iNEXT: An R package for interpolation and extrapolation in measuring species diversity

T. C. Hsieh, K. H. Ma, Anne Chao

2015 台灣生態研究網年會, 14 March 2015

Agenda

- What is species diversity?
- Why we need interpolation and extrapolation?
- Demo and case study.
- Intro to shiny, a web app for R.

What is species diversity?

- **Species diversity** is defined as the number of species and abundance of each species in a given community

- The number of species that live in a certain location, **species richness** is the most widely used diversity measure.

- The **effective number of species** refers to the number of equally abundant species needed to obtain the same mean proportional species abundance observed in the dataset of interest.

Chao and Jost (2012, Ecology Vol. 93)

Effective transform

10 different abundant species

$f(\boldsymbol{p})$

10 equally abundant species

Effective number (Hill 1973)

$$ {}^{q}D = \left(\sum_{i=1}^{S} p_i^q \right)^{1/(1-q)} $$

- If ${}^{q}D = x$, the value is equivalent to that of an idealized assemblage with x equally abundant species, $x = [x \cdot (1/x)^q]^{1/(1-q)}$
- The parameter q determines the sensitivity of the relative frequencies

Order	q = 0	q = 1	q = 2
✔ Name	✔ Richness	✔ exp(Shannon entropy)	✔ 1 / (Simpson index)
✔ Senstive to	✔ All species	✔ Typical species	✔ Dominant species

Interpolation and Extrapolation

- The number of species counted in a biodiversity study is usually a biased **underestimate**.
- The observed number of species is sensitive to the **sample size or sampling efforts**.
- We develop interpolation and extrapolation curve for **abundance-based** and **incidence-based** data

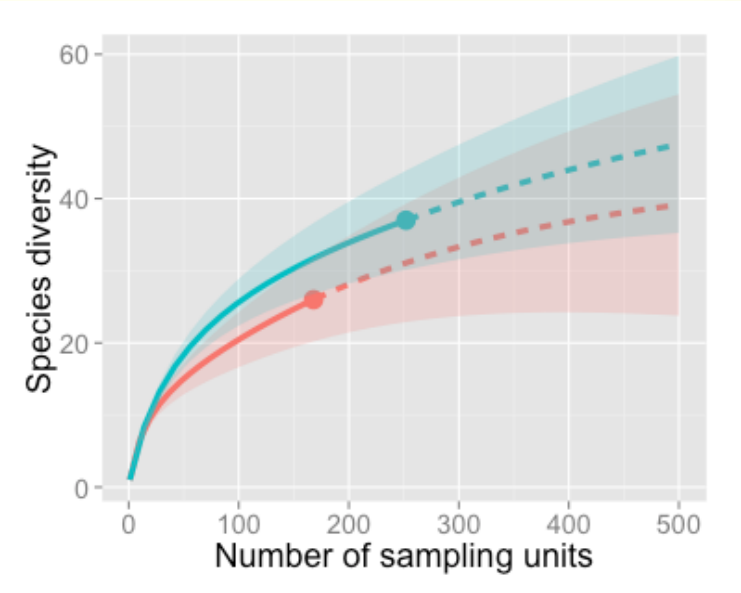
$$ {}^{q}D(m) \approx \left[\sum_{X_i>0} \left(\frac{X_i}{m} \right)^{q} \right]^{1/(1-q)}, $$

$$ \text{where } \sum_{X_i>0} X_i = m. $$

The effective number with sample of size m.

Interpolation and Extrapolation

- The number of species counted in a biodiversity study is usually a biased **underestimate**.
- The observed number of species is sensitive to the **sample size or sampling efforts**.
- We develop interpolation and extrapolation curve for **abundance-based** and **incidence-based** data



The interpolated and extrapolated species diversity curves for abundance-based data.

Coverage-based comparison

- Samples standardized by size will usually have different degrees of **sample completeness** or **sample coverage**.
- Ex: 10 tree species with sample of 100 individuals in a **temperate-zone tree community** vs. 50 species with sample of 100 individuals in a **tropical rain forest community**.

