UNIVERSIDAD DE MURCIA

Departamento de Matemáticas
Cálculo Numérico en una Variable
Curso 2023/24

Prof. S. Sánchez-Pedreño

Práctica 2. Gráficos y funciones en Octave

9 y 13 de octubre de 2023

El objetivo principal de esta práctica es el de familiarizarnos con las herramientas gráficas de Octave: dibujo de gráficas en el plano, diferentes tipos de línea en el gráfico, colores, leyendas, etc. También aprenderemos a manejar las funciones de Octave de distintas formas.

Introducción: herramientas gráficas de Octave

Recordemos que Octave y Matlab trabajan fundamentalmente con vectores y matrices, es importante tenerlo en cuenta, puesto que para dibujar una curva en el plano lo que haremos (con la orden gráfica más elemental) será pasarle un vector de abscisas y un vector de ordenadas. Con esos datos Octave dibujará una poligonal (que puede parecer una línea continua...) que une los puntos del plano dados por las componentes de ambos vectores: eso será una curva gráfica para Octave.

Comenzamos por tanto con algunas herramientas para crear o manejar vectores o matrices.

Atención: los índices de las componentes de un vector o matriz se numeran empezando en 1, no en 0 (al contrario que en Java u otros lenguajes).

Orden	**Descripción**	**Ejemplo (a teclear)**
`[*,*,*;*,*,*]`	Crea una matriz descrita por valores concretos (`*`). Las columnas se separan mediante una coma, las filas mediante punto y coma (`;`)	`>> A=[1,2,3;9,10,11]` `>> B=[2,4,6,8]`

`A(n,m)`	Devuelve el elemento de la matriz `A` en la fila `n` y la columna `m`. Naturalmente para vectores sólo necesitamos una posición. Recuerda: `B(0)` producirá un error.	`>> A(2,3)`
		`>> B(2)`
		`>> C=A(1,:)`
		`>> D=A(:,2)`

`ini:incr:max` `[ini:incr:max]`	Crea un vector cuyo primer elemento es `ini` y los sucesivos se obtienen sumando `incr` al anterior, hasta alcanzar el valor `max`. Si el valor max se sobrepasa (porque al incrementar sucesivamente no lo obtenemos exactamente) el último elemento será menor que `max`.	`>> A=2:3:32` `>> B=[1:0.1:2.005]`

`linspace(ini,fin,n)`	Crea un vector con `n` componentes cuyos valores están equidistribuidos en el intervalo `[ini,fin]`.	`>> linspace(1,10,10)`
		`>> linspace(1,10,11)`
		`>> linspace(pi,pi+1,5)`

`zeros(n,m)`
`zeros(n)`
`zeros(...,"clase")`
	Crea una matriz de ceros, de tamaño `n`×`m`, o `n`×`n` (respectivamente). Las componentes serán de tipo `clase`; si no se incluye, por defecto de tipo `double`.	`>> A=zeros(3)`
		`>> v=zeros(1,5,"uint8")`

`ones(n,m)`
`ones(n)`
`ones(...,"clase")`
	Como la anterior, pero con valores 1.	`>> A=ones(3)`
		`>> v=ones(1,5,"uint8")`

Los primeros gráficos

El comando básico para realizar gráficas bidimensionales es: `plot(x,y)`, donde `x` será el vector de abscisas e `y` será el de ordenadas.

Por ejemplo: teclea en la ventana de comandos

```
>> x=[1,2,3]
>> y=[0.3,2.5,0.9]
>> plot(x,y)
```

Comprobarás que se abre una ventana, con nombre «`Figure 1`», que tiene una botonera con algunas herramientas (desplazamiento, zoom, etc.), puedes moverla, ampliarla, etc. como cualquier otra ventana. Experimenta con los botones... Después cierra la ventana gráfica (con el ratón)

Ahora vamos a crear varias ventanas. Podemos intentar lo siguiente (teclea en la ventana de comandos):

```
>> x=[1,2,3]
>> y=[0.3,2.5,0.9]
>> plot(x,y)
>> plot([4,5,6],[2,-2,0])
```

¿Qué ha ocurrido?... que el segundo gráfico ha reemplazado al primero en la ventana de la «`Figure 1`». Para evitar esto hacemos:

```
>> x=[1,2,3]
>> y=[0.3,2.5,0.9]
>> plot(x,y)
>> figure(2)
>> plot([4,5,6],[1,1.5,1])
```

En los ejemplos que siguen repetiremos con frecuencia algunas órdenes en la ventana de comandos: si utilizas la flecha hacia arriba del teclado, en esta ventana, podrás ir repitiendo órdenes y editarlas, si quieres modificarlas.

Algunas órdenes interesantes para el manejo de distintos gráficos

Cierra la ventanas gráficas creadas antes. A continuación teclea en la ventana de comandos las órdenes indicadas en lo que sigue, en el orden indicado.

