Homework 8 Time & Space:

Updated April, 2016

Table of Contents

Description

Time and space remain at the crux of environmental data analysis. In his 1989 address to the *Ecological Society of America* and upon receiving the society's highest award, Si Levin (1992) stated:

> To address such phenomena, we must find ways to quantify patterns of variability in space and time, to understand how patterns change with scale (e.g., Steele 1978, 1989, Dagan 1986), and to understand the causes and consequences of pattern (Levin 1989, Wiens 1989). This is a daunting task that must involve remote sensing, spatial statistics, and other methods to quantify pattern at broad scales; theoretical work to suggest mechanisms and explore relationships; and experimental work, carried out both at fine scales and through whole-system manipulations, to test hypotheses. Together, these can provide insights as to how information is transferred across scales, and hence how to simplify and aggregate models.

This homework assignment is intended to get you comfortable working with both temporal and spatial dimensions. While this effort will only be scratching the surface, think about aggregation along the two primary axes of the environment: time and space.

Homework Exercise (3 Points total)

These exercises rely on the lab data found in the resource section of CATCOURSES, .

Your document should have the following sections, and provide written explanations formatted in RMarkdown that explains your code, output and graphics in the following format:

NAME
CLASS
DATE

Homework Assignment 8

Objective Statement: [What are you trying to accomplish?]

Methods: [In general terms, what analyses are you doing?]
- **Data**: [What are the data and where did they come from?]
- **Code**: [In specific terms, what is the code that was used to conduct the analysis?]

Results: [What do the results show? Numerical evidence and graphic evidence are required.]

Discussion: [What do the results mean?]

Limitations: [What are the limitations, caveats, and assumptions of the analysis?]

OBJECTIVE

In November 2014, voters were asked to pass a $7Bn bond to better secure California's water future such that times of scarcity (e.g., drought) could be better balanced by times of abundance (e.g., floods). As a new water resources manager for the California Department of Water Resources, you have been asked to explore the temporal and spatial dynamics of discharge in the Cosumnes River, the last major undammed river of the Sierra Nevada. One idea that has been floated by your boss is to examine the possibility of localized recharge of the groundwater aquifer by purposeful flooding of the floodplain. This approach could have the added benefit of promoting fish growth (as suggested by Jeffres et al. 2008) via primary productivity dynamics (Ahearn et al. 2006). One concern is that with the recent drought, it might not flood often enough to be worth the expense of constructing the set back levees. Thus the first step in your analysis is to provide exploratory data analysis of the time series data of discharge at the USGS gaging station located at Michigan Bar (MHB), with a final analysis of determining what percentage of years it floods more than 100 days (a flood is defined as discharges >= 800 cubic feet per second). If at least ⅓ of all years flood for this period of time, localized aquifers will benefit. The second concern is that decreases in dissolved oxygen due to respiration could cause dead zones on the floodplain that would be detrimental to fish growth (a deadzone <= 50% saturation). Your initial report may help DWR determine in setback levees are a good use of Prop 1. funds.

Step 1 - Spatial Interpolation (1 point)

Analyze the dead zone using the "Triangle Floodplain" just north of the "Accidental Forest" in the Cosumnes River Preserve (`allobs.shp`, unzipped from ahearn_allobs.zip) dataset.

Install and load the `maptools[1]` library and use it to load the shape file `allobs.shp` using the `readShapePoints` function (Don't forget to set your working directory).

```
pt.shp = readShapePoints("allobs.shp")
```

Run an EDA on the dataframe to become familiar with the dataset. Notice the date column (useful for time series!) and the location columns (2 projections here - lat and long, x and y. What projection is the X and Y column using?) This is a smaller time series (as can viewed by a summary of datetime) - you can get a sense of the individual dates using one of many commands:

```
levels(as.factor(pt.shp$DateTime_))    # view individual dates

unique(pt.shp$DateTime_)               # TIMTOWTDI
```

Use the date column to subset the data to the 18th of February, 2005. Create another subset for the next consecutive date.

Install and load the `akima` package and interpolate the subsetted points to a raster image showing temperature using the `interp` function from the `akima` package (see `?interp`. Hint: the x and y are easy, but what should you set as the z parameter? Set the duplicate parameter to "mean". You should only have to set the x, y, z and duplicate parameters).

You can plot the raster using the `image` function and add contour lines using the `contour` function (for contours, make sure to include the `add=T` parameter so it adds to the current raster plot instead of making a new plot. Hint: add this color parameter to the `image` when plotting the temperature raster to make make contours more visible: `col=cm.colors(24)`).

Step 2 - Perspective Plot (1 point)

To create a perspective plot (see `?persp`) of the temperature raster, use:

```
persp(ptzTemp.0218, theta=30, phi=45, col="lightblue", zlab="Temp C",
xlab="UTM X (m)", ylab="UTM Y (m)")
```

Where `ptzTemp.0218` is your temperature raster. How does the perspective plot differ from the image plot? Adjust the theta and phi values to get the best perspective.

