Cálculo con GNU Octave
Curso 2016-2017

Departamento de Matemáticas

Prólogo

GNU Octave es un software ampliamente difundido en el campo de la docencia y de la investigación en numerosas universidades, porque es un programa —de alto nivel— de cálculo científico, fácil de usar y con gran potencial computacional y gráfico. Puede considerarse la versión gratuita (*free*) de MatLab, por lo que la sintaxis es esencialmente la misma. Es, por tanto, fundamentalmente un programa de cálculo matricial (los vectores son su estructura esencial). No obstante, GNU Octave —en adelante Octave— es también una potente herramienta usándolo como lenguaje de programación.

En el texto de cada práctica, se introducen los comandos necesarios para resolver los ejercicios propuestos en dicha práctica y, en su caso, en las siguientes.

La calificación final de las prácticas de Cálculo con Octave (que se evalúan sobre 3 puntos) se obtiene al final del cuatrimestre como consecuencia de la realización de los ejercicios propuestos en cada una de las prácticas y/o la evaluación de ejercicios finales.

Los ejercicios evaluables se realizarán durante las horas de clase, por lo que la calificación final se verá afectada por las faltas de asistencia acumuladas a lo largo del cuatrimestre.

La calificación de prácticas obtenida es válida para la segunda oportunidad de julio. Sólo los alumnos con matrícula parcial, que no hayan aprobado la materia en la primera oportunidad y/o no han sido calificados de la parte de prácticas de laboratorio, pueden en la segunda oportunidad de julio realizar un examen para evaluar (sobre tres puntos) las prácticas de laboratorio.

Bibliografía recomendada

Además de la ayuda en línea del propio programa se puede encontrar el manual de ayuda en el enlace:

`http://www.gnu.org/software/octave/doc/interpreter/`

Por otra parte, existen muchos textos que, en momento dado, pueden servir de ayuda en la resolución de los ejercicios propuestos en las prácticas. En particular, dada la similitud entre Octave y MatLab, cualquier texto sobre este último lenguaje, de los muchos que se encuentran en la biblioteca de la facultad, servirá a nuestros propósitos, al menos en aspectos puntuales del programa de la asignatura. Algunos de ellos son:

Iglesias, M.T. *Matlab para Cálculo en una variable*. Andavira Editores, 2011.
(Libro expresamente orientado a las clases de prácticas de Cálculo de esta facultad).

Lantarón, S. y Llanas, B. *MATLAB y MATEMÁTICA COMPUTACIONAL*. Bellisco Ediciones, 2010.

Moore, H. *MATLAB for engineers*. Pearson Prentice Hall, 2012.

Pratap, R. *Getting Started with MATLAB. A Quick Introduction for Scientifics and Engineers*. Oxford University Press, 2010.

Introducción a Octave

1.1. Instalación y/o acceso a Octave

Usaremos OCTAVE bajo plataforma *Windows*. La versión con la que trabajaremos es la versión **4.0.0** de 29 de mayo de 2015.

Si estás trabajando en un equipo fijo de los laboratorios de la Facultad, para acceder al programa es necesario que te conectes al equipo como usuario. Para ello en la pantalla de entrada debes teclear

`usuario@windows.cc.fic.udc.es`

donde, en lugar de la palabra `usuario`, debes introducir tu nombre de usuario (en inglés *login*). El sistema pedirá entonces tu clave personal (*password*).

Encontrarás el icono del programa en el escritorio de Windows.

Si perteneces a un grupo *wifi*, deberás haber instalado el programa en tu portátil CON ANTERIORIDAD A LA PRIMERA CLASE de prácticas.

Los pasos a seguir para realizar la instalación son los siguientes:

1) Descarga el instalador del paquete OCTAVE desde la siguiente dirección:

   `ftp://ftp.gnu.org/gnu/octave/windows/octave-4.0.0_0-installer.exe`

   Como se puede observar en el enlace descargarás la versión 4.0.0 de Octave, que se instalará en `C:\Octave`

2) Instala el paquete Octave dejando todas las opciones por defecto. Es decir, NO MODIFIQUES NADA.

   A lo largo del cuatrimestre realizaremos cálculos de límites, derivadas, integrales, etc. , para lo que debemos tener instalados un conjunto de comandos que se agrupan en una librería (*package*) adicional al programa básico. Es la librería `symbolic`. Así pues,

3) Teclea:

   `https://github.com/cbm755/octsympy/releases/tag/v2.2.2`

   Recorre la página en la que has entrado, hasta el final de la misma. Verás varios enlaces. Debes descargar

   `symbolic-win-py-bundle-2.2.2.zip`

   (Estamos instalando la versión 2.2.2 de la librería `symbolic` de 3 de julio de 2015)

4) Abre el programa Octave (se habrán creado dos iconos en el escritorio); ábrelo desde el icono etiquetado `Octave (GUI)`.

   Ahora, desde la carpeta donde está el fichero `.zip` anterior, ejecuta en la **ventana de comandos** de OCTAVE:

   ```
   pkg install symbolic-win-py-bundle-2.2.2.zip
   ```

   Si todo se ha instalado correctamente, debería aparecer el siguiente mensaje:

   ```
   For information about changes from previous versions of
   the symbolic package, run 'news symbolic'.
   ```

   Dependiendo de la versión de Windows instalada en tu equipo, el mensaje puede ser este otro:

   ```
   Warning: creating installation directory C:\Octave\Octave4.0.0\share\octave\packages

   warning: called from install at line 30 column 5

                           pkg at line 405 column 9

   For information about changes from previous versions of the symbolic
   package, run 'news symbolic'.
   ```

5) Cada vez que se quiera usar `symbolic`, habrá que cargarlo tecleando:

   ```
   pkg load symbolic
   ```

6) Para comprobar que se ha cargado, basta teclear en la ventana de comandos:

   ```
   pkg list
   ```

   que responderá:

```
Package Name  | Version | Installation directory
--------------+---------+-----------------------
    symbolic *|   2.2.2 | C:\Octave\Octave-4.0.0\share\octave\packages\symbolic-2.2.2
```

7) Para comprobar el funcionamiento correcto de la librería `symbolic` podemos, por ejemplo, derivar x^2. Para ello, escribimos:

   ```
   syms x
   ```

   La primera vez que hacemos esto aparecerá el siguiente mensaje:

   ```
   OctSymPy v2.2.2: this is free software without warranty, see source.
   Initializing communication with SymPy using a popen2() pipe.
   Detected Windows: using "winwrapy.bat" to workaround Octave bug #43036
   Some output from the Python subprocess (pid 3556) might appear next.

   OctSymPy: Communication established.  SymPy v0.7.6.
   Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)]
   ```

   Ahora derivaremos x^2. Para ello, teclea:

   ```
   diff(x^2, x)
   ```

   Obtendremos como respuesta:

   ```
   ans = (sym) 2*x
   ```

Explicaremos estos y otros comandos de la librería `symbolic` en prácticas posteriores.

1.2. Empecemos con Octave

Cuando se accede al programa en pantalla se muestran hasta cinco subventanas: el `explorador de archivos`, el `espacio de trabajo`, el `historial de comandos`, el `editor` y la `ventana de comandos`.

La subventana más grande, situada a la derecha de la pantalla, es la `ventana de comandos`. A su izquierda, están el `explorador de archivos` o `directorio activo`, el `espacio de trabajo` y el `historial de comandos`, que es una ventana que va guardando todos los comandos utilizados, por si fuera necesario ejecutar de nuevo una orden (o teclear una parecida).

