PRÁCTICA 4. MATLAB

Gráficas, entrada y salida de datos.

universidad de león

1. Gráficas

La representación de datos en Matlab es uno de sus puntos fuertes ya que dispone de multitud de funciones para representar series de puntos, superficies, curvas de nivel, etc. Además de disponer de diferentes herramientas de visualización los gráficos mostrados son configurables con el fin de poder etiquetarlos hacer presentaciones conjuntas…

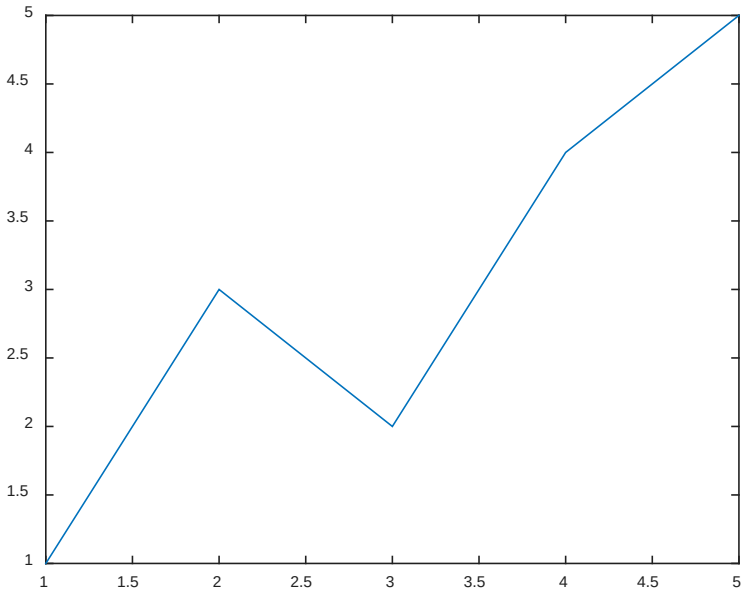
1.1 Representación de curvas en el plano

La forma más sencilla de gráfica es aquella en la que se dispone de dos series de datos x e y que están relacionadas y que se pueden representar siendo x la coordenada del eje de abscisas e y la coordenada del eje de ordenadas. Para representar estos datos se utiliza el comando `plot`

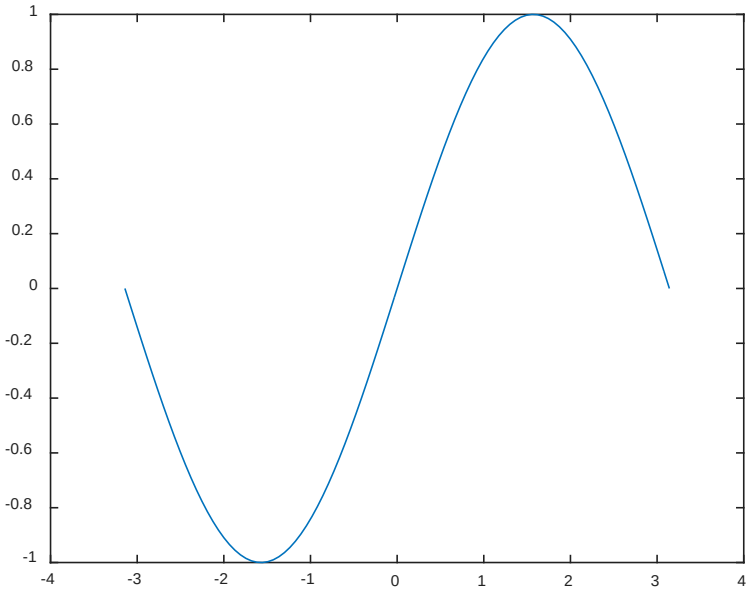
```
plot(x,y,LineSpec)
```

En la función x se corresponde con las abscisas, y es el vector de ordenadas y LineSpec es un especificador del tipo de línea a utilizar. Cuando no se pasa las coordenadas del eje x Matlab asume que el vector `x = 1:length(y)`

```
>> plot([1 3 2 4 5])
```




```
>> x=linspace(-pi,pi,100);
>> plot(x,sin(x))
```



El string LineSpec es una serie de parámetros que se pasan como un string todos juntos que permiten modificar el estilo de la línea. Los atributos de la línea que se puden modificar son:

Colores

x	amarillo
m	magenta
c	cian
r	rojo
g	verde
b	azul
w	blanco
k	negro

Puntos

Matlab lo que hace es crear líneas que unen los puntos que se le pasan a la función plot a través de sus coordenadas. Es posible además de dibujar las líneas, dibujar los puntos en las coordenadas marcadas con un marcador que se define como:

