Curso MATLAB

Introducción

El conocimiento de otros programas de cálculo numérico o simbólico (i.e. Maple, Mathematica, Octave, etc.) o el conocimiento de lenguajes de programación (i.e. C, Pascal, Fortran, etc.) son antecedentes que pueden facilitar el aprendizaje del MATLAB.

Es requisito indispensable para este curso el conocimiento de:

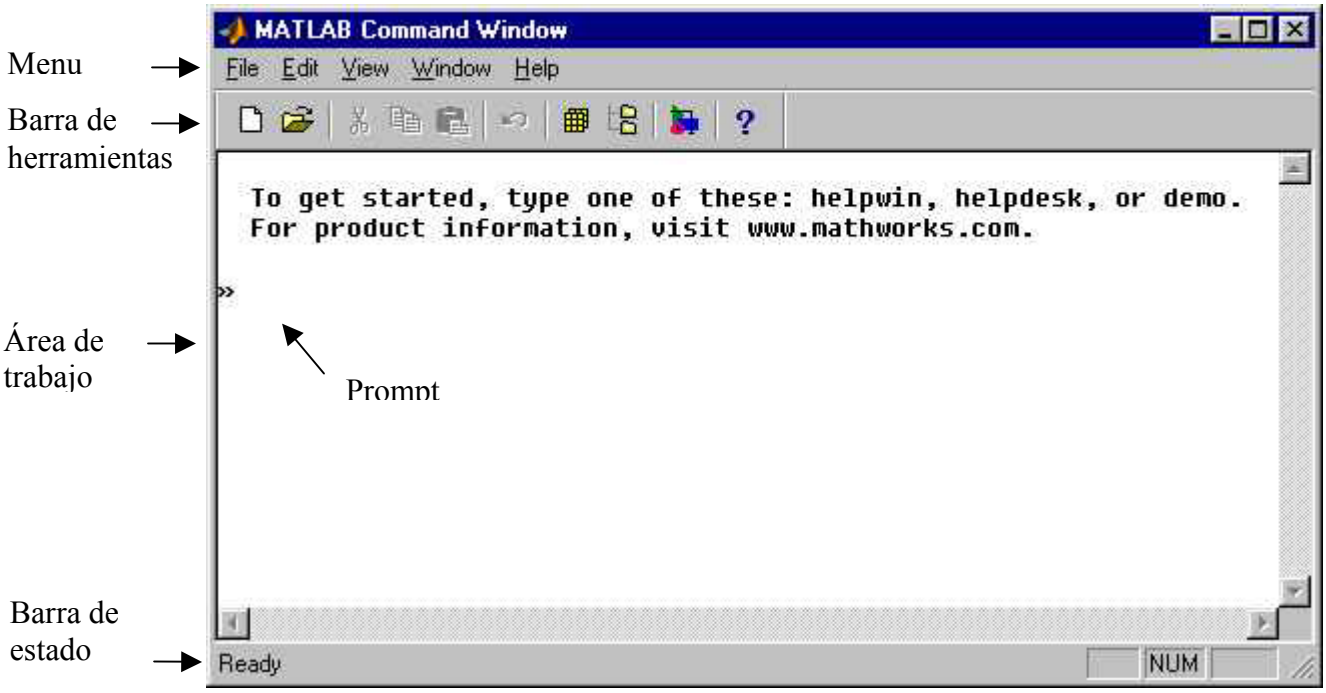
- Manejo del sistema operativo (Windows).
- Conocimientos de Álgebra, Análisis Matemático y Métodos numéricos.

Invocación de la aplicación.

Partes de la interfase de usuario:

- Menú
- Barra de herramientas
- Barra de estado
- **Área de trabajo**.

Las tres primeras se verán más adelante. Por ahora sólo se usará el **Área de trabajo** que es una interfase de usuario (shell) en modo texto (como el DOS).



Existe un "prompt" representado con el símbolo "»" a la izquierda del Área de trabajo. El último "prompt" (el inferior) es el activo, en ella aparece el cursor y sólo en ella se puede escribir. En ella se escriben las expresiones del MATLAB, que pueden ser expresiones aritméticas que involucran operadores, o funciones o la combinación de ambos. La expresión se "ejecuta" cuando se presiona la tecla "enter".

Con las teclas “flecha arriba”y “flecha abajo” se pueden volver a expresiones escritas con anterioridad (cómo el DOSKEY del DOS).

Variables.

Las variables son por defecto:

- De punto flotante (tipo double),
- complejos, y
- matriciales.

Entoces se pueden representar matrices de números complejos. Matrices de números reales (la parte imaginaria es cero), Vectores (marices de una sóla fila o columna), Escalares (matrices de 1x1), etc.

Los nombres de variables consisten de una letra seguida por letras, dígitos ó el caracter de subrayado (_). MATLAB sólo usa los primeros 31 caracteres del nombre. También distingue entre minúsculas y mayúsculas.

Asignación de un escalar real:

$a \leftarrow 3.15$

```
» a = 3.15
```

Asiganación de un escalar complejo:

$a \leftarrow 3.15 + i5.8$

```
» a = 3.15+5.8i     ó     » a = 3.15+5.8*i     ó     » a = 3.15+i*5.8
» a = 3.15+5.8j     ó     » a = 3.15+5.8*j     ó     » a = 3.15+j*5.8
```

Asignación de un vector fila:

$a \leftarrow \begin{bmatrix} 3 & 5 & 8 \end{bmatrix}$

```
» a = [3 5 8]
» a = [3, 5, 8]
```

Asignación de un vector columna:

$$b \leftarrow \begin{bmatrix} 3 \\ 5 \\ 8 \end{bmatrix}$$

```
» b = [3; 5; 8]
```

Asignación de una matriz:

$$c \leftarrow \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
» c = [1, 2, 3;  4, 5, 6; 7, 8, 9]
```

Acceso a elementos de un vector o matriz:

Por ejemplo para asignar a d el segundo elemento del vector a, ($d \leftarrow a_2$):

```
» d = a(2)
```

En el caso de matrices, si por ejemplo se desea asignar a la variable d el escalar de la fila 2 y columna 1 de la matriz c, ($d \leftarrow c_{2,1}$), se procede de la siguiente manera:

```
» d = c(2, 1)
```

dónde el primer número dentro del paréntesis corresponde a la fila, y el segundo a la columna.

Se debe notar que el primer elemento de un vector, la primer fila de una matriz o la primer columna de una matriz es la 1 (a diferencia del lenguaje C, donde el primer elemento es el 0). Por ejemplo la sentencia » b = a(0, 1) resulta en un error. También resulta en un error tratar de acceder a un elemento fuera de la dimensión del vector o matriz.

Acceso a rangos de una matriz:

Por ejemplo para deseamos asignar a d la submatriz izquierda superior de 2x2 de c :

```
» d = c (1:2, 1:2)
```

Para asignar a d la segunda columna de c :

```
» d = c ( : , 2)
```

O para asignar a b la tercera fila de a :

```
» d = c ( 3 , : )
```

También se pueden extraer subconjuntos discontinuos de una matriz, x ej. para crear una matriz formada por la primera y tercera fila de c :

```
» d = c ([ 1, 3] , : )
```

Al margen: si se escribe una expresión sin asignarlo a una variable específica, el resultado del mismo es asignado a la variable ans.

Por ejemplo:

```
» c ( 3 , : )
» ans
»        [ 7       8        9]
```

asigna a ans la tercer fila de a.

Así como se pueden leer elementos de una matriz, también se pueden modificar los elementos de una matriz. Por ejemplo para asignar al elemento (1, 2) ó (fila 1, columna 2) de la matriz c el valor 3.12 :

```
» c (1 , 2) = 3.12
```

También se pueden modificar rangos de una matriz. Por ejemplo para substituir la segunda columna de a por el vector $[1\ 1\ 1]^T$:

```
» a (: , 2) = [1; 1; 1]
```

Al margen: Es conveniente introducir aquí el operador unitario (involucra un sólo operando) transpuesto. Se representa por la comilla simple (‘) y afecta a la expresión que está a la izquierda de la misma. Por ejemplo, para substituir la segunda columna de a por el vector $[1\ 1\ 1]^T$ también se puede hacer de la siguiente manera:

```
» a (: , 2) = [1, 1, 1]'
```

Al margen: Cómo se habrá observado, cada vez que se introdujo una expresión aparecen abajo nuevas líneas con el resultado. Se puede evitar que MATLAB muestre el resultado de una expresión colocando punto y coma (;) al final de la misma.

Para ver las variables actuales presentes en el área de trabajo:

```
» who
```

Si además se desea ver las dimensiones de las mismas:

```
» whos
```

Para guardar variables se usa el comando save, por ejemplo:

```
» save curso a b
```

En este caso las variables a y b se guardan en el archivo curso.mat en el directorio actual. Para ver o cambiar el directorio actual se usa el comando cd. Para ver el contenido del directorio se usa dir. También se puede usar el botón “Path Browser” de la barra de herramientas.

Para guardar todas las variables del área de trabajo se usa:

```
» save curso
```

Para borrar una variable (por ejemplo la variable a) se usa:

```
» clear a
```

O para borrar a y b:

```
» clear a b
```

Para borrar todas la variables:

```
» clear
```

Se puede inicializar o reinicializar una variable asignándole un vector (matriz) vacío:

```
» a=[]
```

Para recuperar variables guardadas en archivos se usa:

```
» load curso
```

Operadores

Al igual que las variables, los operadores son por defecto matriciales y aceptan números complejos.

Se verán ahora los operadores algebráicos más comunes.

- \+ Suma matricial (suma elemento a elemento) las matrices deben ser de las misma dimensión.
- \- Resta. Idem suma.

- `*` Producto matricial. Las matrices deben ser compatibles.
- `/` Solución de un sistema de ecuaciones lineales. Por ej. x = B / A es B*inv(A). Las matrices deben ser compatibles. Notar que en el caso escalar, esto equivale a la división.
- `\` Solución de un sistema de ecuaciones lineales. Por ej. x = A / B ó inv(A)*B es la solución de A*x=B. Las matrices deben ser compatibles. Notar que B/A = (A'\B')'.
- `^` Potencia. B^2 equivale a B*B.
- `'` Transposición. Ya se vio.

Otros operadores útiles:

- `:` Rango. Genera un vector de los números naturales entre el valor del operando izq. y el valor del operando Der. Por ej. 1 : 3 resulta en [1 2 3]. También se pueden generar rangos de números reales a intervalos fijos. Por ej. 1 : 0.1 : 1.3 resulta en [1 1.1 1.2 1.3].
- `;` No volcar el resultado de la sentencia en la pantalla. Ya se vió.
- `.` Se aplica delante de los operadores *, / y ^, para indicar que la misma se debe hacer elemento a elemento (array operation) y no matricial. Los operandos deben tener las mismas dimensiones.
  Por ej. X = A .* B resulta en x(i, j) = a(i,j) * b(i,j) para todo i, j.

Funciones

La función es el "ladrillo" sobre el que se basa el Matlab. Existen funciones internas, las cuales ya están compiladas, y funciones externas que se compilan (o interpretan) al ejecutarlos.

El MATLAB, trae incorporado un conjunto de funciones básicas organizadas en los siguientes grupos:

```
matlab\general       -  General purpose commands.
matlab\ops           -  Operators and special characters.
matlab\lang          -  Programming language constructs.
matlab\elmat         -  Elementary matrices and matrix manipulation.
matlab\elfun         -  Elementary math functions.
matlab\specfun       -  Specialized math functions.
matlab\matfun        -  Matrix functions - numerical linear algebra.
matlab\datafun       -  Data analysis and Fourier transforms.
matlab\polyfun       -  Interpolation and polynomials.
matlab\funfun        -  Function functions and ODE solvers.
matlab\sparfun       -  Sparse matrices.
matlab\graph2d       -  Two dimensional graphs.
matlab\graph3d       -  Three dimensional graphs.
matlab\specgraph     -  Specialized graphs.
matlab\graphics      -  Handle Graphics.
matlab\uitools       -  Graphical user interface tools.
matlab\strfun        -  Character strings.
matlab\iofun         -  File input/output.
matlab\timefun       -  Time and dates.
matlab\datatypes     -  Data types and structures.
matlab\winfun        -  Windows Operating System Interface Files (DDE/ActiveX)
matlab\demos         -  Examples and demonstrations.
```

En el grupo "elfun" se encuentran las funciones trigonométricas. Por ejemplo para obtener la función seno de 30°:

```
» sin(30*pi/180)
```

Se debe notar que los ángulos se expresan siempre en radianes. La contante π está incorporada y se la designa con pi.

La función sin admite arrays, esto implica que si el argumento es un vector o una matriz, la función calcula el seno de los elementos. Por ejemplo,

```
» sin(pi*[15, 30, 60]/180)
```

devuelve un vector cuyos elementos son el seno de 15°, de 30° y de 60°.

Otra función del grupo "elfun" es la funcion exponencial, en cuyo caso el argumento puede ser un escalar:

```
» exp( 1 )
```

puede ser complejo,

```
» exp( i*pi/4 )
```

una matriz cuadrada, en cuyo caso se resuelve la serie $e^{\mathbf{A}} = \mathbf{I} + \mathbf{A} + \frac{1}{2}\mathbf{A}^2 + \frac{1}{4}\mathbf{A}^3 + \cdots$

```
» exp( [1, 2; 3, 4] )
```

o una matriz no cuadrada o un vector, en cuyo caso la función devuelve la exponencial de cada elemento:

```
» exp( [1, 2, 3+4i, 4] )
```

Dada la extensa cantidad de funciones incorporadas no se describirán todas, sino sólo aquellas que se usen en los ejemplos. Sin embargo más adelante se explicarán las herramienta por medio de la cuales se pueden encontrar documentación de las mismas (ayuda).

Además de las funciones incorporadas existen paquetes adicionales, llamados “Toolboxes”, con funciones específicas para cada disciplina, como por ejemplo:

```
toolbox\stats        -  Statistics Toolbox.
toolbox\symbolic     -  Symbolic Math Toolbox.
signal\signal        -  Signal Processing Toolbox.
toolbox\optim        -  Optimization Toolbox.
toolbox\control      -  Control System Toolbox.
simulink\simulink    -  Simulink
powersys\powersys    -  Power System Blockset
```

Se pueden ver las "Toolboxes" ofrecidas por MATLAB en http://www.mathworks.com

Finalmente el usuario puede crear sus propias funciones. Más adelante se mostrará como crearlas.

Uso de la ayuda

Ayuda desde el área del trabajo.

Organización de la ayuda:

"Todo"
- "Capítulos"
  - "Funciones" (operadores, comandos, etc.)

Para invocar la ayuda se usa el comando help. El comando help sin argumento muestra la lista de “capítulos” de la documentación.

```
» help
```

Si después de help se agrega el nombre del “capítulo”, muestra la lista de funciones que comprende ese capítulo.

```
» help elfun
```

Para mostrar la ayuda de una función debe escribir el comando help seguido del nombre de la función:

```
» help exp
```

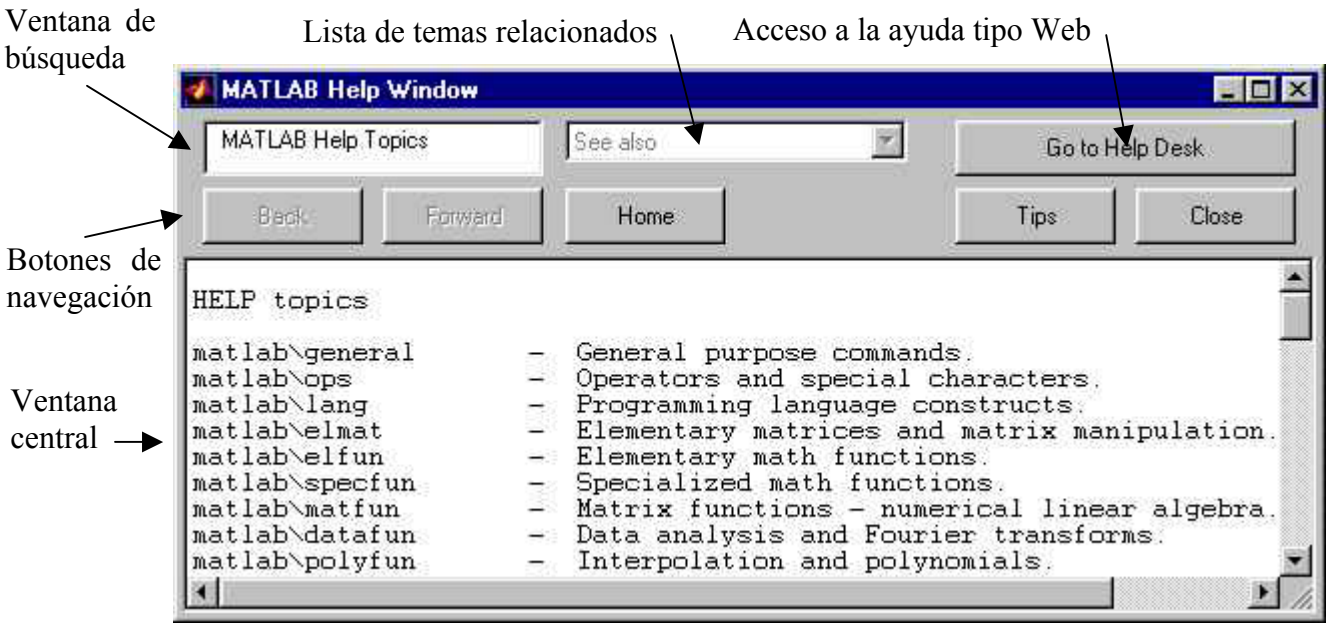
Existen una convención muy importante sobre la sintaxis de las ayudas. Cuando se muestra la forma de uso de una función o comando, el nombre de la mísma aparece en mayúscula para diferenciarlo del resto del texto. Lo mismo ocurre en los ejemplos. Sin embargo para usar esas funciones o comandos, se las debe escribir en minúsculas.

Todas las funciones y comandos de MATLAB están definidas en minúsculas. (En la ayuda se usa mayúscula ya que al estar escrito en texto sin formato, no se pueden usar usar otros recursos para destacarlos como ser la letra en negrita, itálica, colores, etc.)

> **Al margen**: Vale la pena inspeccionar la ayuda de demos (help demos), y ejecutar algunas demostraciones que figuran en ella. También se puede ejecutar el comando demo, la cual abre una interfase gráfica para acceder a las mismas demostraciones que figuran en la ayuda.

Ayuda interactiva.

La ayuda se accede desde la barra de herramientas (Botón “?”), o desde el menú Help. La organización y el contenido de la ayuda interactiva es la misma que la descripta anteriormente.



Al abrir la ayuda interactiva se muestran, en la ventana central de la interfase de la ayuda, la lista de "Capítulos" de la ayuda. Haciendo doble click en el "Capítulo" se muestran las "Funciones" correspondientes a dicho "Capítulo". Finalmente haciendo doble click en la "Función" se obtiene la documentación de la función seleccionada.

Se pueden hacer búsquedas escribiendo el nombre de una función en la ventana que se encuentra arriba a la izquierda de la interfase de ayuda, y presionando luego la tecla "enter".

Si se está viendo la documentación de una "Función" en la ventana central de la ayuda, se puede ir directamente a la ayuda de las funciones relacionadas por medio de la lista desplegable que se encuentra arriba al medio de la interfase de ayuda.

Help desk.

Esta es una ayuda basada en HTML (tipo página de web), y muestra información con mayor detalle que la ayuda desde el área de trabajo o la ayuda interactiva. Se puede acceder a la misma desde el menú "Help" o desde la interfase de ayuda interactiva, usando el botón "Go to Help Desk".

Y como si esto fuese insuficiente, desde el Help Desk se puede acceder también a la documentación en PDF que es la versión electrónica de los manuales impresos.

Ejemplos

Matrices

Se recomiendan usar las herramientas de "ayuda" para ver la documentación de las funciones y comandos usados en los siguientes ejemplos.

Creamos una matriz de 3x3 con elementos aleatorios (entre 0 y 1).

