TEMA 1

Fundamentos de MATLAB

1. Introducción. Programa MATLAB

Origen: MATLAB fue originalmente escrito por Cleve Moler, fundador de *MathWorks Inc*., con el objetivo de proporcionar un acceso fácil al *software* matricial desarrollado en los proyectos de UNIX LINPACK (de *LINear equations PACKage*) y EISPACK (de *EIgenvalue Subroutines PACKage*).

Versiones e historia: La primera versión de MATLAB, a finales de los años 70, se escribió en FORTRAN, siendo la única estructura de datos la matriz. De ahí el nombre del programa, cuyas siglas corresponden a *MATrix LABoratory*.

Posteriormente, en los años 80, MATLAB fue reescrito en C. La versión 3 para MS-DOS es de principios de los años 90. En 1993 aparece la versión 4, para *Windows* 3.11, que ya incorpora la primera versión del *Simulink*. La versión actual es la 7.x (las versiones más recientes también se identifican con el nombre del año, así, R2008a, R2008b, R2009a,…)

Hoy en día MATLAB es un estándar *de facto* en ingeniería y computación científica. Se celebran congresos monotemáticos sobre sus aplicaciones y distintas empresas e instituciones venden sus *toolboxes* como *third parties* o bien las publican en Internet con acceso libre. Para más información, se sugiere entrar en el sitio www.mathworks.com.

Características de MATLAB

- Es un lenguaje sencillo pero potente y rápido. En una sesión de trabajo típica, no hay que compilar o crear ejecutables y los ficheros son de texto, por lo que ocupan poca memoria.

- Muchas de las funciones matemáticas y de aplicación (análisis estadístico, optimización, diseño en ingeniería) están predefinidas y agrupadas en librerías comerciales (llamadas *toolboxes*). El usuario puede acceder a la mayor parte de estas funciones para modificarlas y/o crear las suyas propias.

- Presenta una gran capacidad para generar gráficos, en dos y tres dimensiones, y permite incorporar efectos y animaciones.

- Permite el desarrollo de aplicaciones complejas con ayuda del editor de ventanas, menús y controles de la utilidad GUI (*Graphics User Interface*).

- Puede intercambiar datos con otros lenguajes y entornos. Puede acceder a distintos dispositivos de *hardware* tales como tarjetas de sonido, tarjetas de adquisición de datos y DSPs (*Digital Signal Processors*).

Partes constitutivas: El programa MATLAB está constituido por:

- El entorno (ventanas, variables y ficheros)
- Los objetos gráficos (se verán con más detalle en el Tema 2)
- Un lenguaje propio de programación (se verá con más detalle en el Tema 3)

El presente tema se centra en el entorno de MATLAB.

2. El entorno de MATLAB

El entorno es el conjunto de herramientas que permiten trabajar como usuario o como programador. Permiten importar, procesar y exportar datos; crear y modificar ficheros; generar gráficos y animaciones; y desarrollar aplicaciones de usuario.

El entorno de MATLAB incluye las ventanas, las variables y los ficheros.

Ventanas: Son de diversos tipos. Las ventanas que forman el núcleo (*kernel*) del programa se organizan en el escritorio (*desktop*), pero en una sesión típica se abren y cierran gran número de ventanas secundarias correspondientes a figuras, editores de ficheros o de variables, aplicaciones diversas... Existen, además, ventanas específicas correspondientes a la ayuda y a las demostraciones (*helps* y *demos*).

Variables: Son objetos temporales (al cerrar el MATLAB se borran) y, durante la sesión en curso, se almacenan en el llamado *workspace*.

Ficheros: Son objetos permanentes (al cerrar el MATLAB no se borran). Aparte de los ficheros que conforman el núcleo básico del programa MATLAB, están los ficheros creados por el usuario y los ficheros comerciales correspondientes a las librerías, también llamadas *toolboxes*. Una *toolbox* no es más que un conjunto de ficheros desarrollados para alguna aplicación específica como, por ejemplo, la *Curve Fitting Toolbox*, diseñada para obtener expresiones matemáticas que ajusten curvas.

Existen además dos *toolboxes* "especiales" montadas jerárquicamente sobre MATLAB, tal y como muestra la Fig. 1: *Simulink*, para la simulación numérica de sistemas (dinámicos, de comunicación,…) y *Stateflow*, para la simulación de máquinas de estado.

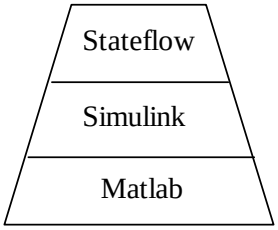


Fig. 1. Módulos de MATLAB

2.1 Ventanas y escritorio

Escritorio: Al abrir el programa MATLAB aparece un escritorio (*desktop*) como el mostrado en la Fig. 2 (versión 7.0.4)

