

MATLAB: un lenguaje para la computación científica

F.Vadillo

Versión 2, Diciembre del 2002

MATLAB (MAThenmatical LABoratory) es un lenguaje para la computación científica cuyos datos fundamentales son vectores y matrices. Se distingue de otros lenguajes como Fortran o C porque opera en un nivel muy alto incluyendo cientos de operaciones matemáticas en un sólo comando.

Desde las primeras versiones de Cleve Moler a finales de los setenta, *MATLAB* se ha convertido en una poderosa herramienta para investigar y resolver problemas prácticos sobre todo con el desalloro de los Toolbox que son programas implementados en *MATLAB* para resolver problemas tipo. Este crecimiento ha motivado que actualmente en muchas facultades y escuelas técnicas cuando se cursan asignaturas de Análisis Numérico, las prácticas se realizan en el lenguaje *MATLAB*, buena prueba de ello es el creciente número de texto que lo hacen de esta manera de los que hemos seleccionado unos pocos en la bibliografía al final de este documento.

Personalidades de la autoridad de L.N.Trefethen llegan a afirmar en la introducción de [10] que con el *MATLAB* ha nacido una nueva era en la computación científica porque puede hacer una cantidad astronómica de cálculos con muy pocas líneas de programa y en unos pocos segundos de tiempo. Charles F. Van Loan en [12] afirma que *MATLAB* ha cambiado la forma de computar e investigar en las ciencias de la computación.

Este documento no pretende ser una guía de *MATLAB* porque muy posiblemente en poco mejoráramos lo que otros autores han hecho con más dedicación y mayores conocimientos. Recomendamos la excelente guía de D.J.Higham y N.J.Higham con el gusto de reconocer un trabajo bien hecho y difícilmente siquiera igualable, que además ha servido en la forma de exponer los argumentos e ilustrar con los ejemplos. Estas **notas de clase** sólo pretenden servir para tener un primer contacto con el lenguaje *MATLAB* y ayudar en las clases prácticas con ejemplos y ejercicios que deben complementar y ampliar los argumentos aquí expuestos.

Evidentemente es mucho más lo omitido que lo comentado, en unas notas de clase no puede ser de otra forma por obvias razones del tiempo disponible, pero pensamos que la afición, la bibliografía y una correcta utilización del comando 'help' pueden hacer el resto. Recordando aquella frase tan bella del poeta que dijo que **se hace camino al andar**, aquí amigo lector estas dando tus primeros pasos en una nueva materia con los inevitable traspiees que pueden motivar caídas pero que nunca te deben llevar al desalientos.

Contents

1	Introducción	3
1.1	Tutoria	4
1.2	Números de Fibonacci	5
1.3	Iteración de Collatz	5
1.4	Conjunto de Mandelbrot	6
1.5	Ecuaciones de Lorenz	6
1.6	Parabolóide hiperbólico	7
2	Control del flujo y M-files	8
2.1	Relaciones y operaciones lógicas	8
2.2	Control del flujo	8
2.3	M-files	9
3	Gráficos	10
3.1	Gráficos bidimensionales	10
3.2	Gráficos tridimensionales	11
4	Algebra Lineal	13
4.1	Construcción de matrices.	13
4.2	Sistemas de ecuaciones lineales.	14
4.2.1	Sistemas cuadrados.	14
4.2.2	Sistemas sobredeterminados.	14
4.2.3	Sistemas subdeterminados.	14
4.3	Factorizaciones de matrices.	14
4.4	Matrices dispersas	15
4.5	Algebra lineal de matrices dispersas.	15
5	Métodos Numéricos	16
5.1	Polinomios	16
5.1.1	Evaluación de polinomios	16
5.1.2	Calculo de raices	16
5.1.3	Interpolación	16
5.2	Ecuaciones no lineales y mínimos	17
5.3	Cuadratura numérica	17
5.4	Ecuaciones diferenciales	17
5.4.1	Un ejemplo con ode45	17
5.4.2	Un problema de depredador presa	19
5.4.3	Problemas Stiff	20
6	El manipulador simbólico	22

Chapter 1

Introducción

MATLAB es un sistema interactivo, cuando escriba despues del prompt `>>` el nombre del comando y presione `'enter'`, el sistema lo ejecuta y si no le ha asignado un nombre concreto lo guardará en una nueva variable que llamará `ans`.

MATLAB tiene tres característica muy importantes que lo diferencian de otros lenguaje:

1. Las variables no necesitan ser previamente declaradas.
2. Contiene una gran colección de funciones matemáticas con un número de argumentos no necesariamente el mismo.
3. El tipo de dato fundamental son vectores y matrices de números complejos almacenados en aritmética flotante de doble precisión.

Comenzamos este breve recorrido por el *MATLAB* con algunos comentarios que siempre se deben tener en cuenta

1. Los caracteres mayúsculas y minúsculas no son equivalentes.
2. Un punto y coma al final de un comando suprime la salida por pantalla.
3. Los `()` y `[]` no son intercambiables.
4. La flecha atrás recupera comandos anteriores.
5. Tecleando `'help topic'` se conocen el formato del comando.
6. Para salir de *MATLAB* se teclea `'exit'` o `'quit'`

Los siguientes comandos son especialmente útiles en todo este curso:

bench : hace un test de velocidad de su computador.

clear : borrar las variables del workspace (variables en memoria).

computer : indica el tipo de computador donde está corriendo *MATLAB*.

demo : accede a una colección de demostraciones.

dir : lista los programas en el directorio actual.

lookfor topic : buscara las funciones relacionadas con lo tópico.

Nan : indica alguna indeterminación en las operaciones.

path : indica los directorios accesibles en ese momento.

ver : le indica la versión *MATLAB* que se está utilizando.

