

# MANUAL BÁSICO DE OCTAVE Y QTOCTAVE

## El programa OCTAVE

Octave o GNU Octave es un programa libre para realizar cálculos numéricos. Como indica su nombre es parte del proyecto GNU. Apoyado en una amplia comunidad de desarrolladores y usuarios, Octave cuenta con herramientas para la resolución de problemas de cálculo numérico lineales y no lineales: álgebra lineal, aproximación de raíces de ecuaciones, integración numérica, integración de ecuaciones diferenciales, etc., así como para la representación de gráficos en dos y tres dimensiones. Es fácilmente extensible y adaptable mediante funciones definidas por el usuario, bien utilizando el propio lenguaje de Octave o bien mediante módulos escritos en C++, C, Fortran u otros lenguajes.

El proyecto fue creado alrededor del año 1988 pero con la finalidad de ser utilizado en un curso de diseño de reactores químicos. Posteriormente en el año 1992 se decide extenderlo y comienza su desarrollo hasta nuestros días.

## Descarga e instalación de Octave

La descarga del programa puede hacerse desde la página oficial del proyecto.

<http://www.gnu.org/software/octave/>

En esta página podemos encontrar el programa octave para las diferentes plataformas: Windows, Linux, Mac, etc.. En este manual detallamos sólo como realizar la instalación para Windows ya que es la que vamos a utilizar durante el curso. Para la instalación en otras plataformas podéis seguir las instrucciones que están en la página web oficial del programa.

Para Windows, existe ya el instalador precompilado y puede descargarse directamente de la dirección:

<http://octave.sourceforge.net/>

El programa se instala como cualquier programa de Windows, por defecto el directorio donde se instala el programa es:

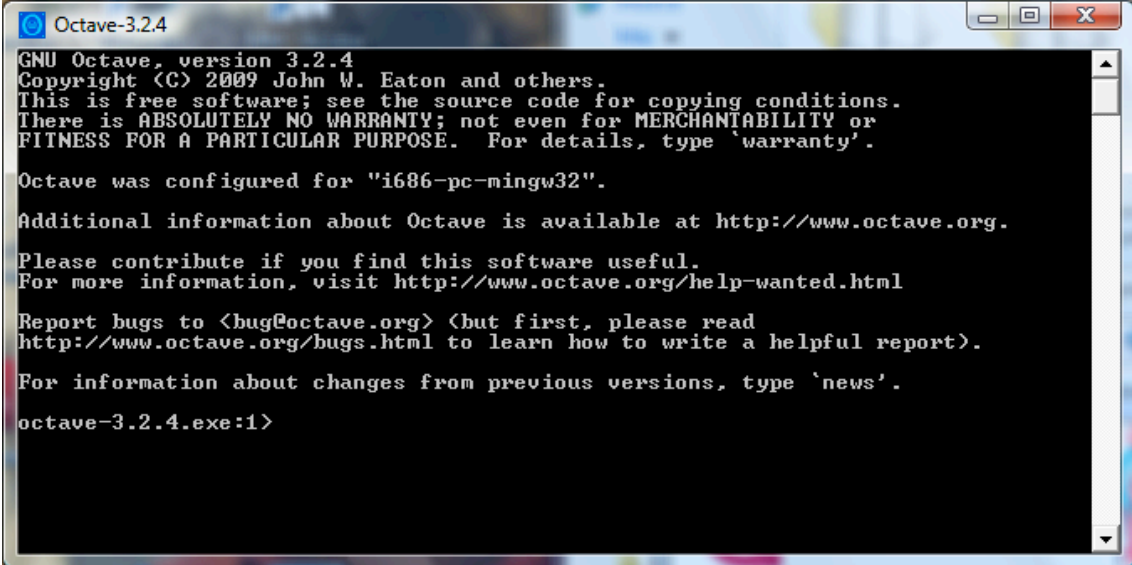
C:/Octave/3.2.4\_gcc-4.4.0

y el ejecutable se encuentra dentro de la carpeta

C:/Octave/3.2.4\_gcc-4.4.0/bin

Esta información es necesaria posteriormente en la instalación de la interface de usuarios.

El ultimo paso que realiza la instalación es la ejecución del programa, la ventana es la siguiente:



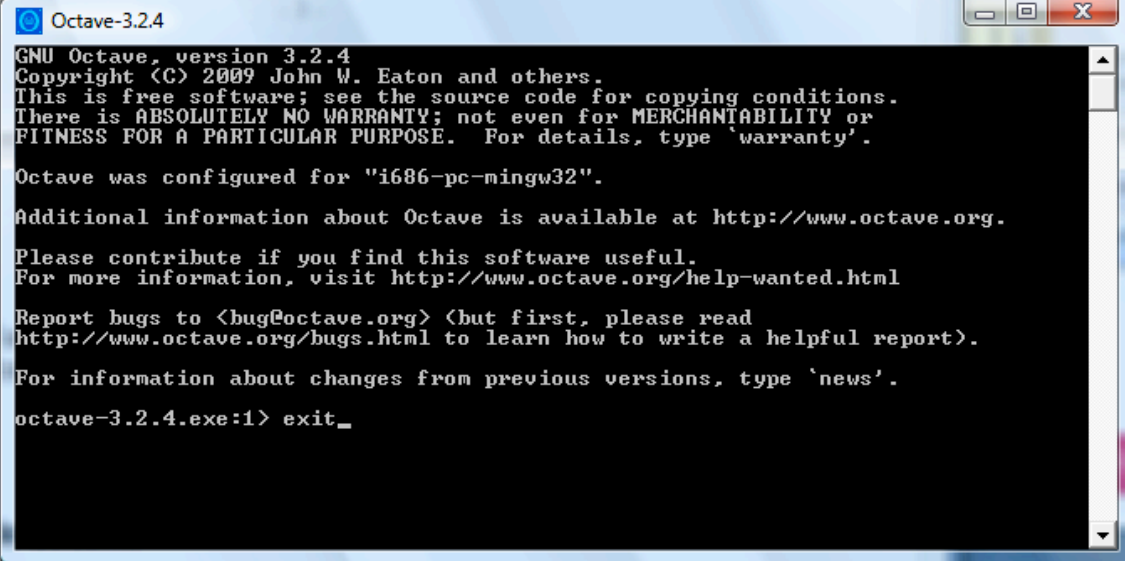
```
Octave-3.2.4
GNU Octave, version 3.2.4
Copyright (C) 2009 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type `warranty'.

Octave was configured for "i686-pc-mingw32".

Additional information about Octave is available at http://www.octave.org.
Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).
For information about changes from previous versions, type `news'.
octave-3.2.4.exe:1>
```

Como podemos ver la interfaz con el usuario no es demasiado agradable, es por ello que otros proyectos también de software libre, se han dedicado a mejorar esta interfaz, para hacer más fácil al usuario la forma de interactuar con Octave. Para salir de esta ventana basta teclear quit o exit.



```
Octave-3.2.4
GNU Octave, version 3.2.4
Copyright (C) 2009 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type `warranty'.

Octave was configured for "i686-pc-mingw32".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).

For information about changes from previous versions, type `news'.

octave-3.2.4.exe:1> exit_
```

## Interfaces de usuario

De entre las interfaces, nosotros utilizaremos QtOctave. Este programa está disponible para distintos sistemas operativos, cuenta con numerosos menús, botones y ventanas de diálogo que, aun encontrándose todavía en fase experimental, podemos decir que se trata de una herramienta que facilita al usuario la comunicación con Octave.

La página principal de este programa es:

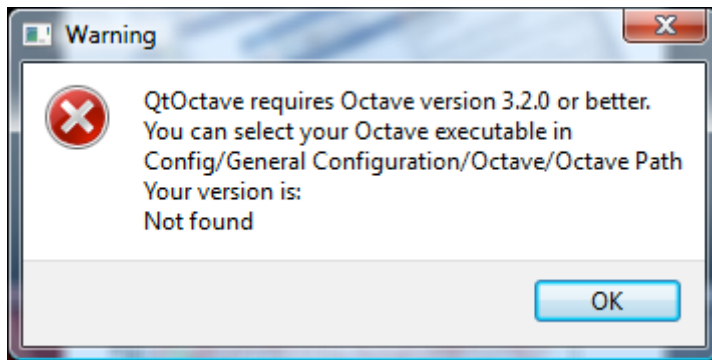
<http://qtoctave.wordpress.com/>

En concreto para Windows hay que descargar el fichero comprimido:

