Análisis de diversidad: Diversidad α

Parte II: Aplicando la teoría en R

Juan D. Vásquez-Restrepo

Posgrado en Ciencias Biológicas, Universidad Nacional Autónoma de México
(mailto:#)juanda037@outlook.com (mailto:juanda037@outlook.com)

En la sección anterior (Análisis de diversidad: Parte I (https://vr-daniel.github.io/Diversidad-en-R/U1P1.html)) abordamos los fundamentos teóricos detrás de los análisis de diversidad, los índices, estimadores, curvas de acumulación y rarefacción, diversidades verdaderas, y perfiles de diversidad. Ahora, aprenderemos a calcularlos y graficarlos usando R (integrado a RStudio).

Vale la pena mencionar que, este no es un tutorial de programación en R (se necesita conocimiento previo), pero los códigos que veremos a continuación no están optimizados, o sea, son más largos de lo que podrían ser para que podamos seguir el paso a paso, además están comentados, de modo que podamos entender qué hace cada línea. Por temas de simplicidad utilizaremos el motor base de R para graficar, aunque personalmente recomiendo y uso el paquete *ggplot2*, el cual ofrece más opciones de personalización pero que a su vez requiere de una sintaxis un poco más complicada.

1. Primeros pasos: librerías y datos

Lo primero que tenemos que hacer es descargar y cargar las librerías que vamos a utilizar. En este caso son cuatro, las tres primeras enfocadas en los análisis de diversidad biológica, y la cuarta en el manejo de datos.

```
# Instalar librerías (usar solo la primera vez)
install.packages("entropart")
install.packages("iNEXT")
install.packages("vegan")
install.packages("dplyr")
```

```
#Cargar librerías
library(entropart)
library(iNEXT)
library(vegan)
library(dplyr)
```

Ahora, para este ejercicio utilizaremos el conjunto de datos BCI, el cual viene precargado en el paquete *vegan* e incluye datos de abundancia para 225 especies de árboles en la isla de Barro Colorado, Panamá, muestreadas en 50 parcelas.

```
# Cargar el conjunto de datos BCI
data(BCI)
```

En el caso de tener nuestros propios datos podemos importarlos desde un archivo de texto delimitado, usando las funciones `read.table`, `read.csv` o `read.delim`. Es importante que nuestros datos estén organizados de modo que las especies estén en las columnas y las unidades de muestreo o comunidades en las filas.

2. Índices de diversidad tradicionales

Índices de Margalef y Menhinick

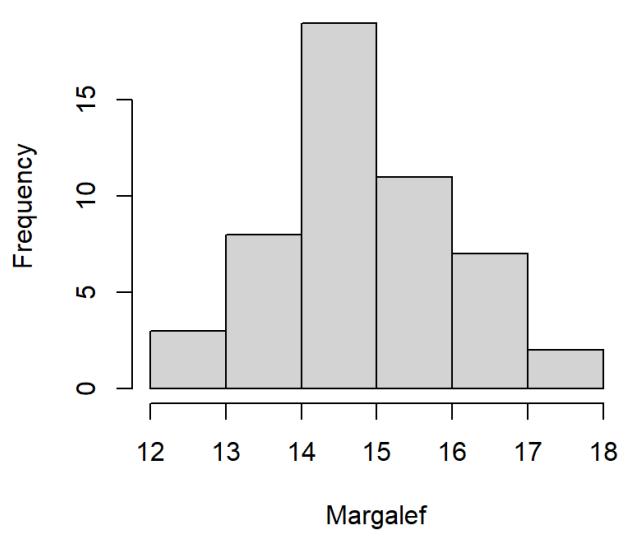
Comenzaremos por calcular los índices de Margalef y Menhinick, para los cuales no tenemos una función predefinida pero que podemos crear aprovechando que su fórmula es bastante sencilla. Para estos índices necesitamos dos datos básicos, el número de especies y el total de individuos, pero como tenemos 50 unidades de muestreo es necesario obtener los datos para cada una.

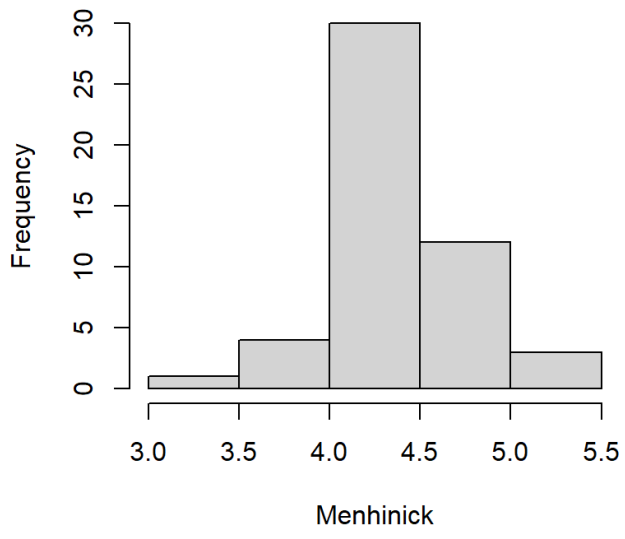
```
# Vector para almacenar el número de especies por unidad de muestreo
S <- c()

# Ciclo para recorrer la matriz de datos
for(i in 1:nrow(BCI)){ # Ciclo desde 1 hasta el total de filas de los datos (unidades de muestreo)
  S.tmp <- length(which(BCI[i, ] > 0)) # Número de especies (columnas) con abundancia > 0 por unidad de muestreo
  S <- append(S, S.tmp) # Añadimos el número de especies del sitio i al vector de especies
}

# Suma de las abundancias por unidad de muestreo (filas)
N <- rowSums(BCI)

# Índice de Margalef
Margalef <- (S - 1) / log(N)

# Índice de Menhinick
Menhinick <- S / sqrt(N)

# Combinamos ambos índices en una misma tabla
indices <- cbind(Ma = Margalef, Me = Menhinick)
```

```
##          Ma       Me
## 1  15.07013 4.393837
## 2  13.66177 4.027492
## 3  14.50048 4.182655
## 4  14.92662 4.170576
## 5  16.06540 4.494441
## 6  13.95112 4.187649
## 7  13.43131 4.020381
## 8  14.34198 4.238811
## 9  14.79950 4.450214
## 10 15.04850 4.277148
```

Como vemos, el resultado es una matriz con los valores de cada índice para los 50 sitios muestreados, aunque aquí para ahorrar espacio solo aparecen los primeros 10. Si hacemos un histograma de frecuencias podemos observar de manera gráfica y general cómo varían los valores para posteriormente interpretarlos.

```
# Histograma índice de Margalef
hist(indices[, 1], # Valores de todas las filas de la primera columna de la tabla índices
     xlab = "Margalef", # Etiqueta del eje X
     main = NA) # Sin  título principal

# Histograma índice de Menhinick
hist(indices[, 2], # Valores de todas las filas de la segunda columna de la tabla índices
     xlab = "Menhinick", # Etiqueta del eje Y
     main = NA) # Sin título principal
```





Índices de dominancia y equitatividad

A continuación calcularemos los índices de Simpson, Shannon, y Pielou. Para esto haremos uso del paquete *vegan*, el cual incluye funciones predefinidas para los dos primeros, mientras que para el tercero nos valdremos de su fórmula. A diferencia de los anteriores, la función `diversity` nos ahorra el trabajo de crear un ciclo para recorrer nuestra matriz, calculando de manera automática cada índice por unidad de muestreo.

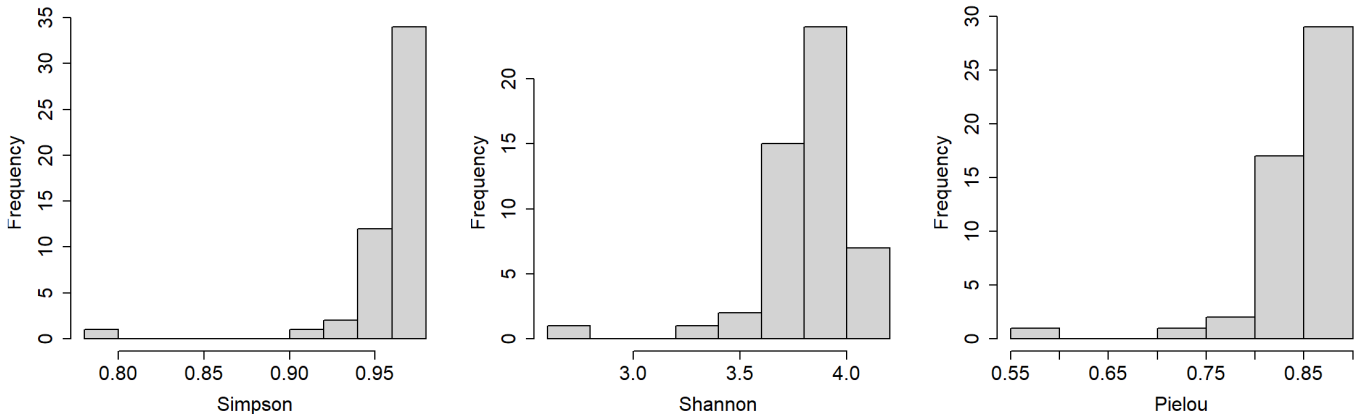
```
# Índice de Gini-Simpson
Simpson <- diversity(BCI, index  = "simpson")

# Índice de Shannon
Shannon <- diversity(BCI, index = "shannon")

# Índice de Pielou
Pielou <- Shannon / log(S)

# Combinamos los índices en una tabla
indices <- cbind(Simpson = Simpson, Shannon = Shannon, Pielou = Pielou)
```

```
##      Simpson  Shannon    Pielou
## 1  0.9746293 4.018412 0.8865579
## 2  0.9683393 3.848471 0.8685692
## 3  0.9646078 3.814060 0.8476046
## 4  0.9716117 3.976563 0.8752597
## 5  0.9678267 3.969940 0.8602030
## 6  0.9627557 3.776575 0.8500724
## 7  0.9672014 3.836811 0.8706729
## 8  0.9671998 3.908381 0.8729254
## 9  0.9534257 3.761331 0.8358867
## 10 0.9663808 3.889803 0.8561634
```

