Lab 2

In this tutorial, we are going to learn to use scripts and to write functions in R.

1. Make a new folder called "lab2" and save the file "globtemp.dat" from the "lab2" directory. Go to the file menu and select "New script". We are going to read in the data from "globtemp.dat" (The global average temperature from 1900-1997). In the first line of the script, enter

   ```
   global= scan("globtemp.dat")
   ```

2. Let us now enter the commands to save a plot of this variable into a pdf file. Enter into your script the following:

   ```
   pdf("globalplot.pdf")
   plot.ts(global)
   dev.off()
   ```

3. Now, we need to save this script. Select "save" from the file menu. Save this script as "lab2.R". Choose, "Source R code..." and select "lab2.R". You will notice that the pdf file is created in your folder. Note that you can take this pdf file and drag and drop it into a word processor program or double click it to view. An alternative to using the "Source R code..." command to run the script is to type

   ```
   source("lab2.R")
   ```

   on the command line.

4. We can add to the script by saving plots of this data differenced and also ACF plots of the data and the differenced data.

```
globaldiff=diff(global)
pdf("globaldiffplot.pdf")
plot.ts(globaldiff)
dev.off()
pdf("globalacf.pdf")
acf(global)
dev.off()
pdf("globaldiffacf.pdf")
acf(globaldiff)
dev.off()
```

Save and run the script.

Writing function

In this lab, we will learn to write our own functions in R. (If you would like more information about writing functions, see section 10 of *An Introduction to R.*) One important reason for learning about functions is to allow us to run simulations. In this section, we will write a function to simulate AR processes. For homework, you will write your own function to simulate MA processes.

1. We will start by writing a very simple function, so we can learn how things work in R. Functions take arguments, do something to the arguments, and return a result. Start a new script titled "add.R". Here is a short function to enter at the top of your script that will simply add two numbers and return the result.

```
add=function(a,b){
result=a+b
result}
```

Now, run your script. Then, type `add(2,3)`, and notice the result. This function has two arguments, `a` and `b`. The result is returned by simply typing the object that you would like to have returned, in this case, the numerical variable `result`.

2. Another example would be to write a function to simulate a Gaussian random walk. We will need to take arguments for the length of the series and the variance of the white noise process and return a vector representing the random walk. Here is one possible implementation:

```
randomwalk=function(sigsq,T ){
x=rep(0,T)
w=rnorm(T,sd=sqrt(sigsq))
for ( i in 2:T){
x[i]=x[i-1]+w[i]
}
x
}
```

   (Note that I am assuming an initial condition of zero.) Try a few sample paths, and calculate the ACF of those sample paths.

3. Now, we will write a function to simulate an $AR(p)$ process of length T. What information is necessary to do this? We will need the AR parameters $\phi_1, ..., \phi_p$, the variance of the white noise σ^2, and the length of the time series we would like to generate T. These will be the arguments for our function.

   It would be nice if we could write one function that will generate an AR time series regardless of p. We will accomplish this by entering the AR parameters as a vector. Our function will be as follows:

```
arsim=function(phis, sigsq, T){
p=length(phis)  #find the number of lags in our AR
noise=rnorm(T+p, sd=sqrt(sigsq))  #generate the white noise plus a few to get started
x=c(noise[1:p],rep(0,T))  #put the initial noise terms in and set the rest to zero
for (i in (p+1):(T+p)){  #this loop generates the AR series with the recursive formula
x[i]=phis %*% x[i-(1:p)] +noise[i]
}
x=x[(p+1):(T+p)]  #throw away those initial starting points
x #return the time series
}
```

We should walk through this function definition to make sure we understand things. First, we see that there are three arguments. The first argument is `phis` which is a vector containing the AR parameters in order: $\phi_1, ..., \phi_p$. The second is `sigsq`, the variance of the white noise, and the last is how many observations we want, `T`.

The first line of the function finds the length of the `phis` which is p. We could pass an additional argument `p`, but this way is simpler. The second line generates $T + p$ independent normal random variables with variance `sigsq`. In terms of how we have been writing our models, this command generates the w_t. We generate p extra white noise terms to use as our initial values for the first p values of our time series.

We are going to use the variable `x` to store our time series. So, the next command sets up the vector `x` with the first p white noise terms and the rest zeros. The next set of statements is a `for` loop. This loop generates the time series using our recursive definition of an AR series. Note the operator `%*%`. This take the inner product of two vectors. In other words, the first element of vector one times the first element of vector two plus the second element of vector one times the second element of vector two plus and so on. The command after the for loop simply gets rid of those first p white noise terms leaving us with the AR process only. The final `x` returns our time series.

4. Now that we have a function to generate an AR series, let us put it to work. Generate an $AR(1)$ time series of length 200 and with $\phi_1 = 0.5$ and $\sigma^2 = 1$ using the command

```
x1=arsim(c(0.5), 1,200)
```

Take a look at a plot of the data and the ACF of this time series. Is this what you would expect?

Also, generate a similar time series but with $\phi_1 = -0.5$. How do the two time series look different? How about the ACF's?

5. We see with a $|\phi_1| < 1$ the process appears reasonable and stationary, and the last simulations you did confirmed that. Try a ϕ_1 of 1.1, 1.01, 1.001, and 0.99. What do you notice?

3 Homework

1. Write a function called `masim` to generate a $MA(q)$ series of length T.

2. Make sure that your function works. Simulate a model with $\sigma^2 = 1$, $\theta_1 = 0.5$, and $\theta_2 = 2$ with $T = 1,000$. Make an ACF plot. Is this consistent with the model that you generated? Try generating a model with the same parameters but only 200 observations. Make an ACF. Repeat this a few times, what do you notice about the autocorrelations and the dotted blue lines?

3 Use arima function to fit a model for the data generated in question 2.