UNIVERSIDAD DE MURCIA

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

Prof. Salvador Sánchez-Pedreño

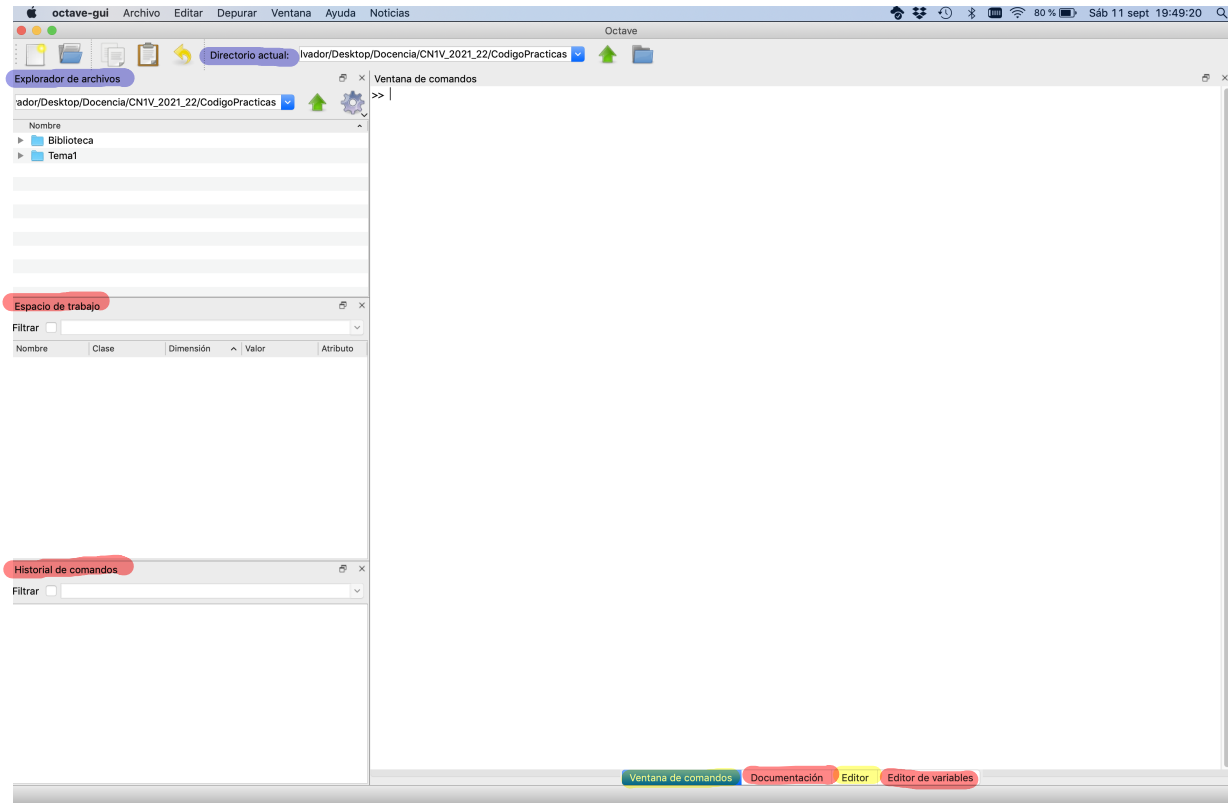
# Introducción a Octave

*Octave es un lenguaje de alto nivel, concebido principalmente para cálculos numéricos. Proporciona una interfaz de línea de comandos para resolver numéricamente problemas lineales y no lineales, y para realizar otros experimentos numéricos utilizando un lenguaje que es casi totalmente compatible con Matlab. También puede ser utilizado como un lenguaje orientado a procesamiento por lotes* (de https://www.gnu.org/software/octave/about).

Posee una interfaz gráfica, que es la que utilizaremos fundamentalmente, y se se suele denominar Octave-Gui. La versión actual de Octave es la 8.3.0.