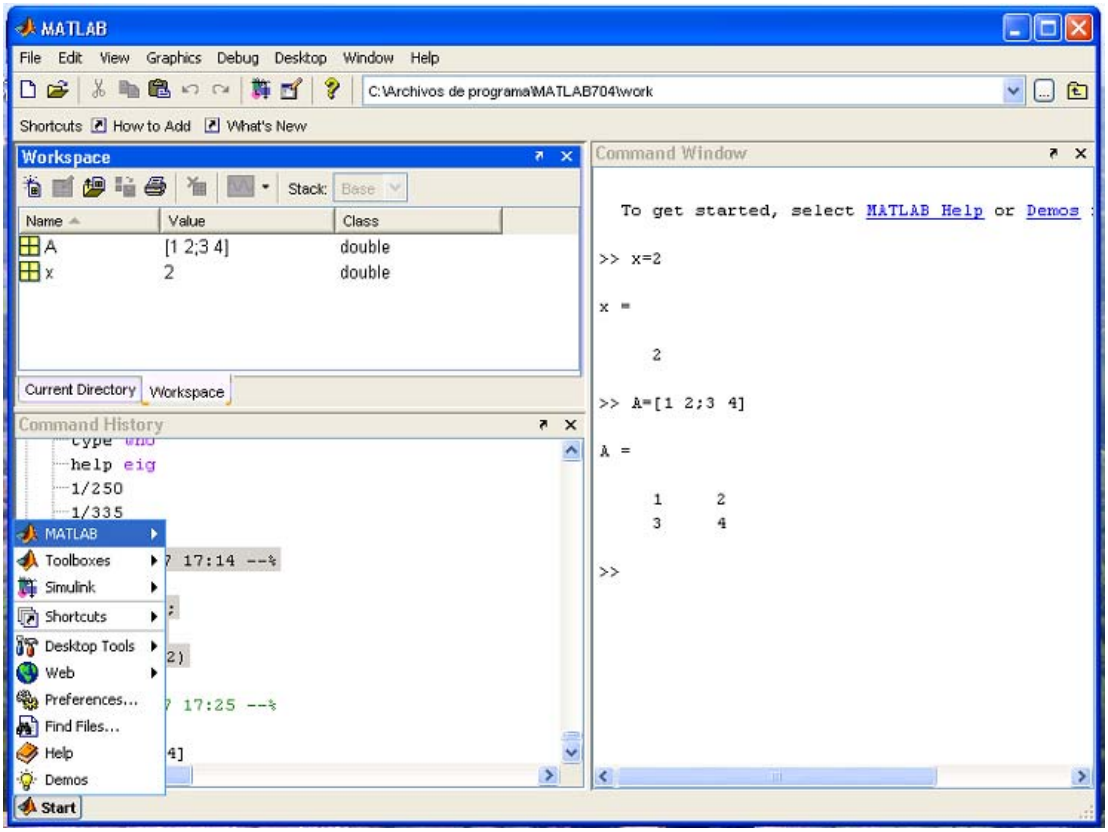


Fig. 2. Escritorio de MATLAB en la versión 7.0.4

Ventana de comandos: La ventana principal es la ventana de comandos (*Command Window*). En ella se escriben los comandos de MATLAB. Las instrucciones se escriben después del *prompt* `>>`. Desde esta ventana es posible ejecutar instrucciones del sistema operativo con sólo poner el signo `!` a continuación del *prompt* (por ejemplo: `>>!dir`). También es posible recuperar instrucciones ejecutadas con anterioridad con ayuda de la tecla <↑> (ello nos ahorra el tener que volver a teclearlas).

Otras ventanas del escritorio: Aparte de la ventana de comandos, las otras ventanas que forman el escritorio por defecto son:

- *Command History*: almacena las instrucciones introducidas en cada una de las sesiones anteriores de MATLAB, indicando fecha y hora de la sesión.
- *Current Directory*: es el listado de ficheros y carpetas en el directorio actual. El directorio seleccionado por defecto es el directorio <*work*>.
- *Workspace*: muestra las variables creadas, así como su tipo y su valor.

Otras ventanas posibles son:

- *Launch Pad*: es una ventana de la versión 6 que lista las diferentes *toolboxes* y que ya no aparece en la versión 7 puesto que su contenido es accesible desde el botón *Start*.
- *Help*: permite acceder a la documentación de ayuda, tanto del núcleo de MATLAB como de las diferentes *toolboxes*.
- *Profiler*: da información sobre los recursos consumidos por las diferentes funciones a fin de orientar en la depuración de los programas.

Notar que, al seleccionar cada una de las diferentes ventanas, las opciones de la barra de menús cambian.

Configuración de la apariencia del escritorio: Es posible cambiar la configuración del escritorio y sus ventanas. Se sugiere probar las opciones disponibles tanto en la barra de menús (*Desktop*) como en el botón *Start*. Si se quieren salvar los cambios en el escritorio (*Desktop* → *Save Layout*), éstos se guardan automáticamente en un fichero llamado `startup.m`. Si este fichero existe, es de los primeros en cargarse al iniciar MATLAB, junto con `matlabrc.m` (este fichero contiene los valores por defecto correspondientes a las fuentes, colores, dimensiones, etc. de los distintos elementos del programa). Las preferencias pueden cambiarse desde *File* → *Preferences…*

Ventanas secundarias: Aparte de las ventanas del escritorio, otras ventanas (tales como figuras, modelos de Simulink, etc.) se van abriendo y cerrando a lo largo de la sesión a medida que se ejecutan comandos o se activan controles (botones, barras deslizantes,…).

Algunos comandos: Algunas instrucciones útiles son:

- `clc`: para "limpiar" la ventana de comandos
- `close [all] [all hidden]`: : Para cerrar todas las ventanas de figuras, aplicaciones,…
- `exit`, `quit`: Para salir de MATLAB
- `diary [on/off]`: Para grabar una sesión en un fichero de texto
- `ver`: Para listar las *toolboxes* instaladas

Para más información se recomienda teclear `>>help nombre_comando`, por ejemplo, `>>help clc`.

Ejercicio 1. Familiarización con el entorno. Escritorio (*Desktop*)

1) Abrir el programa MATLAB y localizar las siguientes ventanas: *Command Window*, *Workspace*, *Current Directory*, *Command History*.

2) Añadir las ventanas *Help* y *Profiler* con ayuda de la opción `Desktop` del menú principal. Usar los botones  y  a fin de encajar/desencajar (*dock*/*undock*) ventanas al/del escritorio. Usar el botón  a fin de cerrar ventanas.

3) Ver qué otras opciones de apariencia de escritorio están disponibles en la opción `Desktop` del menú principal.

4) Volver a la apariencia por defecto: `Desktop` → `Desktop Layout` → `Default`. (Es la más recomendable para trabajar en la mayoría de los casos).

5) Seleccionar la ventana de comandos (*Command Window*). Para ello, hacer clic en su interior. Echar un vistazo a las opciones disponibles dentro del menú principal (`File`, `Edit`, `Debug`, `Desktop`, `Window`, `Help`). Intentar deducir para qué sirven.

6) Ídem con los botones de la barra de herramientas (`Toolbar`).

7) Ídem con las opciones del botón de inicio (`Start`). Notar que las mismas funciones se pueden ejecutar tanto desde la barra de menús, como desde la barra de herramientas, como desde el botón de inicio.

(Nota: Este ejercicio es de familiarización. No hay que entregarlo)

2.2 Variables y workspace

Workspace: Durante una sesión, las variables creadas por los comandos se guardan en el *workspace*, donde pueden ser modificadas y/o utilizadas en otros comandos. Este almacenamiento es temporal, sólo para la sesión en curso, con lo que al cerrar el MATLAB las variables son borradas.

Array Editor: Para ver el contenido de una variable `var1` basta con teclear su nombre en la ventana de comandos `>>var1`. También es posible visualizar su contenido en el *Array Editor*. Para abrirlo basta con ir a la ventana *Workspace* hacer doble clic en el icono de la variable en cuestión, 

. Otra forma de abrir el *Array Editor* es teclear `>>open var1` en la ventana de comandos.

Importación/exportación de variables: Es posible guardar las variables del *workspace* (todas o algunas de ellas) en un fichero para que no se pierdan al cerrar el MATLAB (función `save`) y así posteriormente, en otra sesión, poder cargarlas de nuevo en el *workspace* (función `load`). Los ficheros de datos en MATLAB tienen la extensión `*.mat`. Si al usar `save` no se indica un nombre al fichero de datos, MATLAB le asigna el nombre por defecto `matlab.mat`.

Nomenclatura de variables: Si no se asigna un nombre a una variable, ésta recibe el nombre automático de `ans` (viene de *answer*). La próxima variable que se cree sin nombre sustituirá a la primera puesto que también recibirá el nombre de `ans`.

MATLAB distingue entre mayúsculas y minúsculas. Así, `a` y `A` corresponden a dos variables diferentes.

