

Manual de iniciación a GNU Octave

Autor: José María Valiente Cifuentes

Trabajo realizado dentro de un Proyecto Fin de Carrera dirigido por
Carlos Medrano Sánchez
en la E.U. Politécnica de Teruel

Año 2006

Licencia

Copyright (c) 2006 José María Valiente Cifuentes

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License", at the end of the document.



Manual de Iniciación a GNU Octave

Manual De GNU Octave



1. Operaciones Básicas

1.1. Operaciones con matrices y vectores

1.1.1. Introducción de matrices desde el teclado

1.1.2. Operaciones con matrices

1.1.3. Tipos de datos

1.1.3.1. Números reales de doble precisión

1.1.3.2. Números Complejos

1.1.3.3. Cadenas de caracteres

1.1.3.4. Otras formas de definir matrices

1.1.3.4.1 Tipos de matrices predefinidos

1.1.3.4.2 Formación de una matriz a partir de otras

1.1.3.4.3 Direccionamiento de vectores y matrices a partir de vectores

1.1.3.4.4 Operador <<Dos Puntos>> (:)

1.1.3.4.5 Definición de matrices y vectores desde fichero

1.1.3.5. Operadores Relacionales

1.1.3.6. Operadores Lógicos

1.2. Funciones de Librería

1.2.1 Características Generales de las funciones de librería

1.2.2. Funciones matemáticas elementales que operan de modo escalar

1.2.3. Funciones que actúan sobre matrices

1.2.3.1 Funciones elementales

1.2.3.2 Funciones Especiales

1.2.3.3 Funciones de Factorización y/o Descomposición Matricial

1.3. Más sobre operadores relacionales con vectores y matrices

1.4. Otras funciones que actúan sobre vectores y matrices

2. Otros tipos de datos de GNU Octave

2.1. Cadenas de caracteres

2.2. Hipermatrices (arrays de más de dos dimensiones)

2.2.1 Definición de Hipermatrices



2.3 Estructuras

2.3.1 Creación de Estructuras

2.3.2 Funciones para operar con Estructuras

2.4 Vectores o matrices de celdas (*Cell Array*)

2.4.1 Creación de vectores y matrices de Celdas

2.4.2 Funciones para trabajar con vectores y matrices de celda

3. Programación en GNU Octave

3.1. Bifurcaciones y bucles

3.1.1. Sentencia IF

3.1.2. Sentencia SWITCH

3.1.3. Sentencia FOR

3.1.4. Sentencia DO-UNTIL

3.1.5. Sentencia WHILE

3.1.6. Sentencia BREAK y CONTINUE

3.2 Ficheros *.m

3.2.1 Ficheros de Comandos (*SCRIPTS*)

3.2.2 Definición de Funciones

3.2.3. HELP para las funciones de usuario

4. Gráficos bidimensionales

4.1 Funciones gráficas 2D elementales

4.1.1 Función PLOT

4.1.2 Estilos de Línea y Marcadores para PLOT

4.1.3 Añadir Líneas a un gráfico ya existente

4.1.4 Comando SUBPLOT

4.1.5 Control de los Ejes

4.2 Control de ventanas gráficas: Función Figure

4.3 Otras funciones gráficas 2-D

5. Gráficos tridimensionales

5.1 Tipos de funciones gráficas tridimensionales

5.1.1 Dibujo de líneas: Función PLOT3

5.1.2 Dibujo de mallados: Funciones MESHGRID, MESH Y SURF

5.1.3 Dibujo de líneas de contorno: Función CONTOUR

5.2. Elementos Generales: Ejes, Puntos de vista, líneas ocultas,...



6. Otros aspectos de GNU Octave

6.1 Guardar variables y estados de una sesión: Comandos *save* y *load*

6.2 Guardar sesión: Comando *diary*

6.3 Medida de tiempos y de esfuerzo de cálculo

6.4. Funciones de función

6.4.1 Integración numérica de funciones

6.4.2 Integración Numérica de Ecuaciones Diferenciales Ordinarias



1. Operaciones Básicas

1.1. Operaciones con matrices y vectores

Como se comentó en la introducción que hemos visto en el punto anterior, GNU Octave es un programa creado para trabajar con matrices, por lo tanto, este punto es probablemente el más importante y en el que mejor tenemos que aclararnos para empezar a trabajar. Tenemos muchas opciones para trabajar con ellas, podemos intercambiar matrices, permutarlas, invertirlas; GNU Octave es una herramienta de cálculo muy potente en lo que a matrices se refiere.

1.1.1. Introducción de matrices desde el teclado

Las matrices y vectores son variables del programa cuyos nombres podemos definir, siempre y cuando no utilicemos los caracteres que el programa tiene como caracteres prohibidos.

Para definir una matriz en GNU Octave se determina el número de filas y de columnas en función del número de elementos que se proporcionan (o se utilizan). **Las matrices se definen por filas**; los elementos de una misma fila están separados por **blancos** o **comas**, mientras que las filas están separadas por pulsaciones **intro** o por caracteres **punto y coma** (;). Tomemos como ejemplo:

```
octave:31> a=[1 1; 2 5]
```

Cuya salida será:

```
a =
```

```
1 1  
2 5
```

A partir de este momento la matriz **a** está disponible para hacer cualquier tipo de operación con ella (además de valores numéricos, en la definición de una matriz o vector se pueden utilizar expresiones y funciones matemáticas). Por ejemplo, una



Manual de Iniciación a GNU Octave

sencilla operación con **a** es hallar su **matriz traspuesta**. En GNU Octave, el apóstrofo (**'**) es el símbolo de *trasposición matricial*. Para calcular **a'** (traspuesta de **a**) basta teclear lo siguiente (se añade a continuación la respuesta del programa):

```
octave:32> a'
```

```
ans =
```

```
1 2  
1 5
```

Como el resultado de la operación no ha sido asignado a ninguna otra matriz, GNU Octave utiliza un nombre de variable por defecto (**ans**, de *answer*), que contiene el resultado de la última operación. La variable **ans** puede ser utilizada como operando en la siguiente expresión que se introduzca. También podría haberse asignado el resultado a otra matriz llamada **b**.

Ahora vamos a definir una matriz **b** diferente para hacer operaciones básicas con estas 2 matrices:

```
octave:32> b=[7 6; 8 3]
```

```
b =
```

```
7 6  
8 3
```

Comenzamos con las operaciones más básicas que podemos encontrar, la suma y la resta de matrices:

```
octave:33> a + b
```

```
ans =
```

```
8 7  
10 8
```

```
octave:34> a -b
```

```
ans =
```

```
-6 -5  
-6 2
```

Si realizamos la multiplicación de matrices con el operando ***** tendremos que tener cuidado con que el número de columnas de la primera matriz debe coincidir con el número de filas de la segunda:



Manual de Iniciación a GNU Octave

```
octave:35> a * b  
ans =
```

```
15    9  
54   27
```

También podemos utilizar una multiplicación elemento a elemento, que aunque no tiene demasiado sentido como multiplicación de matrices, si que es muy utilizable en el caso de que la matriz no sea más que un conjunto ordenado de valores.

```
octave:36> a .* b  
ans =
```

```
7    6  
16   15
```

A continuación vamos a definir una nueva matriz **a** a partir de una función que genera valores aleatorios entre 0 y 1.

```
octave:4> a=rand(3); #generar una matriz aleatoria de 3x3  
octave:5> a  
a =  
  
0.6086115  0.0010003  0.3250563  
0.4910289  0.2250230  0.4538064  
0.6238124  0.5558372  0.9002053
```

Vamos ahora a crear una matriz 3x3 para realizar nuevos cálculos a partir de una matriz más manejable si queremos comprobar a mano los datos que creamos.

```
octave:6> a=[1 2 1;1 2 3;4 3.4 4.5]  
a =  
  
1.0000  2.0000  1.0000  
1.0000  2.0000  3.0000  
4.0000  3.4000  4.5000
```

A partir de esta matriz **a** calculamos su inversa con el comando **inv(a)**:

```
octave:7> b=inv(a)  
b =
```



Manual de Iniciación a GNU Octave

```
-0.13043 -0.60870 0.43478  
0.81522 0.05435 -0.21739  
-0.50000 0.50000 0.00000
```

Podemos comprobar multiplicando una por la otra que el cálculo es correcto:

```
octave:9> c=a*b  
c =  
  
1.00000 0.00000 0.00000  
0.00000 1.00000 0.00000  
0.00000 0.00000 1.00000
```

Si los valores no son exactos podemos utilizar el comando **round()** ya que debido a los errores de aproximación en los cálculos podemos encontrar valores como $2.3e-10^9$ que representa un valor extremadamente pequeño.

Si queremos comentar las líneas de código que ejecutamos, a continuación de la operación podemos poner un comentario anteponiendo el carácter **# o %**

```
octave:10> c=a*b #podemos ver que la matriz c es la matriz identidad
```

De igual manera que se define una matriz podemos definir un vector:

```
octave:49> b=[2 0 0]  
b =  
  
2 0 0  
octave:50> b=[2 0 0]',  
b =  
  
2  
0  
0
```

Como podemos observar, podemos definir vectores fila y vectores columna, con sólo hacer la traspuesta del vector en la definición. También podemos definir un vector columna como si hicieramos una matriz de $1 \times n$

```
octave:46> b=[1;0];
```

Como podemos ver, no hemos obtenido resultado tras realizar la operación; esto es debido a que hemos puesto un ";" al final de la línea de comando, esto hace que no salga por pantalla lo que hemos ejecutado, cosa que resulta muy útil cuando las



Manual de Iniciación a GNU Octave

matrices/vectores son de un número muy grande (100, 1000, ...) y por lo tanto, difíciles de manejar visualmente.

En GNU Octave se accede a los elementos de un vector poniendo el índice entre paréntesis (por ejemplo $x(3)$ ó $x(i)$). Los elementos de las matrices se acceden poniendo los dos índices entre paréntesis, separados por una coma (por ejemplo $A(1,2)$ ó $A(i,j)$). Las matrices **se almacenan por columnas** (aunque **se introduzcan por filas**, como se ha dicho antes), y teniendo en cuenta esto puede accederse a cualquier elemento de una matriz con un sólo subíndice. Por ejemplo, si \mathbf{A} es una matriz (3x3) se obtiene el mismo valor escribiendo $A(1,2)$ que escribiendo $A(4)$.



1.1.2. Operaciones con matrices

GNU Octave puede operar con matrices por medio de *operadores* y por medio de *funciones*. Se han visto ya los operadores *suma* (+), *producto* (*) y *traspuesta* ('), así como la función *invertir inv()*. Los operadores matriciales de GNU OCTAVE son los siguientes:

- + adición o suma
- sustracción o resta
- * multiplicación
- ' traspuesta
- [^] potenciación
- \ división-izquierda
- / división-derecha
- .* producto elemento a elemento
- ./ y .\ división elemento a elemento
- .[^] elevar a una potencia elemento a elemento

Estos operadores se aplican también a las variables o valores escalares, aunque con algunas diferencias. Todos estos operadores son coherentes con las correspondientes operaciones matriciales: no se puede por ejemplo sumar matrices que no sean del mismo tamaño. Si los operadores no se usan de modo correcto se obtiene un mensaje de error.

Veamos un ejemplo del uso del divisor:

```
octave:45> #vamos a resolver un sistema de ecuaciones
octave:45> #x+y=1
octave:45> #2x+5y=0
octave:45> a=[1 1;2 5];
octave:46> b=[1;0];
octave:47> x=inv(a)*b
x =
  1.66667
 -0.66667
```



Manual de Iniciación a GNU Octave

```
octave:48> #por tanto x=1.6667 e y=-0.66667
octave:48> a\b
ans =
```

```
1.66667
-0.66667
```

Véase el siguiente ejemplo de tres ecuaciones formadas por una recta que no pasa por el origen y los dos ejes de coordenadas:

```
octave:49> A=[1 2; 1 0; 0 1], b=[2 0 0] '
A =
```

```
1 2
1 0
0 1
```

```
b =
```

```
2
0
0
```

```
octave:50> x=A\b, resto=A*x-b
```

```
x =
```

```
0.33333
0.66667
```

```
resto =
```

```
-0.33333
0.33333
0.66667
```



Manual de Iniciación a GNU Octave

Vamos a ver como funcionan una serie de operadores:

```
octave:37> a/b
```

```
0.185185 -0.037037  
1.259259 -0.851852
```

```
octave:38> a\b
```

```
ans =
```

```
9.0000 9.0000  
-2.0000 -3.0000
```

Si los operadores « / » y « \ » van precedidos de un “.” La operación se realiza elemento a elemento:

```
octave:39> a./b
```

```
ans =
```

```
0.14286 0.16667  
0.25000 1.66667
```

```
octave:40> a.\b
```

```
ans =
```

```
7.00000 6.00000  
4.00000 0.60000
```

Podemos comprobar que premultiplicar con « / » es lo mismo que postmultiplicar con « \ ».

```
octave:41> b./a
```

```
ans =
```

```
7.00000 6.00000  
4.00000 0.60000
```



1.1.3. Tipos de datos

GNU Octave trabaja siempre con el tipo Real de doble precisión, sobre el que se implementan el resto de tipos, avanzados o no. Este tipo de dato se guarda con un tamaño de 8 bytes, que tiene un tamaño de 15 cifras exactas. Además del tipo Real, podremos trabajar con strings, matrices, hipermatrices y estructuras más avanzadas.

