R practice

Eric Gilleland

20th May 2015

1 Preliminaries

1. The data set `RedRiverPortRoyalTN.dat` can be obtained from `http://www.ral.ucar.edu/staff/ericg`. Read these data into R using the `read.table` function with `header = TRUE`, assigning them to an R object named `sf`. Note that `#` is how comments are made in R, so there is no need to skip the first 29 lines when reading this file (R ignores them). However, if these lines did not begin with `#`, then the `skip` argument could be used to force R to ignore them.

2. What is the class of `sf`?

3. Use `names` or `colnames` to see the names of the various columns.

4. What is the class of the `flow` column?

5. Use `summary` to obtain summary statistics for each column of the data frame.

6. Make a line plot of the flow argument against the observation number. Can you make it look just like the one in Figure 1?

7. Now plot flow against year using `type = "h"`.

8. Make boxplots of river flow stratified by the `code` column.

9. Repeat the exercise above, but set `ylim = c(0, 20000)`. Note that the "e" in the code indicates the values have been estimated. What would you say about the distribution of the estimated data versus the measured data?

10. Without using a loop, create a 100×1000 matrix where each column is a sample of size 100 drawn from a standard normal distribution. **Hint**: you need only make one large draw of size 100×1000.

11. Replace the above samples with a single sample of size 1000, where each draw is the maximum of a sample from above.

12. What is the mean and standard deviation from this sample? **Hint**: use `sd` to get the standard deviation.

13. Make a histogram of the resulting sample from 11 above using `freq = FALSE` and make the bars gray in color (i.e., use `col = "gray"`).

14. Superimpose a dashed red line showing the fitted normal distribution (from 12 above) to this sample. Does the normal distribution appear to be a good fit to the sample? **Hint**: the functions `seq`, `dnorm` and `lines` (with `lty = 2`) will be useful here.

15. Now make a normal qq-plot of the sample against the normal distribution. Is the line straight or curved? **Hint**: use the `qqnorm` function.

16. Now let's check if the sample is temporally dependent or independent. Set the plotting region to have one column of two plots. Then use `acf` and `pacf` to plot the autocovariance and partial autocovariance functions. Does the sample appear to be temporally independent? **Hint**: see section 3 of http://www.ral.ucar.edu/staff/ericg/Gilleland2010.pdf

17. Now check the Red River flow data. **Hint**: use `na.action = na.pass`. Do these data appear to be temporally independent?

