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

BUCLES WHILE

1. No aumentar dentro del bucle los valores de la variable de la condición (i=i+a).

Las sentencias dentro de un bucle while se repiten hasta que se incumple una determinada condición que introducimos al inicio. Si por ejemplo tomamos la variable i y queremos que la secuencia se repita hasta alcanzar un valor n es importante que se aumente el valor se i cada vez que se repite la sentencia introduciendo por ejemplo i=i+a (siendo a un número real). Si no se aumenta el valor de la variable i siempre se cumplirá la condición del bucle while y por lo tanto no acabará.

Nota: R no dejará introducir nuevas operaciones y sentencias hasta acabar de realizar el bucle while y tampoco te dejará cerrar el programa. Para detener la ejecución de un bucle que se repite indefinidamente pulsaremos el botón de la barra superior con una señal de STOP.

Ejemplo

Calcular con bucles while el valor de n! e introducirlo en la componente fac. Utilizar valor de n=5.

Como los factoriales son productorios iniciaremos con el valor de fac igual a 1 e introduciremos el valor de n.

```
> n=5
> fac=1
```

Reconocemos que la sentencia que debemos poner en el bucle con i tomando valores desde 1 hasta n y tratándose de un productorio es fac=fac*i. La condición que debe cumplirse para i es que tenga un valor igual o inferior a n, es decir, i<=n. El primer valor que toma i es i=1 así que introducimos:

```
> i=1
```

Ejercicio mal resuelto

Si no incrementamos el valor de i el bucle se repetirá indefinidamente al siempre tener un valor de i menor que n. No nos dará error, pero tampoco ningún resultado. Cuando paremos el bucle dando al STOP, el valor que nos dará para fac será el mismo que hemos introducido, en este caso, 1.

```
> while(i<=n){
+   fac=fac*i
+ }

> fac
[1] 1
```

Ejercicio bien resuelto

Incrementando el valor de i llegará un punto en el que i supere el valor de n de tal forma que el bucle while se detendrá y R podrá darnos un resultado para la operación.

```
> while(i<=n){
+   fac=fac*i
+   i=i+1
+ }
> fac
[1] 120
```

2. No dar valor a i antes de empezar el bucle

Al ejecutar un bucle for no tienes por qué introducir de antemano el valor de i, ya que el bucle for automáticamente te asigna como valor inicial de i el primero del rango que has introducido (por ejemplo, si i va de 1 a n R te asignará como valor inicial de i 1). Sin embargo, en los bucles while hay que dar un valor a i antes de poner el bucle. Un bucle while se ejecutará hasta que se deje de cumplir la condición que hemos introducido, si esa condición es i<=n, es necesario tener un valor de i para ver si se cumple la condición.

Ejemplo

Calcular el valor del factorial de un número n mediante un bucle while. Tomar como valor n=5.

Ya sabemos que los factoriales son productorios de la forma:

$$fac = \prod_{i=1}^{n} i$$

Por lo tanto, sabemos que antes del bucle debemos iniciar el valor de fac como fac=1. Introduciremos también el valor de n.

```
> fac=1
> n=5
```

Ejercicio mal resuelto

Si no introducimos el valor de i nos dará un error, ya que necesita el valor de i para empezar el bucle while.

```
> while(i<=5){
+       fac=fac*i
+       i=i+1
+ }
Error: objeto 'i' no encontrado
```

Ejercicio bien resuelto

Introduciendo la sentencia i=1 R encontrará un valor para i y podrá aplicarlo al comenzar el bucle while y operar con él.

```
> i=1
> while(i<=5){
+       fac=fac*i
+       i=i+1
+ }
> fac
[1] 120
```

ESTRUCTURAS CONDICIONALES

1. Poner sólo un igual en las condiciones

Si queremos que unas determinadas operaciones se realicen sólo para cuando un valor es igual a otro no debemos poner un igual en la condición, si no dos.

