Sentencias de control

Condiciones

Son las preguntas básicas a las que se puede responder sí o no
Para implementar el control del flujo, son muy útiles unas expresiones que permiten comparar dos variables entre sí o una variable con un valor fijo. En un programa, a veces es necesario preguntarse:*¿es x mayor que y? Pues si x es mayor que y, entonces ..., y si no, ...* ; o bien: *caso de que la variable x sea menor que cero, hacer tal cosa*.
Las condiciones se construyen con **operadores relacionales**, como son los siguientes:

> mayor que
< menor que
== igual que
~= diferente que
<= menor o igual que
>= mayor o igual que

Una condición, como se ha indicado, sólo admite dos respuestas: verdadero o falso. Así, a la pregunta ¿es x mayor o igual que y?, que en el lenguaje de Matlab se expresa x>=y, sólo caben dos respuestas posibles: sí, es cierto, x es mayor o igual que y; o bien, no, es falso, x no es mayor o igual que y.
Dos observaciones. La primera: el operador relacional *igual que* se construye con *dos* símbolos ==. El motivo es el siguiente: si se escribe x=3, el resultado es que se le asigna a x el valor 3. No se hace ninguna pregunta. Si lo que se desea es preguntar ¿es x igual a 3?, no se puede escribir x=3, pues en este caso x tomaría el valor 3, independientemente de lo que valiera antes. En cambio,x==3 es el análogo en lenguaje informático a la pregunta: ¿es x igual a 3?. Tras la pregunta x==3 la variable x sigue valiendo lo mismo que antes.La segunda observación es una pequeña astucia: la tilde ~ del operador *diferente que* se consigue apretando la tecla Alt y -sin soltarlaescribiendo
con el teclado el número 126. En algunos sistemas operativos, también
sirve apretar AltGr y, sin soltarla, apretar la tecla 4.
Las condiciones se pueden agrupar en construcciones lógicas: condición1 Y condición2, condición1 O condición2, etcétera. Consúltese el manual de Matlab en lo relativo a este tópico.
En resumen: con las expresiones relacionales se pueden realizar preguntas de respuesta unívoca (verdadero o falso), que permitan luego tomar una decisión.

Sentencia if

La ramificación más simple, expresada en este diagrama de flujo, se obtiene con la siguente sintaxis:

```
if (condición)
sentencias
end
```

(if y end son palabras clave del lenguaje informático, y no se pueden utilizar para otra cosa, p. ej. una variable no puede -no debería- llamarse if).
Un caso concreto:

```
if(length(sitios)>1)
recta=polyfit(x,y,1);
end
```

Leído en lenguaje corriente: si la longitud del vector sitios es mayor que 1, se realiza el ajuste lineal indicado en la instrucción recta=polyfit(x,y,1). Caso contrario (si la longitud del vector sitios es menor o igual a 1) esa instrucción no se ejecuta (y el programa sigue en la instrucción que venga después de end).
Existe la posibilidad de ejecutar ciertas sentencias si la condición es verdadera, y otras diferentes si la condición es falsa:

```
if (condición)
sentencias A
else
sentencias B
end
```

```
if (condición)
sentencias A
elseif (condición)
sentencias B
else
sentencias C
end
```

dicho de otra manera: si la condición se cumple, se ejecutan las sentencias A o B; si no, se ejecutan las sentencias C.

Sentencia switch

Otra posibilidad de ramificación múltiple la ofrece la construcción switch. La sintaxis es:

switch variable
case valor1,
sentencias A
case valor2,
sentencias B
case ...
...
end

Las palabras clave son switch, case, end.

La ramificación switch opera de la siguiente manera. Al llegar a la expresión switch variable, si variable tiene el valor valor1 se ejecutan las sentencias A; si variabletoma el valor valor2, las sentencias B; y así sucesivamente. Es importante notar que la variable sólo debe tomar unos pocos valores: valor1, valor2, etc. para que el programa se ramifique en unas pocas ramas. No tiene sentido intentar una ramificación switch con una variable que pueda tomar un número infinito de valores.

