Introducción a MATLAB

MatLab es una abreviatura de la frase Matrix Laboratory. Es un entorno informático de análisis numérico y representación gráfica de fácil manejo. Originalmente fue escrito para la enseñanza de álgebra lineal, aunque actualmente es, al mismo tiempo, un entorno y un lenguaje de programación. También permite crear funciones propias y programas especiales (denominados archivos-M) en código MatLab, que se pueden agrupar en las llamadas *Toolboxes*: colección especializada de archivos-M para trabajar en distintos tipos de problemas, por ejemplo de optimización, de estadística, de ecuaciones diferenciales parciales, etc.

Se puede considerar, por otro lado, que MatLab es una calculadora totalmente equipada aunque, en realidad, es mucho más versátil que cualquier calculadora para hacer cálculos matemáticos. Se trata de una plataforma para el desarrollo de aplicaciones y para la resolución de problemas en múltiples áreas de aplicación.

Entre sus utilidades, se encuentra:

- Cálculo matricial y Algebra lineal.

- Polinomios e interpolación.

- Regresión y ajuste de funciones.

- Ecuaciones diferenciales ordinarias.

- Integración.

- Funciones y gráficos en dos y tres dimensiones.

Para acceder desde Windows se hace doble click sobre su icono.

El programa está accesible desde el siguiente menú: Inicio `->` Archivos de Programa `->` MatLab.

Sale la pantalla en blanco con una línea de comandos indicada por el símbolo

```
>>
```

desde donde se pueden introducir instrucciones.

Para salir de MatLab se utiliza la instrucción `quit`

El elemento básico de MatLab es una matriz rectangular de elementos reales o complejos. MatLab incluso considera los escalares como matrices. Además todas las variables utilizadas en la línea de comandos son almacenadas por MatLab.

En caso de dudas se puede utilizar siempre el comando

```
help
```

Manejo básico

El manejo de escalares (números) es, básicamente, el mismo que el de una calculadora normal Por ejemplo, la operación $2\left(1-\frac{1}{7}\right)$ se introduce como

```
2*(1 - 1/7)
```

y la operación $\cos^2\left(\frac{\pi}{2}\right)$ se introduce como

```
cos(pi/2)^2
```

Por defecto MatLab muestra los resultados por pantalla con cuatro decimales. Si se quiere un formato fraccionario teclear

```
format rat
```

Así, si se vuelve a realizar la operación anterior, MatLab devuelve

```
12/7
```

Si se quiere realizar la operación anterior con 16 dígitos por pantalla teclear

```
format long

2*(1 - 1/7)

ans =

1.71428571428571
```

Para introducir una matriz, se escriben primero los elementos de la primera fila, luego los de la segunda fila separados por un ENTER, o bien por la tecla ; y así sucesivamente

hasta la última fila. Por ejemplo, la siguiente matriz

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

se introduce en MatLab escribiendo en la línea de comandos

```
A = [1 2 -1; 0 0 0]
```

o bien

```
A = [1 2 -1

0 0 0]
```

Notas:

Los símbolos A y a son diferentes para MatLab: se distingue entre mayúsculas y minúsculas.

Se pueden escribir *comentarios* después del signo de tanto por ciento (`%`).

Podemos colocar órdenes múltiples en una línea si se separan por comas o puntos y comas.

Las comas le dicen a MatLab que *visualice* los resultados.

Los puntos y comas suprimen la impresión.

Para separar una línea en varias se ponen puntos suspensivos: **...**

Para interrumpir la ejecución de una instrucción o programa de MatLab en cualquier momento: Ctrl-C.

