****

Facultad de Ciencias Exactas y Naturales

Curso: Programación

Codigo: 0303158

Tema: Vectores

Semestre: 2020-1

Medellín, 2020

**VECTORES**

Hasta ahora, la única manera de almacenar datos que conocemos es lo que denominamos una variable simple: una celda de memoria en la que se puede almacenar un número, y sólo uno, a la vez.

Agrupando celdas, se obtienen estructuras de datos que permiten almacenar paquetes de información a los que se puede acceder de alguna manera. Una de ellas es el **vector**.

La forma típica de agrupar objetos en Matlab u Octave consiste en encerrar una lista de ellos entre corchetes.

Ejemplo:

|  |  |
| --- | --- |
| Instrucción | Resultado |
| **y = [3,5,2,1,3,0,7]** | **y = 3 5 2 1 3 0 7** |
| **y** | **y = 3 5 2 1 3 0 7** |
| **2\*y** | **6 10 4 2 6 0 14** |

Aparentemente, **y** es una variable en la que se almacenaron 8 valores, pero observemos:

|  |  |
| --- | --- |
| Instrucción | Resultado |
| **y(1)** | **3** |
| **y(3) + 2\*y(5)** | **8** |
| **y(4) = y(2)\*y(7)** | **y = 3 5 2 35 3 0 7** |
| **y(10)** | **error: A(I): Index exceeds matrix dimension** |
| **length(y)** | **7** |
| **y(10) = 6** | **y = 3 5 2 35 3 0 7 0 0 6** |
| **length(y)** | **10** |
| **z** | **error: 'z' undefined near line 32 column 1** |
| **z(4) = -2** | **z = 0 0 0 -2** |

Conclusión: **y** no es una variable que contiene 10 valores, sino 10 variables que contienen un valor cada una, llamadas **y(1), y(2), ... y(10)**.

Vector: conjunto de celdas de memoria con un nombre común y diferenciadas por **un** índice.

Entonces, en Matlab y Octave un vector se puede definir de dos formas:

1. Escribiendo todos sus elementos entre corchetes. Los elementos se separan por medio de comas o espacios en blanco.
2. Asignando un valor a una de sus componentes.

Un vector se crea, entonces, en cualquier momento, con uno de los dos procedimientos anteriores y su tamaño se puede aumentar asignando valores a componentes no definidas.

Ejemplos:

|  |  |
| --- | --- |
| **function z = cubos(n)**  **for k = 1:n**  **z(k) = k^3;**  **end** | **function z = fib(n)**  **z = [1,1];**  **for k = 3:n**  **z(k) = z(k-1) + z(k-2);**  **end** |
| **cubos(5)**  **ans = 1 8 27 64 125** | **a = fib(8)**  **a = 1 1 2 3 5 8 13 21**  **a(7)**  **ans = 8** |
| **function z = cubos2(n)**  **for k = n:-1:1**  **z(k) = k^3;**  **end** | ¿Diferencia con **cubos**? |

Ejemplos:

|  |  |
| --- | --- |
| Instrucción | Resultado |
| **b = []** | **b = [](0x0)** |
| **length(b)** | **ans = 0** |

El tamaño de un vector también se puede disminuir en cualquier momento. Se dice por esto, que Matlab maneja los vectores de forma dinámica. En otros lenguajes, el tamaño del vector y el tipo de las componentes se debe definir por una única vez al principio del programa y su tamaño no se puede modificar una vez definido.

Ejemplos:

|  |  |
| --- | --- |
| Instrucción | Resultado |
| **a = fib(8)** | **a = 1 1 2 3 5 8 13 21** |
| **a(6) = []** | **a = 1 1 2 3 5 13 21** |

Matlab permite manejar varios índices simultáneamente:

Ejemplos:

|  |  |
| --- | --- |
| Instrucción | Resultado |
| **b = cubos(9)** | **b =**  **1 8 27 64 125 216 343 512 729** |
| **b(4:7)** | **ans =**  **1 8 27** |
| **b(4:7) = 0** | **b =**  **1 8 27 0 0 0 0 512 729** |
| **b(4:7) = []** | **b = 1 8 27 512 729** |
| **b(6:10) = b(1:5)** | **b = 1 8 27 512 729 1 8 27 512 729** |
| **c = b(3:6)** | **c = 27 512 729 1** |

**Vectores como elementos de**

En general: un vector es un conjunto de variables con un nombre común y diferenciadas por **un** índice.

