Tema 9

Programación en MATLAB

MATLAB es una herramienta informática que surgió para realizar cálculos matemáticos, especialmente operaciones con matrices. El usuario de MATLAB suele ser una persona que necesita algo más que una calculadora pero que no quiere "mancharse las manos" con un lenguaje de programación. Por eso el entorno de trabajo es sencillo de manejar, casi tan fácil como una calculadora.

Además de realizar cálculos, esta herramienta permite crear gráficos de muchos tipos y presenta grandes ventajas a la hora de trabajar con números complejos, con matrices, con polinomios, con funciones trigonométricas, logaritmos, etc.

Con los años la herramienta ha sido modificada pasando por varias versiones. En cada versión se han incorporado otras funciones distintas y hoy en día proporciona funciones para gran cantidad de aplicaciones ingenieriles: simulación de sistemas dinámicos, visión artificial, análisis estadístico, análisis y diseño de controladores automáticos, etc.

Desde las primeras versiones MATLAB incorpora una característica muy interesante: la capacidad para programar. En efecto, es posible crear archivos que contengan las operaciones que se desean realizar. Además es posible incorporar nuevas funciones de MATLAB realizaedas por el propio usuario.

La programación se lleva a cabo mediante un lenguaje que es muy parecido a lenguajes de alto nivel como BASIC o C. Esto permite que el usuario pueda agrupar sentencias que utiliza frecuentemente dentro de un programa que puede ser invocado posteriormente. De este modo se ahorra tiempo y esfuerzo en sucesivas sesiones pues no es necesario escribir todas las sentencias de nuevo como se mostrará.

Pero no todo son ventajas. Como principal inconveniente hay que señalar el hecho de que

MATLAB no ha sido concebido como lenguaje de programación por lo que carece de algunos elementos o características necesarias para una buena práctica de la programación. Para aliviar estos defectos se van a usar métodos que eviten los peligros derivados de dichas carencias.

9.1 El entorno

El programa MATLAB se maneja (en su mayor parte) escribiendo sentencias dentro de una ventana llamada *de órdenes*. Al arrancar el programa aparecen varias ventanas, pero en una primera toma de contacto es mejor cerrar todas excepto la principal, que es la ventana de órdenes.

Las órdenes se escriben una a una (ya se verá la forma de escribir conjuntos de órdenes o programas más adelante) pulsando la tecla de retorno al final. Por ejemplo, si se escribe `sqrt(16)` el programa realiza la operación indicada y responde en la pantalla con el resultado. Lo que se aprecia en la ventana de órdenes será algo como:

```
>> sqrt(16)
ans =
    4.0000
>>
```

Las operaciones se indican de forma muy intuitiva, por ejemplo para obtener el resultado de $\frac{13\cdot 5}{3+4\cdot 11}$ basta con escribir `13*5/(3 +4*11)`. En la ventana de órdenes se obtendrá lo siguiente:

```
>> 13*5/(3 + 4*11)
ans =
    1.3830
```

Observe que es necesario utilizar paréntesis para que la operación se lleve a cabo correctamente. Esta y otras particularidades del entorno de MATLAB serán comentadas a lo largo del tema.

9.2 Objetos

La herramienta MATLAB permite utilizar nombres simbólicos para referirse a objetos los cuales representan números, vectores o matrices. Como es sabido, en cualquier lenguaje de progra-

mación, los objetos se caracterizan por un tipo, un identificador y otras características. A continuación se describen estas características para el entorno de MATLAB.

9.2.1 Tipos

En MATLAB todos los objetos son matrices de números complejos en punto flotante. Un escalar no es más que una matriz 1×1. Cada elemento de la matriz se almacena en la memoria como un número complejo $a + bi$ siendo $i = \sqrt{-1}$. Cuando se trabaja con números reales MATLAB considera simplemente que $b = 0$. Finalmente, todos los números son tratados en formato de punto flotante. Esto quiere decir que son números con parte entera, parte decimal y un exponente (véase tema 2).

Estas características hacen que MATLAB desperdicie memoria con respecto a otros lenguajes en los cuales las variables pueden ser de otros tipos más pequeños. A pesar de que MATLAB trata a todas las variables del mismo modo se va a continuar con la sana costumbre de indicar en los diagramas de flujo y en los comentarios el tipo de las variables. De este modo se facilita la detección de errores.

9.2.2 Identificadores

Los objetos que se pueden utilizar en MATLAB han de tener un identificador que es un conjunto de caracteres. La palabra resultante ha de cumplir ciertos requisitos:

- No puede comenzar por un número
- No puede coincidir con palabras reservadas como `for`, `if`, etc.
- No puede contener espacios en blanco

9.2.3 Creación y destrucción

Los objetos pueden crearse en cualquier momento. Para ello basta con asignarles un valor en la forma que se indicará más adelante.

Hay que tener en cuenta que los objetos ocupan un espacio en la memoria. Cada vez que se crea una nueva variable el programa MATLAB pide al SE un trozo más de la memoria disponible. A partir de ese momento el trozo en cuestión no puede ser usado para otros fines.

Si se quiere liberar el espacio ocupado por una variable se ha de utilizar la orden `clear`, por ejemplo `clear k` o `clear matriz_a`. Esto causa la destrucción completa de la variable y la liberación del trozo de memoria ocupado por la misma. Una vez que una variable ha sido eliminada no es posible recuperar su valor.

9.2.4 Asignación

La asignación se consigue por medio del signo igual =. No hay que olvidar que asignar es una operación consistente en copiar un cierto valor en la memoria ocupada por una variable. Por tanto no es lo mismo asignar que igualar en el sentido matemático. Lamentablemente el uso del signo = en ambos casos no ayuda nada a clarificar la cuestión.

Debido a que la asignación es una operación distinta de la igualdad matemática se ha insistido en temas anteriores en que se utilice un signo diferente ←. En los diagramas de flujo se mantendrá el uso del símbolo ← para la asignación.

9.3 Operaciones y funciones incorporadas

La mejor manera de aprender a realizar operaciones con MATLAB es probar el programa con ejemplos simples y explorar su comportamiento. En este punto se van a mostrar ejemplos en varias categorías de dificultad creciente. Es muy aconsejable no estudiar de memoria los ejemplos sino probarlos en el entorno MATLAB.

9.3.1 Operaciones elementales

La forma más simple de usar MATLAB es como una calculadora. En lugar de pulsar teclas se ha de escribir la operación a realizar y pulsar Intro para que ésta se lleve a cabo.

Por ejemplo si se escribe

```
6+8
```

y se pulsa Intro se obtiene en la pantalla:

```
>> 6+8
```

```
ans =
    14
>>
```

Puede verse que el resultado viene precedido de `ans=`. Estas letras son la abreviatura de *answer* (respuesta, resultado). Además, como ya se ha dicho, el símbolo `>>` sirve para indicar al usuario que puede escribir ahí su orden. A menudo recibe el nombre de símbolo inductor pues induce al usuario a escribir.

El programa MATLAB admite operaciones con paréntesis, como por ejemplo $2 \cdot (5 + 3)$.

```
>> 2*(5+3)
ans =
    16
>>
```

Nótese que la multiplicación se indica por medio del asterisco *. También es posible realizar operaciones menos elementales, como por ejemplo logaritmos en base 10

```
>> log10(100)
ans =
     2
```

Si uno desea dar nombres simbólicos a los números que ha de manejar debe crear variables. Para ello basta con asignar un valor mediante una expresión del tipo:

```
 nombre_de_la_variable = valor_que_se_asigna
```

por ejemplo:

```
x=22.3
```

el programa responde con un mensaje que indica que se ha creado la variable y que ha tomado el valor adecuado

```
>> x=22.3
x =
   22.3000
```

Es importante saber que estas expresiones hacen dos cosas: crear la variable (tomando cierta cantidad de memoria para almacenar su valor) y dar un valor inicial a la variable.

Esta facilidad para crear variables conlleva algunos peligros que trataremos de minimizar mediante una programación cuidadosa.

Además de las operaciones aritméticas de sobra conocidas existen operaciones lógicas, como por ejemplo la comprobación de igualdad. Considere el ejemplo siguiente consistente en una expresión de comprobación de igualdad:

```
>> x==7
ans =
     0
```

La expresión $x == 7$ equivale a preguntar ¿Es el valor de x igual a 7?. La respuesta obtenida es cero, lo cual indica que x no es igual a 7. De forma equivalente la sentencia $x == 22.3$ produce un valor 1 al ser evaluada.

```
>> x==22.3
ans =
     1
```

De estos ejemplos se desprende que las comprobaciones que el valor lógico *verdadero* es representado en MATLAB mediante el valor numérico "1", mientras que el valor lógico *falso* es representado por "0".

Las operaciones que dan como resultado un valor lógico se pueden combinar utilizando la suma lógica (operación "o"), el producto lógico (operación "y") y la negación (operación "no"). Por ejemplo, para saber si $x < 100$ y al mismo tiempo $x > 5$; es decir, para comprobar si $5 < x < 100$ se usará la expresión `x<100 & x > 5`

```
>> x<100 & x>5
ans =
     1
```

Como era de esperar el resultado es 1. Eso quiere decir que es cierto que "x es menor que 100 y mayor que 5".

Las operaciones que se han comentado se indican en la tabla siguiente de forma más resumida.

