Capítulo 1

MANUAL DE OCTAVE

1.1. Introducción

El propósito de este manual es ofrecer una breve introducción al programa de cálculo **OCTAVE**. Este programa es un entorno de computación para llevar a cabo cálculos numéricos y visualización de datos de una forma potente y a la vez sencilla.

OCTAVE es un sistema interactivo cuyos elementos principales y básicos son las matrices, que facilita la resolución numérica de problemas.

La principal característica de este entorno de programación es su capacidad de expansión mediante paquetes de funciones que se denominan *toolboxes* y que son colecciones de funciones escritas en lenguaje **OCTAVE** que extienden la funcionalidad del programa para resolver tipos de problemas particulares. Las áreas para las que están disponibles esas toolboxes incluyen procesado de señales, diseño de sistemas de control, simulación de sistemas dinámicos, redes neuronales, estadística, optimización y otras.

1.1.1. Editor de la línea de comandos

Cuando se ejecuta el programa **OCTAVE**, se crea, entre otras, la "*Ventana de Comandos*" a través de la cual el usuario se comunica con el intérprete de **OCTAVE**. El intérprete **OCTAVE** se presenta mediante el indicador (*prompt*) "**>>**" indicando que el programa está preparado para aceptar instrucciones. A continuación podemos escribir cualquier comando válido para **OCTAVE,** seguido de una pulsación de la tecla "*Retorno de carro*"(*return*), para que se realice la acción o cálculo correspondiente. Por ejemplo, al introducir dentro de la ventana de comandos la siguiente operación

>> 3 + 4

y después de pulsar sobre la tecla de retorno de carro, el sistema dará la respuesta como

$$ans = 7$$

El sistema trabaja con variables y todos los resultados son almacenados en ellas, en este caso y como no se le ha indicado ninguna, el programa crea una variable automática, *ans,* que es la que contiene la respuesta. Podemos comprobarlo escribiendo en la línea de comandos lo siguiente

>> ans

que nos dará la misma respuesta que antes.

Para simplificar el proceso de entrada de comandos en el intérprete de **OCTAVE**, se pueden utilizar un conjunto de teclas, incluyendo las teclas de cursor, para editar o recuperar comandos introducidos previamente.

Se describe a continuación el conjunto de las teclas utilizadas por el programa OCTAVE para recuperar y modificar instrucciones que ya han sido introducidas en la línea de comandos. La tabla siguiente describe ese conjunto de teclas

Tecla	Función	Tecla Alternativa
↑	Recupera línea anterior	Ctrl + P
↓	Recupera línea siguiente	Ctrl + N
→	Mover hacia la derecha un carácter	Ctrl + F
←	Mover hacia la izquierda un carácter	Ctrl + B
Ctrl + →	Mover hacia la derecha una palabra	Ctrl + R
Ctrl + ←	Mover hacia la izquierda una palabra	Ctrl + L
Home/Inicio	Mover al principio de la línea de comandos	Ctrl + A
End/Fin	Mover al final de la línea de comandos	Ctrl + E
Esc	Borrar la línea de comandos	
Del/Supr	Borrar carácter en el cursor	Ctrl + D
Backspace	Borrar carácter a la izquierda del cursor	
Ctrl + K	Borrar hasta el final de línea	

Los comandos que se introducen durante cada sesión OCTAVE se almacenan en memoria, de manera que podemos recuperar cada comando utilizando las teclas anteriores, podemos ver este historial dentro de la ventana correspondiente del entorno gráfico con el nombre de "*Historial de Comandos*". También podemos recuperar un comando de forma sencilla solamente utilizando un conjunto de caracteres del mismo. Por ejemplo, si escribimos la secuencia de letras "**plo**" y utilizamos la tecla de cursor ↑ , sólo se irán recuperando los últimos comandos utilizados y que comiencen con esas letras.

1.1.2. Variables en OCTAVE

Como se ha indicado, **OCTAVE** trabaja con variables y para crear una, lo único que hay que hacer es asignarle un valor, esta asignación se realiza mediante el operador "=", por ejemplo la secuencia de instrucciones

```
>> radio = 2;
>> volumen = (4/3)*pi*r^3;
```

calculará el volumen de una esfera de radio **2**. Con estas instrucciones, en primer lugar **OCTAVE** crea un variable de nombre **radio** y le asigna el valor **2**. A continuación realiza las operaciones necesarias para calcular el volumen y asigna el resultado de ese cálculo a la variable **volumen**, la variable **pi,** que en realidad es una constante predefinida en **OCTAVE** que contiene al número π. El signo "**;**" al final de la línea indica que **OCTAVE** no presente el resultado de las operaciones realizadas, simplemente que las haga y las almacene en la memoria, luego si queremos conocer el valor de la variable, bastará con escribirla en la línea de comandos. Podemos comprobar qué ocurre al poner o no el punto y coma, para ello prueba las instrucciones anteriores sin incluir el punto y coma del final de cada línea y comprueba el resultado.

Es posible usar varios comandos en una misma línea separándolos, bien mediante el signo de punto y coma **";"**, bien si queremos conocer los resultados de cada operación entonces podemos separarlos mediante una coma **","**. Escribe las siguientes secuencias en la línea de comandos y comprueba sus resultados:

```
>> a=2, b=3; c=a+b;
```

```
>> a=2, b=3, c=a+b
```

A veces la expresión utilizada es muy larga y se extiende más allá de una línea de pantalla, en este caso podemos seguir en la línea siguiente utilizando los signos **"..."**, más un retorno de carro, por ejemplo, en la siguiente instrucción

```
>> a =3; b=2; c=a^3+a^2+...
a+b;
```

la primera línea, continúa en la segunda para completar la definición de **c**. Sin los tres puntos, la expresión sería errónea.

A diferencia de otros lenguajes de programación, como C, en **OCTAVE** no es necesario definir los nombres de las variables, ni tampoco el tipo de datos que contienen, para usarlas basta con asignarles un valor y cualquier variable puede contener cualquier tipo de dato.

Tampoco es necesario indicar el tamaño de los vectores o matrices que se van a utilizar, el límite en el tamaño está dado por la capacidad del ordenador.

Las variables que se van creando se pueden ver en la ventana de **OCTAVE** *.Espacio de trabajo"*, y se puede editar su valor a través del *Editor de Variables,* al que se puede acceder a través de la pestaña correspondiente o haciendo doble clic sobre el nombre de la variable.

En principio puede utilizarse casi cualquier nombre, siempre que sea compatible con **OCTAVE**. Sin embargo, debemos tener presentes dos situaciones confllicitvas. La primera es que **OCTAVE** no acepta nombres que contengan algún carácter no admitido (por ejemplo + o *, por ser operadores aritméticos), o una palabra clave (como **if, for, end,** etc.) tampoco acepta variables que empiecen por un número (**1radio** no sería válida, pero **radio1** sí). La segunda es que se acepta el nombre pero éste cambio anula el significado original de un nombre reservado, por ejemplo, la constante **pi** que hemos utilizado antes. Este tipo de conflicto puede ocurrir con los siguientes tipos de nombres:

1. Nombres de ciertos valores.
2. Nombres de funciones (subrutinas).
3. Nombres de comandos.

Un método para determinar la compatibilidad del nombre de una variable es probarlo en la línea de comandos. Si el nombre no es aceptado, por ejemplo el comando **end=4** que intenta asignar el valor 4 a la instrucción **end**, dará un error y será ignorado. Un ejemplo del segundo conflicto es el siguiente: si se utilizan **sin** y **cos** como nombres de variables, no se emitirá ningún error, pero las funciones trigonométricas seno y coseno no podrán utilizarse como funciones hasta que no se eliminen las variables creadas mediante el comando **clear** o se abandone **OCTAVE**

>> **sin(pi/2)**	Se calcula el valor de la función seno en $\frac{\pi}{2}$ radianes.
>> **sin=2**	Se crea la variable **sin** y se le asigna el valor **2.**
>> **sin(pi/2)**	Error, ya que se está intentando acceder a la componente **pi** de la variable **sin,** que no existe.
>> **clear sin**	Se borra la variable **sin** y la función **sin** recupera su significado.
>> **sin(pi/2)**	Podemos volver a utilizar la función **sin**.

Algunos de los nombres de las variables predefinidas en **OCTAVE** están dados en la siguiente tabla

eps	Tolerancia o epsilón de máquina.
ans	Resultado de la última operación.
pi	$\pi = 3{,}141592653589793$
i,j	Unidad imaginaria, **i=j=**$\sqrt{-1}$.
inf	Infinito. Ejemplo 1/0
NaN	No es un número (Not A Number). Indeterminación, por ejemplo 0/0.
date	Fecha actual del sistema.
nargin	Número de argumentos de entrada de una función.
nargout	Número de argumentos de salida de una función.

Es importante notar que **OCTAVE** distingue entre mayúsculas y minúsculas; por tanto, las variables **radio, Radio** y **RADIO** son todas diferentes.

Cuando una variable no es necesaria podemos eliminarla del "*Espacio de Trabajo"*del sistema mediante la instrucción **clear**. Podemos eliminar una sola variable

clear radio

o varias a la vez

clear radio volumen

e incluso todas de golpe

clear all

Además de usando el nombre de la variable, podemos comprobar el contenido de una variable mediante el comando *disp*, que presenta un número, vector, matriz o cadena de caracteres en la ventana de comandos sin tener que especificar un nombre de variable; por tanto puede utilizarse para presentar mensajes o datos en pantalla

disp(nombre)	Presenta valor de variable **nombre**
disp 'frase'	Presenta la cadena de caracteres **frase**

Se utiliza **format** para controlar la forma de presentación de datos en pantalla. Solamente se afecta la forma y no el fondo, es decir, para las operaciones **OCTAVE** opera internamente con doble precisión. Se utiliza la expresión

format *tipo*

donde *tipo* tiene las siguientes opciones:

short		Formato de punto flotante con 4-5 dígitos. Valor por defecto.	3.1416
short e	**shorte**	Idem anterior con expresión exponencial	3.1416e+00
long		Formato de punto flotante con 15 dígitos	3.141592653589793
long e	**long**	Idem anterior con expresión exponencial	3.141592653589793+00
bank		Formato de punto flotante con 2 dígitos	3.14
hex		Formato hexadecimal	400921fb54442d18
rat		Aproximación a número racional	355/113fo
+		Signo (+ para positivos, - para negativos y blanco para el 0)	+
loose		Pone una línea en blanco antes y después del resultado	
compact		Salida de datos en forma compacta, sin líneas.	

se ha incluido en la tabla el aspecto que presenta la variable **pi** que está definida en el sistema para cada uno de los formatos.

1.1.3. Operadores en OCTAVE

Operadores Aritméticos

Los **operadores aritméticos** que utiliza **OCTAVE** son los usuales y se incluyen en la siguiente tabla

+	Suma: **3+4**
–	Resta: **4-3**
*	Multiplicación: **4*3**.
/	División: **8/2**.
\	División inversa. Esta operación produce el recíproco de /. Es decir **a\b=b/a: 8\2=2/8**
^	Potencia: **2^3** o **2**3**
**	

Operadores Relacionales y Lógicos

Para **OCTAVE** una expresión es VERDAD (TRUE) si tiene valor distinto de 0, mientras que es FALSA (FALSE) si es 0.

OCTAVE utiliza como operadores de comparación de dos variables los dados en la siguiente tabla

>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
= =	Igual que
~=	No igual que

El resultado de la comparación es **0** o **1**, que determina la veracidad o falsedad de la relación. Por ejemplo

>> **(3<4)**

devolverá un valor **1**, mientras que

```
>> (2>4)
```

devolverá un valor **0**.