Ejemplo

Para unos determinados valores de a y b, programar un código tal que:

- *Si a y b son iguales las divida*
- *Si a es mayor que b reste b a a*
- *Si b es mayor que a las sume*

Almacenar el resultado en una variable c. Tomar como valores a=5 y b=5.

Para empezar, introduciremos los valores de a y de b.

```
> a=5; b=5
```

Como son tres condiciones necesitaremos tres sentencias condicionales.

Nota: Aunque nosotros veamos que los valores que nos dan son iguales, es necesario programar el código con todas las condiciones que nos dice. Nuestro código debe ser válido para todos los valores y no sólo para el valor ejemplo que nos dan.

Ejercicio mal resuelto

Si introducimos un igual en vez de dos R nos notificará del error.

```
> if(a=b){
Error: inesperado '=' en "if(a="
> c=a/b
> }
Error: inesperado '}' en "}"
```

Aunque al imprimir c vemos que nos da igual a 1 esto no es porque se haya cumplido la condición de que a y b sean iguales, sino porque la operación se ejecuta como si estuviera fuera de una condicional.

Ejercicio bien resuelto

Si introducimos dos iguales vemos que no nos da ningún error y las sentencias se ejecutan correctamente.

```
> if(a==b){
+ c=a/b
+ }
> if(a>b){
+ c=a-b
+ }
> if(b>a){
+ c=a+b
+ }
> c
[1] 1
```

2. Poner una condición después de else

Utilizamos else después de una sentencia if para realizar una operación alternativa en el caso de que no se cumpla la condición que hemos puesto tras if. Es diferente else que else if, aunque ambas son opciones alternativas en el caso de que no se cumpla la condición inicial, else if contiene una condición que debe cumplirse mientras que else se cumple siempre que no se cumpla la condición inicial independientemente de la situación. Por ende, no debemos poner una condición después de else.

Ejemplo

Se tiene un circuito de dos resistencias de valores a y b y una variable c que es 1 si las resistencias están en serie y 0 si están en paralelo. Programar un algoritmo para hallar el valor de la resistencia equivalente y almacenarlo en la variable R teniendo en cuenta que:

- Si las resistencias están en serie, el valor de la resistencia equivalente será la suma de las resistencias.

- Si las resistencias están en paralelo, el valor de la inversa de la resistencia equivalente será la suma de las resistencias.

Tomar como valores a=5 y b=2 y probar con ambos valores de c.

Introducimos los datos para el primer caso en el que c valdrá 1, es decir, las resistencias se encuentran en serie. Posteriormente, volveremos a introducir el código en la consola y cambiaremos únicamente el valor de c.

```
> a=5
> b=2
> c=1
```

Posteriormente, identificaremos las operaciones que necesitamos realizar. Si c es igual a uno entonces aplicaremos:

```
R=a+b
```

Por el contrario, si c es igual a 0, o lo que es lo mismo, no es igual a 1 entonces aplicaremos la siguiente sentencia:

```
R=(1/a+1/b)^(-1)
```

Nota: Otra opción para este ejercicio es aplicar dos sentencias if, una para c=1 y otra para c=0, pero en este ejemplo aplicaremos una sentencia en la que se valorará si c es igual o no a 1 utilizando para ello la sentencia else.

Ejercicio mal resuelto

Si ponemos una condición después de else, R no la identificará como condición y por lo tanto las operaciones que pongamos dentro de las llaves las tomará como operaciones fuera de la sentencia else que se realizarán independientemente del valor de c.

