Compresión y Recuperación de Información Multimedia

Guión 00.2

Estructuras de Datos en Matlab

Objetivo

Aprender a crear, inicializar y acceder a algunas de las estructuras de datos más usadas en MATLAB.

Objectivos

- Aprender a usar MATLAB en cálculo básico utilizando variables, matrices, vectores.
- Explorar las celdas de matrices multidimensionales
- Recordar las operaciones matriciales
- Aprender el uso de estructuras de datos MATLAB.
- Explorar funciones útiles que pueder utilizarse con las estructuras de datos de MATLAB

Contenidos

Paso 1

Ejecuta las siguientes líneas de código, una después de otra, en la *Ventana de Comandos* (órdenes) para ver como podemos utilizar MATLAB como una calculadora

```
2 + 3
ans =

     5

2*3 + 4*5 + 6*7
ans =

    68
```

¿Quién es la variable `ans` y para qué se usa?

¿Qué ocurre si usamos ; al final de cada orden?

Paso 2

Realiza cálculos utilizando variables

```
fruit_per_box = 20; num_of_boxes = 5;
```

```
total_num_of_fruit = fruit_per_box * num_of_boxes
total_num_of_fruit =

   100
```

Crea tus variables y comprueba si MATLAB es sensible a mayúsculas y minúsculas.
¿Cuál es el objetivo de las variables pi, eps, inf, i?. ¿Podemos sobreescribirlas?.¿Y devolverles su valor original?

Paso 3

Ejecuta las órdenes *who* y *whos*, una tras otra, para ver su función y la diferencia entre ellas.

```
who
Your variables are:

A                    Y_diag               ans
total_num_of_fruit
B                    Y_inv                fruit_per_box
X                    Y_t                  my_images
Y                    Y_trace              num_of_boxes
Y_det                Z                    num_of_images
```

```
whos
  Name                  Size              Bytes  Class     Attributes

  A                     4x3x2               192  double
  B                     5x2x3               240  double
  X                     1x4                 608  cell
  Y                     3x3                  72  double
  Y_det                 1x1                   8  double
  Y_diag                3x1                  24  double
  Y_inv                 3x3                  72  double
  Y_t                   3x3                  72  double
  Y_trace               1x1                   8  double
  Z                     1x3                  24  double
  ans                   1x1                   8  double
  fruit_per_box         1x1                   8  double
  my_images             1x2                 924  struct
  num_of_boxes          1x1                   8  double
  num_of_images         1x1                   8  double
  total_num_of_fruit    1x1                   8  double
```

Paso 4

Limpia una imagen del espacio de trabajo. Una vez ejecutado, observa como ha desaparecido la variable del *Workspace*.

```
clear fruit_per_box
```

Paso 5

Limpia la *Ventana de Comandos* y todas las variables del *Espacio de Trabajo* con las siguientes líneas de código (una tras otra para ver su efecto).

```
clc
clear all
```

Paso 6

Crea una matriz 3×3 ejecutado el código siguiente.

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

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

¿Para qué sirve aquí el ; ?

Paso 7

Un operador muy útil en MATLAB son los dos puntos (:). Podemos usarlo para crear vectores de números.

```
1:5
ans =

     1     2     3     4     5
```

Paso 8

Un tercer parámetro (el central) determina como contar entre los números primero y último.

```
1:1:5
ans =

     1     2     3     4     5

5:-1:1
ans =

     5     4     3     2     1

1:2:9
ans =

     1     3     5     7     9

9:-2:1
ans =

     9     7     5     3     1
```

¿Cómo generamos un vector de valores de 0 a pi con distancia pi/4?

Paso 9

El operador : puede utilizarse también para devolver una fila o columna completa de una matriz.

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

     1     2     3
     4     5     6
     7     8     9

A(:,1)
ans =

     1
     4
     7

A(1,:)
ans =

     1     2     3
```

Usa : para generar matriz anterior (cada una de sus filas)

Paso 10

El operador : puede sustituirse por la función *colon* que realiza la misma operación.

```
colon(1,5)
ans =

     1     2     3     4     5
```

Paso 11

Ejecuta esta orden para ver como funciona *linspace*.