Sentencia for

Hay ocasiones en las que es necesario repetir el mismo conjunto de instrucciones muchas veces, cambiando algunos detalles. Pongamos un caso. Sea un vector x(i) con n componentes; se quiere construir la "media móvil" de x con tres elementos, que consiste en ir tomando la media aritmética de cada tres puntos consecutivos. Es decir: desde i=2 hasta n-1, media(i-1)=(x(i)+x(i-1)+x(i+1))/3.
(Detalles: se empieza a contar en i=2 porque para el primer elemento de x no existe el elemento anterior; y se acaba en n-1 por análoga razón; además, el primer componente de media es el correspondiente a i=2, de ahí que se asigne el resultado a media(i-1)).

Eso es lo que se consigue con un bucle for, cuya sintaxis es:

for contador=inicio:paso:fin,
sentencias
end

Las palabras claves son for y end. Este bucle pone en marcha una variable llamada contador que va desde inicio hasta fin de paso en paso. Cada vez que las sentencias se ejecutan, contador aumenta en un valor paso (que si se omite, se le asigna automáticamente el valor 1). Cuando contador llega al valor fin, el bucle se acaba y el programa continúa con las sentencias que haya más allá de end.

Sentencia while

Obsérvese que un bucle como el indicado se implementa un número fijo de veces: desde inicio hasta fin de paso en paso. En ocasiones, sin embargo, no se sabe de antemano cuántas veces habrá que ejecutar las sentencias del bucle. Por ejemplo: si es necesario repetir una serie de sentencias hasta que se cumpla una determinada condición, y no se sabe a priori cuántas veces será necesario realizar esas operaciones.

En ese caso se emplea un bucle while:

```
while(condición),
sentencias
end
```

Es posible sustituir la condición por una variable. En efecto: una variable que toma el valor cero corresponde a una condición falsa. Si la variable toma un valor diferente de cero, es equivalente a una condición verdadera. Así, se puede escribir

```
x=10;
while(x)
sentencias
x=x-1;
end
```

Para x=10, la "condición" es verdadera puesto que x es diferente de cero. Nótese que el contador x hay que modificarlo manualmente (línea x=x-1) puesto que, al revés que lo que ocurre con el bucle for, este no gestiona ningún contador. En cuanto x tome el valor cero, la "condición" es falsa y el bucle acaba.
Atención: es fácil caer en bucles infinitos. En el ejemplo anterior, si falta la línea x=x-1 y las sentencias no modifican el valor de x, la "condición" siempre será cierta (pues x=10) y el programa nunca saldrá del bucle: ejecutará una y otra vez las sentencias. El programa se "cuelga", y hay que interrumpirlo desde el teclado apretando las teclas Ctrl+C.

Resultados y arreglos

Los resultados pueden ser fácilmente mostrados en pantalla, pero en ocaciones resulta útil almacenar la información.
Es comun obtener varios valores por lo que es importante capturar la respuesta en arreglos. (conjunto de datos)
Un puede ser fácilmente definible como >> y = [1 2 3]
Ahora podemos hacer >> 2*y

Recuerde que matlab toma todo como arreglos y matrices. En el caso anterior la constante multiplica todos los elementos del arreglo.

Para acceder a un elemeneto puntual del arreglo debemos usar >> y(1)
En este caso mostrara el elemento 1 del arreglo 1. Pruebe a ver otros elementos.

Ahora veamos un ejemplo en el que los resultados de un bucle son almacenados en un arreglo. Adicionalmente usamos la función “rand” que genera un numero aleatorio y la función zeros que genera una matriz con todos sus elementos en zeros.
funccion rand que genera números aleatorios

h= rand(a,b) % genera un numero aleatorio entre a y b
función zeros
h=zeros(a,b) % genera un arreglo axb y todos sus elementos son cero
Ejemplo for almacenando en cada casilla del arreglo

```
N=10;
x=rand(1,N);
y=zeros(1,N);
for i=1:N,
y(i)=sum(x(1:i));
end
```

Ejercicios

1. Escribir un programa que tome un número entero al azar entre el 0 y el 9, pregunte un número al usuario y le informe si acertó o no.
2. Escribir un programa que tome un número entero al azar entre el 0 y el 9, y le pregunte un número al usuario hasta que acierte.
3. Escribir una función que devuelva el valor absoluto del argumento. (Nota.- Ya existe en Matlab: abs(x)).
4. Escribir un programa que, tras pedir al usuario un número, le informe de si es par, impar o no entero.
5. Escribir un programa que calcule las N primeras fracciones del tipo 1/i, tras pedir N al usuario.
6. Escribir un programa que calcule los cuadrados de los números enteros, hasta que el cuadrado sea mayor o igual que 100.