```
» x=rand(3)
```

Vemos la dimensión

```
» size(x)
```

Obtenemos la inversa.

```
» y=inv(x)     ó        y=eye(3)/x
```

El producto de la matriz por su inversa resulta en la identidad,

```
» x*y
```

Al margen: Se debe recordar que se están usando métodos numéricos con números de longitud finita. Por lo tanto si bien la solución puede aproximarse con un error mínimo, no será exacto.

La matriz identidad restada del producto entre la matriz x y su inversa no da el error del cálculo numérico.

```
» e=x*y-eye(3)
```

Polinomios representados como vectores. La función poly determina los coeficientes del polinomio característico de una matriz cuadrada, por ejemplo para la matriz A es $\det(\lambda \mathbf{I} - \mathbf{A})$.

```
» p=poly(x)
```

Se pueden multiplicar o dividir polinomios, esto equivale a la convolución o deconvolución de los vectores , respectivamente; expandir en fracciones simples, etc. Ver help polyfun.

Volviendo al ejemplo, si extraemos las raíces de este polinomio, se obtienen los autovalores de la matriz.

```
» a=roots(p)
```

De todos modos hay un camino más fácil para obtener los autovalores de una matriz,

```
» eig(x)
```

Gráficos 2D

```
» x=0:0.1:4*pi;
» y=sin(x);
» z=0.6*cos(x);
```

Ojo recordar de poner los puntos y comas para evitar que se vuelquen en la pantalla vectores extensos.

Para graficar la función seno en función del ángulo:

```
» plot(x, y)
```

Para graficar más de una curva simultáneamente en la misma ventana:

```
» plot(x, y, x, z)
```

También se puede graficar una curva, y luego agregar otras.

```
» plot(x, y)
» hold on
» plot(x, z)
» hold off
```

Se puede abrir una nueva ventana para otras curvas.

```
» figure
» plot(x, z)
```

Se debe notar que la función plot traza la curva sobre la última ventana gráfica que se activó. Para definir los colores y tipo de línea para cada curva:

```
» plot(x, y, 'r', x, z, 'k:')
» plot(x, y, 'ro', x, z, 'bx')
```

La siguiente lista muestra los códigos para colores y para tipos de líneas y puntos (ver help graph2d para mayor detalle):

y	yellow	.	point	-	solid
m	magenta	o	circle	:	dotted
c	cyan	x	x-mark	-.	dashdot
r	red	+	plus	--	dashed
g	green	*	star		
b	blue	s	square		
w	white	d	diamond		
k	black	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

Se pueden ajustar manualmente las escalas de los ejes:

```
» axis([0 2*pi -2 2])
```

Ejemplo integrador: Ciclo de histéresis.

Con un osciloscopio digital se obtuvieron las curvas de tensión y corriente de un inductor con núcleo de hierro. Estas curvas fueron descargadas en dos archivos de texto organizado cada uno en dos columnas separadas por espacios. La primer columna ("array") corresponde al tiempo, y la segunda a la variable muestreada (tensión para el primer archivo y corriente para el segundo).

Las columnas tienen 2500 filas correspondientes a 50 ms de datos muestreados cada 20useg.

Este tipo de archivos se pueden cargar usando el comando load.

```
» cd \curso          ← o el directorio donde estén los archivos.
» load ch1.txt       ← observar que los datos de ch1.txt pasan a la variable ch1.
» load ch2.txt.
» whos               ← para ver las variables nuevas que se crearon.
```

Los 50 mseg (2500 muestras) corresponden a dos ciclos y medio de los datos relevados. Para facilitar los cálculos tomamos una cantidad entera de ciclos, por ejemplo dos (2000 muestras).

```
» t= ch1(1:2000,1);
» v= ch1(1:2000,2);
» i= ch2(1:2000,2);
```

Graficaremos las curvas usando una nueva función que permite crear más de un gráfico en una ventana, es similar a la función figure:

```
» subplot(2,1,1);    ← define una matriz de 2 x 1 gráficos, y activa al primero.
» plot(t, v)
» subplot(2,1,2);    ← activa al segundo gráfico.
» plot(t, i)
```

Se desea obtener el flujo magnético en función de la intensidad de campo. Considerando que la tensión inducida es la derivada del flujo en función del tiempo, por el número de espiras:

$$v = n\frac{d\phi}{dt} \qquad [\text{V}]$$

en consecuencia:

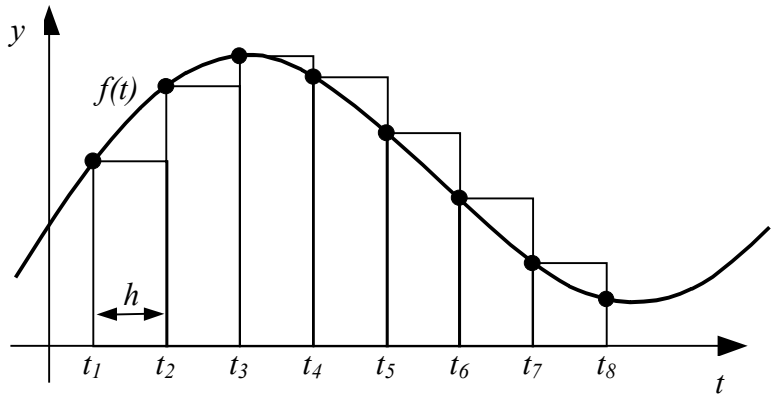
$$\phi = \frac{1}{n}\int_0^t v\,d\tau + C \qquad [\text{V seg}].$$

La intensidad de campo es el producto de la corriente por el número de espiras por la longitud equivalente del circuito magnético:

$$H = i \cdot n \cdot l \qquad [\text{A m}]$$

Para simplificar el cálculo $n = 1$ y $l = 1$.

Para efectuar la integración se usará el método "rectangular". Por ejemplo, se tiene la función *f* muestreada a intervalos *h*. (Ver gráfico). Para obtener la integral desde t_1 hasta t_n se puede aproximar a $\int_{t1}^{tn} f(\tau)\,d\tau \cong h\sum_{i=1}^{n} f(t_i)$.



Por lo tanto al obtener la integral de la tensión desde t_1 hasta t_n , con n correspondiente a cada posición del "array" de la tensión, se tendrá un "array" de flujo. Esto se puede efectuar fácilmente usando la función cumsum (ver help cumsum).

```
» h=diff(t(1:2))          ó      » h= t(2)- t(1)
» f1=h*cumsum(v);
```

También se puede usar la función cumtrapz para efectuar la integración trpezoidal.

Sabiendo que el valor medio del flujo es nulo (para ciclos enteros), se puede determinar la constante de integración cómo menos el valor medio de *f1*.

```
» C=-mean(f1);
» f=f1+C;
```

Ahora, para graficar el ciclo de histéresis,

```
» figure;
» plot(i, f);
```

Se pueden agregar títulos y etiquetas a los ejes de la siguiente manera:

```
» title('Ciclo de histeresis');
» xlabel('H');
» ylabel('\phi');
```

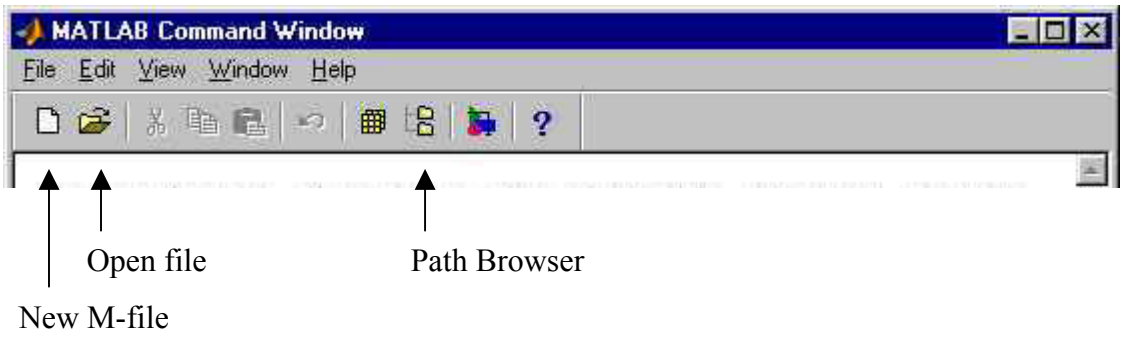
Más adelante se verá la forma de escribir una secuencia de instrucciones en un archivo ("script") de manera de poder ejecutarlo repetidamente. Esto puede ser útil, por ejemplo, en el caso anterior cuando se quiera repetir el cálculo del ciclo de histéresis para distintas muestras de tensión y corriente.

Programación

Los programas en MATLAB ("scripts") consisten en una secuencia de sentencias escritas en un archivo de texto. Las sentencias se escriben tal como si se escribieran en el área de trabajo. El archivo debe tener extensión "m", de allí que también se los llama "M-file".

Para correr el programa, se escribe en el área de trabajo el nombre del archivo (sin la extensión). El directorio actual debe ser el directorio en el que reside el archivo. (Más adelante se mostrará como correr M-files que se encuentren en un directorio distinto al actual.)

Para crear M-files, se puede usar cualquier editor de textos como por ejemplo el Bolck de Notas del Windows. Sin embargo el Matlab incluye un editor con algunas características útiles para la edición y depuración de M-files.
Se puede acceder al editor del Matlab por medio del menú "File → New → M-File" para crear un archivo nuevo o por medio del menú "File → Open..." para abrir un archivo existente. También se pueden usar los botones "New M-file" o "Open File" de la barra de herramientas.



Ejemplo: Regresión (llenado de curvas).

Se realizó un ensayo del cual se obtuvieron cien pares de valores (x_n, y_n) $n = 1 \cdots 100$. Se desean encontrar las funciones polinómicas de primer y segundo orden que aproximen con el menor error cuadrático medio.

El archivo datos1.txt contiene una matriz de 100x2 cuya primer columna corresponde a los valores de x, y la segunda los valores de y.

La siguiente es una secuencia de sentencias que resuelve el problema usando la función polyfit (ver help polyfit). Crear un M-file llamado "regresion.m" con la secuencia de sentencias, y guardarlo el el directorio actual (por ejemplo c:\MATLABR11\work\). Las líneas que comienzan con el signo porcentaje (%) no son ejecutadas y se usan sólo para incorporar comentarios al archivo.

