# Práctica 2

# Gráficas.Ficheros .m

## 2.1. Gráficas en el plano.

### 2.1.1. Polígonos y curvas planas.

Si deseamos dibujar en el plano un polígono con vértices $(x_i, y_i)$ en Matlab, bastaría definir previamente los vectores $x = [x_1, x_2, \ldots, x_n]$, e $y = [y_1, y_2, \ldots, y_n]$, y utilizar el comando $plot(x, y)$.

Nótese que, para que el polígono sea cerrado, el último vértice debe coincidir con el primero.

**Ejemplo 2.1.1.–**

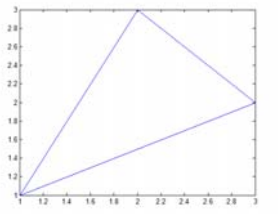
Dibujar el triángulo de vértices $(1,1), (3,2), (2,3), (1,1)$:

$x = [1, 3, 2, 1];$

$y = [1, 2, 3, 1];$

$plot(x, y)$

El resultado será:



Si se desea hacer la gráfica de una función $f(x)$, en un intervalo $[a, b]$, se debe tener en cuenta que, el comando *plot*, dibuja poligonales, por lo que se deben definir dos vectores, $x$ e $y$, de forma que en $x$ tengamos representado el intervalo $[a, b]$, y en $y$, la imagen de $f$ en dicho intervalo. Según realicemos dichas representaciones, más o menos finas serán las gráficas.

**Ejemplo 2.1.2.–** Queremos representar la función $f(x) = xsen(x)$ en el intervalo $[-2\pi, 2\pi]$.

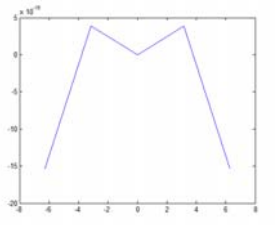
Hagamos primero una representación basta (no deseable), para ello:

$x = -2 * pi : pi : 2 * pi;$

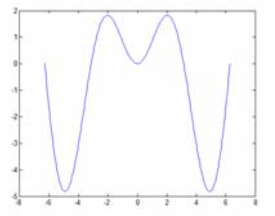
$y = x. * sin(x);$

$plot(x, y)$

El resultado será:



Hagamos ahora, una representación más fina:
$x = -2 * pi : pi/100 : 2 * pi;$
$y = x. * sin(x);$
$plot(x, y)$



En la segunda gráfica, al haber hecho la partición del intervalo mucho más fina, parece que es una curva, aunque, en realidad, siguen siendo segmentos de rectas unidos.

**Ejercicio 2.1.3.**– Representar gráficamente las siguientes funciones:

1. $f(x) = \frac{cos(x)}{x}$, $x \in [\frac{pi}{4}, 3\frac{pi}{4}]$.
2. $f(x) = e^{x^2+1}$, $x \in [-5, 5]$
3. $f(x) = ln(x^2 + 1)$, $x \in [-5, 5]$

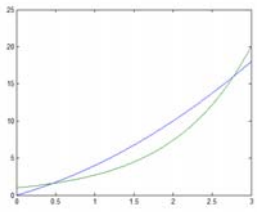
Es posible dibujar varias funciones en una misma gráfica, para ello, tenemos dos opciones:

- Usando únicamente el comando *plot*, evidentemente, todas las gráficas deben estar definidas en el mismo intervalo. Definiríamos las funciones $y1 = f(x)$, $y2 = g(x)$, ..., y escribiríamos $plot(x, y1, x, y2, \ldots)$.

  **Ejemplo 2.1.4.**– Dibujar en el intervalo $[0, 3]$ las funciones $f(x) = x^2 + 3x$ y $g(x) = e^x$.

  $x = 0 : 0{,}01 : 3;$
  $y = x.^2 + 3 * x;$
  $z = exp(x);$
  $plot(x, y, x, z)$

El resultado sería:



- Usando los comandos *hold on* y *hold off*. El comando *hold on* hace que la siguiente gráfica que deseemos dibujar, se realice sobre la última que tenemos activa, mientras que el comando *hold off*, nos devuelve a la situación previa. Si quisiésemos utilizar éste comando con el ejemplo anterior, haríamos:

