ERRORES

Índice

INTRODUCCIÓN

En este documento verás varios errores que puedes cometer a la hora de usar R. Se incluyen tanto errores al introducir una sentencia que el ordenador no pueda realizar – en las que R nos avisará del error – como errores en los que la sentencia está bien escrita pero no nos da el resultado que queremos obtener – en este caso, R no nos notificará de que hay un error, pero nos dará un resultado distinto –. Los errores que presentaremos son algunos de los que hemos cometido en prácticas o a la hora de resolver ejercicios y se muestra en cada caso un ejercicio sencillo en el que podemos cometer un error con su respectiva solución.

Se incluye en cada error

1. Como solucionar el error
2. Ejemplo
    a. Ejercicio mal resuelto (se incluye una breve explicación de por qué no da el resultado que queremos)
    b. Ejercicio bien resuelto

OPERACIONES BÁSICAS

1. No poner la c antes de introducir los valores del vector o matriz

En otros programas no tenemos la necesidad de indicar que el dato que vamos a introducir es un vector, pero en R debemos poner antes de los valores tanto de un vector como de una matriz una c de la forma v=c(a,b,c,…).

Ejemplo

Introducir en R el vector v=(2,5,6,8).

Ejercicio mal resuelto

Si no introducimos el vector con la c delante R no podrá reconocer el vector.

```
> v=(2,5,6,8)
Error: inesperado ',' en "v=(2,"
```

Ejercicio bien resuelto

Si introducimos el vector con la c delante R podrá reconocer que el dato es un vector con varias componentes.

```
> v=c(2,5,6,8)
> v
[1] 2 5 6 8
```

Este error probablemente no suceda en ejercicios con códigos cortos, sin embargo, en ejercicios complejos podemos olvidarnos de poner una c antes de un vector o matriz. En caso de que el programa nos dé un error tras haber introducido un código extenso conviene revisar si hemos puesto la c antes de todos los vectores y matrices introducidos.

2. Confundir el producto matricial con el producto de los componentes de la matriz

Para obtener el producto matricial de dos matrices A y B se introduce en R la sentencia A%*%B que es diferente a la sentencia A*B. Mientras que A%*%B nos calculará el producto matricial la sentencia A*B nos calcula el producto de los componentes de la matriz A por los de la matriz B.

Ejemplo

Calcular el producto matricial en de la matriz A por la matriz B siendo estas:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Tras haber introducido las matrices

```
> a1=c(1, 2); a2=c(3, 4)
> A=rbind(a1, a2)
> b1=c(1,0); b2=c(0,1)
> B=rbind(b1, b2)
```

Introducimos la sentencia del producto matricial. En este caso, al tratarse una matriz de la identidad, sabemos que el resultado del producto matricial va a ser la propia matriz A.

Ejercicio mal resuelto

Si introducimos la sentencia A*B vemos que nos da el siguiente resultado.

```
> A*B
   [,1] [,2]
a1    1    0
a2    0    4
```

No es la matriz A si no el producto componente a componente de los componentes de la matriz A por los de la B (1x1=1, 2x0=0, 3x0=0 y 4x1=4), de forma general A[i,j]*B[i,j].

Ejercicio bien resuelto

Si introducimos la sentencia A%*%B vemos que nos da el siguiente resultado.

```
> A%*%B
   [,1] [,2]
a1    1    2
a2    3    4
```

Esta es la matriz A, resultado que se obtiene al multiplicar A por la identidad. Concluimos que está bien resuelto.

3. Borrar los datos antes de operar con ellos. Utilizar rm(list=ls(all=TRUE)).

Cuando realizamos un nuevo código conviene introducir la sentencia rm(list=ls(all=TRUE)) antes de empezar a introducir otras sentencias para que R borre todos los datos anteriores y no introduzca datos erróneos al operar. Debemos introducir esta sentencia siempre al inicio para que no borre los datos que vayamos a usar posteriormente en nuestro código.