1.1 Tutoria

En su primer contacto con el *MATLAB* le invitamos a teclear y observar los resultados de la siguiente secuencia de comandos:

```
%TUTORIA      % indica una linea de comentario
a=[1 2 3]
c=[4;5;6]
a*c
dot(a,c)
A=c*a
size(A)
openvar('A')
b=a.^2
exp(a)
sqrt(a)
format log
sqrt(a)
format
sum(a), sum(c)
pi
B=[-3 0 1; 2 5 -7;-1 4 8]
size(B)
%              Resuelve un sistema lineal
x=B\c
norm(B*x-c)
%              Calcula autovalores y autovecotes de un matriz
e=eig(B)
[V,D]=eig(B)
%              Otras formas de construir vectores y matrices
v=1:6
w=2:3:10, y=1:-.25:0
C=[A,[8;9;10]]
D=[B;a]
%              Acceso a elementos de una matriz
C(2,3)
C(2:3,1:2)
%              Otras matrices
I3=eye(3), Y=zeros(3,5), Z=ones(2)
rand('state',20), randn('state',15)
F=rand(3), G=randn(1,5)
%              Acceso al workspace
who
whos
clear
%              Suma de fracciones continuas
g=2;
for k=1:10, g=1+1/g; end
g
%              Dibujo de una curva
t=0:.005:1;
z=exp(10*t.*(t-1)).*sin(12*pi*t);
plot(t,z)
```

Seguidamente se mostraran algunos programas en *MATLAB* que resuelven algunos ejercicios clásicos de matemáticas.

1.2 Números de Fibonacci

En el primer ejemplo se construye la sucesión de números de Fibonacci $\{x_n\}$ generada por la iteración:

$$x_{n+1} = x_n \pm x_{n-1}, \quad n \geq 2$$

donde $x_1 = 1, x_2 = 2$ y el signo mas o menos se elige con la misma probabilidad. El análisis de Viswanath demuestra que para n suficientemente grande $|x_n| \sim c^n$ donde $c = 1.13198824\dots$

```
%RFIB                               Sucesion de Fibonacci.

rand('state',100)
m = 1000;                            % Numero de iteraciones.

x = [1 2];                          % datos iniciales.
for n = 2:m-1
    x(n+1) = x(n) + sign(rand-0.5)*x(n-1);
end

semilogy(1:m,abs(x))
c = 1.13198824;                      % Constante de Viswanath.
hold on
semilogy(1:m,c.^(1:m))
hold off
```

1.3 Iteración de Collatz

El siguiente ejemplo analiza la iteración de Collatz: dado un entero positivo x_1 se construye la sucesión $x_{n+1} = f(x_n)$ donde

$$f(x) = \begin{cases} 3x+1 & \text{si } x \text{ es impar,} \\ x/2 & \text{si } x \text{ es par.} \end{cases}$$

Se trata de comprobar que siempre se acaba en el ciclo 4,2 y 1.

```
%COLLATZ                            iteracion de Collatz .

n = input('Enter an integer bigger than 2: ');
narray = n;

count = 1;
while n ~= 1
    if rem(n,2) == 1 % Resto modulo 2.
        n = 3*n+1;
    else
        n = n/2;
    end
    count = count + 1;
    narray(count) = n; % guardamos la iteracion.
end

plot(narray,'*-') % dibujamos un * y unimos con linea continua.
title(['Collatz iteration starting at ' int2str(narray(1))], 'FontSize',16)
```

1.4 Conjunto de Mandelbrot

El conjunto de Mandelbrot puede dibujarse en una pocas líneas de programa *MATLAB*. Recuerde que dicho conjunto son los valores de c en el plano complejo para los que la iteración $z_{n+1} = z_n^2 + c$ permanece acotada cuando $z_1 = c$.

```
%MANDEL      conjunto de Mandelbrot .

h = waitbar(0,'Computing...');
x = linspace(-2.1,0.6,301);
y = linspace(-1.1,1.1,301);
[X,Y] = meshgrid(x,y);
C = complex(X,Y);

Z_max = 1e6; it_max = 50;
Z = C;
for k = 1:it_max
    Z = Z.^2 + C;
    waitbar(k/it_max)
end
close(h)

contourf(x,y,abs(Z)<Z_max,1)
title('Mandelbrot Set','FontSize',16)
```

1.5 Ecuaciones de Lorenz

En el siguiente ejemplo se resuelve el sistema de ecuaciones diferenciales

$$\begin{aligned}\frac{d}{dx}y_1(t) &= 10(y_2(t) - y_1(t)), \\ \frac{d}{dx}y_2(t) &= 28y_1(t) - y_2(t) - y_1(t)y_3(t), \\ \frac{d}{dx}y_3(t) &= y_1(t)y_2(t) - 8y_3(t)/3,\end{aligned}$$

que es un ejemplo de la familia de ecuaciones de Lorenz que construye su famosísimo atractor. Con el dato inicial $(0, 1, 0)$ el siguiente programa *MATLAB* construye dicho atracto en unos pocos segundos.

```
%LRUN      sistema de Lorenz.

tspan = [0 50]; % resuelto para 0 <= t <= 50.
yzero = [0;1;0]; % condiciones iniciales.
[t,y] = ode45(@lorenzde,tspan,yzero);
plot(y(:,1),y(:,3)) % (y_1,y_3) plano de fase.
xlabel('y_1','FontSize',14)
ylabel('y_3','FontSize',14,'Rotation',0,'HorizontalAlignment','right')
title('Lorenz equations','FontSize',16)

function yprime = lorenzde(t,y)
%LORENZDE    ecuaciones de Lorenz .
%           YPRIME = LORENZDE(T,Y).
```

```
yprime = [10*(y(2)-y(1))  
          28*y(1)-y(2)-y(1)*y(3)  
          y(1)*y(2)-8*y(3)/3];
```

1.6 Parabolóide hiperbólico

En el último ejemplo se dibujan el parabolóide hiperbólico $x^2 + y^2 = 1 + z^2$

```
%SWEEP          dibujamos el paraboloides hiperbolico.
```

```
N = 10;          % numero de lados.
```

```
z = linspace(-5,5,N)';  
radius = sqrt(1+z.^2);  
theta = 2*pi*linspace(0,1,N);  
X = radius*cos(theta);  
Y = radius*sin(theta);  
Z = z(:,ones(1,N));
```

```
surf(X,Y,Z)  
axis equal
```


Chapter 2

Control del flujo y M-files

2.1 Relaciones y operaciones lógicas

En *MATLAB* existe un tipo de dato llamado lógico que son también matrices de números pero que deben manipularse de distinta manera y tienen otras utilidades, por ejemplo los vectores lógicos pueden usarse como subíndices de matrices. La forma más sencilla de construirlo estos datos lógicos es aplicando la función `'logical'`.