  $x = 0 : 0{,}01 : 3;$
  $y = x.^2 + 3 * x;$
  $plot(x, y)$
  $hold\ on$
  $z = exp(x);$
  $plot(x, z)$

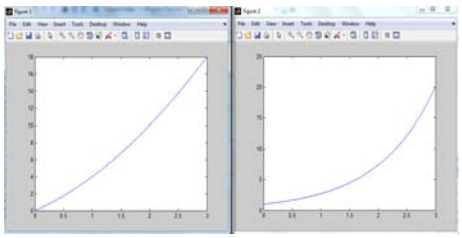
  El resultado es similar al anterior, la única diferencia es que, en esta segunda gráfica, ambas funciones se dibujan del mismo color. Con este comando, no es necesario que ambas funciones estén definidas en el mismo intervalo.

**Ejercicio 2.1.5.**– Representar en un mismo gráfico, las siguientes funciones:
$f(x) = sen(x)$, $g(x) = cos(x)$ y $h(x) = tan(x)$ en $[-2\pi, 2\pi]$

Para tener distintas representaciones gráficas visibles simultáneamente, aunque en distintos gráficos, se usa el comando $figure$, dicho comando, abre una nueva ventana de gráfico para representar la siguiente función. Por defecto, sigue el orden numérico natural.

**Ejemplo 2.1.6.**– $x = 0 : 0{,}01 : 3;$
$y = x.^2 + 3 * x;$
$plot(x, y)$
$figure$
$z = exp(x);$
$plot(x, z)$

El resultado será:



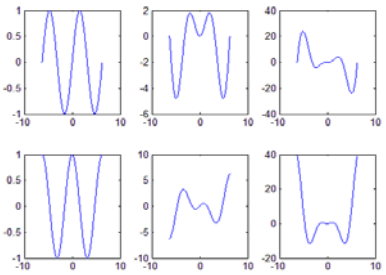
### 2.1.2. *Subplot*.

El comando *subplot*, nos permite realizar distintas gráficas y distribuirla según una matriz, dentro de una misma figura. Su terminología es $subplot(n,m,i)$, donde $n$ nos indica el número de filas de la matriz, $m$, el número de columnas, e $i$, la situación dentro de la matriz, contando por filas y empezando por la izquierda.

**Ejemplo 2.1.1.**– Deseamos representar, dentro de la misma ventana, pero en gráficas diferentes, las funciones $f(x) = sen(x)$, $g(x) = cos(x)$, $h(x) = xsen(x)$, $t(x) = xcos(x)$, $u(x) = x^2 sen(x)$ y $v(x) = x^2 cos(x)$, todas ellas, en el intervalo $[-2\pi, 2\pi]$. Para que se vean comparativamente las funciones seno y coseno, pondremos en la primera fila, todas las que tienen seno, y en la segunda fila, todas las que tienen coseno. Es decir, queremos representarlas en una rejilla de $2 \times 3$ celdas. Para ello:

$x = -2 * pi : pi/100 : 2 * pi;$
$y1 = sin(x); y2 = x. * sin(x); y3 = (x.^2). * sin(x);$
$z1 = cos(x); z2 = x. * cos(x); z3 = (x.^2). * cos(x);$
$subplot(2,3,1); plot(x, y1)$
$subplot(2,3,2); plot(x, y2)$
$subplot(2,3,3); plot(x, y3)$
$subplot(2,3,4); plot(x, z1)$
$subplot(2,3,5); plot(x, z2)$
$subplot(2,3,6); plot(x, z3)$

El resultado es:



**Ejercicio 2.1.2.**– Representar las funciones $f(x) = x^2$, $g(x) = e^{x^2}$, $h(x) = sen(x^2)$ y $t(x) = log_1 0(1 + x^2)$, en una matriz de $2 \times 2$.

## 2.2. Cálculo simbólico.Taylor.

Para definir una función en Matlab, se usa el comando *syms* seguido del nombre de la/s variable/s que tenga la función:

$syms\ x\ y$
$f = x * sin(x)/cos(y);$

Con esas dos líneas, hemos definido la función $f(x,y) = \frac{xsen(x)}{cos(y)}$.