Resistencias en paralelo

```
> a=5
> b=2
> c=1
> if(c==1){
+     R=a+b
+ }else(c==0){
Error: inesperado '{' in:
"   R=a+b
}else(c==0){"
>     R=(1/a+1/b)^(-1)
> }
Error: inesperado '}' en "}"
> R
[1] 1.428571
```

Resistencias en serie

```
> a=5
> b=2
> c=0
> if(c==1){
+     R=a+b
+ }else(c==0){
Error: inesperado '{' in:
"   R=a+b
}else(c==0){"
>     R=(1/a+1/b)^(-1)
> }
Error: inesperado '}' en "}"
> R
[1] 1.428571
```

Ejercicio bien resuelto

Si no ponemos condición en else, las operaciones que pongamos dentro de las llaves se realizarán sólo cuando no se cumpla la condición de la sentencia if, en este caso, c=1.

Resistencias en paralelo

```
> a=5
> b=2
> c=1
> if(c==1){
+     R=a+b
+ }else{
+     R=(1/a+1/b)^(-1)
+ }
> R
[1] 7
```

Resistencias en serie

```
> a=5
> b=2
> c=0
> if(c==1){
+     R=a+b
+ }else{
+     R=(1/a+1/b)^(-1)
+ }
> R
[1] 1.428571
```

3. Poner if en vez de else if

Después de una sentencia if, podemos introducir una sentencia else if. Else if es una sentencia que contiene una condición entre paréntesis y una o varias operaciones entre llaves al igual que las sentencias if. Las operaciones dentro de else if se cumplirán si no se cumplen las condiciones de la sentencia if introducida previamente y sí se cumplen las condiciones dentro del paréntesis de else if. Aunque a veces es indiferente si introducimos dos sentencias if o una if y otra else if, hay casos en los que no da el mismo resultado como veremos en el siguiente ejemplo.

Ejemplo

En una bolsa se tienen bolas del 1 al 100 y se saca una al azar. Programar un código que obtenga la probabilidad de sacar un número par o múltiplo de 5 y almacenar el resultado en una varaible P.

Para hallar la probabilidad introduciremos un vector bolas que contenga los valores del 1 al 100.

```
> bolas=c(1:100)
```

Podemos interpretar P como la suma de las probabilidades de sacar cada una de las bolas que es par o múltiplo de 5. La probabilidad de sacar una bola en concreto es 1/100 (nº de casos favrables/nº de casos posibles). Utilizaremos un bucle for que recorra el vector bolas y si la componente es par o múltiplo de cinco sume a P, que iniciaremos a 0, 1/100 (es lo mismo que 1/n).

```
> n=100
> P=0
```

Para las condiciones utilizaremos una operación que haya el resto de la división de dos números; si este resto es igual a 0 significará que el dividendo es múltiplo del divisor. El resto de la división de dos números se expresa como a%%b.

```
(bolas[i]%%2==0)
(bolas[i]%%5==0)
```

Ejercicio mal resuelto

Si introducimos dos sentencias if en vez de una if y otra else if nos dará un resultado erróneo. En el caso de que la componente cumpla ambas condiciones, como puede ser el 10 que es par y múltiplo de 5, R sumará dos veces 1/100 a P, ya que ha cumplido las dos condiciones de los bulces if. En este caso, vemos claramente que el resultado es equivocado ya que el máximo valor de la probabilidad de un suceso es 1 y el resultado que nos da es 1,3.

```
> for(i in 1:n){
+     if(bolas[i]%%2==0){
+        P=P+1/n
+     }
+     if(bolas[i]%%5==0){
+              P=P+1/n
+     }
+ }
> P
[1] 1.3
```

Ejercicio bien resuelto

Si introducimos una sentencia if y posteriormente else if en el caso de que se cumplan ambas condiciones sólo se sumará una vez 1/100 a P. Pongamos el ejemplo del número 10, como es par se realizarán las operaciones dentro de la sentencia if y como se ha cumplido la sentencia if no se realizarán las operaciones de la sentencia else if. En el caso de números impares múltiplos de cinco al no cumplirse la primera condición R comprobará si se cumple la condición dentro del else if.

```
> for(i in 1:n){
+     if(bolas[i]%%2==0){
+        P=P+1/n
+     }else if(bolas[i]%%5==0){
+              P=P+1/n
+     }
+ }
> P
[1] 0.6
```

FUNCIONES

1. No poner la sentencia return(p)

Cuando acabamos de introducir las operaciones que va a realizar una función debemos introducir la sentencia return(p), siendo p el valor de la imagen que nos dará esa función. Si no introducimos return(p) R nos puede dar un error o bien a la hora de programar la función o bien cuando la apliquemos.