Por defecto, la función `diversity` calcula el índice de Gini-Simpson a partir del índice de Simpson no corregido. Adicionalmente, para el índice de Shannon utiliza el logaritmo natural, a menos que especifiquemos otra base agregando el parámetro `base` (p.e. `base = 2`).

Si graficamos de nuevo los histogramas podemos ver la distribución general para los tres índices, lo cual nos ayuda a tener un panorama de los resultados cuando tenemos muchos datos.

```
#Histograma índice de Simpson
hist(indices[, 1],
     xlab = "Simpson",
     main = NA)

#Histograma índice de Shannon
hist(indices[, 2],
     xlab = "Shannon",
     main = NA)

#Histograma índice de Pielou
hist(indices[, 3],
     xlab = "Pielou",
     main = NA)
```



En el caso en que nuestras unidades de muestreo pertenezcan a una misma comunidad, podemos sumar las abundancias de cada especie y calcular un único índice general utilizando los mismos comandos anteriores.

```
# Número de especies de la comunidad
S <- ncol(BCI)

# Suma de las abundancias de cada especie
BCI.Sum <- colSums(BCI)

# Índice de Gini-Simpson
Simpson <- diversity(BCI.Sum, index  = "simpson")

# Índice de Shannon
Shannon <- diversity(BCI.Sum, index = "shannon")

# Índice de Pielou
Pielou <- Shannon / log(S)

# Combinamos los índices en una tabla
indices <- cbind(Simpson = Simpson, Shannon = Shannon, Pielou = Pielou)
```

```
##    Simpson  Shannon    Pielou
##  0.9736755 4.270409 0.7884656
```

3. Estimadores de riqueza

Estimadores no paramétricos

Siguiendo el órden de la sección anterior vamos a calcular ahora los estimadores de riqueza. Lo primero que debemos hacer es elegir los adecuados, que en este caso son Chao 1 y ACE porque tenemos datos de abundancias. Respecto al uso de los estimadores de Chao tenemos que recordar que existe una versión corregida y una no corregida, que la corregida nos es útil para conjuntos de datos pequeños o aquellos donde el número de *doubletons* (especies con abundancias de dos) es cero. Para este ejemplo, donde tenemos 225 especies, 50 parcelas y abundancias que suman cerca de 21500 individuos, es válido usar la versión no corregida de Chao.

Los estimadores de riqueza pueden calcularse tanto para cada unidad de muestreo como para la comunidad entera, de la misma forma como lo hicimos anteriormente para los índices de diversidad.

```
# Estimadores de riqueza por unidad de muestreo
est.Sites <- estimateR(BCI)
```

```
##                  1          2          3          4          5
## S.obs     93.000000  84.000000  90.000000  94.000000 101.000000
## S.chao1  117.473684 117.214286 141.230769 111.550000 136.000000
## se.chao1  11.583785  15.918953  23.001405   8.919663  15.467344
## S.ACE    122.848959 117.317307 134.669844 118.729941 137.114088
## se.ACE     5.736054   5.571998   6.191618   5.367571   5.848474
```

Aquí vemos para los primeros cinco sitios el número de especies observadas, los estimadores Chao y ACE, y sus respectivos errores estandar. Por defecto, *vegan* utiliza la ecuación de Chao no corregida, a menos que en una de las muestras no hayan *doubletons*. Si bien este paquete decide de manera automática qué ecuación usar, no está de más el validar si en nuestros datos existen unidades de muestreo sin *doubletons*, por si en algún momento decidimos calcular el índice de manera manual, para un conjunto de datos más pequeño, con otra librería, u otro software.

```
# Vector para almacenar el número de doubletons por unidad de muestreo
doub <- c()

# Ciclo para recorrer la matriz de datos
for(i in 1:nrow(BCI)){ # Ciclo desde 1 hasta el total de filas de los datos (unidades de muestre
o)
  doub.tmp <- length(which(BCI[i, ] == 2)) # Número de doubletons por unidad de muestreo
  doub <- append(doub, doub.tmp) # Añadimos el número de doubletons del sitio i al vector de dou
bletons
}
```

```
## Número de especies por unidad de muestreo con abundancia de dos individuos
```

```
##  [1] 18 13 12 19 17 17 15 22 16 16 13 20 19 10 20 22 22 27 15 20 22 18 22 16 20
## [26] 15 20 18 15 16 13 16 13 15 10 19 21 17 18 15 19 13 19 11 14 16 16 18 19 19
```

Para este caso todas nuestras unidades de muestreo tienen al menos una especie con abundancia de dos, por lo que no es necesario utilizar el estimador corregido. En la sección anterior también vimos que aunque por defecto se recomienda usar la versión corregida de Chao, este puede comportarse extraño cuando el coeficiente de variación de las abundancias es mayor a 0.5, así que es algo que también podríamos validar.

```
# Vector para almacenar los coeficientes de variación
CV <- c()

# Ciclo para recorrer la matriz de datos
for(i in 1:nrow(BCI)){ # Ciclo desde 1 hasta el total de filas de los datos (unidades de muestreo)
  zeros <- which(BCI[i, ] == 0) # Validamos qué especies tienen abundancia de 0 para excluirlas del CV
  CV.tmp <- sd(t(BCI[i, -zeros])) / mean(t(BCI[i, -zeros])) # CV de la abundancia por unidad de muestreo
  CV <- append(CV, CV.tmp) # Añadimos el CV del sitio i al vector de CV
}
```

```
##  [1] 1.172285 1.295953 1.486558 1.298631 1.507315 1.480389 1.307802 1.381340
##  [9] 1.796539 1.477646 1.412399 1.675829 1.372574 1.332512 1.313130 1.391203
## [17] 1.807054 1.377983 1.666446 1.236692 1.459292 1.773381 1.324683 1.387341
## [25] 1.376021 1.288875 1.513444 1.815144 1.871841 1.698216 1.352046 1.690600
## [33] 1.606952 1.623638 3.990597 1.502573 1.691263 2.063430 2.104041 2.444927
## [41] 1.325334 1.161212 1.653161 1.562824 1.688636 1.436008 1.538820 1.402234
## [49] 1.606687 1.414040
```

Pese a que los índices de Gini-Simpson y Pielou nos indican que la comunidad presenta una dominancia relativamente baja, la diferencia en las abundancias de algunas especies hacen que la desviación estándar sea mayor que el promedio, por lo que el coeficiente de variación es mayor a uno. De nuevo, esta información nos dice que para este conjunto de datos el estimador no corregido es adecuado.

Volviendo a los estimadores, para calcular el estimador general de la comunidad entera podemos sumar las abundancias de cada especie y utilizar las mismas funciones.

```
# Estimadores de riqueza para la comunidad entera
est.Comm <- estimateR(colSums(BCI))
```

```
##      S.obs    S.chao1   se.chao1      S.ACE     se.ACE
## 225.000000 237.214286   7.434824 238.217659   7.118588
```

El paquete *vegan* también tiene una función llamada `specpool`, con la que podemos obtener algunos estimadores adicionales como los Jacknife y el Bootstrap, con la limitación de que estos se calculan para la comunidad entera y no por unidades de muestreo.

```
# Estimadores de riqueza adicionales para la comunidad entera
est.Comm <- specpool(BCI, smallsample = TRUE)
```