1.1.3.1. Números reales de doble precisión

Los elementos constitutivos de vectores y matrices son números reales almacenados en 8 bytes (53 bits para la mantisa y 11 para el exponente de 2; entre 15 y 16 cifras decimales equivalentes). Es importante saber cómo trabaja GNU Octave con estos números y los casos especiales que presentan. GNU Octave mantiene una forma especial para los *números muy grandes* (más grandes que los que es capaz de representar), que son considerados como *infinito*. Por ejemplo, obsérvese cómo responde el programa al ejecutar el siguiente comando:

```
octave:52> 1/0
warning: division by zero
ans = Inf
```

Así pues, para GNU Octave el *infinito* se representa como *inf* ó *Inf*. GNU Octave tiene también una representación especial para los resultados que no están definidos como números. Por ejemplo, ejecútense los siguientes comandos y obsérvense las respuestas obtenidas:

```
octave:51> 0/0
warning: division by zero
ans = NaN
```

La respuesta es *NaN*, que es la abreviatura de *Not a Number*. Este tipo de respuesta, así como la de *Inf*, son enormemente importantes en GNU Octave, pues permiten controlar la fiabilidad de los resultados de los cálculos matriciales. Los *NaN* se propagan al realizar con ellos cualquier operación aritmética, por ejemplo, cualquier



número sumado a un *NaN* da otro *NaN*. GNU Octave tiene esto en cuenta. Algo parecido sucede con los *Inf*.

Podemos encontrar 3 variables predefinidas por GNU Octave que nos dan los valores máximos y mínimos de este tipo de datos:

- **eps** devuelve la diferencia entre 1.0 y el número de coma flotante inmediatamente superior. Da una idea de la precisión o número de cifras almacenadas. En un computador, **eps** vale 2.2204e-016.
- **realmin** devuelve el número más pequeño con que se puede trabajar (2.2251e-308)
- **realmax** devuelve el número más grande con que se puede trabajar (1.7977e+308)

1.1.3.2. Números complejos (complex)

Muchas veces nos vamos a encontrar que el cálculo que necesitamos ejecutar nos lleva a tener que definir el cuerpo de los números complejos, dado que el cuerpo de los números reales no es suficiente, por ejemplo, para realizar transformadas de Fourier o Laplace. Para ello se define la variable compleja **i** o **j**:

```
octave:61> i  
i = 0 + 1i  
octave:62> j  
j = 0 + 1i  
octave:63> I  
I = 0 + 1i  
octave:64> 2J  
J = 0 + 2i
```

Como podemos ver, se pueden definir tanto en mayúscula como en minúscula, y que no es necesario el uso del operador “*” para multiplicarlo por un escalar. Debemos tener la precaución de no redefinir la variable **i** o **j**, ya que nos podría llevar a equívocos.

```
octave:63> i=2  
i = 2
```

Podemos definir a su vez un número complejo con la función **complex**:

```
octave:77> complex(7, 6)  
ans = 7 + 6i
```

Es importante advertir que el *operador de matriz transpuesta* ('), aplicado a matrices complejas, produce la matriz transpuesta conjugada. Existe una función que



permite hallar simplemente la matriz conjugada (**conj()**) y el operador punto y apóstrofo (.)' que calcula simplemente la matriz transpuesta.

```
octave:78> conj(a)  
ans = 1 - 1i
```

Esta función **conj()** puede ser empleada con escalares debido a que podemos entender dicho escalar como una matriz de 1x1.

1.1.3.3. Cadenas de caracteres

Para crear una cadena de caracteres (string) en GNU Octave podemos hacerlo de estos dos modos:

```
octave:79> s='cadena de caracteres'  
s = cadena de caracteres  
octave:80> s="cadena de caracteres"  
s = cadena de caracteres
```

Debido a que para su uso es necesario un conocimiento previo de las funciones orientadas a matrices, postpondré otras explicaciones hasta que se hayan explicado éstas, al igual que con los **strings**, se postpone la explicación de **hipermatrices**, **structs** y **cell arrays**.

Ya han aparecido algunos ejemplos de **variables** y **expresiones** matriciales. Ahora se va a tratar de generalizar un poco lo visto hasta ahora. Una **variable** es un nombre que se da a una entidad numérica, que puede ser una matriz, un vector o un escalar. El valor de esa variable, e incluso el tipo de entidad numérica que representa, puede cambiar a lo largo de una sesión de GNU OCTAVE o a lo largo de la ejecución de un programa. La forma más normal de cambiar el valor de una variable es colocándola a la izquierda del **operador de asignación** (=).

Una expresión de GNU OCTAVE puede tener las dos formas siguientes: primero, asignando su resultado a una variable,

```
variable = expresión
```

y segundo evaluando simplemente el resultado del siguiente modo, expresión en cuyo caso el resultado se asigna automáticamente a una variable interna de GNU OCTAVE llamada **ans** (de *answer*) que almacena el último resultado obtenido. Se considera por defecto que una expresión termina cuando se pulsa **intro**. Si se desea que una expresión continúe en la línea siguiente, hay que introducir **tres puntos** (...) antes de



pulsar ***intro***. También se pueden incluir varias expresiones en una misma línea separándolas por *comas* (,) o *puntos y comas* (;). Si una expresión **termina en punto y coma** (;) su resultado se calcula, pero no se escribe en pantalla. Esta posibilidad es muy interesante, tanto para evitar la escritura de resultados intermedios, como para evitar la impresión de grandes cantidades de números cuando se trabaja con matrices de gran tamaño.

A semejanza de C, ***GNU OCTAVE distingue entre mayúsculas y minúsculas*** en los nombres de variables. A diferencia del lenguaje C, no hace falta declarar las variables que se vayan a utilizar. Esto hace que se deba tener especial cuidado con no utilizar nombres erróneos en las variables, porque no se recibirá ningún aviso del ordenador. Cuando se quiere tener una *relación de las variables* que se han utilizado en una sesión de trabajo se puede utilizar el comando ***who***. Existe otro comando llamado ***whos*** que proporciona además información sobre el tamaño, la cantidad de memoria ocupada y el carácter real o complejo de cada variable.

El comando ***clear*** tiene varias formas posibles:

- ***clear*** sin argumentos, ***clear*** elimina todas las variables creadas previamente (excepto las variables globales).
- ***clear A, b*** borra las variables indicadas.
- ***clear global*** borra las variables globales.

1.1.3.4. Otras formas de definir matrices

GNU OCTAVE dispone de varias formas de definir matrices. El introducirlas por teclado sólo es práctico en casos de pequeño tamaño y cuando no hay que repetir esa operación muchas veces. Recuérdese que en GNU OCTAVE no hace falta definir el tamaño de una matriz. Las matrices toman tamaño al ser definidas y este tamaño puede ser modificado por el usuario mediante adición y/o borrado de filas y columnas. A continuación se van a ver otras formas más potentes y generales de definir y/o modificar matrices.



1.1.3.4.1 Tipos de matrices predefinidos

Existen en GNU OCTAVE varias funciones orientadas a definir con gran facilidad matrices de tipos particulares. Algunas de estas funciones son las siguientes:

- eye(2) forma la matriz unidad de tamaño (2x2)

```
octave:84> eye(2)
ans =
```

```
1 0
0 1
```

- zeros(3,5) forma una matriz de *ceros* de tamaño (3x5)

```
octave:83> zeros(3,5)
ans =
```

```
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

- zeros(2) ídem de tamaño (2x2)

```
octave:85> zeros(2)
ans =
```

```
0 0
0 0
```

- ones(3) forma una matriz de *unos* de tamaño (3x3)
- ones(3,4) ídem de tamaño (3x4)

```
octave:87> ones(3,4)
ans =
```

```
1 1 1 1
1 1 1 1
1 1 1 1
```

- linspace(x1,x2,n) genera un vector con **n** valores igualmente espaciados entre **x1** y **x2**

```
octave:88> linspace(1,10,9)
ans =
```



Manual de Iniciación a GNU Octave

Columns 1 through 8:

```
1.0000    2.1250    3.2500    4.3750    5.5000    6.6250    7.7500  
8.8750
```

Column 9:

```
10.0000
```

```
octave:89> #semejante a la definicion 1:1.125:10
```

- `logspace(d1,d2,n)` genera un vector con **n** valores espaciados logarítmicamente entre 10^{d1} y 10^{d2} .
- `rand(3)` forma una matriz de números aleatorios entre 0 y 1, con distribución uniforme, de tamaño (3x3)
- `rand(2,5)` idem de tamaño (2x5)
- `randn(4)` forma una matriz de números aleatorios de tamaño (4x4), con distribución normal, de valor medio 0 y varianza 1.
- `magic(3)` crea una matriz (3x3) con los números 1, 2, ... 3*3, con la propiedad de que todas las filas y columnas suman lo mismo.

```
octave:92> magic(3)
```

```
ans =
```

```
8   1   6  
3   5   7  
4   9   2
```

- `compan(pol)` construye una matriz cuyo polinomio característico tiene como coeficientes los elementos del vector **pol** (ordenados de mayor grado a menor).

```
octave:100> compan([1 2 1])
```

```
ans =
```

```
-2   -1  
1     0
```

Hay más tipos predefinidos de matrices, podemos encontrar en la ayuda todos estos tipos y otros muchos más con solo hacer **help -i matrix**.



1.1.3.4.2 Formación de una matriz a partir de otras

GNU Octave ofrece también la posibilidad de crear una matriz a partir de matrices previas ya definidas, por varios posibles caminos:

- recibiendo alguna de sus propiedades (como por ejemplo el tamaño),
- por composición de varias submatrices más pequeñas,
- modificándola de alguna forma.

A continuación se describen algunas de las funciones que crean una nueva matriz a partir de otra o de otras, comenzando por dos funciones auxiliares:

- $[m,n]=\text{size}(A)$ devuelve el número de filas y de columnas de la matriz **A**.

Si la matriz es cuadrada basta recoger el primer valor de retorno

```
octave:103> a=[1 2 1;3 2 5;6 7 5];
octave:104> [m,n]=size(a)
m = 3
n = 3
```

- $n=\text{length}(x)$ calcula el número de elementos de un vector **x**

```
octave:105> x=1:0.00001:2;
octave:106> longitud_vector=length(x)
longitud_vector = 100001
```

- $\text{zeros}(\text{size}(A))$ forma una matriz de *ceros* del mismo tamaño que una matriz **A** previamente creada.

```
octave:107> zeros(size(a))
ans =
```

```
0 0 0
0 0 0
0 0 0
```

- $\text{ones}(\text{size}(A))$ ídem con *unos*
- $A=\text{diag}(x)$ forma una matriz diagonal **A** cuyos elementos diagonales son los elementos de un vector ya existente **x**.
- $x=\text{diag}(A)$ forma un vector **x** a partir de los elementos de la diagonal de una matriz ya existente **A**.



Manual de Iniciación a GNU Octave

```
octave:111> diag(a)
```

```
ans =
```

```
1
```

```
2
```

```
5
```

- `diag(diag(A))` crea una matriz diagonal a partir de la diagonal de la matriz **A**.

```
octave:112> diag(diag(a))
```

```
ans =
```

```
1 0 0
```

```
0 2 0
```

```
0 0 5
```

- `triu(A)` forma una matriz triangular superior a partir de una matriz **A** (no tiene por qué ser cuadrada).

```
octave:110> triu(a)
```

```
ans =
```

```
1 2 1
```

```
0 2 5
```

```
0 0 5
```

- `tril(A)` ídem con una matriz triangular inferior.

```
octave:109> tril(a)
```

```
ans =
```

```
1 0 0
```

```
3 2 0
```

```
6 7 5
```



Manual de Iniciación a GNU Octave

- `rot90(A,k)` Gira $k \cdot 90$ grados la matriz rectangular **A** en sentido antihorario. **k** es un entero que puede ser negativo. Si se omite, se supone **k=1**

```
octave:113> rot90(a,1)
ans =
```

```
1 5 5
2 2 7
1 3 6
```

- `flipud(A)` halla la matriz simétrica de **A** respecto de un eje horizontal

```
octave:114> flipud(a)
ans =
```

```
6 7 5
3 2 5
1 2 1
```

Un caso especialmente interesante es el de crear una nueva matriz **componiendo como submatrices** otras matrices definidas previamente. A modo de ejemplo, vamos a realizar la matriz generadora de un código ortogonal con M=2:

```
octave:124> a=[0 0;0 1];
octave:125> h2=[a a;a not(a)]
h2 =
```

```
0 0 0 0
0 1 0 1
0 0 1 1
0 1 1 0
```



1.1.3.4.3 Direccionamiento de vectores y matrices a partir de vectores

Los elementos de una matriz **a** pueden direccionarse a partir de los elementos de vectores:

```
octave:127> a=rand(2,5)
a =
0.253124  0.517686  0.089229  0.382511  0.177737
0.081242  0.079172  0.841718  0.649800  0.634496
```

```
octave:128> b=[1 2 5];
octave:129> a(b)
ans =
0.253124  0.081242  0.089229
```

Podemos ver que hemos obtenido las posiciones 1, 2 y 5 de la matriz **a**, que debemos contar teniendo en cuenta que la matriz se recorre por columnas y no por filas.

Si queremos ver un elemento concreto de la matriz, podemos ejecutar lo siguiente:

```
octave:132> a(2,1)
ans = 0.081242
```

Creamos esta nueva matriz **a** para que sea más fácil seguir los valores:

```
octave:133> a=randn(4)*10
a =
-22.94025   6.55306   -2.80857   -1.33060
 5.12081    8.98726    1.81236    3.29938
-0.69713    1.40557   25.56103    5.62650
-17.50361   -0.63861   4.61985    1.21845
```

```
octave:134> a(2,3)
ans = 1.8124
```



Ahora podemos ver las columnas 1, 2 y 3 de la 4^a fila

```
octave:135> a(4,1:3)
```

```
ans =
```

```
-17.50361 -0.63861 4.61985
```

```
octave:136> #tercera fila
```

```
octave:136> a(3,:)
```

```
ans =
```

```
-0.69713 1.40557 25.56103 5.62650
```

```
octave:137> #tercera columna
```

```
octave:137> a(:,3)
```

```
ans =
```

```
-2.8086
```

```
1.8124
```

```
25.5610
```

```
4.6199
```

Para ver el último elemento de una matriz podemos usar el direccionamiento

end:

```
octave:139> a(end,end)
```

```
ans = 1.2185
```

1.1.3.4.4 Operador <<Dos Puntos>> (:)

Se trata de una de las formas de definir vectores y matrices más usada y más fácil de utilizar, dada la rápida visualización de la salida sin necesidad de ver el resultado:

```
octave:140> x=1:1:10;
```

```
octave:141> x
```

```
x =
```

```
1 2 3 4 5 6 7 8 9 10
```



En cierta forma se podría decir que el operador (:) representa un *rango*: en este caso, los números enteros entre el 1 y el 10. Por defecto el incremento es 1, pero este operador puede también utilizarse con otros valores enteros y reales, positivos o negativos. En este caso el incremento va entre el valor inferior y el superior, pero podemos hacer que el incremento sea negativo, o que se haga con un incremento mayor o menor:

```
octave:142> x=10:-1:1  
x =  
  
10 9 8 7 6 5 4 3 2 1
```

1.1.3.4.5 Definición de matrices y vectores desde fichero

GNU Octave acepta el uso de scripts desde los que crear matrices, vectores, variables, etc; como si estuviéramos ejecutándolo desde la propia línea de comandos. Por ejemplo, si creamos el fichero **matriz_a.m** donde creamos una matriz a cualquiera, al ejecutarla en GNU Octave podremos ver que se crea como si estuviéramos generándola en el propio programa:

```
#primer script  
a=randn(4)*10  
#fin del script  
octave:133> matriz_a  
a =  
  
-22.94025 6.55306 -2.80857 -1.33060  
5.12081 8.98726 1.81236 3.29938  
-0.69713 1.40557 25.56103 5.62650  
-17.50361 -0.63861 4.61985 1.21845
```

Nota: Tenemos que estar dentro del directorio de trabajo o incluir el directorio donde encontrar la función dentro del path



1.1.3.5. Operadores Relacionales

El lenguaje de programación de GNU Octave dispone de los siguientes operadores relacionales:

- < menor que
- > mayor que
- <= menor o igual que
- >= mayor o igual que
- == igual que
- ~= distinto que

Obsérvese que, salvo el último de ellos, coinciden con los correspondientes operadores relacionales de C. Sin embargo, ésta es una coincidencia más bien formal. En GNU Octave los operadores relacionales pueden aplicarse a vectores y matrices, y eso hace que tengan un significado especial. Al igual que en C, si una comparación se cumple el resultado es 1 (*true*), mientras que si no se cumple es 0 (*false*). Recíprocamente, cualquier valor distinto de cero es considerado como *true* y el cero equivale a *false*. La diferencia con C está en que cuando los operadores relacionales de GNU Octave se aplican a dos matrices o vectores del mismo tamaño, *la comparación se realiza elemento a elemento*, y el resultado es otra matriz de unos y ceros del mismo tamaño, que recoge el resultado de cada comparación entre elementos.

```
octave:147> a=7;
octave:148> b=8;
octave:149> a<b
ans = 1
octave:155> a==b
ans = 0
```



1.1.3.6. Operadores Lógicos

Los operadores lógicos de GNU Octave son los siguientes:

- & and
- | or
- ~ negación lógica

Obsérvese que estos operadores lógicos tienen distinta notación que los correspondientes operadores de C (`&&`, `||` y `!`). Los operadores lógicos se combinan con los relacionales para poder comprobar el cumplimiento de condiciones múltiples.

```
octave:165> a=true
a = 1
octave:166> b=not(a)
b = 0
octave:167> c=a&b
c = 0
octave:168> c=a|b
c = 1
```



1.2. Funciones de Librería

GNU Octave posee un gran número de funciones integradas y de funciones definidas por el usuario, las primeras son funciones optimizadas para Octave, las segundas, con extensión *.m son funciones definidas en ficheros, que pueden ser:

- Definidas por GNU Octave
- Definidas por grupos/usuarios desinteresados que ofrecen su código a los demás usuarios de GNU Octave
- Definidas por el propio usuario, para su uso y/o compartición con otros usuarios.

1.2.1 Características Generales de las funciones de librería

El concepto de función en GNU OCTAVE es semejante al de C y al de otros lenguajes de programación, aunque con algunas diferencias importantes. Al igual que en C, una función tiene **nombre, valor de retorno** y **argumentos**. Una función *se llama* utilizando su nombre en una expresión o utilizándolo como un comando más.

```
#funcion de prueba para evaluar
```

```
function y=prueba(x)
```

```
y=x+3;
```

```
endfunction
```

Podemos ver que esta función es solamente la función de una recta, cuya pendiente es de 45º y desplazada 3 unidades.

y : es el valor de retorno.

prueba : es el nombre de la función.

x : es el argumento de entrada.

Una característica de GNU OCTAVE es que las funciones que no tienen argumentos no llevan paréntesis, por lo que a simple vista no siempre son fáciles de distinguir de las simples variables. Ejemplo:

```
#! hello -qf
printf("hello, world \n")
```



```
octave:1> hello
```

```
hello, word
```

Los nombres de las funciones de GNU Octave no son *palabras reservadas* del lenguaje. Es posible crear una variable llamada **sin** o **cos**, que ocultan las funciones correspondientes. Para poder acceder a las funciones hay que eliminar (**clear**) las variables del mismo nombre que las ocultan.

Podemos encontrar gran variedad de tipos de función según lo que resuelvan:

- 1.- Funciones matemáticas elementales.
- 2.- Funciones especiales.
- 3.- Funciones matriciales elementales.
- 4.- Funciones matriciales específicas.
- 5.- Funciones para la descomposición y/o factorización de matrices.
- 6.- Funciones para análisis estadístico de datos.
- 7.- Funciones para análisis de polinomios.
- 8.- Funciones para integración de ecuaciones diferenciales ordinarias.
- 9.- Resolución de ecuaciones no-lineales y optimización.
- 10.- Integración numérica.
- 11.- Funciones para procesamiento de señal.

Las características principales de estas funciones son:

- Los *argumentos actuales* de estas funciones pueden ser expresiones y también llamadas a otra función.
- Admite valores de retorno matriciales múltiples. Por ejemplo, en el comando:

```
octave:177> a=[1 2;2 3];
octave:179> [vector_propio, valor_propio]=eig(a)
vector_propio =
-0.85065    0.52573
 0.52573    0.85065
```



```
valor_propio =  
  
-0.23607  0.00000  
0.00000  4.23607
```

la función *eig()* calcula los valores y vectores propios de la matriz cuadrada **A**. Los vectores propios se devuelven como columnas de la matriz **vector_propio**, mientras que los valores propios son los elementos de la matriz diagonal **valor_propio**.

- Las operaciones de suma y/o resta de una matriz con un escalar consisten en sumar y/o restar el escalar a todos los elementos de la matriz.
- Recuérdese que tecleando *help nombre_funcion* se obtiene de inmediato información sobre la función de ese nombre.

1.2.2. Funciones matemáticas elementales que operan de modo escalar.

Estas funciones, que comprenden las funciones matemáticas trascendentales y otras funciones básicas, actúan sobre cada elemento de la matriz como si se tratase de un escalar. Se aplican de la misma forma a escalares, vectores y matrices. Algunas de las funciones de este grupo son las siguientes:

- $\sin(x)$: seno
- $\cos(x)$: coseno
- $\tan(x)$: tangente
- $\arcsin(x)$: arco seno
- $\arccos(x)$: arco coseno
- $\arctan(x)$: arco tangente (devuelve un ángulo entre -90° y 90°)
- $\sinh(x)$: seno hiperbólico
- $\cosh(x)$: coseno hiperbólico
- $\tanh(x)$: tangente hiperbólica
- $\operatorname{arsinh}(x)$: arco seno hiperbólico
- $\operatorname{arcosh}(x)$: arco coseno hiperbólico
- $\operatorname{atanh}(x)$: arco tangente hiperbólica
- $\log(x)$: logaritmo natural
- $\log_{10}(x)$: logaritmo decimal
- $\exp(x)$: función exponencial



Manual de Iniciación a GNU Octave

- `sqrt(x)` : raíz cuadrada
- `round(x)` : redondeo hacia el entero más próximo
- `fix(x)` : redondea hacia el entero más próximo a 0
- `floor(x)` : valor entero más próximo hacia $-\infty$
- `ceil(x)` : valor entero más próximo hacia $+\infty$
- `gcd(x)` : máximo común divisor
- `lcm(x)` : mínimo común múltiplo
- `real(x)` : partes reales
- `imag(x)` : partes imaginarias
- `abs(x)` : valores absolutos
- `angle(x)` : ángulos de fase

En realidad estas funciones *se pueden aplicar también a matrices*, pero en ese caso se aplican por separado a cada columna de la matriz, dando como valor de retorno un vector resultado de aplicar la función a cada columna de la matriz considerada como vector. Si estas funciones se quieren aplicar a las filas de la matriz basta aplicar dichas funciones a la matriz traspuesta.



1.2.3. Funciones que actúan sobre matrices

Las siguientes funciones exigen que el/los argumento/s sean matrices. En este grupo aparecen algunas de las funciones más útiles y potentes de GNU OCTAVE. Se clasificarán en varios subgrupos.

1.2.3.1 Funciones elementales

- $B = A'$ calcula la traspuesta (conjugada) de la matriz **A**

```
octave:29> a=[1+i,2+3i;1+2i,2+i];
```

```
octave:30> b=a'
```

```
b =
```

```
1 - 1i  1 - 2i  
2 - 3i  2 - 1i
```

- $B = A.'$ calcula la traspuesta (sin conjugar) de la matriz **A**

```
octave:31> b=a.'
```

```
b =
```

```
1 + 1i  1 + 2i  
2 + 3i  2 + 1i
```

- $v = \text{poly}(A)$ devuelve un vector **v** con los coeficientes del polinomio característico de la matriz cuadrada **A**

```
octave:35> x=[1 2 1];
```

```
octave:36> v=poly(x)
```

```
v =
```

```
1 -4  5 -2
```

- $t = \text{trace}(A)$ devuelve la traza **t** (suma de los elementos de la diagonal) de una matriz cuadrada **A**

```
octave:37> t=trace(c)
```

```
t = 2
```



- $[m,n] = \text{size}(A)$ devuelve el número de filas **m** y de columnas **n** de una matriz rectangular **A**

```
octave:38> [m,n]=size(c)
m = 2
n = 2
```

- $n = \text{size}(A)$ devuelve el tamaño de una matriz cuadrada **A**

```
octave:39> n=size(c)
n = 2
```

1.2.3.2 Funciones Especiales

Las funciones *exp()*, *sqrt()* y *log()* se aplican elemento a elemento a las matrices y/o vectores que se les pasan como argumentos. Existen otras funciones similares que tienen también sentido cuando se aplican a una matriz como una única entidad. Estas funciones son las siguientes (se distinguen porque llevan una "m" adicional en el nombre):

- $\text{expm}(A) : \text{si } A = XDX', \text{ expm}(A) = X * \text{diag}(\exp(\text{diag}(D))) * X'$
- $\text{sqrtm}(A) : \text{devuelve una matriz que multiplicada por sí misma da la matriz } A$
- $\text{logm}(A) : \text{es la función inversa de expm}(A)$

```
octave:39> logm(c)
ans =
0.8047 + 1.5708i 0.3540 - 1.2825i
0.5309 - 1.9238i 0.8047 + 1.5708i
```



1.2.3.3 Funciones de Factorización y/o Descomposición Matricial

A su vez este grupo de funciones se puede subdividir en 4 subgrupos:

- Funciones basadas en la factorización triangular (eliminación de Gauss):
 - $[L,U] = lu(A)$ descomposición de Crout ($A = LU$) de una matriz. La matriz **L** es una permutación de una matriz triangular inferior (dicha permutación es consecuencia del pivotamiento por columnas utilizado en la factorización).

```
octave:42> a=rand(3)
a =
0.521692  0.247373  0.174305
0.245762  0.047457  0.744174
0.468085  0.375004  0.362295

octave:43> [L,U]=lu(a)
L =
1.00000  0.00000  0.00000
0.47109  -0.45134  1.00000
0.89725  1.00000  0.00000

U =
0.52169  0.24737  0.17430
0.00000  0.15305  0.20590
0.00000  0.00000  0.75499
```



- $B = \text{inv}(A)$ calcula la inversa de **A**. Equivale a $B = \text{inv}(U) * \text{inv}(L)$

```
octave:63> a=[1 1;2 2];
octave:65> inv(a)
warning: inverse: matrix singular to machine precision, rcond = 0
ans =
2.00000 2.00000
0.50000 0.00000
```

- $d = \det(A)$ devuelve el determinante **d** de la matriz cuadrada **A**. Equivale a $d = \det(L) * \det(U)$
- $E = \text{rref}(A)$ reducción a forma de escalón (mediante la eliminación de Gauss con pivotamiento por columnas) de una matriz rectangular **A**. Sólo se utiliza la diagonal y la parte triangular superior de **A**. El resultado es una matriz triangular superior tal que $A = U^*U$.

```
octave:46> a=[2 1 3 0;1 2 1 1;1 1 1 3];
octave:47> e=rref(a)
e =
1.00000 0.00000 0.00000 13.00000
0.00000 1.00000 0.00000 -2.00000
0.00000 0.00000 1.00000 -8.00000
```

- $c = \text{rcond}(A)$ devuelve una estimación del recíproco de la condición numérica de la matriz **A** basada en la norma sub-1. Si el resultado es próximo a 1 la matriz **A** está bien condicionada; si es próximo a 0 no lo está.
 - Funciones basadas en el cálculo de valores y vectores propios:
- $[X,D] = \text{eig}(A)$ valores propios (diagonal de **D**) y vectores propios (columnas de **X**) de una matriz cuadrada **A**. Con frecuencia el resultado es complejo (si **A** no es simétrica).
- $[X,D] = \text{eig}(A,B)$ valores propios (diagonal de **D**) y vectores propios (columnas de **X**) de dos matrices cuadradas **A** y **B** ($\mathbf{Ax} = \lambda \mathbf{Bx}$).
 - Funciones basadas en la descomposición QR:



Manual de Iniciación a GNU Octave

- $[Q,R] = qr()$ descomposición QR de una matriz rectangular. Se utiliza para sistemas con más ecuaciones que incógnitas.

