Prácticas de la asignatura

Cálculo Numérico II

Resumen de MATLAB

Grado en Matemáticas por la Universidad de Sevilla
Dpto. de Ecuaciones Diferenciales y Análisis Numérico
Universidad de Sevilla

Curso 2014/15

Índice

1 Introducción y elementos básicos

MATLAB es un potente paquete de software para computación científica, orientado al cálculo numérico, a las operaciones matriciales y especialmente a las aplicaciones científicas y de ingeniería. Ofrece lo que se llama un entorno de desarrollo integrado (IDE), es decir, una herramienta que permite, en una sóla aplicación, ejecutar órdenes sencillas, escribir programas utilizando un editor integrado, compilarlos (o interpretarlos), depurarlos (buscar errores) y realizar gráficas.

Puede ser utilizado como simple calculadora matricial, pero su interés principal radica en los cientos de funciones tanto de propósito general como especializadas que posee, así como en sus posibilidades para la visualización gráfica.

MATLAB posee un lenguaje de programación propio, muy próximo a los habituales en cálculo numérico (Fortran, C, . . .), aunque mucho más *tolerante* en su sintaxis, que permite al usuario escribir sus propios scripts (conjunto de comandos escritos en un fichero, que se pueden ejecutar con una única orden) para resolver un problema concreto y también escribir nuevas funciones con, por ejemplo, sus propios algoritmos, o para *modularizar* la resolución de un problema complejo. MATLAB dispone, además, de numerosas Toolboxes, que le añaden funcionalidades especializadas.

Numerosas contribuciones de sus miles de usuarios en todo el mundo pueden encontrarse en la web de The MathWorks: `http://www.mathworks.es`

1.1 Comenzando

Al iniciar MATLAB nos aparecerá una ventana más o menos como la de la Figura 1.1 (dependiendo del sistema operativo y de la versión)

Si la ubicación de las ventanas integradas es diferente, se puede volver a ésta mediante:

Menú Desktop → Desktop Layout → Default

Se puede experimentar con otras disposiciones. Si hay alguna que nos gusta, se puede salvaguardar con

Menú Desktop → Desktop Layout → Save Layout . . .

dándole un nombre, para usarla en otras ocasiones. De momento, sólo vamos a utilizar la ventana principal de MATLAB: Command Window. A través de esta ventana nos comunicaremos con MATLAB, escribiendo las órdenes en la línea de comandos. Los signos >> indican que MATLAB está desocupado, esperando nuestras órdenes.

1.2 Objetos y sintaxis básicos

Las explicaciones sobre las funciones/comandos que se presentan en estas notas están muy resumidas y sólo incluyen las funcionalidades que, según el parecer subjetivo de la autora, pueden despertar más interés. La mayoría de las funciones tienen mas y/o distintas funcionalidades que las que se exponen

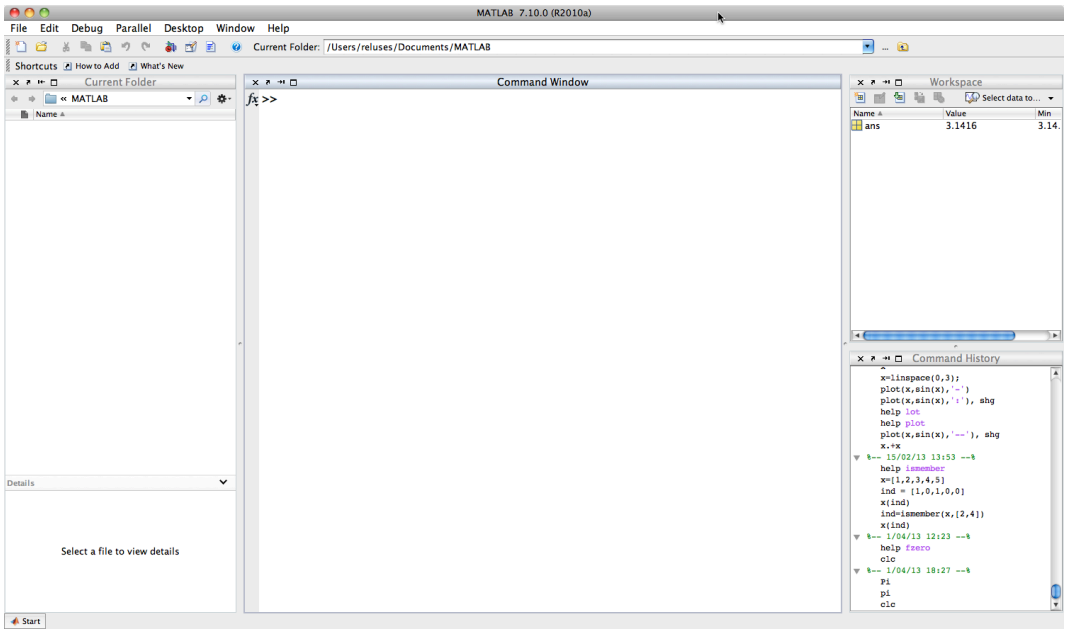


Figura 1.1: La ventana de MATLAB

aquí. Para una descripción exacta y exhaustiva es preciso consultar la Ayuda on-line.

Los tipos básicos de datos que maneja MATLAB son números reales, booleanos (valores lógicos) y cadenas de caracteres (string). También puede manipular distintos tipos de números enteros, aunque sólo suele ser necesario en circunstancias específicas.

En MATLAB, por defecto, los números son codificados como números reales en coma flotante en doble precisión. La precisión, esto es, el número de bits dedicados a representar la mantisa y el exponente, depende de cada (tipo de) máquina.

MATLAB manipula también otros objetos, compuestos a partir de los anteriores: números complejos, matrices, *cells*, estructuras definidas por el usuario, clases Java, etc.

El objeto básico de trabajo de MATLAB es una matriz bidimensional cuyos elementos son números reales o complejos. Escalares y vectores son considerados casos particulares de matrices. También se pueden manipular matrices de cadenas de caracteres, booleanas y enteras.

El lenguaje de MATLAB es **interpretado**, esto es, las instrucciones se traducen a lenguaje máquina una a una y se ejecutan antes de pasar a la siguiente. Es posible escribir varias instrucciones en la misma línea, separándolas por una coma o por punto y coma. Las intrucciones que terminan por punto y coma no producen salida de resultados por pantalla.

Algunas constantes numéricas están predefinidas (ver Tabla 1.1)

MATLAB distingue entre mayúsculas y minúsculas: `pi` no es los mismo que `Pi`.

MATLAB conserva un historial de las instrucciones escritas en la línea de comandos. Se pueden recuperar instrucciones anteriores, usando las teclas de flechas arriba y abajo. Con las flechas izquierda y derecha nos podemos desplazar sobre la línea de comando y modificarlo.

Se pueden salvaguardar todas las instrucciones y la salida de resultados de una sesión de trabajo de MATLAB a un fichero:

```
>> diary nombre_fichero
>> diary off                    % suspende la salvaguarda
```

`i, j`	Unidad imaginaria : `2+3i`, `-1-2j`
`pi`	Número π
`Inf`	*Infinito*, número mayor que el más grande que se puede almacenar. Se produce con operaciones como $x/0$, con $x \neq 0$
`NaN`	*Not a Number*, magnitud no numérica resultado de cálculos indefinidos. Se produce con cálculos del tipo $0/0$ o ∞/∞. `(0+2i)/0` da como resultado `NaN + Inf i`
`eps`, `intmax`, `intmin` `realmax`, `realmin`	Otras constantes. Consultar la ayuda on-line.

Tabla 1.1: Algunas constantes pre-definidas en MATLAB. Sus nombres son reservados y no deberían ser usados para variables ordinarias.

