NOTAS DE MATLAB

MatLab como calculadora

Operaciones con números: "+", "−", "·", "/".
Todas ellas son operaciones **binarias**.

```
>> 5.321+4.127
ans=
    9.448
```

Si encadenamos las operaciones

```
>>2.7+3.21*2.21^2/7+5.21
```

todas ellas siguen siendo operaciones binarias, pero ¿en qué orden? Primero se calculan las potencias

$$2.7 + 3.21 * (2.21^2)/7 + 5.21$$

a continuación los productos (incluidos los cocientes, conviene pensar que `/7` significa `*(1/7)`) agrupados de izquierda a derecha

$$2.7 + ((3.21 * (2.21\text{^}2)/7) + 5.21$$

finalmente sumas (y restas)

$$(2.7 + ((3.21 * (2.21\text{^}2))/7)) + 5.21$$

en caso de duda, sobre todo al principio, pon paréntesis.
Como calculadora MatLab puede utilizar funciones predefinidas. A continuación se listan las más habituales con su notación MatLab y con su notación usual.

`log`	log ó ln	`sin`	sin	`fix`	Redondeo hacia cero
`log10`	$\log_{10}$	`sinh`	sinh	`floor`	Redondeo hacia $-\infty$
`log2`	$\log_2$	`asin`	arcsin	`ceil`	Redondeo hacia $+\infty$
`exp`	$e^{\cdot}$	`cos`	cos	`round`	Redondeo al entero más cercano
`sqrt`	$\sqrt{\cdot}$	`cosh`	cosh	`rem`	Resto en la división entera
`pow2`	$2^{\cdot}$	`acos`	arccos	`mod`	Resto en la división entera

Las funciones `rem` y `mod` pueden diferir en el signo: `rem` es siempre positivo.

La función `help` de MatLab nos ayudará a ver qué otras funciones están predefinidas. Esta función será una de las que usemos con más frecuencia, sobre todo al principio. Prueba con

```
>> help
>> help help
>> help matlab\elfun
```

Variables

Usualmente utilizaremos MatLab con variables en vez de números. Las variables se definen automáticamente dándoles un valor; el nombre de las variables puede ser cualquier combinación de números y letras que no comience por un número (la letra ñ no vale), las mayúsculas y las minúsculas son distintas: `long` y `Long` son distintas. Esto último es cierto también para las funciones: `Sin(.5)` ó `SIN(.5)` darán error, habrá que escribir `sin(.5)`.

El resultado de cualquier operación queda almacenado en una variable denominada `ans` (del inglés *answer*) de forma que este nombre de variable no debe utilizarse ya que se modifica tras cada operación. Otros nombres de variables que están predefinidos son `eps`, `realmax`, `realmin`, `i`, `j`.La constante `eps` es el mayor valor que sumado a 1 no produce ningún efecto. Las constantes `realmax` y `realmin` contienen el mayor y el menor número en representación de coma flotante. Las letras `i`, `j`, son ambas iguales y representan la imaginaria $\sqrt{-1}$.
Una variable `valor` se define asignándole un valor, bien de forma directa, bien por medio de operaciones.

Curiosidad: las constantes `realmax` y `realmin` están definidas con cualquier combinación de mayúsculas y minúsculas.

```
>> valor=2*cos(.5)+3*sin(.2);
```

Si finalizamos la línea con `;` no se produce *output* a la pantalla.

```
>> valor=2*valor;
>> nuevovalor=5*cos(.2)-2*sin(.3);
>> resultado=valor/nuevovalor;
>> valor, nuevovalor, resultado
```

Por supuesto, las variables pueden pasar a ser argumentos de funciones.

```
>> cos(valor);
```

