NYU R Workshop

Thinking in R

`http://drewdimmery.com/r-workshop-2013/`

Drew Dimmery

NEW YORK UNIVERSITY

February 8, 2013

Outline

Why R?

- Free!
- Open!
- Wide array of statistical procedures
- Extensible
- Interface-able
- Large volumes of online support

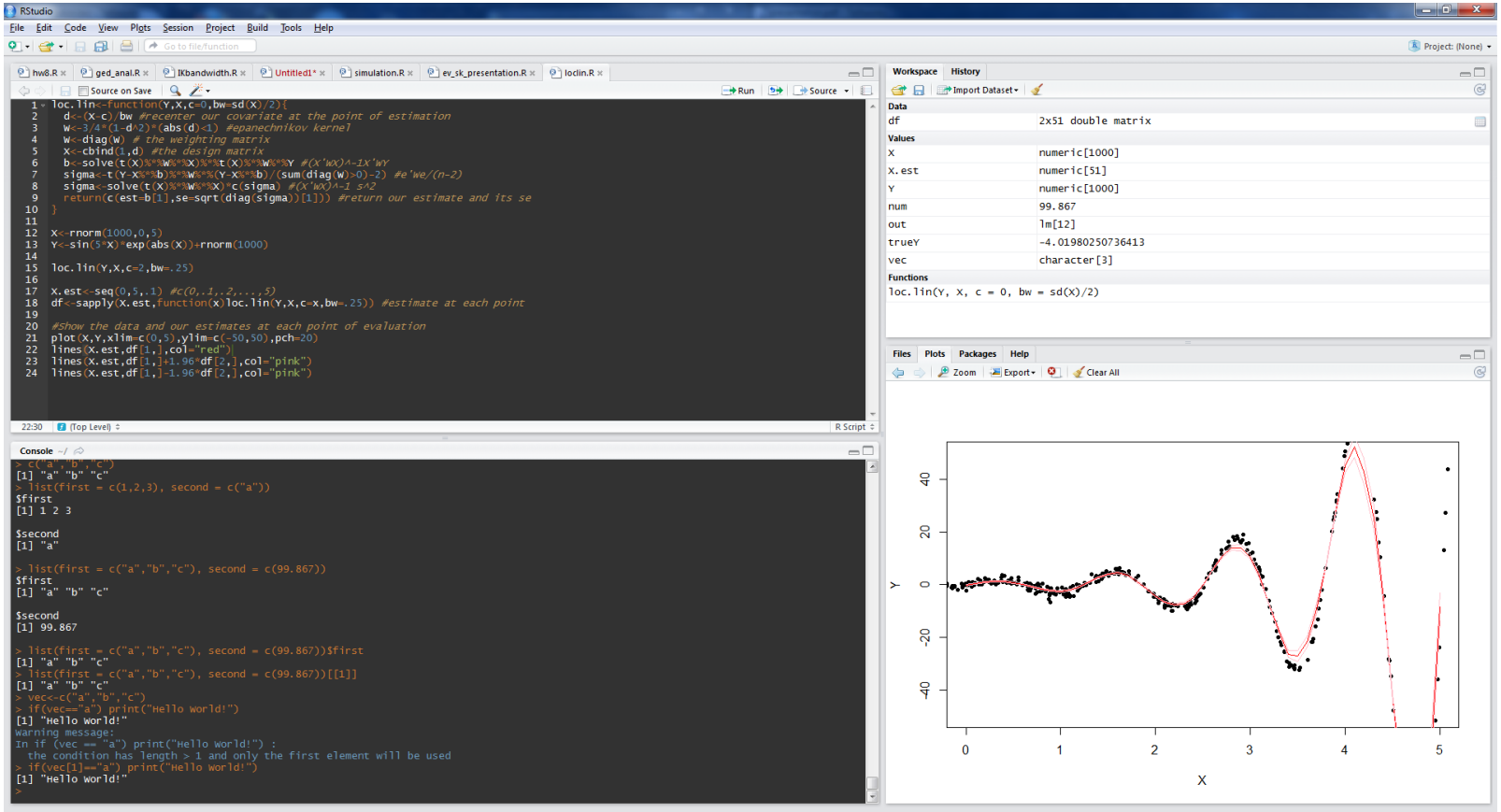
How to setup a workflow

- `http://www.r-project.org/`
- Traditional Console and text editor
  - Notepad++
  - vim
  - Emacs
  - Kate, Textmate, Sublime, etc
- **Rstudio**
- Other IDEs (Eclipse, Netbeans, etc)