.	punto
o	circulo
x	equis
+	cruz
*	asterisco

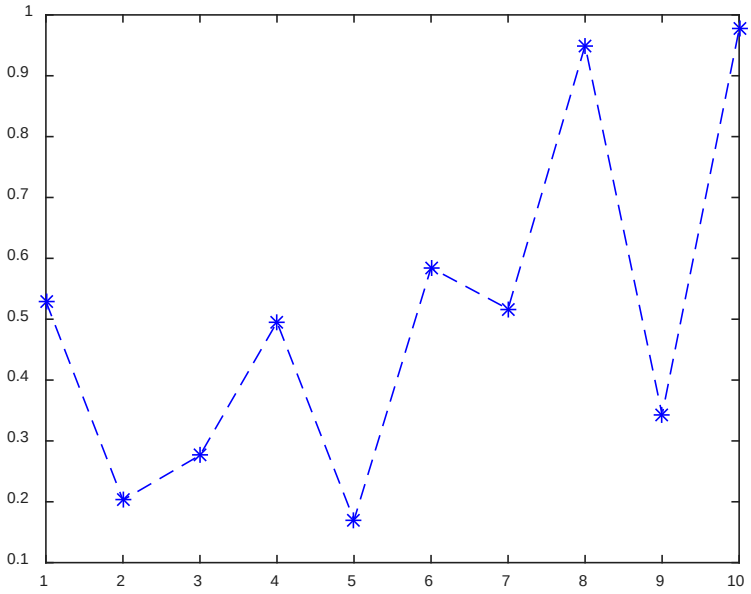
Forma de la línea

Los tramos de línea dibujado pueden representarse de una forma concreta introduciendo los siguientes caracteres

`-`	línea continua
`:`	línea de puntos
`-.`	punto y raya
`--`	línea discontinua

Para plotear una gráfica en color azul discontinua en la que los puntos se marcan con un asterisco:

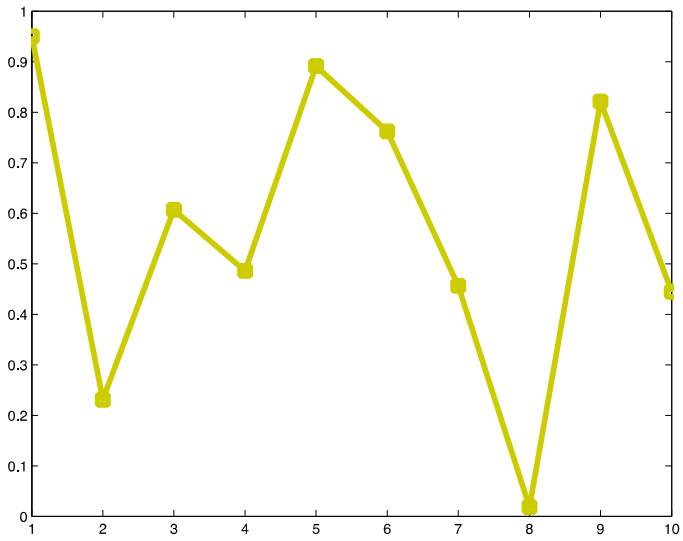
```
>> plot(x,y,'b*--')
```



Los modificadores de propiedad de una gráfica también se pueden pasar por medio de pares (‘Propiedad’, ‘Valor’). Este paso de parámetros permite modificar más parámetros de la gráfica como puede ser el grosor de la línea o el tamaño de los puntos, pudiendo combinarse con el LineSpec que vimos anteriormente. Se pueden modificar varios parámetros simultáneamente, pero siempre pasando el par de propiedad valor, ya que en caso contrario genera un error. Existen muchas opciones que se pueden consultar en la ayuda de Matlab, pero los más utilizados son:

`'LineWidth', ancho`	siendo ancho la anchura de la línea
`'MarkerSize', size`	donde size es el tamaño del símbolo
`'Color',[r,g,b]`	siendo r, g, b, valores ∈ [0, 1]

```
>> plot(x,y,'b*- ','LineWidth',4,'MarkerSize',9,'Color',[.8,.8,0])
```

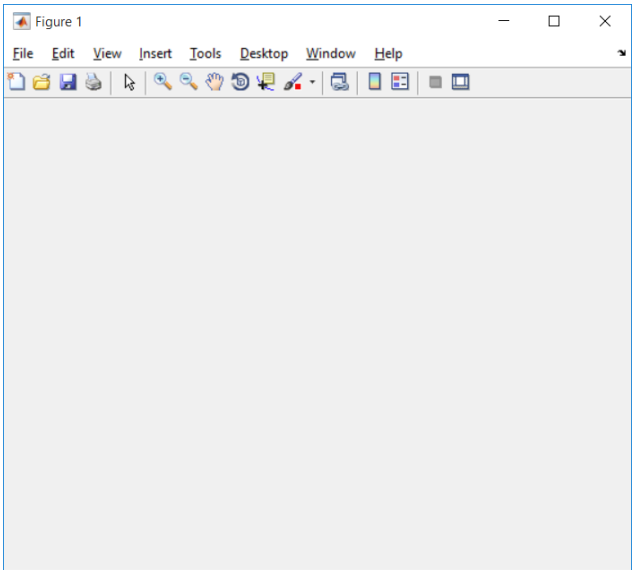


En este caso estamos modificando dos veces el color de la línea, por un lado, con la b del LineSpec y por otro lado con la opción 'Color'. Finalmente prevalece esta última por ser el último valor que se le pasa. Siempre se pasa primero como parámetros el LineSpec y luego los pares de opciones.

1.2 Ventanas de figuras

Cuando llamamos alguna función de representación gráfica Matlab debe antes crear una ventana en el entorno gráfico, para ello usa la función `figure`. No es necesario que usemos esta función siempre; cuando no tengamos ninguna ventana activa Matlab la llamará por nosotros. Será necesario llamarla cuando queramos utilizar varias ventanas a la vez.

```
>> figure(1)
```



Es dentro de este marco en el cual Matlab realiza la visualización de los datos. Así la próxima vez que llamemos a plot, Matlab realizará el dibujado en la ventana que se encuentre activa.