Para calcular el polinomio de Taylor de una función, Matlab tiene dos comandos, *taylor* y *taylortool*. Para utilizar el primero, hay que indicarle la función, el punto $x_0$ en el que se desea calcular el polinomio, y el grado del polinomio de Taylor que se desea (Matlab

considera el número de términos del polinomio, por lo que se debe introducir el grado del polinomio más uno).

**Ejemplo 2.2.3.**– Si queremos el polinomio de Taylor de la función $f(x) = ln(1+x)$ en el punto $x_0 = 0$ de grado 5, escribiremos:

*syms x*
$f = log(1+x)$
$taylor(f, 0, 6)$

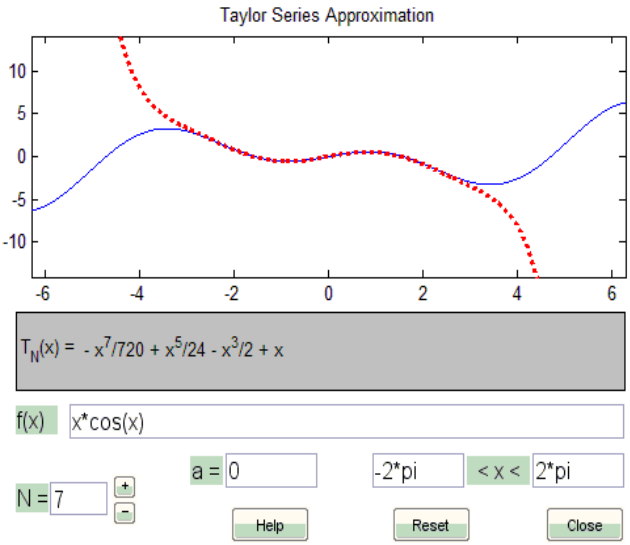
**Ejercicio 2.2.4.**– Calcular el polinomio de Taylor de grado 4 de las siguientes funciones, en los puntos que se indican:

1. $f(x) = sen(x + \frac{\pi}{2})$ en $x = 0$.
2. $f(x) = \frac{1}{x+1}$ en $x = 0$.
3. $f(x) = cos(x)$ en $x = \frac{\pi}{2}$.

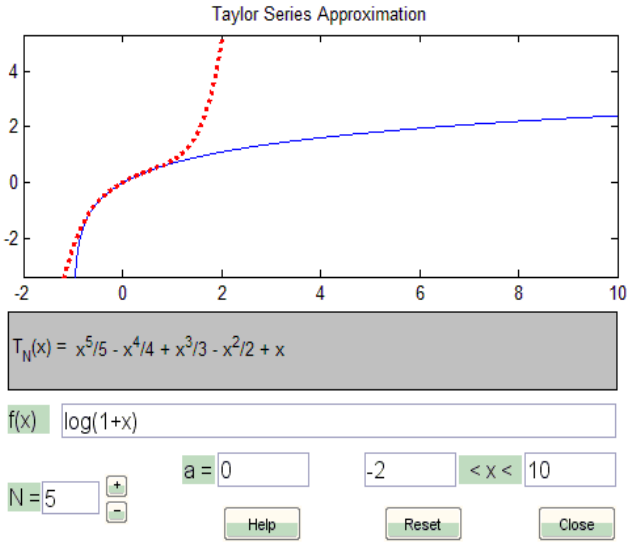
Al usar el comando Taylortool, la forma de trabajar es distinta, se empieza escribiendo simplemente el comando Taylortool:

*taylortool*

Y sale la siguiente ventana gráfica:




En dicha ventana, nos sale la última función que se estudió con dicho comando, en la ventana $f(x)$, podemos sustituir la función por la nuestra, $ln(1+x)$, en $a$, hay que poner el punto $x_0$ en el que se desea desarrollar la función $f(x)$, en nuestro caso, 0, en $N$, el grado del polinomio, y teníamos grado 5 (pulsamos sobre más o menos, según corresponda). También podemos cambiar el rango en el que se observa la función, cambiémoslo por $(-2, 10)$. El resultado será:




En $T_N(x)$, tenemos el polinomio de Taylor, en la gráfica, de color azul, tenemos representada la función $f(x)$, y en color rojo, el polinomio de Taylor asociado, lo que permite observar con claridad el grado de aproximación entre ambos y el intervalo más adecuado para realizar dicha aproximación.