También se pueden usar operadores lógicos, para unir dos o más proposiciones lógicas

&	Y (AND) lógico
|	O (OR) lógico
~	No (NOT) lógico

Por ejemplo

```
>> (3<4) & (2>5)
```

devolverá el valor **0**, ya que la proposición **(2>5)** es falsa y estamos utilizando el operador **AND**, mientras que

```
>> (3<4) & (2>5)
```

devolverá el valor **1**, ya que la proposición **(3<4)** es verdadera y estaríamos utilizando el operador **OR.**

1.2. Matrices

OCTAVE trabaja ensencialmente con matrices rectangulares de dimensión $m \times n$, con elementos reales o complejos, aunque se pueden definir variables de cualquier número de componentes.

Un escalar es una matriz 1×1, mientras que los vectores fila son matrices de dimensión $1 \times n$ y los vectores columna son matrices de dimensión $m \times 1$.

Podemos crear matrices de varias formas diferentes

1. Mediante una lista explícita de elementos usando corchetes []

   ```
   >> A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
   ```

   Se utiliza el signo "**,**" para separar elementos dentro de una misma fila, aunque también se puede utilizar un espacio y para separar filas se utiliza el signo "**;**". Otra forma de generar una matriz de forma explícita es utilizar el retorno de carro en vez del signo "**;**" para comenzar la definición de la siguiente fila

   ```
   >> A = [1, 2, 3
   4, 5, 6
   7, 8, 9 ]
   ```

   Hay que tener en cuenta que **todas las filas tienen que tener el mismo número de elementos**. Como antes, la definición de la matriz termina cerrando corchetes en la última de las filas.

   Un vector fila es una matriz $1 \times n$ y por tanto estaría definido como

   ```
   >> x1 = [1, 2, 3, 4]
   ```

   mientras que un vector columna, que es una matriz $m \times n$, estaría definido como

   ```
   >> x2 = [1; 2; 3; 4]
   ```

   Se utilizará el símbolo "**'** "(apóstrofo) para obtener la matriz transpuesta correspondiente

   ```
   >> A'

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

2. Para construir un vector de una forma rápida y sencilla podemos utilizar los dos puntos, ":". Por ejemplo la sentencia

$$>> \mathbf{x = 1:5}$$

crea un vector fila que contiene los números del 1 al 5, con incrementos unitarios, produciendo el vector

$$x = \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix}$$

Por defecto el incremento es 1, pero podemos utilizar incrementos diferentes. La instrucción

$$>> \mathbf{y = 0:pi/4:pi}$$

produce el siguiente vector

$$y = \begin{matrix} 0.0000 & 0.7854 & 1.5708 & 2.3562 & 3.1416 \end{matrix}$$

También es posible utilizar incrementos negativos, por ejemplo, la instrucción

$$>> \mathbf{z = 6:-1:1}$$

define la variable **z** como

$$z = \begin{matrix} 6 & 5 & 4 & 3 & 2 & 1 \end{matrix}$$

3. También podemos crear una matriz utilizando un fichero **OCTAVE** (extensión *.m*) que contenga la definición de la matriz. Es decir, podemos crear un fichero que contenga las instrucciones que escribiríamos como si se estuviera haciendo desde la línea de comandos. Por ejemplo, utilizamos el *Editor* de texto de **OCTAVE,** que podemos encontrar en las pestañas inferiores, y escribimos

$$\mathbf{A = [1, 2, 3; 4, 5, 6; 7, 8, 9]}$$

A continuación grabamos el fichero con cualquier nombre, por ejemplo **gen_a.m** (siempre con extensión **.m).** Para construir la matriz **A**, es suficiente con escribir **gen_a** (sin la extensión) en la línea de comandos.

4. También podemos utilizar funciones y comandos para leer los datos desde un fichero que haya sido creado por otro programa, en particular **OCTAVE** puede leer ficheros escritos en ASCII, ficheros binarios creados por C o Fortran y los propios ficheros creados por **OCTAVE**. Las funciones **fread** y **load**, son ejemplos de dichas funciones.

5. Otra forma de generar vectores es utilizar funciones específicas para ello

linspace(m,n,[p])	Genera [**p**] valores equiespaciados linealmente, entre **m** y **n**. Si no se indica el valor [**p**], se generan 100.
logspace(m,n,[p])	Genera [**p**] valores equiespaciados logaritmicamente, entre **m** y **n**. Si no se indica el valor [**p**], se generan 50.
zeros(n)	Matriz nula de dimensión **n** x **n**
zeros(m,n)	Matriz nula de dimensión **m** x **n**
ones(n)	Matriz de unos de dimensión **n** x **n**
ones(m,n)	Matriz de unos de dimensión **m** x **n**
diag(v)	Matriz diagonal con los elementos de **v** en la diagonal
rand(n)	Matriz de elementos uniformemente distribuidos de dimensión **n** x **n**
rand(m,n)	Matriz de elementos uniformemente distribuidos de dimensión **m** x **n**
randn(n)	Matriz de elementos normalmente distribuidos de dimensión **n** x **n**
randn(m,n)	Matriz de elementos normalmente distribuidos de dimensión **m** x **n**
eye(n)	Matriz identidad **n** x **n**
eye(m,n)	Matriz identidad **m** x **n**

Para crear una variable (vector o matriz) de números complejos podemos utilizar los siguientes métodos

$$A = [\ ] + i*[\ ]$$

o

$$A = [a+bi,\ c+di,...]$$

por ejemplo

```
>> A1 = [1+i, 3-i; 2*i, 3-i]
```

creará una matriz compleja de dimensión *2x2*. La misma matriz se puede crear de la siguiente forma

```
>> Real_A = [1,3; 0,3];
>> Imag_A=[1,-1; 2,-1];
>> A2 = Real_A + i*Imag_A;
```

Los elementos de una matriz pueden ser cualquier expresión válida en **OCTAVE**, por ejemplo

```
>> x = [-1.3, sqrt(5), (1+2+3)*4/5]
```

es una definición válida para el vector **x**

```
x =
   -1.3000    2.2361    4.8000
```

1.2.1. Subíndices

Los elementos individuales de una matriz puede obtenerse haciendo referencia entre paréntesis a sus subíndices, de este modo si definimos la matriz

```
>> A = [1, 2, 3
4, 5, 6
7, 8, 9]
```

podemos acceder al elemento de la fila 3 y columna 2 como

>> A(3,2)

que nos dará por respuesta

ans = 8

Para vectores sólo es necesario especificar la posición dentro del vector, por ejemplo, si hemos definido el vector fila *x1* como

>> x1 = [1, -1, 2, -2, 3, -3]

la instrucción

>> x1(2)

nos daría como respuesta

ans = -1

que corresponde al segundo elemento del vector. También se puede hacer con vectores columna, por ejemplo, si definimos el vector columna como

>> x2 = [1; -1; 2; -2; 3; -3]

la instrucción

>> x2(3)

nos daría como respuesta

ans = 2

Los elementos de una matriz o un vector también pueden usarse como variables y podemos emplearlos en el lado izquierdo de una asignación, de este modo podemos poner una expresión como

>> A(1,1) = 24

que nos daría como respuesta

A=

24	*2*	*3*
4	*5*	*6*
7	*8*	*9*

donde vemos que sólo se ha cambiado el elemento de la primera fila y primera columna, dejando el resto de elementos fijos, y ahora la sentencia

>> A(3,3) = A(1,3) + A(3,1)

produce

A=

24	*2*	*3*
4	*5*	*6*
7	*8*	*10*

Podemos referirnos a una submatriz dentro de una matriz utilizando el rango de filas y columnas en la forma *A(i1:i2,j1:j2)* que daría los elementos que abarcan desde la fila *i1* hasta la fila *i2* y desde la columna *j1* hasta la columna *j2.* Por ejemplo, si definimos la matriz *A,* de dimensión *4x5* como

>> A = [1, 2, 3, 4, 5
6, 7, 8, 9, 10
11, 12, 13, 14, 15
16, 17, 18, 19, 20]

la instrucción

>> A(2:4, 2:3)

nos proporciona la submatriz de A formada por los elementos que están entre las filas 2 y 4, y entre las columnas 2 y 3:

```
>> ans
  7     8
  12    13
  17    18
```

Mientras que la instrucción

```
>> A(1:3, 3)
```

especifica la submatriz 3×1, o vector columna formado por los 3 primeros elementos de la columna 3

```
>> ans
  3
  8
  13
```

Para extraer todos los elementos de una fila o conjunto de filas se utiliza el símbolo "**:**"sin indicar el rango, por ejemplo, la instrucción

```
>> A(:,1)
```

nos daría como resultado la primera columna

```
>> ans
  1
  6
  11
  16
```

y la instrucción

```
>> A(1, :)
```

nos daría la primer fila

```
>> ans
  1    2    3    4    5
```

Por ejemplo

```
>> A(:,1:3)
```

serían las 3 primeras columnas

```
>> ans
  1     2     3
  6     7     8
  11    12    13
  16    17    18
```

mientras que

```
>> A(1:3, :)
```

son las 3 primeras filas

```
>> ans
  1     2     3     4     5
  6     7     8     9     10
  11    12    13    14    15
```

Como se ha indicado al principio no es necesario definir ni la dimensión ni el tipo de datos que contiene una matriz. Si definimos el vector x como

```
>> x = [-1.3, sqrt(5), (1+2+3)*4/5]
x =
  -1.3000    2.2361    4.8000
```

que tiene 3 componentes, podemos introducir un nuevo dato en la posición 5 del vector simplemente escribiendo su valor, por ejemplo si escribimos

```
>> x(5) = abs(x(1))
```

se obtiene

```
x =
  -1.3000    2.2361    4.8000    0    1.3000
```

Vemos que la matriz se dimensiona automáticamente para poder contener todos los elementos, notar que como el vector original contenía 3 elementos y hemos incorporado el elemento con índice 5, el resto de los elementos no definidos (en este caso sólo el elemento 4) están definidos como *0*.

Usando esta misma propiedad, podemos utilizar matrices ya definidas como elementos para construir otras matrices, por ejemplo si hemos definido la matriz **A** y el vector **r** como sigue

```
>> r = [10, 11, 12];
>> A = [1, 2, 3
4, 5, 6
7, 8, 9];
```

la instrucción

```
>> B = [A; r]
```

definirá la variable **B** como la matriz de 4 filas y 3 columnas

```
B =
   1    2    3
   4    5    6
   7    8    9
  10   11   12
```

mientras que la instrucción

```
C = [A; r' ]
```

definirá la matriz **C** de dimensión 3 filas y 4 columnas siguiente

```
C =
   1    2    3   10
   4    5    6   11
   7    8    9   12
```

Si se quiere construir una tabla de valores se pueden trasponer los vectores obtenidos utilizando la notación de los dos puntos, y unirlos en una sola matriz. Por ejemplo

```
>> x = (0.0:0.2:3.0)';
>> y = exp(-x).*sin(x);
>> [x, y]
```

produce una tabla de valores en la que la primera columna se incluyen los valores de **x** y en la segunda los valores obtenidos al emplear la función $e^{-x}\sin(x)$.

En **OCTAVE** un vector puede actuar como subíndice, de hecho, como ya se ha indicado, la instrucción *x=1:10* define un vector. Si x y v son vectores, siendo las componentes de v números naturales no nulos, entonces la expresión *x(v)* representa al vector *[x(v(1)), x(v(2)),...,x(v(n))]*, siendo n la longitud del vector v, por ejemplo, si definimos el vector x como

```
>> x = 0:0.1:10;
```

y

```
>> v =  [1,3,5,7];
```

la expresión

```
>> x(v)
```

da como resultado un vector con las componentes 1, 3, 5 y 7 del vector x

```
ans =
   0    0.2000    0.4000    0.6000
```

Para matrices, si v y w son vectores de valores naturales no nulos, entonces $A(v,w)$ es la matriz que se obtiene al tomar los elementos de A, con subíndices para las filas tomados de v, y con subíndices para las columnas tomados a partir de w, por ejemplo, si definimos A, v y w como

```
>> A = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12];
>> v = [1,2];
>> w = [3,1];
```

La instrucción

```
>> A(v,w)
```

nos devuelve la matriz

```
ans =
   3   1
   7   5
```

que como vemos, contiene los elementos de A de las filas 1 y 2 que están en las columnas 3 y 1, y en ese orden. La instrucción

