Grado en Ingeniería en Inteligencia Artificial

SEÑALES Y SISTEMAS

Práctica nº 1

Antonio Valle Sánchez

DFESTS

UA | UNIVERSITAT D'ALACANT
UNIVERSIDAD DE ALICANTE

PRÁCTICA 1
INTRODUCCIÓN A MATLAB

1. ¿Qué es MATLAB?

MATLAB (MATrix LABoratory) es un entorno de programación de alto nivel especialmente diseñado para el análisis numérico, la simulación y la visualización de datos; fue desarrollado por MathWorks en los ochenta y es habitualmente utilizado en ingeniería, ciencias, finanzas y otras disciplinas donde se requiere el análisis y procesamiento de datos a nivel avanzado.

Esta herramienta, proporciona un entorno de trabajo que permite realizar desde cálculos numéricos, algoritmos y problemas con formulación matricial, hasta la representación y análisis de señales, adaptándose perfectamente también al desarrollo de cálculos con sistemas. Uno de los motivos por el que se estudia en esta asignatura.

Otro motivo, es la necesidad de enseñar al alumno distintos lenguajes de programación. Como es sabido, en la actualidad, el lenguaje de programación más utilizado en IA es Python, por las ventajas que proporciona, y se estudia en distintas asignaturas de esta carrera. Además de este lenguaje también se realizan desarrollos de Inteligencia Artificial en Java, C++, R o MATLAB. Para cumplir con el objetivo de conocer otros lenguajes de programación, además de Python, las prácticas y los proyectos de Señales y sistemas se realizan en MATLAB.

Con este lenguaje de alto nivel se pueden crear modelos de IA con unas pocas líneas de código o utilizar modelos previamente entrenados, usar herramientas específicas de un área de conocimiento, combinar técnicas de IA con simulación a nivel de sistema para reducir los errores en la producción o intercambiar modelos de IA y funcionalidad de diseño entre MATLAB y otros lenguajes.

Una de las características más importante de MATLAB es la facilidad con la que se puede ampliar, permitiendo a cualquiera crear aplicaciones y funciones integrables de modo que puedan ser utilizadas por el resto de los usuarios. De esta manera, permite a los ingenieros y científicos trabajar en proyectos concretos y hace posible la colaboración entre equipos y organizaciones de diferentes entornos.

MATLAB interpreta los comandos que le son introducidos por teclado, procesándolos inmediatamente y mostrando el resultado. También permite ejecutar secuencias de comandos almacenados en ficheros con extensión **.m**, llamados ficheros m-files. Los ficheros **m** son ficheros de caracteres ASCII que pueden crearse o modificarse con cualquier editor de texto convencional. Los ficheros tipo **m** pueden referenciar a otros ficheros similares, o incluso referenciarse a sí mismos recurrentemente. Además, existe otro formato de archivo con extensión **.mat** que permite almacenar, además de las instrucciones de código, variables y datos.

1.1. Primeros pasos con MATLAB

El símbolo propio de MATLAB es >>

Al aparecer en la pantalla, espera que se escriba una instrucción para ser ejecutada.

A partir de ahora, se aconseja ir repitiendo desde MATLAB todos los ejemplos que van a ir apareciendo. Esto ayudará a coger confianza e ir descubriendo todas las posibilidades básicas que nos ofrece este paquete de simulación.

El tipo de datos básico que utiliza MATLAB es la matriz (como casos particulares son los vectores o las variables unidimensionales). Para definir una variable en modo interactivo simplemente se ha de asignar un valor a un nombre cualquiera. Este nombre ha de comenzar por una letra y no debe tener más de 18 caracteres. Además, MATLAB hace distinción entre letras mayúsculas y minúsculas.

Así, por ejemplo, haciendo:

```
>> a=5
```

MATLAB ha asignado el valor 5 a la variable `a`. La variable **`ans`** está reservada por el sistema para guardar el resultado de una operación sobre la que no se ha especificado ningún nombre, así la instrucción

```
>> 3^5+17
```

proporciona una variable `ans=260` (3 elevado a la quinta + 17). Esta variable puede usarse como cualquier otra, así se puede hacer

```
>> A=2*ans
```

y nos dará `A=520`. El sistema diferencia entre minúsculas y mayúsculas. Con

```
>> aA=a*A
```

se obtiene como respuesta `Aa=2600`. Observamos que si se pone un punto y coma (`;`) al final de la instrucción el resultado no aparece en pantalla. Compruébelo ejecutando:

```
>> aA=a*A;
```

Todas las variables se almacenan en forma de matriz de elementos complejos, aunque pueden introducirse datos reales o enteros, como ya hemos visto. Si la parte imaginaria de todos los elementos de la matriz es cero, entonces MATLAB sólo muestra en pantalla la parte real.