Ejemplo

Programar una función suma_componentes que sume las componentes de un vector v aplicando bucles for.

Analizamos la operación que debe realizar la función y vemos que se trata de un sumatorio en el que se suman las componentes de un vector v. Podemos expresarlo como:

$$p = \sum_{i=1}^{n} v[i]$$

Donde n es la longitud del vector v. Este bucle for lo introduciremos dentro de la función junto con el valor de n.

```
> suma_componentes=function(v){
+ n=length(v)
+ p=0
+ for(i in 1:n){
+     p=p+v[i]
+ }
```

Ejercicio mal resuelto

Si no introducimos return(p) puede que no te salga un error a la hora de programar la función.

```
> suma_componentes=function(v){
+ n=length(v)
+ p=0
+ for(i in 1:n){
+     p=p+v[i]
+ }
+ }
```

Sin embargo, al aplicarla con un vector cualquiera no te dará el resultado. Pongamos el ejemplo de que queremos hallar la suma de los componentes de un vector v=(1,2,0,3).

```
> v=c(1,2,0,3)
> suma_componentes(v)
```

Aun ejecutando el programa no nos da un valor para suma_componentes(v)

Ejercicio bien resuelto

Si introducimos return(p) después de haber introducido el bucle for veremos que al aplicar la función R no tiene ningún problema para darnos la imagen del vector v.

```
> suma_componentes=function(v){
+ n=length(v)
+ p=0
+ for(i in 1:n){
+     p=p+v[i]
+ }
+ return(p)
+ }
> v=c(1,2,0,3)
> suma_componentes(v)
[1] 6
```

2. No conservar el orden al introducir los valores en una función para funciones de dos o más variables

Cuando tenemos una función en la que participan dos variables a y b, al aplicarla debemos tener en cuenta que operaciones hará con a y cuales con b para introducir los valores de los que queremos la imagen en ese orden. Tenemos que tener en cuenta que una vez introducida la función function(a,b)

$$function(a,b) \neq function(b,a)$$

No importa en que orden introduzcamos las variables de una función, lo importante está en conservar ese orden una vez introducida.

Ejemplo

Programar una función div_vectores que divida las componentes de dos vectores de la misma longitud mediante un bucle for y aplicarla para dividir las componentes del vector v=(2,4) entre las de las componentes del vector w=(1,2).

Analizamos las operaciones que nos pide el enunciado, en este caso nos pide dividir componente a componente dos vectores. Obtendremos como resultado de la función otro vector cuyas componentes sean el resultado de la división de las componentes de la forma:

$$d[i] = \frac{v[i]}{w[i]}$$

Donde i comprenderá valores de 1 hasta n siendo n la longitud del vector. Introduciremos la operación dentro del bucle for correspondiente dando previamente un valor a n. Este bucle irá dentro de la función de variables v y w de la siguiente forma:

```
> div_vectores=function(v,w){
+ n=length(v)
+ d=c(0)
+ for(i in 1:n){
+     d[i]=v[i]/w[i]
+ }
+ return(d)
+ }
```

Después de programar la función introduciremos los datos que nos da el ejercicio.

```
> v=c(2,4)
> w=c(1,2)
```

Ejercicio mal resuelto

Si cambiamos el orden de las variables al aplicar la función, es decir, ponemos w primero y v después, lo que hará la función será dividir las componentes del vector w entre las del vector v.

```
> div_vectores(w,v)
[1] 0.5 0.5
```

Ejercicio bien resuelto

Si mantenemos el orden la función dividirá las componentes de v entre las de w.

```
> div_vectores(v,w)
[1] 2 2
```

