# Lección D

# Definición de funciones

## D.1. Funciones de Matlab

Damos la relación de las **funciones escalares** más importantes de Matlab

| Función | Descripción |
|---|---|
| `sin, asin` | seno, arcoseno |
| `cos, acos` | coseno, arcocoseno |
| `tan, atan` | tangente, arcotangente |
| `sinh, asinh` | seno y arcoseno hiperbólico |
| `cosh, tanh` | coseno y tangente hiperbólico |
| `abs(r), abs(z)` | valor absoluto de $r \in \mathbb{R}$ y módulo de $z \in \mathbb{C}$ |
| `angle(z)` | argumento de $z \in \mathbb{C}$ |
| `conj(z)` | conjugado de $z \in \mathbb{C}$ |
| `real(z), imag(z)` | parte real e imaginaria de $z \in \mathbb{C}$ |
| `log, log10` | logaritmo neperiano y en base 10 |
| `sign(z)` | $z/\|z\|$ si $z \neq 0$ y 0 si $z = 0$ |
| `sqrt` | raíz cuadrada |
| `ceil(r)` | redondea hacia infinito $r \in \mathbb{R}$ |
| `fix(r)` | redondea hacia cero |
| `floor(r)` | redondea hacia menos infinito |
| `round(r)` | redondea hacia el entero más próximo |

Y ahora damos la relación de las **funciones vectoriales** más importantes de Matlab

| FUNCIÓN | DESCRIPCIÓN |
|---|---|
| `sum(x)` | suma de las componentes de $x$ |
| `prod(x)` | análogo con el producto |
| `[y,i]=max(x)` | $y$ es la máxima componente de $x$ e $i$ el lugar donde ocurre |
| `[y,i]=min(x)` | análogo al anterior con el mínimo |
| `[y,i]=sort(x)` | $y$ es la ordenación de las componentes de $x$ de menor a mayor, $i$ es el vector tal que $x(i) = y$, *i.e.* la reordenación que hay que hacer |
| `cross(v,w)` | producto vectorial de $v, w \in \mathbb{R}^3$ |

Matlab cuenta con tres tipos fundamentales de funciones: *funciones escalares*, a un escalar le asignan otro escalar pero pueden aplicarse también sobre matrices **componente a componente** siendo, entonces, el resultado otra matriz del mismo orden; *funciones vectoriales*, a un vector (columna o fila) le asigna un escalar y cuando se aplican sobre una matriz le asignan un vector fila que contiene los resultados de su aplicación **sobre cada columna** y las *funciones matriciales* las cuales están asignadas sobre matrices y su resultado puede ser un escalar, un polinomio o también una matriz.

Las funciones matriciales más importantes ya han sido establecidas en la sección B.4. Es importante tener en cuenta que las funciones de Matlab pueden tener dos o más argumentos de salida, como ejemplo sirva la función `[V,D]=eig(A)` que a una matriz cuadrada $A$ le asigna los valores propios $D$ y los vectores propios $V$, $V^{-1}AV = D$.

**Práctica a** Para $r = 10, 50, 100, 250, 600$, y 800 calculamos la suma $\displaystyle\sum_{k=1}^{r} \frac{1}{3^k}$ siguiendo el siguiente guión

```
format long
r0=1:10; r1=1:50; r2=1:100; r3=1:250;
r4=1:600; r5=1:800;
x0=1./(3.^r0);
x1=1./(3.^r1); x2=1./(3.^r2);
x3=1./(3.^r3); x4=1./(3.^r4);
x5=1./(3.^r5);
suma0=sum(x0),
suma1=sum(x1), suma2=sum(x2),
suma3=sum(x3), suma4=sum(x4),
suma5=sum(x5),
```

del resultado obtenido se justifica, que no demuestra, la igualdad $\sum_1^\infty \frac{1}{3^k} = 0.5$.

**Práctica b** Consideramos la función continua $f(x) = \text{sen}(x)e^{-0.4x}$ en el intervalo $[0, 10]$. De ella se nos pide calcular su máximo con una tolerancia de $10^{-4}$ sobre la $x$. Para ello podemos proceder como sigue: en la práctica CB hicimos la representación gráfica de $f$, la podemos volver a repetir y ver que su máximo se encuentra en el intervalo $[1, 2]$. Ahora ejecutando el listado

```
x=1:0.01:2; y =sin(x).*exp(-0.4*x);
plot(x,y)
[ymax,i]=max(y),
xmax=x(i)
```

comprobamos que el máximo de $f$ se produce para $x = 1.19$ con una tolerancia de 0.01. Repitiendo el proceso con el intervalo $[1.18; 1.20]$ llegamos a `xmax=1.1903` y con una tolerancia de acuerdo con lo pedido.