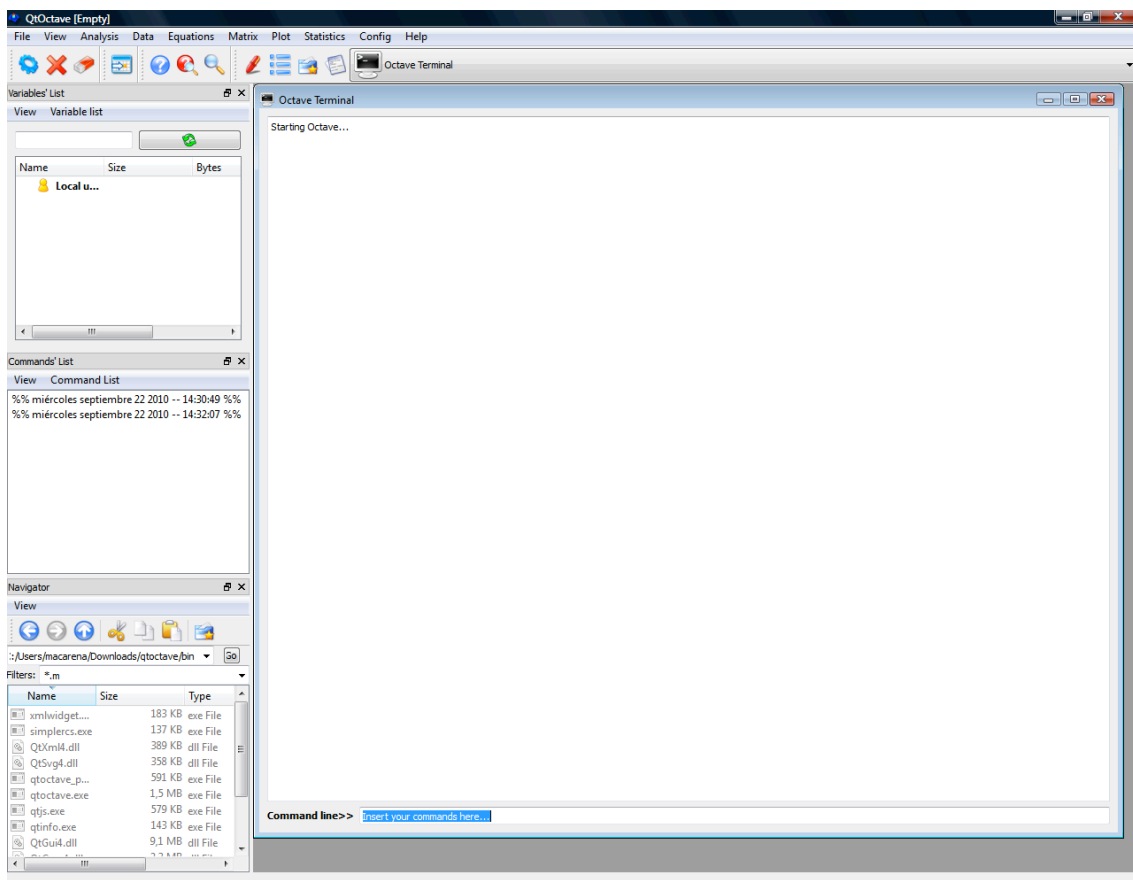
qtoctave-win32-0.9.1-3.zip

Esta interfaz no necesita instalación es suficiente con descomprimir en una carpeta y dentro de la carpeta bin encontramos el fichero ejecutable: qtoctave.exe

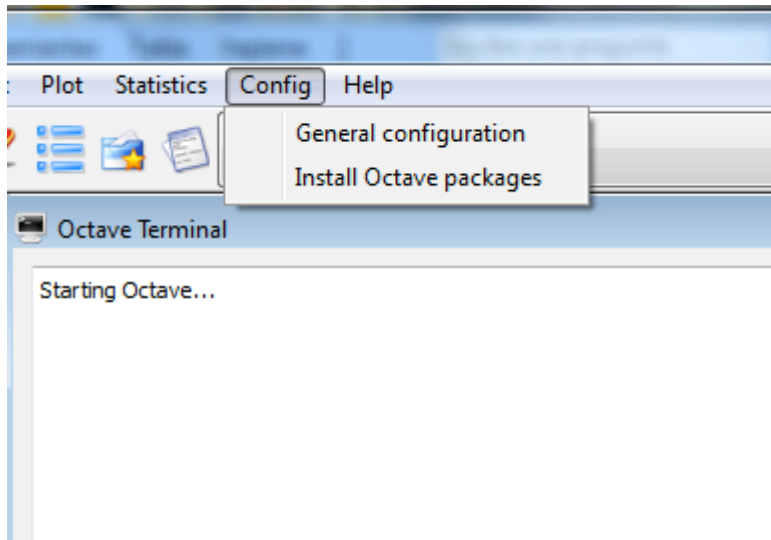
La primera vez que se ejecuta sale un mensaje de error como podéis ver en la figura:



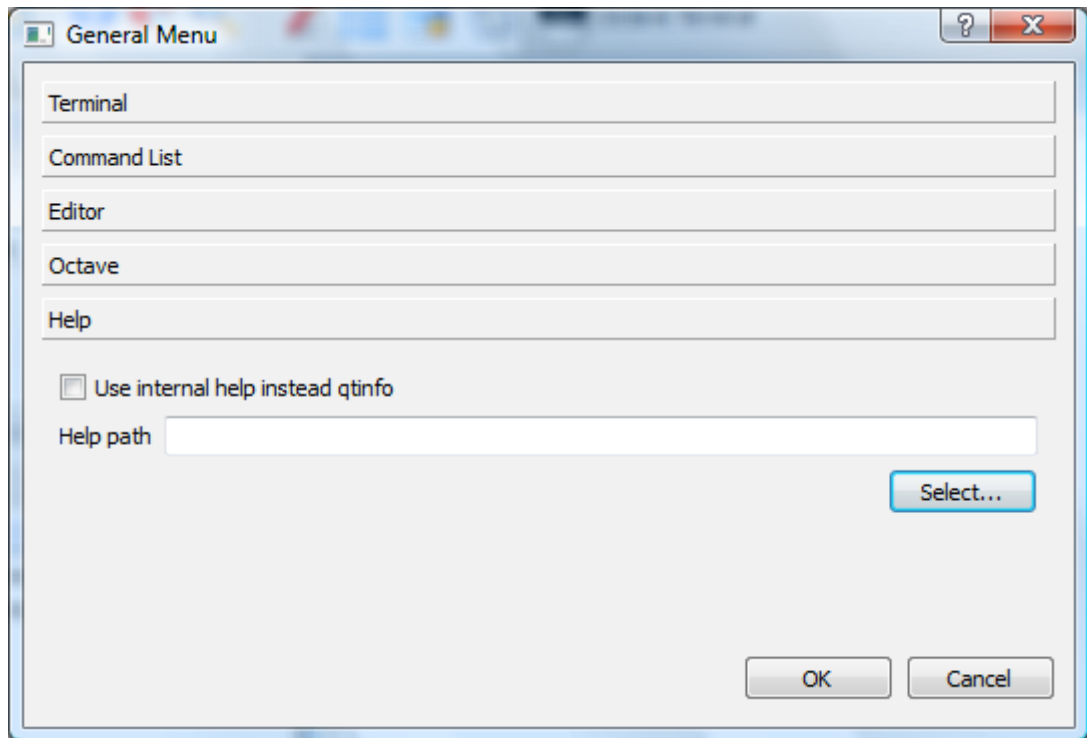
Pulsamos OK y ya se inicia el programa. El mensaje de error corresponde a que no conoce la dirección donde está el ejecutable de Octave. Cuando entramos la ventana es:



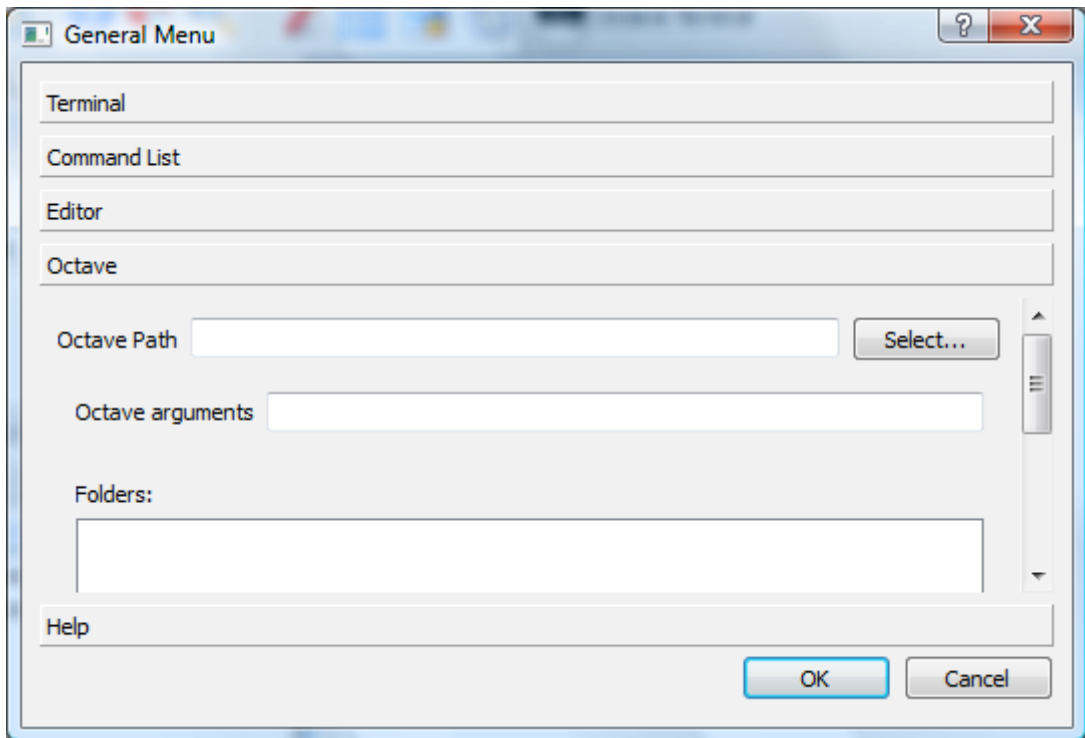
Ahora para definir la dirección de donde está octave vamos a la pestaña **Config** y dentro de ella **General configuration**.