Recuerda: lo importante no es el orden en el que introduzcamos las variables si no conservarlo. Podemos aplicar esto al ejemplo intercambiando al inicio v por w.

```
> div_vectores=function(w,v){
+ n=length(v)
+ d=c(0)
+ for(i in 1:n){
+     d[i]=v[i]/w[i]
+ }
+ return(d)
+ }
> #datos
> v=c(2,4)
> w=c(1,2)
> div_vectores(v,w)
[1] 0.5 0.5
> div_vectores(w,v)
[1] 2 2
```

3. No introducir en el paréntesis todas las variables que se necesitan para el cálculo de la función

En R podemos programar funciones con gran cantidad de variables. Si nos olvidamos de poner una de las variables de la función antes de las operaciones de la misma, es decir, para f de variables w,v y u

ponemos por ejemplo function(v,w) R no encontrará un valor para dicha variable y no podrá realizar la operación.

Ejemplo

Programar una función que nos dé el valor de los polinomios de base de Lagrange para un punto t mediante un vector x que contiene los puntos de soporte.

Aplicar la función para x=(0.5,2,3,7) y t=4.

Utilizamos la fórmula para calcular el polinomio de base Lagrange para cada uno de los valores de x.

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

Donde i toma valores de 1 a n siendo n el número de puntos de soporte que tenemos. Tenemos por lo tanto de imagen de la función un vector L con los polinomios de base Lagrange en cada punto de soporte.

Aplicando esta fórmula obtenemos el siguiente bucle for que introduciremos dentro de una función:

```
L=c(0)
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])
        }
    }
}
```

Ejercicio mal resuelto

Un error que podemos cometer en este caso es olvidarnos de introducir n como variable dentro de la función. Cuando apliquemos la función con los datos que tenemos R no encontrará un valor para n y nos dará un error.

```
> rm(list=ls(all=TRUE))
> PolBase=function(x,t){
+                   L=c(0)
+                   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])
+                           }
+                       }
+                   }
+                   return(L)
+ }
> #datos
> x=c(0.5,2,3,7)
> t=4
> PolBase(x,t)
Error in PolBase(x, t) : objeto 'n' no encontrado
```

Ejercicio bien resuelto

Si introducimos n como una variable después de x y t y al aplicar la función introducimos como valor de n el número de puntos de soporte que tenemos (en este caso 4 o lo que es lo mismo, la longitud del vector x), R podrá darnos el resultado correcto.

Nota: También podríamos haber introducido dentro de la función n=length(x) y ahorrarnos así una variable de la función, ya que si n se obtiene a partir de x no hace falta que introduzcamos el valor de n aparte.

```
> PolBase=function(x,t,n){
+                   L=c(0)
+                   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])
+                           }
+                       }
+                   }
+                   return(L)
+ }
> #datos
> x=c(0.5,2,3,7)
> t=4
> n=length(x)
> PolBase(x,t,n)
[1]  0.24615385 -1.40000000  2.10000000  0.05384615
```

4.En representaciones, no poner comillas cuando queremos poner un color o tipo de gráfico

Cuando queremos representar una función introducimos la sentencia plot(f) siendo f una función o bien plot(x,f(x)) si queremos la representación de unos puntos en concreto. Dentro de plot podemos introducir más datos como el tipo de función, el color, los límites, etc. Para introducir el tipo de representación pondremos “type=” seguido de una letra que indicará el tipo de función (l para una

línea, b para una línea con los puntos marcados, h para un histograma, etc.). También podemos especificar el color introduciendo "col=" seguido del color en inglés. Es importante que tanto la letra para el tipo de representación como el color estén entrecomillados, si no R no lo entenderá.

Nota: Si quieres saber más detalles acerca de cómo presonalizar gráficos te recomendamos que consultes GRáficos en R.

Ejemplo

Representar gráficamente los valores de la función y=x^2 para los números del 1 al 5 con línea y puntos (type b) de color azul.