```
%Archivo regresion.m
%Carga de los datos del experimento
load datos1.txt
x= datos1(:,1);
y= datos1(:,2);

%Grafica los datos
plot(x, y, 'kx')

%Aproximación polinómica de 1er orden
p1=polyfit(x, y, 1)

%Vector con igual rango que x pero uniformemente distribuido
%(usado para graficar)
x1 = (min(x):0.1:max(x))';

%Grafica el polinomio
hold on
plot(x1, polyval(p1,x1), 'b')

%Calcula el error cuadrático medio
e=(y-polyval(p1, x));
```

```
mean(e.^2)

%Aproximación polinómica de 2do orden
p2=polyfit(x, y, 3)

%Grafica el polinomio
plot(x1, polyval(p2,x1), 'r')
hold off

%Calcula el error cuadrático medio
e=(y-polyval(p2, x));
mean(e.^2)
```

Para correr este programa, se debe escribir en el área de trabajo:

» regresion

Se debe recordar que el directorio actual debe ser el que aloje al programa que se desea ejecutar. Recordar también que para ver o cambiar de directorio se puede usar el comando cd, y par ver el contenido de un directorio se puede usar el comando dir. Alternativamente se puede usar la herramienta "Path Browser" el cual se puede acceder con el botón correspondiente de la barra de herramientas del Matlab.

Para interrumpir la ejecución de un M-file presionar la combinación de teclas Ctrl-C.

Control de flujo

En el programa anterior se corrió una secuencia de instrucciones con flujo "lineal". Sin embargo se puede desear efectuar cálculos con una secuencia "repetitiva" o "iterativa", o que ciertas sentencias se ejecuten en forma "condicional". Para ello existen comandos de control de flujo que se describirán a continuación. Se debe prestar atención a los ejemplos que se darán para cada estructura de control de flujo dado que en ellas se introducirán otras funciones útiles en la programación (lectura del teclado, impresión en el área de trabajo, etc.). Ver help lang.

Repetición de sentencias un número conocido de veces (for):

```
for variable = expresión,
   sentencia;
   ...
   sentencia;
end
```

Las "sentencias" se repiten tantas veces como elementos tenga "expresión". La "variable" toma el valor de cada elemento de la expresión, uno por vez.

Por ejemplo:

```
x= rand(10,1);
for n = 1:10,
   y(n,1)=x(11-n);
end
```

En el ejemplo se asigna a la variable y la variable aleatoria x en orden inverso.

Repetición indefinida de sentencias (while):

```
while expresión
```

```
   sentencias;
end
```

Las “sentencias” se repiten mientras “expresión” sea no nulo. La “expresión” generalmente es una comparación tipo

```
expresión operador_relacional expresión
```

dónde el operador relacional puede ser alguno de los siguientes (ver help relop):

== Igual qué.

< Menor qué.

> Mayor qué.

<= Menor o igual qué.

>= Mayor o igual qué.

~= Distinto qué.

Por ejemplo:

```
x=0;
n=0;
disp('Ingrese números menores que 100')
disp('Para finalizar ingrese un número mayor que 100')

while x < 100,
   x=input('Ingrese un número: ');
   n=n+1;
   y(n,1)=x;
end
disp(['Se ingresaron ', int2str(n), ' números'])
y
```

Este ejemplo carga un vector y con valores introducidos por el teclado, mientras éstos sean menores que 100.

Ejecución condicional (if):

```
if expresión
   sentencias;
end
```

Si “expresión” es no nulo, se ejecutan “sentencias”.

```
if expresión_1
   sentencias_A;
else
   sentencias_B;
end
```

Si “expresión_1” es no nulo, se ejecutan las “sentencias_A”, si no, se ejecutan las “sentencias_B”.

```
if expresión_1
   sentencias_A;
elseif expresión_2
   sentencias_B;
else
   sentencias_C;
end
```

Si "expresión_1" es no nulo, se ejecutan las "sentencias_A", si no, si "expresión_2" es no nulo, se ejecutan las "sentencias_B", si no (ninguno de los anteriores), se ejecutan las "sentencias_C".

Selección de casos (switch)

El comando switch sirve para seleccionar el código a ejecutar en función de un número entero o una cadena de caracteres (las cadenas de caracteres se verá en detalle más adelante).

```
switch expresión_llave
   case caso1
      sentencias_A
   case {caso2,caso3,caso4}
   sentencias_B

   otherwise
      sentencias_C
end
```

Si el resultado de "*expresión_llave*" coincide con "*caso1*", se ejecutan "sentencias_A" y luego se continúa ejecutando las sentencias despues de end.
Si el resultado de "*expresión_llave*" coincide con "*caso2*", "*caso3*" o "*caso4*" se ejecutan "sentencias_B" y luego se continúa ejecutando las sentencias después de end. Aquí la a sentencia case responde a más de una condición.
Se pueden crear cuantos casos sean necesarios usando la sentencia case para cada condición o grupo de condiciones.
Si ninguno de los casos anteriores aplica, se ejecutan "sentencias_B" y luego se continúa ejecutando las sentencias después de end.

Biblioteca de funciones

El paquete Matlab tiene una amplia cantidad de funciones incorporadas. No obstante existen bibliotecas de funciones "tooboxes" adicionales que se pueden adquirir por separado. Existen bibliotecas del mismo fabricante o de terceros.

Cada biblioteca de funciones reside en un directorio y a cada función le corresponde un archivo con extensión ".m" dentro de ese directorio.

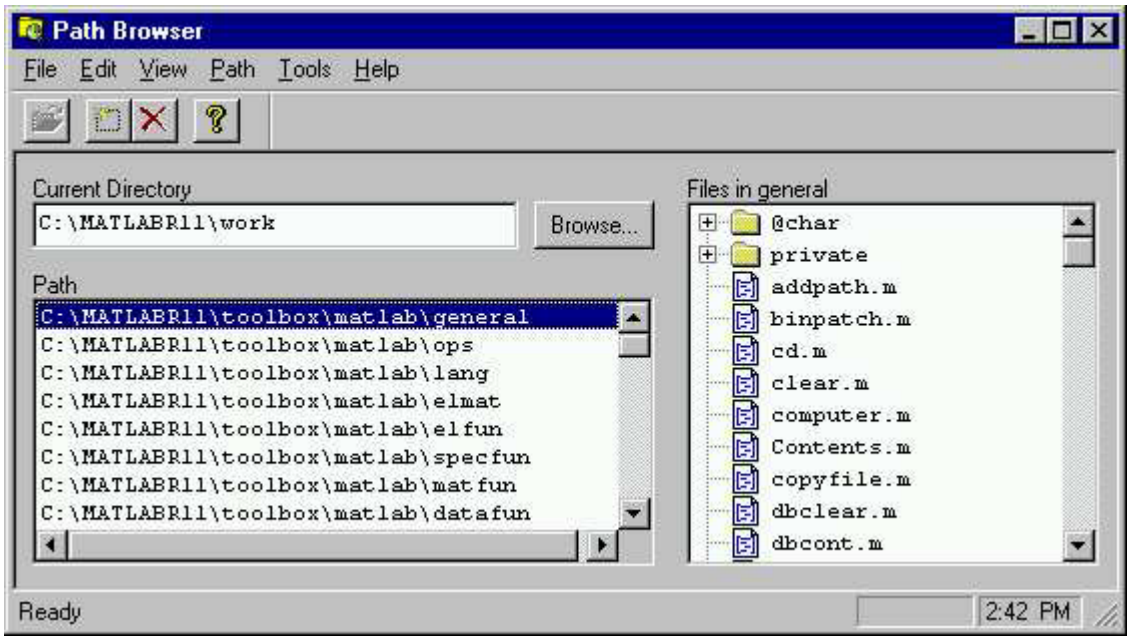
Para que Matlab pueda encontrar las funciones, existe un listado de directorios ("path"). Cada vez que se ejecuta una función, el sistema busca si este se encuentra en el directorio actual, si no lo encuentra, sigue buscando en la lista de directirios.

Para ver el listado actual de directorios, se puede usar el comando path. Este comando también sirve agregar o eliminar directorios a la lista.

Por ejemplo:

```
» path

            MATLABPATH

      C:\MATLABR11\toolbox\matlab\general
      C:\MATLABR11\toolbox\matlab\ops
      C:\MATLABR11\toolbox\matlab\lang
      C:\MATLABR11\toolbox\matlab\elmat
      C:\MATLABR11\toolbox\matlab\elfun
      C:\MATLABR11\toolbox\stats
      C:\MATLABR11\toolbox\symbolic
      C:\MATLABR11\toolbox\ncd
      C:\MATLABR11\toolbox\signal\signal
      C:\MATLABR11\toolbox\optim
      C:\MATLABR11\toolbox\control
      C:\MATLABR11\toolbox\simulink\simulink
      C:\MATLABR11\work
```

La lista mostrada es parcial y a modo de ejemplo. Las funciones que están en "`C:\MATLABR11\toolbox\matlab\`" son las que incluye por defecto el Matlab. Los demás directorios corresponden a bibliotecas adicionales.
Esta lista se puede completar con los directorios donde el usuario creará sus propias funciones o programas. Para modificar la lista agregando o eliminando directorios de la misma, conviene usar la herramienta "Path browser" a la cual se puede acceder desde la barra de herramientas.



Creación de Funciones

Las funciones se deben describir (usando los operadores y las funciones existentes) en un archivo cuyo nombre es el nombre de la nueva función, y con extensión ".m". La primer línea del archivo debe describir la sintaxis de la función, de la siguiente manera:

`function` *var_dev* = nombre_función (*var_arg1, var_arg2*)

Esta línea define el nombre "interno" de las variables argumento de la función y el nombre "interno" de la variable devuelta por la función. Las siguientes líneas del archivo deberán ser operaciones que usen *var_arg1, var_arg2* para generar la variable *var_dev* que será el resultado devuelto. Aquí se muestran dos argumentos (pueden se escalares o matrices), pero se pueden definir cuantos se deseen, incluso ninguno.

Si se desea crear una función que devuelve más de un resultado (pueden se escalares o matrices) se debe definir como se muestra a continuación:

`function` [*var_dev1, var_dev2*] = nombre_función (*var_arg1, var_arg2*)

A diferencia de los programas ("scripts"), las variables usadas dentro de las funciones están protegidas. Esto significa que no pueden ser leídas ni modificadas desde el entorno de trabajo, ni de otra función.

Ejemplo simple

Crear un archivo de texto con nombre "funcion1.m" en el directorio actual, o en otro directorio que se haya agregado a la lista de directorios del Matlab, con el siguiente contenido:

```
function c = funcion1(a, b)

   t = a*pi+b;
   c = cos(t);
```

La función calcula $c = \cos(a\pi + b)$. Las variables a y b son los argumento, c es la variable que debe llevar el resultado, y t es una variable intermedia.

Una vez guardado el archivo, se puede usar la función:

```
» funcion1(1, 0)

ans
      -1
```

Se puede verificar, usando el comando whos, que ninguna de las variables usadas dentro de la función aparecen el entorno de trabajo.

La función creada se puede usar dentro de operaciones, en programas o en otras funciones.

Ejemplo que devuelve dos resultados:

Crear la siguiente función en un archivo con nombre "stat.m":

```
function [med, des] = stat(x)
%Calcula la media y la desviación standard del vector x.
      n = length(x);
      med = sum(x) / n;
      des = sqrt(sum((x - med).^2)/n);
```

Para usar la función se la debe asignas a un vector de dos variables. El nombre de las variables no tiene importancia, sí la posición. El primer elemento devuelto será la media, y el segundo elemento devuelto será la desviación standard.

```
» [m, d]=stat([1, 1.2, 1.5, 0.8])

m =
    1.1250

d =
    0.2586
```

Si se usa la función sin asignárselo a ninguna variable, sólo el primer resultado (en este ejemplo la media) será mostrada y asignada a la variable ans. De igual manera, si la función se la asigna a una sola variable, esta tomará el primer resultado.
Como se muestra en el ejemplo, si se agrega un comentario inmediatamente después de la definición de la sintaxis, ésta se muestra al ejecutar el comando help *nombre_función*.

```
» help stat

 Calcula la media y la desviación standard del vector x.
```

Se pueden agregar cuantas líneas de comentarios se deseen para que aparezcan en la ayuda, siempre que sean consecutivas.

Ejemplo con subfunciones

Crear la siguiente función en un archivo con nombre "stat1.m":

```
function [med, des] = stat1(x)
%Calcula la media y la desviación standard del vector x.
      n = length(x);
      med = avg(x, n);
      des = sqrt(sum((x - med).^2)/n);

%-------------------------
function med = avg(x,n)
%Subfunción par el cálculo de la media
   med = sum(x)/n;
```

La primera definición de función (stat1) usa la función avg que se pude definir dentro del mismo archivo. Se debe definir agregando una nueva línea de sintaxis y la definición misma de la función, después del final de la función que la usará. La subfunción sólo tendrá validez dentro del archivo, y no se puede llamar por separado.
Si se desea crear una subfución que a su vez se pueda usar por separado o en otras funciones, se la debe definir en un archivo aparte.

Otro ejemplo

Crear una función llamada sinc que calcule $y = \frac{\sin(x)}{x}$ salvando la indeterminación para $x = 0$. Debe poder calcular la función elemento a elemento si el argumento es un vector.

```
function y = sinc(x)
%Calcula sin(x)/x.

      n = length(x);
      for i=1:n,
         if x(i) == 0
            y(i)=1;
         else
            y(i)=sin(x(i))/x(i);
         end
      end
```

Función de Funciones

Existe un grupo de funciones que opera sobre funciones (Ver help funfun). Estas incluyen gráficas de funciones, minimización, obtención de ceros (resolución de sistemas de ecuaciones no lineales), resolución de ecuaciones diferenciales ordinarias, etc.

Gráficas de funciones:

Para graficar funciones se puede usar la función fplot.
Sintaxis:

```
fplot(función, límites)
```

Dónde *función* será una cadena de caracteres que contenga el nombre de una función o una expresión, con una sola variable argumento. El argumento *límites* debe ser un vector de dos elementos indicando el límite inferior y superior respectivamente.
Por ejemplo para graficar la función sinc en el rango $x = [-15, 15]$, se debe escribir:

```
» fplot('sinc', [-15, 15])
```

Se puede graficar una función con dos argumentos si se establece una como constante. Por ejemplo:

```
» fplot('diric(x,10)', [-15, 15])
```

También se pueden superponer dos funciones en un mismo gráfico si se escribe cada función como un elemento de un vector de funciones:

```
» fplot('[sinc(x), diric(x,10)]',[-15, 15])
```

Minimización:

Para encontrar el mínimo de una función escalar, dentro de un rango, se puede usar la función fminbnd. Existen otras funciones para encontrar mínimos ya sea para funciones escalares, multidimensionales, etc. Ver help funfun y help optim.

Sintaxis:

```
x_min = fminbnd(función, lím_inf, lím_sup)
```

Dónde *función* es una cadena de caracteres que contenga el nombre o la expresión de una función del tipo $y = f(x)$, con x e y valores escalares reales. La cota inferior y superior de x se define con *lim_inf* y *lim_sup* respectivamente. La función devolverá el valor de x dentro de la cota, para el cuál y tenga el menor valor.

Por ejemplo para encontrar el mínimo de la función sinc en el rango [2, 6], se debe escribier:

```
» xmin = fminbnd('sinc', 2, 6)
```

Para graficar este resultado se puede escribir:

```
» fplot('sinc', [2, 6])
» hold on
» plot(xmin, sinc(xmin), 'o')
```

Para encontrar el máximo de la misma función en el mismo rango:

```
» xmax = fminbnd('-sinc(x)', 2, 6)
```

Si el gráfico anterior todavía está abierto, se puede imprimir la ubicación del máximo sobre el mismo:

```
» plot(xmax, sinc(xmax), 'o')
```

Ceros de una función:

Cuando se desea buscar ceros de una función escalar, se puede usar la función fzero.

Sintaxis:

```
x = fzero(función, x0)
```

Dónde *función* es una cadena de caracteres que contenga el nombre o la expresión de una función del tipo $y = f(x)$, con x e y valores escalares reales. La función fzero devuelve el valor de x cercano al cero o cambio de signo de y. La búsqueda del cero comienza en el valor *x0*.
Ejemplo:
Se desea encontrar el cero más cercano a $x = 1$:

```
» x = fzero('sinc', 1)
```

Para graficar este resultado se puede escribir:

```
» fplot('sinc', [0, 10])
» hold on
» plot(x, sinc(x), 'ro')
```

Integración:

Para la integración definida de una función se pueden usar algunas de las siguientes funciones:

```
quad – Use Adaptive Simpson's rule
quad8 – Use Adaptive Newton Cotes 8 panel rule
```

La sintaxis es igual para ambas:

```
q = quad(F, lim_inf, lim_sup)
```

Dónde quad devuelve la integral de $F(x)$ definido entre *lim_inf*, *lim_sup*. `F` es una función escalar que debe devolver un vector si su argumento es un vector.
Ejemplo (del manual de Matlab) :
Para determinar la integral de la función sinc definido entre $x = 0$, y $x = 2$, se debe escribir:

```
a = quad('sinc', 0, 2);
```

Otro ejemplo:
Cálculo de la longitud de una curva. Considerando la siguiente curva paramétrizada:

$$x(t) = \sin(2t), \quad y(t) = \cos(t), \quad z(t) = t$$

con $0 \le t \le 3\pi$
Para graficar curvas en el espacio, se puede usar la función plot3 (ver la ayuda de la función), como se muestra a continuación:

```
t = 0:0.1:3*pi;
plot3(sin(2*t), cos(t), t);
```

La fórmula de "longitud de arco" dice que la longitud de la curva es igual a la integral del módulo de las derivadas de las ecuaciones parametrizadas. Aplicando este criterio a la curva del ejemplo:

$$l = \int_0^{3\pi} \sqrt{4\cos(2t)^2 + \sin(t)^2 + 1}\, dt$$

Para resolverla integral, escribimos la función del integrando:

```
function f = hcurva(t)

f = sqrt(4*cos(2*t).^2 + sin(t).^2 + 1);
```

Integrando la función anterior, se obtiene la longitud de la curva.