Podemos asimilar, entonces, un vector con una n-tupla de números reales, recordando que, en programación, es unavariable que contiene un número real:

**Tipos de problemas que se pueden resolver con vectores**

1. Generar vectores. Son funciones que crean vectores.
2. Definir funciones booleanas con argumento vectorial, para verificar propiedades de un vector.
3. Definir funciones con argumentos vectoriales, para operar con los elementos de uno o más vectores o para generar vectores a partir de otros.

Ejemplos:

|  |  |
| --- | --- |
| **function z = vector(n,a,b)**  **for k = 1:n**  **z(k) = floor(rand\*(b-a)+a);**  **end** | **function z = var2(x)**  **n = length(x);**  **M = 0; z = 0;**  **for k = 1:n**  **M = M + x(k);**  **end**  **for k = 1:n**  **z = M + (x(k)-M)^2;**  **end** |
| **function z = ordenado(x)**  **n = length(x);**  **z = 1;**  **for k = 1:n-1**  **if x(k) > x(k+1)**  **z = 0;**  **break;**  **end**  **end** | **function z = pertenece(c,x)**  **n = length(x);**  **z = 0;**  **for k = 1:n**  **if x(k) == c**  **z = 1;**  **break;**  **end**  **end** |
| **function z = escalar(x,y)**  **n = length(x); m = length(y);**  **z = 0;**  **if m != n**  **z = 'No conformables';**  **else**  **for k = 1:n**  **z = z + x(k)\*y(k);**  **end**  **end** | **function z = simetrico(x)**  **n = length(x);**  **z = 1;**  **for k = 1:n/2**  **if x(k) != x(n+1-k)**  **z = 0;**  **break;**  **end**  **end**  **% hacerlo sin fórmula** |

**Agregar elementos por agrupación**

Los corchetes permiten pegar vectores o agregarle elementos a un vector.

|  |  |
| --- | --- |
| **function z = impares(x)**  **n = length(x);**  **z = [];**  **for k = 1:n**  **if mod(x(k),2) == 1**  **z = [z, x(k)];**  **end**  **end** | **function z = separar(x)**  **n = length(x);**  **M = 0; A = []; B = [];**  **for k = 1:n**  **M = M + x(k);**  **end**  **M = M/n;**  **for k = 1:n**  **if x(k) <= M**  **A = [A,x(k)];**  **else**  **B = [B,x(k)];**  **end**  **end**  **z = [A,B];** |
| **function z = invertir(x)**  **n = length(x);**  **z = [];**  **for k = 1:n**  **z = [x(k),z];**  **end** |

**Teoría de Conjuntos**

Los vectores pueden utilizarse para representar conjuntos, guardadas algunas proporciones: los elementos no ocupan una posición específica en un conjunto, los elementos repetidos no se cuentan en un conjunto, la igualdad entre vectores es muy diferente a la igualdad entre conjuntos, etc.

La relación de pertenencia puede facilitar espectacularmente la definición de otras relaciones y operaciones entre conjuntos,

Ejercicios

1. Eliminar los elementos repetidos de un vector
2. Contar los elementos distintos de un vector.
3. Contar los elementos repetidos de un vector.
4. Relación de inclusión: verificar si todos los elementos de un vector pertenecen a otro vector.
5. Relación de igualdad: verificar si dos vectores contienen exactamente los mismos elementos.
6. Definir las operaciones unión, intersección, diferencia y diferencia simétrica entre dos conjuntos.