Utilizaremos para esta representación un vector x que contenga los valores del 1 al 5.

```
> x=c(1:5)
```

Posteriormente introduciremos la función.

```
> f=function(x){
+             f=x^2
+             return(f)
+ }
```

Obtendremos los valores de las imágenes mediante un bucle for que vaya de 1 a n, siendo n la longitud del vector x y los almacenaremos en un vector y.

```
> n=length(x)
> y=c(0)
> for(i in 1:n){
+     y[i]=f(x[i])
+ }
```

Ejercicio mal resuelto

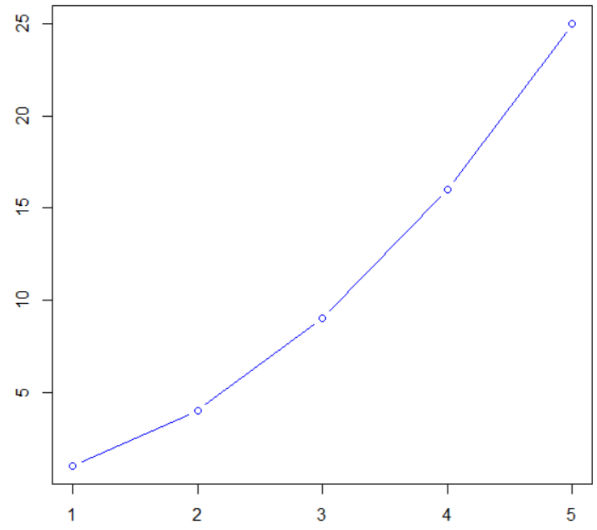
Si no introducimos las comillas R tomará a b y a blue como variables que no tienen valor.

```
> plot(x,y,type=b,col=blue)
Error: objeto 'b' no encontrado
```

Ejercicio bien resuelto

Si introducimos comillas R reconocerá el color y el tipo de gráfico que queremos.

```
> plot(x,y,type=b,col=blue)
```



5. En representaciones, no poner la misma escala cuando queremos representar dos funciones juntas.

Cuando representamos dos funciones juntas R las representa con distinta escala intentándose ajustar lo mejor posible a los puntos introducidos. Para evitar esto basta con poner los mismos límites en ambas representaciones con xlim e ylim.

Ejemplo

Representar en un mismo gráfico con líneas y puntos (type b) el valor de las funciones f(x)=x^2 y g(x)=x^4 para los valores de x del 1 al 5.

Introduciremos el vector x que contenga los valores del 1 al 5.

```
> x=c(1:5)
```

Posteriormente, introduciremos las funciones.

```
> f=function(x){
+             f=x^2
+             return(f)
+ }
> g=function(x){
+              g=x^4
+              return(g)
+ }
```

Calcularemos los valores de las imágenes en ambos casos mediante bucles for, los valores de f(x) en el vector y1 y los de g(x) en el y2.