## D.2. Definición funciones propias

El usuario de Matlab puede definir sus propias funciones o subrutinas y asignarle el nombre que quiera con la misma limitación que se tiene para nombrar un fichero. Esto es así porque de hecho definir una función propia consiste sencillamente en la creación de un fichero m que ha de tener por nombre **el mismo nombre que el de la función**. Para entenderlo, hacemos la siguiente práctica

**Práctica c** Creamos la función de nombre `dcmedcd` que calcula la media y la suma de los cuadrados de los datos dados por las componentes del vector $x$. para ello procedemos como sigue:

Primero editamos el fichero m de listado:

```
function [media, cuadrado]=dcmedcd(x)

n=length(x);
media=sum(x)/n;
cuadrado=sum(x.^2);
```

En un segundo paso asignamos el nombre `work\dcmedcd.m` al fichero editado (seguimos con nuestro criterio de recordar que debemos guardar los ficheros creados en la carpeta de Métodos). Por último, en un tercer paso comprobamos que nuestra función opera correctamente, para ello vamos a la ventana de comandos y ejecutamos el siguiente listado

```
x=[1 2 3 4 5], [xmed,cd]=dcmedcd(x)
```

obteniéndose `xmed=3` y `cd=55`.

**Práctica d** Definimos la función $g(x) = -\operatorname{sen}(x)e^{-0.4x}$ con el listado

```
function y=ddexpn(x)
y=-sin(x).*exp(-0.4.*x);
```

y con el nombre `work\ddexpn.m`. Aprovechamos ahora para calcular su mínimo relativo en el intervalo $[1, 2]$ lo cual lo conseguimos con el comando `fminbnd` ejecutando en la ventana de comandos lo siguiente

```
fminbnd('ddexpn',1,2)
```

obteniéndose que el mínimo se produce para $x = 1.19028$ lo que coincide con el resultado dado en la práctica DB.

Si nuestra tolerancia para el error fuera de $10^{-8}$ entonces ejecutaremos

```
fminbnd('ddexpn',1,2,optimset('TolX',1e-8,'Display','iter'))
```

con ello, además, conseguiremos que se nos informe del número de iteraciones necesario y del carácter del mínimo.

Matlab no cuenta con la operación `fmaxbnd` análoga a la anterior. Si nosotros quisiéramos hallar el máximo de una función $h$ lo que tendríamos que hacer es calcular el mínimo de $-h$. Esto es precisamente lo que hemos hecho para calcular el máximo de la función $f(x) = \text{sen}(x)e^{-0.4x}$ definida en la práctica DB.

**Práctica e** Vemos ahora como obtener el mínimo local relativo de una función de varias variables, lo cual se hace con la orden

`fminsearch`('función',$\overrightarrow{x_0}$,opciones)

que funciona de forma análoga al comando `fminbnd`; en este caso no se considera el intervalo de trabajo como argumento de entrada, sino un vector $\overrightarrow{x_0}$ que indica el punto en torno del cual deseamos minimizar la función.

Como ejemplo vamos a minimizar la función $f(x, y) = \text{sen}(xy)$ en un entorno del origen de coordenadas

```
>fminsearch('sin(x(1)*x(2))',[0,0])
```

ans=

1.0268 -1.5298

**Práctica f** El comando de la práctica anterior también se puede aplicar a la resolución de un sistema de ecuaciones. En efecto, consideremos para $x, y \in [-3, 3]$ las funciones $f(x, y) = x^2 + y^2 - 5x$, $g(x, y) = 2x^4 + y^4 - 9.1y$ y el sistema

$$\begin{cases} f(x,y) &= 0 \\ g(x,y) &= 0 \end{cases}$$

Nos planteamos en esta práctica el problema de resolver dicho sistema para los valores $x, y \in [-3, 3]$. Para ello procedemos como sigue:

1. Realizamos la representación gráfica de las curvas $f(x, y) = 0$ y $g(x, y) = 0$ con el listado `dfmingraf`:

```
x=-3:0.05:3; y=x;
[X,Y]=meshgrid(x,y);
f=X.^2+Y.^2-5*X;
g=2*X.^4+Y.^4-9.1*Y;
h=contour(X,Y,f,[0 0],'g');   % La curva f(x,y)=0 en verde
hold on
hh=contour(X,Y,g,[0 0],'b');  % La curva g(x,y)=0 en azul
```

```
hold off

[px,py]=ginput(2)              %pinchamos en los dos cortes
```

Con ello observamos que sólo existen dos soluciones al sistema, a saber $p_0 = (0, 0)$ y $p_1$ un punto próximo a $(1, 2)$.