Si no existe ninguna ventana activa cuando llamemos a plot Matlab abrirá una ventana de nombre *figure 1* donde va a dibujar todo a partir de ahora. Si llamamos a otra rutina gráfica, sea cual sea, la va a dibujar en la ventana activa, *figure 1*. Si queremos dibujar en otra ventana tendremos que llamar la función `figure` usando como argumento el número de una ventana que no se encuentre activa:

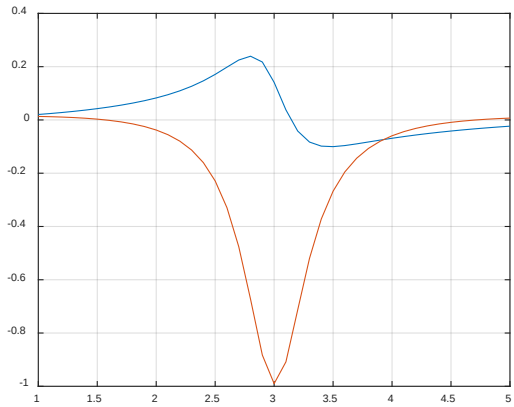
```
>> figure (2)
```

A partir de ahora la ventana 1 va a estar inactiva y todo lo que dibujemos va a expresarse en la ventana 2

Cuando llamamos a la función `plot` el dibujo lo hace en la ventana activa sobrescribiendo la gráfica anterior. En el caso de que queramos dibujar varias gráficas en la misma ventana tenemos que utilizar la función `hold on`. Esta función hace que la gráfica permanezca fija y las siguientes las dibuje en la misma gráfica compartiendo los ejes. Cuando queramos sobrescribir la gráfica llamamos a la función `hold off` que desactiva esta opción.

Ejemplo de hold on

```
x=1:0.1:5;
y = 1./(1 + 10*(x-3).^2).*sin(x);
z = 1./(1 + 10*(x-3).^2).*cos(x);
plot(x,y)
hold on
plot(x,z)
grid on
```



La opción `grid on` muestra una cuadricula en la gráfica. Otra opción para dibujar las dos líneas en esta misma gráfica es haber pasado toda la familia de puntos en la misma línea como pares abscisas y ordenadas.

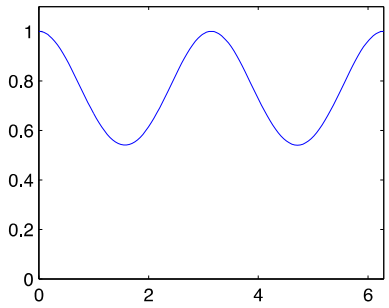
```
>> plot(x, y, x, z)
```

En el caso de que en la misma figura queramos dibujar gráficas con diferentes ejes podemos utilizar el comando `subplot.` Funciona exactamente igual que `figure` pero opera dentro de la misma ventana. Se llama con tres argumentos, el primero son el número de subgráficas por fila, el segundo el número de subgráficas por columna y el tercero es la subgráfica que activamos en cada momento.

`>> subplot (m, n, k)` mxn subventanas y realiza el dibujo en la ventan k contando por filas.

Ejemplo de subplot

```
>> subplot(221); fplot('cos(sin(x))',[0 2*pi 0 1.1])
>> xlabel('x'); ylabel('y'); title('posicion 1')
>> subplot(222); fplot('exp(sin(floor(x)))',[0 2*pi],'r')
>> subplot(223); fplot('abs(sin(x)*x)',[0 4],'b--')
>> subplot(224); fplot('[1+x,1+x+x^2/2,exp(x)]',[-1 1 0 2])
```



1.3 Etiquetas

El paso siguiente es poner etiquetas: un identificador para cada eje y un título si lo creemos necesario. Las etiquetas se aplicarán sólo en la ventana activa.

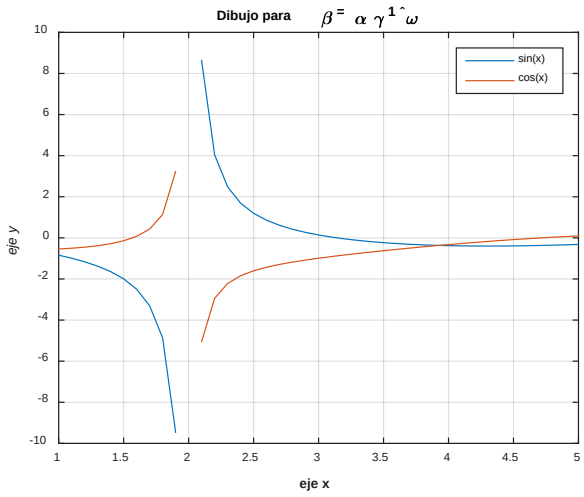
`title(str)`	Añade un título a la figura activa
`xlabel(str)`	Añade una etiqueta al eje *x* de la ventana activa
`ylabel(str)`	Añade una etiqueta al eje *y* de la ventana activa
`legend(...)`	Añade una leyenda a la gráfica

La forma de utilizar la función `legend` es pasarle como argumento tantas cadenas de caracteres como curvas hayamos representado y automáticamente asignará por orden cada curva al identificador. Tanto para la función legend como para las otras funciones de etiquetas se puede usar notación **Tex** para pasar formulas y símbolos en el texto.

