

2. Escalares, vectores y matrices en Matlab

Como ya se mencionó anteriormente es sumamente importante tener en mente el concepto de que para el Matlab **todo es considerado como una matriz**.

Si un número se almacena en una matriz que tiene solamente una fila y una columna, podemos llamar **escalar** a dicho número. De forma semejante, si una matriz tiene una sola fila de varios elementos o una sola columna de varios elementos, la llamamos **vector**; inclusive se hace la especificación de su tipo a través del término **vector fila** o **vector columna** según sea el caso.

De igual forma es necesario recordar que cuando hablamos de matrices, necesitamos de algún mecanismo para hacer referencia a cualquiera de sus elementos ó números individuales que contiene.

Ya sabemos que para localizar a un elemento específico en una matriz se recurre al *número de fila* y al *número de columna* (tomados en ése orden) del lugar en donde se encuentre el elemento buscado. A este par de "coordenadas" se les conoce en Matlab como **subíndices**.

Si se tiene una matriz, digamos M, y se quiere hacer referencia a ella por completo, usaremos su nombre sin subíndices, es decir, como M. Si queremos referirnos al elemento de la fila 4, columna 3 de la matriz M en Matlab, los subíndices se indicarán entre paréntesis; es decir, M(4,3).

2.1 Nombres de variables en Matlab

Al realizar algún cálculo en la pantalla de comandos o al editar algún programa .m de Matlab, por lo general, se asignarán nombres a los escalares, vectores y matrices que usemos. Dichos nombres tendrán que ajustarse a las reglas que siguen para su asignación:

- ❖ Los nombres de variables deben comenzar con una letra.
- ❖ Los nombres de variables pueden contener letras, dígitos y el guión bajo (_).
- ❖ Los nombres de variables pueden tener cualquier longitud, pero sólo los primeros 63 caracteres son reconocidos para diferenciarlas (aunque los restantes sean distintos).

Precaución: Matlab es **sensible** a las mayúsculas y minúsculas. Por ejemplo, *equis*, *Equis* y *EQUIS* representan para Matlab tres variables distintas.

Una buena costumbre en computación es que siempre asignemos nombres de variables que nos den una clara idea del tipo de valor que se está almacenando en dicha variable.

Por otro lado, también es recomendable no utilizar nombres demasiado largos pues esto puede resultar en poca legibilidad de nuestros programas, haciendo más complicada su depuración y mantenimiento.

La recomendación para la mayoría de los casos en Matlab, es utilizar nombres de variables de hasta un máximo de ocho caracteres.

2.2 Inicialización de variables en Matlab

Básicamente son cuatro los métodos para asignar **valores iniciales**, ó **inicializar**, a las matrices en Matlab:

2.2.1 Listas de valores explícitos

La forma más sencilla de definir una matriz es utilizar una lista de números tecleados explícitamente. Por ejemplo:

```
>>A = [ 2.71 ];           (escalar)
>>B = [ 3.8 , -6.2 ];    (vector)
>>C = [-1, 4, 2 ; 2, -3, 1 ; -2, 0, 1 ] ; (matriz)
```

Como se podrá observar, éstas tres definiciones son ejemplos de la instrucción básica de **asignación** (=) en cómputo, que consiste en un nombre de variable situado a la izquierda (lado receptor) seguido de un signo de igual y a la derecha el ó los valores (lado emisor) que se asignarán a la variable.

Dichos valores se encierran entre **corchetes** ordenados por fila; los valores de cada fila se separan mediante comas o espacios, y las filas se separan con signos de punto y coma.

Cuando se define un escalar, vector o matriz, Matlab despliega el o los valores introducidos a menos que se suprima la salida hacia pantalla con un signo de **punto y coma** colocado al final de la instrucción. Por lo general al realizar definiciones de matrices por este método, se suprimirá su despliegue en pantalla.

También puede definirse una matriz escribiendo cada fila en un renglón separado y dando [Enter] para cada una de ellas:

```
>>C = [-1, 4, 2
        2, -3, 1
        -2, 0, 1 ] ;
```

En el ejemplo anterior el [Enter] que se da al final de cada fila sustituye al punto y coma que actúa como separador de ellas.