```
octave:51> a=[1 1 1 1;1 1 2 4; 1 2 1 2; 1 2 2 3];
```

```
octave:52> [q,r]=qr(a);
```

```
octave:53> q
```

```
q =
```

```
-0.50000 -0.50000 0.50000 0.50000  
-0.50000 -0.50000 -0.50000 -0.50000  
-0.50000 0.50000 0.50000 -0.50000  
-0.50000 0.50000 -0.50000 0.50000
```

```
octave:54> r
```

```
r =
```

```
-2.00000 -3.00000 -3.00000 -5.00000  
0.00000 1.00000 0.00000 0.00000  
0.00000 0.00000 -1.00000 -2.00000  
0.00000 0.00000 0.00000 -1.00000
```

- $B = \text{null}(A)$ devuelve una base ortonormal del subespacio nulo (kernel, o conjunto de vectores x tales que $Ax = 0$) de la matriz rectangular A .

```
octave:56> a=[1 1; 2 1];
```

```
octave:57> null(a)
```

```
ans = [] (2x0)
```



Manual de Iniciación a GNU Octave

- $Q = \text{orth}(A)$ las columnas de Q son una base ortonormal del espacio vectorial de las columnas de A , y el número de columnas de Q es el rango de A .

```
octave:62> q=orth(a)
```

```
q =
```

```
0.36060  0.93272  
0.93272 -0.36060
```

– Funciones basadas en la descomposición de valor singular

- $B = \text{pinv}(A)$ calcula la pseudo-inversa de una matriz rectangular A

```
octave:63> a=[1 1;2 2];
```

```
octave:64> pinv(a)
```

```
ans =
```

```
0.100000 0.200000  
0.100000 0.200000
```

- $r = \text{rank}(A)$ calcula el rango r de una matriz rectangular A

- $\text{nor} = \text{norm}(A)$ calcula la norma sub-2 de una matriz (el mayor valor singular)

- $\text{nor} = \text{norm}(A,2)$ lo mismo que la anterior

```
octave:67> x=rand(1,4)
```

```
x =
```

```
0.84525 0.67284 0.87206 0.57248
```

```
octave:68> norm(x,2)
```

```
ans = 1.5018
```



– Cálculo del rango, normas y condición numérica:

Existen varias formas de realizar estos cálculos, con distintos niveles de esfuerzo de cálculo y de precisión en el resultado. El rango se calcula implícitamente (sin que el usuario lo pida) al ejecutar las funciones **rref(A)**, **orth(A)**, **null(A)** y **pinv(A)**.

Normas de vectores:

- `norm(x,p)` norma sub- p , es decir $\text{sum}(\text{abs}(x)^p)^{(1/p)}$.
- `norm(x)` norma euclídea; equivale al módulo o `norm(x,2)`.
- `norm(x,inf)` norma sub- ∞ , es decir `max(abs(x))`.
- `norm(x,1)` norma sub-1, es decir `sum(abs(x))`.

```
octave:67> x=rand(1,4)
x =
0.84525  0.67284  0.87206  0.57248
```

```
octave:68> norm(x,2)
ans = 1.5018
octave:69> norm(x,3)
ans = 1.2072
```



1.3. Más sobre operadores relacionales con vectores y matrices

Cuando alguno de los operadores relacionales vistos previamente ($<$, $>$, \leq , \geq , $=$ y \sim) actúa entre dos matrices (vectores) del mismo tamaño, el resultado es otra matriz (vector) de ese mismo tamaño conteniendo unos y ceros, según los resultados de cada comparación entre elementos hayan sido **true** o **false**, respectivamente.

De ordinario, las matrices "binarias" que se obtienen de la aplicación de los operadores relacionales no se almacenan en memoria ni se asignan a variables, sino que se procesan sobre la marcha. GNU OCTAVE dispone de varias funciones para ello. Recuérdese que cualquier valor distinto de cero equivale a **true**, mientras que un valor cero equivale a **false**. Algunas de estas funciones son:

- `any(x)` función vectorial; chequea si *alguno* de los elementos del vector **x** cumple una determinada condición (en este caso ser distinto de cero). Devuelve un *uno* ó un *cero*

```
octave:70> any(x)
ans = 1
```

- `any(A)` se aplica por separado a cada columna de la matriz **A**. El resultado es un vector de *unos* y *ceros* `all(x)` función vectorial; chequea si *todos* los elementos del vector **x** cumplen una condición. Devuelve un uno ó un cero.

```
octave:71> any(a)
ans =
```

1 1



- `all(A)` se aplica por separado a cada columna de la matriz **A**. El resultado es un vector de *unos y ceros*.

```
octave:74> all(a)
```

```
ans =
```

```
1 1
```

- `find(x)` busca índices correspondientes a elementos de vectores que cumplen una determinada condición. El resultado es un vector con los índices de los elementos que cumplen la condición.

```
octave:82> x
```

```
x =
```

```
0.84525 0.67284 0.87206 0.57248
```

```
octave:83> find(x>0)
```

```
ans =
```

```
1 2 3 4
```

- `find(A)` cuando esta función se aplica a una matriz la considera como un vector con una columna detrás de otra, de la 1^a a la última.



1.4. Otras funciones que actúan sobre vectores y matrices

Las siguientes funciones pueden actuar sobre vectores y matrices, y sirven para chequear ciertas condiciones:

- `exist(var)` comprueba si la variable **var** existe
- `isnan()` chequea si hay valores *NaN*, devolviendo una matriz de unos y ceros

```
octave:89> x=[1 2 3 4 0/0 6]
```

```
warning: division by zero
```

```
x =
```

```
1      2      3      4      NaN      6
```

```
octave:90> isnan(x)
```

```
ans =
```

```
0  0  0  0  1  0
```

- `isinf()` chequea si hay valores *Inf*, devolviendo una matriz de unos y ceros

```
octave:91> isinf(x)
```

```
ans =
```

```
0  0  0  0  0  0
```

- `isfinite()` chequea si los valores son finitos

```
octave:94> isfinite(x)
```

```
ans =
```

```
1  1  1  1  0  1
```

- `isempty()` chequea si un vector o matriz está vacío
- `ischar()` chequea si una variable es una cadena de caracteres (*string*)

```
octave:95> ischar(x)
```

```
ans = 0
```

- `isglobal()` chequea si una variable es global

```
octave:97> isglobal(y)
```

```
ans = 0
```



- issparse() chequea si una matriz es dispersa (*sparse*, es decir, con un gran número de elementos cero)

2. Otros tipos de datos de GNU OCTAVE

En los Capítulos precedentes se ha visto la “especialidad” de GNU OCTAVE: trabajar con vectores y matrices. En este Capítulo se va a ver que GNU OCTAVE puede también trabajar con otros tipos de datos:

1. Conjuntos o cadenas de caracteres, fundamentales en cualquier lenguaje de programación.
2. Hipermatrices, o matrices de más de dos dimensiones.
3. Estructuras, o agrupaciones bajo un mismo nombre de datos de naturaleza diferente.
4. Vectores o matrices de celdas (cell arrays), que son vectores o matrices cuyos elementos pueden ser cualquier otro tipo de dato.

2.1. Cadenas de caracteres

GNU OCTAVE trabaja también con *cadenas de caracteres*, con ciertas semejanzas y también diferencias respecto a C/C++. A continuación se explica lo más importante del manejo de cadenas de caracteres en GNU OCTAVE. Los caracteres de una cadena se almacenan en un vector, con un carácter por elemento. Cada carácter ocupa dos bytes. Las cadenas de caracteres van entre *apóstrofos* o *comillas simples*, como por ejemplo: 'cadena'. Si la cadena debe contener comillas, éstas se representan por un doble carácter comilla, de modo que se pueden distinguir fácilmente del principio y final de la cadena. Por ejemplo, para escribir la cadena **ni 'idea'** se escribiría **'ni''idea'''**.

Una *matriz de caracteres* es una matriz cuyos elementos son caracteres, o bien una matriz cuyas filas son cadenas de caracteres. Todas las filas de una *matriz de caracteres* deben tener el *mismo número de elementos*. Si es preciso, las cadenas (filas) más cortas se completan con blancos.

Las funciones más importantes para manejo de cadenas de caracteres son las siguientes:



- `double(c)` convierte en números ASCII cada carácter
- `char(v)` convierte un vector de números `v` en una cadena de caracteres
- `char(c1,c2)` crea una matriz de caracteres, completando con blancos las cadenas más cortas
- `deblank(c)` elimina los blancos al final de una cadena de caracteres
- `disp(c)` imprime el texto contenido en la variable `c`
- `ischar(c)` detecta si una variable es una cadena de caracteres
- `isletter()` detecta si un carácter es una letra del alfabeto. Si se le pasa un vector o matriz de caracteres devuelve un vector o matriz de unos y ceros
- `isspace()` detecta si un carácter es un espacio en blanco. Si se le pasa un vector o matriz de caracteres devuelve un vector o matriz de unos y ceros
- `strcmp(c1,c2)` comparación de cadenas. Si las cadenas son iguales devuelve un uno, y si no lo son, devuelve un cero (funciona de modo diferente que la correspondiente función de C)
- `strcmpi(c1,c2)` igual que `strcmp(c1,c2)`, pero ignorando la diferencia entre mayúsculas y minúsculas
- `strncmp(c1,c2,n)` compara los `n` primeros caracteres de dos cadenas
- `c1==c2` compara dos cadenas carácter a carácter. Devuelve un vector o matriz de unos y ceros
- `s=[s,' y más']` concatena cadenas, añadiendo la segunda a continuación de la primera
- `findstr(c1,c2)` devuelve un vector con las posiciones iniciales de todas las veces en que la cadena más corta aparece en la más larga
- `num2str(x,n)` convierte un número real `x` en su expresión por medio de una cadena de caracteres, con cuatro cifras decimales por defecto (pueden especificarse más cifras, con un argumento opcional `n`)
- `str2double(str)` convierte una cadena de caracteres representando un número real en el número real correspondiente
- `vc=cellstr(cc)` convierte una matriz de caracteres
- `strrep(c1,c2,c3)` sustituye la cadena `c2` por `c3`, cada vez que `c2` es encontrada en `c1`



Manual de Iniciación a GNU Octave

```
octave:3> c=char('mas','madera')
c =
 
mas
madera

octave:4> a=[11,112,121,23]; #podemos crear los caracteres con su
numero en ascii
octave:5> b=char(a)
b=
py_
octave:6> char(11)
ans=

octave:7> char(112)
ans = p
octave:8> char(121)
ans = y
octave:9> char(23)
ans = _
octave:10> double(b)
ans =

11 112 121 23

octave:11> c='espacios      '
c = espacios
octave:12> size(c)
ans =

1 12

octave:13> deblank(c)          #de este modo podemos eliminar los
espacios existentes en el string
ans = espacios
octave:14> size(ans)
ans =

1 8
```



Manual de Iniciación a GNU Octave

```
octave:15> strcmp(a,b)
ans = 0
octave:16> strcmp(b,b)
ans = 1
octave:17> a='en un lugar de la mancha de cuyo nombre no puedo
acordarme'
a = en un lugar de la mancha de cuyo nombre no puedo acordarme
octave:18> b='n';
octave:19> findstr(a,b)
ans =

2    5   21   34   41

octave:20> strmatch(a,b)
ans = [] (0x0)
octave:21> strrep(a,b,'T')
ans = eT uT lugar de la matcha de cuyo Tombre To puedo acordarme
```



2.2. Hipermatrices (arrays de más de dos dimensiones)

GNU OCTAVE permite trabajar con *hipermatrices*, es decir con matrices de más de dos dimensiones. Una posible aplicación es almacenar con un único nombre distintas matrices del mismo tamaño (resulta una hipermatrix de 3 dimensiones). Los elementos de una hipermatrix pueden ser números, caracteres, estructuras, y vectores o matrices de celdas.

El tercer subíndice representa la tercera dimensión la “profundidad” de la hipermatrix.

2.2.1. Definición de Hipermatrices

Las funciones que operan con matrices de más de dos dimensiones son análogas a las funciones vistas previamente, aunque con algunas diferencias. Por ejemplo, las siguientes sentencias generan, en dos pasos, una matriz de 2x2x2:

```
octave:25> rand(2,2,2)
```

```
ans =
```

```
ans(:,:,1) =
```

```
0.31106 0.70197  
0.48694 0.45170
```

```
ans(:,:,2) =
```

```
0.017063 0.261818  
0.234682 0.340703
```



2.3. Estructuras

Una estructura (*struct*) es una agrupación de datos de tipo diferente bajo un mismo nombre. Estos datos se llaman *miembros* (*members*) o *campos* (*fields*). Una estructura es un nuevo tipo de dato, del que luego se pueden crear muchas variables (*objetos* o *instances*). Por ejemplo, la estructura *alumno* puede contener los campos *nombre* (una cadena de caracteres) y *carnet* (un número).

2.3.1 Creación de Estructuras

En GNU OCTAVE la estructura *alumno* se crea creando un objeto de dicha estructura. A diferencia de otros lenguajes de programación, no hace falta definir previamente el modelo o patrón de la estructura. Una posible forma de hacerlo es crear uno a uno los distintos campos, como en el ejemplo siguiente:

```
octave:35> alumno.nombre='jose';
octave:36> alumno.apellido='escriche';
octave:37> alumno
alumno =
{
    apellido = escriche
    nombre = jose
}
```

También puede crearse la estructura por medio de la función *struct()*, como por ejemplo,

```
octave:38> alumno=struct('nombre','jose maria','dni',77335559)
alumno =
{
    dni = 77335559
    nombre = jose maria
}
```

GNU Octave permite, además, añadir un nuevo campo a una estructura en cualquier momento. La siguiente sentencia añade el campo *edad* a todos los elementos del vector *alumno*.



