Homework 1

Advanced Hydroinformatics

Missing Data Quantification and an introduction to Parallel Computing

Due: 2/12/2021

In this assignment we will develop functions for visualizing missing data in time series data. We will begin with an example dataset and then scale up our function to 50 years worth of data. Last, we will accelerate our computations with parallel programming packages. This assignment can be performed on any computer, although using the Lewis computing cluster may provide greater accelerations in the final, parallel computing, section.

Section 1: An Example

Hydrological datasets, sometimes collected over decades, may contain missing recordings. Depending on the recording issue, this could be an hour, a day, or a year. Let us define a simple time series with data from 4 different weather stations over 5 different days. Some of the readings are missing and their entry contains a `NA`.

```
test=matrix(nrow=5,ncol=4)
test[,1]=1:5
test[2:5,2]=2:5
test[3:5,3]=3:5
test[4:5,4]=4:5
test=data.frame(test)
colnames(test)=c("Station1","Station2","Station3","Station4")
rownames(test)=c("Day1","Day2","Day3","Day4","Day5")
```

Make sure to take a look at your data using `View(test)`.

Question 1.1 What variables are on the x and y axis of this dataset?
Question 1.2 How many missing values are there for each station?
Question 1.3 If we started our analysis on day 2, only including days 2-5, how many missing values are there for each station?
Question 1.4 Day 3?

It will be impossible to find NAs like this in larger datasets, so let us write a function to do so. This function takes a **datafame** (organized like ours) and a **starting row** as input. Remember, rows are equivalent to dates of recording. It outputs the number of NAs in the range of starting_row -> end of recording. As in question 1.3 and 1.4, it gives us an idea of how much missing data there is in this frame depending on where we start counting.

```
addNA = function(data_with_some_nas, starting_row) {
  nas = is.na(data_with_some_nas[starting_row:dim(data_with_some_nas)[1], ])
  return(apply(nas, 2, sum))
}
```

Question 1.5 What will `addNA(test,1)` return? What does this mean?
Question 1.6 What will `addNA(test,4)` return? What does this mean?

To visualize where the NAs are in this dataset, we need to run this function on every row. Try coming up with a loop to do this.

Question 1.7 What are the two types of loops in R? What should we use for this purpose?
Question 1.8 What is the output of `1:dim(test)[1]`? What does it mean?
Question 1.9 What will the dimensions of the output be?

The below code runs the loop and plots the output. Note that packages `ggplot2` and `reshape2` are required and can be installed using `install.packages("reshape2")`. The **melt function** helps us convert the data to a plottable shape.

```
NAs_in_test = test * 0
for (i in 1:dim(test)[1]) {
  NAs_in_test[i, ] = addNA(test, i)
}

library(ggplot2)
library(reshape2)
NAs_in_test$date = 1:5
reshaped = melt(NAs_in_test, id.vars = "date")
ggplot(reshaped, aes(x = date, y = value, col = variable)) + geom_line() +
  facet_wrap( ~ variable)
```

Question 1.10 What do the different plots represent?
Question 1.11 What variables are on the x and y axis?

Section 2: Real data

Set your working directory accordingly using `setwd()`. Be sure that the file `allprcp.Rdata` is in this dir.

```
load("allprcp.Rdata")
```

Question 2.1 What are the dimensions of this data? *Hint: `dim()`.*
Question 2.2 What are the column names of this data? *Hint: `colnames()`.*

Although we can use the `View()` function still, it is good practice to try and understand the data without it. Our later datasets may have thousands of rows and columns.

The data contains a few columns with date information, but our `addNA()` function is not written to deal with this type of data.

Question 2.3 What is the column range of the data? (Names begin with US. . . .) Use this information to make a new data frame called rain.

```
data_start_column = #Enter your answer here
data_stop_column = #Enter your answer here

rain = allprcp[, data_start_column:data_stop_column]
```

Try and adapt the earlier loop and plot to the `rain` dataset. **Note that the loop may take a few minutes to run.** After the loop has ran, we will need to add the date information back to the data frame for plotting. Your algorithm should go as follows:

1. Create an output container. *Hint: remember* `NAs_in_test = test * 0`
2. Loop through the data's x range using addNA.
3. Attach a date column of `allprcp` to the data output.
4. Reshape the data and plot.

```
NAs_in_rain = rain * 0
for (i in 1:dim(rain)[1]) {
  NAs_in_rain[i, ] = addNA(rain, i)
}

NAs_in_rain$date = allprcp$date
rain_reshaped = melt(NAs_in_rain, id.vars = "date")
ggplot(rain_reshaped, aes(x = date, y = value, col = variable)) + geom_line() +
  facet_wrap( ~ variable)
```

Question 2.4 Describe the missing data trends in a few of the stations.
Question 2.5 Describe how this algorithm works.

Section 3: Parallel

With 7 stations and 50 years worth of data, the loop took a while to run! We can speed up this process by using **parallel programming**. Most computers, likely including your laptop, have multiple processors and can perform multiple processes at once–think about how you can move your mouse, run R code, and surf the internet all at the same time. Despite this, most code we write is **serial**, only one line of code is executed at a time. Let us time our **serial** code.

```
NAs_in_rain = rain * 0
system.time({
  for (i in 1:dim(rain)[1]) {
    NAs_in_rain[i, ] = addNA(rain, i)
  }
})
```

Question 3.1 What is the elapsed time of this code block?

Serial code is necessary if the second line of code depends on the first line of code. In the case of our loop, the output of one loop iteration does not depend on the output of the previous. In English: `addNA(rain,2)` can be run before, after, or at the same time as `addNA(rain,3)`. We will now use an R package that allows for-loops to run multiple iterations at the same time.

```
install.packages("doParallel")
library(doParallel)

registerDoParallel()
getDoParWorkers()
```

Question 3.2 What is the output of `getDoParWorkers()`? This is most likely the number of processors your computer has.

Take a look at the parallelized code below. It has many of the same parts of a for loop:
1. A `for()` function, just called `foreach()`
2. An iterator `i = 1:dim(allprcp)[1]`
3. A function that occurs every iteration `addNA(rain, i)`

However, there are a few differences:
1. Instead of writing the output of `addNA()` every iteration, `foreach` arranges the output and gives us the output at the end. See `NAs_in_rain_par = foreach` and the `.combine` option.
2. We need to explicitly tell the `foreach` function what data and functions it will be using. This is listed in the `.export` option.
3. Last, we need to specify the `%dopar%` option.

```
system.time({
  NAs_in_rain_par = foreach(i = 1:dim(allprcp)[1],
                            .export = c("addNA", "rain"),
                            .combine = "rbind") %dopar% {
    addNA(rain, i)
  }
})
```

Question 3.3 What is the elapsed time of this code? Is it faster than our earlier serial code? Why is this?

Plot the code to make sure the function worked.

```
NAs_in_rain_par = data.frame(NAs_in_rain_par)
NAs_in_rain_par$date = allprcp$date

rain_reshaped_par = melt(NAs_in_rain_par, id.vars = "date")
ggplot(rain_reshaped_par, aes(x = date, y = value, col = variable)) + geom_line() +
  facet_wrap( ~ variable)
```