# 21705 - Métodos del Álgebra Lineal

## Introducción a la programación en Octave

Departamento de Ciencias Matemáticas e Informática

Universidad de las Islas Baleares

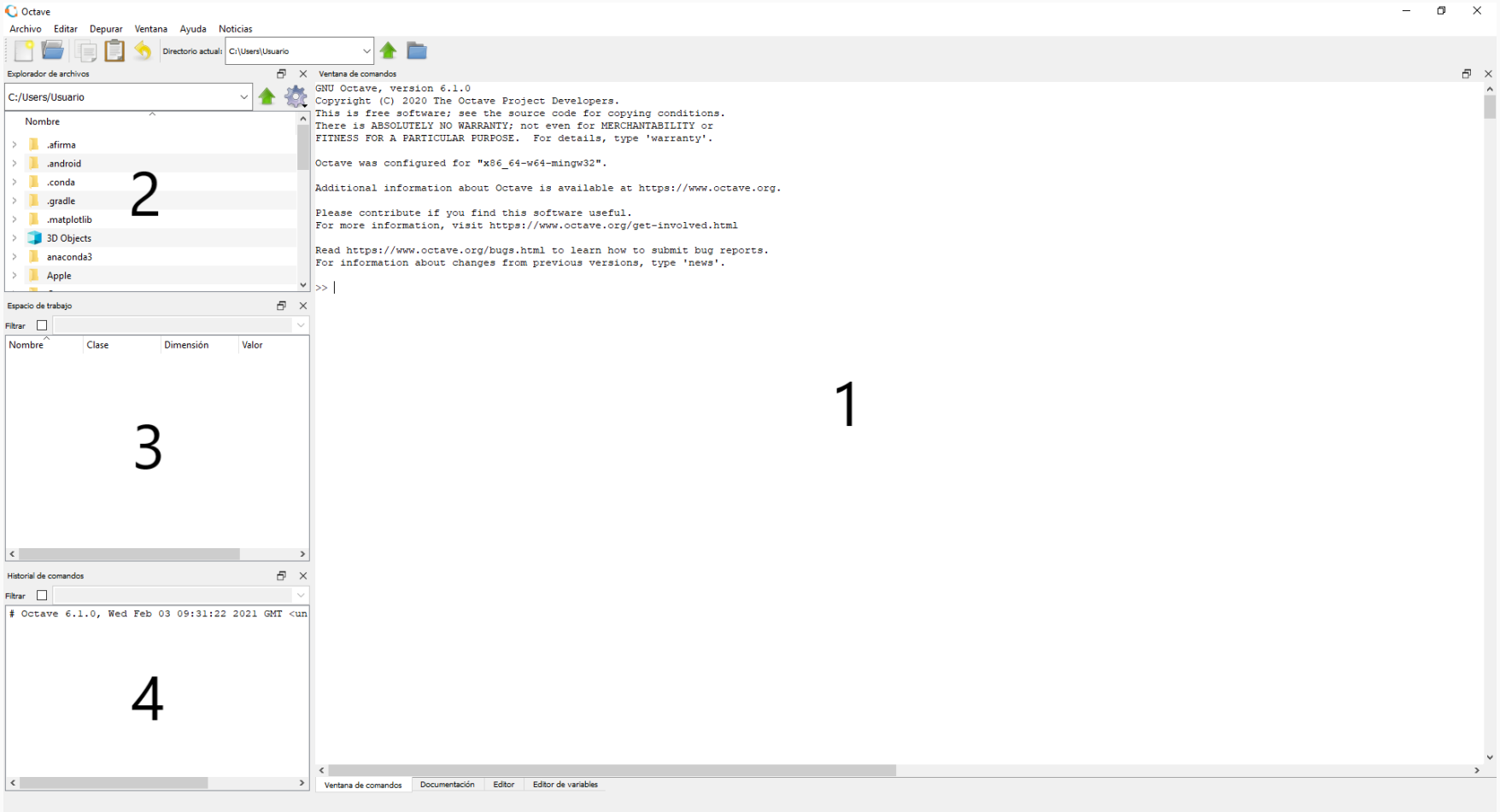
# Table of contents

# Instalación y uso

# Instalación de Octave

- Octave: Versión *open source* de Matlab.
- Descargar desde la fuente oficial: *https://www.gnu.org/software/octave/download.html*
- Seleccionar la versión de cada sistema operativo.