```
>> A(:, 4:-1:1)
```

invierte el orden en las columnas de A

```
ans =
   4    3    2    1
   8    7    6    5
   12   11   10   9
```

Finalmente la instrucción

```
>> A(:)
```

indica todos los elementos de A tal y como **OCTAVE** los almacena en memoria, considerados como un vector columna

```
ans =
   1
   5
   9
   2
   6
   10
   3
   7
   11
   4
   8
   12
```

En la parte izquierda de una sentencia de asignamiento, se puede utilizar $A(:)$, para redefinir la matriz, para ello la variable A debe existir en el espacio de trabajo y en ese caso las dimensiones de A se mantienen, pero con otros elementos. Por ejemplo si definimos A como la matriz

```
>> A = [1, 2; 3, 4; 5, 6];
```

La instrucción

```
>> A(:) = 11:16
```

redefine a la matriz A como

$$A = \begin{matrix} 11 & 14 \\ 12 & 15 \\ 13 & 16 \end{matrix}$$

También podemos utilizar los subíndices a ambos lados de una operación de asignación, por ejemplo si definimos A *como*

$$>> \mathbf{A = rand(5,5);}$$

y definimos la matriz B como

$$>> \mathbf{B = ones(5,5);}$$

entonces la instrucción

$$>> \mathbf{A(:, [1,3,5]) = B(:,1:3)}$$

reemplaza las columnas 1, 3 y 5 de la matriz A con las tres primeras columnas de la matriz B.

Vectores 0-1 como subíndices

Podemos utilizar un vector de 0 y 1, para referirnos a submatrices dentro de una matriz. Si A es una matriz $m \times n$ y L es un vector de longitud m que contiene ceros y unos, la sentencia *A(L,:)* especifica las filas de A donde los elementos de L son distintos de 0. Por ejemplo, si definimos x como

$$>> \mathbf{x = rand(1,100);}$$

especifica un vector cuyas componentes se han extraído de una distribución uniforme en el intervalo (0,1), mientras que la instrucción

$$>> \mathbf{L = x < 0.5;}$$

nos devuelve un vector de ceros y unos que nos indica si la correspondiente componente de x es menor (1) o mayor (0) que 0.5. Finalmente, la instrucción

$$>> \mathbf{y=x(L);}$$

crea un vector y cuyas componentes son las de x que cumplen la condición de ser menores que 0.5.

Matrices vacías

Una matriz vacía es una variable que existe en el espacio de trabajo, pero que no contiene ningún dato. La sentencia

$$>> \mathbf{x = []}$$

define a **x** cómo una matriz de dimensión 0×0. Una matriz vacía existe y por tanto no se producirá ningún error al utilizarla. Está sentencia es diferente de la de borrar una matriz del espacio de trabajo, la orden

$$>> \mathbf{clear\ x}$$

eliminará la variable **x** del espacio de trabajo y cualquier orden o sentencia posterior que la utilice dará lugar a un error.

Las matrices vacías se utilizan para eliminar filas o columnas de una matriz, por ejemplo, la instrucción

$$>> \mathbf{A(:, [2,4]) = []}$$

eliminará las columnas 2 y 4 de la matriz A.

Matrices especiales

OCTAVE contiene un conjunto de funciones para generar diferentes tipos de matrices que aparecen en Álgebra lineal. Algunas de ellas son las siguientes

diag	Generar una matriz diagonal
meshgrid	Para funciones de 2 variables

También hay otro conjunto de funciones para crear matrices más específicas:

compan	Matriz compañera de un polinomio
gallery	Matrices de comprobación
hadamard	Matriz de Hadamard
hankel	Matriz de Hankel
hilb	Matriz de Hilbert
invhilb	Matriz Inversa de Hilbert
kron	Producto tensorial de Kronecker
magic	Cuadrado mágico
pascal	Triángulo de Pascal
toeplitz	Matriz de Toeplitz
vander	Matriz de Vandermonde

1.2.2. Operaciones con matrices

1. **Matriz Traspuesta**

   Si **A** es una variable definida como una matriz $m \times n$, podemos calcular la traspuesta de dicha matriz, simplemente utilizando el operador "' "(apóstrofo) o la función **transpose**

```
>> A = [1,2,3; 4,5,6; 7,8,9];
>> A'
 ans =
   1   4   7
   2   5   8
   3   6   9
```

   y para un vector

```
>> x = [-1,0,2];
>> x'
```

   produce

```
 ans =
  -1
   0
   2
```

   Si **A** es una matriz compleja, entonces **A'** es la transpuesta conjugada de **A**, para construir la transpuesta sin conjugar de una matriz compleja utilizar o bien la operación elemento a elemento, insertando un punto antes del apóstrofo, **A.',** o bien la función **conj(A')**

2. **Suma y diferencia de matrices**

   Los operadores de suma y diferencia de matrices son, respectivamente, + y −. Esta operación está definida siempre que las matrices que se utilicen tengan las mismas dimensiones, en ese caso el resultado de la suma de dos matrices es el usual en matemáticas, es decir, la suma elemento a elemento:

$$\text{Si } A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \text{ y } B = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

entonces

$$C = A + B$$

es aceptable y da como resultado

$$C = \begin{matrix} 2 & 6 & 10 \\ 6 & 10 & 14 \\ 10 & 14 & 18 \end{matrix}$$

También están definidas estas operaciones si alguno de los sumandos es un escalar (matriz 1×1). Por ejemplo si

$$x = [1, 2, 3]$$

La operación

$$y = x - 1$$

dará como resultado

$$y = \begin{matrix} 0 & 1 & 2 \end{matrix}$$

3. **Producto de Matrices**

El operador $*$ es el utilizado para la multiplicación de matrices. Dicha operaciones es la definida en el sentido usual de producto de matrices, y está definida siempre que el número de columnas de la primera matriz sea el mismo que el número de filas de la segunda matriz.

Una operación importante es el producto interior de dos vectores, si x e y son dos vectores de la misma dimensión

$$x = (-1, 0, 2)$$

$$y = (-2, -1, 1)$$

definimos el producto escalar o interior como

$$x * y'$$

definido según la definición matemática

$$x * y = \sum_{j=1}^{n} x_j y_j$$

Y que para los vectores x e y anteriores tienen un valor de -4.

Por supuesto la expresión

$$y * x'$$

dará el mismo resultado.

Existen dos productos exteriores, uno traspuesto del otro definidos por

$$x' * y = \begin{bmatrix} -1 \\ 0 \\ 2 \end{bmatrix} * \begin{bmatrix} -2 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -1 \\ 0 & 0 & 0 \\ -4 & -2 & 2 \end{bmatrix}$$

$$y' * x = \begin{bmatrix} -2 \\ -1 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 0 & -4 \\ 1 & 0 & 2 \\ -1 & 0 & 2 \end{bmatrix}$$

Por supuesto el producto también está definido multiplicamos un escalar por cualquier matriz.

4. **División de matrices**

   Existen dos símbolos de divisón en OCTAVE: \ y /. Si A es una matriz cuadrada no singular, entonces

$$\begin{aligned} A\backslash B &\Rightarrow inv(A) * B \\ B/A &\Rightarrow B * inv(A) \end{aligned}$$

   El resultado se obtiene sin el cálculo de la inversa. En general

$$\begin{aligned} X &= A\backslash B \qquad \text{es una solución de } A * X = B \\ X &= B/A \qquad \text{es una solución de } X * A = B \end{aligned}$$

   $A\backslash B$ está definido si B tiene las mismas filas que A.

   Podemos expresar las operaciones con / en términos de \ mediante

$$B/A = (A'\backslash B')'$$

5. **Potencias de Matrices**

   La expresión

$$A\hat{}p$$

   eleva A a la potencia p-ésima, y está definida para A cuadrada y p escalar. También se puede utilizar el operador $**$

$$\text{A**p} = \text{A}\hat{}\text{p}$$

6. **Funciones trascendentales y elementales sobre matrices**

$expm(A)$	Exponencial Matricial	$\sum_{k=0}^{\infty} \frac{A^k}{k!}$
$logm(A)$	Logaritmo Matricial	
$sqrtm(A)$	Raíz cuadrada Matricial	
$poly(A)$	Polinomio característico	
$det(A)$	Determinante	
$trace(A)$	Traza de la Matriz	
$kron(A, b)$	Producto tensorial de Kronecker	
	Todos los posibles productos que hay entre elementos de A y B	

1.2.3. Operaciones elemento a elemento

Si queremos realizar operaciones de producto, división o exponenciación entre matrices, pero realizando las operaciones elemento a elemento, se tiene que anteponer un punto "." delante de cada operador.

1. **Suma y resta**

   Para la suma elemento a elemento se utiliza el mismo operador que para la suma de matrices, puesto que la suma de matrices se define de ese modo.

$$\begin{aligned} &A + B \\ &A - B \end{aligned}$$

2. **Multiplicación y división**

   Para dos matrices, A y B, con la misma dimensión

$$A. * B \Rightarrow \text{ Multipliación elemento a elemento}$$

Por ejemplo para

$$x = [1, 2, 3]\,; \qquad y = [4, 5, 6]$$

entonces

$$z = x. * y$$

da como resultado

$$z =$$
$$4 \quad 10 \quad 18$$

Para la división elemento a elemento tenemos

$$A./B \Rightarrow \text{ División / elemento a elemento}$$

$$A.\backslash B \Rightarrow \text{ División \ elemento a elemento}$$

por ejemplo, utilizando los vectores anteriores

$$z = x.\backslash y$$

da como resultado

$$z =$$
$$4{,}0000 \quad 2{,}5000 \quad 2{,}0000$$

3. **Potenciación**

El operador ".^" representa potencias de los elementos de una matriz. Podemos utilizar dicho operador de la siguiente forma, se incluye un ejemplo para $x = [1, 2, 3]\,; \qquad y = [4, 5, 6]$

$x.\hat{}y$	Dos vectores o matrices de la misma dimensión. Cada elemento de x es elevado a cada elemento de y $x.\hat{}y = [1, 32, 729]$
$x.\hat{}a$	Un vector elevado a un escalar. Eleva cada elemento de x a ese escalar $x.\hat{}2 = [1, 4, 9]$
$a.\hat{}x$	Un escalar elevado a un vector. Eleva el escalar a cada elemento del vector $2.\hat{}[x, y] = [2, 4, 8, 16, 32, 64]$

4. **Operaciones relacionales**

Se pueden utilizar los operadores relacionales para comparar dos matrices de la misma dimensión. Se comparan las parejas de los correspondientes elementos y el resultado es una matriz de unos y ceros. También pueden relacionarse matrices con escalares, comparando cada elemento de la matriz con el escalar. En este caso el resultado es una matriz de ceros y unos con las mismas dimensiones que la matriz.

La función *find* es muy útil con los operadores relacionales, busca elementos distintos de 0 en una matriz de 0 y 1, y por tanto los elementos de la matriz que satisfacen una condición relacional en particular. Por ejemplo, si Y es un vector, $find(Y < 3{,}0)$, devuelve un vector que contiene los índices de los elementos de Y que son más pequeños que 3,0.

La secuencia

$$\begin{aligned} \text{i} \quad &= \text{find} \quad (Y > 3{,}0)\,; \\ Y(i) \quad &= \quad 10 * ones(i); \end{aligned}$$

reemplaza todos los elementos en Y que son más grandes que 3,0 por 10. Funciona incluso con matrices, puesto que estas se pueden considerar como un vector columna con un único subíndice.

5. **Operaciones lógicas**

   Se pueden utilizar los operadores lógicos entre dos matrices de la misma dimensión, aunque también se puede operar entre matrices y escalares.

   Las siguientes funciones son muy útiles con los operadores lógicos

$any(x)$	Develve 1 si hay algún valor en x que sea $\neq 0$
$all(x)$	Devuelve 1 si todos los valores de x son $\neq 0$
find	Devuelve los índices donde se cumpla la expresión lógica
$exist('var')$	Comprueba si existe la variable *var* en el espacio de trabajo
isnan	Comprueba si hay algún resultado NaN
isinf	Comprueba si hay algún resultado infinito (*inf*)
isfinite	Comprueba si hay algún resultado finito
isempty	Comprueba si la variable es una variable sin datos
isstr	Comprueba si la variable es una cadena de caracteres
isglobal	Comprueba si el argumento es una variable global
issparse	Comprueba si la matriz contiene muchos 0

   Las funciones *any* y *all* actuan sobre columnas, de manera que el resultado de aplicar estas funciones sobre una matriz $m \times n$, es una fila con n elementos que contiene el resultado para cada columna. De manera que si aplicamos la condición 2 veces obtendremos una condición escalar.

6. **Funciones matemáticas**

   Podemos utilizar un conjunto de funciones matemáticas sobre los elementos de una matriz. Las siguientes son un conjunto de las funciones que ya se encuentran definidas en OCTAVE

Funciones Trigonométricas

sin	Seno	*asin*	Arco Seno
cos	Coseno	*acos*	Arco Coseno
tan	Tangente	*atan*	Arco Tangente
sinh	Seno Hiperbólico	*asinh*	Argumento Seno Hiperbólico
cosh	Coseno Hiperbólico	*acosh*	Argumento Coseno Hiperbólico
tanh	Tangente Hiperbólica	*atanh*	Argumento Tangente Hiperbólica
atan2(y,x)	Arco Tangente de y/x		

Funciones Elementales

abs	Valor absoluto	*ceil*	Redondear hacia $+\infty$
angle	Árgumento de un número complejo	*sign*	Signo
sqrt	Raíz cuadrada	*rem*	Resto o módulo de una división entera
real	Parte real de un número complejo	*gcd*	Máximo común divisor
imag	Parte imaginaria de un número complejo	*lcm*	Mínimo común múltiplo
conj	Conjugado de un número complejo	*exp*	Exponencial base e
round	Redondear al entero más cercano	*log*	Logaritmo base e
fix	Redondear hacia 0	*log10*	Logaritmo en base 10
floor	Redondear hacia $-\infty$		

Funciones Especiales

bessel	Función de Bessel	*erf*	Función de error
beta	Funciones beta completa e incompleta	*erfinv*	Inversa de la función de error
gamma	Funciones gamma completa e incompleta	*ellipk*	Integral elíptica completa de 1^a y 2^a clase
rat	Aproximación racional	*ellipj*	Funciones elípticas Jacobianas

Más información sobre funciones matemáticas: *help elfun*
Más información sobre operadores en: *help ops*

1.2.4. Otras funciones sobre matrices

Las siguientes funciones actúan sobre una matriz para producir una rotación, un desplazamiento, extraer una parte, etc..

rot90	Rotación
fliplr	Desplazamiento matricial de izquierda a derecha
flipud	Desplazamiento matricial de arriba a abajo
diag	Extraer la diagonal de una matriz
tril	Parte triangular inferior de una matriz
triu	Parte triangular superior de una matriz
reshape	Reformar una matriz
size	Tamaño de la matriz
length	Longitud de una matriz

Para obtener más información de funciones sobre matrices consultar con: *help matfun* o *help elmat*

1.3. Programación: Instrucciones de control

OCTAVE tiene estructuras de control igual que la mayoría de los lenguajes de progrmación.

OCTAVE cuenta con dos instrucciones de control para crear bucles o ciclos de instrucciones: la instrucción *for-end* y la instrucción *while-end*, y con una instrucción condicional: la sentencia *if*

Para más información: *help lang*

1.3.1. Bucle *for-end*

Con este bucle repetimos una instrucción o grupo de instrucciones un número fijo de veces. La forma general de un bucle *for* es la siguiente

for v=**expresión**
instrucciones
end

Donde **expresión** es una matriz. Las columnas de esa matriz se asignan una por una a la variable v y despues se realizan las **instrucciones.** Una forma más clara de expresar el bucle anterior es la siguiente:

E=**expresión;**
$[m, n] = size(E);$
for $j = 1 : n$
$v = E(:, j);$
Instrucciones
end

En el caso de que **expresión** sea un vector. El contador avanza por cada una de las componentes del vector y el bucle se repite tantas veces como componentes tiene el vector. Podemos, pues, construir un bucle de la forma siguiente:

$v = [1, 4, 5, -6, 0]$
for $k = v$
Instrucciones
end

que es equivalente a

for $j = 1 : 5$
$k = v(j);$
Instrucciones
end

Normalmente, la **expresión** es de la forma $n : m$ o $n : p : m$; en el que la matriz es un vector, y sus columnas son simplemente escalares. En este caso, el bucle *for-do* es equivalente al de otros lenguajes.

Veamos un ejemplo