1.2.1 Constantes y operadores

Números reales	`8.01  -5.2  .056  1.4e+5  0.23E-2` `-.567d-21  8.003D-12`
Números complejos	`1+2i  -pi-3j`
Booleanos	`true  false`
Caracteres	Entre apóstrofes: `'esto es una cadena de caracteres (string)'`
Operadores aritméticos	`+  -  *  /  ^`
Operadores de comparación	`==  ~= (ó <>)  <  >  <=  >`
Operadores lógicos (lo dos últimos sólo para escalares)	`&  |  ~  &&  ||` `&&` y `||` no evalúan el operando de la derecha si no es necesario.)

Tabla 1.2: Constantes y operadores de diversos tipos.

1.2.2 Funciones elementales

Los nombres de las funciones elementales son los *habituales*.

Los argumentos pueden ser, siempre que tenga sentido, reales o complejos y el resultado se devuelve en el mismo tipo del argumento.

La lista de todas las funciones matemáticas elementales se puede consultar en:

Help → MATLAB → Functions: By Category → Mathematics → Elementary Math

Algunas de las más habituales se muestran en la Tabla 1.3:

1.2.3 Uso como calculadora

Se puede utilizar MATLAB como simple calculadora, escribiendo expresiones aritméticas y terminando por RETURN (`<R>`). Se obtiene el resultado inmediatamente a través de la variable del sistema `ans` (de *answer*). Si no se desea que MATLAB escriba el resultado en el terminal, debe terminarse la orden por punto y coma (útil, sobre todo en programación).

`sqrt(x)`	raiz cuadrada	`sin(x)`	seno (radianes)
`abs(x)`	módulo	`cos(x)`	coseno (radianes)
`conj(z)`	complejo conjugado	`tan(z)`	tangente (radianes)
`real(z)`	parte real	`cotg(x)`	cotangente (radianes)
`imag(z)`	parte imaginaria	`asin(x)`	arcoseno
`exp(x)`	exponencial	`acos(x)`	arcocoseno
`log(x)`	logaritmo natural	`atan(x)`	arcotangente
`log10(x)`	logaritmo decimal	`cosh(x)`	cos. hiperbólico
`rat(x)`	aprox. racional	`sinh(x)`	seno hiperbólico
`mod(x,y)` `rem(x,y)`	resto de dividir x por y Iguales si $x, y > 0$. Ver help para definición exacta	`tanh(x)`	tangente hiperbólica
`fix(x)`	Redondeo hacia 0	`acosh(x)`	arcocoseno hiperb.
`ceil(x)`	Redondeo hacia $+\infty$	`asinh(x)`	arcoseno hiperb.
`floor(x)`	Redondeo hacia $-\infty$	`atanh(x)`	arcotangente hiperb.
`round(x)`	Redondeo al entero más próximo		

Tabla 1.3: Algunas funciones matemáticas elementales.

Ejemplos 1.1

```
>> sqrt(34*exp(2))/(cos(23.7)+12)
   ans  =
        1.3058717

>> 7*exp(5/4)+3.54
   ans  =
        27.97240

>> exp(1+3i)
   ans  =
        - 2.6910786 + 0.3836040i
```

1.2.4 Variables

En MATLAB las variables no son nunca declaradas: su tipo y su tamaño cambian de forma dinámica de acuerdo con los valores que le son asignados. Así, una misma variable puede ser utilizada, por ejemplo, para almacenar un número complejo, a continuación una matriz 25×40 de números enteros y luego para almacenar un texto. Las variables se crean automáticamente al asignarles un contenido. Asimismo, es posible eliminar una variable de la memoria si ya no se utiliza.

Ejemplos 1.2

```
>> a=10
   a =
        10.
>> pepito=exp(2.4/3)
   pepito  =
        2.2255
>> pepito=a+pepito*(4-0.5i)
   pepito  =
        18.9022 - 1.1128i
>> clear pepito
```

Para conocer en cualquier instante el valor almacenado en una variable basta con teclear su nombre (Atención: recuérdese que las variables `AB`, `ab`, `Ab` y `aB` **son distintas**, ya que MATLAB distingue entre mayúsculas y minúsculas).

Otra posibilidad es hojear el Workspace ó espacio de trabajo, abriendo la ventana correspondiente. Ello nos permite ver el contenido de todas las variables existentes en cada momento e, incluso, modificar su valor.

Algunos comandos relacionados con la inspección y eliminación de variables se describen en la tabla siguiente:

`who`	lista las variables actuales
`whos`	como el anterior, pero más detallado
`clear`	elimina todas las variables que existan en ese momento
`clear a b c`	elimina las variables `a`, `b` y `c` (atención: sin comas!)

1.2.5 Formatos

Por defecto, MATLAB muestra los números en formato de punto fijo con 5 dígitos. Se puede modificar este comportamiento mediante el comando `format`

`format`	Cambia el formato de salida a su valor por defecto, short
`format short`	El formato por defecto
`format long`	Muestra 15 dígitos
`format short e`	Formato short, en coma flotante
`format long e`	Formato long, en coma flotante
`format rat`	Muestra los números como cociente de enteros

1.2.6 Algunos comandos utilitarios del sistema operativo

Están disponibles algunos comandos utilitarios, como

`ls` `dir`	Lista de ficheros del directorio de trabajo
`pwd`	Devuelve el nombre y la ruta (*path*) del directorio de trabajo
`cd`	Para cambiar de directorio
`clc`	Limpia la ventana de comandos Command Window
`date`	Fecha actual

Tabla 1.4: Comandos utilitarios.

1.3 Documentación y ayuda *on-line*

- Ayuda on-line en la ventana de comandos

```
>> help nombre_de_comando
```

  La información se obtiene en la misma ventana de comandos. Atención:

- Ayuda on-line con ventana de navegador

```
>> helpwin
```

  ó bien Menú Help ó bien Botón Start → Help.

Además, a través del navegador del Help se pueden descargar, desde The MathWorks, guías detalladas, en formato pdf, de cada capítulo.

1.4 *Scripts* y funciones. El editor integrado

1.4.1 *Scripts*

En términos generales, en informática, un *script* (guión o archivo por lotes) es un conjunto de instrucciones (programa), usualmente simple, guardadas en un fichero (usualmente de texto plano) que son ejecutadas normalmente mediante un intérprete. Son útiles para automatizar pequeñas tareas. También puede hacer las veces de un "programa principal" para ejecutar una aplicación.

Así, para llevar a cabo una tarea, en vez de escribir las instrucciones una por una en la línea de comandos de MATLAB, se pueden escribir las órdenes una detrás de otra en un fichero. Para ello se puede utilizar el Editor integrado de MATLAB. Para iniciarlo, basta pulsar el icono *hoja en blanco* (New script) de la barra de MATLAB, o bien

File → New → Script

UN *script* de MATLAB debe guardarse en un fichero con sufijo `.m` para ser reconocido. El nombre del fichero puede ser cualquiera *razonable*, es decir, sin acentos, sin espacios en blanco y sin caracteres «extraños».

Para editar un *script* ya existente, basta hacer *doble-click* sobre su icono en la ventana Current Folder.

Para ejecutar un *script* que esté en el directorio de trabajo, basta escribir su nombre (sin el sufijo) en la linea de comandos.

1.4.2 M-Funciones

Una función (habitualmente denominadas M-funciones en MATLAB), es un programa con una «interfaz» de comunicación con el exterior mediante argumentos de entrada y de salida.

Las funciones MATLAB responden al siguiente formato de escritura:

```
function [argumentos de salida] = nombre(argumentos de entrada)
%
% comentarios
%
....
instrucciones (normalmente terminadas por  ;  para evitar eco en pantalla)
....
```

Las funciones deben guardarse en un fichero con el mismo nombre que la función y sufijo `.m`. Lo que se escribe en cualquier línea detrás de `%` es considerado como comentario.