Funciones Matemáticas Comunes

A continuación se muestra una tabla con las funciones matemáticas en MatLab:

`abs(x)` Valor absoluto o magnitud de un número complejo

`acos(x)` Inversa del coseno

`acosh(x)` Inversa del coseno hiperbólico

`angle(x)` Angulo de un número complejo

`asin(x)` Inversa del seno

`asinh(x)` Inversa del seno hiperbólico

`atan(x)` Inversa de la tangente

`atan2(x,y)` Inversa de la tangente en los cuatro cuadrantes

`atanh(x)` Inversa de la tangente hiperbólica

`ceil(x)` Redondea hacia más infinito

`conj(x)` Complejo conjugado

`cos(x)` Coseno

`cosh(x)` Coseno hiperbólico

`exp(x)` Exponencial

`fix(x)` Redondea hacia cero

`floor(x)` Redondea hacia menos infinito

`imag(x)` Parte imaginaria de un número complejo

`log(x)` Logaritmo natural

`log10(x)` Logaritmo decimal

`real(x)` Parte real de un número complejo

`rem(x,y)` Resto después de la división

`round(x)` Redondea hacia el entero más próximo

`sign(x)` Devuelve el signo del argumento

`sin(x)` Seno

`sinh(x)` Seno hiperbólico

`sqrt(x)` Raíz cuadrada

`tan(x)` Tangente

`tanh(x)` Tangente hiperbólica

Notas:

MatLab sólo opera en radianes.

Para ver las diferentes funciones elementales y trigonométricas teclear

```
help elfun
```

La siguiente orden borra de *memoria* todas las variables

```
clear
```

Archivos M

Se pueden colocar órdenes en un simple archivo de texto (o ascii) y, a continuación, hacer que MatLab lo abra y evalúe las órdenes exactamente como si hubiesen sido escritas desde la línea de comandos.

Estos archivos se llaman archivos script o archivos-M, y deben finalizar con la extensión 'm'.

Para crear un archivo-M se escoge New del menú File y seleccionamos M-file. Una vez guardado este archivo-M, MatLab ejecutará las órdenes en dicho archivo simplemente escribiendo su nombre (sin extensión) en la línea de comandos.

Normalmente, las órdenes leídas desde el archivo-M no se visualizan cuando se evalúan. La orden `echo on` le dice a MatLab que visualice o efectúe un eco de las órdenes en la ventana de Orden cuando se leen y evalúan. También existe la función `echo off`.

Funciones elementales para la construcción de matrices

`zeros(n)` Matriz de ceros $(n \times n)$.

`ones(n,m)` Matriz de unos $(n \times m)$.

`rand(n,m)` Matriz $(n \times m)$ de números aleatorios distribuidos uniformemente entre cero y uno.

`randn(n,m)` Matriz $(n \times m)$ de números aleatorios distribuidos normalmente con media cero y varianza unidad.

`eye(n,m)` Matriz identidad $(n \times m)$.

Ejemplos

```
I = eye(6)
```

crea la matriz identidad 6x6

```
Z = zeros(6,6)
```

crea una matriz de ceros 6x6

```
O = ones(6,1)
```

crea un vector de unos de dimensión 6

```
R = rand(6,6)
```

matriz 6x6 generada aleatoriamente

```
v = diag(R)
```

extrae la diagonal de R

Utilización del carácter :

Este comando es de mucha utilidad a la hora de programar con vectores y matrices. Con el carácter `:` Se pueden extraer partes de una matriz. Los siguientes ejemplos explican su utilización

```
x = [0:0.1:2]
```

vector de 0 a 2 con paso 0.1

```
I = eye(6)
```

matriz identidad 6x6

```
x = I(1,2)
```

elemento (1,2) de I

```
x = I(3,:)
```

toda la fila tercera de I

```
x = I(:,2)
```

toda la columna segunda de I

```
B = I(2:5,2:6)
```

filas 2 a 5 y columnas 2 a 6

```
I(:,[2 4 6]) = R(:,1:3)
```

Reemplaza las columnas 2,4,6 de I por las 3 primeras de R

También son útiles los siguientes comandos:

```
v = R(:)
```

vector que contiene a todas las columnas de R

```
Ru = triu(R)
```

matriz triangular superior de R

```
Rl = tril(R)
```

matriz triangular inferior de R

Operaciones con matrices

Las siguientes operaciones se pueden realizar de forma sencilla en MatLab.

