9. Manejo de datos en Matlab: Estructuras, Matrices de celdas y Tablas

Hasta ahora hemos visto que matlab permite manipular con facilidad vectores y matrices. Ambos tipos de variables se caracterizan porque sus elementos son siempre de la misma clase: son todos valores numéricos o son todos valores de tipo texto (string o char). No es posible en una misma matriz combinar números y texto. Para ello en matlab existen tipos de variables más sofisticados (*estructuras* y *celdas*) cuya finalidad es precisamente almacenar datos de distintas clases.

Estructuras (`Struct`)

Una *estructura* en matlab es una variable que agrupa varios *contenedores de datos* llamados campos. Cada campo puede contener cualquier tipo de datos (números, cadenas de texto, vectores, matrices …), y puede tener cualquier dimensión. Por ejemplo, si la información que se recoge para cada alumno de la universidad está formada por su nombre, apellidos, dni, año de nacimiento y cursos en que está matriculado, dicha información podría almacenarse

en una estructura que llamaremos `alumno` definida del siguiente modo:

⊕

```
>> alumno.nombre="María";
 alumno.apellido="Rodríguez";
 alumno.DNI=12345678;
 alumno.anioNac=2000;
 alumno.cursosMatriculado=[1 2 3];
 alumno
```

```
alumno =

  scalar structure containing the fields:

    nombre = María
    apellido = Rodríguez
    DNI =  12345678
    anioNac =  2000
    cursosMatriculado =

       1   2   3
```

Obsérvese que los campos de la estructura `alumno` se especifican mediante su nombre separado por un punto del nombre de la estructura. En la misma estructura hay variables de tipo *string* (nombre y apellidos), variables numéricas (DNI y año de nacimiento) y vectores (cursos en que está matriculado)

En el ejemplo, hemos creado la estructura definiendo en cada linea el contenido de un campo. De manera equivalente podemos definir la estructura en una sola línea utilizando la función `struct`:

```
>> alumno=struct("nombre", "María","apellido","Rodríguez","DNI",12345678,"anionac",2000,"
 alumno
```

```
alumno =

  scalar structure containing the fields:

    nombre = María
    apellido = Rodríguez
    DNI =  12345678
    anionac =  2000
    cursosMatriculado =

       1   2   3
```

Nótese que para utilizar esta sintaxis los nombres de los campos deben ir entrecomillados; a continuación de cada campo se introduce su valor, que sólo se entrecomilla si es de tipo carácter.

Es posible definir *vectores de estructuras* (*struct array*); así, por ejemplo, si quisiéramos guardar la lista de alumnos de la clase podríamos seguir añadiendo términos a la estructura anterior como si de un vector se tratase:

```
>> alumno(2)=struct("nombre","Pedro","apellido","González","DNI",87654321,"anionac",1999,
 alumno
```

```
alumno =

  1x2 struct array containing the fields:

    nombre
    apellido
    DNI
    anionac
    cursosMatriculado
```

```
>> alumno.nombre
```

```
ans = María
ans = Pedro
```

```
>> alumno.apellido
```

```
ans = Rodríguez
ans = González
```

Existe la posibilidad de convertir todos los valores del mismo campo en un vector; por ejemplo, si queremos extraer todos los nombres de los alumnos a un único vector empleamos la sintaxis:

```
>> [alumno.nombre]
```

```
ans =

  1×2 string array

    "María"    "Pedro"
```

Para acceder a los datos de algún campo particular de un alumno concreto utilizamos la siguiente sintaxis:

```
>> alumno(1).apellido
```

```
ans = Rodríguez
```

```
>> alumno(2).DNI
```

```
ans =  87654321
```

Ver más información sobre estructuras aquí

Vectores y matrices de celdas (`cells arrays`)

Al igual que las estructuras, las matrices de celdas son variables formadas por varios *contenedores de datos* (campos). A diferencia de las estructuras, cuyos campos tienen un nombre asignado (especificado a continuación del nombre de la estructura separado por un punto), en el caso de las celdas los campos que las forman no tienen nombre sino que se indexan numéricamente del mismo modo que los elementos de una matriz.