Si la matriz tiene demasiados números en las filas y no es posible teclearlos en una sola línea podemos recurrir al carácter especial de **continuación** que se define mediante tres puntos seguidos, es decir, *puntos suspensivos* (...).

Este carácter indica que la instrucción continua en la siguiente línea (de hecho se puede usar una ó mas veces en cualquier instrucción Matlab para indicar continuación).

Al definir una matriz, la única condición para que funcione la continuación en otro renglón es que la línea anterior termine en coma justamente antes de los tres puntos. Por ejemplo, resumiendo lo anterior:

```
>>C = [-1, 4, 2
        2, -3, ...
        1 ; -2, ...
        0 , 1 ] ;
```

Define a C, igual que los ejemplos anteriores.

Matlab también permite definir datos usando otros datos definidos previamente. Es decir, se puede definir una matriz en base a otra matriz ya definida anteriormente. Por ejemplo:

```
>>M1 = [ 2.8 3.1 4.7 ];
>>M2 = [ 1.5 M1 ];
```

Que equivale a:

```
>>M2 = [ 1.5 2.8 3.1 4.7 ];
```

También podemos modificar los valores de una matriz o agregar nuevos valores usando subíndices que hagan referencia a una posición específica. Por ejemplo:

```
>>M2 ( 3 ) = 9.8 ;
```

Cambia el valor del tercer **elemento** de la matriz M2 de 3.1 a 9.8. Veamos otro ejemplo:

```
>>M2 ( 8 ) = 5.5 ;
```

Extiende el tamaño de la matriz M2 desde su tamaño original de cuatro elementos hasta ocho. ¿Que ocurre con M2(5), M2(6) y M2(7) para los que no se dieron valores? Resp. Se inicializan automáticamente con cero.

Otro caso. Si tomamos a la matriz M1 tal como está definida anteriormente e inicializamos al elemento:

```
>>M1 ( 2 , 3 ) = 7.7
```

¿Qué ocurre? Resp. Convierte el vector a matriz.

Ejercicios:

Para las siguientes matrices, indique cuál es su contenido y tamaño. Compruebe sus respuestas con el comando whos. Observe que una definición de matriz puede estar basada en otra anterior:

- 1) >>x = [1 ; 5 ; 7 ; 9] ;
- 2) >>y = [4 2 4 ; 6 -3 1] ;
- 3) >>z = [1 2 3 0 ; 5 6 ...
 4 2 ; 1 2 8 5] ;
- 4) >>a = [5 2 9] ;
- 5) >>b = [a 3 a] ;
- 6) >>c = [y(2,1) ; x] ;
- 7) >>d (2 , 2) = -4 ;
- 8) >>e = [y d] ;
- 9) >>f = [[1 ; 2 ; 3] z] ;
- 10) >>g = [[4 ; 7 ; 2] [8; 9; f(3,4)]]

2.2.2 El Operador Dos-Puntos

El **Operador Dos-Puntos** (:) es un operador muy poderoso para inicializar matrices nuevas. Algunos de sus usos típicos son:

- ❖ Crear vectores o matrices a partir de matrices.
- ❖ Referirse a toda una fila de una matriz.
- ❖ Referirse a toda una columna de una matriz.
- ❖ Generar matrices nuevas
- ❖ Seleccionar una **submatriz** de otra matriz

Veamos cada uno de estos casos. Por ejemplo, si:

```
>>C = [ -1, 4, 2 ; 5, 2, -3 ; 4, -4, 0 ; 0, 0, 2 ] ;
```

Entonces,

```
>>X = C ( : , 1)  Almacena la primera columna de
                  C en el nuevo vector columna X.
>>Y = C ( 2 , :)  Almacena la segunda fila de C
                  en el nuevo vector fila Y.
>>Z = C ( : , :)  Crea la matriz Z como una copia
                  íntegra de C (es igual que Z = C).
```

En los tres casos anteriores se puede observar como el operador *Dos-Puntos* puede referirse a todas las filas (primer ejemplo), o a todas las columnas (segundo ejemplo) y finalmente a toda la matriz, usándolo doblemente (último ejemplo).