Nota: Esta sentencia puedes introducirla pulsando en la barra superior Misc > Remover todos los objetos. Te aconsejamos copiarla de la consola y pegarla en el script antes de cada código.

Ejemplo

Introducir un valor a=15 y un valor b=20 y sumarlos en R.

Ejercicio mal resuelto

Si introducimos la sentencia rm(list=ls(all=TRUE)) después de introducir un dato, R borrará este dato y no podrá operar con él.

```
> a=15
> rm(list=ls(all=TRUE))
> b=5
> a+b
Error: objeto 'a' no encontrado
```

Ejercicio bien resuelto

Si introducimos esta sentencia antes de introducir los valores que necesitamos R podrá encontrar los valores a y b y podrá operar con ellos.

```
> rm(list=ls(all=TRUE))
> a=15
> b=5
> a+b
[1] 20
```

Probablemente este error no nos suceda en operaciones simples. En caso de que estemos ante un ejercicio que requiere un código largo y que tiene varios apartados, si R nos dice que no encuentra un objeto para una operación podemos comprobar si hemos metido la sentencia en un lugar incorrecto.

4. Olvidarse de imprimir el valor que nos piden en el ejercicio

Al finalizar un ejercicio en R debemos siempre al final imprimir la variable que nos pide el ejercicio que obtengamos. Para ello podemos usar varias sentencias que nos darán el valor de la variable; para un valor (también sirve para vectores y matrices) a, podemos introducir las sentencias a o print(a). El programa no te informará de este error ya que no es un error en el código; sin embargo, pueden restarte puntos por no dar el valor que te piden al final del ejercicio.

Recuerda poner a o print(a) al final del ejercicio.

5. Confundir mayúsculas y minúsculas

En R a y A son distintas variables independientemente de que se trate de la misma letra. A la hora de ejecutar un código es importante que las variables a las que nos referimos estén perfectamente escritas, una letra distinta supondría otra variable. Este es un error que podemos cometer cuando estamos ante códigos muy largos que requieren mucha atención.

Ejemplo

Sumar la variable a y la variable b en R. Introducir como valores a=100 y b=50.

Introducimos los valores.

```
> a=100
> b=50
```

Ejercicio mal resuelto

Si introducimos por accidente una A mayúscula en vez de a minúscula el programa no encontrará un valor para A.

```
> A+b
Error: objeto 'A' no encontrado
```

Ejercicio bien resuelto

Si introducimos las dos variables perfectamente escritas R podrá encontrar ambos valores y operar.

```
> a+b
[1] 150
```

6. Utilizar valores numéricos en vez de variables. No utilizar códigos para valores genéricos

Cuando hacemos un código es importante que sirva para cualquier valor, no sólo para los valores que nos da el enunciado. Si cometes este error te dará el mismo resultado, pero te descontarán puntos. Introduce sentencias que contengan variables y el menor número de números posible. Mejor será si introduces los datos al inicio del problema y luego realizas cálculos con las variables en las que los has almacenado.

Ejemplo

Dividir dos variables a y b e introducir el resultado en una variable c. Introducir como datos a=4 y b=2.

Ejercicio mal resuelto

Este código no sirve para cualesquiera sean los valores de a y de b.

```
> c=4/2
> c
[1] 2
```

Ejercicio bien resuelto

Este código sirve para cualesquiera sean los valores de a y de b pudiendo aplicarse con otros datos.

```
> a=4
> b=2
> c=a/b
> c
[1] 2
```

Debemos aplicar valores genéricos en todos nuestros códigos. También aplicaremos esto a los bucles con vectores de n componentes introduciendo para ello el valor de la variable n como dato.

BUCLES FOR

1. No introducir el vector a obtener antes del bucle

Es normal que en ejercicios nos pidan un vector cuyas componentes se obtengan a través de un bucle que realice una operación en concreto. Antes de introducir la sentencia del bucle debemos introducir el vector.

Dependiendo de la versión de R que tengamos instalada, deberemos o especificar la longitud del vector. Para ello vale con introducir la sentencia v=c(0).

