Some R basics

1 Getting help

Learn how `function.x` works (assuming you have the required package installed).	`?function.x`
Find out about `function.x` on the web	Google `function.x R cran` There is also "Rseek.org", but I have never found it useful.
Read about how to use R	See references on Prof Wawro's webpage. Work through Gelman/Hill 2009. Search on the web using "R" and "cran" in your Google searches.

2 Loading packages

Packages contain functions that you may need in conducting your analysis. Most of the functions that you need (e.g. random number generators from all univariate distributions, the matrix operators, etc) come in the basic packages that are installed in R automatically. Sometimes you will have to load packages. If you use the Fox book, you are asked to load the `car` package, which contains specialty functions to go along with the examples in the book. If you work with Gelman/Hill 2009, you are asked to load the `arm` package. If you want to work with multivariate normal distributions, then you could load and employ the `mvtnorm` or `MASS` package. If you want to do chained regression multiple imputation, you would use the `mice` or *mi* package. And so on. All packages have documentation and instructions posted to the web.

Download and install a new package onto your computer.	`install.packages()` or use the Packages drop-down menu in the R console.
Loading a package that you have installed on your computer	`library()`
Package that has R import or export Stata, SPSS, or Excel-readable datasets	`foreign`
Some models and tools that are regularly used in social science. (Note: you won't use this for any class assignments. It's just a reference for you.)	`zelig`
Some panel data models for R. (Note: You will not use this package for any assignments. It is just a reference for you. Of course, you could use this to check whether your code is working correctly, but using this to generate results without programming things correctly won't get you credit for your assignments.)	`plm`

There are tons of other packages for every statistical application that you can imagine.

3 Loading data

Setting your working directory	`setwd()`
Navigating to a file outside your working directory	Use `c:/...` (Note "/" always works and "\" sometimes doesn't, so use "/".)
Tab or comma delimited text	`read.table()`
Stata (.dta)	`read.dta()` from `foreign` package
Excel (saved as .csv)	`read.csv()` from `foreign` package
SPSS (.por, .sav)	`read.spss()` from `foreign` package

4 Loops

Create $y_t = y_{t-1} + 5$ for $t = 1, ..., 10$, where $y_1 = 0$.

```
y <- rep(NA,10)
y[1] <- 0
for (i in 2:10){
                y[i] <- y[i-1] + 5
                }
```

Results: `[1] 0 5 10 15 20 25 30 35 40 45`

You will occasionally have to use loops in your programming. It is more efficient to use vector/matrix operations rather than loops whenever you can—it makes the code shorter, reduces space for error, and is often computationally faster. Thus, given $x = (1,2,3)$ and $y = (11,12,13)$, you could compute $x_1y_1 + x_2y_2 + x_3y_3 = 1 \cdot 11 + 2 \cdot 12 + 3 \cdot 13 = 74$ as follows:

```
x <- c(1,2,3)
y <- c(11,12,13)
xy.long <- rep(NA, 3)
for(i in 1:3){
   xy.long[i] <- x[i]*y[i]
}
xy <- sum(xy.long)
```

but you would be much better off using a vector operation. (You should know what operation. Code for vector and matrix operations is below.)

5 Random draws

Create 100 draws of $\nu \sim N(0, 10^2)$.

```
nu <- rnorm(100,mean=0,sd=10)
```

Create $y_t = 0.9y_{t-1} + \varepsilon_t$ for $t = 1, ..., 50$, where $\varepsilon \sim t_{[\text{central},2]}$ and $y_1 = 0$. Do it 20 times. Graph the series on a single plot.

```
outmat <- matrix(NA, nrow=50, ncol=20)
emat <- matrix(c(rep(NA, 20),rt(49*20,df=2)),
               nrow=50,ncol=20,byrow=T)
outmat[1,] <- 0
for(j in 1:ncol(outmat)){
    for(i in 2:nrow(outmat)){
    outmat[i,j] <-.9*outmat[i-1,j]+emat[i,j]
    }
}

ymax <- max(outmat)
ymin <- min(outmat)
colvec <- hsv(seq(from=.1,to=.9,
              length=ncol(outmat)),1,1)

plot(   seq(1:nrow(outmat)),
        outmat[,1], type="l",
        ylim=c(ymin,ymax),
        col=colvec[1], ylab="y_t",xlab="t")

for(i in 2:ncol(outmat)){
    points( seq(1:nrow(outmat)),outmat[,i],
            type="l",col=colvec[i])
}
```

This will come in handy for creating fake data for monte carlo simulations. You also use random draws in imputation algorithms and simulation-based methods. (Quick quiz: Is the latter series stationary? Why or why not? If not, does it converge to a stationary series?)

6 Matrix algebra

You will be doing lots and lots of this for this class. If you search in Google, you will find thousands of tutorials on matrix algebra in R. For example, you can look at Chapters 2 and 5 in Venables and Smith, *An Introduction to R*, which is available for free online. The R references on Prof Wawro's website and the R books on the syllabus also contain lots of information. Here are some basics:

$x'y$	`t(x)%*%y` (`x*y` does elementwise mult.)
AB (assuming conformability)	`A%*%B` (`A*B` does elementwise mult.)
A^{-1} (assuming invertibility)	`solve(A)` (`solve` is a general solver for systems of linear equations; specifying only a matrix inside the argument, with no outcome vector, simply yields an inverse.)
$A \otimes B$	`A%x%B` (you'll need the kronecker product a lot for panel data, because you frequently get block matrices)
$(X'X)^{-1}X'y$	`solve(t(X)%*%X)%*%t(X)%*%y`

$\sqrt{diag\left((X'X)^{-1}X'\begin{pmatrix}\hat{\varepsilon}_1^2 & 0 & \dots \\ 0 & \hat{\varepsilon}_2^2 & \dots \\ \vdots & \vdots & \ddots\end{pmatrix}X(X'X)^{-1}\right)}$	`HatMat <- X%*%solve(t(X)%*%X)`
	`Omega <- diag(diag(e%*%t(e)),n)`
	`VC <- t(HatMat)%*%Omega%*%HatMat`
	`sqrt(diag(VC))`

Quick quiz: what are the last two quantities? You've *got* to know this to follow this class.

7 Saving your work

Storing your commands to run in another session.	This happens automatically if all of your commands and programs are done in a script. Never work through the command line in the console.
Outputting results to a text file (useful when you have lots of results that will exceed the length of the console space).	`sink()`
Save everything so that you can come back again and pick up right where you left off.	`save.image()` or use "File → Save workspace..." from the drop-down menus in the console.