Se recomienda no asignar a una variable un nombre que corresponda también a una función. Un error frecuente es crear una variable de nombre `axis` con lo que, al intentar ejecutar la función `axis`, ésta da error puesto que MATLAB considera que `axis` es una variable antes que una función. (En caso de duda se sugiere hacer `>>help nombre_función` para averiguar si una función con dicho nombre ya existe)

Otros comandos: Algunas instrucciones útiles son:

- `who`, `whos`: para ver las variables del *workspace*
- `size`: dimensiones de una matriz (`>>[nfil,ncol]=size(A)`)
- `length`: longitud de un vector (`>>long=length(v)`)
- `clear [all]`: para borrar variables
  (`>>clear` las borra todas, `>>clear var1` sólo borra la variable `var1`)
- `why`: para ver el sentido de humor que gusta a los autores de MATLAB

Como siempre, para más información se recomienda teclear `>>help nombre_comando`.

Ejemplo 1. Ventana de comandos y *workspace*

```
» x=[1 2 3]

x =

     1     2     3

» y=[4;5;6];
»
» who

Your variables are:

x         y

» A=[1 2 3;4 -5 6;7 8 9];
» q=y'

q =

     4     5     6

» x.*q

ans =

     4    10    18

» whos
  Name      Size         Bytes  Class

  A         3x3             72  double array
  ans       1x3             24  double array
  q         1x3             24  double array
  x         1x3             24  double array
  y         3x1             24  double array

Grand total is 21 elements using 168 bytes

»
```

Punto y coma

Transposición

Operación componente a componente

Ejercicio 2. Variables y *workspace* (I). Entrada y visualización de datos

1) Crear variables: En la ventana de comandos introducir un escalar, una matriz, una cadena de caracteres y un par de operaciones cualquiera. Por ejemplo:

```
>> x=2
>> A=[1 2;3 4;5 6];
>> A
>> s='hola'
>> a=2/0
>> 0/0
```

Notar cómo los comandos introducidos quedan registrados en la ventana *Command History*.

¿Para qué sirve el punto y coma?
¿MATLAB distingue entre mayúsculas y minúsculas?
¿Qué pasa al teclear `>> y=40.5` y, a continuación, `>> y=102.3`?
¿Qué significa `ans`?
¿Qué significa `NaN`?
Teclear ahora `>>who` y `>>whos`. ¿Qué hacen estas dos funciones?

2) Ventana *Workspace*: Seleccionar la ventana *Workspace* (para ello hacer clic en ella). Observar que dicha ventana contiene las variables creadas. Echar un vistazo a las opciones disponibles en la barra de menús (`File`, `Edit`, `View`, `Graphics`, `Debug`, `Desktop`, `Window`, `Help`) y en la barra de herramientas.

3) Ventana *Array Editor*: Al hacer doble clic en una variable o clicar en el icono  se abre la ventana *Array Editor*, que muestra el contenido de las variables. Echar un vistazo.

(Nota: Este ejercicio es de familiarización. No hay que entregarlo)

Ejercicio 3. Variables y *workspace* (II). Importación/exportación de variables.

1) Salvar variables: Para salvar todas las variables del *workspace* en un fichero `*.mat` hacer clic en el botón  . El nombre por defecto es `matlab.mat`. Notar que dicho fichero aparece en la ventana *Current Directory*.

2) Borrar variables: Seleccionar una o varias variables y borrarlas con ayuda del icono  .

3) Cargar variables: Al clicar en el botón  y seleccionar un fichero de datos (`*.mat`, `*.xls`, `*.txt`,...) se abre la ventana *Import Wizard*, que permite cargar todo o parte del contenido del fichero de datos al *workspace*. Cargar de nuevo todas las variables.

4) Funciones de la ventana de comandos (`clear, save, load`). Las operaciones anteriores pueden realizarse también mediante comandos. Por ejemplo:

Salvar las variables `a` y `A` en un fichero de nombre `hola.mat`:

```
>> save hola a A     o bien >> save('hola','a','A')
```

Borrar la variable `s`

```
>> clear s
>> who
```

Borrar todas las variables

```
>> clear, who
```

Cargar el fichero `hola.mat`

```
>> load hola, who
```

(Nota: Este ejercicio es de familiarización. No hay que entregarlo)

2.3 Ficheros y search path

Así como las variables son la información temporal de MATLAB, los ficheros constituyen la información permanente, la que no se borra al cerrar la sesión.

Tipos de ficheros: Hay distintos tipos de ficheros:

- Ficheros de datos: Son los ficheros con extensión `*.mat`, su formato puede ser diverso (*ascii*, binario,...) y se crean y cargan ejecutando las instrucciones `save` y `load` o bien con las opciones de menú y botones de la ventana *Workspace*.

  MATLAB también puede importar datos de otros tipos de ficheros (`*.txt`, `*.xls`,...). Si el fichero en cuestión está dentro del *Current Directory* basta con clicar sobre su icono a fin de abrir el *Import Wizard* (ver Ejercicio 6). También es posible acceder a los datos de un fichero a través de la ventana de comandos, por ejemplo, `>>importdata('prova.txt')`. Otras funciones relacionadas son `fopen`, `fread`. `fprintf`,...

- Ficheros-M: Son ficheros de texto, de extensión `*.m`, que contienen comandos de MATLAB escritos tal y como se introducirían en la ventana de comandos. Los ficheros-M pueden ser creados por el usuario y se guardan, por defecto, en el directorio de trabajo `<work>`.

  Los ficheros-M comerciales se venden agrupados en librerías o *toolboxes* (cada *toolbox* se almacena en un directorio) y también son modificables por el usuario.

  En líneas generales, cada comando o función de MATLAB corresponde a un fichero. Por ejemplo, la función `roots` corresponde al fichero `roots.m`.

- Ficheros *built-in*: Estos ficheros no pueden ser modificados por el usuario, sus extensiones son variadas (`*.dll`, `*.exe`, `*.mex`) y corresponden a los ficheros del kernel del MATLAB (por ejemplo, la función `who` no se puede editar. El fichero `who.m` solo contiene la ayuda de dicha función).

Nomenclatura de ficheros: MATLAB distingue entre mayúsculas y minúsculas. Por ejemplo, ver qué pasa al teclear `>>who` o `>>WHO`. En general, las funciones se invocan en minúsculas.

No son válidos nombres de fichero del tipo `ejercicio1-2.m`, puesto que entonces MATLAB interpreta que debe restar `2` de la variable `ejercicio1`.

Search path: Los ficheros creados por el usuario se guardan por defecto en el directorio `<work>`, pero es posible guardar ficheros en cualquier lugar del árbol de directorios. Para que MATLAB tenga en cuenta la nueva ubicación, hay que ampliar su *path*. Para ello,

seleccionar `File` → `Set Path...` en el menú principal. O bien clicar en el icono  que aparece junto al desplegable `Current Directory` en la barra de herramientas del escritorio.

También es posible usar la función `addpath` para indicarle a MATLAB que tenga en cuenta las funciones de directorios concretos durante la sesión en curso. Por ejemplo:

```
addpath c:\matlab6p5\work\mi_directorio
addpath d:\proyectos
```

Scripts y funciones: Los ficheros-M son de dos tipos: *scripts* o funciones.