El operador *Dos-Puntos* también funciona como una especie de “*Generador de rango*” automático.

Si se utiliza el operador *Dos-Puntos* en medio de dos enteros, se generará un nuevo vector cuyos elementos serán todos los enteros comprendidos en el intervalo dado (incluyendo a los enteros especificados). Por ejemplo:

```
>>R = 1 : 10 ;
```

Genera el nuevo vector R que contiene los números del 1 al 10.

Si se usa el operador *Dos-Puntos* para separar tres números, el operador generará valores en el intervalo desde el primer número y hasta el tercero, usando el segundo número como incremento (ó decremento, según sea su signo). Por ejemplo:

```
>>T = 0.0 : 0.5 : 5.0 ;
```

Generará un nuevo vector T con valores de 0.0, 0.5, 1.0, 1.5, ... , 4.5 y 5.0.

```
>>V = 1.5 : -0.25 : 0.0 ;
```

Resulta en un nuevo vector V con los elementos 1.5, 1.25, 1.0, ... , 0.25 y 0.0.

Ahora ya estamos en condiciones de poder extraer una submatriz a partir de otra. Sea C la matriz definida previamente,

```
>>C = [ -1, 4, 2 ; 5, 2, -3 ; 4, -4, 0 ; 0, 0, 2 ] ;
```

Si ejecutamos los siguientes comandos:

```
>>S1 = C ( : , 2 : 3 );
>>S2 = C ( 3 : 4 , 1 : 2 )
```

Obtendríamos respectivamente:

S1 =		S2 =	
4	2	4	-4
2	-3	0	0
-4	0		
0	2		

Si en algún caso el operador *Dos-Puntos* hace referencia a una matriz con subíndices no válidos, como $C(5:6, :)$, se desplegará un mensaje de error:

“??? Index exceeds matrix dimensions.”

Por otro lado, en Matlab es válido la existencia de **matrices vacías** (sin elementos). Por ejemplo, los órdenes generan matrices vacías:

```
>>A = [ ];
>>B = 3 : -1 : 5 ;
```

Hay que tener en cuenta que una matriz vacía **no** es lo mismo que una matriz que sólo contiene elementos con valor cero.

Finalmente, es muy importante notar que tanto los vectores como las matrices tienen como subíndice(s) del primer elemento al **valor 1** y no a 0, como en C ó C++.

Ejercicios:

Dada la matriz x, indique el contenido de las siguientes matrices:

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

- 1) >>a = x (: , 2) ;
- 2) >>b = 10 : 15 ;
- 3) >>c = [4 : 9 ; 1 : 6] ;
- 4) >>d = 0.0 : 0.1 : 1.0 ;
- 5) >>e = x (4 : 5 , 1 : 3) ;
- 6) >>f = x (1 : 2 : 5 , :) ;
- 7) >>g = x (: , 4 : -2 : 1)
- 8) >>h = x (5 : -3 : 1 , 2 : 2 : 4) ;
- 9) >>i = 0.5 : -0.1 : 0.1 ;
- 10) >>j = x (4 : -2 : 1 , 3 : -1 : 2)

2.2.3 Funciones Especiales de Matlab

Matlab incluye una serie de comandos especiales relacionados con la manipulación de matrices y para generación de matrices nuevas:

- ❖ Para obtener las dimensiones de una matriz:

```
[26] >>size ( matriz );
```

El comando *size* regresa dos argumentos de tipo escalar que representan al número de filas y al número de columnas de la matriz.

Si se tiene una matriz C, se podría efectuar la operación de asignación:

```
>>[ a, b ] = size ( C );
```

Que nos proporciona el número de filas de C en el primer elemento a y el número de columnas de C en el elemento b.