Si la apariencia no es la indicada en estas líneas basta escoger, en el menú de herramientas, el icono etiquetado `Ventana` y marcar las seis primeras opciones. Así habremos activado también la visión del `editor` que aparecerá en una ventana "flotante" que podemos anclar en la parte inferior de la pantalla. Para ello debemos pinchar en el icono situado en la parte superior derecha de la ventana del `editor`, junto al aspa.

Es posible ocultar aparentemente el editor si en la opción `Ventana` del menú de herramientas escogemos la última opción: `Restablecer el esquema de ventana predeterminado`. Entonces podemos observar que, en la parte baja de la ventana de comandos, hay tres pestañas, dos de las cuales permiten pasar del `editor` a la `ventana de comandos` y viceversa. La tercera pestaña se dedica a la documentación y contiene la ayuda del programa (**help**) por la que se puede navegar. Volveremos sobre la ayuda más adelante.

Por defecto el directorio activo es el disco local **C**. Sin embargo, **si trabajas en un equipo fijo** (no en tu portátil) debes cambiar de directorio para trabajar en el disco **S**, creando una carpeta (*New Folder*) para las prácticas de OCTAVE en tu directorio personal que se identifica con tu nombre de usuario.

Grabar prácticas en modo local (en **C**) *no garantiza su permanencia ahí durante todo el cuatrimestre.*

Es en la `ventana de comandos` en donde se escriben las órdenes a ejecutar. Las instrucciones se escriben a la derecha del indicativo `>>`; es decir, en donde parpadea el cursor.

Por ejemplo, la orden **exit** cierra el programa.

La ventana de comandos es similar a la de una calculadora: cuando se apaga se pierden los cálculos realizados.

- Para las cuatro operaciones aritméticas OCTAVE usa los símbolos (`+`), (`-`), (`*`), (`/`).
- Para la potenciación emplea el circunflejo (`^`).

  Al igual que en las calculadoras, las expresiones se evalúan de izquierda a derecha; la potencia tiene el orden de prioridad más alto, seguido del producto y la división (ambas tienen la misma prioridad) y por último están la suma y la resta (con igual prioridad entre ellas). Si se desea alterar este orden se deben introducir, adecuadamente, paréntesis.

  Para que una expresión sea más legible es aconsejable dejar un espacio antes y después del signo de la suma, así como antes y después del signo de la diferencia pero no antes y después del símbolo del producto y/o del cociente.

  Ejemplos:

  ```
  8 + 4/2 - 1
  ```

  ```
  (8 + 4)/2 - 1
  ```

Ejercicio 1.1
Calcula:

a) $\dfrac{3^2+7}{2}$

b) $2^3 - \dfrac{16\,(7+\frac{24}{3})}{2^4 5}$

c) $\sqrt[4]{625}$

Escribe las expresiones con el menor número de paréntesis posibles (¡sólo los imprescindibles!)

- OCTAVE ignora cualquier texto que vaya precedido por el símbolo `%`, por lo que `%` sirve para incluir comentarios. Por ejemplo,

```
15*3    % calculamos el triple de 15
```

- Se pueden introducir varias órdenes, en una misma línea, separadas cada una de las demás por una coma (o un punto y coma).

- Si una orden ocupa más de una línea basta escribir tres puntos suspensivos y continuar su escritura en la siguiente línea.

- Sólo está activa la última línea escrita, pero se pueden usar las flechas del teclado (↑ o ↓) para recuperar líneas ya escritas.

- OCTAVE **distingue entre minúsculas y mayúsculas**. No es lo mismo **sin**(0) que Sin(0). La primera orden calcula el seno de cero radianes, la segunda no. Si has escrito Sin(0), Octave te ayuda sugiriendo posibles soluciones al error cometido.

- **clc** deja en blanco la ventana de comandos, eliminando todas las órdenes introducidas (proviene de ***cl****ear* ***c****onsole*). Sin embargo, siguen estando en el `historial de comandos`.

- Para interrumpir la ejecución de una instrucción se deben pulsar simultáneamente las teclas `CTRL` y `C`.

- **Inf** denota al símbolo infinito (∞)

- **NaN** es la abreviatura de *Not a Number*. Es el resultado de operaciones que no producen un resultado bien definido; por ejemplo, las indeterminaciones $0/0$, $\infty - \infty$.

- Como ya hemos comentado, se puede acceder a la ayuda del programa usando la pestaña que figura en la parte inferior de la ventana de comandos etiquetada como `Documentación`. Pero, si ya conocemos el nombre de un comando y no recordamos su sintaxis, podemos acceder a la ayuda tecleando **help** seguido del nombre del comando. Por ejemplo,

help sqrt

proporciona información sobre la orden **sqrt**.

¿Qué operación realiza ese comando? Utilízalo en un ejemplo.

También obtenemos ayuda sobre un comando si tecleamos

doc *nombre del comando*

Por otra parte, si no recordarmos el nombre completo de un comando puede ayudarnos la tecla del tabulador. Comprébalo tecleando en la ventana de comandos **sq**. Presiona a continuación la tecla de tabulación y obtendrás el nombre de comandos que empiezan por **sq**:

sqp sqrt sqrtm squeeze

1.3. Formatos en Octave para representar las respuestas

Hay varias formas de medir la exactitud con la que una cantidad se almacena en una máquina. La más difundida se basa en el *epsilon de máquina*. El *epsilon de máquina* es la distancia entre 1 y el siguiente número mayor que 1 que se almacena de forma exacta. En OCTAVE **los números se almacenan internamente en doble precisión** guardando cada dato en ocho *bytes* (64 *bits*), lo que se traduce en unas 16 cifras decimales, aproximadamente, al convertirlo al sistema decimal. ¿Por qué? Pues, porque en **doble precisión** el epsilon de máquina es $2^{-52} \approx 2'22 \times 10^{-16}$.

Eso significa que sólo se representan números reales en un determinado rango: entre $2'2 \times 10^{-308}$ y $1'8 \times 10^{308}$ aproximadamente.[1]

Independientemente de cómo se almacenan los datos en OCTAVE, este programa permite ver en pantalla los resultados en diferentes formatos, algunos de los cuales son los siguientes:

- **format short** (es el formato que emplea, por defecto): formato en punto fijo con 4 dígitos en la parte decimal,
- **format rat**: cociente de dos números enteros,
- **format long**: punto fijo con 15 dígitos en la parte decimal,
- **format long** `e`: formato en coma flotante con 15 dígitos en la parte decimal,

Nota 1.1
No se debe confundir la letra `e` *que aparece en la representación de los números en pantalla con la letra que representa en matemáticas —de forma habitual— al número real que es base de los logaritmos neperianos.*

En OCTAVE *el número* `e` *se escribe* **`exp(1)`** *(aunque también sirve* **`e`***.)*

Ejercicio 1.2
Representa en los distintos formatos los siguientes números:

a) $\frac{2}{3}$ *b)* $1'4444$

¿Qué significa la `e` *que aparece al escribir un número en formato* ***long*** `e`*?*

[1] En *simple precisión* el epsilon de máquina es $2^{-23} \approx 1'19 \times 10^{-7}$.

1.4. Variables numéricas

- Para almacenar los resultados de los cálculos se emplean variables.
- Una variable es una etiqueta que identifica una porción de memoria.
- **ans** es el nombre (o identificador) de variable con el que responde OCTAVE por defecto (proviene de *answer*).
- Para asignar un nombre a una variable se debe utilizar el operador de asignación `=` que actúa de forma similar a como se asigna un nombre a un archivo al grabarlo: cuando se actualiza el contenido del archivo y se vuelve a grabar con el mismo nombre se está alterando el contenido del archivo pero se mantiene el nombre.

