Práctica 1
Introducción a Matlab I

Objetivo. El objetivo de esta práctica es aclarar la dinámica de trabajo y de evaluación de esta materia así como una introducción al ambiente de trabajo de Matlab, el manejo de variables y constantes y los comandos básicos para trabajar con matrices y vectores.

Dinámica de Trabajo de esta Materia:

1) El alumno deberá haber leído el instructivo de cada práctica antes de cada sesión.
2) Para asegurarse de lo anterior, el profesor comenzará la sesión haciendo preguntas orales al azar sobre los temas que trata el instructivo.
3) Al final de cada sesión, el profesor aplicará una evaluación individual (de 15 minutos) por escrito, de la cual obtendrá una calificación de la sesión.
4) En algunas prácticas se dejará de tarea elaborar un reporte o resolver algún problema o problemas, lo cual se deberá entregar en la siguiente sesión.
5) Se aplicarán tres exámenes parciales, los cuales podrán consistir en proyectos a realizar o en exámenes por escrito.
6) La calificación final de la materia considerará: asistencia a las sesiones, respuestas a las preguntas orales en clase, evaluaciones escritas de cada sesión, exámenes parciales y reportes.

Introducción.

MATLAB (Matrix Laboratory) es un paquete de herramientas de software desarrollado por *The MathWorks Inc.*, compañía fundada por Cleve Moler, Jack Little and Steve Bangert en 1984. (http://www.mathworks.com). MATLAB fue diseñado para realizar cálculos numéricos, especialmente cálculos basados en matrices y álgebra lineal, análisis y visualización de datos y para facilitar la escritura de nuevos programas dentro de este tipo de objetivos.

La ventaja de Matlab es que combina funciones matemáticas y gráficas con un poderoso lenguaje interpretado de alto nivel.

Matlab es un lenguaje de programación amigable para hacer cálculos numéricos o matemáticos con características más avanzadas y mucho más fácil de usar que los lenguajes de computadoras tales como Basic, Pascal, C o Java, proporcionando un entorno rico para la visualización de datos a través de sus poderosas capacidades gráficas.

Matlab es una plataforma de desarrollo de aplicaciones, para la cual se han desarrollado poderosos conjuntos de herramientas para la resolución de problemas en áreas específicas, a menudo llamadas toolboxes. Las áreas en que los toolboxes están disponibles incluyen Procesamiento de señales, Diseño de sistemas de control, Simulación de sistemas dinámicos, Identificación de sistemas, Estadística, Lógica difusa, Redes neuronales entre otras. Además el usuario puede desarrollar sus propios toolboxes.

☞ Como Matlab es un lenguaje interpretado, algunas secuencias son más lentas que en lenguajes compilados, especialmente las que involucran ciclos escritos por el usuario. Por esta razón una primera recomendación es evitar al máximo esta situación y escribir los ciclos siempre que sea posible en forma vectorizada o recurrir a funciones incorporadas en Matlab que realicen lo que se quiere hacer con el ciclo.

Las versiones más recientes de Matlab incorporan soporte para crear y manipular instancias de clases de Java directamente en los progamas de Matlab. También uno puede hacer llamadas a código escrito en C, C++, Perl, Fortran, o ejecutar comandos del sistema operativo DOS o UNIX. Adicionalmente, el código de Matlab puede ser exportado para usarse en programas escritos en Java, C, C++, y .Net, e inclusive se pueden generar aplicaciones ejecutables (stand alone applications).

Desde su creación hasta la fecha se han desarrollado una gran cantidad de versiones de Matlab. Este manual de prácticas está basado en la versión 7.10 (Versión R2010a), sin embargo, prácticamente todos los temas cubiertos funcionan sin problema para las versiones posteriores.

Iniciando Matlab

Matlab para Windows puede iniciarse seleccionándolo desde el icono de Windows. Al ejecutarse Matlab crea una ventana como la mostrada en la figura 1.1. La ventana principal es la denominada Ventana de comandos (Command Window) que muestra un mensaje inicial y termina con un apuntador consistente en dos signos ">>" seguidos del cursor parpadeante indicando que está listo para recibir comandos en esa línea (Línea de comandos).