La comparación entre dos escalares también produce un resultado de tipo lógico que vale 1 si es cierta y un 0 si es falsa. Las operaciones de relación en *MATLAB* son las siguientes:

<code>==</code>	igualdad
<code>~=</code>	desigualdad
<code><</code>	menor que
<code>></code>	mayor que
<code><=</code>	menor o igual que
<code>>=</code>	mayor o igual que

la igualdad simple `=` asigna valores a variables.

La comparación entre matrices obtiene una matriz también lógica. Para obtener una lista de las funciones lógicas teclee en pantalla `doc is`.

Las operaciones con las variables lógicas son las siguientes:

<code>&</code>	and
<code> </code>	or
<code>~</code>	negación
<code>xor(x)</code>	intercambia ceros por unos y viceversa
<code>all(x)</code>	cierta cuando todos los elementos del vector son unos
<code>any(x)</code>	cierta si algún elemento del vector es no nulo

por ejemplo, `all(all(A==B))` es una forma de hacer un test para saber si $A = B$, y `any(any(A==B))` permite conocer si A y B tienen algún elemento igual.

2.2 Control del flujo

MATLAB tiene cuatro controles de flujo `'if'`, `'for'`, `'while'` y `'switch'`. Puede accederse a sus formatos por la función `help topic`.

2.3 M-files

Son los equivalentes de programas, funciones, subrutinas y procedimientos de otros lenguajes.

Para escribir M-files *MATLAB* tiene su propio editor que aparece cuando se teclea el comando 'edit'. Todos los ficheros llevan la extensión .m y existen dos tipos de M-files:

script : son conjuntos de comandos que operan sobre las variables del workspace.

funciones : contienen el comando 'function' con variables de entrada y variables de salida, aunque no es necesario requerir todos los argumentos a la vuelta. Las variables son locales al menos que se declaren como globales, y el nombre del M-file debe coincidir con el nombre de la función.

En el siguiente ejemplo se describe una función que calcula la derivada en un punto de la función logística $f(x) = x(1 - ax)$:

```
function [f,fprime]=flogist(x,a)
%FLOGIST calcula la funcion logistica y su derivada

f=x.*(1-a*x);
fprime=1-2*a*x
```

En el siguiente ejemplo de M-file se ha implementado el método de Newton para calcular la raíz cuadrada de un número $a > 0$.

```
function [x,iter] = sqrtn(a,tol)
%SQRTN calcula la raiz cuadrada de numero a por el
% metodo de Newton.
% se supone que a >= 0.
% TOL es la tolerancia para la convergencia (por defecto EPS).
% ITER es el numero de iteraciones para la convergencia.

if nargin < 2, tol = eps; end

x = a;
iter = 0;
xdiff = inf;
fprintf(' k          x_k          rel. change\n')

while xdiff > tol
    iter = iter + 1;
    xold = x;
    x = (x + a/x)/2;
    xdiff = abs(x-xold)/abs(x);
    fprintf('%2.0f: %20.16e %9.2e\n', iter, x, xdiff)
    if iter > 50
        error('Not converged after 50 iterations.')
    end
end
```

Chapter 3

Gráficos

MATLAB tiene poderosas y versátiles herramientas de representación gráfica de distintos tipos de datos. En este capítulo se hace un breve recorrido por las más utilizados.

3.1 Gráficos bidimensionales

La gráfica en dos dimensiones más sencilla une los puntos cuyas coordenadas son los vectores \mathbf{x} , \mathbf{y} con el comando `plot(x,y)`. Cuando se ejecuta este comando, *MATLAB* abre una ventana gráfica donde muestra la figura. El comando `'plot'` tiene unas opciones que pueden consultarse con el `'help'`, su formato general es `plot(x,y,'cadena')` donde la cadena tiene tres elementos que indican el color, la marca en los puntos y el tipo de línea. Si \mathbf{X} , \mathbf{Y} fueran matrices $m \times n$ entonces `plot(X,Y)` traza las gráficas por columnas. Para un vector \mathbf{y} de números complejos `plot(y)=plot(Re(y),Im(y))` y si fuera real dibujaría \mathbf{y} contra sus índices.

Otros comandos usados con frecuencia son:

`'loglog'` : igual que el `'plot'` pero en escala logarítmica.

`'semilogx'`, `'semilogy'` : dibuja en la escala semilogarítmica correspondiente.

`'axis'` : controla los ejes.

`'grid on'` : cuadricula la superficie del gráfico.

`'xlabel'`, `'ylabel'` : etiqueta los ejes.

`'title'` : coloca un título en el gráfico.

`'legend'` : crea un legenda que identifica los gráficos.

`'text'`, `'gtext'` : permiten escribir un texto en cualquier lugar de la gráfica.