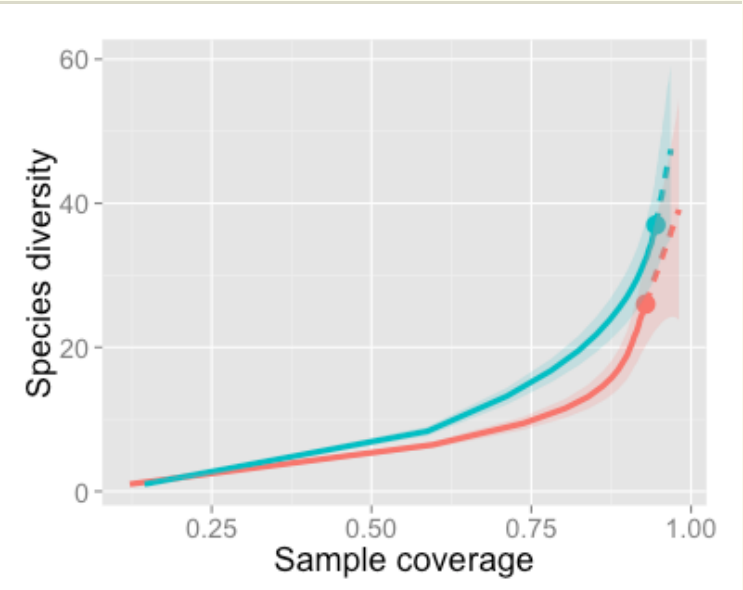
$$C(m) = \sum_{i=1}^{S} p_i I(X_i > 0)$$

$$\approx \sum_{i=1}^{S} p_i [1 - (1 - p_i)^m]$$

Coverage is defined as the total relative abundances of the observed species

Coverage-based comparison

- Samples standardized by size will usually have different degrees of **sample completeness** or **sample coverage**.
- Ex: 10 tree species with sample of 100 individuals in a **temperate-zone tree community** vs. 50 species with sample of 100 individuals in a **tropical rain forest community**.



The interpolated and extrapolated species diversity curves for abundance-based data.

Demo and case study

- Install iNEXT package from github and import package

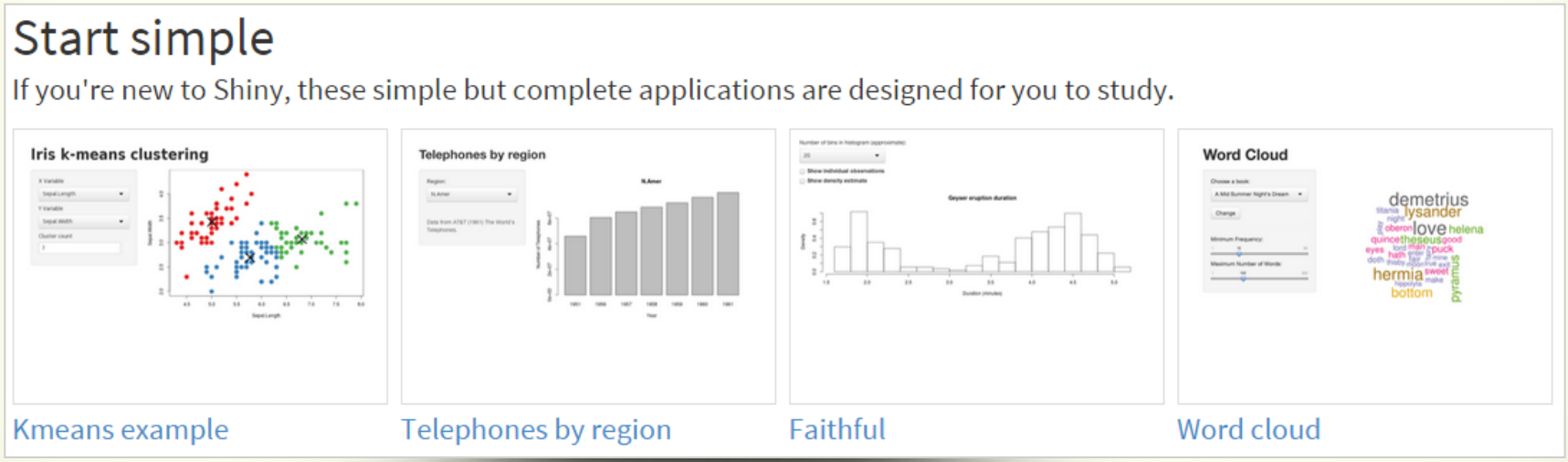
```
install.packages('devtools')
library(devtools)
install_github('iNEXT','JohnsonHsieh')
library(iNEXT)
```