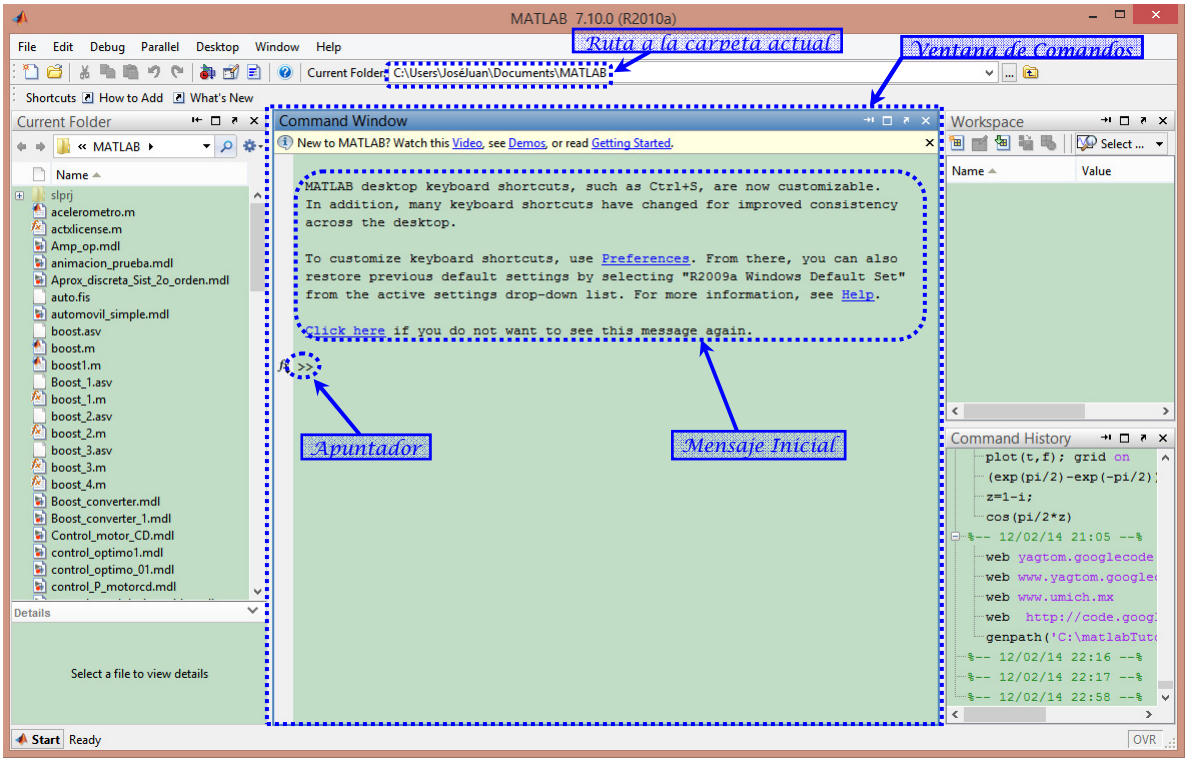


Figura 1.1.- Ventana principal de Matlab

Casi todas las actividades de las prácticas se realizarán en la ventana de comandos, sin embargo, en la figura 1.1 se muestran también otras ventanas que aparecen por default la primera vez que se ejecuta Matlab:
Current Folder.- Explorador de archivos, visualiza los guardados en la carpeta actual.
Workspace.- Explorador de los datos que se van creando o importando de archivos.
Command History.- Visualiza y permite re-ejecutar las instrucciones introducidas en la línea de comandos.

Uso de Matlab como una Calculadora.
La manera más sencilla de utilizar Matlab es introduciendo en la línea de comandos las operaciones que se desea realizar y el resultado se obtiene inmediatamente al terminar el comando con la tecla [enter] en la variable `ans`.

Ejemplo:
```
>> (4+8)/2
ans =
     6
```

Ejemplo: Usando una variable
```
>> x=4+8;
>> x/2
ans =
     6
```

Operaciones aritméticas básicas:
De la misma forma que una calculadora, Matlab ofrece las siguientes operaciones aritméticas básicas:

Operación	Símbolo	Ejemplo	Precedencia
Suma	+	5 + 3	3
Resta	–	23 – 12	3
multiplicación	*	20.5 *1.5	2
división	/ o \	56/8 = 8\56	2
Potencia	^	5^2	1

Precedencia de las operaciones básicas:
Las expresiones se evalúan de izquierda a derecha, con la operación de potencia teniendo la orden de precedencia más alta, seguida por la multiplicación y división que tienen ambas igual precedencia y seguidas finalmente por suma y resta que tienen ambas igual precedencia. Se pueden emplear paréntesis para alterar esta usual ordenación.