- Los *scripts* no tienen argumentos de entrada ni de salida. Para ejecutar este tipo de ficheros basta con teclear su nombre (sin extensión) en la ventana de comandos o bien clicar en el icono  (*save and run*) de la barra de herramientas del editor de ficheros M. Para abrir el editor de ficheros-M clicar en el icono  de la barra de herramientas del escritorio.

- Las funciones sí tienen argumentos de entrada y/o de salida. El nombre de la función se escribe normalmente en minúsculas. Las variables de entrada van entre paréntesis ( ) y las variables de salida, si hay más de una, van entre corchetes [ ].

Otros comandos útiles: Algunas instrucciones útiles relacionadas con los ficheros son:

- `what`: indica qué ficheros se encuentran en el directorio actual.
- `which`: indica el *path* completo de una función, p. ej.: `>>which roots`
- `lookfor`: búsqueda de funciones (ver ejemplo siguiente)
- `type`: muestra el código de los ficheros M, p. ej.: `>>type roots`

Para más información se recomienda teclear `>>help nombre_comando`.

Ejemplo 2. Ficheros. Funciones

Si queremos obtener el determinante de una matriz y no nos acordamos de la función que lo obtiene podemos utilizar la función `lookfor` (ésta busca la palabra en cuestión en todos los nombres y ayudas de las funciones).

```
>> lookfor determinant
DET    Determinant.
DET    Symbolic matrix determinant.
DRAMADAH Matrix of zeros and ones with large determinant or inverse.
>>
>> help det

 DET    Determinant.
    DET(X) is the determinant of the square matrix X.

    Use COND instead of DET to test for matrix singularity.

    See also COND.

 Overloaded methods
```

```
    help sym/det.m

>>

>> det([1 3;-2 4])

ans =

    10

>> type det
det is a built-in function.
>>
```

Ejercicio 4. ***Toolboxes*** **y** ***search path***

1) Identificar cuál es el directorio de trabajo por defecto (ventana *Current Directory* o bien el desplegable de la barra de herramientas del escritorio).

2) Averiguar qué versión de MATLAB está instalada y qué librerías comerciales (*toolboxes*) están disponibles. Para ello teclear `>>ver` en la ventana de comandos.

3) Verificar que las *toolboxes* también son accesibles desde el botón de inicio (`Start`).

4) Observar cuál es el árbol de directorios de MATLAB con la ayuda de las opciones del menú `File` → `Set Path…` Localizar donde se guardan los archivos de las diferentes *toolboxes*.

5) Añadir un directorio cualquiera al *path* de MATLAB.

6) Con ayuda de `which`, averiguar en qué carpeta se encuentra el fichero correspondiente a la función `fminsearch`. Ídem con la función `roots`.

7) Ver para qué sirven las funciones anteriores (función `help`) y averiguar si su código es accesible (función `type`).

(Nota: Este ejercicio es de familiarización. No hay que entregarlo)

2.4 Helps y demos

Aparte de la ayuda típica del entorno de ventanas, MATLAB dispone de diversos niveles de ayuda mediante comandos. En el primero se ofrece una recopilación de todas las utilidades y *toolboxes* instaladas. Se solicita con el comando:

```
» help
```

En un segundo nivel es posible obtener un listado, por *toolboxes*, de los nombres de todas las funciones, acompañado de una breve explicación sobre lo que hace cada una de ellas:

```
» help nombre_toolbox        (» help stats)
```

Finalmente, para solicitar una información más completa sobre el uso y sintaxis de cada función, es preciso teclear:

```
» help nombre_función          (» help bode)
```

Por su lado, las demostraciones (demos) son un conjunto de *scripts* existentes dentro del entorno MATLAB cuya misión es ofrecer una perspectiva de las capacidades del programa por medio de la ejecución automática de sus funciones más significativas.

```
» demo                         (» help ctrldemos)
```

Ejercicio 5. Helps y demos
Elegir una *toolbox* cualquiera y, con ayuda del botón `Start`, entrar en la ventana de *Help* a fin de ver cómo se organiza la ayuda.
Ejecutar asimismo alguna de las demos tanto del programa MATLAB básico como de las *toolboxes* que sean de su interés.

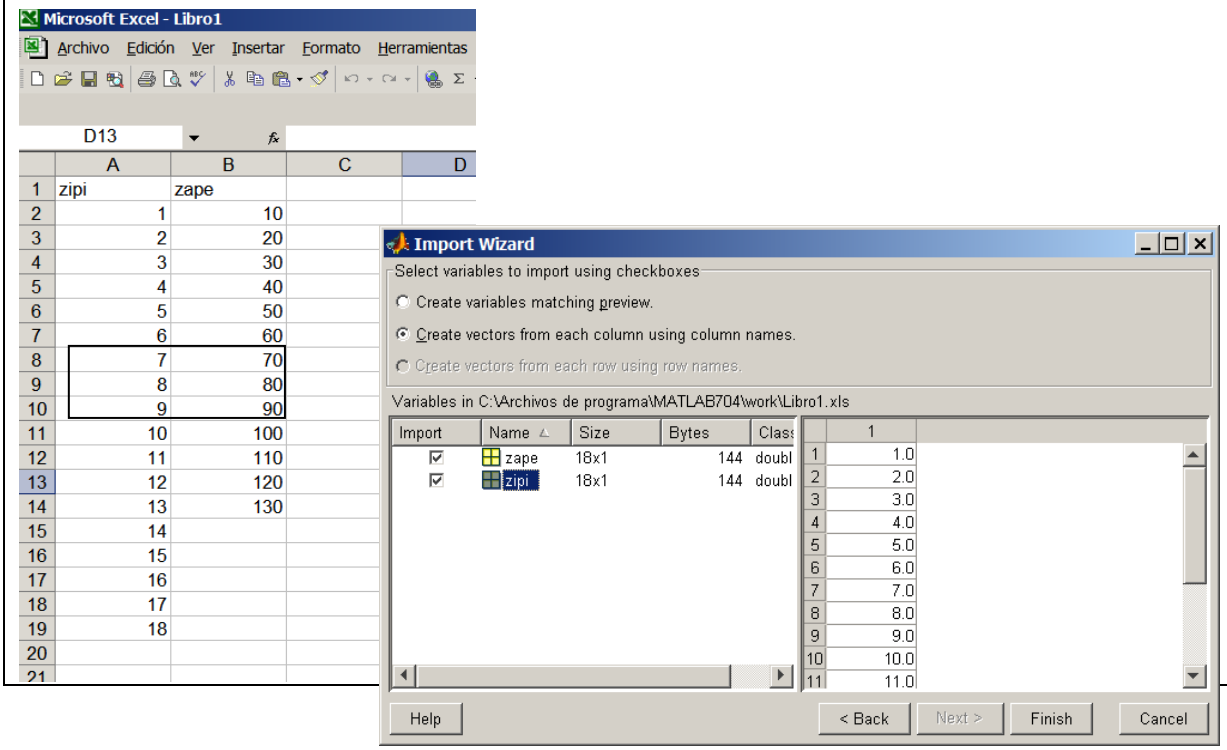
(Nota: Este ejercicio es de familiarización. No hay que entregarlo)

2.5 Relación con EXCEL

MATLAB puede intercambiar datos con otros programas de Windows. El siguiente ejercicio muestra la relación con EXCEL.

