Statistical Software: SAS vs. Minitab vs. SPSS vs. R / S-Plus

There are a wide variety of applications which can perform statistical analysis. The title above only lists the major ones.

SAS is, to a large extent, an industry standard statistical software package. It is used almost exclusively at Pharmaceutical companies to run data analysis for drug trials that will ultimately go to the FDA for final approval. There is great demand for students with SAS skills, but UCSD does not offer any courses on this software at this time. Many companies, though, offer internships where one can pick up the necessary skills. It helps to be familiar with a statistical language to begin with, though.

Minitab was originally developed to help students learn basic statistics. This software is quite accessible to those unfamiliar with computers and its use is primarily limited to the educational community. It used in Math 11 here at UCSD.

SPSS is a more powerful version of Minitab that includes more professional packages to perform statistical analysis. SPSS is commonly used among the biomedical community and in the social sciences.

The last two applications, R and S-Plus, have nearly identical command-line syntax, but the latter is more GUI-oriented. The two programs started out the same. S-Plus was developed into a commercial product that can handle large amounts of data, while R focuses more on small data samples and is more appropriate for university researchers, particularly those interested in computational statistics.

It should also be noted that packages exist for MATLAB which allow it to perform statistical calculations. It is used in many engineering and basic science classes that have an overlap with statistics.

So what are the differences between R and S-Plus?

As was mentioned above, R is primarily command-line based while S-Plus is more GUI-oriented. For example, in either program you can type **t.test**(*Data*) to get detailed hypothesis test which, by default, tests $H_0 : \mu = 0$, and returns the test statistic, p-value, and other potentially useful quantities. But in S-Plus, you can simply click on a menu item instead of typing (and remembering) the **t.test** command. You may think of the analogy as R representing the command-line Linux OS and S-Plus representing the menu-driven Windows OS.

Both are free to students (you), but only R is open-source and under the GNU General Public License. S-Plus is offering a Windows version of its product free to students, but some registration is required. (Also, this version of S-Plus is limited in the amount of data that it can load.)

Where can I get R and S-Plus?

To download R, go to http://cran.stat.ucla.edu/ then continue by clicking on the OS of your choice (Linux/OS X/Windows). If you choose Windows, click on the subdirectory link "base", then choose "R-2.7.2-win32.exe". To download S-Plus, go to http://elms03.e-academy.com/splus/. As mentioned above, some registration is required.

For the purposes of this demonstration, we shall focus on R (and the Windows platform), but again, the syntaxes of the two languages are 99% identical. As a word of warning, the code provided in our textbook is designed for S-Plus. It *should* work in R just fine. Also, near the end of the course, it seems that the code provided for ARCH and GARCH models is intended for use in R, not in (the student version of) S-Plus.

Exercise: Download R or the student version of S-Plus to your computer.

Getting Started in R

Once you have started the R program, try doing some simple computations like: **1+1**, **pi/exp(1)** (that is π/e), **factorial(6)**. Vectors, matrices, and arrays are very important in R since all data sets are stored in these objects, but we will just focus on vectors today. We shall create some vectors/lists: **c(1,2,3,4)**, **1:4**, **seq(1,4)** all create a vector consisting of the numbers 1, 2, 3, 4. Now store the vector 1:4 to the variable **vec** by **vec<-1:4**. Type **vec** and you should see "[1] 1 2 3 4".

To add 3 to every element, simply type **3+vec**; to square every element, **vec^2**; to compute the length, **length(vec)**; the average, **mean(vec)**; standard deviation, **sd(vec)**; and a summary, **summary(vec)**.

Any time you have a question about a command, simply type a question mark before the command, like **?mean**.

Exercises: (a) Make a vector containing {1, 1, 2, 2, 3, 3, 4} and call it **vec2**;
(b) Compute the mean and standard deviation of **vec2**.

Higher-Level Functions

Create a vector of 50 i.i.d. standard normal data by **rnorm(50)** and save it to the variable **x**. (This can be done via the command **x<-rnorm(50,0,1)**, where here 50 represents the number of data to generate, 0 is the mean and 1 is the standard deviation.) Indeed, **x** is an example of a discrete time series, albeit a very trivial one. Knowing that the time series $\{X_i\}$ (i = 1, . . . , 50) is really just i.i.d. data, what is the best predictor of X_{51}? We shall create a time series plot of the data with **ts.plot(x)**. To get see a plot of the autocorrelation function, type **acf(x)**. These plots appear below in Figure 1 and Figure 2, respectively.