Expresión en MATLAB	Operación
+	suma aritmética
-	resta aritmética o cambio de signo
*	multiplicación aritmética
/	división
<	relación "menor que"
>	relación "mayor que"
<=	relación "menor o igual que"
>=	relación "mayor o igual que"
==	relación "igual que"
∼=	relación "distinto que"
&	producto lógico (operación "y")
|	suma lógica (operación "o")
∼	negación (operación "no")

No es preciso aprender ahora mismo de memoria esta tabla. El uso frecuente debería bastar para aprenderla por lo que se recomienda que se practiquen los ejemplos indicados y se exploren las posibilidades que ofrecen.

9.3.2 Funciones incorporadas

De todas las órdenes de MATLAB ninguna debiera ser más útil para el usuario aprendiz que la función de ayuda. Se comentará más adelante como usar la función `help`, de momento se va a usar ahora para conocer la lista de funciones "elementales".

```
>> help elfun
```

El resultado de esta orden es una larga lista de funciones que se muestra en la tabla 9.1.

La forma de uso de estas funciones es simple e intuitiva, por ejemplo `a = sin(1)` asigna a la variable `a` el valor del seno de un radián, o sea sen(1).

```
>> a = sin(1)
a =
    0.8415
```

Trigonometric.	
sin	Sine.
sinh	Hyperbolic sine.
asin	Inverse sine.
asinh	Inverse hyperbolic sine.
cos	Cosine.
cosh	Hyperbolic cosine.
acos	Inverse cosine.
acosh	Inverse hyperbolic cosine.
tan	Tangent.
tanh	Hyperbolic tangent.
atan	Inverse tangent.
atan2	Four quadrant inverse tangent.
atanh	Inverse hyperbolic tangent.
sec	Secant.
sech	Hyperbolic secant.
asec	Inverse secant.
asech	Inverse hyperbolic secant.
csc	Cosecant.
csch	Hyperbolic cosecant.
acsc	Inverse cosecant.
acsch	Inverse hyperbolic cosecant.
cot	Cotangent.
coth	Hyperbolic cotangent.
acot	Inverse cotangent.
acoth	Inverse hyperbolic cotangent.
Exponential.	
exp	Exponential.
log	Natural logarithm.
log10	Common (base 10) logarithm.
log2	Base 2 logarithm and dissect floating point number.
pow2	Base 2 power and scale floating point number.
realpow	Power that will error out on complex result.
reallog	Natural logarithm of real number.
realsqrt	Square root of number greater than or equal to zero.
sqrt	Square root.
nextpow2	Next higher power of 2.
Complex.	
abs	Absolute value.
angle	Phase angle.
complex	Construct complex data from real and imaginary parts.
conj	Complex conjugate.
imag	Complex imaginary part.
real	Complex real part.
unwrap	Unwrap phase angle.
isreal	True for real array.
cplxpair	Sort numbers into complex conjugate pairs.
Rounding and remainder.	
fix	Round towards zero.
floor	Round towards minus infinity.
ceil	Round towards plus infinity.
round	Round towards nearest integer.
mod	Modulus (signed remainder after division).
rem	Remainder after division.
sign	Signum.

Tabla 9.1: Funciones elementales de MATLAB

9.3.3 Vectores

Los vectores se introducen en MATLAB como una colección de valores. Por ejemplo un vector fila es

```
>> v=[ -1 2 -3 4 -5]
v =
    -1     2    -3     4    -5
```

Para acceder a las componentes individuales del vector se usan paréntesis indicando el índice de la componente como se muestra en las dos sentencias del ejemplo siguiente.

```
>> v(3)
ans =
    -3

>> v(1)*8
ans =
    -8
```

Es decir que en MATLAB la expresión `v(k)` equivale a la notación matemática v_k.

También se pueden construir vectores columna. Para ello se separan las filas mediante punto y coma como se muestra a continuación

```
>> u=[0; 1; 0; 1; 0]
u =
     0
     1
     0
     1
     0
```

Los vectores son tratados por MATLAB como matrices con la peculiaridad de tener una sola fila o columna. Por este motivo no merece la pena dedicar más espacio a los vectores y debemos pasar ya a las matrices.

9.3.4 Matrices

En MATLAB una matriz es una variable como otra cualquiera y no precisa mecanismos complicados para su creación y uso. A modo de ilustración considere que pretendemos sumar las dos matrices:

$$A = \begin{pmatrix} 0 & 2 & 4 \\ 1 & 1 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 3 & 3 & 3 \\ -1 & -1 & -1 \end{pmatrix}$$

El primer paso es introducir las matrices en variables. Comenzando con la matriz A escribimos la sentencia

```
A = [ 0 2 4; 1 1 1]
```

en la pantalla se obtiene:

```
>> A = [ 0 2 4; 1 1 1]

A =

     0     2     4
     1     1     1

>>
```

Como puede verse, los elementos de una misma fila se separan por espacios (o comas) y una fila se separa de la siguiente mediante el punto y coma.

Se procede del mismo modo con la matriz B, obteniéndose:

```
>> B = [ 3 3 3; -1 -1 -1]

B =

     3     3     3
    -1    -1    -1
```

9.3.5 Cadenas de caracteres

Las cadenas de caracteres son conjuntos de símbolos tomados de la tabla ASCII. Tienen gran utilidad para trabajar en problemas donde se precisa manipulación de texto.

Un ejemplo de cadena de caracteres es: `clase de 3 a 5 atrasada!!`. Este conjunto de caracteres incluye letras, números y signos como el espacio o la admiración. Es posible introducir la cadena en una variable de MATLAB:

```
>> texto = 'clase de 3 a 5 atrasada!!'

texto =

clase de 3 a 5 atrasada!!
```

Obsérvese que el conjunto de símbolos ha de ir encerrado por dos apóstrofes. Son las apóstrofes las que diferencian una sentencia como `a=barco` de `a='barco'`. Vea en el ejemplo el efecto de cada una de ellas.

```
>> a=barco
??? Undefined function or variable 'barco'.

>> a='barco'

a =

barco
```

En el primer caso MATLAB muestra en la pantalla un mensaje de error pues ha interpretado `a=barco` como una asignación en la cual el término de la derecha (la variable `barco`) no existe. En cambio, en la segunda expresión, `'barco'` es un valor definido que se asigna a la variable `a` por lo que la expresión es tan correcta como hubiera sido `x = 8`.

Las cadenas de caracteres no sirven para cálculos matemáticos propiamente dichos y fueron introducidas en MATLAB para facilitar la programación, el depurado y el uso de programas. No obstante proporcionan interesantes posibilidades para desarrollar algoritmos y practicar el arte de programar.

9.3.6 Funciones para el manejo de datos

Puesto que MATLAB está pensado para ser usado de forma interactiva es frecuente que el usuario necesite saber qué variables ha creado, cuáles son sus características, cuánto espacio ocupan, etc. Para ello se dispone de ciertas funciones que se van a explicar a continuación.

Informes . Se ha comentado anteriormente que la orden `who` proporciona una lista de variables. Vea como ejemplo la secuencia de órdenes y respuestas siguiente:

```
>> x=8
x =
     8

>> y=x+2
y =
    10

>> who

Your variables are:
 x  y
```

Existe una modalidad que proporciona los tamaños de las variables y que es de utilidad cuando se trabaja con matrices. La orden es `whos` como puede verse en el ejemplo.

```
>> A=[1 2 3; 3 4 6]

A =

     1     2     3
     3     4     6

>> whos
  Name      Size                    Bytes  Class

  A         2x3                        48  double array
  x         1x1                         8  double array
  y         1x1                         8  double array

Grand total is 8 elements using 64 bytes
```

Dimensiones A veces es necesario conocer el tamaño o dimensiones de una única variable, por ejemplo para saber el número de filas de una matriz y disponer de este número para posteriores cálculos. Para estos casos se puede usar la función `size`.

```
>> size(A)

ans =

     2     3
```

En el caso de manejar vectores resulta más conveniente la función `length` que proporciona el número de elementos.

```
>> v=[ -1 2 -3 4 -5]
v =
    -1     2    -3     4    -5

>> length(v)
ans =
     5
```

Almacenamiento MATLAB permite almacenar en el disco variables. De este modo es posible parar una sesión de trabajo y continuar en otro momento sin volver a repetir cálculos. La orden más simple para guardar es `save` que puede usarse de varias maneras. En la tabla siguiente se presenta un resumen.

Orden	Operación realizada
`save`	Crea el archivo de nombre *matlab.mat* en la carpeta actual. Dicho archivo contiene todas las variables que existen en ese momento en el entorno MATLAB.
`save nombrearchivo`	Crea el archivo de nombre *nombrearchivo.mat* en la carpeta actual. Dicho archivo contiene todas las variables que existen en ese momento en el entorno MATLAB.
`save nombrearchivo x y z`	Crea el archivo de nombre *nombrearchivo.mat* en la carpeta actual. Dicho archivo contiene únicamente las variables `x`, `y` y `z`.

Recuperación Las variables almacenadas en el disco pueden ser usadas en una sesión diferente. Para ello es preciso que MATLAB las lea del disco mediante la orden `load`. En la tabla siguiente se muestran tres posibilidades.