- Run the demo

```
data(spider)
out <- iNEXT(spider, q=0, datatype="abundance",  endpoint=500)
ggiNEXT(out, type=1, color.var="site", facet.var="site")
ggiNEXT(out, type=2, color.var="site", facet.var="site")
ggiNEXT(out, type=3, color.var="site", facet.var="site")
```

Intro to shiny

”Let R user become web application designer wihtout HTML, CSS, or JavaScript”



http://chao.stat.nthu.edu.tw/blog/software-download/inext-r-package/

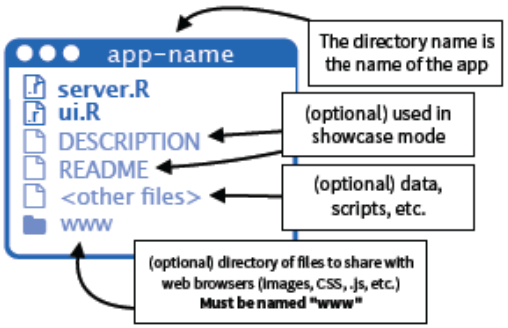
Shiny Cheat Sheet

learn more at shiny.rstudio.com

Shiny 0.10.0 Updated: 6/14

RStudio

1. Structure Each app is a directory that contains a server.R file and usually a ui.R file (plus optional extra files)



2. server.R A set of instructions that build the R components of your app. To write server.R:

- Ⓐ Provide server.R with the minimum necessary code, **shinyServer(function(input, output) {})**
- Ⓑ Define the R components for your app between the braces that follow **function(input, output)**
- Ⓒ Save each R component in your UI as **output$<component name>**
- Ⓓ Create each output component with a render* function.
- Ⓔ Give each render* function the R code the server needs to build the component. The server will note any reactive values that appear in the code and will rebuild the component whenever these values change.
- Ⓕ Refer to widget values with **input$<widget name>**

server.R

```
# load libraries, scripts, data

shinyServer(function(input, output) {

# make user specific variables

  output$text <- renderText({
    input$title
  })

  output$plot <- renderPlot({
    x <- mtcars[ , input$x]
    y <- mtcars[ , input$y]
    plot(x, y, pch = 16)
  })

})
```

render* functions

function	expects	creates
renderDataTable	any table-like object	DataTables.js table
renderImage	list of image attributes	HTML image
renderPlot	plot	plot
renderPrint	any printed output	text
renderTable	any table-like object	plain table
renderText	character string	text
renderUI	Shiny tag object or	UI element (HTML)

Input values are reactive.
They must be surrounded with one of:

render* - creates a shiny UI component
reactive - creates a reactive expression
observe - creates a reactive observer
isolate - creates a non-reactive copy of a reactive object

3. Execution Place code where it will be run the minimum necessary number of times

Run once - code placed *outside of shinyServer* will be run once, when you first launch your app. Use this code to set up the tools that your server will only need one copy of.

Run once per user - code placed *inside shinyServer* will be run once each time a user visits your app (or refreshes his or her browser). Use this code to set up the tools that your server will need a unique copy of for each user.