Ejercicio 6. Variables y *workspace* (III). Relación con Excel
Crear un fichero Excel sencillo como el de la figura siguiente (`Libro1.xls`) y guardarlo en el directorio de trabajo de MATLAB, `<work>`.

1) Importar datos desde la ventana *Workspace*: Para importar los datos de `Libro1.xls` basta con abrir el *Import Wizard*. Para ello hacer doble clic sobre el nombre `Libro1.xls` en la ventana *Current Directory* o bien clicar en el icono de la ventana *Workspace*.



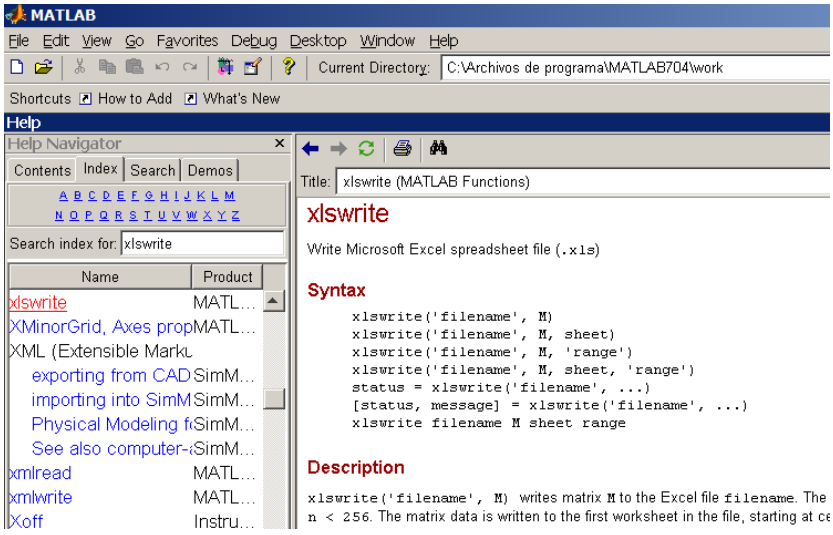
2) Importar datos con ayuda de la función `xlsread`: Para importar tan sólo la matriz formada por los datos recuadrados y almacenarla en el *workspace* con el nombre "chorizo" teclear:

```
>> chorizo=xlsread('Libro1','A8:B10')
chorizo =
     7    70
     8    80
     9    90
```

3) Exportar datos con ayuda de la función `xlswrite`: En primer lugar, crear la siguiente variable de tipo *cell array*:

```
>> datos={'chorizo','mortadela';1 2;3 4}
datos =
    'chorizo'    'mortadela'
    [      1]    [        2]
    [      3]    [        4]
```

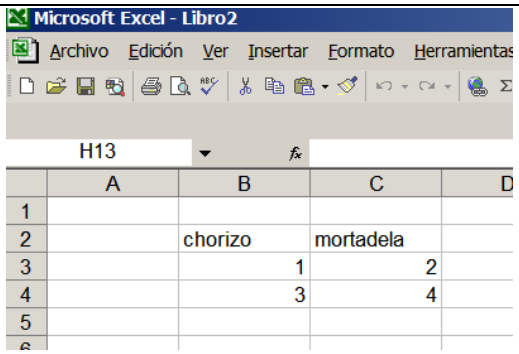
Usar `xlswrite` para guardar la variable `datos` en un fichero `Libro2.xls` en una hoja (inexistente aún) de nombre `Embutidos` y a partir de la celda `B2`. Usar la ventana de *Help* de MATLAB para averiguar cuál es la sintaxis de uso de la función:



Una vez consultada la ayuda, vemos que tenemos que teclear:

```
>> xlswrite('Libro2',datos,'Embutidos','B2')
Warning: Added specified worksheet.
> In xlswrite>activate_sheet at 254
  In xlswrite at 212
```

El resultado es:



(Nota: Este ejercicio es de familiarización. No hay que entregarlo)

3. Lenguaje MATLAB

3.1 Símbolos y comandos básicos

Introducción de matrices: En el programa MATLAB la unidad computacional básica la constituyen las matrices rectangulares con elementos reales o complejos. Las matrices cuadradas, los vectores y los escalares se consideran como casos particulares.

Entrada de matrices: En la expresión de una matriz los elementos han de estar encerrados entre corchetes, `[ ]`. Los elementos de una misma fila se separan por comas o espacios en blanco mientras que el cambio de fila se indica por medio de puntos y coma o pulsando la tecla ↵.

Ejemplo: Introducción de una matriz:

```
» A=[1 2;3 4]      (pulsar ↵)
A =
      1   2
      3   4
```

Punto y coma: Suprime la presentación por pantalla del resultado numérico:

```
» A=[1 2;3 4];     (pulsar ↵)
»
```

Referencia a submatrices: Es posible extraer submatrices de matrices. Por ejemplo, dada una matriz **A**, si queremos tomar la submatriz **B** formada por las filas 3 a 5 y las columnas 4 a 7 de **A**, se puede hacer

`»B=A(3:5,4:7)` o bien `»B=A([3,4,5],[4 5 6 7])`

Si se quiere extraer la submatriz **C** formada por las filas 1, 3 y 4 de **A**, las instrucción es:

```
»C=A([1 3 4],:)
```

Los dos puntos, **:**, después de la coma que separa la referencia de filas de la referencia de columnas indican que hay que tomar todas las columnas.

Símbolos especiales: MATLAB dispone de diversas variables predefinidas. El número complejo $\sqrt{-1}$ puede representarse indistintamente por `i` o por `j`, `Inf` representa al $+\infty$ y `pi` al número π. `NaN` (*Not-a-Number*) es el resultado de operaciones no definidas.

Símbolos especiales: Entrada de una matriz rectangular:

```
» A = [sin(pi/2) -4*j 9;sqrt(2) 0/0 log(0)]

Warning:    Divide by zero
Warning:    Log of zero

A =
      1.0000            0-4.0000i         9.0000
      1.4142          NaN                   -Inf
```

Ans: Cuando no se especifica ninguna variable de salida, MATLAB responde con la variable de defecto `ans`:

```
» 12.4/6.9   (pulsar ↵)
ans =
      1.7971
```

Interrupción de la ejecución: Pulsar la combinación de teclas `<Control>` `<c>`.

Estructura de los comandos: MATLAB es un programa interactivo en el sentido que establece un "diálogo" con el usuario mediante un lenguaje de comandos. Todo comando introducido tiene como respuesta, por parte de MATLAB, su ejecución o, en su defecto, el mensaje de error correspondiente.

Se pueden entrar varios comandos en una sola línea. Basta con separar las instrucciones con comas o puntos y coma.