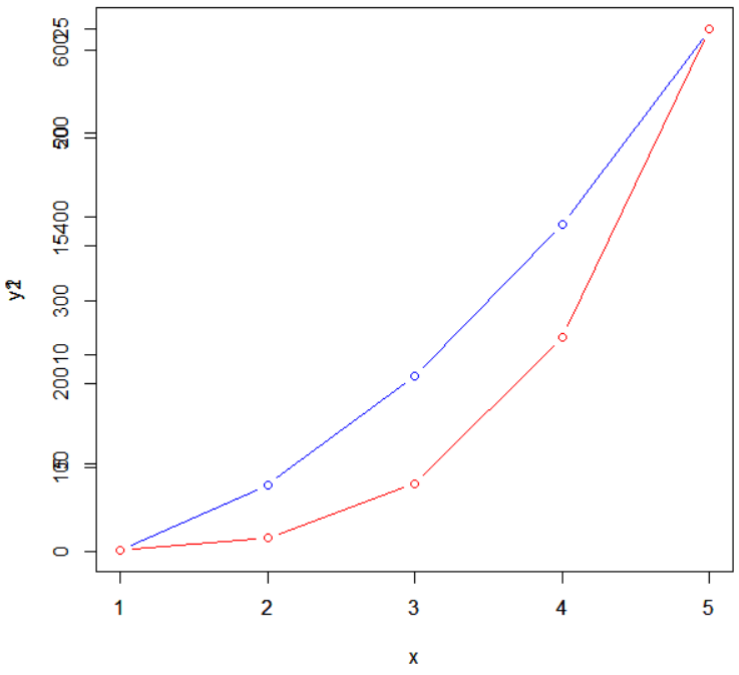
```
> n=length(x)
> y1=c(0)
> for(i in 1:n){
+     y1[i]=f(x[i])
+ }
> y2=c(0)
> for(i in 1:n){
+     y2[i]=g(x[i])
+ }
```

Posteriormente los representaremos. Para que se distingan hemos representado f(x) en azul y g(x) en rojo.

Ejercicio mal hecho

Vemos el resultado de poner ambas representaciones juntas sin límites establecidos donde g(x) parece que tiene valores más pequeños cuando no es así. Si vemos el eje de las y vemos que los valores están solapados a consecuencia de las distintas escalas a las que ambas funciones están representadas.

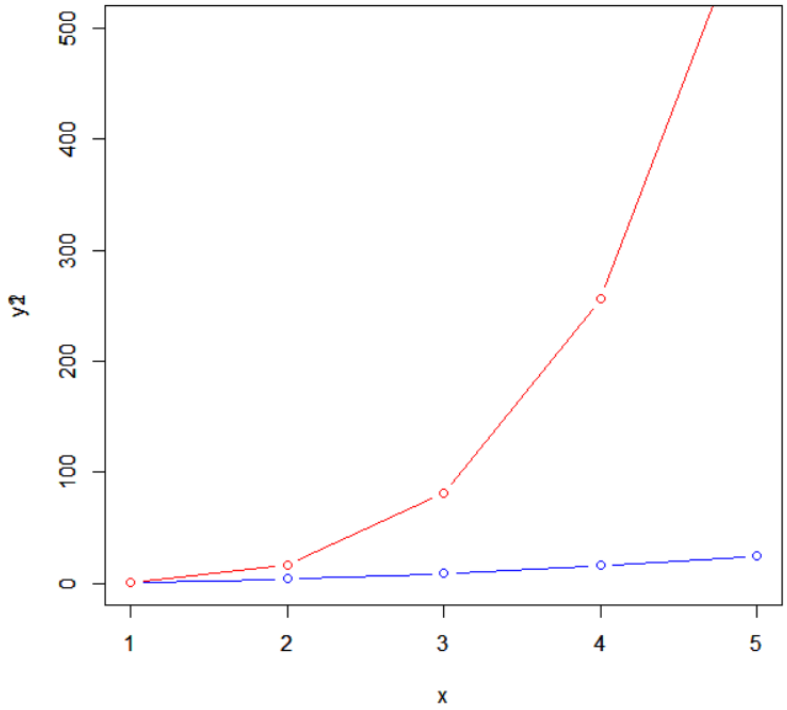
```
> plot(x,y1,type="b",col="blue")
> par(new=TRUE)
> plot(x,y2,type="b",col="red")
```



Ejercicio bien resuelto

Si introducimos unos límites comunes a ambas funciones ambas estarán representadas en la misma escala. Pondremos como límites x del 1 al 5 e y del 1 al 1000.

```
> plot(x,y1,type="b",col="blue",xlim=c(1,5),ylim=c(1,500))
> par(new=TRUE)
> plot(x,y2,type="b",col="red",xlim=c(1,5),ylim=c(1,500))
```


y2
0
100
200
300
400
500
1
2
3
4
5
x