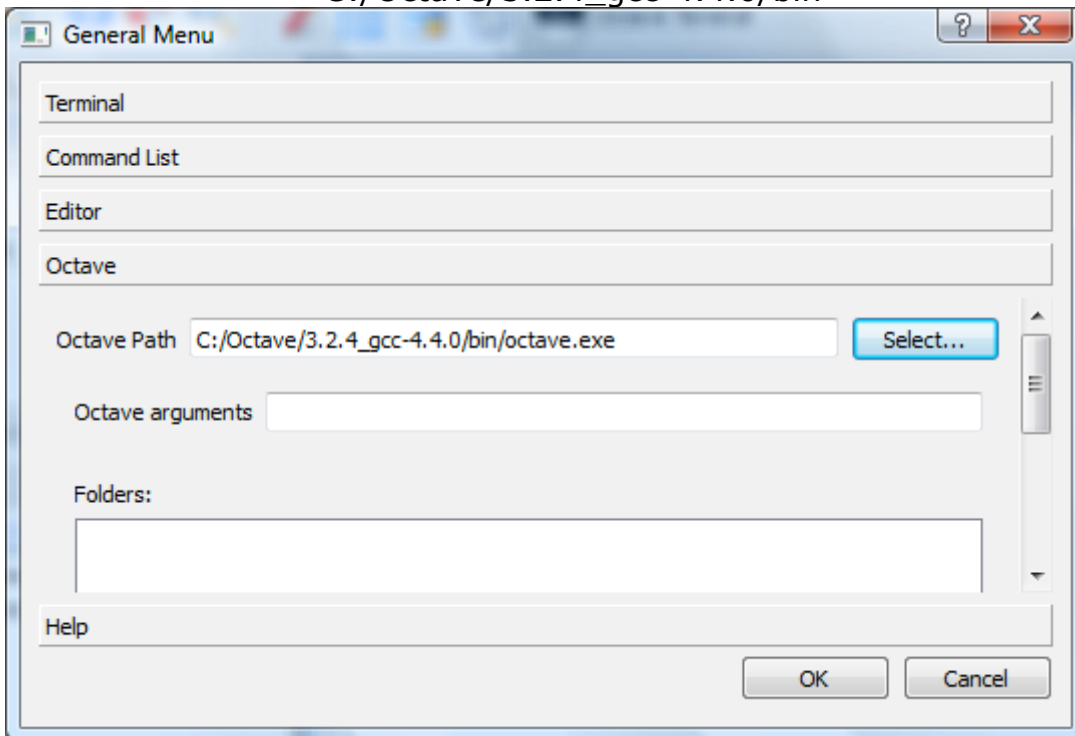
El menú que aparece entonces es el siguiente:



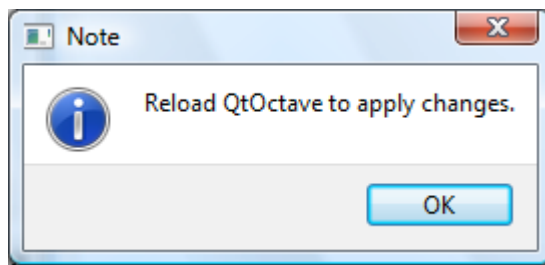
Aquí pulsamos sobre Octave:



En Octave Path, es donde debemos decir donde está el programa octave.exe, lo habitual como ya decíamos antes es:  
C:/Octave/3.2.4\_gcc-4.4.0/bin



Al pulsar OK nos dirá debemos cerrar el programa QtOctave para que se realicen los cambios.



Reiniciamos el programa y ya está listo para poder utilizarlo.

## El entorno gráfico Qt octave

La ventana principal del programa es:

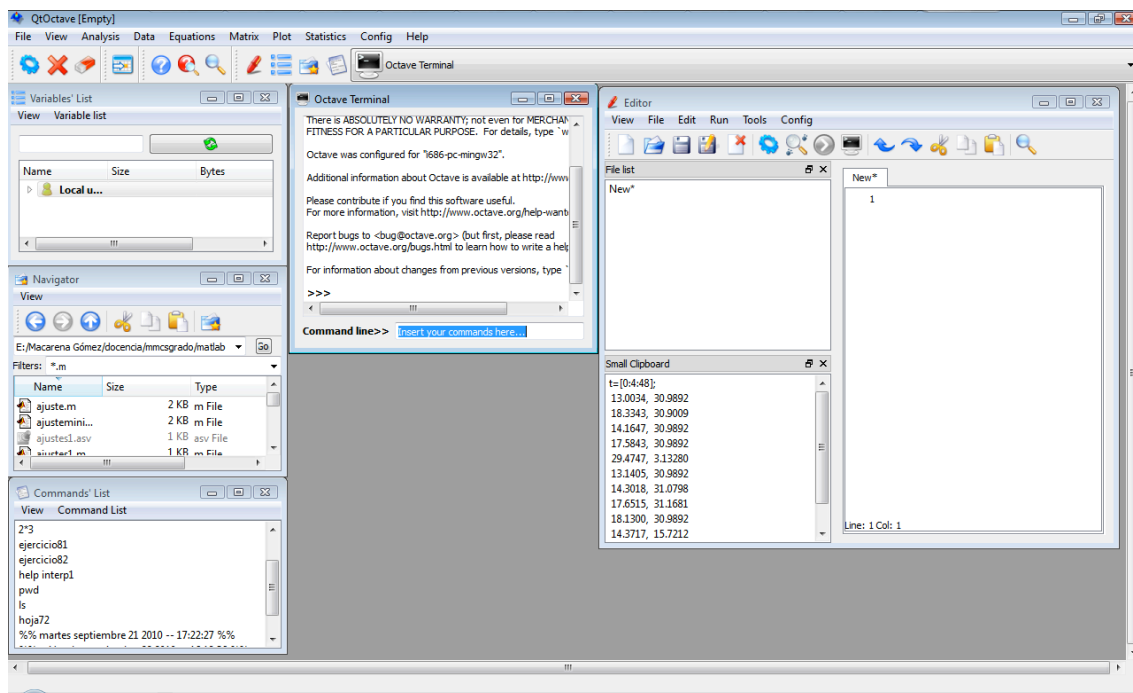


Figura1. Ventana inicial de Qt octave

La parte más importante de la ventana inicial es la **Octave Terminal**, que aparece en la parte derecha (recuadrada en azul en la Figura 2). En esta sub-ventana es donde se ejecutan los comandos de Octave, a continuación del **Command Line >>** que indica que el programa está preparado para recibir instrucciones.



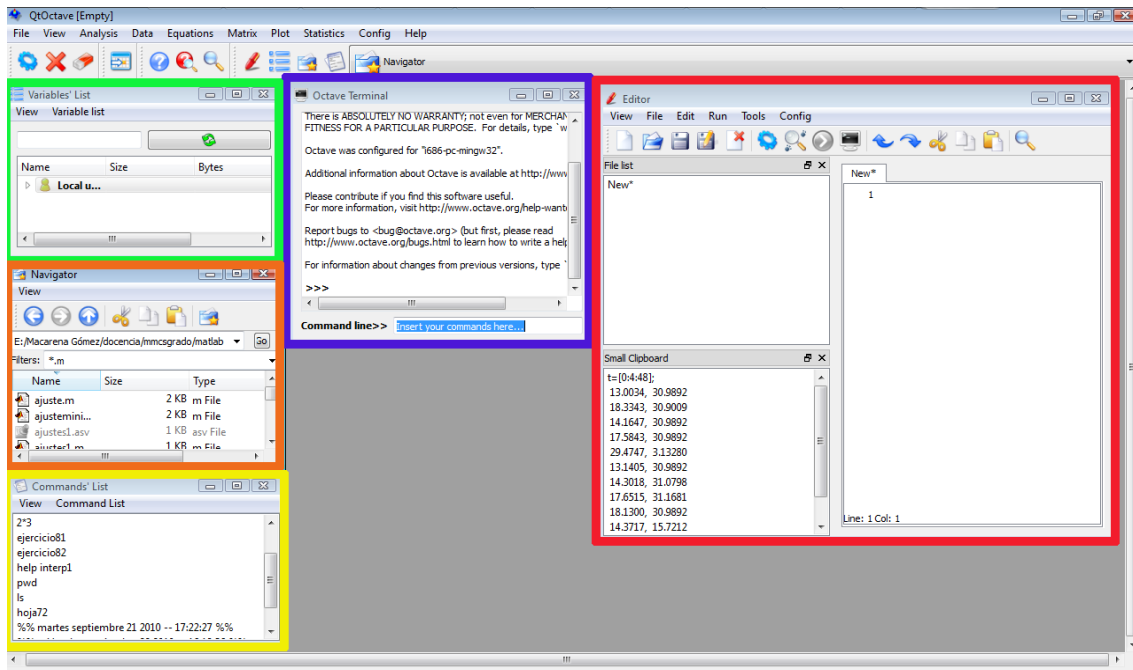


Figura 2. Sub-ventanas de comandos.

En la parte izquierda de la pantalla aparecen tres ventanas también muy útiles: en la parte superior aparece la ventana (recuadrada en verde en la Figura 2) **Variable Lists** que contiene información sobre todas las variables que se hayan definido en esta sesión y permite ver y modificar las matrices y vectores con los que se esté trabajando. A continuación tenemos la pantalla **Navigator** (recuadrada en naranja en la Figura 2) donde se muestra los ficheros del directorio activo o actual.

En la parte inferior aparece la ventana (recuadrada en amarillo en la Figura 2) **Commands List** que muestra los últimos comandos ejecutados en la **Octave Terminal**. Estos comandos se pueden volver a ejecutar haciendo doble clic sobre ellos.

