1. Introduction

To get information about the functions related to MATLAB graphics, type:

```matlab
>> help graph2d  %Two-dimension (2D) plots
>> help graph3d  %Three-dimension (3D) plots
>> help specgraph %specific plots
>> help graphics  %low level commands
```
MATLAB Handle Graphics allows:

- **High-level commands for data presentation.** This includes two-dimensional and three-dimensional plots, photograph/image processing, specific charts for presentations (such as bar diagrams or pie diagrams), and special effects (movies, lighting, camera movements).

- **Low-level commands for creating and modifying objects.** This constitutes the so-called graphics user interface (GUI) for developing complex applications, consisting of figures, menu bars and control elements (such as buttons). With these utilities you can design windows like the ones shown in Fig. 1. These two windows correspond to the Curve Fitting Toolbox. The command is `>>cftool`.

![Fig. 1. Example of a graphics user interface](image)

In the first case you work as a user whereas in the second case you work as a software developer. The second case refers to a more advanced level of MATLAB, which will be discussed in Unit 4. In this unit we introduce the two basic commands `set` and `get`. 
2. Graphic objects

MATLAB graphic objects present the following hierarchy:

```
  root
     figure
        ui objects  axes  annotation
             line  surface  patch  image  text  light
```

*(Fig. 2. Hierarchy of graphic objects in MATLAB.)*

(Note: There are more objects and groups of objects but they are not shown here to make the presentation clearer. For more details, see "graphics objects" in the MATLAB help file.)

The object **root** is the command window. If **root** does not exist then neither do the objects shown in Fig. 2. In other words, if MATLAB is not active, it is not possible to generate **figure**, **axes**, etc.

When we type `>> plot(t,y)` in the **root** object, all objects needed for the representation are automatically generated, **figure** → **axes** → **line**, if they did not already exist.

```
  >> x=0:10;
  >> plot(x,x)
```

*Handle:* Every graphical object is associated to a “handle” (a number that identifies the object) and to a set of “properties” (**color**, **Position**, etc.). Some properties can be modified by the user, whereas others cannot. The handle for the object **root** is 0.

To get the handle of a **figure** object, you can use the **gcf** command (get current figure).

```
  >> gcf
  ans =
       1
```

To get the handle of an **axes** object, you can use the **gca** command (get current axes).

```
  >> gca
  ans =
```
To see all of the properties of an object, you can use `get`, e.g., $\text{get(gca)}$

```matlab
>> get(gca)
ActivePositionProperty = outerposition
ALim = [0 1]
: 
ZTickMode = auto

BeingDeleted = off 
:
Visible = on
```

To change the object properties, you can use `set`, e.g.,

```matlab
>> set(gcf,'NumberTitle','off')
>> set(gcf,'Color','r')
```

These options are also available from the figure window menu bar: Edit $\rightarrow$ Figure Properties.

### 2.1 LINE object

**2D plots:** General steps for two-dimensional curve plotting are listed below. However, it is often not necessary to follow all of these steps explicitly because the functions available in commercial toolboxes automatically execute some of them.

**Step 1) x axis:** Generate a vector with the values for the abscissas axis.

*Functions:* two points (:), `linspace`, `logspace`.

For instance,

```matlab
>> x=0:.2:12; (initial value : space between values : final value)
```
or alternatively,

```matlab
>>x=linspace(0,12,200); % (200 equally spaced values, 0 being the first value and 12 being the last)
```

When logarithmic scales are needed, e.g., for a Bode diagram, we can use the function `logspace`:

```matlab
>>w=logspace(-1,3); % (50 logarithmically spaced values between 10^{-1}=0.1 and 10^3=1000)
```

Note that the third input (vector length) argument in both `linspace` and `logspace` is optional. Default values are 100 and 50, respectively. Also, in function `, if the step between values is not specified, the default value is 1 (e.g., `>>x=1:3` generates a row vector with elements 1 2 3).

**Step 2) y axis:** Generate a vector containing the values corresponding to the y-axis.

Note that vectors x and y must be of equal length. Actually, y is often computed as a function of x, which means that the dimension compatibility is guaranteed.