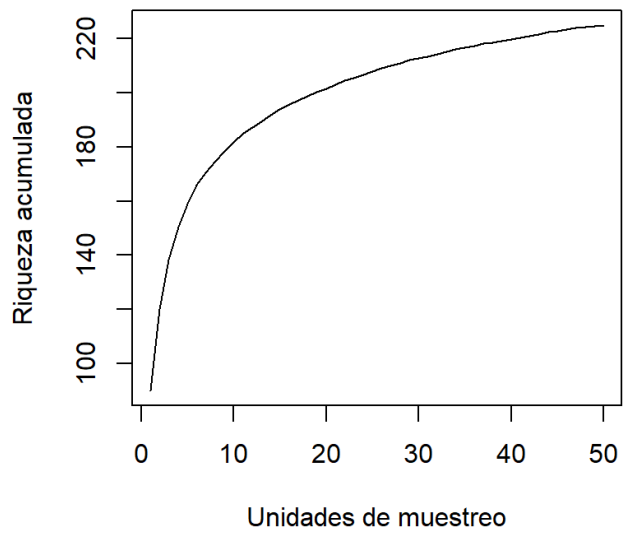
```
##  Species     chao chao.se  jack1 jack1.se    jack2     boot  boot.se  n
##      225 236.3732 6.54361 245.58 5.650522 247.8722 235.6862 3.468888 50
```

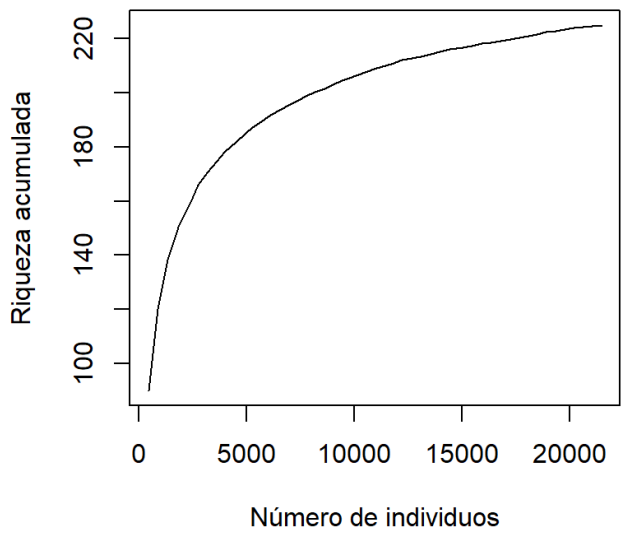
La función `specpool` recibe también el argumento `smallsample`, el cual puede tomar valores de `TRUE` o `FALSE` y nos permite elegir si utilizar por defecto la versión corregida del estimador Chao.

4. Curva de acumulación de especies

Para poder graficar la curva de acumulación de especies de nuestra comunidad, primero tenemos que calcular los valores de la diversidad promedio acumulados. Tal como se mencionó en la teoría, la forma más común de presentar las curvas de acumulación es con las unidades de muestreo en el eje X, sin embargo, también podemos graficarlas usando el número acumulado de individuos.

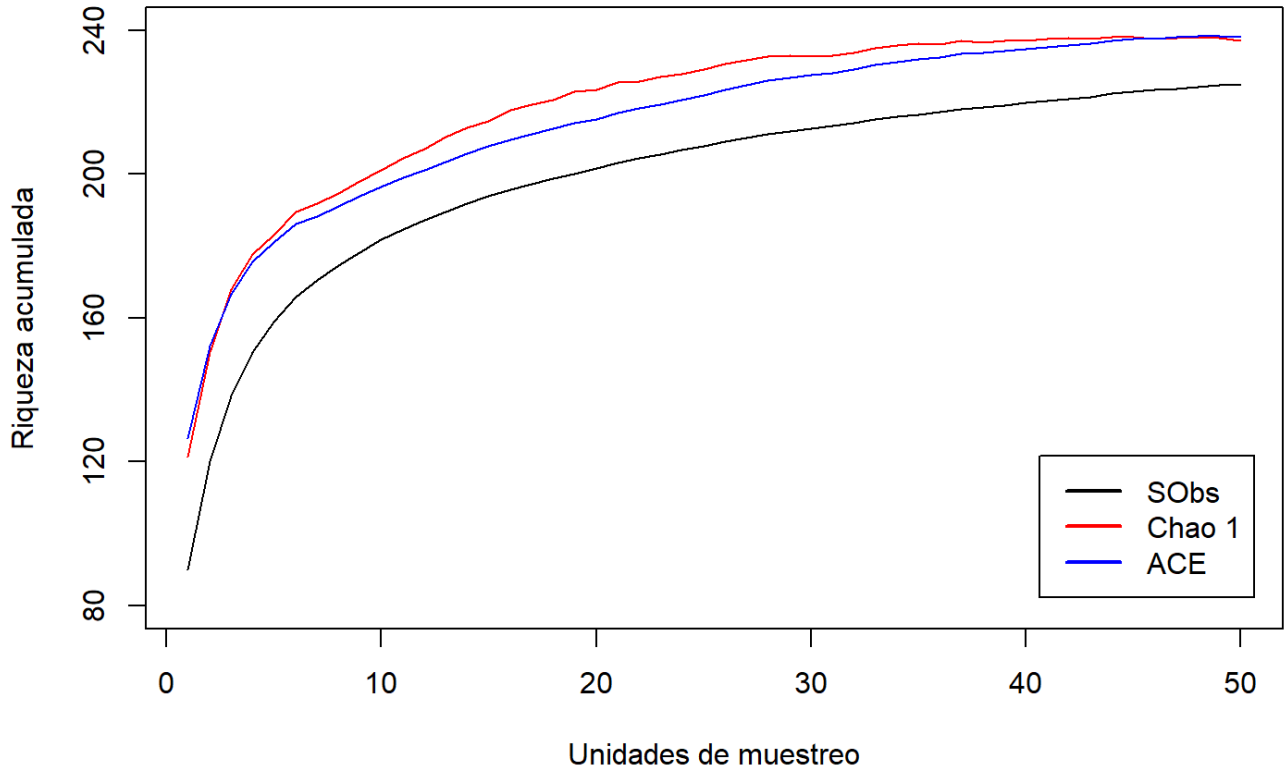
```
# Calculo de la diversidad acumulada para 100 permutaciones
accum <- estaccumR(BCI, permutations = 100)

# Diversidades acumuladas promedio
accum.mean <- as.data.frame(accum$means)

# Abundancias acumuladas
accum.abundance <- cumsum(rowSums(BCI))

# Curva de acumulación usando unidades de muestreo
plot(x = accum.mean$N, # Valores de X
     y = accum.mean$S,# Valores de Y
     xlab = "Unidades de muestreo", # Etiqueta del eje X
     ylab = "Riqueza acumulada", # Etiqueta del eje Y
     type = "l") # Tipo de línea (sólida)

# Curva de acumulación usando abundancias acumuladas
plot(x = accum.abundance, # Valores de X
     y = accum.mean$S, # Valores de Y
     xlab = "Número de individuos", # Etiqueta del eje X
     ylab = "Riqueza acumulada", # Etiqueta del eje Y
     type = "l") # Tipo de línea (sólida)
```





La función `estaccumR` nos da además los estimadores acumulados de Chao y ACE, por lo que a la curva que ya hemos generado podemos adicionarle fácilmente la curva de los estimadores de riqueza. Para ello, primero generamos la grafica base y posteriormente agregamos los demás elementos como si fueran capas.

```
# Curva de acumulación
plot(x = accum.mean$N, # Valores de X
     y = accum.mean$S, # Valores de Y
     xlab = "Unidades de muestreo", # Etiqueta del eje X
     ylab = "Riqueza acumulada", # Etiqueta del eje Y
     type = "l", # Tipo de línea (sólida)
     ylim = c(80, 240), # Límites del eje Y (según los datos)
     yaxt = "n") # Remover la división por defecto del eje Y (para usar personalizada)

# Intervalo personalizado del eje Y de la grafica (según los datos)
axis(side = 2, at = seq(80, 240, 40))

# Curva del estimador Chao
lines(x = accum.mean$N, # Valores de X
      y = accum.mean$Chao, # Valores de Y
      type = "l", # Tipo de línea (sólida)
      col = "red") # Color de la línea

# Curva del estimador ACE
lines(x = accum.mean$N, # Valores de X
      y = accum.mean$ACE, # Valores de Y
      type = "l", # Tipo de línea (sólida)
      col = "blue") # Color de la línea

# Leyenda de la grafica
legend(x = "bottomright", # Posición
       legend = c("SObs", "Chao 1", "ACE"), # Texto de la leyenda
       lty = c(1, 1, 1), # Tipos de línea de los símbolos
       col = c("black", "red", "blue"),  # Colores de los símbolos
       lwd = 2, # Grosor de las líneas de los símbolos
       inset = c(0.025, 0.05)) # Márgenes de la leyenda
```



Si deseamos obtener los intervalos de confianza para los valores de diversidad y sus estimadores, debemos también calcular los valores de desviación estándar. Esto porque la función `estaccumR` solo calcula los valores promedio y necesitamos de ambos para poder aplicar la fórmula de los intervalos.