Run often - code placed within a render*, reactive, or observe function will be run many times. Place here only the code that the server needs to rebuild a UI component after a widget changes.

4. Reactivity When an input changes, the server will rebuild each output that depends on it (even if the dependence is indirect). You can control this behavior by shaping the chain of dependence.

render* - An output will automatically update whenever an input in its render* function changes.


```
output$z <- renderText({
  input$a
})
```

Reactive expression - use reactive to create objects that will be used in multiple outputs.


```
x <- reactive({
  input$a
})

output$y <- renderText({
  x()
})
output$z <- renderText({
  x()
})
```

isolate - use use isolate to use an input without depending on it. Shiny will not rebuild the output when the isolated input changes.


```
output$z <- renderText({
  paste(
    isolate(input$a),
    input$b
  )
)
```

observe - use observe to create code that runs when an input changes, but does not create an output object.


```
observe({
  input$a
  # code to run
})
```

ui.R

```
A shinyUI(fluidPage(

    titlePanel("mtcars data"),
  B sidebarLayout(
      sidebarPanel(

    C   textInput("title", "Plot title:",
          value = "x v y"),

        selectInput("x", "Choose an x var:",
          choices = names(mtcars),
          selected = "disp"),

        selectInput("y", "Choose a y var:",
          choices = names(mtcars),
          selected = "mpg")
      ),

      mainPanel(
        h3(textOutput("text")),
        plotOutput("plot")
      )
    )
  ))
```

C In each panel or column, place…

R components - These are the output objects that you defined in **server.R**. To place a component:

1. Select the ***Output** function that builds the type of object you want to place in the UI.
2. Pass the ***Output** function a character string that corresponds to the name you assigned the object in **server.R**, e.g.

output$plot <- renderPlot({ ... }) ⟷ plotOut put("plot")

*Output functions

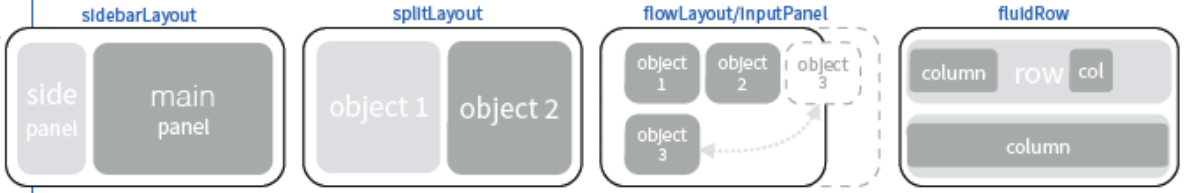
dataTableOutput	tableOutput
htmlOutput	textOutput
imageOutput	uiOutput
plotOutput	verbatimTextOutput

5. ui.R A description of your app's User Interface (UI), the web page that displays your app.

To write ui.R:

A Include the minimum necessary code for ui.R, **shinyUI(fluidPage())**
* note: use **navbarPage** instead of **fluidPage** if you'd like your app to have multiple pages connected by a navbar

B Build a layout for your UI. **sidebarLayout** provides a default layout when used with **sidebarPanel** and **mainPanel**. **splitLayout**, **flowLayout**, and **inputLayout** divide the page into equally spaced regions. **fluidRow** and **column** work together to create a grid-based layout, which you can use to layout a page or a panel.




```
shinyUI(fluidPage(
  sidebarLayout(
    sidebarPanel(…),
    mainPanel(…)
  )
))
```

```
shinyUI(fluidPage(
  splitLayout(
    numericInput(…),
    selectInput(…)
  )
))
```

```
shinyUI(fluidPage(
  flowLayout(
    numericInput(…),
    selectInput(…),
    sliderInput(…)
  )
))
```

```
shinyUI(fluidPage(
  fluidRow(
    column(width = 4, …),
    column(width = 2,
      offset = 3, …),
  ),
  fluidRow(
    column(width = 12, …)
  )
))
```