Ejemplo

Obtener un vector v cuyas componentes sean las componentes del vector w más 1 utilizando bucles for. Tomar valor de w=(1,2,3,4,5).

Primero introducimos los datos

```
> w=c(1,2,3,4,5)
```

Tras ello analizamos la sentencia que nos pide, en este caso se trata de v[i]=w[i]+1

Ejercicio mal resuelto

Si no introducimos datos en el vector v, vemos que nos da un error.

```
> for(i in 1:5){
+                v[i]=w[i]+1
+ }
Error: objeto 'v' no encontrado
> v
Error: objeto 'v' no encontrado
```

El programa no encuentra el objeto v ya que no lo hemos introducido.

Ejercicio bien resuelto

Si introducimos el vector v con la misma longitud que w de antelación vemos que ya no nos da error.
Nota: Para que tuviera la misma longitud de w se ha usado la formula length(w) que calcula la longitud de un vector y la sentencia rep que calcula un vector que repite un elemento tantas veces como se introduzca en el segundo elemento (en este caso, la longitud del vector w).

```
> v=rep(0, length(w))
> v
[1] 0 0 0 0 0
> for(i in 1:5){
+                v[i]=w[i]+1
+ }
> v
[1] 2 3 4 5 6
```

El ejercicio está bien resuelto.

2. No iniciar a 0 o a 1 el componente donde se almacena el resultado del sumatorio o productorio.

En programación los productorios y sumatorios pueden calcularse con bucles for. Para ello introducimos la sentencia *sum=sum + operación dentro del sumatorio* o *prod = prod * operación dentro del productorio.* R necesita que le demos valores previos para prod y para sum para poder aplicar la sentencia. Para los productorios utilizaremos el valor prod=1 y para los sumatorios sum=0.

Ejemplo

Obtener el valor de la suma de los n primeros números naturales y almacenarla en la componente sum y el valor de n! y almacenarlo en la componente prod. Tomar valor de n=5.

Introduciremos antes de realizar los bucles un vector que contenga los valores de 1 a n utilizando las siguientes sentencias.

```
> n=5
> v=(1:n)
```

Ejercicio mal resuelto

Si no ponemos los valores de prod y de sum antes de aplicar el bucle R tomará a prod y a sum como valores no numéricos con los que no puede operar.

```
> for(i in 1:n){
+                prod=prod*v[i]
+                sum=sum+v[i]
+ }
Error in prod * v[i] : argumento no-numérico para operador binario
```

Ejercicio bien resuelto

Si damos valores previos a prod y a sum R utilizará estos valores para las primeras operaciones del bucle y podrá darnos un resultado.

```
> prod=1;sum=0
> for(i in 1:n){
+                prod=prod*v[i]
+                sum=sum+v[i]
+ }
> prod
[1] 120
> sum
[1] 15
```

Comprobando con la calculadora podemos ver que los resultados son correctos.

3. Iniciar a 0 la componente donde se almacena el productorio y a 1 la del sumatorio

Como se ha explicado en el error anterior, debemos introducir valores antes de aplicar el bucle para la componente donde se va a almacenar el resultado del productorio o del sumatorio. Es importante que utilicemos el elemento neutro para la suma en el sumatorio, que es 0, y el elemento neutro para la multiplicación en el productorio, que es el 1. Introducir otros valores dará lugar a modificaciones en el resultado.

Utilizaremos el mismo ejemplo que en el error anterior.

Ejemplo

Obtener el valor de la suma de los n primeros números naturales y almacenarla en la componente sum y el valor de n! y almacenarlo en la componente prod. Tomar valor de n=5.