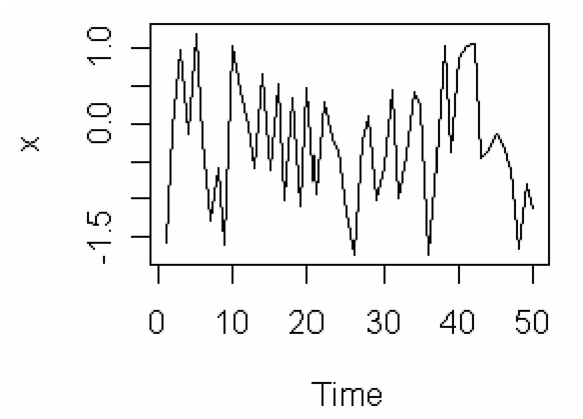

Figure 1: Time Series Plot of x

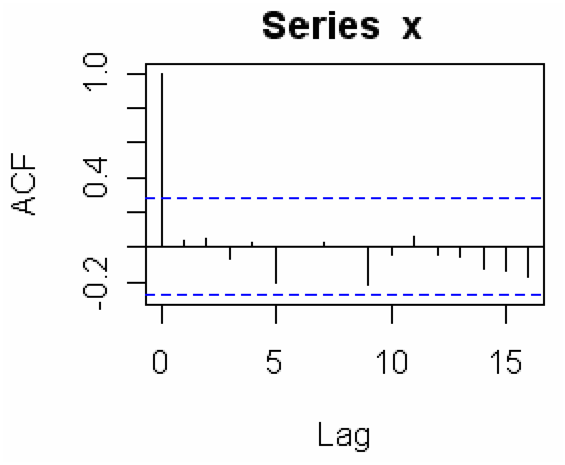

Figure 2: ACF Plot of x

What is the autocorrelation at the zeroth lag? To see the acf numbers (corresponding to the plot), type **acf(x)$acf**. Sometimes a plot of the partial-autocovariance function (pacf) can be useful (do not worry if you do not know what the pacf is), simply type **pacf(x)**.

We are not limited to generating random normal variables. The command **rbinom(*N, n, p*)** generates N data from a Binomial(n, p) distribution. The command **rexp(*N*, λ)** generates N data from an exponential distribution with rate λ. The command **runif(*N*, min, max)** generates N data from a Uniform(min, max) distribution. There are many other distributions built into R. Type in **?rbinom** and scroll through the help file to look for other commands that begin with "r…".

Let us simulate a time series of length 100 from the AR(1) process $X_t = 0.5X_{t-1} + Z_t$ where $Z_t \sim$ i.i.d. $N(0, 1)$. We create the process and store it to the variable **x2** with the command **x2<-arima.sim(n= 100, list(ar = c(.5)))**. Again, we can consider a time series plot of the data with the command **ts.plot(x2)**. This is shown in Figure 3. We can compute the pacf of **x2** as well. This is shown in Figure 4.

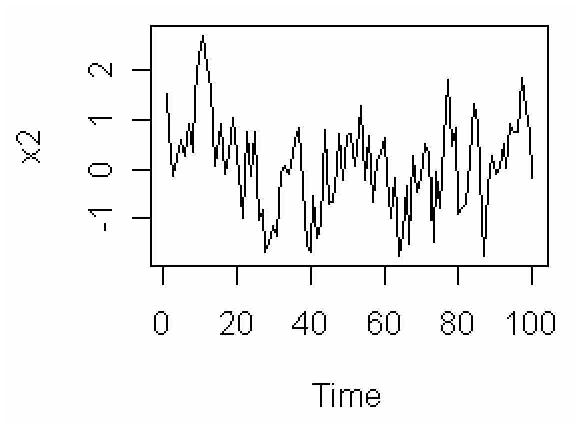

Figure 3: Time Series Plot of x2

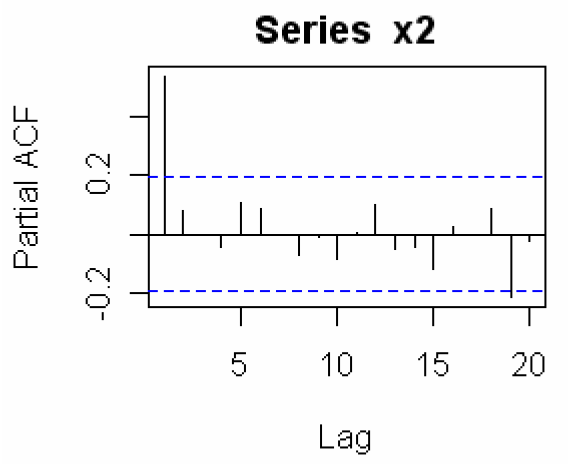

Figure 4: PACF Plot of x2

Exercises: (a) create 100 iid uniform(0,1) stored into variable **y**;
(b) create a time series plot, acf plot, and pacf plot of **y** (to prevent R from overwriting your previous plots, use the command **windows()**)
(c) compute the acf at lag 2 for **y**;
(d) generate 1000 data from the AR(1) process above and store it as **y2**;
(e) plot the pacf of **y2**

Importing Data

There are tons of ways to import data into R. Consider the dataset located in the file http://www.math.ucsd.edu/~wgarner/math181e/sp500.txt. This data set has the daily returns of the S&P500 from August 30, 1979 to August 30, 1991. (It consists of 3,035 data points.)

To load files from a website directly into R, you can use the command
sp500<-scan("http://math.ucsd.edu/~wgarner/math181e/sp500.txt")

Alternatively, go to http://www.math.ucsd.edu/~wgarner/math181e/sp500.txt and download the sp500.txt file to your desktop. Make sure the you change the R command directory to the directory where you downloaded the file with the command **setwd**, like in **setwd("C:/Documents and Settings/User Name/Desktop/")** or by going to File − > Change dir and browse until you find the correct folder. Once the data has been downloaded and the directory set correctly, simply scan in the data to the variable sp500 with the command **sp500<-scan("sp500.txt")**.

Exercises: (a) Create a time series plot of sp500;
(b) Create a time series plot of each day's price difference by **ts.plot(diff(sp500))**;
(c) Fit an AR model to the differenced data with **ar(diff(sp500))**. What is the order of the AR model that was automatically chosen;
(d) Plot the pacf of the differenced data;
(e) Plot an estimate of the spectral density with the command **spec.pgram(diff(sp500),kernel("daniell", 70))**

Acknowledgements

I am grateful to Arthur Berg, whose original handout on R and S-Plus was the basis for these notes.