Ejemplo: Para evaluar la expresión $\left(\frac{8}{7}\right)\times\left(\frac{5}{4}\right)=\frac{8\times5}{7\times4}$ sin usar paréntesis se puede realizar como sigue:
```
>> 8*5/7/4
ans =
    1.4286
```
o también:
```
>> 8/7*5/4
ans =
    1.4286
```

Formato de visualización de números y su representación interna.

Las visualización de las cantidades numéricas se pueden obtener en varios formatos. Los formatos de visualización de números no cambian la representación interna de un número, solamente su visualización. El formato por default es short. Los formatos de visualización disponibles son:

Comando	Visualización de 1/6	Comentario
`format long`	`0.166666666666667`	16 dígitos
`format short e`	`1.6667e-001`	5 dígitos más exponente
`format long e`	`1.666666666666667e-001`	16 dígitos más exponente
`format hex`	`3fc5555555555555`	Hexadecimal
`format bank`	`0.17`	2 dígitos decimales
`format +`	`+`	positivo, negativo o cero
`format rat`	`1/6`	aproximación racional
`format short`	`0.1667`	visualización por defecto

Variables intrínsecas de Matlab.

Matlab proporciona valores predefinidos de acuerdo a la siguiente tabla

Variable	Valor predefinido	Comentario
`i, j`	$\sqrt{-1}$	Unidad imaginaria (permite definir números complejos)
`pi`	$\pi = 4\tan^{-1}(1)$	180^o en radianes
`inf`	∞	Resultado Infinito, por ejemplo: 1/0
`NaN`	No es un Número	Operación no definida, por ejemplo: 0/0, inf-inf

Las variables de la tabla anterior pueden ser redefinidas asignándole valores arbitrarios, en este caso, hay que tener cuidado.

Ejemplo: La siguiente expresión calcula el área de un círculo de radio 1.5:

```
>> pi*1.5^2
ans=
   7.0686
```

pero si se redefine la variable `pi` con otro valor, el resultado no tendrá nada que ver con el área del círculo:

```
>> pi=4;
>> pi*1.5^2
ans =
    9
```

Ejemplo: se pueden hacer operaciones aritméticas con números complejos, por ejemplo

```
>> (1+i)*(2-i)
ans =
  3.0000 + 1.0000i
```

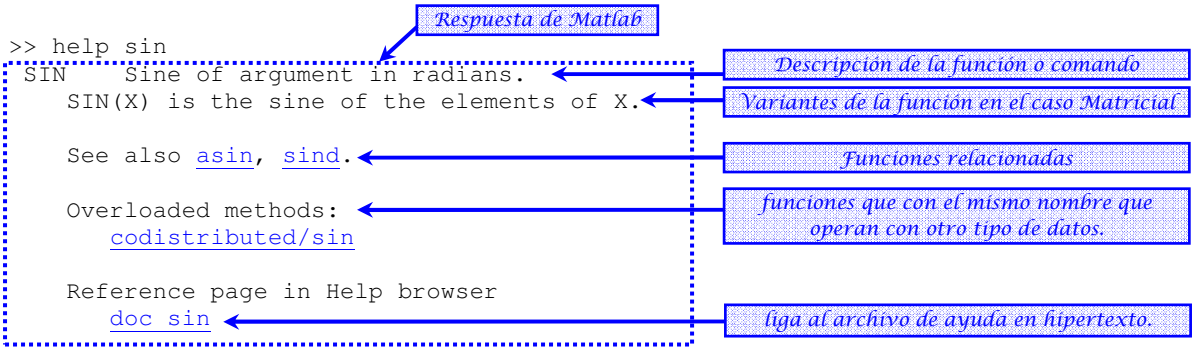
pero si se redefine la variable `i` con otro valor, el resultado no tendrá nada que ver con la multiplicación de números complejos:

```
>> i=1;
>> (1+i)*(2-i)
ans =
    2
```

Ayuda en Línea.

Matlab tiene un comando llamado `help` muy bien documentado. Escribiendo `help`, se despliega

una lista muy larga de tópicos sobre los cuales Matlab puede proporcionar ayuda. También se puede solicitar ayuda sobre un comando o función específica de Matlab, por ejemplo si deseamos ayuda sobre la instrucción `sin`:



Al final Matlab proporciona una liga (`doc sin`) al archivo de ayuda en hipertexto, el cual permite navegar de manera más interactiva y más completa sobre la información acerca de la función `sin`, incluyendo ejemplos sencillos pero ilustrativos de cómo usarla. En el recuadro siguiente se muestra la información correspondiente a la función `sin`.

sin

Sine of argument in radians

Syntax

```
Y = sin(X)
```

Description

`Y = sin(X)` returns the circular sine of the elements of `X`. The `sin` function operates element-wise on arrays. The function's domains and ranges include complex values. All angles are in radians.

