que el valor de la función sea pequeño. En este caso, el criterio de parada será

$$\left|f\left(x_k\right)\right| < tol$$

Conviene también limitar el número máximo de iteraciones, en previsión de que el algoritmo diverja, oscile indefinidamente o converja muy lentamente. Llamaremos *maxiter* al máximo número de pasos que permitimos realizar a un algoritmo iterativo. Si este número se supera, detenemos las iteraciones aunque no se haya alcanzado la convergencia y emitimos un mensaje de advertencia.
Algunos de estos métodos que mostraremos en este capítulo y el siguiente pueden generalizarse para resolver sistemas de ecuaciones no lineales, $f\left(x\right)=0$, con las modificaciones que implica el que *f* sea una función vectorial de varias variables; ello, sin embargo, forma parte del contenido de asignaturas de cursos posteriores.

12.1 Método gráfico

El más simple de los métodos (aunque no el más preciso) para obtener una aproximación a la raíz de una ecuación no lineal $f\left(x\right)=0$ consiste, en primer lugar, en obtener una representación gráfica de la curva $y=f\left(x\right)$ para, posteriormente, observar donde cruza dicha curva el eje de abscisas. Ese valor en el que se anula la función $f\left(x\right)$ es la aproximación que buscamos.
Comenzaremos con un problema clásico: calcular cuál debe ser la arista de un cubo para duplicar su volumen. Buscamos la solución a la ecuación no lineal $f\left(x\right)=x^3-2=0$. La representación gráfica de la curva $y=f\left(x\right)=x^3-2$ la obtenemos ejecutando las expresiones:

```
x=0:.1:2;
y=x.^3-2;
plot(x,y)
```

Partiendo de la representación gráfica podemos deducir que la curva corta el eje de abscisas aproximadamente en el punto (1.25,0). Para confirmar esta estimación basta sustituirla en la ecuación:

```
y=1.25^3-2
```

Para obtener mayor precisión, podemos emplear el comando `ginput` de MATLAB para obtener un resultado más preciso, marcando con el ratón sobre la gráfica el punto cuyas coordenadas queremos averiguar.

```
[X,Y]=ginput(1)
```

No es práctico, sin embargo, tratar de obtener una solución a una determinada ecuación no lineal empleando exclusivamente métodos gráficos, ya que ésta no podría obtenerse con toda la precisión deseada. No obstante, sí es posible obtener aproximaciones de la raíz, que se pueden utilizar a su vez como aproximaciones iniciales en los métodos numéricos que desarrollaremos en este capítulo y el siguiente.

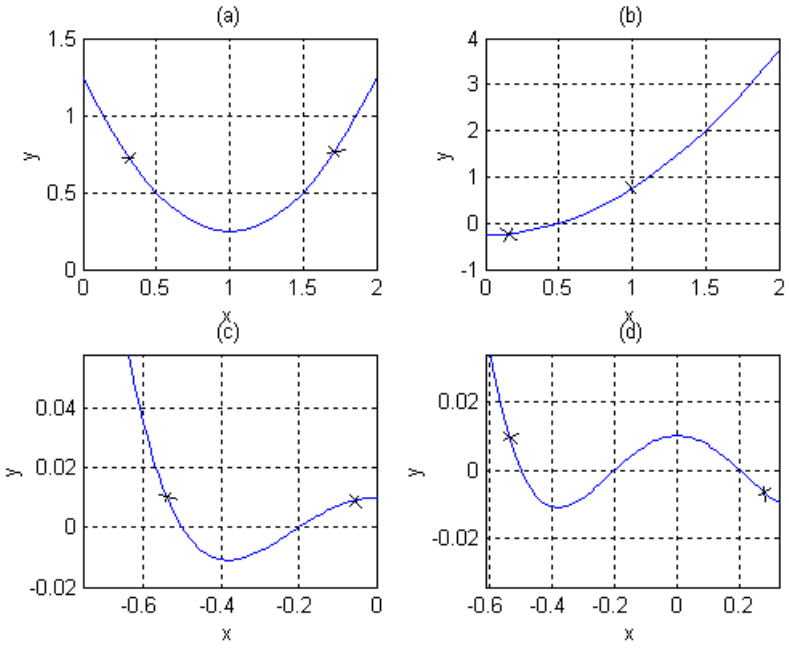
Además, la representación gráfica ayuda en la comprensión de los resultados obtenidos mediante métodos numéricos e incluso permite detectar errores en los mismos. Suponiendo que $f(x)$ es una función continua, en las siguientes gráficas observamos diferentes casos en los que la raíz que buscamos está (o no) en el intervalo definido por dos valores de la variable x, x_1 y x_2:

(a) $f(x_1)\cdot f(x_2)>0$, es decir, ambas imágenes tienen el mismo signo y no existe ninguna raíz en el intervalo $[x_1,x_2]$;

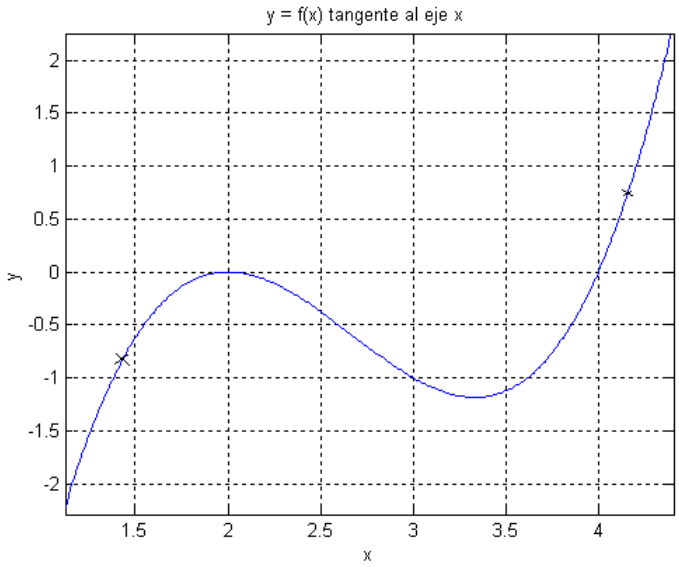
(b) $f(x_1)\cdot f(x_2)<0$, por lo que el teorema de Bolzano nos permite afirmar que existe una raíz (o un número impar de ellas) en el intervalo $[x_1,x_2]$; en este caso, la raíz es única.

(c) $f(x_1)\cdot f(x_2)>0$, nuevamente ambas imágenes tienen el mismo signo; sin embargo, sí existen raices en el intervalo $[x_1,x_2]$, en número par;

(d) $f(x_1)\cdot f(x_2)<0$ por lo que, aplicando de nuevo el teorema de Bolzano, afirmamos que existe un número impar de raices en el intervalo $[x_1,x_2]$, concretamente tres.



Las funciones discontinuas y aquellas que son tangenciales al eje x son excepciones a los casos anteriormente expuestos: observamos en la siguiente figura la representación gráfica de la función $y=(x-2)^2(x-4)$. Notemos que $x = 2$ es una raíz múltiple (de multiplicidad 2) y en este caso, la curva es tangente al eje de abscisas. Así, una mala elección del intervalo (por ejemplo [1.5,2.5]) podría hacernos pensar que no hay raíces que encontrar cuando en realidad hay una raíz doble. Sin embargo, un intervalo más amplio en el que la función cambie de signo puede permitirnos encontrar todas los ceros de la función.



12.2 Métodos de intervalos

En esta sección trataremos aquellos métodos que aprovechan el cambio de signo de la función en el entorno de una raíz. Así, partiendo de dos valores proporcionados inicialmente, dichos métodos tratan de reducir el tamaño del intervalo que encierra a la raíz buscada, hasta converger a ésta con la suficiente precisión.

12.2.1 Método de la bisección

El *método de bisección* garantiza la convergencia a la raíz de una ecuación $f(x)=0$, partiendo de un intervalo en cuyos extremos la función continua *f* toma signos distintos. Como ya hemos observado anteriormente, el llamado Teorema de Bolzano afirma que si una función continua cambia de signo en un intervalo, ésta se anula en algún punto del mismo. La bisección es un proceso sistemático para buscar ese punto: tras comprobar si la función cambia de signo en un intervalo concreto, se evalúa el valor de la función en el punto medio, que es una nueva estimación de la raíz. Si aún no tenemos la precisión deseada, se divide el intervalo por la mitad y se comprueba en cuál de ellas se encuentra la raíz, es decir, se comprueba en qué subintervalo cambia de signo la función. El proceso se repite y la aproximación a la raíz mejora cada vez más a medida que los subintervalos se dividen por la mitad.

Podemos resumir el método en los siguientes pasos:

3. Partimos de un intervalo inicial [*a,b*] en el que suponemos está la raíz. Esto se puede verificar comprobando que $f(a)\cdot f(b)<0$

4. Determinamos la primera aproximación a la raíz como $c=\dfrac{a+b}{2}$.

5. Realizamos las siguientes comprobaciones:

- **a)** Si $f(a)\cdot f(c)<0$, entonces la raíz se encuentra en el intervalo [*a,c*]. Así, consideramos $b = c$ y continuamos en el punto (2) para determinar una nueva aproximación de la raíz.
- **b)** Si $f(c)\cdot f(b)<0$, entonces la raíz está en el intervalo [*c,b*]. Así, consideramos $a = c$ y continuamos en el punto (2) para determinar una nueva aproximación de la raíz.

6.Si $f(c)\cdot f(b)=0$ (ó $f(c)\cdot f(a)=0$), la raíz es igual a c y termina el proceso.

Consideraremos como cota del error cometido en la aproximación de la raíz la amplitud del último subintervalo considerado, siendo el centro de éste nuestra aproximación a la solución. Además, tras un proceso de n iteraciones, puede demostrarse que el error cometido está acotado por el cociente:

$$Error < \left|\frac{b-a}{2^n}\right|$$

donde *a* y *b* son los extremos iniciales del intervalo en el que encontramos la solución.

Por otra parte, como criterio de parada en el algoritmo de bisección, consideraremos la amplitud del intervalo actual en cada iteración (también podríamos utilizar $incr = |f(c)| < tol$), ya que se puede probar que ésta tiende a cero.

Ejemplo 1

Resolveremos a continuación el problema del cálculo de la arista de un cubo para duplicar su volumen, descrito por la ecuación $f(x)=x^3-2=0$. Comprobamos visualmente que en intervalo [1,2] podemos encontrar una raíz y evaluamos la función $f(x)=x^3-2$ en los extremos del intervalo:

```
a=1;
fa=a^3-2;
b=2;
fb=b^3-2;
```

calculamos el punto medio del intervalo y evaluamos en él la función:

```
c =(a+b)/2;
fc=c^3-2;
```

a continuación, comprobamos en cuál de los subintervalos [a,c] y [c,b] se encuantra la raíz y nos centramos en él:

```
if fa*fc<0   %elige [a,c]
      b=c;
      fb=fc;
end
if fc*fb<0   %elige [c,b]
      a=c;
      fa=fc;
end
```

Comprobamos si la amplitud del nuevo intervalo [a,b] es tan pequeña como para considerar que su punto medio es una buena aproximación de la raíz. Si es así, hemos alcanzado el objetivo que nos habíamos propuesto. En caso contrario, calcularemos de nuevo el punto medio de [a,b] y repetiremos el proceso previo. En el caso del cálculo de la arista del cubo, si exigimos una tolerancia de 10^{-6} , son necesarias 20 iteraciones para alcanzar la solución aproximada, x = 1.259921. Además, dado que conocemos la solución exacta, podemos calcular el error exacto cometido en este caso:

```
error=abs(x-2^(1/3))
```

Si estudiamos la tasa de convergencia lineal, obtenemos en todas las iteraciones el valor 0.5, lo que resulta lógico si pensamos en que asociamos la estimación del error a la amplitud del intervalo actual, que se divide por la mitad en cada paso. Así, podemos afirmar que la velocidad de convergencia del método de bisección es siempre lineal, de tasa 0.5.

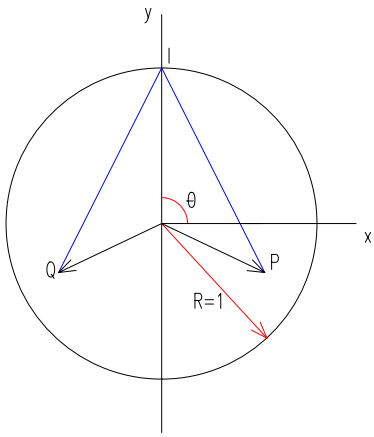
La ventaja del algoritmo de bisección es que siempre converge, supuesto que partimos de un intervalo que contiene a la raíz. Sin embargo, la convergencia es muy lenta y puede darse el caso de que el punto medio calculado en un paso esté más alejado de la raíz que el del paso anterior.

Ejercicio

Implementa en MATLAB el método de bisección, de manera que, dados el intervalo inicial, la tolerancia y el número máximo de iteraciones, proporcione la solución aproximada al problema, la estimación del error y el número de iteraciones empleado.

Ejemplo 2

Para jugar al billar en una mesa circular hemos de golpear la bola Q de la figura con la bola P, tras un impacto I en la banda. Conocido el radio R de la mesa y las posiciones en coordenadas cartesianas (cuyo origen es el centro de la mesa) de los puntos P y Q, (x_P, y_P) y (x_Q, y_Q), el punto de impacto viene definido por el angulo central θ.

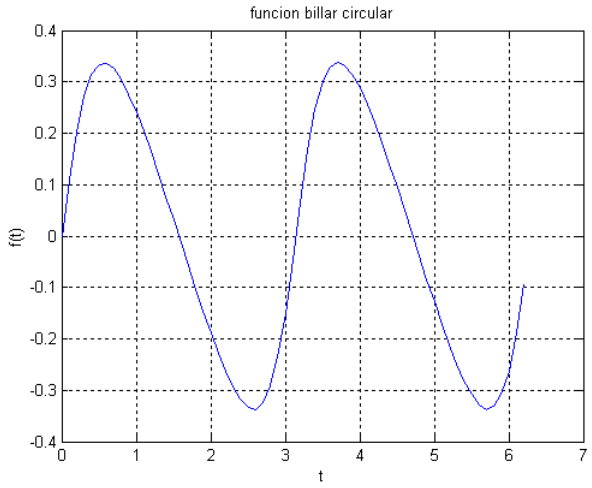


Tras un análisis geométrico del problema, se prueba que los valores de θ que proporcionan los puntos de impacto posibles son los ceros de la función:

$$f(\theta)=\frac{x_P\sin\theta-y_P\cos\theta}{\sqrt{(R\cos\theta-x_P)^2+(R\sin\theta-y_P)^2}}+\frac{x_Q\sin\theta-y_Q\cos\theta}{\sqrt{(R\cos\theta-x_Q)^2+(R\sin\theta-y_Q)^2}}$$

Consideraremos un billar de radio unidad, $P=(0.6,0)$ y $Q=(-0.6,0)$. Si estudiamos gráficamente los ceros de la función podremos proporcionarle al método de bisección un intervalo inicial razonable:

```
t = 0:.1:2*pi;
P = [0.6,0];Q = [-0.6,0];
y1=(P(1)*sin(t)-P(2)*cos(t));
den1=sqrt((cos(t)-P(1)).^2+(sin(t)-P(2)).^2);
y1=y1./den1;
y2=(Q(1)*sin(t)-Q(2)*cos(t));
den2=sqrt((cos(t)-Q(1)).^2+(sin(t)-Q(2)).^2);
y2=y2./den2;
y=y1+y2;
plot(t,y),grid on
```



Podemos deducir que una de las raíces buscadas está en el intervalo [1,2], por lo que aplicamos el método de bisección con este intervalo de partida, tolerancia 10^{-4} y 100 iteraciones como máximo a la función $f(\theta)$, almacenada en el fichero billar.m:

```
[sol,iter]=bisec('billar',1,2,1e-4,100)

sol =    1.5707
iter =     14
```

Observamos en la figura anterior que una solución exacta de la ecuación es $\theta = \frac{\pi}{2} = 90º$, por lo que podemos calcular el error cometido en nuestra aproximación:

```
error=abs(sol-pi/2)
```

Podemos observar en la siguiente tabla las sucesivas aproximaciones con distintas aproximaciones iniciales, el número de iteraciones necesario y el error exacto cometido en las mismas.

k	x_k (rad.)	x_k (grad.)	Error ex.
1	1.5000	85.9437	4.0563
2	1.7500	100.2676	10.2676
3	1.6250	93.1056	3.1056
4	1.5625	89.5247	0.4753
5	1.5938	91.3151	1.3151
6	1.5781	90.4199	0.4199
7	1.5703	89.9723	0.0277
8	1.5742	90.1961	0.1961
9	1.5723	90.0842	0.0842
10	1.5713	90.0282	0.0282
11	1.5708	90.0003	0.0003
12	1.5706	89.9863	0.0137
13	1.5707	89.9933	0.0067
14	1.5707	89.9968	0.0032

Notemos que para alcanzar esta solución la primera aproximación es especialmente buena, c = 1.5, pero el siguiente iterado es 1.75, que evidentemente es una aproximación bastante peor que la anterior. Esto sucede también al acercarnos a la solución exacta: observamos cómo la aproximación proporcionada por el iterado x_{12} es bastante mejor que la siguiente, aunque finalmente obtengamos la precisión requerida. Estudiaremos

otros métodos que nos permitirán evitar estas desviaciones sistemáticas, resultado de considerar el punto medio del intervalo como iterado, independientemente del valor que tome la función en ese punto. Generalmente, el método de bisección se utiliza en combinación con otros métodos más rápidos, bien para proporcionar una estimación inicial suficientemente próxima a la raíz, bien para evitar la divergencia en cualquier paso intermedio.

12.2.2 Método de Regula-Falsi

Nos planteamos a continuación una variante en la que, en lugar de tomar el punto medio, se considera el punto (*c*,0) en el que la recta que pasa por los puntos $\left(a, f\left(a\right)\right)$ y $\left(b, f\left(b\right)\right)$ cortan el eje de abscisas. Para averiguar el valor de *c*, igualaremos dos fórmulas que nos proporcionan la pendiente de esa recta:

$$m = \frac{f(b) - f(a)}{b - a}$$

que se obtiene al usar los puntos $\left(a, f\left(a\right)\right)$ y $\left(b, f\left(b\right)\right)$, y también

$$m = \frac{0 - f(a)}{c - a}$$

Igualando ambas pendientes obtenemos la expresión

$$c = a - f(a) \Big/ \frac{f(b) - f(a)}{b - a}$$

que divide el intervalo en partes proporcionales al valor de la función en los extremos.

Una vez calculado el punto c, se elige el subintervalo [*a,c*], o [*c,b*] en el que la función cambia de signo. Este se denomina método de *Régula Falsi*: aunque el método generalmente converge con menos iteraciones que el de bisección, presenta el inconveniente de que la longitud del intervalo puede no tender a cero, por lo que estimaremos la precisión mediante $incr = \left| f\left(c\right) \right| < tol$, y de este modo medimos cuán cerca está c de ser solución de la ecuación $f\left(x\right) = 0$. Por otra parte, obtendremos la diferencia entre dos iterados consecutivos, lo que nos permitirá calcular la velocidad de convergencia.

El algoritmo del método de regula falsi difiere del de bisección únicamente en el cálculo del nuevo extremo del intervalo, en el que se utiliza la expresión

$$c = a - f(a) \Big/ \frac{f(b) - f(a)}{b - a}$$

junto con la forma en que se estima la precisión alcanzada por el método.

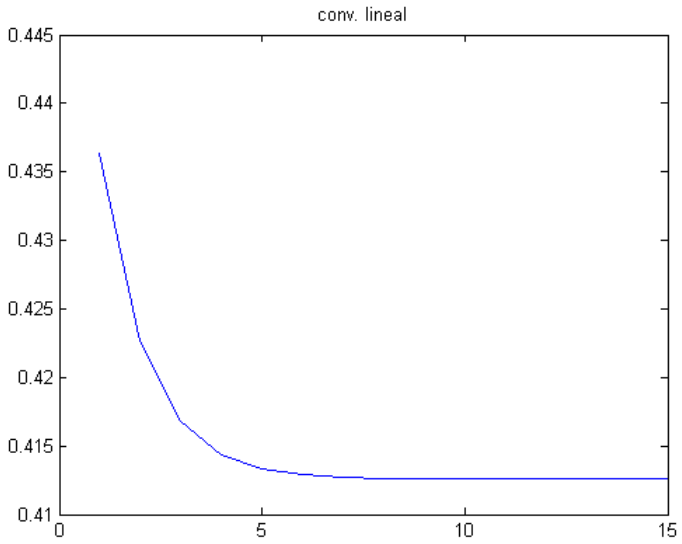
Ejercicio

Implementa en MATLAB el método de Regula Falsi, de manera que, dados el intervalo inicial, la tolerancia y el número máximo de iteraciones, proporcione la solución aproximada al problema, la estimación del error y el número de iteraciones empleado.

Ejemplo 3

Si aplicamos el método de Régula Falsi a la ecuación del ejemplo 1, $f\left(x\right) = x^3 - 2 = 0$. con un itervalo inicial $a = 1$, $b = 2$, tolerancia 10^{-6} y un máximo de 50 iteraciones,

```
[sol,iter]=regula('cubica',1,2,1e-6,50)
```



obtenemos una solución suficientemente precisa, *sol* = 1.25992096146283, en 17 iteraciones, con una velocidad de convergencia lineal, pero ligeramente más rápida que utilizando el método de bisección. En este caso, la tasa de convergencia es de 0.41, aproximadamente.

Ejemplo 4

En el ejemplo del al billar circular de radio unidad, con las posiciones de las bolas $P=(0.6,0)$ y $Q=(-0.6,0)$ y la misma tolerancia, obtenemos el resultado deseado, sol = 1.5708 = 90.0000º, en tan sólo tres iteraciones, siendo en este caso la velocidad de convergencia superlineal (tasa de convergencia lineal prácticamente nula, 0.0345) y el error exacto, 2.9 10^{-9}.

```
[sol,iter]=regula('billar',1,2,1e-4,50)
error=abs(sol-pi/2)
```

Aunque el método de Regula Falsi proporciona mejores resultados que el de Bisección en los ejemplos tratados, no es correcto generalizar esa conclusión, como se deduce del siguiente ejemplo:

Ejemplo 5

Aplicando sobre la función $f(x)=x^{10}-1$ los métodos anteriores, partiendo del intervalo [0,1.5], con tolerancia 10^{-6} y un máximo de 500 iteraciones, observamos que el método de regula falsi converge con mucha más lentitud que el de bisección a la raíz múltiple (de multiplicidad 10) $x = 1$.

```
[sol,iter]=bisec(0,1.5,1e-6,500)
sol =   0.99999976158142
iter =    21

[sol,iter]=regula(0,1.5,1e-6,500)
 sol =   0.99998978416958
iter =   158
```

Más adelante describiremos métodos cuya velocidad de convergencia es, bajo determinadas condiciones, más rápida que la de tipo lineal o superlineal que presentan los anteriores.

12.3 Métodos abiertos

En los métodos de intervalo descritos en la sección anterior partimos siempre de dos valores (los extremos del intervalo) que encierran a la raíz, de manera que convergen a ella con mayor o menor velocidad. A continuación describiremos métodos que, partiendo de una (métodos de Punto Fijo y Newton) o dos aproximaciones (método de la secante), la aproximan con la precisión deseada. Estos métodos pueden converger o no a la solución buscada pero, cuando convergen, lo hacen generalmente con mayor velocidad que los métodos de intervalos.

12.3.1 Método de Punto Fijo

Comenzaremos definiendo un punto fijo de una función $g(x)$ como un número real p que verifica: $g(p)=p$. Geométricamente, los puntos fijos de una función son los puntos de corte de la función $y=g(x)$ con la recta $y=x$.

La clave del método de punto fijo es transformar la ecuación $f(x)=0$ en otra *equivalente* de punto fijo, $g(x)=x$ y construir una sucesión de iterados

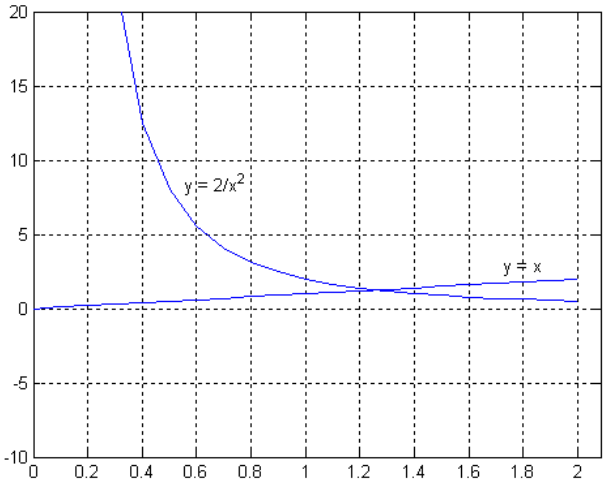
$$x_{k+1}=g(x_k) \qquad k=1,2,...$$

a partir de una estimación inicial x_1 de la solución.
Si la sucesión $\{x_k\}$ converge y la función de iteración g es continua, entonces el límite es punto fijo de g y, por la equivalencia, solución de la ecuación $f(x)=0$.

Usaremos el ejemplo del cálculo de la arista de un cubo para duplicar su volumen, $f(x)=x^3-2=0$ para ilustrar los pasos del *método de punto fijo*: en primer lugar hay que despejar una x de la ecuación $x^3-2=0$, dejándola en función de otras x (si las despejamos todas, ya tenemos la solución). En este caso obtenemos dos ecuaciones de punto fijo equivalentes a la inicial.

$$x=2/x^2 \quad \text{o bien} \quad x=\sqrt{2/x}$$

A continuación sustituimos la x del segundo miembro por una estimación inicial de la solución y tomamos el resultado como nuevo valor de x. Este proceso se repite hasta que x se estabiliza o bien al comprobar que los valores de x divergen u oscilan mucho.
Para elegir una buena estimación inicial, representamos gráficamente la función de punto fijo y buscamos su intersección con la bisectriz del primer cuadrante.



Partiremos de la estimación inicial de la solución $x = 1$, deducida gráficamente:

```
x=1;                % Estimación inicial
```

y evaluamos repetidas veces la función de punto fijo:

```
x=2/x^2           % Ecuación equivalente
```

iteración	sol. aprox.
1	1
2	2
3	0.5
4	8
5	0.03125

Observamos que no converge. Repetimos ahora el proceso iterando con la función de punto fijo $x = \sqrt{2/x}$:

```
x=1;                % Estimación inicial
x=sqrt(2/x)       % Otra ecuación equivalente
```

iteración	sol. aprox.
1	1.41421356237310
2	1.18920711500272
3	1.29683955465101
4	1.24185781207348

Observamos cómo los iterados se estabilizan rápidamente, aproximándose a la solución $x = \sqrt[3]{2}$.

Para mejorar el experimento almacenamos en un vector las diez primeras aproximaciones, calculadas mediante un bucle **for**.

```
k=1; x(k)=2;     % Estimación inicial
for k=1:10
x(k+1)= sqrt(2/x(k)); k=k+1;       % Iteraciones
end
```

siendo la última estimación obtenida:

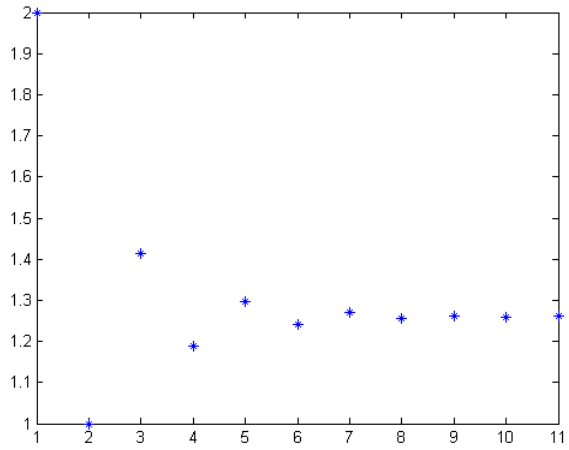
```
x(end)
ans =   1.26048973986985
```

y el error exacto cometido:

```
error=abs(x(10)-2^(1/3))
error =   0.00113661034516
```

Finalmente, para observar la evolución de las componentes de x, representaremos las diez primeras iteraciones con

```
plot(x,'*')
```



Para estudiar la velocidad de convergencia, analizamos la tasa de convergencia lineal a partir de los diez iterados calculados:

```
e = abs(diff(x));          % ek = |xk+1 - xk|
n = length(e);
r = e(2:n)./e(1:n-1)          % rk = ek+1/ek
```

Observamos que las componentes de *r* se estabilizan entorno a 1/2, por lo que la convergencia es lineal de razón 1/2. Podemos llegar también a esta conclusión representando gráficamente el vector *r*.

```
plot(r)
```

Demostraremos que la convergencia del método de punto fijo depende del valor de la derivada de la función de iteración. De hecho, de la elección adecuada de la función de punto fijo depende, no sólo la velocidad de convergencia, sino la misma convergencia del método.

Ejemplo 6

En la resolución de la ecuación $f(x)=x^3-2=0$ mediante la búsqueda de un punto fijo de la ecuación $x=\sqrt{2/x}$, se comprueba que $\left|g'(x^*)\right|=\frac{1}{2}$, siendo x^* el punto fijo de $g(x)=\sqrt{2/x}$. No es casual que el valor de la derivada coincida con la razón de convergencia: si escribimos las componentes de *r* como cocientes incrementales de *g* en valor absoluto,

$$r=\left(\frac{|x_3-x_2|}{|x_2-x_1|},\frac{|x_4-x_3|}{|x_3-x_2|},\frac{|x_5-x_4|}{|x_4-x_3|},\ldots\right)$$

$$=\left(\left|\frac{g(x_2)-g(x_1)}{x_2-x_1}\right|,\left|\frac{g(x_3)-g(x_2)}{x_3-x_2}\right|,\left|\frac{g(x_4)-g(x_3)}{x_4-x_3}\right|,\ldots\right)$$

observamos como, para una función de punto fijo continuamente derivable,

$$\lim_{k\to\infty} r_k=\lim_{k\to\infty}\left|\frac{g(x_{k+1})-g(x_k)}{x_{k+1}-x_k}\right|=g'(x^*)$$

siendo $\lim_{k\to\infty} x_k=x^*$.

Para programar el método de punto fijo en MATLAB, crearemos un fichero de función que, partiendo de una estimación inicial, x_1, construya una sucesión de iterados (en

realidad, un vector finito) de la función *g*, que a su vez estará en otro fichero de función, hasta detectar si hay convergencia o no. Se establecerá como criterio de parada la diferencia, en valor absoluto, entre dos estimaciones sucesivas de la solución. Partiendo de estas diferencias, resultará sencillo calcular posteriormente la tasa de convergencia, tanto lineal como cuadrática.

Ejemplo 7

Comprobaremos el funcionamiento de nuestro algoritmo para encontrar la solución de la ecuación $f(x)=x^3-2=0$ empleando la ecuación de punto fijo $x=\sqrt{2/x}$ que almacenamos en el fichero de función "pf_cubica.m":

```
[sol, incr, it] = pfijo('pf_cubica',1,1e-6,50)
```

La solución obtenida es 1.25992077227690, con precisión estimada de 8.328540990198974 10^{-7}. Hemos necesitado 20 iteraciones (sin contar la estimación inicial). El trabajo de estos algoritmos suele medirse por las veces que evalúan la función, que han sido 20.

Ejercicio

Modifica tu función **pfijo** para que dé como argumento de salida, *x*, el vector de iterados. Observa la convergencia con `plot(x)`.

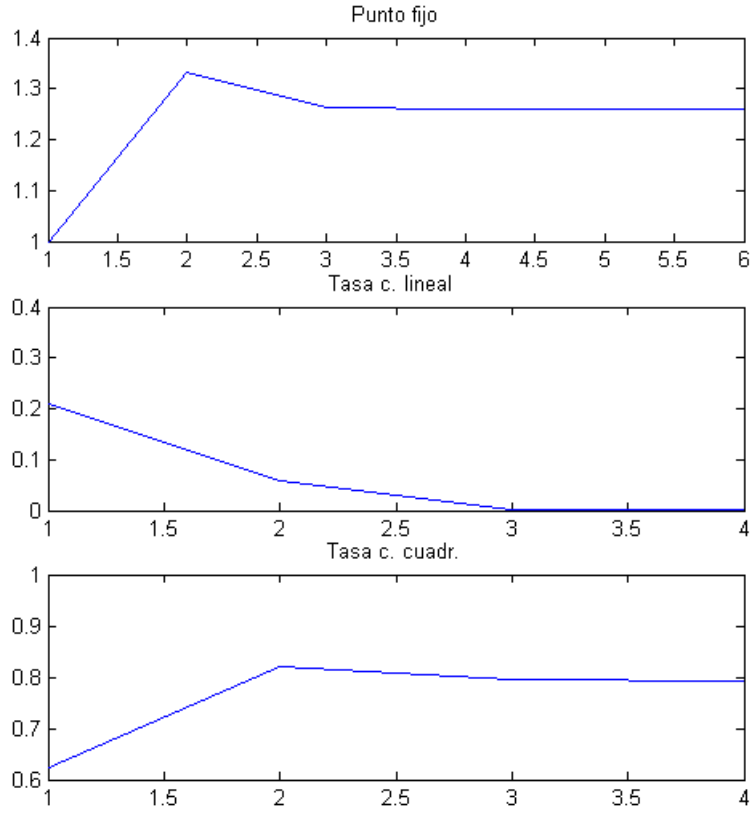
Ejercicio

Pon en **pf_cubica2.m** la función de punto fijo **x=2/x^2** y ejecuta **pfijo** con un máximo de 6 iteraciones. Observa gráficamente la divergencia.

Ejemplo 8

Comprobemos que la función de punto fijo $g(x)=\frac{1}{3}\left(\frac{2}{x^2}+2x\right)$ resuelve el problema del cubo y converge cuadráticamente: al ejecutar el método de punto fijo sobre la función pf_cubica3.m en que almacenamos $g(x)=\frac{1}{3}\left(\frac{2}{x^2}+2x\right)$ (se puede probar fácilmente que esta función de punto fijo es equivalente a la función original).

```
[sol, incr, it] = pfijo('pf_cubica3',1,1e-6,50)

sol =   1.25992104989487
incr =    1.228965818000916e-010
it =     5
```



Observamos que encuentra la solución del problema con una precisión muy elevada y una rapidez mucho mayor de lo esperado: en tan sólo 5 iteraciones. Si estudiamos la velocidad de convergencia a través de la representación gráfica de su tasa de convergencia (lineal y cuadrática), notamos que, con esta función de punto fijo, dicha velocidad es cuadrática, ya que la tasa de convergencia cuadrática tiende a 0.8, aproximadamente.

12.3.1.1 Convergencia del método de Punto Fijo

Deducimos de los resultados obtenidos hasta el momento que la elección de la función de punto fijo es importante a la hora de resolver una ecuación no lineal por este método, no sólo para su convergencia, sino también para acelerar la velocidad a la que ésta se produce. El siguiente resultado es la clave del proceso:

Sea $g:[a,b]\to R$ continuamente derivable tal que $g(x)\in[a,b]$ para todo $x\in[a,b]$.
Además, supongamos que $|g'(x)|\le K<1$ para todo $x\in\,]a,b[$.
Entonces, la iteración de punto fijo

$$x_{k+1}=g(x_k)\quad k=1,2,...$$

converge al único punto fijo x^* para cualquier punto de partida $x_1\in[a,b]$.

Esbozamos a continuación la demostración de este resultado: aplicando el Teorema del Valor Medio a la función g en el intervalo $[x_1, x^*]$, se obtiene que

$$g(x_1)-g(x^*)=g'(\xi_1)(x_1-x^*)$$

para cierto ξ_1 del intervalo $]x_1, x^*[$. Por definición de las iteraciones y por ser x^* punto fijo,

$$g(x_1) - g(x^*) = x_2 - x^* = g'(\xi_1)(x_1 - x^*)$$

Tomando valor absoluto y usando la acotación de la derivada tenemos

$$|x_2 - x^*| = |g'(\xi_1)||x_1 - x^*| < K|x_1 - x^*|$$

Aplicamos de nuevo el TVM al intervalo $[x_2, x^*]$ y obtenemos

$$|x_3 - x^*| = |g'(\xi_2)||x_2 - x^*| < K|x_2 - x^*|$$

Teniendo en cuenta la desigualdad anterior

$$|x_3 - x^*| < K|x_2 - x^*| < K^2|x_1 - x^*|$$

Repitiendo este proceso, para n = 4, 5, ... probamos que

$$|x_n - x^*| < K^{n-1}|x_1 - x^*|$$

Al ser $K < 1$, K^n tiende a 0 y, por tanto, x_n tiende a x^*, como queríamos demostrar.

Además, ya hemos demostrado que la *convergencia* de los iterados de punto fijo es, al menos, *lineal* de razón **$|g'(x_*)|$,** por la hipótesis de continuidad de la derivada. Sin embargo, si $g'(x^*) = 0$**,** se demuestra análogamente, utilizando el desarrollo de Taylor de orden 2, que las iteraciones convergen cuadráticamente. Este es el caso de la función de punto fijo en el ejemplo anterior, $g(x) = \frac{1}{3}\left(\frac{2}{x^2} + 2x\right)$, para la que la convergencia del método era cuadrática ya que, en este caso, $g'(\sqrt[3]{2}) = 0$.
Como regla práctica podemos decir que:

- Si $|g'(x^*)| > 1$ los iterados no convergen a x^*.
- Si $|g'(x^*)| < 1$ los iterados convergen linealmente a x^*,
- Si $|g'(x^*)| = 0$ los iterados convergen cuadráticamente a x^*.

Hemos visto que los casos anteriores corresponden a las funciones de punto fijo $g(x) = \frac{2}{x^2}$, $g(x) = \sqrt{\frac{2}{x}}$ y $g(x) = \frac{1}{3}\left(\frac{2}{x^2} + 2x\right)$, respectivamente, para el ejemplo del cálculo de la raíz cúbica de 2.

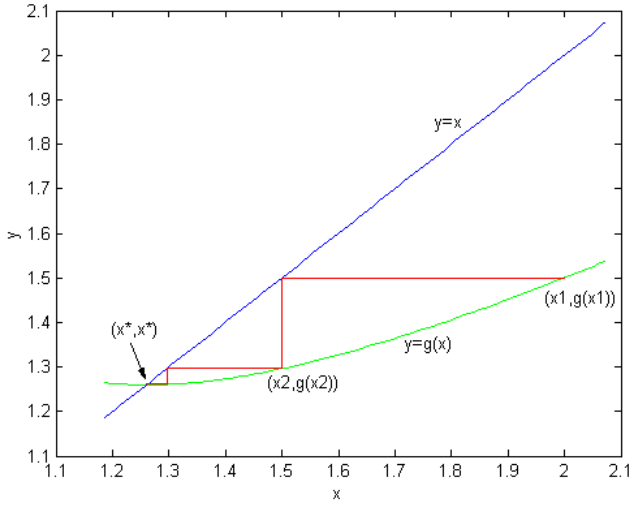
12.3.1.2 Interpretación gráfica de la iteración de Punto Fijo

Sabemos que si la derivada de la función de punto fijo verifica la desigualdad:

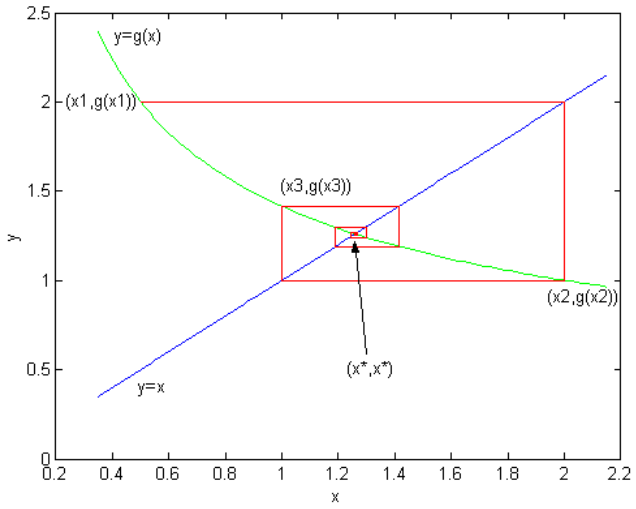
$$|g'(x^*)| < 1$$

los iterados convergen linealmente a x^*. Sin embargo, dado que x^* es un punto fijo de la función g, las curvas $y = g(x)$ e $y = x$ se cortan en el punto (x^*, x^*). Si hay convergencia, la iteración puede ser de dos tipos: monótona y oscilante.

- Si $0 < g'(x^*) < 1$ los iterados forman una sucesión monótona que converge a x^*.

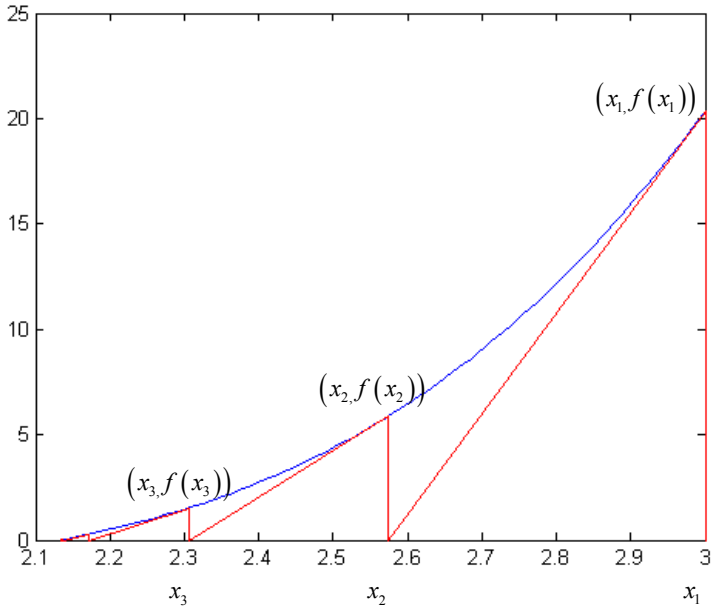


- Si $-1 < g'(x^*) < 0$, los iterados toman valores alternativamente mayores y menores que el punto fijo x^*, aunque cada vez más próximos a éste.



12.3.2 Método de Newton

Los métodos iterativos para resolver la ecuación $f(x) = 0$ vistos en hasta el momento presentan convergencia lineal, salvo un caso especial del método de punto fijo, en el que la derivada de la función de iteración $g(x)$ se anula en la solución.
El método de Newton usa la derivada para aproximar la función por la tangente a su gráfica. Interpretando el método de Newton como caso especial de punto fijo, es fácil justificar que presenta convergencia cuadrática, pues la derivada de la función de iteración se anula en la raíz de $f(x) = 0$.
La idea del método de Newton-Raphson para resolver la ecuación $f(x) = 0$ es aproximar localmente la función f por una función lineal l y resolver la ecuación $l(x) = 0$. Tomamos esta solución como nueva estimación de la raíz de $f(x)$ y repetimos el proceso hasta obtener la precisión deseada.



Sea x_1 la estimación inicial de la solución. El método de Newton halla la tangente a la gráfica de f en el punto $(x_1,f(x_1))$ y toma la intersección x_2 de la tangente con el eje OX como nueva estimación de la solución.
La ecuación de la recta tangente es

$$y = f(x_1) + f'(x_1)(x - x_1)$$

y la intersección con el eje OX se obtiene haciendo $y = 0$

$$x_2 \equiv x = x_1 - \frac{f(x_1)}{f'(x_1)}$$

siempre que la derivada f' no se anule en x_1.
Tomamos x_2 como nueva estimación de la raíz y repetimos el proceso para obtener una nueva estimación:

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$$

Procediendo iterativamente, en el paso $k + 1$ determinamos

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

La función $g(x)$ definida por la relación

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

recibe el nombre de función de iteración de Newton-Raphson. Puesto que buscamos el punto x^* tal que $f(x^*) = 0$, observemos que la solución de nuestra ecuación verifica $g(x^*) = x^*$ (si $f'(x^*) \neq 0$) y, por tanto, es punto fijo de la función de iteración. Notemos que la derivada de la función de iteración se anula en x^*,

$$g'(x^*) = \frac{f(x^*) f''(x^*)}{(f'(x^*))^2} = 0$$

En estas condiciones, el método de punto fijo converge cuadráticamente. En efecto, si estudiamos el error cometido en la iteración *k*:

$$e_{k+1} = x_{k+1} - x^* = g(x_k) - g(x^*)$$

y desarrollando $g(x_k)$ en torno a x^* por la fórmula de Taylor, tenemos:

$$e_{k+1} = g'(x^*)(x_k - x^*) + \frac{1}{2} g''(\eta_k)(x_k - x^*)^2$$

$$= \frac{1}{2} g''(\eta_k) e_k^{\,2}$$

Suponiendo que la derivada segunda de g es aproximadamente constante, deducimos que el error cometido en la iteración k es proporcional al cuadrado del error cometido en la iteración anterior. En la práctica, esto supone que el número de cifras decimales exactas se duplica cada cierto número de iteraciones.

Ejemplo 9

Jugaremos de nuevo al billar en una mesa circular de radio unidad: hemos de golpear la bola Q de coordenadas $(-0.6, 0)$ con la bola $P = (0.6, 0)$, tras un impacto I en la banda.

Sabemos que los ceros de la función:

$$f(\theta) = \frac{0.6 \sin\theta}{\sqrt{(\cos\theta - 0.6)^2 + \sin^2\theta}} + \frac{-0.6 \sin\theta}{\sqrt{(\cos\theta + 0.6)^2 + \sin^2\theta}}$$

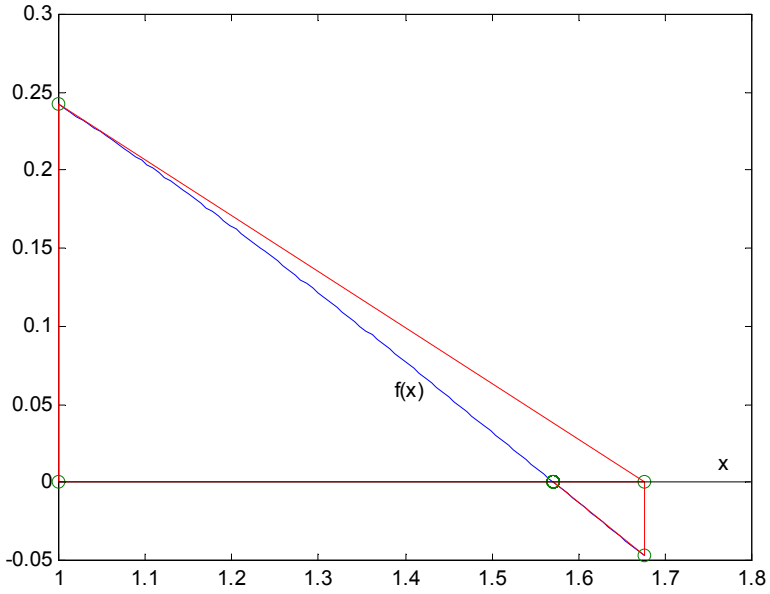
nos proporcionan la posición del punto de impacto. Para resolver este problema mediante el método de Newton, hemos de calcular la derivada de la función *f*, que incluimos, junto con la función, en **nw_billar.m**:

```
x=1; k=0;  % Estimación inicial; paso
xx=x; yy=0;% Datos para la gráfica
k=k+1;% Paso k
[y,dy]=nw_billar(x(k));% Valor de la función y de su derivada
xx=[xx x(k)]; % Datos para la gráfica
yy=[yy y];
delta = -y/dy  %diferencia entre dos estimaciones consecutivas
x(k+1) = x(k)+ delta% Iteración de Newton
xx=[xx x(k+1)]; yy=[yy 0];
```

Repitiendo varias veces desde la línea del paso k y dibujando el resultado con

```
xa=min(xx); xb=max(xx);
xg=linspace(xa,xb); yg=xg.^3-a;
plot(xg,yg,xx,yy,'o',xx,yy,,[xa xb],[0 0],'r')
```

obtenemos la siguiente gráfica, en la que se muestra gráfica el proceso de convergencia del método de Newton.



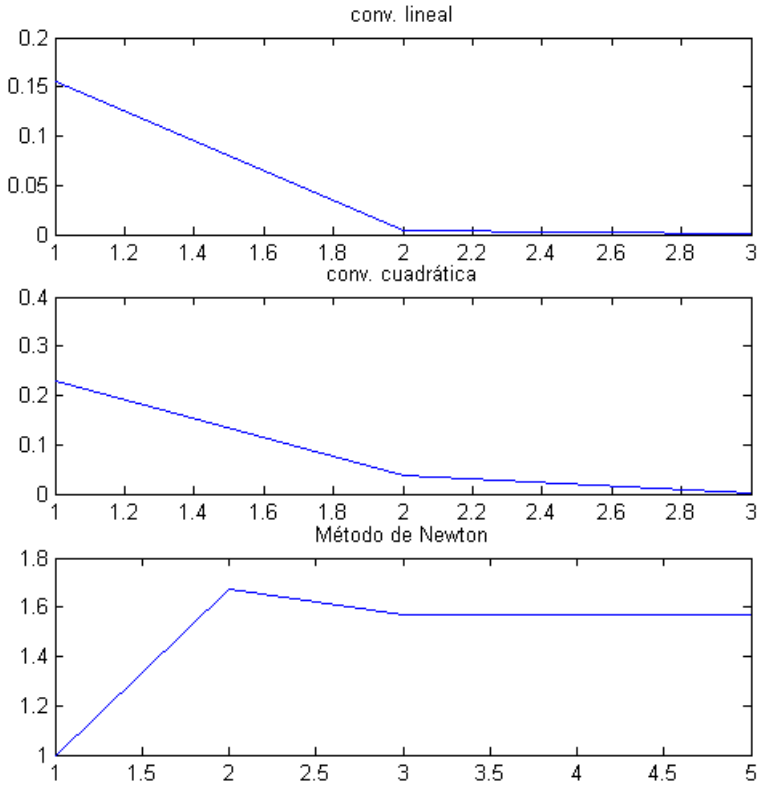
Así, el algoritmo del método de Newton es muy similar al de punto fijo; la diferencia estriba fundamentalmente en la expresión iterativa inherente al método y en que ésta necesita evaluar, no sólo la función *f*, sino también su derivada, en cada iterado:
Como criterio de parada consideraremos la diferencia, en valor absoluto, entre dos estimaciones sucesivas de la solución, aunque en determinadas circunstancias (tales como raíces múltiples) resulta más eficiente comprobar si $\left|f\left(x_k\right)\right| < tol$. Notemos que no es necesario calcular la diferencia entre dos iterados consecutivos, ya que esa información es la almacenada en la variable *delta*. Partiendo de estas diferencias, resultará sencillo calcular posteriormente la tasa de convergencia, tanto lineal como cuadrática.

Ejemplo 10

Si, una vez implementado en MATLAB el método de Newton, lo ejecutamos sobre la función **nw_billar.m**, obtenemos los siguientes resultados:

```
[x, incr, iter] = newton1var('nw_billar',1,1e-6,50)

x =   1.5708
incr =    2.6270 e-011
iter =     4
```

Observamos que bastan 4 iteraciones para encontrar la solución con una precisión estimada de diez cifras decimales exactas, frente a las 14 iteraciones que necesitaba el método de bisección partiendo de c = 1 como primer iterado. En cuanto a la velocidad de convergencia, dado que la tasa de convergencia cuadrática tiende a cero, concluimos que efectivamente, el método de Newton tiene convergencia cuadrática en este caso.



Por otra parte, la convergencia del método de Newton se degrada al acercarse a una raíz múltiple de $f(x)$. De hecho, se puede demostrar que, en general, la velocidad de convergencia del método de Newton pasa a ser lineal en el caso de raíces múltiples.

Ejemplo 11

Si aplicamos el método de Newton al cálculo de la raíz cúbica de dos, la función a considerar es

$$f(x) = x^3 - 2 = 0$$

y su derivada

$$f'(x) = 3x^2$$

que almacenamos en la función **nw_cubica.m**, con lo que la iteración de Newton queda

$$x_{k+1} = x_k - \frac{x_k^3 - a}{3x_k^2} = \frac{1}{3}\left(\frac{a}{x_k^2} + 2x_k\right)$$

Respecto a la velocidad de convergencia, para obtener la solución con una precisión de 10^{-6}, ejecutamos la función:

```
[x, incr, iter] = newton1var('nw_cubica',1,1e-6,50)
x =   1.25992104989487
incr =    1.228965856871140e-010
iter =     5
```

obteniendo la solución en 5 iteraciones. Notemos que el número de iteraciones coincide con el empleado por el método de punto fijo con la función $g(x) = \frac{1}{3}\left(\frac{2}{x^2} + 2x\right)$. De hecho, esta ecuación de punto fijo coincide plenamente con la expresión iterativa de Newton, razón por la cual obteníamos convergencia cuadrática con el método de punto

fijo, mientras que con las otras funciones de punto fijo empleadas, la convergencia era lineal. Por otra parte, ¿qué ocurre si tomamos una estimación inicial compleja, por ejemplo, $x_1 = 1+i$?.

Las siguientes condiciones son suficientes para que el método de Newton converja a la solución de la ecuación $f(x)=0$, siendo $f \in C^2(D),\ D \subseteq R$:

- La estimación inicial debe ser próxima a la raíz;
- $f'(x) \neq 0,\ \ x \in D$;

Notemos que si la derivada de la función *f* se anula en un iterado, el siguiente paso de Newton no está definido. En estos casos, puede hacerse inestable, aunque haya una raíz próxima. Sin embargo, si la derivada se anula en la raíz, el método de Newton converge linealmente, en lugar de cuadráticamente.
Cuando la derivada de la función *f* es difícil de calcular, recurrimos a una modificación consistente en evaluar la derivada una sola vez y utilizar el mismo valor en todas las iteraciones.

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_1)}$$

Este método modificado es mucho más lento, perdiendo la convergencia cuadrática. Una solución intermedia consiste en calcular la derivada cada cierto número de iteraciones. Como solución de compromiso, podemos evaluar la derivada cada cierto número de pasos, para no degradar tanto la convergencia.
En el caso de raíces múltiples, la convergencia cuadrática del método de Newton puede perderse, como es el caso de la función $f(x)=(x-2)^2(x-4)=0$, que necesita de 20 iteraciones para alcanzar la solución $x = 2$, partiendo de la estimación inicial $x_1 = 1.5$ con una precisión estimada de $2.6\ 10^{-9}$. Una modificación del método de Newton que mejora la convergencia en estos casos es aplicar un factor m al cociente entre función y derivada, igual a la multiplicidad de la raíz que buscamos:

$$x_{k+1} = x_k - m\frac{f(x_k)}{f'(x_k)}$$

en el caso $f(x)=(x-2)^2(x-4)$, considerando $m = 2$ obtenemos la solución en tan sólo 4 iteraciones. Sin embargo, no conocemos a priori la multiplicidad de la raíz que buscamos, aunque el tanteo o la comparación de $f(x)$ con la función $(x-r)^m$ (donde *r* es la raíz) pueden dar buenos resultados.