# Editor de código



# Editor de código

Elementos que aparecen en el IDE de Octave:

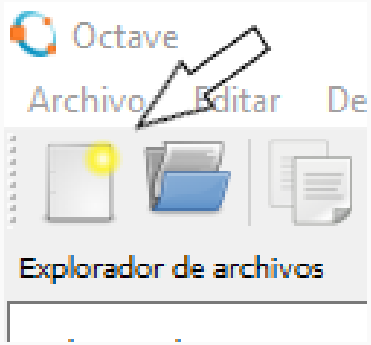
1. Terminal de código. Podemos escribir código libre, como llamar a una función que ya tengamos definida o hacer alguna prueba corta.
2. Explorador. Nos permite fijar la carpeta de trabajo (recomendable seleccionar la carpeta donde tengamos el script). Para ello:
    - Seleccionar carpeta o escribir la ruta que contiene nuestra carpeta de trabajo..
    - Click derecho sobre la carpeta de trabajo, y seleccionar *Add to path / Selected directories and Subdirectories*.

# Editor de código

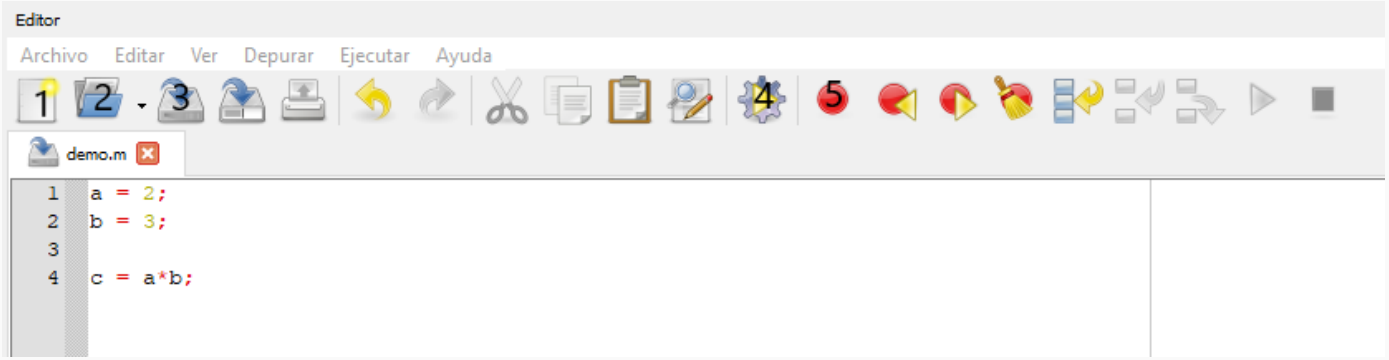
3. Registro de variables. Aparecen todas las variables que hayamos definido en la terminal o en el script que ejecutemos.
    - Importante: **No** aparecen las variables de las diferentes funciones que llamemos desde el script principal (el que ejecutamos). Si queremos verificar el comportamiento de una función, habrá que hacer *debug*.
4. Historial. Podemos consultar todos los comandos que se han ejecutado.

# Creación de un script

- En la interfaz principal, seleccionamos la siguiente opción:




- Nos aparecerá el editor de código:



# Creación de un script

1. Crear un nuevo script.
2. Abrir un script ya existente.
3. Guardar el script actual.
4. Ejecutar el script actual.
    - Importante: Cualquier script que ejecutemos debe estar previamente guardado. En caso contrario, Octave nos pedirá que lo guardemos.
5. Añadir un *breakpoint*. Poniendo el cursor de texto sobre una línea de código, al seleccionar esta opción añade una parada; es decir, cuando se ejecute el script se parará en esa línea en concreto, y nos dejará ver las variables.