Matrices

Resulta que el elemento básico de de MatLab no es el número sino la matriz. Hasta ahora todas las manipulaciones las hemos hecho con números aprovechando que estos pueden interpretarse como matrices 1×1. Las operaciones de las que hemos hablado hasta ahora son las operaciones con matrices (`+,-,*,/,^`) las cuales deben tener las dimensiones adecuadas.
Veamos como definir e introducir las matrices.

```
>> [2.3,4.2,-2.1];
```

es la matriz fila (o vector fila) $(2.3, 4.2, -2.1)$ que también puede teclearse sin escribir las *comas*, simplemente con espacios.
Para introducir matrices con más filas, éstas deben separarse por medio del símbolo `;` o por un retorno de carro (*enter*).

```
>>[2.3,4.2,-2.1;3.2,-2.7,1.0];
>>[2.3 4.2 -2.1
3.2 -2.7 1.0];
```

definen exactamente la misma matriz:

$$\begin{pmatrix} 2.3 & 4.2 & -2.1 \\ 3.2 & -2.7 & 1.0 \end{pmatrix}$$

Por supuesto, introducir una matriz como lo hemos hecho no sirve para nada si no operamos con ella o si no le damos un nombre.

```
>> [1,2;3,-1]*[0,3;-1,2]
ans=
-2      7
1       7
>> x=[1,2;3,-1];
>> y=[0,3;-1,2];
>> z=x*y
ans=
    -2      7
    1       7
```

Con lo que hemos visto hasta ahora puedes producir algunos ejemplos y forzar mensajes de error pidiendo imposibles.
MatLab introduce tres nuevas operaciones para manipular números y matrices que son el producto, la división y la exponenciación componente a componente (MatLab las llama *array operations*) cuyos símbolos son `.*  ./  .^`

```
>> [1,2].*[2,2]
ans=
    2       4
```

es decir, el resultado es el vector $(1\cdot 2, 2\cdot 2)$, producto componente a componente. De igual forma

```
>> [1,2]./[2,2]
ans=
    0.5000  1.0000
```

en este punto conviene indicar que MatLab utiliza también el operador división invertido

```
>> 2\4
ans=
    2
```

en cualquiera de ellos, `/` y `\`, el numerador es el que *queda arriba* y el denominador el que *queda debajo*. El operador `\` también tiene versión componente a componente.

```
>> [1,1]./[2,2]
ans=
    0.5000  0.5000
>>[1,1].\[2,2]
ans=
    2       2
```

Para el operador `.^` también tienen sentido

```
>> [2,2].^2
ans=
    4       4
>>[2,2].^[2,3]
ans=
    4       8
```

Los operadores `+` y `-` funcionan exactamente igual que `.+` y `.-`
El símbolo `:` se utiliza para rangos numéricos, ya sean enteros o de coma flotante. Por ejemplo, si

```
>> x1=[1.1,2.2,3.3;4.4,5.5,6.6];
```

entonces

```
>> x1(2,1)
ans=
    4.4000
>> x1(:,1)
ans=
    1.1000
    4.4000
>> x1(2,2:3)
ans=
    5.5000  6.6000
>> x3=1:5
```

```
ans=
    1       2       3       4       5
>> x4=1:2:10
ans=
    1       3       5       7       9
```

Explicación: 1:2:10 será el vector cuya primera componente es 1 y las siguientes se incrementan de dos en dos unidades. El método es válido para números FT, incluso con incrementos negativos.

```
>> x5=.73:0.05:0.97
ans=
    0.7300  0.7800  0.8300  0.8800  0.9300
>> .73:-.05:.50
ans=
    0.7300  0.6800  0.6300  0.5800  0.5300
```

Tres funciones de MATLAB permiten definir matrices muy especiales que pueden ser muy útiles

- `eye` genera una matriz con *unos*en la diagonal principal y *ceros* en las demás posiciones.
- `ones` genera una matriz con *unos* en todas las posiciones.
- `zeros` genera una matriz con *ceros* en todas las posiciones.

Experimenta a ver que resultados obtienes con las siguientes instrucciones

```
>> eye(5)
>> eye(2,3)
>> eye(3,2)
>> eye(1,5)
>> ones(3)
>> ones(1,5)
>> zeros(4)
>> zeros(2,8)
```

Para transponer una matriz real `x` es `x'`. Si `x` es compleja entonces `x'` es la conjugada de la transpuesta de `x`; `conj(x)` es la matriz conjugada compleja de `x`.