Para limitar los ejes de una gráfica en el caso de que nos de algún valor anormalmente grande se utiliza la función `axis([xmin xmax ymin ymax])` que limita la escala de los ejes a los valores indicados

Ejemplo

```
x=1:0.1:5;
y = 1./(1 + (x-3)).*sin(x);
z = 1./(1 + (x-3)).*cos(x);
plot(x,y,x,z)
title('Dibujo para \beta = \alpha {\gamma 1}^\omega')
xlabel('\bf eje x');
ylabel('\it eje y')
legend('sin(x)','cos(x)');
grid on
```

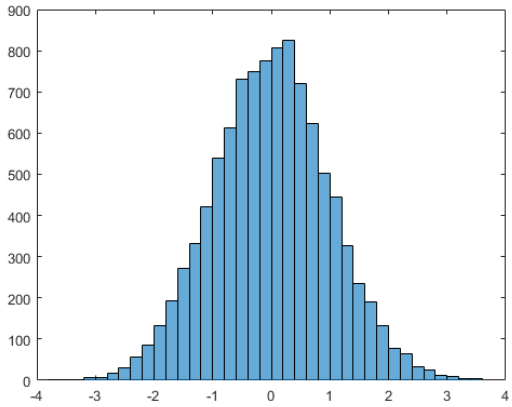


1.4 Otras funciones plot

`histogram(X)`

Crea un gráfico histograma de los datos que se le pasan

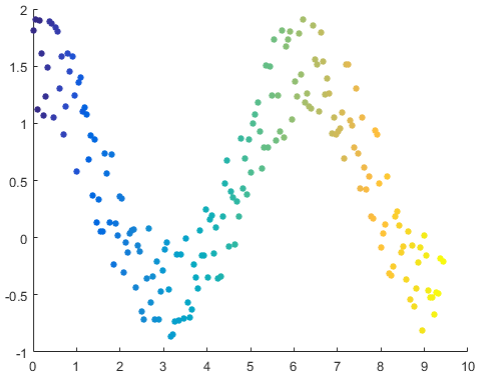
```
x = randn(10000,1);
h = histogram(x)
```



`scatter(x,y,a,c)`

Crea un gráfico de puntos bidimensionales, la diferencia con respecto a `plot` es que permite determinar el tamaño y los colores de los puntos de forma individual. Se utiliza sobre todo para mostrar características que puedan tener los puntos ploteados a parte de las coordenadas. '**a**' especifica el tamaño del círculo y **'c'** el color del mismo, se puede dar la misma propiedad a todos los círculos o pasar un vector del mismo tamaño que x e y con el valor individual. Con la opción **'filled'** el color del elemento es sólido.

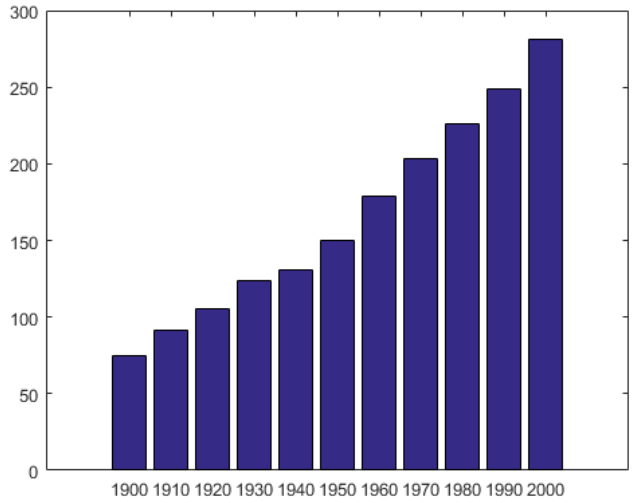
```
x = linspace(0,3*pi,200);
y = cos(x) + rand(1,200);
a = 25;
c = linspace(1,10,length(x));
scatter(x,y,a,c,'filled')
```



```
bar(x,y)
```

Dibuja barras de tamaño y en las posiciones definidas por x.

```
x = 1900:10:2000;
y = [75 91 105 123.5 131 150 179 203 226 249 281.5];
bar(x,y)
```



2. Entrada y salida de datos

En Matlab a la hora de introducir y mostrar los datos se puede realizar bien a través de la pantalla de comandos o bien por medio de la escritura y la lectura de ficheros de datos.

2.1 Entrada y salida de datos por pantalla

Entrada

Existe la opción de introducir valores en Matlab desde la línea de comandos por parte de un usuario. Para ello se utiliza la función `input` de Matlab:

```
variable = input('Ingrese el valor de la variable: ')
```

Esta función siempre requiere un argumento de entrada de tipo string que es el mensaje que aparecerá en pantalla. A continuación del mensaje por pantalla se introduce el valor que será almacenado en la variable. Este valor siempre tiene que ser numérico o una matriz si se introduce en el formato adecuado. Para finalizar la entrada por pantalla hay que pulsar Intro

```
>> a = input('Entrada ')
Entrada 12.25
a =
   12.2500
```

```
>> a = input('Entrada ')
Entrada [2 5.4; 1 0]
a =
    2.0000    5.4000
    1.0000         0
```