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

```
B = [0 1;1 0]
```

```
A*B
```

producto matricial de A y B

```
A.*B
```

producto componente a componente A y B

```
inv(A)
```

inversa de la matriz (cuadrada) A

```
pinv(A)
```

pseudoinversa de la matriz A

```
det(A)
```

determinante de la matriz (cuadrada) A

```
r = rank(A)
```

dimensión de la imagen de A

```
[n,m] = size(A)
```

numero de filas y columnas de A

```
trace(A)
```

traza de A

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

factorización LU de la matriz A, es tal que $P \cdot A = L \cdot U$

```
lambda = eig(A)
```

vector que contiene los autovalores de A

```
[S,lambda] = eig(A)
```

La matriz S contiene los autovectores (columnas) de A, y λ es una matriz diagonal de autovalores tal que $A \cdot S = S \cdot \lambda$

```
[Q,R] = qr(A)
```

factorización QR de A, Q es una matriz ortogonal de dimensión $m \times m$ y R una matriz triangular superior de dimensión $m \times n$ tal que $A = Q \cdot R$

```
expm(A)
```

e^A

```
poly(A)
```

Da el polinomio característico de A

Guardar resultados

En MatLab existen dos modalidades para guardar resultados. La primera es mediante la instrucción,

```
diary('fichero1')
```

Esta orden, poniéndola al principio de la sesión, guarda en el fichero de texto fichero1 todo lo que se haya realizado en una sesión de MatLab. Esta orden sólo guarda los comandos introducidos en la línea de comandos pero no las variables y matrices. Para guardarlas y cargarlas se teclea

```
save fichero2
```

guarda todas las variables en fichero2.mat

Este formato sólo es legible mediante MatLab

```
save fichero3 x A
```

guarda sólo las variables x y A en fichero3.mat

```
save fichero4.dat x A -ascii
```

guarda las variables x y A en ASCII en fichero4.dat

```
load fichero3
```

carga variables de fichero3.mat (binario)

```
load fichero4.dat -ascii
```

carga variables de fichero4.dat (ASCII) sea cual sea su extensión.

```
who
```

Lista todas las variables del espacio de trabajo.

```
clear
```

Borra todas las variables del espacio de trabajo.

```
clear(v1,v2)
```

Borra todas las variables v1 y v2 del espacio de trabajo.

Gráficos

MatLab presenta un entorno gráfico de muy fácil manejo.

El ejemplo más sencillo para crear gráficos es el siguiente

```
x = rand(10,1);
y = rand(10,1);
plot(x,y)
```

El comando `plot` dibuja los puntos $(x_i; y_i)$ uniéndolos por líneas continuas.

Para dibujar un diagrama de dispersión de las variables x e y se teclea

```
plot(x,y,'.')
```

Cuando se pone el comando así

```
plot(x)
```

puede resultar muy útil, por ejemplo en series temporales, ya que dibuja los puntos $(x_i; i)$ uniéndolos por líneas continuas.

```
subplot(m,n,p)
```

Divide el gráfico en $m \times n$ ventanas poniendo el gráfico correspondiente en la posición p (empezando por arriba de izquierda a derecha).

Para dibujar la función $f(x) = xlog(x)$ en el intervalo $[0; 3]$ se puede teclear

```
x = [0:0.01:3];

y = x.*log(x);

plot(x,y)

grid on

title('grafico')

text(1,0.65,'y = x log(x)')
```

Nota: el comando `grid on` imprime una malla en el gráfico.

Ejemplo: para dibujar la función $y = xsin(1/x)$ se escribe

```
plot(x,x.*sin(ones(size(x))./x))
```

El comando `size` obtiene las dimensiones del argumento. Así, con la orden `ones` se dibuja una vector columna de *unos* de tamaño `size` de x (esto es la longitud del vector x) y se divide luego entre el valor de cada x.

