Introducción a *Octave*

Facundo Gutiérrez

Elementos de Cálculo Numérico

Resumen

Instalando Octave

- En Windows: https://www.gnu.org/software/octave
- En Linux: En la terminal:

  sudo apt-add-repository ppa:octave/stable

  sudo apt-get update

  sudo apt-get install octave

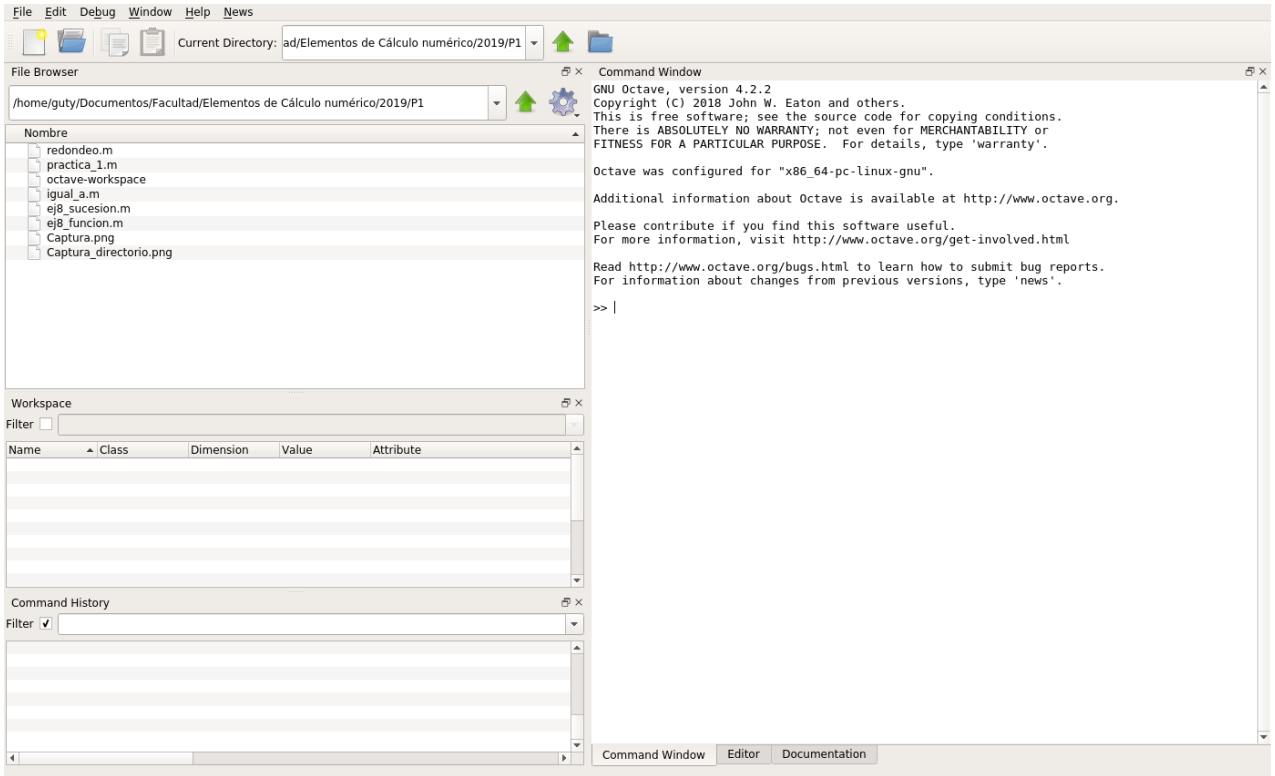
Instalando Octave

- En Windows: https://www.gnu.org/software/octave
- En Linux: En la terminal:

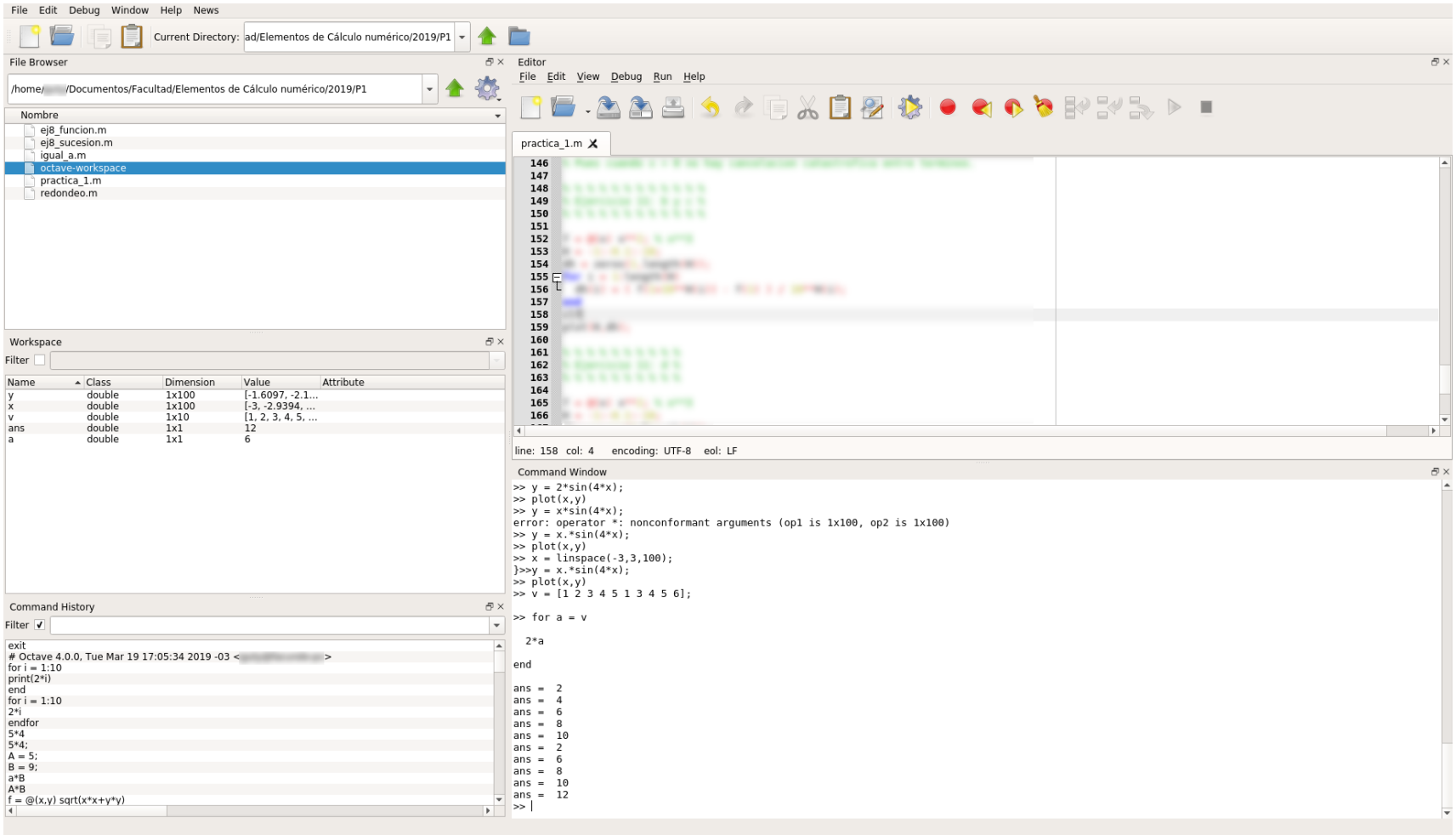
  sudo apt-add-repository ppa:octave/stable