Si queremos introducir un string en lugar de una variable numérica hay que añadir un parámetro en la función de entrada y que es el parámetro `'s'`. Si se añade este parámetro el texto que se introduzca se considerará siempre string, en ausencia de este parámetro Matlab intentará convertir a número y si no tiene tal formato devolverá error.

```
>> a = input('Entrada ', 's')
Entrada String de entrada
a =
     String de entrada
```

Salida

Para mostrar los datos por pantalla de comandos en Matlab existen tres opciones:

1. No utilizar el punto y coma al final de la instrucción para que Matlab muestre el resultado de la asignación de una variable.

```
>> Variable1 = 10
Variable1 =
    10
```

2. Utilizar el comando `disp` de Matlab que imprime por pantalla la variable indicada en la función omitiendo el nombre de la variable.

```
>> disp(Variable1)
    10
```

3. Utilizar el comando `fprintf` que permite configurar el formato de salida por pantalla. El funcionamiento de esta función es similar a otros lenguajes de programación. Se introduce la cadena de texto con los especificadores de formato dentro de la cadena y a continuación las variables que se van a mostrar dentro del string.

```
>> fprintf('Valor de la variable: %6.2f \n', Variable1)
```

```
El valor de la variable es:  10.00
```

2.2 Lectura y escritura de ficheros

Matlab puede leer y escribir los datos directamente sobre un tipo de fichero propietario de Matlab o sobre ficheros estándar que pueden ser leídos por otros programas.

Archivos .mat

Los archivos .mat son un formato propietario de Matlab que permite almacenar las variables del workspace de Matlab. La forma de acceder a estos ficheros es por medio de las funciones `save` y `load`

```
save('nombre del fichero.mat', 'var1', 'var2' ….)
load('nombre del fichero.mat', 'var1', 'var2' ….)
```

Ambas funciones tienen la misma estructura. Se les pasa el nombre del fichero en el que se van a guardar o del que se van a leer las variables. En el caso de que a la función no se le pasen más parámetros Matlab guardará todas las variables del workspace o que lea todas las variables de ese mismo fichero. Si se pasan los nombres de las variables sólo se almacenarán dichas variables o se leerán únicamente esas variables del fichero.

A la hora de cargar las variables se mantiene el nombre de dichas variables y los nombres de las variables se tienen que pasar como strings. Si se pasa la variable directamente a la función (sin introducirla con apóstrofes) la función devuelve error.

Archivos de texto

Desde Matlab se pueden leer y escribir ficheros de texto siempre y cuando estos tenga una estructura regular. Un ejemplo de este tipo de ficheros son ficheros de datos en los que cada línea se corresponde con una nueva muestra y siempre van a tener la misma cantidad de variables que los definan.

Tanto la escritura como la lectura se realiza por medio de una secuencia de instrucciones: `fopen – fprintf / fscanf – fclose`

Apertura y cierre del fichero

La función `fopen` es la que encarga de abrir el fichero a leer / escribir. Una vez abierto, se procede a realizar las distintas operaciones de lectura y/o de escritura en el mismo. Cuando se termina de operar con el fichero hay que cerrarlo con la función fclose**.**

```
file = fopen('mifichero.txt', 'r');
% Operaciones de lectura del fichero
fclose(file);
```

La función `fopen` devuelve un puntero de Matlab que identifica el fichero sobre el que se van a realizar las operaciones. Si se produce un error en el proceso de apertura el puntero tiene un valor de -1, en caso contrario tiene un valor mayor o igual a 3. Como parámetros de entrada a la función se le pasan:

1. Nombre del fichero: la cadena de texto con el nombre del fichero que se quiere abrir. Si incluye la ruta del fichero se utilizará, si no se buscará o creará el fichero en el directorio de trabajo.

2. Comando de permisos: especifica el modo de apertura del fichero. Lo habitual es abrir el fichero para escribir en él o para leer de él, si bien es también posible abrir el fichero en modo lectura y escritura.

- 'r': Abre el fichero para lectura. Es el modo por defecto.
- 'r+': Abre el fichero en modo lectura-escritura.
- 'w': Abre o crea un nuevo fichero en modo escritura. Si existe se sobreescribe.
- 'w+': Abre o crea un fichero para lectura-escritura. Si existe, se sobreescribe.
- 'a': Abre o crea un nuevo fichero para escritura. Si existe el fichero, añade al final del mismo.
- 'a+': Abre o crea un nuevo fichero para lectura-escritura. Si existe, se añade al final del mismo.

A la hora de cerrar el fichero utilizamos la función fclose

```
fclose(file);
```

Esta función cierra el fichero que se identifica por medio de la variable en la que guardamos el puntero. Si devuelve un cero es que pudo cerrar el fichero, mientras que si devuelve un -1 es que hubo un error.

Escritura / Lectura del fichero

Una vez que tenemos el puntero al fichero realizamos el proceso de escritura y lectura del mismo, para ello utilizamos las funciones fscanf para leer y fprintf para escribir.

```
[A,count] = fscanf(file, 'format', size)
```

La función `fscanf` lee los datos del fichero que viene especificado por el puntero **file** que se ha obtenido en la instrucción `fopen`, convierte esos datos al formato especificado en el string **format** y los devuelve en la matriz A. Por último, la variable **size** permite determinar el número de elementos a leer del fichero:

- n lee n elementos en un vector columna
- inf Lee todos los elementos hasta el final del fichero
- [m,n] Lee suficientes elementos para rellenar una matriz de mxn

```
fprintf(file, 'format', A1, …, An)
```

En el caso de la función `fprintf` la estructura es similar. A diferencia de la anterior no se suelen utilizar variables de salida en la función y en lugar de utilizar una variable para indicar el tamaño se le pasa la matriz que se va a escribir en el fichero con el formato especificado o bien todas las variables que se van a escribir.