Orden	Operación realizada
`load`	Lee todas las variables del archivo de nombre *matlab.mat* de la carpeta actual. Si alguna de las variables del disco tiene nombre coincidente con otra que previamente existe en MATLAB se producirá la destrucción de la variable existente para dejar su sitio a la variable del disco.
`save nombrearchivo`	Igual que en el caso anterior, pero leyendo del archivo *nombrearchivo* de la carpeta actual.
`save nombrearchivo x y z`	Igual que el anterior pero leyendo únicamente las variables `x`, `y` y `z`.

Resulta útil la orden `what` pues muestra los archivos que existen en la carpeta actual y que contienen variables o programas utilizables por MATLAB.

9.4 Codificación de nuevos programas

Una de las características del entorno MATLAB es que permite que las órdenes puedan ser tomadas de un archivo en lugar de ser introducidas por el teclado.

La idea es simple: si el usuario va a repetir a menudo un conjunto de órdenes puede escribirlas en un archivo de texto. Posteriormente le indica a MATLAB que lea dicho archivo ejecutando las órdenes una por una. El efecto es el mismo que si el usuario hubiese escrito las órdenes en el entorno de MATLAB. Ahora bien, puesto que el archivo de texto se puede guardar en disco no es preciso volver a escribir las órdenes nunca más. Cada vez que el usuario desee ejecutar de nuevo el conjunto de órdenes podrá indicar nuevamente a MATLAB que lea el archivo. Esto supone un gran ahorro de tiempo en muchos casos.

Es costumbre dar a los archivos que contienen órdenes de MATLAB una extensión predeterminada que los diferencia de otros archivos de texto. Esta extensión es `.m`. Por ejemplo si un archivo contiene las órdenes para dibujar un vector se le puede llamar `dibuvector.m`.

Los archivos que contienen órdenes de MATLAB serán llamados desde ahora *archivos M*. Para escribir el archivo de texto se puede usar cualquier programa como la libreta de notas, el editor de MS-DOS, etc. MATLAB incorpora su propio programa para redacción llamado *M file Editor*, o sea redactor de archivos M.

A modo de ejemplo considere las siguientes órdenes de MATLAB que convierten una cantidad en pesetas a euros.

```
pesetas=input('Escriba la cantidad en pesetas : ')
```

```
euros = pesetas/166.386
```

Supongamos que se introducen este texto dentro de un archivo al cual se le da el nombre de `cpe.m` (el nombre viene de convertidor de pesetas a euros). La forma de indicar a MATLAB que utilice el archivo es simple: se escribe su nombre en el entorno MATLAB. Se obtiene el resultado que se muestra a continuación.

```
>> cpe
Escriba la cantidad en pesetas : 1000

pesetas =
        1000

euros =
    6.0101
```

9.4.1 Legibilidad

Frecuentemente el usuario de MATLAB escribe archivo M que le resultan de utilidad. Pasado un tiempo sin usar un archivo es posible que uno olvide qué tarea realiza exactamente. En tal caso es necesario mirar el contenido del archivo y repasar su contenido. Es en este punto cuando se agradece (o se echa en falta) una buena práctica de programación que haga que el código sea legible.

Los siguientes consejos ayudarán sin duda al usuario de MATLAB a conseguir programas legibles.

- Elegir nombres de variables indicativos de lo que representan.
- No usar una misma variable para representar más que una cosa.
- Incluir comentarios en el código para ayudar a seguir la secuencia del programa.
- Dividir el código en trozos, de forma tal que sea posible abarcar cada trozo de un vistazo en una ventana mediana. La división no ha de ser arbitraria, sino que los trozos deben tener cada uno cometidos claros. Normalmente los diagramas de flujo desarrollados con anterioridad a la codificación indican cómo realizar esta división.

A pesar de haber dado estos consejos hemos de ser conscientes de que la buena programación sólo se logra mediante un proceso de aprendizaje por prueba y error.

9.5 Funciones para trazado de gráficas

Uno de los motivos por el que MATLAB ha sido un entorno favorecido por el público es la facilidad con la que se pueden realizar gráficos de muy distintos tipos. En este punto se va a indicar la forma de realizar algunas representaciones gráficas que pueden ser muy útiles para ilustrar posteriormente otros ejercicios.

La orden de dibujo más simple es `plot`. Esta función puede ser utilizada de muchas maneras. En primer lugar puede usarse para representar las componentes de un vector. Por ejemplo, supongamos que el vector:

```
>> v=[ 15.6 16.2 18 17 16.5 15 ]
```

está formado por las temperaturas (en grados Celsius) medidas cada cuatro horas en una estación meteorológica. Si se escribe lo siguiente en el entorno MATLAB

```
>> v=[ 15.6 16.2 18 17 16.5 15 ]
v =
   15.6000   16.2000   18.0000   17.0000   16.5000   15.0000

>> plot(v)
```

se observa que aparece una nueva ventana conteniendo una gráfica con el aspecto que muestra la figura 9.1. Puede observarse que en el eje vertical MATLAB ha representado los valores v_1, v_2, etc. mientras que en el horizontal aparece el subíndice correspondiente.

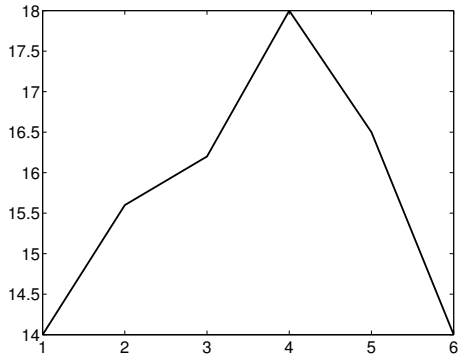


Figura 9.1: Gráfica obtenida con la orden `plot`

Continuando con el ejemplo supongamos ahora que se conoce la hora a la que se realizó cada medida y que deseamos que aparezca en el eje horizontal. Sea $t = [4812162024]$, la orden que necesitamos es simplemente:

```
>> t= [ 4 8 12 16 20 24 ]
t =
     4     8    12    16    20    24

>> plot(t,v)
```

que produce un cambio en la ventana, de forma que ahora se obtiene lo que muestra la figura 9.2. Se observa que el eje horizontal está ahora marcado con las componentes del vector t.

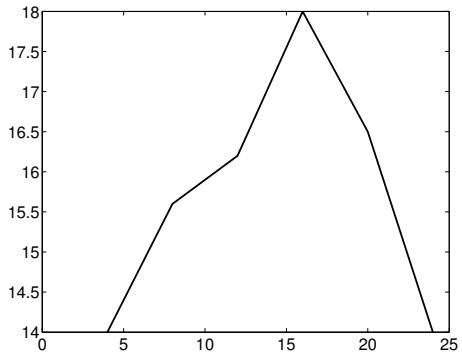


Figura 9.2: Gráfica obtenida con la orden `plot` usando una variable para el eje horizontal y otra para el vertical.

Puede ser interesante ahora añadir unos letreros indicativos de qué es lo que se está representando. Esto se consigue de manera simple con las órdenes siguientes:

```
>> xlabel('hora')
>> ylabel('temperatura')
>> title('Datos estaci{\'o}n meteorol{\'o}gica')
```

El resultado final se muestra en la figura 9.3.

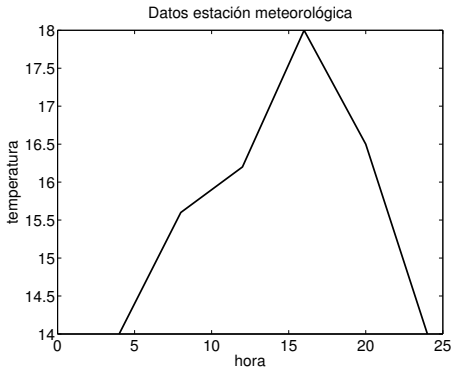


Figura 9.3: Gráfica con título y letreros en los ejes.

9.6 Sentencias de control

9.6.1 La bifurcación

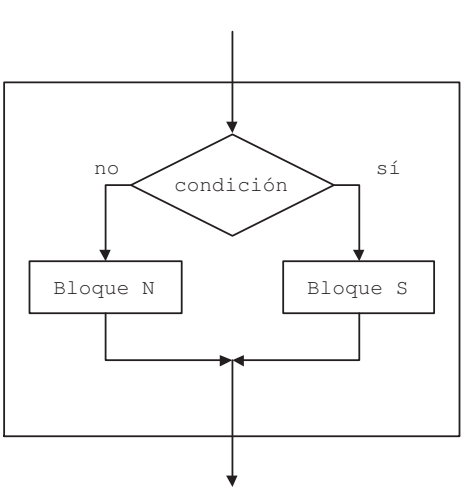
Ya se ha visto en los diagramas de flujo realizados que las bifurcaciones son imprescindibles para llevar a cabo ciertas tareas. En MATLAB es posible realizar la bifurcación básica mediante la sentencia `if-else`. La forma de uso se muestra en el código que aparece a continuación.

```
if condici{\'o}n
  sentencias bloque S
else
  sentencias bloque N
end
```

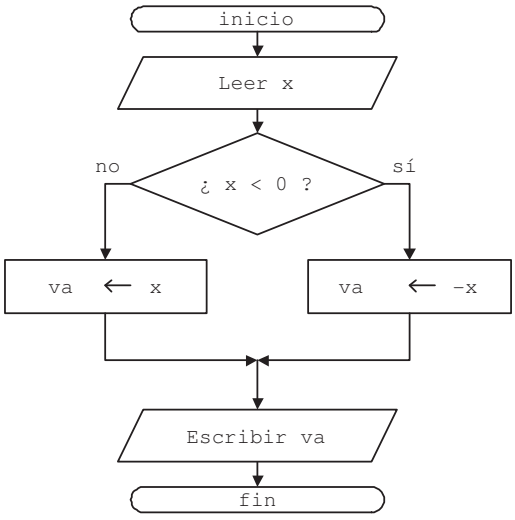
En esta construcción, La condición es una expresión que da como resultado un valor lógico (por ejemplo $x > 2$) que puede ser verdadero o falso. En caso de que la evaluación de la expresión arroje un resultado verdadero se procede a ejecutar las sentencias del bloque S. En caso contrario se ejecutan las sentencias del bloque N. Al terminar uno u otro bloque se pasa a las sentencias posteriores a `end`.