Ejemplo 1.3
El siguiente código debe guardarse en un fichero de nombre `areaequi.m`.

```
function [sup] = areaequi(long)
%
% areaequi(long) devuelve el area del triangulo
%                equilatero de lado = long
%

sup = sqrt(3)*long^2/4;
```

La primera linea de una M-función siempre debe comenzar con la claúsula (palabra reservada) `function`. El fichero que contiene la función debe estar *en un sitio en el que MATLAB lo pueda encontrar*, normalmente, en la carpeta de trabajo.

Se puede ejecutar la M-función en la misma forma que cualquier otra función de MATLAB:

Ejemplos 1.4
```
>> areaequi(1.5)
ans =
    0.9743
>> rho = 4 * areaequi(2) +1;
>> sqrt(areaequi(rho))
```

Los breves comentarios que se incluyen a continuación de la línea que contiene la claúsula `function` deben explicar, brevemente, el funcionamiento y uso de la función. Además, constituyen la ayuda *on-line* de la función:

Ejemplo 1.5

```
>> help areaequi
  areaequi(long) devuelve el area del triangulo
                 equilatero de lado = long
```

Se pueden incluir en el mismo fichero otras funciones, denominadas subfunciones, a continuación de la primera[1], pero sólo serán *visibles* para las funciones del mismo fichero. **NO** se pueden incluir M-funciones en el fichero de un *script*.

1.4.3 Funciones anónimas

Algunas funciones *sencillas*, que devuelvan el resultado de una expresión, se pueden definir mediante una sóla instrucción, en mitad de un programa (script o función) o en la línea de comandos. Se llaman funciones anónimas. La sintaxis para definirlas es:

```
nombre_funcion = @(argumentos) expresion_funcion
```

Ejemplo 1.6 (Función anónima para calcular el área de un círculo)

```
>> area_circulo = @(r) pi * r.^2;
>> area_circulo(1)
ans =
    3.1416
>> semicirc = area_circulo(3)/2
semicirc =
    14.1372
```

Las funciones anónimas pueden tener varios argumentos y hacer uso de variables previamente definidas:

Ejemplo 1.7 (Función anónima de dos variables)

```
>> a = 2;
>> mifun = @(x,t) sin(a*t).*cos(t/a);
>> mifun(pi/4,1)
ans =
    0.7980
```

Si, con posterioridad a la definición de la función `mifun`, se cambia el valor de la variable `a`, la función no se modifica: en el caso del ejemplo, seguirá siendo `mifun(x,t)=sin(2*t).*cos(t/2)`.

[1] También es posible definir funciones anidadas, esto es, funciones «insertadas» dentro del código de otras funciones. (Se informa aquí para conocer su existencia. Su utilización es delicada.)

1.5 *Workspace* y ámbito de las variables

Workspace (espacio de trabajo) es el conjunto de variables que en un momento dado están definidas en la memoria del MATLAB.

Las variables creadas desde la linea de comandos de MATLAB pertenecen al Base Workspace (espacio de trabajo base; es el que se puede «hojear» en la ventana Workspace). Los mismo sucede con las variables creadas por un *script* que se ejecuta desde la linea de comandos. Estas variables permanecen en el Base Workspace cuando se termina la ejecución del script y se mantienen allí durante toda la sesión de trabajo o hasta que se borren.

Sin embargo, las variables creadas por una M-función pertenecen al espacio de trabajo de dicha función, que es independiente del espacio de trabajo base. Es decir, las variables de las M-funciones son **locales**: MATLAB reserva una zona de memoria cuando comienza a ejecutar una M-función, almacena en esa zona las variables que se crean dentro de ella, y «borra» dicha zona cuando termina la ejecución de la función.

Esta es una de las principales diferencias entre un *script* y una M-función: cuando finaliza la ejecución de un *script* se puede «ver» y utilizar el valor de todas las variables que ha creado el script en el Workspace; en cambio, cuando finaliza una función no hay rastro de sus variables en el Workspace.

Para hacer que una variable local de una funcion pertenezca al Base Workspace, hay que declararla `global`: la orden

```
global a suma error
```

en una función hace que las variables `a`, `suma` y `error` pertenezcan al Base Workspace y por lo tanto, seguirán estando disponibles cuando finalize la ejecución de la función.

1.6 Matrices

Como ya se ha dicho, las matrices bidimensionales de números reales o complejos son los objetos básicos con los que trabaja MATLAB. Los vectores y escalares son casos particulares de matrices.

1.6.1 Construcción de matrices

La forma más elemental de introducir matrices en MATLAB es describir sus elementos de forma exhaustiva (por filas y entre corchetes rectos `[ ]`) : elementos de una fila se separan unos de otros por comas y una fila de la siguiente por punto y coma.

Ejemplos 1.8 (Construcciones elementales de matrices)

```
>> v = [1, -1, 0, sin(2.88)]                    % vector fila longitud 4

>> w = [0; 1.003; 2; 3; 4; 5*pi]                % vector columna longitud 6

>> a = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12]  % matriz 3x4
```

Observación. El hecho de que, al introducirlas, se escriban las matrices por filas no significa que internamente, en la memoria del ordenador, estén así organizadas: **en la memoria las matrices se almacenan como un vector unidimensional ordenadas por columnas**.

Ejemplos 1.9 (Otras órdenes para crear matrices)

```
>> v1 = a:h:b      % crea un vector fila de números regularmente espaciados
                   %  a  a+h  a+2h ... hasta  c <= b < c+h

>> v2 = a:b        % como el anterior, con paso h=1

>> v3 = v2'        % matriz traspuesta (conjugada si es compleja)

>> v4 = v2.'       % matriz traspuesta sin conjugar
```

Se pueden también utilizar los vectores/matrices como objetos para construir otras matrices (bloques):

Ejemplos 1.10 (Matrices construidas con bloques)

```
>> v1 = 1:4;

>> v2 = [v1, 5; 0.1:0.1:0.5]

>> v3 = [v2', [11,12,13,14,15]']
```

$$v_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix} \quad v_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 0.1 & 0.2 & 0.3 & 0.4 & 0.5 \end{pmatrix} \quad v_3 = \begin{pmatrix} 1 & 0.1 & 11 \\ 2 & 0.2 & 12 \\ 3 & 0.3 & 13 \\ 4 & 0.4 & 14 \\ 4 & 0.5 & 15 \end{pmatrix}$$

Las siguientes funciones generan vectores de elementos regularmente espaciados, útiles en muchas circunstancias, especialmente para creación de gráficas.

`linspace(a,b,n)`	Si `a` y `b` son números reales y `n` un número entero, genera una partición regular del intervalo `[a,b]` con `n` nodos (`n-1` subintervalos)
`linspace(a,b)`	Como el anterior, con `n`=100

Las siguientes funciones generan algunas matrices especiales que serán de utilidad.

`zeros(n,m)`	matriz `n`×`m` con todas sus componentes iguales a cero
`ones(n,m)`	matriz `n`×`m` con todas sus componentes iguales a uno
`eye(n,m)`	matriz unidad `n`×`m`: diagonal principal = 1 y el resto de las componentes = 0
`diag(v)`	Si `v` es un vector, es una matriz cuadrada de ceros con diagonal principal = `v`
`diag(A)`	Si `A` es una matriz, es su diagonal principal

MATLAB posee, además, decenas de funciones útiles para generar distintos tipos de matrices. Para ver una lista exhaustiva consultar:

Help → MATLAB → Functions: By Category → Mathematics → Arrays and Matrices

1.6.2 Operaciones con vectores y matrices

Los operadores aritméticos representan las correspondientes operaciones matriciales siempre que tengan sentido.

