Cómo escribir funciones en R

Marcelino de la Cruz Rot1

1. Departmento de Biología y Geología, Física y Química Inorgánica. ESCET. Universidad Rey Juan Carlos. Móstoles.

Autor para correspondencia: Marcelino de la Cruz Rot [[marcelino.delacruz@urjc.es](mailto:marcelino.delacruz@urjc.es)]

# Palabras clave

Doble estandarización; métodos S3; ordenación Bray-Curtis; ordenación polar.

# Keywords

Bray-Curtis ordination; double standardization; Polar ordination; S3 methods.

# Porqué es útil escribir funciones en R.

La respuesta más sencilla es porque todo en R (R Core Team, 2019) se basa en funciones. Las funciones permiten repetir de forma sencilla (escribiendo menos código) y fiable (evitando errores, consecuencia de lo anterior) los mismos cálculos sobre diferentes conjuntos de datos. Imaginemos que queremos poner a punto en R el clásico método de la ordenación polar (Bray y Curtis, 1957). Como didácticamente resumen Gauch y Scruggs (1980), los pasos a seguir serían:

1. Realizar una estandarización doble de la matriz de inventarios.
2. Calcular la disimilitud entre inventarios.
3. Proyectar los inventarios sobre el eje de ordenación.

Para este ejemplo, usaremos la comunidad de pastizales de dunas que vienen con el paquete vegan (Oksanen et al., 2019).

library(vegan)  
data(dune)  
head(dune)

La denominada “doble estandarización” consiste en realidad en realizar dos normalizaciones consecutivas de la matriz de datos: la primera, dividiendo los valores de las columnas por el valor máximo de cada una y la segunda, dividiendo las filas por la suma total de cada una. Si no sabemos crear funciones, cada vez que necesitemos estandarizar deberemos escribir dos bucles for()

dune.s1<- NULL  
dune.s2 <- NULL  
for( i in 1: ncol(dune)) dune.s1 <- cbind(dune.s1, dune[,i]/max(dune[,i]))  
for(j in 1:nrow(dune.s1)) dune.s2 <- rbind(dune.s2, dune.s1[j,]/sum(dune.s1[j,]))

Seguramente habremos leído en alguna parte que la función apply() posibilita una manera más elegante de trabajar con matrices y data.frames evitando los engorrosos bucles. Claro, que para ello es necesario disponer de una **función** que *aplicar* a las filas o columnas.

# Definiendo funciones en R

Una función en R se crea usando la función function(). Los requisitos necesarios para crearla son: 1) darle nombre, 2) definir su(s) argumento(s) y 3) escribir el código de la función. Por ejemplo, una función para normalizar un vector por su máximo valor la podríamos definir como

standmax <- function(x) x/max(x)

Aquí, standmax sería el nombre de la función, x sería el argumento (la representación simbólica del objeto que vamos a manipular con el código de nuestra función) y x/max(x) constituiría todo el código de la misma. A la hora de dar nombre a nuestra función o a sus argumentos tenemos casi total libertad, la misma que al crear cualquier nuevo objeto en R.

Siguiendo el mismo proceso, otra función para normalizar por la suma total sería:

standtot <- function(x) x/sum(x)

Y con estas dos funciones podríamos realizar la “doble estandarización” de forma más sencilla que con el engorroso bucle:

dune.s1 <- apply(dune, 2, standmax)  
dune.s2 <- t(apply(dune.s1, 1, standtot))

Alternativamente, en vez de dos funciones, podríamos hacer una que realice varios tipos de normalización, por ejemplo:

stand <- function(x, type="max") if(type=="max") x/max(x) else x/sum(x)

En este caso, además del argumento x, que representa la tabla de datos, incluimos el argumento type, que por defecto dejamos inicializado en "max". Sólo habrá que modificarlo si queremos realizar la normalización por la suma. Por lo tanto, la doble estandarización la realizaríamos así:

dune.s1 <- apply(dune, 2, stand)  
dune.s2 <- t(apply(dune.s1, 1, stand, type="tot"))

Claro que, ya puestos, podríamos crear una función que realice todo el proceso, por ejemplo:

doblestand <- function(x){  
 x.s1 <- apply(x, 2, standmax)  
 x.s2 <- t(apply(x.s1, 1, standtot))  
 return(x.s2)  
}

Nótese que en este caso, como la función consta de varios pasos, encerramos el código entre llaves ({}). Además, como durante el proceso creamos varios objetos, pero sólo el último (x.s2) es el que contiene el resultado que nos interesa, especificamos su devolución con la función return(). Por último, es conveniente emplear identación o sangrado al escribir el código dentro de la función para mejorar su legilibilidad (aunque no es obligatorio).