Las operaciones `sum` y `prod` aplicadas a una matriz suman y (respectivamente) multiplican los elementos de cada columna y dan como resultado un vector fila cuyas componentes son los resultados obtenidos.

```
>> x=[1,3,5,3
4,7,9,2
5,8,3,9
5,8,3,1]
>> sum(x), prod(x)
ans =
    15    26    20    15
ans =
         100         1344          405           54
```

Ejercicio: Utiliza las operaciones recién estudiadas para realizar las sumas

$$\textbf{a.}\ \sum_{n=1}^{20}\frac{1}{n} \qquad \textbf{b.}\ \sum_{n=1}^{20}\frac{1}{n^2} \qquad \textbf{a.}\ \sum_{n=1}^{20}\frac{1}{2^n}$$

Soluciones: **a.** 3.5977 **b.** 1.5962 **c.** 0.99999904632568

Otra instrucción de gran utilidad en la manipulación e introducción de matrices es `diag` que actúa de forma distinta sobre matrices y sobre vectores. Sobre una

matriz que no sea un vector da como resultado el vector columna de los elementos de la diagonal principal. Sobre un vector produce una matriz cuadrada cuya diagonal principal es el vector dado.

```
>> A=[1,2,3;4,5,6]; diag(A)
ans =
     1
     5
>> x=[1,2,3,4];diag(x)
ans =
     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     4
```

La instrucción `diag` admite también un segundo parámetro, que tiene que ser un entero `k`. Sobre vectores produce la matriz cuadrada cuya superdiagonal (o subdiagonal si `k` es negativo) de orden `k` es la dada. Sobre matrices extrae la sub (super) diagonal indicada. Experimenta con algunos vectores y matrices para ver como funciona.

```
>> diag([1,2],2)
ans =
     0     0     1     0
     0     0     0     2
     0     0     0     0
     0     0     0     0
```

Si se quiere una matriz diagonal cuya diagonal coincida con la de una matriz `A` dada se conseguirá con

```
>> diag(diag(A))
```

Otras instrucciones:
`tril(A), triu(A), sort(A), max(A), min(A), diff(A)`.
Esta última instrucción será de especial importancia en el cálculo de polinomios interpoladores.

Representación gráfica

La función básica de MatLab que permite representar gráficamente en el plano los resultados obtenidos es `plot`. En su forma más sencilla tiene como argumentos dos vectores de la misma longitud, `x` e `y`, y representa en el plano los puntos con coordenadas parejas (x_n, y_n) ó `(x(n),y(n)`.

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

El *output* de esta función aparece en una ventana gráfica nueva que tiene el título *Figure No. 1* Si se utiliza `plot` dos veces consecutivas los resultados del segundo borran los del primero. Si se quiere que esto no ocurra y que los resultados se muestren sobre los anteriores ha de darse primeramente la instrucción

```
>>hold
```

Si se quiere que los resultados del segundo `plot` aparezcan en una nueva ventana habrá de efectuarse antes la instrucción

```
figure
```

con lo que aparece una segunda ventana gráfica con el título *Figure No. 2* en la que aparecerán los resultados del segundo `plot`.
Antes de seguir adelante conviene ver que nos ofrece

```
>>help plot
```
parte de los que nos muestra es la siguiente tabla que nos da parámetros para dibujar nuestros datos con diferentes colores y marcas.

```
(...)
 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

For example, PLOT(X,Y,'c+:') plots a cyan dotted line with a plus
at each data point; PLOT(X,Y,'bd') plots blue diamond at each data
point but does not draw any line. (...)
```

Bucles

MatLab admite, bien de forma directa, bien a través de un archivo ejecutable, la programación de bucles del mismo estilo de los que aparecen en los lenguajes de programación científica (fortran, C, pascal).
Los dos bucles más usuales son

```
for (...)%
(...instrucciones...)%
end
```

```
while (...condiciones...)%
(...instrucciones...)%
end
```

Ambos tipos de bucles pueden interrumpirse por medio de la instrucción `break`, que hará falta condicionar por medio de algún condicional.
Podemos introducir instrucciones condicionales por medio de `if`

```
if (...condicion...)%
(...instrucciones...)%
end
```

Un `if` puede ir ramificado con uno o varios `elseif` y puede finalizarse con un `else` que recoja todas las opciones no contenidas en los anteriores

```
if (...condici\'{o}n...)%
(...instrucciones...)%
elseif (...condicion...)%
(...instrucciones...)%
elseif (...condicion...)%
(...instrucciones...)%
else%
(...instrucciones...)%
end%
```