```
for r=1:5
                vol=(4/3)*pi*r^3;
                disp([r,vol])
end
```

En este ejemplo se calcula y presenta el volumen de 5 esferas de radios 1,2,3,4 y 5

Los bucles se pueden anidar, e indentar para una mejor lectura

```
for i = 1 : m
        for j = 1 : n
                A(i,j) = 1/(i+j-1);
        end
end
```

Un punto importante es que cada *for* debe llevar asociado un *end,* en *OCTAVE* se puede utilizar *endfor* (aunque en este caso no es compatible con MATLAB)

1.3.2. Bucle *while-end*

Con este bucle repetimos una instrucción o grupo de instrucciones un número indeterminado de veces, bajo el control de una condición lógica. La forma general de un bucle *while* es la siguiente

```
while expresión
                instrucciones
end
```

Donde las instrucciones se repiten mientras que **expresión** sea una condición lógica verdadera. Igual que el bucle *for-end* **expresión** es una matriz, y por tanto el conjunto de **instrucciones** se repite mientras que la matriz **expresión** tenga todos sus elementos distintos de 0. La matriz **expresión** casi siempre es una expresión relacional simple, por ello todos los elementos distintos de 0 es equivalente a TRUE (cierto). Cuando la matriz **expresión** no es un escalar, podemos reducirla a escalar utilizando las funciones *any* y *all.*

Como ejemplo,

```
r=0
while r<5
        r=r+1
        vol=(4/3)*pi*r^3;
        disp([r,vol])
end
```

que es el mismo ejemplo que para el bucle *for-end*, pero utilizando el bucle *while-end.* Como en la sentencia anterior, *porclaridad, en OCTAVE* se puede utilizar *endwhile* (aunque en este caso tampoco es compatible con MATLAB)

1.3.3. Sentencia *If*

La estructura de la sentencia *If* es la siguiente

```
If expresión
                instrucciones;
end
```

Si **expresión** tiene el valor de verdad (distinto de 0) se realizan el conjunto de **instrucciones,** en caso contrario, se continua sin realizar ninguna operación.

Esta sentencia puede modificarse de forma que en caso de no cumplirse la **expresión** se realice una instrucción alternativa, para ello utilizamos la estructura

```
If expresión
                operaciones1;
else
                operaciones2;
end
```

En este caso si **expresión** es Verdadera se realizan las instrucciones de **operaciones1;** en caso contrario, es decir, si **expresión** es Falsa, se realizan las instrucciones de **operaciones2.**

También podemos anidar varios *if*, mediante la instrucción *elseif,* según el siguiente esquema

```
If expresión1
                operaciones1;
elseif expresión2
                operaciones2;
else
                operaciones3;
end
```

Si **expresión1** contiene un valor distinto de 0 (de VERDAD) se procede con las instrucciones de **operaciones1;** si por el contrario la expresión es falsa (igual que 0) se comprueba si la **expresión2** es cierta, en caso afirmativo se procede con **operaciones2**, y en caso negativo con **operaciones3.**

La anidación de sentencias de control *if* puede hacerse de la siguiente manera

```
If expresión1
                operaciones1;
else             if expresión2
                                operaciones2;
                else
                                operaciones3;
                end
end
```

Por ejemplo

```
If r > 3          b = 1;
elseif r == 3     b = 2;
else              b = 0;
end
```

La sentencia *elseif* se puede repetir tantas veces como se desee.

La sentencia puede terminarse tanto con *end* como con *endif*..

1.4. Programación: ficheros-M

Aunque OCTAVE es usualmente utilizado en modo comando, es decir, introduciendo una única orden en la línea de comandos, también puede ejecutar una secuencia de comandos que se encuentre almacenada en un fichero.

Los ficheros que contienen instrucciones OCTAVE se denominan ficheros-M, porque tienen extensión .m.

Un fichero-M consiste en una secuencia normal de instrucciones OCTAVE, que puede incluir referencias a otros ficheros-M. Un fichero-M puede llamarse a si mismo de forma recursiva. Además del editor incorporado, para crear un fichero-M se puede utilizar un editor normal de textos ASCII.

Podemos utilizar dos tipos de ficheros-M: *scripts* (guiones) y *functions* (funciones). Los ficheros *script* son una secuencia de comandos que se ejecutan secuencialmente. Los ficheros *function* son funciones en el sentido propio, son nuevos comandos que admiten tanto parámetros de entrada como de salida.

1.4.1. Ficheros *Script*

Cuando se llama a un fichero *script,* OCTAVE ejecuta los comandos que se encuentran en el fichero. Las instrucciones dentro de un fichero de tipo *script* actuan globalmente sobre los datos que se encuentran en el espacio de trabajo.

Como ejemplo, supongamos un fichero llamado *fibno.m* que contiene los siguientes comandos OCTAVE

```
%------------------
% Fichero-M para calcular los números de Fibonacci
%------------------
f = [1, 1] ; i = 1
while i < 16
    f(i + 2) = f(i) + f(i + 1);
    i = i + 1;
end
plot( f )
```

El símbolo % indica que el resto de la línea es un comentario que se ignorará. Escribiendo *fibno* en la línea de comandos, se llama al fichero *fibno.m* y OCTAVE ejecuta cada línea del fichero para calcular los 16 primeros números de la serie de Fibonacci, y crea un gráfico. Después de terminar la ejecución completa del fichero, las variables f e i permanecen en el espacio de trabajo.

Cuando llamamos a OCTAVE, automáticamente se ejecuta un fichero *script* denominado *startup.m* .

1.4.2. Ficheros *Function*

Un fichero-M que contiene la palabra *function* al principio de la primera línea del fichero es un fichero de tipo *function.* Un fichero *function* se diferencia de un fichero *script* en que podemos pasar argumentos al fichero, y las variables que se definen y manejan dentro del fichero son locales a la función y no operan globalmente en el espacio de trabajo.

Por ejemplo. Si queremos construir la función *media*, para calcular la media de un conjunto de valores, editamos mediante un editor ASCII, el fichero *media.m* y escribimos

```
% ------------------------
% Fichero de tipo función
% ------------------------
function y = media(x)
% Promedio o valor medio
% Para vectores, media(x) devuelve el valor medio
% Para matrices, media(x) es un vector fila que
% contiene el valor medio de cada columna.