```
octave:39> alumno.edad=23
alumno =
{
  dni = 77334059
  edad = 23
  nombre = jose maria
}
```

Como hemos visto desde un inicio, GNU Octave trabaja con matrices, por lo tanto, todo lo que veamos para un elemento es extrapolable a una matriz, vector o conjunto de estos, por lo tanto, podemos hacer matrices de **structs** al igual que hemos hecho **structs** de vectores/matrices.

2.3.2 Funciones para operar con Estructuras

Las estructuras de GNU OCTAVE disponen de funciones que facilitan su uso. Algunas de estas funciones son las siguientes:

- `isstruct(ST)` permite saber si **ST** es o no una estructura
- `fieldnames(struct)` devuelve un array de celdas con el numero y nombre de los elementos de la estructura
- `isfield (expr, nombre)` nos dice si la structura posee un campo con el nombre indicado.



2.4. Vectores y matrices de Celda

Un vector (matriz o hipermatriz) de celdas es un vector (matriz o hipermatriz) cuyos elementos son cada uno de ellos una variable de tipo cualquiera. En un array ordinario todos sus elementos son números o cadenas de caracteres. Sin embargo, en un *array de celdas*, el primer elemento puede ser un número; el segundo una matriz; el tercero una cadena de caracteres; el cuarto una estructura, etc.

2.4.1 Creación de vectores y matrices de Celdas

Obsérvese por ejemplo cómo se crea, utilizando *llaves* {}, el siguiente vector de celdas,

```
octave:42> vc(1)={[1 1 2]}
```

```
vc =
```

```
{  
[1,1] =
```

```
1 1 2
```

```
}
```

```
octave:43> vc(2)={'jose maria'}  
vc =
```

```
{  
[1,1] =
```

```
1 1 2
```

```
[1,2] = jose maria  
}
```



Manual de Iniciación a GNU Octave

```
octave:44> vc(3)={rand(2,2)}
```

```
vc =
```

```
{
```

```
[1,1] =
```

```
1 1 2
```

```
[1,2] = jose maria
```

```
[1,3] =
```

```
0.39113 0.97938
```

```
0.41501 0.77790
```

```
}
```

Otra nomenclatura alternativa y similar, que también utiliza llaves, es la que podemos ver a continuación.

```
octave:45> v{1}=[1 2 1]
```

```
v =
```

```
{
```

```
[1,1] =
```

```
1 2 1
```

```
}
```



```
octave:46> v{2}='escriche'
v =
{
[1,1] =
1 2 1

[1,2] = escriche
}
```

2.4.2 Funciones para trabajar con vectores y matrices de celdas

GNU OCTAVE dispone de las siguientes funciones para trabajar con *cell arrays*:

- `cell(m,n)` crea un *cell array* vacío de **m** filas y **n** columnas
- `celldisp(ca)` muestra el contenido de todas las celdas de **ca**
- `cellplot(ca)` muestra una representación gráfica de las distintas celdas
- `iscell(ca)` indica si **ca** es un vector de celdas
- `num2cell()` convierte un array numérico en un *cell array*

```
octave:48> c=cell(2,2)
c =
{
[1,1] = []
[2,1] = []
[1,2] = []
[2,2] = []
}
```



Manual de Iniciación a GNU Octave

```
octave:49> c{1,1}='jose maria'  
c =
```

```
{  
[1,1] = jose maria  
[2,1] = [] (0x0)  
[1,2] = [] (0x0)  
[2,2] = [] (0x0)  
}
```

```
octave:50> c{2,2}='azucar'  
c =
```

```
{  
[1,1] = jose maria  
[2,1] = [] (0x0)  
[1,2] = [] (0x0)  
[2,2] = azucar  
}
```

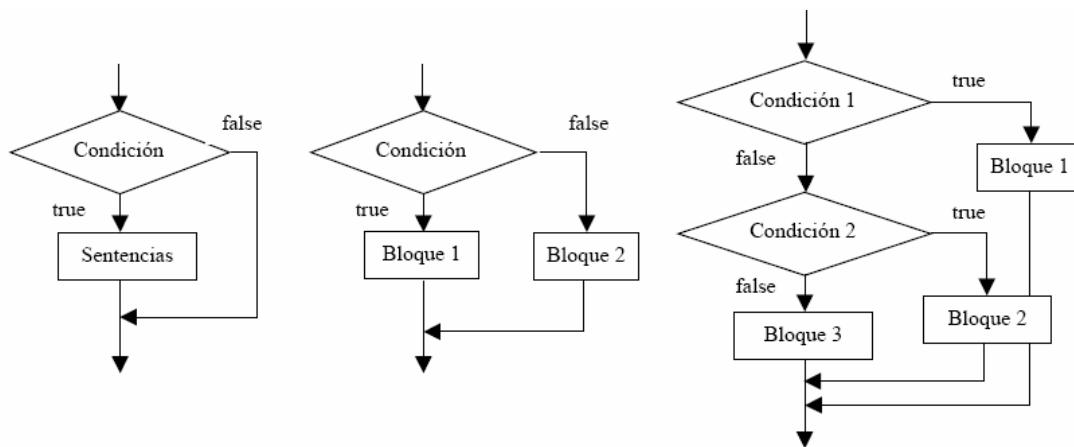


3. Programación en GNU Octave

Este es uno de los puntos flojos de todo este tipo de herramientas de cálculo numérico, ya que no existen grandes posibilidades de programación, aunque sí las formas de programación básicas.

3.1. Bifurcaciones y bucles

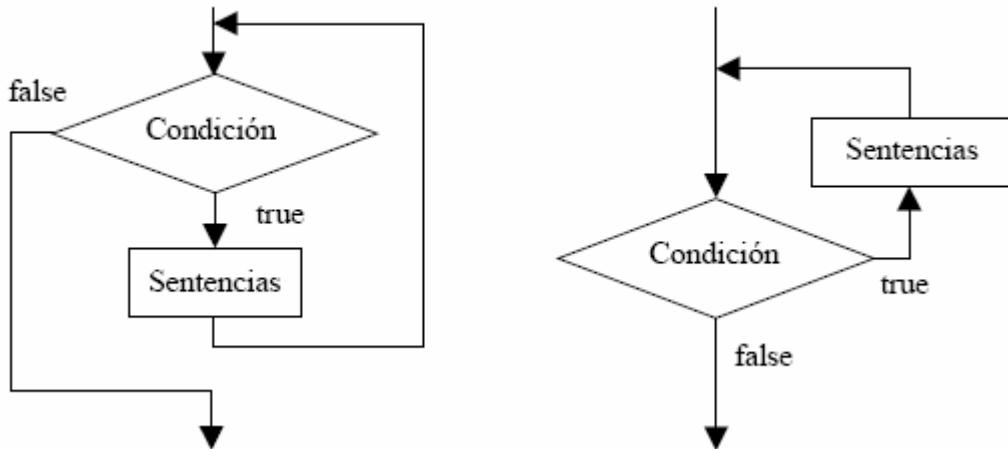
Como todo lenguaje de programación, podemos encontrar bifurcaciones y bucles. Las bifurcaciones sirven para realizar una u otra operación:



Los bucles nos permiten realizar varias iteraciones de un mismo proceso, o sea, realizar una misma operación sobre distintos elementos. Podemos encontrar varios tipos de bucles:

- while
- do-until
- for

Además de las sentencias **break** y **continue** utilizadas dentro de los bucles para salir del proceso.



3.1.1. Sentencia IF

Esta sentencia nos sirve para hacer bifurcaciones, podemos hacer 3 usos diferentes de ella:

- Una sola sentencia que utilizamos si es verdadera y sino no hacemos nada:

```
if (condition)
    then-body
endif
```

- Utilizando la expresión *else* con la que conseguiremos hacer uso de una expresión u otra si es consecuentemente *true o false*.

```
if (condition)
    then-body
else
    else-body
endif
```

- Utilizando la expression *elseif* con la que se pueden anidar bifurcaciones (aunque es mejor usar la sentencia switch)

```
if (condition)
    then-body
elseif (condition)
    elseif-body
else    #es la opción por defecto cuando no se cumple ninguna condición
    else-body
endif
```



Una observación muy importante: la condición del ***if*** puede ser una *condición matricial*, del tipo **A==B**, donde **A** y **B** son matrices del mismo tamaño. Para que se considere que la *condición* se cumple, es necesario que sean *iguales dos a dos todos los elementos* de las matrices **A** y **B**. Basta que haya dos elementos diferentes para que las matrices no sean iguales, y por tanto las sentencias del ***if*** no se ejecuten. Análogamente, una condición en la forma **A~=B** exige que todos los elementos sean diferentes dos a dos. Bastaría que hubiera dos elementos iguales para que la condición no se cumpliese.

En resumen:

- if **A==B** exige que **todos** los elementos sean *iguales* dos a dos
- if **A~=B** exige que **todos** los elementos sean *diferentes* dos a dos

3.1.2. Sentencia SWITCH

Se trata de una sentencia con la que podemos hacer una función similar a la concatenación de sentencias *elseif*, de manera que simplifiquemos el modo de programar:

```
switch expression
case label
    command_list
case label
    command_list
...
otherwise
#con este último agregamos todas las posibilidades que no se
#contemplan en los anteriores
    command_list
endswitch
```

Al principio se evalúa la ***expression***, cuyo resultado debe ser un número escalar o una cadena de caracteres. Este resultado se compara con las ***label***, y se ejecuta el bloque de sentencias que corresponda con ese resultado. Si ninguno es igual a ***expression*** se ejecutan las sentencias correspondientes a ***otherwise***.



3.1.3. Sentencia FOR

Repite una serie de sentencias un número determinado de veces, sin importar los procesos que ocurran dentro, por lo que la única manera de salir del bucle es esperar que acabe (más adelante veremos la sentencia *break*).

```
for var = expression  
    body  
endfor
```

cuando **var** llega al valor **expression** el bucle se detiene.

3.1.4. Sentencia DO-UNTIL

Repite una serie de sentencias hasta que la condición *until* se hace *true*, momento en el que se detiene la ejecución.

```
do  
    body  
until (condition)
```

3.1.5. Sentencia WHILE

Similar a DO-UNTIL salvo que la comprobación de la condición se hace antes de la ejecución de la iteración.

```
while (condition)  
    body  
endwhile
```



3.1.6. Sentencia BREAK y CONTINUE

Al igual que en C/C++/Java, la sentencia ***break*** hace que se termine la ejecución del bucle más interno de los que comprenden a dicha sentencia. La sentencia ***continue*** hace que automáticamente se pare la ejecución de la iteración actual, por lo que vuelve al principio del bucle (sólo sirve para el bucle FOR).

3.2 Ficheros *.m

Los ficheros con extensión (**.m**) son ficheros de texto sin formato (ficheros ASCII) que constituyen el centro de la programación en GNU OCTAVE. Ya se han utilizado en varias ocasiones. Estos ficheros se crean y modifican con un editor de textos cualquiera.

Existen dos tipos de ficheros ***.m**, los ***ficheros de comandos*** (llamados *scripts* en inglés) y las ***funciones***. Los primeros contienen simplemente un conjunto de comandos que se ejecutan sucesivamente cuando se teclea el nombre del fichero en la línea de comandos de GNU OCTAVE. Un fichero de comandos puede llamar a otros ficheros de comandos.

Las ***funciones*** permiten definir funciones enteramente análogas a las de GNU OCTAVE, con su ***nombre***, sus ***argumentos*** y sus ***valores de retorno***. Los ficheros ***.m** que definen funciones permiten extender las posibilidades de GNU OCTAVE. Las funciones definidas en ficheros ***.m** se caracterizan porque la primera línea (que no sea un comentario) comienza por la palabra ***function***, seguida por los ***valores de retorno*** (entre corchetes [] y separados por comas, si hay más de uno), el signo igual (=) y el ***nombre de la función***, seguido de los ***argumentos*** (entre paréntesis y separados por comas).

Recuérdese que un fichero ***.m** puede llamar a otros ficheros ***.m**, e incluso puede llamarse a sí mismo de forma recursiva. Los ficheros de comandos se pueden llamar también desde funciones, en cuyo caso las variables que se crean pertenecen a espacio de trabajo de la función. El espacio de trabajo de una función es independiente del espacio de trabajo base y del espacio de trabajo de las demás funciones. Esto implica por ejemplo que no puede haber colisiones entre nombres de variables aunque varias funciones tengan una variable llamada A, en realidad se trata de variables completamente distintas (a no ser que A haya sido declarada como variable ***global***).



3.2.1 Ficheros de Comandos (*SCRIPTS*)

Los ficheros de comandos o *scripts* son ficheros con un nombre tal como *file1.m* que contienen una sucesión de comandos análoga a la que se teclearía en el uso interactivo del programa. Dichos comandos se ejecutan sucesivamente cuando se teclea el nombre del fichero que los contiene (sin la extensión), es decir cuando se teclea *file1* con el ejemplo considerado.

Cuando se ejecuta desde la línea de comandos, las variables creadas por *file1* pertenecen al espacio de trabajo base de GNU OCTAVE. Por el contrario, si se ejecuta desde una función, las variables que crea pertenecen al espacio de trabajo de la función.

En los ficheros de comandos conviene poner los puntos y coma (;) al final de cada sentencia, para evitar una salida de resultados demasiado cuantiosa. Un fichero *.m puede llamar a otros ficheros *.m, e incluso se puede llamar a sí mismo de modo recursivo.

El comando *echo* hace que se impriman los comandos que están en un *script* a medida que van siendo ejecutados. Este comando tiene varias formas:

- echo on activa el *echo* en todos los ficheros script
- echo off desactiva el *echo*
- echo file on donde 'file' es el nombre de un fichero de función, activa el *echo* en esa función
- echo file off desactiva el *echo* en la función
- echo file pasa de **on** a **off** y viceversa
- echo on all activa el *echo* en todas las funciones
- echo off all desactiva el *echo* de todas las funciones

Mención especial merece el fichero de comandos *octaverc*. Este fichero se ejecuta cada vez que se entra en GNU OCTAVE. En él puede introducir todos aquellos comandos que le interesa se ejecuten siempre al iniciar la sesión, por ejemplo *format compact* y los comandos necesarios para modificar el *path*.



3.2.2 Definición de Funciones

La *primera línea* de un fichero llamado ***name.m*** que define una función tiene la forma:

```
function [lista de valores de retorno] = name(lista de argumentos)
```

donde ***name*** es el nombre de la función. Entre corchetes y separados por comas van los *valores de retorno* (siempre que haya más de uno), y entre paréntesis también separados por comas los *argumentos*. Puede haber funciones sin valor de retorno y también sin argumentos. Recuérdese que los *argumentos* son los *datos* de la función y los *valores de retorno* sus *resultados*. Si no hay valores de retorno se omiten los corchetes y el signo igual (=); si sólo hay un valor de retorno no hace falta poner corchetes. Tampoco hace falta poner paréntesis si no hay argumentos.

Las variables definidas dentro de una función son *variables locales*, en el sentido de que son inaccesibles desde otras partes del programa y en el de que no interfieren con variables del mismo nombre definidas en otras funciones o partes del programa. Se puede decir que pertenecen al propio espacio de trabajo de la función y no son vistas desde otros espacios de trabajo. Para que la función tenga acceso a variables que no han sido pasadas como argumentos es necesario declarar dichas variables como *variables globales*, tanto en el programa principal como en las distintas funciones que deben acceder a su valor.

Dentro de la función, los valores de retorno deben ser calculados en algún sitio. De todas formas, no hace falta calcular siempre todos los posibles valores de retorno de la función, sino sólo los que el usuario espera obtener en la sentencia de llamada a la función. En cualquier función existen dos variables definidas de modo automático llamadas ***nargin*** y ***nargout***, que representan respectivamente el número de argumentos y el número de valores de retorno con los que la función ha sido llamada. Dentro de la función, estas variables pueden ser utilizadas como el programador desee ya que las variables en Octave son pasadas por valor.

La ejecución de una función termina cuando se llega a su última sentencia ejecutable. Si se quiere forzar el que una función termine de ejecutarse se puede utilizar la sentencia ***return***, que devuelve inmediatamente el control al entorno de llamada.



3.2.3. HELP para las funciones de usuario

También las funciones creadas por el usuario pueden tener su ***help***, análogo al que tienen las propias funciones de GNU OCTAVE. Esto se consigue de la siguiente forma: las primeras líneas de comentarios de cada fichero de función son muy importantes, pues permiten construir un ***help*** sobre esa función.

En otras palabras, cuando se teclea en la ventana de comandos de GNU OCTAVE:

```
help mi_func
```

el programa responde escribiendo las primeras líneas del fichero ***mi_func.m*** que comienzan por el carácter “#”, es decir, que son comentarios.



4. Gráficos bidimensionales

A estas alturas, después de ver cómo funciona este programa, a nadie le puede resultar extraño que los gráficos 2-D de GNU Octave estén fundamentalmente orientados a la representación gráfica de vectores (y matrices). En el caso más sencillo los argumentos básicos de la función `plot` van a ser vectores. Cuando una matriz aparezca como argumento, se considerará como un conjunto de vectores columna (en algunos casos también de vectores fila).

GNU Octave utiliza un tipo especial de ventanas para realizar las operaciones gráficas. Ciertos comandos abren una ventana nueva y otros dibujan sobre la ventana activa, bien sustituyendo lo que hubiera en ella, bien añadiendo nuevos elementos gráficos a un dibujo anterior. Todo esto se verá con más detalle en las siguientes secciones.

Todo este sistema de gráficos está implementado a partir de GNU Plot, por lo que si fuera necesario, siempre podríamos llamar directamente a este programa en vez de dejar que sea Octave el que lo haga por nosotros.

4.1 Funciones gráficas 2D elementales

GNU Octave dispone de cuatro funciones básicas para crear gráficos 2-D. Estas funciones se diferencian principalmente por el tipo de escala que utilizan en los ejes de abscisas y de ordenadas. Estas cuatro funciones son las siguientes:

- `plot()` crea un gráfico a partir de vectores y/o columnas de matrices, con escalas lineales sobre ambos ejes.
- `loglog()` ídem con escala logarítmica en ambos ejes
- `semilogx()` ídem con escala lineal en el eje de ordenadas y logarítmica en el eje de abscisas
- `semilogy()` ídem con escala lineal en el eje de abscisas y logarítmica en el eje de ordenadas

En lo sucesivo se hará referencia casi exclusiva a la primera de estas funciones (`plot`). Las demás se pueden utilizar de un modo muy similar.



Existen además otras funciones orientadas a añadir títulos al gráfico, a cada uno de los ejes, a dibujar una cuadrícula auxiliar, a introducir texto, etc. Estas funciones son las siguientes:

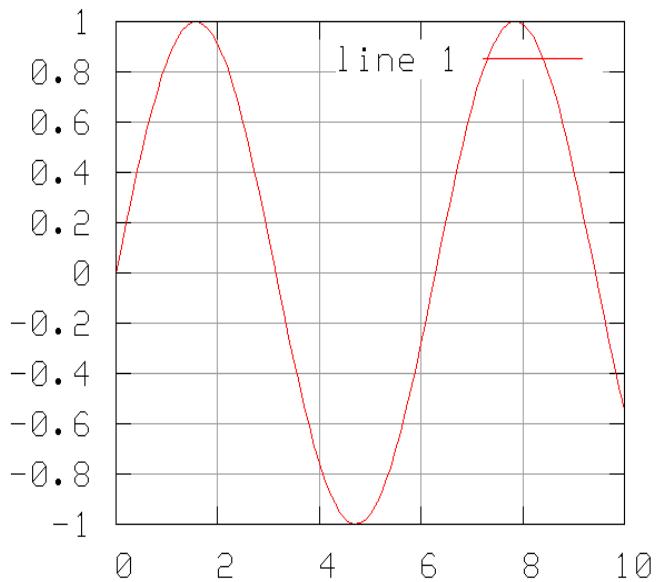
- `title('título')` añade un título al dibujo.
- `xlabel('tal')` añade una etiqueta al eje de abscisas. Con `xlabel off` desaparece.
- `ylabel('cual')` añade una etiqueta al eje de ordenadas. Con `ylabel off` desaparece.
- `text(x,y,'texto')` introduce 'texto' en el lugar especificado por las coordenadas x e y. Si x e y son vectores, el texto se repite por cada par de elementos. Si texto es también un vector de cadenas de texto de la misma dimensión, cada elemento se escribe en las coordenadas correspondientes.
- `legend()` define rótulos para las distintas líneas o ejes utilizados en la figura. Para más detalle, consultar el Help.
- `grid` activa la inclusión de una cuadrícula en el dibujo. Con `grid off` desaparece la cuadrícula.

Los dos grupos de funciones anteriores no actúan de la misma forma. Así, la función *plot* dibuja una nueva figura en la ventana activa (en todo momento GNU Octave tiene una ventana activa de entre todas las ventanas gráficas abiertas), o abre una nueva figura si no hay ninguna abierta, sustituyendo cualquier cosa que hubiera dibujada anteriormente en esa ventana.



Ahora se deben ejecutar los comandos siguientes:

```
octave:2> grid  
octave:3> x=0:0.1:10;y=sin(x);  
octave:4> plot(x,y)
```



Más adelante se verá que con la función hold pueden añadirse gráficos a una figura ya existente respetando su contenido.



4.1.1 Función PLOT

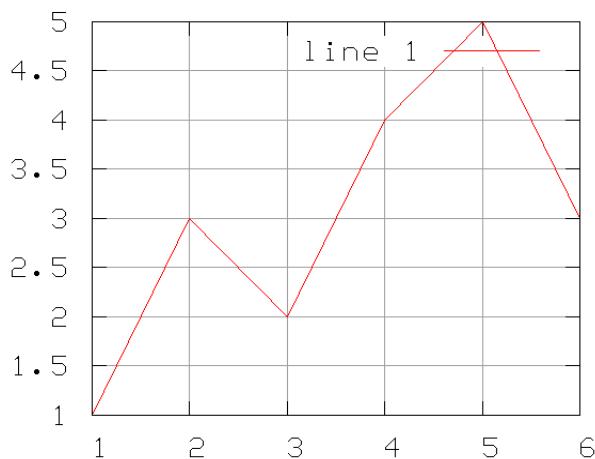
Esta es la función clave de todos los gráficos 2-D en GNU Octave. Ya se ha dicho que el elemento básico de los gráficos bidimensionales es el vector. Se utilizan también cadenas de 1, 2 ó 3 caracteres para indicar colores y tipos de línea. La función `plot()`, en sus diversas variantes, no hace otra cosa que dibujar vectores. Un ejemplo muy sencillo de esta función, en el que se le pasa un único vector como argumento, es el siguiente:

```
octave:6> x=[1 3 2 4 5 3],plot(x)
x =
```

```
1 3 2 4 5 3
```

El resultado de este comando es que se abre una ventana. Por defecto, los distintos puntos del gráfico se unen con una línea continua. También por defecto, el color que se utiliza para la primera línea es el rojo.

Cuando a la función `plot()` se le pasa un único vector –real– como argumento, dicha función dibuja en ordenadas el valor de los n elementos del vector frente a los índices 1, 2, ... n del mismo en abscisas. Más adelante se verá que si el vector es complejo, el funcionamiento es bastante diferente.





En la pantalla de su ordenador se habrá visto que GNU Octave utiliza por defecto color blanco para el fondo de la pantalla y otros colores más oscuros para los ejes y las gráficas.

Una segunda forma de utilizar la función `plot()` es con dos vectores como argumentos. En este caso los elementos del segundo vector se representan en ordenadas frente a los valores del primero, que se representan en abscisas. Véase por ejemplo cómo se puede dibujar un cuadrilátero de esta forma (obsérvese que para dibujar un polígono cerrado el último punto debe coincidir con el primero):

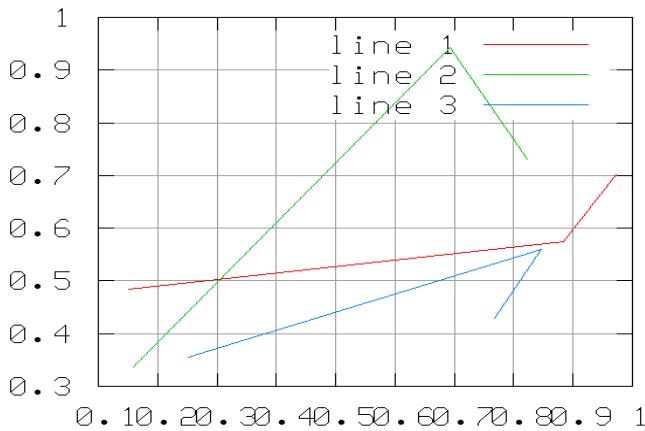
```
octave:8>x=[1 6 5 2 1]; y=[1 0 4 3 1];
octave:9>plot(x,y)
```

La función `plot()` permite también dibujar múltiples curvas introduciendo varias parejas de vectores como argumentos. En este caso, cada uno de los segundos vectores se dibujan en ordenadas como función de los valores del primer vector de la pareja, que se representan en abscisas. Obsérvese bien cómo se dibujan el seno y el coseno en el siguiente ejemplo:

```
octave:10> x=0:pi/25:6*pi;
octave:11>y=sin(x); z=cos(x);
octave:12>plot(x,y,x,z)
```

Ahora se va a ver lo que pasa con los vectores complejos. Si se pasan a `plot()` arios vectores complejos como argumentos, GNU Octave simplemente representa las partes reales y desprecia las partes imaginarias.