## 2.3. Ficheros .m

En Matlab, se pueden crear archivos de extensión.m,que se denominan archivos de función ó archivos de guión (o de instrucciones). Con ellos podemos definir nuevas funciones que se añadirán a las que ya trae Matlab.
Los archivos de instrucciones se llaman así porque pueden consistir en una serie de instrucciones a ejecutar. Veremos que en éstos puede ocurrir que en el momento de su ejecución nos pida una serie de valores (inputs). En todos los casos, se deberá ir a $File - New-$ y aparece una ventana ($Editor - Untitled$) donde podemos escribir el programa correspondiente.

Una vez terminado, vamos al $File$ de dicha ventana y hacemos clic en $Save\ As$ y debemos guardar el archivo .m con el nombre que le hayamos dado a la función (nombre.m). A partir del momento en que se crea la fución, si se desea utilizar, hay que situar el $current\ folder$ en la carpeta en la que hayamos guardado la función.

El nombre de la función debe empezar con una letra y para evitar confusiones, debemos asegurarnos que no coincide con el nombre de alguna de las funciones de que dispone Matlab.

Las dos primeras líneas de un archivo de función tienen la forma $function[y,z,\ldots] = nombre(a,b,\ldots)$

Donde $a,b,\ldots$ denotan las variables de entrada (las variables independientes), mientras que $y,z,\ldots$ son las variables de salida (dependientes), en ambos casos separadas por

comas. El comando $function$, es de Matlab, mientras que en $nombre$, se debe introducir algo que nos sirva para identificar la función en el futuro.

**Ejemplo 2.3.5.**– Crear un archivo de función cuya entrada sea una matriz cuadrada $x$ y cuyas salidas sean su número de componentes total y su determinante.

$function[numero, determinante] = ejemplo1(x)$
$[n, m] = size(x);$
$numero = n * m;$
$determinante = det(x);\ fprintf('\text{El número de elementos de la matriz es :}')$
$numero$
$fprintf('\text{El determinante de la matriz es :}')$
$determinante$
$end$

Aplicar dicha función a la matriz:

$$\begin{pmatrix} 1 & 2 & 3 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

**Ejemplo 2.3.6.**– Crear un archivo de función con el nombre somb.m para añadir al listado de funciones de Matlab la función $f(x) = \frac{sen(x)}{x}$ , si $x \neq 0$, y $f(0) = 1$.

$function[y] = somb(x)$
$if\ x = 0$
$y = 1$
$else$
$y = sin(x)/x$
$end$

Una vez creado el archivo $somb.m$, cuando tecleamos $somb(x)$, obtenemos el valor de $y(x)$. Aplicar la función a $x = 0$ y $x = \frac{\pi}{2}$

**Ejemplo 2.3.7.**– Un archivo de función que determina la suma de las componentes y dicha suma dividida entre el número de componentes (media) de un vector fila.

$function[a, media] = media(x)$
$n = length(x);$
$suma = sum(x);$
$function[a] = med(suma, n)$
$\%Calculalamedia$
$a = suma/n;$
$end$
$fprintf('\text{La suma de las componentes es :}')$
$suma$
$fprintf('\text{La media es :}')$

$media = med(suma, n)$
$end$

Aplicarlo al vector fila $x = [1, 2, 3, 4, 5]$

**Ejemplo 2.3.8.–**
Vamos a crear una función que, dados tres coeficientes $a, b, c$, nos de las soluciones reales de la ecuación de segundo grado $ax^2 + bx + c = 0$

$function[x1, x2] = ecuacion2(a, b, c)$
$Delta = b^2 - 4 * a * c;$
$if Delta < 0$
$disp('\text{no hay raíces reales}')$
$elseif\ Delta > 0$
$x1 = (-b + sqrt(Delta))/2 * a$
$x2 = (-b - sqrt(Delta))/2 * a$
$else$
$x1 = -b/(2 * a)$
$x2 = -b/(2 * a)$
$disp('\text{raíz doble}')$
$end$

Calcular las raíces de las ecuaciones de segundo grado $x^2 - 2x + 1 = 0$, $x^2 + x + 1 = 0$ y $x^2 - 1 = 0$.