De igual forma, el comando :

```
[27] >>length ( vector );
```

Nos proporciona el tamaño del vector especificado (ó número de elementos que contiene). Si se aplica el comando *length* a una matriz de tamaño (m,n) ; se devuelve el número de filas, es decir m.

Veamos ahora como se pueden aplicar estos comandos conjuntamente con otros de tipo especial:

- ❖ Para generar matrices de ceros:

```
[28] >>zeros ( dimensión );
```

En donde el argumento *dimensión* puede ser un número escalar, en cuyo caso el comando *zeros* generará una matriz cuadrada de ceros, usando el valor del argumento dado como el número de filas y el número de columnas.

También *dimensión* podría referirse a dos argumentos escalares como en *zeros(m, n)*. Entonces, se generará una matriz de ceros con m filas y n columnas.

Podemos usar el comando *size* para generar una matriz de ceros que tenga el mismo tamaño que otra matriz mediante la orden:

```
>>D = zeros ( size ( C ) );
```

- ❖ Para generar matrices con unos:

```
[29] >>ones ( dimensión );
```

El comando *ones* genera matrices que contendrán unos en todos sus elementos. Los argumentos de *dimensión* operan de la misma manera que con la orden *zeros*.

Una combinación de éstos comandos sería la obtención de la llamada **matriz identidad**, que tiene unos en la diagonal principal y ceros en todo el resto de sus elementos.

- ❖ Generación de matrices identidad:

```
[30] >>eye ( dimensión ) ;
```

Los argumentos de *eye* que se especifican en *dimensión*, son similares en funcionamiento a los de *zeros* y de *ones*.

Vale la pena recordar que en la mayoría de las aplicaciones matemáticas se usan matrices identidad cuadradas. Este comando puede ser utilizado para matrices no cuadradas si el caso lo requiere.

- ❖ Para generar un vector espaciado linealmente.

```
[31] >>linspace ( x1, x2 ) ;  
ó,  
    >>linspace ( x1, x2, n ) ;
```

En su primera forma *linspace* genera un **vector fila** que contendrá un total de **100** valores, lineales y uniformemente separados, a través de todo el rango desde x_1 hasta x_2 , inclusive.

En la segunda versión, el rango desde x_1 hasta x_2 inclusive, se divide en un total de n valores a solicitud del usuario.

Otra variante de este mismo comando lo constituye la generación de vectores de tipo logarítmico (base 10) con el comando:

```
[32] >>logspace ( x1, x2 ) ;
```

En este caso, *logspace* genera un **vector fila** de **50** valores logarítmicamente e igualmente separados entre el rango especificado, desde 10^{x_1} hasta 10^{x_2} . Se puede hacer que x_2 sea el valor de π y entonces los valores serán desde 10^{x_1} hasta π .

También se puede tener una versión para un número n de puntos dados por el usuario con el comando:

```
>>logspace ( x1, x2, n ) ;
```

- ❖ Números aleatorios en Matlab:

Existen una gran cantidad de problemas de ingeniería que requieren de **números aleatorios** para obtener un solución. En muchos casos estos números aleatorios sirven para la creación de simulaciones de problemas que se repiten y repiten utilizando diferentes valores en cada prueba.

Los números aleatorios **no** se definen por una ecuación; más bien, se caracterizan por la **distribución** de sus valores que presentan.

Los números aleatorios que tienen la misma probabilidad de ser un valor cualquiera entre un límite inferior y un límite superior se denominan **números aleatorios uniformes**.

A veces se necesita generar números aleatorios usando distribuciones en las que algunos valores tienen mayor probabilidad de ser generados que otros.

Las sucesiones aleatorias que presentan este comportamiento probabilístico se denominan **números aleatorios gaussianos** (ó también llamados **números aleatorios normales**).

- ❖ Para la generación de matrices de números aleatorios uniformes:

```
[33] >>rand ( n );
ó
    >>rand ( m, n );
```

El primer caso genera una matriz de $n \times n$ que contiene números aleatorios distribuidos uniformemente en el intervalo $[0, 1]$. El segundo caso funciona de la misma forma pero genera una matriz de tamaño $m \times n$.