Los comandos pueden ser:

- Operaciones matemáticas
- Invocaciones a *scripts* (basta con teclear el nombre del *script*, sin extensión)
- Invocaciones a funciones (es decir, con argumentos de entrada y/o salida)

Llamada a una función: Los argumentos de entrada van entre paréntesis y, los de salida, si son más de uno, van entre corchetes.

```
» [vars_salida]=nombre_función(vars_entrada)   (pulsar ↵)
» nombre_función(vars_entrada)                 (pulsar ↵)
```

Siempre que no se le indique lo contrario, MATLAB mantendrá el valor de las variables que vayan siendo creadas, las almacenará en el *workspace* y las admitirá como parámetros de nuevos comandos del usuario.

Ejemplo 3. Sintaxis de las funciones

Aunque los nombres de las variables de entrada y salida de las funciones son libres, el orden es importante a la hora de interpretar el resultado. Por ejemplo, la función `eig` es la encargada de calcular los autovalores y los autovectores de una matriz.

Si se invoca `eig` con un único argumento de salida (o sin argumento de salida), el resultado son los autovalores:

```
>> A=[-3 2;0 1];
>> eig(A)
ans =
    -3
     1
```

Si se invoca `eig` con dos argumentos de salida, el primero de ellos es una matriz conteniendo los autovectores y el segundo la forma de Jordan (conteniendo los autovalores en la diagonal):

```
>> [autovect,diag]=eig(A)

autovect =
    1.0000    0.4472
         0    0.8944

diag =
    -3     0
     0     1

>>
```

Puesto que la mayoría de las funciones generan diferentes resultados según sea el número de parámetros de entrada y/o salida, se recomienda consultar la ayuda de las funciones antes de utilizarlas por primera vez, `>> help eig`.

Generación de matrices especiales: Es posible generar matrices (o vectores) cuyos elementos sean todos igual a "1" (función `ones`) o todos igual a "0" (función `zeros`). También es posible generar matrices de elementos aleatorios (función `rand` para distribución uniforme o función `randn` para distribución normal). Para generar la matriz identidad la función es `eye`.

Generación de vectores con componentes equiespaciadas: Las dos funciones básicas son `linspace` y los dos puntos, `:`. Notar que con la primera tenemos control sobre el número de puntos mientras que, con la segunda, tenemos control sobre el paso entre puntos.

```
>> x=linspace(0,1,5)   %genera 5 puntos entre 0 y 1
x =
         0    0.2500    0.5000    0.7500    1.0000

>> x=linspace(0,1);   %100 puntos (defecto) entre 0 y 1
```

```
>> y=0:5
y =
     0     1     2     3     4     5

>> y=0:0.5:5
y =
  Columns 1 through 7
    0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000
  Columns 8 through 11
    3.5000    4.0000    4.5000    5.0000
```

Introducción de polinomios y funciones de transferencia: Los polinomios se introducen como vectores fila cuyos elementos son los coeficientes del polinomio.

Por ejemplo, el polinomio $s^4 + 5s^2 + 3s - 10$ se introduce como

```
» polinomio=[1 0 5 3 -10];
```

Notar que el coeficiente correspondiente a s^3 es cero.

Las funciones de transferencia se introducen tecleando por separado los polinomios del numerador y del denominador. Así, la función

$$H(s)=\frac{s}{s^3+2s+5}$$

habrá que definirla por medio de las instrucciones:

```
» num = [1 0];
» den = [1 0 2 5];
```

También es posible agrupar dicho numerador y denominador en una única variable de clase `tf` (*transfer function*).

```
»  H=tf(num,den)
Transfer function:
      s
-------------
s^3 + 2 s + 5
```

Producto de polinomios: Para multiplicar polinomios se puede utilizar la instrucción de convolución, `conv`. Sólo admite dos argumentos de entrada cada vez, pero es posible anidar diferentes funciones. Para multiplicar $(s+2)\cdot(s^2+4s)\cdot(3s^2+7s+2)$, se hace:

```
>> conv([1 2],conv([1 4 0],[3 7 2]))
ans =
     3    25    68    68    16     0
```

El resultado se interpreta como: $3s^5+25s^4+68s^3+68s^2+16s$.

Otros tipos de datos (cell y struct): Si se desea guardar en una única variable datos de diversos tipos, se pueden usar variables de tipo *cell array* o *struct*.

Por ejemplo, para guardar la temperatura máxima de varios días junto con las etiquetas “día” y “temp máx” se puede usar una variable tipo `cell`. Teclear `>>help cell` para más información.

```
>> calor={'día','temp máx';1,25.7;2,25.5;3,25.4}

calor =

    'día'    'temp máx'
    [  1]    [ 25.7000]
    [  2]    [ 25.5000]
    [  3]    [ 25.4000]
```

Para organizar distintos tipos de información por medio de campos se puede utilizar una variable tipo `struct`. Teclear `>>help struct` para más información. Por ejemplo, si se

quiere guardar la entrada y la salida de cierto experimento así como la fecha de realización, se puede hacer:

```
>> x=0:10;
>> y=2*x;

>> experimento.entrada=x;
>> experimento.salida=y;
>> experimento.fecha='1mar10';
>> experimento

experimento =
    entrada: [0 1 2 3 4 5 6 7 8 9 10]
     salida: [0 2 4 6 8 10 12 14 16 18 20]
      fecha: '1mar10'
```

Formato: La instrucción `format` permite cambiar el formato numérico con que MATLAB presenta los resultados. Algunos formatos son:

`short`: coma fija con 4 decimales (defecto)
`long`: coma fija con 15 decimales
`bank`: dos cifras decimales
`rational`: expresa los números racionales como cocientes de enteros

Un par de instrucciones útiles para la presentación de resultados son `disp` y `sprintf`:

```
>> disp 'hola'
hola
>> disp('hola')
hola
>> disp(sprintf('\n\t\t\tTabla 1'))

             Tabla 1
```

Teclear `>>help nombre_comando` para más información.

3.2 Tablas de operadores y funciones matemáticas

MATLAB puede utilizarse como "calculadora". Para ello dispone de los operadores matemáticos y funciones elementales y trigonométricas mostrados en las Tablas 1.1, 1.2 y 1.3.

Operadores matemáticos	
+	suma
-	resta
*	producto
/	división por la derecha
\	división por la izquierda
^	potencia
'	transpuesta conjugada

Tabla 1.1

Funciones elementales	
abs	valor absoluto
angle	fase (argumento)
sqrt	raíz cuadrada
real	parte real
imag	parte imaginaria
conj	complejo conjugado
exp	exponencial base *e*
log	logaritmo natural
log10	logaritmo base 10

Tabla 1.2

Funciones trigonométricas	
sin	seno
cos	coseno
asin	arcseno
acos	arcoseno
tan	tangente
atan	arctangente
sinh	seno hiperbólico
cosh	coseno hiperbólico
tanh	tangente hiperbólica

Tabla 1.3

Punto antes de un operador matemático: Poner un punto antes de un operador entre dos vectores o matrices indica que la operación debe realizarse componente a componente. Por ejemplo, `A*B` es el producto matricial entre las matrices **A** y **B**. Si se teclea `C=A.*B` (siendo **A** y **B** matrices de igual dimensión), el resultado es una matriz **C** cuyas componentes c_{ij} corresponden a $c_{ij}=a_{ij}\times b_{ij}$.