Definitions

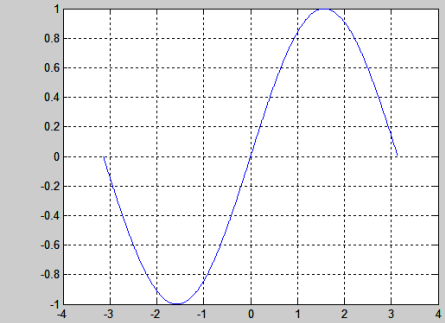
The sine of an angle is: $\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$

For complex x: $\sin(x) = \sin(x)\cosh(x) + i\cos(x)\sinh(x)$

Examples

Graph the sine function over the domain .

```
x = -pi:0.01:pi;
plot(x,sin(x)), grid on
```



References

`sin` uses FDLIBM, which was developed at SunSoft, a Sun Microsystems, Inc. business, by Kwok C. Ng, and others. For information about FDLIBM, see `http://www.netlib.org`.

See Also

`sind`

Manejo de Matrices y Vectores.
Las Matrices son el principal tipo de datos que maneja MATLAB. La manera más sencilla de escribir una matriz es como una lista de elementos. Solamente hay que seguir unas convenciones básicas:

- Toda matriz es una arreglo rectangular de elementos y está organizada por:
  - renglones o filas (en forma horizontal)
  - columnas (en forma vertical).
- Los elementos de una fila se escriben separados con espacios o comas.
- Se Usa ; (punto y coma) para indicar el fin de cada renglón (fila).
- Se encierra la lista entera de elementos con paréntesis cuadrados, [ ].

Ejemplo: Para introducir la siguiente matriz de dos filas y dos columnas: $A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$:

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

Una matriz de una sola fila o de una sola columna se denomina ***vector***:

Ejemplo: Introducir el vector fila $b=\begin{bmatrix}5 & 6\end{bmatrix}$ y el vector columna $c=\begin{bmatrix}7\\ 8\end{bmatrix}$

```
>> b=[ 5 6]
b =
   5 6
>> c=[ 7;8]
c =
   7
   8
```

Para evitar el despliegue de los resultados en la pantalla cada vez que se asigna una variable use el operador ";" al final de la expresión, para el ejemplo anterior:

```
>> b=[ 5 6];
>> c=[ 7;8];
```

Concatenando matrices:
Se pueden juntar dos matrices con el mismo número de columnas o con el mismo número de renglones para formar una matriz más grande:

Ejemplo:

```
>> A=[1 2 3; 4 5 6;7 8 9];
>> R=[10 11 12];
>> B=[A ; R]        %Junta dos matrices de tres columnas
B =
     1     2     3
     4     5     6
     7     8     9
    10    11    12
>> C=[10; 11; 12];
>> B=[A R]          %Junta dos matrices de tres filas
B =
     1     2     3   10
```

```
     4     5     6    11
     7     8     9    12
```

Submatrices:
Si `A` es una matriz, la notación `A(i:j,m:n)` se refiere a la submatriz formada por los elementos desde la fila `i` hasta la fila `j` y desde la columna `m` hasta la columna `n`.

☞ Obsérvese que i, j, m, n sólo pueden ser números enteros mayores a cero y menores al tamaño correspondiente de la matriz.

Ejemplo: Si B es la matriz del ejemplo anterior

```
>> C=B(2:3,2:3)
C =
     5     6
     8     9
```

Manipulación de matrices elemento a elemento:
Si `A` es una matriz, la notación `A(i,j)` hace referencia al elemento individual de la fila `i` y la columna `j`. Esto permite leer o modificar cada elemento de matriz en base a su posición renglón columna especificados por los índices `i, j`

El símbolo dos puntos ":" permite referirse a un rango de valores enteros, lo cual puede ser usado para referirse a un rango de filas o de columnas.