La función *rand* genera la misma sucesión de valores aleatorios en cada sesión de trabajo si se usa el mismo valor semilla para generarla (inicialmente Matlab usa 0 como semilla).

Para modificar de forma voluntaria el valor semilla que se esté utilizando en Matlab usamos:

```
>>rand ( 'seed' , n );
```

Aquí se asigna a n como el nuevo valor semilla para el generador de números aleatorios uniformes.

Y si se quiere ver el valor semilla actual que está usando Matlab, daremos:

```
>>rand ( 'seed' );
```

Con frecuencia se requieren sucesiones aleatorias con valores dentro de intervalos distintos de $[0, 1]$. Por lo tanto, si queremos convertir un valor v que está distribuido entre 0 y 1 en un valor uniformemente distribuido entre un límite inferior $x1$ y un límite superior $x2$, usaremos la siguiente ecuación:

$$x = (x2 - x1) * v + x1 ;$$

- ❖ Para la generación de matrices de números aleatorios gaussianos ó normales:

```
[34] >>randn ( n );
ó
    >>randn ( m, n );
```

El primer caso genera una matriz de $n \times n$ que contiene números aleatorios gaussianos (o normales) con una media de 0 y una varianza de 1. El segundo caso funciona de la misma forma pero genera una matriz de tamaño $m \times n$.

Si se desea modificar valores gaussianos v , con una media de 0 y una varianza de 1 de modo que tengan otra distribución gaussiana; digamos, una media de $x1$ y una desviación estándar de $x2$ se puede aplicar la ecuación:

$$x = x2 * v + x1 ;$$

El valor semilla se puede modificar y ver con instrucciones similares a las de las distribuciones uniformes, sólo que con *randn('seed', n)* y *randn('seed')*.

2.2.4 Entrada interactiva del usuario

Este constituye el cuarto y último método para asignar valores iniciales, ó inicializar, a las matrices en Matlab.

Los valores pueden ser introducidos a través del teclado usando el comando:

```
[35] >>input ( 'mensaje' );
```

Este comando despliega el *mensaje* en pantalla y espera a que el usuario introduzca información. Después, él o los valores introducidos son asignados a la variable que se especifique a la izquierda.

Si el usuario va a introducir más de un valor, debe encerrarlos entre corchetes. Si el usuario oprime *[enter]* sin introducir valores, se generará una matriz vacía.

Otra opción es cuando el usuario va a introducir texto en lugar de valores. Entonces se usará esta versión del comando:

```
>>input ( 'mensaje', 's' );
```

La opción 's' indica que la entrada de datos serán caracteres en lugar de números. Para terminar de teclear la cadena de caracteres deberá de oprimirse *[enter]* al final.

Los espacios intermedios de la cadena **si** son tomados en cuenta y forman parte de su longitud total.

Ejercicios:

Dada la matriz *x*, realice lo solicitado en cada problema a continuación:

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

- 1) >>Despliegue el tamaño de *x*.
- 2) >>Genere una matriz *A* de ceros de igual tamaño que *x*.
- 3) >>Genere una matriz *B* de unos del tamaño de la columna de *x*.
- 4) >>Genere una matriz identidad de dimensiones cambiadas a las de *x*.
- 5) >>Genere un vector de 100 valores lineales desde -5 a 5.
- 6) >>Genere un vector de 250 valores logarítmicos desde -1 hasta 1.
- 7) >>Genere una matriz de números aleatorios uniformes entre 0.0 y 10.0
- 8) >>Genere una matriz de números aleatorios uniformes de igual tamaño que la matriz *x*, y que estén entre $-\pi$ y π .
- 9) >>Genere una matriz de números aleatorios gaussianos con una media de 3 y una desviación estándar de 0.25.
- 10) >>Genere una matriz de números aleatorios gaussianos del tamaño de la fila de la matriz *x*, con una media de 1.0 y desviación estándar de 0.5.
- 11) Se reciba una matriz *y* igual a *x* desde teclado, enviando un mensaje al usuario.