Rstudio

NYU R Workshop

Drew Dimmery



So your code won't run. What now?

- Documentation
  Example: `?lm`
- Google
- CRAN: Reference Manual, vignettes
  Example: `http://cran.r-project.org/web/packages/AER/index.html`
- Journal of Statistical Software
  Example: `http://www.jstatsoft.org/v27/i02`
- Stack Overflow
- Listservs

How to read R documentation

- `http://stat.ethz.ch/R-manual/R-patched/library/stats/html/lm.html`
- Call: `?lm`
- Sections:
  - Description
  - Usage
  - Arguments
  - Details
  - Value
  - Author(s)
  - References
  - See Also
  - Examples

CRAN

- `AER` on CRAN: `http://cran.r-project.org/web/packages/AER/index.html`

`AER: Applied Econometrics with R`

Functions, data sets, examples, demos, and vignettes for the book Christian Kleiber and Achim Zeileis (2008), Applied Econometrics with R, Springer-Verlag, New York. ISBN 978-0-387-77316-2. (See the vignette for a package overview.)

Version:	1.1-9
Depends:	R (≥ 2.10.0), stats, car (≥ 2.0-1), Formula (≥ 0.2-0), lmtest, sandwich, strucchange, survival, zoo
Imports:	stats
Suggests:	boot, dynlm, effects, foreign, ineq, KernSmooth, lattice, MASS, mlogit, nlme, nnet, np, plm, pscl, quantreg, ROCR, sampleSelection, scatterplot3d, systemfit, rgl, truncreg, tseries, urca
Published:	2011-12-10
Author:	Christian Kleiber [aut], Achim Zeileis [aut, cre]
Maintainer:	Achim Zeileis <Achim.Zeileis at R-project.org>
License:	GPL-2
Citation:	AER citation info
In views:	Econometrics, Survival, TimeSeries
CRAN checks:	AER results

`Downloads:`

Package source:	AER_1.1-9.tar.gz
MacOS X binary:	AER_1.1-9.tgz
Windows binary:	AER_1.1-9.zip
Reference manual:	AER.pdf
Vignettes:	Applied Econometrics with R: Package Vignette and Errata Sweave Example: Linear Regression for Economics Journals Data
News/ChangeLog:	NEWS
Old sources:	AER archive

`Reverse dependencies:`

Reverse depends:	rdd, tonymisc
Reverse suggests:	censReg, lmtest, micEconCES, mlogit, plm, REEMtree, sandwich

Journal of Statistical Software

› `plm` in JSS: `http://www.jstatsoft.org/v27/i02/`

Journal of Statistical Software

July 2008, Volume 27, Issue 2. *http://www.jstatsoft.org/*

Panel Data Econometrics in R: The plm Package

Yves Croissant
Université Lumière Lyon 2

Giovanni Millo
University of Trieste and Generali SpA

Abstract

Panel data econometrics is obviously one of the main fields in the profession, but most of the models used are difficult to estimate with R. **plm** is a package for R which intends to make the estimation of linear panel models straightforward. **plm** provides functions to estimate a wide variety of models and to make (robust) inference.

Keywords: panel data, covariance matrix estimators, generalized method of moments, R.

1. Introduction

Panel data econometrics is a continuously developing field. The increasing availability of data observed on cross-sections of units (like households, firms, countries etc.) *and* over time has given rise to a number of estimation approaches exploiting this double dimensionality to cope with some of the typical problems associated with economic data, first of all that of unobserved heterogeneity.