For instance,

```matlab
>>y1=bessel(1,x);
>>y2=bessel(2,x);
>>y3=bessel(3,x);
```

If we want to plot a constant value along the x axis, we can type `>>plot(x,2*ones(size(x))`. The function `ones` generates a matrix with the same dimension as x and with all components equal to “1”. Another possibility is `>>plot(x,x*0+2)`.

**Step 3) Plot:** Call a graphics generation command.

- **Functions:** `plot`, `semilogx`, `semilogy`, `loglog`, `polar`, `plotyy`, `stem`, `stairs`.

For instance:

```matlab
>>plot(x,y1,x,y2,x,y3)
```

When working with complex numbers (e.g., `>>n=3+j*5;`), statement `>>plot(n,'x')` is equivalent to `>>plot(real(n),imag(n),'x')`.

**Step 4) Zoom:** To change the axes values, you can use `axis` and `zoom` (zoom on and zoom off). The syntax for `axis` is as follows:

```matlab
axis([xmin xmax ymin ymax])
```
The following are also possible:

```
axis square
axis normal
```

**Step 5) Grid:** To add a grid, you can use function `grid` (also `grid on` and `grid off`). See also `box on/off` and `axis on/off`.

```
>> th=linspace(0,2*pi,101);
>> x=sin(th);
>> y=sin(2*th+pi/4);
>> plot(x,y,'k-')
>> grid on
>> box off
>> axis off
```

![Fig. 4. Grid, box off and axis off](image)

**Step 6) Line properties:** It is possible to change the line style (solid or not) and use different colours and symbols. We recommend exploring the statement `>>help plot`. Different options can be combined (the order is not important) and are called inside symbol `'` (e.g., `'r'`).

Colours: `r`, `b`, `g`, `m`, `k`, `c`, `y`, ...

Line style: `'-'`, `'-.'`, `'-..'`, `'o'`, `'*'`, `'+``, ``, `s`, `h`, `p`, `d`, ...

For instance,

```
>>plot(t,y1,t,y2,'r--',t,y2,'-.g')
```

![Fig. 5. Line styles](image)
Step 7) Holding plots: To add new graphics over existing ones, use the function `hold` (also `hold on`, `hold off`).

```matlab
>> plot(x,y1), hold
Current plot held
>> plot(x,y2)
>> plot(x,y3)
>> hold
Current plot released
```

Alternatively,

```matlab
>> plot(t,y1), hold on
>> plot(t,y2)
>> plot(t,y3), hold off
```

Step 8) Subplots: Use `subplot(a,b,c)` to subdivide the figure window into several graphic areas, where `a` is the number of rows, `b` is the number of columns and `c` refers to the current plotting area (from 1 to `a*b`).

For instance, two plots:

```matlab
>> subplot(212), plot(x,y2)
>> subplot(211), plot(n,'og')
```

Four plots:

```matlab
>> x=linspace(0,12); y1=bessel(1,x); y2=bessel(2,x); y3=bessel(3,x);
>> subplot(221), plot(x,y1)
>> subplot(222), plot(x,y2)
>> subplot(223), plot(x,y3)
>> subplot(224), plot(x,x)
```

![Fig. 6. Use of the `subplot` command](image)

Step 9) Graphics input: To capture coordinate values `x, y` from a plot, use `ginput` (`graphics input`). The simplest usage is:
>> ginput

A cross appears over the current plot. Use the mouse to capture several points and when you have enough, return to the command window to see the captured points by pressing the <Enter> key.