Ejemplo: Generar todos los números enteros del 1 al 20, con incrementos de 2 en 2

```
x=1:2:10
x =
     1     3     5     7     9
```

Si el incremento no se especifica, se toma por default 1:

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

Usado como índice, la notación de dos puntos permite omitir no solamente el incremento, sino el inicio y el final:

Ejemplo: Partiendo del ejemplo anterior

```
>> x(1:5) %los elementos del 1 al 5
ans =
     1     2     3     4     5
>> x(:)  %los elementos desde el primero hasta el último en forma de columna
ans =
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
```

Es decir, el uso de dos puntos para la designación de filas y columnas implica, respectivamente,

todas las filas o columnas;

Ejemplo: `A(:,3)` representa todas las filas en la columna tres, `A(2,:)` representa todas las columnas en la fila dos

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

☞ **Observaciones**:

- Usar sólo dos puntos, por ejemplo `A(:)`, reagrupa todos los elementos de una matriz en un vector columna.
- Colocar datos fuera del rango actual de una matriz rellena con ceros las zonas no especificadas, manteniendo la forma rectangular de la matriz.
- Fijar las filas o columnas de una matriz igual a la matriz vacía `[ ]` elimina estas filas o columnas.
- Los valores lógicos `0` y `1` pueden utilizarse para seleccionar partes de un vector. Los elementos falsos (`0`) se eliminan, los elementos verdaderos (`1`) se retienen.
- Los valores lógicos pueden ser generados como resultado de desigualdades.

Ejemplo: Prosiguiendo con la matriz del ejemplo anterior

```
>> A(:)  %%%% Convierte a columna
ans =
     1
     4
     7
     2
     5
     8
     3
     6
     9
>> A(2,5)=10 %%%% Agrega un elemento fuera de rango
A =
     1     2     3     0     0
     4     5     6     0    10
     7     8     9     0     0
>> A(:,4:5)=[] %%%% Elimina las columnas 4 y 5
A =
     1     2     3
     4     5     6
     7     8     9
>> B=A(:)'   %%%% Convierte a renglón
B =
     1     4     7     2     5     8     3     6     9
>> C = (B>5)  %%%% Genera un vector de valores lógicos
C =
     0     0     1     0     0     1     0     1     1
>> B(C)       %%%% Selecciona los elementos de B con valor mayor a 5
```

```
ans =
     7     8     6     9
```

Ejemplos varios:

```
» A=[1 2 3; 4 5 6; 7 8 9]  %Crea la matriz A
» A(3,3)=0                 %Cambia un elemento de la matriz
» A(2,6)=1;                %Agrega un elemento fuera de rango
» A=[1 2 3; 4 5 6; 7 8 9]  %Vuelve a crear la matriz A
» B=A(3:-1:1,:)            %Crea la matriz B con las filas A en orden inverso
» C=[A B(:,[1 3])]         %Añade la 1ª y 3ª columna de B a la derecha de A
» B=A(1:2,2:3)             %Extrae submatriz de A
» B=A(:)                   %Convierte A en vector columna
» B=B'                     %transpone la columna para convertirla a fila
» B(:,2)=[]                %Elimina la segunda columna de B
» B=B'                     %Transpone una matriz
» A=B                      %Copia la matriz B en A
» B(2,:)=[]                %Elimina la segunda fila de B
» x=-3:0.5:3               %Crea un vector desde -3 a 3 con incrementos de 0.5
» y=abs(x)>0               %Marca con 1's los elementos positivos de x
» y=x(y)                   %Selecciona los elementos positivos de x
» y=x([1 1 1 1])           %Crea y tomando el primer elemento de X tres veces
» x(abs(x)<2)=[]           %Elimina valores de x tales que -2<x<2
```

Matrices con elementos complejos.

Matlab permite el manejo de números complejos como ya se dijo, mediante el uso de las variables predefinidas: `i=j=`$\sqrt{-1}$.

Ejemplo: Para introducir la siguiente matriz con elementos complejos: $A=\begin{bmatrix} 1+i & 2-3i \\ 3-3i & 4+2i \end{bmatrix}$, se puede proceder como sigue:

```
>> A=[1+j 2-3j; 3-3j 4+2j];
```

ó bien, separando en parte real y parte imaginaria:

```
>> A=[1 2; 3 4]+i*[1 -3; -3 2];
```

El conjugado y el traspuesto de una matriz.

El operador comilla (') obtiene el traspuesto conjugado de una matriz, si se desea solamente trasponer la matriz se debe usar punto comilla (.') o bien, la función `conj`.