Observar el uso del comando `ones` y del `./`

Existe una función que evalúa cuidadosamente la función que se va a representar. Como entrada, esta función necesita conocer el nombre de la función en forma de una cadena de caracteres y el rango de representación como un array de dos elementos: `fplot('nombre',[a,b])`

```
fplot('t*sin(1/t)',[0 3])
```

Para definir los ejes X e Y de un gráfico se utiliza la orden

```
axis([Xmin Xmax Ymin Ymax])
```

Para gráficos en tridimensional se pone:

```
plot3(x,y,z)
```

Para los demás comandos gráficos consultar usando:

```
help plotxy
```

```
help graphics
```

Operadores Relacionales y Operadores Lógicos

los operadores relacionales típicos son:

`<` Menor que

`<=` Menor que o igual a

`>` Mayor que

`>=` Mayor que o igual a

`==` Igual a

`~=` No igual a

La salida de las operaciones lógicas se pueden utilizar también en operaciones matemáticas.

Los operadores lógicos proporcionan un medio de combinar o negar expresiones rela-

cionales.

`&` AND

`|` OR

`~` NOT

Ejemplo

```
A=1:9, B=9-A
```

```
A = 1 2 3 4 5 6 7 8 9
```

```
B = 8 7 6 5 4 3 2 1 0
```

```
tf=A>4
```

Encuentra elementos de A que son mayores que 4.

```
tf = 0 0 0 0 1 1 1 1 1
```

```
tf=A==B
```

Encuentra elementos de A que son iguales a aquellos en B.

```
tf = 0 0 0 0 0 0 0 0 0
```

```
tf=B-(A>2)
```

Encuentra donde A>2 y resta el resultado de B.

```
tf = 8 7 5 4 3 2 1 0 -1
```

Ejemplo 2:

```
A=1:9; B=9-A;
```

```
tf=A>4
```

Encuentra donde A es mayor que 4.

```
tf = 0 0 0 0 1 1 1 1 1
```

```
tf=~(A>4)
```

Niega el resultado anterior.

```
tf = 1 1 1 1 0 0 0 0 0
```

```
tf=(A>2)&(A<6)
```

Devuelve unos donde A es mayor que 2 y menor que 6.

```
tf = 0 0 1 1 1 0 0 0 0
```

Controles de flujo

```
for x = array
    ordenes
end
```

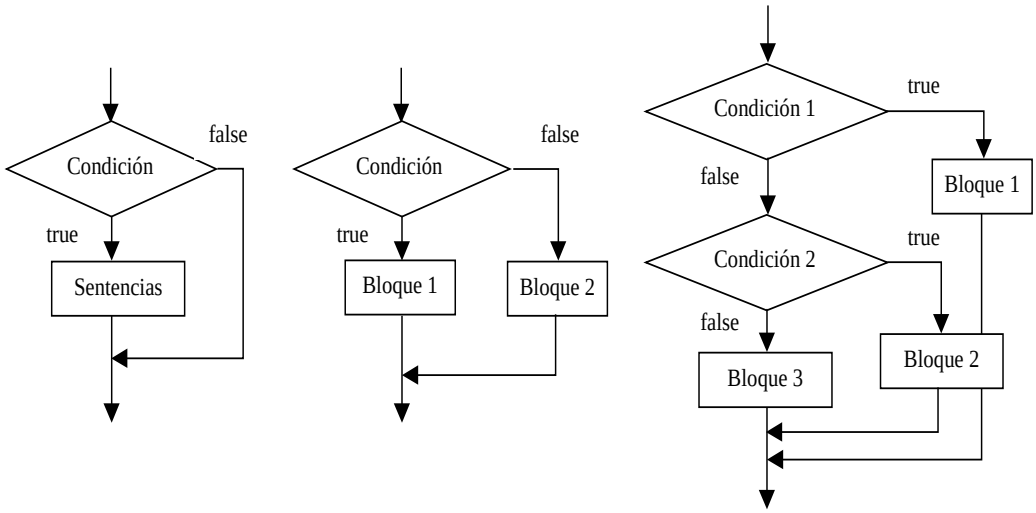
Un bloque `for` que en cada iteración asigna `x` a la columna i-ésima de `array` y ejecuta `ordenes`.

```
while expresion
    ordenes
end
```