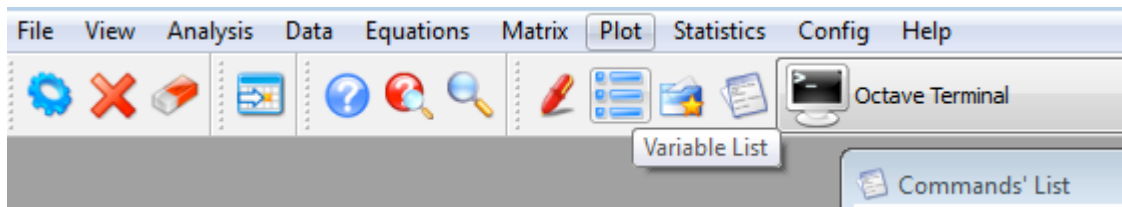
Por último aparece una ventana (recuadrada en rojo en la Figura 2) que corresponde al Editor de texto para los programas.

Estas ventanas pueden ser colocadas en cualquier orden dentro del espacio de trabajo, pueden estar cerradas si no van a ser utilizadas, y pueden abrirse en cualquier momento. Para ello nos fijamos en la barra de herramientas, a continuación vemos los iconos que corresponden a cada ventana:

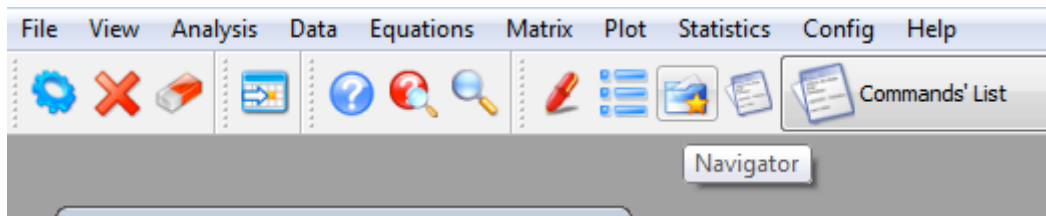
## Editor:



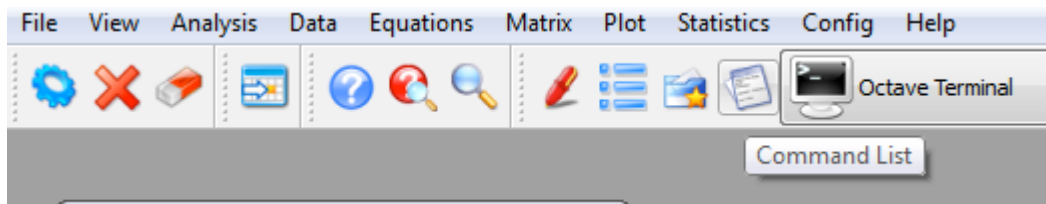
### Variable List:



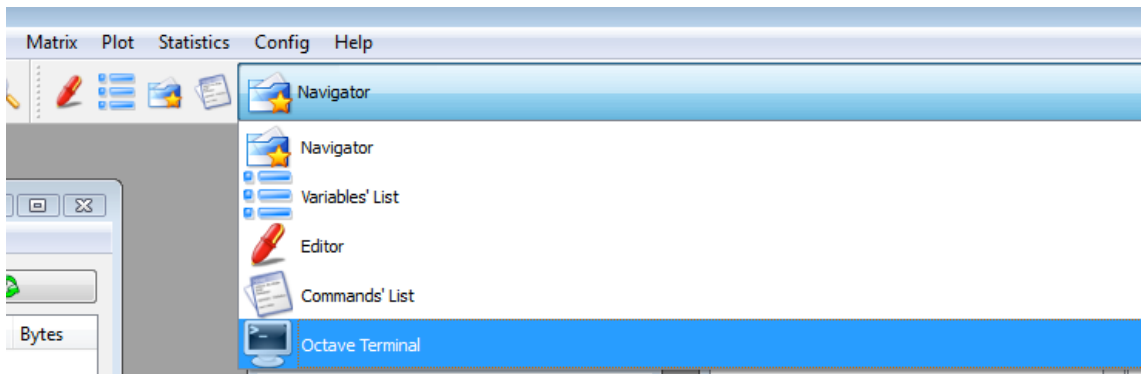
### Navigator:



### Command List



O bien podemos utilizar el menú despegable que aparece a continuación:



A continuación vamos a ver el resto de los iconos que aparecen en esta barra de Herramientas:



Este icono es para ejecutar un fichero, al pulsar sobre el, sale un ventana que nos permite elegir el fichero, con una pestaña de tipo examinar.



Este icono es muy útil y sirve para parar un proceso, procesos bloqueados, bucles infinitos, etc. Esto es equivalente a Ctrl+C.



Este icono limpia todas las variables que se ha utilizado hasta ahora en esta sección y es equivalente al comando **clear all**.

Para salir de la aplicación basta elegir **Quit** en el menú **File** o utilizar cualquiera de los medios de terminar una aplicación en Windows.

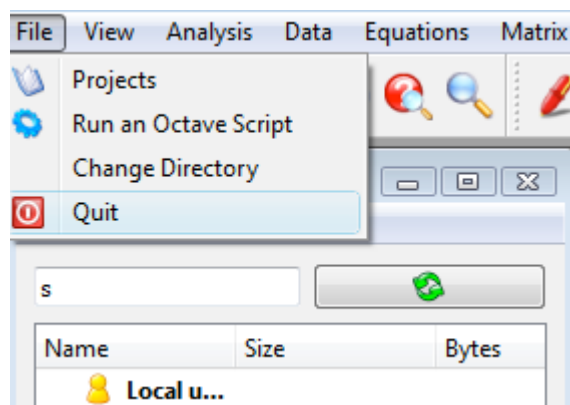
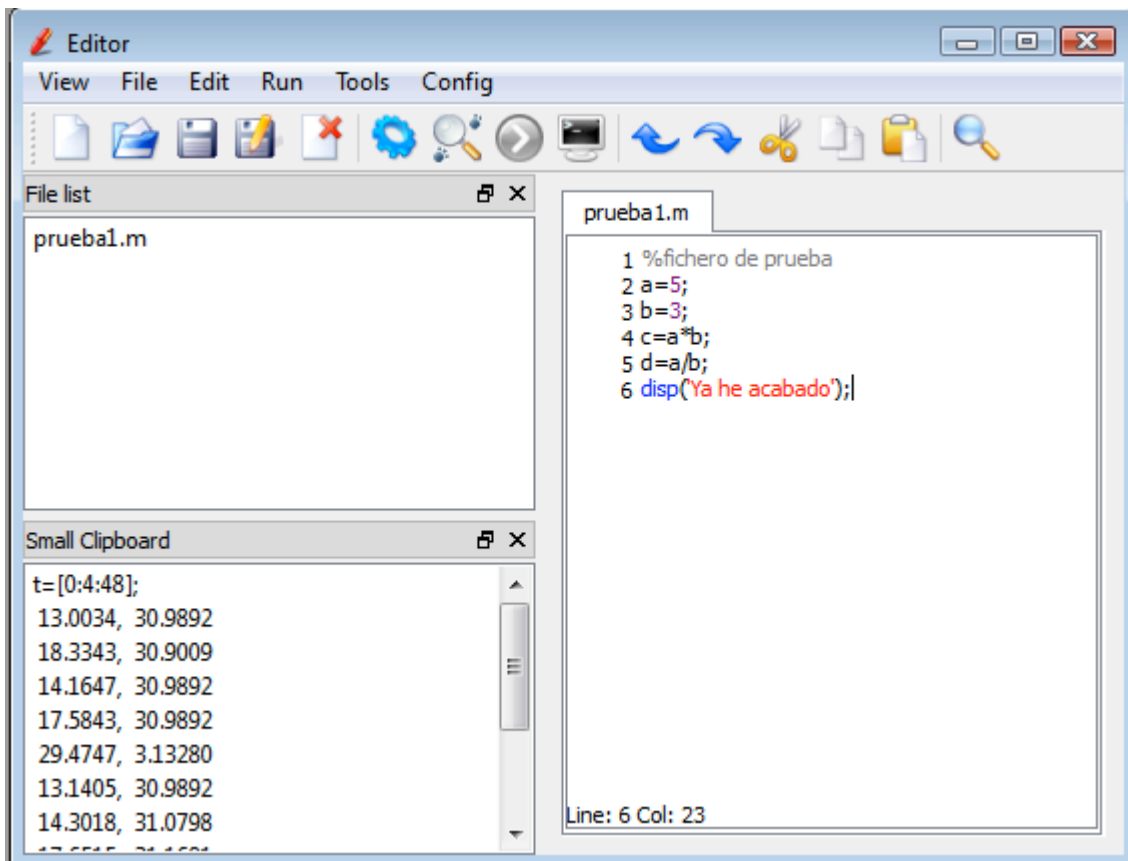


Figura 3: Salir de Qt octave

Las instrucciones se pueden proporcionar directamente en la ventana de comando (**Octave terminal**) o a través de **ficheros-M** (o **M-files**). Estos ficheros tienen la extensión **\*.m** y contienen conjuntos de comandos o definiciones de funciones. La importancia de estos **ficheros-M** es que al teclear su nombre en la línea de comandos y pulsar **Intro**, se ejecutan uno tras otro todos los comandos contenidos en dicho fichero. El poder guardar las instrucciones en un fichero permite ahorrar mucho tiempo y trabajo.

Aunque los ficheros **\*.m** se pueden crear con cualquier editor de texto tal como Bloc de Notas, Notepad, Word, etc, Qt octave dispone de un **editor** que permite crear y modificar estos ficheros, como ejecutarlos paso a paso para ver si contiene errores. En la ventana editor hemos tecleado un **fichero-M** llamado **prueba1.m**. El editor muestra con diferentes colores los diferentes tipos o elementos constitutivos de los comandos (en gris los comentarios, en rojo las cadenas de caracteres, etc.). Las líneas de comentario se indican con el carácter **%** delante del comentario.