Sean `A` y `B` dos matrices de elementos respectivos a_{ij} y b_{ij} y sea `k` un escalar.	
`A+B`, `A-B`	matrices de elementos respectivos $a_{ij}+b_{ij}$, $a_{ij}-b_{ij}$ (si las dimensiones son iguales)
`A+k`, `A-k`	matrices de elementos respectivos $a_{ij}+k$, $a_{ij}-k$.
`k*A`, `A/k`	matrices de elementos respectivos $k\,a_{ij}$, $\frac{1}{k}\,a_{ij}$
`A*B`	producto matricial de `A` y `B` (si las dimensiones son adecuadas)
`A^n`	Si `n` es un entero positivo, `A*A*...*A`

Además de estos operadores, MATLAB dispone de ciertos operadores aritméticos que operan **elemento a elemento**. Son los operadores `.*` `./` y `.^`, muy útiles para aprovechar las funcionalidades vectoriales de MATLAB.

Sean `A` y `B` dos matrices de las mismas dimensiones y de elementos respectivos a_{ij} y b_{ij} y sea `k` un escalar.	
`A.*B`	matriz de la misma dimensión que `A` y `B` de elementos $a_{ij}\times b_{ij}$
`A./B`	ídem de elementos $\frac{a_{ij}}{b_{ij}}$
`A.^B`	ídem de elementos $a_{ij}{}^{b_{ij}}$
`k./A`	matriz de la misma dimensión que `A`, de elementos $\frac{k}{a_{ij}}$
`A.^k`	ídem de elementos a_{ij}^{k}
`k.^A`	ídem de elementos $k^{a_{ij}}$

Por otra parte, la mayoría de las funciones MATLAB están hechas de forma que admiten matrices como argumentos. Esto se aplica en particular a las funciones matemáticas elementales y su utilización

debe entenderse en el sentido de **elemento a elemento**. Por ejemplo, si `A` es una matriz de elementos a_{ij}, `exp(A)` es otra matriz con las mismas dimensiones que `A`, cuyos elementos son $e^{a_{ij}}$.

Algunas otras funciones útiles en cálculos matriciales son:

`sum(v)`	suma de los elementos del vector `v`
`sum(A)` `sum(A,1)`	suma de los elementos de la matriz `A`, por columnas
`sum(A,2)`	suma de los elementos de la matriz `A`, por filas

`prod(v)`, `prod(A)`
`prod(A,1)`
`prod(A,2)`
	como la suma, pero para el producto

`max(v)`, `min(v)`	máximo/mínimo elemento del vector `v`
`max(A)`, `min(A)`	máximo/mínimo elemento de la matriz `A`, por columnas
`mean(v)`	promedio de los elementos del vector `v`
`mean(A)`	promedio de los elementos de la matriz `A`, por columnas.

1.7 Dibujo de curvas

La representación gráfica de una curva en un ordenador es una linea poligonal construida uniendo mediante segmentos rectos un conjunto discreto y ordenado de puntos: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.

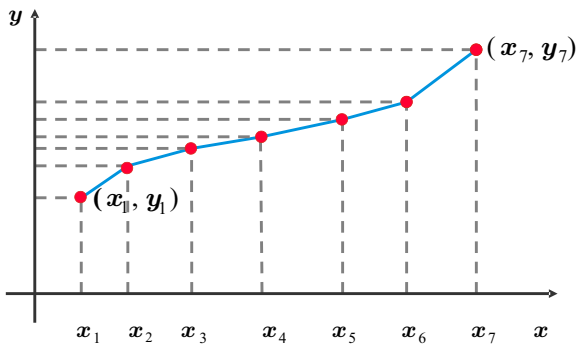


Figura 1.2: Linea poligonal determinada por un conjunto de puntos.

La línea así obtenida tendrá mayor apariencia de «suave» cuanto más puntos se utilicen para construirla, ya que los segmentos serán imperceptibles.

Para dibujar una curva plana en MATLAB se usa el comando

```
plot(x,y)
```

siendo `x` e `y` dos vectores de las mismas dimensiones conteniendo, respectivamente, las abscisas y las ordenadas de los puntos de la gráfica.

Ejemplo 1.11
Dibujar la curva $y = \dfrac{x^2+2}{x+5}$ para $x \in [-2, 3]$

```
>> f = @(x) (x.^2+2)./(x+5);
>> x = linspace(-2,3);
>> plot(x,f(x))
```

Se pueden dibujar dos o más curvas de una sóla vez, proporcionando al comando `plot` varios pares de vectores abscisas-ordenadas, como en el ejemplo siguiente.

Ejemplo 1.12
Dibujar las curvas $y = 2\,\mathrm{sen}^3(x)\,\cos^2(x)$ e $y = e^x - 2x - 3$ para $x \in [-1.5, 1.5]$

```
>> f1 = @(x) 2 * sin(x).^3 .* cos(x).^2;
>> f2 = @(x) exp(x) - 2*x -3;
>> x = linspace(-1.5,1.5);
>> plot(x,f1(x),x,f2(x))
```

A cada par de vectores abscisas-ordenadas en el comando `plot` se puede añadir un argumento opcional de tipo cadena de caracteres, que modifica el aspecto con que se dibuja la curva. Para más información, hojear el help (`help plot`).

Ejemplo 1.13
Las siguientes órdenes dibujan la curva $y = \mathrm{sen}^3(x)$ en color rojo (`r`, de **r**ed) y con marcadores $*$, en lugar de una línea contínua.

```
>> x = linspace(0,pi,30);
>> plot(x,sin(x).^3,'r*')
```

La orden siguiente dibuja la curva $y = \mathrm{sen}^3\, x$ en color negro (`k`, de blac**k**), con marcadores $*$ y con línea punteada, y la curva $y = \cos^2 x$ en color azul (`b`, de **b**lue) y con marcadores $+$

```
>> plot(x,sin(x).^3,'k*:', x, cos(x).^2,'b+')
```

Ademas, mediante argumentos opcionales, es posible modificar muchas otras propiedades de las curvas. Esto se hace siempre mediante un par de argumentos en la orden de dibujo que indican

```
'Nombre de la Propiedad', Valor de la propiedad
```

Esta propiedad afectará a todas las curvas que se dibujen con la misma orden.

Ejemplo 1.14
Para establecer un grosor determinado de las líneas se usa la propiedad `LineWidth` (atención a las mayúsculas):

```
>> plot(x,sin(x).^3,'k*:', x, cos(x).^2,'b+', 'LineWidth', 1.1)
```

Se pueden añadir elementos a la gráfica, para ayudar a su comprensión. Para añadir una leyenda que identifique cada curva se usa el comando siguiente, que asigna las leyendas a las curvas en el orden en que han sido dibujadas.

```
legend('Leyenda1', 'Leyenda2')
```

Para añadir etiquetas a los ejes que aclaren el significado de las variables se usan los comandos

```
xlabel('Etiqueta del eje OX')
ylabel('Etiqueta del eje OY')
```

Se puede añadir una cuadrícula mediante la orden

```
grid on
```

También es muy útil la orden, siguiente, que define las coordenadas mínimas y máxima del rectángulo del plano OXY que se visualiza en la gráfica.

```
axis([xmin, xmax, ymin, ymax])
```

Cada nueva orden de dibujo borra el contenido previo de la ventana gráfica, si existe. Para evitar esto existen la órdenes

```
hold on
hold off
```

La orden `hold on` permanece activa hasta que se cierre la ventana gráfica o bien se dé la orden `hold off`

Ejemplos 1.15
Las siguientes órdenes darán como resultado la gráfica de la figura 1.3

```
x = linspace(0,pi,30);
axis([-0.5,pi+0.5,-0.5,1.5])
hold on
plot(x,sin(x).^3,'g', x, cos(x).^2,'b+', 'LineWidth', 1.1)
x = linspace(-0.95, pi);
plot(x, log(x+1)/2, 'r', 'LineWidth', 1.1)
plot([-5,5], [0, 0], 'k', 'LineWidth', 1)
plot([0, 0], [-5,5], 'k', 'LineWidth', 1)
legend('Leyenda1', 'Leyenda2', 'Leyenda3')
xlabel('Etiqueta del eje OX')
ylabel('Etiqueta del eje OY')
hold off
shg
```