**Step 10) Save and open figures:** To save a figure object, select the following options in the menu bar: File → Save. The figure will be saved in a file with the extension *.fig, for instance, figu.fig. To recover the figure, simply type:

```matlab
>> openfig('figu')
```

**3D plots:** Three-dimensional line plots follow the same steps as two-dimensional line plots. The only difference is in the graphic commands. We use commands such as `plot3` or `comet3`.

```matlab
>> plot3(y1,y2,y3)
```

![3D line object](image)

**Fig. 7. 3D line object**

Axes can be modified using the `axis` command:

```matlab
axis([xmin xmax ymin ymax zmin zmax])
```

### 2.2 TEXT object

Again, consider the graphical representation of the three Bessel functions. To label the representation, we can use the following functions: `xlabel`, `ylabel`, `zlabel`, `title`, `text`, `gtext`, `legend`. Observe how we establish the format for sub- and super-indexes.

```matlab
>> xlabel('x')
>> ylabel('y_1 , y_2 , y_3')
>> title('Funciones de Bessel de primera especie')
>> legend('1^e^r orden','2^o orden','3^e^r orden','-1')
>> text(6,0.5,'hoolaaa')
```
Function `gtext` (graphics text) is similar to `text` but it inserts the text using the mouse instead of indicating the coordinates for the text. To write several text lines, use the symbol \{\}:

```matlab
>>gtext({'y_1: 1^{s^{t}} order','y_2: 2^{n^{d}} order','y_3: 3^{r^{d}} order'});
```

It is possible to use symbols from the Greek alphabet. Just type symbol \ before the symbol name: \alpha, \beta, …

```matlab
>>title('y_1(\phi)=(\phi-sin(\phi))/2');
```

We suggest typing `>>help TeX`. This utility also allows for mathematical expressions such as rational functions, square roots, and so on.

Finally, you can insert other symbols, such as arrows:

```matlab
>> text(0.3,0.4,\text{\downarrow},'FontSize',10)
```

Another useful function for use within a text object is `num2str` (number to string):

```matlab
>> r=2.5;
>> text(0.4,0.3,['radio = ',num2str(r)])
```

## 2.3 PATCH object

Patch objects are objects composed of one or more polygons that may or may not be connected. They are useful for presentations and animations since they can be used to draw complex pictures.

The three basic functions are `fill`, `fill3` and `patch`. The user must specify the polygon vertices and the fill-in colours.

The order in the vertices specification is very important. See the following example:
>> x=[3 3 7 7];
>> y=[5 6.5 6.5 5];
>> fill(x,y,'r')
>> axis([0 10 4 7.5])

Fig. 9. Specification of vertices in a PATCH object

An alternative is \texttt{fill(x1,y1,c1,x2,y2,c2,...)}.

It is possible to use pre-specified colours or to define our own colours using the triple \([r\ g\ b]\), where the three components (red, green, blue) take values between 0 and 1.

Example:

Dark red = \([0.5\ 0\ 0]\)
Copper = \([1.62\ .4]\)
Grey = \([0.5\ 0.5\ 0.5]\)

Pre-specified:
Red = \([1\ 0\ 0]\) ‘r’
Blue = \([0\ 0\ 1]\) ‘b’
Magenta = \([1\ 0\ 1]\) ‘m’
Black = \([0\ 0\ 0]\) ‘k’
Green = \([0\ 1\ 0]\) ‘g’
Cyan = \([0\ 1\ 1]\) ‘c’
Yellow = \([1\ 1\ 0]\) ‘y’
White = \([1\ 1\ 1]\) ‘w’

There also exist pre-specified colour maps (\texttt{colormap}): \texttt{hsv}, \texttt{hot}, \texttt{cool}, \texttt{summer}, \texttt{gray}, \texttt{jet}.

Example 1. PATCH objects

This example shows how to fill in an area \([\text{mag},0.5]\):

\begin{verbatim}
>> x=linspace(0,6,100);
>> plot(x,cos(x),'k-',x,1./cosh(x),'k--',[4.73 4.73],[[-1 1]],'k')
>> hold on
>> xn=linspace(0,4.73,50);
>> fill([xn,flip(xn)], [1./cosh(xn),flip(cosh(xn))]),'c')
\end{verbatim}
>> x=linspace(0,6,100);
>> plot(x,cos(x),'k-',x,1./cosh(x),'k--',[4.73 4.73],[-1 1],'k')
>> hold on
>> xx=linspace(0,4.73,20);
>> plot([xx;xx],[cos(xx);1./cosh(xx)],'k-')