[m, n] = size(x);
if m == 1
    D = n;
else
    D = m;
end
y = sum(x)/D;
```

Este fichero define una nueva función que se llama *media.* La nueva función *media* se utiliza de la misma forma que las demás funciones en OCTAVE. Por ejemplo, si z es el vector de enteros desde 1 hasta 99

$$z = 1 : 99;$$

el valor medio se encuentra utilizando la función *media*

$$media(z)$$

que tiene por respuesta

$$ans = 50$$

Veamos algunos detalles del fichero *media.m*

- La primera línea declara el nombre de la función, los parámetros de entrada, y los parámetros de salida. Sin esa línea, el fichero sería un fichero *script* en vez de un fichero *function*
- El símbolo % indica que el resto de la línea es un comentario que se ignorará.
- Para obtener una descripción del contenido y uso de la función emplearemos la siguiente sección

```
## -*- texinfo -*-
## @deftypefn {} {@var{y} =} media (@var{input1}, @var{input2})
##
## @seealso{}
## @end deftypefn
```

- Las variables m, n e y son variables locales a la función *media* y desaparecen después de que la función *media* acabe.
- No es necesario poner los enteros desde el 1 hasta el 99 en una variable de nombre x. De hecho, hemos utilizado la función *media* con la variable z. El vector z contiene los enteros desde el 1 hasta el 99, y el contenido de este vector se copia en *media* donde se transforma en una variable local de nombre $\dot{x}$

Podemos crear una versión un poco más complicada de *media*, llamada *estat*, que calcula también la desviación estándar. Editamos el fichero *estat.m* y escribimos

```
function [media, desv] = estat(x)
% Promedio o valor medio y desviación estándar
% Para vectores, estat(x) devuelve el valor medio en media
% y la desviación estándar en desv
% Para matrices, estat(x) devuelve un vector en media que
% contiene el valor medio de cada columna. Y otro vector en
% desv con la desviación estándar de cada columna

[m, n] = size(x);
if m == 1
    D = n;
else
    D = m;
end
media = sum(x)/m;
desv = sqrt(sum(x.^2)/m - media.^2)
```

La función *estat.m* ilustra el hecho de que es posible devolver varios argumentos de salida. También podemos construir funciones que admitan multiples parámetros de entrada.

Un par de variables interesantes y útiles en la definición de funciones son *nargin* que devuelve el número de parámetros de entrada en la función y *nargout* que da el número de parámetros de salida de la función.

1.4.3. Comandos de programación y variables globales

Orden de búsqueda

Cuando introducimos un nombre en la línea de comandos, por ejemplo *masa,* el interprete de OCTAVE sigue los siguientes pasos para intentar identificar dicho nombre:

1. Busca *masa* entre las variables

2. Verifica si *masa* es una función predefinida en OCTAVE

3. Busca en el directorio actual el fichero *masa.m*

4. Busca en los directorios especificados por la ruta de búsqueda el fichero *masa.m*

Por tanto, OCTAVE usará, si existe, *masa* como una variable, antes de utilizarla como una función.

Comando *echo*

Normalmente, mientras que se ejecuta un fichero-M, los comandos del fichero aparecen en la pantalla. El comando *echo* cambia este comportamiento. Utilizando *echo on* todos los comandos que se ejecuten dentro de un fichero-M aparecerán en pantalla, mientras que *echo off*, desactiva esta opción. El comando *echo* por si sólo intercambia entre el estado *on* y *off*. Este comando es interesante para depurar errores en una función o para demostración

Comando *input*

La función *input* es utilizada para pedir información al usuario. Por ejemplo la instrucción

$$d = input('Distancia\ recorrida')$$

presenta el texto entre comillas simples, y espera a que el usuario introduzca información y la asigna a la variable que hay a la izquierda, en este caso, la variable d. La entrada introducida por el usuario puede ser cualquier expresión OCTAVE, que es evaluada, utilizando las variables que existen en el espacio de trabajo.

Podemos utilizar la función *input*, para introducir un texto o una expresión sin evaluar, utilizando

$$nombre = input('¿Cuál\ es\ tu\ nombre?','s')$$

en este caso la función espera a que el usuario introduzca una cadena de caracteres. La entrada no es evaluda; y lo caractres simplemente se almacenan como una variable alfanumérica

Comando *Keyboard*

Este comando es similar a *input* pero mucho más potente. Esta función invoca al teclado de la computadora como un *script.* Si se utiliza dentro de un fichero-M, es útil para depurar programas o modificar variables en tiempo de ejecución.

Comando *Pause*

La función *pause* espera una respuesta del usuario. Las opciones son las siguientes:

pause	El programa para hasta que se pulsa una tecla
pause(n)	El programa para durante n segundos
pause off	Indica que cualquier utilización posterior del comando *pause* no tenga efecto
pause on	Indica que cualquier utilización posterior del comando *pause* si tenga efecto

Variables globales

Normalmente, cada función en OCTAVE, definida mediante un fichero-M, tiene sus propias variables locales, que están separadas de las de otras funciones, y también de las variables del espacio de trabajo. Sin embargo, si varias funciones, y posiblemente el espacio de trabajo, declaran un nombre particular como *global*, entonces todas esas funciones comparten una única copia de esa variable. Cualquier asignación a esa variable, en cualquier función donde esté definida como *global*, está disponible al resto de funciones que la declaren como *global.*

Podemos declarar una variable como global utilizando el comando *global*

$$global\ ALPHA, BETA$$

declara las variables $ALFA$ y $BETA$ como variables globales. Para que una función puede utilizar estas variables también tienen que estar definidas como *global* dentro del fichero-M que define a la función.

1.4.4. Variables alfanuméricas

Las variables de texto en OCTAVE se introducen mediante comillas, que pueden ser simples o dobles. Por ejemplo

s = 'Hola'

o bien

s = "Hola"

El texto se almacena en un vector, un carácter por elemento. En este caso

size(s)
ans =
1 4

indica que s tiene cinco elementos. Los caracteres se almacenan como su valor ASCII. Podemos observar ese valor mediante la función *abs*

abs(s)
ans=
72 111 108 97

Se pueden utilizar corchetes para concatenar variables de texto en cadenas alfanuméricas más grandes

$s = [s, \text{'Mundo'}]$
$s =$
Hola Mundo

Comando *setstr*
La función *setstr* transforma enteros entre 0 y 255 en sus correspondientes símbolos ASCII

*setstr(*65 : 90*)*
ans=
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Comando *disp*
El comando *disp* muestra una matriz como texto, sin presentar el nombre de la matriz

disp(x)

Comando *int2str*
La función

$s = int2str(n)$

convierte el valor escalar entero n en una representación alfanumérica del mismo
Comando *num2str*
La función

$t = num2str(x)$

convierte el valor escalar x en una representación alfanumérica con 4 dígitos y si se requiere un exponente. Este comando es útil para poner texto en gráficos mediante los comandos *title, xlabel, ylabel* y *text.*

Se puede suministrar un argumento alternativo para indicar la precisión o el número de dígitos a guardar, en este caso

$t = num2str(x, prec)$

convierte el valor escalar x en una representación alfanumérica con *prec* dígitos y si se requiere un exponente. lar number X into a string

Para más información acerca de funciones alfanuméricas utilizar: *help strfun*

La función *eval*

La función *eval* trabaja con variables de texto. El empleo de *eval(t)* evalúa el texto contenido en la variable t. Si **expresión** es una cadena de texto que contiene una instrucción o expresión en OCTAVE, entonces

$t =$ **'expresión';**

almacena el texto en la variable t. Escribiendo t en la línea de comandos se presenta el texto que la variable contiene, mientras que *eval(t)* interpreta el texto escrito, bien como una instrucción o como factor en una expresión. Por ejemplo:

```
t = '1/(i + j - 1)';
for i = 1 : n
      for j = 1 : n
            a (i, j) =eval(t);
      end
end
```

genera la matriz de Hilbert de orden n

Podemos utilizar *eval* junto con *input* para elegir una de entre varias tareas definidas por ficheros-M. En este ejemplo, los ficheros tienen nombres como *circulo.m, recta.m, elipse.m, etc..*

```
dibujo = ['circulo','recta','elipse','ovalo']hel
k = input('Elige gráfico: ');
eval(dibujo(k,:))
```

Por último un ejemplo para mostrar como *eval* puede utilizar el comando *load* para cargar secuencialmente 10 ficheros de datos

```
fname = 'misdatos'
for i = 1 : 10
      eval(['load ',fname,int2str(i)])
end
```

1.5. Ficheros en disco

1.5.1. Manipulación de ficheros

Podemos utilizar las siguiente funciones y comandos de manipulación ficheros, y que son genéricos a todos los sistemas operativos.

dir	Mostrar contenido del directorio actual Se pueden utilizar rutas y comodines Ejemplo: *dir *.mat* o *dir('*.mat')*
type	Mostrar contenido de un fichero Se puede utilizar la ruta del fichero Ejemplos: *type media.m* o *type('media.m')* *type c:\autoexec.bat*
delete	Borrar un fichero Se pueden utilizar rutas y comodines Ejemplos: *delete media.m* o *delete('media.m')*
cd	Cambiar de directorio o muestra el directorio actual en caso de no utilizar argumentos. Ejemplo: *cd c:\temp*

Para cambiar de unidad de disco se utiliza el comando *cd*