Ejemplo: Partiendo del ejemplo anterior

```
>> A'    %%% Obtiene la matriz traspuesta y conjugada
ans =
   1.0000 - 1.0000i   3.0000 + 3.0000i
   2.0000 + 3.0000i   4.0000 - 2.0000i
>> A.'    %%% Obtiene la matriz traspuesta sin conjugar
ans =
   1.0000 + 1.0000i   3.0000 - 3.0000i
   2.0000 - 3.0000i   4.0000 + 2.0000i
>> conj(A')    %%% Misma operación que A.'
ans =
   1.0000 + 1.0000i   3.0000 - 3.0000i
   2.0000 - 3.0000i   4.0000 + 2.0000i
```

Operaciones Aritméticas con matrices.

Las operaciones aritméticas básicas: suma, resta, multiplicación, e inclusive división se pueden realizar entre matrices de dimensiones compatibles mediante los operadores +, -, *, /.

☞ En el caso de la división matricial, ésta debe interpretarse como sigue, Si A y B son matrices de dimensiones compatibles (número de filas de B igual a número de columnas de A), y si B tiene inversa, entonces

$$A / B = A * B^{-1}$$

Además de las operaciones aritméticas básicas entre matrices anteriores, Matlab dispone de operaciones especiales elemento a elemento, las cuales se enumeran en la siguiente tabla, en la cual se supone que `c` es un escalar arbitrario y las matrices `A` y `B` siguientes:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ & & \dots & \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix},\ B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ & & \dots & \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix}$$

Operaciones elemento a elemento con matrices	
Suma con escalar `A+c = c+A`	$\mathtt{A+c} = \begin{bmatrix} a_{11}+c & a_{12}+c & \dots & a_{1m}+c \\ a_{21}+c & a_{22}+c & \dots & a_{2m}+c \\ & & \dots & \\ a_{n1}+c & a_{n2}+c & \dots & a_{nm}+c \end{bmatrix}$

Multiplicación por escalar
`A*c = c*A =`
`A.*c = c.*A`
	$\mathtt{A*c} = \begin{bmatrix} a_{11}*c & a_{12}*c & \dots & a_{1m}*c \\ a_{21}*c & a_{22}*c & \dots & a_{2m}*c \\ & & \dots & \\ a_{n1}*c & a_{n2}*c & \dots & a_{nm}*c \end{bmatrix}$

Multiplicación elemento a elemento `A.*B = B.*A`	$\mathtt{A.*B} = \begin{bmatrix} a_{11}*b_{11} & a_{12}*b_{12} & \dots & a_{1m}*b_{1m} \\ a_{21}*b_{21} & a_{22}*b_{22} & \dots & a_{2m}*b_{2m} \\ & & \dots & \\ a_{n1}*b_{n1} & a_{n2}*b_{n2} & \dots & a_{nm}*b_{nm} \end{bmatrix}$
División elemento a elemento `A./B`	$\mathtt{A./B} = \begin{bmatrix} a_{11}/b_{11} & a_{12}/b_{12} & \dots & a_{1m}/b_{1m} \\ a_{21}/b_{21} & a_{22}/b_{22} & \dots & a_{2m}/b_{2m} \\ & & \dots & \\ a_{n1}/b_{n1} & a_{n2}/b_{n2} & \dots & a_{nm}/b_{nm} \end{bmatrix}$

Elevación a Potencia elemento a elemento
`A.^c`
`c.^A`
`A.^B`
	$\mathtt{A.\hat{}c} = \begin{bmatrix} a_{11}^{\,c} & a_{12}^{\,c} & \dots & a_{1m}^{\,c} \\ a_{21}^{\,c} & a_{22}^{\,c} & \dots & a_{2m}^{\,c} \\ & & \dots & \\ a_{n1}^{\,c} & a_{n2}^{\,c} & \dots & a_{nm}^{\,c} \end{bmatrix}$ $\mathtt{c.\hat{}A} = \begin{bmatrix} c^{a_{11}} & c^{a_{12}} & \dots & c^{a_{1m}} \\ c^{a_{21}} & c^{a_{22}} & \dots & c^{a_{2m}} \\ & & \dots & \\ c^{a_{n1}} & c^{a_{n2}} & \dots & c^{a_{nm}} \end{bmatrix}$
	$\mathtt{A.\hat{}B} = \begin{bmatrix} a_{11}^{\,b_{11}} & a_{12}^{\,b_{12}} & \dots & a_{1m}^{\,b_{1m}} \\ a_{21}^{\,b_{21}} & a_{22}^{\,b_{22}} & \dots & a_{2m}^{\,b_{2m}} \\ & & \dots & \\ a_{n1}^{\,b_{n1}} & a_{n2}^{\,b_{n2}} & \dots & a_{nm}^{\,b_{nm}} \end{bmatrix}$