  sudo apt-get update

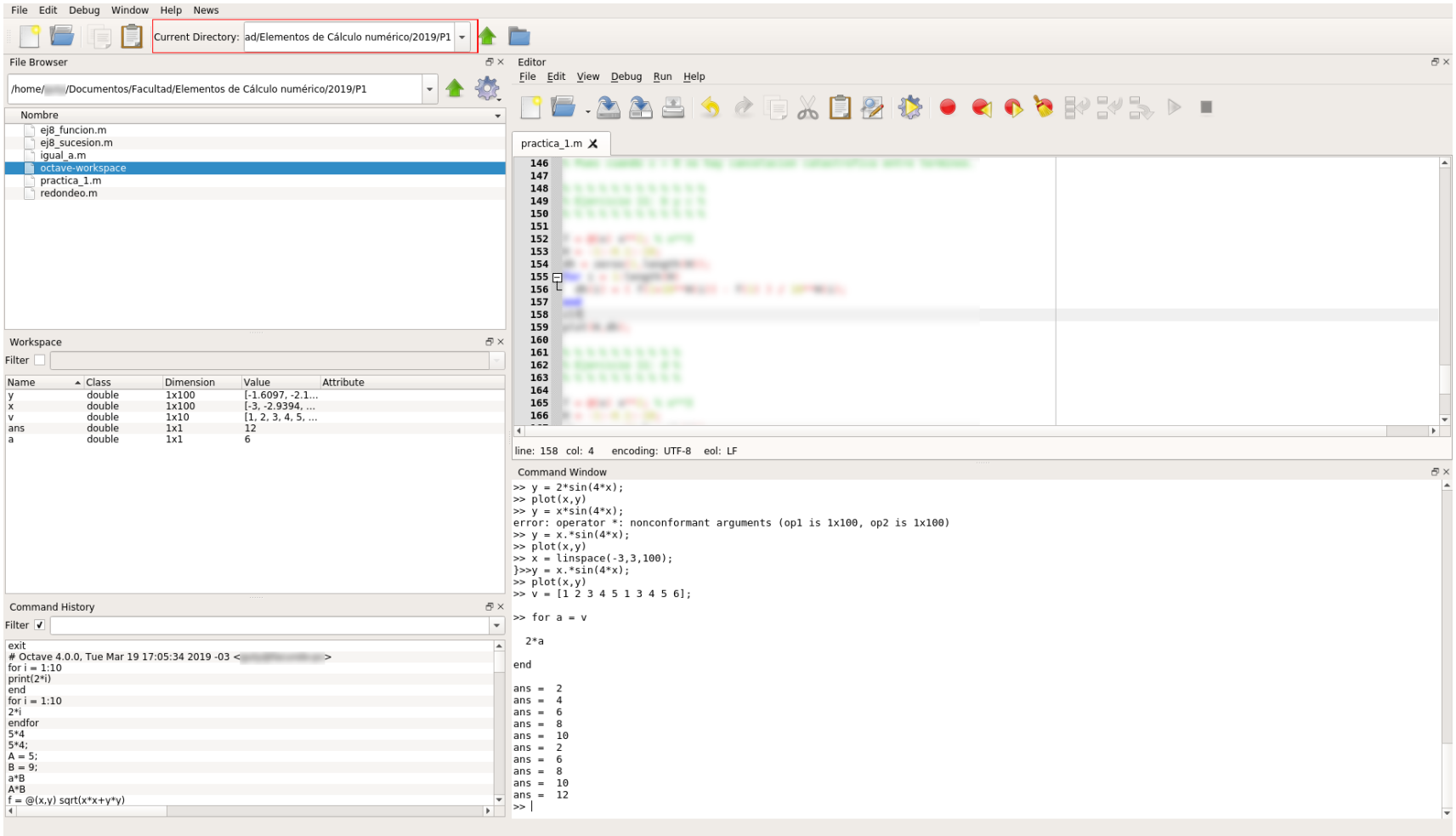
  sudo apt-get install octave



Interfaz gráfica

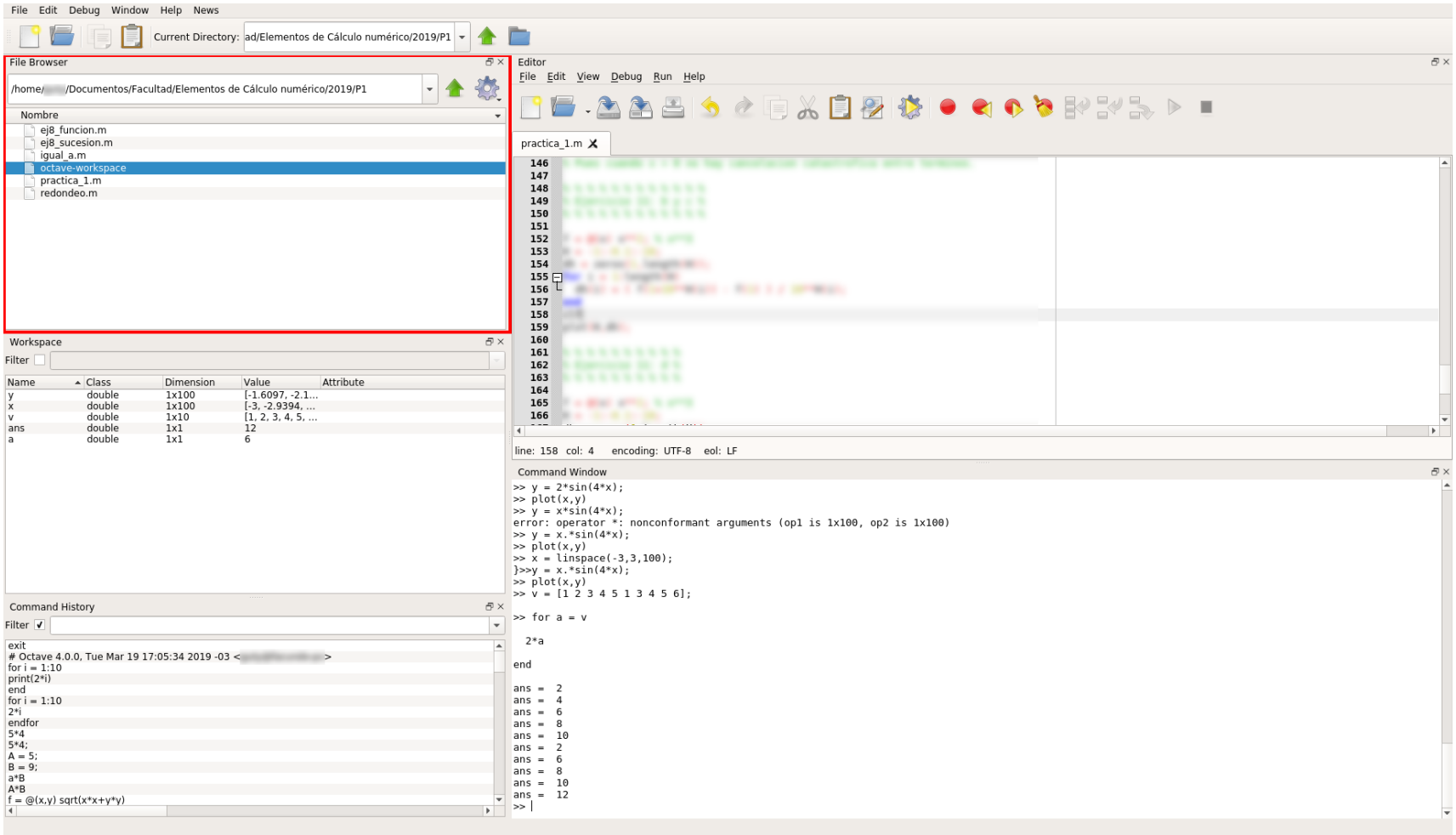