Para introducir números complejos, MATLAB asigna por defecto la raíz cuadrada de -1 a las variables de nombre `i` y `j`. Podrá emplearse, por tanto, cualquiera de estas dos variables como identificador de la parte imaginaria de un número. El usuario del entorno tendrá que tener la precaución de no asignar a estas variables otro valor distinto del que ya tienen.

Por ejemplo, para crear una variable `A` de valor 2-3j, basta con introducir el comando:

```
>> A=2-3*j;
```

o bien:

```
>> A=2-3*i;
```

Las instrucciones en la ventana de comandos se pueden encadenar en una misma línea, separadas por comas (`,`) o por punto y coma (`;`). En el primer caso aparecen en la pantalla los resultados de cada instrucción, mientras que en el segundo caso no. Si una instrucción es demasiado larga y ocupa más de una línea, se finaliza la línea con tres puntos (`...`) y se continua en la línea siguiente.

Los operadores aritméticos básicos son `+`, `-`, `*`, `/`, `\`, `^` (suma, resta, producto, división, división inversa y potencia, respectivamente). Ejemplos:

```
>>1/4    %División normal

ans=0.25

>>1\4...%División inversa

ans=4

>>c=ans^2 ...%Potenciación

c=16
```

1.1.1. Matrices

Existen cuatro modos de definir una matriz en MATLAB.

- Introduciendo explícitamente la lista de elementos de la matriz.
- Generándola mediante comandos y declaraciones disponibles en MATLAB.
- Creándola con ficheros M.
- Cargándola de ficheros MAT.

Un ejemplo típico del primer modo lo tenemos en la siguiente declaración:

```
>>A=[1 2 3;4 5 6;7 8 9]
```

que representa una matriz 3 por 3, llamada `A`, con los números naturales del 1 al 9.

Para ver el tamaño de la matriz:

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

Los corchetes, `[ ]`, se emplean como delimitadores de los elementos de la matriz. El punto y coma se emplea aquí para marcar el fin de cada fila, aunque también puede hacerse esto último con el retorno de carro. Los elementos individuales de cada fila pueden separarse por espacios o por comas. Así que `[1 -2 3]` es lo mismo que `[1,-2,3]` pero no es igual que `[1-2,3]`.

Los elementos de una matriz pueden ser expresiones matemáticas. Por ejemplo. la sentencia:

```
>> B = [-1.3  sqrt(3)  (1+2+3)*4/5]
```

produce el vector `[-1.3000  1.7321  4.8000]`.

Puede accederse a los elementos de una matriz de la misma forma que se hace en los lenguajes de programación convencionales, es decir, indicando los subíndices entre paréntesis, donde el primer subíndice es el número de fila y el segundo subíndice el número de columna. Por ejemplo: `A(2,3)` hace referencia al tercer elemento de la segunda fila de la matriz `A`. Tecléese `A(2,3)` en la línea de comandos del MATLAB seguido de un retorno de carro y compruébese que el resultado devuelto es `6`.

También puede tenerse acceso a una fila o columna completa por medio del símbolo dos puntos `(:)` como subíndice libre. Por ejemplo `A(2,:)` es la segunda fila de `A`, y `B(:,j)` es la j-ésima columna de `B`.

Pueden manipularse submatrices empleando un vector como subíndice. El vector puede ser una matriz definida previamente, o puede constituirse con los delimitadores `[ ]` tal como se explicó más arriba. Por ejemplo, si hacemos `v = [1 3]`, y cogemos el vector `B` anterior, tendríamos que `B(v)` es igual a `[-1.3000  4.8000]`.