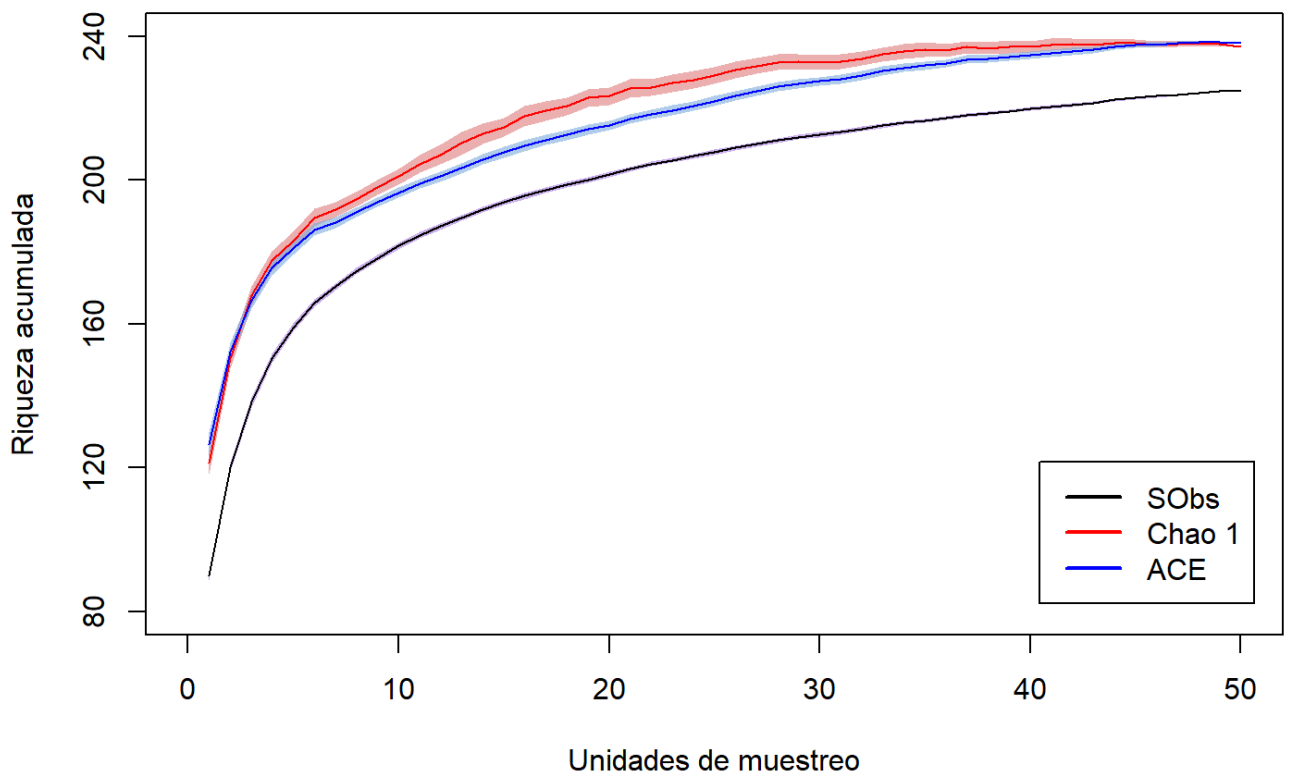
```
# Matriz para almacenar las desviaciones estandar
accum.sd <- matrix(ncol = 3, nrow = nrow(BCI))

# Nombres de las columnas de la matriz de desviaciones
colnames(accum.sd) <- c("S.SD", "Chao.SD", "ACE.SD")

# Ciclo para recorrer las matrices de permutaciones
for(i in 1:nrow(BCI)){ # Ciclo desde 1 hasta el total de filas de los datos (unidades de muestre
o)
  accum.sd[i, 1] <- sd(accum$S[i, ]) # Desviación estándar S
  accum.sd[i, 2] <- sd(accum$chao[i, ]) # Desviación estándar Chao
  accum.sd[i, 3] <- sd(accum$ace[i, ]) # Desviación estándar ACE
}

# Matriz para almacenar los intervalos de confianza
CI <- as.data.frame(matrix(ncol = 6, nrow = nrow(BCI)))

# Nombres de las columnas de la matriz de intervalos de confianza
colnames(CI) <- c("S.LCI", "S.UCI", "Chao.LCI", "Chao.UCI", "ACE.LCI", "ACE.UCI")

# Valor Z para el 95% de confianza
Z <- 1.96

# Ciclo para calcular los intervalos de confianza
for(i in 1:nrow(BCI)){ # Ciclo desde 1 hasta el total de filas de los datos (unidades de muestre
o)
  CI[i, 1] <- accum.mean[i, 2] - (Z * (accum.sd[i, 1] / sqrt(ncol(accum$S)))) # Limite inferior
 95% CI S
  CI[i, 2] <- accum.mean[i, 2] + (Z * (accum.sd[i, 1] / sqrt(ncol(accum$S)))) # Limite superior
 95% CI S
  CI[i, 3] <- accum.mean[i, 3] - (Z * (accum.sd[i, 2] / sqrt(ncol(accum$chao)))) # Limite inferi
or 95% CI Chao
  CI[i, 4] <- accum.mean[i, 3] + (Z * (accum.sd[i, 2] / sqrt(ncol(accum$chao)))) # Limite superi
or 95% CI Chao
  CI[i, 5] <- accum.mean[i, 4] - (Z * (accum.sd[i, 3] / sqrt(ncol(accum$ace)))) # Limite inferio
r 95% CI ACE
  CI[i, 6] <- accum.mean[i, 4] + (Z * (accum.sd[i, 3] / sqrt(ncol(accum$ace)))) # Limite superio
r 95% CI ACE
}
```

```
##      S.LCI     S.UCI Chao.LCI Chao.UCI  ACE.LCI  ACE.UCI
##   88.65514  91.26486 118.2890 124.7280 123.3370 129.9614
##  118.77918 121.34082 147.7460 153.3575 149.7865 154.9186
##  136.92629 139.75371 165.1015 170.5663 164.2989 169.0721
##  149.27234 151.70766 174.9935 180.1989 173.7075 177.3966
##  157.93798 160.16202 180.6191 185.6054 179.3860 182.9295
##  164.91414 166.92586 187.2040 192.0029 184.7151 187.8756
##  169.45965 171.40035 189.7054 193.9516 186.7862 189.8157
##  173.78485 175.59515 192.6480 196.6600 189.7042 192.4718
##  177.42679 179.17321 195.8585 200.1527 192.5529 195.1920
##  180.83637 182.60363 198.8732 203.2294 195.2416 197.9265
```

Los intervalos calculados usando el código anterior corresponden al 95% de confianza, en caso de querer un nivel diferente basta con modificar el valor *Z* por su equivalente al nivel de confianza deseado, por ejemplo, 90% = 1.64, 95% = 1.96, 99% = 2.58 (existen tablas con estos valores ya calculados). Como ya hemos visto, los intervalos de confianza nos permiten incorporar incertidumbre y nos dan información indirecta sobre nuestra comunidad, e incluirlos en nuestras gráficas es relativamente sencillo. De nuevo, aquí tenemos que ir agregando cada elemento uno tras otro, comenzando por aquellos que estarían ubicados en el plano de más atrás.

```
# Lienzo vacío para la grafica (con los limites de X e Y según los datos)
plot(NA, # Lienzo sin datos
     xlab = "Unidades de muestreo", # Etiqueta del eje X
     ylab = "Riqueza acumulada", # Etiqueta del eje Y
     xlim = c(0, 50), # Límites del eje X (según los datos)
     ylim = c(80, 240), # Límites del eje Y (según los datos)
     yaxt = "n") # Remover la división por defecto del eje Y (para usar personalizada)

# Intervalo personalizado del eje Y de la grafica (según los datos)
axis(side = 2, at = seq(80, 240, 40))

# Polígono con los intervalos de confianza para S
polygon(x = c(seq(1, nrow(BCI)) , rev(seq(1, nrow(BCI)))), # Valores X del intervalo
        y = c(CI$S.LCI, rev(CI$S.UCI)), # Valores Y del intervalo
        col = rgb(166/255, 122/255, 222/255, 0.6), # Color (en RGB) y transparencia del polígono
        border = NA) # No mostrar el borde del polígono

# Polígono con los intervalos de confianza para Chao
polygon(x = c(seq(1, nrow(BCI)) , rev(seq(1, nrow(BCI)))), # Valores X del intervalo
        y = c(CI$Chao.LCI, rev(CI$Chao.UCI)), # Valores Y del intervalo
        col = rgb(222/255, 122/255, 122/255, 0.6), # Color (en RGB) y transparencia del polígono
        border = NA) # No mostrar el borde del polígono

# Polígono con los intervalos de confianza para ACE
polygon(x = c(seq(1, nrow(BCI)) , rev(seq(1, nrow(BCI)))), # Valores X del intervalo
        y = c(CI$ACE.LCI, rev(CI$ACE.UCI)), # Valores Y del intervalo
        col = rgb(122/255, 170/255, 222/255, 0.6), # Color (en RGB) y transparencia del polígono
        border = NA) # No mostrar el borde del polígono

# Curva de acumulación (especies observadas)
lines(x = accum.mean$N, # Valores de X
      y = accum.mean$S, # Valores de Y
      type = "l", # Tipo de línea (sólida)
      col = "black") # Color de la línea

# Curva del estimador Chao
lines(x = accum.mean$N, # Valores de X
      y = accum.mean$Chao, # Valores de Y
      type = "l", # Tipo de línea (sólida)
      col = "red") # Color de la línea

# Curva del estimador ACE
lines(x = accum.mean$N, # Valores de X
      y = accum.mean$ACE, # Valores de Y
      type = "l", # Tipo de línea (sólida)
      col = "blue") # Color de la línea

# Leyenda de la grafica
legend(x = "bottomright", # Posición
       legend = c("SObs", "Chao 1", "ACE"), # Texto de la leyenda
       lty = c(1, 1, 1), # Tipos de línea de los símbolos
       col = c("black", "red", "blue"), # Colores de los símbolos
```

```
        lwd = 2, # Grosor de las líneas de los símbolos
        inset = c(0.025, 0.05)) # Márgenes de la leyenda
```



5. Curvas de rarefacción

Para poder continuar con el ejercicio de la rarefacción tendremos que cambiar de conjunto de datos, porque como lo hemos visto, la idea de este método es comparar entre comunidades diferentes con diferente esfuerzo de muestreo, y los datos con los que hemos venido trabajando no cumplen con esta condición.