De la explicación anterior debe resultar evidente que al usar la bifurcación se ejecutan las sentencias S o las N pero no ambas.

El diagrama de flujo de la figura 9.4 a) corresponde a una bifurcación genérica. Dentro de los bloques S y N se puede colocar cualquier conjunto de sentencias, incluyendo nuevas bifurcaciones como se verá más adelante.



a)



x	Variable real	Dato
va	Variable real	Resultado. Valor absoluto de x

b)

Figura 9.4: a) Diagrama de flujo correspondiente a una bifurcación genérica. b) Diagrama de flujo correspondiente al ejemplo de bifurcación

Como ejemplo sencillo considere el siguiente trozo de código:

```
x=input('Introduzca valor de x ')
if x < 0,
  va = -x;
else
 va=x;
end
disp('El valor absoluto es')
va
```

Este programa sería de utilidad en caso de que no dispusiésemos en MATLAB de otros medios para calcular el valor absoluto.

Un ejemplo igualmente simple es:

```
a=input('Coeficiente a? ')  %coeficientes de a x^2 + b x + c = 0
b=input('Coeficiente b? ')
c=input('Coeficiente c? ')
if b*b-4*a*c < 0,
  disp('Las ra{\'\i}ces son complejas')
else
 disp('Las ra{\'\i}ces son reales')
end
```

Es fácil ver que este programa toma como datos los coeficientes a, b y c de una ecuación de segundo grado $ax^2 + bx + c = 0$, los analiza y escribe en la pantalla si las raíces (soluciones de la ecuación) serán reales o complejas (con parte imaginaria).

En los ejemplos anteriores la condición de la bifurcación depende de una única relación lógica. Ocurre a menudo que la condición es más elaborada dependiendo de varias relaciones lógicas combinadas. A modo de ejemplo considere el problema de determinar si un número leído pertenece o no al intervalo $(2, 5)$. En otras palabras, se ha de determinar si el número leído (llamémosle x) cumple o no cumple la condición $x > 2$ y simultáneamente $x < 5$. El operador & permite combinar las dos desigualdades como muestra el siguiente ejemplo:

```
x=input('Introduzca valor de x ')
if x > 2 & x < 5,
  disp('x est{\'a} en el intervalo')
else
```

```
  disp('x est{\'a} fuera del intervalo')
end
```

El operador & permite pues realizar la suma lógica de los resultados de las dos comparaciones. De forma similar el operador | realiza la suma lógica. El ejemplo siguiente es un programa que permite calcular si el número leído x es exactamente igual a tres o a cuatro.

```
x=input('Introduzca valor de x ')
if x == 3 | x == 4,
  disp('x cumple la condici{\'o}n')
else
  disp('x no cumple la condici{\'o}n')
end
```

9.6.2 El bucle "mientras"

El bucle "hacer mientras que la condición sea cierta" se construye en MATLAB mediante un conjunto de líneas de código que tienen la forma siguiente:

```
while condici{\'o}n
  sentencias del cuerpo del bucle
end
```

En la figura 9.5 a) se muestra el diagrama de flujo correspondiente al bucle. La condición es una expresión que da como resultado un valor lógico, por ejemplo $x > 2$. El cuerpo del bucle son sentencias cualesquiera.

Examinando el diagrama de flujo de la figura 9.5 a) es fácil deducir que mientras la condición se cumpla se repetirá el bucle una y otra vez. Es decir, si al evaluar la expresión lógica de la condición se obtiene un resultado verdadero se pasa a ejecutar el cuerpo del bucle, en caso contrario se termina el bucle.

Como ejemplo sencillo considere el siguiente trozo de código:

```
x=0; suma=0;
while x<10,
suma = suma + x;
```

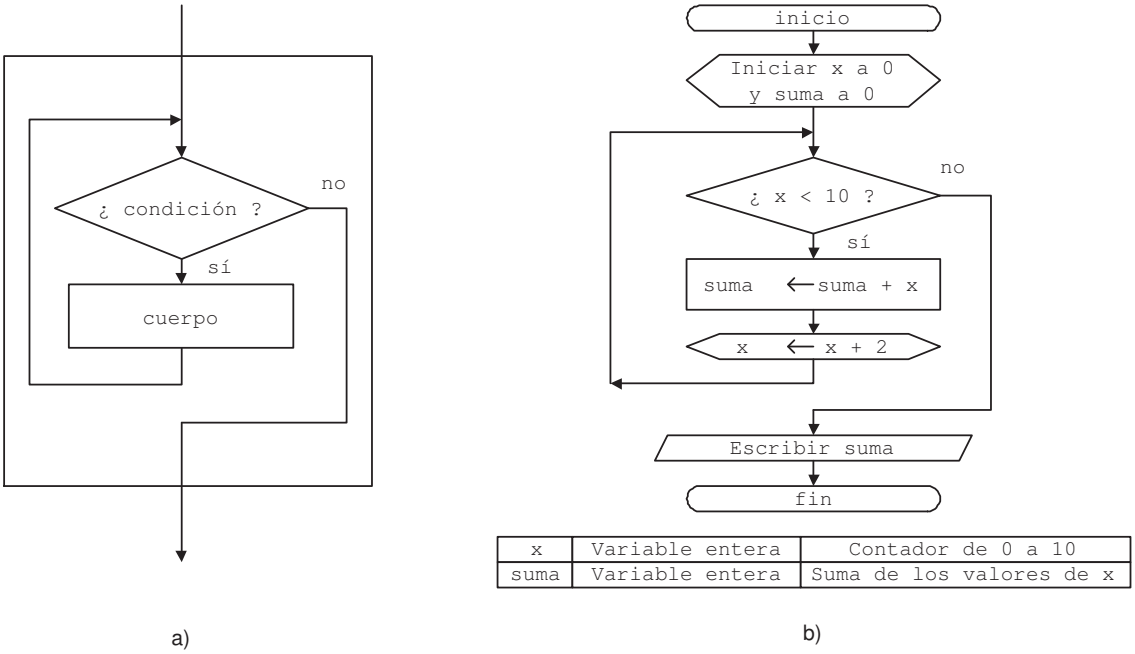


Figura 9.5: a) Diagrama de flujo correspondiente a un bucle genérico del tipo "repetir mientras la condición sea cierta". b) Diagrama de flujo correspondiente al ejemplo de uso del bucle "mientras"

```
x = x+2;
end
suma
```

Es fácil adivinar qué ocurre al ejecutar este programa, sobre todo si se dibuja el diagrama de flujo equivalente (véase figura 9.5 b).

Para mejorar la legibilidad de los programas es muy aconsejable incluir comentarios y ordenar la apariencia en la pantalla de las sentencias. De este modo el ejemplo en cuestión resulta más fácil de interpretar.

```
x=0;     % variable contador para el bucle
suma=0;  % variable suma parcial

while x<10,          % condici{\'o}n bucle
  suma = suma + x;   %
  x = x+2;           % actualizaci{\'o}n
end
suma                 % escritura resultado
```

9.6.3 La sentencia for

En muchas situaciones es preciso repetir una tarea un número conocido de veces. Para ello se puede disponer un bucle con un contador. En cada pasada o repetición el contador se incrementa y se comprueba que no se ha sobrepasado el límite. Estas tareas se realizan fácilmente en MATLAB con el uso de la sentencia for.

Los bucles que utilizan for tienen en MATLAB el aspecto siguiente:

```
for contador = valorinicial:valorfinal,
  sentencias del cuerpo del bucle
end
```

En la figura 9.6 a) se muestra el diagrama de flujo correspondiente al bucle anterior. En él se observa que `contador` es una variable que sirve para controlar el bucle. Esta variable toma inicialmente el valor especificado por `valorinicial`. La variable se incrementa al final de cada pase o repetición en una unidad. La condición de salida del bucle consiste en que el contador no sobrepase el límite fijado por el valor `valorfinal`.

A fin de aclarar las ideas se muestra el siguiente ejemplo de bucle usando la sentencia for.

```
for k = 1:6,
  k
end
```

En la figura 9.6 b) se presenta el diagrama de flujo correspondiente a este ejemplo. Es fácil ver que el bucle no realiza cálculo alguno, simplemente escribe en la pantalla los valores sucesivos que va adquiriendo la variable `k`.