```
octave:12> a=rand(3,3)+rand(3,3)*i;
octave:13> plot(a)
```



Como ya se ha dicho, si se incluye más de un vector complejo como argumento, se ignoran las partes imaginarias. Si se quiere dibujar varios vectores complejos, hay que separar explícitamente las partes reales e imaginarias de cada vector, como se acaba de hacer en el último ejemplo.

El comando `plot` puede utilizarse también con matrices como argumentos. Véanse algunos ejemplos sencillos:

- `plot(A)` dibuja una línea por cada columna de A en ordenadas, frente al índice de los elementos en abscisas
- `plot(x,A)` dibuja las columnas (o filas) de A en ordenadas frente al vector x en abscisas. Las dimensiones de A y x deben ser coherentes: si la matriz A es cuadrada se dibujan las columnas, pero si no lo es y la dimensión de las filas coincide con la de x, se dibujan las filas
- `plot(A,x)` análogo al anterior, pero dibujando las columnas (o filas) de A en abscisas, frente al valor de x en ordenadas
- `plot(A,B)` dibuja las columnas de B en ordenadas frente a las columnas de A en abscisas, dos a dos. Las dimensiones deben coincidir
- `plot(A,B,C,D)` análogo al anterior para cada par de matrices. Las dimensiones de cada par deben coincidir, aunque pueden ser diferentes de las dimensiones de los demás pares.

Se puede obtener una excelente y breve descripción de la función `plot()` con el comando `help plot`. La descripción que se acaba de presentar se completará en la siguiente sección, en donde se verá cómo elegir los colores y los tipos de línea.



4.1.2 Estilos de Línea y Marcadores para PLOT

En la sección anterior se ha visto cómo la tarea fundamental de la función plot() era dibujar los valores de un vector en ordenadas, frente a los valores de otro vector en abscisas. En el caso general esto exige que se pasen como argumentos un par de vectores. En realidad, el conjunto básico de argumentos de esta función es una tripleta formada por dos vectores y una cadena de 1, 2 ó 3 caracteres que indica el color y el tipo de línea o de marcador. En la tabla siguiente se pueden observar las distintas posibilidades.

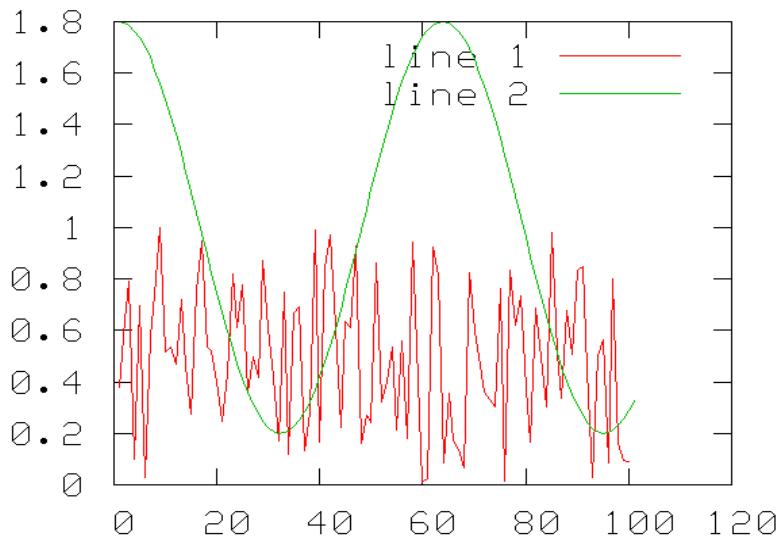
Símbolo	Color	Símbolo	Marcadores
y	yellow	.	puntos
m	magenta	o	círculos
c	cyan	x	marcas en x
r	red	+	marcas en +
g	green	*	marcas en *
b	blue	s	marcas cuadradas (square)
w	white	d	marcas en diamante (diamond)
k	black	^	triángulo apuntando arriba



4.1.3 Añadir Líneas a un gráfico ya existente

Existe la posibilidad de añadir líneas a un gráfico ya existente, sin destruirlo o sin abrir una nueva ventana. Se utilizan para ello los comandos `hold on` y `hold off`. El primero de ellos hace que los gráficos sucesivos respeten los que ya se han dibujado en la figura (es posible que haya que modificar la escala de los ejes); el comando `hold off` deshace el efecto de `hold on`. El siguiente ejemplo muestra cómo se añaden las gráficas de `x2` y `x3` a la gráfica de `x` previamente creada (cada una con un tipo de línea diferente):

```
octave:16> closeplot
octave:17> x1=rand(100,1);
octave:18> x2=1+0.8*cos(0:0.1:10);
octave:19> plot(x1)
octave:20> hold on
octave:21> plot(x2)
octave:22> print('plot4.png', '-dpng')
```





4.1.4 Comando SUBPLOT

Una ventana gráfica se puede dividir en m particiones horizontales y n verticales, con objeto de representar múltiples gráficos en ella. Cada una de estas subventanas tiene sus propios ejes, aunque otras propiedades son comunes a toda la figura. La forma general de este comando es:

```
subplot(m,n,i)
```

donde m y n son el número de subdivisiones en filas y columnas, e i es la subdivisión que se convierte en activa. Las subdivisiones se numeran consecutivamente empezando por las de la primera fila, siguiendo por las de la segunda, etc. Por ejemplo, la siguiente secuencia de comandos genera cuatro gráficos en la misma ventana:

```
octave:23> x=0:0.01:10; y=sin(x); z=cos(x); w=exp(-x*.1).*y; v=y.*z;
octave:24> subplot(2,2,1), plot(x,y)
octave:25> subplot(2,2,2), plot(x,z)
octave:26> subplot(2,2,3), plot(x,w)
octave:27> subplot(2,2,4), plot(x,v)
```

Se puede practicar con este ejemplo añadiendo títulos a cada gráfica, así como rótulos para los ejes. Se puede intentar también cambiar los tipos de línea.

4.1.5 Control de los Ejes

También en este punto GNU Octave tiene sus opciones por defecto, que en algunas ocasiones puede interesar cambiar. El comando básico es el comando axis. Por defecto, GNU Octave ajusta la escala de cada uno de los ejes de modo que varíe entre el mínimo y el máximo valor de los vectores a representar. Este es el llamado modo "auto", o modo automático. Para definir de modo explícito los valores máximo y mínimo según cada eje, se utiliza el comando:

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

mientras que :

```
axis('auto')
```

devuelve el escalado de los ejes al valor por defecto o automático.



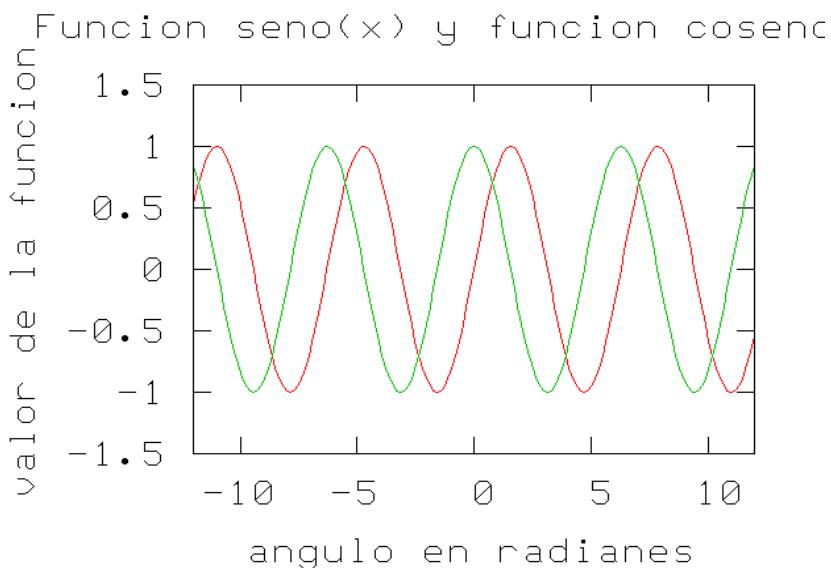
4.2 Control de ventanas gráficas: Función figure

Si se llama a la función *figure* sin argumentos, se crea una nueva ventana gráfica con el número consecutivo que le corresponda. El valor de retorno es dicho número.

Por otra parte, el comando *figure(n)* hace que la ventana n pase a ser la ventana o figura activa. Si dicha ventana no existe, se crea una nueva ventana con el número consecutivo que le corresponda (que se puede obtener como valor de retorno del comando). La función *close* cierra la figura activa, mientras que *close(n)* cierra la ventana o figura número n.

Para practicar un poco con todo lo que se acaba de explicar, ejecútense las siguientes instrucciones de GNU Octave, observando con cuidado los efectos de cada una de ellas en la ventana activa.

```
octave:41> plot(x,sin(x), 'r',x,cos(x), 'g')
octave:42> title('Función seno(x) -en rojo- y función coseno(x) -en
verde-')
octave:43> xlabel('ángulo en radianes')
octave:43> ylabel('valor de la función trigonométrica')
octave:44> axis([-12,12,-1.5,1.5])
```





4.3 Otras funciones gráficas 2-D

Existen otras funciones gráficas bidimensionales orientadas a generar otro tipo de gráficos distintos de los que produce la función `plot()` y sus análogas. Algunas de estas funciones son las siguientes (para más información sobre cada una de ellas en particular, utilizar `help nombre_función`):

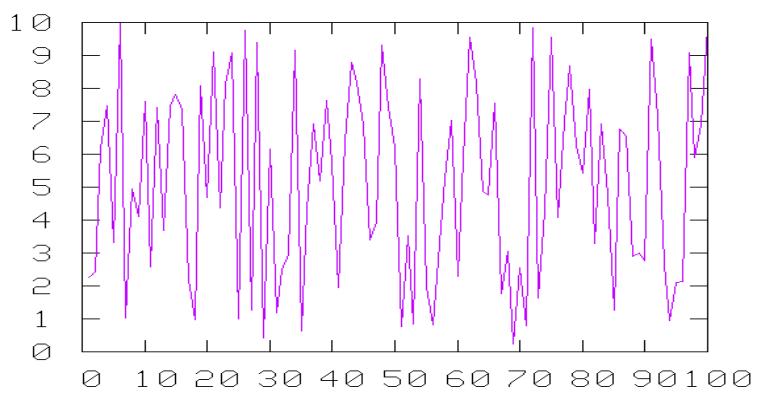
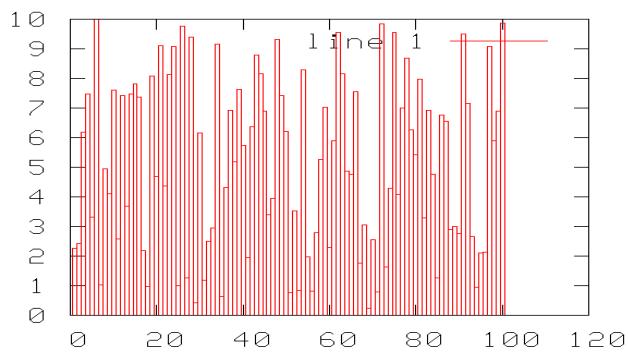
- `bar()` crea diagramas de barras
- `barh()` diagramas de barras horizontales
- `bar3()` diagramas de barras con aspecto 3-D
- `bar3h()` diagramas de barras horizontales con aspecto 3-D
- `pie()` gráficos con forma de “tarta”
- `pie3()` gráficos con forma de “tarta” y aspecto 3-D
- `area()` similar `plot()`, pero rellenando en ordenadas de 0 a y
- `stairs()` función análoga a `bar()` sin líneas internas
- `errorbar()` representa sobre una gráfica –mediante barras– valores de errores.
- `compass()` dibuja los elementos de un vector complejo como un conjunto de vectores partiendo de un origen común
- `feather()` dibuja los elementos de un vector complejo como un conjunto de vectores partiendo de orígenes uniformemente espaciados sobre el eje de abscisas.
- `hist()` dibuja histogramas de un vector

A modo de ejemplo, genérese un vector de valores aleatorios entre 0 y 10, y ejecútense los siguientes comandos:

```
octave:53> x=[rand(1,100)*10];
octave:69> plot(x, 'm')
octave:71> bar(x)
```



Manual de Iniciación a GNU Octave





5. Gráficos tridimensionales

Quizás sea ésta una de las características de GNU OCTAVE que más admiración despierta entre los usuarios no técnicos (cualquier alumno de ingeniería sabe que hay ciertas operaciones algebraicas como la descomposición de valor singular, sin ir más lejos, que tienen dificultades muy superiores, aunque "luzcan" menos).

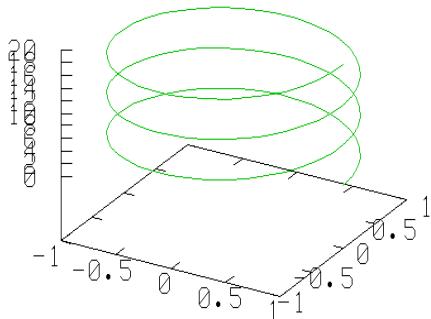
5.1 Tipos de funciones gráficas tridimensionales

GNU Octave tiene posibilidades de realizar varios tipos de gráficos 3D. Para darse una idea de ello, lo mejor es verlo en la pantalla cuanto antes, aunque haya que dejar las explicaciones detalladas para un poco más adelante.

La primera forma de gráfico 3D es la función `plot3`, que es el análogo tridimensional de la función `plot`. Esta función dibuja puntos cuyas coordenadas están contenidas en 3 vectores, bien uniéndolos mediante una línea continua (defecto), bien mediante markers.

Asegúrese de que no hay ninguna ventana gráfica abierta y ejecute el siguiente comando que dibuja una línea espiral:

```
octave:1> fi=[0:pi/20:6*pi]; plot3(cos(fi),sin(fi),fi,'g')
```



Ahora se verá cómo se representa una función de dos variables. Para ello se va a definir una función de este tipo en un fichero llamado `test3d.m`. La fórmula será la siguiente:



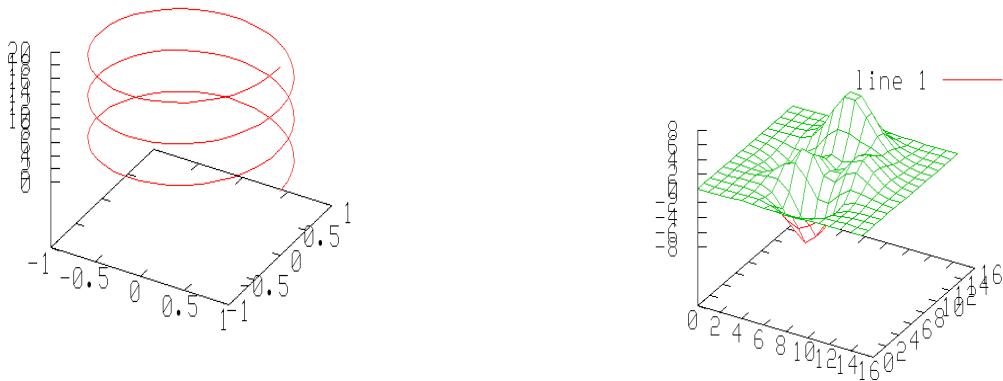
Manual de Iniciación a GNU Octave

```
function z=test3d(x,y)
z = 3*(1-x).^2.*exp(-(x.^2)-(y+1).^2) ...
- 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
- 1/3*exp(-(x+1).^2 - y.^2);
```

Ahora, ejecútese la siguiente lista de comandos (directamente, o mejor creando un fichero test3dFC.m que los contenga):