Creación de una matriz de celdas

Por ejemplo, supongamos que queremos crear una tabla en la que figuren los nombres de las montañas más altas del mundo junto a su altura:

Las cuatro montañas más altas del planeta

nombre	altura
Everest	8848
K2	8611
Kanchenjunga	8586
Llhotse	8516

Para crear una matriz de celdas con esta información, usamos la misma sintaxis que para las matrices, con la única diferencia de que deberemos usar llaves { } en lugar de corchetes [ ]:

⊕

```
>> mountains={"Everest" "K2" "Kanchenjunga" "Llhotse";
   8848 8611 8586 8516 }
```

```
mountains =

  2×4 cell array

  Columns 1 through 4

    {["Everest"]}    {["K2"]}    {["Kanchenjunga"]}    {["Llhotse"]}
    {[     8848]}    {[8611]}    {[           8586]}    {[     8516]}
```

Las matrices de celdas, igual que las matrices numéricas, se pueden trasponer:

```
>> mountains'
```

```
ans =

  4×2 cell array

    {["Everest"     ]}    {[8848]}
    {["K2"          ]}    {[8611]}
    {["Kanchenjunga"]}    {[8586]}
    {["Llhotse"     ]}    {[8516]}
```

Dado que las matrices de celdas se indexan igual que las matrices numéricas, podemos extraer sus términos indicando índice de fila y columna, si bien en el caso de las matrices de celdas estos índices deben especificarse entre llaves:

```
>> mountains{1,2}
```

```
ans = K2
```

Si especificáramos el índice entre paréntesis, en lugar de obtener el **contenido** de la celda, obtendríamos **una nueva celda** que contiene ese valor:

```
>> mountains(1,2)
```

```
ans =

  1×1 cell array

    {["K2"]}
```

Podemos seleccionar una o varias filas o columnas pero, y **esto es importante**, si queremos que el resultado de la selección sea una nueva matriz de celdas, debemos especificar los índices entre

paréntesis. Si no, lo que se seleccionan son los valores de las celdas y se pierde la estructura de celda:

Obsérvese la diferencia entre usar paréntesis en la selección (se obtiene una matriz de celdas como resultado):

```
>> mountains(1,:)
```

```
ans =

  1×4 cell array

  Columns 1 through 4

    {["Everest"]}    {["K2"]}    {["Kanchenjunga"]}    {["Llhotse"]}
```

y usar llaves (se obtienen los valores que estaban dentro de las celdas)

```
>> mountains{1,:}
```

```
ans =
    "Everest"

ans =
    "K2"

ans =
    "Kanchenjunga"

ans =
    "Llhotse"
```

Añadir nuevos términos a una matriz de celdas

Si queremos añadir, por ejemplo, el año de la primera ascensión a cada una de estas montañas (dichos años fueron respectivamente 1953, 1954, 1955 y 1956) lo haríamos del siguiente modo:

```
>> mountains(3,:)={1953, 1954, 1955, 1956}
```

```
mountains =

  3×4 cell array

    {["Everest"]}    {["K2"]}    {["Kanchenjunga"]}    {["Llhotse"]}
    {[     8848]}    {[8611]}    {[          8586]}    {[     8516]}
    {[     1953]}    {[1954]}    {[          1955]}    {[     1956]}
```

Conversión de matrices de celdas en matrices o vectores numéricos o de texto.

Muchas veces resultará conveniente convertir una matriz de celdas (o una parte de la misma) a vector o matriz (numéricos o de texto). Por ejemplo, si de nuestra matriz de montañas queremos extraer simplemente los nombres (la primera fila), ya hemos visto que `mountains{1,:}` nos devuelve los valores en dicha fila. Para guardar estos valores en un vector bastará con poner esta expresión entre corchetes:

```
>> nombres = [mountains{1,:}];
 nombres
```

```
ans =

  1×4 string array

    "Everest"    "K2"    "Kanchenjunga"    "Llhotse"
```

De igual modo podemos extraer las alturas:

```
>> alturas = [mountains{2,:}];
 alturas
```

```
ans =

        8848        8611        8586        8516
```

Para más conversiones, ver la función cell2mat en la web de Mathworks.