12.3.3 Método de la secante

Si no disponemos de la derivada, la estimamos mediante un cociente incremental para h pequeño. Así se "simula" bastante bien el método de Newton. En cada paso evaluamos dos veces la función *f*, pero no necesitamos la expresión de la derivada (de hecho, en la práctica, sólo hay que evaluar la función *f* en el último iterado).
El método que veremos a continuación utiliza como iterados los puntos que intervienen en el cálculo del cociente incremental, reduciendo a una las evaluaciones de la función por cada paso. Es el método de la secante.
Geométricamente, el método de la secante aproxima la función *f* por la recta que pasa por los puntos $(x_1, f(x_1))$ y $(x_2, f(x_2))$ y toma su intersección, x_3, con el eje OX como siguiente estimación.

En el segundo paso, traza la secante por los puntos de abscisa x_2 y x_3 y la intersecta de nuevo con OX para hallar la tercera estimación x_4. El proceso se repite, como siempre, hasta alcanzar la precisión deseada.
Expresamos la iteración genérica de la secante en términos análogos a la de Newton como

$$x_{k+1} = x_k - f(x_k) \Big/ \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Existe una diferencia fundamental entre los métodos de la secante y regula falsi: las dos estimaciones iniciales x_1 y x_2 en el método de regula falsi deben verificar $x_1 < x^* < x_2$ (siendo x* la raíz buscada), mientras que en el método de la secante partimos de dos estimaciones cualesquiera. Precisamente debido a esta diferencia, el cociente

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

es una buena aproximación de la derivada de la función f en el caso de la secante, ya que x_k y x_{k-1} son valores próximos entre sí.

Ejemplo 12

Para resolver la ecuación del billar circular sin necesidad de calcular la derivada de la función

$$f(\theta) = \frac{0.6\sin\theta}{\sqrt{(\cos\theta - 0.6)^2 + \sin^2\theta}} + \frac{-0.6\sin\theta}{\sqrt{(\cos\theta + 0.6)^2 + \sin^2\theta}}$$

emplearemos el método de la secante, partiendo de las estimaciones iniciales 1 y 2:

```
x = [1,2];   % Abscisas iniciales
y = billar(x);% Ordenadas
delta = x(2)-x(1);k = 1;% Paso inicial
```

y ejecutando varias veces las siguientes sentencias

```
k = k+1% Paso k
delta = -y(k)/((y(k)-y(k-1))/delta)
x(k+1) = x(k)+delta
y(k+1) = billar(x(k+1))
```

obtenemos la solución en 4 iteraciones, sin contar las estimaciones iniciales, con un estimación del error de $2.9\ 10^{-9}$. La velocidad de convergencia es, en este caso, superlineal, ya que la tasa de convergencia lineal tiende a cero. Viéndolo en positivo, con una evaluación por paso (y sin necesidad de utilizar $f'(x)$), se logra una convergencia mucho más rápida que con cualquier otro método.

12.4 Cálculo de raíces con MATLAB

La resolución numérica de ecuaciones polinómicas ha suscitado gran interés históricamente, dada la limitación de los métodos analíticos en cuanto al grado del polinomio. La función **roots** de MATLAB obtiene todas las raíces de un polinomio construyendo la matriz asociada al polinomio (considerado característico) y obteniendo los valores propios de dicha matriz.

Ejemplo 13

Consideramos la ecuación polinómica $x^5 - 3x^4 - 10x^3 + 10x^2 + 44x + 48 = 0$. Basta definir el polinomio,

```
p=[1 -3 -10 10 44 48];
```

para a continuación obtener sus raíces, tanto reales como imaginarias, con el comando **roots**:

```
roots(p)
```

Por otra parte, para resolver ecuaciones no polinómicas, MATLAB dispone de la orden **fzero**. Su primer argumento es el nombre (del fichero) de la función cuyos ceros tratamos de hallar. Tal nombre es una cadena de caracteres y, por tanto, ha de ponerse entre comillas. Por ejemplo,

```
fzero('f',x0,tol)
```

siendo f.m el fichero que contiene la ecuación a resolver o el nombre de una función reconocida por MATLAB.

```
fzero('atan',1,1e-6)

fzero('x.^3-2',1,1e-7)
```

12.5 Problemas

1. La raíz de la ecuación $tan(x)-c=0$ es relativamente difícil de encontrar cuando c es una cantidad grande. Trata de averiguar la razón de éste comportamiento a través de los siguientes pasos:

a) Dibuja la función $f(x)=tan(x)-c$ para $c=5$ y $c=10$ y estima un intervalo en que se encuentre la raíz buscada.

b) Aproxima, para cada uno de los valores de c especificados en el apartado (a), la solución a la ecuación no lineal mediante los métodos de Bisección y Regula Falsi empleando el intervalo estimado en el apartado (a), con una tolerancia de 10^{-4}.

c) Analiza razonadamente la velocidad de convergencia de ambos métodos para el intervalo estimado en el apartado (a).

2. Las temperaturas en el interior de un material con fuentes de calor incrustadas pueden determinarse si se resuelve la siguiente ecuación:

$$e^{-\frac{1}{2}t}\cosh^{-1}\left(e^{\frac{1}{2}t}\right)=\sqrt{\frac{1}{2}L}$$

Dado que L = 0.088, encuentra el valor de la temperatura t:

a) Gráficamente, representando la ecuación a resolver. Estima además un intervalo en el que se encuentre la solución, que sirva de estimación inicial al resto de apartados.

b) Empleando el método de Bisección con el intervalo inicial obtenido a partir de la representación. Calcula la velocidad de convergencia.

c) Emplea el método de Punto Fijo con el punto medio del intervalo anterior como estimación inicial. Calcula la velocidad de convergencia. ¿Se verifican las condiciones de convergencia? Compruébalo.

d) Compara los resultados obtenidos en los apartados anteriores.

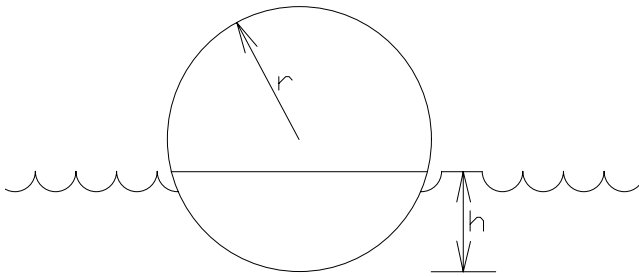
3. Se construye una caja sin tapadera a partir de una hoja metálica rectangular que mide 10 por 16 centímetros. Queremos averiguar cuál debe ser el lado de los cuadrados que hay que recortar en cada esquina para que el volumen de la caja sea 100 centímetros cúbicos.

a) Plantea la ecuación que nos proporcionará el volumen de la caja en función del lado l de los cuadrados a recortar.
b) Representa gráficamente la ecuación a resolver y obtén un intervalo inicial en el que esté el valor de l que estamos buscando.
c) Resuelve dicha ecuación mediante el método de la Bisección, empleando una tolerancia de 10^{-9} y tomando como intervalo inicial el obtenido en el apartado (b).
d) ¿Mejora el método de Regula Falsi el resultado obtenido por bisección?
e) Obtén el lado de dichos cuadrados empleando el método de Punto Fijo, con estimación inicial el punto medio del intervalo utilizado en apartados anteriores.
f) Compara los resultados obtenidos por ambos métodos (solución, número de iteraciones, velocidad de convergencia,…).

4. La curva que pasa por un cable colgante se llama catenaria. Supongamos que el punto más bajo de una catenaria es el origen (0,0), entonces la ecuación de la catenaria es $y = C\cosh\left(\frac{x}{C}\right) - C$. Si queremos determinar la catenaria que pasa por los puntos $(\pm a, b)$, debemos resolver la ecuación $b = C\cosh\left(\frac{a}{C}\right) - C$ de incógnita C.

a) Prueba que la catenaria que pasa por los puntos $(\pm 10, 6)$ es $y = 9.1889\cosh\left(\frac{x}{9.1889}\right) - 9.1889$.
b) Halla la catenaria que pasa por los puntos $(\pm 12, 5)$ empleando el método de Regula Falsi. Utiliza la representación gráfica para obtener el intervalo inicial. Estima la velocidad de convergencia del método en este caso.
c) Repite el apartado (b) aplicando el método de Newton, utilizando la representación gráfica para obtener la estimación inicial. Analiza las condiciones de convergencia del método de Newton en este caso y su velocidad de convergencia
d) Compara el resultado obtenido en los apartados (b) y (c).

5. Una esfera de densidad d y radio r pesa $\frac{4}{3}\pi\left(3rh^2 - h^3\right)$. Encuentra la profundidad a que una esfera de densidad 0.6 se hunde en el agua, en términos de una fracción del radio de la esfera.



Con una tolerancia de 10^{-4}, utilizando los algoritmos de Regula Falsi, Punto Fijo y Newton. Establece una tabla que permita comparar resultados, número de iteraciones, velocidad de convergencia, etc...

6. Consideremos el siguiente sistema no lineal:

$$x^2 + x - y^2 = 1$$
$$y - \text{sen}\left(x^2\right) = 0$$

a) Determina gráficamente todas las soluciones del sistema.
b) Transforma el sistema de ecuaciones en una única ecuación no lineal y aproxima con precisión de 10^{-6} sus soluciones mediante el método de Punto Fijo. Indica las aproximaciones iniciales utilizadas y las iteraciones necesarias para cada una de las raíces.
c) Repite el apartado (b) empleando el método de Newton y comprueba la convergencia cuadrática del método.

7. Halla la menor raíz positiva de la ecuación

$$e^{-x^2} - \cos(x) = 0$$

Con una tolerancia de 10^{-6}, utilizando los algoritmos de Bisección y Punto Fijo. Establece una tabla que permita comparar resultados, número de iteraciones, etc...

8. Consideremos el siguiente sistema no lineal:

$$e^x e^y + x\cos\left(y\right) = 0$$
$$x + y - 1 = 0$$

a) Analiza gráficamente las raíces del sistema para $x \in [-6,6]$.
b) Transforma el sistema de ecuaciones en una única ecuación no lineal y aproxima las raíces encontradas en el apartado anterior mediante el método de Newton con una tolerancia de 10^{-5}. ¿Cuántas iteraciones son necesarias en cada caso?. Estima la velocidad de convergencia en una de las raíces.

c) Para acelerar la convergencia del método de Newton se introduce un factor de sobrerrelajación 0<w<2 que modifica el incremento en cada paso según la fórmula

$$x_{k+1} = x_k - w\frac{f(x_k)}{f'(x_k)}$$

Repite el apartado (b) con este método y representa en una tabla el número de iteraciones necesarias para w = 0.2, 0.4, 0.6,.....,1.6, 1.8.

9. El método de Halley es una forma de acelerar la convergencia del método de Newton-Raphson. La fórmula iterativa de Halley es:

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

A partir de $f(x) = x^2 - A$, determina la función de iteración de Halley para hallar $\sqrt{A}$. Empieza con x_0 = -2.4. ¿Cuál es la velocidad de convergencia del método en este caso? Compara estos resultados con los obtenidos aplicando otros métodos: Bisección, Regula Falsi, Punto Fijo y Newton.

10. Para el estudio del movimiento de cuerpos celestes con órbitas elípticas es necesario resolver la llamada ecuación de Kepler:

$$E - e\,\text{sen}\,E = M$$

Donde la incógnita es la anomalía excéntrica E y son conocidas la excentricidad de la órbita e = 0.96727464 y la anomalía media M = 4.527594 10^{-3} .

a) Dibuja la función F(E) = E – e sen(E) –M, y obtén una aproximación de la solución.

b) Resuelve la ecuación de Kepler mediante el método de Punto Fijo con valor inicial E_0 = 1 y tolerancia TOL = 0.00005. ¿Cuál es el valor de la anomalía excéntrica?. ¿Cuántas iteraciones han sido necesarias?. ¿Cuál es la estimación del error cometido?

c) Analiza las condiciones de convergencia del método de Newton en este caso y su velocidad de convergencia.

d) Resuelve la ecuación de Kepler mediante el método de la secante bajo las mismas condiciones que en el apartado (b) y con las estimaciones iniciales E = 0.5 y E = 1. ¿Cuál es el valor de la anomalía excéntrica?. ¿Cuántas iteraciones han sido necesarias?

e) Compara los resultados obtenidos.

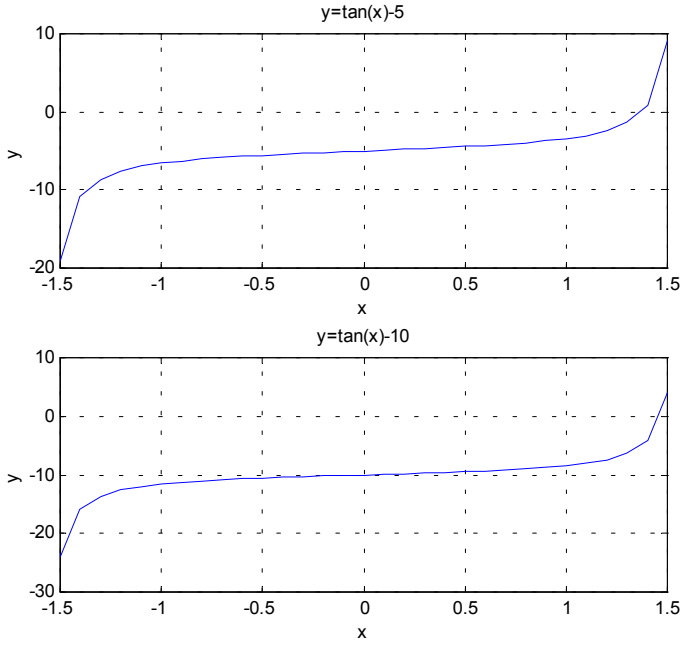
12.6 Soluciones

Problema 1

En primer lugar, obtenemos la representación gráfica de ambas funciones:

```
x=-1.5:.1:1.5;y1=tan(x)-5;
subplot(2,1,1),plot(x,y1)
y2=tan(x)-10;
```

```
subplot(2,1,2),plot(x,y2)
```



Podemos concluir de las gráficas anteriores que las raíces se encuentran en el intervalo [1,1.5].

Almacenamos la función $tan(x)-c=0$ en un fichero de función:

```
  function y=f1(x,c)
y=tan(x)-c;
```

Modificamos el algoritmo de bisección para introducir el valor del parámetro c como variable de entrada. Al aplicar este método sobre el intervalo [1,2] con una tolerancia de 10^{-4}, obtenemos los siguientes resultados para c = 5:

```
[sol,x,iter]=bisec1('f1',1,1.5,1e-4,100,5)
sol =    1.3734
iter =    13
```

y para c = 10:

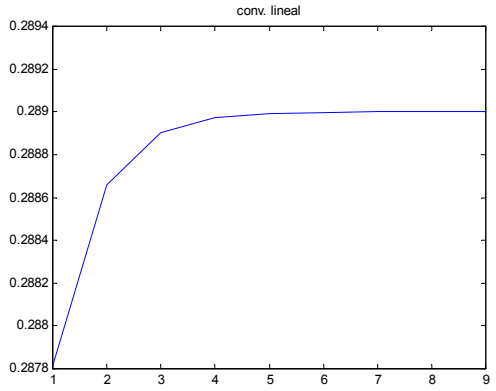
```
    [sol,x,iter]=bisec1('f1',1,1.5,1e-4,100,10)
    sol =    1.4711
    iter =    13
```

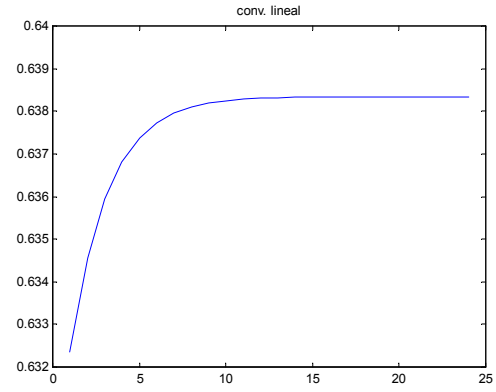
y mediante regula falsi, con idéntica estimación del error:

```
[sol,iter]=regula1('f1',1,1.5,1e-4,100,5)
sol =    1.3734
iter =    26

    [sol,iter]=regula1('f1',1,1.5,1e-4,100,10)
    sol =    1.4711
    iter =    11
```

Observamos que, aunque el método de bisección necesita el mismo número de iteraciones para resolver la ecuación $tan(x)-c=0$ con c = 5 o c=10, el método de regula falsi muestra una convergencia más lenta en el primer caso (justo el doble que las

que necesita el método de bisección) y más rápida en el segundo. Al estudiar la velocidad de convergencia de ambos métodos, recordamos que en el caso del método de bisección ésta es siempre lineal de tasa 0.5, mientras que podemos observar en las siguientes gráficas cómo la convergencia de regula falsi es lineal en ambos casos, aunque con tasa 0.29 para c = 10 y una tasa mayor, aproximadamente 0.64 para c = 5.





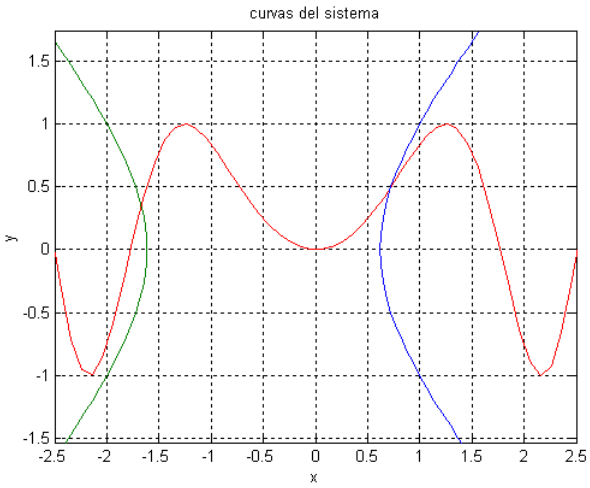
Sin embargo, estos resultados parecen contradecir la afirmación del enunciado en cuanto a la dificultad de los métodos numéricos para aproximar las raíces de la función *f* para valores grandes de c.

Problema 6

Representaremos en primer lugar las curvas que definen el sistema en el plano, para lo que obtenemos en primer lugar la forma explícita de ambas:

$$y = \pm\sqrt{x^2 + x - 1}$$

$$y = \sin\left(x^2\right)$$



Observemos que únicamente hay dos puntos de intersección entre ambas curvas, cuyas aproximaciones obtenidas mediante el comando "ginput" de MATLAB son: (-1.66,0.27) y (0.76,0.5).

A continuación, transformamos el sistema en una ecuación no lineal:

$$x^2 + x - \text{sen}^2\left(x^2\right) = 1$$

y la transformamos en la ecuación de punto fijo equivalente:

$$x = 1 - x^2 + \text{sen}^2\left(x^2\right)$$

Generamos el fichero de función:

```
function y=g6(x)
y=1-x.^2+sin(x.^2).^2;
```

en el que almacenamos la función de punto fijo $g\left(x\right) = 1 - x^2 + \text{sen}^2\left(x^2\right)$.

Aplicamos el método de punto fijo con las aproximaciones iniciales -1.6 y 0.7 obtenidas gráficamente y la tolerancia exigida en el enunciado del problema:

```
[sol, incr, iter] = pfijo('g6',-1.6,1e-6,100)

sol =   0.72595082969515
incr =    6.497037460251320e-007
iter =    11
[sol, incr, iter] = pfijo('g6',0.7,1e-6,100)
sol =   0.72595067846056
incr =    2.991704356469782e-007
iter =     8
```

Por tanto, la primera solución del sistema es:

```
x=sol,y=sin(sol^2)

x =   0.72595067846056
y =   0.50294644122539
```

Observamos cómo, partiendo de aproximaciones inicales próximas a cada una de las raíces, el método de punto fijo converge a una única raíz. La razón que explica este comportamiento es simple: la derivada de la función de punto fijo en la raíz (en valor absoluto) no está acotada por la unidad, por lo que no se cumplen las condiciones de convergencia, y ésta no está asegurada.

Para encontrar la otra raíz emplearemos una ecuación de punto fijo equivalente:

$$x = -\sqrt{1 - x + \text{sen}^2\left(x^2\right)}$$

```
[sol, incr, iter] = pfijo('g6b',-1.6,1e-6,100)
sol =  -1.67009279255709
incr =    9.352586394228979e-007
iter =    12
```

Por tanto, la solución del sistema que buscábamos es:

```
x=sol,y=sin(sol^2)

x =  -1.67009279255709
y =   0.34513509208359
```

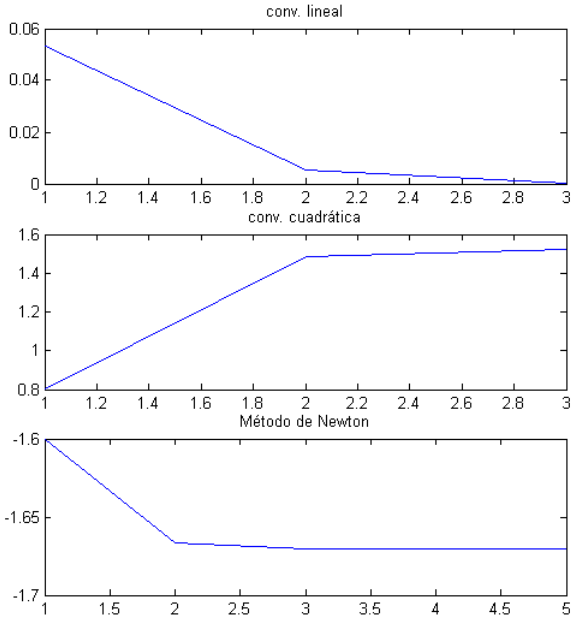
Si resolvemos ahora la ecuación no lineal mediante el método de Newton, necesitaremos una función que nos permita evaluar en un punto, tanto la función $f(x)$, como su derivada:

```
function [y,dy]=f6(x)
y=x.^2+x-sin(x.^2).^2;
dy=-2*x+4*x.*sin(x.^2).*cos(x.^2)
```

Ejecutamos nuestro algoritmo de Newton con las mismas estimaciones iniciales empleadas anteriormente. Analizamos en cada caso la velocidad de convergencia:

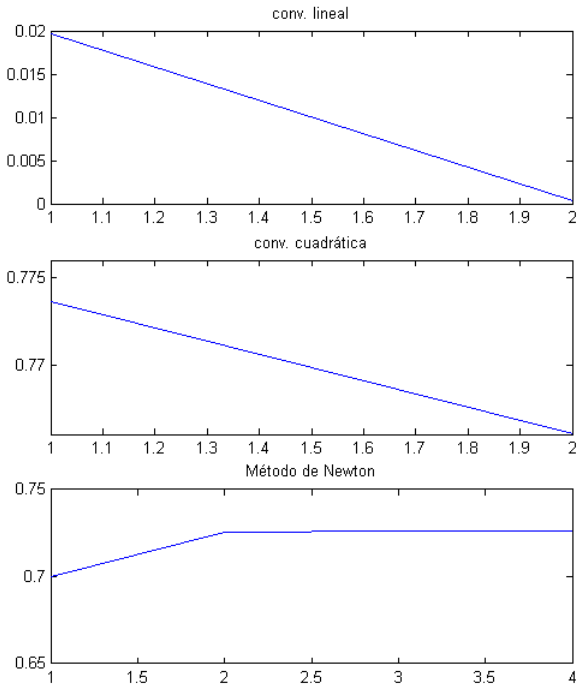
```
[sol, incr, iter] = newton1var('f6',-1.6,1e-6,100)
sol =  -1.67009303424438
incr =    5.340967319075955e-010
iter =     4
```

En las siguientes gráficas podemos observar la convergencia superlineal del método de Newton en este caso (la tasa de convergencia cuadrática no está acotada por debajo de la unidad, mientras que la tasa lineal tiende a cero), junto con la evolución de los iterados en el proceso. Es fácil comprobar que f' se anula en las proximidades de la raíz, lo que ralentiza la convergencia del método.



Al aplicar el método de Newton con la segunda estimación inicial obtenemos la solución en tan sólo 3 iteraciones y con una velocidad de convergencia cuadrática, ya que esta tasa de convergencia tiende a 0.765, como puede observarse en las gráficas, junto con la evolución de la tasa lineal y de los iterados.

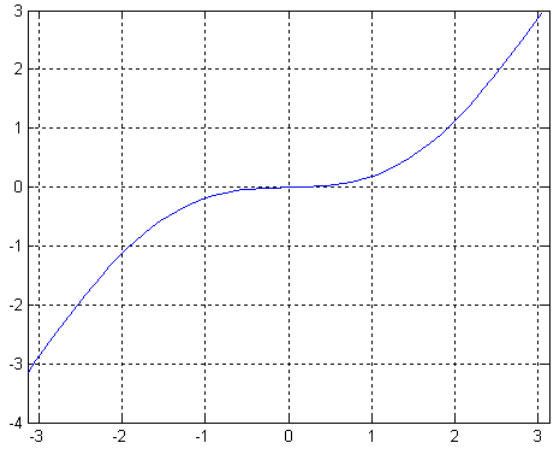
```
[sol, incr, iter] = newton1var('f6',0.7,1e-6,100)
sol =   0.72595072614316
incr =    1.923265868745633e-007
iter =     3
```



Problema 10

Conocidas la excentricidad de la órbita e = 0.96727464 y la anomalía media M = 4.527594 10^{-3}, obtenendremos la representación gráfica de la función:

```
e=0.96727464 ;
M = 4.527594e-3;
E=-pi:.1:pi;
FE=E-e*sin(E)-M;
plot(E,FE)
```



Observamos en la gráfica que el valor que buscamos de la anomalía excéntrica es aproximadamente nula. Si ampliamos la imagen observamos que $E \cong 0.1$.
Para aplicar el método de punto fijo, obtenemos la ecuación equivalente:

$$E = M + e\,\mathrm{sen}\,E$$

Empleando la función:

```
function E2=g10(E)

e = 0.96727464;
M = 4.527594e-3;

E2=M+e*sin(E);
```

y bajo las condiciones del enunciado, la solución obtenida es:

```
[E, incr, iter] = pfijo('g10',1,5e-5,500)
E =   0.12915859636994
incr =    4.819453403098750e-005
iter =   131
```

Si aplicamos el método de Newton, necesitamos almacenar función y derivada en el fichero:

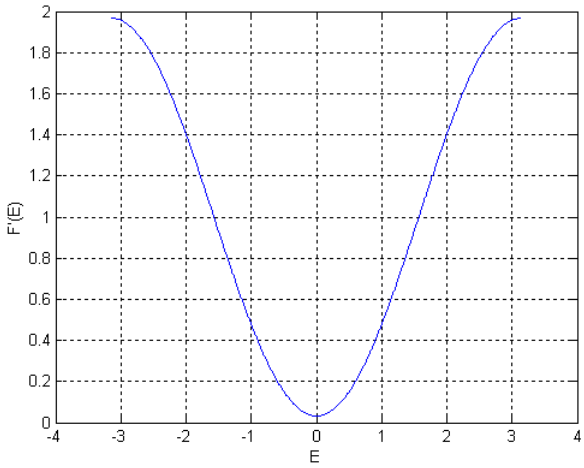
```
function [y,dy]=f10(E)

e = 0.96727464;
M = 4.527594e-3;

y=-M+E-e*sin(E);
dy=1-e*cos(E);
```

y la solución obtenida por Newton en este caso es:

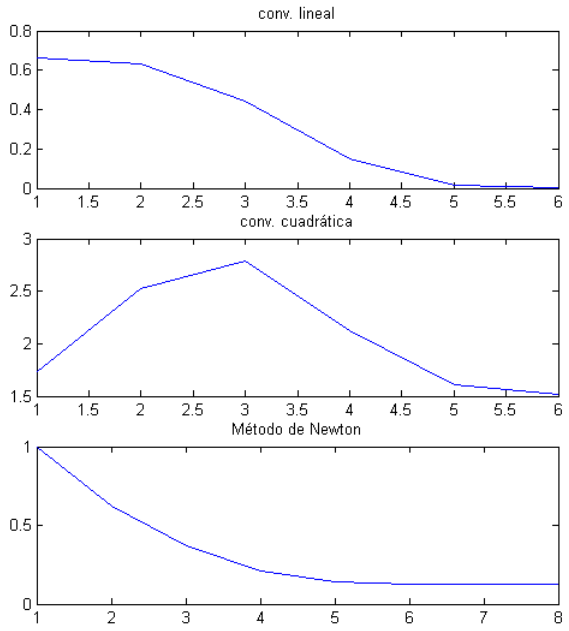
```
[E, incr, iter] = newton1var('f10',1,5e-5,100)
E =   0.12802307540635
incr =    4.924433887513424e-008
iter =     7
```

Respecto a las condiciones de convergencia del método, notemos que no hay problemas en cuanto a la continuidad y derivabilidad de la función, su derivada toma valores próximos a cero para una anomalía excéntrica nula, como se observa en la siguiente gráfica:

```
x=-pi:.01:pi;[y,dy]=f10(x);plot(x,dy)
```

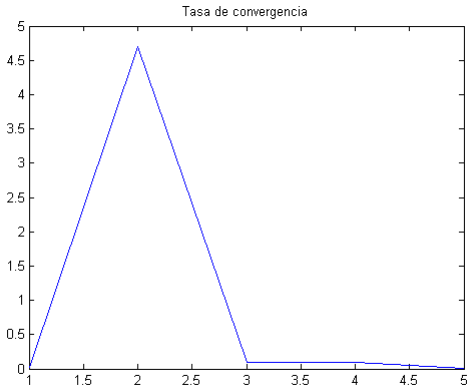


Por esta razón, la velocidad de convergencia no es cuadrática, sino superlineal, como se desprende de las siguientes gráficas:



Aplicando el método de la secante, obtenemos la solución en tan sólo una iteración más que empleando el método de Newton y con velocidad de convergencia superlineal:

```
[E, incr, iter] = secante('f10',0,1,5e-5,100)
E =0.12802309212324
incr =    1.182783157985257e-005
iter =     6
```



En este problema el método de la secante se muestra más eficaz, ya que con una velocidad superlineal (al igual que Newton), consigue encontrar la solución en una iteración menos con un algoritmo más simple, que no implica el conocimiento de la derivada de la función. El comportamiento del método de punto fijo en este caso es muy poco efectivo, con un número de iteraciones comparativamente muy elevado y convergencia lineal.

--- aproximacion grafica

```
>> x = linspace(1,2,1000);
>> y=x.^3;
>> plot(x,y,'.')
>> grid
>> [X,Y] = ginput(1)

X =

    1.2599


Y =

    2.0000
```

--- metodo iterativo

```
>> a = 1; fa = a^3-2
fa =
    -1
>> b = 2; fb = b^3-2
fb =
     6
>> c = (a+b)/2; fc = c^3-2
fc =
    1.3750
>> b = c;fb = fc; %movemos el extremo derecho
>> c = (a+b)/2; fc = c^3-2
fc =
   -0.0469
>> a = c;fa = fc; %movemos el extremo izquierdo
>> c = (a+b)/2; fc = c^3-2
fc =
    0.5996
```

etc...

-------- biseccion.m

```
function c=biseccion(f,a,b,tol,maxiter)

fa = feval(f,a);
fb = feval(f,b);
k = 0;

if fa*fb>0
   disp('Intervalo inadecuado. No hay garantia de existencia de raices.')
else
   while b-a>tol & k<maxiter
      c = (a+b)/2;
      fc = feval(f,c);
      if fa*fc < 0 % Semiintervalo izquierdo
         b = c; fb=fc;
      elseif fc*fb<0 %Semiintervalo derecho
         a = c; fa = fc;
      elseif fc==0 %c es la solucion
         a=c;b=c;
      end
      k = k+1;
   end
end

if k == maxiter
   disp('Alcanzado maximo de iteraciones')
end
if b-a < tol
   disp('Alcanzado error menor que tol')
end
```

Solucion al problema del billar

```
>> x = linspace(0,2*pi,10000);
>> y = billar(x);
>> plot(x,y,'.')
>> grid
>>
>> c=biseccion('billar',1,2,1e-6,40)
```

```
Alcanzado error menor que tol
c =
   1.5708
>> c=biseccion('billar',3,4,1e-6,40)
Alcanzado error menor que tol
c =
   3.1416
>> c=biseccion('billar',4,5,1e-6,40)
Alcanzado error menor que tol
c =
   4.7124
>> c=biseccion('billar',-1,1,1e-6,40)
Alcanzado error menor que tol
c =
    0
>> c=biseccion('billar',2*pi-1,2*pi+1,1e-6,40)
Alcanzado error menor que tol
c =
   6.2832
```

Ejemplo utilizando variables globales

```
>> global P Q
>> P = [0.7 -0.7]; Q=[0.5 0];
>> z = linspace(0,2*pi);
>> plot (cos(z), sin(z), P(1), P(2), 'ro', Q(1), Q(2), 'go')
>> [sol,c]=regulaf('billar',2,3,1e-6,40)
Alcanzado error menor que tol
sol =
   2.6697
>> x=cos(sol); y = sin(sol);
>> hold on
>> plot(x,y,'*')
```

---

ejercicio

y = x.^10 - 1

solucion x=1

con tol = 1e-6

biseccion: 21 iteraciones
regulaf: 160 iteraciones

```
>> %metodo del PUNTO FIJO
-------------------------

>> format compact
>> x = 1;
>> x = sqrt(2/x); %hemos reorganizado la ecuacion x^3 - 2 = 0
>> x = sqrt(2/x)
x =
   1.1892
>> x = sqrt(2/x)
x =
   1.2968
>> x = sqrt(2/x)
x =
   1.2419
>> x = sqrt(2/x)

...

>> x = sqrt(2/x)
x =
   1.2599


>> %mejora obvia
---------------

>> x = 1;
>> k=1;
>> x(k+1) = sqrt(2/x(k)), k = k + 1;
x =
   1.0000    1.4142
>> x(k+1) = sqrt(2/x(k)), k = k + 1;
x =
   1.0000    1.4142    1.1892
>> x(k+1) = sqrt(2/x(k)), k = k + 1;
x =
   1.0000    1.4142    1.1892    1.2968
>> x(k+1) = sqrt(2/x(k)), k = k + 1;
x =
   1.0000    1.4142    1.1892    1.2968    1.2419
>> x(k+1) = sqrt(2/x(k)), k = k + 1;
```

```
x =
   1.0000    1.4142    1.1892    1.2968    1.2419    1.2691
```

Permite un mejor analisis cuantitativo

```
>> plot(x);
>> e = diff(x);
>> tasa_convergencia = e(2:end)./e(1:end-1)
tasa_convergencia =
  Columns 1 through 7
   -0.5432   -0.4784   -0.5108   -0.4946   -0.5027   -0.4986   -0.5007
  Columns 8 through 14
   -0.4997   -0.5002   -0.4999   -0.5000   -0.5000   -0.5000   -0.5000
  Columns 15 through 16
   -0.5000   -0.5000
>> %significa que g'(x*) = -1/2
```

Representacion grafica:

(Hay que arreglarlo un poco)
Lo que falla es construir la matriz para representar los puntos:
Los puntos a representar son:

(x1,x1)
(x1,x2)
(x2,x2)
(x2,x3)

(x3,x3)
(x3,x4)
(x4,x4)
(x4,x5)

(x5,x5)
(x5,x6)
....

seria facil haciendo coord = [coord x(k) x(k)] con k=1->n-1

```
                absc = [absc x(k) x(k+1)] con k=1->n-1



xcos = linspace(0,pi/2);
ycos = cos(xcos);
[x, solucion, tasa_convergencia] = puntofijo('cos',1,1e-6,100);

%construir coord y absc
      %pequeña modificacion, que el primer punto sea x1,0 para
      %que mole mas
      coord = [x(1) x(1)];
      absc = [0 x(2)];       %absc = [x(1) x(2)];
      k = 2;
      >> for k = 2:34
            coord = [coord x(k) x(k)];
            absc = [absc x(k) x(k+1)];
            k = k+1;
      end

>> plot(xcos,ycos,'g',coord,absc,'b')
```

Solucion al billar

```
>> global P Q
P = [0.6,0];Q = [-0.6,0];
[aprox, solucion, tasa_convergencia] = puntofijo('billarf',1,1e-6,100);
>> solucion

solucion =

   1.5708


>> %la otra solucion no se obtiene porque la derivada es mayor que cero
>> %se hace con billarf2, donde y = x - billar(x)
```

```
>> [aprox, solucion, tasa_convergencia] = puntofijo('billarf2',1,1e-6,100);
>> solucion

solucion =

 -3.7824e-008

>> %Asi se han obtenido AMBAS soluciones
```

14 Aproximación Mínimo-Cuadrática. Ajuste de datos discretos

La aproximación mínimo-cuadrática trata de obtener una solución aproximada a un problema lineal, óptima en el sentido de que haga mínima una expresión cuadrática bajo ciertas condiciones.

Uno de los problemas de este tipo más usuales consiste en ajustar funciones a un número finito de valores dados; concretamente, se trata de encontrar la función de cierta clase que "mejor" representa dichos valores. Como clase de referencia, podemos considerar los polinomios de grado no superior a n.

En otras ocasiones, intentamos aproximar una función dada mediante otra función "más simple", por ejemplo un polinomio o una combinación de funciones trigonométricas.

Un problema aparentemente distinto, pero que engloba a los anteriores, es resolver aproximadamente un sistema de ecuaciones lineales incompatible.

Matemáticamente, los problemas mencionados se traducen en hallar el punto de un subespacio más próximo a un punto dado de un espacio euclídeo. La solución teórica es la proyección ortogonal del punto dado sobre el subespacio.

La solución numérica requiere obtener de una base ortogonal de dicho subespacio, lo que se consigue mediante el algoritmo de factorización QR en el caso finito-dimensional. En los espacios de funciones (de dimensión infinita) se suele partir ya de una base ortogonal, para reducir el trabajo computacional y garantizar la estabilidad numérica de la solución.

En este tema tratamos del ajuste de datos discretos, o caso de dimensión finita y en los dos siguientes, el ajuste funcional en espacios de dimensión infinita.

14.1 Aproximación Polinómica Mínimo-Cuadrática

Dada una nube de puntos $\{(x_i, y_i),\ i = 1, 2, \ldots, m\}$, tratamos de hallar el polinomio de grado no superior a n, $P_n(x)$, que mejor se ajusta a la nube, en el sentido de minimizar el error cuadrático

$$E^2 = \sum_{i=1}^{m} \left(P_n(x_i) - y_i\right)^2 .$$

La expresión del error se ha escogido de modo que todos los sumandos sean positivos para evitar que los errores se compensen y para que la función resultante sea derivable, y así poder resolver el problema por métodos analíticos.

14.1.1 Regresión lineal

Ejemplo 1

Consideremos los datos de implantación de telefonía fija en España que se muestran en la tabla adjunta.

Año	Líneas /100 hab
1.988	28.1
1.989	30.0
1.990	32.3
1.991	34.6
1.992	35.3

1.993	36.4
1.994	37.5
1.995	38.5

Suponiendo que el número de líneas instaladas por cada 100 habitantes crece linealmente en el periodo considerado, queremos obtener la recta o polinomio de primer grado, $y = P_1(x) = a_1 x + a_2$ que mejor aproxime los valores de la tabla, concretamente, que minimice el error cuadrático dado por

$$E^2 = \sum_{i=1}^{m} (a_1 t_i + a_2 - y_i)^2$$

El problema puede resolverse analíticamente, considerando E^2 como función de dos variables a_1 y a_2, y hallando los extremos relativos. En este caso existe un único extremo que es el mínimo buscado. Para ello hacemos cero las derivadas parciales de E respecto a a_1 y a_2:

$$\left.\begin{aligned} \frac{\partial E^2}{\partial a_1} &= 2\sum_{i=1}^{m} t_i (a_1 x_i + a_2 - y_i) = 0 \\ \frac{\partial E^2}{\partial a_2} &= 2\sum_{i=1}^{m} (a_1 x_i + a_2 - y_i) = 0 \end{aligned}\right\}$$

Obtenemos así un sistema de dos ecuaciones lineales con dos incógnitas, llamadas *ecuaciones normales*, cuya solución nos da los coeficientes de la recta buscada, llamada *recta de regresión de Y sobre X*.

$$\left.\begin{aligned} (\Sigma_i x_i^2) a_1 + (\Sigma_i x_i) a_2 &= \Sigma_i x_i y_i \\ (\Sigma_i x_i) a_1 + m a_2 &= \Sigma_i y_i \end{aligned}\right\}$$

Efectuamos los cálculos con MATLAB:

```
x = (1988:1995)'
y = [28.1 30 32.3 34.6 35.3 36.4 37.5 38.5]'
m = length(x)
```

Representamos los datos mediante un diagrama de barras con

```
bar(x,y)
```

Calculamos los coeficientes de las ecuaciones normales usando `sum`, aunque posteriormente se obtendrán de modo más rápido.

```
N = [sum(x.^2), sum(x); sum(x), m]
```

```
N =
    31728620       15932
       15932           8
```

Obtenemos a continuación los términos independientes y resolvemos el sistema

```
c = [sum(x.*y); sum(y)]
p = N\c
```

```
c =
  1.0e+005 *
    5.4314
    0.0027
p =
```

```
  1.0e+003 *
    0.0015
   -2.8892
```

La solución proporciona los coeficientes de la recta de regresión que aproxima los datos tal como muestra la Figura 1.

```
xg = linspace(1987,1996);
yg = polyval(p,xg);
plot(x,y,'*',xg,yg)
```

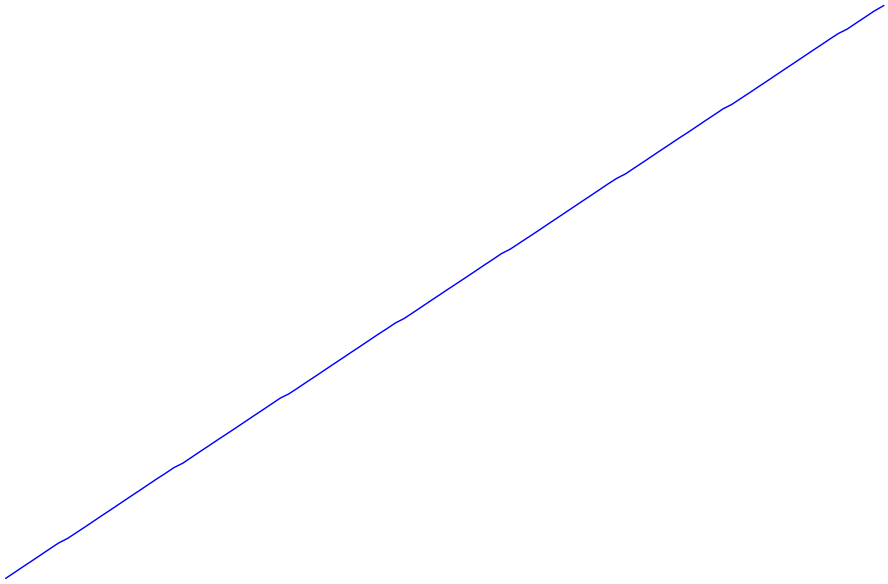


Figura 1. Recta de regresión

El error cuadrático se obtiene del modo siguiente:

```
E2 = norm(polyval(p,x)-y)^2

E2 =
    3.4554
```

Las ecuaciones normales pueden obtenerse por un procedimiento algebraico que simplifica las expresiones y es fácilmente generalizable. Consideremos la ecuación general de la recta $y = P_1(x) = a_1 x + a_2$ e intentemos determinar los coeficientes a_1 y a_2 imponiendo que los datos satisfagan dicha ecuación. Obtenemos así un sistema de ecuaciones lineales probablemente incompatible.

$$\begin{pmatrix} 1988 & 1 \\ 1989 & 1 \\ 1990 & 1 \\ 1991 & 1 \\ 1992 & 1 \\ 1993 & 1 \\ 1994 & 1 \\ 1995 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 28.1 \\ 30.0 \\ 32.3 \\ 34.6 \\ 35.3 \\ 36.4 \\ 37.5 \\ 38.5 \end{pmatrix}$$

Expresamos matricialmente este sistema como

$$(\boldsymbol{x} \quad \boldsymbol{u})\boldsymbol{p} = \boldsymbol{y}$$

donde $\boldsymbol{x}$ e $\boldsymbol{y}$ son vectores columna que contienen los datos, $\boldsymbol{u}$ es una columna de unos y $\boldsymbol{p}$ la columna de los coeficientes a determinar. Premultiplicando por la matriz transpuesta $(\boldsymbol{x} \ \boldsymbol{u})^T$ se obtiene

$$\begin{pmatrix} \boldsymbol{x}^T \\ \boldsymbol{u}^T \end{pmatrix}(\boldsymbol{x} \quad \boldsymbol{u})\boldsymbol{p} = \begin{pmatrix} \boldsymbol{x}^T \\ \boldsymbol{u}^T \end{pmatrix}\boldsymbol{y}$$

o sea

$$\begin{pmatrix} \boldsymbol{x}^T\boldsymbol{x} & \boldsymbol{x}^T\boldsymbol{u} \\ \boldsymbol{u}^T\boldsymbol{x} & \boldsymbol{u}^T\boldsymbol{u} \end{pmatrix}\boldsymbol{p} = \begin{pmatrix} \boldsymbol{x}^T\boldsymbol{y} \\ \boldsymbol{u}^T\boldsymbol{y} \end{pmatrix}$$

que coincide precisamente con el sistema de ecuaciones normales. Mientras el sistema inicial era incompatible, éste tiene solución única que proporciona la recta buscada.
Repetimos los cálculos en MATLAB:

```
A = [x, x.^0]
N = A'*A
c = A'*y
p = N\c

A =
        1988           1
        1989           1
        1990           1
        1991           1
        1992           1
        1993           1
        1994           1
        1995           1
N =
    31728620       15932
       15932           8
c =
  1.0e+005 *
    5.4314
    0.0027
p =
  1.0e+003 *
    0.0015
   -2.8892
```

14.1.2 Desplazamiento del origen

Las ecuaciones normales tienen, desde el punto de vista numérico, una ventaja y un inconveniente. La ventaja es que su matriz es simétrica definida positiva, lo que

garantiza que puede resolverse de modo estable sin necesidad de pivotación. El inconveniente es que esta matriz puede tener un número de condición alto si no se elige con cuidado la matriz original $\boldsymbol{A}$, como ocurre en el ejemplo.
Una solución obvia es modificar convenientemente la variable $\boldsymbol{x}$ para que sus valores sean moderados.
En el ejemplo anterior, si centramos las abscisas a su media, tomando $\boldsymbol{xn} = \boldsymbol{x} - 1991.5$, se observa una importante mejora en el número de condición de la matriz del sistema de ecuaciones normales $\boldsymbol{A}^T\boldsymbol{A}$.

```
cond(N)                 % Condición del sistema inicial
xn = x - mean(x)        % Desplazo el origen a la media de x
A = [xn xn.^0]          % Condiciones de ajuste
N = A'*A                % El sistema nuevo
cond(N)                 % tiene mejor condición
```

```
ans =
  2.9961e+012
xn =
   -3.5000
   -2.5000
   -1.5000
   -0.5000
    0.5000
    1.5000
    2.5000
    3.5000
A =
   -3.5000    1.0000
   -2.5000    1.0000
   -1.5000    1.0000
   -0.5000    1.0000
    0.5000    1.0000
    1.5000    1.0000
    2.5000    1.0000
    3.5000    1.0000
N =
    42     0
     0     8
ans =
    5.2500
```

Las nuevas ecuaciones normales son más estables que las anteriores. Su solución da los coeficientes de la recta de regresión desplazada al valor medio $y = b_1(x - \bar{x}) + b_2$.

```
c = A'*y
p = N\c
```

```
c =
   61.6500
  272.7000
p =
    1.4679
   34.0875
```

Para evaluar el polinomio hay que tener en cuenta que está referido a las nuevas abscisas. Sin embargo, en la representación gráfica utilizo las abscisas antiguas.

```
yp = polyval(p,xn)
plot(x,y,'*',x,yp)
```

14.1.3 Ajuste polinómico

Con la versión algebraica, es fácil generalizar las ecuaciones normales para ajustar por polinomios de grado mayor, en lugar de por rectas. Obtendremos así mejores aproximaciones, con menor error cuadrático.

Por ejemplo, para ajustar con un polinomio de grado 2,

$$P_2(x) = a_1 x^2 + a_2 x + a_3$$

basta añadir a la matriz A una columna con los cuadrados de las abscisas

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{x}^2 & \boldsymbol{x} & \boldsymbol{u} \end{pmatrix}$$

y efectuar las demás operaciones como en el caso de la recta de regresión. El sistema de ecuaciones normales es de tamaño 3×3, pero su condición es muy elevada, por lo que trabajaremos con las abscisas desplazadas.

Realicemos las operaciones con MATLAB.

```
A = [xn.^2, xn, xn.^0]
N = A'*A            % Ecuaciones normales
c = A'*y            % Términos independientes
p = N\c             % Coeficientes del polinomio
yp = polyval(p,xn) % Valor del polinomio en xn
E2 = norm(y-yp)^2  % Error cuadrático
```

```
A =
   12.2500   -3.5000    1.0000
    6.2500   -2.5000    1.0000
    2.2500   -1.5000    1.0000
    0.2500   -0.5000    1.0000
    0.2500    0.5000    1.0000
    2.2500    1.5000    1.0000
    6.2500    2.5000    1.0000
   12.2500    3.5000    1.0000
N =
  388.5000         0   42.0000
         0   42.0000         0
   42.0000         0    8.0000
c =
  1.0e+003 *
    1.4098
    0.0616
    0.2727
p =
   -0.1304
    1.4679
   34.7719
yp =
   28.0375
   30.2875
   32.2768
   34.0054
   35.4732
   36.6804
   37.6268
   38.3125
E2 =
    0.6005
```

Notemos cómo el error se ha reducido considerablemente. Para representar gráficamente el polinomio obtenido, lo evaluamos en el rango `xg`, previo desplazamiento a la media de `x`.

```
xgn = xg-mean(x);
yg = polyval(p,xgn);
plot(x, y, '*', xg, yg, x, yp, '+')
```

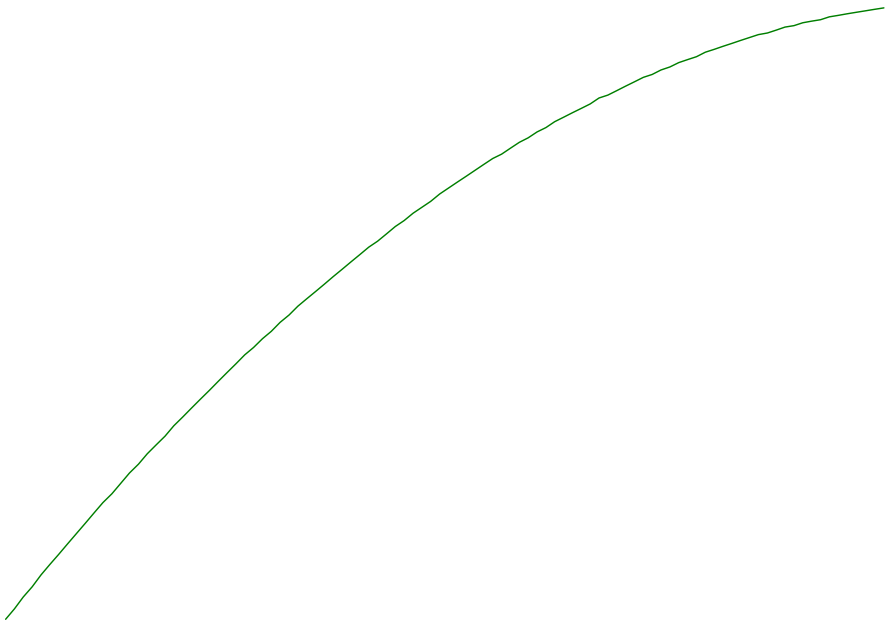


Figura 2: Ajuste polinómico de segundo grado

El ajuste por rectas y parábolas se generaliza de modo directo a polinomios de cualquier grado. En general suponemos que el grado del polinomio, n, es mucho menor que el número de puntos a interpolar, m; aunque, en teoría, el grado puede llegar a ser m–1, con lo que se obtiene la interpolación polinómica normal como caso particular de la mínimo cuadrática.

Imponiendo que el polinomio

$$P_n(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

pase por los puntos $\{(x_i, y_i),\ i = 1, 2, \ldots, m\}$, queda un sistema lineal $m\times(n+1)$ incompatible:

$$\begin{pmatrix} x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ x_2^n & x_2^{n-1} & \cdots & x_2 & 1 \\ x_3^n & x_3^{n-1} & \cdots & x_3 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ \vdots & \vdots & & \vdots & \vdots \\ x_m^n & x_m^{n-1} & \cdots & x_m & 1 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \\ a_{n+1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ y_m \end{pmatrix}$$

A partir de este sistema se forman las ecuaciones normales y se resuelven para obtener el polinomio mínimo-cuadrático

14.1.4 Índice de determinación

Conforme aumenta el grado del polinomio, disminuye el error cuadrático del ajuste de un conjunto dado de puntos. Deseamos establecer un criterio para elegir un polinomio de grado no demasiado alto que aproxime los datos adecuadamente.
El error mínimo cuadrático no es buen indicador de la aproximación, pues depende del número de puntos y de las unidades de medida empleadas. Como alternativa definimos el *índice de determinación*,

$$I = \frac{\sum_{i=1}^{m} (P_n(x_i) - \bar{y})^2}{\sum_{i=1}^{m} (y_i - \bar{y})^2}$$

donde $\bar{y}$ es la media de las ordenadas.
El índice de determinación es una cantidad adimensional entre 0 y 1. Cuanto más próximo a 1, mejor es el ajuste. Dado un valor umbral entre 0 y 1, tomaremos el polinomio mínimo-cuadrático de menor grado cuyo índice de determinación supere dicho umbral.
La aproximación de grado 0 es el polinomio constante igual a la media de las ordenadas. Su índice de determinación es también 0. En el otro extremo, si exigimos que el índice sea muy próximo a 1, el grado llegara a ser el número de datos menos 1 con lo que obtendremos el polinomio de interpolación que pasa exactamente por todos los puntos. El índice valdrá 1 y el error cuadrático, 0.
Si el ajuste se hace para predecir la evolución futura del fenómeno, los polinomios de grado elevado no son fiables pues oscilan mucho fuera del intervalo.

Ejemplo 2

Calculemos el índice de determinación del ajuste de grado 2 de los datos anteriores.

```
ym = mean(y)
I = (norm(yp-ym)/norm(y-ym))^2
```

Obtenemos un valor de 0.9936. Si queremos una mejor aproximación, aumentamos el grado del polinomio a ajustar.

```
A = [xn.^3 A]
N = A'*A
c = A'*y
p = N\c
yp = polyval(p,xn)
I = (norm(yp-ym)/norm(y-ym))^2
```

Ahora el índice ha subido a 0.9944.