```
cd drive:
```

1.5.2. Importación de datos

Podemos introducir datos desde otros programas en OCTAVE mediante diversos métodos.

La mejor forma de importar datos depende de la cantidad de datos que haya, de la forma de estos datos, etc.. A continuación hay varias elecciones:

- Introducir los datos como una lista de elementos explícita. Si existe una pequeña cantidad de datos, digamos por ejemplo, de 10 a 15 elementos, es fácil introducir manualmente los datos utilizando corchetes. Este método no es aconsejable para una cantidad de datos grande porque no podemos editar los datos si se comete un error.

- Crear un fichero-M. Utilizar un editor de textos para crear un fichero-M de tipo *script* que introduzca los datos como una lista explícita de elementos. Este método es útil cuando los datos no están en formato legible por el ordenador y no hay que introducirlos en cualquier caso. Aunque esencialmente es igual que el primer método, este método tiene la ventaja de que podemos utilizar un editor para cambiar los datos o corregir errores. Posteriormente se vuelven a introducir los datos ejecutando de nuevo el fichero-M.

- Cargar los datos desde un fichero de texto ASCII. Un fichero de texto ASCII almacena los datos en formato ASCII, con filas de longitud fija que terminan con un carácter de nueva línea (retorno de carro), y utilizando espacios para separar los números. (Estos ficheros pueden editarse utilizando un editor de textos normal). Los ficheros de texto ASCII pueden leerse directamente desde OCTAVE utilizando el comando *load*, el resultado se pone en una variable cuyo nombre es el nombre del fichero.

  load nombre.ext -ascii

- Leer datos utilizando *fopen, fread* y otras funciones de entrada/salida (E/S o I/O) de bajo nivel. Este método es útil para cargar ficheros de datos desde otras aplicaciones que tienen sus propios formatos de ficheros.

1.5.3. Exportación de datos

Se dan a continuación algunos métodos para exportar datos

- Para matrices pequeñas, utilizar el comando *diary* para crear un fichero diario, y listar las variables en ese fichero. Posteriormente se puede utilizar un editor de textos para manipular el fichero de diario. El fichero diario incluye los comandos de OCTAVE utilizados durante la sesión.

- Grabar los datos en formato ASCII utilizando la función *save* con la opción *-ascii*

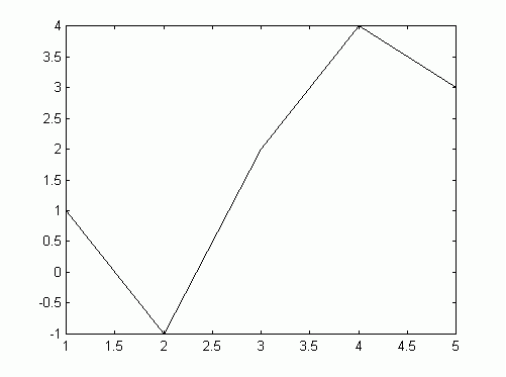
  save nombre.ext -ascii

- Escribir los datos en formato especial utilizando *fopen, fwrite,* y otras funciones de E/S de bajo nivel. El método es útil para escribir ficheros de datos en formatos requeridos por otras aplicaciones.

1.6. Gráficos

El sistema gráfico de OCTAVE proporciona una gran variedad de técnicas para la presentación y visualización de datos. Este sistema está construído sobre la base de una colección de objetos gráficos, como líneas y superficies, cuya apariencia puede controlarse mediante los valores de las propiedades de los objetos. Aunque gracias a la gran cantidad de funciones y programas gráficos que incorpora OCTAVE no es necesario acceder a esos objetos a bajo nivel.

Las siguientes secciones describen como utilizar las capacidades gráficas de OCTAVE para la representación de datos en 2 y en 3 dimensiones.



1.6.1. Gráficos 2D

OCTAVE proporciona una gran variedad de funciones para representar datos mediante gráficos en 2D y anotaciones en esos gráficos. Se describen a continuación dichas funciones.

Tipos de Ejes

plot	Crear una gráfica con escala linear en ambos ejes
loglog	Crear una gráfica con escala logarítmica en ambos ejes
semilogx	Crear una gráfica con escala logarítmica en el eje x, y linear en y
semilogy	Crear una gráfica con escala logarítmica en el eje y, y linear en x

Leyendas

title	Añadir un título a la gráfica
xlabel	Añadir un nombre al eje x
ylabel	Añadir un nombre al eje y
text	Añadir texto en una posición específica
gtext	Añadir texto utilizando el ratón
grid	Poner enrejado a la gráfica

Si y es un vector, la sentencia

$$plot\,(y)$$

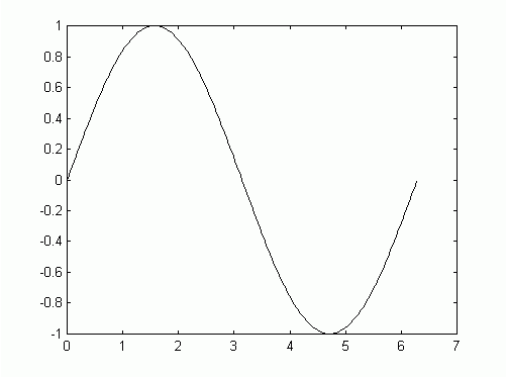
produce una gráfica lineal de los elementos de y en función de los índices de los elementos de y. Por ejemplo si $y = [1, -1, 2, 4, 3]$, entonces $plot(y)$ producirá la figura 1.6.1Si se utilizan dos vectores como argumentos, $plot(x, y)$ produce un gráfico de y respecto a x. Por ejemplo

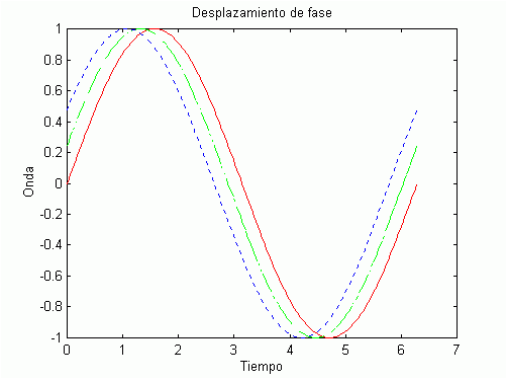
$$\begin{array}{l} x = 0 : pi/100 : 2 * pi; \\ y = \sin(x); \\ plot\,(x, y) \end{array}$$

producirá la gráfica 1.6.1 También podemos representar un conjunto múltiple de datos y definir el estilo de línea y el color para cada uno de los conjuntos de datos en una única llamada de la función, así como colocar un título a la gráfica, y a los ejes coordenados. La secuencia de sentencias

$$\begin{array}{l} t = 0 : pi/100 : 2 * pi; \\ x = \sin(t); \\ y = \sin\left(t + 0{,}25\right); \\ z = \sin\left(t + 0{,}5\right); \\ plot\,(t, x, ' r-', t, y, ' g - -', t, z, ' b :') \\ title(\text{'Desplazamiento de fase'}) \\ xlabel(\text{'Tiempo'}) \\ ylabel(\text{'Onda'}) \end{array}$$

produce la gráfica de la figura 1.6.1El argumento de tipo carácter que se incluye en la llamada a la función *plot*





indica el tipo de línea utilizada, y el color de la misma. La siguiente tabla contiene los estilos de línea y los colores posibles en la utilización de *plot*

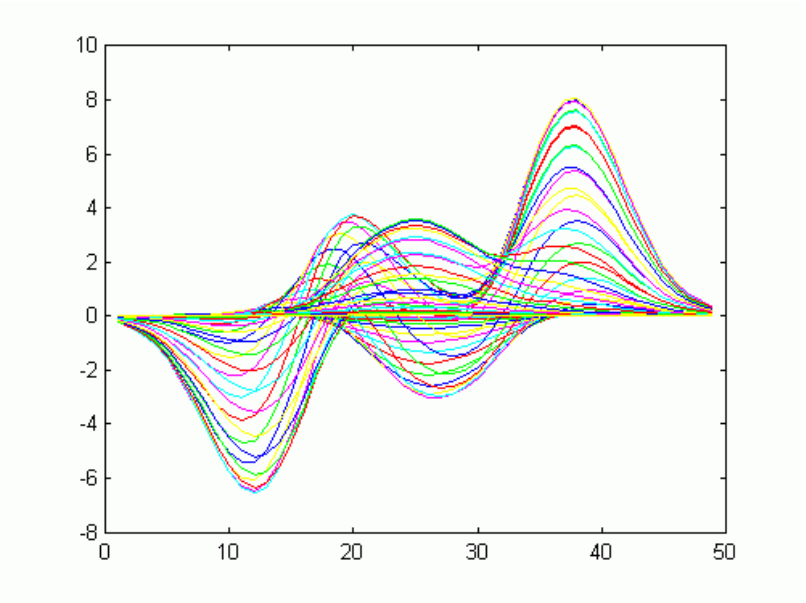
Símbolo	**Color**	**Símbolo**	**Estilo de línea**
y	Amarillo	.	Punto individual
m	Magenta	o	Círculo en cada punto
c	Cian	x	Una x en cada punto
r	Rojo	$+$	Un $+$ en cada punto
g	Verde	$*$	Un $*$ en cada punto
b	Azul	$-$	Continua (Sólida)
w	Blanco	:	Punteada
k	Negro	$-.$	Puntos y rayas
		$--$	Discontinua

Si no especificamos los colores, la función *plot*, automáticamente toma colores de la tabla anterior. Para una línea, el color es azul. Para múltiples líneas, la función *plot* elige los colores cíclicamente entre los 6 primeros colores de la tabla (salvo el negro).

El tamaño de los puntos, círculos, cruces, signos +, y asteríscos se puede modificar.

Podemos añadir nuevos gráficos a uno ya existente, utilizando el comando *hold.* Si utilizamos *hold on* no se borra el gráfico previo, sino que se incorpora el nuevo gráfico al actual, aunque hay re-escalamiento de los ejes si los datos están fuera de los mismos. Por ejemplo

$plot\,(t, x,'\,r-')$
$hold\ on$
$plot\,(t, y,'\,g--')$
$plot\,(t, z,'\,b:')$
$title$('Desplazamiento de fase')
$xlabel$('Tiempo')
$ylabel$('Onda')



producirá la misma gráfica que la figura 1.6.1.

Cuando el vector que se utiliza en la función *plot* es complejo, se ignora la parte imaginaria, salvo cuando se utiliza un único argumento, en este caso se representa la parte imaginaria respecto a la parte real. Por tanto, si Z es un vector o matriz complejo, estas sentencias son equivalentes

$$plot\left(Z\right) \Longleftrightarrow plot\left(real\left(Z\right), imag\left(Z\right)\right)$$

Si se quiere representar más de una matriz compleja, la parte real y la imaginaria tienen que introducirse explícitamente.

La función *plot* también admite una matriz como argumento

$$plot\left(Y\right)$$

En este caso, representa una curva por cada columna de la matriz Y, mientras que en el eje x se representan el vector de índices, $1:m$, donde m es el número de filas de Y. Por ejemplo si utilizamos la función *peaks*, para generar una matriz de datos

$$Y = peaks$$

El comando

$$plot\left(Y\right)$$

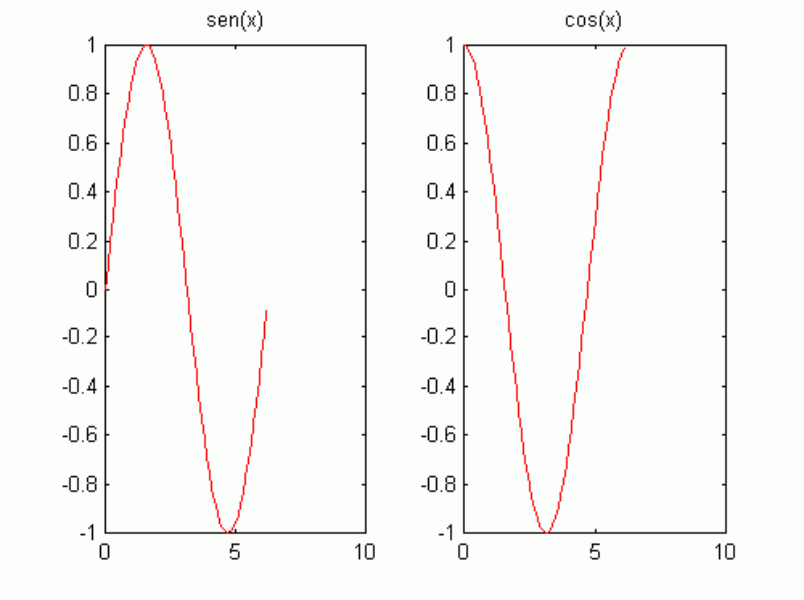
produce el gráfico con 49 curvas de la figura

La función *peaks* genera una matriz de datos basados en la función de 2 variables

$$f\left(x,y\right) = 3\left(1-x\right)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