Este ejemplo revela porqué se le llama bucle "para". Se observa que *para cada valor de k entre 1 y 6* se escribe k en la pantalla, que es lo que literalmente dice la sentencia

```
for k = 1:6,
```

Los valores iniciales y finales no necesariamente son constantes, pueden ser variables como en el programa siguiente.

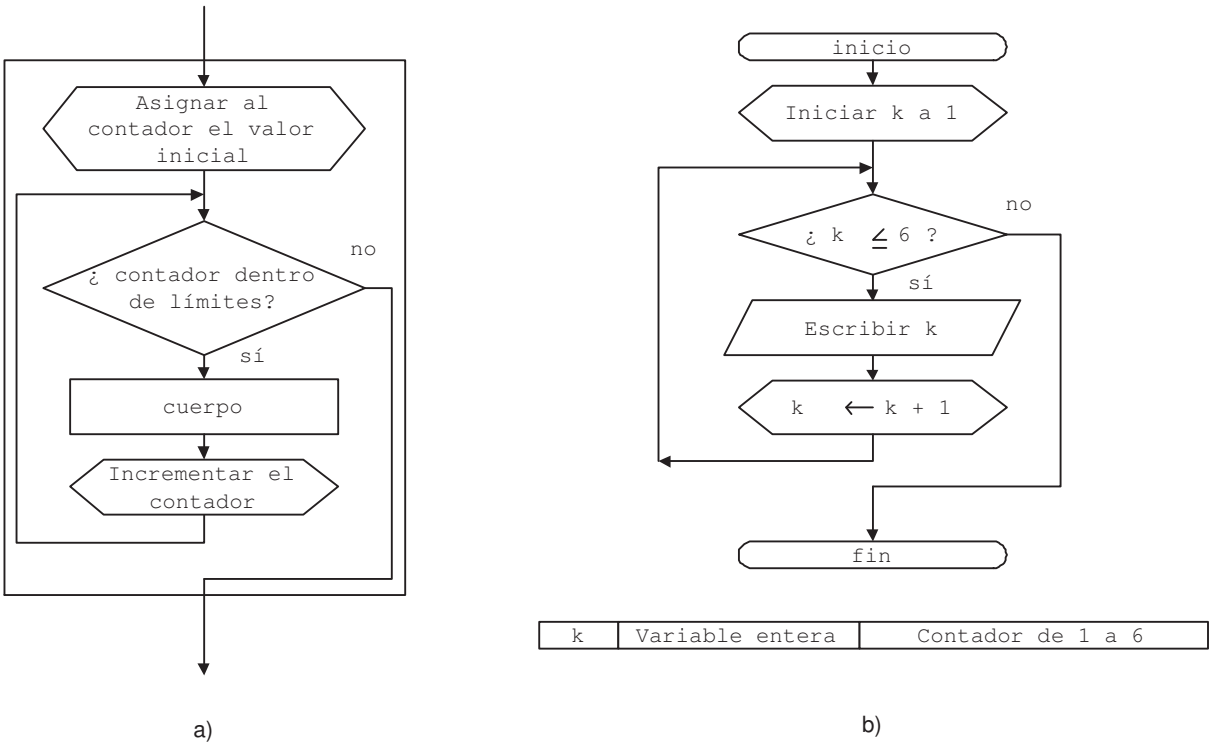


Figura 9.6: a) Diagrama de flujo correspondiente a un bucle genérico del tipo "para". b) Diagrama de flujo correspondiente al ejemplo de uso del bucle "para"

```
% valores inicial y final para bucle
v_inicial = input('Introduce valor inicial')
v_final = input('Introduce valor final')

% bucle
for k = v_inicial:v_final,
  k %escritura en pantalla
end
```

El incremento de la variable contador puede ser distinto de uno. Por ejemplo para realizar una cuenta atrás (mediante decremento del contador) o para saltar de dos en dos, etc. La forma de indicar un incremento en la sentencia for es muy simple:

```
for contador = valorinicial:incremento:valorfinal,
  sentencias del cuerpo del bucle
end
```

A modo de ejemplo considere el programa:

```
for k=1:2:20,
  k
end
```

9.7 Ejercicios propuestos

En estos ejercicios se debe realizar un algoritmo mediante diagrama de flujo y tabla de objetos. Antes de codificar el algoritmo debe comprobar que sea correcto.

A fin de poder resolver un mayor conjunto de problemas se permitirá en los algoritmos el uso de funciones de MATLAB tal y como las trigonométricas, logarítmicas, raíz cuadrada y funciones de dibujo.

Posteriormente ha de codificar los algoritmos en módulos de MATLAB y probarlos con conjuntos de datos controlados.

1. Leer las 5 componentes de un vector v y escribirlo luego al derecho y al revés.

2. Realizar un programa que permita pasar de grados Farentheit a Celsius.

3. Crear un vector x de dimensión 100, cuyas componentes son los valores

$$x_k = -1.001 + 0.01 \cdot k$$

Calcular a continuación un vector y cuyas 100 componentes vienen dadas por $y_k = \frac{\text{sen}(x_k)}{x_k}$. Finalmente se ha de representar gráficamente y frente a x.

4. Leer del teclado un par de abcisas x_1 y x_2. Representar gráficamente la función $y = x - \text{sen}(x)$ utilizando cien puntos equiespaciados entre x_1 y x_2.

5. Utilizar el programa anterior para calcular gráficamente la solución de la ecuación dada por $\text{sen}(x) = x$ (calculando el paso por cero de la función $y = x - \text{sen}(x)$ mediante llamadas sucesivas al programa con valores de x_1 y x_2 que estén a izquierda y derecha de dicho pase por cero).

6. Programar un menú matemático simple. El programa ha de leer un número x y luego aceptar una opción elegida por el usuario. Si la opción es 1 se ha de escribir el valor absoluto de x, si es 2 se ha de escribir x^2, si es 3 aparecerá x^3, finalmente si es 4 se escribirá $x!$.

7. Codificar y probar el algoritmo que calcula la raíz cuadrada aproximada entera por defecto.

8. Realizar un programa que resuelva la ecuación de segundo grado $x^2 + bx + c = 0$ leyendo los coeficientes b y c. Considere separadamente los casos de soluciones reales y complejas.

9. Realizar un programa que dibuje en la pantalla un polígono regular de n lados, siendo $n \in \mathbb{N}$, $n > 2$ un número leído previamente. Utilize las funciones trigonométricas para calcular la posición de los vértices en el plano y las funciones de dibujo para trazar el polígono. Suponga que el centro del polígono está en el origen y que el primer vértice está sobre el eje OX. La longitud del lado será 1 en todo caso.

10. Realice un programa que permita dibujar una circunferencia aproximada como un polígono regular de 200 lados.

11. Generalice el programa anterior para dibujar elipses. Se tomará el semieje horizontal igual a uno y el vertical igual a v siendo v un número que habrá de leerse del teclado.

12. Los vértices de un polígono irregular de n lados en el plano consisten en n parejas (x_i, y_i) con $i = 1, ..., n$. Suponga que dispone de una matriz $V \in \mathbb{R}^{n\times 2}$ cuyas componentes son $a_{i1} = x_i$ y $a_{i2} = y_i$. Realice un programa que calcule el perímetro del polígono dado por V.

13. Se ha de escribir un uno en el caso de que exista un trío (x, y, z) de números enteros positivos tales que $x^2 + y^2 = z^2$. Limite la búsqueda a $x \in (0, 100)$, $y \in (0, 100)$. En caso de que no se encuentre solución se ha de escribir un cero.

14. Repita el ejercicio anterior pero escribiendo todas las soluciones que encuentre en el intervalo $x \in (0, 1000)$, $y \in (0, 1000)$.

15. Se desea calcular la suma $s = \sum_{k=1}^{n} 1/a_k$ siendo los valores a_k los elementos de la sucesión dada por $a_k = a_{k-1} + a_{k-2}$ con $a_1 = 1$ y $a_2 = 1$. El límite n ha de leerse del teclado y se supone mayor que dos.

16. Se desea calcular la suma $s = \sum_{k=1}^{n} 1/k^m$ siendo m y n dos números enteros positivos que se suponen dados.

17. Escriba los n primeros términos de la sucesión dada por $a_n = (1 + \frac{1}{n})^n$, siendo n un número entero positivo dado.

9.8 Puesta a punto y mantenimiento de programas

El proceso de resolución de problemas requiere realizar un algoritmo que posteriormente es codificado. Incluso si el algoritmo es correcto puede ocurrir que el programa resultante no funcione. Esto pasa pues al realizar la codificación pueden introducirse errores variados. En tal caso es necesario pasar a *depurar* el código.

9.8.1 Tipos de error

En primer lugar se van a clasificar los errores en varios grupos:

- Errores sintácticos. Son debidos a una mala escritura del código, por ejemplo al cambiar un carácter por otro (3]2 en lugar de $3+2$ o `sen(1.2)` en lugar de `sin(1.2)`) o al escribir mal el nombre de una variable que se estaba utilizando.

  Estos errores suelen detectarse fácilmente pues MATLAB produce mensajes orientativos en cuanto llega al primero de estos errores. Para corregirlos basta con mirar el lugar donde MATLAB indica que se ha producido el error (la línea del archivo del módulo en cuestión).

- Errores de codificación. Suelen provenir de una mala transcripción del algoritmo en código. Por ejemplo al colocar los valores iniciales de variables, o al crear condiciones para bifurcaciones y ciclos.