14.1.5 Ajuste polinómico con MATLAB

MATLAB obtiene el polinomio de aproximación por mínimos cuadrados mediante la orden `polyfit`. La sintaxis `p = polyfit(x,y,n)` proporciona los coeficientes del polinomio mínimo cuadrático de grado n determinado por la nube de puntos (x, y). El polinomio puede evaluarse con `polyval`.
Por ejemplo, `polyfit(xn,y,4)` proporciona el polinomio de grado 4 que mejor ajusta los datos (x, y). El uso de `polyfit` no evita los problemas de mal condicionamiento, por lo que conviene utilizar las abscisas desplazadas.
Alternativamente, podemos utilizar `polyfit` con la sintaxis siguiente, que además de centrar automáticamente las abscisas, las normaliza para que su desviación típica sea la unidad, dividiéndolas por dicha desviación.

```
[p,s,mu] = polyfit(x,y,4)
```

```
p =
    0.3989    0.1670   -1.6321    3.3380   34.9775
s =
        R: [5x5 double]
       df: 3
    normr: 0.5492
mu =
  1.0e+003 *
    1.9915
    0.0024
```

mu es un vector cuyas componentes mu(1) y mu(2) son, respectivamente, la media $\bar{x}$ y la desviación típica σ de $\boldsymbol{x}$. El polinomio $\boldsymbol{p}$ obtenido está referido a la nueva variable $u = \dfrac{x - \bar{x}}{\sigma}$, lo que ha de tenerse en cuenta para evaluarlo respecto a los datos iniciales.

```
ug = (xg-mu(1))/mu(2);
yg = polyval(p,ug);
plot(xg, yg, x, y, '*')
```

La normalización mejora el comportamiento numérico, tanto del proceso de obtención del polinomio, como del propio polinomio resultante.

14.2 Sistemas Sobredeterminados

Las condiciones de ajuste de un polinomio a una serie de datos originaban en el apartado anterior un sistema de ecuaciones lineales incompatible. Esto ocurre habitualmente cuando se trabaja con valores experimentales. Para reducir la influencia de los errores, se realizan medidas redundantes, dando lugar a ecuaciones incompatibles. Llamamos a estos sistemas *sobredeterminados*, en lugar de incompatibles, para enfatizar el hecho de que todas las ecuaciones contienen información igualmente válida (o con distinto grado de confianza) que utilizaremos en la determinación de la solución aproximada.

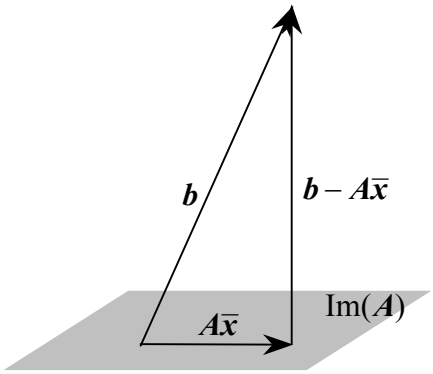
Sea

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

un sistema lineal posiblemente incompatible de tamaño $m \times n$ y de rango total por columnas, es decir, de rango n. La solución óptima $\bar{\boldsymbol{x}}$ es, por definición, la que minimiza el error cuadrático

$$E^2 = \left\| \boldsymbol{A}\bar{\boldsymbol{x}} - \boldsymbol{b} \right\|^2$$

$\boldsymbol{A}\bar{\boldsymbol{x}}$ es una combinación lineal de las columnas de $\boldsymbol{A}$, por tanto será un vector del espacio columna de $\boldsymbol{A}$, Im($\boldsymbol{A}$), precisamente el más próximo a $\boldsymbol{b}$. Como medimos la distancia mediante una norma euclídea, tal vector es la proyección ortogonal de $\boldsymbol{b}$ sobre el subespacio Im($\boldsymbol{A}$).



Así pues, $\boldsymbol{b} - \boldsymbol{A}\bar{\boldsymbol{x}}$ será ortogonal al espacio columna de $\boldsymbol{A}$, y por tanto, a las columnas de $\boldsymbol{A}$. Matricialmente esto se expresa como

$$\boldsymbol{A}^T (\boldsymbol{A}\bar{\boldsymbol{x}} - \boldsymbol{b}) = \boldsymbol{0}$$

de donde se obtienen las *ecuaciones normales*

$$\boldsymbol{A}^T \boldsymbol{A}\bar{\boldsymbol{x}} = \boldsymbol{A}^T \boldsymbol{b}\,.$$

La solución óptima se expresa formalmente como

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

aunque en la práctica resolvemos el sistema en lugar de calcular la inversa. La proyección ortogonal de $\boldsymbol{b}$ sobre el espacio columna de $\boldsymbol{A}$ es

$$\boldsymbol{p} = \boldsymbol{A}\bar{\boldsymbol{x}} = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}\ .$$

La matriz de la *proyección ortogonal*, $\boldsymbol{P}$, transforma directamente $\boldsymbol{b}$ en su proyección $\boldsymbol{p}$:

$$\boldsymbol{P} = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$$

Frecuantemente las ecuaciones normales resultan mal condicionadas, por lo que se requiere un tratamiento previo de la matriz $\boldsymbol{A}$ para poder efectuar los cálculos de modo estable.

Ejemplo 3

La tarifa del taxi se compone de una parte fija (bajada de bandera), otra parte proporcional a la distancia recorrida y otra proporcional al tiempo de espera. La tabla siguiente recoge los datos de distintas carreras.

Distancia	8.5	2.0	8.4	6.8	8.3	7.1	3.0	1.9	3.0
T. espera	10.5	13.4	0.4	7.6	10.1	8.6	3.8	13.6	10.8
Precio	10.5	6.6	7.7	8.5	10.2	9.0	4.7	6.6	6.6

El precio P de cada carrera viene dado por la expresión

$$P = a + bD + cT$$

donde D y T son la distancia y el tiempo de espera. Se trata de determinar los precios unitarios a,b y c, a partir de los datos de la tabla. Sustituyendo los datos de cada carrera en la expresión anterior se obtiene un sistema incompatible. Hallemos la solución mínimo-cuadrática con MATLAB.

La matriz del sistema consta de una columna de unos, que corresponden a la bajada de bandera, la columna de las distancias y la de los tiempos de espera:

```
U = ones(9,1);
D = [ 8.5;  2.0; 8.4; 6.8;  8.3; 7.1; 3.0;  1.9;  3.0];
T = [10.5; 13.4; 0.4; 7.6; 10.1; 8.6; 3.8; 13.6; 10.8];
A = [U D T]           % Matriz del sistema sobredeterminado
```

```
A =
    1.0000    8.5000   10.5000
    1.0000    2.0000   13.4000
    1.0000    8.4000    0.4000
    1.0000    6.8000    7.6000
    1.0000    8.3000   10.1000
    1.0000    7.1000    8.6000
    1.0000    3.0000    3.8000
    1.0000    1.9000   13.6000
    1.0000    3.0000   10.8000
```

El término independiente es la columna de los precios.

```
b = [10.5; 6.6; 7.7; 8.5; 10.2; 9.0; 4.7; 6.6; 6.6];
```

Premultiplicando por $\boldsymbol{A}^T$ se obtienen las ecuaciones normales.

```
N = A'*A
c = A'*b
```

```
N =
    9.0000   49.0000   78.8000
   49.0000  333.9600  385.6200
   78.8000  385.6200  839.7400
c =
```

```
   70.4000
  419.9300
  625.6900
```

Resolviendo obtenemos la estimación mínimo cuadrática de los factores de la tarifa.

```
x = N\c
```

```
x =
    1.5296
    0.7203
    0.2708
```

La bajada de bandera cuesta 1.53, el precio del Km es de 0.72 y el minuto de espera cuesta 0.271.
Con estos datos podemos estimar el precio de un recorrido cualquiera. Por ejemplo, una carrera de 5 Km con 18 min de espera costará `[1 5 18]*x` = 10.01€.

14.3 Factorización QR

El principal problema de las ecuaciones normales desde el punto de vista de la estabilidad numérica es la multiplicación de $\boldsymbol{A}$ por su transpuesta. Al hallar la recta de regresión, hemos visto que centrar las abscisas a su media mejora la condición de las ecuaciones normales. La matriz del sistema normal centrado es diagonal, lo que indica que las columnas de $\boldsymbol{A}$ son ortogonales.
Apliquemos la idea de ortogonalizar las columnas a la matriz de un sistema sobredeterminado con columnas independientes. Esto nos evitará efectuar productos de matrices y obtendremos la solución óptima resolviendo un sistema triangular.
Ortogonalizar las columnas de $\boldsymbol{A}_{m\times n}$ equivale a descomponerla en dos factores $\boldsymbol{Q}$ y $\boldsymbol{R}$. El primero está formado por las columnas de $\boldsymbol{A}$ ortogonalizadas y el segundo es una matriz $n\times n$ triangular superior invertible (si $\boldsymbol{A}$ es de rango n) que almacena las transformaciones efectuadas.
Al sustituir la factorización $\boldsymbol{A}=\boldsymbol{QR}$ en el sistema de ecuaciones normales

$$\boldsymbol{A}^T\boldsymbol{A}\bar{\boldsymbol{x}}=\boldsymbol{A}^T\boldsymbol{b}$$

queda

$$(\boldsymbol{R}^T\boldsymbol{Q}^T)(\boldsymbol{QR})\bar{\boldsymbol{x}}=\boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{b}$$

Como $\boldsymbol{Q}$ es ortogonal, el término $\boldsymbol{Q}^T\boldsymbol{Q}=\boldsymbol{I}$ desaparece, y al ser $\boldsymbol{R}$ invertible, simplificamos $\boldsymbol{R}^T$ en los dos miembros, quedando el sistema triangular

$$\boldsymbol{R}\bar{\boldsymbol{x}}=\boldsymbol{Q}^T\boldsymbol{b}$$

cuya solución es la solución óptima en sentido de mínimos cuadrados del sistema inicial.
Obtendremos la factorización QR mediante el algoritmo de Gram-Schmidt modificado. El coste de la factorización se compensa por la mejora de la estabilidad numérica en la resolución del sistema resultante.

14.3.1 Método de Gram-Schmidt modificado

Sea $\boldsymbol{A}$ una matriz $m\times n$ con columnas linealmente independientes. El método de Gram-Schmidt modificado obtiene una matriz $\boldsymbol{Q}$ de columnas ortonormales $(\boldsymbol{q}_1,\boldsymbol{q}_2,...,\boldsymbol{q}_n)$ y una matriz R triangular superior invertible tales que $\boldsymbol{A}=\boldsymbol{QR}$.
En cada paso del método se obtiene una columna más de la matriz $\boldsymbol{Q}$ y una fila de $\boldsymbol{R}$. En el paso k-ésimo, se obtiene la k-ésima columna de $\boldsymbol{Q}$ normalizando la correspondiente

columna de $\boldsymbol{A}$ y se hacen ortogonales a $\boldsymbol{q}_k$ las siguientes columnas de $\boldsymbol{A}$, sustrayéndoles su proyección en la dirección de $\boldsymbol{q}_k$. Los coeficientes de la proyección son los elementos de la fila k de la matriz $\boldsymbol{R}$. Como resultado, las $n-k$ últimas columnas de $\boldsymbol{A}$ son ortogonales a las k columnas ya obtenidas de $\boldsymbol{Q}$, puesto que en pasos anteriores se han eliminado las correspondientes proyecciones.
Podemos establecer formalmente el algoritmo como

Para $k = 1, 2, ..., n$

Normalizar a_k:

$$r_{kk} = \|\boldsymbol{a}_k\|$$

$$q_k = \boldsymbol{a}_k / r_{kk}$$

Para $j = k+1, ..., n$

Restar de a_j su proyección sobre q_k:

$$r_{kj} = \langle \boldsymbol{a}_j, \boldsymbol{q}_k \rangle$$

$$\boldsymbol{a}_j \leftarrow \boldsymbol{a}_j - r_{kj}\boldsymbol{q}_k$$

Fin j

Fin k

Ejercicio 1

Traduce este algoritmo a una función de MATLAB que operando columna a columna proporcione una factorización QR de la matriz $\boldsymbol{A}$.

14.3.2 Factorización QR en MATLAB

El algoritmo presentado en punto anterior, aunque es más estable numéricamente que la ortogonalización ordinaria de Gram-Schmidt, tiene el inconveniente de que los errores de redondeo se transmiten de un paso a otro, con lo que se puede perder la ortogonalidad. MATLAB obtiene la factorización QR mediante un proceso iterativo que trata de evitar este problema. La función correspondiente se denomina `qr`.
Veamos cómo se utiliza en el ejemplo del taxi.

```
[Q,R] = qr(A,0)
```

```
Q =
   -0.3333    0.3728    0.3370
   -0.3333   -0.4202    0.2192
   -0.3333    0.3606   -0.5842
   -0.3333    0.1654   -0.0254
   -0.3333    0.3484    0.2890
   -0.3333    0.2020    0.0828
   -0.3333   -0.2982   -0.5922
   -0.3333   -0.4324    0.2315
   -0.3333   -0.2982    0.0422
R =
   -3.0000  -16.3333  -26.2667
         0    8.1965   -5.2952
         0         0   11.0346
```

Comprobamos que $\boldsymbol{Q}$ tiene las columnas ortonormales ya que `Q'*Q` = I, que $\boldsymbol{R}$ es triangular y que `Q*R` es $\boldsymbol{A}$.
La solución óptima del sistema $\boldsymbol{Ax} = \boldsymbol{b}$ utilizando la descomposición QR se obtiene con

```
x = R\(Q'*b)
```

```
x =
    1.5296
```

```
    0.7203
    0.2708
```

14.3.3 Resolución de sistemas incompatibles con MATLAB

La contrabarra produce la solución mínimo cuadrática en el caso de sistemas lineales sobredeterminados.

```
x = A\b

x =
    1.5296
    0.7203
    0.2708
```

14.4 Modelos Lineales

Si los polinomios no proporcionan un ajuste adecuado de los datos, podemos recurrir a funciones cualesquiera, $\varphi_1(t), \varphi_2(t), \cdots, \varphi_n(t)$, con tal que sus valores en los puntos dados sean independientes.

Exigiendo que una combinación lineal genérica de estas funciones

$$g(x) = a_1\varphi_1(x) + a_2\varphi_2(x) + \cdots + a_n\varphi_n(x)$$

ajuste los puntos $\{(x_i, y_i),\ i = 1, 2, \ldots, m\}$, obtenemos un sistema de m ecuaciones y n incógnitas.

$$\begin{pmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_n(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \cdots & \varphi_n(x_2) \\ \varphi_1(x_3) & \varphi_2(x_3) & \cdots & \varphi_n(x_3) \\ \vdots & \vdots & \vdots & \\ \varphi_0(x_m) & \varphi_1(x_m) & \cdots & \varphi_n(x_m) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_m \end{pmatrix}$$

Si el sistema resulta incompatible, planteamos las ecuaciones normales y procedemos como en el caso del ajuste polinómico.

Ejemplo 4

Tenemos una mezcla de dos sustancias radiactivas con constantes de desintegración conocidas y queremos determinar las cantidades C y D de dichas sustancias. El número de desintegraciones y obedece teóricamente a la expresión

$$y = C\,\mathrm{e}^{t} + D\,\mathrm{e}^{2t}$$

donde t es el tiempo medido en unidades adecuadas. Las lecturas de un contador Geiger se resumen en la tabla siguiente:

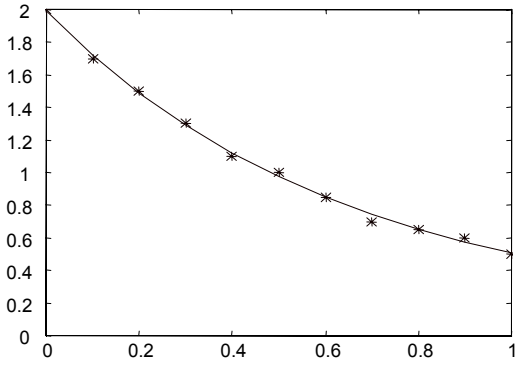
t	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	2	1.7	1.5	1.3	1.1	1	0.85	0.7	0.65	0.6	0.5

Estimemos C y D por mínimos cuadrados. Introducimos los datos en MATLAB:

```
t = (0:.1:1)';
y = [2; 1.7; 1.5; 1.3; 1.1; 1; 0.85;
0.7; 0.65; 0.6; 0.5];
```

Formamos las ecuaciones normales

```
A = [exp(-t), exp(-2*t)]
N = A'*A
```



```
c = A'*y

A =
    1.0000    1.0000
    0.9048    0.8187
    0.8187    0.6703
    0.7408    0.5488
    0.6703    0.4493
    0.6065    0.3679
    0.5488    0.3012
    0.4966    0.2466
    0.4493    0.2019
    0.4066    0.1653
    0.3679    0.1353
N =
    4.9054    3.7160
    3.7160    2.9960
c =
    8.6073
    6.6996
```

La solución mínimo cuadrática es

```
x = N\c

x =
    1.0043
    0.9906
```

Por tanto, la mezcla consta de partes iguales de ambas sustancias.
El error cuadrático indica la precisión del ajuste

```
E2 = norm(y-A*x)^2

E2 =
    0.0044
```

Evaluamos el modelo en los datos y representamos gráficamente ambos.

```
yp = A*x;
plot(t,y,'*',t,yp,'o',t,yp,':')
```

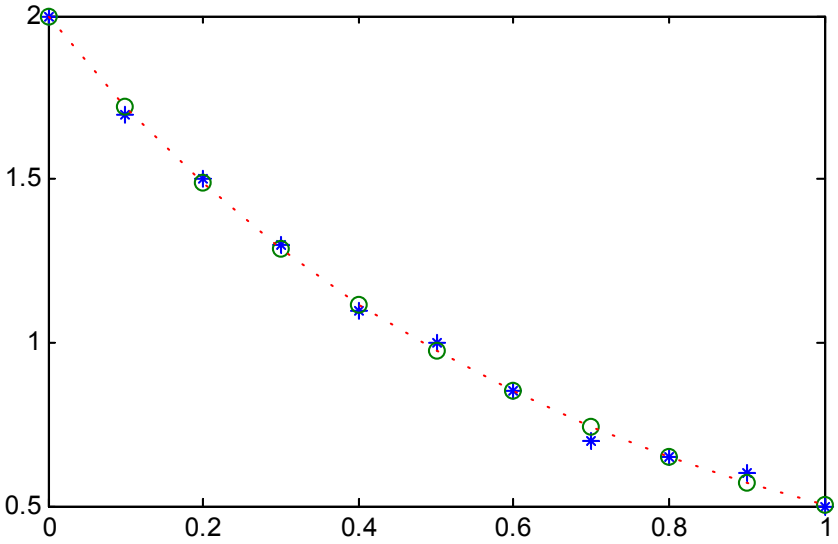


Figura 3: Modelo lineal exponencial

14.4.1 Linealización por cambio de variables

En ocasiones, el modelo propuesto para explicar un fenómeno no tiene inicialmente expresión lineal, pero no es difícil hallar una transformación que linealice el problema y aplicar el método de mínimos cuadrados lineales.

Ejemplo 5

La tabla muestra la evolución del número de abonados a teléfono móvil en España en determinado periodo.

Año	Abonados Teléfono Móvil
1.988	11.689
1.989	29.783
1.990	54.712
1.991	108.451
1.992	180.296
1.993	257.250
1.994	411.930
1.995	928.955

Suponiendo que el número de abonados crece exponencialmente, tratamos de ajustar una función del tipo

$$y = Ke^{at}$$

donde K y a son parámetros a determinar. Como la expresión no es lineal en estos parámetros, efectuamos la transformación

$$\log y = \log K + at$$

Cambiando a la variable $z = \log y$ y llamando b al $\log K$, queda un modelo lineal

$$z = at + b$$

cuyos parámetros a y b estimamos por el método de mínimos cuadrados. Efectuamos los cálculos con MATLAB.

Introducimos las abscisas en columna y las centramos

```
t = (1988:1995)';
tm = mean(t);
tn = t-tm;
```

En y introducimos los miles de abonados

```
y = [ 11.689  29.783  54.712 108.451 ...
     180.296 257.250 411.930 928.955]';
```

Efectuamos el cambio de variable

```
z = log(y);
```

La matriz del sistema lineal sobredeterminado es

```
A = [tn, tn.^0];
```

Planteamos las ecuaciones normales

```
N = A'*A
c = A'*z
```

```
N =
    42     0
     0     8
c =
   24.4573
   38.1405
```

La solución mínimo-cuadrática es

```
p = N\c

p =
    0.5823
    4.7676
```

que proporciona los valores de *a* y *b*. Representamos gráficamente el ajuste, deshaciendo los cambios de variable

```
tg = linspace(1988,1995);
yg = exp(polyval(p,tg-tm));
plot(t,y,'*',tg,yg)
```

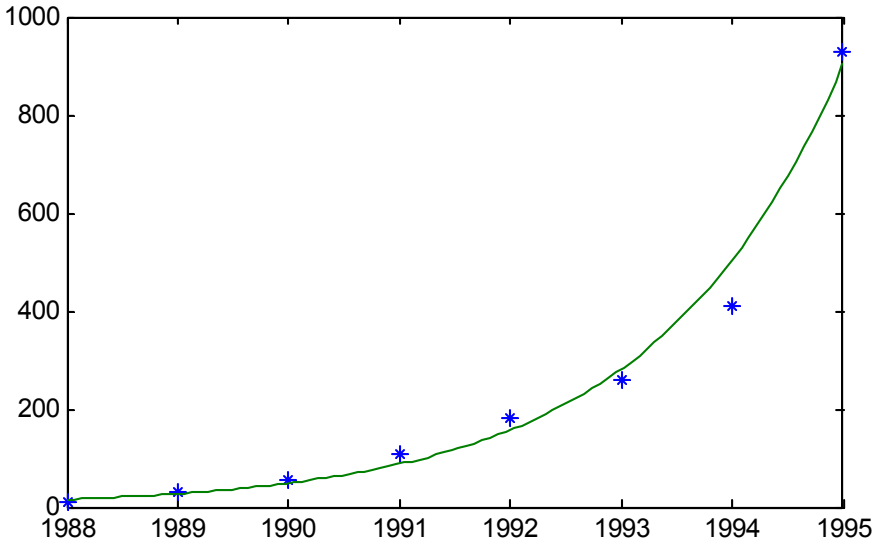


Figura 4. Ajuste exponencial

14.5 Problemas

Problema 1

El paro registrado en febrero de los últimos años, según datos del INEM se resume en la siguiente tabla

Año	Nº de parados
1991	2.362.373
1992	2.337.472
1993	2.471.228
1994	2.774.579
1995	2.575.873
1996	2.427.015
1997	2.262.721
1998	2.067.830
1999	1.783.940
2000	1.659.820
2001	1.598.920

a) Halla la recta de regresión, previa formación de las ecuaciones normales.
b) Ajusta un polinomio mínimo-cuadrático de grado 3. Compara el número de condición de las ecuaciones normales según se efectúe o no el centrado de las abscisas.
c) Halla el error cuadrático y el índice de determinación del polinomio de grado 3.
d) Estima el número de parados en febrero de 2002, de seguir esa tendencia.
e) Obtén el mismo polinomio con polyfit, de tres maneras: a partir de los datos iniciales, centrando las abscisas y centrando y escalando. Compara la magnitud relativa de los coeficientes. Genera gráficos con los polinomios obtenidos de cada manera.

Problema 2

La densidad del aire ρ (en g/m^3) varía con la altura h según la tabla siguiente

h	7	10	15	21	27	34	39	43	47	51	55	59	61
ρ	556	369	191	75	26.2	9.9	4.4	2.3	1.4	0.8	0.5	0.33	0.25

a) Halla el ajuste polinómico de menor grado con índice de determinación mayor que 0.99. ¿Cuál es el error cuadrático?
b) Dibuja los datos y el polinomio ajustado. Explica por qué este ajuste no resulta adecuado para alturas grandes.
c) Ajusta los datos a un modelo exponencial. Compara el error y el comportamiento del ajuste con el caso anterior.

Problema 3

La tabla siguiente muestra los valores de la temperatura ambiente medida a distintas horas del día.

Hora	6	8	10	12	14	16	18	20
Grados	7	9	12	18	21	19	15	10

Normaliza las abscisas y ajusta un polinomio mínimo-cuadrático de cuarto grado por los siguientes procedimientos:

a) Planteando un sistema lineal sobredeterminado y resolviendo directamente las ecuaciones normales.
b) Aplicando previamente la factorización QR según el algoritmo de Gram-Scmidt a la matriz del sistema.
c) Aplicando el algoritmo qr de MATLAB para obtener la factorización.
d) Resolviendo el sistema sobredeterminado mediante la contrabarra, \.
e) Calculando directamente el polinomio mediante polyfit.

Problema 4

La tabla adjunta muestra valores de una señal sinusoidal

$$y = K\sin(\omega t + \varphi)$$

de la que se conoce la frecuencia $\omega = 1$ y se quiere estimar la amplitud K y la fase φ. El cambio

$$P = K\cos(\varphi),\ \ Q = K\sin(\varphi)$$

convierte el problema en uno lineal:

$$y = P\sin(\omega t) + Q\cos(\omega t).$$

t	0	1.5	3	4.5	6
y	-0.0713	-2.1641	0.8403	2.2658	-0.5283

Estima los valores desconocidos, halla el error cuadrático y representa los datos y el ajuste.

Problema 5

La tabla adjunta muestra la distribución de los errores en la medida de 100 piezas salidas de una máquina.

Error	–1	–0.8	–0.6	–0.4	–0.2	0	0.2	0.4	0.6	0.8	1
Frec.	4	4	11	10	15	18	16	8	9	2	3

a) Suponiendo que la distribución es de la forma

$$y = \frac{1}{ax^2 + b},$$

donde x es el error e y la frecuencia, se trata de determinar los parámetros desconocidos a y b.

b) Ajusta una distribución del tipo

$$y = K\,\mathrm{e}^{-ax^2}$$

c) ¿Cuál de los dos modelos explica mejor los datos?

Problema 6

Considera los datos de matrícula en estudios universitarios de los últimos años, según datos del INE que se muestran en la siguiente tabla

Centros	Cursos							
	1993-94	1994-95	1995-96	1996-97	1997-98	1998-99	1999-00[1]	2000-01[1]
TOTAL	1.365.737	1.415.612	1.471.441	1.536.409	1.570.588	1.580.158	1.582.698	1.540.596
Licenciaturas	799.750	808.944	832.977	862.436	872.856	861.651	844.822	807.865
Arq. e Ingenierías	115.359	124.958	133.556	141.731	148.272	153.203	158.905	158.600
Diplomaturas	281.945	302.184	315.282	330.209	338.967	349.239	354.984	347.131
Arq. e Ingen. Técn.	168.683	179.526	189.626	202.033	210.493	216.065	223.987	227.000

a) Construye una tabla que estime los valores correspondientes al periodo 2001-2006 mediante aproximación mínimo cuadrática.
b) Justifica la elección del polinomio aproximante.
c) Representa gráficamente los resultados.

Problema 7

Genera una nube de puntos (x_i, y_i) dividiendo el intervalo $[0, a]$ en n partes iguales, siendo a>0 y n>5 dos números fijados por ti, y tomando $y_i = \mathrm{erf}(x_i)$.

a) Ajusta los puntos por polinomios de grado 0, 1, 2,... , $n-1$ y represéntalos junto a la función erf(x) en el intervalo $[0, a]$. Compara cada polinomio ajustado con la función erf(x) para bastantes puntos y observa la relación entre el error máximo y el grado del polinomio.
b) Como la función erf(x) es impar (erf($-x$) = –erf(x)), parece razonable aproximarla mediante un polinomio impar

$$\mathrm{erf}(x) \approx c_1 x + c_2 x^3 + \cdots + c_k x^{2k-1}$$

Analiza de nuevo cómo varía el error máximo en función del grado del polinomio.

c) Los polinomios no son muy adecuados para aproximar erf(x) porque no están acotados cuando x se hace grande, mientras que erf(x) tiende a 1 al tender x a infinito. Por ello, utilizando los mismos datos, ajusta un modelo de la forma

$$erf(x) \approx c_1 + e^{-x^2}(c_2 + c_3 z + c_4 z^2 + c_5 x^3)$$

donde $z = \dfrac{1}{1+x}$. Compara el error máximo con el de los ajustes polinómicos.

Problema 8

(Nota: en estas tablas, el punto indica millares)

Año	14.5.1 Abonados Teléfono Móvil
1.988	**11.689**
1.989	**29.783**
1.990	**54.712**
1.991	**108.451**
1.992	**180.296**
1.993	**257.250**
1.994	**411.930**
1.995	**928.955**

a) Ajusta un polinomio de grado 7 a los datos de la tabla anterior. Modifica previamente el origen de tiempos para mejorar el comportamiento numérico del problema. Representa gráficamente el polinomio obtenido.

b) Elige el menor grado del polinomio mínimo cuadrático que aproxime estos datos con la condición de que el índice de determinación sea mayor que 99%.

c) Compara las predicciones efectuadas mediante los polinomios anteriores con los datos de la tabla siguiente.

Año	14.5.2 Miles de Abonados Teléfono Móvil
1996	**2.988**
1997	**4.330**
1998	**7.051**
1999	**14.884**
2000	**24.266**
2001	**29.656**

d) Con toda la información de que dispones, elabora un informe para una operadora de telefonía móvil estimando la evolución del mercado en 2002 y 2003. Recuerda comprobar tus predicciones cuando se publiquen los datos reales.

14.6 Soluciones

Problema 1

a) Introducimos los datos en vectores columna:

```
x = (1991:2001)';
```

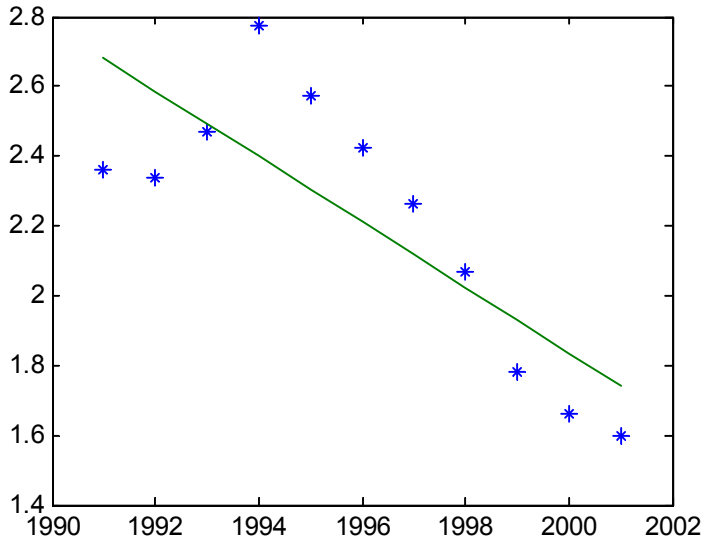
```
y = [2.362373; 2.337472; 2.471228; 2.774579; ...
     2.575873; 2.427015; 2.262721; 2.067830; ...
     1.783940; 1.659820; 1.598920];
```

Generamos las ecuaciones normales de la recta de regresión:

```
m = length(x);
N = [sum(x.^2), sum(x); sum(x), m]
c = [sum(x.*y); sum(y)]

N =
    43824286       21956
       21956          11
c =
  1.0e+004 *
    4.8536
    0.0024
```

La solución proporciona los coeficientes de la recta $P_1(x) = a_1 x + a_2$. La evaluamos en t y la representamos junto con los datos.

```
p = N\c
yp = polyval(p,x);
plot(x,y,'*',x,yp)
title('Recta de regresión')

p =
   -0.0938
  189.4066
```



b) Para determinar el ajuste mínimo-cuadrático de grado 3, construimos primero la matriz de van der Monde, obtenemos las ecuaciones normales, multiplicando por la transpuesta y las resolvemos.

```
A = [x.^3, x.^2, x, x.^0];
N = A'*A
c = A'*y
p = N\c

N =
```

```
  1.0e+020 *
    6.9562    0.0035    0.0000    0.0000
    0.0035    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000
c =
  1.0e+011 *
    1.9329
    0.0010
    0.0000
    0.0000
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 3.205179e-035.
(Type "warning off MATLAB:nearlySingularMatrix" to suppress this
warning.)
p =
  1.0e+005 *
    0.0000
   -0.0000
    0.0023
   -1.7791
```

El número de condición el sistema es muy elevado. El sistema es casi singular.

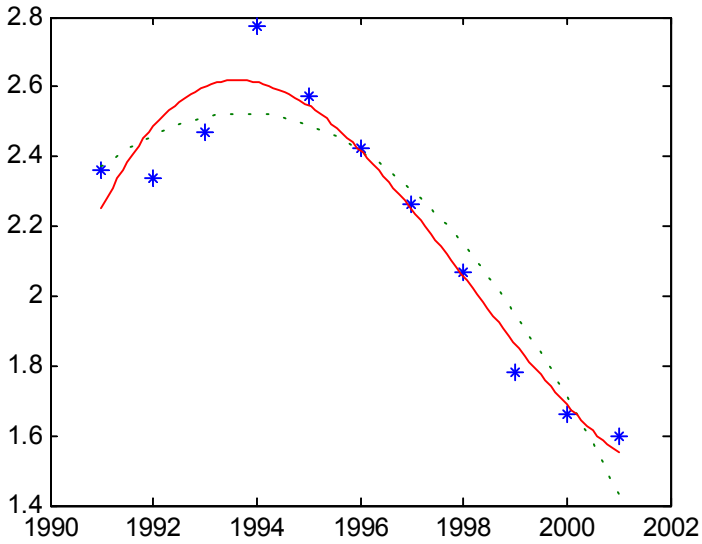
```
cond(N)

ans =
  1.1361e+035
```

Repretimos el cálculo centrando los datos

```
xm = mean(x);
xn = x-xm;
A = [xn.^3, xn.^2, xn, xn.^0];
N = A'*A
c = A'*y;
pn = N\c
cond(N)

N =
       41030           0        1958           0
           0        1958           0         110
        1958           0         110           0
           0         110           0          11
pn =
    0.0033
   -0.0207
   -0.1523
    2.4176
ans =
  8.5584e+003
```

Comparando gráficamente los resultados se aprecia que al ajustar con las abscisas iniciales se pierde el término de grado 3, debido a la diferencia relativa de los valores de los coeficientes del polinomio.

```
xg = linspace(1991,2001);
yg = polyval(p,xg);
xgn = xg-xm;
ygn = polyval(pn,xgn);
plot(x,y,'*',xg,yg,':',xg,ygn),
```



c) Error cuadrático: evaluamos el polinomio en las abscisas y lo comparamos con las ordenadas:

```
yp = polyval(pn,xn);
E2 = norm(y-yp)^2

E2 =
    0.0876
```

Índice de determinación: comparamos las desviaciones de las ordenadas calculadas y de las ordenadas de los datos respecto a su media:

```
ym = mean(y);
I = (norm(yp-ym)/norm(y-ym))^2

I =
    0.9411
```

d) El valor del polinomio en 2002 (previo desplazamiento) estima el número de parados de ese año.

```
polyval(pn,2002-xm)

ans =
    1.4702
```

e) Repetimos los cálculos con `polyfit`. Primero con las abscisas originales: MATLAB avisa de la necesidad de centrar los datos, aunque el resultado es más preciso que el obtenido a partir de las ecuaciones normales. Evaluamos el polinomio para comparar con los otros ajustes.

```
p = polyfit(x,y,3)
yg = polyval(p,xg);

Warning: Polynomial is badly conditioned. Remove repeated data points
         or try centering and scaling as described in HELP POLYFIT.
(Type "warning off MATLAB:polyfit:RepeatedPointsOrRescale" to suppress
this warning.)
> In C:\Matlab\toolbox\matlab\polyfun\polyfit.m at line 75
p =
  1.0e+007 *
```

```
    0.0000   -0.0000    0.0039   -2.6221
```

Los coeficientes del polinomio ajustado tienen escalas muy diferentes: de 10^{-4} a 10^{4}. Centrando las abscisas a la media evitamos el aviso de inestabilidad y obtenemos un polinomio cuyos coeficentes están mejor escalados:

```
p = polyfit(xn,y,3)
ygn = polyval(p,xgn);

p =
    0.0033   -0.0207   -0.1523    2.4176
```

La sintaxis más compleja de `polyfit` centra y normaliza las abscisas, proporcionando el polinomio con coeficientes más equilibrados.

```
[p,s,mu] = polyfit(x,y,3)
xgu = (xg-mu(1))/mu(2);
ygu = polyval(p,xgu);

p =
    0.1199   -0.2272   -0.5051    2.4176
s =
        R: [4x4 double]
       df: 7
    normr: 0.2960
mu =
  1.0e+003 *
    1.9960
    0.0033
```

Los valores de los tres ajustes son suficientemente aproximados para no ser gráficamente distinguibles. La diferencia entre los dos últimos es del orden de 10^{-15} y la de éstos con el primero, del orden de 10^{-8}.

```
plot(x,y,'*',xg,yg,xg,ygn,'+',xg,ygu,'o')
max(abs(ygn-yg))
max(abs(ygn-ygu))

ans =
  6.8915e-008
ans =
  2.8866e-015
```

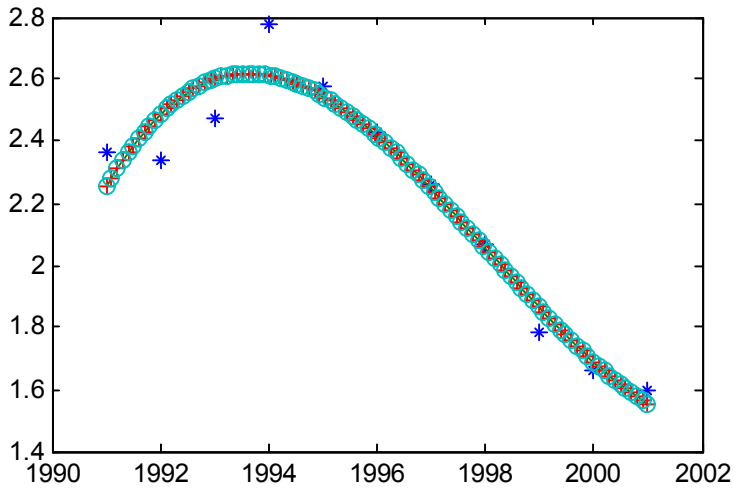


Figura 5 Ajustes de `polyfit` con distintas abscisas

Problema 2

a) Ajuste polinómico

Almacenamos los datos en sendos vectores

```
h = [7 10 15 21 27 34 39 43 47 51 55 59 61];
r = [556 369 191 75 26.2 9.9 4.4 2.3 1.4 0.8 0.5 0.33 0.25];
```

y probamos con un polinomio de tercer grado

```
p = polyfit(h,r,3)
p =   -0.0131    1.7017  -70.8466  941.9188
```

Para hallar el índice de determinación evaluamos el polinomio y comparamos con los datos:

```
rp =polyval(p,h);
rm = mean(r);
indice=(norm(rp-rm)/norm(r-rm))^2

indice = 0.9886
```

Aumentamos el grado y comprobamos que el polinomio pedido es el de grado 4.

```
p = polyfit(h,r,4)
rp =polyval(p,h);
rm = mean(r);
indice=(norm(rp-rm)/norm(r-rm))^2

p =  1.0e+003 *    0.0000   -0.0001    0.0039   -0.1097    1.1436
indice = 0.9990
```

El error cuadrático será

```
norm(rp-r)^2
ans = 359.1253
```

b) Evaluamos el polinomio para varias alturas y lo representamos junto a los datos

```
hg = 5:5:65;
rg = polyval(p,hg);
plot(h,r,'*',h,rp,'o',hg,rg)
```

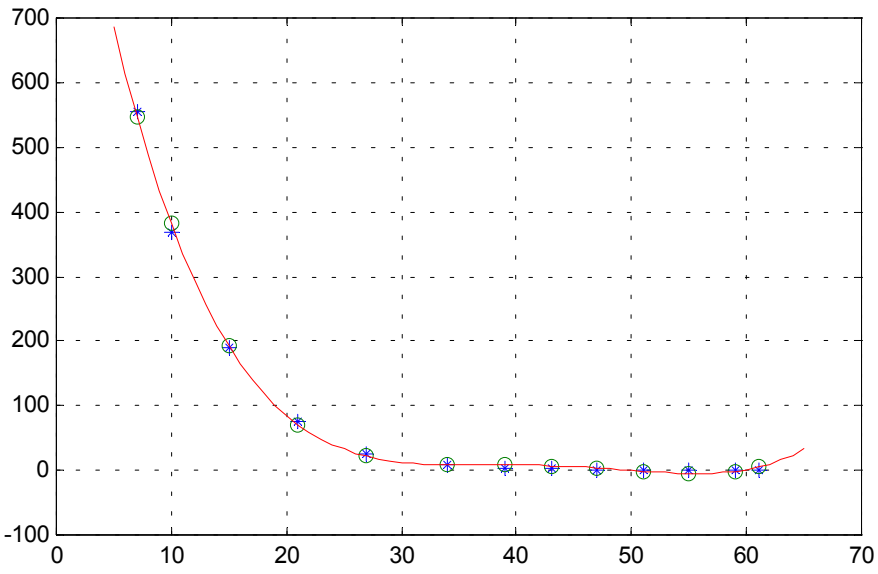


Figura 6: Ajuste polinómico

El ajuste no representa adecuadamente los datos porque para altura grande toma valores negativos y finalmente se hace creciente.

c) Proponemos un modelo del tipo

$$\rho = \mathrm{e}^{Ah+B}$$

que se linealiza tomando logaritmos:

$$\log \rho = Ah + B$$

Realizamos los cálculos con MATLAB ajustando un polinomio de grado 1 al logaritmo de la presión:

```
lr=log(r);
p=polyfit(h,lr,1)

p =
   -0.1459    7.3088
```

Evaluamos el polinomio en *h* para obtener el error cuadrático

```
ly=polyval(p,h);
y=exp(ly);
norm(r-y)^2

ans =
  1.3918e+003
```

El error cuadrático es mayor que el del ajuste polinómico, pero ahora se obtiene una función positiva decreciente, con menor error para valores grandes de *h*, como se aprecia en el gráfico siguiente.

```
lyg=polyval(p,hg);
yg=exp(lyg);
plot(h,r,'*',hg,yg)
```

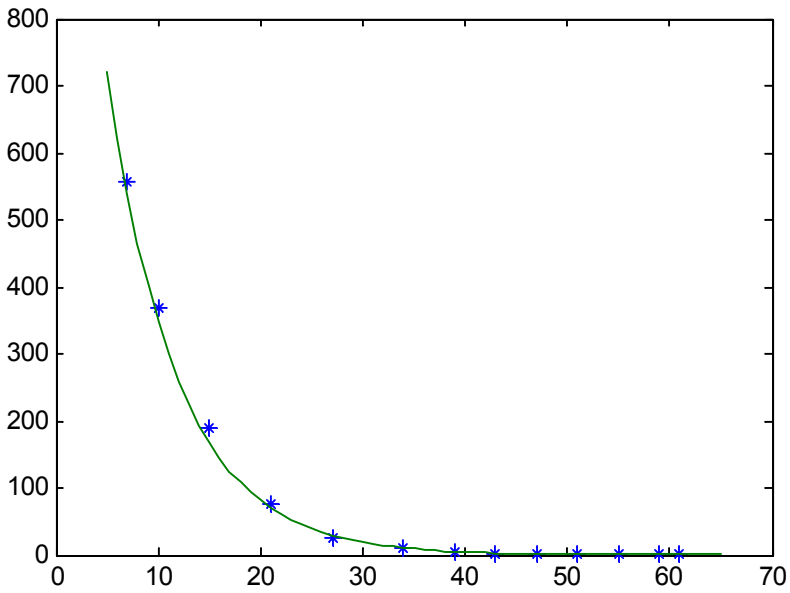


Figura 7: Ajuste exponencial

Problema 3

Introducimos los datos en columna:

```
t = [6; 8; 10; 12; 14; 16; 18; 20];
y = [7; 9; 12; 18; 21; 19; 15; 10];
```

Normalizamos las abscisas:

```
m = mean(t);
s = std(t);
x = (t-m)/s;
```

a) La matriz del sistema sobredeterminado es

```
A = [x.^4 x.^3 x.^2 x x.^0]
```

```
A =
    4.1684   -2.9173    2.0417   -1.4289    1.0000
    1.0851   -1.0631    1.0417   -1.0206    1.0000
    0.1406   -0.2296    0.3750   -0.6124    1.0000
    0.0017   -0.0085    0.0417   -0.2041    1.0000
    0.0017    0.0085    0.0417    0.2041    1.0000
    0.1406    0.2296    0.3750    0.6124    1.0000
    1.0851    1.0631    1.0417    1.0206    1.0000
    4.1684    2.9173    2.0417    1.4289    1.0000
```

La columna de términos independientes es el vector de temperaturas. Formamos las ecuaciones normales:

```
N = A'*A;
c = A'*y;
```

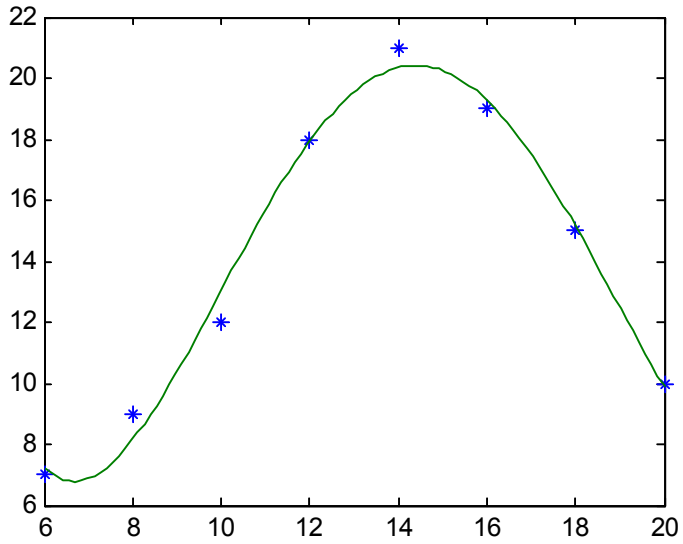
Resolviéndolas obtenemos los coeficientes del polinomio buscado:

```
p = N\c
```

```
p =
    2.2159
   -2.4866
   -9.9006
    6.0206
   19.5488
```

Representémoslo gráficamente:

```
xg = (tg-m)/s;
yg = polyval(p,xg);
plot(t,y,'*',tg,yg)
```



b) El algoritmo de Gram-Schmidt modificado se traduce en la siguiente función de MATLAB

```
function [Q,R]=miqr(A)
%
% Factorización de la matriz a en q ortogonal
% y r triangular superior invertible
% Las columnas de a deben ser independientes

Q = [];R = [];
[m,n] = size(A);
for j = 1:n
   R(j,j) = norm(A(:,j));
   Q(:,j) = A(:,j)/R(j,j);
   for k = j+1:n
      R(j,k) = dot(A(:,k),Q(:,j));
      A(:,k) = A(:,k)-R(j,k)*Q(:,j);
   end
end
```

Esta función descompone ***A*** en los factores

```
Q
R

Q =
    0.6839   -0.6626   -0.1632    0.1957    0.0423
    0.1780   -0.2415    0.5794   -0.4304   -0.1946
    0.0231   -0.0522    0.3676   -0.4863    0.2398
    0.0003   -0.0019    0.0497   -0.2001    0.6347
    0.0003    0.0019    0.0497    0.2001    0.6347
    0.0231    0.0522    0.3676    0.4863    0.2398
    0.1780    0.2415    0.5794    0.4304   -0.1946
    0.6839    0.6626   -0.1632   -0.1957    0.0423
R =
    6.0947         0    3.1810    0.0000    1.7707
         0    4.4031         0    2.4509         0
         0         0    0.8204         0    1.6670
         0         0         0    0.9965   -0.0000
         0         0         0         0    1.4443
```

La solución mínimo cuadrática del sistema sobredeterminado $\boldsymbol{Ax} = \boldsymbol{b}$ se obtiene resolviendo el sistema triangular $\boldsymbol{R\bar{x}} = \boldsymbol{Q}^T\boldsymbol{b}$.

```
pb = R\(Q'*y)
```

```
pb =
    2.2159
   -2.4866
   -9.9006
    6.0206
   19.5488
```

c) En lugar de Gram-Schmidt, factorizamos $\boldsymbol{A}$ por el algoritmo `qr` de MATLAB

```
A = [x.^4 x.^3 x.^2 x x.^0];
[Q,R] = qr(A,0)
```

```
Q =
   -0.6839   -0.6626    0.1632    0.1957   -0.0423
   -0.1780   -0.2415   -0.5794   -0.4304    0.1946
   -0.0231   -0.0522   -0.3676   -0.4863   -0.2398
   -0.0003   -0.0019   -0.0497   -0.2001   -0.6347
   -0.0003    0.0019   -0.0497    0.2001   -0.6347
   -0.0231    0.0522   -0.3676    0.4863   -0.2398
   -0.1780    0.2415   -0.5794    0.4304    0.1946
   -0.6839    0.6626    0.1632   -0.1957   -0.0423
R =
   -6.0947         0   -3.1810         0   -1.7707
         0    4.4031   -0.0000    2.4509    0.0000
         0         0   -0.8204         0   -1.6670
         0         0         0    0.9965    0.0000
         0         0         0         0   -1.4443
```

y la solución se obtiene igual

```
pc = R\(Q'*y)
```

```
pc =
    2.2159
   -2.4866
   -9.9006
    6.0206
   19.5488
```

d) El polinomio que resulta resolviendo el sistema sobredeterminado es

```
pd = A\y
```

```
pd =
    2.2159
   -2.4866
   -9.9006
    6.0206
   19.5488
```

e) Obtenemos ahora el polinomio directamente con `polyfit`

```
pe = polyfit(x,y,4)
```

```
pe =
    2.2159   -2.4866   -9.9006    6.0206   19.5488
```

Comprobamos que, con la precisión mostrada, el resultado es siempre el mismo. Los apartados c) y e) dan exactamente el mismo resultado, pues polyfit aplica qr a la matriz del sistema. Las diferencias en valores del polinomio entre los distintos apartados son del orden de 10^{-14}.

Problema 4

Determinamos M y N resolviendo por mínimos cuadrados el sistema obtenido al sustituir cada par de datos en la ecuación

$$y = M\sin(\omega t) + N\cos(\omega t).$$

Efectuamos los cálculos con MATLAB. Almacenamos los datos en columna:

```
t=(0:1.5:6)';
y=[-0.0713 -2.1641 0.8403 2.2658 -0.5283]';
```

Construimos la matriz del sistema y resolvemos las ecuaciones normales:

```
A=[sin(t) cos(t)];
N=A'*A;
c=A'*y;
x=N\c
```

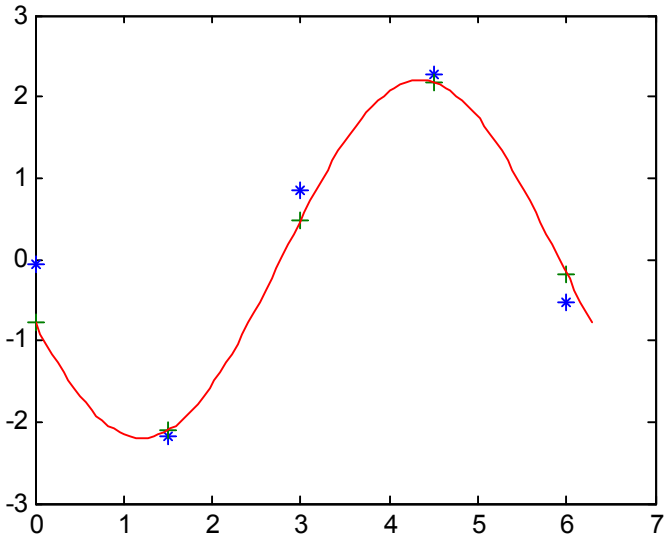
```
x =
   -2.0552
   -0.7831
```

Deshacemos el cambio de variables para obtener el módulo y la fase de la señal:

$$F = \sqrt{P^2 + Q^2}; \qquad \tan(\varphi) = \frac{Q}{P}$$

```
F = norm(x)
fi = atan2(x(2),x(1))
```

```
F =
    2.1994
fi =
   -2.7776
```

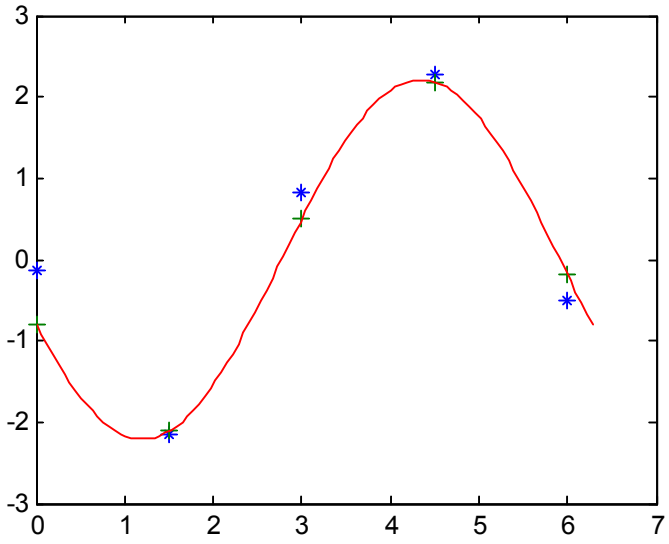


Evaluamos el resultado en los datos y calculamos el error:

```
yp = A*x;
E2 = norm(y-yp)^2

E2 =
    0.7675
```

Para hacer el gráfico, evaluamos el ajuste en un rango más detallado:

```
tg = linspace(0,2*pi)';
yg = F*sin(tg+fi);
plot(t,y,'*',t,yp,'+',tg,yg)
```



Problema 5

a) Editamos unos vectores columna con los datos:

```
x = (-1:0.2:1)';
y = [4 4 11 10 15 18 16 8 9 2 3]';
```

Invirtiendo ambos términos se obtiene un modelo lineal:

$$ax^2 + b = 1/y$$

El sistema sobredeterminado tiene matriz ***A*** y términos independientes 1/***y***.

```
A=[x.^2 x.^0];
c=1./y;
```

Lo resolvemos directamente con MATLAB:

```
p = A\c

p =
    0.2908
    0.0605
```

Hallamos el error cuadrático en términos de los datos originales

```
zp = A*p;
E2= norm(y-1./zp)^2

E2 =
   49.0025
```

Representamos gráficamente el ajuste:

```
xg = linspace(-1,1);
```

```
zg = 1./(p(1)*xg.^2+p(2));
plot(x,y,'*',xg,zg)
```

b) Linealizamos el modelo tomando logaritmos

$$\log y = \log K - ax^2$$

Resolvemos las ecuaciones normales:

```
A=[x.^0 -x.^2];
c=log(y);
p = A\c
```

```
p =
    2.6840
    1.7298
```

Hallamos el error cuadrático en términos de los datos originales, observando que es ligenrmente superior al del modelo anterior.

```
zp = A*p;
E2= norm(y-exp(zp))^2
```

```
E2 =
   51.4532
```

Representamos gráficamente el ajuste:

```
xg = linspace(-1,1);
zg = exp(p(1)-p(2)*xg.^2);
plot(x,y,'*',xg,zg)
```

Recta de Regresión

```
>> x = (1988:1995)';
>> y = [28.1 30 32.3 34.6 35.3 36.4 37.5 38.5]';
>> bar(x,y)

>> N = [sum(x.^2) sum(x); sum(x) length(x)];
>> c = [sum(x.*y); sum(y)];

>> p = N\c
p =
  1.0e+003 *
    0.0015   --> la pendiente
   -2.8892   --> corte con eje y
```

-- Representacion Gráfica: --

```
>> x_graf = linspace(1988, 1995);
>> y_graf = polyval(p,x_graf);
>> plot(x,y,'.',x_graf,y_graf)
```

-- Error Cuadrático: --

```
>> yp = polyval(p,x);
>> E2 = norm(y-yp)^2
E2 =
    3.4554
```

(NOTA:
%norm hace la raiz de la suma de los cuadrados
NORM(V,P) = sum(abs(V).^P)^(1/P).
NORM(V) = norm(V,2).
)

-- Jiustificacion utilizando la premultiplicacion por la traspuesta: --

```
>> A = [x x.^0]
A =
       1988          1
       1989          1
       1990          1
```

```
      1991     1
      1992     1
      1993     1
      1994     1
      1995     1

>> N = A'*A
N =
   31728620      15932
      15932          8

>> c=A'*y
c =
  1.0e+005 *
    5.4314
    0.0027


>> p = N\c
```

Ajuste con polinomio de mayor grado:

```
>> A = [x.^2 x x.^0];
>> N = A'*A;
>> c = A'*y;
>> hold on
>> p = N\c;
>> y_graf = polyval(p,x_graf);
>> plot(x,y,'.',x_graf,y_graf)
```

-- Error cuadrático: --

```
>> yp = polyval(p,x);
E2 = norm(y-yp)^2
E2 =
    0.6005
```

Es un error cuadrático bastante menor

-- Reduccion del condicionamiento --

```
>> xn = x - mean(x)
xn =
   -3.5000
   -2.5000
   -1.5000
   -0.5000
    0.5000
    1.5000
    2.5000
    3.5000
>> A = [xn.^2 xn xn.^0];
>> N = A'*A;
>> c = A'*y;
>> p = N\c;


>> y_graf = polyval(p,x_graf-mean(x));
>> plot(x,y,'.',x_graf,y_graf,'g')
```

-- Mayor grado --

```
>> A = [xn.^3 xn.^2 xn xn.^0];
>> c = A'*y;
>> N = A'*A;
>> p = N\c;


>> y_graf = polyval(p,x_graf-mean(x));
>> plot(x,y,'.',x_graf,y_graf,'g')
>> yp = polyval(p,x);
```

-- Error cuadratico --

```
>> yp = polyval(p,xn);
>> E2 = norm(y-yp)^2
E2 =
    0.5238
```

-- Indice de determinacion --

```
>> I=norm((yp-mean(y))/norm(y-mean(y))).^2
I =
    0.9944
```

--- Factorizacion QR --

```
 [Q,R] = qr(A)
>> p = R\(Q'*y)
p =
    0.0114
   -0.1304
    1.3627
   34.7719
```

% Utilizando la 'magnificiencia' de Matlab xD

```
>> A\y
ans =
    0.0114
   -0.1304
    1.3627
   34.7719
```

y = [11.689 29.783 54.712 108.451 180.296 257.250 411.930 928.955]'

Linealizacion de una exponencial

```
>> y_exp = [11.689 29.783 54.712 108.451 180.296 257.250 411.930 928.955]';
>> x_exp = [1988:1995]';
```

%Linealizacion y = Ke^(a*t) --> log y = log K + a*t

```
>> y = log(y_exp);
>> t = x_exp;

>> A = [t t.^0];
>> N = A'*A;
>> c=A'*y;
>> p = N\c;
```

% Comprobacion de que lo lineal esta bien

```
>> x_graf = linspace(1988,1995);
>> plot(t,y,'.',x_graf,polyval(p,x_graf))

% 'Deshacer el cambio'

>> x_graf = linspace(1988,1995);
>> y_graf = exp(polyval(p,x_graf));
>> plot(x_exp,y_exp,'.',x_graf,y_graf)
```

Estudiando...