Introduciremos antes de realizar los bucles un vector que contenga los valores de 1 a n utilizando las siguientes sentencias.

```
> n=5
> v=(1:n)
```

Ejercicio mal hecho

Si confundimos los valores que tenemos que introducir antes de realizar el productorio o sumatorio no nos saldrá un error, pero sí distintos resultados. Si iniciamos el productorio a 0 el resultado será 0

ya que todo número multiplicado por 0 es 0. Si iniciamos el sumatorio a 1 el resultado será el sumatorio más uno.

```
> prod=0;sum=1
> for(i in 1:n){
+                prod=prod*v[i]
+                sum=sum+v[i]
+ }
> prod
[1] 0
> sum
[1] 16
```

Ejercicio bien hecho

Si introducimos 1 como valor inicial de la componente donde se almacena el productorio no altera el resultado ya que cualquier número multiplicado por 1 es él mismo, lo mismo si iniciamos a 0 el valor de la componente donde se almacena el sumatorio.

```
> prod=1;sum=0
> for(i in 1:n){
+                prod=prod*v[i]
+                sum=sum+v[i]
+ }
> prod
[1] 120
> sum
[1] 15
```

4. En bucles anidados, no iniciar la componente donde se va a introducir el sumatorio o productorio justo antes del bucle.

Los bucles anidados los podemos utilizar cuando hay dos componentes i y j en una operación. En el caso de que una de estas operaciones sea un productorio o sumatorio debemos iniciarlas con los valores correspondientes justo antes del bucle para evitar errores en los resultados.

Ejemplo

Almacenar en un vector w utilizando bucles for los factoriales de los n primeros números naturales de la forma w=(1!, 2!, …, n!). Tomar valor de n=5.

Los factoriales son productorios que van desde 1 hasta n. Podemos expresar lo que nos pide el enunciado de la siguiente forma:

$$w[i] = \prod_{j=1}^{i} v[j]$$

Siendo v un vector que contiene los n primeros números naturales. Para empezar, introducimos los valores que necesitamos.

```
> n=5
> v=c(1:n)
> w=c(0)
```

Utilizaremos además la componente prod para almacenar el valor del productorio para los diferentes valores. Como hay dos componentes en el productorio, i y j, utilizaremos dos bucles.

Ejercicio mal resuelto 1

Si iniciamos a 1 el valor de prod dentro del bucle, cada vez que se produzcan las operaciones del bucle el valor de la componente prod se iniciará a 1. Esto provoca que la siguiente componente del vector en vez de ser multiplicada por el valor anterior calculado sea multiplicada por uno. De esta forma, el valor final de cada componente del vector w será cada componente del vector v multiplicada por 1.

```
> for(i in 1:n){
+                 for(j in 1:i){
+                                 prod=1
+                                 prod=prod*v[j]
+                 }
+                 w[i]=prod
+ }
> w
[1] 1 2 3 4 5
```

Se cumple que w[i]=v[j]*1 (1=1x1, 2=2x1, …, 5=5x1)

Ejercicio mal resuelto 2

Si iniciamos a 1 el valor de prod fuera de ambos bucles, cada vez que se repita la sentencia dentro del primer bucle el valor de la componente prod no se iniciará a 1 antes del segundo bucle. Esto provoca que en vez de obtener el resultado de un productorio en cada componente del vector w, obtengamos el productorio que queremos obtener multiplicado por el anterior.

```
> prod=1
> for(i in 1:n){
+                  for(j in 1:i){
+                                   prod=prod*v[j]
+                  }
+                  w[i]=prod
+ }
> w
[1]     1     2    12   288 34560
```

En este caso w=(1!, 1!x2!, …, 1!x2!x3!x4!x5!).

Ejercicio bien resuelto

Si introducimos la sentencia prod=1 justo antes del bucle cada vez que se inicie el primer bucle el valor de prod se iniciará a 1 dando lugar a que el segundo bucle nos dé el valor del productorio que queremos.

```
> for(i in 1:n){
+                 prod=1
+                 for(j in 1:i){
+                                 prod=prod*v[j]
+                 }
+                 w[i]=prod
+ }
> w
[1]   1   2   6  24 120
```

5. Usar vector for para valores de 1 a n cuando en las operaciones del bucle hay una sentencia que contiene v[i-1].

Es importante ver que valores va a tomar i dentro del bucle y asegurarnos de que se puede operar con todos los valores desde el primero hasta n. Si no se puede operar en este caso con i=1 debemos iniciar el bucle en 2.