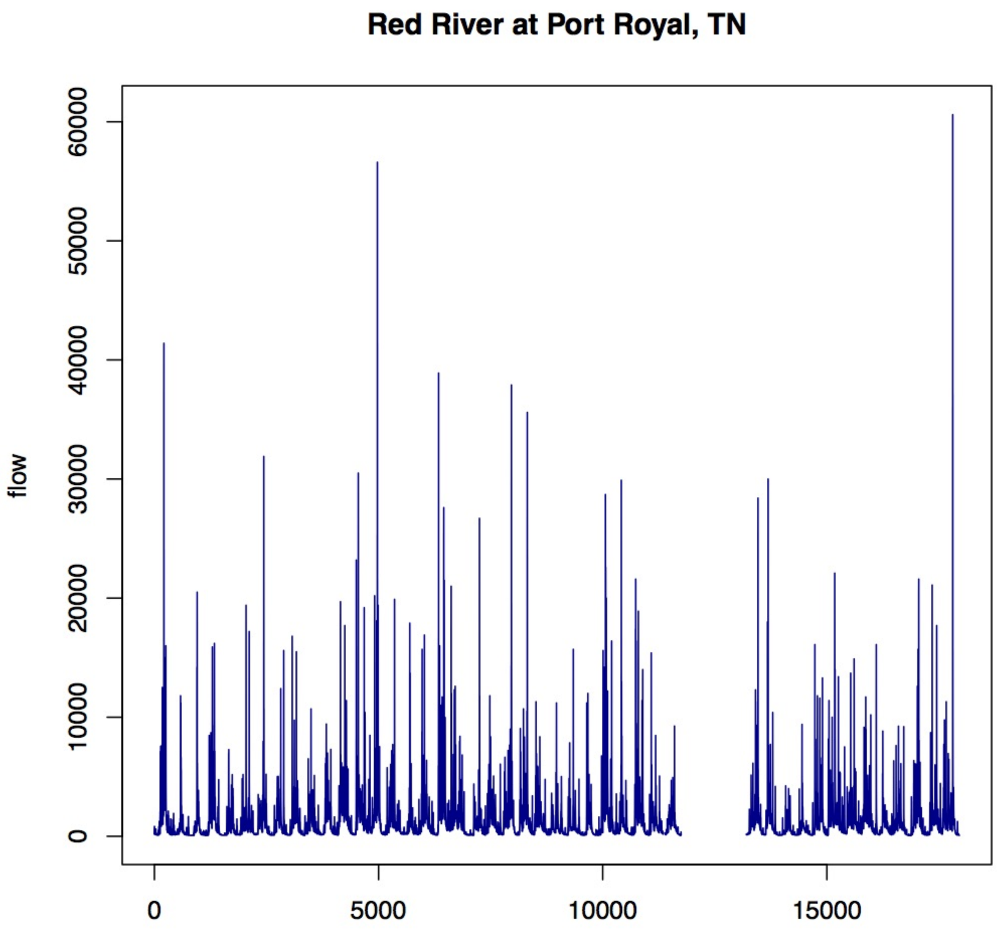
2 Bootstrapping in R

1. Write a function to take a data frame with a (integer or numeric) component named `flow` (e.g., the Red River data in `sf` from 1.1 above) as an argument called `d` and an indexing argument called `i`. Inside the function, first assign to an object called `y`, the flow component as indexed by `i` (i.e., `y <- d$flow[ i ]`). Finally, return a numeric vector containing the mean and variance of `y` (**Hint**: use the `var` function). Be sure to do something about possible missing values! Name this function `booter` (or whatever name you like, but I will refer to it as booter for brevity).

2. Load the **boot** package, and check how to cite the package in a paper.

3. A useful function in R is the `args` function, which simply shows a function's arguments. The degree of usefulness depends on the specific function. See what the arguments for `boot` are.

Figure 1: Result for Preliminaries exercise number 6.



4. Apply the `boot` function to the `sf` data frame using `booter` created in 2.1 above, use `R = 500` replicate samples. Assign the output value to an object called `booted` (or, again, whatever you like).

5. Now check the arguments of `boot.ci`.

6. Apply `boot.ci` to the `booted` object. Use `type = "perc"`. What is the 95% iid bootstrap confidence interval for the mean?

7. Repeat 2.6, but this time use `index = 2`. What is the 95% iid bootstrap confidence interval for the variance?

8. Repeat 2.6 and 2.7 using the `type = "basic"` bootstrap.

9. We had previously decided that the flow data are not independent, so the intervals found in 2.6 through 2.8 will be too narrow (we are over confident because we effectively have fewer data points than we think we have). To account for this issue, we can apply a block bootstrap procedure. To this end, write a function similar as before, but this time have it only take one argument called `data` and assign `data$flow` to `y`. I will call this function `tsbooter`.

10. Check the arguments for `tsboot`.

11. Use `tsboot` to evaluate `sf` using `tsbooter`. Use `R = 500` replicate samples again, as well as `l = 134`[1] and `sim = "fixed"`. I will refer to the assigned output value as `tsbooted`.

12. Repeat 2.6 through 2.8 using the `tsbooted` object. Do the intervals change at all?

3 Avoiding loops

For each problem below, do not invoke an explicit loop (i.e., do not use a for or while loop).

1. Create a $500 \times 600 \times 10$ array filled with realizations from a standard normal distribution, and assign it to an object named `Z`.

[1]The `l` argument is the block length, which needs to be longer than the correlation length, but much shorter than the length of data. A good rule of thumb is to use the square root of the sample size, which in this case is 134.

2. Suppose `Z` represents a model's error at each grid point for ten time points on a spatial field with dimensions 500×600. Find the temporal average error at each grid point and assign it to an object called `Zbar`. Use `image.plot` from the **fields** package to plot the result.