Un bloque `while` que ejecuta órdenes mientras todos los elementos de `expresion` son verdaderas o distinto de cero.

```
if expresion
    ordenes
end
```

Una simple estructura `if-else-end` donde `ordenes` se ejecutan si todos los elementos en expresión son verdaderas.




```
if expresion
    ordenes evaluadas si exp=verdadero
else
```

```
    ordenes evaluadas si exp=falso
end
```

Una estructura `if-else-end` con dos caminos. Un grupo de órdenes se ejecuta si `expresion` es verdadera. El otro conjunto se ejecuta si `expresion` es falsa o cero.

```
if expresion1
    ordenes evaluadas si expresion1 es verdadera
elseif expresion2
    ordenes evaluadas si expresion2 es verdadera
elseif
...
else
    ordenes evaluadas si ninguna otra expresion es verdadera
end
```

Es la estructura más general `if-else end`.

`break` Termina la ejecución de bucles `for` y bucles `while`.

Ejemplo: (bucle for)

```
for n=1:5
    for m=5:-1:1
    A(n,m)=n^2+m^2;
    end
disp(n)
end
A
```

Notas:

La condición del if puede ser una condición matricial, del tipo `A==B`, donde A y B son matrices del mismo tamaño. Para que se considere que la condición se cumple, es necesario que sean iguales dos a dos todos los elementos de las matrices A y B. Basta que haya dos elementos diferentes para que las matrices no sean iguales, y por tanto las sentencias del if no se ejecuten.

Análogamente, una condición en la forma `A~=B` exige que todos los elementos sean diferentes dos a dos. Bastaría que hubiera dos elementos iguales para que la condición no se cumpliese.

En resumen:

`if A==B` exige que todos los elementos sean iguales dos a dos

`if A~=B` exige que todos los elementos sean diferentes dos a dos.

Ejemplo: (estructura if-else-end)

```
manzanas=10;

coste=manzanas*25 % Numero y coste de manzanas.

coste

if manzanas > 5 % Aplicar 20% de descuento por cantidad.

coste=(1-20/100)*coste;

end

coste
```

Operaciones sobre funciones

Para encontrar mínimos de funciones unidimensionales y n-dimensionales usamos, respectivamente, las funciones:

```
fmin('nombre_funcion',a,b)

fmins('nombre_funcion',a,b)
```

Para buscar el cero de una función unidimensional usamos:

```
fzero('nombre_funcion',a)
```

donde a es el punto cerca del cual se busca el cero. A la función `fzero` debe darse el nombre de una función hecha por el usuario cuando se la llama.

También puede utilizarse para encontrar dónde una función es igual a cualquier

constante.

La función `inline` transforma en función una cadena de caracteres.

```
g = inline(expresion)
```

Por ejemplo,

```
 fmin(inline('cos(x)'),3,4)
```

```
solve('ecuacion','x')
```

resuelve la ecuación en la variable x.

MatLab proporciona tres funciones para calcular numéricamente el área bajo una función sobre un rango finito: `trapz`, `quad` y `quad8`.

`trapz(x,y)` aproxima la integral bajo una función al sumar el área de los trapezoides formados con los puntos.

Las funciones `quad` y `quad8` realizan aproximaciones de un orden más elevado que un simple trapezoide, mediante la regla de Simpson. Funcionan igual que `fzero`.