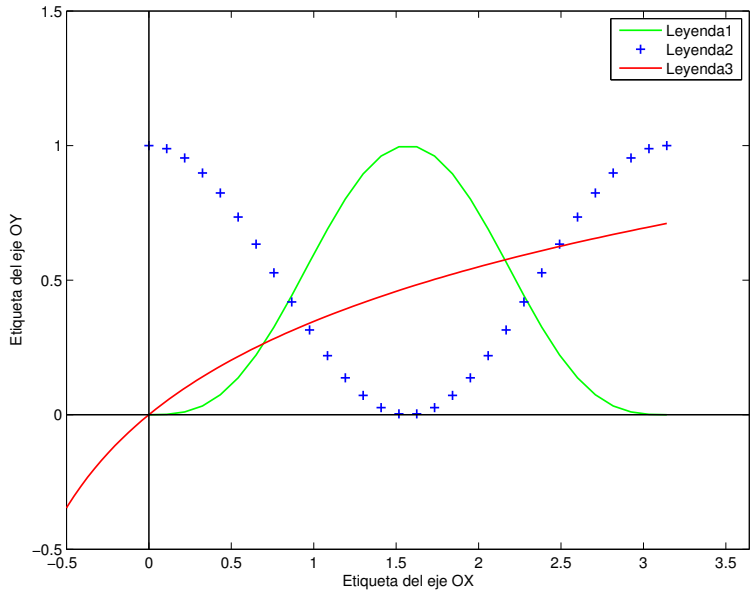


Figura 1.3: Varias curvas con leyenda y etiquetas.

2 Programación con MATLAB

Los condicionales y los bucles o repeticiones son la base de la programación estructurada. Sin ellas, las instrucciones de un programa sólo podrían ejecutarse en el orden en que están escritas (orden secuencial). Las estructuras de control permiten modificar este orden y, en consecuencia, desarrollar estrategias y algoritmos para resolver los problemas.

Los **condicionales** permiten que se ejecuten conjuntos distintos de instrucciones, en función de que se verifique o no determinada condición.

Los **bucles** permiten que se ejecute repetidamente un conjunto de instrucciones, ya sea un número pre-determinado de veces, o bien mientras que se verifique una determinada condición.

2.1 Estructuras condicionales: if

Son los mecanismos de programación que permiten "romper" el flujo secuencial en un programa: es decir, permiten hacer una tarea si se verifica una determinada **condición** y otra distinta si no se verifica.

Evaluar si se verifica o no una condición se traduce, en programación, en averiguar si una determinada **expresión con valor lógico** da como resultado **verdadero** o **falso**

Las estructuras condicionales básicas son las representadas en los diagramas de flujo siguientes:

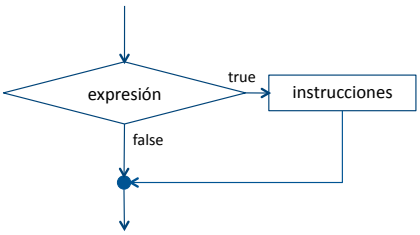


Figura 2.1: Estructura condicional simple: Si `expresión` toma el valor lógico `true`, se ejecutan las instrucciones del bloque `instrucciones` y, después, el programa se continúa ejecutando por la instrucción siguiente. Si, por el contrario, la `expresión` toma el valor lógico `false`, no se ejecutan las `instrucciones`, y el programa continúa directamente por la instrucción siguiente.

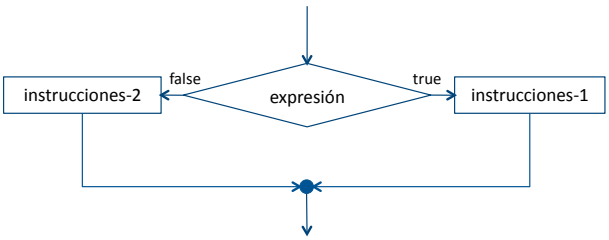


Figura 2.2: Estructura condicional doble: Si `expresión` toma el valor lógico `true`, se ejecutan las instrucciones del bloque `instrucciones 1`. Si, por el contrario, la `expresión` toma el valor lógico `false`, se ejecuta el bloque de `instrucciones 2`. En ambos casos, el programa continúa ejecutándose por la instrucción siguiente.

En casi todos los lenguajes de programación, este tipo de estructuras se implementan mediante una instrucción (o super-instrucción) denominada `if`, cuya sintaxis puede variar ligeramente de unos lenguajes a otros. En MATLAB, concretamente, su forma es la siguiente:

```
instruccion-anterior
if expresion
  bloque-de-instrucciones
end
instruccion-siguiente
```

Esta super-instrucción tiene el funcionamiento siguiente: Se evalúa `expresion`. Si el resultado es `true`, se ejecuta el `bloque-de-instrucciones` y, cuando se termina, se continúa por `instruccion-siguiente`. Si el resultado no es `true`, se va directamente a `instruccion-siguiente`.

```
instruccion-anterior
if expresion
  bloque-de-instrucciones-1
else
  bloque-de-instrucciones-2
end
instruccion-siguiente
```

Su funcionamiento es el siguiente: Se evalúa `expresion`. Si el resultado es `true`, se ejecuta el `bloque-de-instrucciones-1` y, cuando se termina, se continúa por `instruccion-siguiente`. Si el resultado no es `true`, se ejecuta el `bloque-de-instrucciones-2` y cuando se termina se va a la `instruccion-siguiente`.

Ejemplo 2.1 (Uso de un condicional simple)
Escribir una M-función que, dado $x \in \mathbb{R}$, devuelva el valor en x de la función definida a trozos

$$f(x) = \begin{cases} x+1 & \text{si } x < -1, \\ 1-x^2 & \text{si } x \geq -1. \end{cases}$$

```
function [fx] = mifun(x)
%
%  v = mifun(x)  devuelve el valor en x de la función
%                f(x) = x+1   si x < -1
%                f(x) = 1-x^2 si no
%
fx = x + 1;
if x > -1
  fx = 1 - x^2;
end
```

Ejemplo 2.2 (Uso de un condicional doble)
Escribir una M-función que, dados dos números $x, y \in \mathbb{R}$, devuelva un vector cuyas componentes sean los dos números ordenados en orden ascendente.

```
function [v] = Ordena(x, y)
%
%  v = Ordena(x, y)  es un vector que contiene los dos
%          numeros x e y ordenados en orden creciente
%
if x <= y
  v = [x, y];
else
  v = [y, x];
end
```

Observación. La orden de MATLAB `sort([x,y])` tiene el mismo efecto que esta M-función.

Estas estructuras se pueden "anidar", es decir, se puede incluir una estructura condicional dentro de uno de los bloques de instrucciones de uno de los casos de otra.