El string de formato especifica cómo se van a leer o escribir los datos en el fichero. En el caso de la lectura lo que Matlab hace es ajustar el valor a la forma que se especifica, si coincide sigue leyendo, si no consigue ajustar los datos leídos, entonces la lectura del fichero termina. En el caso de la escritura convierte los valores de la matriz al formato especificado y los escribe como un string en el fichero.

La especificación de formato consiste en una serie de caracteres que indican el tipo de conversión a realizar sobre los datos. Estos especificadores se componen del símbolo de % y una letra. Los conversores utilizados por Matlab son:

%d	números decimales
%e, %f, %g	números en punto flotante

%s	un string

Entre el porcentaje y la letra que indica la conversión se pueden añadir algunos de los siguientes caracteres:

Asterisco (*)	Salta el valor leído, existe coincidencia, pero el valor no se lee o no se escribe.
Un digito	Máximo ancho del campo
Una letra	Indica el tamaño del objeto a recibir; por ejemplo, una h para un entero corto, l para formato long

A la hora de escribir también se pueden introducir en el especificador de formato caracteres especiales, como el de nueva línea que en la escritura es el que nos va a marcar como se van a introducir las filas de la matriz.

`\b`	Borrar
`\n`	Nueva línea
`\t`	Tabulación
`\\`	Barra
%%	Porcentaje

Las funciones `fscanf` y `fprintf` tienen el mismo funcionamiento que sus homólogas en C, la única diferencia es que son operaciones vectorizadas, por lo que la cadena de formato es reutilizada en cada línea para poder trabajar en formato matricial.

Ejemplo

Se va a realizar la exportación de un cell array a un fichero de texto utilizando el comando fprintf.

```
% Creamos el cell C
C = {'Atkins',32,77.3,'M';'Cheng',30,99.8,'F';'Lam',31,80.2,'M'}
C =

    'Atkins'    [32]    [77.3]    'M'
    'Cheng'     [30]    [99.8]    'F'
    'Lam'       [31]    [80.2]    'M'

% Abrimos un fichero llamado celldata
file = fopen('celldata.dat','w');
```

```
% Generamos el descriptor de formato que va a tener el fichero
% 'String' 'Entero' 'Flotante con un decimal' 'String' 'Nueva
linea'
formatSpec = '%s %d %2.1f %s\n';

% Obtenemos las dimensiones de la matriz
[nrows,ncols] = size(C);

% Recorre cada fila y se va escribiendo cada fila de lamatriz
% Si no se pusiese el caracter de nueva linea lo escribiría todo
% en la misma
for row = 1:nrows
    fprintf(file,formatSpec,C{row,:});
end

% Cerramos el fichero
fclose(file);
```

Archivos especiales

Matlab tiene funciones de escritura y lectura para formatos específicos de ficheros que utilizan otros programas y que evitan tener que utilizar las funciones de scan y de print ya que la estructura interna de estos ficheros es conocida y es siempre la misma.

Para leer y escribir ficheros numéricos con algún delimitador fijo que separa los números entre sí como puede ser un tabulador o un punto y coma, Matlab tiene las funciones `dlmread` y `dlmwrite`

```
dlmwrite(file, M, delimiter)
M = dlmread(file, delimiter)
```

En estas funciones directamente se pasa el fichero que se va a leer y los caracteres que van a separar los datos. En el caso de que el separador sea una coma, es un caso particular de los ficheros csv, Matlab emplea funciones dedicadas que son `csvread` y `csvwrite` que tienen la misma sintaxis pero que no se les pasa el delimitador ya que internamente saben que es la coma.

Por último, también destacar que Matlab puede leer y escribir directamente de ficheros Excel .xlsx por medio de las funciones `xlsread` y `xlswrite`

Importdata

Por último, cabe destacar el uso de la función `importdata` en Matlab para leer ficheros grandes de los que no queremos pasar el formato de los ficheros o bien porque mezclan datos de texto y numéricos por lo que no podemos leerlo directamente en una matriz y hay que separar los campos de texto.

Esta función se caracteriza porque es capaz de leer un fichero de datos que tenga una estructura regular, introducir en una matriz los valores numéricos y aislar en un cell en el caso de que haya valores de tipo string.