3.3 Operadores lógicos

La ejecución de las instrucciones en MATLAB puede someterse a un proceso de programación, tal como ejecutar un determinado grupo de funciones sólo si se satisface cierta condición booleana, realizar bucles, etc.

Álgebra de Boole: Los operadores relacionales son `==` (igualdad), `<`, `<=` (menor, menor o igual) y `>`, `>=` (mayor, mayor o igual). También existen funciones que relacionan variables, por ejemplo, `gt(i,1)` indica si el valor de la variable "i" es mayor que "1" (el nombre de la función corresponde a *greater than*).

Las condiciones booleanas se suelen expresar entre paréntesis, por ejemplo `(a==2)`. Si esta condición es cierta el valor de salida es "1" y si es falsa el valor de salida es "0".

Los operadores lógicos son `&` (*and*), `|` (*or*) y `~` (*not*); en MATLAB `xor` no es un operador sino una función.

Control de flujo: Las instrucciones básicas para el control del flujo de instrucciones son `for...end`, `if...elseif...else...end`, `while...end`, `switch... case... otherwise ...end`

Ejemplo: Suma de los elementos de un vector mediante un bucle

```
»x=[1 2 3 4 5];
»suma=0;
»for i=1:length(x)
        suma=suma+x(i);
end
»suma
suma =
      15
```

Nota: El ejemplo es sólo para ilustrar el uso del bucle. Para una operación de este tipo basta con teclear: `>>sum([1 2 3 4 5])`.

Paréntesis y corchetes: Notar que, en el ejemplo anterior, para acceder a la componente “i” del vector `x` se han utilizado los paréntesis. Los paréntesis `()` sólo se usan para indexar, para establecer la prioridad en las operaciones matemáticas y para contener los parámetros de entrada de las funciones. Los corchetes `[]` se usan para definir vectores y matrices y para contener las variables de salida en las funciones que dan más de una.

Instrucciones útiles: Dos instrucciones muy útiles son `size` (devuelve las dimensiones de una variable) y `length` (si la variable es un vector, devuelve la longitud de éste), puesto que permiten verificar que los comandos se están ejecutando correctamente. En programación, es importante la función `find` (que devuelve el índice de las componentes que, dentro de un vector o matriz, satisfacen una determinada condición booleana).

3.4 Creación de funciones por parte del usuario

La construcción por parte del usuario de funciones sólo tiene sentido en aquellas situaciones en las que se vaya a repetir el uso de una estructura en la que tan sólo varíen los parámetros y éstos puedan ser introducidos "exteriormente". El nombre del fichero-M asociado debe coincidir con el nombre de la función.

Llamada: La llamada a una función se realiza siguiendo el esquema:

```
»[variables_salida] = nombre_función (variables_entrada)
```

Estructura: Un fichero-M de función se estructura en tres partes:

- Cabecera: `function` [args de entrada]=nombre_función(args de salida)

```
function [seno,coseno,tangente]=func1(ang)
```

- Comentarios de ayuda: Esto es opcional. Los comentarios de ayuda es lo que aparece en pantalla al teclear `>>help nombre_función`.

```
% FUNC1 Función de prueba
% [seno,coseno,tangente]=func1(ang) calcula el seno,
% el coseno y la tangente del ángulo indicado
% por la variable 'ang'
```

- La colección de instrucciones

```
seno=sin(ang);
coseno=cos(ang);
tangente=tan(ang);
```

Ejercicio 7. Creación de funciones por parte del usuario
1) Abrir el editor de ficheros M (clicar en ) y escribir las instrucciones del ejemplo anterior.
2) Guardar el fichero llamándolo igual que a la función, `func1.m` en nuestro caso.
3) Teclear `>>help func1` en la ventana de comandos.
4) Llamar la función `func1` para diversos valores de `ang` y ver los resultados.

(Nota: Este ejercicio es de familiarización. No hay que entregarlo)

4. Generación de gráficos en MATLAB

Para generar un gráfico simple en MATLAB se necesita

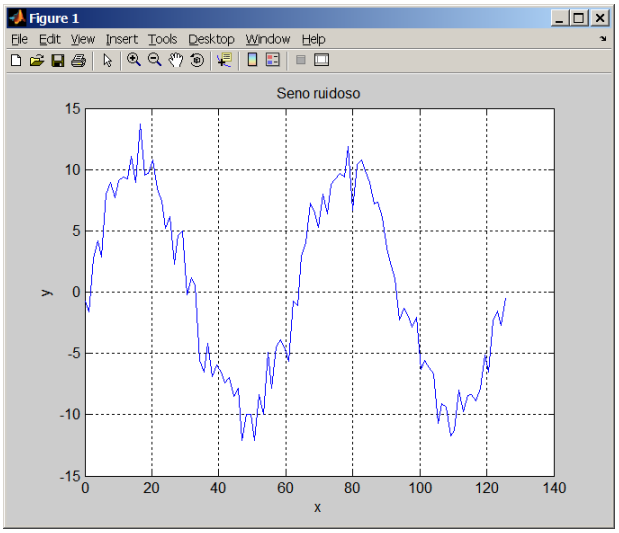
1. Generar (o cargar) un vector `x` conteniendo los valores del eje de abscisas.
2. Generar (o cargar) un vector `y` conteniendo los valores del eje de ordenadas (`y` deberá tener la misma longitud que `x`)
3. Ejecutar una instrucción gráfica, por ejemplo, `>>plot(x,y)`.
4. Opcionalmente, poner una rejilla (`>>grid`), etiquetar los ejes (`xlabel`, `ylabel`), poner un título (`title`)

Nota: Para superponer dos gráficos, teclear `>>plot(x1,y1,x2,y2)`.

Ejemplo 4. Gráfico simple

A continuación se generan y representan 2 periodos de un seno de frecuencia 0.1rad/s de amplitud 10 y contaminado por ruido blanco Gaussiano aditivo de media nula e intensidad (varianza) 3:

```
>> x=linspace(0,2*2*pi/0.1);
>> y=10*sin(0.1*x)+randn(size(x))*sqrt(3);
>> plot(x,y)
>> grid,xlabel('x'),ylabel('y'),title('Seno ruidoso')
```



Para copiar la figura en un documento (*word*, por ejemplo) basta con seleccionar *Edit* → *Copy Figure* en la barra de menús de la figura.

5. Uso de las *toolboxes*

Para saber de qué *toolboxes* se dispone hay que teclear `>>ver`. Aquí se ilustra el uso de algunas funciones de la *Signal Processing Toolbox*.

Ejemplo 5. Espectro de una señal. Densidad espectral de potencia

Se desea obtener y representar las densidades espectrales de potencia (PSDs) de las siguientes señales:

1) Combinación de senoides: $x(t) = 0.5\,\mathrm{sen}(2\pi 3t) - 0.2\,\mathrm{sen}(2\pi 9t)$.