Use the layout matrix (review homework 2) to create two (2) separate plots of 2 rows x 3 cols for each of the two dates (as above). The two rows will have separate plot types (`image(xyz)` combined with `contour(xyz, add = TRUE)`, and the second the wireframe `persp(xyz)`)

and the three columns will be one each of Temperature (use `cm.colors` palette, Chlorophyll (`topo.colors` palette), and DO (`heat.colors` palette). See Ahearn et al. 2006 for notes on units for response variables.

Temperature	Chlorophyll	Dissolved Oxygen
`image() & contour()`	`image() & contour()`	`image() & contour()`
`persp()`	`persp()`	`persp()`

Step 3 – Shiny Map (2 points)

You decide to publish your analysis on the department blog along with a dynamic figure that allows users to choose which among
Shiny is an open source R package developed by RStudio that allows for the easy creation of interactive web applications using R. Shiny can enable you to add interactive features to the analysis you do in R and the graphs that you create. Shiny has been in active development since 2012. Shiny produces fairly lightweight web applications—simple apps that just display text and images can be created without any javascript dependencies, whereas very interactive apps (with sliders, knobs, or 3D visualizations) become fairly script heavy. Shiny uses Bootstrap for page creation which allows for very "theme-able" and extensible web applications without having to be a web design expert (at the cost of sacrificing some backwards compatibility) [3]. To understand the types of applications that can be created with Shiny, please browse the Shiny Gallery:

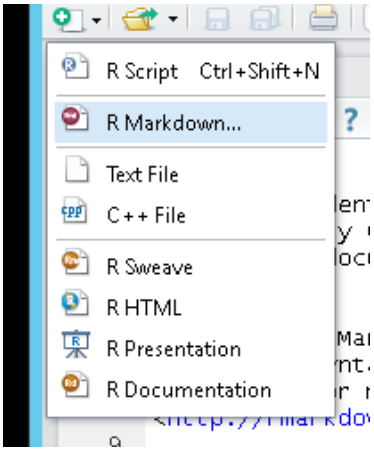
http://shiny.rstudio.com/gallery/

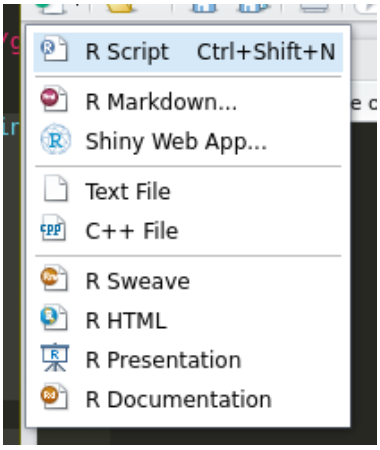
A Shiny application is comprised of two R files – a server and a user interface (UI) file. The `ui.R` file is where you create your buttons, checkboxes, images, other interactive widgets and graphic outputs. You essentially assign each interactive element to a variable, the user sets the variable value in the browser, and shiny passes the variable value to the shiny server. The `server.R` file is where the computation happens. Here, you create functions that accept variables from the `ui.R` file, load your necessary R libraries, perform computations, and return outputs to the Shiny UI. It is also possible to combine both the UI and server portions of your project into one `.R` script.

Let's begin by reviewing lessons 1 and 3 and completing Lesson #5 of the shiny tutorial found at the following link:

http://shiny.rstudio.com/tutorial/

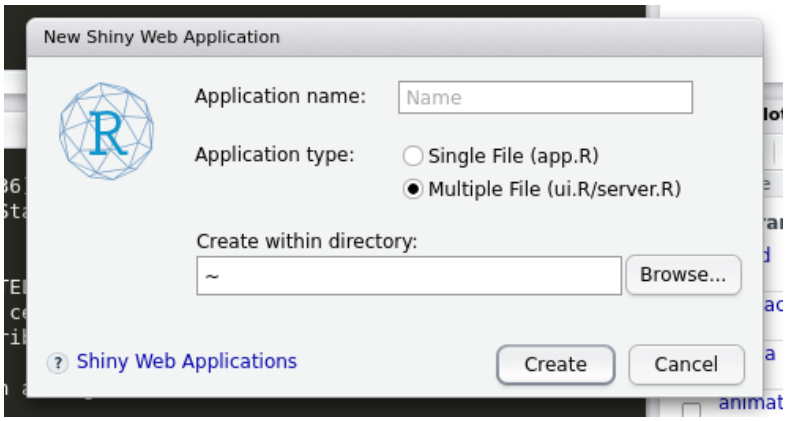
You can have RStudio create a blank shiny app (with example boilerplate) via the same menu used to create new Rmarkdown documents. The exact look of this menu will depend on your version of RStudio:





If your `New File` doesn't look like the image to the far right, update your version of RStudio. Select `Shiny Web App...`

In this exercise we will use the `Multiple File` method of constructing Shiny applications. Feel free to use a single file if you wish. Shiny apps run within their own folder, so it makes sense to create a dedicated folder within your working directory for your shiny application.