Ejemplo

Sumar a cada componente del vector v la componente anterior e introducir los resultados en un vector w utilizando bucles for. Utilizar valor de v=(1,2,3,4,5).

Para empezar, introducimos los valores que necesitamos.

```
> v=c(1,2,3,4,5)
> w=c(0)
> n=length(v)
```

Nota: Se utiliza la sentencia length(v) para calcular la longitud del vector v.

Tras leer y analizar el enunciado vemos que cada componente del vector w será el resultado de v[i]+v[i-1]. Introduciremos por lo tanto dentro del bucle for la sentencia w[i]=v[i-1]+v[i].

Ejercicio mal resuelto

Si iniciamos el bucle for con valor de i igual a 1 la operación del bucle no se podrá realizar y por lo tanto se detendrá. R no puede encontrar el valor de v[0] (v[1-1]) y por lo tanto detendrá el bucle ahí y no realizará ninguna operación.

```
> for (i in 1:n){
+                 w[i]=v[i-1]+v[i]
+ }
Error in w[i] <- v[i - 1] + v[i] : replacement has length zero
> w
[1] 0
```

El resultado que nos da es el que hemos introducido para el vector w.

Ejercicio bien resuelto

Como para el valor de w[1] no podemos aplicar la misma sentencia lo calcularemos por separado. Como el enunciado no nos especifica el valor de w[1] asignaremos el valor de v[1]. Si estás en un examen probablemente lo especifique y en caso de que no lo haga recomendamos preguntar al profesor.

```
> w[1]=v[1]
```

Tras calcular el valor de w[1] podemos aplicar el bucle ya que a partir de w[2] los valores de w[i] sí que serán v[i-1]+v[i] (w=(v[1], v[1]+v[2], v[2]+v[3], …, v[n-1]+v[n]).

```
> for (i in 2:n){
+                 w[i]=v[i-1]+v[i]
+ }
> w
[1] 1 3 5 7 9
```

6. Introducir en el bucle una sentencia que no se puede aplicar con un determinado valor

Sobre todo, en sentencias que incluyan divisiones, es importante comprobar que se puede realizar la operación para cualquier componente del vector dentro de la sentencia. En caso de las divisiones, sabemos que el denominador no puede ser 0. Este caso se da, por ejemplo, en la interpolación de Lagrange, donde para que el denominador no sea 0 introducimos una sentencia condicional como veremos en el siguiente ejemplo.

Ejemplo

Dados el vector x=(0,3,8,11) que contiene los puntos de soporte cuyas imágenes para una determinada función están contenidas en el vector f=(2,7,-3,0.5), calcular mediante interpolación de Lagrange el valor de un punto t con bucles for en R y almacenarlo en una variable p. Asignar a t el valor 6.

Primero introduciremos los datos que nos da el problema.

```
> x=c(0,3,8,11)
> f=c(2,7,-3,0.5)
> t=6
```

Tras ello analizaremos las sentencias que necesitamos para obtener la imagen de t. Recordamos que la imagen p de un punto t aplicando la interpolación de Lagrange es:

$$p = \sum_{i=1}^{n} L[i] * f[i]$$

Donde n es el número de puntos de soporte que tenemos, es decir, la longitud del vector donde se están almacenando los puntos de soporte, y L es el polinomio de base de Lagrange en cada punto de soporte que se calcula mediante la expresión:

$$L[i] = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{t - x[j]}{x[i] - x[j]}$$

Al ser p el resultado de un sumatorio iniciaremos su valor a 0 y para el primer bucle introduciremos también el valor de n y el valor del vector L iniciado a 0.

```
> n=length(x)
> p=0
> L=c(0)
```

Como hay dos componentes (i y j) se deben introducir dos bucles. En el segundo bucle (el de la j) introduciremos la sentencia dentro del productorio para calcular L[i] que necesitamos para calcular el valor p. Como L[i] es el resultado de un productorio, antes del segundo bucle iniciaremos su valor a 1. Después de haber calculado L[i] cerramos el segundo bucle e introducimos la sentencia para el sumatorio que nos da el valor de p.