  Tras cada cálculo realizado el valor de **ans** cambia; por esta razón, es aconsejable asignar nombres a los resultados u operaciones con los que nos interese trabajar posteriormente. (Observa en el `espacio de trabajo` el valor actual de **ans**).
- El identificador de una variable puede contener letras, números y guión subrayado. No puede empezar por un número. Además, las letras deben ser del alfabeto inglés; es decir, no se permiten caracteres propios como nuestra "ñ." **No se permiten tampoco espacios ni tildes**.

  Veamos un ejemplo de asignación de un nombre o identificador a una variable numérica:

  ```
  lado = 3    % lado de un cuadrado, medido en cm
  ```

  `lado` es el nombre de la variable numérica.
- OCTAVE tiene asignados valores por defecto a ciertas variables: **i**, **j**, **pi**, ... al igual que reserva ciertos identificadores concretos para diversas órdenes, como las destinadas a la creación de bucles, por ejemplo. Estos últimos se denominan *palabras reservadas.*
  No se deben utilizar las *palabras reservadas* del programa, (aunque **i**, **j** no lo sean, no los emplearemos para acciones distintas a aquellas para las que fueron asignados).

  Para obtener una lista de palabras reservadas, teclea:

  iskeyword()

  Observa las primeras líneas de la respuesta:

  The following identifiers are keywords, and may not be used as variable or functions names:
  ...

Ejercicio 1.3
De los siguientes nombres de variables indica cuáles son incorrectos y por qué:

```
altura, resultado-1, 1_resultado, resultado.1, resultado_1, resultado*,
resultado total,  _resultTotal, Alt\&ura, matriz_3*2, matriz_3x2, ancho=,
x*y, x,  Pi
```

1.5. Algunas funciones predefinidas

- **abs** calcula el valor absoluto de su argumento,
- **rats** aproxima su argumento por una fracción,
- **sqrt** calcula la raíz cuadrada de su argumento,
- **exp** calcula la exponencial (con base **e**) de su argumento,
- **log** calcula el logaritmo neperiano (su argumento debe ser un número real no negativo) (*idem* para el logaritmo en base 10, **log10** o en cualquier base positiva **a**, **loga**),
- **sin**, **cos**, **tan**, **sec**, **csc**, **cot** calculan las razones trigonométricas de sus argumentos medidos en radianes,
- **asin**, **acos**, **atan**, **asec**, **acsc**, **acot** son las funciones trigonométricas inversas de las anteriores, respondiendo con un ángulo en el intervalo $(-\pi/2, \pi/2)$.

Nota 1.2
En el nombre de cada una de las funciones anteriores sólo se emplean letras ***minúsculas****.*

1.6. Representaciones gráficas

Cuando se realiza un gráfico OCTAVE abre una ventana (`figure`) que denomina `figure1`.

Una sencilla forma de obtener un gráfico es dibujar un conjunto de puntos en el plano que el programa unirá con segmentos de recta. Se forma así una línea quebrada (poligonal). Para introducir esos conjuntos de puntos se utilizan vectores.

1.6.1 Vectores

Para generar el vector fila `v` $= (3, 6, 2, 4, 1, 5)$ basta teclear:

```
v = [3 6 2 4 1 5]
```

En un vector columna cada componente debe separarse de la siguiente por un punto y coma (o bien pulsando la tecla `Intro`). Por ejemplo, para obtener el vector columna `w` $= \begin{pmatrix} 3 \\ 6 \\ 9 \\ 4 \\ 2 \end{pmatrix}$ se debe introducir la orden:

```
w = [3; 6; 9; 4; 2]
```

Los vectores columna también se pueden obtener como traspuestos de los correspondientes vectores fila. El símbolo que identifica las trasposición de vectores es el apóstrofo[2] (').

[2]El apóstrofo (') se encuentra en la tecla del símbolo de cierre de interrogación (?)

Ejercicio 1.4
Qué hacen las siguientes órdenes?

```
v = [3 6 2 7 1 5];
v(3)
v([2, 4])
zeros(1, 5);
linspace(2, 10, 3)
```

Escribe un comentario para cada una de ellas que explique su acción.

Operador dos puntos

El operador dos puntos (`:`) permite obtener vectores en los que se proporciona el elemento inicial, el salto entre componentes y el valor último o, en su defecto, el valor que determina la última componente del vector. Observa las dos siguientes órdenes:

```
5:3:17
5:3:18
```

Si las órdenes son distintas, ¿por qué el resultado es el mismo?

¿Qué hace la orden siguiente?:

```
1/2:7/2
```

Ejercicio 1.5

- *Obtén el vector v = (3, 1'5, 0, -1'5, -3, -4'5) usando:*
  - *el operador dos puntos introducido en el ejercicio anterior,*
  - *el comando* **linspace.**
- *Extrae las componentes, del vector v = (3, 1'5, 0, -1'5, -3, -4'5), situadas en los lugares impares.*

1.6.2 Operaciones con vectores elemento a elemento

Además de las operaciones habituales de suma de vectores de la misma dimensión y producto por escalares, OCTAVE permite operar con vectores elemento a elemento. Esto permite, por ejemplo, elevar cada componente de un vector a un número.

Para operar elemento a elemento se deben emplear los operadores habituales precedidos de un punto (`.*`, `./`, `.^`). Así, la respuesta a la orden

```
v = [-1,  3,  5];  v.^3
```

es

```
ans =

   -1    27   125
```

y, si tecleamos:

```
v = [-4, 3, 10];  w = [-1, 3, 4] ;   v./w
```

entonces obtenemos como respuesta:

ans =

```
    4.0000    1.0000    2.5000
```

Aunque parezca una operación incorrecta, en OCTAVE se pueden sumar un escalar y un vector y la respuesta es un vector en el que cada componente es el resultado de sumar el escalar a cada componente del vector de partida. Por ejemplo, son equivalentes las órdenes de las dos siguientes líneas:

```
v = [2, 3, -1, 0];  5 + v
```

```
v = [2, 3, -1, 0];  [5, 5, 5, 5] + v
```

Por otra parte,

- El comando **sum** calcula la suma de las componentes de un vector.
- El comando **max** calcula el valor máximo entre las componentes de un vector.
- El comando **min** calcula el valor mínimo entre las componentes de un vector.
- El comando **length** calcula el número de componentes de un vector.

Ejercicio 1.6
Dado `v` $= (4,\ -\sqrt{2},\ 2'5)$ *calcula el vector cuyas componentes son las del vector* `v` *elevadas al cuadrado. Hazlo de dos formas distintas.*

1.6.3 Comando plot

- **plot** dibuja puntos del plano generados al utilizar las componentes del primer vector introducido *versus* las componentes del segundo vector. Por ejemplo, para dibujar la poligonal que une los puntos $(1, 6)$, $(2, -1)$, $(3, 7)$, $(-4, 9)$, $(5, -2)$, la orden es:

  ```
  x = [1 2 3 -4 5]; y = [6 -1 7 9 -2]; plot(x, y)
  ```

- Los vectores que son argumentos de **plot** deben ser de la misma dimensión (es decir, deben tener el mismo número de componentes).
- Algunas opciones gráficas que añaden claridad a una gráfica son:
  - **title**(′Para titular′)
  - **xlabel**(′esta es la etiqueta del eje de abscisas′)
  - **ylabel**(′esta es la etiqueta del eje de ordenadas′)
  - **text**(a, b, ′este es el punto (a,b)′)
  - **legend**(′este texto identifica una linea′)

  Nota: Hemos prescindido de las tildes en los comentarios porque, aunque en MatLab no plantean problemas, Octave no las admite.