# Declaración de variables y operaciones elementales

# Declaración de variables y operaciones

- A diferencia de otros lenguajes (como Java), Octave no es *tipado*. Automáticamente elije el tipo de la variable.
- Cada línea de código debe terminar en punto y coma.
  - No es obligatorio, pero tiene más rendimiento.
  - El punto y coma evita que por detrás esté imprimiendo cada resultado de cada línea.

# Declaración de variables y operaciones

- Variables numéricas.

```
a = 2;
b = 3;

sum = a+b;
subtr = a-b;
mult = a*b;
div = a/b;
pow = a**b;
modul = mod(b, a) %Para calcular b mod a.
```

- Variables lógicas.

```
t = true;
f = false;
and = t & f;
or = t | f;
```

# Declaración de variables y operaciones

- Declaración de matrices. Se declara como una lista, separando las filas por punto y coma.
  Por ejemplo, las matrices

$$A = \begin{bmatrix} 3 & 2 & 1 \\ 5 & 3 & 4 \\ 1 & 1 & -1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 3 \\ 2 & 2 \\ 4 & 5 \end{bmatrix}$$

  se declaran como:

```
A = [3, 2, 1; 5, 3, 4; 1, 1, -1];
B = [1, 3; 2, 2; 4, 5];
```

# Declaración de variables y operaciones

Podemos realizar operaciones con las matrices:

- Multiplicación por escalar:

```
>> 2*A
ans =
     6     4     2
    10     6     8
     2     2    -2
```

- Suma de matrices:

```
>> A+2*A'
ans =
     9    12     3
     9     9     6
     3     9    -3
```

- Matriz transpuesta:

```
>> A'
ans =
     3     5     1
     2     3     1
     1     4    -1
```

- Producto de matrices:

```
>> A*B
ans =
    11    18
    27    41
    -1     0
```

# Declaración de variables y operaciones

Podemos realizar operaciones con las matrices:

- Obtener la matriz identidad:

```
>> eye(3)
ans=
     1     0     0
     0     1     0
     0     0     1
```

- Determinante:

```
>> det(A)
ans = -1.0000
```

- Rango de la matriz:

```
>> rank(A)
ans = 3
```

- Matriz inversa (si existe):

```
>> inv(A)
ans =
    7.000   -3.000   -5.000
   -9.000    4.000    7.000
   -2.000    1.000    1.000
```

- Dimensiones de la matriz:

```
>> [m, n] = size(A)
m = 3
n = 3
```

## Ejercicio resuelto

Implementar un código que permita resolver el sistema de ecuaciones:

$$\begin{bmatrix} 3 & 2 & 1 \\ 5 & 3 & 4 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

## Solución

Primero de todo, declaramos las dos variables que harán falta: la matriz del sistema y el vector independiente.

**Importante:** Hay que tener presente que el vector independiente es un vector columna, y por lo tanto hay que construirlo como tal.

## Solución

Si $A \cdot x = b$, y $A$ es invertible, entonces una manera de calcular las soluciones es $x = A^{-1} \cdot b$. Así, la solución es:

```
system_matrix = A;
indep_term = [1; 2; 1]
if det(system_matrix) != 0
    solution = inv(system_matrix)*indep_term;
    disp(x)
else
    disp("The system can not be solved")
endif
```

## Ejercicio resuelto

Dado un sistema de ecuaciones $A \cdot x = b$, implementar un código para clasificar el sistema según el número de soluciones.

## Solución

Tengamos presente el teorema de Rouché-Frobenius para resolver el ejercicio. Primero, calculemos la matriz ampliada, añadiendo el vector independiente como columna.

```
aug_matrix = [system_matrix, indep_term]
[m, number_unknowns] = size(system_matrix);
```

Y ahora discutamos los rangos.