`figure(n)` Crea una ventana gráfica con nombre «`Figure n`». Si ya existe esta ventana entonces la declara como activa (tiene el foco): es decir, los comandos gráficos siguientes le afectarán a dicha ventana.

```
>> figure(1)
>> plot([1,2,3],[3,4,5])
>> figure(2)
>> plot([1,2,3],[-3,-4,-5])
>> figure(1)
>> plot([3,4,8],[-2,1,-5])
```

¿En qué ventana se ha dibujado la última gráfica?

`clf(n)` Vacía el contenido de la ventana «`Figure n`», pero la ventana sigue abierta. Qué es exactamente el contenido y qué desaparece es complicado de explicar ahora: una ventana gráfica es como un contenedor... `clf` elimina ciertas cosas del contenedor, pero no otras y esta orden tiene otras sintaxis más ricas, mediante las que se puede especificar más detalles de los elementos a eliminar... De momento nos contentamos con esto.

```
>> clf(1)
```

`hold on` Permite añadir elementos gráficos a la ventana gráfica que tiene el foco, sin eliminar los que ya existen.

```
>> figure(2)%%Por si no tiene el foco en estos momentos, lo ponemos en la fig. 2
>> hold on
>> plot([9,10,11,12],[1,2,1,2])
>> hold on
>> plot([3,4,8],[-2,1,-5])
```

`close(n)` Cierra la ventana «`Figure n`».

```
>> close(1)
```

`close all` Cierra todas las ventanas «Figure...»

Elementos gráficos adicionales

Las posibilidades gráficas en Octave son bastante completas. Aquí vamos a describir sólo algunas de ellas.

En primer lugar, y **muy importante**:

- Podemos incluir varias gráficas simultáneamente en un mismo comando `plot`:

  ```
  >> close all%% Para empezar de nuevo
  >> x1=[1,2,3]
  >> y1=[0.1,2.5,-1.2]
  >> x2=[1,2,3,4]
  >> y2=[-0.1,-2.5,1.2,-3.4]
  >> plot(x1,y1,x2,y2)
  ```

- Octave escala automáticamente el eje de ordenadas para que se ajuste al tamaño de la gráfica (también el de abscisas, si por ejemplo se añade un segundo gráfico a una ventana ya existente). Podemos fijar los límites de los intervalos de abscisas y ordenadas, con las órdenes `xlim([xmin,xmax])`, `ylim[ymin,ymax]`. Por ejemplo, siguiendo con las órdenes ya tecleadas en el punto anterior:

  ```
  >> xlim([-3,2])
  >> ylim[-2,0]
  ```

  Observa lo que ocurre, y utiliza el botón de desplazamiento de la ventana gráfica para comprobar que las gráficas «siguen ahí». Estas órdenes sólo se ocupan de determinar la parte visible del plano. Cada gráfico añadido a una ventana gráfica cambia la escala de los ejes, así `xlim` e `ylim` deben utilizarse después de incluir un gráfico.

Antes de introducir otros elementos de control de gráficos vamos a dibujar algunas gráficas menos triviales que las de los ejemplos anteriores.

Por ejemplo, para dibujar la función $\mathrm{sen}(x)$ en una malla de puntos (201) en el intervalo $[-\pi, \pi]$:

```
>> x=linspace(-pi,pi,201)
>> plot(x,sin(x))%% observa el intervalo de abscisas en el gráfico
>> xlim([-pi,pi])%% ahora el intervalo de abscisas ha cambiado
```

Otro ejemplo, utilizando «funciones anónimas»:

```
>> mifun=@(x) 2*x./(1+x.^2);
>> x=linspace(-3,3,201);
>> plot(x,mifun(x))
```

Incluyendo puntos en un gráfico

Se pueden incluir puntos en un gráfico pasando a `plot` dos vectores: un vector con las abscisas de los puntos que se quieren dibujar, y un vector con sus ordenadas, e incluyendo además una indicación de cómo dibujar un punto, mediante el símbolo adecuado, por ejemplo `'*'`, `'o'`:

```
>> mifun=@(x) 2*x./(1+x.^2);
>> x=linspace(-3,3,11);
>> plot(x,mifun(x))
>> hold on
>> plot(x,mifun(x),'*',[0],[0],'o')
```

Colores y leyendas

Podemos controlar muchos otros elementos, como los colores con los que se dibujan las gráficas o los puntos, el estilo de las líneas y los puntos, así como la inclusión de leyendas, títulos de la ventana, etc. He aquí algunos ejemplos.

La orden `plot` tiene numerosas variantes en cuanto a las variables que espera. Una de ellas incluye el «formato», una cadena de caracteres que controla cómo se muestra cada elemento gráfico.