- Aunque cada vez que se genera un gráfico éste substituye a cualquier otro que se hubiese creado con anterioridad esto no significa que no se puedan dibujar conjuntamente varias gráficas. Existen varias formas de hacerlo, una de ellas es incluir en la orden **plot** tantos pares de vectores como gráficos se desea representar conjuntamente. Por ejemplo, la siguiente línea produce el dibujo conjunto de las curvas $y = cos(x)$ e $y = cos(3x)$ en el intervalo $[0, 3\pi]$:

  ```
  x = 0:pi/100:3*pi;  y = cos(x);  z = cos(3*x);   plot(x, y, x, z)
  ```

  Otra forma alternativa de dibujar simultáneamente varias gráficas la podemos conocer si analizamos con atención las órdenes del siguiente ejercicio:

Ejercicio 1.7
Incluye un comentario a la derecha de cada una de las siguientes órdenes que explique lo que hace cada una.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x, y)
hold on
z = cos(x);
plot(x, z, ' g-.')
hold off
```

Otra cuestión distinta es que necesitemos tener dos (o más) ventanas de gráficos distintas. Como ya sabemos el programa al ejecutar un gráfico etiqueta la ventana en la que aparece con el nombre `Figure 1`. Si deseamos mantener ese gráfico en esa ventana, y poder generar otro en una ventana distinta, deberemos teclear, por ejemplo,

```
figure(2)
```

que generará la ventana (en blanco) etiquetada con el nombre `Figure 2`. Ahora, las órdenes que se introduzcan tras la línea anterior crearán el gráfico en esa segunda ventana sin alterar el que ya hubiese, en su caso, en otra ventana. En general la orden

figure(n)

crea la ventana de gráficos `Figure n`.

Cada vez que OCTAVE genera un gráfico utiliza ciertos criterios —opciones por defecto— sobre la apariencia del mismo, como son, por ejemplo, los sucesivos colores o trazados que emplea en las curvas que dibuja. Veamos cómo modificar algunas de esas opciones. Por ejemplo, al dibujar una circunferencia con las instrucciones:

```
t = 0:pi/180:2*pi; x = cos(t); y = sin(t); plot(x,y)
```

la apariencia que se obtiene es la de una elipse. No obstante, se puede modificar esto para que su apariencia sea la que esperamos. Esto se consigue con la orden **axis equal**:

axis equal

axis equal obliga al programa a emplear la misma escala en ambos ejes, de ahí que ahora la circunferencia se vea circular y no elíptica como antes.

Se puede modificar el rectángulo en el que se dibuja la gráfica variando los valores que OCTAVE genera, de forma automática, para los ejes cartesianos. Así, si deseamos representar una curva en el rectángulo `[a, b]`×`[c, d]`, fijamos ese rango en cada uno de los ejes con la orden:

axis`([a b c d])`

Compruébalo cambiando el rango de los ejes en el ejemplo anterior.

Por otra parte,

`v =` **axis**

almacena los valores mínimo y máximo de cada eje en el vector `v`, es decir,

`v = [xmin, xmax, ymin, ymax]`.

Recordemos que se puede conseguir más información sobre **axis** con:

help axis

Como se ha comentado previamente, se puede elegir el color y trazo de la curva a dibujar.

Si se desea establecer un color o trazado concreto, se debe introducir como una opción en el comando **plot** (debe ir entre apóstrofes), como ya has podido comprobar en el ejercicio 1.7. Los posibles colores y algunos de los posibles trazos se relacionan a continuación, junto con diferentes marcas que permiten resaltar detalles de un gráfico:

Color		Marcas	
`b`	azul (***b****lue*, opción por defecto)		
`c`	azul celeste (***c****yan*)	`d`	**d**iamante
`g`	verde (***g****reen*)	`o`	círculo
`k`	negro (*blac****k***)	`s`	cuadrado (***s****quare*)
`m`	**m**agenta	`p`	estrella de cinco puntas (***p****enta*)
`r`	rojo (***r****ed*)	`h`	estrella de seis puntas (***h****exa*)
`w`	blanco (***w****hite*)		
`y`	amarillo (***y****ellow*)	`*`, `.`, `+`, ×	
Trazo		`<`	triángulo apuntando a la izquierda
		`>`	triángulo apuntando a la derecha
`-`	línea continua (opción por defecto)	`^`	triángulo apuntando arriba
`--`	discontinuo (línea y línea)	`v`	triángulo apuntando abajo
`.-`	discontinuo (línea y punto)		
`:`	discontinuo (línea de puntos)		

Si deseamos introducir una cuadrícula debemos teclear **grid** (o **grid on**) tras haber escrito el código del gráfico (el comando que desactiva una cuadrícula es **grid off**).

Ejercicio 1.8
Escribe la orden que dibuja el punto del plano de coordenadas (-4, 9) en color rojo y marcado con un masterisco.

1.7. Los archivos .m

Cada una de las prácticas a realizar durante el curso incluye un apartado con ejercicios que son evaluables. Esos ejercicios se entregarán escritos en un determinado tipo de archivos: los archivos con extensión `.m`

1.7.1. Archivos de instrucciones

Los archivos `.m` son archivos de texto ASCII (*scripts*) que se pueden crear desde cualquier editor de texto, aunque OCTAVE posee su propio editor que es el que utilizaremos nosotros. Los archivos `.m` permiten almacenar conjuntos de instrucciones que vayan a utilizarse en más de una ocasión y el programa las ejecuta secuencialmente.

Para crear un archivo `.m` abrimos el editor de textos tecleando **edit** en la ventana de comandos. Aparecerá la ventana del editor en blanco, con una pestaña etiquetada `<sin nombre>`. Vamos a crear un archivo que calcule el volumen de un cilindro circular recto, conocidos su altura y el radio de la base. Lo grabaremos con el nombre `volumen_cilindro`. Para ello, en el menú del editor escogemos la opción Archivo y a continuación Guardar archivo como ....

Observa que, en la ventana flotante que se abre, la extensión por defecto es `.m`, por lo que **SOLO INTRODUCIREMOS EL NOMBRE DEL ARCHIVO, SIN SU EXTENSIÓN**.

Ahora ya podemos teclear en la hoja del editor las instrucciones:

```
% Este fichero contiene la formula del volumen de un cilindro circular recto
% de radio 15 y altura 3
radio_base = 15;
altura  = 3
volumen = pi*radio_base^2*altura
```

Ahora podemos ejecutarlo tecleando en la ventana de comandos su nombre SIN LA EXTENSIÓN.