2. Definimos la función `dfminFun(x)` mediante el listado

```
function F=dfminFun(x)

X=x(1); Y=x(2);

F=[
X.^2+Y.^2-5*X
2*X.^4+Y.^4-9.1*Y
];
```

3. Calculamos ahora $p_1$ con una tolerancia de $10^{-6}$ para ello ejecutamos el comando

```
p1=fsolve('dfFun',[1 2],optimset('TolX',1e-6))
```

Obteniéndose $p_1 = (1.00790330; 2.00590314)$.

**Práctica g** Realizamos ahora la representación gráfica de la función `ddexpn` en el intervalo $[0, 10]$ con el comando `fplot` para ello ejecutamos

```
fplot('ddexpn',[0,10],'.'), grid
```

De la gráfica es fácil observar que la orden `fplot` utiliza más puntos en la representación allí donde la variación de la función es mayor.

A partir de la representación gráfica de la función `ddexpn` es fácil observar que posee 4 ceros cada uno de ellos en los intervalos $[0, 1]$, $[3, 4]$, $[6, 7]$ y $[9, 10]$ . El cálculo explícito de dichos ceros lo podemos conseguir con el comando `fzero` ejecutando

```
x1=fzero('ddexpn',0), x2=fzero('ddexpn',3)
x3=fzero('ddexpn',6), x4=fzero('ddexpn',9)
```

obteniéndose `x1=0`, `x2=3.1415`, `x3=6.2831` y `x4=9.4247`. Estos puntos están obtenidos todos ellos con una tolerancia de $10^{-4}$ si quisiéramos modificar dicha tolerancia lo más sencillo que podemos hacer es poner en la ventana de comandos

```
help fzero
```

con ello conseguimos que nos salga toda la referencia acerca del comando `fzero` y en concreto se nos dice la forma de modificar su tolerancia. Teniendo en cuenta lo que se nos dice vamos ahora a calcular `x3` con una tolerancia de $10^{-6}$

```
format long
x3=fzero(@ddexpn,6,optimset('TolX',1e-6))
```

obteniéndose `x3=6.2831853..`

## D.3. Control de flujo programando en Matlab

Tal y como hemos podido observar en las últimas prácticas los ficheros m y las funciones nos permiten programar de una forma rápida y sencilla. Vamos a ver ahora la forma en que Matlab controla el flujo de programación estudiando las construcciones `for`, `while`, e `if`:

**for:** Las instrucciones

```
x=[]; for ii=1:33, x=[x,ii^2]; end, x
```

dan como resultado final el vector $x$ cuyas componentes son las 33 primeros cuadrados naturales.

El listado siguiente tiene el mismo efecto que el anterior, siendo ambos equivalentes

```
x=[];                %[] es la matriz vacía
   for ii=1:33
      x=[x,ii^2];
   end
   x
```

**while:** La forma general de un bucle `while` es

```
while  relación
    instrucciones
end
```

Como ejemplo de aplicación realizamos el siguiente listado que nos permite calcular el menor entero no negativo $n$ tal que $2^n > 8973445$:

```
n=0;
   while 2^n <= 8973445
      n=n+1;
   end
   n
```

**if:** La forma general de un bucle `if` simple es:

```
if   relación
    instrucciones
end
```

Las instrucciones se ejecutarán sólo si la relación es cierta. También son posibles las ramificaciones multiples, como se ilustra con el siguiente listado que guardamos con el nombre `work\d3IFparid.m`

```
n=28

  while n ~=0
  n=input('Introduzca un número entero (ó 0 para terminar) ')

     if n < 0
      paridad=0,
     elseif rem(n,2) == 0
      paridad=2,
     else
      paridad=1,
     end
  end
```

Hagamos notar que se usa '=' en las asignaciones mientras que para las relaciones lógicas se usa '=='. La tabla de los *operadores relacionales y lógicos* que usa Matlab es la siguiente

| Relación | Descripción | Operador | Descripción |
|---|---|---|---|
| < | menor que | & | y |
| > | mayor que | \| | ó |
| <= | menor o igual que | ~ | no |
| >= | mayor o igual que | | |
| == | igual que | | |
| ~= | no igual que | | |