El «formato» es una cadena de caracteres compuesta por cuatro partes, todas ellas optativas. Estas partes son: estilo de la línea, estilo de puntos (marcadores), color y leyenda. Atención: la leyenda, si se incluye, debe encerrarse entre signos de punto y coma, es decir, debe ser de la forma `;leyenda;`.

Veamos un ejemplo rápido:

```
>> close all
>> x=linspace(-3,3,200)
>> plot(x,mifun(x),'g--;Mi función;')
>> hold on
>> plot(x,mifun(x),'m*;Puntos;',[0,-3,3],[0,0,0],'o')
```

En el primer caso hemos especificado en el formato el color verde (`g`, de «green>), un estilo de línea discontinua (`--`), y la leyenda (`Mi función`). En el segundo caso se dibujan sólo puntos (debido al formato especificado con `*`), en color magenta (`m`) y una nueva leyenda para este elemento gráfico adicional.

A continuación tenéis una lista de valores admitidos en el formato:

Estilos de línea Posibles valores: `-` (línea continua, valor por defecto), `--` línea discontinua, `:` línea de puntos, `-.` discontinua compuesta de raya y punto.

Colores Posibles valores (entre otros): `b`, `r`, `g`, `y`, `m`, `c`, `w`.

Marcas de puntos Entre otros: `+`, `o`, `*`, `.`, `x`, `s`, etc.

Una sintaxis alternativa, y que ofrece más posibilidades es

```
plot(x,y, prop1,valor1,prop2,valor2,...)
```

donde `prop1`, `prop2` representan el nombre de distintas propiedades, seguidas cada una de su valor. Entre las propiedades: `linestyle`, `linewidth`, `color`, `marker`, `markersize`, `markeredgecolor`,...

El nombre de la propiedad y los valores que no sean numéricos deben escribirse entre apóstrofos o comillas (como todas las cadenas). Un ejemplo con pequeñas variaciones sobre el anterior:

```
>> close all
>> x=linspace(-3,3,200)
>> plot(x,mifun(x),'color','c','linestyle','-.','linewidth',2)
>> legend('Mi función')
>> hold on
>> plot(x,mifun(x),'marker','*',[0,-3,3],[0,0,0],'marker','o','markerfacecolor','y')
```

La orden `legend` es otra forma de incluir una leyenda, admite muchas variantes, pero no entraremos en ellas ahora.

Para más detalles, puedes consultar los siguientes enlaces sobre Octave:
https://octave.org/doc/v4.0.0/Two_002dDimensional-Plots.html
https://octave.org/doc/v4.0.0/Two_002ddimensional-Function-Plotting.html

Ejercicios para la práctica 1

1. Este primer ejercicio consiste en dibujar algunas gráficas sencillas, para familiarizarnos con los elementos gráficos. Debemos escribir el código en un archivo denominado `Ejercicio1.m`. Como se van añadiendo elementos apartado a apartado, utiliza `close all`, desde la ventana de comandos, para cerrar la ventana gráfica entre apartado y apartado, antes de volver a ejecutar el script (también puedes cerrar directamente la ventana con el ratón).

   a) Dibuja en una ventana la gráfica de la función $f(x) = \pi + \sqrt{1+x^2}$ en el intervalo $[-3, 3]$, utilizando 179 puntos: en un script podemos definir una función con el código:

   ```
   f=@(x) pi+sqrt(1+x.^2);
   ```

   Aunque de forma alternativa podemos definirla, dentro del propio archivo script, mediante el código

   ```
   function ret=f(x)
   ret=pi+sqrt(1+x.^2)
   endfunction
   ```

   Aunque en este ejemplo no se ve la necesidad de la segunda opción, es conveniente recordar que la primera sintaxis sólo permite una expresión tras la parte de código `@(x)`. En ambas formas, la definición de la función debe preceder en el código a cualquier llamada a la misma.

   b) Cambia el color de la gráfica y el grosor de la línea (el valor del grosor por defecto es `0.5`). Añade una leyenda.

   c) Añade a la misma ventana la gráfica de la función $g(x) = \text{sen}(x)$ sobre el mismo intervalo. Elige el color de esta nueva gráfica y fija un estilo de línea discontinuo. Incluye también una leyenda.

   d) Finalmente, añade en la misma ventana la gráfica de $h = f + g$.

   e) Añade la orden `title('Mi primer gráfico')`. ¿Qué efecto produce?

   f) Conviene iniciar el archivo `.m`, con el código `close all`, para que las ventanas gráficas se cierren (si es lo que deseamos) al iniciar una nueva ejecución del script, evitando que los gráficos y leyendas se acumulen con los de la ejecución anterior, ¡recuérdalo!