Interfaz gráfica



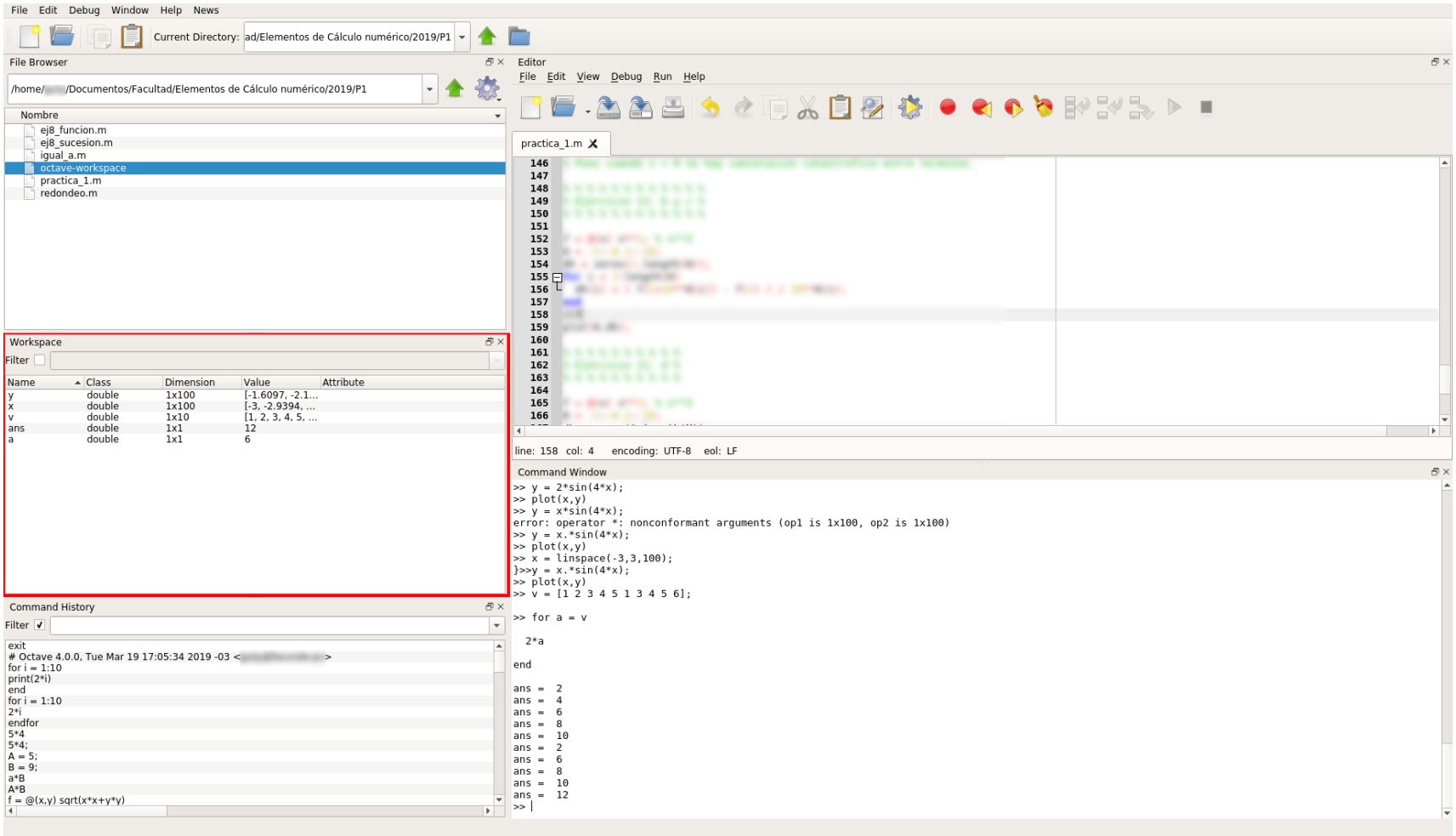
Directorio actual: Indica en qué carpeta estamos trabajando actualmente (importa al trabajar con funciones).