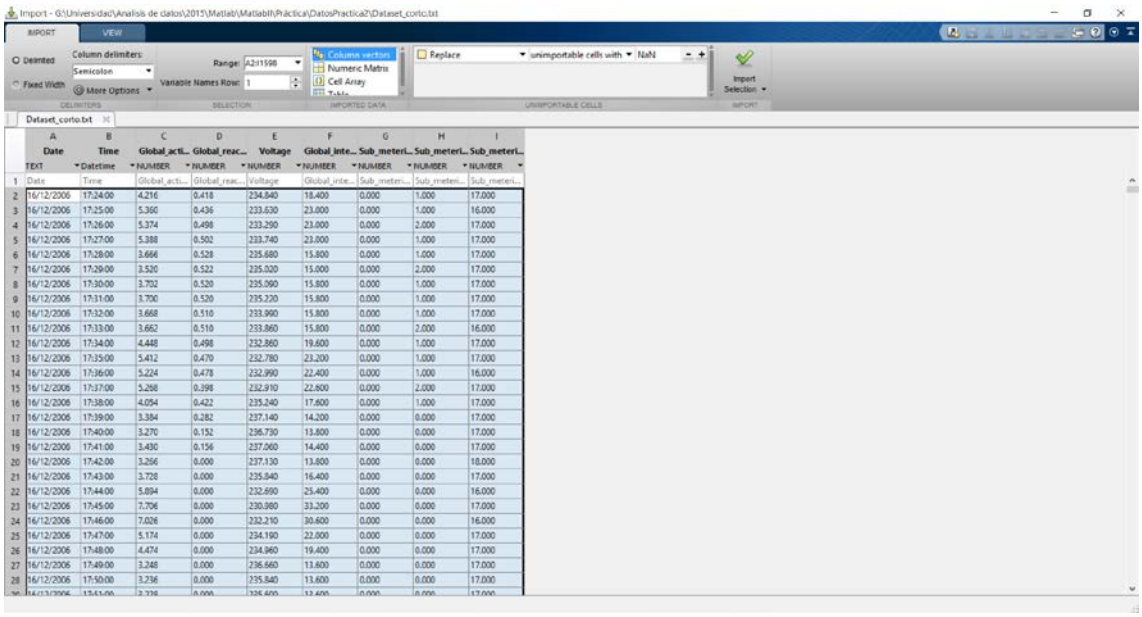
```
>> Datos = importdata('Dataset_corto.txt')
Datos =
           data: [1597x7 double]
       textdata: {1598x9 cell}
```

La función devuelve una variable de tipo estructura en la que va a haber dos campos, uno de los campos es la matriz con los valores numéricos y el otro campo es el cell con los valores separados de lo que considera que es un string.

En el caso de que queramos optimizar la lectura de un fichero queda la alternativa de utilizar el asistente de importación de datos de Matalb que permite además generar código utilizable por scripts propios.

Para lanzar el asistente de importación

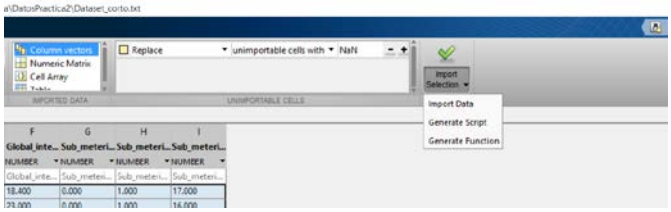
Una vez lanzado se abre un dialogo para seleccionar un fichero con datos para abrir y una vez seleccionado se abre una ventana en donde podemos ver el fichero y configurar como queremos leerlo.



En esta ventana podemos seleccionar que rango de datos queremos leer (si queremos leer todas las columnas o queremos eliminar alguna fila concreta) podemos decir si el fichero tiene una fila con el nombre de las variables para que nos aísle esos valores.

Se puede indicar el formato de la variable en la que lo vamos a guardar, si es un cell o queremos guardar cada variable por separado. Hay que tener presente que en función del formato de salida es posible que algunas columnas no las lea ya que no puede establecer el formato de salida a ese tipo de datos. Por ejemplo, si establecemos que lo guarde en una matriz las columnas texto no las guardará. Por último, también podemos especificar que queremos que haga con las celdas que no puede importar para hacer un pretratamiento de muestras perdidas.

Una vez configurado, podemos importar directamente los datos para que nos lo guarde en una variable del workspace de Matlab o especificar que nos genere el código necesario para leer ese fichero desde un código personal.



Ejemplo

A continuación, se muestra el resultado de utilizar el asistente de `importdata` para generar una función que se puede llamar desde un script propio.

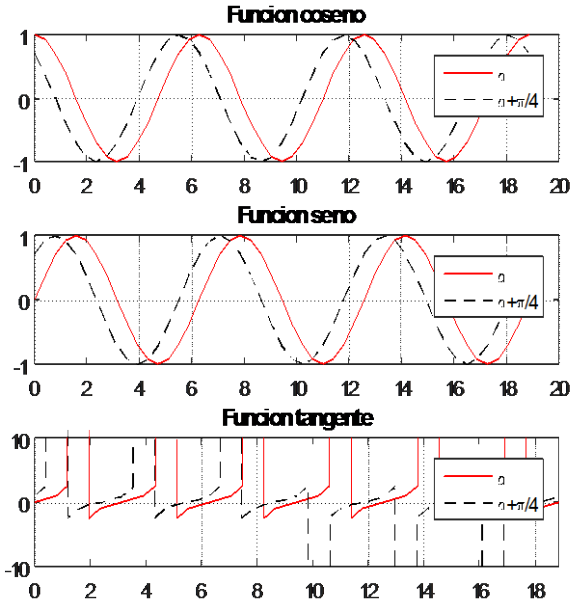
```
function Datasetcorto = importfile(filename, startRow, endRow)
%IMPORTFILE Import numeric data from a text file as a matrix.
%   DATASETCORTO = IMPORTFILE(FILENAME) Reads data from text file FILENAME
%   for the default selection.
%
%   DATASETCORTO = IMPORTFILE(FILENAME, STARTROW, ENDROW) Reads data from
%   rows STARTROW through ENDROW of text file FILENAME.
%
% Example:
%   Datasetcorto = importfile('Dataset_corto.txt', 2, 1598);
%
%    See also TEXTSCAN.

% Auto-generated by MATLAB on 2016/03/10 14:24:14

%% Initialize variables.
delimiter = ';';
if nargin<=2
    startRow = 2;
    endRow = inf;
end

%% Format string for each line of text:
%   column1: text (%s)
%   column2: text (%s)
%   column3: double (%f)
%   column4: double (%f)
%   column5: double (%f)
%   column6: double (%f)
%   column7: double (%f)
%   column8: double (%f)
%   column9: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%s%f%f%f%f%f%f%f%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1, 'Delimiter',
delimiter, 'HeaderLines', startRow(1)-1, 'ReturnOnError', false);
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-startRow(block)+1,
'Delimiter', delimiter, 'HeaderLines', startRow(block)-1, 'ReturnOnError', false);
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end

%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Create output variable
dataArray([3, 4, 5, 6, 7, 8, 9]) = cellfun(@(x) num2cell(x), dataArray([3, 4, 5, 6, 7,
8, 9]), 'UniformOutput', false);
Datasetcorto = [dataArray{1:end-1}];
```