En el siguiente ejemplo se dibujan los cuatro primeros polinomios de Legendre con una leyenda que identifica a cada uno de ellos:

```
%LEGENDRE14
```

```
x=-1:.01:1;
p1=x;
p2=(3/2)*x.^2-1/2;
p3=(5/2)*x.^3-(3/2)*x;
p4=(35/8)*x.^4-(15/4)*x.^2+3/8;
```

```

plot(x,p1,'r:',x,p2,'g--',x,p3,'b-',x,p4,'m-')
box off

legend('\itn=1','n=2','n=3','n=4',4)
xlabel('x','FontSize',12,'FontAngle','italic')
ylabel('P_n','FontSize',12,'FontAngle','italic')
title('Polinomios de Legendre','FontSize',14)
text(-.6,.7,'(n+1)P_{n+1}(x)=(2n+1)xP_n(x)-n P_{n-1}(x)',...
      'FontSize',12,'FontAngle','italic')

```

El comando 'subplot' permite dibujar varias gráficas en la misma ventana y 'fplot' dibuja funciones matemáticas, el siguiente ejemplo muestra como funcionan ambas instrucciones:

%PLOTMUL ejemplo de dibujo multiple

```

subplot(221), fplot('exp(sqrt(x)*sin(12*x))',[0,2*pi])
subplot(222), fplot('sin(round(x))',[0,10], '--')
subplot(223), fplot('cos(30*x)/x',[0.01 1 -15 20], '-')
subplot(224), fplot('[sin(x),cos(2*x),1/(1+x)]',[0 5*pi -1.5 1.5])

```

3.2 Gráficos tridimensionales

El comando 'plot3' es el equivalente al 'plot' en el espacio de tres dimensiones. `plot3(x,y,z)` dibuja una curva que une los puntos de cordenadas cartesianas en los vectores x, y, z . A continuación mostramos un ejemplo muy sencillo:

%DIBUJO3D ejemplo de utilizacion del comando plot3

```

t=-5:.005:5;
x=(1+t.^2).*sin(20*t);
y=(1+t.^2).*cos(20*t);
z=t;

plot3(x,y,z)
grid on
xlabel('x(t)'), ylabel('y(t)'), zlabel('z(t)')
title('\it{ejemplo plot3 }','FontSize',14)

```

Para dibujar superficie existen varias opciones aunque las más utilizadas son:

'mesh' 'meshgrid' : dibuja superficie de la manera que puede verse en el ejemplo superficie1 .

'surf', 'surfc' : como indica el ejemplo superficie2.

'waterfall' : dibuja ondas viajeras.

%SUPERFICIE1 ejemplo de trazado de superficies

```

x=1:.1:pi; y=x;
[X,Y]=meshgrid(x,y);
Z=sin(Y.^2+X)-cos(Y-X.^2);

subplot(221)

```

```
mesh(Z)

subplot(222)
meshc(Z)

subplot(223)
mesh(x,y,Z)
axis([0 pi 0 pi -5 5])

subplot(224)
mesh(Z)
hidden off

%SUPERFICIE2 otro ejemplo de superficies

Z=membrane; FS='FontSize';
subplot(221), surf(Z), title('\bf{surf}',FS,14)
subplot(222), surfc(Z), title('\bf{surfc}',FS,14),colorbar
subplot(223), surf(Z), shading flat, title('\bf{shading flat}',FS,14)
subplot(224), waterfall(Z), title('\bf{waterfall}',FS,14)
```

La posición del observador en la gráficas se define en el comando `view(a,b)` donde `a`, `b` definen rotaciones sobre el eje `z`.

Las graficas creadas en *MATLAB* evidentemente pueden enviarse directamente a una impresora o también puede almacenarse en un fichero PostCript con la linea de comando:

```
print -deps2 nombre.eps
```

Evidentemente este fichero `*.eps` despues puede enviarse a una impresora o utilizarse en otros ficheros de procesadores de textos para obtener gráficos.

Chapter 4

Algebra Lineal

MATLAB fue originariamente diseñado para resolver problemas computacionales de algebra lineal, por esto no sorprende la gran versatilidad de funciones y aplicaciones en este campo. Para recordar o conocer los temas esenciales en esta disciplina del Análisis Numérico se recomiendan la referencia [12] y [11].

Con caracter general, las funciones trabajan indistintamente para matrices de números reales o matrices de valores complejas.

4.1 Construcción de matrices.

Como ya se comento en la introducción, el tipo de dato fundamental en *MATLAB* son los vectores y las matrices por lo que es fundamental aprender a generar y manipular con agilidad estos tipos de datos.

Además de construir matrices enumerando cada uno de sus elementos, recuerde el capítulo de la tutoría, existen funciones que construyen unas matrices concretas, a continuación se enumeran algunas de ellas:

- `zeros(m,n)`: construye una matriz de ceros $m \times n$.
- `ones(m,n)`: construye matrices de unos.
- `eye(m,n)`: matriz identidad.
- `rand(m,n)`: matriz de números aleatorios uniformemente distribuidos.
- `randn(n,m)`: matriz de números aleatorios normalmente distribuidos.
- `linspace(m:n:p)`: vector de elementos linealmente espaciados.
- `lgspase(m:n:p)`: vector de elementos logaritmicamente espaciados.

en general los parametros m y n indican las dimensiones.

La función '`gallery`' permite acceder a una gran colección de matrices test creadas por *N.J.Higham*.

El determinante de una matriz se calcula con la función `det(A)` y si se trata de una matriz inversible, la inversa es `inv(A)`, aunque en computación científica raramente se necesita calcular inversas y el test del determiante es poco aconsejable.

También pueden manipularse matrices por bloques: '`blkdiag`' crea una matriz diagonal por bloques y '`repmat(A,m,n)`' crea una matriz $m \times n$ donde A se repite en cada bloque.

El producto de Kronecker de dos matrices A y B es `kron(A,B)`

También se puede redimensionar matrices, `reshape(A,m,n)` ordena los elementos de la matriz A en m filas y n columnas, o se puede conocer sus elementos diagonales con la función `diag(A)` o las matrices triangulares inferiores y superiores con las funciones `tril(A)` y `triu(A)`.

4.2 Sistemas de ecuaciones lineales.

La norma de una matriz mide su tamaño y la función `norm(A,p)` donde $p=1,2,\text{inf}$ calcula la norma p de la matriz A .