A partir de ahora utilizaremos parte de los datos de Janzen (1973) para dos comunidades tropicales de escarabajos, los cuales fueron muestreados en dos tipos de bosque diferentes, un bosque maduro y un bosque en crecimiento secundario. Estos datos no vienen precargados como los anteriores, así que tendremos que generarlos a partir de los originales. En este conjunto de datos la primera comunidad tiene 140 especies y 976 individuos (bosque secundario), mientras que la segunda 112 especies y 237 individuos (bosque maduro).

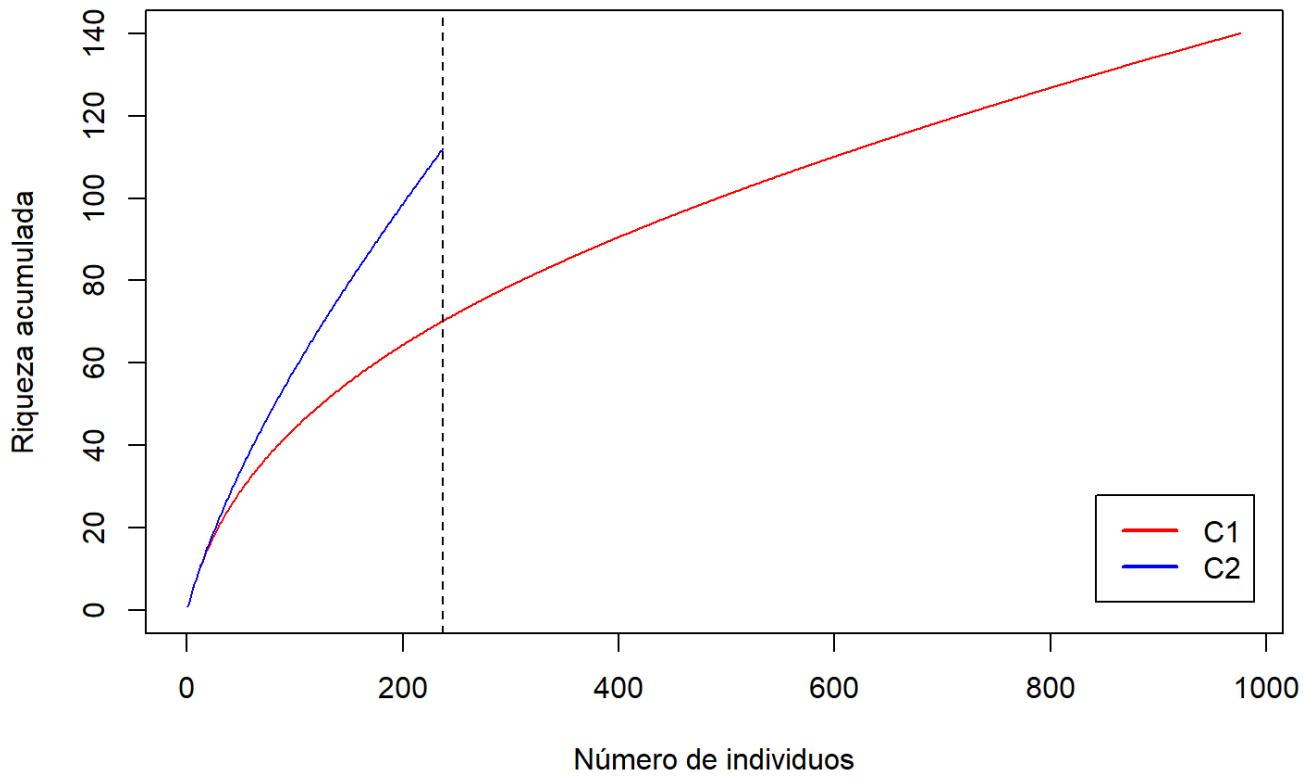
```
# Semilla para generar pseudoaleatorizaciones (solo para que el ejemplo sea replicable)
set.seed(42)

# Vector de abundancias para la Comunidad 1
C1.Jan <- c(rep(1, 70), rep(2, 17), rep(3, 4), rep(4, 5), rep(5, 5), rep(6, 5), rep(7, 5), rep(8
, 3), rep(9, 1), rep(10, 2), rep(11, 3), rep(12, 2), rep(14, 2), rep(17, 1), rep(19, 2), rep(20,
3), rep(21, 1), rep(24, 1), rep(26, 1), rep(40, 1), rep(57, 2), rep(60, 1), rep(64, 1), rep(71,
1), rep(77, 1))

# Aleatorizamos el orden de los datos
C1.Jan <- sample(C1.Jan, 140)

# Vector de abundancias para la Comunidad 2
C2.Jan <- c(rep(1, 84), rep(2, 10), rep(3, 4), rep(4, 3), rep(5, 5), rep(6, 1), rep(7, 2), rep(8
, 1), rep(14, 1), rep(42, 1), rep(0, 28))

# Aleatorizamos el orden de los datos
C2.Jan <- sample(C2.Jan, 140)

# Añadimos ambos vectores a una tabla
Janzen <- as.data.frame(rbind(C1 = C1.Jan, C2 = C2.Jan))
```

```
##  V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20
##   1  1  2 12 77 20  1  1  2   5   3   7   1   1   8   7   1   1  20   1
##   3  1  2  6  1  1  0  3  1   1   0   1   1   0   0   1   1   1   0   0
```

El código anterior genera una matriz de datos de abundancias para dos comunidades (se muestran solo las primeras 20 especies), para ello usamos el comando `rep` que repite un número una N cantidad de veces, de modo que, si sabemos el número de especies y sus abundancias podemos recrear los datos. En este caso hemos añadido los datos en orden, primero los que tienen abundancia de uno, luego dos, tres, y así sucesivamente, lo cual no tiene efecto alguno a la hora de graficar ya que al rarificar se aleatoriza el orden, pero que sí importaría a la hora de interpretar otros resultados si conociésemos la identidad de cada especie. Por lo tanto, para darle un tinte más real a los datos, una vez generados vamos a aleatorizarlos. Para efectos de replicabilidad del ejercicio establecí una semilla para generar números pseudoaleatorios usando el comando `set.seed`, de modo que siempre se generen los mismos números aleatorios y así podamos trabajar exactamente con los mismos datos. También es importante tener presente que, para poder correr los análisis, la tabla de datos debe estar balanceada, es decir, debe tener el mismo número de datos. Es por esta razón que al final del vector para la Comunidad 2 adicionamos 28 especies con abundancia de cero.

Una vez tenemos nuestros datos podemos rarificar fácilmente usando la función `rarecurve` del paquete *vegan*, la cual genera una curva de acumulación basada en tamaño (número acumulado de individuos en el eje X) para cada una de las filas en nuestra matriz de datos.

```
# Curvas de acumulación basadas en tamaño
rare <- rarecurve(Janzen, # Matriz de datos
                  col = c("red", "blue"), # Colores de las líneas
                  label = FALSE, # Sin etiquetas de datos
                  xlab = "Número de individuos", # Etiqueta del eje X
                  ylab = "Riqueza acumulada", # Etiqueta del eje Y
                  ylim = c(0, 140), # Límites del eje Y (según los datos)
                  yaxt = "n") # Remover la división por defecto del eje Y (para usar personaliza
da)

# Intervalo personalizado del eje Y de la grafica (según los datos)
axis(side = 2, at = seq(0, 140, 20))

# Obtenemos el valor de abundancia mínima común entre las comunidades
ab.min <- min(rowSums(Janzen))

# Línea punteada en el valor de abundancia mínima común
abline(v = ab.min, lty = 2)

# Leyenda de la grafica
legend(x = "bottomright", # Posición
       legend = c("C1", "C2"), # Texto de la leyenda
       lty = c(1, 1), # Tipos de línea de los símbolos
       col = c("red", "blue"), # Colores de los símbolos
       lwd = 2, # Grosor de las líneas de los símbolos
       inset = c(0.025, 0.05))# Márgenes de la leyenda

# Valores de diversidad al mismo tamaño de muestra
S.rare <- cbind(C1 = rare[[1]][ab.min], C2 = rare[[2]][ab.min])
```