Ejemplo

```
>> x = (1988:1995)';
>> y = [28.1 30 32.3 34.6 35.3 36.4 37.5 38.5]';
bar(x,y)
>> bar(x,y)
>> A = [x.^2 x x.^0];
>> N = A'*A;
>> c = A'*y;
>> sol = N\c

>> xg = linspace(1988,1995);
>> yg = polyval(sol, linspace(1988,1995));
>> plot(linspace(1988,1995), yg)
```

Aproximacion funcional mínimo cuadrática

-- Los puntos serviran tanto para plot como para integrar con trapz

```
>> h = 0.001;
>> x = 0:h:1;
>> y = sin(pi*x.^2);
```

-- Dibujamos la funcion

```
>> plot(x,y)
```

-- Creamos el sistema

```
>> N = hilb(3); %matriz coeficientes
>> b(1) = trapz(x,y);
>> b(2) = trapz(x,y.*x);
>> b(3) = trapz(x,y.*x.^2);
>> b = b(:); %vector terminos independientes
```

-- Resolvemos el sistema

```
>> c = N\b
c =
   -0.3552
    3.5790
   -2.7884
```

el polinomio se ha obtenido de menor a mayor grado, pero matlab funciona al reves.
Hay que darle la vuelta.

```
>> p = flipud(c);
```

-- Dibujamos la aproximacion

```
>> z = polyval(p,x);
>> plot(x,y,':',x,z)
```

-- Convendria aumentar el grado del polinomio

El problema de este método es que los calculos hechos no tienen

ninguna utilidad. Hay que recalcularlo todo.

```
>> N = hilb(4);
>> b(4,1) = trapz(x,y.*x.^3);
>> c = N\b
c =
    0.0811
   -1.6560
   10.2993
   -8.7251
>> p = flipud(c);
>> z = polyval(p,x);
>> plot(x,y,':',x,z)
```

-- Condicionamiento

```
>> cond(N)

ans =

  1.5514e+004
```

Funcion desarrollo en serie de Legendre

---

15. Aproximación funcional mínimo-cuadrática. Polinomios ortogonales

En el tema anterior obteníamos el polinomio *p(x)* de cierto grado que mejor aproximaba un número finito de puntos en el sentido de hacer mínimo el error cuadrático. En este tema, el objetivo es aproximar, no un número finito de puntos, sino toda una función en un intervalo mediante una combinación lineal de funciones base.

Para ello usaremos también un producto escalar, pero ahora definido en un espacio infinito dimensional, que contiene en particular a las funciones continuas en el intervalo de trabajo.

Tal producto se define mediante una integral y su cálculo práctico es más costoso que el de un producto escalar en dimensión finita. Conviene, por lo tanto, trabajar con bases ortonormales para efectuar el mínimo número posible de productos escalares.

Analizaremos varias de estas bases ortonormales, adecuadas cada una de ellas para aproximar distintos tipos de funciones y que tienen importantes aplicaciones, además de la aproximación propiamente dicha, como son la integración numérica y la resolución de ecuaciones en derivadas parciales.

Planteamos en primer lugar el problema de la *aproximación funcional mínimo cuadrática* y lo interpretamos como la minimización de una norma procedente de un producto escalar. La solución viene dada por una proyección ortogonal respecto a dicho producto escalar.

El cálculo de la proyección ortogonal desemboca en la formulación de las *ecuaciones normales.* En la práctica, la obtención de estas ecuaciones resulta muy costosa, y lo que es peor, el sistema lineal resultante suele ser mal condicionado.

Estos inconvenientes aconsejan la utilización de *bases ortonormales* en los subespacios de las funciones aproximantes.

La aproximación polinómica se puede realizar considerando distintos sistemas de polinomios ortogonales, obtenidos al aplicar el método de ortogonalización de Gram-Schmidt a los polinomios básicos 1, x, x^2, ... , con respecto a distintos productos escalares. Los polinomios ortogonales más utilizados en intervalos acotados son los de *Legendre* y los de *Chebyshef*.

15.1. Aproximación Funcional Mínimo-Cuadrática

Dada una función f real o compleja, definida en un intervalo $[a, b]$, se trata de aproximarla mediante una función g combinación lineal de funciones sencillas, llamadas funciones base:

$$g(x) = c_1 g_1(x) + c_2 g_2(x) + \cdots + c_n g_n(x)$$

Las funciones base g_i son dadas y los coeficientes c_i se determinan de modo que se minimice la distancia entre f y g. En este tema consideramos distancias que provienen de un producto escalar. Veamos pues, en primer lugar, cómo se define el producto escalar en un espacio de funciones.

Supongamos que las funciones toman valores reales. Se define el producto escalar mediante la integral

$$< f, g > = \int_a^b w(x) f(x)\, g(x)\, dx$$

donde $w(x)$ es una función no negativa en el mismo intervalo, denominada *peso*. Si las funciones toman valores complejos, para que el producto escalar té bien definido, hemos de conjugar uno de los factores

$$< f, g > = \int_a^b w(x) f(x)\, \overline{g(x)}\, dx$$

Distintas elecciones del peso dan lugar a diferentes productos escalares. Una elección sencilla consiste en considerar un peso constante.

A partir del producto escalar se define la norma

$$\|f\|^2 = < f, f > = \int_a^b w(x) f(x)\, \overline{f(x)}\, dx = \int_a^b w(x) |f(x)|^2\, dx$$

y a partir de la norma, la distancia entre f y g:

$$\mathrm{d}(f, g) = \|f - g\|.$$

Volviendo a nuestro problema, tratamos de hallar la función g, combinación lineal de las funciones base, a menor distancia de f. Por las propiedades del espacio euclídeo, g es la proyección ortogonal de f sobre el subespacio engendrado por las funciones base. Para obtener dicha proyección, descomponemos f en dos sumandos: uno, combinación lineal de las funciones base, y el otro, ortogonal a las mismas:

$$f = g + h = c_1 g_1 + c_2 g_2 + \cdots + c_n g_n + h$$

Al imponer la ortogonalidad de h a las funciones base, se obtiene el sistema de ecuaciones normales, cuya solución proporciona los coeficientes de la proyección buscada.

$$< f, g_i > = c_1 < g_1, g_i > + c_2 < g_2, g_i > + \cdots + c_n < g_n, g_i >$$

donde hemos omitido el término que se anula $< h, g_i >$.

Ejemplo 1

Aproximemos la función $f(x) = \sin(\pi x^2)$ en el intervalo [0, 1] mediante un polinomio $p(x)$ de grado 2 que minimice el error cuadrático

$$E^2 = \|f - p\|^2 = \int_0^1 |f(x) - p(x)|^2\, dx$$

Descomponemos $f(x)$ en suma de un polinomio de grado 2, $p(x)$ y una función, $q(x)$.

$$f(x) = c_0 + c_1 x + c_2 x^2 + q(x)$$

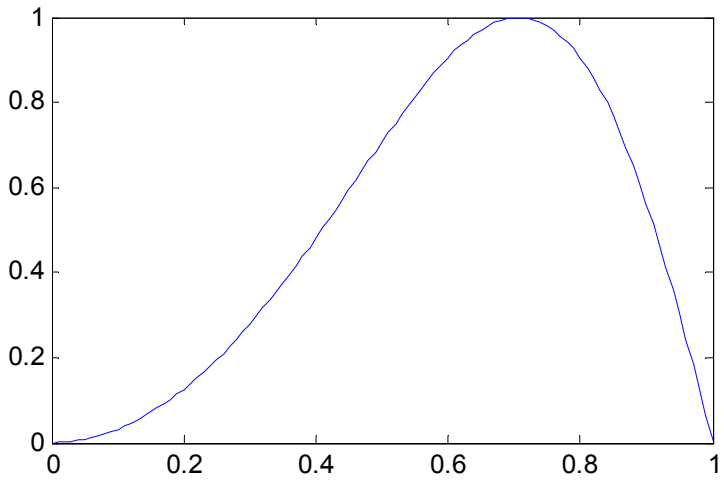
Exigimos que q sea ortogonal a 1, x y x^2:

$$\left.\begin{array}{llll} < f(x),1 > & = c_0 < 1,1 > & + c_1 < x,1 > & + c_2 < x^2,1 > \\ < f(x),x > & = c_0 < 1,x > & + c_1 < x,x > & + c_2 < x^2,x > \\ < f(x),x^2 > & = c_0 < 1,x^2 > & + c_1 < x,x^2 > & + c_2 < x^2,x^2 > \end{array}\right\}$$

Los elementos de la matriz del sistema se pueden calcular analíticamente

$$n_{ij} = < x^{i-1}, x^{j-1} > = \int_0^1 x^{i-1} x^{j-1}\, dx = \frac{1}{i+j-1}$$

resultando así la llamada *matriz de Hilbert*



$$\boldsymbol{N} = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}$$

Los términos independientes se obtienen analíticamente en este ejemplo con un poco más de esfuerzo, pero en general hay que aplicar una fórmula de cuadratura numérica. Para no tener que editar un fichero de función por cada integrando, utilizamos un método de integración que acepte como parámetro de entrada los valores del integrando en los nodos, en lugar de la función que evalúa el integrando. Por ejemplo, podemos aplicar la función `trapz` de MATLAB o editar un programa como el siguiente que aplica el método de Simpson.

```
function s = simpson(x,y)
n = length(x);
h = x(2)-x(1);              % Nodos equiespaciados
s = h/3*(y(1) + y(n) + 4*sum(y(2:2:n-1)) + 2*sum(y(3:2:n-2)));
```

Determinemos el polinomio mínimo-cuadrático operando con MATLAB. Tomamos bastantes nodos para que la estimación de la integral por trapecios sea suficientemente

precisa (en caso de duda, deberíamos usar trapecios iterativo u otro método de mayor orden). Representamos la función a aproximar.

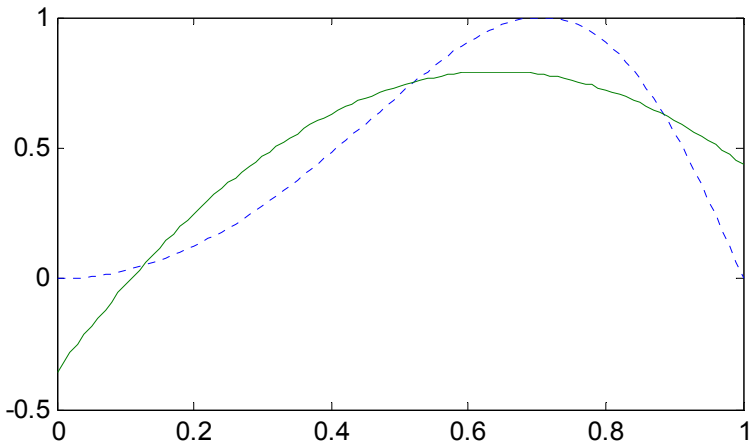
```
h = 0.001;
x = 0:h:1;
y = sin(pi*x.^2);
plot(x,y)
```

Construimos y resolvemos el sistema de ecuaciones normales. La matriz, directamente y los términos independientes evaluando las integrales.

```
N = hilb(3);
b(1) = trapz(x,y);
b(2) = trapz(x,y.*x);
b(3) = trapz(x,y.*x.^2);
b = b(:)
c = N\b

b =
    0.5049
    0.3183
    0.2187
c =
   -0.3552
    3.5790
   -2.7884
```

Ordenamos los coeficientes del polinomio de mayor a menor grado y lo representamos gráficamente.

```
p = flipud(c);
z = polyval(p,x);
plot(x,y,x,z)
```



Estimamos el error cuadrático integrando el cuadrado de la diferencia entre la función y la aproximación.

```
E2 = trapz(x,(y-z).^2)
```

```
E2 =
    0.0278
```

Ejercicio 1

Dibuja el polinomio mínimo cuadrático de tercer grado que aproxima la función dada.

Ejemplo 2

Aproxima la función $f(x)=1/x$ por un polinomio de cuarto grado, mínimo-cuadrático con respecto a la norma integral en el intervalo [1,10].

Procedemos como en el ejemplo anterior, pero realizando las integrales en el intervalo [1,10]. La matriz del sistema tiene elementos

$$n_{ij} =< x^{i-1}, x^{j-1} >= \int_1^{10} x^{i-1}\, x^{j-1}\, dx = \frac{10^{i+j-1}-1}{i+j+1}$$

y los términos independientes son

$$b_i =< f(x), x^{i-1} >= \int_1^{10} \frac{1}{x}\, x^{i-1}\, dx = \frac{10^{i-1}-1}{i+j+1}$$

para y los términos independientes son

Efectuamos los cálculos con MATLAB. Podemos obtener la matriz del sistema mediante dos bucles anidados:

```
for i=1:5
    for j=1:5
        k=i+j-1;
        N(i,j)=(10^k-1)/k;
    end
end
```

Una alternativa más eficiente por no usar bucles, sugerida por el código de la función `hilb` es la siguiente:

```
J = 1:5;
J = ones(5,1)*J;
I = J';
K = I+J-1;
N = (10.^K-1)./K;
```

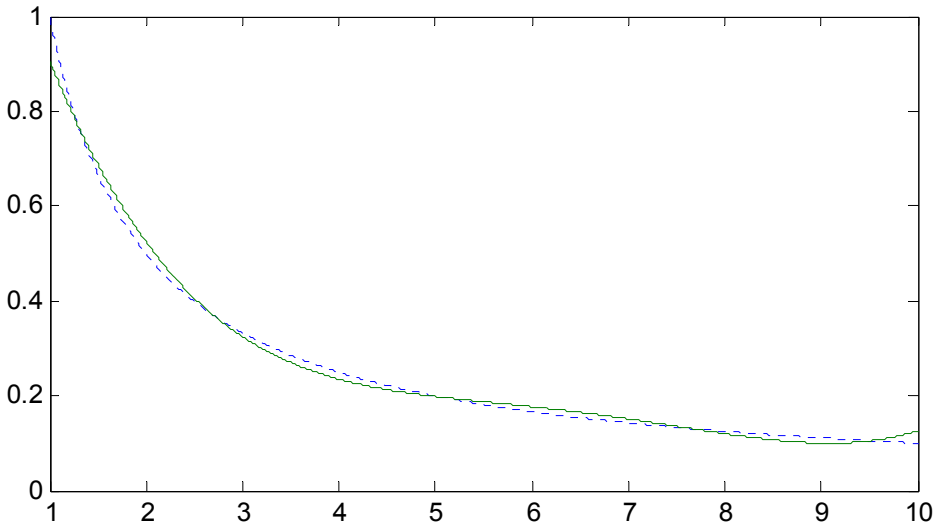
Aplicando `trapz` a una matriz cuyas columnas son los sucesivos integrandos, obtenemos de una vez los términos independientes:

```
h = 0.01;
x = (1:h:10)';
y = 1./x;
Y = [y, y.*x, y.*x.^2, y.*x.^3, y.*x.^4];
b = trapz(x,Y)'

b =
```

```
  1.0e+003 *
    0.0023
    0.0090
    0.0495
    0.3330
    2.4998
```

Resolvemos el sistema para obtener el polinomio mínimo-cuadrático y lo representamos junto a la función a aproximar.

```
c = N\b
p = flipud(c);
z = polyval(p,x);
plot(x,y,x,z)

c =
    1.5641
   -0.8302
    0.1920
   -0.0199
    0.0008
```



El error cuadrático es:

```
E2 = trapz(x,(y-z).^2)

E2 =
    0.0019
```

La condición del sistema de ecuaciones normales es elevada, lo que limita en la práctica el grado del polinomio de aproximación.

```
cond(N)

ans =
  6.8266e+009
```

15.2. Bases ortogonales

En los ejemplos anteriores, el sistema de ecuaciones normales del que se obtiene el polinomio mínimo-cuadrático es mal condicionado. Esto se debe a que las funciones 1, x, x^2, ..., son "poco" independientes en el intervalo [0, 1], como sugiere la semejanza de sus gráficas.

Aproximando en otros intervalos o respecto a otras funciones, la matriz no tiene una expresión analítica tan sencilla, obteniéndose mediante el cálculo aproximado de $n(n+1)/2$ integrales (es simétrica), lo que supone un coste considerable.

Para evitar estos problemas, recurrimos a una base ortogonal $\{g_1, g_2, \ldots, g_n\}$ del subespacio sobre el que proyectamos. De este modo, se asegura, por una parte, la "máxima" independencia posible entre las funciones base y, por otra, se reduce el número de integrales a calcular (la matriz es diagonal, pues $\langle g_k, g_j \rangle = 0$, para $i \neq j$) y se mejora la estabilidad numérica. Multiplicando escalarmente la descomposición

$$f = g + h = c_1 g_1 + c_2 g_2 + \cdots + c_n g_n + h$$

por g_j se obtienen unas ecuaciones normales de solución inmediata

$$< f, g_k > = c_k < g_k, g_k >, \text{ para } k = 1, 2, \ldots, n.$$

Cada sumando $c_k g_k$ es la proyección ortogonal de f sobre la dirección g_k y al ser estas direcciones ortogonales, la proyección ortogonal g sobre el subespacio generado es la suma de estas proyecciones.

Debido también a la ortogonalidad, el error cuadrático se expresa sencillamente como suma de los cuadrados de las normas de las proyecciones en la dirección de los vectores básicos.

$$\|f - g\|^2 = \|f\|^2 - \|g\|^2 = \|c_1 g_1\|^2 + \|c_2 g_2\|^2 + \cdots + \|c_n g_n\|^2$$

El añadir un elemento más a la base ortogonal supone el mínimo trabajo, pues las operaciones previas se aprovechan completamente. Así, por ejemplo, si tenemos un sistema ortogonal de polinomios de grado creciente, podemos obtener sucesivamente la mejor aproximación de cada grado. En el caso de base no ortogonal, al cambiar de grado, cambia completamente la solución de las ecuaciones normales.

Por estas razones, a partir de ahora, trabajaremos siempre con bases ortogonales para obtener aproximaciones funcionales mínimo cuadráticas.

15.3. Polinomios de Legendre

Consideremos el espacio de las funciones continuas en el intervalo [−1, 1] con el producto escalar definido por la integral

$$< f, g > = \int_a^b f(x)\, g(x)\, dx$$

Tratamos de obtener una base ortogonal del subespacio de los polinomios de grado menor o igual que n. Para ello, podemos ortogonalizar sucesivamente los polinomios $1, x, x^2, \ldots x_n$, con respecto a dicho producto escalar. Los polinomios ortogonales resultantes, salvo un factor constante, se denominan *polinomios de Legendre*. Los primeros polinomios son:

$$P_0(x)=1 \qquad P_3(x)=\frac{5x^3-3x}{2}$$
$$P_1(x)=x$$
$$P_2(x)=\frac{3x^2-1}{2} \qquad P_4(x)=\frac{35x^4-30x^2+3}{8}$$

Los polinomios de Legendre verifican la siguiente relación recursiva, que se utiliza para generarlos.

$$P_0(x)=1$$
$$P_1(x)=x$$
$$P_{k+1}(x)=\big((2k+1)xP_k(x)-kP_{k-1}(x)\big)/(k+1)$$
para $k = 1, 2, \ldots$

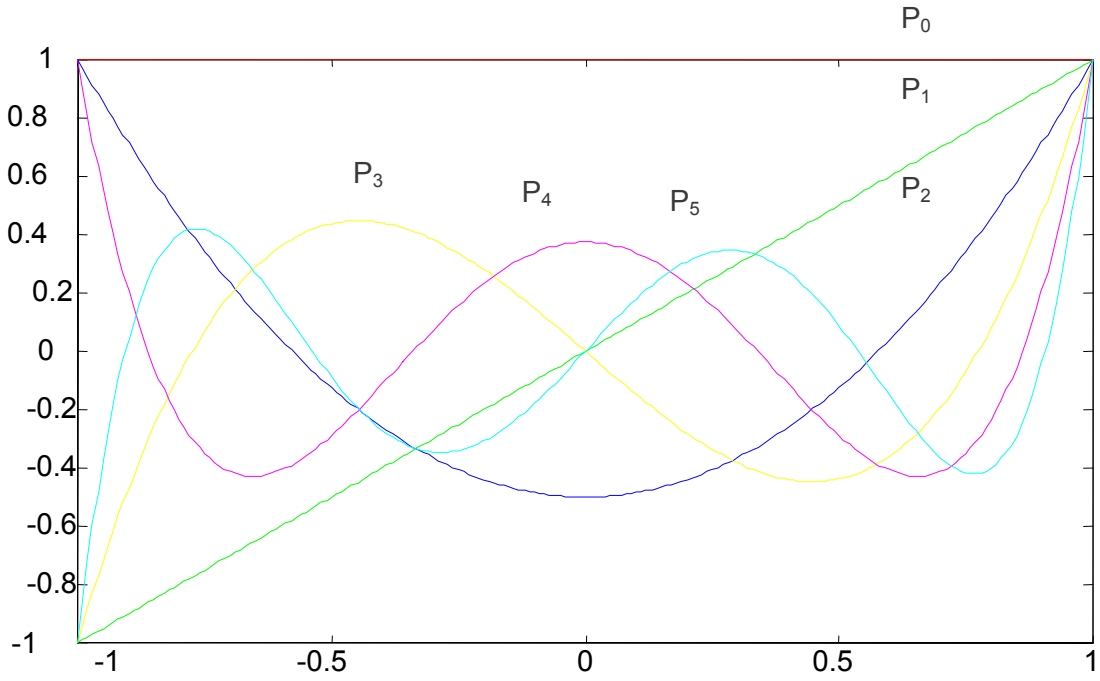


Figura 1: Polinomios de Legendre en [−1,1]

Aplicando formalmente la recursión, obtenemos los coeficientes del polinomio de Legendre de cualquier grado. Si ejecutas el programa siguiente con `format rat`, puedes comprobar el resultado tal como aparece en las tablas.

```
function p = polegend(n)
r = 1                   % p0(x) = 1
q = [1 0]               % p1(x) = x
for k = 1 : n-1         % Polinomio de grado k+1
    p=((2*k+1)*[q 0]- k*[0 0 r])/(k+1)
    r = q;
    q = p;
end
```

Ejemplo 3

Aproximemos la función de Runge $f(x)=\dfrac{1}{1+25x^2}$ en [−1, 1] por un polinomio mínimo-cuadrático de grado 6, utilizando los polinomios de Legendre P_0, P_1, P_2, …, P_6.

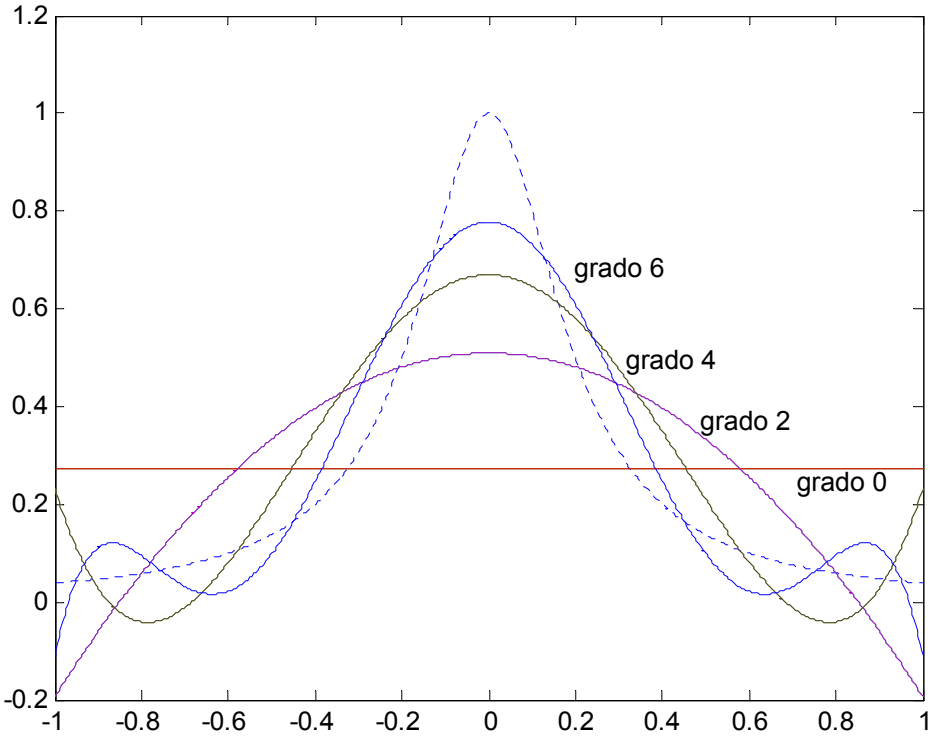
La aproximación pedida es

$$g = c_0P_0 + c_1P_1 + +c_2P_2 + \cdots + c_6P_6$$

donde los coeficientes c_k se obtienen de las expresiones

$$c_k = \frac{< f, P_k >}{< P_k, P_k >} = \frac{\int_{-1}^{1} f(x)\, P_k(x)\, dx}{\int_{-1}^{1} (P_k(x))^2\, dx} = \frac{\int_{-1}^{1} \dfrac{P_k(x)}{1+25x^5}\, dx}{\dfrac{2}{2k+1}}$$

El denominador $\dfrac{2}{2k+1}$ es el cuadrado de la norma del polinomio P_k. Para efectuar los cálculos editamos un programa de MATLAB que trabaja con los valores de los polinomios de Legendre, en lugar de su expresión analítica. El programa obtiene un polinomio ortogonal en cada paso utilizando la relación de recurrencia, proyecta al mismo tiempo la función a aproximar sobre dicho polinomio y acumula las proyecciones obteniendo la mejor aproximación para cada grado.




```
function c = serielegnorm(f,n)

% Aproximación de f por polinomios de Legendre de grado n en [-1,1]

m = 100*(n+1);
```

```
x = linspace(-1,1,m)';
y = feval(f,x);
plot(x,y,':'), pause
r = ones(size(x));                          % P0
c(1) = trapz(x,y.*r)/2;                     % Coeficiente de P0
z = c(1)*r;                                 % Aproximación de grado 0
Z = z;
plot(x,y,':',x,Z), pause
q = x;                                      % P1
c(2) = trapz(x,y.*q)*3/2;                   % Coeficiente de P1
z = z+c(2)*q;                               % Aproximación de grado 1
Z = [Z, z];
plot(x,y,':',x,Z), pause
for k = 1:n-1
     p = ((2*k+1)*x.*q-k*r)/(k+1);          % Pk+1
     c(k+2) = trapz(x,y.*p)*(2*k+3)/2;      % Coeficiente de Pk+1
     z = z+c(k+2)*p;                        % Aproximación de grado k+1
     Z = [Z, z];
     plot(x,y,':',x,Z), pause
     r = q; q = p;                          % Términos de la recursión
end
```

15.4. Polinomios de Chebyshev

Las funciones 1, cos *t*, cos 2*t*, ..., cos *nt* forman un sistema ortogonal con respecto al producto escalar definido en el intervalo [0, π].

$$\int_0^{\pi}\cos(jt)\cos(kt)\,dt=\int_0^{\pi}\frac{\cos(j+k)t+\cos(j-k)t}{2}\,dt=\begin{cases}0 & si\ j\neq k\\ \pi & si\ j=k=0\\ \pi/2 & si\ j=k>0\end{cases}$$

Si hacemos el cambio de variable $x = \cos(t)$, mediante relaciones trigonométricas podemos expresar estas funciones como polinomios en x:

$$T_0(x)=1$$
$$T_1(x)=\cos(t)=x$$
$$T_2(x)=\cos(2t)=\cos^2(t)-\sin^2(t)=\cos^2(t)-(1-\cos^2(t))=2\cos^2(t)-1=2x^2-1$$
$$T_3(x)=\cos(3t)=\cos(2t)\cos(t)-\sin(2t)\sin(t)=4\cos^3(t)-3\cos(t)=4x^3-x$$
$$\vdots$$

En general podemos poner

$$T_n(x)=\cos\big(n\arccos(x)\big)$$

teniendo el cuenta que en realidad es un polinomio, llamado *polinomio de Chebyshef de grado n*.

Tras el cambio de variable, la ortogonalidad de los cosenos se convierte en ortogonalidad entre polinomios

$$\int_0^{\pi} \cos(jt)\cos(kt)\,dt = \left[\begin{array}{l} \cos t = x \\ dt = -\dfrac{1}{\sqrt{1-x^2}} \end{array}\right] = \int_{-1}^{1} \frac{T_j(x)\,T_k(x)}{\sqrt{1-x^2}}\,dx = \begin{cases} 0 & si\ j \neq k \\ \pi & si\ j = k = 0 \\ \pi/2 & si\ j = k > 0 \end{cases}$$

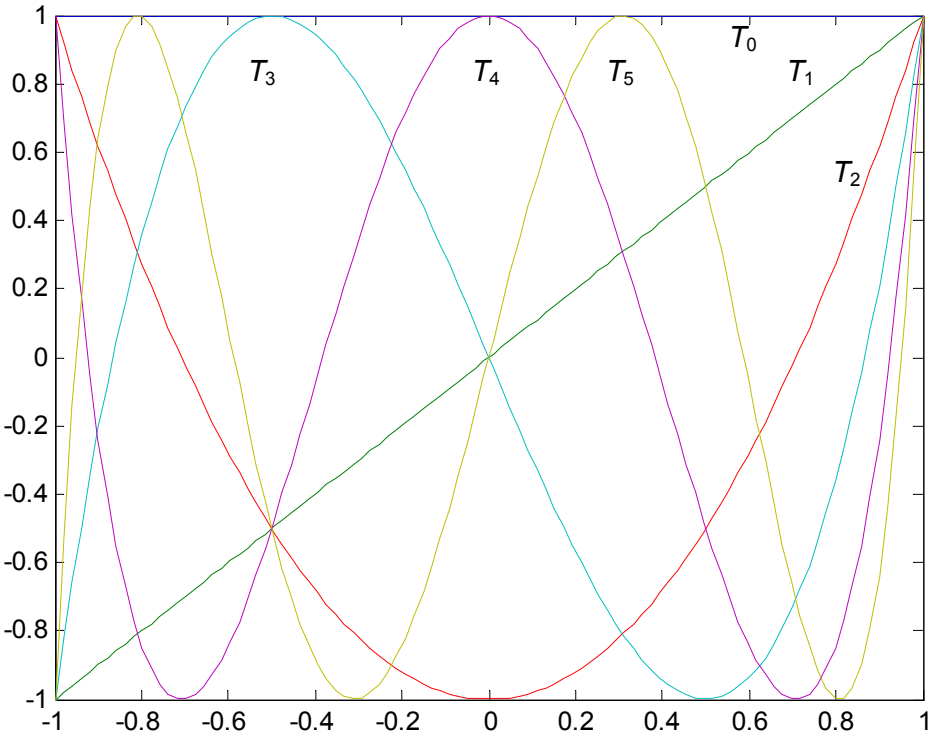
es decir, que los polinomios de Chebyshef son ortogonales respecto a un producto escalar integral con peso $\dfrac{1}{\sqrt{1-x^2}}$ en $[-1,1]$

$$< f, g >= \int_{-1}^{1} \frac{f(x)\,g(x)}{\sqrt{1-x^2}}\,dx$$

Alternativamente, estos polinomios se obtienen ortogonalizando las funciones 1, x, x^2, ..., con respecto a este producto escalar.

Como en el caso de Legendre, los polinomios de Chebyshef cumplen una relación de recurrencia que es la vía más eficaz para su obtención.

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x)$$
para $k = 1, 2, \ldots$



En el gráfico apreciamos la propiedad fundamental de los polinomios de Chebyshef: que toman alternativamente valores máximos y mínimos iguales a +1 y –1 entre cada par de raíces consecutivas. Dichos valores se alcanzan en el intervalo [–1, 1], en concreto, el polinomio de grado n tiene extremos en los puntos

$$x_k = \cos\frac{k\pi}{n},\ k = 0, 1, \ldots, n.$$

Otra propiedad interesante, que comparten con otros polinomios ortogonales como los de Legendre, es que el polinomio de grado *n* tiene *n* raíces distintas en el intervalo [−1, 1]. Las raíces se obtienen fácilmente de su definición trigonométrica: el polinomio de grado *n* se anula en

$$r_k = \cos\frac{(2k-1)\pi}{2n},\ k = 1, 2, \ldots, n.$$

Aproximando una función con respecto a la norma que genera los polinomios de Chebyshef, obtenemos el polinomio de determinado grado que menos se desvía en valor absoluto de la función dada en el intervalo [−1, 1].

Los coeficientes de la aproximación

$$g = c_0T_0 + c_1T_1 + +c_2T_2 + \cdots + c_nT_n$$

se obtienen de las expresiones

$$c_k = \frac{< f, T_k >}{< T_k, T_k >}$$

donde el producto escalar viene dado por la integral

$$< f, g >= \int_{-1}^{1} \frac{f(x)\ g(x)}{\sqrt{1-x^2}}\,dx$$

Aunque el integrando tiende a infinito en los extremos del intervalo, la integral existe si *f* es acotada. Como la evaluación numérica de integrales impropias es poco precisa, haremos el cambio de variable $x = \cos(t)$ que evita la anulación del denominador.

$$< f, g >= \int_{-1}^{1} \frac{f(x)\ g(x)}{\sqrt{1-x^2}}dx = \begin{bmatrix} \cos(t) = x \\ dt = -\dfrac{1}{\sqrt{1-x^2}} \end{bmatrix} = \int_0^{\pi} f(\cos(t))g(\cos(t))dt$$

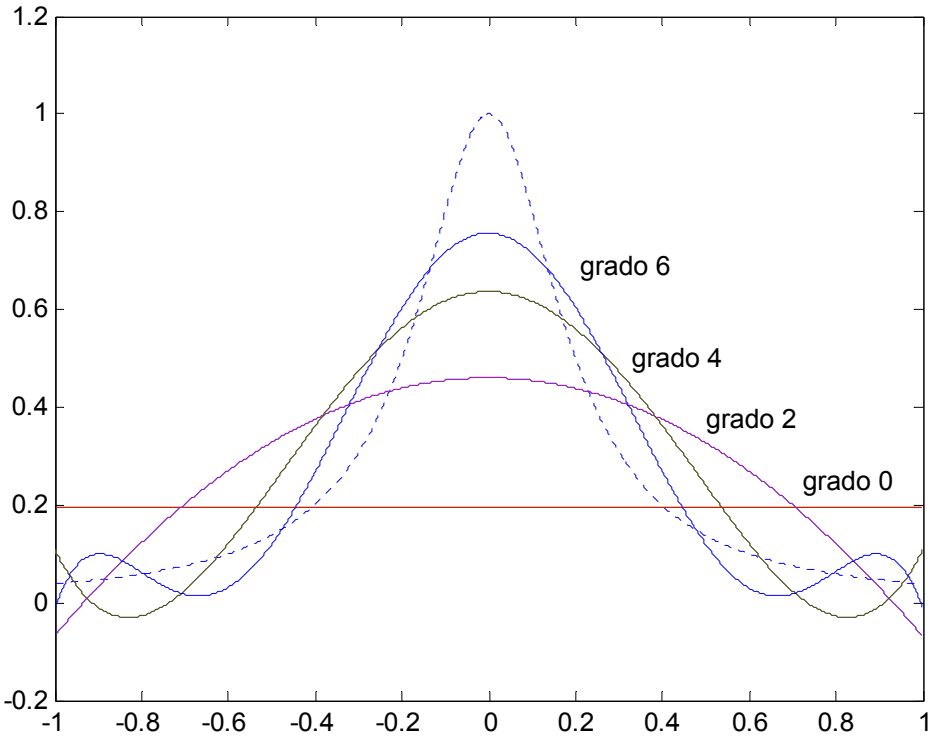
Ejemplo 4

Aproximemos la función de Runge $f(x) = \dfrac{1}{1+25x^2}$ en [−1, 1] por un polinomio de grado 6, mínimo-cuadrático respecto a la norma asociada a los polinomios de Chebyshef.

Utilizamos un programa análogo al caso de Legendre, modificando adecuadamente la recurrencia para generar los polinomios de Chebyshef y efectuando las integrales con el cambio de variable $x = \cos(t)$ para evitar las singularidades del integrando.

```
function c = seriechebnorm(f,n)

% Aproximación de f por polinomios de Chebyshef de grado n en [-1,1]

m = 100*(n+1);
h = pi/m;
t = (0:h:pi)';                          % Nodos de integracion
x = cos(t);                             % Cambio de variable
y = feval(f,x);
```

```
plot(x,y,':'), pause
r = ones(size(x));                      % P0
c(1) = trapz(t,y)/pi;                   % Coeficiente de P0
z = c(1)*r;                             % Aproximación de grado 0
Z = z;
plot(x,y,':',x,Z), pause
q = x;                                  % P1
c(2) = trapz(t,y.*q)*2/pi;              % Coeficiente de P1
z = z+c(2)*q;                           % Aproximación de grado 1
Z = [Z, z];
plot(x,y,':',x,Z), pause
for k = 1:n-1
     p = 2*x.*q - r;                    % Pk+1
     c(k+2) = trapz(t,y.*p)*2/pi;       % Coeficiente de Pk+1
     z = z+c(k+2)*p;                    % Aproximación de grado k+1
     Z = [Z, z];
     plot(x,y,':',x,Z), pause
     r = q; q = p;                      % Términos de la recursión
end
```



15.5. Problemas

15.5.1. Enunciados

1.- Aproxima la función $f(x) = e^x$ en el intervalo [0, 3] mediante un polinomio quinto grado. Halla el error cuadrático. Aproxímala ahora por el polinomio de Taylor del mismo grado. Halla el error cuadrático. Compara gráficamente los errores de ambas aproximaciones.

2.- Halla a, b, c y d para que la integral $\int_{-1}^{1} \left| e^x - (a + bx + cx^2 + dx^3) \right|^2 dx$ sea mínima.

3.- Aproxima $f(x) = |x|$ en [−1, 1] por un polinomio de grado 4, utilizando polinomios de Legendre.

4.- Verifica la fórmula de Rodrigues para los primeros polinomios de Legendre:

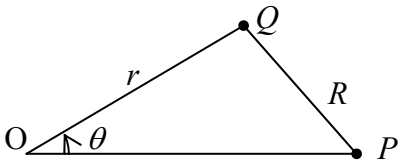
$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n$$

5.- El potencial en un punto P creado por una carga situada en el punto Q es inversamente proporcional a la distancia R entre ambos. Sitúa el punto P en (0,1) y el punto Q en una posición arbitraria de coordenadas polares (θ, r)

a) Prueba que $R^2 = 1 + r^2 - 2r\cos\theta$

Por tanto, el potencial en el punto P está determinado por la función

$$f(r) = \frac{1}{\sqrt{1 + r^2 - 2rx}}$$



donde $x = \cos\theta$.

b) Comprueba que f es una función generatriz de los polinomios de Legendre, es decir, que éstos son los coeficientes de r^n en el desarrollo de f en serie de potencias de r.

6.- Utilizando la relación

$$e^{in\theta} = \cos n\theta + i \sin n\theta = (\cos\theta + i \sin\theta)^n$$

y la definición de los polinomios de Chebyshef

$$T_n(x) = \cos n\theta = \mathrm{Re}(\cos\theta + i \sin\theta)^n$$

donde $x = \cos\theta$, demuestra la fórmula explícita de los mismos:

$$T_n(x) = \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{2k} x^{n-2k} (x^2 - 1)^k$$

7.- Desarrollando $\cos(n \pm 1)\theta$, demuestra la ley de recurrencia que verifican los polinomios de Chebyshef.

8.-Expresa el polinomio de Tchebyshev de grado n como producto de polinomios de grado 1.

9.- Expresa las funciones 1, x, x^2, …, x^5 como combinación lineal de los polinomios de Chebyshef de grado no superior a 5. Dado un polinomio cualquiera de grado no superior a 5, explica cómo calcularías los coeficientes de su expresión como combinación lineal de los polinomios de Chebyshef.

10.- Aproxima $f(x)=|x|$ en [−1, 1] por un polinomio de grado 4, respecto de la norma de Chebyshev. Compara el resultado con el del problema 3.

15.5.2. Soluciones

1.-Procediendo como en el Ejemplo 2, se obtiene el polinomio mínimo-cuadrático que es

$$P(x)=0.9915+1.1083x+0.1796x^2+0.5342x^3-0.1455x^4+0.0474x^5$$

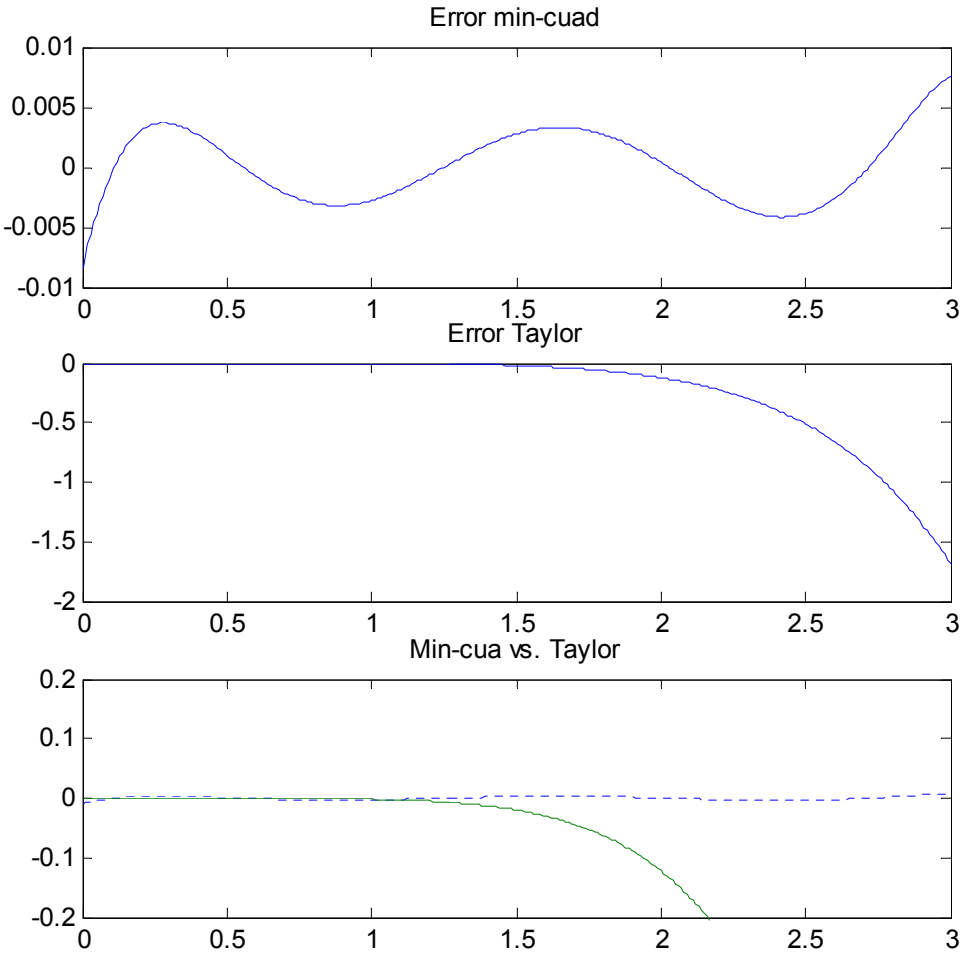
Gráficamente, este polinomio es indistinguible de la función exponencial. El error cuadrático es $2.5360\cdot10^{-5}$.

El polinomio de Taylor del mismo grado es

$$P(x)=1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}$$

El polinomio de Taylor se ajusta mucho mejor cerca del origen, pero tiene un error muy grande en la parte derecha del intervalo. Su error cuadrático es 0.6044, obviamente mayor que el del polinomio mínimo-cuadrático.

Los errores puntuales de ambos ajustes se comparan en la gráfica adjunta.



2.- Los valores pedidos son los coeficientes del polinomio mínimo-cuadrático de tercer grado que ajusta la función exponencial en el intervalo [−1, 1]. Efectuando las integrales por trapecios con paso 0.001 se obtienen los valores

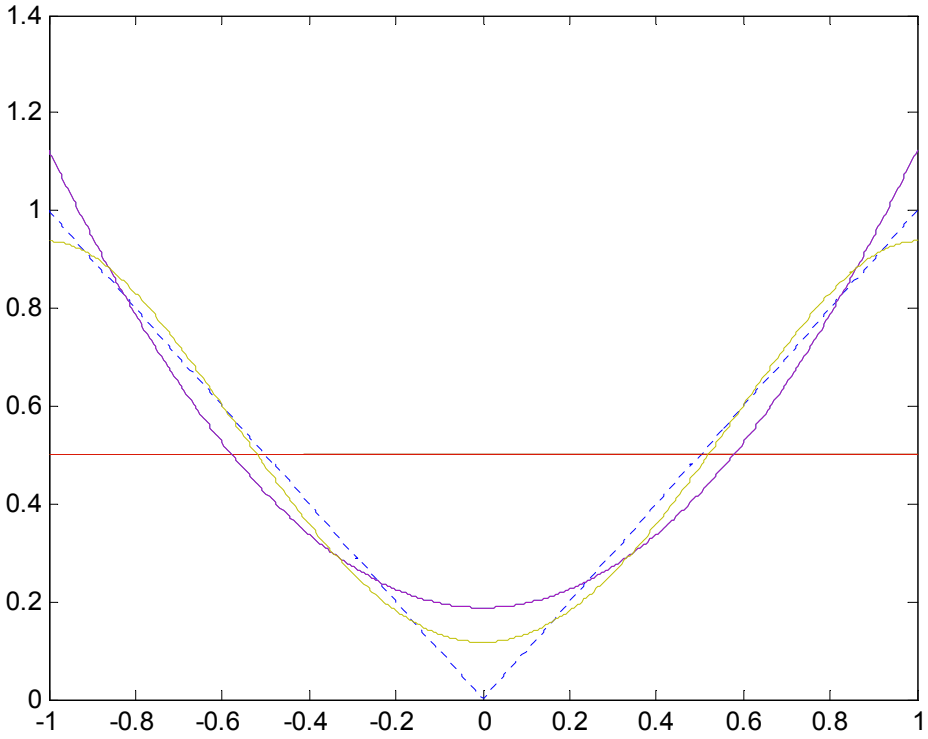
$$a = 0.9963,\ b = 0.9979,\ c = 0.5367,\ d = 0.1762$$

y el valor mínimo de la integral, 2.2289e−005.

3.- El desarrollo de Legendre de la función $f(x) = |x|$ en [−1, 1] es

$$0.5P_0(x) + 0.6250P_2(x) - 0.1874P_4(x).$$

Como la función es par, sólo aparecen en el desarrollo polinomios pares. La figura muestra los sucesivos polinomios de aproximación.



4.- La fórmula de Rodrigues es $P_n(x) = \dfrac{1}{2^n n!}\dfrac{d^n}{dx^n}(x^2-1)^n$.

Para $n = 0$, se obtiene trivialmente $P_0(x) = 1$.

$$P_1(x) = \frac{1}{2}\frac{d}{dx}(x^2-1) = x$$

$$P_2(x) = \frac{1}{2^2 2!}\frac{d^2}{dx^2}(x^4 - 2x^2 + 1) = \frac{1}{8}(12x^2 - 4) = \frac{3}{2}x^2 - \frac{1}{2}$$

$$P_3(x)=\frac{1}{2^3 3!}\frac{d^3}{dx^3}(x^6-3x^4+3x^2-1)=\frac{1}{48}(120x^3-72x)=\frac{5}{2}x^3-\frac{3}{2}x$$

$$P_4(x)=\frac{1}{2^4 4!}\frac{d^4}{dx^4}(x^8-4x^6+6x^4-4x^2+1)=$$
$$=\frac{1}{16\cdot 24}(1680x^4-1440x^2+144)=\frac{35}{8}x^4-\frac{30}{8}x^2+\frac{3}{8}$$

5.- a) El cuadrado de la distancia de $P(1,0)$ a $Q(r\cos\theta, r\sin\theta)$ es

$$R^2=(1-r\cos\theta)^2+(r\sin\theta)^2=1-2r\cos\theta+r^2(\cos^2\theta+\sin^2\theta)=1-2r\cos\theta+r^2$$

b) Calculemos los coeficientes del desarrollo de Taylor de $f(r)=\dfrac{1}{\sqrt{1+r^2-2rx}}$.

$$f(0)=1=P_0(x)$$

$$f'(r)=-\frac{2r-2x}{2\left(1+r^2-2rx\right)^{3/2}};\qquad f'(0)=x=P_1(x)$$

$$f''(r)=-\left[(r-x)\left(1+r^2-2rx\right)^{-3/2}\right]=$$
$$=-\left[\left(1+r^2-2rx\right)^{-3/2}+(r-x)\left(\frac{-3}{2}\right)\left(1+r^2-2rx\right)^{-5/2}(2r-2x)\right]$$

$$\frac{f''(0)}{2!}=-\frac{1}{2}(1-3x^2)=P_2(x).$$

7.- Aplicando las fórmulas del ángulo suma, tenemos

$$\cos(n+1)\theta=\cos n\theta\cos\theta-\sin n\theta\sin\theta$$
$$\cos(n-1)\theta=\cos n\theta\cos\theta+\sin n\theta\sin\theta$$

de donde podemos despejar

$$\cos(n+1)\theta=2\cos n\theta\cos\theta-\cos(n-1)\theta$$

Haciendo $x=\cos\theta$ y teniendo en cuenta la definición de los polinomios de Chebyshev obtenemos la relación pedida.

$$T_{n+1}(x)=2xT_n(x)-T_{n-1}(x)$$

9.- Los polinomios de Chebishev hasta el grado 5 son

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_2(x) = 2x^2 - 1$$
$$T_3(x) = 4x^3 - 3x$$
$$T_4(x) = 8x^4 - 8x^2 + 1$$
$$T_5(x) = 16x^5 - 20x^3 + 5x$$

Construimos una matriz cuyas columnas sean los coeficientes de los polinomios de Chebyshev. La inversa de esta matriz proporcionará las expresiones pedidas. Efectuamos los cálculos con MATLAB.

```
A = [ 1   0   0   0   0   0
      0   1   0   0   0   0
     -1   0   2   0   0   0
      0  -3   0   4   0   0
      1   0  -8   0   8   0
      0   5   0 -20   0  16]';
```

La matriz inversa es

```
B = inv(A)
```

```
B =
    1.0000         0    0.5000         0    0.3750         0
         0    1.0000         0    0.7500         0    0.6250
         0         0    0.5000         0    0.5000         0
         0         0         0    0.2500         0    0.3125
         0         0         0         0    0.1250         0
         0         0         0         0         0    0.0625
```

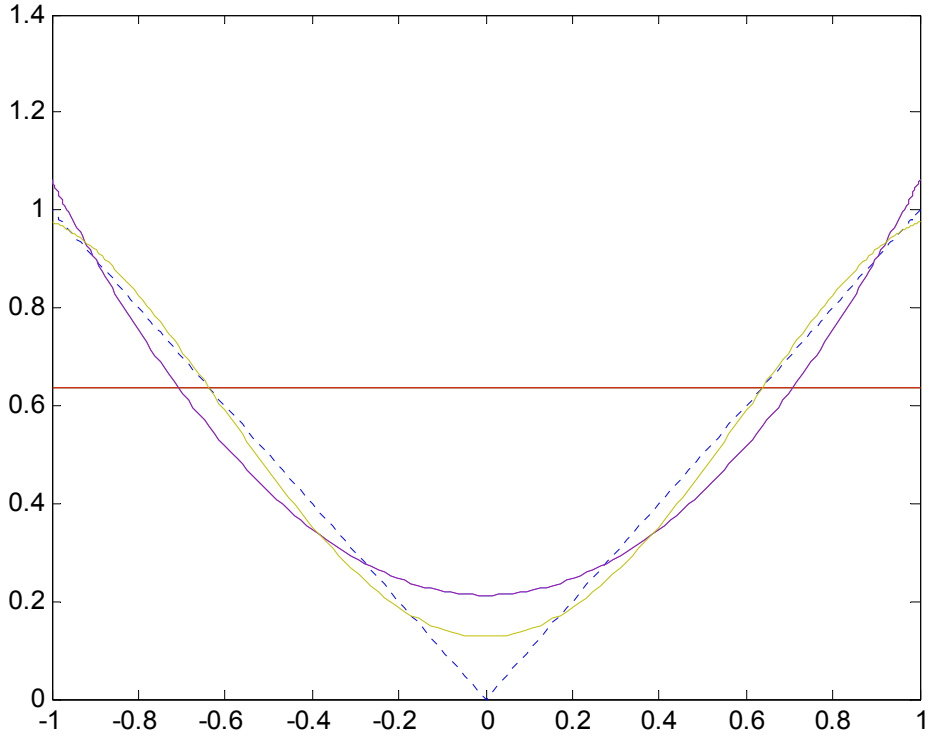
Para expresar un polinomio cualquiera en función de polinomios de Chebyshev, basta multiplicar esta última matriz por los coeficientes del polinomio. Por ejemplo, la última columna expresa x^5 como

$$x^5 = \frac{10T_1(x) + 5T_3(x) + T_5(x)}{16}$$

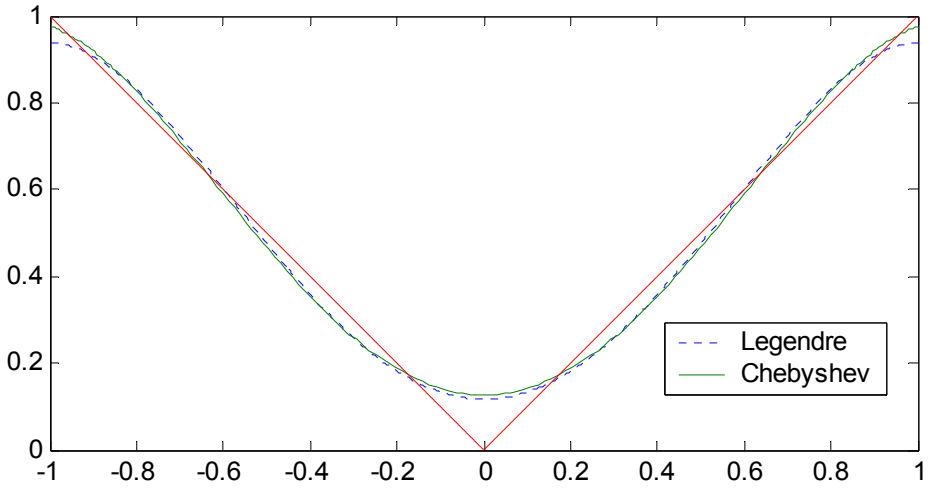
10.- El desarrollo de Chebyshev de la función $f(x) = |x|$ en [−1, 1] es

$$0.6366T_0(x) + 0.4244T_2(x) - 0.0849T_4(x).$$

Como la función es par, sólo aparecen en el desarrollo polinomios pares. La figura muestra los sucesivos polinomios de aproximación.