2) Tren de pulsos de frecuencia 0.3 (T = 3.33s) y duración del pulso 0.3 (ciclo de trabajo del 10%).

Solución:

En primer lugar hay que generar un vector temporal cuyo número de muestras sea potencia de 2, como por ejemplo,

```
t=0:0.01:10.23;
```

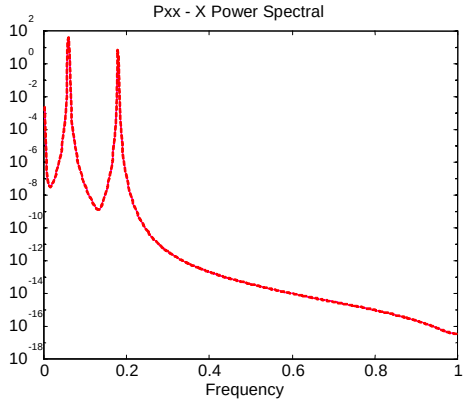
que tiene 1024 muestras. En este vector es donde se establece el periodo de muestreo, $T_s = 0.01s$, con lo que la frecuencia de muestreo es $f_s = \frac{1}{T_s} = 100Hz$.

A continuación se generan y representan las señales:

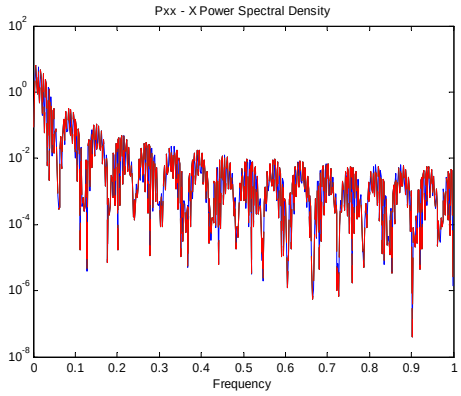
```
x1=0.5*sin(2*pi*3*t)-0.2*sin(2*pi*9*t); plot(t,x1)
x2=0.5*square(2*pi*0.3*t,10)+0.5;       plot(t,x2)
```

y, finalmente, se aplican las funciones **`spectrum`** (para generar el vector que contendrá las muestras de la PSD) y **`specplot`** (para su representación):

```
px1=spectrum(x1,1024);specplot(px1)
px2=spectrum(x2,1024);specplot(px2)
```



PSD de la combinación de senos.



PSD del tren de pulsos.

Notar que **`specplot`** sólo representa el espectro hasta la frecuencia $f_s/2$ y que además la normaliza de manera que ésta sea 1. Así pues, con ayuda de la función **`ginput`** es posible comprobar que, para el caso de las senoides, las frecuencias normalizadas son $f_{1n} = 0.06$ y $f_{2n} =$

0.18, las cuales una vez desnormalizadas (multiplicándolas por $f_s/2 = 50$) son, efectivamente, $f_1 = 3$ y $f_2 = 9$.

Ejemplo 6. Autocorrelación del ruido blanco

Se trata de obtener y representar la autocorrelación de una señal de ruido blanco Gaussiano de media cero y varianza 4.

Solución:
En primer lugar hay que generar un vector temporal cuyo número de muestras sea potencia de 2, como por ejemplo,

```
Ts=0.01;fs=1/Ts;fn=fs/2;
t=0:Ts:10.23;
```

que tiene 1024 muestras. Con esta elección, se establece un periodo de muestreo de $T_s = 0.01s$, y, por tanto, una frecuencia de muestreo, $f_s = \frac{1}{T_s} = 100Hz$ y una frecuencia de Nyquist, $f_N = \frac{f_s}{2} = 50Hz$.

A continuación se genera la señal de ruido:

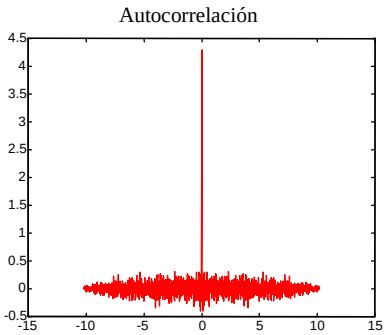
```
n=randn(size(t))*sqrt(4);plot(t,n)
```

y se comprueba el valor de la media y la varianza

```
mean(n),cov(n)
```

La autocorrelación se obtiene con la función **`xcorr`** y tiene el doble de muestras menos una que la señal de ruido. Verificar, con ayuda de **`ginput`** o **`max`**, que el valor en el origen corresponde a la potencia de la señal.

```
rn=xcorr(n,'biased');
plot(-10.23:0.01:10.23,rn)
```



---

6. Comentarios finales y entrega de prácticas

Una de las características de MATLAB es que se pueden hacer las mismas cosas de formas muy distintas. En estos apuntes se han presentado los comandos (de texto) más importantes y útiles. Las acciones de algunos de estos comandos también pueden ser ejecutadas desde los menús y barras de herramientas pero, puesto que el entorno de ventanas es autoexplicativo, no se ha considerado necesario describir lo que hace cada submenú o botón. De todas formas se recomienda, antes de empezar a trabajar, echar un vistazo a todas las opciones y botones para hacerse una idea de todas las posibilidades del entorno.

Instrucciones para la entrega de las prácticas: En cuanto a la entrega de las prácticas, para cada una habrá que subir un único fichero al Moodle.

El nombre del fichero será vuestro apellido seguido del número de la práctica, por ejemplo, `fernandez_p1.pdf`. Este fichero contendrá tanto las instrucciones que resuelven la práctica como los resultados obtenidos debidamente comentados. A continuación se muestra una propuesta de presentación:

Práctica 1. Fundamentos de MATLAB

Nombre:

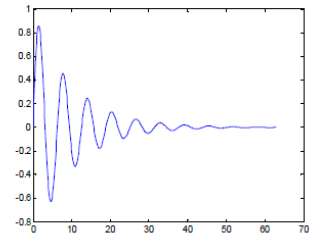
Fecha:

Ejercicio 1. Representación de una sinusoide amortiguada.

Instrucciones:

```
x=linspace(0,10*2*pi,500);
y=sin(x).*exp(-0.1*x);
plot(x,y)
```

Resultados:



Comentarios:

Ejercicio 2. Operaciones con matrices

Instrucciones:

Inversa:

```
>> A=[1 2 3;4 -5 6;7 8 9];
>> inv(A)
```

Resultados:

```
ans =
   -0.7750    0.0500    0.2250
    0.0500   -0.1000    0.0500
    0.5583    0.0500   -0.1083
```

Valores singulares:

```
>> M=[250 1;1 0];
>> svd(M)
```

Resultados:

```
ans =
  250.0040
    0.0040
```

Comentarios: La matriz M está muy cerca de la singularidad puesto que uno de sus valores singulares es muy pequeño. Ello indica que una pequeña variación de los coeficientes, p. ej.: si a_{22} pasa a valer 0.004, puede hacer M singular (su determinante valdrá 0).