2. Para calcular el valor de $p(x) = (x-1)^5$ podemos optar por cualquiera de las tres formas siguientes:

   a) $(x-1)^5$;
   b) $x^5 - 5x^4 + 10x^3 - 10x^2 + 5x - 1$;
   c) $-1 + x(5 + x(-10 + x(10 + x(-5 + x))))$ (forma anidada o de Horner).

   Dibuja la gráfica de cada una de estas formas de evaluación, en una ventana gráfica distinta, en el intervalo $[0.998, 1.002]$, utilizando una malla de al menos 1000 puntos. Haz el cálculo y los gráficos primero en precisión doble y después en precisión simple. ¿Qué observas? ¿Qué implicaciones te parece que puede tener este efecto a la hora de aproximar una raíz de un polinomio?

   Como verás las ventanas gráficas se superponen unas a otras o tienen un tamaño o proporciones inadecuadas. A veces por tanto conviene poder decidir dónde se van a colocar en pantalla y qué tamaño tendrán. Para esto disponemos de la orden siguiente:

   ```
   set(1,"position",[x0,y0,x1,y1])
   ```

   Con esta orden especificamos valores para la ventana gráfica correspondiente a la `figure 1`; los valores `x0`, `y0` son las coordenadas de pantalla de la ventana de la figura y `x1`, `y1` son las unidades de medida del ancho y alto. Para orientarte sobre los valores de estas dimensiones puedes obtener los valores de cualquier ventana gráfica que esté abierta: suponiendo que es la figura `1`,

   ```
   get(1,"position")
   ```

3. En este último ejercicio vamos a manipular funciones de Octave e intentar comprender cómo funcionan para evitar posibles problemas en el futuro.

   Recuerda:

   - Un archivo de función en Octave es un archivo de extensión `.m` que comienza con la orden

     `function... NombreFuncion...`

     y en ese caso el nombre del archivo debe coincidir con el nombre de la función. El archivo puede contener otras funciones, pero solo la inicial (principal) será accesible desde otros script o funciones (aunque todas serán accesibles desde el propio archivo de función).
   - Un archivo script también tiene extensión `.m`, pero no tiene ninguna estructura particular y puede contener una o varias funciones.

   Las tareas en este ejercicio serán:

   a) Definir en el script `Ejercicio3.m` la función $f(x,a) = \exp(-x^2) * \exp(-a^2)$ mediante una sentencia `function`. En esta función (de dos variables) interpretamos la variable a como parámetro: el objetivo es dibujar la gráfica de $f_a(x) = f(x,a)$ para distintos valores de a y, además, aproximar el valor máximo de f_a en cierto intervalo.

   b) Crearemos en la carpeta `Practica2` el archivo de función `auxiliar.m` que contendrá una única función: `auxiliar(func,par,s)`. La función `auxiliar` recibirá como parámetro el nombre de una función (variable `func`, que será el nombre de f), un valor del parámetro a (variable `par`) y un vector de abscisas en el intervalo $[-5,5]$ (variable `s`). La función `auxiliar` debe definir la función f_a, calcular el máximo de los valores de esta función en el vector de abscisas `s` y dibujar su gráfica con el mismo vector de abscisas.

   c) El script `Ejercicio3.m` incluirá un bucle que asigne valores al parámetro a, variando desde -2 a 2 con incrementos de 0.1, y para cada uno de esos valores hará una llamada a la función `auxiliar`.

   d) Una vez conseguido que funcione todo lo anterior introduciremos algunos detalles: las gráficas se deben dibujar todas en la misma ventana, cada una reemplazando a la anterior, manteniendo una escala vertical fija (con `ylim[-0.1,1.1]`) y entre un paso del bucle y otro introduciremos la orden `pause(0.15)`, es decir, una pausa de 0.15 segundos.

   Observaciones sobre Octave.

   - Recuerda que la sintaxis `@func` es un puntero a una función de nombre `func`. Por otro lado la sintaxis

     `@(x,y)... expresión en x e y`

     define una función anónima (y a la vez un puntero a dicha función). Por tanto,

     `f=@(x,y)... expresión en x e y`

     establece `f` como puntero a una función anónima.
   - **Para pasar como argumento de una función el nombre de otra función, debemos utilizar un puntero a la misma**. Así, por ejemplo, el código siguiente dará un error:

     ```
     function ... f(...)
     ...
     ...
     auxiliar(f,par,x)
     ```

     Asumiendo definida la función

     ```
     function ... f(...)
     ```

     las siguientes sintaxis son alternativas que no producen error:

     ```
     auxiliar(@f,par,x)
     ...
     ```

```
h=@(x,a) f(x,a)
auxiliar(h,par,x)
...
auxiliar(@(x,a) f(x,a),par,x)
...
auxiliar(str2func("f"),par,x)
```

La última sentencia es extraña: `str2func("cadena")` produce un puntero a la función de nombre `cadena` (hay que pasar una cadena a `str2func`).