Ejemplo: Encontrar la solución del sistema de tres ecuaciones con tres incógnitas siguiente

$$x_1 + x_2 + x_3 = 1$$
$$x_2 + x_3 = 2$$
$$x_3 = 3$$

```
>> A=[1 1 1;0 1 1;0 0 1];  %se define la matriz del sistema
>> b=[1 2 3]';             %se define el vector columna de términos independientes
>> x=A^-1 *b               %se obtiene la solución
x =
    -1
    -1
     3
>> x=inv(A)*b              %inv(A) es lo mismo que A^-1
x =
    -1
    -1
     3
```

Ejemplos varios: Continuando con el ejemplo anterior

```
>> A^-1       %%% Obsérvese la diferencia del operador ^
>> A.^-1      %%% y el operador .^
>> A^3        %%% Calcula el producto matricial A*A*A
>> A.^3       %%% Eleva al cubo cada elemento de A
>> B=A*A'     %%% Obtiene una nueva matriz simétrica
>> A*B        %%% Obsérvese la diferencia entre la multiplicación matricial *
>> A.*B       %%% y la multiplicación elemento a elemento .*
```

Ejemplos de Errores comunes: Continuando con el ejemplo anterior se pueden intentar las siguientes operaciones que fallarán por no respetar las dimensiones de las matrices y su compatibilidad con las operaciones:

```
>> b^2        %%%% Intento de multiplicar un vector columna por si misma
??? Error using ==> mpower
Inputs must be a scalar and a square matrix.
>> b*A        %%% Intento de multiplicar matrices de dimensiones incompatibles
??? Error using ==> mtimes
Inner matrix dimensions must agree.
>> A(3,4)     %%% Intento de acceder a un elemento fuera de las dimensiones de A
??? Index exceeds matrix dimensions.
>> A/0        %%% Obsérvese que dividir entre cero NO ES ERROR
ans =
   Inf   Inf   Inf
   NaN   Inf   Inf
   NaN   NaN   Inf
```

Funciones intrínsecas de Matlab

Matlab proporciona un número gigantesco de funciones. Algunas funciones son intrínsecas o construidas en el propio núcleo ejecutable de Matlab. Otras están disponibles en librerías externas archivos-M distribuidos con Matlab (Toolboxes). Y otras son adicionadas por los usuarios, o grupo de usuarios, para alguna aplicación especifica en archivos-M.

Usando el comando `help <nombre de Función>` Matlab despliega una explicación concisa (en inglés) sobre la función así como del parámetro o parámetros necesarios para su correcta ejecución y de los resultados que produce.

Ejemplo:

```
>> help sqrt
 SQRT   Square root.
    SQRT(X) is the square root of the elements of X. Complex
    results are produced if X is not positive.

    See also sqrtm, realsqrt, hypot.

    Overloaded methods:
       codistributed/sqrt
    Reference page in Help browser
       doc sqrt
```

Funciones básicas con matrices:

Se puede obtener un listado de las funciones matemáticas elementales de Matlab con el comando `help elfun`.

La mayoría de las funciones de Matlab pueden operar tanto sobre escalares como sobre matrices, cuando operan sobre matrices la función la aplican elemento a elemento, a menos que por definición se especifique lo contrario.

Ejemplo:

```
>> A=[1 1 1;0 1 1;0 0 1]*pi/2     %Define una matriz
A =
    1.5708    1.5708    1.5708
         0    1.5708    1.5708
         0         0    1.5708
>> sin(A)                         %Calcula el seno de cada elemento
ans =
     1     1     1
     0     1     1
     0     0     1
>> cos(A)                         %Calcula el coseno de cada elemento
ans =
    0.0000    0.0000    0.0000
    1.0000    0.0000    0.0000
    1.0000    1.0000    0.0000
>> exp(A)                         %Calcula el exponencial de cada elemento e.^A
ans =
    4.8105    4.8105    4.8105
    1.0000    4.8105    4.8105
    1.0000    1.0000    4.8105
>> expm(A)                        %Calcula la matriz exponencial e^A
ans =
    4.8105    7.5563   13.4910
         0    4.8105    7.5563
         0         0    4.8105
```

La siguiente es una lista de funciones básicas útiles que operan sobre matrices:

Función	Descripción
`det(A)`	Determinante de la matriz cuadrada A
`inv(A)`	Inversa de la matriz cuadrada A
`eig(A)`	Valores y vectores propios de la matriz cuadrada A
`[n,m]=size(A)`	Dimensiones de la matriz A (n renglones, m columnas)
`length(A)`	Mayor dimensión de A
`zeros(n,m)`	Genera una matriz de puros ceros de dimensiones nxm