```
x=[-3:0.4:3]; y=x;
closeplot
figure(), fi=[0:pi/20:6*pi];
plot3(cos(fi),sin(fi),fi,'r')
[X,Y]=meshgrid(x,y);
Z=test3d(X,Y);
figure(), mesh(Z)
```

Estos son unos ejemplos de las posibilidades que GNU Octave ofrece para el uso de gráficos 3-D.





5.1.1 Dibujo de líneas: Función PLOT3

La función plot3 es análoga a su homóloga bidimensional plot. Su forma más sencilla es la siguiente:

```
octave:81>plot3(x,y,z)
```

que dibuja una línea que une los puntos $(x(1), y(1), z(1))$, $(x(2), y(2), z(2))$, etc.

Y la proyecta sobre un plano para poderla representar en la pantalla. Al igual que en el caso plano, se puede incluir una cadena de 1, 2 ó 3 caracteres para determinar el color, los markers, y el tipo de línea:

```
octave:82>plot3(x,y,z,s)
```

También se pueden utilizar tres matrices X, Y y Z del mismo tamaño:

```
octave:83>plot3(X,Y,Z)
```

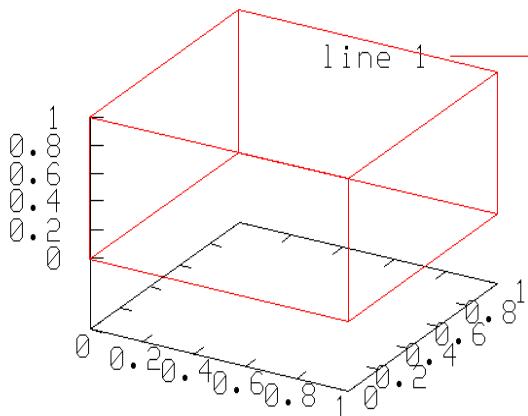
en cuyo caso se dibujan tantas líneas como columnas tienen estas 3 matrices, cada una de las cuales está definida por las 3 columnas homólogas de dichas matrices.

A continuación se va a realizar un ejemplo sencillo consistente en dibujar un cubo. Para ello se creará una matriz que contenga las aristas correspondientes, definidas mediante los vértices del cubo como una línea poligonal continua (obsérvese que algunas aristas se dibujan dos veces). La matriz A cuyas columnas son las coordenadas de los vértices, y cuyas filas son las coordenadas x, y y z de los mismos:

```
octave:86>A=[0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 1 0 0  
0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 1 1  
0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 1 0];
```

Ahora basta ejecutar los comandos siguientes (el trasponer los vectores en este caso es opcional):

```
octave:87>plot3(A(1,:) ',A(2,:) ',A(3,:) ')
```



5.1.2 Dibujo de mallados: Funciones MESHGRID, MESH Y SURF

Ahora se verá con detalle cómo se puede dibujar una función de dos variables ($z=f(x,y)$) sobre un dominio rectangular. Se verá que también se pueden dibujar los elementos de una matriz como función de los dos índices.

Sean x e y dos vectores que contienen las coordenadas en una y otra dirección de la retícula (grid) sobre la que se va a dibujar la función. Después hay que crear dos matrices X (cuyas filas son copias de x) e Y (cuyas columnas son copias de y). Estas matrices se crean con la función `meshgrid`. Estas matrices representan respectivamente las coordenadas x e y de todos los puntos de la retícula. La matriz de valores Z se calcula a partir de las matrices de coordenadas X e Y . Finalmente hay que dibujar esta matriz Z con la función `mesh`, cuyos elementos son función elemento a elemento de los elementos de X e Y . Véase como ejemplo el dibujo de la función $\sin(r)/r$ (siendo $r=\sqrt{x^2+y^2}$; para evitar dividir por 0 se suma al denominador el número pequeño eps). Para distinguirla de la función `test3d` anterior se utilizará u y v en lugar de x e y .

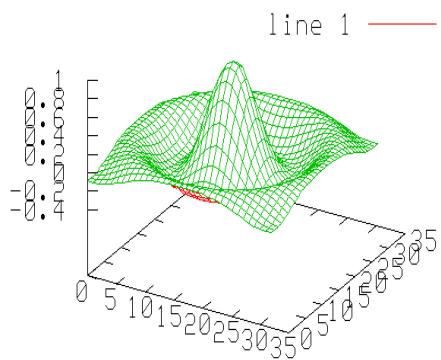
Créese un fichero llamado `sombrero.m` que contenga las siguientes líneas:

```
closeplot  
u=-8:0.5:8; v=u;  
[U,V]=meshgrid(u,v);  
R=sqrt(U.^2+V.^2)+eps;
```



Manual de Iniciación a GNU Octave

```
W=sin(R)./R;  
mesh(W)
```



Se habrá podido comprobar que la función `mesh` dibuja en perspectiva una función en base a una retícula de líneas de colores, rodeando cuadriláteros del color de fondo, con eliminación de líneas ocultas. Ejecútese ahora el comando:

```
octave:91> surf(W)
```

En vez de líneas aparece ahora una superficie facetada (aunque no es fácilmente visible, pero de manera teórica es así). El color de las facetas depende también del valor de la función.



5.1.3 Dibujo de líneas de contorno: Función CONTOUR

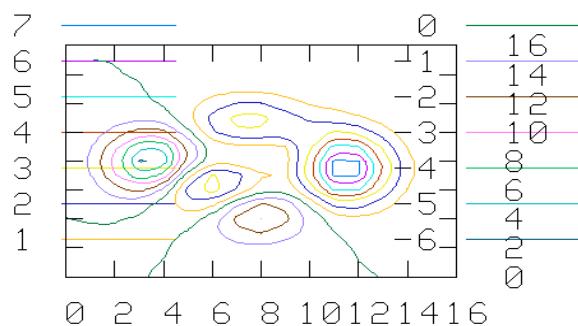
Una forma distinta de representar funciones tridimensionales es por medio de isolíneas o curvas de nivel. A continuación se verá cómo se puede utilizar estas representaciones con las matrices de datos Z y W que se han calculado previamente:

```
octave:3>contour(Z,20)
octave:5>contour(W,20)
```

donde "20" representa el número de líneas de nivel. Si no se pone se utiliza un número por defecto.

Otras posibles formas de estas funciones son las siguientes:

- `contour(Z, val)` siendo val un vector de valores para las isolíneas a dibujar.
- `contour(u,v,W,20)` se utilizan u y v para dar valores a los ejes de coordenadas.





5.2. Elementos Generales: Ejes, Puntos de vista, líneas ocultas,...

Las funciones *surf* y *mesh* dibujan funciones tridimensionales en perspectiva. La localización del punto de vista o dirección de observación se puede hacer mediante la función *view*, que tiene la siguiente forma:

```
view(azimut, elev)
```

donde *azimut* es el ángulo de rotación de un plano horizontal, medido sobre el eje z a partir del eje x en sentido antihorario, y *elev* es el ángulo de elevación respecto al plano (x-y). Ambos ángulos se miden en grados, y pueden tomar valores positivos y negativos. También se puede definir la dirección del punto de vista mediante las tres coordenadas cartesianas de un vector (sólo se tiene en cuenta la dirección):

```
view([xd, yd, zd])
```

En los gráficos tridimensionales existen funciones para controlar los ejes, por ejemplo:

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

También se pueden utilizar las funciones siguientes: *xlabel*, *ylabel*, *zlabel*, *axis('auto')*, *axis(axis)*, etc.

Las funciones *mesh* y *surf* disponen de un algoritmo de eliminación de líneas ocultas (los polígonos o facetas, no dejan ver las líneas que están detrás). El comando *hidden* activa y desactiva la eliminación de líneas ocultas.

En el dibujo de funciones tridimensionales, a veces también son útiles los NaNs. Cuando una parte de los elementos de la matriz de valores Z son NaNs, esa parte de la superficie no se dibuja, permitiendo ver el resto de la superficie.



6. Otros aspectos de GNU Octave

6.1 Guardar variables y estados de una sesión: Comandos *save* y *load*

En muchas ocasiones puede resultar interesante interrumpir el trabajo con GNU Octave y poderlo recuperar más tarde en el mismo punto en el que se dejó (con las mismas variables definidas, con los mismos resultados intermedios, etc.). Hay que tener en cuenta que al salir del programa todo el contenido de la memoria se borra automáticamente.

Para guardar el estado de una sesión de trabajo en el *directorio actual* existe el comando **save**. Si se teclea:

```
octave:2>save
```

antes de abandonar el programa, se crea un fichero binario con el estado de la sesión (excepto los gráficos, que por ocupar mucha memoria hay que guardar aparte).

Dicho estado puede recuperarse la siguiente vez que se arranque el programa con el comando:

```
octave:3>load
```

Esta es la forma más básica de los comandos **save** y **load**. Se pueden guardar también matrices y vectores de forma selectiva y en ficheros con nombre especificado por el usuario. Por ejemplo, el comando:

```
octave:4>save "filename" A, x, y
```

guarda las variables **A**, **x** e **y** en un fichero binario llamado *filename*. Para recuperarlas en otra sesión basta teclear:

```
octave:5>load "filename"
```



Manual de Iniciación a GNU Octave

Si no se indica ningún nombre de variable, se guardan todas las variables creadas en esa sesión.

El comando *save* permite guardar el estado de la sesión en formato ASCII utilizándolo de la siguiente forma:

```
octave:6>save -ascii #guarda en formato compatible con matlab  
octave:7>save -text #guarda en formato texto  
octave:8>save -mat-binary #guarda en formato compatible con matlab en  
binario  
octave:9>save -hdf5 #guarda en formato hdf5
```

Cuando se recuperan estos ficheros con *load -tipo* la información se recupera, hay que tener cuidado si guardamos en -ascii o en -text dado que la información recuperada se hace en una sola matriz.



6.2 Guardar sesión: Comando *diary*

Los comandos *save* y *load* crean ficheros binarios o ASCII con el estado de la sesión. Existe otra forma más sencilla de almacenar en un fichero un texto que describa lo que el programa va haciendo. El comando *diary* nos permite guardar el proceso de trabajo en un archivo de tipo texto:

```
octave:32>diary "filename"
```

6.3 Medida de tiempos y de esfuerzo de cálculo

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

- **cputime** devuelve el tiempo de CPU (con precisión de centésimas de segundo) desde que el programa arrancó. Llamando antes y después de realizar una operación y restando los valores devueltos, se puede saber el tiempo de CPU empleado en esa operación. Este tiempo sigue corriendo aunque GNU Octave esté inactivo.
- **etime(t2, t1)** tiempo transcurrido entre los vectores **t1** y **t2** (¡atención al orden!), obtenidos como respuesta al comando *clock*.
- **tic** *operacioness toc* imprime el tiempo en segundos requerido por *ops*. El comando **tic** pone el reloj a cero y **toc** obtiene el tiempo transcurrido.



6.4. Funciones de función

En GNU Octave existen funciones a las que hay que pasar como argumento el nombre de otras funciones, para que puedan ser llamadas desde dicha función. Así sucede por ejemplo si se desea calcular la integral definida de una función, resolver una ecuación no lineal, o integrar numéricamente una ecuación diferencial ordinaria (problema de valor inicial). Estos serán los tres casos –de gran importancia práctica– que se van a ver a continuación. Se comenzará por medio de un ejemplo, utilizando una función llamada *prueba* que se va a definir en un fichero llamado *prueba.m*

```
function y=prueba(x)
y = 1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6;
```

6.4.1 Integración numérica de funciones

Lo primero que se va a hacer es calcular la integral definida de esta función entre dos valores de la abscisa *x*. En inglés, al cálculo numérico de integrales definidas se le llama *quadrature*. Sabiendo eso, no resulta extraño el comando con el cual se calcula el área comprendida bajo la función entre los puntos 0 y 1 (obsérvese que el nombre de la función a integrar se pasa entre apóstrofos, como cadena de caracteres):

```
octave:45>area = quad('prueba', 0, 1)

area =
29.8583
```

Si se teclea *help quad* se puede obtener más de información sobre esta función.



6.4.2 Integración Numérica de Ecuaciones Diferenciales Ordinarias

GNU Octave dispone de varias funciones capaces de realizar la integración de ecuaciones diferenciales; si hacemos uso del fichero **tiropar.m**:

```
function deriv=tiropar(t,y)
fac=-(0.001/1.0)*sqrt((y(1)^2+y(2)^2));
deriv=zeros(4,1);
deriv(1)=fac*y(1);
deriv(2)=fac*y(2)-9.8;
deriv(3)=y(1);
deriv(4)=y(2);
end
```

donde se han supuesto unas constantes con los valores de $c=0.001$, $m=1$ y $g=9.8$. Falta fijar los valores iniciales de posición y velocidad. Se supondrá que el proyectil parte del origen con una velocidad de 100 m/seg y con un ángulo de 30° , lo que conduce a los valores iniciales siguientes:

$$u(0)=100\cos(\pi/6), v(0)=100\sin(\pi/6), x(0)=0, y(0)=0.$$

Los comandos para realizar la integración son los siguientes (se podrían agrupar en un fichero llamado **tiroparMain.m**):

```
t0=0;
tf=9;
y0=[100*cos(pi/6) 100*sin(pi/6) 0 0]';
[t,Y]=ode23('tiropar',[t0,tf],y0);
plot(t,Y(:,4)) # dibujo de la altura en función del tiempo
```

podemos encontrar varios tipos de funciones que resuelven ecuaciones diferenciales:

- `ode23` : resuelve ecuaciones diferenciales de 2^{o} y 3er orden
- `ode45` : resuelve ecuaciones diferenciales de 4^{o} y 5^{o} orden
- `ode78` : resuelve ecuaciones diferenciales usando fórmulas de 7^{o} orden

GNU Free Documentation License

GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially.

Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall

subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any

Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have

received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.