El símbolo dos puntos ":" permite una gran flexibilidad al construir matrices y submatrices, tal y como muestran los ejemplos siguientes:
(habrá que dar valor a i, j y k)

```
>>x = j:k; %Es equivalente a [j,j+1,j+2,...,k] (será vacío si j>k).

>>x = j:i:k; %Es equivalente a [j,j+i, j+2i,....k].

>>A([1,3],:) = A([3,1],:);%Intercambia las filas 1 y 3 de la matriz A.

>>A(2:3,1:2); %Submatriz de A formada por las filas 2a3 y las columnas 1a2.

>>A(:); %Es un vector columna con TODOS los elementos de A.
```

En MATLAB las matrices se redimensionan automáticamente. Por ejemplo, si hacemos

```
>> B = [-1.3000  1.7321  4.8000],
```

entonces

```
>> B(5) = abs(B(1)),
```

produce

```
>> B = [-1.3000  1.7321  4.8000  0.000  1.3000]
```

Otra operación matricial importante es el cálculo de la **traspuesta de una matriz**. La forma de realizarla es empleando el apóstrofe. Existen dos versiones:

- `A.'` calcula la matriz traspuesta de `A`, mientras que
- `A'` calcula la matriz traspuesta conjugada de `A`.

(*sólo si diferencian si la matriz está formada por elementos complejos*)

Una aplicación interesante y directa de esta operación es el cálculo del producto escalar de dos vectores, `a` y `b`, escribiendo `a*b.'`.

Por ejemplo, si `a = [1  -1  1]` y `b =[0  2  4]`, entonces tendríamos:

```
>>a*b.' = [2]
>>a*b' = [2]
```

Pero sin embargo, si tenemos `c = [1  i  1]` y `d=[0  j  4]`, entonces:

```
>>c*d.' = [3]
>>c*d' = [5]
```

Finalmente, la sentencia `A=[]`, borra la variable `A` asignándole una matriz de dimensión cero por cero.

1.1.2. Números complejos y matrices

El formato de los números complejos en MATLAB es como sigue:

El número imaginario `i` o `j` al igual que el número `pi` están predefinidos:

```
>>i              %Devuelve ans = 0+1.000i
>>j              %Devuelve ans = 0+1.000i
```

Genere, por ejemplo, un número complejo de la siguiente manera:

```
>>z1=j
```

MATLAB incorpora funciones para la **representación gráfica** (como veremos más adelante). La función básica es `plot()`. Ejecute:

```
>>plot(z1)
%Representa la parte imaginaria en función de la parte real.
```

Observe que se crea una ventana para el gráfico. Al dibujar se ha hecho una selección automática de los ejes de la representación, y se ha representado un punto en las coordenadas correspondientes del número complejo `z1`.

En ocasiones puede suceder que al dibujar un punto, este no sea visible en la pantalla. La función `plot` permite seleccionar entre uno de los siguientes marcadores: "`.`", "`o`", "`+`", "`x`", "`*`". También es posible elegir el color de la representación: "`r`", "`g`" , "`b`", "`w`" (rojo, verde, azul o blanco, respectivamente).

Por ejemplo:

```
>>plot(z1,'o')
%Representa el símbolo o en las coordenadas del punto z1

>>z2=1+2+2*z1
%Definición de un segundo número complejo


>>plot(z2,'x')
%Representa el símbolo x en las coordenadas del punto z2
```

Para representar conjuntamente los dos gráficos en uno sólo podemos utilizar la sentencia `hold on`:

```
>>hold on
%Congela un gráfico para permitir la superposición

>>plot(z1,'x')
%Superpone el gráfico de z1 al de z2
```

Es preciso tener en cuenta que los ejes de la representación son los del primer gráfico dibujado, y si un punto queda fuera de los ejes de la representación, simplemente no aparecerá en la pantalla.

```
>>z3=1/z1
>>plot(z3,'+')
%Representa z3 superpuesto a z2 y z1
```

Ahora `z3` no ha aparecido en el gráfico por quedar fuera de los márgenes de la representación. Para anular la superposición de gráficos se opera con la orden

```
>>hold off
```

Para imponer unos ejes concretos a la representación se emplea la función `axis`.

```
>>axis([-1 4 -2 4])
%Fija unos ejes adecuados a la representación

>>plot(z1,'o')
%Representa el símbolo "o" en las coordenadas del pto z1

>>hold on
%Congela un gráfico para permitir la superposición

>>plot(z2,'x')
%Representa el símbolo "x" en las coordenadas del punto z2

>>plot(z3,'+')
%Representa z3 superpuesto a ambos z2 y z1

>>hold off
%Desactiva el sistema de superposición de gráficos
```