Damos también la relación de las *funciones relacionales y lógicas* que usa Matlab

| Operador | Descripción |
|---|---|
| `isnan` | detecta `NaN` |
| `isinf` | detecta infinitos |
| `isempty` | detecta si una matriz es vacía |
| `any` | cierto si alguna componente lo verifica |
| `all` | cierto si todas las componentes lo verifican |
| `find` | calcula las componentes que lo verifican |

Es importante tener en cuenta que las funciones `any` y `all` son funciones vectoriales lo cual significa que a la hora de aplicarse sobre una matriz opera columna a columna.

**Práctica h** Ejecutamos las siguientes instrucciones

```
R1=(3>4 | 4>1)
R2=(3>2 & 4>5)
x=[1 2 3]; y=[2 2 4]; R3=any(x==y)
R4=find(x==y)
R5=isnan(2^inf)
R6=isinf(2^inf)
```

obteniendo los resultados `R1=1` (cierta), `R2=0` (falsa), `R3=1` (cierta), `R4=2` (para la segunda componente es cierto), `R5=0` (falsa) y `R6=1` (cierta).

**Práctica i** La ejecución de un bucle `for` y `while` se puede romper intercalando la estructura

`if`, relación, instrucción, `break`, `end`

Como ejemplo sirva el siguiente listado que calcula el volumen de una esfera de radio $r$ menor que 10000

```
r=0;
while r <= 10000
   r=input('Teclee el radio (o -1 para terminar): ')
   if r<0, error('El radio es negativo'), break, end
   vol=(4/3)*pi*r^3;
   fprintf('Volumen= %7.3f\n', vol)
end
```

La orden `fprintf` sirve para imprimir `vol` con un formato, `%7.3f` lo que significa que usará la notación del punto fijo con 3 cifras decimales y que nos reservará una longitud mínima de 7 dígitos para escribir el campo `vol`, la opción `\n` indica que el programa usará una nueva línea cada vez que dé una respuesta.

Crear un fichero m de nombre `work\diradio.m` con el listado anterior y comprobar su funcionamiento con distintos radios positivos y negativos.

**Práctica j** Llamamos $s$ al vector columna $(-1, 0, 1, 3, 5)^t$. Se define la matriz de Vandermonde asociada al vector $s$ como la matriz cuadrada de orden la longitud de $s$ cuya primera columna son las potencias quintas de las componentes de $s$, la segunda columna las potencias cuartas y así sucesivamente hasta llegar a la potencia 0-ésima de $s$. Comprobar que el listado

```
s=[-1 0 1 3 5].' n=length(s) A(:,n)=ones(n,1);
   for jj=n-1:-1:1
      A(:,jj)=s.*A(:,jj+1);
   end
A
```

nos calcula la matriz de Vandermonde de $s$.

## D.4. Ejercicios

**Práctica p** 1) Para $r = 10, 50, 100, 250, 600$, y 800 calcular la sumas

$$\sum_{k=1}^{r} \frac{(-1)^k}{3^k}; \quad \sum_{k=1}^{r} \frac{1}{k}; \quad \sum_{k=1}^{r} (\frac{1}{k} - \frac{1}{k+1})$$

2) ¿Es posible encontrar números reales $r_1$, y $r_2$, y $r_3$ con una tolerancia de $10^{-6}$ de modo que

$$\sum_{k=1}^{\infty}\frac{(-1)^k}{3^k}=r_1,\quad \sum_{k=1}^{\infty}\frac{1}{k}=r_2\quad \text{y}\quad \sum_{k=1}^{\infty}(\frac{1}{k}-\frac{1}{k+1})=r_3\ ?$$

**Práctica q** Se define la matriz de Hilbert $H_n=(h_{ij})$ de orden $n$ como la matriz caracterizada por ser $h_{ij}=1/(i+j-1)$. Elaborar guiones que nos definan la matriz de Hilbert $H_n$ para los casos $n=4,5,6$.

**Práctica r** Definir una función propia, `dr2grado(a,b,c)`, que nos calcule las dos soluciones $x_1,x_2$ de la ecuación $ax^2+bx+c=0$ con sólo ejecutar el comando `dr2grado(a,b,c)`.