```
l = quad('hcurva',0,3*pi)
```

Resolución de ecuaciones diferenciales ordinarias:

Para la resolución de ecuaciones diferenciales ordinarias, primero se debe describir al problema como un sistema de ecuaciones de primer orden (Representación en ecuaciones de estado):

Dado una ecuación diferencial ordinaria:

$$y^{(n)} = f(t, y', y'', \cdots, y^{(n-1)})$$

realizando las siguientes sustituciones:

$$y_1 = y,\ y_2 = y',\ y_3 = y'',\ y_n = y^{(n-1)}$$

Se obtiene el siguiente sistema de ecuaciones de primer orden:

$$y_1' = y_2$$
$$y_2' = y_3$$
$$\cdots$$
$$y_n' = f(t, y_1, y_2, \cdots, y_n)$$

El problema del valor inicial de este sistema queda descrito (en forma vectorial) como:

$$Y' = F(t, Y), \text{ con } Y(0) = Y_0 \qquad \text{dónde} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}, \quad F = \begin{bmatrix} f_1 \\ f_2 \\ \cdots \\ f_n \end{bmatrix}$$

Para resolver ecuaciones diferenciales en Matlab, se debe crear la función *F* del sistema de ecuaciones. Luego se puede usar alguno de los métodos de resolución de EDOs, cuya sintaxis general es:

```
[T, Y] = método(F, rango_de_tiempo,Y0)
```

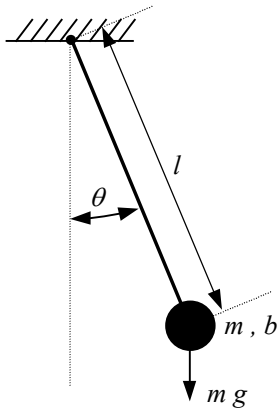
Dónde *método* es el nombre del "resolvedor" de EDOs (luego se verán los nombres). *F* es el nombre de la función vectorial de estado, de la forma $Y' = F(t, Y)$. *rango_de_tiempo* es un vector de dos elementos, cuyos valores son el tiempo inicial y el tiempo final. *Y0* es el vector de valores iniciales. El *método* devolverá un vector columna de tiempo *T* con tantos elementos como intervalos usó el método para resolver la EDO, y una matriz *Y*, con *n* columnas (cantidad de variables de estado) y tantas filas como *T*.

Métodos:

ode45 - Solve non-stiff differential equations, medium order method.
ode23 - Solve non-stiff differential equations, low order method.
ode113 - Solve non-stiff differential equations, variable order method.
ode23t - Solve moderately stiff differential equations, trapezoidal rule.
ode15s - Solve stiff differential equations, variable order method.
ode23s - Solve stiff differential equations, low order method.
ode23tb - Solve stiff differential equations, low order method.

Se recomienda usar inicialmente ode45 (Runge-Kuta 45). Usar los otros métodos en caso que el primero resulte muy lento y seleccionarlos en función de la "rigidez" y orden del sistema a resolver.

Ejemplo:
Supongamos el caso del péndulo de la figura:



Dónde: θ es el ángulo del péndulo con respecto a la vertical (se supone un brazo rígido); l es la longitud del brazo; m es la masa del péndulo; b es el coeficiente de rozamiento y g es la aceleración de la gravedad.
Equiparando las fuerzas tangenciales de la masa, se puede obtener la ecuación diferencial que describe al sistema:

$$\ddot{\theta}\, l\, m + \dot{\theta}\, l\, b + m\, g \sin(\theta) = 0$$

Substituyendo $\theta = y_1$, $\dot{\theta} = y_2$ se puede escribir la ecuación en variables de estado:

$$\dot{y}_1 = y_2$$

$$\dot{y}_2 = \frac{1}{l\, m}\left[-y_2\, l\, b - m\, g \sin(y_1)\right]$$

Para poder resolver este sistema, se debe crear la función de las ecuaciones de estado. Para ello se creará el archivo "pendulo.m" con el siguiente contenido:

```
function dy = pendulo(t, y)
% y es el vector de estados [pos_ang; vel_ang].

l = 1;
m = 1;
g = 9.8;
b = 0.1;

dy = [y(2); (-y(2)*l*b -m*g*sin(y(1)))/(l*m)];
```

Una vez creada la función de las ecuaciones de estado, se puede resolver el sistema usando la función ode45. Se parten de los siguientes valores iniciales: $\theta(0) = y_1(0) = \pi/4$, y $\dot{\theta}(0) = y_2(0) = 0$.

```
» [T, Y]=ode45('pendulo', [0 30], [pi/4; 0]);
» plot(T, Y(:,1), 'b', T, Y(:,2), 'r')
```

En el gráfico se muestran las curvas de posición angular (azul) y velocidad angular (rojo) en función del tiempo.
Se puede ver el comportamiento del péndulo cuando es soltado desde el punto de equilibrio inestable, con una mínima velocidad inicial.

```
» [T, Y]=ode45('pendulo', [0 30], [pi; 0.0001]);
» plot(T, Y(:,1), 'b', T, Y(:,2), 'r')
```

Si se desea modificar alguno de los parámetros, de debe modificar la función.

Cadenas de Caracteres, Estructuras, Listas y Matrices Multidimensionales.

Existen formas de "almacenar" datos, diferentes a las que se han visto hasta este punto (las matrices). No se pretende un manejo detallado de estas formas, pero se debe conocer cómo acceder a ellas dado que muchas funciones de las librerías las usan.

Cadenas de caracteres (char array ó string):

Esta es la forma de que tiene Matlab para representar textos. Básicamente, las cadenas de caracteres son vectores cuyos elementos son caracteres. La forma de crear una cadena de caracteres, es usando las comillas. Por ejemplo:

```
nombre = 'Villa Mercedes';
```

En este caso se asignó la cadena de caracteres a la variable "nombre". El comando whos volcará la siguiente información:

```
  Name         Size         Bytes  Class

  nombre       1x14            28  char array

Grand total is 14 elements using 28 bytes
```

Donde se observa que la dimensión corresponde a la cantidad de caracteres cargados en la variable (incluyendo espacios), y que ocupa dos bytes de memoria por cada caracter. La clase es "char array". La "clase" indica la forma o tipo de información almacenada. En el caso de vectores o matrices, la clase en "double array".
Se mostrarán a continuación algunas funciones útiles para manipular cadenas de caracteres.
Para convertir una cadena de caracteres en un vector donde cada elemento es el código ASCII del correspondiente caracter, se usa la función double :

```
» nombre = double(nombre)

nombre =

  Columns 1 through 12

    86   105   108   108    97    32    77   101   114    99   101   100

  Columns 13 through 14

   101   115
```

La función inversa, que convierte vectores de números (que representan caracteres en código ASCII) en cadena de caracteres es char :

```
» nombre = char(nombre)

nombre =

Villa Mercedes
```

Los operadores relacionales actúan sobre las cadenas de caracteres de igual manera que sobre vectores, esto es, elemento a elemento. Por ejemplo:

```
» cadena1 = 'hola';
» cadena2 = 'halo';
» cadena1 == cadena2
ans =
```

```
     1     0     1     0
```

Se observa que el vector resultante de la comparación indica cuáles caracteres coinciden y cuales no.
Para verificar si una cadena es igual a otra, se puede usar la función strcmp :

```
» strcmp(cadena1, cadena2)
ans =
     0
```

Conversión de números a cadenas de caracteres. Considerando el siguiente escalar:

```
» x = 5246
x =
        5246
```

Se puede convertir a cadena de caracteres como se muestra a continuación:

```
» x = 5246
x =
        5246

» y = int2str(x)
y =
5246
```

No parece haber diferencia entre los resultados de las dos sentencias anteriores. Sin embargo x es un escalar, mientras que y es un vector de cuatro elementos que representa en forma de texto al número cargado en x (Notar los resultados numéricos aparecen alineados a la derecha, mientras que las cadenas aparecen alineados a la izquierda.) Con las sentencia whos so puede observar la dimensión y clase de cada variable.

```
» whos
  Name      Size          Bytes  Class

  x         1x1               8  double array
  y         1x4               8  char array
```

La función int2str sólo convierte enteros en cadena de caracteres. para números reales, se puede usar al función num2str que tiene dos argumentos: el número a convertir la cantidad de cifras (mantisa) que se representarán.

```
» y = num2str(pi, 9)
y =
3.14159265
```

Esta función puede ser de utilidad para agregar títulos o etiquetas a los gráficos, por ejemplo:

```
» x=-rand(1):0.01:rand(1);
» plot(x, sin(x))
» cadena2 = num2str(max(x));
» cadena1 = num2str(min(x));
» cadena3 = ['Función seno desde ', cadena1, ' hasta ', cadena2];
» title(cadena3)
```

Donde se observa la concatenación de textos en la penúltima línea. Esta operación es equivalente a la de unir varios vectores numéricos para formar uno nuevo.
Finalmente si se tiene una variable que contiene el texto de una expresión matemática que pueda interpretar Matlab, se puede evaluar para obtener así el resultado numérico con la función eval :

```
» cadena='5*sin(pi/4)';
» eval(cadena)
ans =
    3.5355
```

Arreglos multidemensionales.

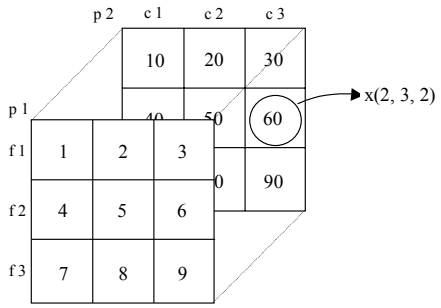
Desde el comienzo del curso se han usado arreglos de una dimensión (vectores) y de dos dimensiones (matrices). Sin embargo en algunas circunstancias pueden ser de utilidad los arreglos de más de dos

dimensiones, por ejemplo si se desean organizar páginas de matrices, o para representar campos en el espacio.
El tratamiento es similar al de las matrices, se puede hacer referencia a un elemento del arreglo usando los subíndices, o rangos de subíndices. A modo de ejemplo se mostrará el caso de un arreglo de tres dimensiones, sin embargo el número de dimensiones puede ser superior.

Se muestra a continuación cómo organizar las matrices x1 y x2 de 3x3 en dos páginas, o sea en un arreglo x de 3x3x2. (La variable x debe estar inicialmente vacía o no existir):