3. Find the maximum error over time at each grid point, assign the result to an object called `Zmax`, and plot the result as in 3.2 above.

4. Create a $3,000,000 \times 3$ matrix whose rows index each grid point in `Z`. That is, the first row will have values `c(1, 1, 1)`, the second row will have values `c(1, 1, 2)`, the eleventh row will have `c(1, 2, 1)`, etc. (please use this ordering), and assign the result to an R object called `ind`. **Hint**: `cbind` and `rep` might be useful here.

5. Create a logical array of the same dimension as `Z` whose values are `TRUE` wherever `Z` is positive and `FALSE` otherwise. Assign the result to `posid`. How many positive values did you get? What is the frequency of occurrence of positive values for your sample? Given the center of mass and symmetry of the standard normal distribution, does your answer make sense?

6. Write a function whose first argument is called `ijk`, and whose second argument is `x`. Argument `ijk` will be a length-three integer vector that indexes a particular grid point of the three-dimensional array, `x`. Have the function return a numeric vector whose components are: the mean and variance of the grid points within a cubic neighborhood of length three[2] of the point indexed by `ijk`, as well as the rank and order of the point `ijk` among these neighbors. **Hint**: see `?order` and `?rank`.

7. Using the function from 3.6 above, `posid` and `ind`, evaluate the function for each positive-valued grid point in `Z`. Assign the result to `res`. Note: this exercise may take a few moments to compute, but should not be ridiculously long.

8. What is the class, mode and dimension of `res`? What are the row names?

9. Transpose `res` and apply `summary` to it. does `res` contain any missing values? Why?

[2]In general, the length-three neighborhood of a point indexed by i, j and k is given by all the points inside `((i - 1):(i + 1), (j - 1):(j + 1), (k - 1):(k + 1))`. For points near boundaries, use a smaller neighborhood of points (i.e., any indexes less than 1 (R indexes beginning with 1 rather than 0) or greater than the respective dimension).

10. create a 100×10 matrix whose columns each represent an independent time series simulated by `arima.sim` (simulate any type of time series model you like), and call it `Zt`.

11. Without plotting them, use `acf` to find the autocovariances for each column (use `type = "covariance"`), and store all the results in one list (i.e., with ten unnamed components, where each component is an `"acf"` object) named `acfres`.

12. In this spectral representation, the variance of a time series is the sum of its (lag or auto-) covariances. Empirically, if all possible lag covariances are summed, the result can be proven to be zero, which is usually a very bad estimate of the variance. Here, however, we used the default `lag.max` argument, so it should be possible to get a reasonable estimate of the variance (that accounts for serial dependence) of each series by summing the lag covariances that we have estimated.

    (a) What is the length of `acfres`?

    (b) What are the names of any one, say the first, component of `acfres`?

    (c) What is the class and dimension of the `acf` component?

    (d) Estimate the variances by summing the values in the `acf` component for each of the ten series using the `lapply` function.

4 Method Functions

1. Write a generic function called `acfer` that has arguments `x`, and '. . .'.

2. Write a method function for `acfer` that works on numeric vectors. The function should return a list object with attributes containing `call` (the function call) and `data.name` giving the name of the object passed to `x` (i.e., use `deparse(substitute( x ))` inside the function call), class `acfed`, and named components `lag`, `acf` and the estimated variance of the series as estimated in 3.12 (d) above. Also return a component that is just a copy of the data passed to `x`, called `x`. The . . . argument should allow the user to pass any arguments to `acf`.

3. Write a similar method function that works on matrices whereby it applies the function for numeric vectors to each column of the matrix.

4. Write a print method for `"acfed"` class objects that prints the function call, name of the data used and the estimated variance. You may (or may not)

want to use the `paste` function. Be sure the print method can handle either numeric or matrix output. In the latter case, print the estimated variance for each column of the matrix.

5. Write a `plot` method function for `"acfed"` class objects. For numeric vectors, make panels of plots showing the ACF and PACF plots, as well as another plot showing lags against covariances. For matrices, produce one single plot that plots the autocovariances for each column using different colors and/or symbols. Give the plot a legend too that should be placed in the `"topright"` (also use argument `bty = "n"`).