Vectores

Oscar Gerardo Hernández Martínez

1/7/2019

Para crear un vector podemos definirlo mediante los siguientes comandos:

1. c()
2. scan() Este lo crearemos a mano introduciendo todos y cada uno de los elementos en los renglones subsecuentes.
3. fix(x) Esto primero definiendo el vector mediante la siguiente instrucción: x <- c() y, mediante esta opción, podemos arreglarlo en el block de notas y, una vez modificado, lo guardamos dentro del block de notas para así modificar el vector.
4. rep(a, n) Donde "a" representa el elemento que queremos que contenga el vector y "n" representa el número de veces que queremos que se repita dicho elemento.

Es importante recalcar que no se puede declarar un vector que incluya diferentes tipos de datos, todos deben ser el mismo tipo. Si introduzco un valor de tipo integer, todos deben ser de tipo integer; si introducimos otro tipo de dato, se transformarán en el tipo de dato dominante siguiendo la siguiente jerarquía: character, complex, numeric, integer y logical.

Para poder saber qué tipo dominó al vector, podemos hacer uso de la función class() la cuál nos indicará el tipo de datos que contiene el vector.

Otro uso que le podemos dar a la función scan es importar datos desde una URL para así formar el vector con los datos en una página web. Solo debemos pegar la URL dentro del paréntesis de la función scan y entonces jalará automáticamente los datos de dicho sitio web. También podemos importar los datos desde algún fichero de nuestra computadora colocando las direcciones entre los paréntesis y entre comillas.

IMPORTANTE: Si requieres utilizar datos importados cuya estructura para indicar decimales sea con "," entonces debes colocar scan(dec = ",") y así se creará un vector con valores decimales y no generará errores.

Ejercicios

1. Repite tu año de nacimiento 10 veces *rep(1996, 10)*
2. Crea el vector que contenga las entradas 16,0,1,20,1,7,88,5,1,9 y usa la función fix() y modifica la cuarta entrada *x <- c(16,0,1,20,1,7,88,5,1,9)... fix(x)*

Progresiones y secuencias

Una progresión aritmética es una sucesión de números tales que la diferencia, d, de cualquier par de términos sucesivos de la secuencia es constante.

Para generar una progresión aritmética de diferencia d que empieza en a y termina en b, utilizamos el comando *seq(a, b, by = d)*.

EJEMPLO: *seq(5, 60, by = 5)*

5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60 Este comando nos regresará una progresión aritmética desde el número 5 hasta el 60 y con una diferencia de 5.

Las diferencias no necesariamente deben ser enteras, pueden asignarse valores decimales, teniendo en cuenta que, en ocasiones, `R` nos regresará el último número que no sobrepasa el valor de b.

EJEMPLO: *seq(5, 60, by = 3.5)*

5, 8.5, 12, 15.5, 19, 22.5, 26, 29.5, 33, 36.5, 40, 43.5, 47, 50.5, 54, 57.5 El último valor que nos regresa R es 57.5 pues el el último valor que no sobrepasa 60.

IMPORTANTE La diferencia, no necesariamente puede ser positiva, también se pueden crear progresiones con diferencias negativas.

Si ahora nosotros queremos una progresión aritmética que comience en a y termine en b pero con una longitud n, usaremos el comando *seq(a, b, length.out = n)*. Con esto, R calcula la diferencia d con la siguiente fórmula $d = (b - a)/(n - 1)$

Es decir, si escribo *seq(1, 10, length.out = 10)* R generará la siguiente progresión aritmética:

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10$$

donde d adquiere el valor 1 pues, $d = (10 - 1)/(10 - 1) = 1$

Por otro lado, si nosotros queremos generar una progresión aritmética cuyo primer elemento sea a, tenga longitud n y diferencia d, el comando a escribir será *seq(a, by = d length.out = n)*, así R calculará automáticamente el último número de esa progresión.

EJMEPLO: *seq(4, by = 2 length.out = 15)* 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32

Por último, el comando a:b, define una secuencia de número enteros ($\mathbb{Z}$) desde a hasta b.

Ejercicios

1. Imprime los números del 1 al 20 *1:20*
2. Imprimer los primeros 20 números pares *seq(2, length.out = 20, by = 2)*
3. Imprime 30 números equidistantes entre el 17 y el 98, mostrando solo 4 cifras significativas *round(seq(17, 98, length.out = 30), 4)*

Funciones

Cuando queremos aplicar una función a cada uno de los elementos de un vector de datos, la función sapply nos ahorra tener que programar con bucles en R:

1. *sapply(nombre_de_vector,FUN=nombre_de_función):* para aplicar dicha función a todos los elementos del vector
2. *sqrt(x):* calcula un nuevo vector con las raíces cuadradas de cada uno de los elementos del vector x

Dado un vector de datos x, podemos calcular muchas medidas estadísticas acerca del mismo:

1. **length(x)** Calcula la longitud del vector x
2. **max(x)** Calcula el máximo del vector x
3. **min(x)** Calcula el mínimo del vector x
4. **sum(x)** Calcula la suma de las entradas del vector x
5. **prod(x)** Calcula el producto de las entradas del vector x
6. **mean(x)** Calcula la media aritmética de las entradas del vector x
7. **diff(x)** Calcula el vector formado por las diferencias sucesivas entre entradas del vector original x
8. **cumsum(x)** Calcula el vector formado por las sumas acumuladas de las entradas del vector original x
   - Permite definir sucesiones descritas mediante sumas.
   - Cada entrada de **cumsum(x)** es la suma de las entradas de x hasta su posición.
9. **sort(x)** Ordena el vector en orden natural de los objetos que lo forman, *i.e.*, numérico, alfabético, etc. (**NOTA** en caso de quererlos ordenados de manera decreciente, la estructura será **sort(x, decreasing = TRUE)**).
10. **rev** Invierte el orden de los elementos del vector x

Ejercicios

1. Combina las dos funciones anteriores, sort y rev para crear una función que dado un vector x os lo devuelva ordenado en orden decreciente. *ordenar = function(x){sort(x) -> a rev(a) ->b print(b)}*

2. Razona si aplicar primero **sort** y luego **rev** a un vector x daría en general el mismo resultado que aplicar primero **rev** y luego **sort**.

Subvectores

- **vector[i]:** Da la i-ésima entrada del vector
  - Los índices de `R` empiezan en 1
  - **vector[length(vector)]:** Nos da la última entrada del vector.
  - **vector[a:b]:** si a y b son dos números naturales, nos da el subvector con las entradas del vector original que van de la posición a-ésima hasta la b-ésima.
  - **vector[-i]:** si i es un número, este subvector está formado por todas las entradas del vector original menos la entrada i-ésima. Si i resulta ser un vector, entonces es un vector de índices y crea un nuevo vector con las entradas del vector original, cuyos índices perteneces a i.
  - **vector[-x]:** si x es un vector (de índices), entonces este es el complementario del vector[x].

Operadores lógicos con vectores:

- == : $=$
- != : $\neq$
- ">=": $\geq$
- <= : $\leq$
- ">" : $>$
- < : $<$
- ! : NO lógico
- & : Y lógico
- | : O lógico

Condicionales

- **which(x cumple condición):** para obtener los índices de las entradas del vector x que satisfacen la condición dada.
- **which.min(x):** nos da la primera posición en la que el vector x toma su valor mínimo.
- **which(x==min(x)):** da todas las posiciones en las que el vector x toma sus valores mínimos.
- **which.max(x):** nos da la primera posición en la que el vector x toma su valor máximo.
- **which(x==max(x)):** da todas las posiciones en las que el vector x toma sus valores máximos.

OJO Las funciones **which** nos regresarán las POSICIONES en el vector de los elementos que cumplen la condición dada. Para poder obtener los valores que corresponden a esa posición es necesario combinar las funciones con el comando **x[].**

Ejemplos

1. **x[length(x)-1]** Nos regresa el penúltimo elemento del vector x
2. **x[length(x)-2]** Nos regresa el antepenúltimo elemento del vector x
3. **x[seq(2, length(x), by = 2)]** Nos regresa los elementos pares del vector x
4. **x[seq(1, length(x), by = 2)]** Nos regresa los elementos impares del vector x
5. **x[x%%2==0]** Otra forma de obtener los elementos pares del vector x
6. **x[x%%2==1]** Otra forma de obtener los elementos impares del vector x
7. **x[n] = 15** Añade un nuevo valor en la posición n (cuyo valor será 15) a un vector con *n-1* elementos.
8. **x[2:5] = x[2:5] + 3** Suma 3 a los valores que se encuentran en las posiciones 2, 3, 4 y 5.

9. **x[(length(x)-2):length(x)] = 0** Sustituye los valores que se encuentran a partir de la antepenúltima entrada del vector x hasta la última por el valor 0.
10. **x[length(x) + 5] = 9** Añade el valor de 9 en una entrada que se encuentra 5 posiciones adelante de la última y los valores anteriores adquieren el valor de *NA*.

OJO Si tenemos valores NA, para poder realizar operaciones como sum, prod, etc. necesitamos agregar un parámetro dentro de nuestra función llamado *na.rm* y debemos colocarlo de la siguiente manera: **sum(x, na.rm = TRUE)**. Las demás funciones llevan exactamente la misma estructura. En particular, la función cumsum NO acepta el parámetro *na.rm* por lo que requerimos de la siguiente estructura: **cumsum(x[!is.na(x)])**

Si necesitamos saber qué valores de nuestro vector tienen el valor NA usaremos la función *is.na* y para saber su posición, usaremos una combinación de **which** con esa función; terminando en esta estructura: **which(is.na(x))**

Trabajando en estadística, por lo general los valores de NA son reemplazados por la media para facilitar cálculos y el manejo de los datos dentro de un vector.

Funciones NA

- **na.rm** Hace que los valores NA no se tomen en cuenta al momento de realizar operaciones con vectores.
- **is.na** Nos regresa los valores dentro de un vector que son NA.
- **na.omit** Elimina los valores NA dentro de un vector.