Para ejecutar el fichero basta teclear en la ventana **Octave Terminal** en **Command Line** **>> prueba1**.

La respuesta aparece en la ventana **Octave Terminal** y en este caso solo es la que vemos a continuación:

```
>>> prueba1
Ya he acabado
>>>
```

## Definición de vectores desde el teclado

Como en casi todos los lenguajes de programación las matrices y vectores son variables que tienen nombres. Para definir una matriz o un vector no hace falta establecer de antemano su tamaño, de hecho, se puede definir inicialmente un tamaño y cambiarlo posteriormente. Octave determina el número de filas y de columnas en función del número de elementos que se proporcionan.

Las matrices se definen por filas, los elementos de la misma fila están separados por blancos o por comas, mientras que las filas están separadas por pulsaciones intro o por caracteres punto y coma (;). Por ejemplo el siguiente comando define una matriz **A** de dimensión 3x3:

```
>>>A=[1 2 3 ; 4, 5, 6
7 8 9]
```

La respuesta del programa es:

**A =**

```
 1   2   3
 4   5   6
 7   8   9
```

De forma análoga a las matrices, es posible definir un vector fila **x** en la forma siguiente:

```
>>>x=[10 20 30]
```

La respuesta del programa es:

**x =**

```
10  20  30
```

En Octave se accede a los elementos de un vector poniendo el índice entre paréntesis, por ejemplo

```
>>> x(2)
```

La respuesta es:

**ans =**

## 20

Para definir un vector columna **y** hay que separar los elementos por (;) o intro,

```
>>>y=[11; 12; 13]
y =
```

```
11
12
13
```

Y de la misma forma

```
>>> y(3)
ans =
```

```
13
```

Octave tiene en cuenta la diferencia entre vectores fila y vectores columna. Si intentamos sumar los vectores **x** e **y** obtendremos el siguiente mensaje de error:

```
>>> x + y
>>>error: operator +: nonconformant arguments (op1 is 1x3, op2 is 3x1)
```

Para poder sumar los vectores necesitamos que ambos sean vectores filas o ambos columna.

Si hacemos

```
>> x'
```

Transforma el vector fila x en un vector columna

```
ans =
```

```
10
20
30
```

Si sumamos ahora obtenemos:

```
>>> x'+y
ans =
```

```
21
32
43
```

Si transformamos y en vector fila y sumamos, entonces

```
>> x+y'
```

**ans =**

**21 32 43**

## Operaciones entre vectores

Octave puede operar con vectores por medio de **operadores** y por medio de **funciones**. Los operadores son los siguientes:

- Suma (+) : `>>> x+y'`  
**ans =**  
**21 32 43**
- Resta (-) : `>>> x-y'`  
**ans =**  
**-1 8 17**
- Multiplicación (\*):
  1. Vector fila por vector columna, el resultado es un número.  
`>>> x*y`  
**ans =**  
**740**
  2. Vector columna por vector fila, el resultado es una matriz.  
`>>> y*x`  
**ans =**  
**110 220 330**  
**120 240 360**  
**130 260 390**
- Producto elemento a elemento (.\*)
  1. Por un número: `>>>x.*3`  
**ans =**  
**30 60 90**
  2. Por un vector elemento a elemento: `>>> x.*y'`  
**ans=**  
**110 240 390**
- División elemento a elemento
  1. División de todos los elementos del vector por un número(./)  
`>>> x./3`  
**ans=**  
**3.3333 6.6667 10.0000**

2. División de un número por todos los elementos de un vector (./)

```
>>> x.\3
ans=
    0.3000    0.1500    0.1000
```

- Elevar a una potencia elemento a elemento (.^)

```
>>> x.^3
ans=
    1000     8000    27000
```

Si anteriormente no hemos definido el vector x se hace de la misma forma, es decir:

```
>>> [1 2 3 4]^2
>>> error: for A^b, A must be square
>>>
```

```
>>> [1 2 3 4].^2
ans =
```

```
    1    4    9   16
```

```
>>> [1 2 3 4]*[1 -1 1 -1]
>>> error: operator *: nonconformant arguments (op1 is 1x4, op2 is 1x4)
>>>
```

```
>>> [1 2 3 4].*[1 -1 1 -1]
ans =
```

```
    1   -2    3   -4
```

## Tipos de datos

**Números reales de doble precisión:** Los elementos constitutivos de los vectores y matrices son números reales. Octave mantiene una forma especial para los números muy grandes (más grandes que



lo que es capaz de representar), que son considerados como **infinito**. Por ejemplo, obsérvese como responde el programa al ejecutar el siguiente comando:

```
>>> 1.0/0.0
ans = Inf
>>> warning: division by zero
>>>
```

Así pues, para Octave el **infinito** se representa como **inf** o **Inf**. Octave tiene también una representación especial para los resultados que no están definidos como números. Por ejemplo:

```
>>> 0/0
ans = NaN
>>> warning: division by zero
>>>
```

```
>>> inf/inf
```

```
ans =
```

```
NaN
```

En ambos casos, la respuesta es **NaN**, que es la abreviatura de **Not a Number**.

Las funciones `realmin` y `realmax` devuelven el número más pequeño y el más grande respectivamente con el que se puede trabajar.

```
>>> realmin
ans =
```

```
2.2251e-308
```

```
>>> realmax
```

```
ans =
```

```
1.7977e+308
```

**Cadenas de caracteres.** En Octave las cadenas de texto van entre apóstrofes o comillas simples. Por ejemplo

```
>>> s='hola soy una cadena de caracteres'
```

```
s =
```

## **hola soy una cadena de caracteres**

**Variables.** Una **variable** es un nombre que se da a una entidad numérica que puede ser una matriz, un vector o un escalar. El valor de esta variable puede cambiar a lo largo de una sesión de Octave o a lo largo de la ejecución de un programa. La forma más normal de cambiar el valor de una variable es colocándola a la izquierda del **operador de asignación (=)**.

Por ejemplo:

```
>>> t=1
```

```
t =  
    1
```

```
>>> t=t+1
```

```
t =  
    2
```

```
>>> t=5
```

```
t =  
    5
```

Cuando a un resultado no le asignamos ningún nombre, MATLAB le asigna automáticamente una variable interna llamada **ans** que almacena el último resultado obtenido.

```
>>> t+3
```

```
ans =  
  
    8
```

Si se desea que una expresión continúe en la línea siguiente, hay que introducir **tres puntos (...)** antes de pulsar **intro**.

```
>>> t*2 ...
```

```
*5  
ans =  
    50
```

También se pueden incluir varias expresiones en la misma línea separándolas por comas (,) o punto y coma (;)

```
>>> t*2, t*5
```

```
ans =  
    10
```

```
ans =  
    25
```

Si una expresión **termina en punto y coma (;)** su resultado se calcula, pero no se escribe en pantalla.

```
>>> t*2;t*5
```

```
ans =
```

```
>>> t*2; t*5;
>>>
```

Esta posibilidad es muy interesante, tanto para evitar la escritura de resultados intermedios, como para evitar la impresión de grandes cantidades de números cuando se trabaja con matrices o vectores de gran tamaño.

**Otras formas de definir vectores.** Octave dispone de varias formas de definir los vectores. El introducirlos por teclado sólo es práctico en casos de pequeño tamaño y cuando no hay que repetir esa operación muchas veces. Recordemos también que el tamaño de los vectores puede ser modificado por el usuario mediante la adición o sustracción de elementos. A continuación vamos a ver otras formas de definir los vectores:

Tipos de vectores predefinidos mediante funciones. Existen en MATLAB funciones para definir vectores con gran facilidad, algunas de estas funciones son las siguientes:

- Vectores fila o columna donde todos los elementos son cero.

**Zero(n,1)**= Vector **columna** donde los **n** elementos son **cero**.

**Zero(1,n)**= Vector **fila** donde los **n** elementos son **cero**.

```
>>> zeros(3,1)
```

```
ans =
```

```
0
0
0
```

```
>>> zeros(1,3)
```

```
ans =
```

```
0 0 0
```

Vectores fila o columna donde todos los elementos son unos.

**ones(n,1)**= Vector **columna** donde los **n** elementos son **uno**.

**ones(1,n)**= Vector **fila** donde los **n** elementos son **uno**.

```
>>> ones(3,1)
```

```
ans =
```

```
1
1
1
```

```
>>> ones(1,3)
```

```
ans =
```

```
1 1 1
```

Generar un vector con **n** valores igualmente espaciados entre **x1** y **x2**, la orden se escribe **linspace(x1,x2,n)**

```
>>> linspace(1,10,10)
```

```
ans =
```

```
1 2 3 4 5 6 7 8 9 10
```

```
>>> linspace(0,1,10)
```

```
ans =
```

```
Columns 1 through 7
```

```
0 0.1111 0.2222 0.3333 0.4444 0.5556
0.6667
```

```
Columns 8 through 10
```

```
0.7778 0.8889 1.0000
```

Calcular el número de elementos de un vector **x**, la orden es **length(x)**

```
>>> x=linspace(1,10,20);
```

```
>>> length(x)
```

```
ans =
```

```
20
```

Tipos de vectores predefinidos mediante el operador dos puntos (:)

El operador dos puntos es muy importante en MATLAB y puede usarse de varias formas.