## 1. Ejecutamos la interfaz gráfica

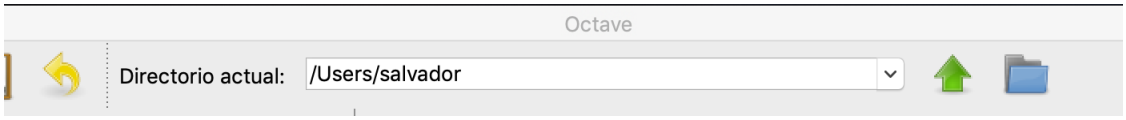
Nos encontramos con esta ventana:



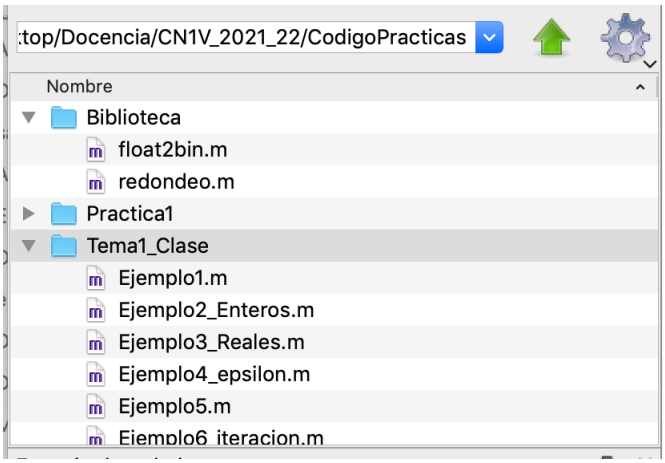

Además de la barra de menús observamos distintas áreas en la ventana:

1. **Directorio actual** (o Current Directory). Es el directorio actual, donde Octave irá a buscar los archivos que debe ejecutar.

   Nosotros fijaremos una carpeta llamada «`/CN1V_2023_24/CodigoPracticas`».



2. **Explorador de archivos** (o File Manager): se comporta como un administrador de archivos, donde podemos movernos por las carpetas y seleccionar el archivo que deseemos editar o ejecutar.