  Al utilizar estructuras repetitivas hay que poner especial cuidado en no crear bucles infinitos (que nunca terminan) o bucles vacíos (que nunca realizan ni siquiera la primera iteración). En ambos casos hay que vigilar la condición de salida, los valores iniciales y las actualizaciones de las variables que controlan los ciclos.

- Errores derivados del uso de variables. El caso más común es utilizar una variable a la cual no se le ha proporcionado previamente ningún valor.

  En otras ocasiones el error proviene de usar una variable más de una vez con significados distintos. Es posible que una parte del programa cambie esta variable siendo necesario el valor antiguo para una parte posterior del programa.

  Estos dos tipos de errores se resuelven fácilmente si se ha preparado cuidadosamente la tabla de objetos teniendo cada variable un único cometido.

  Otro tipo de errores surge al considerar funciones. Es frecuente confundir los argumentos y otras variables locales de las funciones con variables del programa principal. Estas cuestiones se explicarán en un punto posterior.

- Errores durante la ejecución. Incluso si el programa no contiene errores sintácticos puede ocurrir que al ejecutarlo se den circunstancias que impidan un correcto desarrollo del mismo. Las causas más frecuentes son

  – El desbordamiento. Si se realiza una operación cuyo resultado excede la capacidad de representación de la máquina se produce un error de desbordamiento. Esto ocurre por ejemplo si se intenta realizar la operación 1/0.
  – Índices fuera de intervalo. Los índices para acceso a vectores y matrices han de ser enteros y mayores que cero.
  – Números complejos. Si se realiza la raíz cuadrada de un número negativo en algunos lenguajes el resultado es un error de desbordamiento. En el caso de programar en MATLAB se obtiene como resultado un número complejo. Esto puede dar lugar a errores si no se tiene en cuenta que dicho número posee parte real e imaginaria que han de ser tenidas en cuenta.

9.8.2 Pruebas de los programas

Para detectar errores en la codificación de los programas es preciso realizar pruebas sobre los mismos. En estas pruebas es conveniente tener en cuenta algunos consejos. Se supone que el algoritmo que da lugar a la codificación ha sido ya comprobado previamente, por lo que los errores que se detecten provienen de una mala codificación.

Conjunto de datos Ya es sabido que las pruebas se realizan con un conjunto controlado de datos para los cuales se conoce el resultado correcto. Este conjunto debiera incluir los casos "difíciles" en los que los datos toman valores "extremos". Por ejemplo, en un programa que calcule el factorial puede ser interesante ver si responde adecuadamente al cálculo de 0! y 1!.

Mensajes de error Un programa bien diseñado debiera defenderse cuando los datos que se le suministran no son adecuados. Así, el programa que calcula la raíz cuadrada aproximada

por defecto (véase tema 6) no debiera admitir que el dato sea un número negativo. La forma más fácil es hacer que el programa escriba en la pantalla un mensaje de error y termine su propia ejecución. Este método expeditivo no debe usarse en programas de envergadura, pero sí resulta muy efectivo para los pequeños programas de un curso introductorio.

Mensajes orientativos Si un programa consta de muchas partes puede ser útil que en la fase de depuración el programa vaya indicando mediante mensajes en la pantalla la parte en la que se halla. De este modo es posible cazar bucles infinitos y otras situaciones indeseables.

A pesar de estos consejos la depuración es una tarea que se complica mucho a medida que el programa crece de tamaño. Por este motivo se recomienda siempre hacer uso de módulos pequeños que han de comprobarse intensivamente antes de ser incorporados a módulos mayores.

9.8.3 Documentación

Los programas deben ir provistos de documentación tanto para los futuros usuarios como para facilitar su posible modificación por parte del mismo creador o de otro programador.

La documentación necesaria puede clasificarse en varios grupos:

Interna Es la que está incluida en el propio código del programa. A su vez puede ser:

Comentarios. Sólo se ven en el código fuente.

Mensajes en pantalla. Aparecen durante el uso del programa para guiar al usuario.

Ayuda directa. Equivale al manual del usuario pero en formato digital.

Externa Es la que consta en documentos adicionales, incluyendo:

Manual de usuario

Diagramas de flujo o pseudocódigo y tablas de objetos

Descripción de los datos utilizables

Descripción de módulos y funciones y de los archivos que los contienen

9.9 Funciones

En matemáticas una función f calcula un valor (y variable dependiente) a partir de otro dado (x variable independiente). Tanto x como y pueden ser escalares, vectores o matrices. Con las

funciones de MATLAB ocurre otro tanto. El papel de la variable independiente es representado por los datos que se le suministran a la función, llamados normalmente argumento. Se dice habitualmente que la función "devuelve" un resultado que es el equivalente a la variable dependiente.

Para aclarar las ideas consideremos la sentencia de MATLAB `y = sin(x)`. Hemos de recordar que la ejecución de esta sentencia provoca lo siguiente:

1. El cálculo mediante la función `sin` del seno del dato proporcionado, que en este caso es el valor de `x`.
2. La asignación a la variable `y` del valor calculado por la función `sin` (que es, lógicamente $\text{sen}(x)$).

Es muy conveniente no perder de vista estos pasos en las explicaciones que se darán posteriormente.

Además de las funciones existentes en MATLAB hay mecanismos para que el usuario escriba funciones nuevas. Para ello sólo tiene que preparar en un archivo de texto las órdenes de MATLAB que realizan un determinado cálculo y posteriormente añadir una cabecera que permita a esas órdenes trabajar como debe hacerlo una función.

Se va a ilustrar el procedimiento con un ejemplo. Supongamos que se necesita una función que calcule el factorial de un número dado n. Escribimos en un archivo las órdenes:

```
producto=1;
for multiplicador=2:n,
  producto = producto*multiplicador;
end
factorial = producto;
```

y con esto tenemos resuelto parte del problema. Ahora bien, si queremos utilizar este trozo de código debemos recordar siempre que hemos de usar la variable `n` para el dato. Dicho de otro modo, este trozo de código no sirve si se pretende calcular el factorial de `x` o de `q`. Lo que se necesita es añadir una cabecera para que el trozo de código se convierta en una función.

```
function [factorial] = mi_fact(n)
```

```
producto=1;
for multiplicador=2:n,
  producto = producto*multiplicador;
end
factorial = producto;
```

Se ha de guardar este texto en un archivo M cuyo nombre ha de ser `mi_fact`. Este nombre se ha escogido para recalcar que es una función mía y que calcula el factorial.

Para probar mediante ejemplos que este archivo M es una función escribimos en la ventana de órdenes de MATLAB lo siguiente:

```
>> mi_fact(5)
ans =
   120
```

Cada vez que se utiliza una función se dice que se hace "una llamada" a la misma, o que se la "invoca". Nótese que esta invocación, uso o llamada produce un efecto interesante: el valor que se le proporciona (el valor 5 en el ejemplo) es copiado en la variable `n` de la función, de modo que las órdenes que se habían escrito y que calculan el factorial de `n` están en realidad calculando el factorial del dato escrito entre paréntesis. A este fenómeno se le suele llamar "pase de argumento".

Por otra parte, el resultado que se calcula y que almacena en la variable `factorial` aparece tras la llamada en el entorno de MATLAB. Si esto no se entiende del todo considere este otro ejemplo:

```
>> y=0
y =
     0

>> y=mi_fact(4)
y =
    24

>> y
y =
    24
```

Ahora debe ser obvio que el valor de `factorial` ha sido asignado a la variable `y`. A este fenómeno se le da el nombre de "devolución de resultados".

Con esta explicación ya tiene sentido una frase habitual entre programadores como "le pasé a la función el valor 4 y me devolvió en `y` el factorial de 4".

La sintaxis para la escritura de funciones es simple. En primer lugar debe aparecer una línea en la que se indica:

- el nombre de la función (en el ejemplo anterior es `mi_fact`).
- el nombre de la variable resultado (en el ejemplo anterior es `factorial`).
- el nombre de los argumentos (en el ejemplo anterior hay un único argumento de nombre `n`).

A continuación se escribirán sentencias de MATLAB incluyendo todo tipo de cálculos. Es obligatorio que en algún punto se le de valor a la variable resultado pues de otro modo la función no sabrá qué devolver al ser invocada y generará un error.

9.10 Variables locales y globales

Continuando con el ejemplo anterior es instructivo utilizar la orden `who` para ver qué variables se están utilizando:

```
>> who
Your variables are:

y
```

el resultado obtenido quizá debiera sorprender, porque, ¿dónde está la variable `producto`?, ¿y `factorial`?, ¿y `multiplicador`?, ¿y `n`?. La única variable que parece existir (según indica el resultado de `who`) es `y`.

La respuesta es que estas variables no son visibles desde el entorno de MATLAB pues están ocultas dentro de la función `mi_fact`. Esta característica es muy útil pues de este modo cada función puede usar variables con el nombre que quiera sin que haya que preocuparse porque esta variable ya exista previamente en otra función.