2.4 SURFACE object

The procedure for plotting surfaces in three-dimensional axes is as follows,

First, define the values for axes x, y

```matlab
>> x=-10:0.1:10;
>> y=x;
```

Then combine the values of x and y (vectors) to obtain the grid xx, yy (matrix) where we are going to represent the z values.

```matlab
>> [xx,yy]=meshgrid(x,y);
```

z values are computed over the grid values xx, yy:

```matlab
>> z=xx.^3+yy^2+2*xx*yy;
```

Finally, plot the surface (use `mesh`, `surf`, `surface`, `waterfall`). If you want to change the viewpoint, use the `view` function.

```matlab
>> mesh(x,y,z)
```

Contours are represented with `contour` or `meshc` (which combines the surface and the contour plot). The contour diagrams are labelled with `clabel`.

To change the representation colour properties, use `colormap` (e.g., `>>colormap gray`), `shading`, `hidden`, `brighten`. You can also use the menu bar (`Edit → Colormap...`)
Example 2. SURFACE objects

The following example can be found in the MATLAB demos:

```matlab
z = peaks;
surf(z); hold on
shading interp;
[c ch] = contour3(z,20); set(ch, 'edgecolor', 'b')
[u v] = gradient(z);
h = streamslice(-u,-v); % downhill
set(h, 'color', 'k')
for i=1:length(h);
    zi = interp2(z,get(h(i), 'xdata'), get(h(i),'ydata'));
    set(h(i),'zdata', zi);
end
view(30,50); axis tight
```

To change the axes, use `axis`, `zoom`, `grid`, `box`, `hold`, `axes`, `subplot`. You can rotate the plot using `rotate3d`, `viewmtx` or `view` (in the menu bar: View → Camera...).
Toolbar). We recommend exploring the options from the menu bar in the figure window.

\[
\begin{align*}
\text{>> } & \text{z=peaks; surf(z)} \\
\end{align*}
\]

![Fig. 12. Toolbar in the figure window.](image)

For more information, type \texttt{>>help graph2d y >>help graph3d}.

\textit{Prespecified volumes}: Use the functions \texttt{cylinder, sphere, ellipsoid}.

\[
\begin{align*}
\text{>> } & \text{cylinder([2 1 1 0.5],20);} \\
\text{>> } & \text{sphere(50), axis('square')} \\
\end{align*}
\]

![Fig. 13. Prespecified volumes.](image)

### 2.5 LIGHT object

This object is used to change the appearance of 3D representations. The most important functions are \texttt{lighting (flat, none, phong, gouraud), material (metal, dull, shiny), surf1, specular, diffuse, surfnorm}.

\textbf{Example}:

\[
\begin{align*}
\text{>> } & \text{z=peaks; surf(z)} \\
\text{>> } & \text{colormap('gray')} \\
\end{align*}
\]
>> lighting phong

Select *Insert \(\rightarrow\) Light* on the menu bar.

![Fig. 14. LIGHT object.](image)

### 2.6 IMAGE object

MATLAB writes/reads several graphical formats (TIFF, JPEG, BMP, PCX, XWD, HDF). The main functions are `imread`, `imwrite` and `imfinfo`.

```matlab
>> X=imread('earth1','gif');
>> X=imread('earth1.gif');
>> imfinfo('earth1.gif')
```

```
ans =
    Filename: 'earth1.gif'
    FileModDate: '17-May-2000 01:49:46'
    FileSize: 58178
    Format: 'GIF'
    FormatVersion: '87a'
    Width: 601
    Height: 353
    BitDepth: 7
    ColorType: 'indexed'
    FormatSignature: 'GIF87a'
    BackgroundColor: 0
    AspectRatio: 0
    ColorTable: [128x3 double]
    Interlaced: 'no'
```

There are two data types available to display the image: `double` (floating double precision, 64 bits) and `uint8` (unsigned integer, 8 bit). Functions are `image` and `imagesc`. It is possible to add a bar showing the present colours, `colorbar`. 
An image consists of one data matrix $X$ (each component is a pixel) and one matrix containing the colours for every pixel in $X$. There are four image types: indexed, of intensity, binary and truecolor.