Ejemplo de una función

```
function[x1,x2]=ecua2(a,b,c)
% Esta funcion resuelve la ecuacion de segundo grado
% ax^2 + bx +c = 0
% a, b, c son parametros de entrada
```

```
% x1, x2 son parametros de salida


d=b^2-4*a*c;

x1=(-b+sqrt(d))/(2*a);

x2=(-b-sqrt(d))/(2*a);
```

La sucesión de Fibonacci $\{a_n\}$ se define con la recurrencia

$$\begin{aligned} a_1 &= 1 \\ a_2 &= 1 \\ a_n &= a_{n-1} + a_{n-2} \end{aligned}$$

```
function y=fibo(x)

if x<=1 y=1;

else

y=fibo(x-1)+fibo(x-2);

end
```

Poner

```
fibo(2)

fibo(8)

fibo(20)
```

Lectura y grabación de datos

Supongamos un fichero llamado datos.txt que tiene, por ejemplo, dos columnas de valores que son números reales.

Para leer los datos del fichero:

```
[x1,x2] = textread('datos.txt','%f %f');
```

Probar

```
help textread
```

para otras opciones de formatos de entrada.

Otra opción más:

```
load 'datos.txt' -ascii
 x1 = datos(:,1);
 x2 = datos(:,2);
```

Para grabar en ficheros de datos ascii:

Por ejemplo, grabar en el fichero sale.txt las matrices V y W:

```
save 'sale.txt' V W -ascii
```

Ecuaciones diferenciales

Una ecuación diferencial ordinaria se puede expresar como

$$\begin{aligned} y' &= f(t,y) \\ y(t_0) &= y_0 \end{aligned}$$

que se puede generalizar para un vector $y = (y_1, \ldots, y_n)$.

Comandos Básicos:

`ode45`: Ecuaciones diferenciales por el método de Runge-Kutta

`ode15s`: Cuando con el método anterior la convergencia a la solución es lenta. Para problemas complejos.

`ode113`: Ecuaciones diferenciales por el método de Adams. Para situaciones computacionalmente intensivas.

USO:

```
[t,y]=ode##(funcion, vectorIntervalo, y0)
```

Ejemplo:

Defino un sistema de ecuaciones:

$$\begin{aligned} y_1' &= y_2 y_3 & \text{con} \quad y_1(0) &= 0 \\ y_2' &= -y_1 y_3 & \text{con} \quad y_2(0) &= 1 \\ y_3' &= -0{,}51 y_1 y_2 & \text{con} \quad y_3(0) &= 1 \end{aligned}$$

Se define una función aparte en un fichero M:

```
function dy=sistema1(t,y)

% defino el vector columna

dy=zeros(3,1);

dy(1) = y(2) * y(3);

dy(2) = -y(1) * y(3);

dy(3) = -0.51 * y(1) * y(2);
```

Para resolver el sistema, se pone

```
[T,Y] = ode45('sistema1', [0 12] , [0 1 1]);
```

Para interpretar los resultados se pone

```
plot(T,Y(:,1),'b-',T,Y(:,2),'r-.',T,Y(:,3),'g.')
```

Ejemplo:

Resolver el sistema de ecuaciones diferenciales de *Van der Pol*:

$$\begin{aligned} y_1' &= y_2 && \text{con} \quad y_1(0) = 0 \\ y_2' &= 10\,(1 - y_1^2)\,y_2 - y_1 && \text{con} \quad y_2(0) = 1 \end{aligned}$$

Se define primero una función en un fichero M:

```
function dy=vpool(t,y)

% defino el vector columna

dy=zeros(2,1);

dy(1) = y(2);

dy(2) = 10 * (1- y(1)^2)*y(2) - y(1);
```

Para resolver el sistema, se pone

```
[T,Y] = ode15s('vpool', [0 50] ,[2 0]);
```

Para interpretar los resultados se pone

```
subplot(1,2,1)

plot(T,Y(:,1),'b-')

subplot(1,2,2)

plot(T,Y(:,2),'r-')
```