Ejercicio mal resuelto

Si no introducimos la sentencia condicional dentro del bucle j, R no será capaz de calcular el valor de p. Esta sentencia condicional implica que para realizar la operación dentro del productorio i tiene que

ser distinto de j, ya que x[i]-x[i]=0 y el denominador no puede ser 0. El programa no nos dará un error, pero al imprimir p no nos dará un valor.

```
> for(i in 1:n){
+     L[i]=1
+     for(j in 1:n){
+         L[i]=L[i]*(t-x[j])/(x[i]-x[j])
+     }
+     p=p+f[i]*L[i]
+ }
> p
[1] NaN
```

Ejercicio bien resuelto

Si introducimos la sentencia condicional R será capaz de obtener todos los valores del vector L en el segundo bucle y por ende el valor de p.

```
> for(i in 1:n){
+     L[i]=1
+     for(j in 1:n){
+         if(i!=j){
+            L[i]=L[i]*(t-x[j])/(x[i]-x[j])
+         }
+     }
+     p=p+f[i]*L[i]
+ }
> p
[1] 0.9545455
```

7. Introducir el vector del que queremos los resultados como v=c(a) siendo a distinto de 0

Antes de introducir un vector en una sentencia dentro de un bucle for (como se explica en el error 1 de bucles for), debemos introducir la variable donde se almacenará el vector solución como v=c(0). Si la iniciamos a otro número R lo tomará como un vector de una sola componente y no podrá realizar las sentencias del bucle n veces.

Ejemplo

Dada una matriz A calcular el producto total de los componentes de cada fila y almacenarlos en un vector v. Tomar como valor:

$$A = \begin{pmatrix} 1 & 5 & 5 \\ 6 & 9 & 4 \\ 2 & 2 & 0 \end{pmatrix}$$

Para empezar, introduciremos la matriz que nos da el enunciado. Para ello, utilizaremos tres vectores a1, a2 y a3 con los componentes de cada fila y los uniremos en una matriz mediante la función rbind.

```
> a1=c(1,5,5)
> a2=c(6,9,4)
> a3=c(2,2,0)
> A=rbind(a1,a2,a3)
> A
   [,1] [,2] [,3]
a1    1    5    5
a2    6    9    4
a3    2    2    0
```

Posteriormente, analizaremos la sentencia que debemos introducir dentro de uno o varios bucles for. Lo que queremos obtener es un vector v que sea v=(A[1,1]*A[1,2]*… A[1,n], A[2,1]*A[2,2]*...A[2,n],…, A[m,1]*...A[m,n]) siendo m el número de filas y n el de columnas.

Vemos que cada componente del vector v es un productorio de la forma:

$$v[i] = \prod_{j=1}^{n} A[i,j]$$

Como hay dos componentes (i y j) introduciremos dos bucles para los cuales necesitamos los valores de m y n. Calcularemos el valor de m con la sentencia nrow(A) que calcula el número de filas de A y n con ncol(A) que calcula el número de columnas.

```
> m=nrow(A)
> n=ncol(A)
```

Ejercicio mal resuelto

Al tratarse de un productorio podemos pensar que lo óptimo sería introducir el valor del vector v inicialmente como 1, sin embargo, esto R lo tomará como un vector de una sola componente que es 1. Al sólo tener una componente el vector v sólo podrá realizar cálculos cuando i=1, ya que v[2], por ejemplo, no existe para R.

```
> v=c(1)
> for(i in 1:m){
+     for(j in 1:n){
+         v[i]=v[i]*A[i,j]
+     }
+ }
> v
[1] 25 NA NA
```