Aunque de la gráfica podríamos estimar los valores aproximados de diversidad para cada comunidad al mismo tamaño, es mucho más fácil y preciso extraerlos directamente de los datos.

```
# Valores de diversidad al mismo tamaño de muestra
S.rare <- cbind(C1 = rare[[1]][ab.min], C2 = rare[[2]][ab.min])
```

```
##        C1  C2
##  70.18935 112
```

Recordemos que este tipo de rarefacción basada en tamaño (número de individuos) no es la mejor opción, porque la cobertura de muestra de cada comunidad puede ser diferente. Para verificar la cobertura lo haremos de manera manual utilizando su fórmula.

```
# Vectores para almacenar el número de singletons y doubletons por comunidad
sing <- c()
doub <- c()

# Ciclo para recorrer la matriz de datos
for(i in 1:nrow(Janzen)){ # Ciclo desde 1 hasta el total de filas de los datos (comunidades)
  sing.tmp <- length(which(Janzen[i, ] == 1)) # Número de singletons por comunidad
  sing <- append(sing, sing.tmp) # Añadimos el número de singletons del sitio i al vector de sin
gletons
  doub.tmp <- length(which(Janzen[i, ] == 2)) # Número de doubletons por comunidad
  doub <- append(doub, doub.tmp) # Añadimos el número de doubletons del sitio i al vector de dou
bletons
}

# Abundancias totales por comunidad
abundances <- rowSums(Janzen)

# Matriz para almacenar los datos de cobertura por comunidad
coverage <- matrix(ncol = nrow(Janzen), nrow = 1)
colnames(coverage) <- c("C1", "C2") # Nombres de las columnas de la matriz de coberturas

# Ciclo para calcular la cobertura por comunidad
for(i in 1:nrow(Janzen)){ # Ciclo desde 1 hasta el total de filas de los datos (comunidades)
  f1 <- sing[i]
  f2 <- doub[i]
  n <- abundances[i]
  coverage[1, i] <- 1 - ((f1 / n) * (((n - 1) * f1) / (((n - 1) * f1) + (2 * f2)))) # Cobertura
 de muestra
}
```

```
##     C1    C2
##  0.928 0.646
```

Ahora, vamos a rarificar pero usando la cobertura de muestra, y para ello necesitaremos la librería *iNEXT*. Es importante saber que a diferencia del paquete *vegan*, esta librería recibe los datos con las especies en las filas y las unidades de muestreo en las columnas, de modo que tendremos primero que transponer nuestra matriz.

```
# Matriz de datos transpuesta
Janzen.t <- as.data.frame(t(Janzen))

# Comando general de iNEXT (calcula muchas cosas)
est.Comms <- iNEXT(Janzen.t, q = 0, datatype = "abundance") # q = 0 es la riqueza (diversidades
 verdaderas)
```

```
## Compare 2 assemblages with Hill number order q = 0.
## $class: iNEXT
##
## $DataInfo: basic data information
##   site   n S.obs     SC f1 f2 f3 f4 f5 f6 f7 f8 f9 f10
## 1   C1 976   140 0.9283 70 17  4  5  5  5  5  3  1   2
## 2   C2 237   112 0.6459 84 10  4  3  5  1  2  1  0   0
##
## $iNextEst: diversity estimates with rarefied and extrapolated samples.
## $C1
##       m       method order      qD  qD.LCL  qD.UCL    SC SC.LCL SC.UCL
## 1     1 interpolated     0   1.000   1.000   1.000 0.034  0.031  0.037
## 10  488 interpolated     0  99.658  93.377 105.940 0.902  0.889  0.915
## 20  976     observed     0 140.000 128.627 151.373 0.928  0.916  0.941
## 30 1438 extrapolated     0 169.592 153.601 185.583 0.943  0.929  0.957
## 40 1952 extrapolated     0 195.425 173.687 217.162 0.956  0.941  0.971
##
## $C2
##      m       method order      qD  qD.LCL  qD.UCL    SC SC.LCL SC.UCL
## 1    1 interpolated     0   1.000   1.000   1.000 0.040  0.026  0.054
## 10 118 interpolated     0  66.440  60.968  71.912 0.576  0.527  0.626
## 20 237     observed     0 112.000 101.071 122.929 0.646  0.594  0.698
## 30 349 extrapolated     0 149.518 133.163 165.872 0.684  0.625  0.743
## 40 474 extrapolated     0 186.679 163.456 209.902 0.721  0.654  0.788
##
##
## $AsyEst: asymptotic diversity estimates along with related statistics.
##   Site         Diversity Observed Estimator    s.e.     LCL     UCL
## 1   C1  Species richness  140.000   283.970  50.474 213.868 420.600
## 2   C1 Shannon diversity   51.194    59.202   2.939  53.442  64.962
## 3   C1 Simpson diversity   28.876    29.726   1.632  28.876  32.924
## 4   C2  Species richness  112.000   463.311 136.273 280.660 843.766
## 5   C2 Shannon diversity   58.277   123.467  20.392  83.499 163.435
## 6   C2 Simpson diversity   22.549    24.815   4.976  22.549  34.568
##
## NOTE: Only show five estimates, call iNEXT.object$iNextEst. to show complete output.
```

La salida de datos de esta función es una lista compuesta por tres objetos. En el primero (*DataInfo*) encontraremos información general sobre nuestra comunidad, como el número de especies, abundancias totales, singletons y doubletons. En el segundo (*iNextEst*) se encuentran los estimadores de diversidad para diferentes valores de riqueza, tanto la observada como valores interpolados y extrapolados, y sus respectivas coberturas. Finalmente, en el tercer objeto (*AsyEst*) están los índices de diversidad verdadera asintóticos para nuestras comunidades.

Ya que la salida de datos de este paquete tiene un formato estándar, también podemos extraer directamente las coberturas de muestra para cada comunidad de manera programática.

```
# Cobertura de muestra C1
SC.C1 <- est.Comms$iNextEst$C1[which(est.Comms$iNextEst$C1$method == "observed"), 7]

# Cobertura de muestra C2
SC.C2 <- est.Comms$iNextEst$C2[which(est.Comms$iNextEst$C2$method == "observed"), 7]

# Añadimos las coberturas a una tabla
coverage <- cbind(C1 = SC.C1, C2 = SC.C2)
```

```
##     C1    C2
##  0.928 0.646
```

Para rarificar con base en la cobertura de muestra vamos a utilizar los datos de los estimadores que calculamos usando *iNEXT*. Por defecto, esta función calcula los estimados para 40 valores de riqueza, desde uno hasta el doble del total de individuos por comunidad y en intervalos igualmente espaciados. No obstante, es posible que al dividir 40 entre el total de valores de diversidad no obtengamos coberturas de muestra con el valor exacto que queremos para rarificar, de modo que incluiremos un parámetro adicional (`knots`) a la función para que estime los valores de uno en uno. Por ejemplo, si para la primera comunidad tenemos una abundancia de 976 individuos, esta función estimará los valores hasta 1952 en intervalos igualmente espaciados. Por lo tanto, al decirle que lo haga para un número de intervalos igual al máximo del número de individuos a estimar, obtendremos intervalos de una unidad.

```
# Abundancias totales por comunidad
abundances <- colSums(Janzen.t)

# Valor máximo para extrapolar (no más del doble de las abundancias)
max.Extrapol <- abundances * 2

# Estimadores para cada comunidad
rare.C1 <- iNEXT(Janzen.t[, 1], q = 0, datatype = "abundance", knots = max.Extrapol[1])
rare.C2 <- iNEXT(Janzen.t[, 2], q = 0, datatype = "abundance", knots = max.Extrapol[2])
```

Una vez hecho esto podemos extraer el valor de riqueza estimado para nuestras dos comunidades si ambas hubiesen sido muestreadas al mismo nivel de cobertura, que en este caso es de 65%. Por temas de decimales es posible que obtengamos más de un valor de riqueza para una misma cobertura en una comunidad, en cuyo caso simplemente promediamos.