## Solución

```
if ((rank(aug_matrix) == rank(system_matrix)) &&
    (rank(system_matrix) == number_unknowns))
  disp("System independent with one solution");

elseif ((rank(aug_matrix) == rank(system_matrix)) &&
        (rank(system_matrix) < number_unknowns))
  disp("System independent with infinite solutions")

elseif (rank(aug_matrix) != rank(system_matrix))
  disp("System inconsistent");

endif
```

# Iteraciones sobre matrices

- Supongamos que queremos iterar sobre una matriz. En Java, tenemos los bucles *for*:

```
for (int i = 0; i < 10; i++) {
    //Do something
}
```

  El equivalente en Octave es:

```
for i = 0:9
    //Do something
endfor
```

- **Importante**: En Java las listas (y por tanto las matrices) se indexan a partir del cero. En Octave no, hay que empezar en 1. Esto es una ventaja, ya que así se sigue la notación usual que usamos en álgebra.

# Iteraciones sobre matrices

## Ejercicio resuelto

Implementar un código que determine si una matriz es simétrica o no.

## Solución

Para que una matriz sea simétrica, primero comprobaremos si es cuadrada o no.

```
if m == n
    % Check if matrix is symmetric.
else
  disp("The matrix entered is not a square matrix.")
endif
```

## Solución

Ahora, una matrix $A = (a_{ij})$, con $i, j = 1, \ldots, n$ es simétrica si $a_{ij} = a_{ji}$. Por lo tanto, bastará comprobar si no se cumple la igualdad.

```
for i=1:m
    for j=1:n
        if matrix(i,j) != matrix(j,i)
            disp("The matrix is not symmetric");
        endif
    endfor
endfor
```

## Solución

Otra solución es considerar directamente la transpuesta de la matriz. Una matriz es simétrica si $A = A^t$ o, equivalentemente, cuando $A - A^t = 0$. Recurriremos a dos funciones de Octave:

- `isequal`: Comprueba si dos matrices son iguales.
  - Importante: Hay que cogerlo con pinzas. A veces errores pequeños de redondeo harán que matrices iguales den como resultado que no son iguales.
- `zeros(m, n)`: Genera una matriz nula de dimensión $m \times n$.

```
if isequal(matrix-matrix', zeros(m, n))
    disp("The matrix is symmetric")
else
    disp("The matrix is not symmetric")
endif
```

# Declaración de funciones

# Declaración de funciones

- En programación, la reutilización de código es fundamental para un código limpio.
- Esto se consigue usando funciones que realicen tareas por separado.
- La estructura básica de una función es:

```
function value_to_return = function_name(args)
    % Do something...
    value_to_return = something
    return;
endfunction
```

# Declaración de funciones

- Por ejemplo, para determinar si una matriz es simétrica:

```
function is_symmetric = check_symmetry(matrix)
  [m, n] = size(matrix);
  if m == n
    if isequal((matrix-matrix'),zeros(m, n))
      is_symmetric = true;
      return;
    else
      is_symmetric = false;
      return;
    endif
  else
    is_symmetric = false;
    return;
  endif
endfunction
```

# Indexación de matrices

# Indexación de matrices

- Supongamos que necesitamos elegir todos los valores de una fila o columna de una matriz.
- Posible solución: fijado $k$, iterar sobre todos los $i$ para elegir $a_{ki}$ o $a_{ik}$.
- Alternativa: Octave permite, de forma rápida y con alto rendimiento, elegir un subconjunto de una matriz.
- La sintaxis es `matrix(a:b, c:d)`; es decir:
  - La submatriz de `matrix` cuyas filas están entre la fila `a` y `b`, y cuyas columnas están entre la columna `c` y `d`.
  - Si queremos coger toda una fila, y un determinado rango de columnas, la sintaxis es `matrix(:, c:d)`.