## 2.4. ANEXOS

### 2.4.1. Anexo I.Aspectos del gráfico.

Si se desea, Matlab permite modificar el aspecto del gráfico con algunos comandos:

- $xlabel('\ldots')$, $ylabel('\ldots')$, permiten poner una etiqueta en los ejes $x$ e $y$ respectivamente. También tenemos el comando $gtext('\ldots')$, que nos permite poner la etiqueta donde deseemos usando el ratón sobre el gráfico.

- Para modificar el aspecto de una línea, se tienen, entre otras, las siguientes opciones: $+$, para poner símbolos de suma, $-$, para poner una línea discontinua, ., para poner una línea con puntos, $x$, $*$, ..... Para usarlos, se pondría, por ejemplo, $plot(x, y, '+')$.

- Para cambiar el color de una gráfica, se añade (en general), la inicial del nombre del color que se desea (en inglés), por ejemplo $r$ es para ponerlas en rojo, $b$, para azul, $y$ para amarillo, $g$ para verde, $k$ para negro, ..., se usaría de manera análoga a los estilos de línea: $plot(x, y, 'r')$

- Si se desean cambiar simultáneamente el estilo de la línea y el color, se ponen ambos comando juntos entre las comillas: $plot(x, y, 'r+')$.

- Para modificar la amplitud de los ejes (siempre dentro de los límites en los que tenemos definidas las variables), se puede usar el comando $axis([xmin, xmax, ymin, ymax])$, donde $xmin$ simboliza la mínima $x$ que deseamos ver, $xmax$, la máxima, y análogamente, con $ymin$ e $ymax$. Otras opciones son $axis\ square$, para que los ejes definan

un cuadrado, o $axis('equal')$, si queremos que en ambos ejes tengamos la misma escala.

- Para que nos aparezca la cuadrícula (o no), tenemos el comando $grid\ on$ ($grid\ off$).

### 2.4.2. Anexo II.Operadores lógicos.

Matlab emplea los tres operadores lógicos siguientes: & (y), |(ó) y $\sim$ (negación).

Cuando operan sobre una matriz, lo hacen elemento a elemento. Cualquier número no nulo se considera como verdadero y el cero como falso. Consideremos, por ejemplo, la instrucciones siguientes:

$A = [1\ 0; 2\ 1];$
$B = [4\ 0; 0\ 3];$
$C = A\&B$
$C =$
1 0
0 1

Es decir, produce una matriz con unos y ceros , dando como resultado un 1 cuando ambos elementos sean distintos de cero, y 0, cuando al menos uno sea igual a 0.

### 2.4.3. Anexo III.If, else y elseif

$If$ evalúa una expresión lógica y ejecuta un grupo de instrucciones si la expresión lógica es verdadera. La forma más simple es la siguiente

$if$expresión lógica
*instrucciones*
*end*

$elseif$ evalúa la expresión lógica que aparece en su misma línea si $if$ o los anteriores $elseif$ resultan falsos. Si dicha expresión lógica es verdadera se ejecutan las instrucciones siguientes hasta el próximo $elseif$ o *else*. Por el contrario, *else* no lleva expresión lógica a evaluar y matlab realiza las instrucciones siguientes hasta *end* si las expresiones lógicas de $if$ y los $elseif$ anteriores son falsos. La forma general de un if es la siguiente:

$if$ expresión lógica
*instrucciones*
$elseif$ expresión lógica
*instrucciones*
$elseif$ expresión lógica
*instrucciones*
...
*else*
*instrucciones*
*end*

### 2.4.4. Anexo IV.For

La sintaxis es la siguiente:

*for* índice = *inicio* : *incremento* : *final*
*instrucciones*
*end*

Si el incremento es la unidad, no hace falta indicarlo.

### 2.4.5. Anexo V.while

Su sintaxis es la siguiente:

*while* expresión
*instrucciones*
*end*
Mientras que la expresión que controla el *while* sea verdadera, se ejecutan todas las instrucciones comprendidas entre *while* y *end*.