```
linspace(pi/4,pi,4)
ans =

    0.7854    1.5708    2.3562    3.1416
```

Paso 12

Compara el resultado del paso previo con estos valores.

```
pi/4
ans =

    0.7854

pi/2
ans =
```

```
    1.5708
```

```
3*pi/4
ans =

    2.3562
```

```
pi
ans =

    3.1416
```

Paso 13

Ejecuta las siguientes líneas de código, una después de otra:

```
zeros(3,4)
ans =

     0     0     0     0
     0     0     0     0
     0     0     0     0
```

```
ones(3,4)
ans =

     1     1     1     1
     1     1     1     1
     1     1     1     1
```

```
ones(3,4) * 10
ans =

    10    10    10    10
    10    10    10    10
    10    10    10    10
```

```
rand(3,4)
ans =

    0.2399    0.4899    0.7127    0.0596
    0.8865    0.1679    0.5005    0.6820
    0.0287    0.9787    0.4711    0.0424
```

```
randn(3,4)
ans =

   -2.3193    0.4115   -0.6912    0.8261
    0.0799    0.6770    0.4494    0.5362
   -0.9485    0.8577    0.1006    0.8979
```

¿Cuál es la diferencia entre `rand` y `randn`?

Paso 14

Combina tres vectores individuales en una matriz 3×3.

```
X = [1 2 3]; Y = [4 5 6]; Z = [7 8 9];
A = [X;Y;Z]
B = cat(1,X,Y,Z)
```
```
A =

     1     2     3
     4     5     6
     7     8     9


B =

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

Paso 15

Borra la última fila (fila 3) de la matriz *A*. Observa que el operador : se utiliza para especificar la fila completa.

```
A(3,:) = []
```
```
A =

     1     2     3
     4     5     6
```

Paso 16

Utiliza las funciones *ones* y *rand* para crear matrices multimensionales.

```
A = ones(4,3,2);
B = rand(5,2,3);
size(A)
```
```
ans =

     4     3     2
```
```
size(B)
```
```
ans =

     5     2     3
```
```
disp(A)
```
```
(:,:,1) =

     1     1     1
     1     1     1
     1     1     1
     1     1     1


(:,:,2) =

     1     1     1
     1     1     1
```

```
     1     1     1
     1     1     1
```

```
disp(B)
```

```
(:,:,1) =

    0.4538    0.1734
    0.4324    0.3909
    0.8253    0.8314
    0.0835    0.8034
    0.1332    0.0605


(:,:,2) =

    0.3993    0.2920
    0.5269    0.4317
    0.4168    0.0155
    0.6569    0.9841
    0.6280    0.1672


(:,:,3) =

    0.1062    0.9516
    0.3724    0.9203
    0.1981    0.0527
    0.4897    0.7379
    0.3395    0.2691
```

¿Qué hacen las funciones `size` y `disp`?

Paso 17

Multiplica dos matrices:

```
X = [1 2 -2; 0 -3 4; 7 3 0]
Y = [1 0 -1; 2 3 -5; 1 3 5]
X*Y
```

```
X =

     1     2    -2
     0    -3     4
     7     3     0


Y =

     1     0    -1
     2     3    -5
     1     3     5


ans =

     3     0   -21
    -2     3    35
    13     9   -22
```

Paso 18

Realiza multiplicación elemento a elemento.

```
X .* Y
ans =

     1     0     2
     0    -9   -20
     7     9     0
```

Paso 19

Realiza otra multiplicación de dos matrices.

```
X = eye(3,4)
X =

     1     0     0     0
     0     1     0     0
     0     0     1     0
```

```
Y = rand(4,2)
Y =

    0.4228    0.9831
    0.5479    0.3015
    0.9427    0.7011
    0.4177    0.6663
```

```
X*Y
ans =

    0.4228    0.9831
    0.5479    0.3015
    0.9427    0.7011
```

```
%Y*X
```

¿Por qué no funcionaría la orden anterior?

Paso 20

Utiliza las funciones *diag* y *trace* para realizar operaciones sobre la diagonal de una matriz.

```
Y = rand(3,3)*4
Y =

    2.1565    0.7125    0.6845
    2.7924    0.5121    0.1304
    2.6661    3.9963    2.2448
```

```
Y_diag = diag(Y)
```

```
Y_diag =

    2.1565
    0.5121
    2.2448
```

```
Y_trace = trace(Y)
Y_trace =

    4.9134
```

¿Qué hacen las funciones diag y traza?, ¿podrías tú calcular la traza de otra forma?