Con la nueva función, nuestros análisis serían mucho más simples:

doblestand(dune)

# Una función más compleja.

De las 36 variantes existentes de ordenación polar (Gauch y Scruggs, 1980), la que más nos conviene para este ejemplo sustituye el cálculo del porcentaje en disimilaridad (en el paso 2) por el cálculo de la distancia ecuclídea entre inventarios (Orl’oci, 1974). En el paso final (3), la opción mas sencilla consistiría en colocar los inventarios sobre el eje representado por la distancia entre los más alejados ( y ) mediante una proyección pitagórica, en base a la ecuación:

donde es la posición del inventario en el eje de ordenación, es la distancia entre los inventarios extremos y , y y son las distancias del inventario a ambos extremos.

opolar<- function(x){  
  
 # doble estandarización  
 x.s<- doblestand(x)  
   
 # matriz de distancia ecuclidea  
 D <- as.matrix(dist(x.s))  
   
 # proyección  
 DAB <- max(D)  
 AB <- which(!is.na(apply(D,1,function(x)  
 ifelse(DAB%in%x, which(x==DAB), NA))))  
 A <-AB[1]  
 B<- AB[2]  
 X<- sapply((1:nrow(D)), function(i) ((D[A,i]^2)  
 -(D[B,i]^2)+(DAB^2))/(2\*DAB))  
   
 # prepara y devuelve el resultado  
 name.x <- deparse(substitute(x))  
 result <- list(A=A, B=B, X=X, namex= name.x)  
 class(result) <- c("miopolar", class(result))  
 return(result)  
}

En esta función hemos incluído ya los tres pasos de la ordenación polar. Es importante notar que, excepto el tercero, cada paso está codificado en una función independiente: esto otorga **modularidad** a nuestro código, lo que facilita su actualización y corrección si fuese necesario, además de mejorar su legibilidad y comprensión. Para esto último es conveniente también incluir comentarios (precedidos por el símbolo **#**), explicando lo que hace cada bloque (o en su caso línea) de código. Como resultado de la ordenación polar tenemos varios objetos: el número de los inventarios extremos (A y B), la posición de los inventarios en el eje de ordenación (X) y el nombre de la tabla de datos original (name.x). La forma más conveniente de devolver varios objetos es dentro de una lista (a la que denominamos resultado). Y como queremos facilitarnos la vida, a dicha lista (que es un objeto de clase list) le asignamos una nueva clase (miopolar), completamente inventada, lo que nos permitirá desarrollar métodos específicos para presentar los resultados de nuestra función de manera cómoda.

La simplicidad que aporta una función de R queda parente en la siguiente frase

dune.o <- opolar(dune)

# Funciones especiales: métodos S3.

A muchos usuarios de R novatos les parece mágico cómo, usando la misma función plot(), R dibuja el gráfico adecuado para cada tipo de datos. Lo que ocurre es que no hay una sino muchas funciones plot() (en realidad, muchos *métodos*: teclear en la consola methods(plot)). De la misma manera, existen numerosos métodos print, que controlan la información que se presenta en pantalla cuando escribimos el nombre de cualquier objeto de R. Podemos crear fácilmente métodos (print, plot, summary, etc) para objetos creados con nuestra función combinando el nombre del método con el de la clase del objeto devuelto por la función. Por ejemplo, un método print que nos describa nuestra ordenación sería algo así:

print.miopolar <- function(x,...){  
 cat("Ordenación polar de", x$name, "a lo largo de un gradiente de longitud", max(X), "\n")  
}

El método print no es necesario invocarlo explícitamente. Una vez que lo hayamos creado, cada vez que escribamos el nombre de un objeto de la clase nos escribirá en pantalla la descripción.