La función *plot* también admite dos matrices como argumentos. En general, si *plot* se utiliza con dos argumentos, X e Y, y alguno de ellos tiene más de una fila o columna, entonces:

- Si Y es una matriz, y x es un vector, $plot(x,Y)$ representa sucesivamente las filas o columnas de Y respecto a x, utilizando diferentes colores o estilos de línea. La elección entre filas y columnas se hace en función del número de elementos de x, si Y es cuadrada, se utilizan sus columnas.
- Si X es una matriz e y es un vector, $plot(X,y)$ representa cada fila o columna de X respecto al vector y. La elección entre filas y columnas se hace en función del número de elementos de y, si X es cuadrada, se utilizan sus columnas.



- Si X e Y son matrices de la misma dimensión, $plot(X, Y)$ representa las columnas de Y respecto a las columnas de X.

Podemos utilizar *plot* con múltiples pares de matrices como argumentos

$$plot\,(X1, Y1, X2, Y2, \ldots)$$

Se puede dividir la ventana de gráficos de forma que podamos observar dos o más curvas en ella utilizando cada una sus repectivos ejes. Para ello utilizaremos la función *subplot*. La estructura de esta función es

$$subplot\,(m, n, p)$$

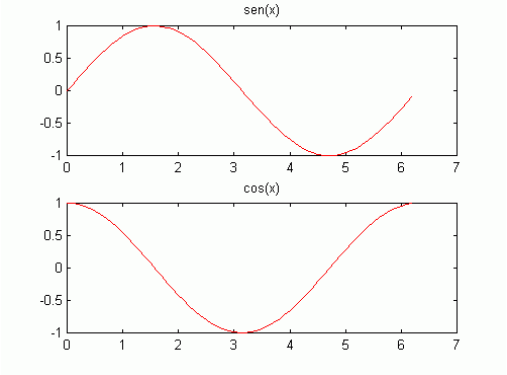
que divide a la ventana de gráficos en $m \times n$ zonas con ejes independientes, asignádole a cada zona un índice desde 1 hasta $m \times n$, de izquierda a derecha y de arriba a abajo, y declara como zona activa para gráficos la zona de índice p. Por ejemplo, para representar las funciones: $sin(x)$ y $cos(x)$ en dos zonas individuales, podemos utilizar la secuencia de instrucciones siguiente:

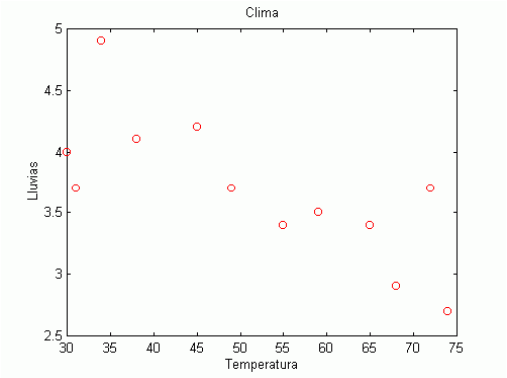
$x = 0 : 0{,}1 : 2 * pi$
$y = sin(x)$
$z = cos(x)$
$subplot(1, 2, 1)$
$plot(x, y)$
$title('sin(x)')$
$subplot(1, 2, 2)$
$plot(x, y)$
$title('cos(x)')$

que produce la gráfica 1.6.1Mientras que la secuencia

$subplot(2, 1, 1)$
$plot(x, y)$
$title('sin(x)')$
$subplot(2, 1, 2)$
$plot(x, y)$
$title('cos(x)')$

produce la gráfica Podemos situar texto en una localización del gráfico utilizando la función *text*. El siguiente





ejemplo muestra el uso de esta función. Definimos dos variables

$$\begin{aligned} temp &= [30,31,38,49,59,68,74,72,65,55,45,34] \\ precip &= [4{,}0,3{,}7,4{,}1,3{,}7,3{,}5,2{,}9,2{,}7,3{,}7,3{,}4,3{,}4,4{,}2,4{,}9] \end{aligned}$$

que indica la temperatura y el volumen de lluvia en una determinada zona. Podemos hacer una gráfica que represente una variable frente a la otra,

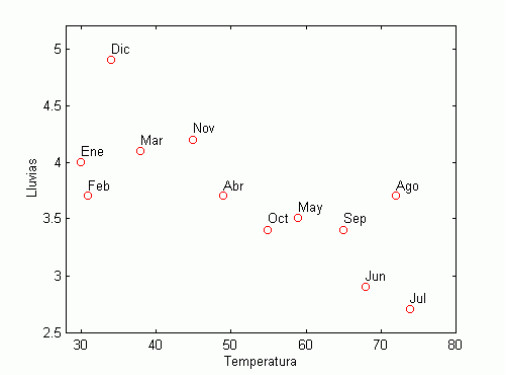
$$\begin{aligned} &plot(temp, precip, 'o') \\ &xlabel('Temperatura') \\ &ylabel('Lluvias') \\ &title('Tiempo') \end{aligned}$$

que proporciona la gráfica 1.6.1Podemos mejorar la gráfica anterior introduciendo una etiqueta en cada punto, que identifique el mes al que se refiere la temperatura y lluvia correspondiente, y modificando los ejes para evitar que existan puntos en los bordes de la misma, estas modificaciones declarando la siguiente variable de texto

$$\begin{aligned} meses = \; & ['Ene';'Feb';'Mar';'Abr';'May';'Jun';... \\ & 'Jul';'Ago';'Sep';'Oct';'Nov','Dic'] \end{aligned}$$

y utilizando la siguiente secuencia de instrucciones

$$\begin{aligned} &plot(temp, precip, 'o') \\ &xlabel('Temperatura') \\ &ylabel('Lluvias') \\ &title('Tiempo') \\ &axis([28,80,2{,}5,5{,}2]) \\ &text(temp, precip + 0{,}1, meses) \end{aligned}$$



Donde se ha empleado un pequeño desplazamiento en el eje y, $(precip + 0{,}1)$, para que no se solape el texto con el punto. El resultado puede observarse en la figura 1.6.1En la siguiente tabla se incluye un conjunto de funciones para relizar tipos de gráficos específicos

Funciones Gráficas	
bar	Crea un gráfico de barras
compass	Gráfico de flechas para representar complejos
errorbar	Crea un gráfico con barras de error
feather	Gráfico de flechas para representar complejos
fplot	Representación de funciones analíticas
hist	Histogramas
polar	Gráficos en coordenadas polares
quiver	Representa campos de velocidades
rose	Diagrama de Sectores
stairs	Diagrama de barras sin líneas internas
fill	Polígono relleno

Ejemplos:

1. **Relleno de polígonos**

   La función *fill* define un área poligonal cuyo interior se rellena de un color específico. Si utilizamos un color por cada puntos, entonces, se utiliza interpolación bilineal para determinar el color del interior. La secuencia

   $$t = 0 : 0{,}05 : 2 * pi;$$
   $$x = sin(t)\,;$$
   $$fill(x, t, 'b')$$

   produce el polígono relleno de azúl de la figura 1

Gráficas de funciones matemáticas

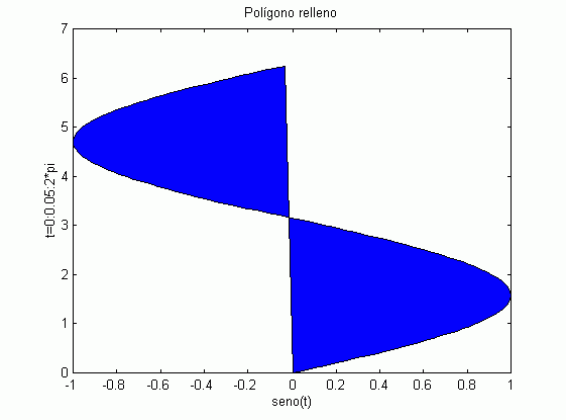
La función *fplot* es una forma de dibujar funciones matemáticas cuando conocemos su expresión analítica. La función *fplot* evalua la función en regiones donde la tasa de variación de la función es grande, para poder representarla mejor.

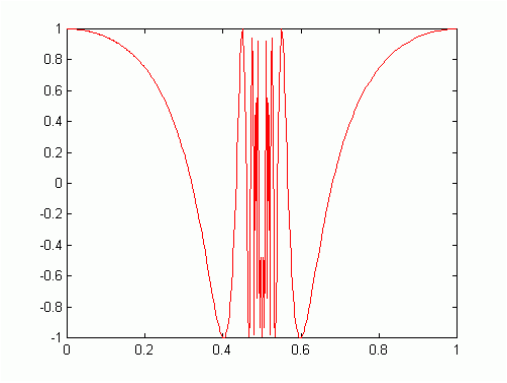
Para representar una función, creamos un fichero *.m* con la definición de la función y pasamos el nombre de este fichero a la función *fplot.* Por ejemplo, el siguiente fichero define una función llamada *fof(x)*

$$function\ y = fofx(x)$$
$$y = cos(tan(pi*x));$$

Pasamos el nombre de la función al comando *fplot*, junto con el intervalo donde queremos evaluar la función.

$$fplot('fofx', [0, 1])$$





que produce la gráfica 1.6.1El comando *fplot* admite dos parámetros optativos, que son el tipo y color de línea, de la misma forma que el comando *plot*; y el nivel de tolerancia que indica el número de evaluaciones, n, de la función según la siguiente fórmula

$$n = (1/TOL) + 1$$

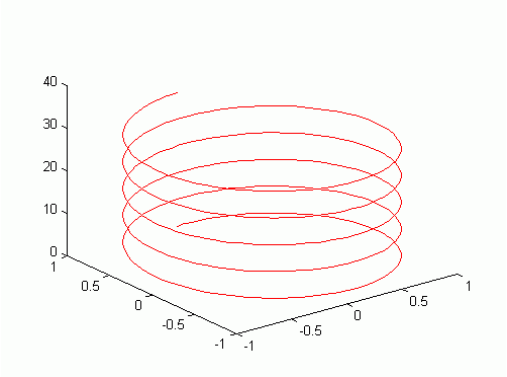
1. También podemos indicar el rango de la variable dependiente, de forma que la llamada completa de la función podría ser

   fplot('nombre',[xmin, xmax, ymin, ymax],tipo,tol*)*

1.6.2. Gráficos 3D

OCTAVE proporciona una gran variedad de funciones para representar datos $3D$. Algunos dibujan líneas en tres dimensiones, mientras que otros representan superficies. La siguiente tabla resumen un conjunto de esas funciones

plot3	Dibuja líneas y puntos en 3D
contour	Crea isocontornos
contour3	Crea isocontornos en 3D
pcolor	Dibuja una matriz rectangular de celdas, cuyos colores están determinados por los elementos de la matriz
image	Dibujar una matriz como una imagen
mesh, meshc, meshz	Representa una superficie
surf, surfc, surfl	Representa una superficie
fill3	Crea un polígono relleno en 3D
zlabel	Crea una etiqueta en el eje z
clabel	Pone etiquetas en los isocontornos
view	Cambia las perspectivas de visión de una gráfica 3D
viewmtx	Calcula una matriz de transformación



Dibujos de líneas

La función para dibujar líneas en 3D es *plot3.* Si x, y y z son tres vectores de la misma longitud

$$plot3(x, y, z)$$

genera una línea en el espacio utilizando los puntos cuyas coordenadas son los elementos de x, y y z. Y reproduce una proyección 2D de esa recta en la pantalla. Por ejemplo

$$t = 0 : pi/50 : 10 * pi$$
$$plot3(sin(t),cos(t),t);$$

produce la hélice de la gráfica 1.6.2Si X,Y y Z son matrices de las mismas dimensiones,

$$plot3(X, Y, Z)$$

dibuja las curvas obtenidas de las columnas de X, Y y Z. Para especificar diversos tipos de línea, símbolos y colores, utilizamos el comando

$$plot3(X, Y, Z, s)$$

donde s es una cadena de 1, 2 o 3 caracteres como en el caso de *plot.* Al igual que el comando *plot* también podemos realizar los gráficos de varias curvas con una sola llamada a la función *plot3*

$$plot3(x1, y1, z1, s1, x2, y2, z2, s2, \dots)$$

Función meshgrid