**Práctica s** Definir una función propia, `dsseno(a,b,c)`, que nos calcule la media ponderada de la función seno en tres puntos $a,b,c$ por la fórmula:

$$\texttt{dsseno(a,b,c)}=\frac{\operatorname{sen}a+2\operatorname{sen}b+\operatorname{sen}c}{4}$$

Comprobar el correcto funcionamiento de la función definida para las ternas $(0,\pi,2\pi)$, $(1,2,3)$ y $(0,\pi/2,\pi)$.

**Práctica t** 1) Crear una función propia, `dtex(x)`, que calcule la siguiente función:

$$f(x)=0.5e^{x/3}-x^2\operatorname{sen}x$$

La función debe aceptar tanto un escalar como un vector. Probar la función definida ejecutando `dtex(3)` y `dtex([1 2 3])`

2) Hacer lo mismo para la función:

$$g(x)=\operatorname{sen}x\log(1+x)-x^2,\quad x>0$$

**Práctica u** Consideramos la función $y(x)$ que para $x,y\in[-1,5]$ está definida por la ecuación implícita

$$y^3+e^x+\operatorname{sen}x-2=0$$

Se pide, calcular los ceros de $y(x)$, $x\in[-1,5]$.

(`duimplic`)

**Práctica puntuable v** (0.3 puntos) El señor Hernández tiene tres hijos cuyos nombres y datos son los siguientes

| Nombre | Fecha de Nacimiento |
|---|---|
| Fernando | 17/2/1990 |
| Pablo | 4/4/1991 |
| Javier | 20/4/1998 |

Este señor ha ido tomando las alturas de sus tres hijos y con los datos ha confeccionado el fichero `dvaltur_x.dat` donde aparecen: en las tres primeras columnas las fechas en que midió a sus hijos, siendo la primera columna el año, la segunda el mes y la tercera el día; en la cuarta quinta y sexta columna introdujo las alturas (en centímetros) que en las citadas fechas tenían sus hijos Fernando, Pablo y Javier, respectivamente.

Se pide, representar en un mismo dibujo y con diferentes colores las tres gráficas de las alturas de los tres hijos en relación a sus edades.

**Práctica puntuable w** (0.3 puntos) Resolver los siguientes sistemas de ecuaciones con una tolerancia de $10^{-5}$ y para valores de las variables comprendidos entre $-10$ y 10

$$1)\ \begin{cases} x_1^2 + x_2^2 &= 1 \\ x_1^2 - x_2 &= 0 \end{cases} \qquad 2)\ \begin{cases} -3x_1(x_1+1)(x_1-1) &= x_2^2 \\ x_2^2 - 5x_1^2x_2 - x_1^3 &= 1 \end{cases}$$

$$3)\ \begin{cases} (x_2-1)^2 + 4x_1(x_1^2-4) &= 0 \\ (x_1 + x_2^2/5)\cdot \ln|x_2/2+8| &= 3 \end{cases} \qquad 4)\ \begin{cases} \operatorname{sen}^2 x_1 + \frac{1}{2}\operatorname{sen} x_2 &= 0 \\ x_1^2 + x_2 &= 0 \\ \frac{1}{3}x_1 + x_2 &= 0 \end{cases}$$

**Práctica puntuable x** (0.3 puntos) Considérese el sistema de ecuaciones no lineales siguiente:

$$\begin{cases} x_1 &= 1 - x_2 \\ x_i^2 &= \frac{\sqrt{i-1}}{(2i-1)}(1 - x_{i+1} - x_{i-1}) \qquad i = 2, \ldots, n \\ x_{n+1} &= -x_n \end{cases}$$

Resolverlo para $n = 5$ en las proximidades del origen, y para $n = 10$ en las proximidades de $b_0 = (1, \ldots, \sqrt{i-1}/(2i-1), \ldots, 0)$, $i = 2, \ldots, n$.

(`dxnolnV_x` y `dxnolnX_x`)

**Práctica puntuable y** (0.4 puntos) Determinado señor desea comprarse un todo-terreno que tiene una altura de 177 cm y una distancia entre ejes de 270 cm. Antes de comprarlo quiere saber si dicho automóvil va a poder entrar en el garaje situado en el sótano de su vivienda. Sabiendo que la puerta de su garaje es de 200 cm de altura y que la pendiente de la rampa de acceso a su garaje es del 15 % se nos pide:

1. Determinar si dicho automóvil entrará en el garaje.
2. Determinar si es posible además colocarle una baca que sobresale 14 cm por encima del techo del citado todo-terreno.

(`dyterra`)