Widgets - The first argument of each widget function is the **<name>** for the widget. You can access a widget's current value in **server.R** with **input$<name>**

widget	function	common arguments
Action button	actionButton	inputId, label
checkbox	checkboxInput	inputId, label, value
checkbox group	checkboxGroupInput	inputId, label, choices, selected
date selector	dateInput	inputId, label, value, min, max, format
date range selector	dateRangeInput	inputId, label, start, end, min, max, format
file uploader	fileInput	inputId, label, multiple
Number field	numericInput	inputId, label, value, min, max, step
Radio buttons	radioButtons	inputId, label, choices, selected
select box	selectInput	inputId, label, choices, selected, multiple
slider	sliderInput	inputId, label, min, max, value, step
submit button	submitButton	text
text field	textInput	inputId, label, value

HTML elements - Add html elements with shiny functions that parallel common HTML tags.

a	tags$col	tags$form	tags$input	tags$output	tags$sub
tags$abbr	tags$colgroup	h1	tags$ins	p	tags$summary
tags$address	tags$command	h2	tags$kbd	tags$param	tags$sup
tags$area	tags$data	h3	tags$keygen	pre	tags$table
tags$article	tags$datalist	h4	tags$label	tags$progress	tags$tbody
tags$aside	tags$dd	h5	tags$legend	tags$q	tags$td
tags$audio	tags$del	h6	tags$li	tags$ruby	tags$textarea
tags$b	tags$details	tags$head	tags$link	tags$rp	tags$tfoot
tags$base	tags$dfn	tags$header	tags$mark	tags$rt	tags$th
tags$bdi	div	tags$hgroup	tags$map	tags$s	tags$thead
tags$bdo	tags$dl	hr	tags$menu	tags$samp	tags$time
tags$blockquote	tags$dt	HTML	tags$meta	tags$script	tags$title
tags$body	em	tags$i	tags$meter	tags$section	tags$tr
br	tags$embed	tags$iframe	tags$nav	tags$select	tags$track
tags$button	tags$eventsource	img	tags$noscript	tags$small	tags$u
tags$canvas	tags$fieldset	includeCSS	tags$object	tags$source	tags$ul
tags$caption	tags$figcaption	includeMarkdo	tags$ol	span	tags$var
tags$cite	tags$figure	wn	tags$optgroup	strong	tags$video
code	tags$footer	includeScript	tags$option	tags$style	tags$wbr

6. Run your app

runApp - run from local files
runGitHub - run from files hosted on www.GitHub.com
runGist - run from files saved as a gist (gist.github.com)
runURL - run from files saved at any URL

7. Share your app Launch your app as a live web page that users can visit online.

ShinyApps.io

Host your apps on RStudio's server. Free and paid options
www.shinyapps.io

Shiny Server

Build your own linux server to host apps. Free and open source.
shiny.rstudio.com/deploy

Shiny Server Pro

Build a commercial server with authentication, resource management, and more.
shiny.rstudio.com/deploy