Construcción de una matriz de celdas vacía

La función `cell(m,n)` permite crear una matriz de celdas vacía de dimensión $m \times n$. Si solo se especifica `cell(m)` se crearía una matriz de celdas vacía de dimensión $m \times m$.

```
>> A=cell(3)
```

```
A =

  3×3 cell array

    {0×0 double}    {0×0 double}    {0×0 double}
    {0×0 double}    {0×0 double}    {0×0 double}
    {0×0 double}    {0×0 double}    {0×0 double}
```

Una vez definida una matriz de celdas vacía, podemos rellenar sus celdas. En el ejemplo anterior, en la matriz de celdas `mountain` la primera fila eran nombres de montañas y la segunda eran alturas, pero en general en una matriz de celdas, cada celda puede contener cualquier tipo de datos: números, texto, matrices, incluso otras matrices de celdas. Así, en el siguiente ejemplo, la matriz de celdas `A` tiene su primera fila compuesta por números, la segunda por matrices y la tercera por una mezcla de variables de distinto tipo.

```
>> A(1,:)={1 2 3};
 A(2,:)= {zeros(2), ones(3), [1:5]};
 A(3,:)= {"Texto", [1 1; 2 3], ["Matriz"; "de";"texto"]};
 A
```

```
A =

  3×3 cell array

    {[       1]}    {[       2]}    {[       3]}
    {2×2 double}    {3×3 double}    {1×5 double}
    {["Texto" ]}    {2×2 double}    {3×1 string}
```

Ver más información sobre matrices de celdas aquí

Tablas

En matlab una *tabla* es una matriz especial, caracterizada porque sus columnas tienen nombre, y cada columna puede ser de un tipo distinto (string o numérico). Los datos de alturas de montañas que vimos en la sección anterior también podrían codificarse en matlab como una tabla. En esta sección utilizaremos como ejemplo los siguientes datos, referidos al año 2016 en Canarias:

Datos de Canarias, 2016

Isla	Extensión (km2)	Población	Municipios	Capital
Lanzarote	846	145084	7	Arrecife
Fuerteventura	1660	107521	6	Puerto del Rosario
Gran Canaria	1560	845195	21	Las Palmas de GC

Isla	Extensión (km2)	Población	Municipios	Capital
Tenerife	2034	891111	31	Santa Cruz de Tenerife
La Gomera	370	20940	6	San Sebastián
La Palma	708	81486	14	Santa Cruz de La Palma
El Hierro	269	10587	3	Valverde

Para crear una tabla con estos datos, definimos primero los vectores que conforman cada una de las variables (columnas) de la tabla:

```
isla = ["Lanzarote" "Fuerteventura" "Gran Canaria" ...
      "Tenerife" "La Gomera" "La Palma" "El Hierro" ]';
extension = [846 1660 1560 2034 370 708 269]';
poblacion = [145084 107521 845195 891111 20940 81486 ...
             10587]';
municipios = [7 6 21 31 6 14 3]';
capital = ["Arrecife" "Puerto del Rosario" ...
           "Las Palmas de GC" ...
           "Santa Cruz de Tenerife" "San Sebastián" ...
           "Santa Cruz de La Palma" "Valverde"]';
```

NOTA 1: cuando el comando que escribimos en matlab no cabe en una linea y debe continuar en la linea siguiente, hay que utilizar tres puntos suspensivos … para indicarlo.

NOTA 2: Nótese que hemos definido cada vector como vector fila y al final le hemos añadido un apóstrofe para que se guarde como vector columna. **Es importante que todas las columnas de una tabla se definan como vectores columna**.

Una vez definidos estos vectores creamos la tabla usando la función `table()`:

```
>> canarias = table(isla, extension, poblacion, municipios,capital)
```

```
canarias =

  7×5 table

        isla          extension    poblacion     municipios            capital
    ---------------    ---------    ----------    ----------    ------------------------

    "Lanzarote"           846       1.4508e+05         7        "Arrecife"
    "Fuerteventura"      1660       1.0752e+05         6        "Puerto del Rosario"
    "Gran Canaria"       1560        8.452e+05        21        "Las Palmas de GC"
    "Tenerife"           2034       8.9111e+05        31        "Santa Cruz de Tenerife"
    "La Gomera"           370            20940         6        "San Sebastián"
    "La Palma"            708            81486        14        "Santa Cruz de La Palma"
    "El Hierro"           269            10587         3        "Valverde"
```

Igual que en las estructuras, podemos acceder a cada variable simplemente añadiendo su nombre a continuación del nombre de la tabla, separando ambos con un punto:

```
>> canarias.isla
```

```
ans =

  1×7 string array

  Columns 1 through 5

    "Lanzarote"    "Fuerteventura"    "Gran Canaria"    "Tenerife"    "La Gomera"

  Columns 6 through 7

    "La Palma"    "El Hierro"
```

En la práctica, la mayor parte de las veces las tablas se leerán desde archivos de texto externo mediante la función `readtable` (ver la sección 6-Lectura de datos desde archivos csv).

Más información sobre el uso de tablas aquí