Una forma alternativa para representar un conjunto de números complejos es construir un vector complejo cuyo valor de las coordenadas sea el de los valores complejos a representar. Una manera de hacerlo es asignar cada valor a una componente del vector:

```
>>z(1)=z1;z(2)=z2;z(3)=z3;
%Creación del vector z de orden 1x3
```

Otra forma diferente de definir el vector es:

```
>>z=[z1 z2 z3]
%Creación    del vector z de orden 1x3

>>plot(z)
%Representa las coordenadas de z unidas por líneas

>>plot(z,'o')
%Representa el símbolo "o" para cada coordenada del vector
```

MATLAB permite además obtener la parte real, parte imaginaria, módulo y argumento de un número complejo mediante las funciones predefinidas: `real`, `imag`, `abs`, `angle`, respectivamente.
También incorpora la función `conj` que calcula el complejo conjugado.

Es posible generar un vector de números complejos mediante:

```
>>V_real=1:10              %Parte real del vector
>>V_imaginaria=1:10        %Parte imaginaria del vector
>>complex=V_real+V_imaginaria*j
>>rcmp=real(complex)       %Componente real del vector complex
>>icmp=imag(complex)       %Componente imaginaria del vector complex
>>mcmp=abs(complex)        %Módulo del vector complex
>>fcmp=angle(complex)      %Argumento del vector complex
>>ccmp=conj(complex)       %Vector complex conjugado
```

Para operar con matrices elemento a elemento, uno a uno, se emplean los símbolos aritméticos básicos precedidos de un punto: `.+`, `.-`, `.*`, `./`, `.^`.

```
>>cmp1=mcmp.*exp(j*fcmp)
%Recupera el complejo a partir del módulo y el argumento
```

Genere una matriz cuadrada cualquiera y calcule:

```
>> C=A.^2       % Cada valor de la matriz se eleva al cuadrado
>> D=A^2        % El elemento 1 es el resultado de multiplicar cada
                valor de la primera fila por cada valor de la
                primera columna y sumarlos, o sea, A*A.
```

Por otra parte, como ya hemos avanzado, los dos puntos "`:`" se utilizan para indicar un margen de valores. Así por ejemplo, si se quiere generar un vector cuyas componentes sean números enteros desde el 1 al 10 se hace simplemente:

```
>>lista=0:10
```

El primer valor será el primero de la lista, el segundo el último. Por defecto el incremento es la unidad. Si se quiere hacer una lista de los números pares entre 0 y 10, podemos asignar un incremento de dos unidades a la instrucción anterior, intercalando el incremento correspondiente entre la primera y la última componente del vector:

```
>>pares=0:2:10            %Incremento de 2
>>decrece=10:-1:0         %Incremento de -1
```

1.1.3. Operadores aritméticos y lógicos

Los símbolos aritméticos: `+`, `-`, `*`, `^`, se emplean respectivamente como operadores para sumar, restar, multiplicar y elevar una cantidad a una potencia. Las matrices que se sumen o resten deben tener la misma dimensión. Mientras que para la multiplicación de `A*C`, el número de filas de `A` debe coincidir con el número de columnas de `C`. Para la potenciación, la matriz debe ser cuadrada y el exponente debe ser un número positivo real o entero.

Las barras de división a izquierda y derecha, `A\C` y `C/A`, corresponden formalmente a las multiplicaciones izquierda y derecha de `C` por la inversa dela matriz `A`, esto es $A^{-1}*C$ y $C*A^{-1}$. O dicho de otro modo, el operador división se define como la matriz solución, `X`, de la ecuación `A*X=C`, para la división izquierda `A\C`, y de `X*A=C`, para la división derecha `C/A`.

Como ya hemos visto, también es posible efectuar la multiplicación o la división elemento a elemento de dos matrices. Para ello sólo hay que utilizar los símbolos `.*` (multiplicación), `.\` (división izquierda), y `./` (división derecha).

Introduzca en MATLAB por teclado dos matrices `A` y `C` y pruebe a efectuar con ellas todas las operaciones aritméticas que se han presentado.

