Programación en Matlab

Vamos a poder implementar nuestros propios algoritmos programando con Matlab. La estructura de dichos programas propios será la habitual

Programa Principal + Rutinas

Como es usual, en el programa principal presentaremos al usuario nuestro algoritmo, introduciremos los datos necesarios (bien por teclado o bien leyéndolos de un fichero), organizaremos la computación del algoritmo llamando a las rutinas necesarias y finalmente devolveremos al usuario los resultados solicitados.

En muchos casos se puede usar un fichero-m de tipo script para almacenar el programa principal. Pero es recomendable hacerlo con ficheros-m de tipo función (sin argumentos) cuando necesitemos usar subfunciones y funciones anidadas (*en versiones de Matlab 7 o superiores*) como ya veremos.

Es evidente que todas las rutinas van a almacenarse siempre en ficheros-m de tipo función.

VARIABLES Y OPERADORES LÓGICOS

Matlab tiene un tipo de datos lógicos. El 1 representa lo cierto (true) y el 0 lo falso (false):

```
a=true
b=false
```

Los operadores relacionales en Matlab son

==	igual
~=	no igual
<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que

La comparación entre escalares produce un resultado lógico 1 si la relación es cierta y 0 si es falsa. La comparación entre matrices de la misma dimensión produce una matriz lógica cuyos elementos son 1's ó 0's según las comparaciones componente a componente sean ciertas o falsas. Cuando se compara una matriz y un escalar se entiende que se compara cada elemento de la matriz con el escalar y produce una matriz lógica de 1's y 0's.

Estos operadores relacionales se pueden combinar con los operadores lógicos

&	y lógico
&&	y lógico con "corto-circuito" (entre escalares)
|	ó lógico
||	ó lógico con "corto-circuito" (entre escalares)
~	no lógico

xor	o lógico exclusivo
all	cierto si todos los elementos de un vector son no nulos
any	cierto si algún elemento de un vector es no nulo

```
x=[-1,1,1]; y=[1,2,-3];
x>0
x>0 & y>0
x>0 | y>0
xor(x>0,y>0)
any(x>0)
all(x>0)
```

Los operadores && y || son especiales. Por una parte sólo trabajan entre escalares, y deben usarse más que & y | entre escalares. Por otra parte, cortan la evaluación de la expresión lógica que sigue si la anterior es falsa:

expr1 && *expr2*: si *expr1* es falsa, no se evalúa *expr2*.
expr1 || *expr2*: si *expr1* es cierta, no se evalúa *expr2*.
Esto se suele usar para evitar divisiones por cero, por ejemplo:
if x>1.e-14 && sin(1/x)<0.5

COMANDO IF

IF más simple:

if *expr*
    sentencias a ejecutar si *expr* es cierto
end

Hay que entender que *expr* es cierto si todos los elementos de *expr* son 1. En otras palabras, si *expr* es una comparación matricial, tiene que ser ciertas todas las componentes.
Si las sentencias a ejecutar se ponen junto a *expr*, se tiene que separar por comas.

```
if x>0, x=sqrt(x); end
```

IF-ELSE:

if *expr*
    sentencias a ejecutar si *expr* es cierto
else
    sentencias a ejecutar si *expr* es falso
end

IF-ELSEIF-ELSE:

if *expr1*
    sentencias a ejecutar si *expr1* es cierto
elseif *expr2*
    sentencias a ejecutar si *expr2* es cierto
else
    sentencias a ejecutar si *expr1* y *expr2* son falsas
end

Como en cualquier otro lenguaje de programación, los if se pueden anidar unos dentro de otros, teniendo en cuenta que hay que cerrarlos con end apropiadamente.

Un comando alternativo más simple cuando queremos hacer un menú para que el usuario elija un caso de una lista de posibilidades, es el comando **switch** (ver ayuda de Matlab).

BUCLES FOR

```
for variable=expr
        sentencias a ejecutar
end
```

Normalmente *expr* es una expresión del tipo i:inc:j. Las sentencias se ejecutan cada vez que *variable* toma un valor diferente de *expr.*

```
s=0;
for i=1:25, s=s+1/i; end, s
```

También puede ser

```
s=0;
for x=[pi/6,pi/4,pi/3], s=s+sin(x); end, s
```

Y puede ser también una expresión matricial. Por ejemplo,

```
for z=eye(5), disp(z), end
```

donde z es vectorial y toma sucesivamente los valores de los vectores unitarios de dimensión 5.