```
>>  volumen_cilindro
```

También se puede ejecutar activando, con el puntero del ratón, la opción `Ejecutar` del menú del editor.

Nota 1.3
Observa que en la ventana de comandos no aparece la línea que define el radio de la base (porque finaliza con un punto y coma) pero sí se ejecuta ya que el cálculo del volumen sí se realiza.

Ejercicio 1.9
Teclea en la ventana de comandos:

help `volumen_cilindro`

¿Qué función desempeñan las líneas de comentarios introducidas al principio en el archivo?

1.7.2. Archivos de función

Aunque ya hemos utilizado funciones predefinidas en OCTAVE (`sin`, `tan`, `log`, $\cdots$) en este apartado veremos una forma de definir nuestras propias funciones, usando *archivos de función*.

Los archivos de función son archivos `.m` que se caracterizan porque la primera línea ejecutable (que no es un comentario) comienza con la palabra reservada **function**.

El nombre que con el que grabaremos el archivo será el nombre de la función.

Veamos un par de ejemplos:

```
% Este archivo implementa la funcion f dada por f(x)=(3x + 4)sen(2x+pi/2)
function  y = f(x)
y = (3*x + 4)*sin(2*x+pi/2);
end   % tambien es correcto escribir endfunction
```

Para utilizar esta nueva función, calculando por ejemplo $f(0)$, primero grabamos el archivo con el nombre `f` (su extensión será `.m`) y después tecleamos en la ventana de comandos:

```
f(0)
```

Calcula a mano el valor para comprobar que el resultado de la pantalla es el correcto.

El siguiente es un ejemplo de una función de dos variables b y h. Es decir, la función tiene dos *parámetros de entrada*:

```
% Este archivo implementa la fórmula del área de un triángulo
% como función de la base y de la altura;
% la función tiene pues dos argumentos de entrada: b  h
%
function  y = area(b, h)
y = b*h/2;    % b es la base y h es la altura
endfunction
```

Una vez tecleadas las líneas anteriores, grabamos el archivo con el nombre `area` (su extensión será `m)` y lo invocamos desde la ventana de comandos para calcular el área de un triángulo de altura `h = 2` y con base `b = 10` tecleando:

```
area(10, 2)
```

1.8. Algo de programación en Octave

1.8.1. Operadores lógicos y relacionales

OCTAVE reconoce expresiones lógicas asociándoles el valor **1** si son verdaderas (**true**) y el valor **0** si son falsas (**false**). Para ello dispone de seis operadores relacionales (todos con el mismo orden de prioridad, evaluándose de izquierda a derecha) y de tres operadores lógicos:

Operador relacional	Significado	Operador lógico	Significado
$<$	menor que	&	y (**and**)
$>$	mayor que		
$<=$	menor o igual que	|	o (**or**)
$>=$	mayor o igual que		
$==$	igual a	$\sim$	no (**not**)
$\sim=$ $!=$	distinto a		

El símbolo $\sim$ se consigue con Alt+1+2+6 y el símbolo | se consigue con AltGr+1.

Nota 1.4

- *Además de los anteriores existen los operadores:*

  . &&, *también llamado* and breve *porque si el primer operando es falso ya no se evaluará el segundo pues el resultado final de la comparación será* **false**.

  . || —*también llamado* or breve— *si el primer operando tiene valor* **true** *ya no tiene en cuenta el segundo pues el resultado será* **true**.

  Tanto este operando como el anterior evitan operaciones innecesarias.

Veamos un ejemplo de aplicación de estos operadores. Teclea las siguientes instrucciones:

```
>> v = -1:1:5;
```

```
>> v >= 2
```

Verás que la respuesta es:

ans =

0 0 0 1 1 1 1

y si tecleamos

```
>>  (v >= 2)&(v < 4)
```

obtenemos

ans =

0 0 0 1 1 0 0

Es decir, donde se cumple la condición pedida aparece un 1 y donde no se cumple la condición aparece un 0.

Los operadores relacionales pueden simplificar considerablemente la escritura de un algoritmo, como se puede observar en la instrucción

```
y = v( v.^2 >= 3)
```

que proporciona el vector llamado `y` cuyas componentes se obtienen a partir de las componentes de otro vector llamado `v`, escogiendo sólo aquellas tales que su cuadrado es mayor o igual que `3`. Compruébalo con el vector $v = (-1,\ 5\,,1/2,\ 3,\ -2,\ 0,\ 1)$.

1.8.2. Cómo dibujar funciones definidas a trozos

Supongamos que deseamos dibujar en el intervalo $[-2, 3]$ la función f dada por

$$f(x) = \begin{cases} 1 - 2x & \text{si } x \leq -1 \\ (x+1)^3 + 2x & \text{si } x > -1, \end{cases}$$

bastará escribir las siguientes órdenes, usando las operaciones entre vectores elemento a elemento:

```
>>  x = linspace(-2, 3, 1000);
>>  y = (1-2*x).*(x <= -1)+((x + 1).^3 + 2*x).*(x > -1);
>>  plot(x, y, '.');
>>  title('funcion definida a trozos')
```

¡Observa cuándo son imprescindibles los puntos que preceden a las operaciones elemento a elemento y cuándo no!

Observa también que hemos dibujado el trazado con puntos. Es conveniente hacerlo así (o con asteriscos u otras marcas), para que se reflejen las discontinuidades (si las hubiere).

1.8.3. Estructuras de repetición (bucles)

***Estructura* for**:

```
for k = 1:n
 sentencias
endfor
```

Nota 1.5

- *Una forma más general de escribir el contador de un bucle es:* `k = n1:incr:n2,` *donde* `incr` *es el incremento del contador, que puede ser negativo.*
- *No son aconsejables instrucciones del tipo* `k = k+1` *y no se deben utilizar ni* `i` *ni* `j` *para denotar el contador.*
- *Los bucles anidados se ejecutan desde los más internos hacia los más externos.*
- *Si se introducen directamente en la ventana de comandos, sin utilizar el editor de texto, las órdenes se ejecutan sólo tras teclear el último* **endfor** *que completa el bucle.*

Como ejemplo de las estructuras de repetición incluimos un archivo que implementa el cálculo de los cuadrados de los diez primeros números naturales

```
% Archivo que, mediante un bucle, calcula los cuadrados de los diez
% primeros numeros naturales
 for  k = 1:10
    k^2
 endfor
```

Ejercicio 1.10
Genera un archivo que calcule el cubo de los números naturales pares, comprendidos entre 5 y 21. Hazlo usando un bucle. ¿Sabrías hacerlo vectorizando la orden (sin emplear un bucle)?

1.8.4. Estructuras de selección (decisión)

Forma simple:

```
if condición
 sentencias
endif
```

Forma general:

```
 if condición
  sentencias
 else
  sentencias alternativas
 endif
```

o bien, en una sola línea:

if *condición* `sentencias,` **else** `sentencias alternativas,` **endif**

Nota 1.6

- *¡Las comas son imprescindibles!*
- *también sirve* **end** *pero* **endif** *hace más claro el código*

***Forma anidada con* elseif**:

if *condición 1*

  grupo 1 de sentencias

elseif *condición 2*

  grupo 2 de sentencias

elseif *condición 3*

  grupo 3 de sentencias

  ⋮

else

  sentencias alternativas

endif