Para realizar el gráfico de una función de dos variables, $z = f(x, y)$, primero hay que generar dos matrices X e Y que consisten en la repetición de filas y columnas, respectivamente, sobre el dominio de la función. Posteriormente se utiliza esas matrices para evaluar y dibujar la función.

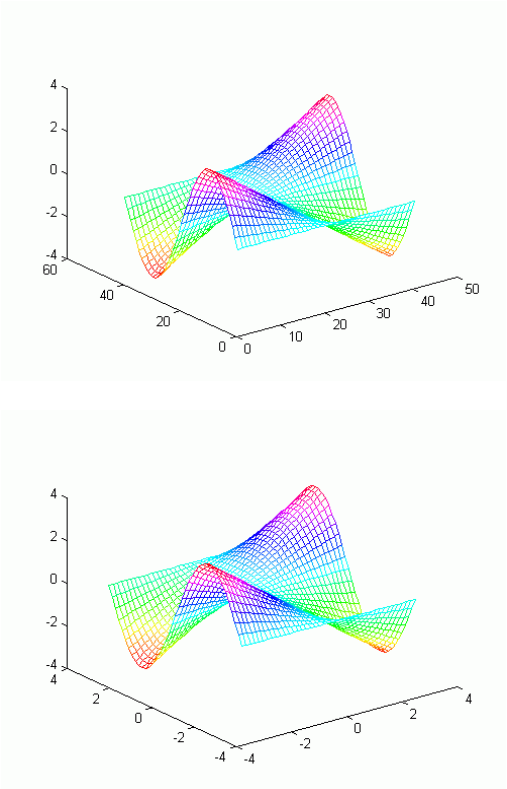
La función *meshgrid* transforma el dominio especificado por dos vectores x e y, en dos matrices X e Y. Podemos, entonces, utilizar esas matrices para evaluar funciones de dos variables. Las filas de X son copias del vector x, y las columnas de Y son copias del vector y.

Veamos un ejemplo que ilustra la utilización de la función *meshgrid,* que utilizaremos para dibujar la función $x*sin(y)$, en el cuadrado $[-\pi, \pi] \times [-\pi, \pi]$

$$x = -pi : pi/20 : pi;$$
$$y = x;$$
$$[X, Y] = meshgrid(x, y)$$
$$Z = X.*sin(Y);$$
$$mesh(Z)$$

Se ha utilizado la función *mesh* para dibujar la superficie de la función. Podemos ver los resultados en la gráfica 1.6.2 Si queremos utilizar el verdadero rango para las variables x e y, utilizamos la expresión

$$mesh(X, Y, Z)$$



que produce la imagen 1.6.2

La función *surf* representa también una superficie, y difiere de *mesh* en que mientras esta representa una superficie utilizando líneas; *surf* representa la superficie, utilizando cuadrados de color. Podemos ver el resultado en la figura 1.6.2

Dibujos de Contornos

Se pueden generar isocontornos tanto en 2D como en 3D. Las funciones *contour* y *contour3* son las encargadas de realizar estas labores, representando líneas de valor constante a partir de una matriz. Opcionalmente podemos especificar el número de isocontornos, la escala de los ejes, y los valores donde va a dibujarse un isocontorno.

Por ejemplo la siguiente instrucción genera un dibujo de contornos, con 20 líneas de isocontorno, de la función $x* \ sin(y)$, del apartado anterior

$$contour(Z, 20)$$

para producir la figura 1.6.2o

$$contour(X, Y, Z, 20)$$

si queremos la escalar original para los ejes x e y.

Para dibujar isocontornos en $3D$ utilizamos

$$contour3(Z, 20)$$

que nos proporciona la gráfica

Variaciones de surf y mesh

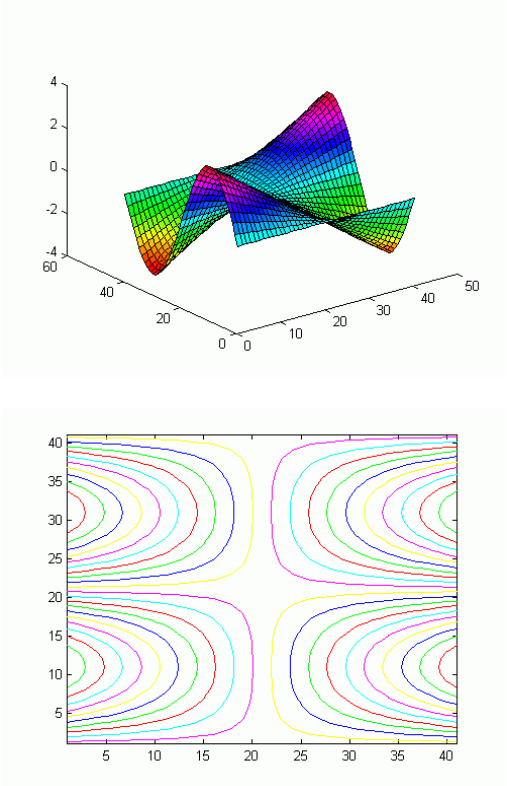
Podemos combinar superficies e isocontornos en un mismo gráfico. Por ejemplo, *surfc*, genera una superficie junto con un gráfico de isocontornos 2D dibujados en el plano $z = 0$. Por ejemplo

$$surfc(Z)$$

produce el gráfico de la figura 1.6.2

La función *meshz* genera una superficie utilizando una plano de referencia en la z mínima. Por ejemplo

$$meshz(Z)$$



dibuja la superficie definida en Z, y un plano en el mínimo de la superficie. Este plano mínimo se conecta con el plano $z = 0$. Vemos los resultados en la figura 1.6.2

La función *surfl* genera una superficie sombreada como si estuviera iluminada desde una determinada posición. Por ejemplo

surfl(Z, [−10, 50])
colormap(gray)
shading flat

genera la gráfica de la figura 1.6.2. Se ha utilizado la función *colormap* para cambiar la paleta de colores a *gray* (grises), y se ha utilizado el comando *shading flat* para eliminar la malla creada por el comando *surfl.*

1.6.3. Funciones gráficas generales

Viewpoint

Podemos cambiar el ángulo desde el cual observamos una superficie 3D. Con la función *view* podemos seleccionar el ángulo de visión en coordenadas esféricas, especificando el azimuth y la elevación del punto de vista respecto del origen de coordenadas. El azimuth es un ángulo polar en el plano $x - y$, medido en el sentido contrario a las agujas del reloj que indica la rotación del punto de vista. La elevación es un ángulo por encima (positivo) o por debajo (negativo) del plano $x - y$.

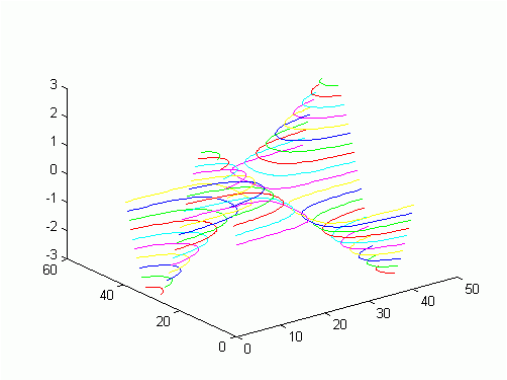
Las imágenes 1.6.3 y 1.6.3 han sido obtenidas mediante los comandos *view(−7, 80)* y *view(−7, −10)* respectivamente, a partir de la imagen de la figura 1.6.2

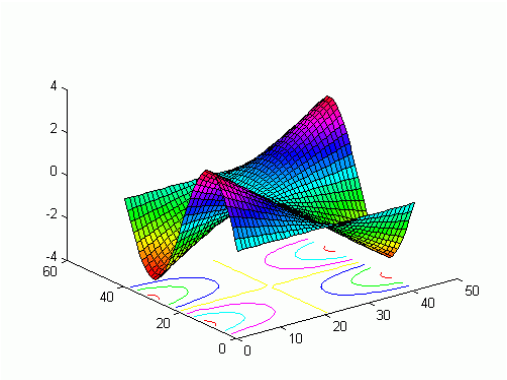
Ejes

La función *axis* tiene un número de opciones que permite modificar la escala, orientación y aspecto de los dibujos.

Normalmente, OCTAVE encuentra el mínimo y máximo de los datos que hay que dibujar y elige los ejes apropiadamente. Se puede elegir el rango de valores en los ejes, utilizando la siguiente instrucción

$$axis(\ [x_{\text{mín}}, x_{\text{máx}}, y_{\text{mín}}, y_{\text{máx}}]\ )$$





para dibujos en 2D o

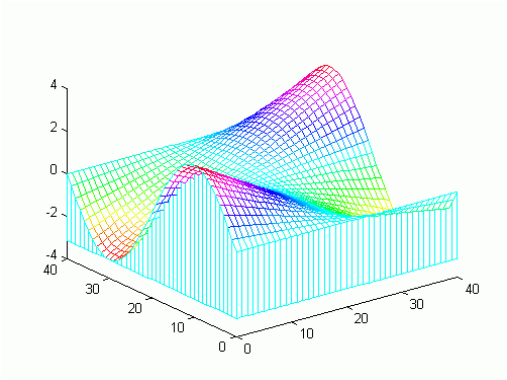
$$axis(\ [x_{\text{mín}}, x_{\text{máx}}, y_{\text{mín}}, y_{\text{máx}}, z_{\text{mín}}, z_{\text{máx}}]\ )$$

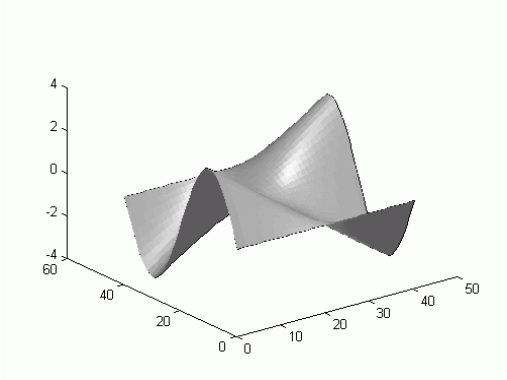
para gráficos en 3D.

La siguiente tabla proporciona los

axis('auto')	Retorno al modo automático de elección de límites
$v = axis$	Guarda los límites actuales en la variable v Podemos utilizarlos después con: *axis(v)*
axis(axis)	Libera la escala de los límites actuales
axis('ij')	Ejes en modo matriz. El origen está en la esquina superior derecha. El eje i vertical y se numera de arriba a abajo. El eje j es el horizontal y numerado de izquierda a derecha.
axis('xy')	Ejes en modo cartesiano.
axis('square')	Relación entre eje x e y:
axis('equal')	Relación entre eje x e y
axis('on') *axis('off')*	Activa y desactiva la presencia de los ejes

Líneas Ocultas





El comando *hidden* permite activar o desactivar la eliminación de líneas ocultas. Desactivar la ocultación de líneas puede ser útil si queremos añadir diversos gráficos en un único dibujo. Por ejemplo

*mesh(peaks(*20*)+*7*)*
hold on
pcolor(peaks(20*))*
hidden off

Dibuja una superficie utilizando la función *peaks,* encima de una superficie de pseudocolores generados mediante la función *pcolor* y utilizando los mismos datos. La función *hidden off* desactiva la ocultación de líneas, permitiendo ver la superficie de pseudocolores a traves de la malla. Podemos ver el resultado en la figura 1.6.3.

Figure

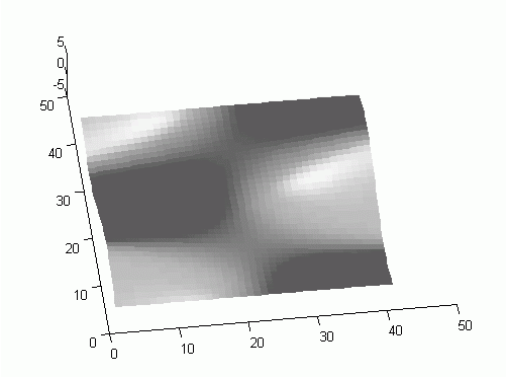
Mediante la función *figure* sin argumentos creamos otra nueva ventana gráfica. Mientras que *figure(N)* hace activa la ventana gráfica *N*.

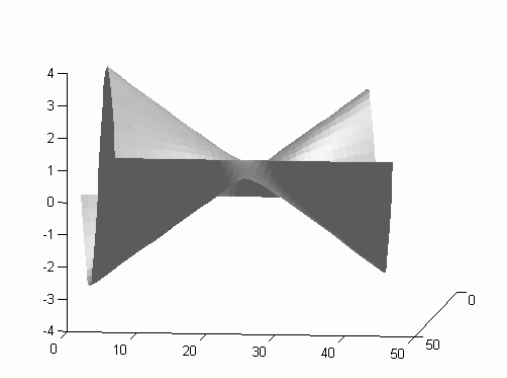
Impresión de Gráficos

Se puede generar una salida PostScript del contenido de cualquier ventana gráfica en OCTAVE utilizando el comando *print.* El comando *print* envía la salida directamente a la impresora del sistema o escribe la salida en un fichero si especificamos un nombre de fichero. Podemos especificar el tipo de PostScript según la tabla siguiente:

-dps	PostScript
-dpsc	PostScript Color
-dps2	PostScript Nivel 2
-dpsc2	PostScript Color Nivel 2
-deps	PostScript Encapsulado
-depsc	PostScript Color Encapsulado
-deps2	PostScript Encapsulado Nivel 2
-depsc2	PostScript Color Encapsulado Nivel 2

En general, los ficheros del Nivel 2 son más pequeños y se imprimen más rapidamente, sin embargo, no todas las impresoras PostScript tienen el Nivel 2.





Normalmente la impresión se hace en modo invertido, es decir, el negro del fondo de los gráficos se convierte en blanco y los ejes y las gráficas se dibujan en negro. Si se quiere mantener el aspecto del gráfico podemos utilizar el comando

set(gcf,'InverHardCopy','off')

Portapapeles

Se puede intercambiar información entre OCTAVE y otras aplicaciones utilizando el portapapeles de Windows.

Para transferir un dibujo en una ventana de gráficos de OCTAVE (figure windows), usar la secuencia de teclas:

Alt+Print Screen ó Alt+Impr. Pant.

para copiar el contenido de la ventana en el portapapeles. A continuación utilizar la opción de *Pegar* (*Paste*) para incorporar el contenido del portapapeles en la aplicación.

Se puede utilizar las opciones **Copy to Bitmap** o **Copy to Metafile** del menú **Edit** de una ventana de gráficos para copiar una imagen.

Se puede incorporar texto desde otra apliación en la línea de comandos utilizando la opción **Paste** del menú **Edit.**