`ones(n,m)`	Genera una matriz de puros unos de dimensiones nxm
`eye(n)`	Genera una matriz identidad de dimensiones nxn
`magic(n)`	Genera matriz nxn cuyos renglones, columnas y diagonales suman lo mismo.
`linspace(a,b,n)`	Genera vector fila de n valores desde a hasta b en incrementos constantes de (b-a)/(n-1). Equivale a `a:(b-a)/(n-1):b`
`logspace(a,b,n)`	Genera vector fila de n valores desde 10^a hasta 10^b en incrementos exponenciales. Equivale a `10.^linspace(a,b,n)`

El espacio de Trabajo de Matlab:
Se denomina espacio de trabajo al conjunto de variables creadas por el usuario en cada sesión. El espacio de trabajo está vacío al inicio de cada sesión, o después de un comando `clear`. Se puede obtener información en el espacio de trabajo con el comando `who` o `whos`; por ejemplo:

Ejemplo:
```
>> clear %%% Limpia el espacio de trabajo
>> who   %%% obsérvese que el espacio de trabajo está vacío
>> A=magic(3)
A =
     8     1     6
     3     5     7
     4     9     2
>> [n,m]=size(A)
n =
     3
m =
     3
>> who
Your variables are:
A  m  n

>> whos
  Name      Size            Bytes  Class     Attributes

  A         3x3                72  double
  m         1x1                 8  double
  n         1x1                 8  double
```

Guardar el espacio de trabajo de una sesión: Comandos *save* y *load*
Cuando uno ejecuta el comando `clear` o cierra la sesión de Matlab, el espacio de trabajo se pierde y todas las variables creadas así como sus valores desaparecen de la memoria. En ocasiones puede ser necesario interrumpir el trabajo con Matlab y poder recuperarlo más tarde en el mismo punto en el que se dejó (con el mismo espacio de trabajo.

Para guardar el estado de una sesión existe el comando `save`. Si se teclea
```
» save
```

el espacio de trabajo se guarda en el archivo `matlab.mat` y puede recuperarse la siguiente vez que se arranque el programa con el comando
```
» load
```

si no se desea guardar el espacio de trabajo completo se pueden guardar también matrices y vectores de forma selectiva y en archivos con nombre especificado por el usuario. Por ejemplo, el comando (sin comas entre los nombres de variables)
```
» save filename A x y
```

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

```
» load filename
```

Ejemplo

```
>> who        %%% checamos como está el espacio de trabajo
Your variables are:
A    ans  m    n
>> save prueba A m n %%% salvamos algunas variables
>> clear      %%% borramos el espacio de trabajo
>> who        %%% checamos espacio de trabajo vacío
>> load prueba      %%% rescatamos las variables salvadas
>> who
Your variables are:
A  m  n
```

Guardar sesión y copiar salidas: Comando *diary*

El comandos `save` crea archivos binarios o ASCII que guardan el estado de la sesión. También se puede almacenar en un archivo de texto toda la actividad (comandos introducidos y respuestas obtenidas) en una sesión o fragmento de sesión de Matlab. Esto se hace con el comando `diary` en la forma siguiente:

```
» diary filename.txt       % comienza la ejecución de diary
…
» diary off          % suspende la ejecución de diary
….
» diary on           %reanuda la ejecución de diary
…
```

El simple comando `diary` pasa de `on` a `off` y viceversa. Para poder acceder al fichero `filename.txt` con *Notepad* o *Word* es necesario que `diary` esté en `off`. Si no se especifica el nombre del archivo, se utiliza por defecto un archivo llamado `diary` (sin extensión).

Desarrollo de la Práctica.

Durante el desarrollo de esta práctica se presenta el funcionamiento del ambiente de trabajo de Matlab, en cada uno de las explicaciones se dan ejemplos del funcionamiento del ambiente de Matlab, y se proponen algunos ejercicios.

1. Probar todos los ejemplos propuestos por el profesor conforme los va explicando.
2. Realizar todos los ejercicios propuestos.
3. Contestar el cuestionario de evaluación de la práctica.

Reportar:

1. Reportar el listado de funciones elementales que se obtiene con el comando `help elfun`.
2. Elegir de la lista, dos de las funciones más raras que te llamen la atención (no vistas en esta clase) y explicar con tus propias palabras para que sirven. Dar ejemplos de aplicación de cada una éstas dos.