Para medir la sensibilidad de la solución del sistema $A \cdot x = b$ respecto de perturbaciones de A o b , se define el número de condición de la matriz $\kappa(A) = \|A\| \cdot \|A^{-1}\|$, *MATLAB* calcula estas cantidades con la función `cond(A,p)` incluso cuando se encuentran dificultades existen otras funciones más sofisticadas.

La herramienta fundamental para resolver un sistema de ecuaciones lineales $A \cdot x = b$ es el operador backslash `\`, aunque este operador también resuelve sistema más general $A \cdot X = B$ donde B es una matriz con m columnas, en este caso se resuelven los m sistemas para cada columna de B .

Las tres situaciones posibles son los sistemas cuadrados (igual número de ecuaciones que de incógnitas), sistemas sobredeterminados (con más ecuaciones que incógnitas) y finalmente los sistemas con más indeterminadas que ecuaciones.

4.2.1 Sistemas cuadrados.

Si A es una matriz no singular $n \times n$, simplemente `A\b` proporciona la solución del sistema $A \cdot x = b$. *MATLAB* reconoce dos formas especiales de sistemas cuadrados:

- Matrices triangulares o permutaciones de matrices triangulares.
- Matrices definidas positivas.

4.2.2 Sistemas sobredeterminados.

Si la matriz del sistema es $m \times n$ con $m > n$ se tiene un sistema que puede tener o no solución. En general, no existirá x resolviendo exactamente el sistema. Cuando el rango de la matriz A es n , `A\b` da una solución de mínimos cuadrados, es decir, x minimiza el residuo $A \cdot x - b$ en la norma euclídea. Si el rango de A es menor que n se calcula una solución básica.

Otra forma de obtener la misma solución es calculando la pseudoinversa de la matriz A , entonces `x=pinv(A)*b`. Si buscamos soluciones con componentes no negativas `x=lsqnonneg(A,b)`.

4.2.3 Sistemas subdeterminados.

Cuando se tienen más incógnitas que ecuaciones ($m < n$), el sistema o no tiene solución o existen infinitas de ellas. Ahora `A\b` proporciona una solución con a lo más k elementos no nulos, donde k es el rango de A .

4.3 Factorizaciones de matrices.

Las funciones para calcular las factorizaciones clásicas del Análisis Numérico Matricial son las siguientes:

- `[L,U,P]=lu(A)`: calcula la factorización LU tal que $P \cdot A = L \cdot U$.
- `R=chol(A)`: obtiene la factorización de Cholesky de una matriz definida positiva.
- `[Q,R,P]=qr(A)`: calcula la factorización QR tal que $A \cdot P = Q \cdot R$ donde P es una matriz de permutaciones para que los elementos diagonales de R sean no crecientes en módulo.

- `[U,S,V]=svd(A)`: computa la descomposición en valores singulares de la matriz A , es decir, construye matrices unitarias U y V y una matriz diagonal S tal que $A = U \cdot S \cdot V^*$.

4.4 Matrices dispersas

. Una matriz dispersa es una matriz con un alto porcentaje de ceros. Son matrices muy frecuentes en los sistemas lineales que resultan de discretizar ecuaciones diferenciales usando diferencias finitas, elementos finitos o también métodos espectrales. Como la dimensión del sistema puede ser muy elevada resulta imprescindible aprovechar memoria y reducir el número de operaciones.

MATLAB tiene un tipo de dato denominado '**sparse**' que sólo almacena los elementos no nulos de una matriz junto con su índices de filas y columnas. Son tipos de datos especiales para los que los algoritmos son diferentes del caso de las matrices llenas, aunque en ocasiones reciban el mismo nombre.

Una manera habitual de construir matrices dispersas es la función **A=sparse(i,j,x)** donde **x** es un vector de valores que se sitúan según los vectores de índices **i** y **j**. Para convertir una matriz dispersa **A** en otra **B** pero llena, basta teclear **B=full(A)**, y al revés **A=sparse(B)**. El número de elementos no nulos de una matriz lo da la función '**nnz**'.

La matriz dispersa identidad es **speye(n)** o **speye(m,n)**, mientras que **spones(A)** coloca unos en donde **A** no es cero. Las funciones '**spdiag**', '**sprand**' y '**sprandn**' son las versiones estudiadas anteriormente pero para matrices dispersas.

4.5 Algebra lineal de matrices dispersas.

MATLAB tiene funciones que permiten resolver sistemas, calcular autovalores y valores singulares de matrices dispersas, y aunque el nombre en ocasiones coincida con las matrices llenas, el algoritmo implementado no es el mismo.

El backslash $x = A \backslash b$ resuelve el sistema lineal $A \cdot x = b$, y la función '**condest**' calcula el número de condición de la matriz.

Las funciones '**lu**', '**chol**', '**eig**' ... actúan como en el caso de las matrices llenas, y la función **spy(A)** dibuja un esquema indicando donde se encuentran los elementos no nulos de la matriz.

En matrices dispersas un tema muy importante es su posible reordenamiento para que los algoritmos sean más eficaces. *MATLAB* dispone de funciones que implementan los algoritmos habituales como el de Cuthill-McKee llamado '**symrcm**'. Este tema que es muy importante supera el alcance de estas notas y por ello remitimos al manual general.

Chapter 5

Métodos Numéricos

En este capítulo se describen las funciones *MATLAB* relacionadas con las técnicas numéricas clásicas de manipulación de polinomios, resolución de ecuaciones no lineales, cuadratura numérica y resolución de ecuaciones diferenciales.

5.1 Polinomios

MATLAB representa el polinomio

$$p(x) = p_1x^n + p_2x^{n-1} + \cdots + p_nx + p_{n+1},$$

por un vector fila $p = [p(1), \dots, p(n+1)]$ formado por sus coeficientes. Los polinomios pueden multiplicarse y dividirse con `'conv'` y `'deconv'` respectivamente.