**Index image:** A colormap matrix $\text{map}$ is defined with 3 columns corresponding to R, G, and B, and as many rows as colours are present in the image. The elements of the image matrix $X$ refer to a row number for the colour matrix.

```
>> load earth
>> image(X),colormap(map),colorbar('vert')
```

![Fig. 15. Index image](image)

**Intensity image:** Matrix $I$ contains intensities (grey levels). These levels are from black to white (from 0 to 1, from 0 to 255 or from 0 to 65535)

```
>> Y=X/64;
>> imagesc(Y,[0 1]),colormap(gray),colorbar('vert')
```

![Fig. 16. Intensity image](image)
**Binary image:** Components of matrix \( X \) are 1s and 0s.

**Truecolor image:** A 3D image that does not use a colormap. \( X \) has dimensions \( m \times n \times 3 \). Each pixel in matrix \( X \), \( X(m,n) \) is defined by three numbers: \( \text{RGB}(m,n,1) \), which corresponds to the red level; \( \text{RGB}(m,n,2) \), which corresponds to the green level; and \( \text{RGB}(m,n,3) \), which corresponds to the blue level.

```matlab
>> rgb=imread('ngc6543a.jpg');
>> size(rgb)
ans =
   650   600     3
>> image(rgb)
```

**Fig. 17. Truecolor image**

The default colormap is \texttt{colormap(‘default’)\texttt{}} which corresponds to the \texttt{hsv (Hue Saturation Value)}. Statement \texttt{>>help graph3d} gives information about other colour maps.

There are specific toolboxes that make a more intensive use of the images. For examples, see the demos of the Image Processing Toolbox, Mapping Toolbox Virtual Reality Toolbox.

### 3. Plots for specific applications

Depending on the applications, special graphics are required. For instance, in statistical analysis, data are presented by means of histograms, scattering diagrams, error bars, etc. Specific toolboxes include special representations facilities.

#### 3.1 Plots for presentations

**Pie diagrams:** The command is \texttt{pie}. If the sum of values is less than 1, the pie is not complete.

```matlab
>> x = [.19 .22 .41];
>> pie(x)
>> pie3(x)
```
If you want to take a piece:

```matlab
>> x = [1 3 0.5 2.5 2];
>> pct = x / sum(x)
pct =
         0.1111    0.3333    0.0556    0.2778    0.2222
>> piece = [0 1 0 0 0];
>> pie(x, piece)
>> pie3(x, piece), colormap summer
```

Fig. 18. Pie plots

**Histograms:** Functions are `hist, histfit`

```matlab
yn = randn(10000,1);
hist(yn),
colormap autumn

Y = randn(10000,3);
hist(Y),
colormap summer

r = normrnd(10,1,100,1);
histfit(r)
```

Fig. 19. Histograms
**Bar plots:** Functions are `bar`, `barh`

\[
Y = \text{round}([\text{rand}(5, 3) \times 10]) ; \\
\text{subplot}(2, 2, 1), \text{bar}(Y, 'group'), \text{title}('\text{Grupo}') \\
\text{subplot}(2, 2, 2), \text{bar}(Y, 'stack'), \text{title}('\text{Pila}') \\
\text{subplot}(2, 2, 3), \text{barh}(Y, 'stack'), \text{title}('\text{Pila horizontal}') \\
\text{subplot}(2, 2, 4), \text{bar}(Y, 1.5), \text{title}('\text{Grosor 1.5}')
\]

![Bar plots](image1.png)

**Fig. 20.** Bar plots

### 3.2 Probability and statistics

**Example 3. Histograms**

The histogram in the first figure has been generated as:

\[
\text{>>data=randn(1000,1)}; \quad \% \text{data is a vector of 1000 random elements} \\
\text{>>hist(data,30)} \quad \% \text{30 bin histogram}
\]