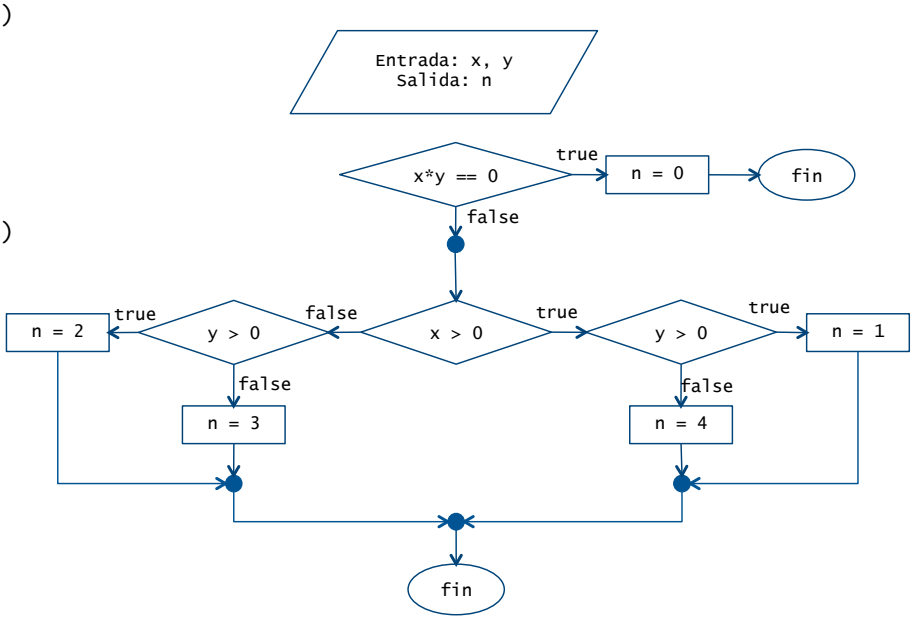
En ocasiones interesa finalizar la ejecución de un programa en algún punto "intermedio" del mismo, es decir, no necesariamente despues de la última instrucción que aparece en el código fuente. Esto se hace mediante la orden

```
return
```

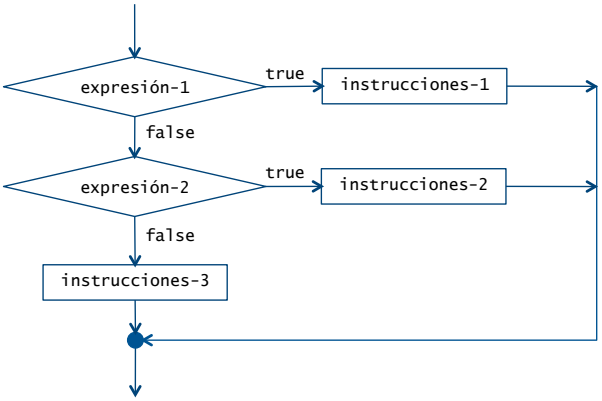
Ejemplo 2.3 (Condicionales anidados)
Escribir una M-función que, dados dos números $x, y \in \mathbb{R}$, devuelva el número del cuadrante del plano OXY en el que se encuentra el punto (x, y). Si el punto (x, y) está sobre uno de los ejes de coordenadas se le asignará el número 0.

```
function [n] = Cuadrante(x, y)
%
%  n = Cuadrante(x, y)  es el número del cuadrante del plano OXY
%          en el que se encuentra el punto (x,y).
%          n = 0 si el punto está sobre uno de los ejes.
%
if x*y == 0
  n = 0;
  return
end

if (x > 0)
  if (y > 0)
    n=1;
  else
    n=4;
  end
else
  if (y > 0)
    n=2;
  else
    n=3;
  end
end
```



Se pueden construir estructuras condicionales más complejas (con más casos). En MATLAB, estas estructuras se implementan mediante la versión más completa de la instrucción `if`:



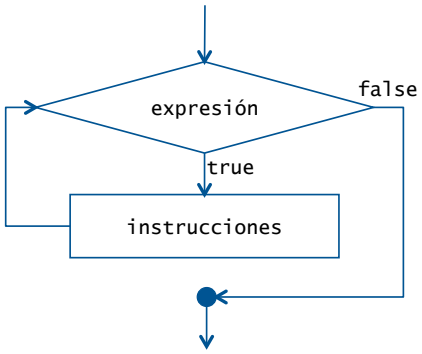

```
instruccion-anterior
if expresion-1
  bloque-de-instrucciones-1
elseif expresion-2
  bloque-de-instrucciones-2
else
  bloque-de-instrucciones-3
end
instruccion-siguiente
```

Esta estructura tiene el funcionamiento siguiente:
Se evalúa `expresion-1`.
Si el resultado es `true`, se ejecuta el `bloque-de-instrucciones-1` y, cuando se termina, se continúa por `instruccion-siguiente`.
Si el resultado de `expresion-1` no es `true`, se evalúa `expresion-2`.
Si el resultado es `true`, se ejecuta el `bloque-de-instrucciones-2` y, cuando se termina, se continúa por `instruccion-siguiente`.
Si el resultado de `expresion-2` no es `true`, se ejecuta el `bloque-de-instrucciones-3` y luego se va a `instruccion-siguiente`.

Es **muy importante** darse cuenta de que, en cualquier caso, se ejecutará **sólo uno** de los bloques de instrucciones.
La cláusula `else` (junto con su correspondiente bloque-de-instrucciones) puede no existir. En este caso es posible que no se ejecute ninguno de los bloques de instrucciones.

2.2 Estructuras de repetición o bucles: while

Este mecanismo de programación permite repetir un grupo de instrucciones **mientras que** se verifique una cierta condición. Su diagrama de flujo y su sintaxis en MATLAB son los siguientes:



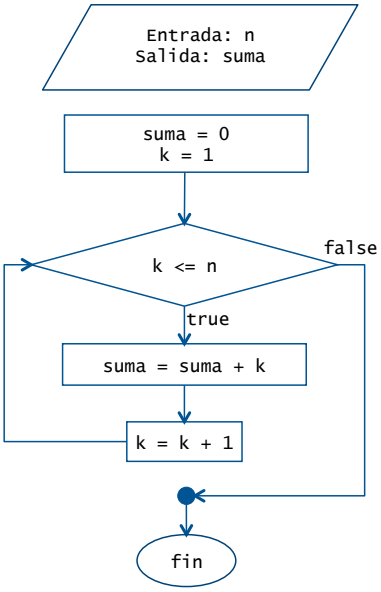

```
instruccion-anterior
while  expresion
  bloque-de-instrucciones
end
instruccion-siguiente
```

Esta estructura tiene el funcionamiento siguiente:
Al comienzo, se evalúa `expresion`.
Si el resultado **no es** `true`, se va directamente a la `instruccion-siguiente`. En este caso, no se ejecuta el `bloque-de-instrucciones`.
Si, por el contrario, el resultado de `expresion` es `true`, se ejecuta el `bloque-de-instrucciones`. Cuando se termina, se vuelve a evalúar la `expresion` y se vuelve a decidir.
Naturalmente, este mecanismo precisa que, dentro del `bloque-de-instrucciones` se modifique, en alguna de las repeticiones, el resultado de evaluar `expresion`. En caso contrario, el programa entraría

en un *bucle infinito*. Llegado este caso, se puede detener el proceso pulsando la combinación de teclas CTRL + C.

Ejemplo 2.4 (Uso de un while)
Escribir una M-función que, dado un número natural n, calcule y devuelva la suma de todos los números naturales entre 1 y n.

```
function [suma] = SumaNat(n)
%
%  suma = SumaNat(n)  es la suma de los n primeros números naturales
%
suma = 0;
k    = 1;
while  k <= n
  suma = suma + k;
  k = k + 1;
end
```



Observación. La orden de MATLAB `sum(1:n)` tiene el mismo efecto que `SumaNat(n)`.

Ejercicio.
Partiendo de la M-función `SumaNat` del ejemplo anterior, escribe una M-función que calcule y devuelva la suma de todos los números naturales impares entre 1 y n.

Ejemplo 2.5 (Uso de un while)
Escribir un *script* que genere de forma aleatoria un número natural del 1 a 9 y pida repetidamente al usuario que escriba un número en el teclado hasta que lo acierte.