```
% Archivo que, mediante una sentencia de decision, calcula los factoriales
% de los numeros naturales pares menores o iguales que 8.
% Calcula el cubo de los numeros naturales impares menores o iguales que 9
% y muestra un mensaje de advertencia si el numero
% introducido es mayor que 9

function      factorialesYcubos(n)

 if rem(n,2) ==  0   &&   n  <= 8
 factorial(n)
 elseif rem(n,2) ~= 0  &&    n <= 9
   n^3
 else disp('debes introducir un natural menor que 10')
 endif     % tambien sirve end

endfunction     % tambien sirve end
```

A la vista del programa anterior, ¿para qué se usa `rem`?

Ejercicio 1.11
En este ejercicio, a partir de la ecuación genérica de segundo grado, $ax^2+bx+c=0$, *deberás escribir un archivo de función con tres parámetros de entrada que son los coeficientes de la ecuación de segundo grado.*
El archivo deberá realizar ***uno de los apartados siguientes****:*

1. *Debe comprobar si* $a = 0$ *y en caso afirmativo resolverá la ecuación. Además mostrará en pantalla un mensaje que indique que la ecuación es de grado 1. En caso contrario, es decir, si* $a \neq 0$, *deberá indicarnos si las raíces son reales o complejas.*
2. *Debe comprobar si* $a = 0$ *y en caso afirmativo producirá un mensaje de texto indicando que la ecuación no es de segundo grado. En caso contrario, es decir, si* $a \neq 0$, *nos indicará si las raíces son reales o complejas y en caso de ser reales deberá indicar si son distintas o es una raíz doble.*
3. *Debe comprobar si* $a = 0$ *y en caso afirmativo debe enviar un mensaje de error indicando que se deben introducir otros datos. En caso contrario, es decir, si* $a \neq 0$, *calculará las raíces de la ecuación.*

1.9. Algunos comandos más

Esta sección reune algunos comandos que pueden ser útiles en la elaboración de archivos `.m`.

- **disp**

  su nombre proviene de *display* y muestra en pantalla el texto entre apóstrofos que figure como argumento:

  disp(*'Este texto se muestra en pantalla'*)

  Así pues,

  disp(' ')

  genera una línea en blanco.

  disp también permite mostrar mensajes en los que se combinan cadenas de caracteres con resultados de cálculos (una vez convertidos estos a cadenas de caracteres). En nuestro caso esto será útil, entre otras, en aquellas ocasiones en las que deseemos indicar por ejemplo el número de iteraciones empleadas por un método iterativo para obtener una aproximación:

  disp(['el algoritmo ha empleado ', **num2str**(iter), ' iteraciones '])

  Nota: El comando **num2str** que aparece en la línea anterior se verá con detalle más adelante.

- **error**

  muestra en pantalla el mensaje de error que figura como argumento situado entre apóstrofos. Detiene la ejecución y devuelve el control al usuario a través del teclado. Su sintaxis genérica es:

  error(*'texto del mensaje'*)

- **factor**(`argumento`)

  factoriza `argumento`.

- **factorial**`(n)`

  calcula el factorial del número `n`. No se debe confundir con el comando **factor**. (Observa la diferencia entre **factorial**(6) y **factor**(6)).

- **fix**`(x)`

  redondea hacia cero el valor de `x`. Simplemente elimina la parte fraccionaria.

- **floor**`(x)`

  redondea al número entero más próximo a `x` hacia menos infinito.

- **fprintf**

  permite controlar la respuesta combinando texto y valores de variables en una misma línea, ajustando el formato numérico. El cursor no cambia de línea cuando se muestra la respuesta en pantalla. La forma más general de este comando es:

  fprintf(*'cadena'*, `lista de variables`)

  donde *cadena* contiene un mensaje con especificadores.

  Los especificadores —`%e`, `%f`, `%g`— controlan el formato de las variables de la lista que aparecen combinadas con el texto en *cadena*.(`%g` mezcla formato decimal y exponencial, decidiendo el programa qué formato emplea). Algunos ejemplos de estos especificadores son:

  `%8.3f` indica que la variable debe aparecer en formato en coma flotante con al menos 8 cifras (entre la parte entera y la decimal) y, tras el punto decimal, habrá 3 dígitos.

  En particular, `%2.0f` indica que la respuesta será un número entero de, al menos, dos dígitos.

  Por ejemplo, la respuesta de la orden:

  fprintf`('cantidad: %5.2f', 5.6789)`

  es:

  ```
  cantidad: 5.68
  ```

  mientras que la respuesta de:

  fprintf`('%5.2e',345.6789)`

  es: `3.46e+002`

- **fprintf**(*'\ n'*)

  fuerza un salto de línea.

- **input**

  permite la introducción de datos por pantalla. Su sintaxis genérica es

  `variable =` **input**(*'texto del mensaje'*)

  que escribe en pantalla el texto del mensaje y espera el valor de la variable que debe teclear el usuario en la ventana de comandos. Si la variable que se va a introducir es de tipo carácter, debe ir entre apóstrofos o se debe indicar como segundo argumento de entrada **s** (proviene de ***s****tring*):

  `variable =` **input**(*'texto del mensaje'*, *'s'*)

- **menu**

  interactúa con el usuario a través de la pantalla y el teclado, proporcionando un menú con distintas opciones para que el usuario escoja una de ellas

  `opcion = menu('mensaje', 'opcion_1', 'opcion_2',...)`

  y almacena en la variable llamada `opcion` el valor `n` si el usuario ha escogido `opcion_n`. Por ejemplo,

  `menu('elija el algoritmo de aproximacion', 'biseccion', 'Newton-Raphson')`

- **pause**

  detiene la ejecución del algoritmo hasta que el usuario pulsa una tecla.

- **pause(n)** interrumpe la ejecución durante `n` segundos y luego continúa.

- **rand**

  genera un número aleatorio en el intervalo abierto (0, 1).

- **round(x)**

  redondea el valor de `x` al entero más próximo.

1.10. Cálculo simbólico

1.10.1. Variables simbólicas

Además de las variables numéricas existen las variables simbólicas que permiten calcular límites, derivadas, integrales, etc., como se hace habitualmente en las clases de matemáticas.

Como ya se ha comentado en la primera parte de esta práctica, para poder realizar estas operaciones, habituales en un curso de Cálculo, es necesario tener instalada la librería `symbolic`. Como este es un proceso ya realizado, cada vez que queramos trabajar en modo simbólico bastará cargar esa librería tecleando en la ventana de comandos la orden

```
pkg load symbolic
```

Esta orden permanece activa mientras no cerremos la sesión de trabajo, por lo que si lo hacemos al inicio de cada clase, no tendremos que preocuparnos más tarde por posibles olvidos.

Para trabajar en modo simbólico es necesario definir *variables simbólicas* y para hacer esto utilizaremos el comando **syms**. Veamos algunos ejemplos de su uso:

```
syms   x   % define la variable simbolica x

 f = 3*x + 5   % ahora hemos definido la expresion simbolica f

syms  a  b  c    % define como simbolicas las variables a, b, c.

expresion = a^3 + b^2 + c
```

Las dos últimas instrucciones definen, a partir de las variables simbólicas `a`, `b` y `c`, la que hemos llamado `expresion` y que contiene la expresión simbólica $a^3 + b^2 + c$.

syms es la forma abreviada de uso del comando **sym** cuya sintaxis, para definir la variable simbólica `x` es:

`x = sym('x')`

Al manejar expresiones simbólicas obtenemos resultados simbólicos.

1.10.2. Constantes simbólicas

En OCTAVE se pueden definir constantes simbólicas de forma sencilla usando el comando **sym**. Por ejemplo `a = sym('1/3')` define la constante simbólica 1/3.

Observa la diferencia en la respuesta que se obtiene, en la ventana de comandos, al teclear 1/3 o al teclear la orden anterior. Observa también la diferencia en las clases asignadas en el `espacio de trabajo`.

1.10.3. Cambio de tipo de variable

Las constantes simbólicas se convierten en valores numéricos utilizando la orden **double**. Por ejemplo,