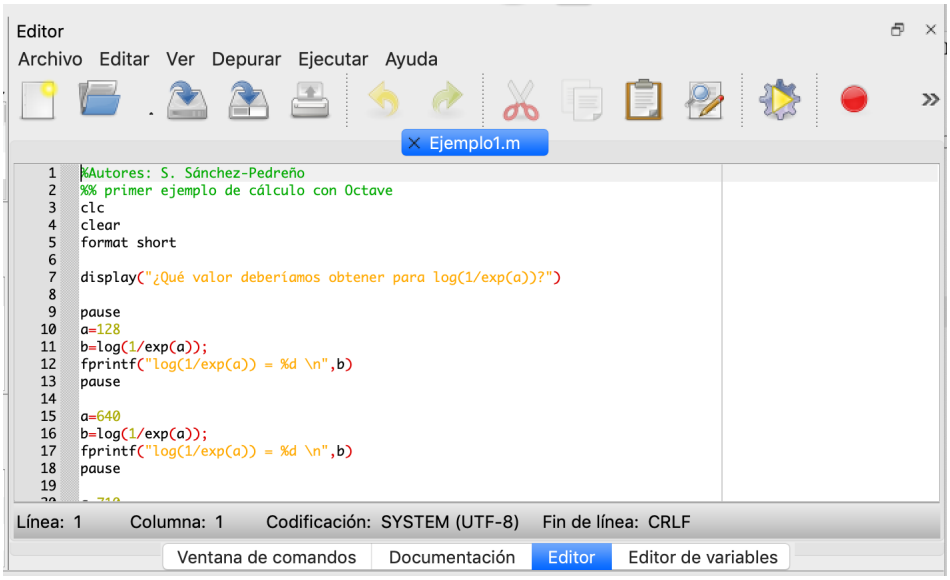



3 En la imagen se muestran las carpetas:

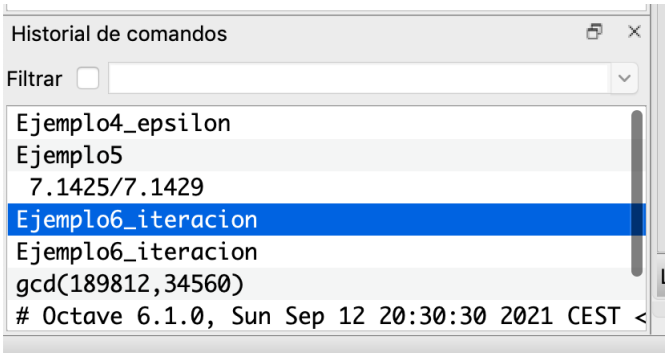
- «Biblioteca», donde guardaremos todas las funciones y scripts auxiliares que utilizaremos en prácticas.
- «Tema1_Clase», donde incluiremos los archivos de los ejemplos mostrados en clase del tema 1, etc.
- «Practica1», donde añadiremos los archivos de código correspondientes a los ejercicios de la primera práctica...

3. **Ventana de comandos** (o Command Window). Se utiliza para introducir órdenes de cálculo directas: las órdenes se escriben después del «prompt» (>>) y una vez escrita se pulsa la tecla «Enter». El resultado de la orden se muestra a continuación, salvo que la orden acabe con «;», en cuyo caso el resultado se guarda en memoria pero no se muestra.


4. **Editor** (o Editor Window). Es la ventana en la que se escriben, ejecutan y depuran los programas archivos de tipo «`script`» o «`function`»: todos estos archivos se guardan con extensión «`.m`».



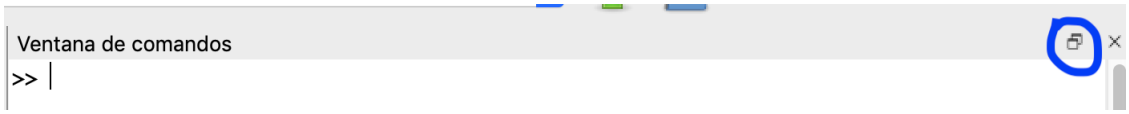

5. **Historial de comandos** (o Command History). Se incluyen en esta ventana todas las órdenes ejecutadas en la ventana de comandos, así como los nombres de los programas ejecutados. Haciendo doble «click» sobre cualquiera de ellas en esta ventana de historial se vuelve a ejecutar. El contenido de la ventana se puede limpiar desde el menú «Editar».




6. **Espacio de trabajo** (o Workspace). Contiene las variables creadas previamente, incluyendo detalles sobre cada una de ellas: dimensiones, tamaño, tipo. El contenido de la ventana se puede limpiar desde el menú «Editar».


7. **Editor de variables** (o Variable Editor). Permite editar y modificar el contenido de las variables. Para activarlo basta con seleccionar la variable en el Espacio de Trabajo y pulsar sobre ella con el ratón.
8. **Documentación** (o Documentation). Ventana con sistema de búsqueda y navegación en la documentación del programa. Como alternativa acudir directamente a Google.

Todas las ventanas son independientes, al iniciar el programa están acopladas en la misma ventana. Se pueden desacoplar (y volver a acoplar) una a una pulsando en el icono de las ventanitas que aparecen en la esquina superior derecha de la barra del título de la ventana.




También se pueden cerrar estas ventanas de forma independiente. Para volverlas a abrir hay que acudir al menú «Ventana» y marcar la ventana que queremos restaurar.

## 2. Operaciones básicas

En Octave (como en Matlab) todas las operaciones están diseñadas para operar directamente con vectores y matrices. De hecho, a priori todas las variables son matrices, de ahí que en la ventana de «Espacio de trabajo» una de las características de las variables que se indica sea su dimensión (aunque esta pueda ser $1 \times 1$).