Este hecho se produce con las funciones, pero no con cualquier archivo M. El siguiente ejemplo puede contribuir a aclarar esta afirmación. Considere los dos archivos M que se indican a continuación.

```
function [y] = f_recta(x)
 a = 8; % pendiente
 b = 2; % ordenada en el origen
 y = a*x + b; %valor de la recta en x
```

Archivo `f_recta.m`

Este primer archivo M contiene una función. Es fácil ver que la función calcula la ordenada sobre la recta $y = 8x + 2$ correspondiente a un punto de abcisa x que es el dato proporcionado a la función.

```
 a = 8; % pendiente
 b = 2; % ordenada en el origen
 y = a*x + b; %valor de la recta en x
```

Archivo `m_recta.m`

Este otro archivo no es una función, es simplemente un conjunto de órdenes de MATLAB que se han escrito y guardado en disco.

Se va a mostrar mediante ejemplos las diferencias de uso de ambos archivos.

```
>> a=-2
a =
     -2

>> f_recta(1)
ans =
    10
```

```
>> a
a =
     -2
```

Las líneas anteriores presentan un ejemplo en el que se hace uso de la función `f_recta` en el que se observa que la variable `a` permanece inalterada a pesar de haber ejecutado la función.

```
>> a=-2
a =
    -2

>> x=1
x =
     1

>> m_recta
>> y
y =
    10

>> a
a =
     8
```

Se observa en este ejemplo de uso del archivo M `m_recta` que la variable `a` cambia de valor. Queda claro pues que la variable que usa el archivo M es la misma que existe en el entorno de MATLAB.

9.10.1 Clasificación de las variables

Las variables pueden clasificarse de varias maneras. Atendiendo al campo donde pueden ser vistas y utilizadas se distinguen dos grupos:

Globales Estas variables pueden ser vistas y utilizadas desde cualquier archivo M y desde el entorno de MATLAB.

Locales Son variables que sólo pueden ser vistas y utilizadas dentro de alguna función. Fuera de la misma no son visibles y por tanto no son utilizables.

Por otra parte, atendiendo a la durabilidad de la variables se pueden hacer dos grupos:

Persistentes Estas variables existen desde el momento en que son creadas hasta que se las destruye explícitamente mediante la orden `clear`. Las variables globales son de este tipo. También se las suele llamar variables *estáticas*.

Las variables estáticas o persistentes no pierden su valor por culpa de llamadas a funciones u otros acontecimientos. Tan sólo se ven afectadas por órdenes de asignación.

Efímeras Son variables locales que son creadas por una función y desaparecen al terminar la función. Debido a esto las variables que están dentro de una función no conservan su valor de una llamada a otra de la función. Existe un procedimiento para convertir las variables locales en persistentes mediante la orden `persistent`.

Las características de disponibilidad (global o local) y durabilidad (persistente o efímera) pueden combinarse dando lugar a varias situaciones. El mejor modo de poner de manifiesto todo esto es utilizando ejemplos como los que se proporcionan a continuación.

Ejemplo 1. Las variables del entorno de MATLAB son locales.

Para poner de manifiesto esta afirmación considere la función `f1` cuyas órdenes se guardan en el archivo `f1.m`.

```
function [y] = f1(x)
 a = 8; % pendiente
 y = a*x + b; %valor de la recta en x
```

Archivo `f1.m`

Puede verse que el valor de b no es asignado dentro de la función, por lo que intentaremos hacerlo desde el entorno. Para ello creamos la variable y le damos un valor:

```
>> clear all
```

```
>> b=4
b =
     4
```

y a continuación utilizamos la función `f1`

```
>> f1(9)
??? Undefined function or variable 'b'. Error in ==> f1.m
On line 3  ==>  y = a*x + b; %valor de la recta en x
```

Aparece un mensaje de error que nos indica que la variable `b` no es conocida dentro de la función `f1`. La explicación es simple: la variable `b` pertenece al entorno de MATLAB. No es una variable global. No puede ser utilizada en el interior de funciones.

Ejemplo 2. Las variables interiores de una función son locales y no pueden ser accedidas desde otras funciones o desde el entorno de MATLAB. Para ilustrarlo se retoma el archivo `f1` y se modifica para dar lugar a una función que se guardará en el archivo `f_recta` que se reproduce a continuación.

```
function [y] = f_recta(x)
 a = 8; % pendiente
 b = 2; % ordenada en el origen
 y = a*x + b; %valor de la recta en x
```

Archivo `f_recta.m`

Haremos una llamada a la función y luego intentaremos averiguar desde el entorno de MATLAB cuánto vale la variable `a` que es la pendiente de la recta.

```
>> clear all
>> ordenada2 = f_recta(2)
ordenada2 =
    18
```

```
>> a
??? Undefined function or variable 'a'.
```

Obtenemos un mensaje de error que era esperable pues `a` es una variable local de la función `f_recta` y no es accesible fuera de esta función.

Ejemplo 3. Las variables globales son accesibles desde cualquier función.

Para crear una variable global es preciso escribir `global nombrevariable` en las funciones que la vayan a usar, incluyendo la ventana de órdenes de MATLAB en caso necesario.

En este ejemplo comenzamos por tanto escribiendo:

```
>> global b
```

que no produce respuesta alguna, pero que crea la variable `b` aunque sin valor asignado. Para poner esto de manifiesto usamos la orden `whos`.

```
>> whos
  Name      Size                    Bytes  Class
  b         0x0                         0  double array (global)

Grand total is 0 elements using 0 bytes
```

La función que vaya a utilizar la variable global ha de contener también la orden `global b`. Creamos la función `f2` que es una modificación de `f1` (véase ejemplos anteriores).

```
function [y] = f2(x)
global b
a = 8; % pendiente
y = a*x + b; %valor de la recta en x
```

Archivo `f2.m`

Ahora intentamos asignar un valor a `b` y veamos si la función `f2` es capaz de utilizar dicho valor.

```
>> b=-5
b =
    -5

>> f2(1)
ans =
     3
```

La respuesta obtenida es la deseada, con lo que queda probado que las variables globales pueden ser usadas dentro y fuera de funciones.

9.10.2 Fases de la ejecución de una instrucción

De los ejemplos anteriores es posible sacar conclusiones generales acerca de cómo utiliza MATLAB las funciones. Para terminar de exponer estas ideas se indican a continuación las fases de la ejecución de una instrucción. Conviene comprenderlas bien pues se trata de los pasos que MATLAB realiza cada vez que se invoca a una función. El éxito o fracaso de las funciones que uno escriba estará sin duda influenciado por el buen conocimiento de estas fases.

1. Creación de variables locales para argumentos formales.

2. Recogida del valor de los argumentos de la llamada por parte de las variables locales.

3. Cálculos incluidos en el cuerpo de la función.

4. Envío de valores de las variables resultados a las variables de la sentencia llamante.

Resulta un ejercicio interesante comprobar en los ejemplos anteriores que dichas fases tienen lugar y que son necesarias para el desempeño de las tareas encomendadas a una función.

9.11 Ejemplos con funciones en MATLAB

A continuación se presentan algunos ejemplos resueltos. Cada problema ha sido planteado para ser resuelto y programado en MATLAB.

9.11.1 Cola de montaje

Se desean codificar dos funciones de MATLAB que sirvan para simular el comportamiento de una cola de montaje (hilera de piezas distintas que esperan a la vez). Estas funciones trabajarán con los números de identificación de las piezas (enteros positivos). Dichos números se almacenarán en un vector v por orden de llegada a la cola.

La primera función se llamará `entrada` y admitirá como argumentos el número de la pieza que acaba de incorporarse a la cola y el vector v. La función ha de introducir el número de la pieza recién llegada al final del vector. Como resultado ha de devolver el vector de la cola modificada.

La segunda función se llamará `salida`. Su único argumento es el vector v. La función sacará del vector v el número de la pieza que lleve más tiempo en la cola. Como resultado ha de devolver el nuevo vector v en el que se ha retirado la pieza que ha salido, además ha de aparecer en la pantalla el número de la pieza saliente.

A fin de aclarar el ejercicio considere los siguientes ejemplos de uso:

```
>> v = [ 12 25 3];
>> w=entrada(44, v);
>> w
w =
    12 25  3 44
```

En este primer ejemplo puede verse que la función entrada ha introducido el nuevo valor (44) en la cola. El resultado devuelto por la función se asigna en este caso a un nuevo vector w.

```
>> [z,p]salida(w);
>> z
z =
    25     3    44
>> p
p =
```

```
    12
```

El segundo ejemplo ilustra la forma en que trabaja la función salida. Se observa que los resultados son dos: el vector z que contiene las nuevas componentes de la cola tras la salida de una de ellas y el número p de la pieza que ha salido.

Para la codificación es necesario usar la función `length` que devuelve el número de elementos de un vector. Por ejemplo si `v = [ 1 2 3 4]` entonces:

```
>> length(v)
ans =
    4
```

Solución. La función `entrada` tiene como misión añadir el identificador de la nueva pieza al final del vector. El siguiente trozo de código puede ser solución (se ha obviado el diagrama de flujo por ser trivial).

```
function [nv]=entrada(nropieza, vector)
  n = length(vector);
  for k=1:n,
    nv(k)=vector(k);
  end
  nv(n+1)=nropieza;
```

Para una mejor comprensión de la función considere la tabla de objetos de la misma.