En esta figura superponemos las aproximaciones de Legendre y de Chebyshef.



Aproximacion Teorica

Funcion Onda cuadrada:

```
>> m = 2000;
>> h = 2*pi/m;
>> t = (0:h:2*pi)'; %abscisas, nodos de integracion
>>
>> y = sign(pi-t); %funcion a evaluar
>> plot(t,y,'.')
```

Creamos los coeficientes

```
>> b(1) = trapz(t,y.*sin(t))/pi; %se evalua la integral

>> b*pi %comprobacion
ans =
   4.0000

>> b(2) = trapz(t,y.*sin(2*t))/pi;
>> b(3) = trapz(t,y.*sin(3*t))/pi;
>> b(4) = trapz(t,y.*sin(4*t))/pi;
>> b(5) = trapz(t,y.*sin(5*t))/pi;
>> b(6) = trapz(t,y.*sin(6*t))/pi;
>> b(7) = trapz(t,y.*sin(7*t))/pi;

b =
   1.2732   -0.0000    0.4244   -0.0000    0.2546   -0.0000    0.1819
```

%Se comprueba que para k par, en la onda cuadrada, el coeficiente da cero, porque la onda es impar, y sólo tiene componentes en el seno, que tambien es impar.

```
>> %Vamos a representar
>> s = b(1)*sin(t);
>> plot(t,y,'.',t,s,'r')
>> hold on

>> %Vamos acumulando en s la funcion con cada vez mas términos
s = s + b(3)*sin(3*t) %recordemos que en este caso, sólo hay terminos del seno y sólo k impar
>> plot(t,y,'.',t,s,'g')

>> s = s + b(5)*sin(5*t);
```

```
>> plot(t,y,'.',t,s,'m')


>> %Comparamos coeficientes teoricos y numericos
>> c = 4/pi*[1 1/3 1/5 1/7 1/9];
c =
   1.2732    0.4244    0.2546    0.1819    0.1415
>> b
b =
   1.2732   -0.0000    0.4244   -0.0000    0.2546   -0.0000    0.1819


Aproximacion con la fórmula discreta
------------------------------------

>> N = 12
N =
   12
>> h = 2*pi/N;
>> t = (0:h:2*pi-h)';

>> y = sign(pi-t); %funcion a evaluar
>> y(1) = 0; % arreglamos el principio para obtener simetria
>>
>> stem(t,y) %obtener representacion

%para evaluar d eun golpe todos los productos escalares implicados:
>> M = [t.^0 cos(t) sin(t) cos(2*t) sin(2*t) cos(3*t) sin(3*t) cos(4*t) sin(4*t) cos(5*t) sin(5*t) cos(6*t) sin(6*t)];

%Me he equivocado pues he puesto 13 vectores independientes, sólo puedo 12
>> M = [t.^0 cos(t) sin(t) cos(2*t) sin(2*t) cos(3*t) sin(3*t) cos(4*t) sin(4*t) cos(5*t) sin(5*t) cos(6*t)]

>> T = M'*y*2/N
T =
         0
    0.0000
    1.2440
         0
   -0.0000
    0.0000
    0.3333
   -0.0000
   -0.0000
   -0.0000
    0.0893
```

```
        0
>> T(12) = T(12)/2 %en general

>> b %comparandolos con B
b =
   1.2732   -0.0000    0.4244   -0.0000    0.2546   -0.0000    0.1819
>> T'
ans =
  Columns 1 through 8
        0    0.0000    1.2440         0   -0.0000    0.0000    0.3333   -0.0000
  Columns 9 through 12
   -0.0000   -0.0000    0.0893         0
```

Hay bastante diferencia utilizando 12 puntos en comparacion con 2000.

Aunque hay uqe pensar que la aproximacion obtenida asi es de la funciom
discreta de sólo 12 puntos, y NO de la onda cuadrada.

Representemosla

```
>> tg = linspace(0,2*pi); %a efectos graficos utilizamos una variable continua
>> Z = T(3)*sin(tg);
>> hold on
>> plot(tg,Z,'r')
>> Z = Z + T(7)*sin(3*tg) %como se que se me anulan todos menos el 3, el 7 y el 11, me ahorro trabajo. En general habria uqe evaluar todos los términos: mirar transparencias
>> plot(tg,Z,'g')
>> Z = Z + T(11)*sin(5*tg)
>> plot(tg,Z,'k')
```

Vemos que la aproximacion ya es perfecta en el mundo discreto

16. Aproximación trigonométrica

Los fenómenos ondulatorios son muy frecuentes en la naturaleza y en las aplicaciones técnicas. Quizás la luz y el sonido sean los ejemplos más cotidianos, pero ni mucho menos los únicos. La luz, como sabemos, ocupa una estrecha banda en el amplio espectro de las radiaciones electromagnéticas y el sonido es un caso de vibración mecánica.

Desde el punto de vista teórico, la característica fundamental de las ondas es la periodicidad. El origen del sonido, por citar un ejemplo, está en las variaciones rítmicas de la presión del aire o medio en el que se produce. La velocidad con la que se suceden los incrementos y disminuciones de la presión determinan el tono del sonido. Estas variaciones se producen con gran rapidez en relación con la duración de un sonido y son muy uniformes en un sonido puro, salvo que se atenúa su intensidad. El ciclo elemental consta de un incremento de la presión que llega a un máximo y disminuye de nuevo provocando ahora una depresión. Esta a su vez se corrige para volver a la presión normal y comienza de nuevo el ciclo, repitiéndose periódicamente.

Las oscilaciones se estudian matemáticamente descomponiéndolas en suma de funciones periódicas sencillas como son las funciones trigonométricas seno y coseno. La afirmación del matemático francés J.B. Fourier de que cualquier función periódica podía representarse como suma de senos y cosenos causo una notable controversia de la que surgieron importantes avances en el conocimiento matemático, como la noción de convergencia uniforme o la aclaración de la propia idea de función.

Desde el punto de vista práctico, las componentes elementales de una onda reflejan muchas de sus propiedades físicas, por lo que su determinación reviste gran importancia en las aplicaciones técnicas. Generalmente basta un reducido número de estas componentes, también llamadas armónicos, para reproducir la información significativa del fenómeno ondulatorio estudiado. La amplitud y frecuencia de los armónicos se obtiene técnicamente con los osciloscopios y analíticamente mediante la teoría de Fourier que desarrollaremos en este tema.

En las aplicaciones no se suele disponer de una expresión analítica de la función a analizar, sino de valores de la misma medidos a intervalos regulares de tiempo. Es necesario por tanto, elaborar versiones discretas de la aproximación de Fourier, que proporcionen información a partir de un número finito de datos. Además, estos métodos se adaptan mejor al cálculo con ordenador, que trabaja con una cantidad finita de información.

16.1. El Sistema Trigonométrico

Una función de variable real *f* es periódica de periodo L, si $f(t+L)=f(t)$ para todo valor de *t*. Consecuentemente, $f(t+kL)=f(t)$ para cualquier *k* entero. Por comodidad,

consideraremos funciones de periodo 2π, pero los resultados se aplican fácilmente a funciones de periodo L arbitrario.
Una función periódica queda determinada por sus valores en cualquier intervalo de longitud igual al periodo.

El *sistema trigonométrico* está constituido por las funciones 1/2, $\sin(x)$, $\cos(x)$, $\sin(2x)$, $\cos(2x)$, ..., $\sin(nx)$, $\cos(nx)$, ... Estas funciones son ortogonales respecto al producto escalar dado por la integral

$$< f, g > = \frac{1}{\pi}\int_0^{2\pi} f(t)\, g(t)\, dt$$

Además, son unitarias, salvo 1/2, cuya norma es $1/\sqrt{2}$. La integral puede efectuarse en cualquier intervalo de longitud 2π, por ejemplo en $[-\pi, \pi]$.

La ortogonalidad del sistema trigonométrico puede comprobarse aplicando conocidas fórmulas de trigonometría que transforman sumas en productos (ver problema 1).

Dada una función f de periodo 2π, la combinación lineal de las funciones anteriores

$$s_n(t) = \frac{a_0}{2} + \sum_{k=1}^{n} \left(a_k \cos(kt) + b_k \sin(kt)\right)$$

que mejor se aproxima a f, se denomina *polinomio de Fourier de orden n* asociado a f. La mejor aproximación es la que minimiza el error cuadrático

$$E^2 = \|f - s_n\|^2 = \frac{1}{\pi}\int_{-\pi}^{\pi} |f(t) - s_n(t)|^2\, dt$$

y es precisamente la proyección ortogonal de f sobre el subespacio generado por las $2n+1$ primeras funciones del sistema trigonométrico. Debido a la ortogonalidad del mismo, los llamados *coeficientes de Fourier*, se calculan mediante las expresiones siguientes, conocidas como *fórmulas de Euler*:

$$a_0 = \frac{< f, \frac{1}{2} >}{< \frac{1}{2}, \frac{1}{2} >} = \frac{1}{\pi}\int_0^{2\pi} f(t)dt$$

$$a_k = \frac{< f, \cos(kt) >}{< \cos(kt), \cos(kt) >} = \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(kt)dt \qquad (16.1)$$

$$b_k = \frac{< f, \sin(kt) >}{< \sin(kt), \sin(kt) >} = \frac{1}{\pi}\int_0^{2\pi} f(t)\sin(kt)dt$$

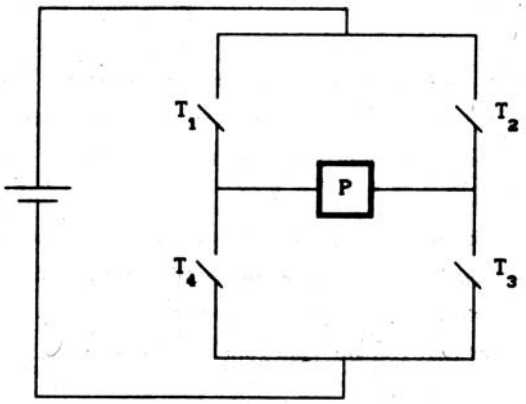
También gracias a la ortogonalidad, podemos expresar el error cuadrático como una suma de cuadrados

$$E^2 = \|f - s_n\|^2 = \|f\|^2 - \|s_n\|^2 = \|f\|^2 - (a_0^2/2 + a_1^2 + b_1^2 + a_2^2 + b_2^2 + \cdots + a_n^2 + b_n^2)$$

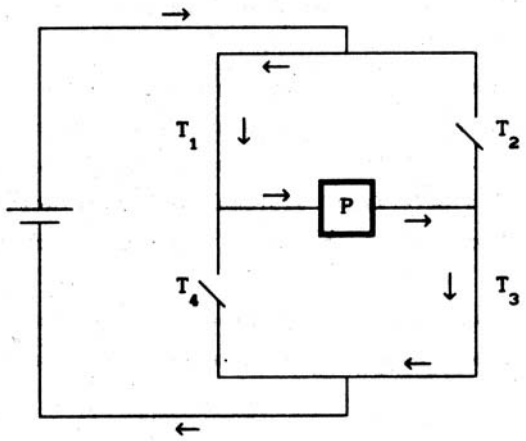
Ejemplo 1

Los sistemas de alimentación ininterrumpida (SAI) almacenan energía en baterías para suministrarla a determinados aparatos en caso de cortes del fluido eléctrico. Dichos aparatos suelen funcionar con corriente alterna, por lo que el SAI debe convertir la

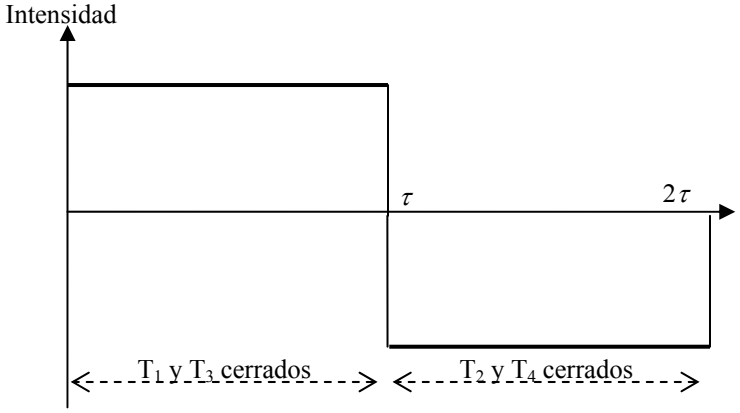
corriente continua que genera en corriente alterna. Los dispositivos que realizan esta conversión se denominan inversores. La figura muestra el esquema de un sencillo circuito inversor, llamado de puente.



La carga P está conectada a la batería a través de cuatro interruptores. Los interruptores T_1 y T_3 se cierran simultáneamente, lo mismo que T_2 y T_4, pero alternando su estado con los anteriores. Es decir, cuando los interruptores impares están cerrados, los pares están abiertos y viceversa. La figura siguiente muestra el paso de la corriente cuando los interruptores impares están cerrados. Al invertir el estado de los interruptores, se invierte el sentido de la tensión y la corriente en la carga.



Si durante un tiempo τ mantenemos la corriente en un sentido, la invertimos entonces durante un periodo igual, τ, y así sucesivamente, la intensidad en la carga presentará el aspecto de la figura.



La eficiencia de un circuito de este tipo puede analizarse teóricamente mediante la aproximación de Fourier. En efecto, si el aparato conectado al inversor está diseñado para trabajar con una corriente alterna de forma sinusoidal, aprovechará la parte de la intensidad recibida correspondiente al primer armónico del desarrollo de Fourier de dicha intensidad. Es importante obtener este armónico y calcular la potencia del resto de los armónicos, pues sus efectos son indeseables en todas las aplicaciones del inversor (perturbaciones, calentamientos, etc...); por lo que se impone su minimización o anulación mediante los blindajes y filtros adecuados.

Supongamos que el inversor produce una corriente unidad durante el intervalo de 0 a $\tau = \pi$ y una corriente opuesta en el intervalo siguiente de la misma duración, repitiéndose este ciclo indefinidamente. Dicha corriente se representa mediante la función 2π-periódica cuyos valores entre 0 y 2π son

$$f(t) = \operatorname{sign}(\pi - t) = \begin{cases} 1 & \text{si} \quad 0 < t < \pi \\ 0 & \text{si} \quad t = \pi \\ -1 & \text{si} \quad \pi < t < 2\pi \end{cases}$$

Aplicando las fórmulas de Euler (1), obtenemos una expresión general para el polinomio de Fourier de orden $2n$ de esta función. Observa que sólo contiene términos en seno por tratarse de una función *impar* $(f(-t) = -f(t))$.

$$s_{2n}(t) = \frac{4}{\pi}\left(\sin(t) + \frac{\sin(3t)}{3} + \frac{\sin(5t)}{5} + \cdots + \frac{\sin((2n-1)t)}{2n-1}\right)$$

Para el cálculo práctico de los coeficientes de Fourier, evaluaremos las integrales por el método de los trapecios, tomando suficientes nodos para obtener la precisión adecuada. Cuantos más términos del desarrollo de Fourier queramos obtener, más nodos utilizaremos en su cálculo. Trabajamos, por ejemplo, con 200 nodos para obtener el polinomio trigonométrico de orden 10.

```
m = 200;
h = 2*pi/m;
t = (0:h:2*pi)';
```

Evaluamos la función en los nodos y la representamos gráficamente.

```
y = sign(pi-t);
plot(t,y)
```

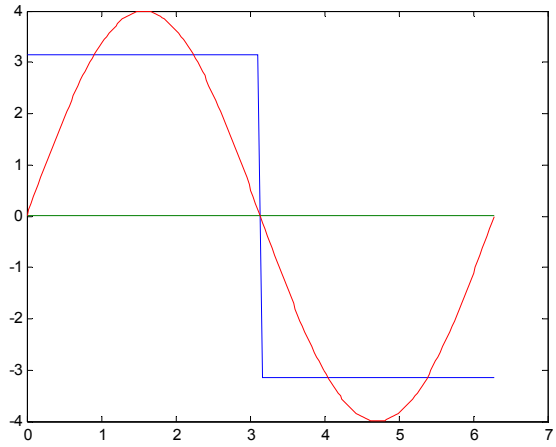
Calculamos el coeficiente del término constante del desarrollo y representamos éste gráficamente. Acumularemos en *s* los sucesivos términos del desarrollo.

```
a0 = trapz(t,y)/pi;
s = a0/2*ones(size(t));
plot(t,y,t,s)
```

En este caso, el término constante es nulo. Calculemos los siguientes coeficientes.

```
a(1) = trapz(t,y.*cos(t))/pi;
b(1) = trapz(t,y.*sin(t))/pi;
s = s+a(1)*cos(t)+b(1)*sin(t);
```

```
plot(t,y,t,s)
```

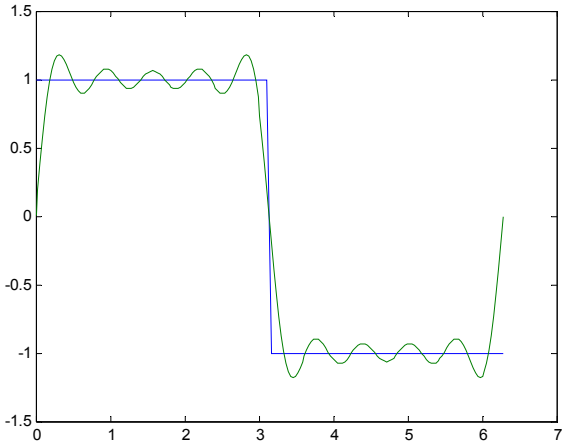


Observamos que el término en cos(t) es nulo, debido a que la función es impar y, por tanto, ortogonal al coseno. Los términos de orden 2 se anulan ambos, debido al distinto tipo de simetría con respecto a $\pi/2$ de la función y de sin($2t$) y cos ($2t$):

```
a(2) = trapz(t,y.*cos(2*t))/pi;
b(2) = trapz(t,y.*sin(2*t))/pi;
s = s+a(2)*cos(2*t)+b(2)*sin(2*t);
```

Evaluamos los términos restantes y representamos la aproximación.

```
for k=3:10
    a(k) = trapz(t,y.*cos(k*t))/pi;
    b(k) = trapz(t,y.*sin(k*t))/pi;
    s = s+a(k)*cos(k*t)+b(k)*sin(k*t);
end
plot(t,y,t,s)
```



Podemos hallar el error cuadrático directamente o a partir de los coeficientes.

```
E2 = trapz(t,(y-s).^2)/pi
E2 = trapz(t,y.^2)/pi-a0^2/2-norm(a)^2-norm(b)^2
```

```
E2 = 0.0721
E2 = 0.0721
```

Los resultados numéricos se aproximan bastante a los teóricos, aunque la precisión disminuye conforme aumenta el orden. En este ejemplo, la función a desarrollar es discontinua, por lo que el error es mayor.

```
b(1:2:9) - 4/pi./(1:2:9)

ans =  1.0e-003 *
   -0.1047   -0.3142   -0.5238   -0.7336   -0.9437
```

16.2. Transformada de Fourier Discreta

Al estimar los coeficientes de Fourier por el método de trapecios siempre con el mismo número de nodos, observamos que en las operaciones solamente intervienen los valores de la función a desarrollar y los de las funciones trigonométricas en un número finito de nodos equiespaciados. En realidad, estamos resolviendo un problema de aproximación discreta, en lugar de continua. En otras palabras, trabajamos en un espacio euclídeo de dimensión finita en el que el producto escalar se define mediante sumas y productos, en lugar de trabajar en un espacio de dimensión infinita, cuyo producto escalar es una integral.

Vamos a considerar el problema de aproximación trigonométrica desde este punto de vista, que se adapta mejor al cálculo con ordenador, así como a situaciones experimentales en las que no conocemos la función a aproximar, sino únicamente valores de la misma medidos a intervalos regulares.

Supongamos que f es una función de periodo 2π y que estimamos sus coeficientes de Fourier de modo aproximado mediante la fórmula de los trapecios con N intervalos. En el apartado anterior hemos tomado N suficientemente grande para que la precisión de los coeficientes hasta el orden n fuera buena. Conociendo analíticamente la función, podíamos evaluarla en tantos puntos como quisiéramos.

En la práctica, sin embargo, sucede que disponemos de un número limitado N de valores de f en el intervalo del periodo, $[0, 2\pi]$, y nos preguntamos cuántos coeficientes del desarrollo de Fourier podemos estimar a partir de dichos valores.

Sea $h = 2\pi / N$ y $t_j = jh$, para $j = 0, 1, 2, \ldots, N$. Denotamos por y_j el valor de f en t_j. Estimamos el coeficiente de Fourier a_k, $k = 0, 1, 2, \ldots$, de la función f mediante la fórmula de los trapecios con N subintervalos

$$a_k = \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(kt)dt \cong$$

$$\cong \frac{1}{N}\left(y_0\cos(kt_0) + 2y_1\cos(kt_1) + \cdots + 2y_{N-1}\cos(kt_{N-1}) + y_N\cos(kt_N)\right)$$

Al ser las funciones del integrando 2π-periódicas, el primer sumando y el último son iguales

$$a_k \cong A_k := \frac{2}{N}\left(y_0\cos(kt_0) + y_1\cos(kt_1) + \cdots + y_{N-1}\cos(kt_{N-1})\right)$$

La expresión entre paréntesis es el producto escalar en el espacio euclídeo N-dimensional de los vectores cuyas componentes son los valores de $f(t)$ y de $\cos(kt)$ en los puntos t_j, $j = 0, 1, 2, \ldots, N-1$. Llamaremos *funciones discretas* a estos vectores y las denotaremos igual que las correspondientes funciones reales.

Análogamente se aproximan los coeficientes en seno.

$$b_k = \frac{1}{\pi}\int_0^{2\pi} f(t)\sin(kt)dt \cong$$

$$\cong B_k := \frac{2}{N}\left(y_0\sin(kt_0) + y_1\sin(kt_1) + \cdots + y_{N-1}\sin(kt_{N-1})\right)$$

Vamos a comprobar que los n primeros sumandos de la expresión

$$T_n(t) = \frac{A_0}{2} + A_1\cos(t) + B_1\sin(t) + A_2\cos(2t) + B_2\sin(2t) + \cdots \qquad (16.2)$$

donde $n \le N$, proporcionan la proyección ortogonal de la función discreta $f(t)$ sobre el subespacio generado por n primeros vectores que aparecen en ella. La clave está en la ortogonalidad de estos vectores. Como el espacio es N-dimensional, a lo sumo hay N generadores independientes.

En primer lugar, comprobemos la ortogonalidad de las funciones trigonométricas discretas en los casos particulares de $N = 3$ y $N = 4$.

```
N = 3;
h = 2*pi/N;
t = (0:h:2*pi-h)';
M = [t.^0 cos(t) sin(t)];
G = M'*M

G =
    3.0000    0.0000   -0.0000
    0.0000    1.5000   -0.0000
   -0.0000   -0.0000    1.5000
```

La matriz de Gram $\boldsymbol{G}$ es diagonal, por lo que los vectores son ortogonales. El cuadrado de la norma de la función discreta constante 1 es obviamente $3 = N$, mientras que para los vectores restantes es $1.5 = N/2$.

En el caso $N = 4$ tenemos

```
N = 4;
h = 2*pi/N;
t = (0:h:2*pi-h)';
M = [t.^0 cos(t) sin(t) cos(2*t)];
G = M'*M

G =
    4.0000   -0.0000    0.0000         0
   -0.0000    2.0000    0.0000    0.0000
    0.0000    0.0000    2.0000    0.0000
         0    0.0000    0.0000    4.0000
```

En este caso notamos que el cuadrado de la norma del último vector, que es $(1,-1,1,-1)$, es $4 = N$, al igual que la del primero, $(1,1,1,1)$. La norma al cuadrado de los otros vectores es $2 = N/2$.

En general, si N es impar, el último vector es $\sin\left(\frac{N-1}{2}t\right)$ y el cuadrado de su norma, $N/2$. Si N es par, el último vector es $\cos\left(\frac{N}{2}t\right)$ y el cuadrado de su norma, N. La fórmula (16.2) proporciona las proyecciones ortogonales buscadas, ya que los coeficientes que aparecen en ella son:

$$A_k = \frac{2}{N}\left(y_0\cos(kt_0)+y_1\cos(kt_1)+\cdots+y_{N-1}\cos(kt_{N-1})\right)=$$
$$=\frac{\langle f(t),\cos(kt)\rangle}{\langle\cos(kt),\cos(kt)\rangle},\quad k=1,2,\ldots\lfloor(N-1)/2\rfloor$$

$$A_{N/2} = \frac{1}{N}\left(y_0\cos(\tfrac{N}{2}t_0)+y_1\cos(\tfrac{N}{2}t_1)+\cdots+y_{N-1}\cos(\tfrac{N}{2}t_{N-1})\right)=$$
$$=\frac{\langle f(t),(1,-1,\ldots,1,-1)\rangle}{\langle(1,-1,\ldots,1,-1),(1,-1,\ldots,1,-1)\rangle},\quad \text{si } N \text{ es par}$$

$$\frac{A_0}{2}=\frac{1}{N}\left(y_0+y_1+\cdots+y_{N-1}\right)=\frac{\langle f(t),1\rangle}{\langle 1,1\rangle}$$

$$B_k = \frac{2}{N}\left(y_0\sin(kt_0)+y_1\sin(kt_1)+\cdots+y_{N-1}\sin(kt_{N-1})\right)=$$
$$=\frac{\langle f(t),\sin(kt)\rangle}{\langle\sin(kt),\sin(kt)\rangle},\quad k=1,2,\ldots\lfloor(N-1)/2\rfloor$$

La aproximación discreta $T_N(t)$ de mayor orden coincide con el vector $f(t)$. Si f es una función de variable real t, la aproximación $T_N(t)$ es un polinomio trigonométrico que la interpola en los N nodos del muestreo.

Los términos de la aproximación tienen frecuencias que son múltiplo de la frecuencia fundamental. La frecuencia más alta que aparece es $N/2$ veces la frecuencia fundamental y se denomina frecuencia de Nyquist. Para que la transformada detecte frecuencias más altas, hay que tomar más valores de la función. En otros términos, hay que muestrear la señal a frecuencias mayores, concretamente al doble de la frecuencia máxima deseada.

Ejemplo 2

Para aproximar los tres primeros términos no triviales del desarrollo de Fourier de la función $f(t)=\text{sign}(\pi-t)$ del Ejemplo 1,

$$f(t)\approx\frac{4}{\pi}\left(\sin(t)+\frac{\sin(3t)}{3}+\frac{\sin(5t)}{5}\right)$$

hemos de tomar al menos $N = 11$, pues el último término es un seno y $(N-1)/2=5$. Tomaremos sin embargo $N = 12$, para preservar la simetría de la función. Ajustamos el valor de la función en 0 para que resulte periódica. Como el valor de f en 2π no interviene en el cálculo y la función tiene una discontinuidad, le asignamos en 0 el valor medio entre los valores por la izquierda y por la derecha.

```
N = 12;
h = 2*pi/N;
```

```
t = (0:h:2*pi-h)';
y = sign(pi-t);
y(1) = 0;
plot(t,y,'*')
```

Evaluamos las funciones base en los nodos t_k y hallamos los productos escalares.

```
M = [t.^0   cos(t)   sin(t)   cos(2*t)   sin(2*t) ...
        cos(3*t)   sin(3*t)   cos(4*t)   sin(4*t) ...
        cos(5*t)   sin(5*t)   cos(6*t)];
T = M'*y*2/N;
T(12) = T(12)/2;
```

Los coeficientes del término independiente A_0, de los cosenos A_k y de los senos B_k son, respectivamente,

```
A0 = T(1)
Ak = T(2:2:12)
Bk = T(3:2:11)

A0 =
     0
Ak =
  1.0e-015 *
    0.3701
         0
    0.5934
   -0.2405
   -0.3331
         0
Bk =
    1.2440
   -0.0000
    0.3333
   -0.0000
    0.0893
```

Los coeficientes en seno, A_k son prácticamente nulos. Los B_k no nulos corresponden a los coeficientes de los senos impares del desarrollo de Fourier ordinario

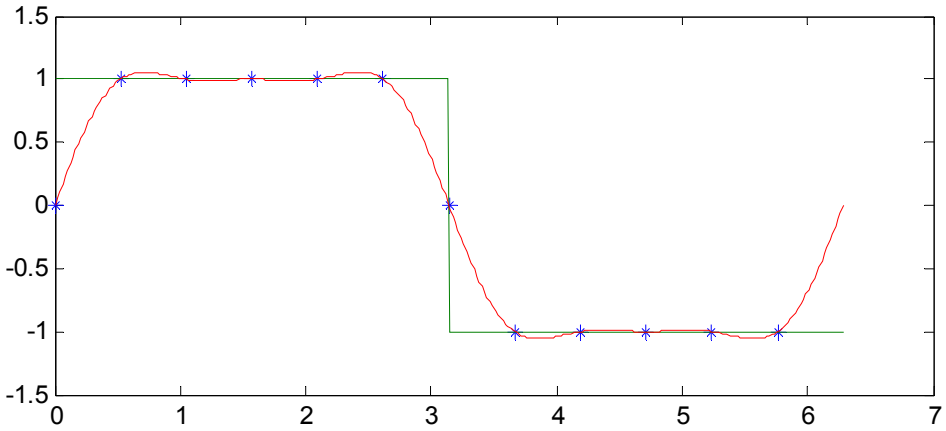
```
S = 4/pi*[ 1 1/3 1/5]

S =    1.2732    0.4244    0.2546
```

Salvo el primero, la aproximación es bastante pobre. Se puede alcanzar más precisión tomando N más elevado, como hemos visto en el Ejemplo 1, pero ahora lo que pretendemos es observar las propiedades de la aproximación discreta. Representamos primero la función y los nodos usados para hallar la transformada.

```
tg = linspace(0,2*pi,300);
yg = sign(pi-tg);
plot(t,y,'*',tg,yg)
```

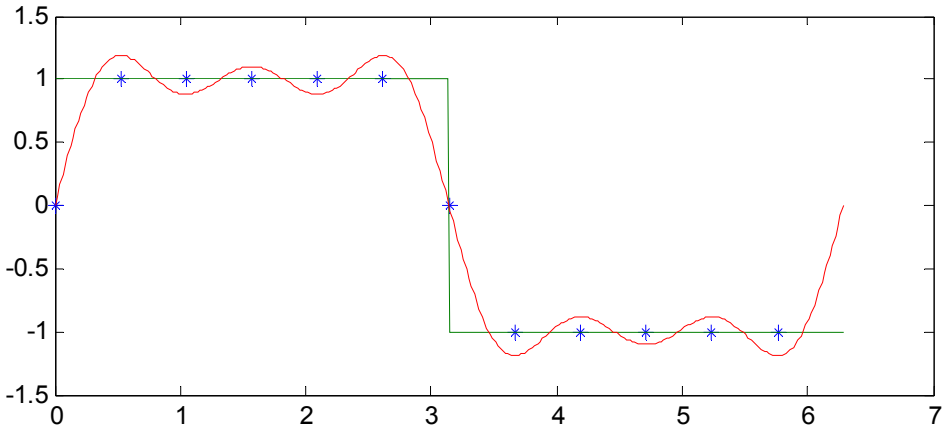
Evaluamos la transformada discreta.

```
z = T(1)/2*ones(size(tg));
for k=1:5
    z=z+T(2*k)*cos(k*tg);
    z=z+T(2*k+1)*sin(k*tg);
end
z=z+T(12)*cos(12*tg);
plot(t,y,'*',tg,yg,tg,z)
```



Comparamos ahora con el desarrollo de Fourier ordinario.

```
s = S(1)/2*ones(size(tg));
for k=1:5
    s=s+S(2*k)*cos(k*tg);
    s=s+S(2*k+1)*sin(k*tg);
end
plot(t,y,'*',tg,yg,tg,s)
```



Observamos que esta aproximación oscila más que la discreta, aunque tal vez ajuste mejor la función entorno a las discontinuidades.

16.3. Transformada rápida de Fourier (FFT)

La transformada de Fourier tiene innumerables aplicaciones y en la mayoría se necesitan valores de N elevados. El coste de la obtención de la transformada es del

orden de N^2, lo que no parece mucho en comparación con otros algoritmos como el de eliminación gaussiana o la factorización QR que son de orden N^3. Sin embargo, la popularidad de la transformada discreta se debe en gran parte al descubrimiento de algoritmos de menos coste, llamados rápidos, que permiten su aplicación a problemas de gran tamaño. El algoritmo de la transformada rápida de Fourier (FFT) tiene un coste del orden $N \cdot \log(N)$, que para N grande, es mucho menor que N^2.

Para introducir la FFT, es conveniente utilizar la versión compleja del desarrollo y de la transformada de Fourier discreta.

16.3.1. Desarrollo de Fourier complejo

Las funciones complejas $\left\{e^{ikt},\ k = 0, \pm 1, \pm 1, \ldots, \pm n\right\}$ forman un sistema ortonormal para el producto escalar definido por

$$< f, g >= \frac{1}{2\pi}\int_0^{2\pi} f(t)\,\overline{g(t)}\,dt$$

Dada una función $f(t)$ compleja, de variable real y periódica de periodo 2π, se define su desarrollo de Fourier complejo de orden n como

$$s_n(t) = \sum_{k=-n}^{n} c_k e^{ikt}$$

donde

$$c_k =< f, e^{ikt} >= \frac{1}{2\pi}\int_0^{2\pi} f(t)\, e^{-ikt}\ dt$$

En otros términos, s_n es la proyección ortogonal de f sobre el subespacio generado por las funciones $\left\{e^{ikt},\ k = 0, \pm 1, \pm 1, \ldots, \pm n\right\}$.

Si f toma valores reales, los coeficientes de su desarrollo de Fourier en senos y cosenos están relacionados con las del desarrollo complejo mediante las expresiones

$$c_k = \overline{c_{-k}} = \frac{a_k - ib_k}{2}$$

que permiten pasar fácilmente del desarrollo real al complejo y viceversa.

16.3.2. Transformada de Fourier discreta

Los coeficientes de la transformada discreta de N valores de una función 2π-periódica $y_j = f(t_j) = f(jh) = f(j\frac{2\pi}{N})$ se definen como

$$C_k = \frac{1}{N}\sum_{j=0}^{N-1} y_j \exp(-ikt_j) = \frac{1}{N}\sum_{j=0}^{N-1} y_j \exp\left(-ikj\frac{2\pi}{N}\right), \text{ para } k = 0, 1, \ldots, N-1.$$

Estos coeficientes permiten expresar el vector y como combinación lineal de exponenciales complejas ortogonales

$$y_j = \frac{1}{N}\sum_{j=0}^{N-1} C_k \exp\left(ikj\frac{2\pi}{N}\right)$$

Si f es real, los coeficientes C_k están relacionados con los de su transformada de Fourier discreta en senos y cosenos mediante las expresiones

$$C_0 = \frac{A_0}{2}, \qquad C_k = \frac{A_k - iB_k}{2}, \qquad 1 \le k < N/2$$

Si N es par, $C_{N/2} = A_{N/2} - iB_{N/2}$. Los restantes coeficientes son los conjugados de los anteriores por lo que pueden ignorarse, concretamente

$$C_{N-k} = \frac{A_k + iB_k}{2}, \qquad 1 \le k \le N/2.$$

Recíprocamente,

$$\begin{aligned} A_k &= 2\,\mathrm{real}(C_k) && 0 \le k < N/2 \\ A_{N/2} &= \mathrm{real}(C_{N/2}) && \text{si } N \text{ es par} \\ B_k &= -2\mathrm{imag}(C_k), && 1 \le k < N/2 \end{aligned}$$

16.3.3. Transformada rápida de Fourier

El cálculo de la transformada de Fourier discreta se optimiza mediante el algoritmo FFT, programado en una función de MATLAB del mismo nombre. Aplicando esta función a un vector y, se obtiene un vector de elementos NC_k, donde N es el número de componentes de y y C_k son los coeficientes de la transformada de Fourier discreta compleja. A partir de ellos obtenemos los coeficientes de la transformada real, que a su vez estiman los coeficientes del desarrollo de Fourier ordinario.

Ejemplo 3

Calculemos la transformada rápida de 12 muestras de la función $f(t) = \mathrm{sign}(\pi - t)$ en el intervalo $[0, 2\pi]$ y comparemos los resultados con los del ejemplo anterior.

```
N = 12;
h = 2*pi/N;
t = 0:h:2*pi-h;
y = sign(pi-t);
y(1) = 0;
C = fft(y)/N
```

```
C =
  Columns 1 through 3
        0                  0 - 0.6220i        0
  Columns 4 through 6
        0 - 0.1667i        0                  0 - 0.0447i
  Columns 7 through 9
        0                  0 + 0.0447i        0
  Columns 10 through 12
        0 + 0.1667i        0                  0 + 0.6220i
```

Todas las componentes tienen parte real nula, correspondiendo a la anulación de los coeficientes en coseno de la transformada discreta. En general, podemos obtener estos coeficientes a partir de los 6 primeros coeficientes complejos. En el caso real, los restantes coeficientes contienen información redundante.

```
A0 = 2*C(1);
Ak = 2*real(C(2:6));
```

Los coeficientes en seno son

```
Bk = -imag(C(2:6))

Bk =  0.6220         0    0.1667         0    0.0447
```

La inversión de la transformada discreta compleja es fácil con MATLAB, obteniéndose el vector inicial.

```
z = zeros(size(t));
for k=1:12
    z=z+C(k)*exp(i*(k-1)*t);
end
z

z =
  Columns 1 through 3
        0             1.0000 - 0.0000i   1.0000 - 0.0000i
  Columns 4 through 6
   1.0000 - 0.0000i   1.0000 - 0.0000i   1.0000 - 0.0000i
  Columns 7 through 9
  -0.0000            -1.0000 + 0.0000i  -1.0000 + 0.0000i
  Columns 10 through 12
  -1.0000 - 0.0000i  -1.0000 + 0.0000i  -1.0000 + 0.0000i
```

16.4. Problemas

16.4.1. Enunciados

1.- Comprueba la ortogonalidad del sistema trigonométrico 1/2, sin(x), cos(x), sin($2x$), cos($2x$), ..., sin(nx), cos(nx), ... con respecto al producto escalar integral

$$< f, g > = \frac{1}{\pi} \int_0^{2\pi} f(t)\, g(t)\, dt\, .$$

2.- Los dispositivos que transforman la corriente alterna en continua se denominan rectificadores. El circuito rectificador más elemental contiene de un diodo que deja pasar la corriente en un sentido mientras que impide que circule en sentido opuesto. Sometido el diodo a una tensión sinusoidal, la corriente por el circuito es una sinusoide en la que se ha anulado la parte negativa. Estudia la aproximación de Fourier continua y discreta de la función $f(t) = \begin{cases} \sin(t) & 0 < t < \pi \\ 0 & \pi < t < 2\pi \end{cases}$ que da la corriente por el circuito.

3.- El barrido del haz de electrones por la pantalla de un tubo de rayos catódicos se consigue mediante un campo magnético variable en forma de diente de sierra, $f(t) = t/2, \quad -\pi < t < \pi.$
Representa gráficamente diversas aproximaciones periódicas de esta función.

4.- Aproxima la función de Runge $f(t)=\dfrac{1}{1+t^2}$, $-\pi<t<\pi$ mediante polinomios trigonométricos.

16.4.2. Soluciones

1.- Comprobemos en primer lugar la ortogonalidad de la función constante con respecto a los senos y cosenos.

$$<\tfrac{1}{2},\cos(kt)>=\frac{1}{\pi}\int_0^{2\pi}\frac{\cos(kt)}{2}\,dt=\left.\frac{\sin(kt)}{2k\pi}\right|_0^{2\pi}=0,\quad k=1,2,\ldots$$

$$<\tfrac{1}{2},\sin(kt)>=\frac{1}{\pi}\int_0^{2\pi}\frac{\sin(kt)}{2}\,dt=-\left.\frac{\cos(kt)}{2k\pi}\right|_0^{2\pi}=0,\quad k=1,2,\ldots$$

Las integrales se anulan por ser el sen(kt) y cos(kt) 2π-periódicas. Comprobemos la ortogonalidad de los cosenos. Aplicamos la fórmula de transformación de productos de funciones trigonométricas en suma.

$$<\cos(jt),\cos(kt)>=\frac{1}{\pi}\int_0^{2\pi}\cos(jt)\cos(kt)\,dt=$$

$$=\frac{1}{\pi}\int_0^{2\pi}\frac{\cos((j+k)t)+\cos((j-k)t)}{2}\,dt=\begin{cases}0, & \text{si } j\neq k\\ \dfrac{1}{\pi}\displaystyle\int_0^{2\pi}\frac{1}{2}\,dt=1, & \text{si } j=k\end{cases}$$

Análogamente se prueba para los senos.

$$<\sin(jt),\sin(kt)>=\frac{1}{\pi}\int_0^{2\pi}\sin(jt)\sin(kt)\,dt=$$

$$=\frac{1}{\pi}\int_0^{2\pi}\frac{\cos((j-k)t)-\cos((j+k)t)}{2}\,dt=\begin{cases}0, & \text{si } j\neq k\\ \dfrac{1}{\pi}\displaystyle\int_0^{2\pi}\frac{1}{2}\,dt=1, & \text{si } j=k\end{cases}$$

La ortogonalidad entre senos y cosenos se comprueba igualmente, o de forma directa teniendo en cuenta que los senos son impares y los coseno pares.

$$<\sin(jt),\cos(kt)>=\frac{1}{\pi}\int_0^{2\pi}\sin(jt)\cos(kt)\,dt=$$

$$=\frac{1}{\pi}\int_0^{2\pi}\frac{\sin((j+k)t)-\sin((j-k)t)}{2}\,dt=0$$

2.- Calculemos los coeficientes del desarrollo de Fourier

$$a_0=\frac{1}{\pi}\int_0^{2\pi}f(t)dt=\frac{1}{\pi}\int_0^{\pi}\sin(t)dt=-\frac{1}{\pi}(\cos(\pi)-\cos(0))=\frac{2}{\pi}$$

$$a_k = \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(kt)dt = \frac{1}{\pi}\int_0^{\pi} \sin(t)\cos(kt)dt =$$

$$= \frac{1}{\pi}\int_0^{\pi} \frac{\sin((k+1)t)-\sin((k-1)t)}{2}dt = \frac{1}{2\pi}\left[-\frac{\cos((k+1)t)}{k+1}\Big|_0^{\pi} + \frac{\cos((k-1)t)}{k-1}\Big|_0^{\pi}\right] =$$

$$= \begin{cases} 0 & \text{si } k \text{ impar} \\ \frac{1}{2\pi}\left(\frac{2}{k+1}-\frac{2}{k-1}\right) = \frac{-2}{\pi}\frac{1}{k^2-1} & \text{si } k \text{ par} \end{cases}$$

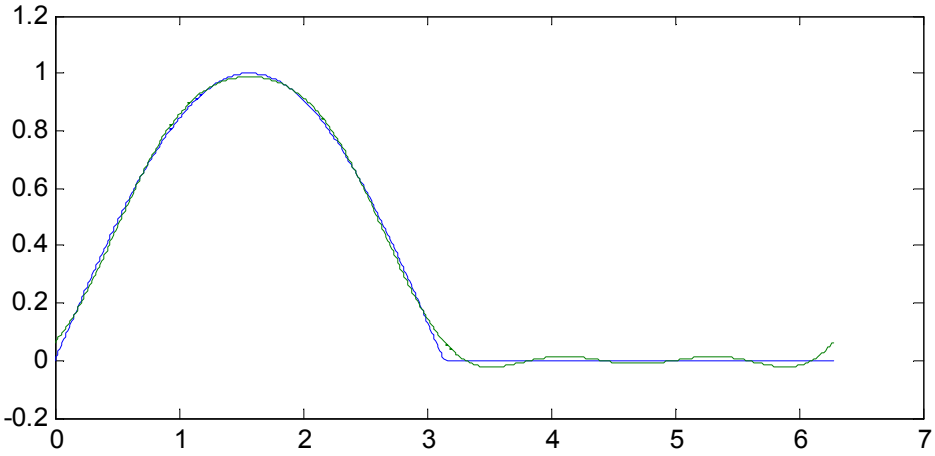
$$b_1 = \frac{1}{\pi}\int_0^{2\pi} f(t)\sin(t)dt = \frac{1}{\pi}\int_0^{\pi} \sin^2(t)dt = \frac{1}{\pi}\int_0^{\pi} \frac{1-\cos(2t)}{2}dt = \frac{1}{\pi}\frac{\pi}{2} = \frac{1}{2}$$

$$b_k = \frac{1}{\pi}\int_0^{2\pi} f(t)\sin(kt)dt = \frac{1}{\pi}\int_0^{\pi} \sin(t)\sin(kt)dt =$$

$$= \frac{1}{\pi}\int_0^{\pi} \frac{\cos((k-1)t)-\cos((k+1)t)}{2}dt = \frac{1}{2\pi}\left[\frac{\sin((k-1)t)}{k-1}\Big|_0^{\pi} - \frac{\sin((k+1)t)}{k+1}\Big|_0^{\pi}\right] = 0$$

El polinomio de Fourier de orden $2n$ es

$$s_n(t) = \frac{\sin(t)}{2} + \frac{2}{\pi}\left(\frac{1}{2} - \frac{\cos(2t)}{3} - \frac{\cos(4t)}{15} - \frac{\cos(6t)}{35} - \cdots - \frac{\cos(2nt)}{(2n)^2-1}\right)$$



La figura muestra el seno rectificado y su aproximación de Fourier de orden 4.

Estimamos ahora los coeficientes de esta aproximación mediante la transformada rápida de Fourier. Como el coeficiente en sen($4t$) es nulo, basta tomar 8 puntos.

```
N = 8;
h=2*pi/8;
t = 0:h:2*pi-h;
y = (t<pi).*sin(t);
C = fft(y)/N

C =
```

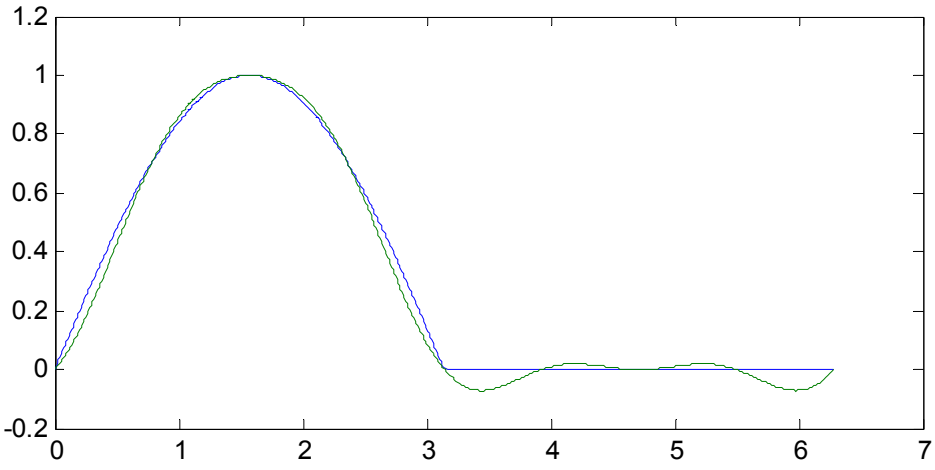
```
  Columns 1 through 3
   0.3018            -0.0000 - 0.2500i  -0.1250 + 0.0000i
  Columns 4 through 6
   0.0000            -0.0518             0.0000
  Columns 7 through 8
  -0.1250 - 0.0000i  -0.0000 + 0.2500i
```

Obtengamos de aquí los coeficientes de la transformada discreta.

```
A0 = 2*C(1)
Ak(1:3) = 2*real(C(2:4));
Ak(4) = real(C(5))
Bk = -2*imag(C(2:5))

A0 =
    0.6036
Ak =
   -0.0000   -0.2500    0.0000   -0.0518
Bk =
    0.5000   -0.0000         0         0
```

Los valores correspondientes del desarrollo analítico antes calculado son bastante aproximados a los de la transformada discreta.

```
a0 = 2/pi
a = 2/pi*[0 -1/3 0 -1/15]
b = [1/2 0 0 0]

a0 =   0.6366
a =         0   -0.2122         0   -0.0424
b =    0.5000         0         0         0
```



En el gráfico de la aproximación discreta se aprecia la diferencia respecto a la aproximación analítica.

4.- a) Coeficientes del desarrollo de Fourier.

En primer lugar, notemos que al tratarse de una función par, o sea, que $f(-t) = f(t)$, el desarrollo sólo tendrá términos en coseno. Estimamos los coeficientes del desarrollo

de Fourier mediante la fórmula de trapecios con un número relativamente alto de intervalos, pongamos 500.

```
m = 500;
h = 2*pi/m;
t = (-pi:h:pi)';
y = 1./(1+t.^2);
```

Calculamos el coeficiente del término constante del desarrollo y Acumularemos en *s* los sucesivos términos del desarrollo.

```
a0 = trapz(t,y)/pi
s = a0/2*ones(size(t));
```

```
a0 = 0.8038
```

Calculamos los siguientes coeficientes, recordando que los b_k son nulos.

```
for k=1:5
    a(k) = trapz(t,y.*cos(k*t))/pi;
    s = s+a(k)*cos(k*t);
end
```

Los coeficientes en coseno son

```
a
```

```
a =  0.3889    0.1282    0.0532    0.0163    0.0080
```

El polinomio de Fourier resultante se confunde casi con la función.

b) Transformada de Fourier discreta
Obtendremos los mismos coeficientes con distintos valores de *N* para observar cómo se aproximan a los del desarrollo analítico. Calculamos únicamente los coeficientes es las funciones pares.

```
N = 12;
h = 2*pi/N;
t = (-pi:h:pi-h)';
y = 1./(1+t.^2);
M = [t.^0   cos(t)    cos(2*t)    cos(3*t) ...
          cos(4*t)    cos(5*t)    cos(6*t)];
T = M'*y*2/N;
T(7) = T(7)/2
```

```
T =
    0.8031
    0.3897
    0.1275
    0.0542
    0.0157
    0.0101
    0.0013
```

La tabla adjunta muestra cómo se estabilizan los valores calculados al aumentar N.

Coef	$N = 12$	$N = 60$	$N = 120$
A_0	0.80305543116051	0.80378254202805	0.80381223853720
A_1	0.38969093028561	0.38892207380461	0.38889236036422
A_2	0.12745841540200	0.12819495390619	0.12822471822888
A_3	0.05423075163348	0.05326441580220	0.05323456637931
A_4	0.01567242107834	0.01627556617975	0.01630553536884
A_5	0.01007848390572	0.00807805792346	0.00804793366750
A_6	0.00134128211459	0.00076457501605	0.00077973274210

c) Transformada rápida

Para obtener la transformada mediante el algoritmo rápido, hay que evaluar la función en el intervalo $[0, 2\pi]$. Por la periodicidad, los valores de π a 2π son los mismos que los de $-\pi$ a 0. Basta cortar el vector y en dos trozos y pegarlos cambiando el orden.

```
n=floor((N+1)/2);
yb = y([n+1:N,1:n]);
fft(yb)*2/N

ans =
   0.8031
   0.3897 + 0.0000i
   0.1275
   0.0542
   0.0157
   0.0101 + 0.0000i
   0.0027
   0.0101 - 0.0000i
   0.0157
   0.0542
   0.1275
   0.3897 - 0.0000i
```

De las 7 primeras componentes se obtienen los coeficientes de la transformada discreta.

17 Ecuaciones Diferenciales Ordinarias

Los problemas de valor inicial de ecuaciones y sistemas de ecuaciones diferenciales suelen resolverse por métodos directos, a diferencia de los problemas de frontera, que suelen reducirse a la resolución de sistemas algebraicos, lineales o no.
Esta Práctica aborda los métodos numéricos para el problema de valor inicial de ecuaciones diferenciales ordinarias de primer orden. En la siguiente, aplicamos estos métodos a las ecuaciones diferenciales de orden superior al primero y a los sistemas de ecuaciones diferenciales de primer orden.
Formulamos el llamado *Problema de Valor Inicial* para la ecuación diferencial de primer orden e indicamos su aplicación a los *Modelos de Población*, que usaremos como ejemplos ilustrativos de los métodos numéricos.
La interpretación geométrica de una ecuación diferencial con un *campo de direcciones* que siguen las curvas solución, nos proporciona una idea intuitiva del método de las *poligonales de Euler*, pero otros métodos más sofisticados se obtienen más fácilmente por vía analítica, a partir de la formulación de la ecuación diferencial como una ecuación integral. La evaluación numérica de esta integral por distintas fórmulas de cuadratura da lugar a los métodos básicos de resolución de ecuaciones diferenciales.
Programaremos los métodos de *Euler, Heun, Euler modificado y Runge-Kutta* y los aplicaremos a algunos ejemplos, comprobando el orden de error de cada uno de ellos.