Por otro lado la sintaxis y orden de preferencia en las operaciones elementales es la habitual. Por ejemplo, ejecuta el código siguiente en la «Ventana de comandos»:

```
>> 5+27^(1/3)
>> (0.1+0.7)/0.4
```

Ahora, realiza los siguientes cálculos con datos vectoriales:

```
>> [2,3]*[4;5]
>> [2,3].*[4,5]
>> [2,3;1,2]
>> [2,3;1,2]^2
>> [2,3;1,2].^2
```

Observa cómo se escriben las filas y columnas en una matriz y cómo se utiliza un punto («`.*`») antes de cualquier símbolo de operación («`*`») para operar con vectores o matrices coordenada a coordenada. La inclusión del punto en algunas operaciones será crucial y fuente de frecuentes errores: cuando definas una función o una operación y las cosas no vayan bien revisa si has olvidado algún punto. La diferencia entre «*» y «.*», por ejemplo, es sencilla de entender: la primera significa multiplicar dos matrices con el producto habitual (por tanto las dimensiones respectivas deben ser las adecuadas); la segunda significa multiplicar coordenada a coordenada las dos matrices, para crear una nueva con las componentes igual a los respectivos productos, or tanto ambas necesitarán tener la misma dimensión (aunque esto no es siempre necesario porque Octave contempla intrínsecamente una operación de «broadcasting», de la que ahora no nos vamos a ocupar). Por ejemplo, Octave sabe calcular $\operatorname{sen}(x)$, siendo $x$ un vector: nos da el vector cuyas componentes son el seno de las componentes de $x$. Pero si queremos calcular $\operatorname{sen}^2(x)$ entonces no debemos hacer `(sin(x))^2`, sino `(sin(x)).^2`. Cuando operamos con escalares o multiplicamos un escalar por un vector el punto no es necesario (hasta la versión 7 de Octave utilizarlo no era un problema, a partir de dicha versión la inclusión del punto en esta situación puede dar un error...).

## 3. Algunas órdenes, variables y funciones

Comenzamos con algunas órdenes de carácter general. Teclea los ejemplos que se muestran para experimentar con ellos.

| Orden | Descripción | Ejemplo (a teclear) |
|---|---|---|
| clc | Limpia el contenido (visible) de la ventana de comandos. | `>> clc` (después de ejecutarlo, desplázate mediante el scroll por la ventana ¿qué observas?). |
| `clear` | Borra todas las variables locales del «Espacio de trabajo». | `>> x=1+1` `>> clear` `>> x` |
| `clear all` | Borra todas las variables (locales y globales) | |
| `%` o `#` | Símbolo para comentarios: se ignora lo que sigue en la línea. | `>>%Hola` `>> #Hola` |
| `;` | Al final de una orden evita que el resultado se muestre en la «Ventana de comandos». | `>> x=1+1;` `>> x` |
| `disp('...')` `display('...')` | Muestra en la «Ventana de comandos» el texto entre las comillas: después inicia nueva línea. | `>> display('Hola desde Octave')` |

Una orden importante para obtener la salida de Octave es

`printf('...',x1,x2,...)`

que muestra en la «Ventana de comandos» el texto entre comillas y las variables o valores indicados por `x1`, `x2`, etc. El texto entre comillas permite incluir referencias a las variables o cantidades con indicación del formato.

Por ejemplo, teclea:

```
>> x=2.1
>> printf('x =%f ; e=%f \n',x,e)
```

Como ves, cada indicación de variable con formato se escribe en la forma «`%c`», donde `c` puede tomar diversos valores, entre otros los siguientes: `%d` (notación decimal usual), `%f` (en punto fijo), `%e` (en notación exponencial o científica), `%g` (equivale a `%f` o `%e`, véase más abajo la orden `format`), `%s` (cadena de caracteres) o `%u` (entero sin signo), y otros...

Observa también que `\n`, dentro del texto entre comillas, produce un salto de línea en la salida; se pueden incluir tantos como se quieran. De forma análoga, `\t` actúa como tabulador en la salida.