Problemas

1. Gráficas I

Crear en una figura tres gráficas en las que se representen las relaciones trigonométricas del ángulo alpha y alpha+pi/4. Una gráfica será continua en color rojo y la otra discontinua en color negro. Etiquetar cada una de ellas convenientemente.

El resultado tiene que ser idéntico al que se muestra a continuación.



2. Gráficas II

Implementar un código que permita calcular el centro de gravedad de una serie de partículas introducidas por el usuario en un sistema bidimensional. El script se comportará de la siguiente manera:

- Pedir al usuario por pantalla el número de partículas que habrá en el sistema.
- Pedir por pantalla al usuario por cada partícula las coordenadas y la masa que se le asigna a la partícula.

```
>> Introduzca el número de partículas del sistema:
>> Partícula 1. Posición x:
>> Posición y:
>> Masa:
>> Partícula 2. Posición x:
>> Posición y:
```

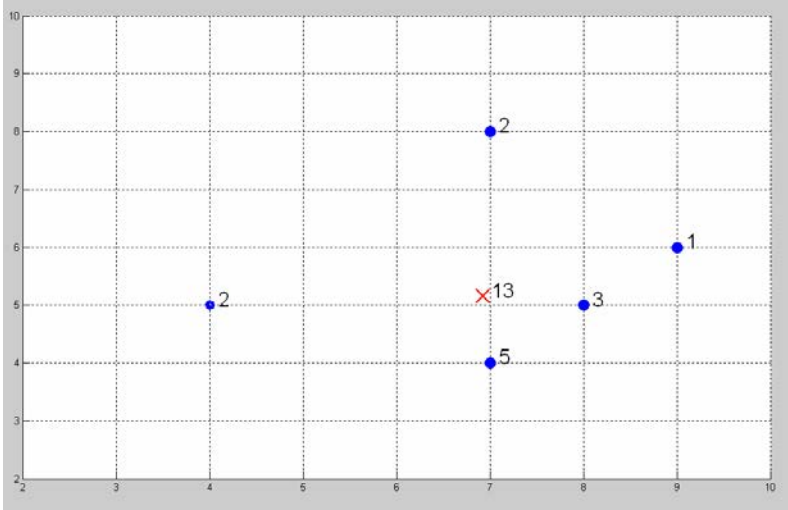
```
>> Masa:
```

……

- Calcular las coordenadas del centro de masas con la siguiente función:

$$Centro\ de\ masas = \frac{\sum_{i=1}^{n}(x_i, y_i)m_i}{\sum_{i=1}^{n} m_i}$$

- Dibujar cada una de las partículas en su posición correspondiente escribiendo también la masa que le ha sido asignada. Además dibujar también el centro de masas del sistema de partículas en su posición apropiada asignándole la masa total de todo el sistema. Las partículas se representarán mediante un círculo y el centro de masas del sistema mediante una 'x'. Utilizar la función `scatter`.



3. Salida de datos

Generar una matriz mágica de tamaño 6

1. Escribe su contenido en un fichero de texto (.txt) en el que los valores internos estén separados por tabulación. Utilizar para ello la función `fopen` y `fprintf`. El fichero tiene que llamarse 'matriz.txt'.
2. Escribe el contenido en un fichero .csv separado por ';' para que lo pueda leer Excel. Utilizar la función que se considere oportuna. El fichero tiene que llamarse 'matriz.csv'.

4. Entrada de datos

Leer el fichero de datos 'Maqueta.csv' e importar los datos mediante un script creado por medio del asistente de '**Import Data'** para que se importen como un cell de Matlab. El fichero está compuesto por las siguientes variables:

- DateStamp = Fecha
- Consigna = Númerico

- Nivel = Númerico
- Converbomba = Númerico
- Valvula = Númerico
- Salida = Númerico
- On/Off B1 = String Booleano
- On/Off B2 = String Booleano

Una vez que los datos estén leídos en un **cell** la variable Fecha hay que convertirla a **datenum** y los String Booleanos convertirlos a 1 y 0. Aprovechar el script que se puede generar con la función ‘import data’ para modificarlo y hacer las transformaciones de los datos para generar como variable final una matriz numérica con todos los datos.