Cómo dar salida estructurada a los resultados

Los resultados obtenidos se imprimen directamente en la pantalla si no se evita por medio de un punto y coma. También, tras especificar el nombre de la variable, su valor aparece por pantalla.
La forma en que estos resultados aparecen en la pantalla puede modificarse por medio de la instrucción `format` que tiene las siguientes versiones, relacionadas por la instrucción

```
>>help format
```

```
FORMAT Set output format.
    All computations in MATLAB are done in double precision.
    FORMAT may be used to switch between different output
    display formats as follows:
      FORMAT         Default. Same as SHORT.
      FORMAT SHORT   Scaled fixed point format with 5 digits.
      FORMAT LONG    Scaled fixed point format with 15 digits.
      FORMAT SHORT E Floating point format with 5 digits.
      FORMAT LONG E  Floating point format with 15 digits.
      FORMAT SHORT G Best of fixed or floating point format with 5 digits.
      FORMAT LONG G  Best of fixed or floating point format with 15 digits.
      FORMAT HEX     Hexadecimal format.
      FORMAT +       The symbols +, - and blank are printed
                     for positive, negative and zero elements.
                     Imaginary parts are ignored.
      FORMAT BANK    Fixed format for dollars and cents.
      FORMAT RAT     Approximation by ratio of small integers.

    Spacing:
      FORMAT COMPACT Suppress extra line-feeds.
      FORMAT LOOSE   Puts the extra line-feeds back in.
```

Cómo recoger varias instrucciones en un archivo

En muchas ocasiones nos interesa efectuar varias operaciones de forma secuencial y el tiempo de proceso de alguna de ellas puede ser prolongado, por lo que en vez de esperar ante la pantalla a que aparezcan los resultados es mejor recoger todas las instrucciones de forma secuencial en un archivo y procesarlo directamente. Además, después de procesado y obtenidos los resultados podemos reusar el archivo, tal vez con alguna modificación, posteriormente.
Para esto podemos escribir las líneas de instrucciones en un archivo de texto que guardaremos con con un nombre al que pondremos la extensión `.m`, por ejemplo `programa.m`. Si este programa está en nuestro directorio de trabajo, o en el *path* de MatLab, la sola instrucción

```
>>programa
```

hace que las líneas de `programa.m` se procesen secuencialmente como instrucciones de MatLab.
Todo el texto que se encuentre en una línea tras el símbolo `%` MatLab lo interpretará como un comentario y no lo tomará en consideración. Además, todas las líneas que aparezcan precedidas de `%` antes de la primera instrucción o de la primera línea en blanco, MatLab las mostrará con la instrucción

```
>>help programa
```

Ejemplo. Un programa que suma los inversos de los veinte primeros números naturales. El nombre del programa será `UnaSuma.m`.

Recuerda que mayúsculas y minúsculas son diferentes para MatLab.

```
%UnaSuma
%
%Este programa suma los inversos de los 20 primeros naturales.
%
%
suma=0;          %Inicializamos la variable suma.
                 %Esto es necesario porque en su primera aparicion
                 %se utiliza para redefinirse a si misma.

for n=1:20       %
suma=suma+1/n;   %se a\~{n}ade el ";" para que no se muestre en pantalla
                 %el resultado en cada vuelta del bucle.
end
disp('La suma vale ')   %Este texto precede al valor de la suma,
                        %que se exhibe en la linea siguiente.
disp(suma)
```

Tras la instrucción

```
>> UnaSuma
```

se obtiene la respuesta

```
La suma vale
    3.5977
```

Y tras la instrucción

```
>> help UnaSuma
```

se obtiene

```
 UnaSuma

 Este programa suma los inversos de los 20 primeros naturales.
```

Observa los comentarios de las tres últimas líneas de `UnaSuma.m`, en ellas se comenta la instrucción `disp` que permite introducir texto en los resultados y escribir el valor de una variable sin que aparezca su nombre. Más adelante veremos otras formas de modificar la salida de resultados.
Como continuación del ejemplo anterior vemos otras formas de guardar nuestro trabajo con MatLab

Cómo guardar nuestro trabajo