Todas estas opciones de formato de salida son muy útiles, por ejemplo, para construir tablas de datos legibles en la ventana de comandos. Utilizaremos con mucha frecuencia el formato `% n.me` o `% n.mf`, cuyo significado es el siguiente: `f` y `e`, como antes, especifican formato en punto fijo o flotante, `n` será el número total de caracteres que ocupará la escritura y `m` el número de cifras significativas que se imprimirán. Así, si `n` es mayor que `m` se incluirán caracteres en blanco a izquierda del número (esto se puede cambiar, para ajustar a izquierda...). Escribe en la ventana de comandos:

```
>> printf('%25.16f\n',e)
>> printf('%25.5f\n',e)
>> printf('%25.5e\n',e)
>> printf('%25.16e\n',e)
```

## 3.1. Declaraciones de formato

| Orden | Descripción | Ejemplo (a teclear) |
|---|---|---|
| `format short` | Formato de punto fijo con 4 dígitos decimales si $0.001 \leq x \leq 1000$; en otro caso equivale a `short e`. | `>> format short`<br>`>> 290/7` |
| `format long` | Punto fijo con 14 dígitos decimales. si $0.001 \leq x \leq 1000$, en otro caso equivale a `long e`. | `>> format long`<br>`>> 290/7` |
| `format short e` | Coma flotante con 4 dígitos decimales. | `>> format e`<br>`>> 290/7` |
| `format long e` | Coma flotante con 15 dígitos decimales. | `>> format long e`<br>`>> 290/7` |
| `format short g` | El mejor entre coma flotante y punto fijo con 5 decimales. | `>> format short g`<br>`>> 290/7` |
| `format long g` | El mejor entre coma flotante y punto fijo con 15 decimales. | `>> format long g`<br>`>> 290/7` |
| `format bank` | Punto fijo con dos decimales | `>> format bank`<br>`>> 290/7` |
| `format bit` | La representación interna en bits, con el bit más significativo en primer lugar. | `>> format bit`<br>`>> 290/7` |

Cada una de estas declaraciones estará activa hasta que la modifique una nueva declaración. Para obtener salidas con `d` cifras decimales, se puede utilizar la orden
`output_precision(d)`
donde `d` debe ser menor o igual que 16. Por ejemplo, teclea:

```
>> format long
>> output_precision(4)
>> 290/7
```

## 3.2. Variables y constantes

Hay dos tipos de variables: locales y globales. Las variables locales se utilizan desde la «Ventana de comandos» o desde un programa (script) concreto. Las variables globales (este atributo aparece indicado en el «espacio de trabajo») pueden ser utilizadas por distintos programas. Para declarar una variable como global se utiliza la sintaxis:

`global` *`var`*

Algunos valores (variables o constantes) predifinidos:

| Nombre | Descripción | Ejemplo (a teclear) |
|---|---|---|
| `ans` | La variable que contiene el último valor calculado. | `>> 2+3`<br>`>> ans` |
| `Inf` o `inf` | Infinito. | `>> 1/0`<br>`>> 1/Inf` |
| `Nan` o `nan` | «Not a Number». | `>> 0/0` |
| `pi` | El número $\pi$. | `>> 2*pi` |
| `e` | El número $e$. | `>> e^2` |
| `i` o `j` | La unidad imaginaria $\sqrt{-1}$ | `>> (2i)^2`<br>`>> 2j*2i` |

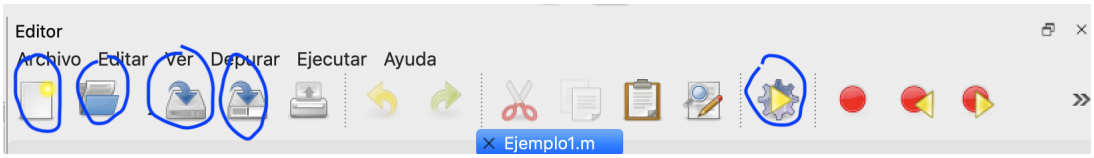
## 3.3. Funciones predefinidas

Algunas de las funciones predefinidas en Octave:

| Nombre | Descripción | Ejemplo (a teclear) |
|---|---|---|
| `eps(x,"clase")` | El épsilon de la máquina: para Octave es la distancia entre $x$ y el siguiente número en coma flotante en precisión `clase`. | `>> eps`<br>`>> eps(2,"single")`<br>`>> eps(2,"double")` |
| `sqrt(x)` | La raíz cuadrada de $x$ | `>> sqrt(81)`<br>`>> sqrt([25, 36; 16, 4])` |
| `exp(x)` | La exponencial de $x$ | `exp([1,2])` |
| `log(x)` | Logaritmo neperiano de $x$ | `log([1,e])` |
| `sin(x)`, `sind(x)` | El seno del ángulo de $x$ radianes, y de $x$ grados sexagesimales. | `>> sin(pi/4)`<br>`>> sind(45)` |
| `cos(x)`, `cosd(x)` | El coseno del ángulo de $x$ radianes, y de $x$ grados sexagesimales. | `>> cos(pi/4)`<br>`>> cosd(45)` |
| `tan(x)`, `tand(x)` | La tangente del ángulo de $x$ radianes, y de $x$ grados sexagesimales. | `>> tan(pi/2)`<br>`>> tand(90)` |
| `abs(x)` | Valor absoluto o módulo de $x$. | `>> abs(-5.2)`<br>`>> abs(2+4i)` |
| `fix(x)` | Redondea $x$ al entero más próximo en la dirección del 0. | `>> fix(-3.7)` |
| `floor(x)` | Redondea $x$ al entero más próximo en la dirección del $-\infty$. | `>> floor(-3.7)` |
| `ceil(x)` | Redondea $x$ al entero más próximo en la dirección del $+\infty$. | `>> ceil(-3.7)` |
| `round(x)` | Redondea $x$ al entero más próximo. | `>> round(3.7)`<br>`>> round(-3.7)` |

## 4. Archivos script y function

Para guardar ciertos cálculos y/o realizar programas disponemos de los archivos `script` y `function`, ambos con extensión «`.m`». Estos archivos se editan desde la ventana «Editor», desde donde también pueden ser ejecutados.




- Para crear un archivo `.m` podemos utilizar el menú «Archivo» de Octave (→ Nuevo guión (script), o → Nueva función), el menú «Archivo» de la ventana «Editor» o el botón adecuado de la barra de botones de la ventana «Editor».
- Para abrir un archivo `.m` puedes utilizar el menú «Archivo» de Octave, de la ventana «Editor» o el botón adecuado.
- Para guardar un archivo utiliza el menú «Archivo» de la ventana «Editor» (→ Guardar archivo..., o → Guardar archivo como...)
- Para ejecutar un archivo utiliza el menú «Ejecutar» de la barra de menús de la ventana «editor» o el botón correspondiente.

**Los nombre de archivos `script` o `function` y de directorios no pueden contener caracteres acentuados ni espacios en blanco y no pueden comenzar con un número.**

**Un archivo de `function` debe tener el mismo nombre que la función que define en su interior.**

En general, los archivos `script` no necesitan tener ninguna estructura particular. Para este curso, recomendamos que comiencen con las líneas:

```
clc
clear all
display('Mensaje sobre el contenido... por ejemplo -Ejercicio 1-')
```

La definición más sencilla de una función en un archivo `function.m` debe tener la siguiente estructura:

```
function salida=NombreFuncion(var1,var2,...)
...
cuerpo de la definición
...
endfunction
```

donde

- *salida* representa el nombre de las variables que devolverán el valor o valores que produce la función. Puede contener varias variables, en cuyo caso debe escribirse en la forma `[f1,f2,...]`. Hay otras posibilidades que iremos aprendiendo.
- *var1* , *var2*,... son las variables de las que dependerá la función.

Con esta sintaxis, las llamadas a la función se harán mediante `NombreFuncion(x1,x2,...)`. Dentro de un archivo de función se pueden definir varias funciones, pero sólo la función principal (la que da nombre al archivo) será accesible desde fuera.