Cuatro son los operadores lógicos que tiene MATLAB, el operador AND: `&`, el operador OR: `|`, el operador NOT: `~`, y el operador OR EXCLUSIVO: `xor(a,b)`.
Su utilidad radica en que permiten aunar o modificar operadores relacionales conforme a las reglas del álgebra de Boole.

Por ejemplo, `5>4 & 6>4` será cierto (TRUE),
mientras que `1==2 | 2==3` será falso (FALSE).

Al responder a una operación lógica, MATLAB asigna 1 a los resultados TRUE y 0 a los FALSE.

En el siguiente apartado, dedicado a los bucles, se entenderá mejor su uso en MATLAB.

1.1.4. Bucles

MATLAB permite ejecutar repetidamente una serie de instrucciones controlando el momento en que se desea terminar esta operación. Es lo que se conoce por el nombre de bucles. MATLAB tiene dos tipos de bucles: la sentencia `FOR` y la sentencia `WHILE`. La forma general de una sentencia `for` es:

```
for  variable=expr
        declaración;
        ........
        declaración;
end
```

En la descripción anterior expr hace referencia a una expresión aritmética, usualmente un vector del tipo `j:k`, mientras que variable es una variable convencional de MATLAB cuyo contenido puede usarse dentro del bucle. En la sentencia `for` expuesta, la *variable* va tomando a cada paso el contenido de *expr*, ejecutándose todas las declaraciones que figuran hasta la palabra `end`. El bucle se lleva a efecto tantas veces como diversos valores pueda tomar *expr*.

El comando `for` permite entre otras cosas, generar matrices de modo distinto al visto en el apartado anterior. Concretamente, el siguiente ejemplo muestra cómo generar una matriz de Hilbert de 5 por 5 :

```
n=5;                        %Declara el tamaño de la matriz
for i = 1:n                 %para las filas 1 a la n
     for j= 1:n             %y para las columnas 1 a la n
       A(i,j)=1/(i+j-1);    %Crea elemento(i,j)de la matriz
     end
end
```

La forma general de una sentencia `while` es:

```
while  exprl relop expr2
        declaración;
        ..........
        declaración;
end
```

En esta sintaxis, se ha llamado *exprl y expr2* a sendas expresiones aritméticas y *relop* a uno de los siguientes operadores relacionales: == (igual que), < (menor que), > (mayor que), <= (menor o igual que), >= (mayor o igual que) o ~= (distinto a).
En un bucle `while` todas las declaraciones contenidas hasta la palabra `end`, se ejecutan repetidamente mientras que *exprl y expr2* estén relacionadas de forma verdadera. En el siguiente ejemplo, MATLAB pide que se introduzca un número hasta que éste esté comprendido entre 1 y 5 ambos inclusive:

```
A=0
while A<1 | A>5
A=input ('Para salir teclee un numero del 1 ... al 5:  ');
end
```

1.1.5. Caminos condicionados

Una vez que un programa MATLAB comienza su ejecución, es posible hacer que ejecute cosas distintas conforme a alguna condición establecida. Es lo que se conoce como sentencia IF -ELSE. La forma general de una sentencia if-else es la siguiente:

```
if expr1 relop exp2
        declaración A;
        …
        decleración A;
else
        declaración B;
        …
        declaración B;
end
```

En la sintaxis expuesta, *exprl, expr2 y relop*, tienen el mismo significado que tenían cuando se explico la sentencia `while`. Si *exprl* está relacionada de forma cierta con *expr2*, entonces se ejecutan las declaraciones etiquetadas con A; y, en caso contrario, se ejecutan las declaraciones etiquetadas con B.

En la sentencia if-else la palabra `else` puede ser omitida. Una variante de la sentencia if-else es la if-elseif-else que permite anidar bucles condicionales dentro de otros bucles condicionales tantas veces como se desee. El siguiente, es un ejemplo de uso de la sentencia if-elseif-else:

```
if I==J
    A(I,J) =2;
elseif abs(I-J) == 1;
    A(I,J)=-1;
else
    A(I,J)=0;
end
```

1.1.6 Formatos de presentación de resultados de MATLAB

Para presentar los resultados numéricos por pantalla, MATLAB posee diversos formatos. Los más relevantes para nosotros son:

short	Punto fijo con 5 dígitos.
long	Punto fijo con 15 dígitos.
short e	Punto flotante con 5 dígitos.
long e	Punto flotante con 15 dígitos.
hex	Hexadecimal.
+	Imprime el signo de los resultados. Ignora la parte imaginaria.
rat	Formato fraccional de presentación de los resultados.