17.1 Problemas de valor inicial

El Problema de Valor Inicial (PVI) consiste en hallar la solución y(t) de la ecuación diferencial ordinaria de primer orden

$$y'(t) = f(t, y(t)), \quad a \le t \le b,$$

que en t = a toma un valor determinado y_0. Bajo ciertas condiciones, se demuestra que el PVI tiene solución única. Sin embargo, la solución casi nunca puede obtenerse analíticamente mediante una fórmula cerrada. Afortunadamente, disponemos de diversos métodos numéricos que proporcionan aproximaciones de la solución con la precisión requerida en las aplicaciones. Los ejemplos que analizamos sí tienen solución cerrada, lo que nos permite estudiar el error de los métodos numéricos desarrollados.

Ejemplo 1: Modelos de población de Malthus y Verhulst.

Malthus supone que la velocidad de crecimiento de una población es proporcional al tamaño de la misma en cada momento. Si llamamos a al coeficiente de proporcionalidad, p a la población y p' a su derivada respecto al tiempo, la ecuación resultante es p' = a·p
Si p_0 es la población en es tiempo t = 0, la solución del PVI es

$$p = p_0 \cdot e^{at}$$

El modelo de Malthus se aplica con bastante precisión a poblaciones pequeñas con recursos prácticamente ilimitados o a otros fenómenos como la desintegración radiactiva (con a < 0). Sin embargo, cuando los recursos de que dispone la población son limitados, su crecimiento se ve afectado por la competencia por conseguirlos entre los miembros de la población. Verhulst introduce en la ley de Malthus una corrección proporcional al número de encuentros que pueden producirse entre los miembros de la población en la búsqueda de los recursos, quedando la ecuación diferencial como

$$p' = a \cdot p - bp^2$$

Notemos que el modelo de Malthus es un caso particular del de Verhulst con b = 0. La solución del PVI correspondiente al modelo de Verhulst es

$$p(t) = \frac{a\ p_0}{b\ p_0 + (a - b\ p_0)e^{-at}}$$

17.2 Campo de direcciones

Si representamos en el plano cartesiano las soluciones de una ecuación diferencial para distintos valores iniciales, obtenemos curvas cuya pendiente en cada punto (t, y) es el valor de f en dicho punto.

Si no conocemos las soluciones, la ecuación diferencial proporciona sus pendientes en cada punto del plano. Representándolas obtendremos la forma aproximada de dichas soluciones.
MATLAB dispone de una función, *quiver*, apropiada para visualizar el campo de direcciones de una ecuación diferencial.
En primer lugar, editamos un fichero f.m que devuelva los valores de la ecuación diferencial del ejemplo 4

function z = verhulst (t,p)

```
a = 4; b = 1;
z = a*p - b*p.^2;
```

Para representar su campo de direcciones hacen falta dos matrices con las coordenadas de los nodos en los que trazar las direcciones y otras dos con las coordenadas de vectores que en cada punto tienen la pendiente deseada. Los vectores son escalados convenientemente por la función *quiver*.

```
t = 0:.2:3; y = 0:.2:5;
[tt,yy] = meshgrid(t,y);
uno = ones(size(tt));
dy = f(tt,yy);
quiver(tt,yy,uno,dy)
```

Siguiendo las flechas visualizamos la forma de las soluciones.

17.3 Métodos Numéricos para el Problema de Valor Inicial

Los métodos numéricos no dan una expresión analítica de la función que permitiría evaluarla en cualquier **t**, sino valores aproximados de la misma en un número finito de instantes

$$a = t_0 < t_1 < \ldots < t_n = b$$

Los t_k se consideran como nodos de integración y suelen tomarse equiespaciados para simplificar la aplicación del método. La distancia entre nodos consecutivos es el *paso* de integración h. Denotamos por y_0, y_1, ..., y_n las aproximaciones calculadas de los valores de la solución en los nodos correspondientes, $y(t_0)$, $y(t_1)$, ..., $y(t_n)$. Para obtenerlas, expresamos el problema de valor inicial en la forma

$$y(t) = y(t_0) + \int_{t_0}^{t} f(s, y(s))ds$$

y evaluamos numéricamente la integral del segundo miembro.
Curiosamente, esta ecuación integral es la base tanto para la demostración teórica de la existencia y unicidad de la solución del problema de valor inicial, como para la aproximación numérica de las mismas. Por el contrario, los métodos analíticos obtienen la solución mediante transformaciones efectuadas sobre la forma diferencial del problema de valor inicial.
En el caso teórico, se considera la ecuación integral como una ecuación de punto fijo de incógnita y(t) y se itera sustituyendo a la derecha la estimación inicial $y_0(t) = y_0$. El resultado se asigna a $y_1(t)$, que es el siguiente iterado, y así hasta obtener la convergencia. El límite es la solución buscada. En el caso numérico, la evaluación de la integral mediante fórmulas de cuadratura ha de enfrentar la dificultad de que en el integrando aparece la función y(t) que se trata de obtener. Para resolverla, algunos métodos hacen en cada paso una *predicción* del valor buscado y luego integran de nuevo para *corregir* esta predicción.

Para analizar la precisión de un método numérico, comparamos la solución obtenida con la exacta para problemas en que ésta puede determinarse analíticamente.

Cuando no se dispone de la solución exacta, se comparan dos soluciones aproximadas, obtenidas con paso distinto.

Tomamos como medida del error el máximo de los errores locales, definidos éstos como la distancia en cada nodo entre la solución exacta y la aproximada.

Se dice que un método es *convergente* si el error máximo tiende a 0 cuando el paso se hace pequeño. Además de ser convergente, interesa que un método se aproxime lo más rápidamente posible al límite. Comparamos esta rapidez con la tendencia a cero de las potencias de h.
Decimos que un método es de orden p, si el error máximo tiende a cero a la misma velocidad que h^p, cuando h tiende a 0.

Ejemplo 2: (Continuación de ejemplo 1)

Codifiquemos la solución exacta del modelo de población de Verhulst:

$$p(t) = \frac{a\ p_0}{b\ p_0 + (a - b\ p_0)e^{-at}}$$

function p = logistic(t,p0)

```
a = 4; b = 1;
p = a*p0./(b*p0+(a-b*p0)*exp(-a*t));
```

Guardamos la función en un fichero *logistic.m* que utilizaremos para hallar los errores de los distintos métodos.

17.4 Método de Euler

El método de Euler aproxima la integral de forma bastante rudimentaria. Para estimar $y(t_1)$, hemos de evaluar la integral

$$\int_{t_0}^{t_1} f(t, y(t))dt$$

en el intervalo $[t_0, t_1]$, pero sólo conocemos el integrando en el punto t_0. Supondremos pues que el integrando toma en todo el intervalo el valor $f(t_0, y_0)$, con lo que la integral buscada se aproxima por

$$h \cdot f(t_0, y_0).$$

Llevando este valor al segundo miembro de la ecuación integral, obtenemos la estimación de la solución en el nodo t_1:

$$y_1 = y_0 + h \cdot f(t_0, y_0).$$

Repitiendo este proceso sucesivamente para cada nodo, obtenemos las aproximaciones de la solución en los nodos del intervalo, por el llamado método de Euler.
Para k = 0, 1, 2, ...,n-1 hacemos

$$y_{k+1} = y_k + h \cdot f(t_k, y_k)$$

El siguiente algoritmo calcula estos valores y los almacena en un vector **y**.

Algoritmo de Euler

- Entrada: *a, b, y_0, n*

- Proceso: Paso de integración *h = (b–a)/n*
  Nodos de integración *t = (a : h : b)'*
  valor inicial *y(1) = y_0*
  for *k = 1 : n*
  y(k+1) = y(k) + h f(t(k), y(k))*

end

- Salida: *t, y*

Codificamos el algoritmo de Euler en MATLAB y lo guardamos en *mieuler.m* (no se puede usar euler.m porque es una función del sistema).

Es conveniente inicializar *y* como un vector columna de *n+1* ceros. Así ahorraremos tiempo en las asignaciones de cada iteración.

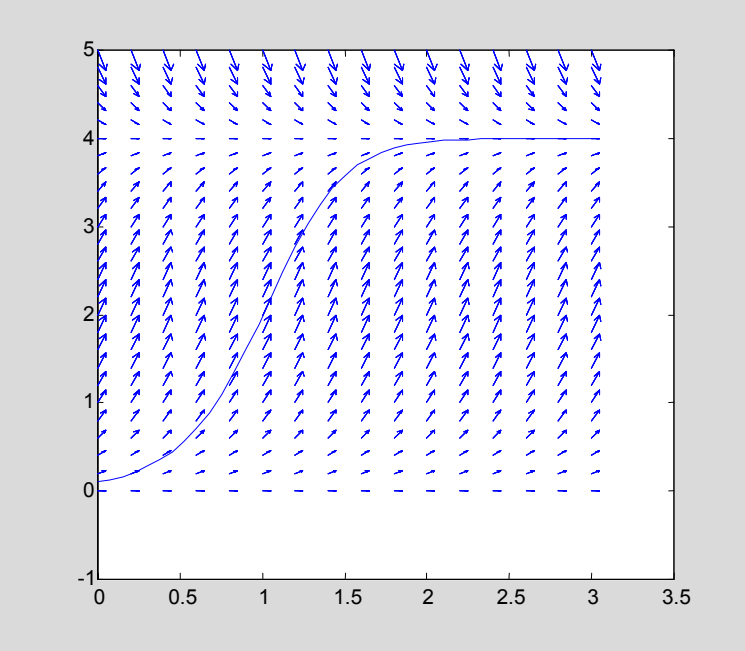
```
y = zeros(n+1,1)
```

Almacenamos los nodos y las iteraciones en columna para facilitar la representación gráfica de la solución de sistemas de ecuaciones diferenciales que veremos en la Práctica siguiente. Para resolver el modelo de población, ejecutamos

```
[t,y] = mieuler(0,3,0.1,20)
```

y representamos la salida sobre el campo de direcciones anterior:

```
quiver(tt,yy,uno,dy)
```




```
hold
plot(t,y)
```

Si hemos puesto la solución exacta en el fichero *logistic.m*, calculamos el error mediante

```
max(abs(y-logistic(t,0.1)))
```

```
ans =
    0.6616
```

Comprobamos que duplicando el número de subintervalos, el error se divide aproximadamente por 2. El método de Euler es de orden *h*.

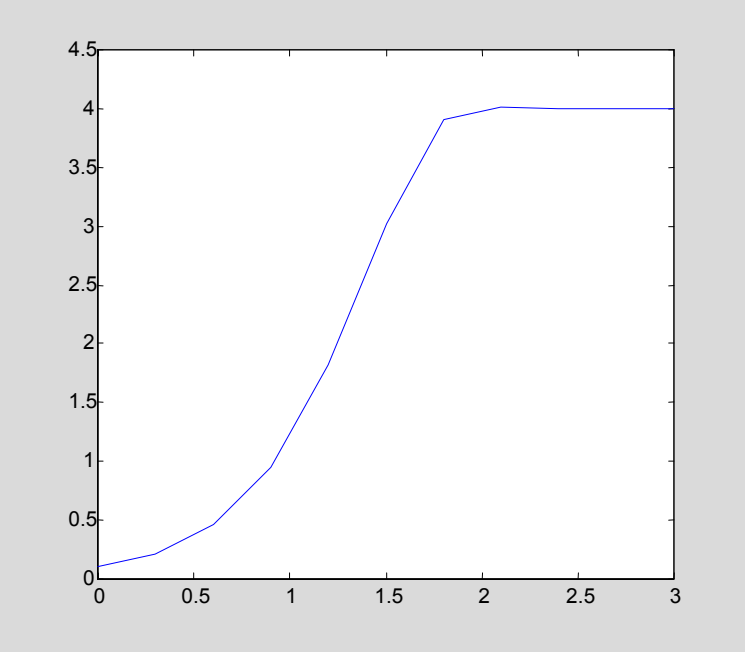
Ejemplo 3

```
[t,y] = mieuler(0,3,0.1,40)
```

Tomado menos intervalos, 10 por ejemplo, se advierte que el método se hace inestable, produciéndose oscilaciones en el resultado.

```
[t,y] = mieuler(0,3,0.1,10)
```

```
plot(t,y)
```



Repetimos el proceso con distintos valores iniciales para comprobar que la solución tiende al valor límite *a/b*, partiendo de un valor positivo, y a 0 si la población inicial es nula.

17. 3 Método de Heun

Para mejorar la precisión del método de Euler, basta aproximar la integral en cada paso mediante un método más preciso, como puede ser el de los trapecios.

Sin embargo, tenemos el inconveniente de que, por ejemplo en el primer paso, no conocemos $f(t_1, y_1)$, dado que y_1 es justamente el resultado de este paso. A falta del valor de y_1, lo estimamos por Euler:

$$y_1^p = y_0 + h \cdot f(t_0, y_0)$$

Ahora ya podemos aplicar la fórmula de trapecios y corregir el valor de y_1.

$$y_1 = y_0 + h/2 \cdot \left(f(t_0, y_0) + f(t_0, y_1^p) \right)$$

Análogamente obtenemos y_2 a partir de y_1, prediciendo primero un valor por Euler y ajustándolo después por trapecios. En general, el paso k consiste en:

a) Predecir un valor de y_{k+1} mediante

$$y_{k+1}^p = y_k + h \cdot f(t_k, y_k)$$

b) Corregir la predicción mediante trapecios

$$y_{k+1} = y_k + h/2 \cdot \left(f(t_k, y_k) + f(t_k + h, y_{k+1}^p) \right)$$

Obtenemos así el llamado *método de Heun*. La estructura del algoritmo correspondiente es igual a la del algoritmo de Euler, variando sólo la forma de obtener el paso siguiente.

Algoritmo del método de Heun

Para	*k = 1 : n*
	k1 = f(t(k), y(k))
Predicción	*yp = y(k) + h*k1*
	k2 = f(t(k)+h, yp)
Corrección	*y(k+1) = y(k) + h/2*(k1+k2)*
fin para	*k*

Codificamos el algoritmo de Heun en MATLAB, modificando el fichero de Euler, del que sólo se diferencia en la forma de obtener la **y** en cada paso, y lo guardamos en *heun.m.*

Ejemplo 4: Caída en un medio resistente

Un proyectil de masa $m = 0.11\ kg$ se lanza verticalmente hacia arriba en el aire con una velocidad inicial de $8\ m/s$. Si la resistencia del aire es proporcional al cuadrado de la velocidad, se satisface la ecuación diferencial

$$v' = -g - k \cdot v \cdot |v| / m$$

donde $g = 9.8\ m/s^2$ y $k = 0.02\ kg \cdot m^{-1}$ es el coeficiente de resistencia del aire.
a) Resuelve el PVI en el intervalo *[0,1]* con $h = 0.1$, por el método de Heun.
b) Representa la solución **v** gráficamente e interpreta físicamente el resultado obtenido. ¿Hacia qué tiende **v**?

Solución
Codificamos la ecuación diferencial en *f.m*

```
function z = proyectil(t,y)
k = 0.002; m = 0.11; g = 9.8;
z = -g - k/m*y.*abs(y);
```

En el intervalo *[0,1]* la velocidad parece decrecer linealmente como si fuese un movimiento de caída uniformemente acelerado. La velocidad tendería a $-\infty$.
Al tomar un intervalo mayor observamos que la velocidad tiende a cierto límite y el proyectil cae finalmente con movimiento uniforme, a velocidad constante.

a) `[t,y]=heun(0,1,8,10)`

```
t =
         0
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
    0.6000
    0.7000
    0.8000
    0.9000
    1.0000
y =
    8.0000
    6.9185
    5.8638
    4.8315
    3.8170
    2.8166
    1.8264
    0.8427
   -0.1379
   -1.1168
   -2.0916
```

17.5 Método de Euler Modificado

Utilizando como método de integración el del punto medio, obtenemos el llamado *método de Euler modificado* para ecuaciones diferenciales.

Para evaluar la integral en un intervalo, pongamos por ejemplo en $[t_0, t_1]$, necesitamos el valor de y en el punto medio del mismo, $t_0 + h/2$. Como aún no disponemos de él, lo aproximamos por Euler

$$y_{1/2} = y_0 + h/2 \cdot f(t_0, y_0)$$

Con esta aproximación, evaluamos la integral en $[t_0, t_1]$ por la fórmula de punto medio que da

$$h \cdot f(t_0 + h/2, y_{1/2})$$

El primer paso del método modificado de Euler queda pues como

$$y_1 = y_0 + h \cdot f(t_0 + h/2, y_{1/2})$$

En la etapa k , para k = 0, 1, 2, ...n-1, obtenemos y_{k+1} a partir de y_k mediante las operaciones

$$y_{k+1/2} = y_k + h/2 \cdot f(t_k, y_k)$$

$$y_{k+1} = y_k + h \cdot f(t_k + h/2, y_{k+1/2})$$

El método de Euler modificado tiene error de orden 2, como el de Heun.

Algoritmodel método de Euler modificado

Para k = 1 : n
%Estimación de y_k + h/2 *y12 = y(k) + h/2* f(t(k), y(k))*
% Estimación de y_{k+1} *y(k+1) = y(k) + h* f(t(k) + h/2, y12)*
fin para *k*

Ejemplo 5

```
[t,y]=mieulerm(0,1,8,10)

t =
         0
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
    0.6000
    0.7000
```

```
    0.8000
    0.9000
    1.0000
y =
    8.0000
    6.9190
    5.8649
    4.8330
    3.8190
    2.8190
    1.8292
    0.8460
   -0.1342
   -1.1135
   -2.0889
```

17.6 Método de Runge-Kutta

El método elemental que más se utiliza para resolver numéricamente ecuaciones diferenciales es, sin duda, el de Runge-Kutta. Basado en la regla de Simpson de integración numérica, resulta como éste de orden 4.
Observando los términos de la fórmula de integración aproximada vemos que, en el primer paso, hay que estimar $f(t_{1/2}, y_{1/2})$ y $f(t_1, y_1)$, que dependen de valores de **y** aún no disponibles. Denotamos por $t_{1/2}$ a $t_0+h/2$.
Llamando

$$k_1 = f(t_0, y_0)$$

estimamos $y_{1/2}$ por Euler y evaluamos f en $(t_{1/2}, y_{1/2})$

$$k_2 = f(t_{1/2}, y_0 + h/2 \cdot k_1)$$

Estimamos ahora $y_{1/2}$ por Euler con pendiente k_2, en lugar de k_1 y volvemos a evaluar f

$$k_3 = f(t_{1/2}, y_0 + h/2 \cdot k_2)$$

Finalmente, estimamos y_1 por Euler con pendiente k_3 y evaluamos f en (t_1,y_1)

$$k_4 = f(t_1, y_0 + h \cdot k_3)$$

Combinando adecuadamente estas estimaciones de f en la regla de Simpson, obtenemos el primer paso del método de Runge-Kutta.

$$y_1 = y_0 + h/6 \cdot (k_1 + 2k_2 + 2k_3 + k_4)$$

En general, obtenemos y_{k+1} de y_k mediante las expresiones siguientes

$$\begin{aligned} k_1 &= f(t_k, y_k) \\ k_2 &= f(t_k+h/2, y_k + h/2 \cdot k_1) \\ k_3 &= f(t_k+h/2, y_k + h/2 \cdot k_2) \\ k_4 &= f(t_{k+1}, y_k + h \cdot k_3) \\ y_{k+1} &= y_k + h/6 \cdot (k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

Ejemplo 6

```
[t,y]=rk4('proyectil',0,1,8,10)

t =
         0
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
    0.6000
    0.7000
    0.8000
    0.9000
    1.0000
y =
    8.0000
    6.9187
    5.8643
    4.8322
    3.8180
    2.8179
    1.8279
    0.8445
   -0.1358
   -1.1150
   -2.0901
```

17.7 Problemas

17.7.1 Enunciados

1. Un proyectil de masa m = 0.11 Kg. se lanza verticalmente hacia arriba con una velocidad inicial de 8 m/s siendo frenado por la acción de la gravedad y por la resistencia del aire. Se satisface la ecuación diferencial

$$m \cdot v' = - m \cdot g - k \cdot v|v|$$

donde g = 9.8 m/s2 y k = 0.002 Kg/m. Halla la velocidad al cabo de 2 segundos usando el

a) Método de Euler con h = 20
b) Método de Euler con h = 10
c) Método de Euler con h = 5
d) Método de Heun con h = 20
e) Método de Heun con h = 10
f) Método de Heun con h = 5
g) Método de Euler modificado con n = 20
h) Método de Euler modificado con n = 10
i) Método de Euler modificado con n = 5
j) Método de Runge-Kutta con n = 20
k) Método de Runge-Kutta con n = 10
l) Método de Runge-Kutta con n = 5

Halla la solución analíticamente. Justificar el orden de error de cada método. Representa gráficamente la solución.
Tomando un intervalo de tiempo mayor, comprobar que la velocidad "crece" hasta llegar a cierto límite. Obtén éste analítica y numéricamente.

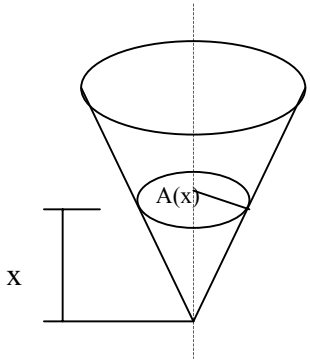
2. Un líquido de baja viscosidad fluye de un tanque cónico invertido por un orificio circular a razón de $\frac{dx}{dt} = -0.6\pi r^2\sqrt{2g}\frac{\sqrt{x}}{A(x)}$ donde r = 3 mm es el radio del orificio, x es la altura de la superficie del líquido desde el vértice del cono y A(x) es el área transversal del depósito a una altura x. Supongamos que la altura inicial del líquido es de 3 m y el volumen inicial de 180p/3 m3. Halla el nivel del agua al cabo de una hora, mediante

a) Método de Euler con h = 360
b) Método de Euler con h = 180
c) Método de Euler con h = 90
d) Método de Heun con h = 360
e) Método de Heun con h = 180
f) Método de Heun con h = 90
g) Método de Euler modificado con h = 360
h) Método de Euler modificado con h = 180
i) Método de Euler modificado con h = 90
j) Método de Runge-Kutta con h = 360
k) Método de Runge-Kutta con h = 180
l) Método de Runge-Kutta con h = 90

Halla la solución analíticamente. Justificar el orden de error de cada método. Representa gráficamente la solución.

17.7.2 Soluciones

2.-



A(x)= 3*Volumen/altura=pi*20*x^2/3

Supongamos: x0=3m.
V0=60*pi m^3
Nos piden calcular el nivel del agua, x, al cabo de una hora t=3600s

Entonces la ecuación diferencial quedara de la siguiente manera :

Dx/dt= -0.6πr^{2}sqrt(2g)*sqrt(x)/(π*20/3*x^2)

b.) Método de Euler con h = 180

El algoritmo del método de euler lo modificaremos porque en este ejercicio nos dan como dato el paso de integración luego tendremos que sacar n. Si h=180 n sera lo siguiente : n=(b-a)/h. Donde b=t=3600s y a=t0=0. Luego :

N=(3600-0)/180=20 ;

La nomenclatura :

y0=x0=3 m.
y=x

Antes de aplicar los algoritmos de cada método tendremos que crear un archivo de ombre f11.m en el que pondremos la ecuación diferencial. El algoritmo es el siguiente :

```
function z=f11(t,y)
%ejercicio 2 de propuestos.
r= 0.003;
g=9.8;
z=-0.6*pi*r^2*sqrt(2*g)*sqrt(y)./(pi*20/3*y.^2);
```

Ahora pasamos a aplicar los métodos.

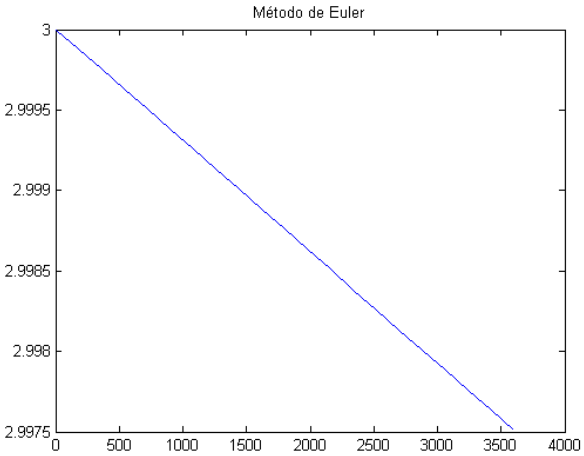
El algoritmo del método de euler es el siguiente :

```
function [t,y]=mieuler(a,b,y0,n)
%
%  [t,y]=mieuler(a,b,y0,n)
%
%  Obtenemos las aproximaciones dela solución en los nodos del intervalo,
%  por el llamado método de euler.
%  El siguiente algoritmo calcula estos valores y los almacena en un
%  vector y.
%
%       ¡¡ NOTA !!:    Es conveniente inicializar 'y' como un vector columna
%  de n+1 ceros:   y=zeros(n+1,1)
%  Así ahorraremos tiempo en las asignaciones de cada
%  iteración.
%
% >>>>>>>>>>  CONCLUSIONES:  duplicando el numero de subintervalos, el error
%  obtenido por la formula [max(abs(y-logistic(t,y0)))] se divide
%  aproximadamente por 2. El método de euler es de orden h.
%

h=(b-a)/n;       %paso de integración
t=(a:h:b)';       % Nodos de integración
y=zeros(n+1,1);     % Valor inicial
y(1,1)=y0;
for k=1:n
   y(k+1,1)=y(k,1)+h*f11(t(k),y(k,1));
end
```

Si lo ejecutamos con los valores dados en el enunciado :

```
[t,y]=mieuler(0,3600,3,20)
```

```
t =
           0
         180
         360
         540
         720
         900
        1080
        1260
        1440
        1620
        1800
        1980
        2160
        2340
        2520
        2700
        2880
        3060
        3240
        3420
        3600
y =
    3.0000
    2.9999
    2.9998
    2.9996
    2.9995
    2.9994
    2.9993
    2.9991
    2.9990
    2.9989
    2.9988
    2.9986
    2.9985
    2.9984
    2.9983
    2.9981
    2.9980
    2.9979
    2.9978
    2.9976
    2.9975
```



El nivel del agua al cabo de una hora será : x=2.9975m.

e.) Método de Heun con h = 180

Al igual que en el apartado anterior el paso de integración es un dato del problema entonces tendremos que hallar la n para poder introducirla como datos de entrada en la cabecera del algoritmo.

Entonces : n=(b-a)/h siendo b=t=3600s y a=t0=0s. Luego : n=3600/180=20.

y0=x0=3m.
y=x Es el resultado que buscamos.

El algoritmo del método de Heun es el siguiente :

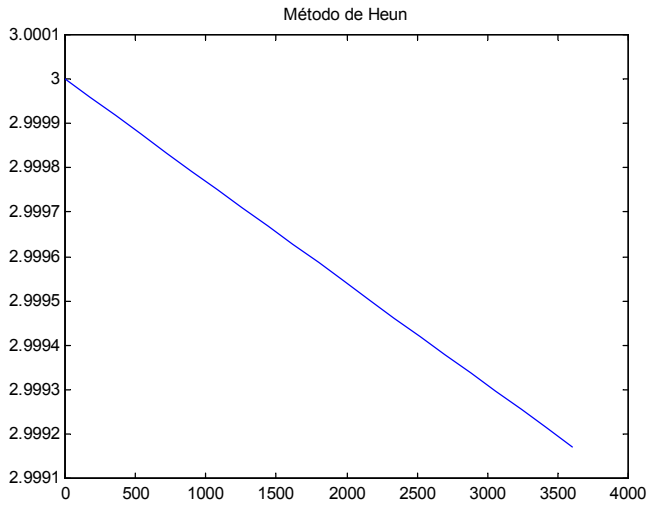
```
function [t,y]=Heun(a,b,y0,n)
%
%
% [t,y]=Heun(a,b,y0,n)
%
% Para mejorar la precision del metodo euler basta aproximar la integral en cada paso mediante un metodo mas preciso (como puede ser el de trapecios).
% Pero tenemos el inconveniente de que en el primer paso no conocemos f(t1,y1) dado que y1 es el resultado de este paso, entonces estimaremos el valor de y1 por euler.
% Entonces ya podremos utilizar trapecios y corregir el valor de y1.
% Obtenemos el llamado metodo Heun.


h=(b-a)/n;          % Paso de integración
t=(a:h:b)';         % Nodos de integración
```

```
y=zeros(n+1,1);     % Valor inicial
y(1,1)=y0;
for k=1:n
   k1=f11(t(k),y(k,1));
   yp=y(k,1)+h*k1;        % Predicción
   k2=f11(t(k)+h,yp);
   y(k+1,1)=y(k,1)+h/2*(k1+k2);  % Corrección
end
```

Introducimos los datos y ejecutamos el algoritmo y los resultados son :

```
[t,y]=Heun('deposito',0,3600,3,20)

t =
           0
         180
         360
         540
         720
         900
        1080
        1260
        1440
        1620
        1800
        1980
        2160
        2340
        2520
        2700
        2880
        3060
        3240
        3420
        3600
y =
   3.0000
   2.9998
   2.9997
   2.9996
   2.9995
   2.9993
   2.9992
   2.9991
   2.9990
   2.9988
   2.9987
   2.9986
   2.9985
   2.9983
   2.9982
   2.9981
   2.9980
   2.9978
   2.9977
   2.9976
```

```
2.9975
```



El nivel del agua al cabo de una hora serà : x=2.9975m.

El error se calcula utilizando la función sdeposito, en la cual tenemos que poner la solución exacta de la ecuación para luego poder hallar el error máximo poniendo
max(y-sdeposito(t,y0));

La solución exacta de la ecuanción diferencial :

La ecuación diferencial se puede resolver por partes ya que tiene la forma siguiente :

$Dx/dt=A*(sqrt(x)/x^{3/2})$ entonces la solución exacta es :

Utilizando la función logistic :

```
function y=sdeposito(t,y0)

r= 0.003;
g=9.8;
y=(y0^(5/2)-t*0.6*pi*r^2*sqrt(2*g)./(pi*20/3)*5/2).^(2/5);
```

Con esta función obtenemos que el herror para el método de Euler es $7.73\ 10^{-8}$ y para el método de Heun $2.68\ 10^{-13}$.

CAMPO DE DIRECCIONES

```
>> campo('verhulst',0,3,0,6)
>> hold on
```

METODO NUMERICO: EULER

```
>> for k=0:0.1:6
[t,y] = mieuler('verhulst',0:0.01:3,k);
plot(t,y,'k')
end
```

CALCULO DEL ERROR

GRAFICAMENTE

```
>> [t,y] = mieuler('verhulst',0:0.01:3,0.2);
>> plot(t,y,'g')
>> [t,y] = mieuler('verhulst',0:0.005:3,0.2);
>> plot(t,y,'r')
>> [t,y] = mieuler('verhulst',0:0.0025:3,0.2);
>> plot(t,y,'b')
```

CALCULO

```
>> [t,y] = mieuler('verhulst',0:0.01:1,0.2);
y1 = y(end)
>> [t,y] = mieuler('verhulst',0:0.005:1,0.2);
y2 = y(end)
>> [t,y] = mieuler('verhulst',0:0.0025:1,0.2);
y3 = y(end)

>> y2-y1
ans =
   0.0110

>> y3-y2
ans =
```

0.0054

El doble de orden reduce el error a la mitad. Lineal.
Es muy malo

METODO NUMERICO: HEUN

CONVERGENCIA

```
>> [t,y] = heun('verhulst',0:0.01:1,0.2);
>> y1 = y(end)
y1 =
   2.9670

>> [t,y] = heun('verhulst',0:0.005:1,0.2);
>> y2= y(end)
y2 =
   2.9673

>> [t,y] = heun('verhulst',0:0.0025:1,0.2);
>> y3 = y(end)
y3 =
   2.9673

>> (y2 - y1)/(y3-y2)
ans =
   3.9609 % aprox 4.
```

Aumentar al doble de puntos mejora en 4 la aproximacion
es cuadrático

METODO NUMERICO: EULER MODIFICADO

CONVERGENCIA

```
>> [t,y] = eulermod('verhulst',0:0.01:1,0.2);
>> y1 = y(end)

>> [t,y] = eulermod('verhulst',0:0.005:1,0.2);
>> y2= y(end)
```

```
>> [t,y] = eulermod('verhulst',0:0.0025:1,0.2);
>> y3 = y(end)

>> (y2 - y1)/(y3-y2)
ans =
   3.9475
```

Convergencia cuadratica

Justificacion de la importancia de utilizar método preciso

Utilizando un metodo malillo:

```
>> [t,y] = mieuler('verhulst',0:0.3:3,0.2);
>> plot(t,y,'k')

>> [t,y] = mieuler('verhulst',0:0.7:3,0.2);
>> plot(t,y,'r')

>> [t,y] = mieuler('verhulst',0:0.6:40,0.2);
plot(t,y,'r')
```

se desmadra totalmente

18. Sistemas de ecuaciones diferenciales

En el tema anterior hemos desarrollado métodos numéricos para resolver ecuaciones diferenciales ordinarias de primer orden, que aparecen en las aplicaciones cuando se conoce cómo varía una magnitud y con respecto a otra, generalmente el tiempo t. En este tema, generalizaremos dichos métodos a situaciones que involucren la variación con respecto a t de varias magnitudes, o bien, aquellas en la que intervienen derivadas de y de orden mayor que 1. En el primer caso se trata de los sistemas de ecuaciones diferenciales, y en el segundo de las ecuaciones diferenciales de orden superior.

Un ejemplo típico de sistema diferencial surge al estudiar la evolución de dos poblaciones biológicas relacionadas entre sí. Conocidas leyes físicas, como la ley de Newton, $F = m \cdot a$, dan lugar a ecuaciones diferenciales de orden superior, en este caso de orden 2, pues la aceleración es la derivada segunda de la posición.

La generalización buscada es directa y sencilla, consistiendo básicamente en trabajar con vectores en lugar de escalares. Los programas se modifican mínimamente gracias a la capacidad vectorial de MATLAB. Por la misma razón, en lugar de desarrollar métodos específicos para ecuaciones de orden superior, se prefiere transformarlas en sistemas de primer orden, con tantas ecuaciones como el orden de la ecuación dada.

Aplicaremos los métodos desarrollados a la resolución de circuitos eléctricos LRC, que dan lugar a ecuaciones diferenciales lineales de segundo orden con coeficientes constantes. Como estas ecuaciones se resuelven analíticamente, en cualquier momento podemos comprobar la precisión de los resultados numéricos. Naturalmente, éstos son realmente útiles cuando no se dispone de una expresión analítica adecuada de la solución.

18.1. Ecuaciones de orden superior y sistemas

Una *sistema de ecuaciones diferenciales* expresa las relaciones que verifican varias funciones desconocidas, $y_1(t)$, $y_2(t)$, ..., $y_m(t)$ y sus derivadas respecto a la variable t. El objetivo es determinar estas funciones en un intervalo [a, b]. En el caso más sencillo, la derivada primera de cada función puede expresarse explícitamente en términos de la variable independiente y de las funciones incógnita.

$$\left.\begin{aligned} y_1'(t) &= f_1(t, y_1(t), y_2(t), ..., y_m(t)) \\ y_2'(t) &= f_2(t, y_1(t), y_2(t), ..., y_m(t)) \\ &\vdots \\ y_m'(t) &= f_m(t, y_1(t), y_2(t), ..., y_m(t)) \end{aligned}\right\}$$

Las *m* ecuaciones admiten una escritura compacta en términos vectoriales. Considerando las incógnitas como componentes de una función vectorial de una variable

$$\boldsymbol{y}(t) = \big(y_1(t), y_2(t), \ldots, y_m(t)\big)\colon \mathbb{R} \;\rightarrow\; \mathbb{R}^m$$

y las ecuaciones, como componentes de una función vectorial de varias variables

$$\boldsymbol{f}(t, \boldsymbol{y}(t)) = \big(f_1(t, \boldsymbol{y}(t)), f_2(t, \boldsymbol{y}(t)), \ldots, f_m(t, \boldsymbol{y}(t))\big)\colon \mathbb{R}^{m+1} \;\rightarrow\; \mathbb{R}^m$$

el sistema diferencial se escribe formalmente igual que una ecuación diferencial

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t))$$

El Problema de Valor Inicial consiste en resolver este sistema dado el valor en el instante inicial *a,*

$$\boldsymbol{y}(a) = \big(y_1(a), y_2(a), \ldots, y_m(a)\big) = \boldsymbol{y_a}$$

La notación vectorial sugiere la extensión de los métodos numéricos de ecuaciones diferenciales ordinarias a los sistemas de ecuaciones diferenciales, efectuando componente a componente las operaciones de estos métodos.

Ejemplo 1

Dos especies viven en determinado medio y una de ellas se alimenta de la otra, por ejemplo, zorros y conejos. Si no fuera por los zorros, los conejos se reproducirían proporcionalmente a su población, pues encuentran en el medio suficientes recursos para alimentarse. Los conejos son devorados por los zorros a velocidad proporcional al producto del número de zorros, $z(t)$, por el número de conejos, $c(t)$.

La proliferación de los zorros depende de la cantidad de conejos que pueden cazar, que suponemos proporcional al número de encuentros entre zorros y conejos, que, a su vez, es proporcional al producto de sus poblaciones respectivas. Asimismo, si hay demasiados zorros, han de competir entre ellos por el alimento. El factor de competencia se considera proporcional al cuadrado de la población de zorros.

Estas ideas se resumen en el sistema diferencial

$$\left.\begin{aligned} c' &= \alpha c - \beta c z \\ z' &= -\gamma z^2 + \delta c z \end{aligned}\right\}$$

donde α, β, γ y δ son constantes apropiadas.

Las *ecuaciones diferenciales de orden m* involucran la función incógnita y(t) y sus derivadas hasta el orden *m*. Supondremos que la ecuación está en forma explícita, es decir, con la derivada de mayor orden expresada en función de las demás y en su caso de la variable independiente.

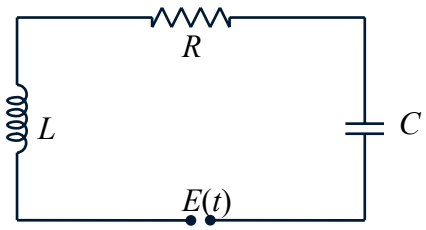
$$y^{(m)}(t) = f\big(t, y(t), y'(t), y''(t), \ldots y^{(m-1)}\big)$$

Las ecuaciones diferenciales de orden superior al primero se transforman en sistemas mediante un cambio de variables. Llamando y_1 a la función incógnita *y,* y_2 a su derivada *y',* y_3 a la derivada segunda *y'',* y así sucesivamente, queda el sistema equivalente

$$\left.\begin{aligned} y_1'(t) &= y_2(t) \\ y_2'(t) &= y_3(t) \\ y_3'(t) &= y_4(t) \\ &\vdots \\ y_m'(t) &= f(t, y_1(t), y_2(t), ..., y_m(t)) \end{aligned}\right\}$$

Ejemplo 2

Un circuito eléctrico LRC consta de una resistencia, una bobina y un condensador conectados en serie con una fuente de energía eléctrica. Si denotamos por R el valor de la resistencia, por L la inductancia de la bobina y por C la capacidad del condensador y suponemos que la tensión en la fuente viene dada por una función del tiempo $E(t)$, entonces, el comportamiento del circuito viene dado por la ecuación



$$LI' + RI + \frac{Q}{C} = E(t)$$

donde Q es la carga en el condensador e $I = Q'$ es la intensidad de la corriente por el circuito. Al fijar los valores de Q e I en $t = 0$ se obtiene un problema de valor inicial.

Si expresamos I e I' en función de Q, queda una ecuación diferencial de segundo orden

$$LQ'' + RQ' + \frac{Q}{C} = E(t)$$

Tomando en cambio como variables Q e I, obtenemos el sistema equivalente de primer orden

$$\left.\begin{aligned} Q' &= I \\ I' &= \left(E(t) - \frac{Q}{C} - RI \right) \Big/ L \end{aligned}\right\}$$

18.2. Método de Euler

Consideramos el problema de valor inicial asociado a un sistema de m ecuaciones diferenciales y a unos valores iniciales dados

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)), \quad t \in [a, b]; \quad \boldsymbol{y}(a) = \boldsymbol{y}_a$$

Para cada t, $\boldsymbol{y}(t)$ e $\boldsymbol{y}'(t)$ son vectores m-dimensionales. Salvo este pequeño detalle, el método de Euler se aplica igual que en el caso unidimensional. Dividimos el intervalo $[a, b]$ en n partes iguales

$$h = (b - a)/n$$
$$t_k = a + (k - 1) * h$$

y estimamos iterativamente el valor de la solución en cada nodo según

$$\boldsymbol{y}(t_1) = \boldsymbol{y}_1 = \boldsymbol{y}_a$$

Para $k = 1, 2, \ldots, m$

$$\boldsymbol{y}(t_{k+1}) \cong \boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h \cdot \boldsymbol{f}(t_k, \boldsymbol{y}_k)$$

Vamos a programar el método de Euler para sistemas diferenciales, mediante una función cuyas entradas y salidas sean análogas a las de los métodos implementados en MATLAB. Consultando la ayuda de ODE23, vemos que la sintaxis más sencilla es

```
[t,y] = ode23(odefun,tspan,y0)
```

donde `odefun` es el fichero de la ecuación diferencial, `tspan` es el intervalo de integración e `y0` la condición inicial. La salida proporciona una matriz `y` cuyas columnas son las componentes de la solución en los instantes dados por `t`. Cada fila de `y` contiene el valor de la solución en el instante correspondiente de `t`.

El vector `tspan` puede ser de la forma `[a,b]`, siendo `a` y `b` los extremos del intervalo en que se quiere calcular la solución, o bien un vector que contenga todos los instantes en los que se requiere conocerla. En el primer caso, el algoritmo determina los puntos a evaluar, en función de la precisión requerida. Nuestros algoritmos usan la versión extensa de `tspan`, ya que son de paso fijo.

La llamada a la función `odefun`

```
dy = odefun(t,y)
```

tiene dos argumentos de entrada, aunque no utilice el primero, y debe producir una columna `dy` con las derivadas dadas por el sistema diferencial, $\boldsymbol{f}(t, \boldsymbol{y})$.

Damos a continuación un posible código del método de Euler, de acuerdo con estas observaciones. En cada paso, al algoritmo añade una fila a la matriz y con el valor calculado de la solución para el instante siguiente.

```
function [t,y]=mieuler(f,t,y0)

% Método de Euler para el problema de valor inicial
%
%    y'(t) = f(t,y(t));   y(t0)=y0;
%
% en los instantes dados por t

n = lenght(t);
h = diff(t);
y(1,:) = y0;
for k=1:n
   y(k+1,:) = y(k,:) + h(k)*feval(f,t(k),y(k,:))';
end
```

Curiosamente, MATLAB requiere que al evaluar un sistema diferencial, las derivadas se obtengan en columna, mientras que genera la solución por filas. Por este motivo aparece la transpuesta en el código de `mieuler`.

Ejemplo 3

El fichero siguiente evalúa el sistema diferencial del Ejemplo 1 con determinados valores de los parámetros.

$$\left.\begin{aligned} c' &= \alpha c - \beta c z \\ z' &= -\gamma z^2 + \delta c z \end{aligned}\right\}$$

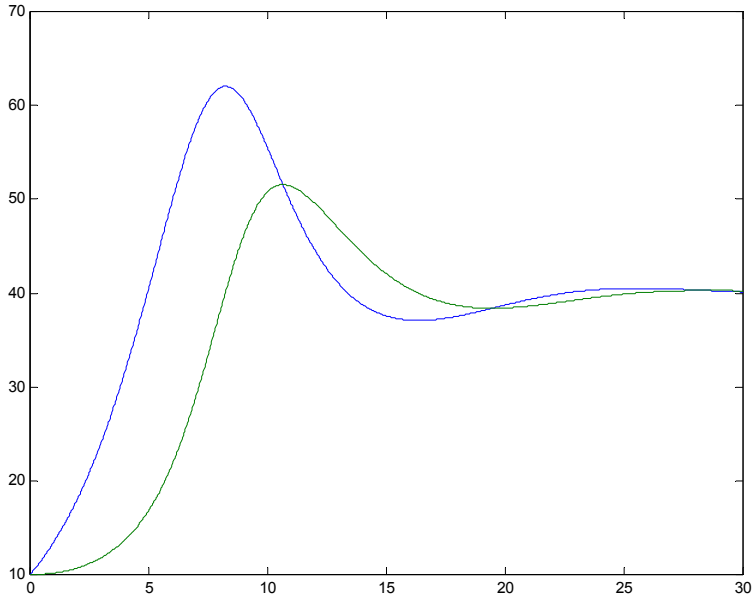
```
function dy = presarapaz(t,y)

% Modelo presa-rapaz (zorros, conejos)

% Parametros
a=0.4; b=0.01; g=0.01; d=0.01;

% Variables
c = y(1);
z = y(2);

% Ecuacion
dc = a*c-b*c*z;
dz = g*c*z-d*z^2;

% Salida
dy = [dc; dz];
```

Resolvamos el problema de valor inicial con partiendo de $c = z = 10$, en un tiempo de 30 unidades, por el método de Euler con 1000 subintervalos.

```
tspan = 0:0.03:30;
[t,y] = mieuler('presarap',tspan,[10,10]);
```

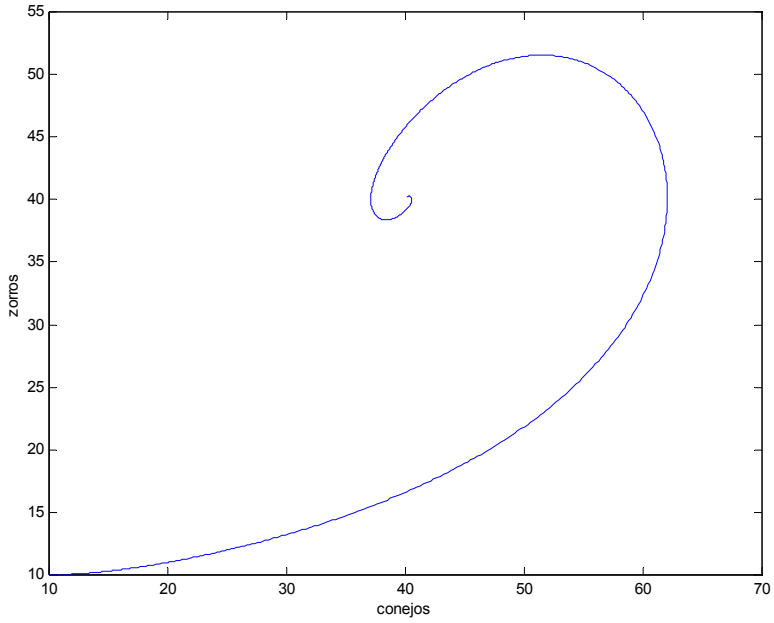
El gráfico muestra la evolución de las poblaciones con el tiempo, observándose que tienden a una situación estable con 40 individuos de cada especie.

```
plot(t,y)
```



En este otro gráfico se observa la interrelación entre ambas poblaciones.

```
plot(y(:,1),y(:,2))
xlabel('conejos')
ylabel('zorros')
```



Cada punto de la trayectoria muestra las poblaciones de ambas especies en un momento dado. La trayectoria se recorre en sentido antihorario. Se observa que cuando la población de conejos es grande, la de zorros aumenta, lo que a su vez hace disminuir

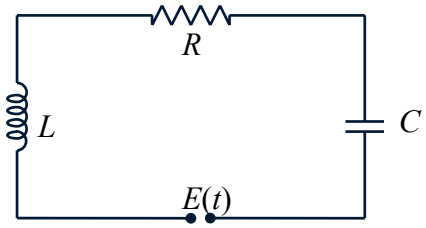
el número de conejos. el aumento de zorros provoca mayor competencia entre ellos además de escasez de conejos, con lo que la población de zorros se reduce.

En este caso los valores de los parámetros hacen que el sistema tienda al equilibrio rápidamente, pero con otros valores, el comportamiento puede ser distinto, provocando, por ejemplo, la extinción de una especie.

Ejemplo 4

Consideremos circuito eléctrico LRC en el que no hay tensión impresa y la resistencia es cero. La solución general de la ecuación diferencial del circuito

$$LQ'' + \frac{Q}{C} = 0$$



es

$$Q = A\cos\omega_0 t + B\sin\omega_0 t$$

donde A y B dependen de las condiciones iniciales. En ausencia de resistencia, el circuito no pierde energía y presenta oscilaciones periódicas sinusoidales de frecuencia angular $\omega_0 = 1/\sqrt{LC}$.

El fichero siguiente codifica el sistema diferencial

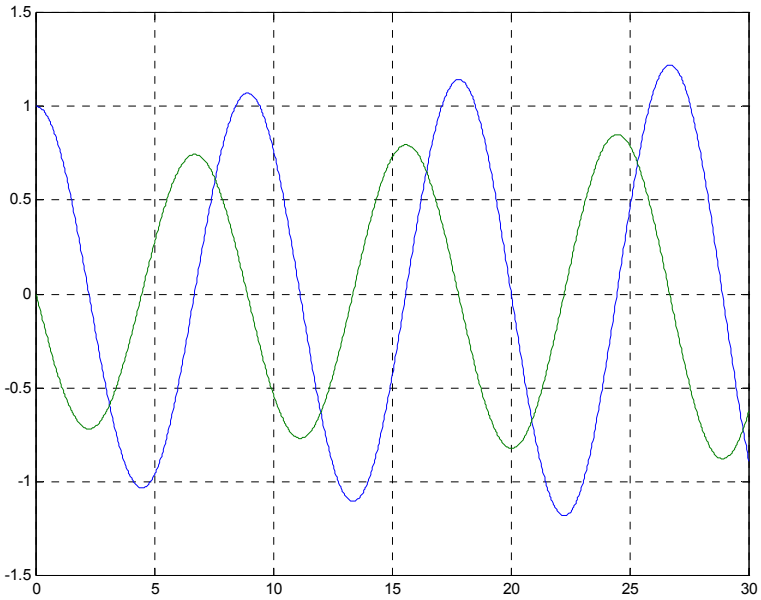
$$\left.\begin{aligned} Q' &= I \\ I' &= \left(E(t) - \frac{Q}{C} - RI\right)\Big/L \end{aligned}\right\}$$

de un circuito genérico LRC. En el propio fichero se establecen los parámetros del circuito concreto. Damos valores convencionales a los datos para simplificar los cálculos; los valores reales suelen estar en diferente escala.

```
function dy=lrc(t,y)

% Circuito LRC
% LI' + RI + Q/C = E(t)

% Prametros del circuito
L = 2;
R = 0;
C = 1;
E = 0;

% Variables
Q = y(1);
I = y(2);

% Ecuación
dQ = I;
dI = (E-Q/C-R*I)/L;

% Salida
dy = [dQ;dI];
```

Resolvamos el circuito por el método de Euler en un intervalo de 30 segundos con 1000 subintervalos

```
tspan = 0:0.03:30;
[t,y] = mieuler('lrc',tspan,[1,0]);
plot(t,y)
grid
```



El gráfico tiene algo inesperado, pues la amplitud de las oscilaciones va en aumento, en lugar de mantenerse constante. Esto no tiene sentido físico, porque supone que el circuito genera energía. Concluimos que es debido a la limitada precisión del método de Euler. Podemos mejorar la precisión aumentando el número de pasos de integración, pero así aumenta también el coste computacional. La alternativa es recurrir a un método de mayor orden, como puede ser el método de Heun o el de Runge-Kutta.

18.3. Método de Runge-Kutta

Recordemos las fórmulas de Runge-Kutta del caso escalar y las adaptamos al problema de valor inicial

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)), \quad t \in [a, b]; \quad \boldsymbol{y}(a) = \boldsymbol{y}_a$$

Dividimos el intervalo [a, b] en n partes iguales

$$h = (b-a)/n$$
$$t_k = a + (k-1) * h$$

y estimamos iterativamente el valor de la solución en cada nodo según

$$\boldsymbol{y}(t_1) = \boldsymbol{y}_1 = \boldsymbol{y}_a$$

Para $k = 1, 2, \ldots, m$

$$\boldsymbol{k}_1 = \boldsymbol{f}(t_k, \boldsymbol{y}_k)$$
$$\boldsymbol{k}_2 = \boldsymbol{f}(t_k + h/2, \boldsymbol{y}_k + h/2 \cdot \boldsymbol{k}_1)$$
$$\boldsymbol{k}_3 = \boldsymbol{f}(t_k + h/2, \boldsymbol{y}_k + h/2 \cdot \boldsymbol{k}_2)$$
$$\boldsymbol{k}_4 = \boldsymbol{f}(t_k + h, \boldsymbol{y}_k + h \cdot \boldsymbol{k}_3)$$

$$\boldsymbol{y}(t_{k+1}) \cong \boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h/6(\boldsymbol{k}_1 + 2 * \boldsymbol{k}_2 + 2 * \boldsymbol{k}_3 + \boldsymbol{k}_4)$$

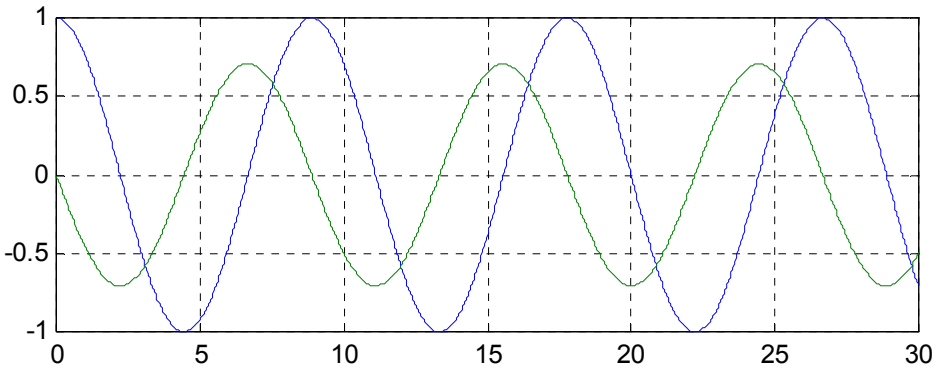
Codificamos el método de Runge-Kutta para sistemas, teniendo en cuenta las observaciones anteriores sobre la sintaxis de estas funciones.

```
function [t,y]=rungekut(f,t,y0)

% Método de Runge-Kutta para el problema de valor
inicial
%
%    y'(t) = f(t,y(t));   y(t0)=y0;
%
% en los instantes dados por t

h = diff(t);
n = length(t);
y(1,:) = y0;
for k=1:n-1
    k1 = feval(f,t(k), y(k,:))';
    k2 = feval(f,t(k)+h(k)/2, y(k,:)+h(k)/2*k1)';
    k3 = feval(f,t(k)+h(k)/2, y(k,:)+h(k)/2*k2)';
    k4 = feval(f,t(k+1),y(k,:)+h(k)*k3)';
    y(k+1,:) = y(k,:)+h(k)/6*(k1+2*k2+2*k3+k4);
end
```

Ejemplo 5

Aplicando Runge-Kutta al problema de valor inicial del ejemplo anterior con el mismo número de subintervalos, obtenemos un resultado más coherente con lo esperado desde el punto de vista físico.

```
tspan = 0:0.03:30;
[t,y] = rungekut('lrc',tspan,[1,0]);
plot(t,y)
grid
```



18.4. Paso de parámetros mediante variables globales

Para analizar el modelo del circuito LRC con distintos parámetros no resulta muy cómodo tener que editar el fichero de la función lrc.m cada vez que se desea modificar uno de ellos. Este inconveniente se puede evitar definiendo como variables globales aquellos parámetros que deseamos modificar.