5.1.1 Evaluación de polinomios

La función `polyval(p,x)` calcula el valor del polinomios de coeficientes **p** en el punto **x**. Si **x** fuera una matriz **X**, entonces resultaría la matriz $Y = \text{polyval}(p, X)$.

5.1.2 Cálculo de raíces

Las raíces del polinomio de coeficientes **p** son $z = \text{root}(p)$ y la función $q = \text{poly}(z)$ hace la operación contraria, construye un polinomio de coeficientes **q** con las raíces **z** y coeficiente director uno. La función `'poly'` admite un argumento matricial `poly(A)` que es el polinomio característico de la matriz **A**.

5.1.3 Interpolación

Dado el conjunto de datos $\{x_i, y_i\}_{i=1}^m$ con las abscisas x_i distintas, el comando $p = \text{polyfit}(x, y, n)$ construye el polinomio mínimo cuadrático de grado $\leq n$. Si $n \geq m$ entonces **p** es el polinomio interpolador.

Para conseguir una interpolación más exacta se utiliza el comando $yy = \text{spline}(x, y, xx)$ que devuelve el spline cúbico con los datos **x,y** evaluado en los puntos **xx**. La función `interp1` admite varias opciones que pueden consultarse con el comando `'help'`.

MATLAB también tiene funciones para interpolar en dos dimensiones `'griddata'` e `'interp2'`, y para dimensión tres `'interp3'`.

5.2 Ecuaciones no lineales y mínimos

MATLAB tienen rutinas para hallar ceros de una función de una variable '**fzero**' y para minimizar funciones de una variable '**fminbnd**' o de n variable '**fminsearch**'.

El comando **fzero(f,x0)** halla un cero de la función **f** cercano a **x0**, y **fzero(f,[x1,x2])** un cero en el intervalo indicado. Los parámetros para la convergencia se controlan con '**optimset**'.

Para hallar un mínimo local en el intervalo **[x1,x2]** se teclea **fminbnd(f,x1,x2)**.

Los problemas de optimización más complicados deben resolver con el paquete '**Optimization Toolbox**' que implementa técnicas más sofisticadas.

5.3 Cuadratura numérica

Las dos funciones más importantes para estimas integrales propias $\int_a^b f(x)dx$ son '**quad**' y '**quadl**'. Las dos funciones utilizan fórmulas de cuadratura compuesta y adaptativas, '**quad**' está basada en la regla de Simpson y '**quadl**' emplea la fórmula de Gauss-Lobatto de cuatro puntos.

El formato es el mismo para las dos funciones, **q=quad(f,a,b,tol)** donde **tol** controla el error absoluto, por defecto **tol=eps**.

Otra función de cuadratura es '**trapz**' que aplica la regla de los trapecios compuesta.

Las integrales dobles también pueden evaluarse con '**dblquad**'.

5.4 Ecuaciones diferenciales

En *MATLAB* existen varias funciones para resolver problemas de valores iniciales:

$$\begin{aligned}\frac{d}{dt}y(t) &= f(t,y(t)), \\ y(t_0) &= y_0.\end{aligned}$$

donde t es el tiempo e $y(t)$ es un vector m-dimensional. La forma más sencilla de resolver este problema es escribir en *MATLAB* un función que defina la ecuación diferencial y despues llamar a alguno de los resolvedores que se comentan seguidamente.

5.4.1 Un ejemplo con ode45

La función '**ode45**' implementa el método de Dormand-Price que es un par encajado de métodos Runge-Kutta de ordenes 4 y 5.

En este ejemplo se resuelve la ecuación del péndulo simple:

$$\frac{d^2}{dt^2}\theta(t) + \sin \theta(t) = 0,$$

que con el cambio de variable $y_1(t) = \theta(t)$ e $y_2(t) = \frac{d}{dt}\theta(t)$ se convierte en el sistema:

$$\begin{aligned}\frac{d}{dt}y_1(t) &= y_2(t), \\ \frac{d}{dt}y_2(t) &= -\sin y_1(t),\end{aligned}$$

que escrito en *MATLAB* resulta:

```
function yprime=pend(t,y)
%PEND pendulo simple
%
yprime=[y(2); -sin(y(1))];
```

La siguiente secuencia de comandos construye el plano de fases y traza tres trayectorias para los valores iniciales indicados:

```
% EJEM541
```

```
tspan=[0 10];
yazero=[1; 1]; ybzero=[-5; 2]; yczero=[5; -2];
[ta,ya]=ode45('pend',tspan,yazero);
[tb,yb]=ode45('pend',tspan,ybzero);
[tc,yc]=ode45('pend',tspan,yczero);
%
[y1,y2]=meshgrid(-5:.5:5,-3:.5:3);
Dy1Dt=y2; Dy2Dt=-sin(y1);
quiver(y1,y2,Dy1Dt,Dy2Dt)
hold on
%
plot(ya(:,1),ya(:,2),yb(:,1),yb(:,2),yc(:,1),yc(:,2))
axis equal, axis([- 5 5 -3 3])
xlabel y_1(t), ylabel y-2(t),
hold off
```

con el resultado:

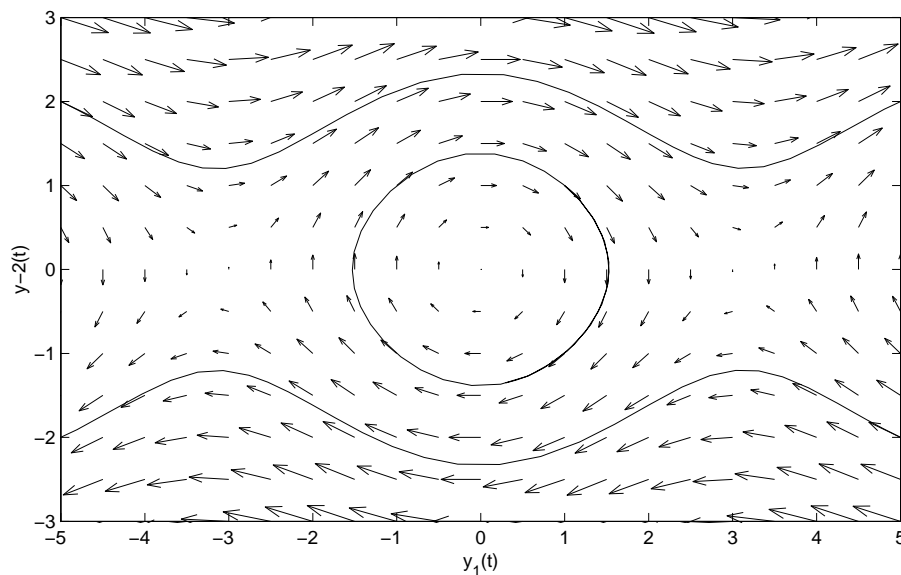


Figura : Pendulo simple

El formato general de 'ode45' es:

```
[t,y]=ode45(f',tspan,yzero,options,p1,p2,...)
```

donde p_1, p_2, \dots son parámetros de la propia f y 'options' controla muchas posibilidades, desde la tolerancia con

```
options=odeset('AbsTol',valor,'RelTol',valor)
```

lo que significa que el test de error será:

$$err(i) \leq \max\{AbsTol, RelTol * |x(i)|\}.$$

El estado de la opciones se puede conocer con la función 'odeset'.

En el siguiente ejemplo se vera otra aplicación muy interesante del 'options'

5.4.2 Un problema de depredador presa

En este ejemplo se supondrá que un conejo sigue un camino prefijado $(r_1(t), r_2(t))$ y que un zorro le está persiguiendo con dos hipótesis:

- En cada instante la tangente del camino del zorro apunta al conejo.
- La velocidad del zorro es k veces la del conejo.

Si se supone que $(y_1(t), y_2(t))$ es la ruta que sigue el zorro, segun las hipótesis anteriores deben ser la solución del sistema de ecuaciones diferenciales:

$$\begin{aligned}\frac{d}{dt}y_1(t) &= s(t)(r_1(t) - y_1(t)), \\ \frac{d}{dt}y_2(t) &= s(t)(r_2(t) - y_2(t)),\end{aligned}$$

donde:

$$s(t) = \frac{k\sqrt{(\frac{d}{dt}r_1(t))^2 + (\frac{d}{dt}r_2(t))^2}}{\sqrt{(r_1(t) - y_1(t))^2 + (r_2(t) - y_2(t))^2}}$$

Cuando el zorro es más rápido que el conejo ($k > 1$), el zorro debe alcanzar la presa de forma que el denominador de $s(t)$ se hace muy pequeño y no puede completarse la integración numérica. Para estos casos 'ode45' permite introducir una opción llamada 'Event' de forma que si durante la ejecución del programa se da una situación concreta se detiene la ejecución y se devuelven los resultados en ese momento. En nuestro ejemplo el evento pudiera definirse diciendo que la distancia del zorro al conejo es menor que 10^{-4} , lo cual queda definido en la función siguiente:

```
function [value,isterminal,direction] = events(t,y)
%EVENTS      Events function for FOX2.
%            Locate when fox is close to rabbit.

r = sqrt(1+t)*[cos(t); sin(t)];
value = norm(r-y) - 1e-4;      % Fox close to rabbit.
isterminal = 1;                % Stop integration.
direction = -1;                % Value must be decreasing through zero.
```

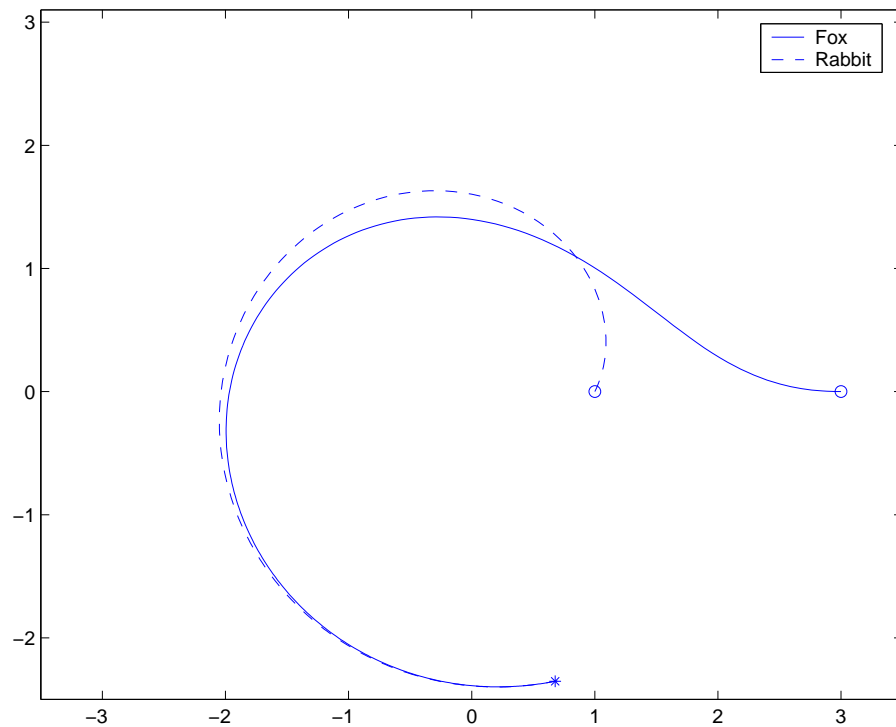
ahora el segundo miembro del sistema diferencial es:

```
function yprime = fox2(t,y)
%FOX2      Fox-rabbit pursuit simulation with relative speed parameter.
%          YPRIME = FOX2(T,Y,K).
k=1.1;
r = sqrt(1+t)*[cos(t); sin(t)];
r_p = (0.5/sqrt(1+t)) * [cos(t)-2*(1+t)*sin(t); sin(t)+2*(1+t)*cos(t)];
dist = max(norm(r-y),1e-6);
factor = k*norm(r_p)/dist;
yprime = factor*(r-y);
```