dune.o

## Ordenación polar de dune a lo largo de un gradiente de longitud 0.6477424

Un método plot lo podemos definir así:

plot.miopolar <- function(x, ...){  
 X<- x$X  
 Y <- rep(0, length(X))  
 borde <- max(X)/10  
 plot(cbind(c(-borde, max(X)+borde), c(-1,1)), type="n",   
 main="ordenación", xlab="", ylab="")  
 segments(0,0,max(X),0)  
 points(cbind(X,Y), ...)  
 text(cbind(X,jitter(Y, amount=1/7)), labels=1:length(X),cex=0.7)  
}

En los métodos S3 es muy frecuente incluir los puntos suspensivos (*dots*, ...) en el campo de argumentos (aunque se pueden incluir en cualquier función). Se emplean para pasar argumentos adicionales a alguna función interna del método. En este ejemplo, cualquier argumento adicional que incluyamos, se traspasará directamente a la función points(), que es donde vuelven a aparecer los .... Por lo tanto deberán ser argumentos aceptados por dicha función. Es el caso de la Figura 1. Nótese que no es necesario escribir el nombre completo de la function (plot.miopar) sino simplemente el método que codifica.

plot(dune.o)  
plot(dune.o, pch=19, col=2)

# Mejorando las funciones.

Entre las posibles mejoras que podríamos aplicar a nuestra función estarían compactar el paso 3 (proyección) en una función independendiente e implementar la extracción de ejes de ordenación complementarios (Bray y Curtis, 1957) para que pudiésemos representar los típicos diagramas de ordenación basados en dos ejes, como en un PCA o en un NMDS (Borcard et al., 2018). Lógicamente, deberíamos modificar también el método plot.miopolar(). Para un lector de la revista Ecosistemas con interés por la ecoinfomática esto no debería resultar muy complicado. Y dado que estamos creando muchas funciones complementarias, podríamos ir pensando en crear un paquete de R para distribuirlas conjuntamente.

# Agradecimientos

REMEDINAL TE-CM (S2018/EMT-4338).

###### REFERENCIAS

Borcard, D., Gillet, F., Legendre, P. 2018. *Numerical Ecology with R*. Springer International Publishing.

Bray, J.R., Curtis, J.T. 1957. An Ordination of the Upland Forest Communities of Southern Wisconsin. *Ecological Monographs* 27: 325-349.

Gauch, H., Scruggs, W. 1980. Variants of polar ordination. *Vegetatio* 40: 147-153.

Oksanen, J., Blanchet, F.G., Friendly, M., Kindt, R., Legendre, P., McGlinn, D., Minchin, P.R. et al. 2019. *vegan: Community Ecology Package*.

Orl’oci, L. 1974. Revisions for the Bray and Curtis ordination. *Canadian Journal of Botany* 52: 1773-1776.

R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

###### PIES DE FIGURA

**Figura 1**. Representación gráfica de la ordenación polar de dune.

###### FIGURE LEGENDS

**Figure 1**. Graphical representation of the polar ordination of the ‘dune’ data.

###### FIGURA 1

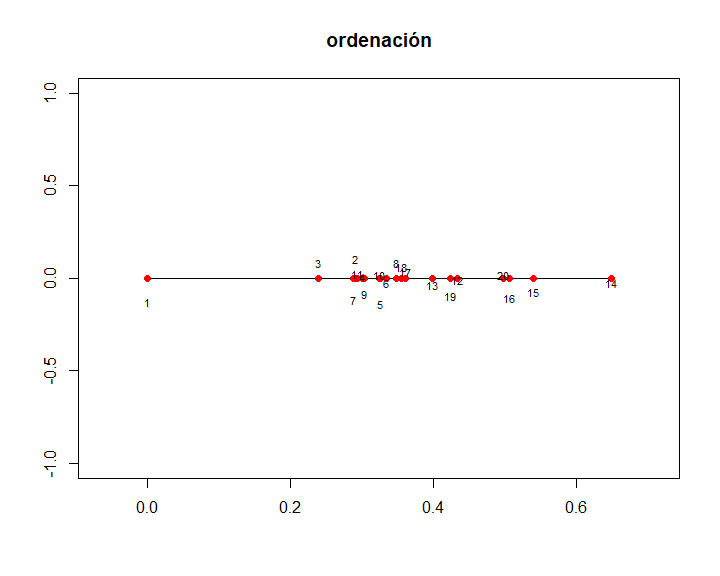


Figura 1. Representación gráfica de la ordenación polar de los datos ‘dune’