Para empezar, definimos un vector  $\mathbf{x}$  con el siguiente comando:

```
>>> x=1:10  
x =
```

```
1 2 3 4 5 6 7 8 9 10
```

En cierta forma se podría decir que el operador (:) representa un rango; en este caso, los números enteros entre 1 y 10. Por defecto el incremento es 1, pero este operador puede también utilizarse con otros valores enteros y reales, positivos y negativos. En este caso el incremento va entre el valor inferior y el superior, en las formas que se muestran a continuación:

```
>>> x=1:2:10  
x =
```

```
1 3 5 7 9
```

```
>>> x=1:1.5:10  
x =
```

```
1.0000 2.5000 4.0000 5.5000 7.0000 8.5000  
10.0000
```

```
>>> x=10:-1:1  
x =
```

```
10 9 8 7 6 5 4 3 2 1
```

Como puede verse, por defecto este operador produce vectores fila. Si se desea obtener un vector columna basta transponer es decir  $\mathbf{x}'$ .

El siguiente ejemplo genera un vector donde las 10 primeras componentes del vector son los números naturales, y las 10 siguientes sus cuadrados:

```
>>> x=[1:10]  
x =
```

```
1 2 3 4 5 6 7 8 9 10
```

```
>>> y=x.^2  
y =
```

```
1 4 9 16 25 36 49 64 81 100
```

```
>>> [x y]
```

ans =

**Columns 1 through 13**

**1 2 3 4 5 6 7 8 9 10 11 12 13**

**Columns 14 through 20**

**16 25 36 49 64 81 100**

## **FUNCIONES DE OCTAVE**

**Funciones Matemáticas elementales:** Las funciones que a continuación vamos a describir corresponden a funciones matemáticas elementales y se aplican a valores escalares o a vectores elemento a elemento.

<b>sin(x)</b>	seno
<b>cos(x)</b>	coseno
<b>tan(x)</b>	Tangente
<b>log(x)</b>	logaritmo neperiano
<b>log10(x)</b>	logaritmo decimal
<b>exp(x)</b>	Exponencial
<b>sqrt(x)</b>	raíz cuadrada
<b>round(x)</b>	redondeo hacia el entero mas próximo
<b>abs(x)</b>	valor absoluto

**Funciones que actúan sobre vectores:** Las siguientes funciones solo actúan sobre vectores.

<b>[xm,im ]=max(x)</b>	Devuelve el valor máximo <b>xm</b> y la posición <b>im</b>
<b>[ym,jm]=min(x)</b>	Devuelve el valor mínimo <b>ym</b> y la posición <b>jm</b>
<b>sum(x)</b>	Suma de los elementos de un vector
<b>mean(x)</b>	Valor medio de los elementos del vector
<b>std(x)</b>	Desviación típica.

**Funciones para cálculos con polinomios:** Para Octave un polinomio se puede definir mediante un vector de coeficientes. Por ejemplo, el polinomio:

$$x^4-8x^2+6x -10=0$$

se puede representar mediante el vector  $[1, 0, -8, 6, -10]$ . Octave puede realizar diversas operaciones sobre él, como por ejemplo evaluarlo para un determinado valor de  $x$

```
>>> pol=[1 0 -8 6 -10]
```

```
pol =
```

```
    1    0   -8    6  -10
```

```
>>> polyval(pol,1)
```

```
ans =
```

```
   -11
```

Otras funciones orientadas al cálculo de polinomios son las siguientes:

**polyfit(x,y,n)**: calcula los coeficientes de un polinomio  $p(x)$  de grado  $n$  que se ajusta a los datos  $p(x(i)) \approx y(i)$ , en el sentido de mínimo error cuadrático medio.

Por ejemplo, consideremos los puntos  $x=[1 2 3 4 5]$  y el vector  $y=[1 -1 3 0 0.5]$ , y queremos calcular un polinomio que aproxime a esos puntos. (Figura 4)

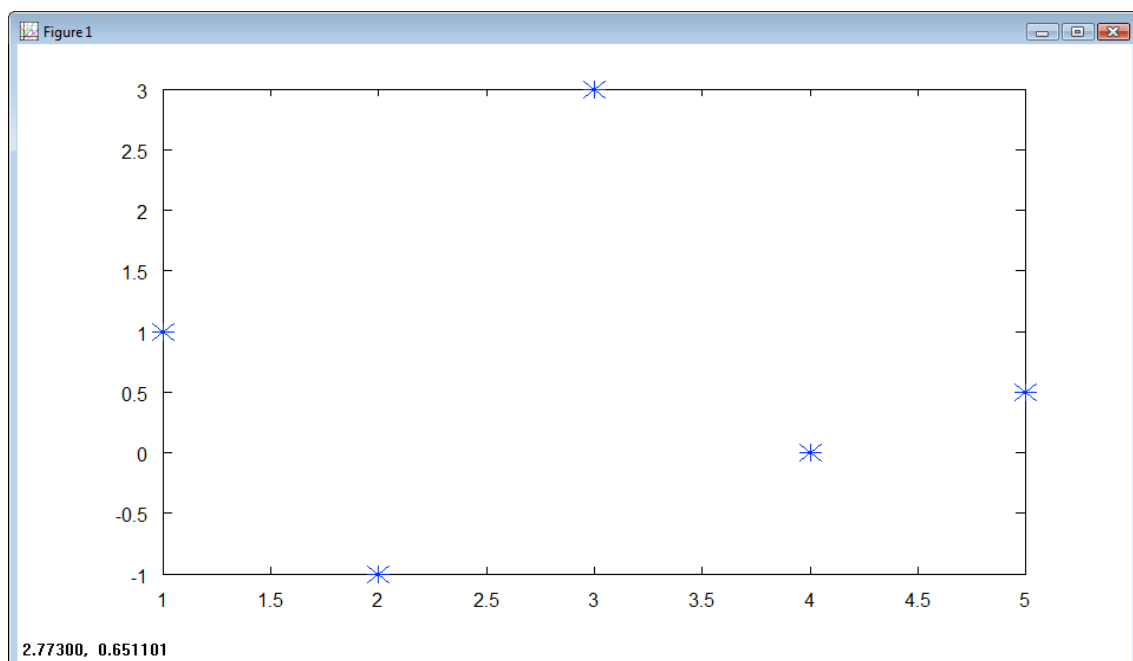


Figura 4: Puntos que queremos aproximar

Ejecutamos los siguientes comandos

```
>>> x
```

```
x =
```

```
1 2 3 4 5
```

```
>>> y
```

```
y =
```

```
1.0000 -1.0000 3.0000 0 0.5000
```

```
>>> polyfit(x,y,2) (Figura 5: gráfica roja)
```

```
ans =
```

```
-0.1429 0.8571 -0.3000
```

```
>>> polyfit(x,y,4) (Figura 5: gráfica verde)
```

```
ans =
```

```
0.9792 -11.9583 50.2708 -83.7917 45.5000
```

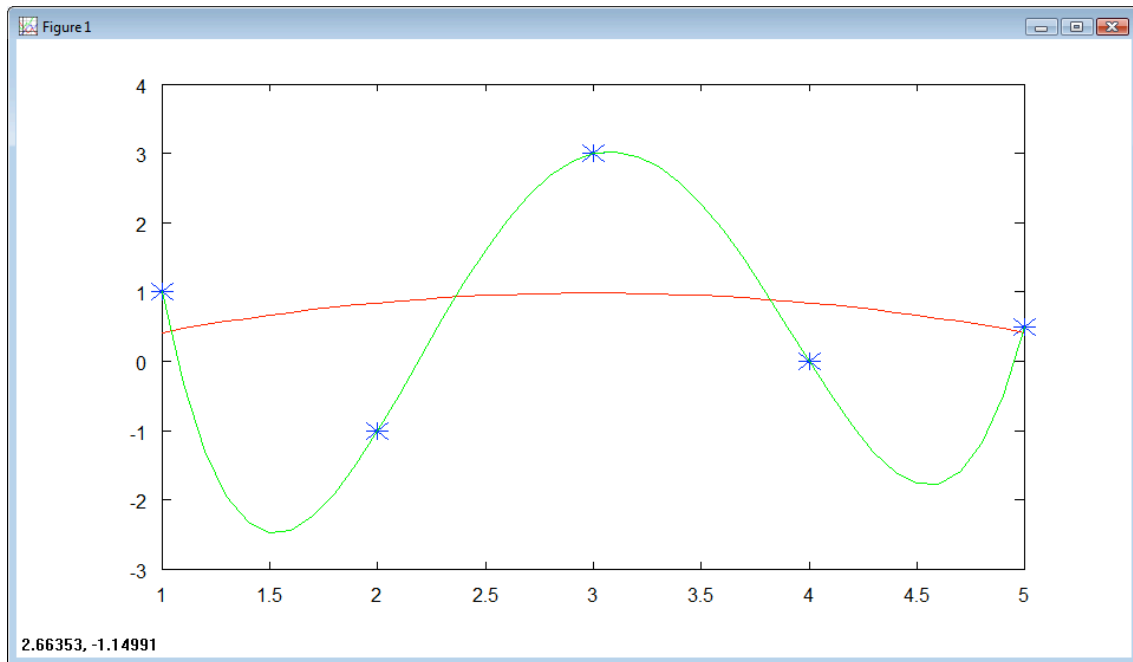


Figura 5: Polinomios de aproximación a los puntos dados.