Observaciones:
(a) La orden `randi(n)` genera de forma aleatoria un número natural entre 1 y n.
(b) La orden `var=input('Mensaje')` escribe `Mensaje` en la pantalla, lee un dato del teclado y lo almacena en la variable `var`.
(c) La orden `beep` emite un sonido.

```
%-----------------------------------------------------------------
%  Script Adivina
%  Este script genera un numero aleatorio entre 1 y 9
%  y pide repetidamente que se teclee un numero hasta acertar
%-----------------------------------------------------------------
%
n = randi(9);

disp(' ')
disp(' Tienes que acertar un numero del 1 al 9')
disp(' Pulsa CTRL + C si te rindes ...')
disp(' ')

M = 0;
while M~=n
    M=input('Teclea un numero del 1 al 9 :  ');
end
beep
disp(' ')
disp('  ¡¡¡¡Acertaste!!!!  ')
```

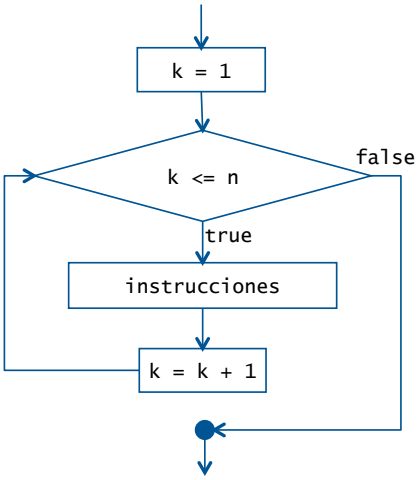
Ejercicio.
Partiendo del *script* `Adivina` del ejemplo anterior, escribe una M-función que reciba como argumento de entrada un número natural m, genere de forma aleatoria un número natural n entre 1 y m, y pida repetidamente al usuario que escriba un número en el teclado hasta que lo acierte.

2.3 Estructuras de repetición o bucles indexados: for

En muchas ocasiones, las repeticiones de un bucle dependen en realidad de una variable entera cuyo valor se va incrementando hasta llegar a uno dado, momento en que se detienen las repeticiones. Esto sucede, especialmente, con los algoritmos que manipulan vectores y matrices, que es lo más habitual cuando se programan métodos numéricos.

En estas ocasiones, para implementar el bucle es preferible utilizar la estructura que se expone en esta sección: **bucle indexado**. En MATLAB (igual que en muchos otros lenguajes) este tipo de mecanismos se implementan mediante una instrucción denominada `for`.

Por ejemplo, el bucle representado mediante el diagrama siguiente, se puede implementar con las órdenes que se indican:




```
instruccion-anterior
for k = 1 : n
  bloque-de-instrucciones
end
instruccion-siguiente
```

Estructura de repetición `for`: Para cada valor de la variable `k` desde `1` hasta `n`, se ejecuta una vez el `bloque-de-instrucciones`. Cuando se termina, el programa se continúa ejecutando por la `instrucción-siguiente`.
Se observa que, con esta instrucción, no hay que ocuparse ni de inicializar ni de incrementar dentro del bucle la variable-índice, `k`. Basta con indicar, junto a la cláusula `for`, el conjunto de valores que debe tomar. Puesto que es la propia instrucción `for` la que gestiona la variable-índice `k`, **está prohibido modificar su valor** dentro del bloque de instrucciones.

El conjunto de valores que debe tomar la variable-índice `k` tiene que ser de números enteros, pero no tiene que ser necesariamente un conjunto de números consecutivos. Son válidos, por ejemplo, los conjuntos siguientes:

```
for k = 0 : 2 : 20           % números pares de 0 a 20
for ind = 30 : -5 : 0        % números 30, 25, 20, 15, 10, 5, 0
for m = [2, 5, 4, 1, 7, 20]  % los números especificados
```

Observación. Siempre que en un bucle sea posible determinar *a priori* el número de veces que se va a repetir el bloque de instrucciones, es preferible utilizar la instrucción `for`, ya que la instrucción `while` es más lenta.

Ejemplo 2.6 (Uso de for)
Escribir una M-función que, dado un vector `v`, calcule el valor máximo entre todos sus componentes.

```
function [vmax] = Maximo(v)
%
%  Maximo(v) es el máximo de las componentes del vector v
%
vmax = v(1);
for k = 2:length(v)
  if v(k) > vmax
    vmax = v(k);
  end
end
```

Ejemplo 2.7 (Uso de for)
Observación:
La orden `error('mensaje')` imprime `mensaje` en la pantalla y detiene la ejecución del programa.

Escribir una M-función que calcule la traza de una matriz.

```
function [y] = Traza(A)
%
% Traza(A) es la suma de los elementos diagonales de la
%          matriz cuadrada A
%
[n,m] = size(A);
if n ~= m
    error('La matriz no es cuadrada')
end
y = 0;
for k = 1:n
    y = y + A(k,k);
end
```

2.4 Ruptura de bucles de repetición: break y continue

En ocasiones es necesario interrumpir la ejecución de un bucle de repetición en algún punto interno del bloque de instrucciones que se repiten. Lógicamente, ello dependerá de que se verifique o no alguna condición. Esta interrupción puede hacerse de dos formas:

- Abandonando el bucle de repetición definitivamente.
- Abandonando la iteración en curso, pero comenzando la siguiente.

Las órdenes respectivas en MATLAB son (tanto en un bucle `while` como en un bucle `for`):

```
break
continue
```

El funcionamiento de estas órdenes queda reflejado en los diagramas de flujo correspondientes (Figuras 2.3 y 2.4).

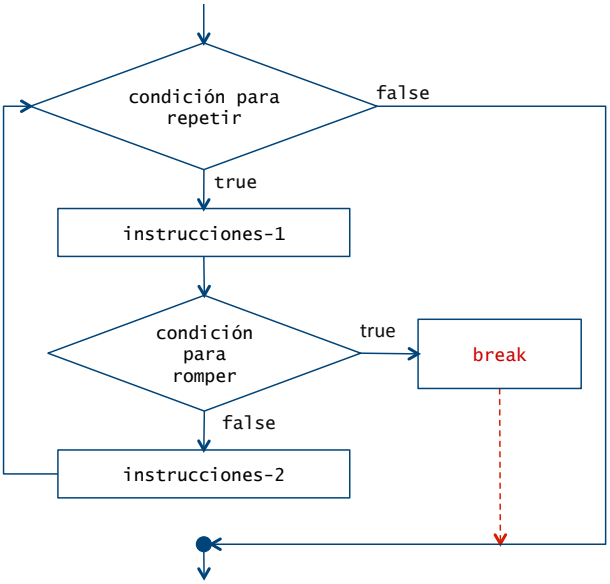


Figura 2.3: Ruptura de bucle `break`: Si en algún momento de un bucle de repetición (de cualquier tipo) se llega a una instrucción `break`, el ciclo de repetición se interrumpe inmediatamente y el programa continúa ejecutándose por la instrucción siguiente a la cláusula `end` del bucle. Si se trata de un bucle indexado `for`, la variable-índice del mismo conserva el último valor que haya tenido.

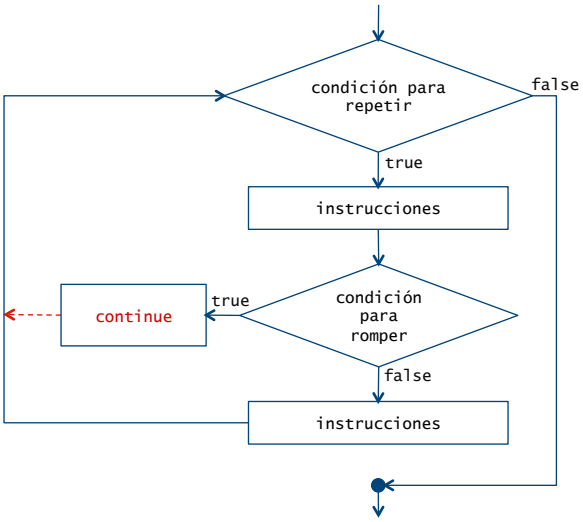


Figura 2.4: Ruptura de bucle `continue`: Si en algún momento de un bucle de repetición (de cualquier tipo) se llega a una instrucción `continue`, la iteración en curso se interrumpe inmediatamente, y se inicia la siguiente iteración (si procede).