```
c = sym('2/5')     % c es simbolica
d = double(c)      % d es numerica
```

Observa la diferencia en la respuesta que se obtiene en la ventana de comandos para `c` y para `d`. Observa también la clase asociada a `c` y a `d` en el `espacio de trabajo`.

Otra forma de obtener un valor aproximado de la variable símbólica `c` es utilizar el comando **vpa**.

vpa son las iniciales de ***v**ariable **p**recision **a**rithmetic*. Este comando se emplea para calcular aproximaciones con un número predeterminado de dígitos. La sintaxis genérica es:

vpa`(valor, d)`

donde `valor` es la cantidad de la que se busca la aproximación y `d` representa el número de dígitos de precisión –dígitos significativos– que se desea fijar. Por ejemplo, la respuesta a la orden

vpa`(1/3, 5)`

es `0.33333` y la respuesta a la orden

vpa`(1/300, 5)`

es `0.0033333`. De nuevo, observa en el `espacio de trabajo` la clase asignada a esta respuesta y compárala con la clase asignada a la variable `d`.

OCTAVE también maneja cadenas de texto que deben ir entre apóstrofes o comillas. Mientras el apóstrofe se usa también para trasponer vectores o matrices, las comillas no tienen ninguna otra utilidad. Si el texto va entrecomillado, debemos duplicar las comillas. Observa la diferencia en la respuesta a las dos órdenes siguientes en las que interviene el comando **disp** que muestra en pantalla el texto introducido como argumento:

disp`(" 'ole' ")`

disp`(' "ole" ')`

Para convertir valores numéricos en cadenas de caracteres se usa la orden **num2str**. Por ejemplo, la orden **num2str**`(d)` convierte la variable numérica `d` definida antes en un *string*.

A continuación se introducen algunos otros comandos de uso habitual al trabajar de forma simbólica.

1.10.4. Comandos que actúan sobre expresiones simbólicas

- **clear all** (o simplemente **clear**)

  borra todas las variables del `espacio de trabajo`.

  ¡Observa la diferencia con el comando **clc**, visto en la sección 1.2!

- **clear** `c  % esta orden borra la variable c`

- **findsym**`(expr)`

  halla las variables simbólicas en la expresión simbólica `expr`. Por ejemplo, teclea las tres siguientes y observa la respuesta:

  syms `alpha  beta`

  `expresion = 1/4^3 + alpha^2 + beta`

  findsym`(expresion)`

- **solve**

  La capacidad que OCTAVE posee para trabajar con variables simbólicas permite manejar ecuaciones de manera simbólica y resolverlas después reemplazando cada una de las distintas variables que aparezcan en la ecuación por un valor, análogamente a como se hace habitualmente en una clase de matemáticas. Por ejemplo, para resolver la ecuación $3x + 15 = 0$ basta teclear:

  syms `x,` **solve**`(3*x + 15 == 0, x)`

  o abreviadamente —no hay ambigüedad sobre cuál es la incógnita—,

  syms `x,` **solve**`(3*x + 15 == 0)`

  Para obtener ayuda sobre los comandos que están en la librería `symbolic`, debes escribir:

  help @sym/`nombreComandoBuscado`

  por ejemplo,

  help @sym/solve

  proporciona ayuda sobre el comando **solve**.

- **subs**

  permite calcular un valor concreto para la(s) variable(s) simbólica(s) que se encuentre(n) en una expresión simbólica. Por ejemplo, la orden

  syms `x;  pol = 3*x^2 - 5*x + 4;` **subs**`(pol, x, 1)`

  proporciona el valor de `pol` en $x = 1$. (La respuesta que debes obtener al teclear la orden anterior es 2). Por otra parte, la orden

```
syms  a   b; pol = a*x^2 - b*x + 4; subs(pol, {a, b}, {-1, 3})
```

permite la sustitución simultánea de las variables simbólicas `a` y `b` por los valores `-1` y `3`, respectivamente. Como puedes observar la sintaxis genérica de este comando, que será muy útil, es:

```
subs(expresión, valor_antiguo, valor_nuevo)
```

o bien,

```
subs(expresión,{valorAntiguo_1,..., valorAntiguo_n},{valorNuevo_1,..., valorNuevo_n})
```

Ejercicio 1.12
¿Qué se obtiene con la orden siguiente?

```
syms a b c x; solve(a*x^2 + b*x + c == 0, x)
```

¿y con la siguiente?

```
syms a b c x; solve(a*x^2 + b*x + c == 0, a)
```

¿y con la siguiente?

```
[x, y, z] = solve(2*x + 2*y - z == 0, x + y + z == 3, y - z == -2)
```

Ejercicio 1.13
Teclea las siguientes líneas y analiza los resultados para saber qué responde OCTAVE *a cada una de las siguientes órdenes:*

```
y = (x^2 - 1)/(2*(x + 4)^3); numden(y)
[num, den] = numden(y)
factor(y)
expand(den)
z=(x^2 - 1)/(x^2 + 2*x + 1); simplify(z)
```

1.10.5. Otras formas de obtener gráficas

En este apartado veremos otra forma de obtener gráficas, trabajando ahora en modo simbólico.

Existen "variaciones"del comando **plot** que funcionan en modo simbólico. Algunas de ellas son **plotyy**, **ezplot**, **fplot** y **subplot**, que permiten obtener gráficas en dos dimensiones. Nosotros nos centraremos sólo en el comando **ezplot** (para obtener información sobre cualquiera de los restantes comandos, como siempre, puedes utilizar la ayuda del programa).

Comando ezplot

Veamos cómo funciona este comando a través de un ejemplo:

```
syms x;   y = x^2 + sin(3*x) - 1/2;  ezplot(y, [-10, 10])
```

o simplemente

```
syms x;   ezplot(x^2 + sin(3*x) - 1/2, [-10, 10])
```

Nota 1.7

- *Con* **ezplot** *se puede dibujar una gráfica sin necesidad de introducir los límites de representación en el eje de abscisas. En ese caso el programa dibuja la gráfica en el intervalo* $[-2\pi, 2\pi]$.
- *Con* **ezplot** *el programa introduce, por defecto, un título a la gráfica.*
- *Si se desea varíar el rango de valores de la variable dependiente (eje de ordenadas), se puede hacer incluyendo una línea con la orden* **axis**, *como se ha explicado en la sección 1.6.*
- *Si se desea cambiar el color o el trazo de la gráfica podemos hacerlo usando el comando* **set**, *como se muestra en el siguiente ejemplo:*

```
>> syms x; gr = ezplot(x^2 - 4);
>>  hold on;
>>  set(gr, 'color', 'g', 'linestyle', '-.');
```

Ejercicios

Ejercicio 1.14
Calcula:

a) $2 * (3/4)$

b) $(2 * 3)/4$

c) ¿Son iguales los resultados de los apartados anteriores?

d) ¿Es correcta la siguiente orden?:

```
2*3/4
```

Justifica la respuesta.

Ejercicio 1.15
Calcula:

a) $(2^3)^4$

b) $2^{(3^4)}$

c) ¿Son iguales los resultados de los apartados anteriores?

d) ¿Es correcta la siguiente orden?:

```
2^3^4
```

Justifica la respuesta.

Ejercicio 1.16
Obtén la representación de $51'555$ *en formato* **long e** *y en formato* **bank**.

A la vista del resultado obtenido, ¿cuál será la representación en formato **bank** *de* $51'554$*? ¿Por qué?*

Ejercicio 1.17

a) ¿Qué calcula la instrucción siguiente?