Note that the data probability distribution does indeed correspond to a zero mean and unit variance Gaussian distribution (“n” in `randn` comes from “normal”). Which probability distribution corresponds to `rand`?
The second figure illustrates the energy (MW) spent by a town for 10 days. The commands used are:

```matlab
> days=19:28;
> power=[10.2, 11.5, 12.7, 18, 16.3, 14.7, 13.0, 13.9, 12.3, 13.1];
> bar(days,power);
```

---

**Example 4. Error intervals**

Consider a system whose output $y$ is the decreasing exponential of the squared input $u$. Output measurements for different input values $u$ are uncertain (uncertainty here has been generated in a random manner). Such uncertainty can be represented using `errorbar`:

```matlab
> u=-2.9:0.1:2.9;
> e=0.1*rand(size(u));
> y=exp(-u.*u);
> errorbar(u,y,e)
```

---

**Example 5. Box and scatter diagrams**

*Scatter diagrams:*

```matlab
load carsmall

figure,scatter(Weight,MPG),xlabel('Weight'),ylabel('MPG')
figure,gscatter(Weight,MPG,Model_Year,'bgr','xos')
```

---

*Fig. 21. Scatter plots*
File *carsmall.mat* contains data about 100 cars: *Acceleration, Cylinders, Displacement, Horsepower, MPG* (consumption: miles-per-gallon), *Model, Model_Year, Origin, Weight*.

**Box plots:**

```matlab
boxplot(MPG, Origin)
```

![Box plots](image)

Fig. 22. Box plots

There is one outlier: a car with the characteristic MPG>40. You can use `find` to identify it:

```matlab
>> find(MPG>40)
ans =
 97
```

It is a German car:

```matlab
>> Origin(97,:)
ans =
    Germany
```

We can also identify the model and year:

```matlab
>> Model(97,:)
ans =
    vw pickup
```

```matlab
>> Model_Year(97,:)
ans =
    82
```

---

**Example 6. Probability plots**

*Distribution diagrams:*
Normal probability plot: Used to determine if a given sample is Gaussian distributed. The solid line connects 25 and 75 percentiles.

```matlab
x=normrnd(10,3,100,1);
figure,normplot(x)
x=exprnd(10,100,1);
figure,normplot(x)
```

Fig. 23. Normal probability plots

Clearly, the second figure does not correspond to a normal distribution. Another way to see that it is not normal is by means of the Kolmogorov-Smirnov test:

```matlab
>> h=kstest(x)
h =
    1
```

The interpretation is as follows: if the result is $h = 1$, we can reject the null hypothesis. The null hypothesis is that the sample $x$ is standard normal distribution (mean 0 and variance 1). The test result says that you can reject this hypothesis. Therefore, the sample is not distributed as $N(0,1)$.

We may also ask whether the distribution is normal but with a different mean and deviation, $N(m,\sigma)$. We already know that it is not the case for the PP plot, but we will make sure:

```matlab
>> [m,s]=normfit(x); %buscar media y desv que ajusten la muestra
>> [h,p]=kstest(x,[x normcdf(x,m,s)]) %y aplicar el test
h =
    1
p =
    0.0032
```

Since $h = 1$ again, we can reject the null hypothesis that the sample is distributed as $N(m,\sigma)$. We are left with the alternative hypothesis, which says that the sample is not distributed as $N(m,\sigma)$.

The `kstest` function rejects the null hypothesis ($h = 1$) by default if the significance level is 5%, that is, if the value of $p$ is less than 0.05, as is true in our case (if the $p$ value had been higher than 0.05, $h$ would have given 0).
For other distributions, use `probplot`.

```matlab
x=wblrnd(3,3,100,1);
probplot('weibull',x)
```

![Probability plot for Weibull distribution](image)

**Fig. 24.** Probability plot (other distributions)

**Q-Q (quantile-quantile) plot:** shows whether two samples are from the same probability distribution family.

```matlab
x=poissrnd(10,50,1);
y=poissrnd(5,100,1);
figure,qqplot(x,y)
```

```matlab
x=normrnd(5,1,100,1);
y=wblrnd(2,0.5,100,1);
figure,qqplot(x,y)
```