Interfaz gráfica



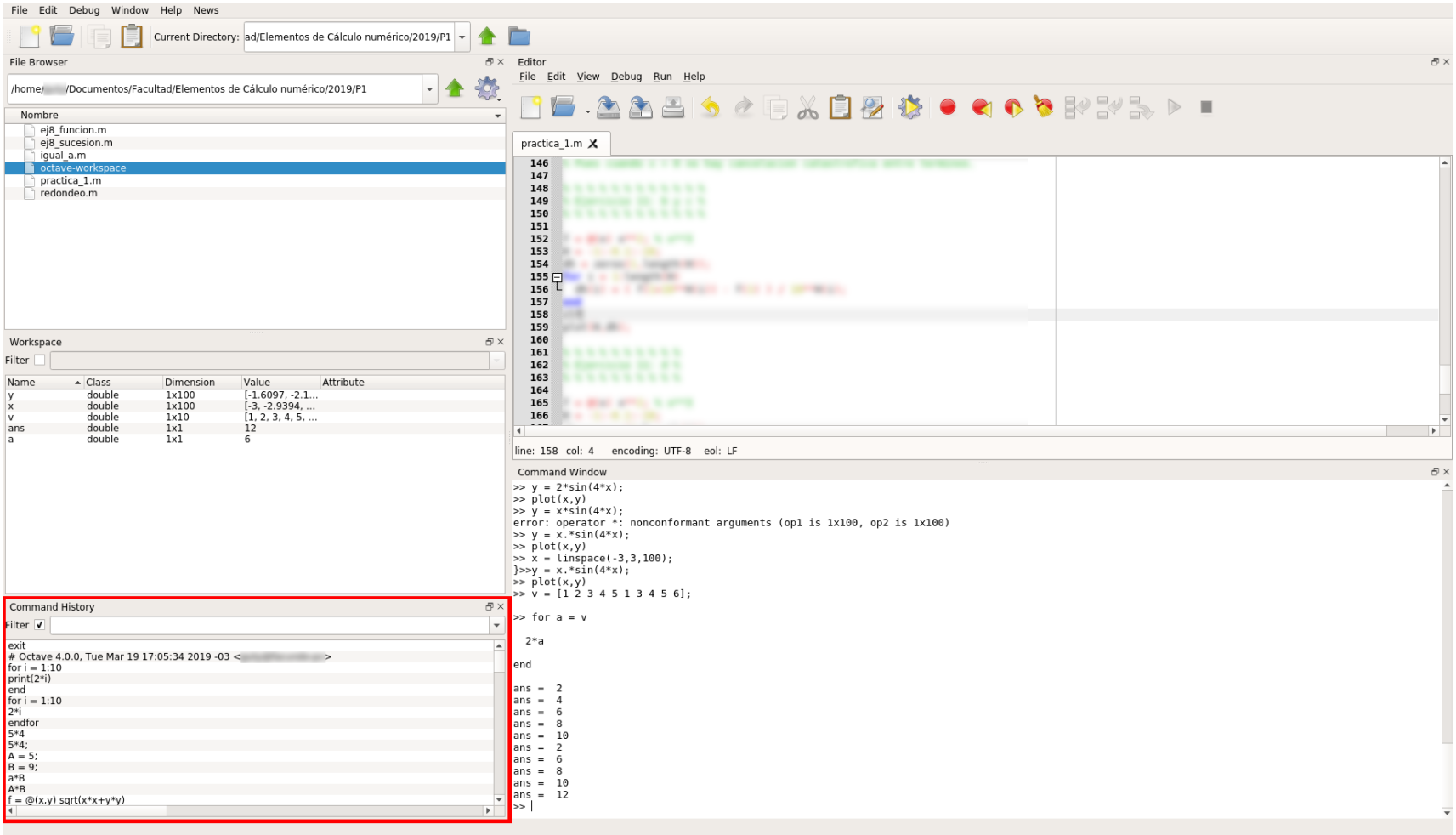
Explorador de archivos: Nos permite ubicarnos en el directorio que deseamos.