```
x = [1/5, 3, 2, -7, 0.2];  v = [2, 4, 5];   x(v)
```

b) Crea una tabla formada por tres columnas. En la primera mostrará los valores de las componentes del vector `x`, *en la segunda el valor de sus raíces cuadradas y en la tercera sus raíces cúbicas en formato largo.*

Ejercicio 1.18
La letra del DNI se obtiene así: se calcula el resto de dividir el valor numérico del DNI entre 23. La correspondencia entre la letra a asignar y el valor del resto obtenido figura en el cuadro siguiente.

Resto	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*
Letra	*T*	*R*	*W*	*A*	*G*	*M*	*Y*	*F*	*P*	*D*	*X*	*B*	*N*	*J*	*Z*	*S*	*Q*	*V*

Resto	*18*	*19*	*20*	*21*	*22*
Letra	*H*	*L*	*C*	*K*	*E*

Obtén la fórmula que calcula el resto, ***usando solo comandos vistos en esta práctica****. Comprueba, con tu DNI y con el de algún compañero, que la fórmula es correcta.*

Ejercicio 1.19
Representa gráficamente el coseno de un ángulo α *de* 0 *a* 2π *a saltos de* 1/8*. Etiqueta los ejes y ponle un título. Dibuja, conjuntamente con el coseno, la recta horizontal que pasa por el valor más alto que toma la función* coseno *en el intervalo* $[0, 2\pi]$*. Es decir, la recta tangente a la gráfica del coseno en el punto donde el coseno alcanza su valor máximo. Dibuja esta recta en otro color y otro trazado. Crea una leyenda identificativa de cada una de las dos funciones dibujadas.*

Ejercicio 1.20

a) *El tramo más inclinado del tranvía del monte Washington en New Hampshire tiene un grado de inclinación (excepcional) del 37.1. A lo largo de este tramo, en sentido de subida, los asientos de la primera fila están 4.27 metros más arriba que los de la última fila. Qué distancia separa a la primera fila de la última?*

   Indicación*:* El grado de inclinación se define como la medida (**dada en porcentaje**) de la pendiente.

b) *Representa la situación usando vectores y el comando* **plot**.

Ejercicio 1.21
Las escalas de temperatura Fahrenheit *(F) y* Celsius *(C) se relacionan a través de una ecuación lineal. Sabiendo que el punto de congelación del agua es a 0°C o a 32°F, y que el punto de ebullición se alcanza a 100°C o a 212°F, se pide:*

a) *Halla la relación lineal entre ambas escalas.*

b) *Representa la función cuya expresión has hallado en el apartado anterior usando vectores y el comando* **plot**.

c) *Existe alguna temperatura en la cual un termómetro en grados* Celsius *proporcione la misma lectura que un termómetro en grados* Fahrenheit*? No se pide que resuelvas ninguna ecuación para responder a esta pregunta. Lo que debes hacer es interpretar gráficamente la situación.*

Ejercicio 1.22
Se desea medir la altura que alcanza un cultivo, sabiendo que dicha altura es función del tiempo. Las mediciones se hacen una vez a la semana y los resultados obtenidos se resumen en el siguiente cuadro:

Tiempo *(semanas)*	*1*	*2*	*3*	*4*	*5*
Altura *(cm)*	*5'2*	*6'6*	*7'3*	*8'6*	*10'7*

Obtén la gráfica que explique ese crecimiento. Dicha gráfica debe incluir un título apropiado y etiquetas en los ejes que indiquen cuál es la variable correspondiente a cada uno de ellos. Usa un trazado y color distinto para cada eje y para la función.

A la vista de la gráfica obtenida, ¿cuándo crees que crece más rápidamente el cultivo, entre la tercera y la cuarta semana o entre la cuarta y la quinta?

Ejercicio 1.23
Un cañón dispara un proyectil con velocidad inicial v_0 y ángulo de inclinación θ. La posición del proyectil en cada instante viene dada por las expresiones:

$$x = v_0 \cos(\theta)\, t \qquad\qquad y = v_0 \sin(\theta)\, t - \frac{1}{2} g\, t^2$$

donde $g = 9.81$ m/s^2 es la aceleración de la gravedad. Supongamos que la velocidad inicial es de 20 m/s.

1. *Dibuja —en un mismo gráfico— las trayectorias del proyectil en los primeros tres segundos cuando el ángulo de inclinación es de $\pi/3$, $\pi/4$ y $\pi/6$.*

2. *Calcula a qué distancia del cañón cae el proyectil en cada uno de los casos anteriores.*

Ejercicio 1.24
Una progresión geométrica es una sucesión en la que cada término se construye como el producto del anterior por una razón r: $a_n = ra_{n-1}$ ($n \in \mathbb{N}$). Sabemos que la suma de sus N primeros términos es:

$$S = a_0 \frac{r^{N+1} - 1}{r - 1}\,.$$

Construye una progresión geométrica a partir del primer término y la razón, y comprueba la exactitud de la expresión anterior para distintos valores de N.

Ejercicio 1.25
Una serie es, grosso modo, la suma de los infinitos términos de una sucesión; se dice que la serie es convergente si dicha suma es finita. Determina si las siguientes series son convergentes:

1. $\displaystyle\sum_{n=0}^{\infty} \frac{1}{2^n}$

2. $\displaystyle\sum_{n=0}^{\infty} \frac{1}{n^2}$

3. $\displaystyle\sum_{n=0}^{\infty} \frac{1}{n}$

4. $\displaystyle\sum_{n=0}^{\infty} \left(1 + \frac{1}{n}\right)^n$

Para ello, calcula las sumas de los N primeros términos para distintos valores (suficientemente grandes) de N.

Ejercicio 1.26
Genera un archivo de función tal que, usando estructuras de decisión, defina la función f dada por:

$$f(x) = \begin{cases} |x-1| & si \quad x \in (-\infty, 0) \\ x^2 + x + 1 & si \quad 0 \leq x \leq 1 \\ 3 + ln(x) & si \quad x > 1. \end{cases}$$

Calcula el valor en puntos de los diferentes subconjuntos de definición para comprobar que está bien definida.

Ejercicio 1.27
Elabora un archivo `.m` *que lea una fecha dd/mm/aaaa (día, mes y año), compruebe si es correcta y calcule el número de días transcurridos desde el 1 de enero de ese mismo año.*

Nota: son años bisiestos los múltiplos de 4, excepto los múltiplos de 100 que no lo sean de 400. Por ejemplo, el año 2000 fue bisiesto, mientras que el año 1000 de nuestra era no lo fue.

Ejercicio 1.28
Se conoce como ecuación de Van der Waals *para n moles de un gas a:*

$$(P + \frac{n^2\, a}{V^2})(V - nb) = nRT$$

donde P es la presión del fluido, medido en atmósferas, V es el volumen (en litros), T es la temperatura en grados Kelvin y R (atm.l/mol.K) es la constante universal de los gases. Los valores de a y b son característicos de cada gas.

Se llaman isotermas *las curvas que se obtienen al representar la ecuación de Van der Waals para una temperatura fija. Consideremos un gas que verifica la ecuación:*

$$(P + \frac{3}{V^2})(3V - 1) = 8T.$$

Escribe un archivo que represente una gráfica con las isotermas de ese gas (P en función de V), para distintos valores de T: 0'5, 0'7, 0'9, 1'1 y 1'3. Deberás ajustar los valores de los ejes para tener una visión aceptable de todas ellas.

Además, cada curva debe ir con un trazo diferenciado, con el texto que indique la isoterma que se ha representado, así como el título de la gráfica y la etiqueta de los ejes. Escoge una presión constante de valor α. Dibuja la recta $p = \alpha$ conjuntamente con las isotermas. ¿Hay algún volumen a esa presión? ¿Hay más de uno? Razona la respuesta.

Índice terminológico