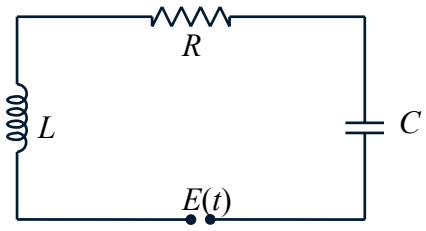
La sintaxis del fichero del sistema de ecuaciones diferenciales quedaría como sigue.

```
function dy=lrcg(t,y)

% Circuito LRC
% LI' + RI + Q/C = E(t)

% Prametros del circuito
global L R C
E = 0;

% Variables
Q = y(1);
I = y(2);

% Ecuación
dQ = I;
dI = (E-Q/C-R*I)/L;

% Salida
dy = [dQ;dI];
```

Las variables globales se asignarían desde el espacio de trabajo o desde otro fichero. En ambos casos, previamente hay que declarar globales dichas variables.

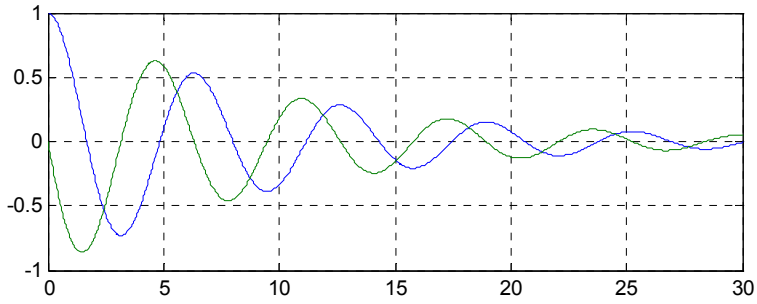
Ejemplo 6

En un circuito eléctrico real, la resistencia siempre es positiva y parte de la energía eléctrica se transforma en calor. En ausencia de fuentes de potencial, la intensidad y la carga en el circuito se anulan con el tiempo. Dependiendo del valor de la resistencia, se producen oscilaciones o no, hasta que se anula la corriente. Por ejemplo,



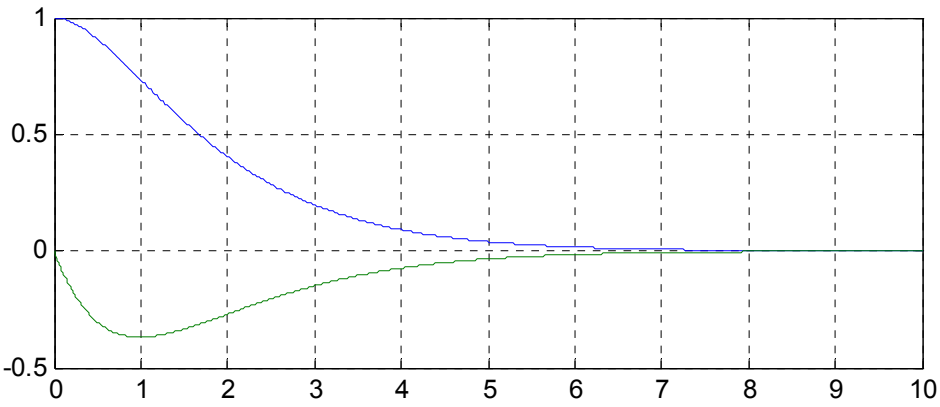
tomando $R = 0.2$, se producen oscilaciones de amplitud decreciente (subamortiguadas).

```
global L R C
L=1;C=1;R=0.2;
tspan = 0:0.03:30;
[t,y] = rungekut('lrcg',tspan,[1,0]);
plot(t,y)
grid
```



En cambio, con R = 2, el circuito está sobreamortiguado y no se producen oscilaciones en el tránsito a la estabilidad.

```
R=2;
tspan = 0:0.01:10;
[t,y] = rungekut('lrcg',tspan,[1,0]);
plot(t,y)
grid
```



18.5. Uso de las funciones de MATLAB para sistemas diferenciales

MATLAB dispone de diversas funciones para resolver ecuaciones diferenciales, aplicando en cada caso métodos apropiados. Los métodos elementales corresponden a las funciones ODE23 y ODE45. Consultando la ayuda de cualquiera de ellas, se obtiene además la lista de métodos disponibles.

La diferencia fundamental entre los métodos elementales que hemos programado, Euler, Heun, Runge-Kutta y los de MATLAB, es que en estos últimos, el paso de integración se determina dinámicamente, en función del comportamiento de la ecuación

y con el objetivo de que el error estimado esté dentro de cierta tolerancia, mientras que en los métodos que hemos programado nosotros, el paso es fijo.

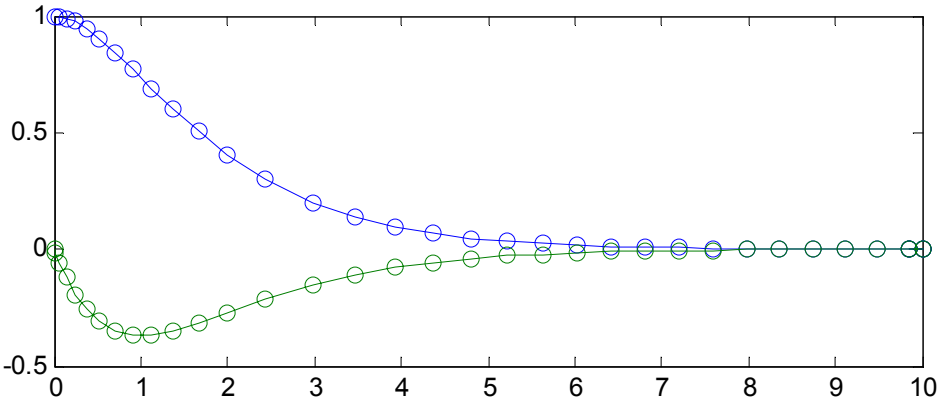
Como ya hemos indicado, la llamada básica a las funciones de MATLAB es del tipo

```
[t,y] = ode23(odefun,tspan,y0)
```

donde `tspan` puede contener, como antes, todos los nodos, o bien, sólo los extremos del intervalo.

En la ayuda se describen numerosas variantes y opciones de estos comandos. Por ejemplo, omitiendo los argumentos de salida, se obtiene la gráfica de la solución.

```
ode23('lrcg',[0,10],[1,0]);
```



Como se aprecia en la figura, los nodos no están equiespaciados. Si queremos forzar que la solución se evalúe en puntos determinados, hemos de introducirlos en `tspan`, tal como la hacemos con nuestros algoritmos.

18.6. Problemas

18.6.1. Enunciados

1.- Emplea el método de Runge-Kutta aplicado a sistemas para aproximar las soluciones a los siguientes problemas de valor inicial. Compara, numérica y gráficamente, el resultado obtenido en cada caso con la solución exacta.

a)

$$\begin{aligned} x' &= 3x + 2y - \left(2t^2+1\right)e^{2t} \qquad x(0)=1 \quad t\in[0,1] \\ y' &= 4x + y + \left(t^2+2t-4\right)e^{2t} \quad y(0)=1 \end{aligned}$$

Solución exacta: $x(t) = \frac{e^{5t}}{3} - \frac{e^{-t}}{3} + e^{2t} \qquad y(t) = \frac{e^{5t}}{3} + \frac{2e^{-t}}{3} + t^2 e^{2t}$

b)

$$\begin{aligned} x' &= y & x(0)&=1 \quad t\in[0,2] \\ y' &= -x - 2e^t + 1 & y(0)&=0 \\ z' &= -x - e^t + 1 & z(0)&=1 \end{aligned}$$

Solución exacta:

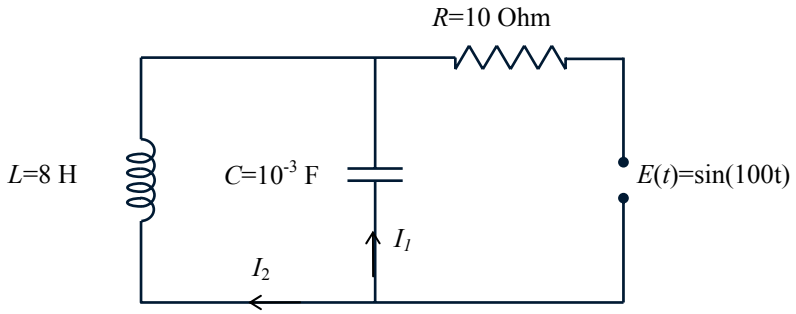
$$x(t) = \cos t + \operatorname{sen} t - e^t + 1 \qquad y(t) = -\operatorname{sen} t + \cos t - e^t \qquad z(t) = -\operatorname{sen} t + \cos t$$

2.- Una masa *m* suspendida de un alambre inextensible de longitud l y masa despreciable oscila en un plano vertical, sometido a la acción de la gravedad y a la resistencia del aire. Denotamos por *y* la desviación en radianes del péndulo respecto a la vertical y suponemos que la resistencia del aire es proporcional a la velocidad del péndulo. La ecuación que rige su movimiento será

$$y'' = -\frac{g}{l} - \operatorname{sen} y(t) - \frac{k}{m} y'$$

a) Resuelve el movimiento del péndulo no amortiguado (*k*=0) con distintos métodos y diferente número de puntos.
b) Estudia el comportamiento del péndulo amortiguado para distintos valores de *k*.
c) Considera ahora que el péndulo que recibe un impulso exterior *h*(*t*)=cos(*t*). Estudia su comportamiento para distintos valores iniciales de la velocidad.

3.- Estudia el circuito



Comprueba numéricamente que el método de Runge-Kutta es de orden 4 comparando los resultados en un punto con diferentes pasos.
4.- Un péndulo de Foucault proporciona la demostración empírica de la rotación terrestre: el plano de oscilación del péndulo gira con el tiempo de manera que tras un cierto periodo que depende de la latitud del observador, vuelve a su posición original. Las ecuaciones de movimiento son:

$$x'' - 2\omega \operatorname{sen}(\varphi) y' + k^2 x = 0$$
$$y'' + 2\omega \operatorname{sen}(\varphi) x' + k^2 y = 0$$

donde $\omega = 7.29 \cdot 10^{-5}\ \ s^{-1}$ es la velocidad angular de la rotación terrestre, φ es la latitud del observador y la constante $k^2 = \frac{g}{l}$ depende de la aceleración de la gravedad *g* y la longitud del péndulo *l*. Para iniciar el movimiento se sitúa el péndulo en la posición *x* = –6, *y* = 0 y se deja en movimiento libre. Calcula la posición del péndulo en el instante t = 3600 con un paso temporal de 0.5 segundos. ¿Cuánto tiempo es necesario para que un péndulo de 10 m de longitud gire su plano de oscilación 45º en Valencia, con una latitud de 40º? ¿Cuánto tiempo tarda el plano de oscilación en girar 360º en el polo norte? ¿Y en el ecuador?

-Las ecuaciones del movimiento de un satélite puesto en órbita desde la Estación Espacial Internacional son

$$\left.\begin{aligned} x'' &= -\mu\frac{x}{r^3}, \quad x(0)=x_0, \quad x'(0)=v_{x0} \\ y'' &= -\mu\frac{y}{r^3}, \quad y(0)=0, \quad y'(0)=v_{y0} \end{aligned}\right\}$$

donde $r=\sqrt{x^2+y^2}$ es la distancia a la Tierra, situada en el origen de coordenadas y $\mu = G\cdot(M_T+m_S)\cong 398598.309\ km^3s^{-2}$ es el llamado parámetro gravitacional, en el que se considera nula la masa de satélite respecto a la de la Tierra. Consideramos que la estación espacial está situada en el punto x_0 del eje de abscisas y v_{x0}, v_{y0} son las componentes de la velocidad inicial.

a) Convierte el sistema de orden 2 en uno de primer orden.
b) Considerando las condiciones iniciales:

$$x(0)=42167.911\ km \quad x'(0)=-1.07168\ km\ s^{-1}$$
$$y(0)=0\ km \quad y'(0)=2.8827 km\ s^{-1}$$

Resuelve el problema de valor inicial empleando la función de MATLAB ode23, encontrando la posición del satélite respecto a la Tierra transcurridas 24 horas desde su lanzamiento.

c) Dado que las variables x e y dependen del tiempo, podemos considerar la trayectoria del satélite en el espacio situada en un plano que contiene al ecuador terrestre. Representa dicha trayectoria situando en el origen de coordenadas una esfera de radio $R_T = 6378.1\,\text{km}$.

7.- Utiliza el algoritmo de Runge-Kutta para aproximar la solución del problema de valor inicial:

$$y''' = -6y^4 \qquad t\in[1,1.9]$$
$$y(1)=-1 \qquad y'(1)=-1 \qquad y''(1)=-2$$

Emplea el paso $h = 0.05$ y compara los resultados, numérica y gráficamente, con la solución exacta $y(t)=\dfrac{1}{t-2}$.

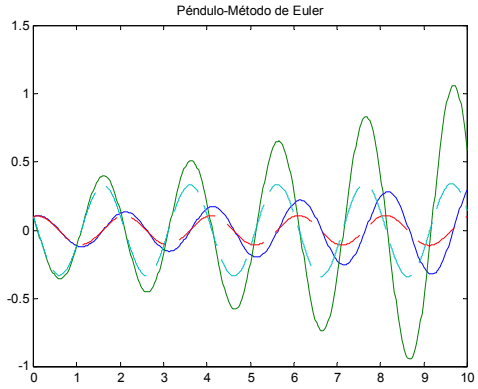
18.6.2. Soluciones

2.- a) Primero definimos la función

```
function dy=pendulo(t,y)

 k=0;          %Resistencia del medio
 m=1;          %Masa
 g=9.81;       %Aceleración de la gravedad
 l=1;          %Longitud del péndulo

 dy=[y(2);-g/l*sin(y(1))-k/m*y(2)];
```

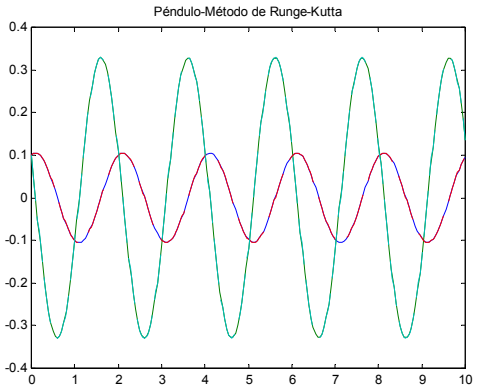
Luego calculamos la solución en el intervalo [0,10] con el método de Euler con dos pasos diferentes:

```
t=linspace(0,10,401);
[t,y]=mieuler('pendulo',t,[0.1,0]);
t2=linspace(0,10,10001);
[t2,y2]=mieuler('pendulo',t2,[0.1,0]);
 plot(t,y,'-',t2,y2,'--')
title('Péndulo-Método de Euler')
```



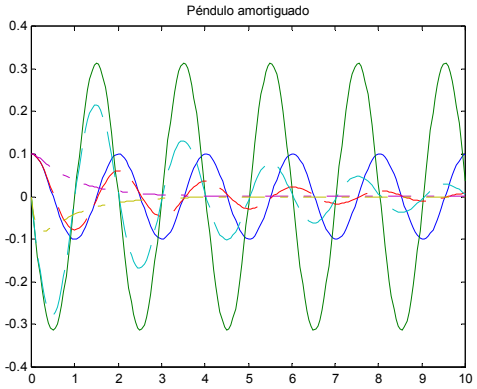
Como pasaba con el circuito eléctrico, si se toman pocos puntos la energía aumenta. Con el método de Runge-Kutta no sucede esto

```
t=linspace(0,10,201);
[t,y]=rungekut('pendulo',t,[0.1,0]);
t2=linspace(0,10,401);
[t2,y2]=rungekut('pendulo',t2,[0.1,0]);
 plot(t,y,'-',t2,y2,'--')
```



b) Vamos a tomar por ejemplo k=0,0.5 y 10. Si utilizamos el método de Runge Kutta dividiendo el intervalo en 200 subintervalos se obtiene

```
t=linspace(0,10,201);
k=0;[t,y]=rungekut('pendulo',t,[0.1,0.]);
k=0.5;[t,y2]=rungekut('pendulo',t,[0.1,0.]);
k=10;[t,y3]=rungekut('pendulo',t,[0.1,0.]);
plot(t,y,'-',t,y2,'--',t,y3,'-.')        ,title('Péndulo
amortiguado')
```



Para valores pequeños de k el péndulo va oscilando hasta que se detiene. Para valores grandes no se producen oscilaciones.

c) Añadimos al fichero de la ecuación del péndulo el término del forzamiento externo.

```
function dy=pendulo(t,y)

 k=0.;       %Resistencia del medio
 m=1;         %Masa
 g=9.81;      %Aceleración de la gravedad
 l=10;         %Longitud del péndulo

 dy=[y(2);-g/l*sin(y(1))-k/m*y(2)+cos(t)];
```

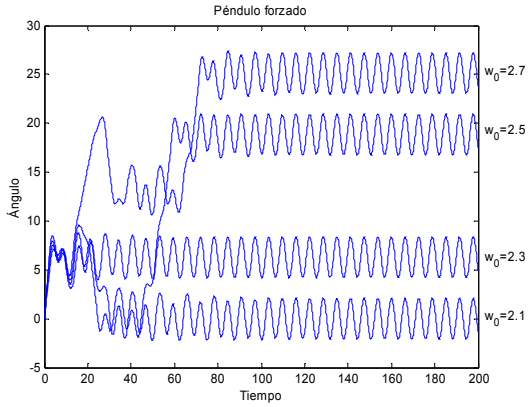
Por analogía con el caso del oscilador lineal, se puede suponer que hay una parte transitoria que decrece exponencialmente y al final de la oscilación sigue la de cos(t). Sin embargo el comportamiento del sistema hasta alcanzar el estado estacionario es difícil de prever. Vamos a resolver el sistema partiendo del reposo y con una velocidad inicial 2.7.

```
t=linspace(0,200,2001);[t,y]=rungekut('pendulof',t,[0,2.7]);
plot(t,y(:,1)),hold
```

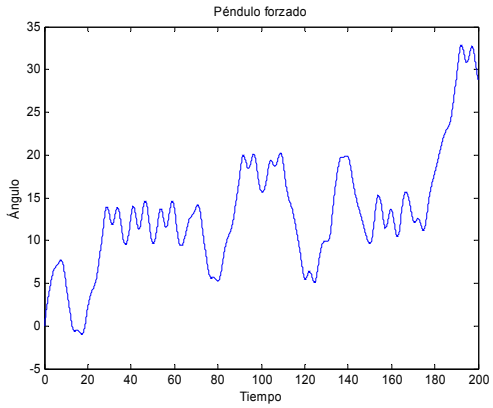
La gráfica indica que el péndulo da unas cuantas vueltas completas antes de estabilizarse. Esto podría ser debido a que se le ha dado demasiado impulso. Podemos probar con una velocidad inicial 2.5.

```
[t,y]=rungekut('pendulof',t,[0,2.5]);
```

Ahora el péndulo se estabiliza antes. Parece que cada vez el comportamiento es menos caótico. Se puede probar con menos velocidad, por ejemplo 2.3 y 2.1.



En el caso de 2.3 hay menos oscilaciones, pero en para 2.1 parece que vuelva a empezar a oscilar. Podemos probar con 1.9



Ahora el comportamiento vuelve a ser bastante caótico. Se observa que es difícil hacer predicciones para un tiempo futuro.

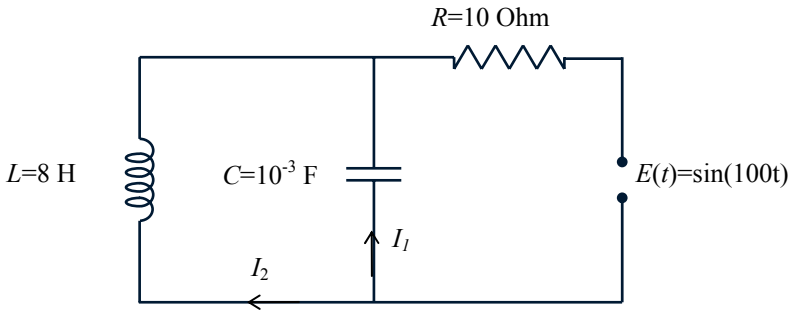
3.- Igualando la diferencia de potencial en el condensador y en la bobina obtenemos la ecuación

$$\frac{Q_1}{C} = LI_2'$$

donde Q_1 es la carga del condensador. La caída de potencial en la malla derecha y la ley de los nudos proporcionan

$$\frac{Q_1}{C} + (I_1 + I_2)R = E(t)$$

Derivando y teniendo en cuenta que $Q_1' = I_1$, se obtiene



$$\frac{I_1}{C}+(I_1'+I_2')R=E'(t)$$

Tomando como variables Q_1, I_1 e I_2, queda el sistema diferencial siguiente

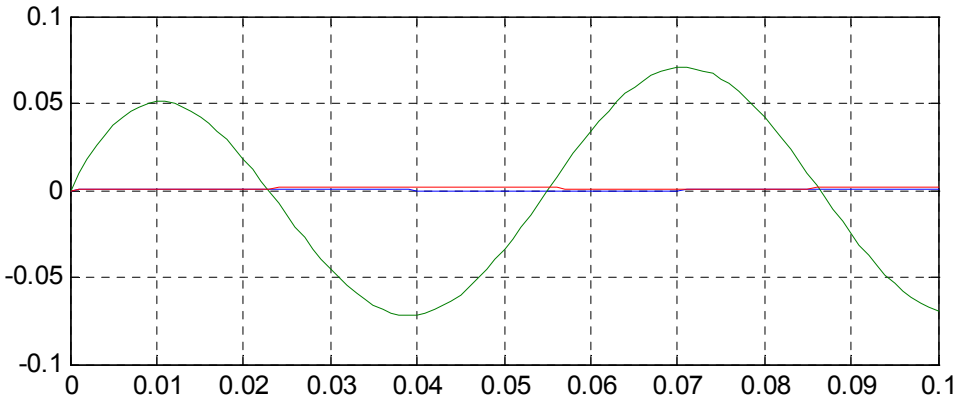
$$Q_1'=I_1$$
$$I_1'=\left(E'(t)-\frac{I_1}{C}-\frac{Q_1}{LC}\right)\Big/R$$
$$I_2'=\frac{Q_1}{LC}$$

Este sistema se codifica en el fichero siguiente

```
function dy=lrc2(t,y)

L=8;R=10;C=1e-3;
dE = 100*cos(100*t);

% Variables
Q1 = y(1);
I1 = y(2);
I2=y(3);
% Ecuación
dQ1 = I1;
dI1 = (dE-I1/C-Q1/L/C)/R;
dI2=Q/L/C;
% Salida
dy = [dQ;dI1;dI2];
```

Resolvemos el problema en un intervalo de 0.1 segundos y representamos el resultado.

```
[t1,y1] = rungekut('lrc2',0:1e-3:1e-1,[0,0,0]);
plot(t1,y1)
grid
```



El aparente aumento de la amplitud de las oscilaciones cesa en poco tiempo, quedando un comportamiento periódico.

Para comprobar el orden del método de Runge-Kutta, obtenemos la solución en un punto, 0.1 por ejemplo con pasos diferentes, cada uno la mitad del anterior. Estimando el error mediante la diferencia entre la solución con paso h y la solución con paso $h/2$,

comprobamos que ésta se divide aproximadamente por 2^4=16 cada vez que el paso se divide por 2.

```
[t2,y2] = rungekut('lrc2',0:0.5e-3:1e-1,[0,0,0]);
[t3,y3] = rungekut('lrc2',0:0.25e-3:1e-1,[0,0,0]);
(y1(end,:)-y2(end,:))./(y2(end,:)-y3(end,:))

ans =   16.1252   16.1313   16.2393
```

4.- Comenzaremos transformando el sistema de orden dos:

$$x'' - 2\omega \operatorname{sen}(\varphi) y' + k^2 x = 0$$
$$y'' + 2\omega \operatorname{sen}(\varphi) x' + k^2 y = 0$$

en uno de primer orden. para ello definimos las variables:

$$z_1 = x \quad z_2 = x' \quad z_3 = y \quad z_4 = y'$$

cuyas derivadas nos permiten definir el sistema de primer orden:

$$z_1' = z_2$$
$$z_2' = 2\omega \operatorname{sen}(\varphi) z_4 - k^2 z_1$$
$$z_3' = z_4$$
$$z_4' = -2\omega \operatorname{sen}(\varphi) z_2 - k^2 z_3$$

que almacenamos en la función:

```
function w=foucault(t,z)
global fi k om
w=[z(2) ; 2*om*sin(fi)*z(4)-k*z(1) ; z(4) ;
 -2*om*sin(fi)*z(2)-k*z(3)];
```

donde declaramos como variables globales la latitud φ y las constantes w y k. Para estudiar el comportamiento del péndulo de Foucault en Valencia (latitud 40º), definimos los valores de las constantes:

```
global fi k2 om
fi=40*pi/180; om=7.29e-5; k2=9.8/10;
```

Definimos el vector de condiciones iniciales y el vector de nodos:

```
z0=[-6 0 1 0];
t=0:.5:3600;
```

y resolvemos con el método de Runge-Kutta y las variables de entrada ya definidas:

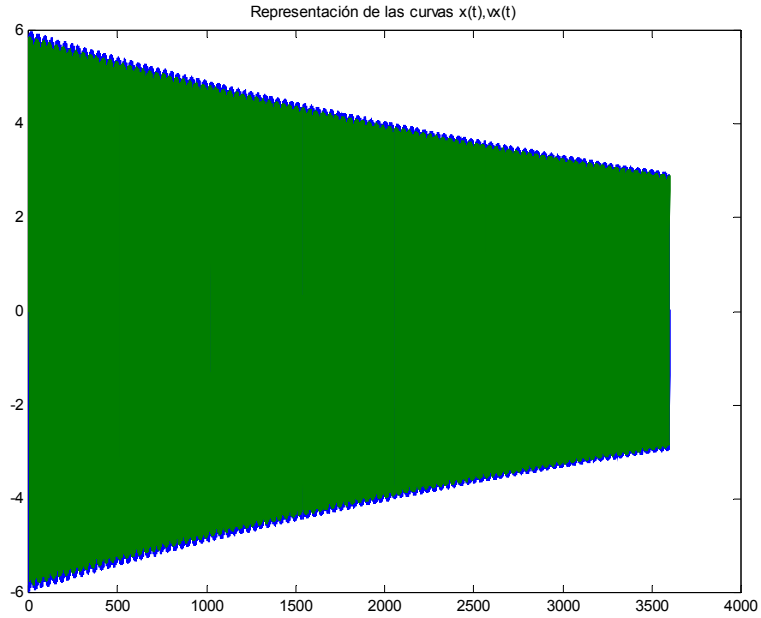
```
[t,y]=rungekut('foucault',t,z0);
sol1=y(end,:)
```

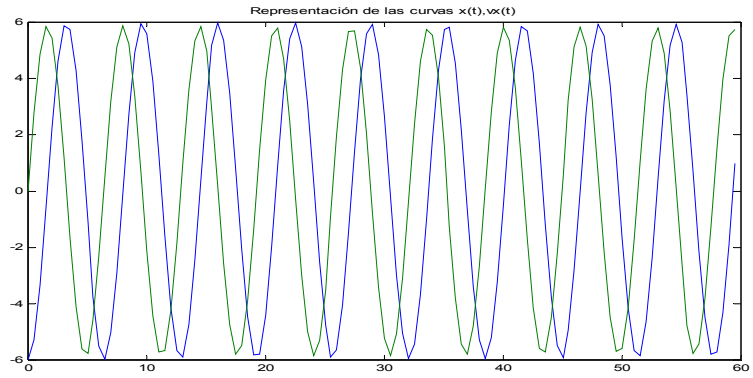
Si representamos gráficamente el movimiento cada una de las componentes de la posición y velocidad del péndulo respecto al tiempo):

```
plot(t,y(,1:2)),
title('Representación de las curvas x(t),vx(t)')
```

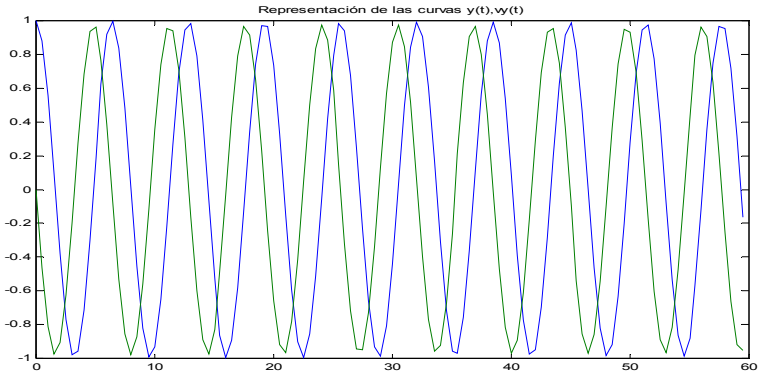
observamos cómo la atracción gravitatoria hace que la amplitud de oscilación (en ambas variables) disminuya con el tiempo. Al observar en detalle la posición del péndulo el primer minuto notamos el carácter ondulatorio del movimiento.

```
plot(t(1:120),y(1:120,1:2)),
title('Representación de las curvas x(t),vx(t)')
```



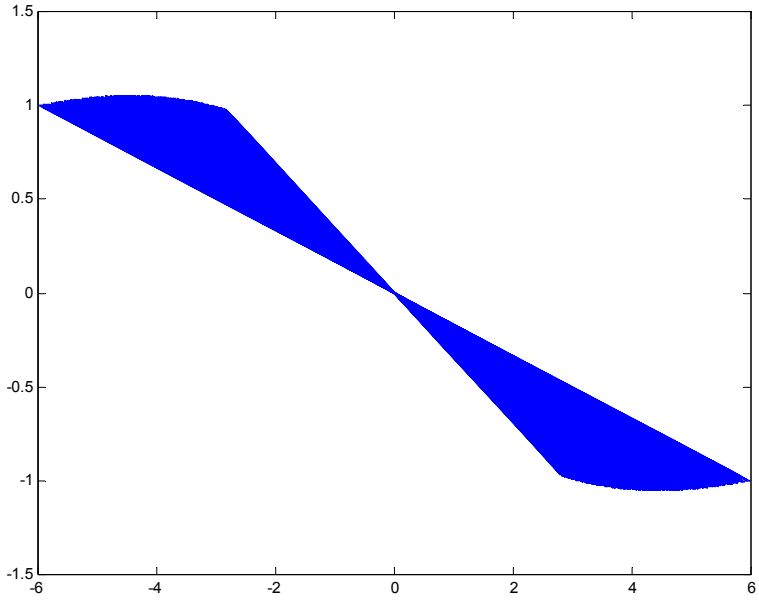



```
plot(t,y(:,3:4))
title('Representación de las curvas y(t),vy(t)')
```



Por otra parte, la representación conjunta de las variables *x* e *y* nos muestra cómo el plano de oscilación gira con el tiempo:

```
plot(y(:,1),y(:,3))
```



Para averiguar cuánto tiempo necesita el plano de oscilación del péndulo de Foucault situado en Valencia para girar 45 grados, calculamos el ángulo (en grados) girado en la primera hora de movimiento:

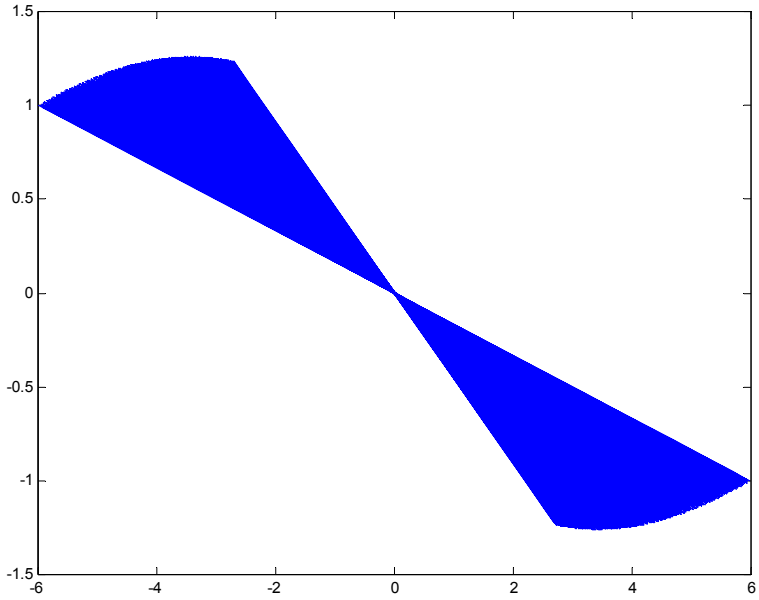
```
angfi=atan2(y(end,3),y(end,1))
angin=atan2(1,-6)
horang=abs(angfi-angin)*180/pi

angfi =
    2.8080
angin =
    2.9764
horang =
    9.6498
```

y el tiempo necesario para girar 45º es de 4 horas, 39 minutos y 47.88 segundos:

```
t45=45/horang

t45 =
    4.6633
```

Al cambiar de situación geográfica, un aumento de latitud provoca una aceleración del giro del plano de oscilación del péndulo, mientras que el acercamiento del péndulo al ecuador hace que el plano de oscilación permanezca constante. Para observar este efecto, basta con modificar el valor de la variable latitud. En el polo norte, la latitud es de 90º; observemos el giro del plano de oscilación en el mismo intervalo de tiempo:

```
fi=pi/2   % latitud en radianes
[t,y]=rungekut('foucault',t,z0);
plot(y(:,1),y(:,3))
```



A continuación calculamos el ángulo de giro por hora:

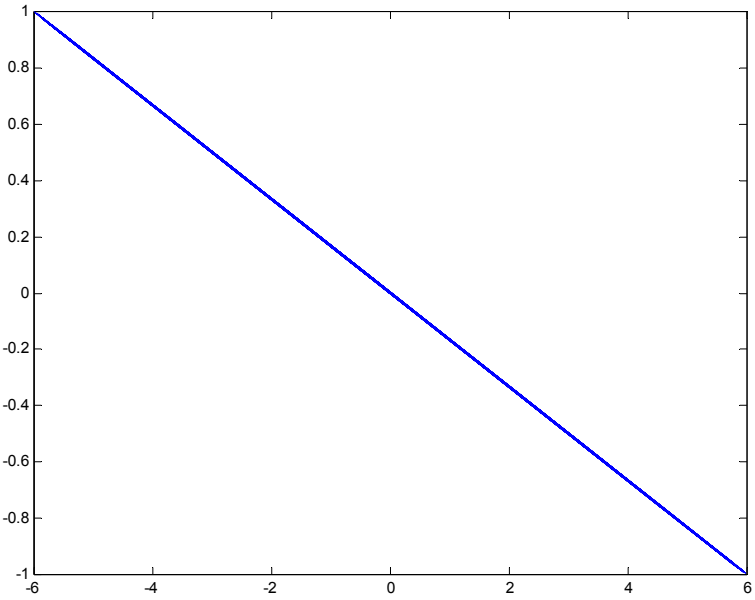
```
angfi=atan2(y(end,3),y(end,1))
angin=atan2(1,-6)
horang=abs(angfi-angin)*180/pi

angfi =
    2.7144
angin =
    2.9764
horang =
   15.0123
```

de lo que se deduce que, en un día, el plano de oscilación del péndulo situado en el polo norte da una vuelta completa:

```
horang*24

ans =
  360.2964
```

Sin embargo, en el ecuador el plano de oscilación no gira:

```
fi=0; [t,y]=rungekut('foucault',t,z0);
plot(y(:,1),y(:,3))
```



El problema de valor inicial

$$\left.\begin{aligned} x'' &= -\mu\frac{x}{r^3}, \quad x(0)=x_0, \quad x'(0)=v_{x0} \\ y'' &= -\mu\frac{y}{r^3}, \quad y(0)=0, \quad y'(0)=v_{y0} \end{aligned}\right\}$$

puede reescribirse como un sistema de cuatro ecuaciones diferenciales de primer orden, introduciendo las nuevas variables:

$$z_1 = x \quad z_2 = x' \quad z_3 = y \quad z_4 = y'$$

cuyas derivadas definen el sistema:

$$\begin{aligned} z_1' &= z_2 \\ z_2' &= -nu\cdot\frac{z_1}{\left(z_1^2+z_3^2\right)^{3/2}} \\ z_3' &= z_4 \\ z_4' &= -nu\cdot\frac{z_3}{\left(z_1^2+z_3^2\right)^{3/2}} \end{aligned}$$

Almacenamos este sistema en la función satelite.m:

```
function w=satelite(t,z)
nu=398598.309;
den=(z(1)^2+z(3)^2)^(3/2);
w=[z(2); -nu*z(1)/den; z(4); -nu*z(3)/den];
```

Para aproximar la trayectoria del satélite en torno a la Tierra, definimos en primer lugar el vector de condiciones iniciales:

```
z0=[39511.0557, -1.07168, 14693.8542, 2.8827];
```

El vector de nodos que emplearemos en la resolución del problema es

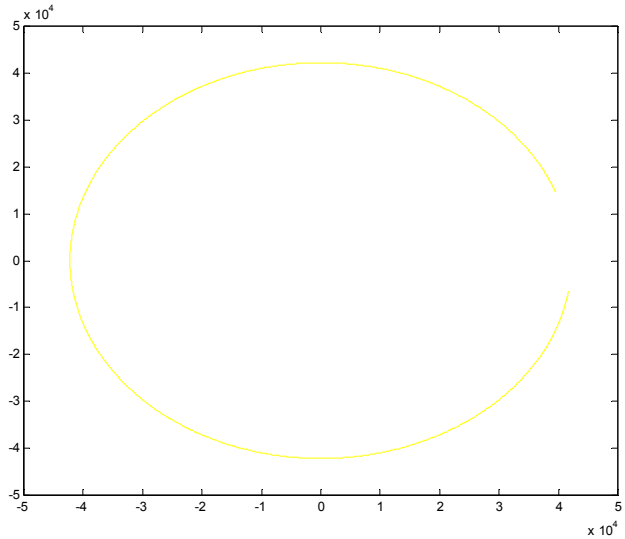
```
t=0:60:24*3600;
```

y la posición y velocidad del satélite en ese instante:

```
[t,y]=ode23('satelite',t,z0);y(end,:)

ans =
  1.0e+004 *
   4.48250114876932            0.00002440421634          -
1.95975100215809   0.00026074263059
```

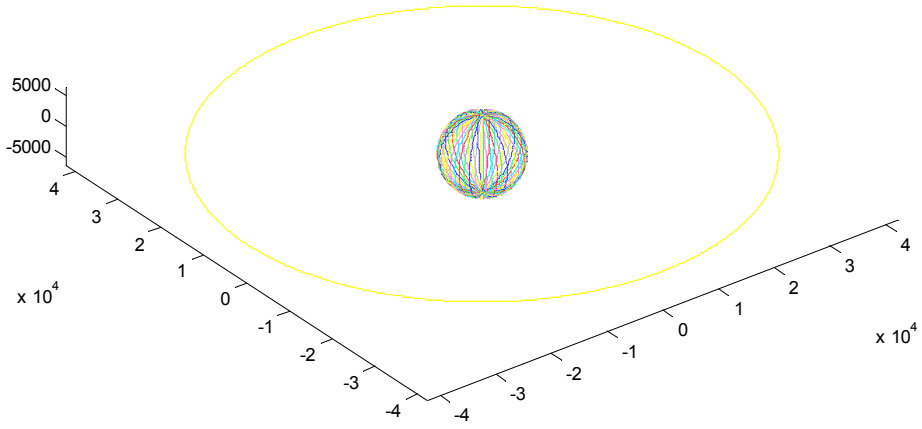
Se comprueba fácilmente que un intervalo de tiempo menor no permite al satélite completar la órbita en torno a la Tierra. Concluimos, por tanto, que el periodo orbital es de un día.

```
[t,y]=ode23('satelite',0:60:22*3600,z0);plot(y(:,1),y(:,3))
```



Para representar una esfera centrada en el origen con radio R_T, y la trayectoria del satélite en torno a ella:

```
[t,y]=ode23('satelite',t,z0);
plot3(y(:,1),y(:,3),zeros(size(y(:,1)))), hold on
[X,Y,Z]=sphere(50);
X=6378.1*X;
Y=6378.1*Y;
Z=6378.1*Z;
plot3(X,Y,Z), axis equal
```


5000
0
-5000
4
3
2
1
0
-1
-2
-3
-4
x 10^4
4
3
2
1
0
-1
-2
-3
-4
x 10^4

Zorros contra Conejos

```
>> tspan = 0:0.03:30;
>> [t,y] = mieuler('presarapaz',tspan,[10,10]);
```

Grafico vs tiempo

```
>> plot(t,y(:,1),'g');
>> hold on
>> plot(t,y(:,2),'r');
```

Conejos vs Zorros

```
>> plot(y(:,1),y(:,2))
```

Circuito LRC (sin R)

```
>> [t,y] = mieuler('lrc',tspan,[10,10]);
>> plot(t,y(:,1),'g');
hold on
plot(t,y(:,2),'r');
```

Sale que va creciendo sin haber fem! Imposible
Se debe a fallos en Euler.
Como es lineal requiere muchos calculos -> errores redondeo
habria que aumentar la 'resolucion'

```
[t,y] = mieuler('lrc',0:0.003:30,[10,10]);
```

Aun asi sigue aumentando bastante...

Solucion: Runge-Kutta (adaptada a vectores)

```
[t,y] = rungekut('lrc',0:0.03:30,[1,0]);
plot(t,y(:,1),'g');
hold on
plot(t,y(:,2),'r');
```

Circuito LRC (con R) (fichero lrcg con variables globales)

```
>> global L R C
>> L = 2; R=0.3; C=1;

>> [t,y] = rungekut('lrcg',0:0.03:30,[1,0]);
plot(t,y(:,1),'g');
hold on
plot(t,y(:,2),'r');
```

se puede ir probando

```
>> R = 3
>> close; [t,y] = rungekut('lrcg',0:0.03:30,[1,0]); plot(t,y(:,1),'g'); hold on; plot(t,y(:,2),'r'); grid
>> C = 3
>> close; [t,y] = rungekut('lrcg',0:0.03:30,[1,0]); plot(t,y(:,1),'g'); hold on; plot(t,y(:,2),'r'); grid
>> R =0.3
>> close; [t,y] = rungekut('lrcg',0:0.03:30,[1,0]); plot(t,y(:,1),'g'); hold on; plot(t,y(:,2),'r'); grid
```

Funcion de Matlab

(ODE45)

```
>> close; [t,y] = ode45('lrcg',0:0.03:30,[1,0]); plot(t,y(:,1),'g'); hold on; plot(t,y(:,2),'r'); grid
```

Si no le damos salido lo dibuja el solito.
Ademas le podemos dar el intervalo y pone los puntos segun convenga

```
>> C = 0.3; R = 2
>> ode45('lrcg',[0,30],[1,0]);
```

Comprobacion de que el error Runge-Kutte es orden 4.

El doble de puntos hace un error 2^4 = 16 veces menor

```
>> h=0.05;
[t,y1] = rungekut('lrcg',0:h:2,[1,0]);
[t,y2] = rungekut('lrcg',0:h/2:2,[1,0]);
[t,y3] = rungekut('lrcg',0:h/4:2,[1,0]);
```

```
(y2(end)-y1(end))/(y3(end)-y2(end))

ans =
   17.2394

>> (y2(end,:)-y1(end,:))./(y3(end,:)-y2(end,:))
ans =
   15.7684   17.2394
```

19. Teoría cualitativa

La *teoría cualitativa* analiza el comportamiento de las soluciones de un sistema de ecuaciones diferenciales sin necesidad de obtenerlas de forma explícita. Dado que los métodos analíticos sólo nos permiten resolver algunos casos particulares, como los lineales con coeficientes constantes, el análisis cualitativo de dichas soluciones (si hay comportamientos periódicos, puntos a los que tienden las trayectorias o de los que se alejan,…) resulta de gran interés.

En este capítulo analizaremos las propiedades cualitativas de las soluciones de sistemas de ecuaciones diferenciales autónomos:

$$\begin{aligned} x_1' &= f_1\left(x_1, x_2, ..., x_n\right) \\ x_2' &= f_2\left(x_1, x_2, ..., x_n\right) \\ &\vdots \\ x_n' &= f_n\left(x_1, x_2, ..., x_n\right) \end{aligned}$$

es decir, sistemas de ecuaciones diferenciales en los que no aparece de forma explícita la variable independiente *t*. Empleando notación vectorial, podemos reescribir el sistema anterior del siguiente modo:

$$\boldsymbol{x}' = \boldsymbol{f}\left(\boldsymbol{x}\right)$$

donde $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x} = \left(x_1, ..., x_n\right)$ y $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$, $\boldsymbol{f}\left(\boldsymbol{x}\right) = \left(f_1\left(x_1, ..., x_n\right), ..., f_n\left(x_1, ..., x_n\right)\right)$.

En primer lugar, estudiaremos los sistemas lineales con coeficientes constantes, introduciendo ideas que se concretarán más adelante para sistemas no lineales:

- Soluciones de equilibrio:valores en los que el sistema puede permanecer indefinidamente; en el caso de sistemas lineales, el origen es un punto de equilibrio;
- Órbitas periódicas: trayectorias cerradas en las que, siendo solución del sistema, el comportamiento se repite tras un cierto tiempo;
- Estabilidad: dadas dos soluciones muy próximas en un instante concreto, ¿permanecerán próximas en el futuro? ¿Se acercarán a una solución del sistema, se alejarán de ella o quizá seguirán caminos distintos?.

Las herramientas que nos permitirán describir este tipo de comportamiento serán los valores propios de la matriz de coeficientes del sistema: si los valores propios son simples, las trayectorias tienden o se alejan de los puntos de equilibrio; los valores propios múltiples provocan comportamientos degenerados, mientras que si son imaginarios puros, las soluciones del sistema serán elipses en torno al punto crítico; dichas elipses degeneran en espirales cuando dichos valores propios tienen parte real no nula.

En la segunda sección, el análisis de los sistemas no lineales se centra, por una parte, en encontrar un sistema lineal próximo, cuyo comportamiento en el entorno del punto

de equilibrio pueda extrapolarse al sistema no lineal; por otra parte, los sistemas no lineales admiten la existencia de soluciones periódicas a las que tienden (o de las que parten) las trayectorias de su entorno, son los llamados cilcos límite. En esta sección de proporcionan algunos resultados teóricos que permiten deducir o descartar la existencia de trayectorias cerradas.

19.1. Sistemas lineales

Consideraremos en esta sección sistemas de n ecuaciones diferenciales de primer orden lineales, con coeficientes constantes:

$$\begin{aligned} x_1' &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ x_2' &= a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ &\vdots \\ x_n' &= a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n \end{aligned}$$

donde a_{ij}, $i = 1,\ldots,n$, $j = 1,\ldots,n$ son n^2 constantes reales. Es posible reescribir este sistema, empleando notación matricial,

$$\boldsymbol{x}'(t) = \boldsymbol{A}\cdot\boldsymbol{x}$$

de tal modo que la matriz **A** es denotada por

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

y puede interpretarse como un campo vectorial sobre $\mathbb{R}^n$, $\boldsymbol{A}:\mathbb{R}^n \to \mathbb{R}^n$ que actúa sobre $x \in \mathbb{R}^n$. Así, una solución de este sistema es una curva $\boldsymbol{x}:\mathbb{R} \to \mathbb{R}^n$ cuyo vector tangente en un cierto instante t es $A\cdot x(t)$.

19.1.1. Puntos críticos y soluciones periódicas

Una solución específica de la ecuación diferencial autónoma

$$\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x})$$

se llama **solución de equilibrio** (o **punto crítico**) si es constante en el tiempo, es decir, $\boldsymbol{x}(t) = \boldsymbol{x}_e$ constante. Para que $\boldsymbol{x}_e$ sea solución de equilibrio debe verificar la ecuación diferencial y, al no depender del tiempo, se verifica:

$$\boldsymbol{x}_e' = \boldsymbol{f}(\boldsymbol{x}_e) = \boldsymbol{0}$$

Así, todos los ceros de la función $\boldsymbol{f}(\boldsymbol{x})$ son soluciones de equilibrio. En el caso particular de los sistemas lineales, $\boldsymbol{x}'(t) = \boldsymbol{A}\cdot\boldsymbol{x}$, cuando la matriz $\boldsymbol{A}$ es no singular puede concluirse que $\boldsymbol{x}_e = \boldsymbol{0}$ es el único punto crítico del sistema.

Ejemplo 1

En el modelo de población de Malthus y Verhulst, las soluciones de equilibrio $\boldsymbol{p}_e$ de la ecuación diferencial autónoma de primer orden:

$$p' = p(a - b\cdot p), \quad a,b > 0$$

son los ceros de la función $f(p) = p(a - b \cdot p)$, por lo que $p_e = 0$ y $p_e = \frac{a}{b}$ serán los puntos críticos de la ecuación diferencial.

Por otra parte, consideremos una solución $\boldsymbol{x} = \boldsymbol{x}(t)$ del problema $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x})$. Si existe $p \in \mathbb{R}$ tal que $\boldsymbol{x}(t+p) = \boldsymbol{x}(t)$, dicha solución recibe el nombre de **órbita** o **solución periódica** de la ecuación diferencial o sistema. De este modo, la solución es una trayectoria cerrada que puede recorrerse en p unidades de tiempo.

19.1.2. Plano de fase

En dimensión dos, recordemos que una solución del sistema

$$x' = f_1(x, y)$$
$$y' = f_2(x, y)$$

es una función vectorial $(x, y)^T = \varphi(t)$ que satisface las ecuaciones diferenciales del sistema. Podemos pensar en dicha solución como la representación paramétrica de una curva en el plano xy.

Resulta útil con frecuencia imaginarse la curva solución del sistema como la trayectoria recorrida por una partícula en movimiento cuya velocidad (respecto a cada una de sus componentes) queda especificada en el sistema de ecuaciones diferenciales. Así, diremos que un conjunto de trayectorias en el plano *xy*, o **plano fase** o **plano de fases**, se llama **retrato fase** del sistema.

19.1.3. Análisis de estabilidad

Consideremos un punto crítico $\boldsymbol{x}_e$ de un sistema autónomo lineal $\boldsymbol{x}'(t) = \boldsymbol{f}(\boldsymbol{x})$ y una solución $\boldsymbol{x} = \boldsymbol{x}(t)$ de dicho sistema que verifica la condición inicial $\boldsymbol{x}(0) = \boldsymbol{x}_0$. Si interpretamos la solución como la trayectoria de una partícula en movimiento, sometida a las condiciones descritas por el sistema de ecuaciones diferenciales, podemos preguntarnos cuál será el comportamiento de dicha partícula en las proximidades del punto crítico. Es decir, cuando $\boldsymbol{x}_0$ es próximo a $\boldsymbol{x}_e$, ¿se mantendrá la trayectoria en las proximidades del punto de equilibrio o se alejará definitivamente de él?; y en el primer caso, ¿tenderá dicha trayectoria al valor de equilibrio?. Responderemos a continuación a estas preguntas para el caso particular de sistemas lineales, dejando para la sección siguiente la respuesta correspondiente al caso no lineal, que precisa de otro tipo de herramientas.

Trabajaremos en lo sucesivo con dimensión 2, es decir, nos ceñiremos al caso de un sistema autónomo lineal formado por dos ecuaciones diferenciales. En casos de dimensión superior, se trata de reducir el problema a varios problemas bidimensionales, para aplicar sobre cada uno de ellos las técnicas que describiremos a continuación.

Para analizar la estabilidad del sistema, interpretaremos las soluciones de

$$x' = a \cdot x + b \cdot y$$
$$y' = c \cdot x + d \cdot y$$

en términos de los valores y vectores propios de la matriz de coeficientes:

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Para asegurar la unicidad del punto crítico (0,0), supondremos que el determinante $\delta = a \cdot d - c \cdot b \neq 0$. Si denotamos la traza de la matriz por $\tau = a + d$, entonces la ecuación característica $\det(\boldsymbol{A} - \lambda \boldsymbol{I}) = 0$ se puede reescribir como $\lambda^2 - \tau\lambda + \delta = 0$. Así, los valores propios de $\boldsymbol{A}$ son $\lambda = \dfrac{\tau \pm \sqrt{\tau^2 - 4\delta}}{2}$, y por tanto podemos afirmar que los valores propios de A serán reales y distintos cuando $\tau^2 - 4\delta > 0$, reales e iguales cuando $\tau^2 - 4\delta = 0$ y complejos si $\tau^2 - 4\delta < 0$. A continuación analizaremos, mediante un ejemplo, el comportamiento de las trayectorias solución del sistema en las proximidades del origen en cada una de estas circunstancias.

Ejemplo 2

Consideraremos el sistema lineal en términos de un parámetro c:

$$x' = -x + y$$

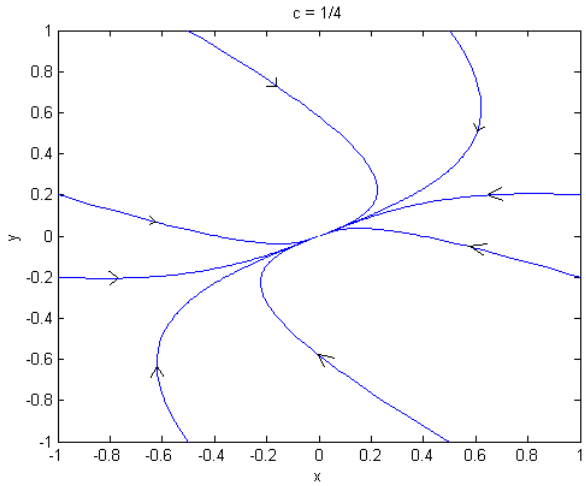
$$y' = c \cdot x - y$$

Obviamente el origen de coordenadas (0,0) es un punto de equilibrio del sistema. Emplearemos el algoritmo de Runge-Kutta implementado en la práctica anterior para representar los espacios de fase de dicho sistema en los casos $c = \dfrac{1}{4}, 4, 0, -9$.

