Apuntes Bucles en R

Bucles for

Bucles for simples

Una vez que se ha entendido cómo funcionan los bucles en algoritmia, traspasar los conocimientos a R es muy sencillo. Los bucles se inician introduciendo en R lo siguiente:

for (i in 1:n){

La palabra for indica al programa que se trata de un bucle for. Lo de dentro del paréntesis es lo que pondríamos en el hexágono de algoritmia, es decir, "i" va de 1 hasta "n".

```
for (i in 1:n){
```

Es muy importante la llave del final, que abre el bucle. Para finalizar el bucle es necesario una llave de cierre.

```
for (i in 1:n){

}
```

Entre la llave de inicio y la de cierre se introduce la operación que se quiere realizar. Por ejemplo, creamos dos vectores (**v** y **u**) y se quiere la suma de ambos vectores (con resultado **w**). Recuerdo que para inicializar vectores, es necesario poner la letra "c" antes del paréntesis donde están los componentes de los vectores.

```
v=c(3,-5,9)
u=c(1,0,3.5)
w=0
n=length(v)

for (i in 1:n){
      w[i]=u[i]+v[i]
}
w
```

```
> v=c(3,-5,9)
> u=c(1,0,3.5)
> w=0
> n=length(v)
>
> for (i in 1:n){
+ w[i]=u[i]+v[i]
+ }
> w
[1]  4.0 -5.0 12.5
```

Cabe destacar la necesidad de inicializar el vector **w** a 0, porque si no se hace, en el bucle no va a encontrar ningún vector **w** donde almacenar el resultado. También, el margen que dejamos en la operación que está debajo del bucle nos ayuda a identificar que está dentro del bucle y queda una

estructura más organizada. Por último, decir que podríamos poner que "n" es igual a la longitud del vector antes del bucle, o directamente en el bucle, en vez de "n" escribir length(v).

```
for (i in 1:length(v)){
```

Los bucles son muy útiles para hacer sumatorios y productorios, como es el caso del producto escalar. Lo probamos con los vectores **u** y **v**.

```
v=c(3,-5,9)
u=c(1,0,3.5)
n=length(v)
Prodesc=0

for (i in 1:n){
      Prodesc=Prodesc+v[i]*u[i]
}
Prodesc

v%*%u
```

```
> v=c(3,-5,9)
> u=c(1,0,3.5)
> n=length(v)
> Prodesc=0
>
> for (i in 1:n){
+ Prodesc=Prodesc+v[i]*u[i]
+ }
> Prodesc
[1] 34.5
>
> v%*%u
     [,1]
[1,] 34.5
```

La operación (v%*%u) es una manera de hacer el producto escalar sin necesidad de hacer el bucle, así que comprobamos que el bucle esté bien hecho con esa operación.

Bucles for anidados

La estructura de los bucles anidados, aunque parezca muy confusa, es muy simple. Realmente son dos o más bucles como los anteriores uno detrás de otro. En estos casos un bucle se encuentra dentro de otro.

```
> A= rbind(v,u)    #rbind significa en filas
> B= cbind(v,u)#cbind significa en columnas
> A
  [,1] [,2] [,3]
v    3   -5  9.0
u    1    0  3.5
> B
      v   u
[1,]  3 1.0
[2,] -5 0.0
[3,]  9 3.5

> #Multiplicar A y B
>
> C=matrix(c(0),nrow=nrow(A),ncol=ncol(B))
> n=length(v)
>
> for (i in 1:nrow(A)){
+ for (j in 1:ncol(B)){
+ for (k in 1:n){
+ C[i,j]= C[i,j]+ A[i,k]*B[k,j]
+ }
+ }
+ }
> C
      [,1]  [,2]
[1,] 115.0 34.50
[2,]  34.5 13.25
>
> A%*%B
      v     u
v 115.0 34.50
u  34.5 13.25
```

En este caso lo vamos a comprobar con la multiplicación de matrices. Para definir matrices, por si se necesitase repasar hemos subido otro recurso sobre vectores y matrices.

A y **B** están formados por los vectores **v** y **u** definidos con anterioridad. En la imagen de abajo se observa mejor la anidación de bucles escalonadamente. **A** tiene 2 filas y 3 columnas, mientras que **B** tiene 3 filas y 2 columnas. Se pueden multiplicar sin problemas y el resultado es una matriz **C** de 2x2. El componente "i" indica la fila de **C** y **A**, y "j" indica las columnas de **C** y **B**. "k" es otro valor que necesitamos para la multiplicación, que iría recorriendo las columnas de **A** y las filas de **B**, que son en ambas 3, pero que no comparten con la matriz **C**. De nuevo, se puede comprobar el resultado haciendo: A%*%B

```
A= rbind(v,u)     #rbind significa en filas
B= cbind(v,u)      #cbind significa en columnas
A
B

#Multiplicar A y B

C=matrix(c(0),nrow=nrow(A),ncol=ncol(B))
n=length(v)

for (i in 1:nrow(A)){
      for (j in 1:ncol(B)){
            for (k in 1:n){
                  C[i,j]= C[i,j]+ A[i,k]*B[k,j]
            }
      }
}
C

A%*%B
```

Estructuras condicionales

Bucles while

Lo que hace este bucle es hacer una determinada operación siempre que se cumpla una condición. Cuando esa condición deja de cumplirse el bucle termina.
Para comenzar el bucle se introduce:

```
while (i<=n){
}
```

Lo que hacemos es indicar que empezamos un bucle while, que se realizará siempre que "i" sea menor o igual a "n". Es importante de nuevo cerrar el bucle con una llave de cierre. Entre medias se pondrá la operación a realizar mientras esa condición se cumpla. Lo probamos con la suma de los componentes de un mismo vector **v**.

```
v=c(3,-5,9)
a=0
i=1
while(i<=length(v)){
      a=a+v[i]
      i=i+1
}
a
```

```
> v=c(3,-5,9)
> a=0
> i=1
> while(i<=length(v)){
+ a=a+v[i]
+ i=i+1
+ }
> a
[1] 7
```

Una manera de indicar que "i" aumenta 1 cada vez que se da una vuelta al bucle es introducir dentro del bucle la expresión i=i+1.

Condiciones (if,else if...)

Estas estructuras siguen un proceso condicional. Se detalla una condición, si se cumple se realiza una cosa, si no se cumple se realiza otro proceso o no se hace nada.Para empezar una condición basta con poner la palabra "if" a continuación de una condición. Se ponen la llave de inicio y la de cierre, y entre medias la operación o proceso que se quiere realizar si se cumple dicha condición. Sin embargo, lo que ocurre aquí es que al tratarse de una condición necesitamos que se realice un proceso en caso de que no se cumpla la condición. Esto se marca mediante la palabra "else", que se introduce después de la llave de cierre del if.

```
if (x=3){
        y=2
} else{
        y=0
}
```

En estas estructuras se pueden introducir tantas condiciones como se desee, pero siempre debe haber una condición final, y solo puede haber una, es decir, solo puede haber un "else" por llavero.

```
a=14; b=17
if (a>b){
      c=a+b
}else if (b<2*a){
      c=10*b-a
}else{
      c=0
}
c
```

```
> a=14; b=17
> if (a>b){
+ c=a+b
+ }else if (b<2*a){
+ c=10*b-a
+ }else{
+ c=0
+ }
> c
[1] 156
```