![Q-Q plots](image)

**Fig. 25.** Q-Q plots

**Cumulative distribution diagram:** Function is `cdfplot`.

```matlab
y=evrnd(0,3,100,1);
figure,cdfplot(y)
```
Example 7. Probability density function generation

Generating pdfs: Functions that generate probability density functions (pdfs) end with “…pdf” and start with the probability family name. Hence normpdf generates the normal distribution pdf and chi2pdf generates the pdf corresponding to the chi-square distribution.

\[ x = \text{linspace}(-10,10); y = \text{normpdf}(x,2,3); \text{figure}, \text{plot}(x,y) \]
\[ x = \text{linspace}(0,15); y = \text{chi2pdf}(x,4); \text{figure}, \text{plot}(x,y) \]

Other functions are betapdf (Beta), binopdf (binomial), exppdf (exponential), unifpdf (uniform), etc

Multivariable normal distribution: The function is mvnpdf.

\[ \text{media} = [0 \ 0]; \]
\[ \text{matriz\_cov} = [.25 \ .3; .3 \ 1]; \]
\[ x = -3:.2:3; \]
\[ y = x; \]
\[ [xx, yy] = \text{meshgrid}(x, y); \]
\[ F = \text{mvnpdf}([xx(:) \ yy(:)], \text{media}, \text{matriz\_cov}); \]
\[ F = \text{reshape}(F, \text{length}(xx), \text{length}(yy)); \]

\[ \text{figure}, \text{surf}(x, y, F), \text{xlabel}('x'), \text{ylabel}('y') \]
\[ \text{figure}, \text{contour}(x, y, F), \text{xlabel}('x'), \text{ylabel}('y') \]
3.3 Frequency response of linear systems

Transfer function: Enter the numerator polynomial and the denominator polynomial (notice the brackets). For instance, \( H(s) = \frac{2}{s^2 + 0.5s + 1} \), is introduced as

\[
>> \text{num}=2;
>> \text{den}=[1 \ 0.5 \ 1];
\]

Functions: They are \texttt{bode} (for Bode diagrams), \texttt{nyquist} (for polar plots), \texttt{nichols} (for phase-gain diagrams) and \texttt{freqs} (to obtain the complex value for the frequency response). We recommend exploring the help of the functions presented: \texttt{>>help function_name}.

Syntax: There are several levels.

The simplest one (see Fig. a) is:

\[
>> \text{bode(num,den)}
\]

To specify the frequencies axis (see Fig. b), use \texttt{logspace}:

\[
>> \text{w=logspace(-1,5)}; \ %\text{frequencies from 0.1 to 1e5}
>> \text{bode(num,den,w)}
\]
To store the values for the phase and magnitude (to represent them later,) use output arguments (see Fig. c):

\[
\begin{align*}
&\texttt{>> [mag,fase]=bode(num,den,w);} \\
&\texttt{>> subplot(211),semilogx(w,20*log10(mag),'r')} \\
&\texttt{>> subplot(212),semilogx(w,fase,'g')} \\
\end{align*}
\]

(a)  
(b)  
(c)  

**Fig. 26.** Frequency response

### 3.4 Time response of linear systems

System \( H(s) = \frac{2}{s^2 + 0.5s + 1} \) is entered as shown in the previous section.

**Functions:** The main functions are `impulse` (for impulse responses), `step` (for step responses) and `lsim` (linear simulation, for arbitrary excitations such as ramps, sinusoids, mixed signals, etc.). For more information, type `>>help function_name`.