Objeto	Identificador	Tipo	Valor
Número que identifica la pieza que va a entrar en la cola. Argumento.	`nropieza`.	entero	variable
Vector que contiene los identificadores de las piezas en la cola. Argumento.	`vector`	vector de enteros	variable
Nuevo vector que incluye la pieza recién entrada. Resultado.	`nv`	vector de enteros	variable
Número de piezas en la cola antes de entrar la nueva. Variable auxiliar.	`n`	entera	variable
Función de biblioteca para calcular el número de elementos de un vector.	`length`		

Por otra parte, la función `salida` ha de proporcionar el elemento que ocupa el primer lugar en el vector y deshacerse de él, moviendo los otros elementos una posición.

```
function [nv, pieza]=salida(vector)
  n=length(vector);
  for k=2:n,
    nv(k-1)=vector(k);
  end
  pieza = vector(1)
```

La tabla de objetos resulta ser en este caso:

Objeto	Identificador	Tipo	Valor
Vector que contiene los identificadores de las piezas en la cola. Argumento.	`vector`	vector de enteros	variable
Nuevo vector que incluye la pieza recién entrada. Resultado.	`nv`	vector de enteros	variable
Número que identifica la pieza que ha salido de la cola. Resultado.	`pieza`.	entero	variable
Número de piezas en la cola antes de salir la primera. Variable auxiliar.	`n`	entera	variable
Función de biblioteca para calcular el número de elementos de un vector.	`length`		

9.11.2 Reordenación de matrices

Se desea codificar una función de MATLAB que sirva para reordenar las filas de una matriz. Dada una matriz $A \in \mathbb{R}^{m\times n}$ la tarea consiste en colocar las filas de la matriz en un nuevo orden indicado por un vector de índices v. Las componentes del vector v son m números enteros tales que $1 \leq v_k \leq m \ \forall k = 1, \cdots, m$.

La reordenación tiene lugar del siguiente modo: la primera fila de A ha de colocarse en el lugar indicado por v_1, la segunda fila de A ha de colocarse en el lugar indicado por v_2, etc.

La función se llamará `mezclar` y admitirá como únicos argumentos: la matriz A, Las dimensiones de A, m y n y el vector v. La función devolverá únicamente la matriz A con sus filas debidamente reordenadas. A fin de aclarar el ejercicio considere el siguiente ejemplo de uso:

```
>> A = [ 0.5 0.5 0.5; 7 7 7];
>> mezclar(A, 2,3, [2 1])
ans =
    7.0000    7.0000    7.0000
    0.5000    0.5000    0.5000
```

Se observa que el resultado obtenido al ejecutar la función `mezclar` es `ans` que resulta ser la matriz A con sus filas cambiadas de lugar.

Solución. El ejercicio se resuelve copiando las filas de la matriz A en otra matriz B teniendo cuidado de colocar la fila k en el lugar indicado por v_k; es decir, se ha de verificar que para todo $j = 1, \cdots, n$ se cumpla que $b_{p,j} = a_{k,j}$ siendo $p = v_k$ el nuevo lugar para la fila k.

```
function [B] = mezclar(A, m, n, v)
 for fila=1:m,
    nuevolugar = v(fila);
    for columna=1:n,
      B(nuevolugar, columna) = A(fila, columna);
    end
 end
```

9.11.3 Ejercicios propuestos

Presente mediante diagramas de flujo y tablas de objetos algoritmos que resuelvan los problemas planteados. Posteriormente codifique la solución en MATLAB. No utilice funciones propias de MATLAB (como seno, coseno, raíz cuadrada, etc.) a menos que se indique expresamente. En su lugar escriba el código de sus propias funciones. Recuerde que debe usar únicamente el conjunto elemental de operaciones que se permite para los diagramas de flujo.

1. Sea un número entero n y un número real x, y que calcule mediante una función la expresión x^n. Preste atención a los diferentes casos para $n > 0$, $n = 0$ y $n < 0$.

2. Sean dos enteros positivos m y n, escriba el código de una función que calcule el número combinatorio $\binom{m}{n}$.

3. Sea un vector $v \in \mathbb{R}^{n\times 1}$ con $n > 1$ entero, escriba el código de una función que calcule $v^t \cdot v$.

4. Los coeficientes de un polinomio de grado $n > 1$ son las componentes de un vector, $a \in \mathbb{R}^{1\times(n+1)}$, $a = (a_1, a_2, ..., a_n, a_{n+1})^t$, teniendo el polinomio la forma:

$$\mathrm{A}(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}.$$

   Escriba el código de una función que evalúe el polinomio A en un cierto punto x siendo los argumentos el vector a, el grado n y el punto x.

5. Escriba el código de una función que calcule el determinante de una matriz genérica $A \in \mathbb{R}^{2\times 2}$. El argumento de la función será la matriz A.

6. Sea $f : \mathbb{R} \rightarrow \mathbb{R}$ una función definida en un intervalo $[x_{inf}, x_{sup}]$. Suponga que se conocen $n+1$ puntos sobre la curva $y = f(x)$. Sean estos puntos: $P_1 = (x_1, y_1)$, ..., $P_n = (x_n, y_n)$. Suponga además que las abscisas de los puntos están ordenadas, de forma que $x_{inf} = x_1 < x_2 \cdots < x_n = x_{sup}$. Realice una función que calcule la longitud de la curva entre los puntos x_{inf} y x_{sup} aproximando dicha longitud por la suma de las longitudes de los segmentos dados por los puntos P_i, $\forall i = 0, \cdots, n$. Para este ejercicio se ha de usar la raíz cuadrada. Indíquela en el diagrama de flujo con el signo habitual $\sqrt{.}$. Para codificar en MATLAB recuerde que la función es `sqrt()`.

7. Interpolación lineal a trozos. Sea $f : \mathbb{R} \rightarrow \mathbb{R}$ una función definida en un intervalo $[x_{inf}, x_{sup}]$. Suponga que se conocen $n+1$ puntos sobre la curva $y = f(x)$. Sean estos puntos: $P_1 = (x_1, y_1)$, ..., $P_n = (x_n, y_n)$. Suponga además que las abscisas de los puntos están ordenadas, de forma que $x_{inf} = x_1 < x_2 \cdots < x_n = x_{sup}$.

   Se necesita una función que calcule la aproximación lineal o estimación de $f(x)$ interpolando mediante un segmento en el subintervalo adecuado.

8. Función que calcule el adjunto $Adj(a_{kj})$ en una matriz $A \in \mathbb{R}^{m \times n}$ dados A, k y j.

9. Se desea realizar varias funciones en MATLAB para trabajar con parábolas del tipo $y = ax^2 + bx + c$ en un intervalo $[x_{inf}, x_{sup}]$.

   En primer lugar se precisa una función que requiera del usuario los valores a, b, c que definen la parábola y los puntos inicial y final del intervalo x_{inf} y x_{sup}. La función devolverá los valores proporcionados por el usuario.

   En segundo lugar realice una función que dibuje en la pantalla la gráfica de la parábola en el intervalo dado. Es necesario utilizar la función de MATLAB `plot`.

   Finalmente escriba una función que calcule y devuelva el menor valor tomado por la parábola en el intervalo. Tenga en cuenta los casos posibles en función de a. El cálculo ha de ser exacto, no aproximado.

10. Función que realice el trazado gráfico de una curva polinómica $y = P(x)$ entre dos abscisas x_i y x_f con $x_i < x_f$ dados el vector de coeficientes del polinomio, las abscisas inicial y final y el número de segmentos ns en que se descompondrá la curva para ser trazada. Es necesario utilizar la función de MATLAB `plot`.

11. Función para el cálculo aproximado de máximos de un polinomio $P(x)$ en un intervalo $[x_i, x_f]$ comprobando las ordenadas de todos los puntos en el intervalo en una rejilla de ancho h dado.

12. Función que evalúe la derivada dy/dx de una función $y = P(x)$, siendo P un polinomio. La derivada ha de evaluarse en en un punto z. Los argumentos son el vector de coeficientes del polinomio y el punto en el que se obtendrá la derivada.

13. Función que evalúe en un punto z la derivada $n-$ésima $d^n y/dx^n$ siendo $y = P(x)$ un polinomio. Los argumentos son el vector de coeficientes del polinomio y el punto en el que se obtendrá la derivada.

14. Utilice las dos funciones anteriores para resolver el problema de calcular los ceros (raíces) de polinomios en un intervalo dado siguiendo el método de Newton.

15. Se necesita un conjunto de funciones que permitan realizar operaciones con vectores de $\mathbb{R}^n$. Dichas funciones permitirán a un programa: leer las componentes de un vector v genérico, calcular el producto escalar de dos vectores, calcular el vector suma de dos vectores. Escriba el código MATLAB de las funciones con las siguientes condiciones:

    1. `LeeVect` Para leer un vector de $\mathbb{R}^n$ del teclado.
    2. `ProdEsc` Para multiplicar escalarmente dos vectores v^a y v^b de $\mathbb{R}^n$, $(e = v^a \cdot v^b)$
    3. `SumVect` Para sumar dos vectores de $\mathbb{R}^n$, $(v^s = v^a + v^b)$
    4. `EscVect` Para escribir en la pantalla un vector de $\mathbb{R}^n$.

16. Continuando con el ejercicio anterior, escriba además un programa de prueba que utilice estas funciones para leer n, dos vectores v^a y v^b y posteriormente escribir su producto escalar y su suma. Este programa no tendrá variables globales y deberá utilizar exclusivamente las funciones `LeeVect`, `ProdEsc`, `SumVect` y `EscVect`.