Key references

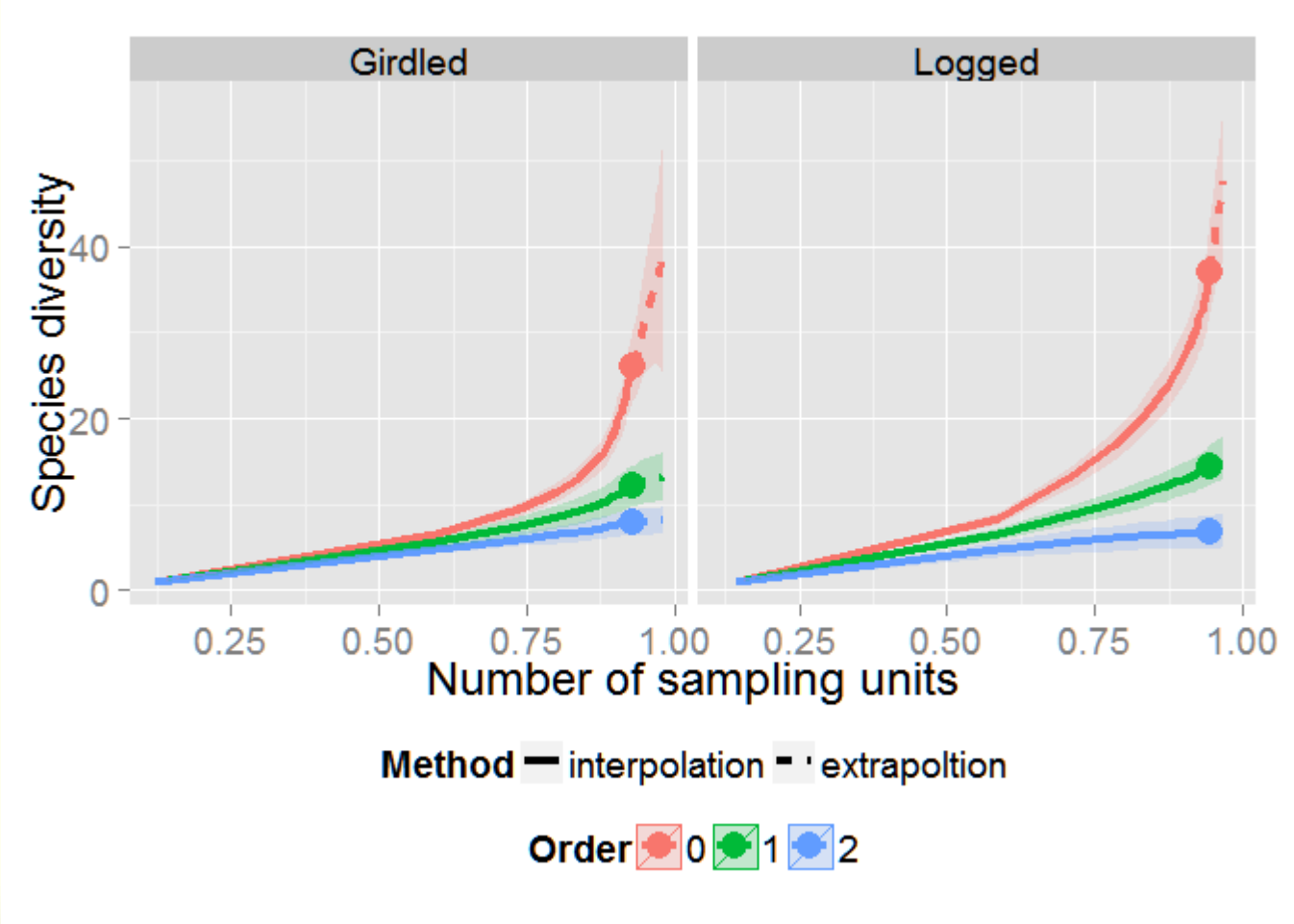
1. Chao, A., N. J. Gotelli, **T. C. Hsieh**, E. L. Sander, K. H. Ma, R. K. Colwell, and A. M. Ellison 2014. Rarefaction and extrapolation with Hill numbers: a unified framework for sampling and estimation in biodiversity studies, Ecological Monographs 84:45-67.

2. Chao, A., and L. Jost. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93:2533-2547.

3. Colwell, R. K., A. Chao, N. J. Gotelli, S. Y. Lin, C. X. Mao, R. L. Chazdon, and J. T. Longino. 2012. Models and estimators linking individual-based and sample-based rarefaction, extrapolation and comparison of assemblages. Journal of Plant Ecology 5:3-21.

4.Hsieh, T. C., K. H. Ma, and A. Chao. 2013. iNEXT online: interpolation and extrapolation (Version 1.3.0) [Software]. Available from http://chao.stat.nthu.edu.tw/blog/software-download/.

5.Hsieh T. C., K. H. Ma, and A. Chao. 2014. iNEXT: An R package for interpolation and extrapolation in measuring species diversity. Unpublished manuscript.



Thanks For Listening