**Syntax:** There are several levels (notice the semicolon use).

The simplest syntax (see Fig. a) is:

\[
\begin{align*}
&\texttt{>> step(num,den)} \\
\end{align*}
\]

To specify time span (see Fig. b), type:

\[
\begin{align*}
&\texttt{>> t=linspace(0,35);} \\
&\texttt{>> step(num,den,t)} \\
\end{align*}
\]

To generate a variable with the time response samples to plot it later (see Fig. c), type:

\[
\begin{align*}
&\texttt{>> y=step(num,den,t);} \\
&\texttt{>> plot(t,y,'r--')} \\
&\texttt{>> grid,title('Respuesta indicial'),xlabel('Tiempo [s]')} \\
\end{align*}
\]
To simulate general excitations, first define them:

\[
\begin{align*}
&> t = \text{linspace}(0, 60, 100); \\
&> u = \sin(0.2 * t); \\
&> y = \text{lsim}(\text{num}, \text{den}, u, t); \\
&> \text{plot}(t, u, t, y) \\
&> \text{legend}(\text{u}', \text{y}', \\
&> \text{xlabel}('\text{Tiempo [s]}'))
\end{align*}
\]

3.5 Another function for systems theory

To compute and plot the Evans root locus, you can use the command \textit{rlocus}:

\[
\begin{align*}
&> \text{num} = 1; \\
&> \text{den} = \text{conv}([1 3 0], [1 2]); \\
&> \text{rlocus}(	ext{num}, \text{den})
\end{align*}
\]

4. Movies

\textit{Movies}: The main functions are \texttt{moviein} (beginning), \texttt{getframe} (to capture each frame) and \texttt{movie}. (Note: \texttt{moviein} is not necessary in the newer versions.)

\textbf{Example 8. Brownian noise:}

\[
n = 300; s = 0.02;
\]
n_tr=50;
x=rand(n,1)-0.5;
y=rand(n,1)-0.5;
h=plot(x,y,'.')
set(h,'MarkerSize',18)
axis([-1 1 -1 1])
axis square
grid off

M=moviein(n_tr);

for k=1:n_tr
    x=x+s*randn(n,1);
y=y+s*randn(n,1);
set(h,'XData',x,'YData',y)
M(:,k)=getframe;
end
movie(M,5)

Try other movie options, e.g., movie(M,-1,37).

When you run the movie function, first it does a quick preview of the animation and then it presents the animation itself. You have several options, for example, movie(M,0) executes the quick preview but not the animation, movie(M,-1) executes the quick pass and then plays the animation forward and then backwards.

By default, the video timing is that of the capture. If you want to change the timing, you must use the third argument. In movie(M,-2,50), the timing is 50 frames per second. This means that a film with 50 frames will be shown in 1 second.
**Video files:** To generate a video file, you can use `avifile`. Frames are added to the video by using the `addframe` function. Finally, close the video with `close`.

The following example illustrates the generation of a video file (named `noise.avi`) based on the Brownian noise of the previous section.

**Example 9. Creating an avi file**

```matlab
mov = avifile('noise.avi')

n=300;
s=0.02;
x=rand(n,1)-0.5;
y=rand(n,1)-0.5;

h=plot(x,y,'.');
set(h,'MarkerSize',18)
axis([-1 1 -1 1]),axis square,grid off

n_frames=50;
for k=1:n_frames
    x=x+s*randn(n,1);
y=y+s*randn(n,1);
    set(h,'XData',x,'YData',y)
    F=getframe(gca);
    mov=addframe(mov,F);
end
mov=close(mov);
```

The generated video file can then be inserted into other programs, such as PowerPoint (Insert ➔ Movies and sounds ➔ Movie file). This is useful for presentations of projects or master’s theses.

**Animated images:** To display animated images (multi-frame files, etc.), you can use the `immovie` function. To illustrate it, see the following example from MATLAB:

```matlab
>> load mri
>> mov=immovie(D,map);
>> movie(mov)
```

To capture an image frame, you can do the following:
[x,map]=imread('nombre_fichero.extensión',1:num:fotogramas);

or

[x,map]=imread('nombre_fichero.extensión','frames','all');

You can also create an image by taking different frames separately and putting them together using the cat function:

A = cat (4, A1, A2)

Here, A1 and A2 are the two images that form the animation. The 4 is because this type of variable is an array of 4 dimensions m x n x 3 x 2, where m x n are the pixels of the image, 3 corresponds to a true-colour image (contains an array of three columns rgb, and each pixel refers to one of these 256 colours) and 2 is the number of frames.