Todos los cálculos en MATLAB se realizan con precisión de tipo doble aunque luego la presentación de los resultados no lo muestre. El comando `format` permite cambiar el formato de presentación de un tipo a otro de los mostrados arriba; para ello, hay que ejecutar `format` seguido de una de las palabras clave que acaban de verse.

1.1.7 Variables reservadas de MATLAB

Aparte de las variables que el usuario de MATLAB pueda declarar para su uso personal, existen una serie de ellas cuya definición y manejo corren por cuenta de MATLAB. Ello no significa que no puedan modificarse, sino que si se toma la determinación de alterar su valor, perderán el propósito para el que fueron diseñadas. Algunas de estas variables son:

eps Es una variable de tolerancia usada en varios cálculos, por ejemplo, rango y singularidad. Su valor inicial es el número real positivo más pequeño disponible en el ordenador, pero puede ser fijado a otro valor cualquiera.

realmax Es el número más grande que puede computarse en punto flotante. Cualquier otro mayor produce desbordamiento.

realmin Es el número más pequeño que puede computarse en punto flotante.

NaN Not a Number. Es la representación IEEE que el MATLAB da a resultados de operaciones indeterminadas como 0/0.

inf Es una variable permanente en formato IEEE que representa una infinidad positiva, como por ejemplo 1/0.

pi Número π.

Medidas de tiempos y de esfuerzo de cálculo.

MATLAB dispone de funciones que permiten calcular el tiempo empleado en las operaciones matemáticas realizadas. Algunas de estas funcione son las siguientes:

cputime Devuelve el tiempo de CPU (con precisión de centésimas de segundo) desde que el programa arrancó. Llamando antes y después de realizar una operación y restando los valores devueltos, se puede saber el tiempo de CPU empleado en esa operación. Este tiempo sigue corriendo aunque MATLAB esté inactivo.

etime(t2,t1) Tiempo transcurrido entre los vectores t1 y t2 obtenidos como respuesta al comando clock.

tic ops toc Imprime el tiempo en segundos requerido por ops. El comando tic pone el reloj a cero y toc obtiene el tiempo transcurrido.

Por ejemplo, el siguiente código mide de varias formas el tiempo necesario para resolver un sistema de 500 ecuaciones con 500 incógnitas. Téngase en cuenta que los tiempos pequeños (del orden de las décimas o centésimas de segundo), no se pueden medir con gran precisión.

```
>>A=rand(500); b=rand(500,1); x=zeros(500,1);
>>tiempo=clock; x=A\b; tiempo=etime(clock, tiempo)
>>time=cputime; x=A\b; time =cputime-time
>>tic; x=A\b; toc
```

Observe que se han puesto varias sentencias en la misma línea para que se ejecuten todas sin tiempos muertos al pulsar la tecla intro. Esto es especialmente importante en la línea de comandos en la que se quiere medir los tiempos. Todas las sentencias de cálculos matriciales van seguidas de punto y coma con objeto de evitar la impresión de los resultados.

1.1.8. Otros comandos de MATLAB

help Da una breve descripción de diversas características de MATLAB. Si help se antepone a un nombre de función, devuelve una breve ayuda sobre ésta.

demo Muestra un menú con las demostraciones disponibles.

clear Borra todas las variables excepto las reservadas de MATLAB. `Clear x` borra únicamente la variable `x`.

who Lista las variables actualmente utilizadas.

whos Muestra las variables actualmente utilizadas así como su tamaño.

what Lista los comandos y funciones actualmente disponibles en el directorio de trabajo del disco.

dir Da una relación de los ficheros que hay en el directorio actual.

echo Reproduce por consola el texto íntegro de la línea de comandos que se ejecuta en cada instante.

chdir Cambia el directorio actual por el que se indique a continuación de chdir.

type Muestra el contenido de cualquier fichero ASCII. Si no se indica extensión, se toma **m** por defecto.

quit o exit Finaliza la sesión de trabajo con MATLAB.

why Siempre tiene respuesta para cualquier problema.