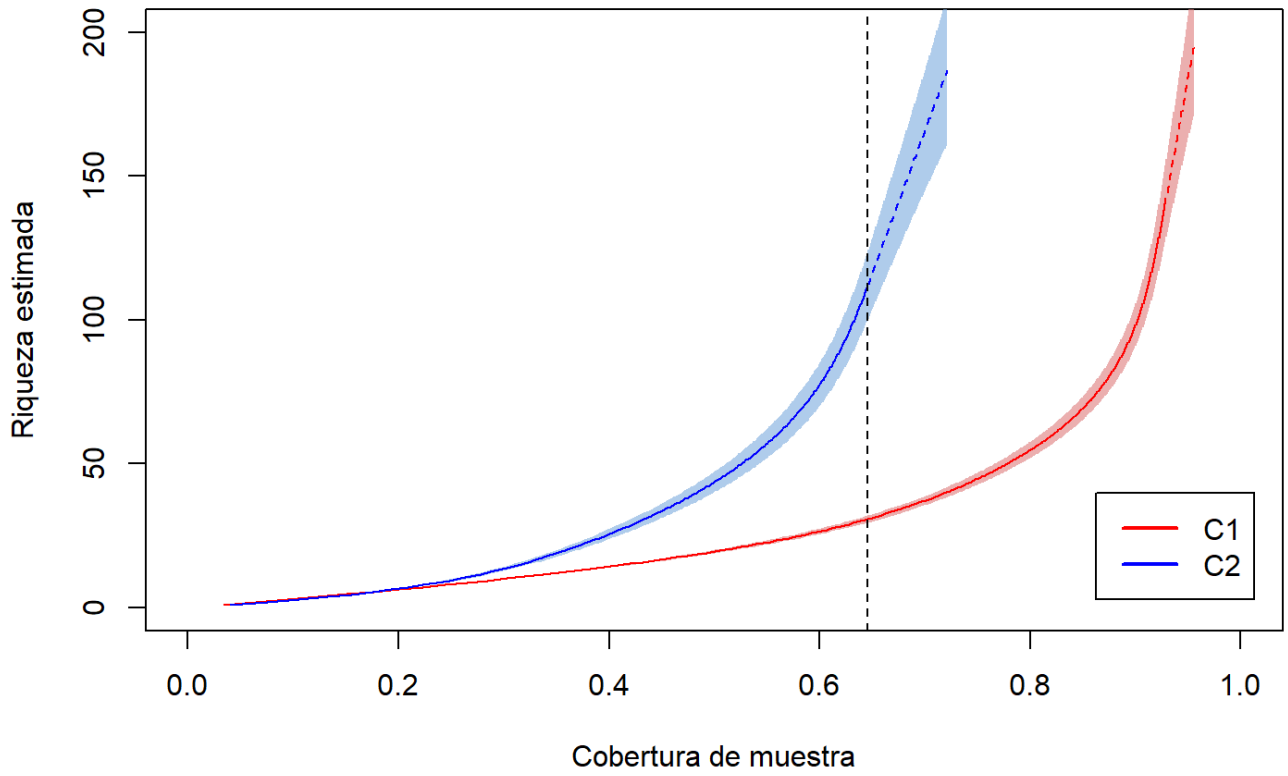
```
# Filtro para extraer los valores de ambas comunidades al 65% de cobertura (redondeado a dos dec
imales)
S.C1.rare <- dplyr::filter(rare.C1$iNextEst, round(SC, 2) == 0.65)$qD
S.C2.rare <- dplyr::filter(rare.C2$iNextEst, round(SC, 2) == 0.65)$qD

# Añadimos los valores promediados a una tabla
S.Comms.rare <- cbind(C1 = mean(S.C1.rare), C2 = mean(S.C2.rare))
```

```
##        C1       C2
##  31.25367 116.0405
```

Finalmente, podemos llevar el resultado de la rarificación basada en cobertura a una grafica, en la cual incluiremos los valores interpolados, extrapolados, y sus respectivos intervalos de confianza.

```
# Filtro para separar los datos interpolados y extrapolados de ambas comunidades
# (abundancia menor o igual a la observada = interpolación, mayor = extrapolación)
C1.inter <- dplyr::filter(rare.C1$iNextEst, m <= abundances[1])
C1.extra <- dplyr::filter(rare.C1$iNextEst, m > abundances[1])
C2.inter <- dplyr::filter(rare.C2$iNextEst, m <= abundances[2])
par(mfrow = c(1, 1), mar = c(4, 4, 1, 2), oma = c(0, 2, 0, 0))

# Lienzo vacío para la grafica (con los limites de X e Y según los datos)
plot(NA, # Lienzo sin datos
     xlab = "Cobertura de muestra", # Etiqueta del eje X
     ylab = "Riqueza estimada", # Etiqueta del eje Y
     xlim = c(0, 1), # Límites del eje X (según los datos)
     ylim = c(0, 200)) # Límites del eje Y (según los datos)

# Polígono con los intervalos de confianza para la Comunidad 1
polygon(c(rare.C1$iNextEst$SC , rev(rare.C1$iNextEst$SC)), # Valores X del intervalo
        c(rare.C1$iNextEst$qD.LCL, rev(rare.C1$iNextEst$qD.UCL)), # Valores Y del intervalo
        col = rgb(222/255, 122/255, 122/255, 0.6), # Color (en RGB) y transparencia del polígono
        border = NA) # No mostrar el borde del polígono

# Polígono con los intervalos de confianza para la Comunidad 2
polygon(c(rare.C2$iNextEst$SC , rev(rare.C2$iNextEst$SC)), # Valores X del intervalo
        c(rare.C2$iNextEst$qD.LCL, rev(rare.C2$iNextEst$qD.UCL)), # Valores Y del intervalo
        col = rgb(122/255, 170/255, 222/255, 0.6), # Color (en RGB) y transparencia del polígono
        border = NA) # No mostrar el borde del polígono

# Curva de acumulación basada en cobertura para la Comunidad 1 (valores interpolados)
lines(rare.C1$iNextEst[which(rare.C1$iNextEst$m <= abundances[1]), 7], # Valores de X
      rare.C1$iNextEst[which(rare.C1$iNextEst$m <= abundances[1]), 4], # Valores de Y
      type = "l", # Tipo de línea (sólida)
      col = "red") # Color de la línea

# Curva de acumulación basada en cobertura para la Comunidad 1 (valores extrapolados)
lines(rare.C1$iNextEst[which(rare.C1$iNextEst$m > abundances[1]), 7], # Valores de X
      rare.C1$iNextEst[which(rare.C1$iNextEst$m > abundances[1]), 4], # Valores de Y
      lty = 2, # Tipo de línea (punteada)
      col = "red") # Color de la línea

# Curva de acumulación basada en cobertura para la Comunidad 2 (valores interpolados)
lines(rare.C2$iNextEst[which(rare.C2$iNextEst$m <= abundances[2]), 7], # Valores de X
      rare.C2$iNextEst[which(rare.C2$iNextEst$m <= abundances[2]), 4], # Valores de Y
      type = "l", # Tipo de línea (sólida)
      col = "blue") # Color de la línea

# Curva de acumulación basada en cobertura para la Comunidad 2 (valores extrapolados)
lines(rare.C2$iNextEst[which(rare.C2$iNextEst$m > abundances[2]), 7], # Valores de X
      rare.C2$iNextEst[which(rare.C2$iNextEst$m > abundances[2]), 4], # Valores de Y
      lty = 2, # Tipo de línea (punteada)
      col = "blue") # Color de la línea

# Línea punteada en el valor de cobertura mínima común
abline(v = min(coverage), lty = 2)
```

```
# Leyenda de la grafica
legend(x = "bottomright", # Posición
       legend = c("C1", "C2"), # Texto de la leyenda
       lty = c(1, 1), # Tipos de línea de los símbolos
       col = c("red", "blue"), # Colores de los símbolos
       lwd = 2, # Grosor de las líneas de los símbolos
       inset = c(0.025, 0.05)) # Márgenes de la leyenda
```



El paquete *iNEXT* también incluye una función llamada *ggiNEXT*, la cual automáticamente genera la gráfica anterior simplificando bastante el proceso. Esta función grafica utilizando *ggplot2*, de modo que si conocemos la sintaxis de esta librería podemos personalizarla mucho más.

6. Índices de diversidad verdadera

Para calcular los índices de diversidad verdadera tenemos dos opciones, la primera es transformar los índices tradicionales que anteriormente aprendimos a calcular, o la segunda, utilizando la función `renyi` del paquete *vegan*. Con el objetivo de mostrar que en ambos casos el resultado es el mismo lo haremos las dos formas. Para esto continuaremos con el conjunto de datos de Janzen (1973), ya que el resultado de esta sección nos será de utilidad para construir los perfiles de diversidad en la siguiente.