La traza de la matriz de coeficientes es $\tau = -2$ y el determinante $\delta = 1 - c$ (notemos que en nuestro caso $c \neq 1$, luego el determinante no se anula y está asegurada la unicidad del punto crítico), por lo que los valores propios asociados a la matriz son

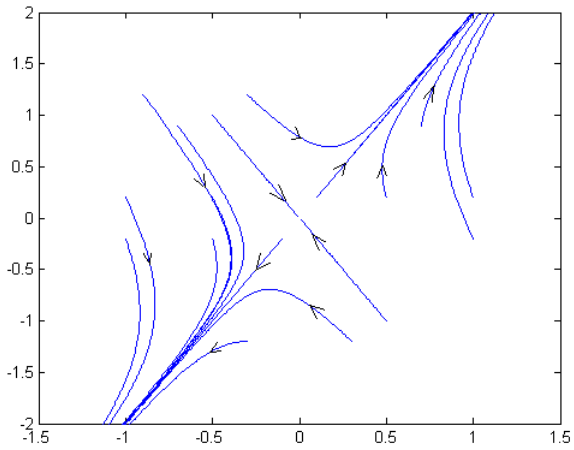
$$\lambda = \frac{\tau \pm \sqrt{\tau^2 - 4\delta}}{2} = -1 \pm \sqrt{c}$$

Así, para $c = \dfrac{1}{4}$, los valores propios son negativos y distintos, $\lambda = -\dfrac{1}{2}$ y $\lambda = -\dfrac{3}{2}$. Aplicando el método de Runge-Kutta al sistema con este valor del parámetro con distintas condiciones iniciales ($[\pm 0.5, \pm 1]$ y $[\pm 1, \pm 0.2]$), obtenemos el siguiente retrato de fase:

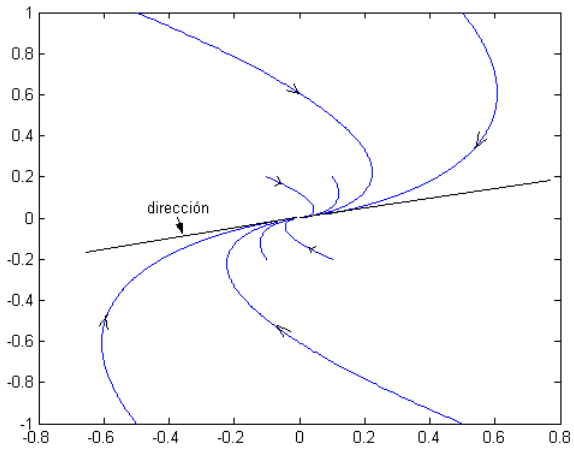


Notemos que, para este valor del parámetro c, todas las trayectorias parecen tender al origen desde cualquier dirección.

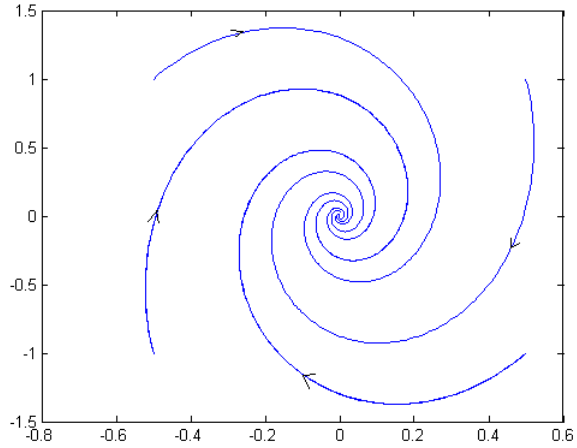
Cuando c = 4, los valores propios tienen signos contrarios, $\lambda = 1$ y $\lambda = -3$. Observamos en el retrato de fase que todas las trayectorias se alejan del origen, aparentemente en una dirección concreta, a excepción de las soluciones que comienzan sobre una determinada recta, que se acercan al punto crítico.



Si el parámetro c toma valor nulo, se llega a un único valor propio real. En este caso, al dibujar el retrato de fase observamos que todas las trayectorias se acercan al punto crítico, pero siguiendo una dirección concreta.



Por último, para c = -9, los valores propios son complejos conjugados, con parte real negativa: $\lambda = -1 \pm 3i$. La trayectoria de las soluciones es una espiral hacia el origen cuando aumenta la variable independiente t.

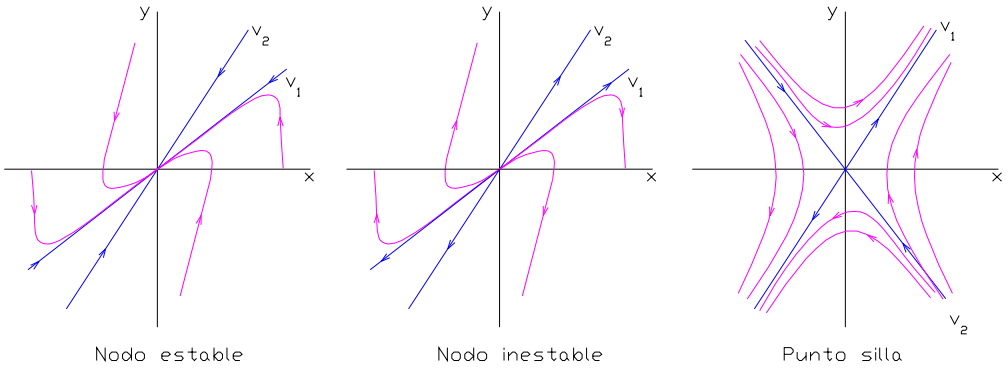


En general, para analizar en detalle el comportamiento de las trayectorias estudiaremos los valores y vectores propios asociados a la matriz de coeficientes del sistema. Teniendo en cuenta que la solución general del sistema es:

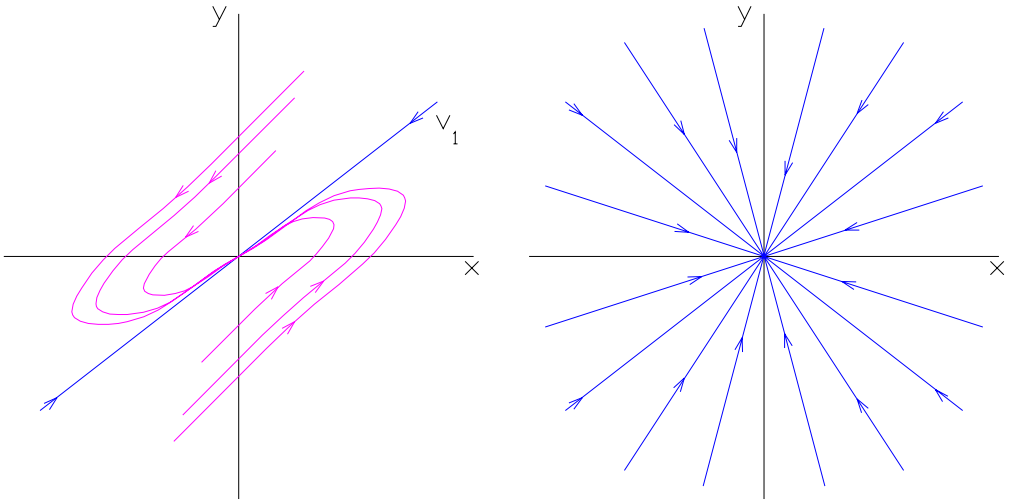
$$\boldsymbol{x}(t) = c_1\boldsymbol{v}_1 e^{\lambda_1 t} + c_2\boldsymbol{v}_2 e^{\lambda_2 t}$$

distinguiremos los siguientes casos, especificando en cada uno de ellos el signo del determinante y traza de la matriz de coeficientes, así como la estabilidad del punto de equilibrio.

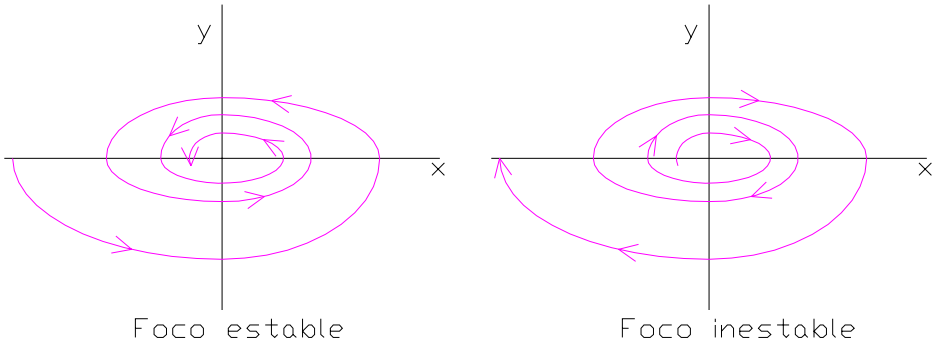
1. Valores propios distintos ($\tau^2 - 4\delta > 0$)
   a. Ambos valores propios son negativos ($\tau^2 - 4\delta > 0$, $\tau < 0$ y $\delta > 0$): las trayectorias solución tienden al origen desde cualquier valor inicial del plano de fase a lo largo de la recta determinada por el vector propio v_1 (si $c_1 \neq 0$) o por el vector propio v_2, si $c_1 = 0$. En este caso, el punto de equilibrio recibe el nombre de nodo estable, $\lambda_2 < \lambda_1 < 0$.
   b. Ambos valores propios son positivos ($\tau^2 - 4\delta > 0$, $\tau > 0$ y $\delta > 0$): las soluciones se alejan del origen desde cualquier valor inicial en una de las direcciones determinadas por el vector propio v_1 (si $c_1 \neq 0$) o por el vector propio v_2, si $c_1 = 0$; Este tipo de punto de equilibrio recibe el nombre de nodo inestable, $0 < \lambda_2 < \lambda_1$.
   c. Los valores propios tienen signos opuestos ($\tau^2 - 4\delta > 0$ y $\delta < 0$). En este caso, el análisis de las soluciones es similar al caso anterior, salvo por una circunstancia: cuando $c_1 = 0$, la solución tiende a cero siguiendo la recta determinada por v_2. Sin embargo, para $c_1 \neq 0$, la recta determinada por v_1 es una asíntota para las trayectorias que se alejan del origen. Este punto crítico recibe el nombre de punto silla.



2. Un valor propio real de multiplicidad dos ($\tau^2 - 4\delta = 0$): en este caso el punto crítico recibe el nombre de nodo estable (inestable) degenerado si las trayectorias solución tienden (se alejan) al origen. Además, la solución general tiene comportamientos distintos según si se pueden determinar uno o dos valores propios linealmente independientes: en el primer caso, la recta determinada por el vector propio es una asíntota para todas las soluciones, mientras que en el segundo caso la distribución de las trayectorias solución respecto al punto crítico es radial.



3. Valores propios complejos ($\tau^2 - 4\delta < 0$): distinguiremos dos casos según si los valores propios son, o no, imaginarios puros.
   a. Valores propios imaginarios puros ($\tau^2 - 4\delta < 0$, $\tau = 0$): En este caso las soluciones del sistema son periódicas y describen elipses con el mismo sentido en torno al punto crítico, que recibe el nombre de centro.
   b. Valor propio imaginario no puro ($\tau^2 - 4\delta < 0$, $\tau \neq 0$): las trayectorias describen espirales en torno al punto crítico. Dichas espirales se acercan cada vez más al origen cuando la parte real de los valores propios sea negativa (en cuyo caso hablaremos del origen como punto espiral estable o foco estable) y se alejarán del punto crítico (espiral/foco inestable) cuando dicha parte real sea positiva.



Retomemos a continuación el ejemplo anterior para analizar la estabilidad del origen para cada uno de los valores del parámetro c:

Ejemplo 3

Consideremos de nuevo el sistema lineal para $c = \frac{1}{4}, 4, 0, -9$.

$$x' = -x + y$$
$$y' = c \cdot x - y$$

Sabemos que los valores de la traza y el determinante son, respectivamente $\tau = -2$ y $\delta = 1 - c$. Así, los valores propios asociados a la matriz son $\lambda = \frac{\tau \pm \sqrt{\tau^2 - 4\delta}}{2} = -1 \pm \sqrt{c}$

Por tanto, es fácil construir la siguiente tabla para clasificar en cada caso el punto crítico:

c	τ	δ	$\tau^2 - 4\delta$	**Punto crítico**
0.25	-2	0.75	1>0	Nodo estable
4	-2	-3	16>0	Punto de silla
0	-2	1	0	Nodo estable degenerado
-9	-2	10	-36<0	Espiral/foco estable

Por otra parte, también es posible dicha clasificación atendiendo únicamente a los valores propios de la matriz de coeficientes del sistema. Empleando el comando **eig** de MATLAB, obtenemos la información necesaria:

```
c=1/4;
A=[-1,1;c,-1];
eig(A)

ans =
   -0.5000
   -1.5000
```

A la vista de los resultados obtenidos, y dado que se trata de dos valores propios reales y distintos, del mismo signo, podemos concluir que el origen es, para este valor del parámetro c, un nodo estable. Repitiendo los cálculos para el resto de los valores posibles del parámetro, obtenemos idénticas conclusiones a las mostradas en la tabla anterior.

19.2. Sistemas no lineales

En la presente sección consideraremos un sistema autónomo no lineal bidimensional $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x})$ y examinaremos el comportamiento de las trayectorias del sistema en el

entorno de un punto crítico $\boldsymbol{x}_e$. Es conveniente que dicho punto crítico esté en el origen; si no es así, un pequeño cambio de variable ($\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x}_e$) nos proporcionará un sistema equivalente al inicial cuyo punto de equilibrio estará en el origen del plano de fases.

Para estudiar la estabilidad del sistema en el entorno de un punto crítico, trataremos de encontrar un sistema lineal próximo a nuestro sistema no lineal:

$$\boldsymbol{x}' = \boldsymbol{A} \cdot \boldsymbol{x} + \boldsymbol{g}(\boldsymbol{x})$$

de manera que $\boldsymbol{x}_e = \boldsymbol{0}$ sea un punto crítico aislado del sistema, es decir, existe un entorno del origen en el cual no hay otros puntos críticos. Además, supondremos que $\det \boldsymbol{A} \neq 0$, por lo que el origen también será un punto crítico del sistema lineal asociado. Por último, supondremos que las componentes de g tienen primeras derivadas parciales continuas y satisfacen la condición:

$$\frac{\|\boldsymbol{g}(\boldsymbol{x})\|}{\|\boldsymbol{x}\|} \to 0 \quad \text{cuando} \quad \boldsymbol{x} \to \boldsymbol{0}$$

es decir, que cerca del origen el comportamiento del sistema original es casi lineal.

A continuación describiremos un método general para encontrar el sistema lineal $\boldsymbol{x}' = \boldsymbol{A} \cdot \boldsymbol{x}$ correspondiente a un sistema casi lineal $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x})$ cerca de un punto crítico dado $\boldsymbol{x}_e$.

Consideremos el sistema no lineal bidimensional:

$$x' = f_1(x, y)$$
$$y' = f_2(x, y)$$

tal que $f_i \in C^2(D)$, $i = 1, 2$ siendo $D \subseteq \mathbb{R}^2$ un entorno del punto crítico; dicho sistema es casi lineal en el entorno del punto de equilibrio, ya que aplicando el desarrollo de Taylor a ambas funciones:

$$f_1(x, y) = f_1(x_e, y_e) + \frac{\partial f_1}{\partial x}(x_e, y_e)(x - x_e) + \frac{\partial f_1}{\partial y}(x_e, y_e)(y - y_e) + \eta_1(x, y)$$

$$f_2(x, y) = f_2(x_e, y_e) + \frac{\partial f_2}{\partial x}(x_e, y_e)(x - x_e) + \frac{\partial f_2}{\partial y}(x_e, y_e)(y - y_e) + \eta_2(x, y)$$

los términos del error $\eta_i(x, y), i = 1, 2$ verifican la condición

$$\frac{\eta_i(x, y)}{\|(x - x_e, y - y_e)\|} \to 0 \quad \text{cuando} \quad (x, y) \to (x_e, y_e)$$

Así, el sistema se puede expresar

$$\begin{pmatrix} x - x_e \\ y - y_e \end{pmatrix}' = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix} \begin{pmatrix} x - x_e \\ y - y_e \end{pmatrix} + \begin{pmatrix} \eta_1(x, y) \\ \eta_2(x, y) \end{pmatrix}$$

y en notación vectorial:

$$\boldsymbol{u}' = J_f(\boldsymbol{x}_e) \cdot \boldsymbol{u} + \boldsymbol{\eta}(\boldsymbol{x})$$

donde $\boldsymbol{u}^T = (x - x_e, y - y_e)$ y $\boldsymbol{\eta} = (\eta_1, \eta_2)$. Dado que el término no lineal $\boldsymbol{\eta}(\boldsymbol{x})$ es pequeño en comparación con el lineal $J_f(\boldsymbol{x}_e) \cdot \boldsymbol{u}$, cuando $\boldsymbol{u}$ es pequeño (es decir, cuando estamos en las proximidades del punto crítico), es razonable esperar que las trayectorias solución del sistema lineal sean buenas aproximaciones a las del sistema no lineal. Esto es cierto en la mayoría de los casos, aunque en algunos, como los de tipo

centro (los más susceptibles a pequeñas perturbaciones del sistema), puede haber problemas para determinar la estabilidad.

Ejemplo 4

El movimiento no amortiguado de un péndulo de longitud $l = 1$ se describe por el sistema:

$$x' = y$$

$$y' = -\frac{g}{l}\sin x$$

donde g es la aceleración de la gravedad. Sabemos que $(0,0)$ y $(\pi,0)$ son puntos de equilibrio; vamos a continuación a estudiar su estabilidad.
El sistema lineal asociado al problema del péndulo, cuya matriz de coeficientes es la matriz jacobiana del sistema, es:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

en el caso de $(x_e, y_e) = (0,0)$, y

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \frac{g}{l} & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

en el caso $(x_e, y_e) = (\pi,0)$, donde $u = x - \pi, v = y$.

Deduciremos la estabilidad de sendos puntos crítico empleando los valores propios de la matriz jacobiana en cada caso:

```
A=[0 1; -9.8/1 0];
  eig(A)

  ans =
          0 + 3.1305i
          0 - 3.1305i
```

de donde se deduce que el origen es un punto crítico de tipo centro. Respecto al segundo punto crítico,

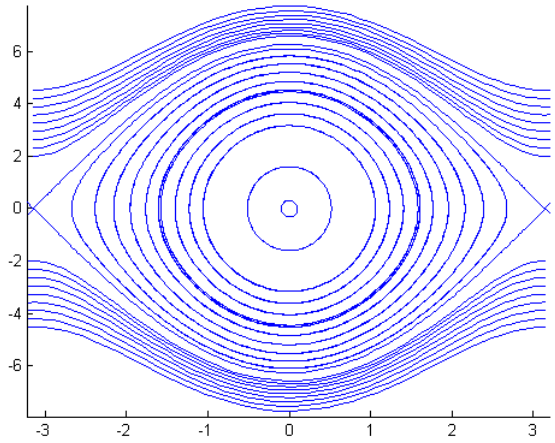
```
A=[0 1; 9.8/1 0];
  eig(A)

  ans =
      3.1305
     -3.1305
```

al tener dos valores propios reales y de distinto signo, se deduce que el punto crítico $(\pi,0)$ es un punto silla.

Podemos confirmar dichas conclusiones representando el espacio de fase del problema del péndulo con valores iniciales en el entorno de sendos puntos críticos: mediante las instrucciones

```
  [t,y]=rk4('pendulo',0,5,[0.1,0.1],500);
  plot(y(:,1),y(:,2))
```

representamos una elipse en torno al punto crítico del origen. Podemos representar estas curvas junto con las correspondientes a un entorno del otro punto crítico, en las que se confirman las conclusiones a las que habíamos llegado mediante el estudio de la estabilidad.



En general, dado un sistema no lineal de n ecuaciones de primer orden $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x})$, diremos que un punto de equilibrio $\boldsymbol{x}_e$ es hiperbólico si ningún valor propio de la matriz jacobiana $J_f(\boldsymbol{x}_e)$ tiene parte real nula, en otro caso se dice que el punto crítico es elíptico.

Por otra parte, un punto crítico hiperbólico $\boldsymbol{x}_e$ se llama sumidero (o punto crítico asintóticamente estable) si todos los valores propios de $J_f(\boldsymbol{x}_e)$ tienen parte real negativa; diremos que dicho punto es una fuente (o punto crítico inestable) si todos tienen parte real positiva y punto de silla si tiene al menos un valor propio con parte real positiva y otro con parte real negativa.

19.2.1. Soluciones periódicas y ciclos límite

Consideremos de nuevo el sistema no lineal bidimensional:

$$x' = f_1(x, y)$$
$$y' = f_2(x, y)$$

tal que las funciones f_i , $i = 1, 2$ y sus primeras derivadas parciales son continuas en el espacio de fases. Empleando las técnicas descritas hasta el momento somos capaces de obtener información sobre las trayectorias solución del sistema únicamente en los entornos de sus puntos críticos. Para obtener las propiedades globales de las trayectorias en regiones amplias del plano de fases es útil saber si existen o no trayectorias cerradas, o lo que es equivalente, soluciones periódicas.

Sabemos por la sección anterior que un sistema lineal tiene trayectorias cerradas si y sólo si los valores propios asociados a la matriz de coeficientes del sistema son imaginarios puros, en cuyo caso toda trayectoria es cerrada. Así, para un sistema lineal se puede concluir que todas las trayectorias son cerradas y, por tanto periódicas, o ninguna lo es. Al contrario, un sistema no lineal puede tener una trayectoria cerrada de tal manera que no haya ninguna otra próxima a ella, como se muestra en el siguiente ejemplo.

Ejemplo 5

Sea el sistema:

$$x' = -y + x\left(1 - x^2 - y^2\right)$$
$$y' = x + y\left(1 - x^2 - y^2\right)$$

en el que resulta más fácil emplear coordenadas polares: $r^2 = x^2 + y^2$ y $\theta = \arctan\frac{y}{x}$. De este modo, las ecuaciones equivalentes en polares son:

$$r' = r\left(1 - r^2\right)$$

$$\theta' = 1$$

El único punto de equilibrio es el origen del plano de fase, ya que r = 1 sólo anula una de las ecuaciones diferenciales. El sistema linealizado en torno al origen es:

$$\begin{pmatrix} r' \\ \theta' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} r \\ \theta \end{pmatrix}$$

donde la matriz de coeficientes es singular. Para estudiar la estabilidad del origen utilizaremos la linealización del sistema en coordenadas cartesianas:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

que es no singular. Dado que la traza $\tau = 2 > 0$ y el determinante $\delta = 2 > 0$ (o alternativamente que los valores propios son complejos, $1 \pm i$), podemos afirmar que el punto crítico es un foco o espiral inestable, y por tanto las trayectorias en un entorno del origen, se alejarán de éste.

Por otra parte, r = 1, $\theta = t + t_0$ es solución del sistema de ecuaciones diferenciales (expresado en polares). Esta es una solución periódica del sistema y además sabemos que las trayectorias solución en un entorno del origen se alejan en espiral de éste y, por tanto, se aproximan a la circunferencia unidad. De hecho, si observamos la ecuación diferencial $r' = r\left(1 - r^2\right)$, para $0 < r < 1$ se verifica que $r' > 0$ y por tanto la variación de la distancia al origen es positiva y cada vez mayor hasta que para r = 1 se anula dicha variación. Esto confirma que el origen sea un foco inestable y que las trayectorias solución tiendan a la órbita periódica pero, ¿qué ocurre más allá de dicha órbita? Siguiendo el mismo razonamiento, para $r > 1$ se verifica $r' < 0$ cada vez mayor en valor absoluto. Así, la variación de la distancia al origen es cada vez menor hasta que para r = 1 se anula dicha variación. Por tanto, no es descabellado atribuir a la órbita periódica un carácter atractor.

En general, una trayectoria cerrada en el plano fase tal que otras trayectorias no cerradas tienden en espiral hacia ella, desde el interior o desde el exterior, se conoce como ciclo límite. De este modo, en el ejemplo anterior la curva r = 1 es un **ciclo límite** del sistema. Concretamente, es un ciclo **estable** o **atractor**, ya que todas las trayectorias que se inician en sus proximidades tienden en espiral hacia ella. Si las trayectorias en uno de sus lados se aproximan en espiral a la trayectoria cerrada y por el otro lado se alejan en espiral de ésta, se dice que el **ciclo límite** es **semiestable**. Sin embargo, si las trayectorias próximas al ciclo límite se alejan en espiral de éste, se dice que dicha trayectoria cerrada es **inestable**.

En el ejemplo anterior hemos deducido la existencia de la trayectoria periódica mediante manipulaciones de las ecuaciones diferenciales. Sin embargo, esto no siempre es posible; son necesarios ciertos resultados generales que nos permitan conocer la existencia o no existencia de ciclos límite en sistemas autónomos no lineales.

Teorema (Poincaré):

Una trayectoria cerrada del sistema autónomo no lineal

$$x' = f_1(x, y)$$
$$y' = f_2(x, y)$$

tal que las funciones f_i , $i = 1, 2$ y sus primeras derivadas parciales son continuas en el espacio de fases, necesariamente debe encerrar un punto crítico. Además, si sólo encierra un punto crítico, éste no puede ser un punto silla.

Ejemplo 6

En el sistema:

$$x' = -y + x\left(1 - x^2 - y^2\right)$$
$$y' = x + y\left(1 - x^2 - y^2\right)$$

se ha demostrado que la órbita periódica $x^2 + y^2 = 1$ contiene al punto crítico (0,0).

Este resultado proporciona un criterio negativo: un sistema sin puntos críticos en una cierta región no puede tener en ella trayectorias cerradas. El siguiente resultado también da un criterio negativo:

Teorema (Bendixon):
Consideremos el sistema autónomo no lineal

$$x' = f_1(x, y)$$
$$y' = f_2(x, y)$$

tal que las funciones f_i , $i = 1, 2$ y sus primeras derivadas parciales son continuas en un dominio D simplemente conexo del espacio de fases. Si $\frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y}$ es siempre positiva o siempre negativa en una cierta región del espacio de fases, el sistema no tiene trayectorias cerradas en esa región.

Ejemplo 7

Volviendo de nuevo al sistema:

$$x' = -y + x\left(1 - x^2 - y^2\right)$$
$$y' = x + y\left(1 - x^2 - y^2\right)$$

calculamos

$$\frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y} = 2 - 4\left(x^2 + y^2\right) = 2\left(1 - 2r^2\right)$$

Es fácil probar que $\frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y}$ es positiva para $0 \le r < \frac{1}{\sqrt{2}}$; como esta región se simplemente conexa (no tiene "agujeros"), podemos afirmar que en este disco no hay trayectorias cerradas. Sin embargo, para $r > \frac{1}{\sqrt{2}}$, $\frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y} < 0$ pero no podemos aplicar el teorema porque la región en este caso no es simplemente conexa. De hecho, hemos visto antes que sí existe dicha trayectoria periódica.

En el siguiente teorema se dan las condiciones que garantizan la existencia de una trayectoria cerrada:

Teorema (Poincaré-Bendixon):

Sea R una región acotada del espacio de fases junto con su frontera y supongamos que R no contiene puntos de equilibrio del sistema. Si C es una trayectoria del sistema que está en R en cierto instante t_0 y permanece en R para todo $t \geq t_0$, entonces C o bien es una trayectoria cerrada o tiene en espiral hacia una trayectoria cerrada.

Ejemplo 8

Para aplicar el teorema de Poincaré-Bendixon al sistema:

$$x' = -y + x\left(1 - x^2 - y^2\right)$$

$$y' = x + y\left(1 - x^2 - y^2\right)$$

emplearemos una región R anular en torno al origen: $0.5 \leq r \leq 2$. En esta región el sistema no posee ningún punto crítico. Trabajando con la ecuación en polares:

$$r' = r\left(1 - r^2\right)$$

observamos que para r = 0.5, r' > 0 luego la distancia al origen crece; mientras que para r = 2, r' >0 por lo que r decrece. Así, cualquier trayectoria que cruce la frontera de R permanece en su interior. Aplicando el teorema de Poincaré-Bendixon, podemos afirmar que existe una trayectoria cerrada en R.

Pudiera parecer a partir del ejemplo mostrado que es fácil demostrar la existencia de ciclos límite en los sistemas no lineales; nada de eso. Muchas veces los teoremas de Poincaré y de Bendixon no nos llevan a ninguna conclusión y el tercer teorema no es aplicable, como se muestra en el siguiente ejemplo. En estos casos, se ha de recurrir a otras técnicas más complejas.

Ejemplo 9

La ecuación de Van der Pol $u'' - \mu\left(1 - u^2\right)u' + u = 0$, donde la constante $\mu > 0$ es positiva, describe la corriente *u* en un triodo oscilador.

Introduciendo las variables $x = u, y = u'$ transformamos la ecuación en el sistema:

$$x' = y$$

$$y' = -x + \mu\left(1 - x^2\right)y$$

Observamos que el único punto de equilibrio del sistema es el origen y el sistema lineal asociado cerca del origen es:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & \mu \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$$

Los valores propios de la matriz de coeficientes son:

$$\lambda = \frac{\tau \pm \sqrt{\tau^2 - 4\delta}}{2} = \frac{\mu \pm \sqrt{\mu^2 - 4}}{2}$$

y, por tanto, si $\mu = 2$, el origen es un nodo inestable degenerado; para $2 > \mu > 0$ los valores propios son imaginarios con parte real positiva, por lo que el origen es un punto espiral o foco inestable; por último, si $\mu > 2$, los valores propios son reales positivos y distintos, luego el origen es un nodo inestable. En todos los casos, una trayectoria que se inicie cerca del origen, se alejará de él.

Con respecto a las soluciones periódicas, del primer teorema de Poincaré se deduce que, si existen soluciones periódicas, deben rodear al origen. Por otra parte, al calcular

$$\frac{\partial f_1}{\partial x}+\frac{\partial f_2}{\partial y}=\mu\left(1-x^2\right)$$

observamos que se produce un cambio de signo en $|x|=1$. Esto significa que mientras $|x|<1$, no habrá ninguna trayectoria periódica.

Por otra parte, una región anular R en torno al origen no nos permite aplicar el teorema de Poincaré-Bendixon, ya que no podemos asegurar que las trayectorias que entren en R permanezcan en su interior.

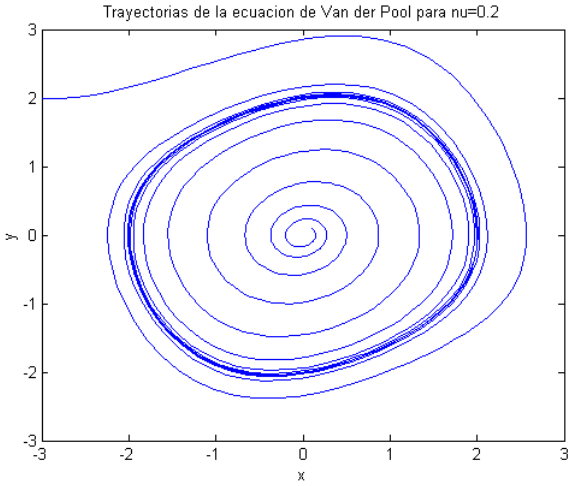
Es posible demostrar, mediante un análisis más complejo, que la ecuación de Van der Pol admite un ciclo límite único. Sin embargo, emplearemos un enfoque diferente: representaremos gráficamente las trayectorias solución obtenidas numéricamente, en función del parámetro μ.

Almacenamos el sistema correspondiente a la ecuación de Van der Pol en una función vandpol.m con un primer valor para el parámetro:

```
function z = vandpol(t,y)
nu = 0.2;
z(:,1) = y(:,2);
z(:,2) = -y(:,1)+mu*(1-y(:,1).^2)*y(:,2);
```

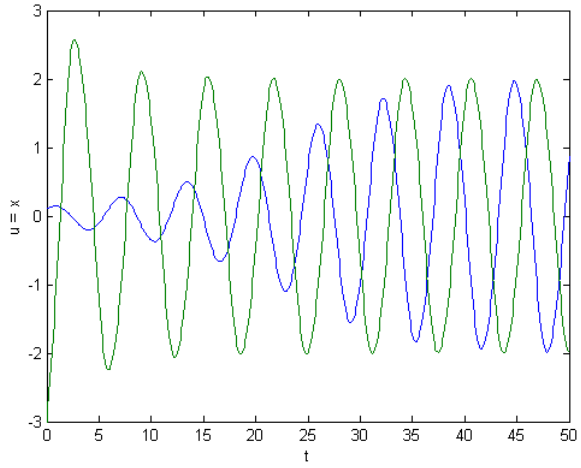
y representamos las trayectorias solución para diferentes valores iniciales:

```
[t,z]=rk4('vandpol',0,50,[0.1,0.1],500);
plot(z(:,1),z(:,2)), hold on
[t,y]=rk4('vandpol',0,50,[-3,2],500);
plot(y(:,1),y(:,2))
```



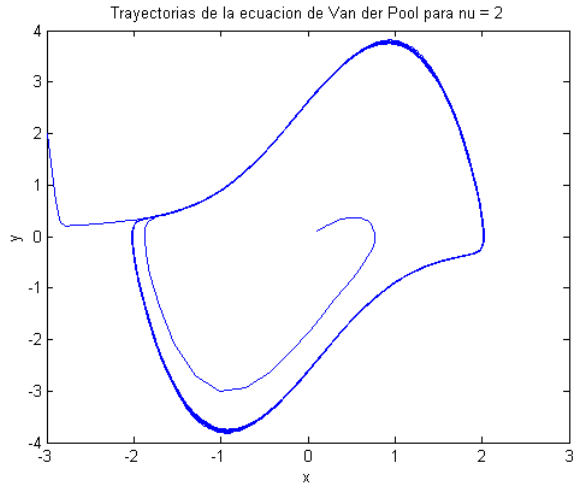
Tal y como habíamos deducido, el origen es un foco inestable ya que la trayectoria solución con estimación inicial (0.1,0.1) traza una espiral en el sentido de las agujas del reloj que se aleja del origen mientras se aproxima a la solución periódica. Al introducir un valor inicial alejado del origen, se observa que la trayectoria describe asimismo una espiral en el sentido de las agujas del reloj, que tiende a la órbita periódica. Si observamos la gráfica de u respecto de t, observamos que las dos soluciones varían su amplitud inicial para tender a un movimiento periódico estable que corresponde al ciclo límite. También se observa una diferencia de fase entre las dos soluciones al aproximarse al ciclo límite. Observando la gráfica se puede determinar la amplitud y el periodo de la órbita periódica, una vez estabilizado el movimiento.

```
plot(t,[z(:,1),y(:,1)])
```



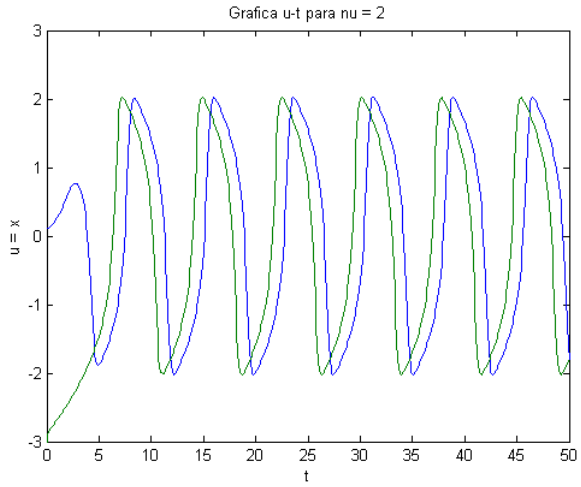
Cuando $\mu = 2$, se observa un comportamiento similar, pero el ciclo límite dista de ser una circunferencia. Las trayectorias se acercan más rápidamente al ciclo límite, perdiéndose la simetría. .

```
[t,z]=rk4('vandpol',0,50,[0.1,0.1],500);
plot(z(:,1),z(:,2)), hold on
[t,y]=rk4('vandpol',0,50,[-3,2],500);
plot(y(:,1),y(:,2))
```



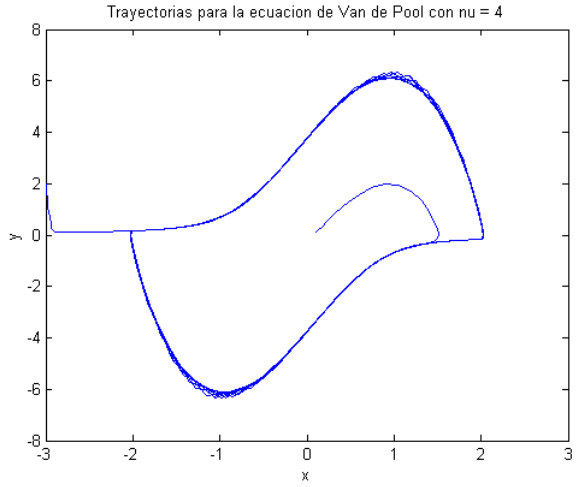
De nuevo se observa en la variación de u respecto del tiempo una diferencia en fase:

```
plot(t,[z(:,1),y(:,1)])
```

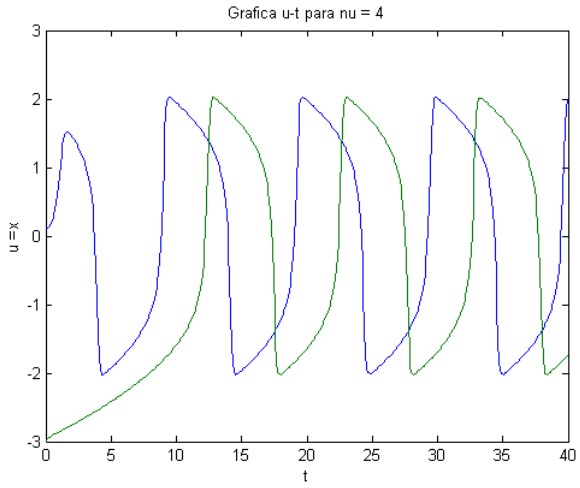


Cuando tomamos $\mu = 4$, partiendo de las mismas condiciones iniciales, obtenemos las siguientes gráficas:

```
[t,z]=rk4('vandpol',0,40,[0.1,0.1],500);
plot(z(:,1),z(:,2)), hold on
[t,y]=rk4('vandpol',0,40,[-3,2],500);
plot(y(:,1),y(:,2))
```




```
plot(t,[z(:,1),y(:,1)])
```



Se observa que el ciclo límite es más alargado en la dirección *y*; las trayectorias siguen tendiendo a él a mayor velocidad que en los casos anteriores (para intervalos de tiempo mayores, las trayectorias exteriores retroceden y se repite en movimiento en sentido opuesto). Además, se observa que la gráfica de *u* respecto al tiempo dista ahora bastante de una sinusoidal.

19.3. Problemas

19.3.1. Enunciados

1.- En los sistemas lineales siguientes, clasifica el punto crítico (0,0), calculando la traza y el determinante. Comprueba los resultados obtenidos representando algunas trayectorias solución en el espacio de fases partiendo de diferentes valores iniciales, empleando para ello el método de Runge-Kutta de orden 4.

(a) $\begin{aligned} x' &= -5x + 3y \\ y' &= 2x + 7y \end{aligned}$ (b) $\begin{aligned} x' &= -5x + 3y \\ y' &= 2x - 7y \end{aligned}$

(c) $\begin{aligned} x' &= -5x + 3y \\ y' &= -2x + 5y \end{aligned}$ (d) $\begin{aligned} x' &= -5x + 3y \\ y' &= -7x + 4y \end{aligned}$

2.- Determina las condiciones bajo las cuales el origen es un centro del sistema lineal:

$$\begin{aligned} x' &= -\mu x + y \\ y' &= -x + \mu y \end{aligned}$$

3.- Determina una condición de la constante μ tal que (0,0) sea un punto espiral (foco) estable del sistema lineal:

$$\begin{aligned} x' &= y \\ y' &= -x + \mu y \end{aligned}$$

4.- Demuestra que el origen es un punto crítico inestable del sistema lineal:

$$\begin{aligned} x' &= \mu x + y \\ y' &= -x + y \end{aligned}$$

donde μ es una constante real tal que $\mu \neq 1$. ¿Bajo qué condiciones será el origen un punto silla? ¿Y un punto espiral inestable?

5.- En los siguientes problemas no lineales clasifica, si es posible, cada punto crítico como nodo estable, foco estable nodo inestable, foco inestable o punto silla. Comprueba en cada caso los resultados obtenidos representando algunas trayectorias solución en el espacio de fases partiendo de diferentes valores iniciales empleando para ello el método de Runge-Kutta de orden 4.

(a) $\begin{aligned} x' &= 1 - 2xy \\ y' &= 2xy - y \end{aligned}$ (b) $\begin{aligned} x' &= x^2 - y^2 - 1 \\ y' &= 2y \end{aligned}$

(c) $x'' + x = \left(\frac{1}{2} - 3\left(x'\right)^2\right)x' - x^2$ (d) $x'' + 4\frac{x}{1+x^2} + 2x' = 0$

(e) $\begin{aligned} x' &= -2xy \\ y' &= y - x + xy - y^3 \end{aligned}$ (f) $\begin{aligned} x' &= x\left(1 - x^2 - 3y^2\right) \\ y' &= y\left(3 - x^2 - 3y^2\right) \end{aligned}$

6.- Demuestra gráficamente que el sistema autónomo plano

$$\begin{aligned} x' &= -x + y - x^3 \\ y' &= -x - y + y^2 \end{aligned}$$

tiene dos puntos críticos. Clasifica dichos puntos críticos y comprueba numéricamente dicha clasificación representando el espacio de fases algunas trayectorias solución del sistema (obtenidas mediante el algoritmo de Runge-Kutta), con distintas condiciones iniciales.

7.- Una interacción depredador-presa entre dos especies sucede cuando una de ellas (el depredador) se alimenta de la segunda de ellas (la presa). Por ejemplo, el lince ibérico se alimenta casi exclusivamente de conejos, mientras que éstos usan las plantas campestres como alimento. El interés de la aplicación de las matemáticas en las relaciones depredador-presa reside en determinar, no sólo el número de individuos de ambas poblaciones que establece el equilibrio en el ecosistema, sino también puede servir para poner remedio a la extinción de alguna de las especies, o de ambas.

El primer modelo depredador-presa fue el de Lotka-Volterra:

$$x' = -ax + bxy$$
$$y' = -cxy + dy$$

en el que x representa la cantidad de depredadores, y la de presas y a, b, c y d son constantes positivas.

a) Calcula los puntos críticos del modelo de Lotka-Volterra.
b) Suponiendo que a = 0.1, b = 0.002, c = 0.0025 y d = 0.2, clasifica los puntos críticos del sistema.
c) Representa el espacio de fases algunas trayectorias solución del sistema obtenidas mediante el algoritmo de Runge-Kutta, con distintas condiciones iniciales.
d) ¿Existen órbitas periódicas? Razona la respuesta.

8.- En cada uno de los siguientes problemas se expresan un sistema autónomo en coordenadas polares. Determina todos los puntos críticos, soluciones periódicas, ciclos límite así como sus características de estabilidad.

(a) $\begin{aligned} r' &= r^2\left(1-r^2\right) \\ \theta' &= 1 \end{aligned}$ (b) $\begin{aligned} r' &= r\left(1-r\right)^2 \\ \theta' &= -1 \end{aligned}$

(c) $\begin{aligned} r' &= r\left(r-1\right)\left(r-3\right) \\ \theta' &= 1 \end{aligned}$ (d) $\begin{aligned} r' &= sen\left(\pi r\right) \\ \theta' &= 1 \end{aligned}$

9.- Determina las soluciones periódicas (y sus características de estabilidad), en caso de haber, del sistema:

$$x' = y + \frac{x}{\sqrt{x^2+y^2}}\left(x^2+y^2-2\right)$$
$$y' = -x + \frac{y}{\sqrt{x^2+y^2}}\left(x^2+y^2-2\right)$$

19.3.2. Soluciones

1.- (a) Definimos la matriz del sistema lineal y calculamos su traza y determinante:

```
A=[-5 3;2 7];
t = trace(A)
d = det(A)
```

```
t =
     2
d =
   -41
```

de este modo, los valores propios son:

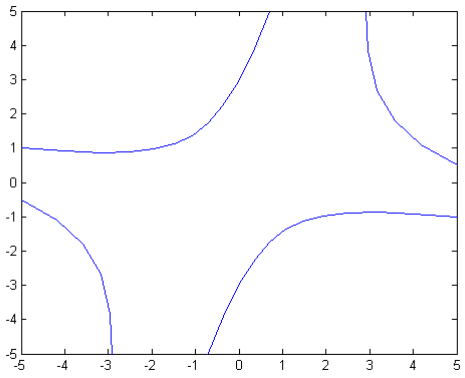
```
lambda1 = (t+sqrt(t-4*d))/2
lambda2 = (t-sqrt(t-4*d))/2

lambda1 =
    7.4420
lambda2 =
   -5.4420
```

o, calculados directamente:

```
lambda = eig(A)

lambda =
   -5.4807
    7.4807
```

Dado que los valores propios son reales y de signos opuestos, concluimos que el punto crítico es un punto de silla. Para comprobar esta afirmación, almacenaremos el sistema en un fichero de función y representaremos algunas curvas solución en el espacio de fases empleando el algoritmo de Rung-Kutta:

```
[t,z]=rk4('ej1',0,2,[5,0.5],50);
plot(z(:,1),z(:,2)), hold on

[t,y]=rk4('ej1',0,2,[-5,1],50);
plot(y(:,1),y(:,2))

[t,z]=rk4('ej1',0,2,[5,-1],50);
plot(z(:,1),z(:,2))

[t,y]=rk4('ej1',0,2,[-5,-0.5],50);
plot(y(:,1),y(:,2))
axis([-5 5 -5 5])
```



2.- Para que el origen sea un centro del sistema lineal

$$x' = -\mu x + y$$
$$y' = -x + \mu y$$

debe cumplirse que los valores propios de la matriz de coeficientes sean imaginarios puros, es decir, que $\tau - 4\delta < 0$, siendo $\tau = \mathrm{tr}(A)$ y $\delta = \det(A)$. Es decir,

$$A = \begin{pmatrix} -\mu & 1 \\ -1 & \mu \end{pmatrix};\ \ \tau = 0;\ \ \delta = -\mu^2 + 1$$

debe verificarse:

$$-4\left(1-\mu^2\right)<0 \Leftrightarrow 1-\mu^2>0 \Leftrightarrow |\mu|<1$$

5.- (a) Buscaremos, en primer lugar, los puntos críticos del sistema, aquellos en los que se verifican las ecuaciones:

$$1-2xy=0$$
$$2xy-y=0$$

Despejando *2xy* en la primera ecuación y sustituyéndolo en la segunda obtenemos que la única solución (y, por tanto, el único punto crítico) es el punto (0.5,1). Para estudiar la estabilidad de las soluciones en un entorno de este punto, trabajaremos con el sistema linealizado:

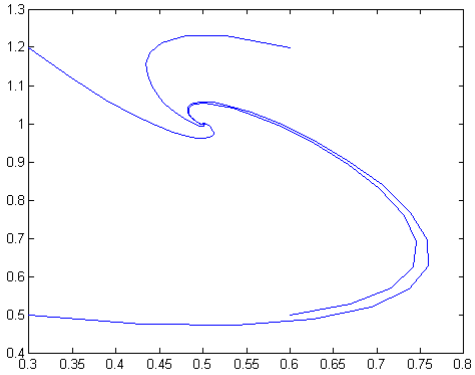
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} -2y & -2x \\ 2y & 2x-1 \end{pmatrix}\Bigg|_{\left(\frac{1}{2},1\right)} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -2 & -1 \\ 2 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Obtenemos los valores propios de la matriz de coeficientes:

```
A=[-2 -1;2 0];
   lambda=eig(A)

lambda =
  -1.0000 + 1.0000i
    -1.0000 - 1.0000i
```

que son complejos de parte real negativa. Por tanto, el punto crítico es un punto espiral estable. Para comprobarlo basta calcular algunas curvas solución mediante el algoritmo de Runge-Kutta:

```
[t,z]=rk4('ej5',0,10,[0.3,0.5],50);
plot(z(:,1),z(:,2)), hold on

[t,y]=rk4('ej5',0,10,[0.6,0.5],50);
plot(y(:,1),y(:,2))

[t,z]=rk4('ej5',0,10,[0.3,1.2],50);
plot(z(:,1),z(:,2))

[t,y]=rk4('ej5',0,10,[0.6,1.2],50);
plot(y(:,1),y(:,2))
```



8.- (d) Teniendo en cuenta la periodicidad de la función seno, los valores críticos de la variable radial son 0 y 1, únicos valores naturales en el intervalo $0 \le r < 2$. Sin

embargo, sólo r = 0 corresponde a un punto crítico. Con toda probablilidad, r = 1 corresponderá a una órbita periódica pero, ¿será ciclo límite, y en su caso, de qué tipo? Estudiemos en primer lugar la estabilidad del punto crítico a través del sistema linealizado:

$$\begin{pmatrix} r' \\ \theta' \end{pmatrix} = \begin{pmatrix} \pi\cos(\pi\cdot 0) & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r \\ \theta \end{pmatrix}$$

pero dado que el determinante de la matriz de coeficientes es nulo, no podemos clasificarlo directamente. Indirectamente, clasificaremos el punto crítico al estudiar el comportamiento de las trayectorias solución en su entorno: si observamos la ecuación diferencial $r' = \sin(\pi r)$, para $0 < r < 1$ se verifica que $r' > 0$ y por tanto la variación de la distancia al origen es positiva y cada vez mayor hasta que para r = 1 dicha variación se anula. Esto nos permite afirmar que el origen es inestable y que las trayectorias solución tiendan a la órbita periódica pero, ¿qué ocurre más allá de dicha órbita? Siguiendo el mismo razonamiento, para $1 < r < 2$ se verifica $r' < 0$. Así, la variación de la distancia al origen es cada vez menor hasta que para r = 1 ésta se anula. Por tanto, la órbita periódica será un ciclo límite de carácter atractor. Sin embargo, no nos basamos en criterios probados para realizar estas afirmaciones, son simple intuición.

Por otra parte, del primer teorema de Poincaré se deduce que, si existen soluciones periódicas, deben rodear al origen. Esto no aporta nueva información, pero no contradice las conclusiones anteriores. Por otra parte, al calcular

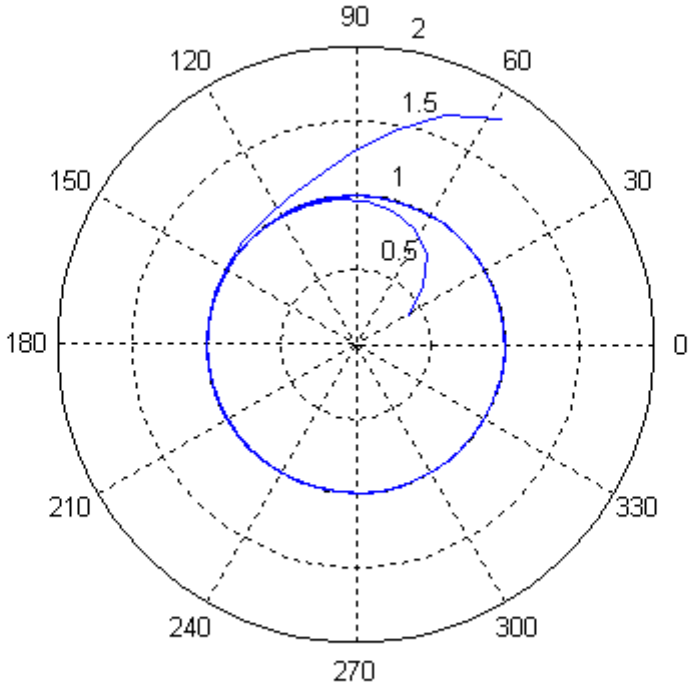
$$\frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y} = \pi\cos(\pi r)$$

observamos que se produce un cambio de signo en $|r| = \frac{1}{2}$. Esto significa que mientras $|r| < \frac{1}{2}$, no habrá ninguna trayectoria periódica.

Una región anular R tal que $2 - \varepsilon \le r \le \frac{1}{2}$ en torno al origen, con $\varepsilon > 0$ próximo a cero, nos permite aplicar el teorema de Poincaré-Bendixon, ya que hemos observado que para r = 0.5, r' > 0 luego la distancia al origen crece; mientras que para $r = 2 - \varepsilon$, r' >0 por lo que r decrece. Así, cualquier trayectoria que cruce la frontera de R permanece en su interior. Aplicando el teorema de Poincaré-Bendixon, podemos afirmar que existe una trayectoria cerrada en R.

A continuación representaremos distintas trayectorias en el espacio de fases que nos permitan observar en comportamiento deducido:

```
[t,z]=rk4('ej8',0,10,[1.8,1],50);
polar(z(:,2),z(:,1)), hold on
[t,y]=rk4('ej8',0,10,[0.4,0.5],50);
polar(y(:,2),y(:,1))
```


90
2
60
120
1.5
30
150
1
0.5
180
0
210
330
240
300
270

Ecuación diferencial no lineal

PENDULO

```
>> [t,y] = rungekut('pendulo',0:0.01:10,[0,0])
```

solucion, todo ceros, el pendulo ha empezado parado, solucion estacionaria

```
>> [t,y] = rungekut('pendulo',0:0.01:10,[0,1]);
>> plot(t,y)
```

ondas aparentemente sinusoidales: quiere decir que el termino no lineal no influye mucho, porque el seño es propio de sistema lineal con coeficientes constantes.

```
>> [t,y] = rungekut('pendulo',0:0.01:10,[0,5]);
>> plot(t,y)
```

la velocidad ya no se curva como el seno, es mas picuda

```
>> [t,y] = rungekut('pendulo',0:0.01:10,[0,6.5]);
>> plot(t,y)
```

el pendulo esta dando vueltas!

Plano de fases
---

en lugar de usar diagrama de tiempos y angulos, en vez de dibujar las dos magnitudes respecto al tiempo, represento una magnitud respecto a la otra. Ademas dibujaremos un campo con los vectores de la derivada.

```
>> planofases('pendulo',-2*pi,2*pi,-8,8);
```

Se pueden ver los puntos estacionarios (ej. 0, 2*pi - derivada cero), los puntos silla (ej.pi,-2*pi)(direcciones que se alejan y otra que se acercan)

```
>> [t,y] = rungekut('pendulo',0:0.01:10,[0,3]);
>> hold on
>> plot(y(:,1),y(:,2))
>> [t,y] = rungekut('pendulo',0:0.01:10,[0,6]);
>> plot(y(:,1),y(:,2))
>> [t,y] = rungekut('pendulo',0:0.01:10,[0,6.5]);
>> plot(y(:,1),y(:,2))

>> axis([-2*pi,2*pi,-8,8]) % lo centra en lo interesante
```

Se puede comparar con lo que ocurria en la otra representacion:

con velocidad 3 -> sinusoidal = elipse en 2D
con velocidad 6 -> sinusoidal picuda = elipse picuda en 2D
con velocidad 6.5 -> se va

Vamos a ver si conseguimos darle el impulso exacto para que se quede en posicion pi (hacia arriba)

```
[t,y] = rungekut('pendulo',0:0.01:10,[0,6.23]); % no llega
>> plot(y(:,1),y(:,2))
[t,y] = rungekut('pendulo',0:0.01:10,[0,6.28]); % se pasa
>> plot(y(:,1),y(:,2))
[t,y] = rungekut('pendulo',0:0.01:10,[0,6.26]); % se pasa
>> plot(y(:,1),y(:,2))
```

Cambiando el Modelo

```
>> k=0.2;
>> planofases('pendulo',-2*pi,2*pi,-8,8);
>> [t,y] = rungekut('pendulo',0:0.01:10,[0,2*pi]);
>> hold on
>> plot(y(:,1),y(:,2),'r')
>> [t,y] = rungekut('pendulo',0:0.01:10,[0,7]);
>> plot(y(:,1),y(:,2),'r')
```

% la 7 primero da una vuelta y luego empieza a oscilar.

Comentarios.

El modelo del pendulo es casi lineal (seno de x =- x)

con otros modelos salen cosas como ciclos limites (ver transpa)

Con ED de mas de 2 varibles salen nuevos fenomenos (atractores extraños)
ej lorenz