Interfaz gráfica



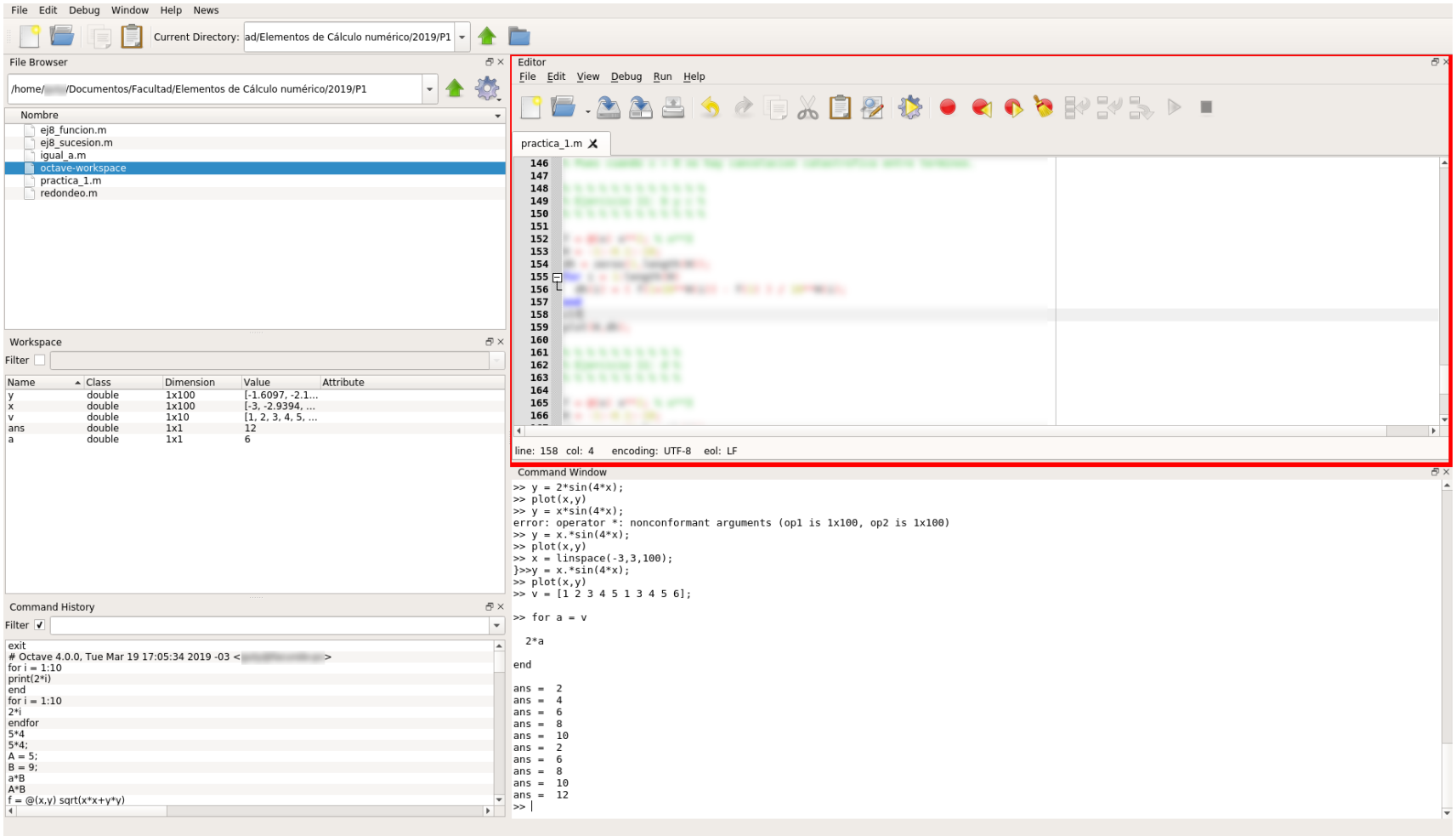
Espacio de trabajo: Nos indica qué variables ya tienen un valor asignado (y su valor).