; Al final de una declaración inhibe la salida por pantalla del resultado.

... Indica que la declaración continua en la siguiente línea física.

% Indica que el resto de la línea es un comentario.

! Invoca a un comando del sistema operativo.

1.2 Modo de trabajo y ficheros editados por el usuario

El sistema MATLAB permite editar en un fichero la secuencia de instrucciones que se quiere que sean ejecutables (scrip-file). Para ello ha de crear y editar un fichero de extensión .M. Por ejemplo, crear para cada uno de los ejemplos del apartado siguiente un fichero con un nombre determinado (ejemplo.m); este fichero deberá situarse en un directorio accesible desde MATLAB (o añadir el directorio donde lo hemos almacenado en el path) y bastará teclear

```
>>ejemplo
```

desde la línea de comando de MATLAB para ejecutar su contenido.

Se puede utilizar cualquier editor que permita crear un fichero de caracteres. En cambio, no es válido un procesador de textos que guarde informaciones de formato, tipos de letra, etc., ya que los correspondientes caracteres de control serán mal interpretados por MATLAB. En general usaremos el propio editor de texto de MATLAB.

1.3 Algunos ejemplos sencillos con MATLAB

A parte de los ejemplos que se expondrán seguidamente, es muy recomendable ejecutar las demostraciones de que dispone MATLAB. Para ello, lo único que hay que hacer es escribir `demo` en la línea de comandos y pulsar la tecla de retorno de carro. Tras esto aparecerá un menú con todas las demostraciones disponibles. Para moverse por ellas basta con seguir las explicaciones que MATLAB va dando.

A continuación, se dan dos ejemplos ilustrativos de programas escritos para MATLAB. Es aconsejable que el alumno intente razonar cómo funcionan y el efecto que produce cada uno de estos programas antes de pasar a ejecutarlos.

Ejemplo 1. El ejemplo siguiente es una función que demuestra cómo se pueden calcular medias estadísticas. Si la variable de entrada es un vector, calcula la media y la desviación típica de todos sus elementos. Mientras que, si la variable de entrada es una matriz, obtiene un vector fila con los valores medios y las desviaciones típicas de cada columna. Asimismo, este ejemplo sirve para ilustrar cual es el formato típico de un fichero de función tipo M.

```
function [mean,stdev] = stat(x)
% Declaración de una función

% Esta función calcula la media y la desviación típica
%de todos los elementos de la entrada si
%la variable de entrada es un vector.
%Si la variable de entrada es una matriz,
%obtiene un vector fila con los valores medios
%y las desviaciones típicas de cada columna.

[m,n]=size(x);
if m==1
     m=
     n;
end
mean=sum(x)/m;
stdev=sqrt(sum(x.^2)/m-mean.^2);
```

Para ejecutar el programa, crear un vector `x`, por ejemplo `x=[4.5 6 7 3 10]`, y teclear en la línea de comandos de MATLAB: `[m,d] = stat (x)`. El resultado que debe obtenerse es `m = 6.1` y `d = 2.3749`.

Ejemplo 2. El presente ejemplo muestra un problema de teoría de números. Consiste en lo siguiente: cojamos cualquier número entero positivo; si es par, se divide por dos; si es impar, se multiplica por 3 y se le suma 1 al resultado. Así se va repitiendo el proceso hasta que se consiga un número entero igual a uno. Lo atractivo del problema, aún no resuelto, es lo siguiente: ¿Existe algún entero para el cual el proceso no termine nunca? El siguiente programa en MATLAB ilustra el empleo de los comandos `while` y `if`. También muestra cómo la función `input` permite introducir un número desde el teclado y como el comando `break` permite salir de un bucle:

```
%Problema "3n+1" clásico de teoría de números

while 1
   n=input ('Introduzca un número positivo:(para
   ... valores negativos finaliza) ');
   if n<=0,
      break,
   end
   while n>1
      if rem(n,2)==0
           n=n/2;
      else n=3*n+1;
      end;
   n  %permite ver por pantalla la evolución de n
   end;
end;
```

Este programa representa un buen ejemplo de un fichero de tipo script. Para ejecutarlo solo hay que teclear su nombre en la línea de comandos de MATLAB. El nombre que se le puede dar al programa puede ser **tres.m.**