2.5 Gestión de errores: warning y error

La instrucción siguiente se utiliza para alertar al usuario de alguna circunstancia no esperada durante la ejecución de un programa, pero no lo detiene. La orden

```
warning('Mensaje')
```

imprime `Warning: Mensaje` en la pantalla y continúa con la ejecución del programa.
Por el contrario, la instrucción

```
error('Mensaje')
```

imprime `??? Mensaje` en la pantalla y detiene en ese punto la ejecución del programa.

Además de estas funciones y sus versiones más completas, MATLAB dispone de otras órdenes para gestionar la detección de errores. Véase Error Handling en la documentación.

2.6 Operaciones de lectura y escritura

2.6.1 Instrucción básica de lectura: input

La instrucción `input` permite almacenar en una variable un dato que se introduce a través del teclado. La orden

```
var = input('Mensaje')
```

imprime `Mensaje` en la pantalla y se queda esperando hasta que el usuario teclea algo en el teclado, terminado por la tecla `return`. Lo que se teclea puede ser cualquier expresión que use constantes y/o variables existentes en el Workspace. El resultado de esta expresión se almacenará en la variable `var`. Si se pulsa la tecla `return` sin teclear nada se obtendrá una matriz vacía: `[ ]`.

2.6.2 Instrucción básica de impresión en pantalla: disp

La instrucción `disp` permite imprimir en la pantalla el valor de una (matriz) constante o variable, sin imprimir el nombre de la variable o `ans` y sin dejar líneas en blanco. Su utilidad es muy limitada.

```
disp(algo)
```

Ejemplos 2.8 (Uso de disp)
Observaciones:
(a) La orden `date` devuelve una cadena de caracteres con la fecha actual.
(b) La orden `num2str(num)` devuelve el dato numérico `num` como una cadena de caracteres.

```
>> disp(pi)
    3.1416

>> v = [1, 2, 3, pi];
>> disp(v)
    1.0000    2.0000    3.0000    3.1416

>> disp('El metodo no converge')
El metodo no converge

>> disp(['Hoy es ', date])
Hoy es 18-Feb-2015

>> x = pi;
>> disp(['El valor de x es: ',num2str(x)])
El valor de x es: 3.1416
```

2.6.3 Instrucción de impresión en pantalla con formato: fprintf

Esta orden permite controlar la forma en que se imprimen los datos. Su sintaxis para imprimir en la pantalla es

```
fprintf( formato, lista_de_datos )
```

donde:

`lista_de_datos` son los datos a imprimir. Pueden ser constantes y/o variables, separados por comas.

`formato` es una cadena de caracteres que describe la forma en que se deben imprimir los datos. Puede contener combibaciones de los siguientes elementos:

- Códigos de conversión: formados por el símbolo `%`, una letra (como `f`, `e`, `i`, `s`) y eventualmente unos números para indicar el número de espacios que ocupará el dato a imprimir.
- Texto literal a imprimir
- Caracteres de escape, como `\n`.

Normalmente el `formato` es una combinación de texto literal y códigos para escribir datos numéricos, que se van aplicando a los datos de la lista en el orden en que aparecen.

En los ejemplos siguientes se presentan algunos casos simples de utilización de esta instrucción. Para una comprensión más amplia se debe consultar la ayuda y documentación de MATLAB.

Ejemplos 2.9 (Uso de fprintf)

```
>> long = 32.067
>> fprintf('La longitud es de %12.6f metros ',long)
La longitud es de    32.067000 metros
```

En este ejemplo, el formato se compone de:

- el texto literal `'La longitud es de '` (incluye los espacios en blanco),
- el código `%12.6f` que indica que se escriba un número ocupando un total de 12 espacios, de los cuales 6 son para las cifras decimales,
- el texto literal `' metros '` (también incluyendo los blancos),
- el carácter de escape `\n` que provoca un salto de línea.

```
>> x = sin(pi/5);
>> y = cos(pi/5);
>> fprintf('Las coordenadas del punto son x= %10.6f e y=%7.3f \n', x, y)
Las coordenadas del punto son x=   0.587785 e y=  0.809
```

Observamos que el primer código `%10.6f` se aplica al primer dato a imprimir, `x`, y el segundo código `%7.3f` al segundo, `y`.

```
>> fprintf('En la iteracion k=%3i el error es %15.7e \n', k, err)
En la iteracion k= 23 el error es   3.1406123e-06
```

- el código `%3i` indica que se escriba un número entero ocupando un total de 3 espacios,
- el código `%15.7e` indica que se escriba un número en formato exponencial ocupando un total de 15 espacios, de los cuales 7 son para los dígitos a la derecha del punto decimal.

2.7 Comentarios generales

Algunos comentarios generales sobre la escritura de programas:

- A los argumentos de salida de una M-función **siempre hay que darles algún valor**.
- Las instrucciones `for`, `if`, `while`, `end`, `return`, `...` no necesitan llevar punto y coma al final, ya que no producen salida por pantalla. Sí necesitan llevarlo, en general, las instrucciones incluídas dentro.
- Las M-funciones no necesitan llevar `end` al final. De hecho, a nivel de este curso, que lo lleven puede inducir a error. Mejor no ponerlo.
- Todos los programas deben incluir unas líneas de comentarios que expliquen de forma sucinta su cometido y forma de utilización. En las M-funciones, estas líneas deben ser las siguientes a la instrucción `function`, ya que de esta manera son utilizadas por MATLAB como el texto de `help` de la función. En los *scripts* deben ser las primeras líneas del archivo.
- Es bueno, en la medida de lo posible y razonable, respetar la notación habitual de la formulación de un problema al elegir los nombres de las variables. Ejemplos
  - Llamar `S` a una suma y `P` a un producto, mejor que `G` o `N`.
  - Llamar `i, j, k, l, m, n` a un número entero, mejor que `x` ó `y`.
  - Llamar `T` o `temp` a una temperatura, mejor que `P`.
  - Llamar `e`, `err` ó `error` a un erro, mejor que `vaca`.
- Hay que comprobar los programas que se hacen, dándoles valores a los argumentos para verificar que funciona bien en **todos los casos que puedan darse**. Esto forma parte del proceso de diseño y escritura del programa.
- Los espacios en blanco, en general, hacen el código más legible y por lo tanto más fácil de comprender y corregir. Se deben dejar uno o dos espacios entre el carácter `%` de cada línea de comentario y el texto del comentario (comparénse los dos códigos siguientes).

```
function[Ma,Mg]=Medias(N)
%Ma=Medias(N)  devuelve la media aritmetica
%de los numeros naturales menores o iguales
%que N
%[Ma,Me]=Medias(N) devuelve tambien la
media
%geometrica
Ma=0;
Mg=1;
for i=1:N
Ma=Ma+i;
Mg=Mg*i;
end
N1=1/N;
Ma=Ma*N1;
Mg=Mg^(N1);
```

```
function [Ma,Mg]=Medias(N)
%-------------------------------------
%  Ma=Medias(N)  devuelve la media
%     aritmetica de los numeros
%     naturales menores o iguales que N
%  [Ma,Me]=Medias(N) devuelve tambien la
%     media geometrica
%-------------------------------------
%

Ma = 0;
Mg = 1;

for i=1:N
  Ma = Ma+i;
  Mg = Mg*i;
end

N1  = 1/N;
Ma = Ma*N1;
Mg = Mg^(N1);
```