Ejemplos

|  |  |
| --- | --- |
| % Eliminar elementos repetidos  % de un vector x  **function R = sinrepetidos(x)**  **R = [];**  **for i = 1: length(x)**  **if pertenece(x(i),R) == 0**  **R = [R, x(i)];**  **end**  **end** | **function z = contiene(X,Y)**  % Contiene X a Y?  **z = 1;**  **for k = 1:length(Y)**  **if pertenece(Y(k),X) == 0**  **z = 0;**  **break;**  **end**  **end** |

**Modificación del argumento de una función**

El contenido de una variable, utilizada como argumento en una función, no se modifica al aplicar ésta, aunque se utilice el nombre de un argumento como valor a devolver. En programación, se dice que Matlab no acepta el paso de argumentos por dirección, única forma de modificar el contenido de una variable por parte de una función.

Para cambiar el valor de **x**, por fuera de la función se debe asignar a **x** el valor devuelto por la función.

|  |  |
| --- | --- |
| % Eliminar ceros de x  **function z = sinceros(x)**  **n = length(x);**  **z = [];**  **for k = 1:n**  **if x(k) != 0**  **z = [z,x(k)];**  **end**  **end** | % Dividir cada elemento del  % vector x por el valor c != 0  **function x = dividir(x,c)**  **n = length(x);**  **for k = 1:n**  **x(k) = x(k)/c;**  **end** |
| **b = [14 0 19 7 0 18 3]** | **b = 14 0 19 7 0 18 3** |
| **sinceros(b)** | **ans = 14 19 7 18 3** |
| **b** | **b = 14 0 19 7 0 18 3** |

|  |  |  |
| --- | --- | --- |
| **function x = rotar(x)**  **n = length(x);**  **v = x(1);**  **for k = 1:n-1**  **x(k) = x(k+1);**  **end**  **x(n) = v;** | **b** | **b = 13 19 8 7 15 16** |
| **rotar(b)** | **ans = 19 8 7 15 16 13** |
| **b** | **b = 13 19 8 7 15 16** |
| **b = rotar(b)** | **b = 19 8 7 15 16 13** |
| **b** | **b = 19 8 7 15 16 13** |

**Las cadenas de caracteres como vectores**

Habíamos dicho, alguna vez, que Matlab no es fuerte en el manejo de cadenas de caracteres. La razón de ello es que, para Matlab, una cadena de caracteres es un vector de caracteres, es decir, un vector cuyas componentes son caracteres simples o, más exactamente, códigos Ascii.

|  |  |
| --- | --- |
| **n1 = 'Pedro'** | **n1 = Pedro** |
| **n2 = 'Espinal'** | **n2 = Espinal** |
| **n3 = [n1, n2]** | **n3 = PedroEspinal** |
| **n3(1)** | **ans = P** |
| **n3(1)** | **ans = e** |
| **simetrico('anitalavalatina')** | **ans = 1** |
| **c = 'anitalavalatina'** | **c = anitalavalatina** |
| **c = c - 35** | **c =**  **65 78 73 84 65 76 65 86 65 76 65 84 73 78 65** |
| **c = char(c)** | **c = ANITALAVALATINA** |

**Ordenamiento de vectores**

El ordenamiento es un proceso muy necesario en muchos problemas de la vida real y requiere que los elementos a ordenar estén almacenados en algún tipo de estructura de datos.

Ejemplo.

La función **ordenado,** que vimos en un ejemplo anterior, verifica si un vector está ordenado, pero no hace nada por mejorar el ordenamiento del vector. Si la hacemos un poco más proactiva, podríamos diseñar un primer método de ordenamiento.

|  |  |
| --- | --- |
| **function z = ordenado(x)**  **n = length(x);**  **z = 1;**  **for k = 1:n-1**  **if x(k) > x(k+1)**  **z = 0;**  **break;**  **end**  **end** | **function z = ordenar(x)**  **n = length(x);**  **for k = 1:n-1**  **if x(k) > x(k+1)**  **aux = x(k);**  **x(k+1) = x(k);**  **x(k) = aux;**  **end**  **end** |

|  |  |
| --- | --- |
| **b = vector(6,10,20)** | **b = 14 10 19 19 12 18** |
| **ordenado(b)** | **ans = 0** |
| **b = ordenar(b)** | **b = 10 14 19 12 18 19** |
| **b = ordenar(b)** | **b = 10 14 12 18 19 19** |
| **b = ordenar(b)** | **b = 10 12 14 18 19 19** |
| **b = ordenar(b)** | **b = 10 12 14 18 19 19** |