Ejemplo:

Supongamos que se trata de estudiar el efecto de los coeficientes de interacción α y β en un modelo de Lotka Volterra de depredador–presa

$$\begin{aligned} \frac{dy_1}{dt} &= y_1 - \alpha y_1 y_2 \\ \frac{dy_2}{dt} &= -y_2 + \beta y_1 y_2 \end{aligned}$$

La tasa de incremento de la población de presas disminuye de modo proporcional al número de encuentros con la población de depredadores. Del mismo modo, la tasa de incremento de la población de depredadores disminuye según su número aumenta, al competir más individuos por las presas.

Este sistema de ecuaciones es difícil de tratar desde el punto de vista analítico, por lo que consideramos su desarrollo en forma numérica. Se puede considerar el siguiente programa en MatLab:

```
% cd c:\cajon
```

```
% Se definen como globales las siguientes variables

% alfa: se relaciona con el efecto de los depredadores

% sobre las presas

% beta: se relaciona con el efecto de los presas

% sobre los depredadores

global alfa beta

alfa = input('Dime cual es la tasa del efecto

de los depredadores sobre las presas: ');

beta = input('Dime cual es la tasa del efecto

de las presas sobre los depredadores: ');

[t,y] = ode45('lotka',[0 60],[100 150]);

% de t0 a tN unidades de tiempo

% condiciones iniciales

feo = input('Que grafica quieres: 1 (temporal), 2 (conjunta): ');

if (feo==1)

% grafico de la evolucion de las especies con el tiempo

 plot(t,y(:,1),t,y(:,2))

 elseif (feo==2)

% grafico de la evolucion conjunta de las especies

 plot(y(:,1),y(:,2))

  else

fprintf('teclea bien \n');

end;
```

donde se tiene que definir previamente la función *lotka.m* en otro fichero que contiene:

```
function [yp] = lotka(t,y)

% Lotka-Volterra predator-prey model.

global alfa beta

yp = [y(1) - alfa*y(1)*y(2); -y(2) + beta*y(1)*y(2)];
```

Se pueden considerar las gráficas tanto de la evolución de las especies con el tiempo, como de la evolución conjunta de las especies.

Matrices de Leslie

supongamos una población con 3 grupos de edad asociados cuya matriz asociada es

$$\begin{pmatrix} 0 & 4 & 3 \\ 0{,}5 & 0 & 0 \\ 0 & 0{,}25 & 0 \end{pmatrix}$$

cada hembra en el segundo grupo da lugar a 4 descendientes, y en el tercer grupo da lugar a 3. Desde el primer grupo pasan al segundo el 50 %, y del segundo al tercero pasan el 25 %.

Supongamos que se parte de la población inicial

$$x^{(0)} = \begin{pmatrix} 10 \\ 10 \\ 10 \end{pmatrix}$$

```
L=[0 4 3; 0.5 0 0; 0 0.25 0];

X=zeros(3,11);

X(:,1)=[10; 10; 10];

for (k=2:11)

X(:,k)=L*X(:,k-1);

end

format short;

X

plot(X');

% Se dibuja el logaritmo de la poblacion
```

```
semilogy(X');
```

Si se parte de una población inicial X^0

```
X0=[10; 10; 10];

X10=L^10*X0;

% para calcular los autovalores

roots(poly(L));

% o bien

[V,D]=eig(L);

V(:,1)

V1=V(:,1)/sum(V(:,1))

% se compara con

X100=L^100*X0;

V1N=X100/sum(X100)

% compruebo que la tasa de crecimiento coincide con el primer autovalor

X(:,100)./X(:,99)

eig(L)
```