# Indexación de matrices

- Si queremos coger filas o columnas sueltas, la sintaxis es `matrix([a1,a2,...], [b1,b2,...])`.
- A veces queremos coger todas las filas excepto una (por ejemplo, calculando adjuntos). Para ello:
  1. Generamos una lista con los números desde 1 hasta $k$:

     ```
     range = 1:k;
     ```

  2. Eliminamos el elemento $i$ de esa lista:

     ```
     range(i) = [];
     ```

  3. Indexamos la matriz por esa lista.

# Indexación de matrices

- Por ejemplo, consideremos la matriz

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 \\ 31 & 32 & 33 & 34 & 35 & 36 \end{bmatrix}$$

- `A(2:4, 3:5)` genera la submatriz

$$\begin{bmatrix} 9 & 10 & 11 \\ 15 & 16 & 17 \\ 21 & 22 & 23 \end{bmatrix}$$

# Indexación de matrices

- `A(4:6, 4:6)` genera la submatriz

$$\begin{bmatrix} 22 & 23 & 24 \\ 28 & 29 & 30 \\ 34 & 35 & 36 \end{bmatrix}$$

- `A(2:4, :)` genera la submatriz:

$$\begin{bmatrix} 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix}$$

- `A(2, :)` genera la submatriz (o vector fila):

$$\begin{bmatrix} 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix}$$

# Modificación de matrices por bloques

- Supongamos que necesitamos copiar una matriz dentro de una matriz más grande.
- Igual que el caso anterior: dos bucles *`for`*, uno dentro de otro, para realizar la copia.
- Alternativa: Usar la indexación de matrices para hacer la copia.

# Modificación de matrices por bloques

- Consideremos la misma matriz $A$ del ejemplo anterior.
- Para copiar la matriz identidad entre las filas 2 y 5, y entre las columnas 3 y 6, haremos:

```
A(2:4, 3:5) = eye(3);
```

Para generar la matriz

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 1 & 0 & 0 & 12 \\ 13 & 14 & 0 & 1 & 0 & 18 \\ 19 & 20 & 0 & 0 & 1 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 \\ 31 & 32 & 33 & 34 & 35 & 36 \end{bmatrix}$$

# Modificación de matrices por bloques

- Naturalmente, las dimensiones de la matriz que vayamos a copiar deben concordar con las dimensiones de destino.
- En caso contrario, Octave nos avisará que esa operación no está bien definida.

# Ejercicios propuestos

# Ejercicios propuestos

1. Resolver la ecuación matricial $2X + 4A = 3BA$, donde

$$A = \begin{bmatrix} 0 & -1 \\ 2 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

2. Resolver la ecuación matricial $AXB^t = C$, donde

$$A = \begin{bmatrix} -3 & -2 \\ 3 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 5 & 3 \\ 9 & 4 \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

3. Encontrar dos matrices, $A$ y $B$, tales que

$$2A + B = \begin{bmatrix} 1 & 2 & 2 \\ -2 & 1 & 0 \end{bmatrix}$$

$$A - 3B = \begin{bmatrix} -4 & -3 & -2 \\ -1 & 0 & -1 \end{bmatrix}$$

# Ejercicios propuestos

4. La función *rand(m, n)* de Octave genera aleatoriamente una matriz de tamaño $m \times n$. Usadla para generar un sistema de ecuaciones $5 \times 5$ (generando dos matrices, la del sistema y el vector independiente). Después, resolved el sistema y comprobad que la solución que os da es, efectivamente, una solución.
5. Dada la matriz $A$ de tamaño $6 \times 6$ del ejemplo anterior, calculad la matriz transpuesta usando la indexación de matrices. Verificad que el resultado es la matriz transpuesta.
6. Generad aleatoriamente una matriz $A$ de tamaño $6 \times 6$, y calculad la matriz de adjuntos.

Las soluciones estarán publicadas próximamente en el repositorio *https://github.com/mmunar97/21705-LinearAlgebra*