Analizando cuál sería el número mínimo de veces que se debería repetir el recorrido el recorrido que se hace en la función ordenar, encontramos que es **n-1**, por lo que incorporamos esas repeticiones al programa, obteniendo:

|  |  |
| --- | --- |
| **function x = ordenar(x)**  **n = length(x);**  **for i = 1:n-1**  **for k = 1:n-1**  **if x(k) > x(k+1)**  **aux = x(k);**  **x(k) = x(k+1);**  **x(k+1) = aux;**  **end**  **end**  **end** | Este es el algoritmo de ordenamiento por intercambios sucesivos. |

|  |  |
| --- | --- |
| **b = vector(6,10,20)** | **b = 13 16 14 11 10 16** |
| **b = ordenar(b)** | **b = 10 11 13 14 16 16** |

**Burbuja**

Observando el algoritmo de ordenamiento por intercambios sucesivos, vemos que, con tantos intercambios, en un recorrido sólo se logra mover los elementos menores un lugar a la izquierda, lo cual permite sugerir que lo que interesa en un recorrido es localizar el menor elemento no considerado y, con un sólo intercambio, llevar dicho elemento a su posición.

|  |  |
| --- | --- |
| **Algoritmo burbuja(x)**  **Para i de 1 hasta n-1:**   1. **Hallar la posición p del menor de los x(k), para k de i hasta n.** 2. **Intercambiar x(i) con x(p).** | **b = 14 10 19 19 12 18** |
| **i = 1:** |
| **b = 10 14 19 19 12 18** |
| **i = 2:** |
| **b = 10 12 19 19 14 18** |
| **i = 3:** |
| **b = 10 12 14 19 19 18** |
| **i = 4:** |
| **b = 10 12 14 18 19 19** |
| **i = 5:** |
| **b = 10 12 14 18 19 19** |
| **function x = burbuja(x)**  **n = length(x);**  **for i = 1:n-1**  **p = i;**  **m = x(i);**  **for k = i+1:n**  **if x(k) < m**  **m = x(k);**  **p = k;**  **end**  **end**  **aux = x(i);**  **x(i) = x(p);**  **x(p) = aux;**  **end** | **function p = menor(x,j,k)**  **m = x(j); p = j;**  **for i = j+1:k**  **if x(i) < m**  **m = x(i); p = i;**  **end**  **end** |
| **function x = burbuja(x)**  **n = length(x);**  **for i = 1:n-1**  **p = menor(x,i,n);**  **aux = x(i);**  **x(i) = x(p);**  **x(p) = aux;**  **end** |

Pruebas en Octave:

**octave-3.2.4.exe:84> a = vector(1000,30000,40000);**

**octave-3.2.4.exe:85> a(1:9)**

**ans =**

**37213 39778 31262 38175 37172 30694 36531 34710 34477**

**octave-3.2.4.exe:86> b = ordenar(a);**

**Elapsed time is 15.7716 seconds.**

**octave-3.2.4.exe:87> b(1:9)**

**ans =**

**30010 30011 30021 30037 30042 30052 30052 30055 30079**

**octave-3.2.4.exe:88> c = burbuja(a);**

**Elapsed time is 3.27601 seconds.**

**octave-3.2.4.exe:89> c(1:9)**

**ans =**

**30010 30011 30021 30037 30042 30052 30052 30055 30079**

**octave-3.2.4.exe:90> a = vector(2000,30000,40000);**

**octave-3.2.4.exe:91> b = ordenar(a);**

**Elapsed time is 63.8977 seconds.**

**octave-3.2.4.exe:92> c = burbuja(a);**

**Elapsed time is 12.8856 seconds.**