y el programa siguiente da la solución que despues se dibuja las trayectorias de los dos animales con la caza que tiene lugar en el tiempo $t = 5.071$ en el punto $(.8646, -2.3073)$.

```
% ZORROCO2 ejercicio del zorro y el conejo con evento

k=1.1;
tspan=[0 6];yzero=[3;0];
options=odeset('RelTol',1e-6,'AbsTol',1e-6,'Events','events');
[tfox,yfox]=ode45('fox2',tspan,yzero,options);
plot(yfox(:,1),yfox(:,2)),hold on
plot(sqrt(1+tfox).*cos(tfox),sqrt(1+tfox).*sin(tfox),'--')
plot([3 1],[0 0],'o'), plot(yfox(end,1),yfox(end,2),'*')
axis equal, axis([-3.5 3.5 -2.5 3.1])
legend('Fox','Rabbit',0), hold off
```



5.4.3 Problemas Stiff

La función 'ode45' como ya se conoce no resuelve de manera eficaz todos los problemas diferenciales, por ejemplo los problemas stiff (ver [2] y [3]). Para estos otros tipos de problemas de valores iniciales, *MATLAB* implementa otros resolutores que se indican en la siguiente tabla:

Nombre	Problema tipo	Tipo de algoritmo
ode45	no stiff	Runge-Kutta de orden 4 y 5
ode23	no stiff	Runge-Kutta de orden 2 y 3
ode113	no stiff	Métodos lineales multipaso de orden 1 a 13
ode15s	stiff	BDF de orde 1 a 5
ode23s	stiff	Rosembrock modificados de orden 2 y 3
ode23t	poco stiff	Regla de los trapecios
ode23tb	stiff	Runge-Kutta implicito de orden 2 y 3

En lineas generales estas funciones estan diseñadas para que sea facilmente intercambiables, es decir, las características de '`ode45`' son extensibles a las otras, aunque los experimentadores recomiendan primeras utilizar '`ode45`' y si el problema es stiff usar '`ode15s`'

Chapter 6

El manipulador simbólico

Aunque *MATLAB* nació pensado para la computación científica que manipula sólo números, también ha desarrollado el *Symbolic Math Toolbox* que permite en una pantalla de *MATLAB* hacer cálculo simbólico y de precisión variable basado en *Maple*. El comando `'help symbolic'` lista las operaciones y funciones disponibles con este manipulador.

El *Symbolic Math Toolbox* define un nuevo tipo de dato llamado simbólico (`'sym object'` y la forma más sencilla de crearlos es con los comandos `'sym'` o `'syms'` simplemente escribiendo `syms a b c x` o bien `a=sym('a') b=sym('b')...`

En la siguiente lista se enumeran algunas funciones

<code>diff</code>	deriva una función
<code>factor</code>	simplifica una expresión
<code>int</code>	integra una función
<code>jacobian</code>	calcula la matriz jacobiana
<code>limit</code>	calcula un límite
<code>pretty</code>	escribe la expresión más comodamente
<code>simplify</code>	simplifica una expresión
<code>solve</code>	resuelve una ecuación algebraica
<code>subs</code>	sustitución simbólica
<code>symsum</code>	suma series
<code>taylor</code>	escribe un desarrollo de Taylor

En la siguiente lista aparecen funciones relacionadas con la manipulación simbólica de matrices:

<code>det</code>	calcula el determinante
<code>diag</code>	escribe la diagonal de una matriz
<code>eig</code>	calcula autovalores y autovectores
<code>expm</code>	exponencial matricial
<code>inv</code>	calcula la matriz inversa
<code>null</code>	escribe una base del espacio nulo
<code>poly</code>	escribe el polinomio característico
<code>rank</code>	rango de una matriz
<code>svd</code>	calcula los valores y vectores singulares

El *Symbolic Math Toolbox* tiene otras funciones algunas de ellas dan acceso al *Maple* llamadas `'mfun'` cuya lista puede verse escribiendo `'mfunlist'`.

Bibliography

- [1] L.V.Fausett *Applied Numerical Analysis Using MATLAB*. Prentice-Hall 1999.
- [2] E.Hairer, S.P.Norsett and G.Wanner *Solving O.D.E. Nonstiff Problemas*. Springer 1987.
- [3] E.Hairer and G.Wanner *Solving O.D.E. Stiff and Differential-Algebraic Problems*. Springer 1991.
- [4] D.J.Higham and N.J.Higham. *MATLAB Guide*. Society for Industrial and Applied Mathematics, Philadelphia 2000.
- [5] *MATLAB. The Language of Technical Computing*. Disponible en <http://www.mathworks.com>.
- [6] J.H.Mathews and K.D.Fink. *Métodos Numéricos con MATLAB. Tercera Edición*. Prentice-Hall 1999.
- [7] *The Student Edition of MATLAB*. The Mathworks, Inc. Published by Prentice-Hall 1997.
- [8] César Pérez *Matlab y sus Aplicaciones en las Ciencias e Ingenierías..* Prentice-Hall 2002.
- [9] A.Quarteroni, R.Sacco and F.Saleri *Numerical Mathematics*. Springer 2000.
- [10] L.N.Trefethen *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics, Philadelphia, 2000.
- [11] J.M.Sanz-Serna *Diez lecciones de Cálculo Numérico*. Universidad de Valladolid, Valladolid 1998.
- [12] C.F.Van Loan. *Introduction to Scientific Computing. A Matrix-Vector Approach Using MATLAB. 2nd Edition* Prentice-Hall 2000.

F.Vadillo

Dpto de Matemática Aplicada, Estadística e I.O.

UPV/EHU

<http://picasso.lc.ehu.es/~fernando>