```
» x1=[1 2 3; 4 5 6; 7 8 9]
» x2=[10 20 30; 40 50 60; 70 80 90]
» x(:,:,1)=x1;
» x(:,:,2)=x2;
```

La información quedará organizada como se muestra en la figura:



Dónde la primer dimensión corresponde a las filas, la segunda a las columnas y la tercera a las páginas. Para ver la dimensión de x se puede usar la función size:

```
» size(x)

     3     3     2
```

Un arreglo multidimensional se puede reorganizar de modo de reducir la cantidad de dimensiones, pero conservando la cantidad de elementos. Para ello se usa la función reshape que tiene la siguiente sintáxis:
:

```
arreglo_B = reshape(arreglo_A, dim1, dim2, ...)
```

Dónde *arreglo_A* es el arreglo que se desea convertir, y *dim1*, *dim2*, etc. son las dimensiones del *arreglo_B* que se desea obtener. La cantidad de elementos se debe conservar, por lo tanto el producto de las dimensiones del *arreglo_A*, debe se igual al producto de *dim1*, *dim2*, etc. Esta función opera inicialmente sobre columnas.
Ejemplo. Sea x el arreglo de 3x3x2 del ejemplo anterior, que se desea convertir en una matriz 3x6:

```
» reshape(x, 3, 6)

ans =

     1     2     3    10    20    30
     4     5     6    40    50    60
     7     8     9    70    80    90
```

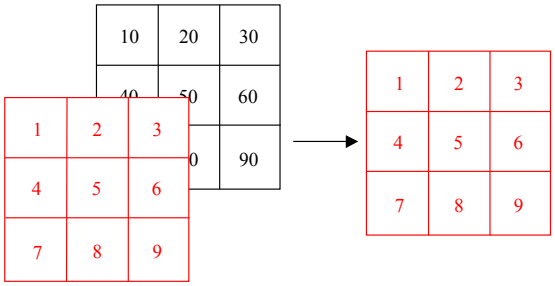
En el caso de querer convertirlo en una matriz de 6x3:

```
» reshape(x, 6, 3)

ans =

     1     3    20
     4     6    50
     7     9    80
     2    10    30
     5    40    60
     8    70    90
```

Donde se observa que la columna 1 está formada por las columnas 1 y 2 de la página 1 de x, la columna 2 está formada por la columna 3 de la página 1 y la columna 1 de la página 2 de x, y así.

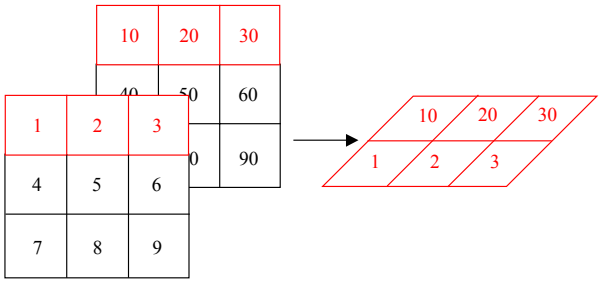
Cuando se hace referencia a un rango de un arreglo, de manera que la última dimensión es de tamaño 1, el resultado se reduce en una dimensión. Por ejemplo la operación de hacer referencia a la primera página de x resulta en una matiz:

```
» a = x(:,:,1);
» size(a)

     3     3
```



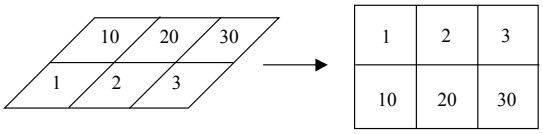
Sin embargo si se hace referencia a las filas 1 del arreglo x, el resultado mantiene tres dimensiones. Esto

```
» a = x(1,:,:)
» size(a)

     1     3     2
```



Esto resulta en una matriz que no está "paralela" a la página. Para eliminar las dimensiones de tamaño 1, se puede usar la función squeeze:

```
» a = squeeze(x(1,:,:))
» size(a)

     3     2
```

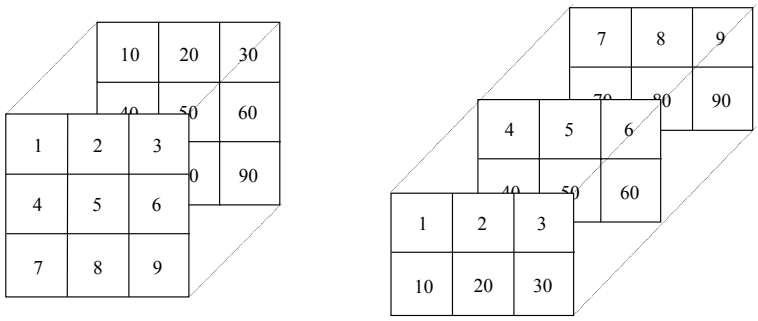


Finalmente existe la posibilidad de permutar dimensiones, esto es "rotar" el cubo que forma al arreglo multidimensional. Para ello se usa la función permute, cuya sintaxis es la siguiente:

```
arreglo_B = reshape(arreglo_A, [d1, d2, ...])
```

Dónde *arreglo_A* es el arreglo cuyas dimensiones de desean permutar, y *d1*, *d2*, etc. indica el nuevo orden de las dimensiones, o sea el que tendrá el *arreglo_B*.
Por ejemplo, para convertir un sistema de 3x3x2 a uno de 2x3x3, se debe .

```
» a = permute(x, [3, 2, 1])
» size(x)

      3     3     2


» size(a)

      2     3     3
```



La funciones y operadores que efectúan los cálculos elemento a elemento sobre vectores y matrices (+ , - , .* , ./ , .^ , etc.), también lo hacen sobre arreglos multidimensionales.

Ejemplo con manejo de arreglos multidimensionales y gráficos de superficies en el espacio.
El objetivo de este ejemplo es graficar la posición angular del péndulo (ejemplo de ODEs) en función del tiempo y del coeficiente de rozamiento. esto resulta en una superficie en el espacio.
Para ello se modificará primeramente la función pendulo, para poder pasarle como argumento el rozamiento, (ver help ode45, help odefile):

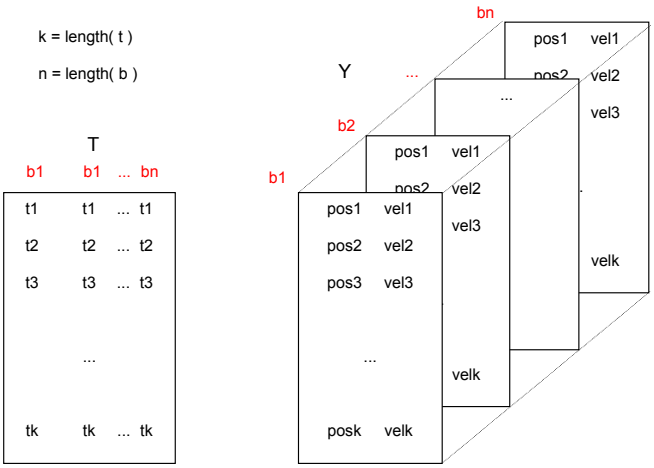
```
function dy = pendulo(t, y, FLAG, b)

l = 1;
m = 1;
g = 9.8;

dy = [y(2); (-y(2)*l*b -m*g*sin(y(1)))/(l*m)];
```

Los argumentos correspondiente a los parámetros de una función ODE, deben estar a partir de la cuarta posición. Por lo tanto en la tercera posición se agrega una variable cualquiera (ver help odefile).

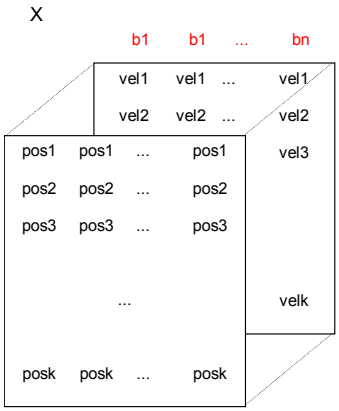
Ahora se debe resolver esta ODE para distintos coeficientes de rozamiento b. Para ello conviene elaborar un programa como el que se muestra a continuación (guardar como "graf3d.m"):

```
clear Y T
% Vector de coeficientes de rozamiento
b=[0:0.1:2];
% Vector de tiempo
t=[0:0.1:6];
% Aquí se resuelve la ODE para cada elemento de b
for n=1:length(b),
   [T(:,n),Y(:,:,n)]=ode45('pendulo', t, [pi/4, 0], [], b(n));
end
% Se permutan la segunda y tercera dimensión de Y
X=permute(Y,[1, 3, 2]);
% Gráficos
surf(b, t, X(:,:,1))
shading interp;
figure
mesh(b, t, X(:,:,2))
```

Cuando se resuelve una ODE con parámetros usando ode45, éstas se deben ubicar del quinto argumento en adelante. Si no se usa el cuarto parámetro (para opciones), se debe colocar una matriz vacía (ver help ode45).
Los resultados de simulación son puestos en la matriz T y el arreglo Y. La matriz T tiene una columna de tiempos para elemento de b. El arreglo Y tiene una página por cada elemento de b, donde cada página tiene la matriz de resultados de posición y velocidad para cada intervalo de tiempo.



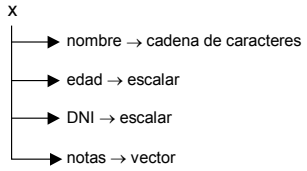
Al permutar la segunda con la tercera dimensión de Y, se obtiene:



Una vez permutadas las dimensiones, el programa grafica la página de posiciones como una superficie continua, y las velocidades como una malla.

Estructuras (structures):

Las estructuras son una clase de variables que tiene "contenedores de datos" con nombre. Estos contenedores de datos se llaman campos. Los campos pueden se de cualquier tipo de datos ya vistos (escalares, matrices, textos, etc.).
Supóngase que se desea organizar la información de alumnos de la siguiente manera:



Con las estructuras, se puede organizar esta información es una sola variable que contirnr "subvariables". Para crear una variable de este tipo se procede de la siguiente manera, suponiendo que x es una variable vacía:

```
» x.nombre = 'Pedro Lopez'
» x.edad = 26;
» x.DNI = 25684752;
» x.notas = [9.5 6.25 7.0 7.75 8.0]
```

Para ver las dimensiones, memoria ocupada y clase de la variable creada:

```
» whos
```

Para ver el contenido de la estructura, se debe escribir el nombre de la variable en el área de trabajo:

```
» x
```

Para acceder a un campo de la variable, se debe escribir el nombre de la variable seguida del nombre del campo separado por un punto, por ejemplo:

```
» x.nombre
```

Si a su vez el campo tiene varios elemento (por ejemplo es un vector) se puede acceder a ellos con la convención usada para arreglos:

```
» y = x.notas(2:3)
```

Dónde la variable y pasa a ser un vector de dos elementos.
Cuando una variable tipo estructura es asignada a otra, se transfiere tanto la información como la estructura de la misma:

```
» z=x
```

Se pueden crear vectores o matrices de estructuras, estructuras de estructuras, etc. :

```
» a(1).nombre = 'Pedro Lopez'
» a(1).edad = 26;
» a(2).nombre = 'Juan Estevez'
» a(3).edad = 25;
...
```

Ejemplo: para modificar las opciones de la función ode45, éstas se deben pasar en forma de estructura. Los nombres de los campos y sus funciones se pueden encontrar en help odeset. Con ellas se puede modificar, por ejemplo, el intervalo máximo de integración, el factor de refinamiento, la tolerancia absoluta, la tolerancia relativa, etc. usada para la resolución de las ODEs.
En este simple ejemplo se activará la opción "Stats", que muestra en la pantalla estadísticas relacionadas con el costo computacional del cálculo realizado:

```
» ops.Stats='on'
» [T,Y]=ode45('pendulo', t, [pi/4, 0], ops);
» plot(T,Y);
```

Listas (cell array):

Es un vector de contenedores. Estos contenedores pueden tener elementos de cualquier clase. A diferencia de las estructuras, las listas no tienen ningún orden (organización).
Para crear una lista se escriben los objetos (o variables que contengan los objeto) separados por comas, entre llaves. En el siguiente ejemplo, se acomodan en una lista un texto, una matriz y un escalar:

```
» x={'Hola', [1 2 3; 4 5 6], 5.248}
```

Para ver las dimensiones, memoria ocupada y clase de la variable creada:

```
» whos
```

Para ver el contenido de la lista, se debe escribir el nombre de la variable en el área de trabajo:

```
» x
```

Para acceder a un elemento de la lista, se debe escribir el nombre de la variable y seguido, entre llaves, el número correspondiente a la posición del mismo dentro de la lista. El resultado será de la misma clase que el elemento contenido. Por ejemplo para acceder al primer elemento de x:

```
» a = x{1}
```

También se puede acceder a un rango de la lista poniendo entre las llaves el rango de elementos que se desea acceder. En este caso el resultado seguirá siendo de la clase lista:

```
» a = x{1:2}
```

Si a su vez el elemento de la lista tiene varios elemento (por ejemplo es un vector) se puede acceder a ellos con la convención usada para arreglos:

```
» a = x{2}(:,1)
```

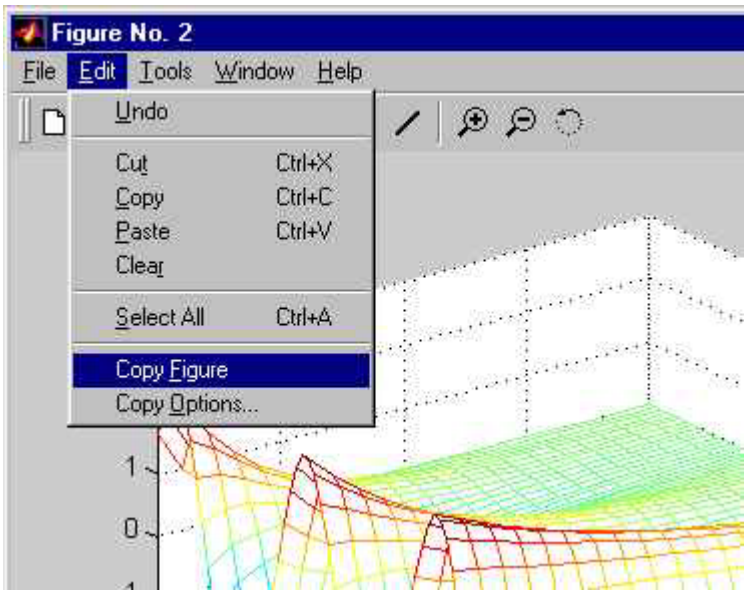
Dónde a pasa a ser un vector columna de dos elementos.

Comunicación con Archivos.

Exportar Gráficos

Los gráficos producidos por Matlab pueden ser copiados al portapapeles del Windows mediante el menú Edit_Copy Figure de la ventana del gráfico. Una vez copiada la figura al portapapeles del

Windows, éste se puede insertar en cualquier aplicación del Windows mediante el menú Edición_Pegar (Edit_Paste) de la aplicación de destino.
Mediante el menú Edit_Copy Options..., se puede establecer el formato gráfico con el se copiará la figura al portapapeles (Metafile o Bitmap). Si se desea pegar la figura en Word, se recomienda usar la opción Metafile, para que ocupe menos memoria en el documento de destino. Por otra parte se recomienda no editar la figura una vez pegada en el Word, debido que se suelen producir cambios indeseados (por ej. el texto con orientación vertical se pierde) y la memoria ocupada por el gráfico se incrementa significativamente. Por lo tanto conviene agregar los rótulos necesarios antes de copiar la figura.



Guardar variables de Matlab en planillas de cálculo.

El Matlab posee la función wk1write que permiten escribir variables en planillas de cálculo (ver help iofun). La sintaxis general de esta función es:

```
wk1write('Nombre_archivo', Variable)
```

Donde *Nombre_archivo* es el nombre del archivo que se creará. El archivo creado tiene formato Lotus (El cual puede se leído por Excel) y si no se especifica la extensión, ésta será ".wk1". el argumento *Variable* es el vector o una matriz que se desea pasar al archivo. Por ejemplo:

```
» wk1write('matriz', [1 2 3; 4 5 6; 7 8 9])
```

El archivo "matriz.wk1" se creará en el directorio actual, y puede ser leído con el Lotus o Excel.

Leer variables del Matlab desde planillas de cálculo.
Para leer celdas de una planilla de cálculo y asignarlas a una variable, se debe usar la función wk1read, cuya sintaxis general es:

```
Variable = wk1read('Nombre_archivo'
```

El primer argumento *Nombre_archivo* es el nombre del archivo, cuyo contenido se desea asignar a *Variable*. El archivo debe estar en formato Lotus y si no se escribe la extensión se asume ".wk1".

```
Variable = wk1read('Nombre_archivo', F, C)
```

En este caso se leen las celdas a partir de la fila *F* y la columna *C*. Las filas y columnas inician en cero.

```
Variable = wk1read('Nombre_archivo', F, C, Rango)
```

El argumento opcional *Rango* indicar el rango de celdas que se desean leer. Se puede especificar cómo *Rango* = [*F1 C1 F2 C2*], dónde (*F1 C1*) es la celda superior izquierda y (*F2 C2*) es la celda inferiro derecha del rango que se desea leer. Otra forma de especificar es con la notación de planilla de cálculo, por ejemplo *Rango* = 'A1..B7'.

Por ejemplo para leer la planilla "matriz.wk1" ubicada el directorio actual, desde la primer fila y la segunda columna:

```
» H=wk1read('matriz', 0, 1)
```

Para leer el mismo archivo, pero restringido a las tres primeras filas y las dos primeras columnas:

```
» H=wk1read('matriz', 0, 1, [1 1 3 2])
```

ó

```
» H=wk1read('matriz', 0, 1, 'A1..B3')
```

Guardar/leer variables de Matlab hacia/desde archivos de texto.
Las función dlmwrite permite escribir vectores o matrices en archivos de texto(ver help iofun).
Sintaxis:

```
» dlmwrite('Nombre_archivo', X, 'Delim')
```

El vector o matriz *X* se guardará en el archivo *Nombra_archivo* separando las columnas con el caracter delimitador *Delim* (puede ser un espacio, una coma, '\t' para una tabulación, etc.) y las filas por un salto de línea. Se debe notar que para ahorrar espacio, cualquier elemento nulo de la matriz será omitido. Por ejemplo el vector [2 4 0 1] se guardará como '2,4,,1'.
Ejemplo:

```
» dlmwrite('matriz.txt', [1 2 3; 4 5 6; 7 8 9], ' ')
```

Para leer variables desde archivos de texto se puede usar la función dlmread, cuya sintaxis es la siguiente:

```
» X = dlmread('Nombre_archivo', 'Delim')
```

Dónde *Nombre_archivo* deberá ser un archivo con números organizados en columnas separadas por espacios, comas, tabulaciones, etc. y las filas por saltos de línea. El delimitador se deberá especificar en *Delim* usando '\t' para el caso de que sean tabulaciones. El separador de decimales de las cifras deberán ser puntos, y no deberá tener separador de miles.