Interfaz gráfica



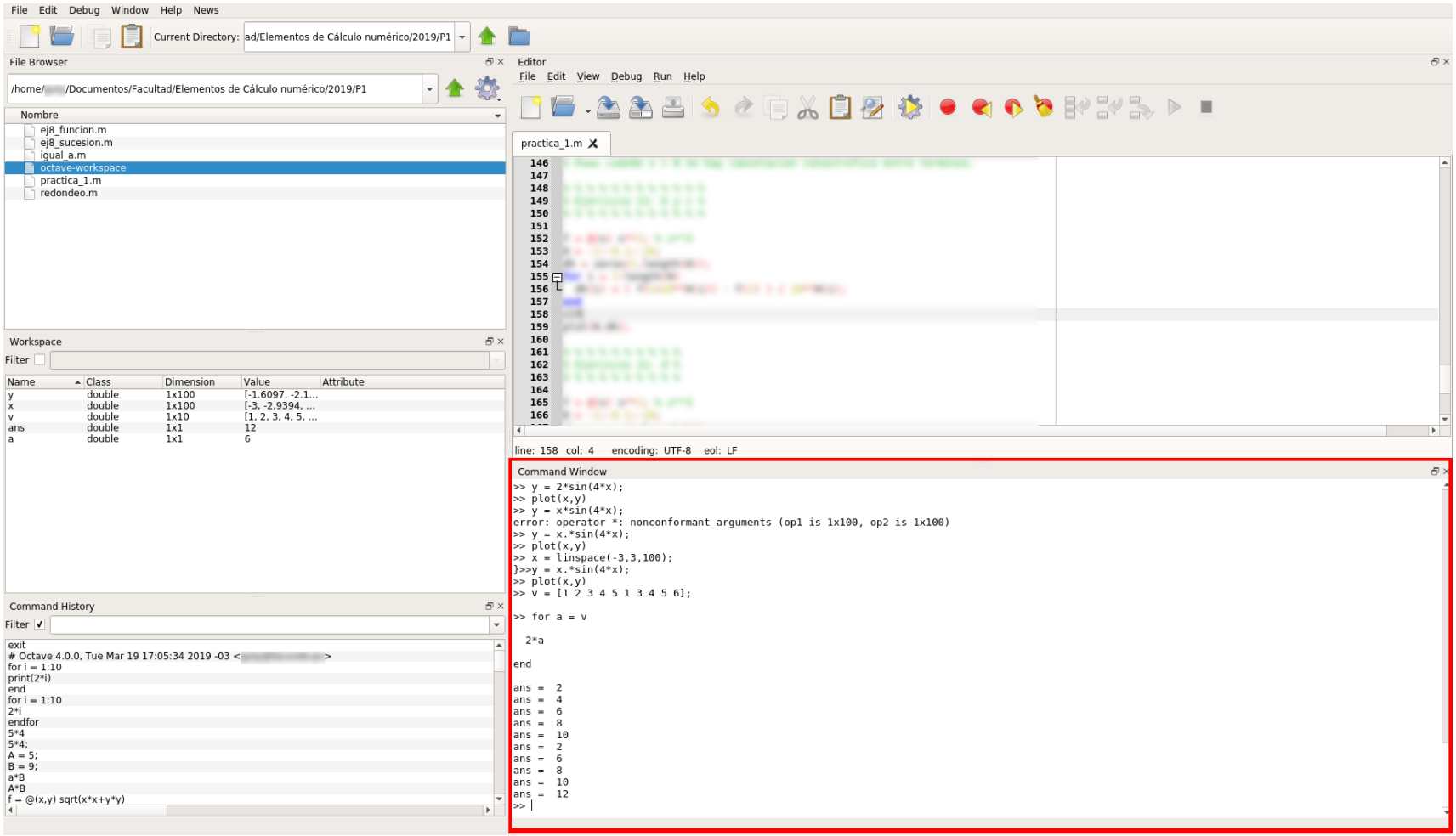
Historial de comandos: Almacena los comandos que fueron ingresados por el usuario.

Interfaz gráfica



Editor: Nos permite escribir listas de comandos que serán ejecutadas secuencialmente.

Interfaz gráfica



Ventana de comandos: Aquí se imprimen los resultados a los comandos que fueron ingresados.

Operadores básicos

- + para la suma

Operadores básicos

- `+` para la suma
- `-` para la resta

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)
- `^` o `**` para la potenciación y `.^` o `.**` para la potenciación *elemento a elemento* (ídem)

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)
- `^` o `**` para la potenciación y `.^` o `.**` para la potenciación *elemento a elemento* (ídem)
- `/` para la división y `./` para la división *elemento a elemento* (ídem)

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)
- `^` o `**` para la potenciación y `.^` o `.**` para la potenciación *elemento a elemento* (ídem)
- `/` para la división y `./` para la división *elemento a elemento* (ídem)

Ingresar en la *ventana de comandos*:

```
(1.5+2)*0.5
```

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)
- `^` o `**` para la potenciación y `.^` o `.**` para la potenciación *elemento a elemento* (ídem)
- `/` para la división y `./` para la división *elemento a elemento* (ídem)

Ingresar en la *ventana de comandos*:

```
(1.5+2)**0.5
```

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)
- `^` o `**` para la potenciación y `.^` o `.**` para la potenciación *elemento a elemento* (ídem)
- `/` para la división y `./` para la división *elemento a elemento* (ídem)

Ingresar en la *ventana de comandos*:

```
(1.5-2)/0.5
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable` $\leftarrow$ `expresion`

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable` $\leftarrow$ `expresion`

Ingresar en la *ventana de comandos*:

```
var_1 = (1.5+2)*0.5
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable` $\leftarrow$ `expresion`

Ingresar en la *ventana de comandos*:

```
var_2 = (1.5+2)**0.5
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable` $\leftarrow$ `expresion`

Ingresar en la *ventana de comandos*:

```
resultado = var_1+var_2
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable` $\leftarrow$ `expresion`

Ingresar en la *ventana de comandos*:

```
format long
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable` $\leftarrow$ `expresion`

Ingresar en la *ventana de comandos*:

```
resultado
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable` $\leftarrow$ `expresion`