Let's first work on your server.R script, the file where Shiny will execute computation routines. The structure of a minimal server.R that displays and updates a map may look like the following:

```
# server.R (Note: This will run but won't produce anything)

# Load libraries and datasets here. Things here will only run once.
library(shiny)

shinyServer(
  function(input, output) {

  # Code placed here will run whenever a user opens
  # your shiny app or refreshes the page

  output$map <- renderPlot({

    # Code placed here will run whenever a user changes a
    # widget associated with output$map

    })
  }
)
```

A corresponding ui.R script may be structured as follows:

```
# ui.R (Note: This will run, but won't produce anything)

library(shiny)

# Begin defining the Shiny UI
shinyUI(fluidPage(
  # Add a title panel
  titlePanel("Triangle Floodplain"),

  # Add a side-panel and add some elements to it
  sidebarLayout(
    sidebarPanel(
      helpText("Visualize different dates and you can add more text
here."),

      # As written, the selection of this input will be passed
      # to server.R as the variable 'var' with a default
      # value of 'Feb 18, 2005'
      selectInput("var",
                  label = "Choose a date to display",
                  choices = c("Feb 18, 2005"),
                  selected = "Feb 18, 2005"),

      selectInput("var.var",
                  label = "Choose a variable to display",
                  choices = c("Temperature","Chlorophyl"),
                  selected = "Temperature")
    ),
    mainPanel(plotOutput("map"))
  )
))
```

You can try running the script above as written by pressing `Run App` button in RStudio (it's near where your knit button was). As written, `ui.R` will create a `selectInput` object dropdown menu in the sidebar panel and a `plotOutput` object in the main panel. The `selectInput` object is called *`var.date`* and the `plotOutput` object is called *`map`*.

If you look at the `server.R` script, the shinyServer function contains an input variable and an output variable. *`input$var.date`* will contain the values selected by the user (as defined in `ui.R`) and we will assign our computed plot to *`output$map`*, for the ui.R script to display.

Often, we want to show the user a short label in the dropdown menu, which would in turn correspond to an identifier in our data frame of interest. We can use the `switch()` command to assign the string shown in the ui.R input to a subsetting operation on our data frame.

Let's take a look at a more complete server.R example, using the map generated in Step 1:

```
# server.R (Note: This will run but won't produce anything)

# Load libraries and datasets here. Things here will only run once.
library(shiny)

shinyServer(
  function(input, output) {

  output$map <- renderPlot({

    pt <- switch(input$var,
         "Feb 18, 2005" = pt.shp[pt.shp$DateTime_ == "2005-02-18",])

    pt.filled = interp(pt$X,pt$Y,pt$Temp,duplicate='mean')

    image(pt.filled,col=cm.colors(24), xlab="UTM X (m)",
          ylab="UTM Y (m)", main="Title")

    })
  }
}
```

Note, the above command won't work because we still need to read in the pt.shp data set. It can be accomplished by adding the following lines to your server.R file (the exact path may vary). Can you think of where these lines should go?

```
library(maptools)
library(akima)
pt.shp = readShapePoints("../Input/ahearn_allobs/allobs.shp")
```

Now, finish up this shiny application by adding options for all of the available dates and by fixing the second drop-down menu so that users can choose between Temperature, Chlorophyll, and DO. This will require adding the missing variables to the second `selectInput` object and correcting `server.R` file to switch between `Temp`, `Chlorophyl`, and `DO__`. Also, try adding the other variables (TDS, Turbidity, etc) to your menu if they work!

(Advanced users are encouraged to use `tabPanel` objects to create a new tab for contour and perspective plots [1pt extra credit]).

Resources

Data

- http://waterdata.usgs.gov/nwis/dv?cb_00060=on&format=rdb&site_no=11335000&referred_module=sw&period=&begin_date=1907-10-01&end_date=2014-09-30
- ahearn_allobs.zip (must unzip this)

References

Citations

[1] maptools - http://cran.r-project.org/web/packages/maptools/index.html
[2] alkima - http://cran.r-project.org/web/packages/akima/index.html
[3] Bootstrap3 browsers - http://v4-alpha.getbootstrap.com/getting-started/browsers-devices/
[4] Shiny tutorials - http://shiny.rstudio.com/tutorial/
[5] Shiny tab panels - http://shiny.rstudio.com/reference/shiny/latest/tabPanel.html

Keywords

`read.table, droplevels, as.numeric, ifelse, acf, stl, readShapePoints, persp, shiny`