**spline(x,y)**: calcula un polinomio de grado tres en cada intervalo  $[x(i),x(i+1)]$ , tal que  $p(x(i))=y(i)$ . (Figura 6)

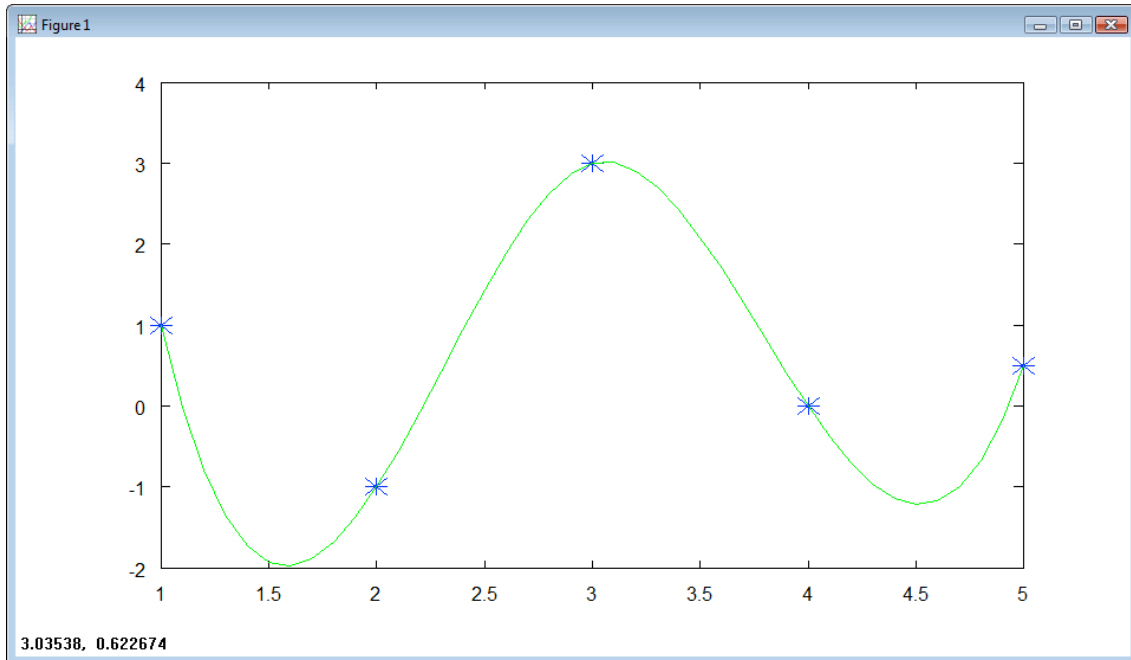


Figura 6: Aproximación por splines cúbicos.

**trapz(x,y)** : calcula la integral de la poligonal que une los puntos  $(x(i), y(i))$ , con su correspondientes signos. En la gráfica 7 hemos rellenado en verde la parte que tiene integral negativa. En el caso que todos los elementos del vector  $y$  sean positivos, esta función nos proporciona el área.

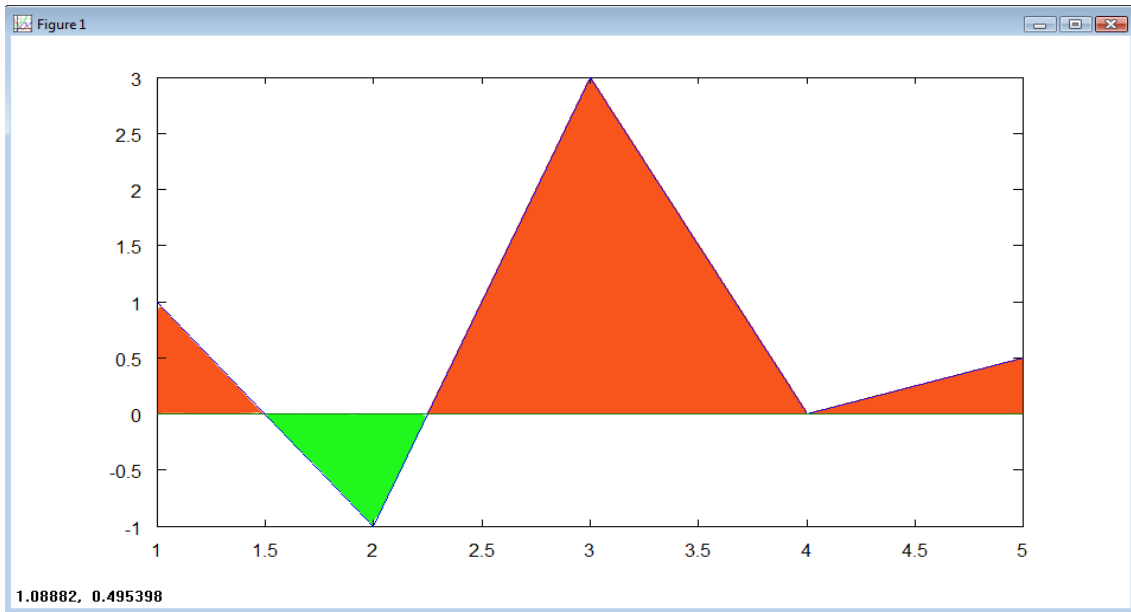


Figura 7: Cálculo de la integral.

Figura 9: Cálculo de la integral

## PROGRAMACIÓN DE OCTAVE

### Bifurcaciones y bucles

Octave es una aplicación que se puede programar muy fácilmente. Las más básicas son las **bifurcaciones** y **los bucles**. La Figura 8 muestra dos posibles formas de bifurcaciones.

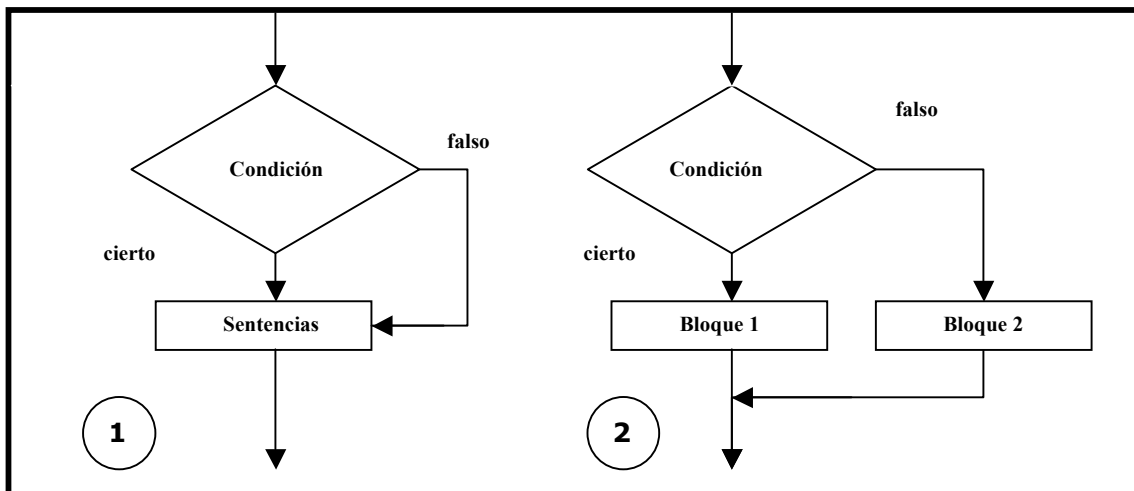


Figura 8: Ejemplos gráficos de bifurcaciones.

Los **bucles** permiten repetir las mismas o análogas operaciones sobre datos distintos. La figura 9 muestra dos posibles formas de bucle, con el control situado al principio o al final del mismo. Si el control está situado al comienzo del bucle es posible que las sentencias no se ejecuten ninguna vez, por no haberse cumplido la condición cuando se llega al bucle por primera vez. Sin embargo, si la condición está al final del bucle las sentencias se ejecutarán por lo menos una vez, aunque la condición no se cumpla.

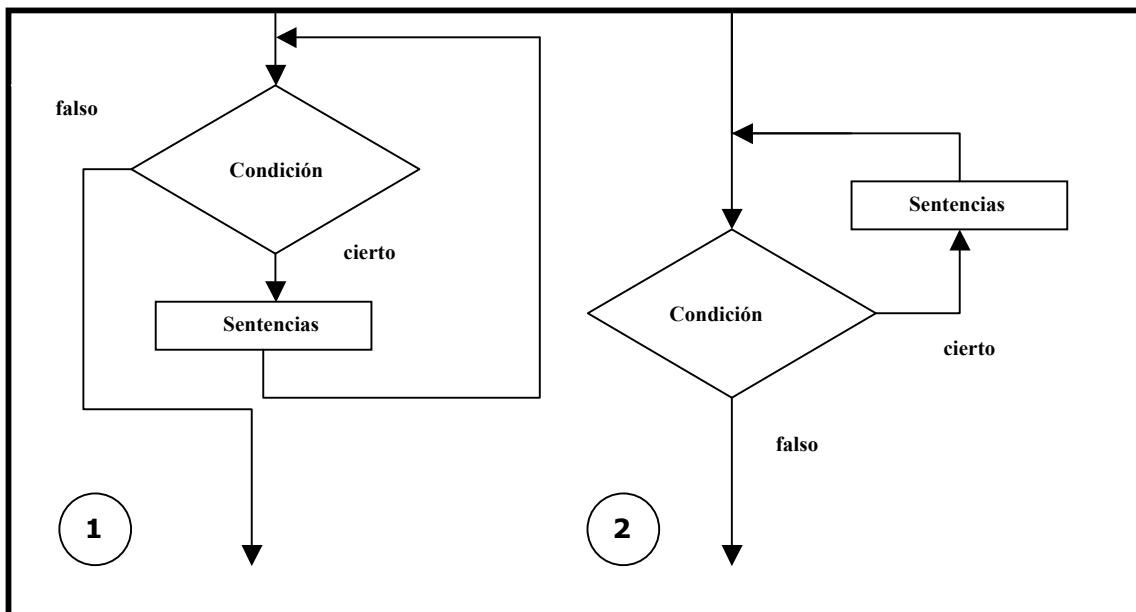


Figura 9: Bucles con control al principio y al final.