Ingresar en la *ventana de comandos*:

```
format short
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable` $\leftarrow$ `expresion`

Ingresar en la *ventana de comandos*:

```
resultado
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0$ tal que $0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
v = [1.5, 2, -3.5, 5]
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
v(3)
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
v+2
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
v*2
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
v**2
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
v.**2
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
u = -1:0.75:1
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
v+u
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
v*u
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
v.*u
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
linspace(-1,2,10)
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
linspace(-1,2,10);
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
1:10
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
v(2:3)
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:
  `v = [1.5, 3.1, -3.4, 4.8]`
  $v \leftarrow (1.5, 3.1, -3.4, 4.8)$
- Usando el operador *dos puntos* (`:`)
  `v = comienzo:paso:final`
  $v_{i+1} = comienzo + i \cdot paso, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{final - comienzo}{paso} \rfloor$
- Usando funciones de *Octave*:
  `v = linspace(comienzo,final,n)`
  $v_{i+1} = comienzo + i \cdot \frac{final - comienzo}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$

Ingresar en la *ventana de comandos*:

```
[v, u]
```

Gráficos

Para graficar una función utilizamos el comando `plot`. En principio toma *dos vectores como argumentos*.

Grafica los *pares ordenados* formados por cada coordenada del primer vector y su correspondiente del segundo vector (deben ingresarse vectores con el mismo tamaño). Sin ninguna opción extra, se interpola linealmente entre pares consecutivos.

Gráficos

Para graficar una función utilizamos el comando `plot`. En principio toma *dos vectores como argumentos*.

Grafica los *pares ordenados* formados por cada coordenada del primer vector y su correspondiente del segundo vector (deben ingresarse vectores con el mismo tamaño). Sin ninguna opción extra, se interpola linealmente entre pares consecutivos.

```
x = linspace(-3,3,100)
```

```
y = x.*sin(4.*x)
```
$y_i = x_i \cdot sen(4x_i)$

```
plot(x,y)
```
Grafica los puntos (x_i, y_i)

Sentencia `for`

Podemos utilizar este comando para recorrer un vector y ejecutar una misma serie de comandos a medida que se recorre el vector (usualmente se utiliza un vector de índices).

Sentencia `for`

Podemos utilizar este comando para recorrer un vector y ejecutar una misma serie de comandos a medida que se recorre el vector (usualmente se utiliza un vector de índices).

```
for  elemento = vector
```

elemento $\in$ *vector*

```
  serie de comandos

end
```

Sentencia `for`

Podemos utilizar este comando para recorrer un vector y ejecutar una misma serie de comandos a medida que se recorre el vector (usualmente se utiliza un vector de índices).

```
for  elemento = vector
```
$elemento \in vector$

```
  serie de comandos

end
```

Si tenemos varios vectores de largo n, un patrón común es:

```
for  i = 1:n
```
$i \in (1, \ldots, n)$

```
  serie de comandos

end
```

Expresiones lógicas

- $\boxed{<}$ menor que $2 < 3 \rightarrow 1(\mathit{True})$

Expresiones lógicas

- $\boxed{<}$ menor que $2 < 3 \rightarrow 1(\mathit{True})$
- $\boxed{>}$ mayor que $2 > 3 \rightarrow 0(\mathit{False})$

Expresiones lógicas

- `<` menor que $2 < 3 \rightarrow 1(True)$
- `>` mayor que $2 > 3 \rightarrow 0(False)$
- `<=` menor o igual $2 <= 3 \rightarrow 1(True)$

Expresiones lógicas

- `<` menor que $2 < 3 \rightarrow 1(True)$
- `>` mayor que $2 > 3 \rightarrow 0(False)$
- `<=` menor o igual $2 <= 3 \rightarrow 1(True)$
- `>=` mayor o igual $2 >= 3 \rightarrow 0(False)$

Expresiones lógicas

- `<` menor que $2 < 3 \rightarrow 1(True)$
- `>` mayor que $2 > 3 \rightarrow 0(False)$
- `<=` menor o igual $2 <= 3 \rightarrow 1(True)$
- `>=` mayor o igual $2 >= 3 \rightarrow 0(False)$
- `==` igual que $2 == 3 \rightarrow 0(False)$

Expresiones lógicas

- `<` menor que $2 < 3 \rightarrow 1(True)$
- `>` mayor que $2 > 3 \rightarrow 0(False)$
- `<=` menor o igual $2 <= 3 \rightarrow 1(True)$
- `>=` mayor o igual $2 >= 3 \rightarrow 0(False)$
- `==` igual que $2 == 3 \rightarrow 0(False)$
- `~=` distinto que $2 \sim= 3 \rightarrow 1(True)$

Expresiones lógicas

- `<` menor que $2 < 3 \rightarrow 1(True)$
- `>` mayor que $2 > 3 \rightarrow 0(False)$
- `<=` menor o igual $2 <= 3 \rightarrow 1(True)$
- `>=` mayor o igual $2 >= 3 \rightarrow 0(False)$
- `==` igual que $2 == 3 \rightarrow 0(False)$
- `~=` distinto que $2 \sim= 3 \rightarrow 1(True)$

Si queremos expresar que varias condiciones ocurran simultáneamente (conocido como el operador `and`), utilizamos `&` entre las condiciones.

Si queremos expresar que solo alguna de varias condiciones valga (conocido como el operador `or`), utilizamos `|` (*barra vertical*) entre las condiciones.

Si queremos negar una condición, podemos utilizar `~`

Sentencia `if`

Nos sirve para ejecutar una serie de comandos específica en el caso de que ocurra la condición que queremos.