Al igual que con los if, los for se pueden anidar unos dentro de otros cerrándolos de manera apropiada cada uno con su end correspondiente.

Cuando trabajamos con bucles es útil la sentencia

break

que hace que el programa salte del bucle y siga después de su end correspondiente.

Ligeramente diferente es el comando

continue

que lo que provoca es que el bucle salte a la siguiente iteración sin ejecutar ninguna sentencia más.

Por otra parte, si lo que queremos es acabar la ejecución de una función cuando, por ejemplo, se cumple una condición antes de llegar al final, se usa el comando

return

que vuelve a la rutina de llamada o a la sesión Matlab abierta.

BUCLES WHILE

while *expr*
sentencias a ejecutar mientras *expr* sea cierto
end

Ejemplo:
x=1; while x>0, xmin=x; x=x/2; end, xmin

Su principal diferencia respecto al for es que en este caso no tenemos que saber previamente cuántas vueltas ha de dar el bucle.
Para salir de ellos también se puede usar break o continue.

ENTRADA DE DATOS DESDE EL TECLADO Y SALIDA A PANTALLA

Comando input: entrada de datos más sencilla, un solo objeto a la vez.

Si es un escalar
n=input('Valor de n? ')

Si se pide una cadena de caracteres hay que introducirla entre apóstrofes
nf=input('Introduce el nombre del fichero de datos? ')
excepto si en el input se añade 's':
nf=input('Introduce el nombre del fichero de datos? ','s')

Si los datos son matriciales, hay que introducirlos como tales (entre corchetes)
v0=input('Introduce los dos valores iniciales x0,y0 ')

Comando disp: salida a pantalla más simple, un solo objeto a la vez que puede ser de cualquier tipo
A=[1,2,3; 3,4,5;7,8,9]
disp(A)
disp('A='), disp(A)

COMANDO FPRINTF: salida de datos con un **formato** específico:

fprintf('*formatos', lista de expresiones*)

El orden de los *formatos* se ha de corresponder con el orden de las expresiones a las que queremos dar formatos y van precedidos del símbolo %. Según los distintos tipos de expresiones tenemos diferentes formatos:

%d	enteros (sin punto decimal)
%*m.n***f**	reales sin exponente, *m*=anchura total, *n*=nº de decimales, *m>n*
%*m.n***e**	reales con exponente, *m*=anchura total, *n*=nº de decimales, *m>n+7*
%g	reales con/sin exponente: Matlab elige la forma más corta
%s	cadena de caracteres
\n	salto de línea

Dentro de los apóstrofes se pueden incluir “etiquetas” explicativas o de cualquier tipo.

```
fprintf('%d, %6.3f, número e =%18.9e\n, %g\n, %s\n',…
        4^5, pi^10, exp(1), exp(20), 'esto es una prueba')
```

Un comando que usaremos a veces (sobre todo en los gráficos) es

num2str(x)

que convierte el número real x en una cadena de caracteres. También está el análogo

int2str(n)

específico cuando el número es un entero.

FICHEROS DE SALIDA DE DATOS

Se usan también para la entrada de datos, pero no lo necesitaremos por ahora.

Primero hay que crearlos dándoles un nombre:

fid=**fopen**(*nombre*,'w')

ó

fid=**fopen**(*nombre*,'wt')

donde el *nombre* es una cadena de caracteres que almacena el nombre que tiene el fichero. La opción 'w' indica que solo vamos a escribir en él. Si dicho fichero se va a leer posteriormente desde un editor de textos en Windows, se debe poner la opción ‘wt’ para que se lea correctamente. Si hay un fichero con el mismo nombre se sobrescribirá.

Dicho *nombre* debe tener la **extensión .txt.** Además puede variarse si hacemos que lo dé el usuario desde el teclado.

Con esto se crea el fichero *nombre* y se le asocia la variable **fid**, que va a ser el valor con el que Matlab identificará dicho fichero.

Para escribir en él hay que usar el comando fprintf visto anteriormente pero añadiéndole que tiene que escribir en **fid:**

fprintf(**fid**, ‘*formatos’, lista de expresiones*)

Cuando se deja de usar el fichero, se cierra

fclose(**fid**)

Si no se cierra no se podrá acceder a él posteriormente.

Para leerlo basta con ir a File->Open, seleccionando la opción All Files para que aparezca en el listado del directorio.