Se puede hacer el gráfico correspondiente

```
L=[0 4 3; 0.5 0 0; 0 0.25 0];

X=zeros(3,101);

X(:,1)=[10; 10; 10];

for (k=2:101)
```

```
X(:,k)=L*X(:,k-1);

end

for (k=2:101)

G(:,k)=X(:,k)/sum(X(:,k));

end

plot(G');

h=plot(G');

set(h(1),'LineStyle','-')

set(h(2),'LineStyle',':')

legend('Primer grupo de edad','Segundo grupo de edad','Tercer grupo de edad')
```

```
%-----------------------------------------------------------------

% dibuja la ecuacion en diferencias x(n)=(1.02^n)*x0

x0=100;

t =1:1:100;

s=((1.02).^t).*x0;

plot(t,s,'.');

%-----------------------------------------------------------------

% dibuja la ecuacion en diferencias x(n)=(a^n)*x0

t =1:1:100;

a=0.99;

x0=500;

s=((a).^t).*x0;

plot(t,s,'.')

xlabel('tiempos');

%-----------------------------------------------------------------

% dibuja la ecuacion en diferencias T(n) = 80 + 100(0.95)^n

t = 0:1:60;

s=((0.95).^t).*100 + 80;
```

```
plot(t,s,'.')

xlabel('tiempos');

axis([0 60 0 180]);

hold on;

fplot('80',[0 60],':');

hold off;


%-----------------------------------------------------------------


function b=par(n)

% b=par(n). If n es un numero entero par, entonces b=1

% de otro modo, b=0.


if mod(n,2)==0,

   b=1;

   else b=0;

end


%-----------------------------------------------------------------


function c=add(a,b)

% Esta funcion suma las matrices a and b.

% Duplica la funcion a+b de MATLAB
```

```
[m,n]=size(a);

[k,l]=size(b);

if m~=k | n~=l,

   r='ERROR usando add: matrices no son del mismo tama{\~n}o';

   return,

end

c=zeros(m,n);

for i=1:m,

   for j=1:n,

      c(i,j)=a(i,j)+b(i,j);

   end

end


%-----------------------------------------------------------------


function c=mult(a,b)

% Esta funcion multiplica las matrices a and b.

% Duplica la funcion a*b de MATLAB


[m,n]=size(a);

[k,l]=size(b);

if n~=k,
```

```
   c='ERROR usando mult: matrices no son compatibles
      para su multiplicacion',
   return,
end,
c=zeros(m,l);
for i=1:m,
   for j=1:l,
      for p=1:n,
         c(i,j)=c(i,j)+a(i,p)*b(p,j);
      end
   end
end

%-----------------------------------------------------------------

function y=twoexp(n)
% Este es un programa recursivo para calcular y=2^n.
% El programa se para solo si n es un numero no negativo

if n==0, y=1;
   else y=2*twoexp(n-1);
end
```

```
%-----------------------------------------------------------------

% Ecuaciones en diferencias

% Para resolver ecuaciones en diferencias en MatLab se usa

% filter(b,a,x)

% poner help filter


% Ejemplo

%     y(n) + 1/2*y(n-1) = x(n) + 2*x(n-2)

% donde

%     x(n) = {1, 2, 3, 4, 2, 1}


% Defines x(n)

x = [1 2 3 4 2 1]    % Observar que no se necesita simetria en este ejemplo


% Defines a

a = [1 1/2 0];


% Defines b

b = [1  0  2];


y = filter(b,a,x)
```

```
%-----------------------------------------------------------------

% Segundo ejemplo

%  y(n) - 0.8*y(n-1) = 2*x(n) + 3*x(n-1)

% Defines n

n = 0 : 9;  % Consideras 10 valores (que se pueden cambiar)

%  Defines x(n)

x = (n==0);  % Esto da una funcion impulso donde x(0)=1, x(n)=0 para el resto

% Defines a

a = [1 -0.8];

%  Defines b

b = [2  3];

y = filter(b, a, x)

% Observar que se necesita colocar todos los y's en la ecuacion (y(n), y(n-1) ...)

% en el lado izquierdo de la ecuacion y los de x en el lado derecho

% entonces hay que tomar los coeficientes con sus signos
```