Vamos a comenzar calculando los índices tradicionales y luego realizando sus respectivas transformaciones.

```
# Vector para almacenar el número de especies por unidad de muestreo
S <- c()

# Ciclo para recorrer la matriz de datos
for(i in 1:nrow(Janzen)){ # Ciclo desde 1 hasta el total de filas de los datos (comunidades)
  S.tmp <- length(which(Janzen[i, ] > 0)) # Número de especies (columnas) con abundancia > 0 por unidad de muestreo
  S <- append(S, S.tmp) # Añadimos el número de especies del sitio i al vector de especies
}

# Índice de Gini-Simpson
Simpson <- diversity(Janzen, index  = "simpson")

# Índice de Shannon
Shannon <- diversity(Janzen, index = "shannon")

# Índice de Pielou
Pielou <- Shannon / log(S)

# Combinamos los índices en una tabla
indices <- cbind(S = S, Simpson = Simpson, Shannon = Shannon, Pielou = Pielou)

# Transformamos los índices tradicionales en índices de diversidad verdadera
indices <- dplyr::mutate(as.data.frame(indices),
                         q0 = S, # Riqueza (q = 0)
                         q1 = exp(Shannon), # Exponencial de Shannon (q = 1)
                         q2 = 1 / (1 - Simpson)) # Inverso de Simpson (q = 2)
```

```
##    S   Simpson  Shannon    Pielou  q0       q1       q2
##  140 0.9653697 3.935616 0.7964185 140 51.19366 28.87644
##  112 0.9556517 4.065210 0.8615473 112 58.27713 22.54878
```

Ahora, haremos el calculo de las diversidades verdaderas usando el paquete *vegan* y las compararemos con las que obtuvimos en el paso anterior.

```
# Vector para almacenar el número de especies por unidad de muestreo
indices <- renyi(Janzen, # Matriz de datos
                 scales = c(0, 1, 2), # Ordenes de diversidad
                 hill = TRUE) # Números de Hill (cuando es FALSE calcula entropías de Renyi)

# Nombre de las columnas de la tabla de índices
colnames(indices) <- c("q0", "q1", "q2")
```

```
##   q0       q1       q2
##  140 51.19366 28.87644
##  112 58.27713 22.54878
```

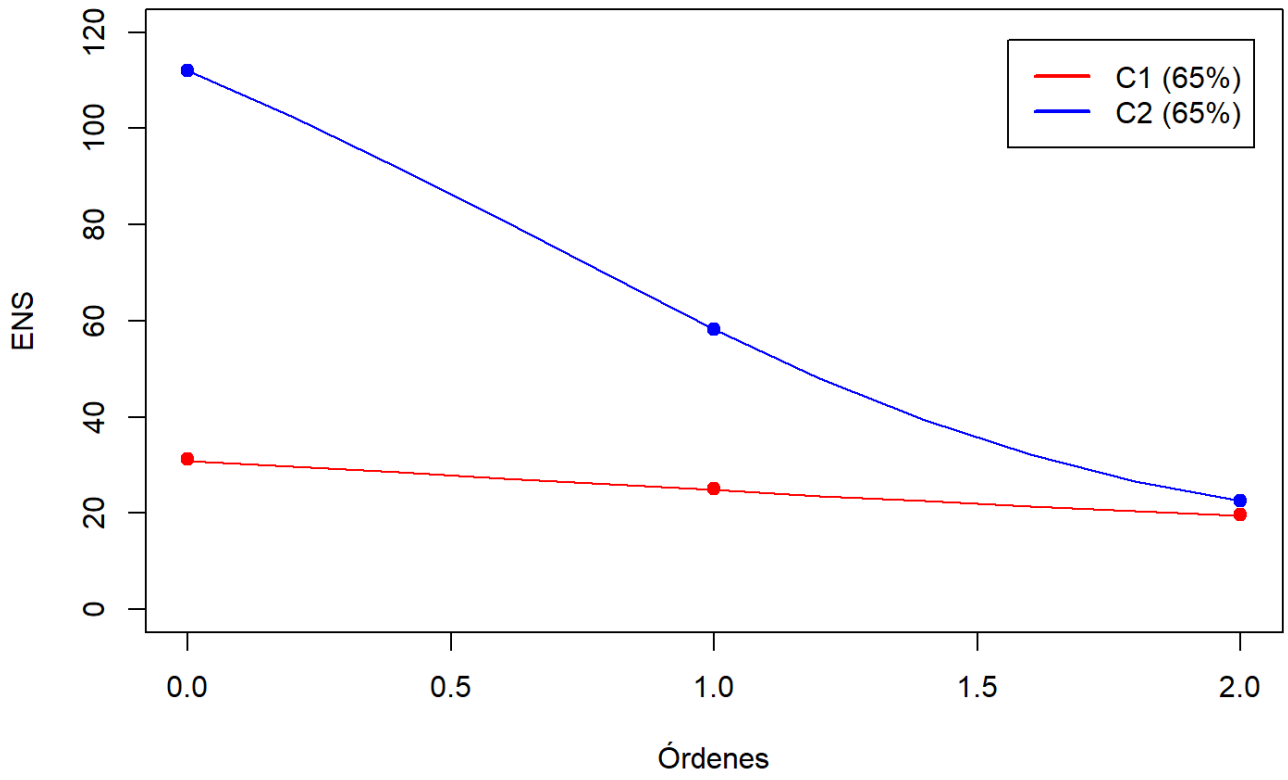
Al comparar el resultado vemos que los valores obtenidos en ambos casos son los mismos, siendo el segundo método mucho más rápido. Dado que aquí solo teníamos los datos generales para dos comunidades, el resultado obtenido es un único índice por cada orden por cada comunidad, pero por ejemplo, si utilizásemos el conjunto de datos BCI obtendríamos un índice por cada orden por cada unidad de muestreo.

El paquete *iNEXT* también calcula diversidades verdaderas, anteriormente solo utilizamos el orden cero, pero podemos especificar otros (`q = c(0, 1, 2)`). La diferencia de usar esta librería radica en el hecho de que nos permite obtener fácilmente los valores de las diversidades verdaderas a diferentes abundancias o coberturas de muestra.

7. Perfiles de diversidad

Finalmente, lo único que nos resta es calcular los perfiles de diversidad, para lo cual necesitamos los tres órdenes con significado biológico de las diversidades verdaderas. No obstante, para darle a la gráfica un acabado más suavizado calcularemos algunos órdenes adicionales en intervalos de 0.2 desde el orden cero hasta el orden dos, y utilizaremos los valores para ambas comunidades al mismo esfuerzo de muestreo.

```
# Tamaño (número de individuos) de cada comunidad a la misma cobertura de muestra
# (los valores de cobertura los tomamos de la variable coverage previamente calculada)
size.C1 <- Coverage2Size(Janzen.t[, 1], SampleCoverage = min(coverage))
size.C2 <- Coverage2Size(Janzen.t[, 2], SampleCoverage = min(coverage))

# Estimados de diversidad para cada comunidad a la misma cobertura de muestra
# (iNEXT no permite el calculo de un único tamaño, así que ponemos un cero que luego filtraremo
s)
C1.DivProf <- iNEXT(Janzen.t[, 1], q = seq(0, 2, by = 0.2), size = c(0, size.C1))$iNextEst
C2.DivProf <- iNEXT(Janzen.t[, 2], q = seq(0, 2, by = 0.2), size = c(0, size.C2))$iNextEst

# Filtro para eliminar los valores de cero que creamos en el paso anterior
C1.DivProf <- dplyr::filter(C1.DivProf, m > 0)
C2.DivProf <- dplyr::filter(C2.DivProf, m > 0)

# Lienzo vacío para el perfil de diversidad
plot(NA, # Lienzo sin datos
     xlab = "Órdenes", # Etiqueta del eje X
     ylab = "ENS", # Etiqueta del eje Y
     xlim = c(0, 2), # Límites del eje X (según los datos)
     ylim = c(0, 120)) # Límites del eje Y (según los datos)

# Perfil de la Comunidad 1
lines(x = C1.DivProf$order, # Valores de X
      y = C1.DivProf$qD, # Valores de Y
      type = "l", # Tipo de línea (sólida)
      col = "red") # Color de la línea

# Perfil de la Comunidad 2
lines(C2.DivProf$order, # Valores de X
      C2.DivProf$qD, # Valores de Y
      type = "l", # Tipo de línea (sólida)
      col = "blue") # Color de la línea

# Puntos de la Comunidad 1
points(x = c(0, 1, 2), # Valores de X
       y = c(31.255, 24.994, 19.648), # Valores de Y
       pch = 19, # Forma del símbolo (19 = círculo)
       col = "red") # Color del símbolo

# Puntos de la Comunidad 2
points(x = c(0, 1, 2), # Valores de X
       y = c(112, 58.277, 22.549), # Valores de Y
       pch = 19, # Forma del símbolo (19 = círculo)
       col = "blue") # Color del símbolo

legend(x = "topright", # Posición
       legend = c("C1 (65%)", "C2 (65%)"), # Texto de la leyenda
       lty = c(1, 1), # Tipos de línea de los símbolos
       col = c("red", "blue"), # Colores de los símbolos
       lwd = 2, # Grosor de las líneas de los símbolos
       inset = c(0.025, 0.05)) # Márgenes de la leyenda
```



8. Consideraciones finales

Con esto damos por terminada esta guía. Los análisis de diversidad se pueden complejizar aún más, métricas y gráficas adicionales, diversidades a otras escalas o dimensiones, análisis espaciales, filogenéticos, etcétera, aunque por lo pronto, esto es lo básico que debemos saber. Espero que esta información haya sido clara y de utilidad, para que podamos entregar mejores productos, tomar mejores decisiones, pero más importante aún, entender qué es lo que estamos haciendo. En la literatura que se encuentra al final de la primera sección hay muchos recursos de utilidad para quien desee ahondar en el los temas.

9. Referencias

Janzen, D. H. (1973). Sweep Samples of Tropical Foliage Insects: Description of Study Sites, With Data on Species Abundances and Size Distributions. *Ecology*, 54(3), 659–686.