## Sentencia if

En su forma más simple, la sentencia **if** se escribe de la forma siguiente

Caso 1 de la Figura 8:

```
if condicion  
    sentencias  
end
```

Caso 2 de la Figura 8:

```

if condicion
    bloque 1
else
    bloque 2
end

```

Existe también la bifurcación múltiple, en la que pueden concatenarse tantas condiciones como se desee, y que tiene la forma:

```

if condicion1
    bloque 1
elseif condicion2
    bloque 2
elseif condicion3
    bloque 3
else %opcion por defecto para cuando no se cumplan
1,2,3
    bloque4
end

```

donde la opción por defecto **else** puede ser omitida: si no está presente no se hace nada en caso de que no se cumpla ninguna de las condiciones que se han chequeado.

### Sentencia for

La sentencia **for** repite un conjunto de sentencias un número predeterminados de veces. La siguiente construcción ejecuta sentencias con valores de **i** de **1** a **n**, variando de uno en uno.

```

for i=1:n
    sentencias
end

```

Por ejemplo,

```

>>> x=linspace(0,1,5)
x =
    0    0.2500    0.5000    0.7500    1.0000
>>> n=length(x)
n =
    5
>>> for i=1:n
i,x(i)
end

i =

```

```

    1
ans =
    0
i =
    2
ans =
    0.2500
i =
    3
ans =
    0.5000
i =
    4
ans =
    0.7500
i =
    5
ans =
    1

```

El caso más general para la variable del bucle (valor-inicial:incremento:valor-final);

En el mismo ejemplo anterior, si ponemos **for i=1:2:n**, quiere decir que va del valor **i=1** al **n** pero de dos en dos:

```

>>> for i=1:2:n
i,x(i)
end

```

```

i =
    1
ans =
    0
i =
    3
ans =
    0.5000
i =
    5
ans =
    1

```

El incremento también puede ser negativo, por ejemplo:

```

>>> for i=5:-2:1
i,x(i)

```

```

end

i =
    5
ans =
    1
i =
    3
ans =
    0.5000
i =
    1
ans =
    0

```

### Setencia while

Su sintaxis es la siguiente:

```

while condicion
    sentencias
end

```

donde condición puede ser una expresión vectorial. Las sentencias se siguen ejecutando mientras haya elementos distintos de cero en condición, es decir mientras haya algún o algunos elementos **true**. El bucle se termina cuando todos los elementos de condición son **false**.

Veamos el ejemplo anterior de escribir **i, x(i)** para **i=1:n**, usando ahora la sentencia while

```

>>> i=1
i =
    1
>>> while i<=n
i,x(i)
i=i+1;
end

i =
    1
ans =
    0
i =
    2
ans =

```

```

0.2500
i =
  3
ans =
  0.5000
i =
  4
ans =
  0.7500
i =
  5
ans =
  1
Ficheros *.m

```

Los ficheros con extensión (**.m**) son ficheros de texto sin formato que constituyen el centro de la programación en Octave.

Existen dos tipos de ficheros **\*.m**, los **ficheros de comandos** y las **funciones**. Los primeros contienen simplemente un conjunto de comandos que se ejecutan sucesivamente cuando se teclea el nombre del fichero en la línea de comando de Octave Terminal o se incluye dicho nombre en otro fichero **\*.m**. Un fichero de comandos puede llamar a otros ficheros de comandos. Si un fichero de comando se llama desde la línea de comandos de Octave, las variables que crea pertenecen al espacio de trabajo base de Octave y permanecen en él cuando se termina la ejecución de dicho fichero.

Las **funciones** permiten definir funciones enteramente análogas a las ya predeterminadas, con su **nombre**, sus **argumentos** y sus **valores de retorno**. Las funciones definidas en ficheros **\*.m** se caracterizan porque la primera línea (que no sea de comentario) comienza por la palabra **function**, seguida por los valores de retorno (entre corchetes [ ] y separados por comas, si hay más de uno), el signo (=) y el nombre de la función, seguido de los argumentos (entre paréntesis y separados por comas).

Los ficheros de comandos se pueden llamar también desde funciones, en cuyo caso las variables que se crean pertenecen al espacio de trabajo de la función. El espacio de trabajo de una función es independiente del espacio de trabajo base y del espacio de trabajo de las demás funciones.

A continuación se verá un poco más detallado ambos tipos de ficheros.

## **Ficheros de comandos**

Como ya hemos dicho los ficheros de comando, contienen una sucesión de comandos análoga a la que se teclearía en el uso interactivo del programa. Dichos comandos se ejecutan sucesivamente cuando se tecldea el nombre del fichero que los contiene sin la extensión. En la figura 10 vemos el fichero prueba2.m, y la ejecución resultante es:

```
>>> prueba2
```

```
y =
```

```
2.7183
```

```
y =
```

```
2.7183 7.3891
```

```
y =
```

```
2.7183 7.3891 20.0855
```

```
y =
```

```
2.7183 7.3891 20.0855 54.5982
```



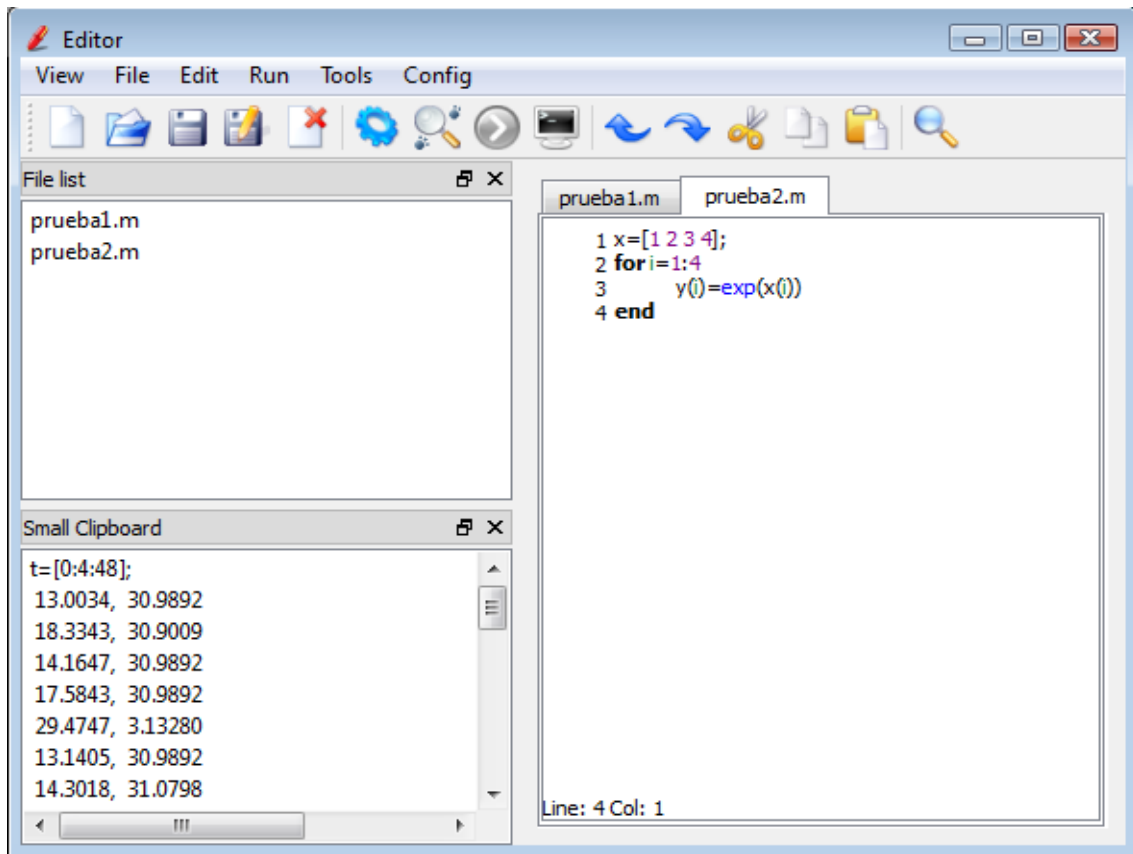


Figura 10: Ejemplo de fichero de comando

### Ficheros de definición de funciones.

La **primera línea** de un fichero llamado **name.m** que define una función tiene la forma:

**function [lista de argumentos de salida] = name (lista de argumentos de entrada)**

donde **name** es el nombre de la función. Entre corchetes y separados por comas van los argumentos de salida (siempre que haya más de uno), y entre paréntesis también separados por comas los argumentos de entrada.

Las variables definidas dentro de una función son variables locales, en el sentido de que son inaccesibles desde otras partes del programa y en el que no interfieren con variables del mismo nombre definidas en otra funciones o partes del programa.

Ejemplo de un fichero que define una función:

`function poblacion(dt)`

```

%Tasas de natalidad b y mortalidad d
b=0.2;
d=0.2;
%Numero de individuos inicialmente
N0=100;
%Calculo del tablero de los tiempos
t=[0:dt:20];
nt=length(t);
%Solucion exacta

for i=1:nt
    sol(i)=N0*exp((b-d)*t(i))
end
%solucion aproximada
N(1)=N0;
for i=2:nt
    N(i)=N(i-1)+b*N(i-1)*dt-d*N(i-1)*dt;
end
%dibujo de las solucion exacta y aproximada, la aproximada en rojo
%y la exacta con *
plot(t,sol,'*',t,N,'r')

```