Ejercicio bien resuelto

Iniciaremos v como v=c(0) y no como v=c(1). Así R podrá tomar v como un vector de n componentes. Como cada componente del vector es un productorio, antes del segundo bucle iniciaremos la componente v[i] a 1.

```
> v=c(0)
> for(i in 1:n){
+     v[i]=1
+     for(j in 1:n){
+         v[i]=v[i]*A[i,j]
+     }
+ }
> v
[1]  25 216   0
```

8. Olvidarse de cerrar un bucle o cerrarlo en un sitio equivocado

Para abrir y cerrar un bucle utilizamos llaves "{"y "}". Cuando estamos ante un código que contiene una cantidad considerable de bucles podemos olvidarnos de poner la llave que cierra el bucle. Aconsejamos para que este error no te suceda poner ambas llaves antes de introducir las sentencias que necesitas dentro del bucle. Además, para que no te confundas entre las sentencias dentro de un bucle o de otro cuando estés ante bucles anidados es mejor que alinees el cierre del bucle con el bucle for correspondiente.

Ejemplo

Calcular el valor del vector v con bucles for mediante la siguiente fórmula:

$$v[i] = \prod_{j=1}^{i} \left(w[j] + \sum_{k=1}^{j} u[k] \right)$$

Siendo i=(1,…,n) donde n es la longitud del vector v igual a la del vector w. Tomar como valores:

w=(1,2,3,4,5)

u=(5,4,3,2,1)

Primero introduciremos los datos que nos da el problema.

```
> w=c(1,2,3,4,5)
> u=c(5,4,3,2,1)
> n=length(w)
```

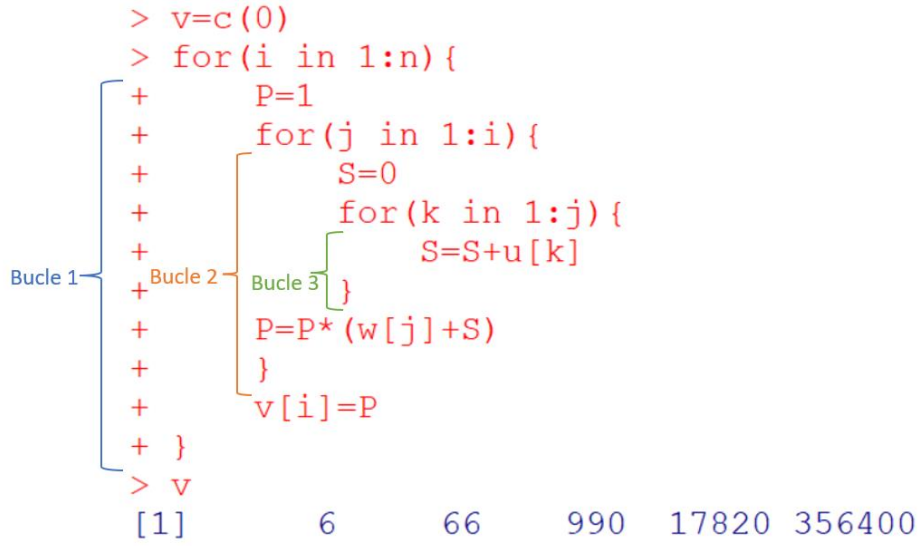
Tras ello introduciremos los bucles que necesitamos con su cierre correspondiente. Al tener tres variables (i,j y k) introduciremos tres bucles.

```
> for(i in 1:n){
+     for(j in 1:i){
+         for(k in 1:j){
+         }
+     }
+ }
```

Para facilitar el ejercicio dividiremos el sumatorio y el productorio de la sentencia de la forma:

$$S = \sum_{k=1}^{j} u[k] \quad P = \prod_{j=1}^{i} w[j] + S$$

Iniciaremos S a 0 y P a 1 antes de cada bucle como hemos explicado en los errores anteriores y gracias a alinear los bucles con las sentencias podremos ver claramente las sentencias dentro de cada bucle.



Si alineas de esta forma es menos probable que te líes con los distintos bucles y donde se cierran.

Nota: Si no cierras un bucle R te alertará poniendo un símbolo más que indica que debes introducir algo más para que el programa pueda realizar las operaciones.