Paso 21

Calcula la traspuesta de una matriz.

```
Y
Y =

    2.1565    0.7125    0.6845
    2.7924    0.5121    0.1304
    2.6661    3.9963    2.2448
```

```
Y_t = Y'
Y_t =

    2.1565    2.7924    2.6661
    0.7125    0.5121    3.9963
    0.6845    0.1304    2.2448
```

Paso 22

Calcula la inversa de una matriz y prueba que YY^(-1) = Y^(-1)Y = I, donde I es la matriz identidad.

```
Y_inv = inv(Y)
Y_inv =

    0.1636    0.2958   -0.0671
   -1.5418    0.7854    0.4245
    2.5504   -1.7495   -0.2306
```

```
Y * Y_inv
ans =

    1.0000         0         0
    0.0000    1.0000    0.0000
         0   -0.0000    1.0000
```

```
Y_inv * Y
ans =

    1.0000         0    0.0000
   -0.0000    1.0000    0.0000
    0.0000    0.0000    1.0000
```

Paso 23

Calcula el determinante de una matriz

```
Y_det = det(Y)
Y_det =

    3.8403
```

Paso 24

Ejecuta la siguiente línea de código, una tras otra, para ver como las matrices de celdas son manejadas en MATLAB.

```
clear X
X{1} = [1 2 3;4 5 6;7 8 9]
X =

    [3x3 double]

X{2} = 2+3i
X =

    [3x3 double]    [2.0000 + 3.0000i]

X{3} = 'String'
X =

    [3x3 double]    [2.0000 + 3.0000i]    'String'

X{4} = 1:2:9
X =

    [3x3 double]    [2.0000 + 3.0000i]    'String'    [1x5 double]

X
X =

    [3x3 double]    [2.0000 + 3.0000i]    'String'    [1x5 double]

celldisp(X)

X{1} =

     1     2     3
     4     5     6
     7     8     9



X{2} =

   2.0000 + 3.0000i



X{3} =
```

```
String


X{4} =

     1     3     5     7     9
```

```
X(1)
ans =

    [3x3 double]
```

```
X{1}
ans =

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

¿Qué hace la función celldisp?. ¿Cuál es la diferencia entre las dos últimas líneas anteriores?

Paso 25

Ejecuta esta línea para ver otra forma de asignar valores a las celdas de matrices.

```
X(1) = {[1 2 3;4 5 6;7 8 9]};
```

Paso 26

Las siguientes líneas de código muestran formas correctas e incorrectas de asignar valores a celdas de matrices cuando trabajamos con hileras.

```
%X(3) = 'This produces an error'
X(3) = {'This is okay'}
X =

    [3x3 double]    [2.0000 + 3.0000i]    'This is okay'    [1x5
double]
```

```
X{3} = 'This is okay too'
X =

    [3x3 double]    [2.0000 + 3.0000i]    'This is okay too'    [1x5
double]
```

Paso 27

Crea una matriz con dos estructuras que representa dos imágenes y sus tamaños.

```
my_images(1).imagename = 'Image 1';
my_images(1).width = 256;
my_images(1).height = 256;
```

```
my_images(2).imagename = 'Image 2';
my_images(2).width = 128;
my_images(2).height = 128;
```

Paso 28

Mira los detalles de la estructura y muestra el contenido de un campo.

```
my_images(1)
ans =

    imagename: 'Image 1'
        width: 256
       height: 256
```

```
my_images(2).imagename
ans =

Image 2
```

Paso 29

Muestra información sobre la estructura

```
num_of_images = prod(size(my_images))
num_of_images =

     2
```

```
fieldnames(my_images)
ans =

    'imagename'
    'width'
    'height'
```

```
class(my_images)
ans =

struct
```

```
isstruct(my_images)
ans =

     1
```

```
isstruct(num_of_images)
ans =

     0
```

¿Qué hace la función fieldname?, ¿Qué significa que el resultado de isstruct sea 0 o 1?

¿