Sentencia `if`

Nos sirve para ejecutar una serie de comandos específica en el caso de que ocurra la condición que queremos.

```
if condicion_1
  serie de comandos 1
elseif condicion_2    opcional
  serie de comandos 2
else    opcional
  serie de comandos 3
end
```

Sentencia `while`

Mientras que se cumpla una cierta condición se ejecutará una serie de comandos específica.

Sentencia `while`

Mientras que se cumpla una cierta condición se ejecutará una serie de comandos específica.

```
while  condicion

  serie de comandos

end
```

Sentencia `while`

Mientras que se cumpla una cierta condición se ejecutará una serie de comandos específica.

```
while  condicion

  serie de comandos

end
```

Observación: Al salir del `while` necesariamente deja de valer `condicion`

Funciones de una línea

Para no repetir una misma expresión cambiando solo unos parámetros dentro de ella podemos utilizar funciones.

Funciones de una línea

Para no repetir una misma expresión cambiando solo unos parámetros dentro de ella podemos utilizar funciones.

```
f = @(x) x.*sin(4.*x)
```

$f(x) = x \cdot sen(4x)$

Funciones de una línea

Para no repetir una misma expresión cambiando solo unos parámetros dentro de ella podemos utilizar funciones.

```
f = @(x) x.*sin(4.*x)
```
$f(x) = x \cdot sen(4x)$

También funciona si tenemos más de un parámetro.

Funciones de una línea

Para no repetir una misma expresión cambiando solo unos parámetros dentro de ella podemos utilizar funciones.

```
f = @(x) x.*sin(4.*x)
```
$f(x) = x \cdot sen(4x)$

También funciona si tenemos más de un parámetro.

```
f = @(x,y,z) sqrt(x.**2 + y.**2 + z.**2)
```
$f(x, y, z) = \sqrt{x^2 + y^2 + z^2}$

Funciones creadas por el usuario

Si tenemos que crear una función más sofisticada, que realiza varias operaciones con los parámetros para llegar al resultado, es conveniente crear una función en un archivo, por ejemplo `nombre.m`.

Funciones creadas por el usuario

Si tenemos que crear una función más sofisticada, que realiza varias operaciones con los parámetros para llegar al resultado, es conveniente crear una función en un archivo, por ejemplo `nombre.m`.

El archivo `nombre.m` deberá tener la siguiente forma:

```
function [res_1,res_2,...,res_k] = nombre(arg_1,...,arg_m)

  serie de comandos que puede usar arg_1,...,arg_m

end
```

Funciones creadas por el usuario

Si tenemos que crear una función más sofisticada, que realiza varias operaciones con los parámetros para llegar al resultado, es conveniente crear una función en un archivo, por ejemplo `nombre.m`.

El archivo `nombre.m` deberá tener la siguiente forma:

```
function [res_1,res_2,...,res_k] = nombre(arg_1,...,arg_m)

  serie de comandos que puede usar arg_1,...,arg_m

end
```

Al finalizar de correr la serie de comandos secuencialmente, retornará los valores que estén guardados en las variables $\texttt{res}_1,\ldots,\texttt{res}_k$.

Para utilizar una función, debemos **estar situados en la misma carpeta donde está guardada** (ya sea desde otro archivo en el *editor* o en la *ventana de comandos*), y llamarla de la siguiente forma:

```
[var_1,var_2,...,var_k] = nombre(arg_1,...,arg_m)
```

Funciones creadas por el usuario

Si tenemos que crear una función más sofisticada, que realiza varias operaciones con los parámetros para llegar al resultado, es conveniente crear una función en un archivo, por ejemplo `nombre.m`.

El archivo `nombre.m` deberá tener la siguiente forma:

```
function [res_1,res_2,...,res_k] = nombre(arg_1,...,arg_m)

   serie de comandos que puede usar arg_1,...,arg_m

end
```

Al finalizar de correr la serie de comandos secuencialmente, retornará los valores que estén guardados en las variables $\texttt{res}_1,\ldots,\texttt{res}_k$.

Para utilizar una función, debemos **estar situados en la misma carpeta donde está guardada** (ya sea desde otro archivo en el *editor* o en la *ventana de comandos*), y llamarla de la siguiente forma:

$[\texttt{var}_1,\texttt{var}_2,\ldots,\texttt{var}_k] = \texttt{nombre}(\texttt{arg}_1,\ldots,\texttt{arg}_m)$ **Importante**.

Algunos tips

- Trabajar en el editor.
- Podemos agregar comentarios con el símbolo `%` o el símbolo `#`.
- Guardar el trabajo continuamente (`CTRL + S` guarda directamente).
- Si tipeamos `help nombre_funcion` *Octave* nos detalla cómo utilizar la función en cuestión.
- Podemos ejecutar un conjunto de líneas seleccionadas apretando `F9`.
- Podemos guardar y ejecutar un script del editor con la tecla `F5`.
- Cuando algo que escribimos no funciona *Octave* nos detalla por qué (en la *ventana de comandos*).
- En la ventana de comandos, usando la flecha `↑` podemos visitar los últimos comandos ingresados.
- Para no escribir *10^38, podemos escribir e38 (38 es un ejemplo).
- Con `CTRL + R` podemos comentar la línea actual, y descomentarla con `CTRL + SHIFT + R`.
- Indentar correctamente (alinear correctamente dónde comienza y termina cada sentencia) es muy importante.

Referencias

John W. Eaton, David Bateman, Soren Hauberg, Rik Wehbring (2019)

Documentación oficial

https://octave.org/doc/octave-5.1.0.pdf

José María Valiente Cifuentes (2006)

Manual de iniciación a GNU Octave

http://softlibre.unizar.es/manuales/aplicaciones/octave/manual_octave.pdf