Para definir una función interna a un archivo `script` podemos utilizar varias sintaxis:

- La usual para funciones mostrada antes;
- Definiéndola como un puntero (o *handle*) a una función ya existente: esto se realiza con la sintaxis `@NombreFuncion`, por ejemplo: `f=@sin` establece un puntero denominado `f` para la función `sin` (propia de Octave). Esto permite usar este nombre como argumento en otras funciones, por ejemplo:

  ```
  f=@sin
  quad(f,0,pi)
  ```

  calcula (numéricamente) la integral del seno entr 0 y $\pi$. La última orden podríamos haberla escrito también en la forma `quad(@sin,0,pi)`.

  Después de la orden `f=@sin`, también podemos usar el nombre `f` como función habitual, por ejemplo, en `f(pi/4)`.
- Como función «anónima»: se trata de crear funciones sin nombre, a partir de expresiones sencillas o de otras funciones definidas para utilizarlas en llamadas a otras funciones. La sintaxis para definir una función anónima es:

  ```
  @(var1,var2,...) expresion;
  ```

  con este código queda definida una función anónima de las variables `var1`, etc.

  Podemos asignar un puntero (nombre) a estas funciones, por ejemplo en

  ```
  f=@(x) x.^2;
  ```

  y entonces podemos usar `f` para calcular por ejemplo el cuadrado de 2, mediante `f(2)`, o una integral mediante `quad(f,0,1)`. También podemos evaluarla mediante `feval(f,2)`, y en este caso podríamos usar directamente la función anónima, mediante `feval(@(x) x.^2,2)`.

  Sin embargo esta forma de definir funciones tiene una restricción importante: la definición debe contener sólo una expresión sencilla, ni siquiera puede contener sintaxis complicadas como un bucle o un condicional.

Ahora tendríamos dos formas de «manipular» el seno:

```
quad(@sin,0,pi)
quad(@(x) sin(x),0,pi)
```

la primera de ellas es hasta cinco veces más rápida que la segunda.

En general, y según mi experiencia, el manejo de las funciones es complejo, principalmente por la accesibilidad de un script a las funciones definidas. Iremos tratando de comprender este aspecto...

## 5. Organización de las prácticas

En el curso dedicaremos una carpeta para incluir los ejercicios de cada práctica, como ya comentábamos antes: `Practica1`, `Practica2`,... `EntregaPractica1`, etc.

Todos los archivos `script` y `function` que construyamos, o se proporcionen, como herramientas generales (o de propósito general) para las prácticas se incluirán en la carpeta `Biblioteca`.

Para permitir que Octave encuentre esos archivos debemos incluir la sintaxis

```
addpath('Biblioteca')
```

y conviene terminar el programa con

```
rmpath('Biblioteca')
```

La cadena `'Biblioteca'` en las órdenes anteriores debe complementarse con una indicación de la ubicación de la carpeta (o «`path`»). Si seguimos la estructura indicada antes, lo normal es que tengamos dentro de la carpeta `/CN1V_2023_24/CodigoPracticas` dos carpetas llamadas`Biblioteca` y `Practica1`. Si estamos escribiendo un script en la carpeta `Practica1`, debemos escribir en él:

```
addpath('../Biblioteca')
```

La cadena `'../Biblioteca'` indica que se debe subir un nivel de carpetas y luego bajar un nivel a la carpeta `Biblioteca`. Por otra parte, la orden `addpath('./Biblioteca')` indicaría que buscara en la carpeta `Biblioteca` que está situada en el mismo nivel en que se encuentra situado el script.

El explorador de archivos de Octave funciona realmente como tal explorador: si cambiamos de carpeta dentro de él, el «prompt» del sistema cambia. Así, por ejemplo, si intentamos ejecutar un script situado en la carpeta `Practica1`, pero el explorador indica que estamos en `Biblioteca` (recuerda que en la parte superior de la ventana se indica el «Directorio actual») Octave nos indicará que se debe cambiar de carpeta y nos propondrá hacerlo inmediatamente