Timewise observation of data from different observational units has long been common in other fields of statistics (where they are often termed *longitudinal* data). In the panel data field as well as in others, the econometric approach is nevertheless peculiar with respect to experimental contexts, as it is emphasizing model specification and testing and tackling a

Things that can be confusing

- At the prompt, `>` means you're good to go, `+` means a parenthesis or bracket is open.
- Case sensitive
- Use `/` in path names.
- R uses variables – there is no "sheet" here, like in Stata
- **R is a programming language**

Using third party code

- Relevant commands: `install.packages`, `library`
- How does one find and install an appropriate package?
- Example:

```
1 ?covariance
2 ??covariance
3 install.packages("sandwich")
4 library("sandwich")
5 ?vcovHC
```

- In reality, Google to see if an appropriate package exists.

Data Types

- Character – strings
- Double / Numeric – numbers
- Logical – true/false
- Factor – unordered categorical variables
- Objects – it's complicated

Character

- Example: `str<-"This is a string"`
- String concatenation:

```
1 > paste("This","is","a","string",sep=" ")
2 [1] "This is a string"
```

- Numbers can be strings: `as.character(99)`
- Possible to determine type: `class(str)`

Numeric

- Most of the time you'll work with these.
- `num<-99.867`
- `round(str, digits=2)`
- Many useful (and not often useful) operations:
  - `sin`
  - `exp`
  - `log`
  - `factorial`
  - `choose`
  - `BesselJ`
  - `pi` (not a function)
  - And many more

Logical

- The logical type allows us to make statements about truth

```
> 2==4
[1] FALSE
> str != num
[1] TRUE
> "34" == 34
[1] TRUE
```

- Equal to: `==`
- Not equal to: `!=`
- Greater than `>`
- Greater than or equal to: `>=`
- Less than: `<`
- Less than or equal to: `<=`
- Not: `!`
- Logical and: `&`
- Logical or: `|`
- Any are true: `any`
- All are true: `all`

Objects

- Many functions will return objects rather than a single datatype
- Example (no, this won't work, since the data doesn't exist):

```
1 > out.lm <- lm(Y~X)
2 > class(out.lm)
3 [1] "lm"
```

- Objects can have other data embedded inside them.

```
1 > out.lm$rank
2 [1] 2
```

- Sometimes operations will work differently for different sorts of objects.

Other ways to “hold” data

- Vectors
- Lists
- Matrices
- Dataframes

Vectors

- **Everything** is a vector (almost).
- Just like a vector in math

```
> as.vector(4)
[1] 4
> 4
[1] 4
```

- We can **c**ombine vectors with `c`

```
> vec <- c("a","b","c")
> vec
[1] "a" "b" "c"
> c(2,3,c(4,5))
[1] 2 3 4 5

> c(1,2,3,4) + c(1,2)
[1] 2 4 4 6    #Recycles shorter vector
```

Vectors

- Vectors are indexed as you would probably expect – just like in math. Or you can name elements yourself.

```
> vec[1]
[1] "a"
> names(vec) <- c("first","second","third")
> names(vec)
[1] "first" "second" "third"
> vec$first
[1] "a"
> vec["first"]
[1] "a"
```

- Vectors can include missing values via `NA`

```
> vec[1]<-NA
> vec
[1] NA b c
> is.na(vec)
[1]  TRUE FALSE FALSE
>vec[!is.na(vec)] #vec[complete.cases(vec)]
[1] "b" "c"
```

Lists

- Lists are sort of like a vector, but they allow for mixing of types and lengths in arbitrary ways

```
> listie <- list(first = vec , second = num)
> listie
$first
[1] "a" "b" "c"

$second
[1] 99.867

> listie[[1]]
[1] "a" "b" "c"

> listie$first
[1] "a" "b" "c"
```

- One could even save an object to one element of a list.

Matrices

› We can create matrices too. They're just like normal matrices.

$$A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

$$A_{ij} \qquad A_{1,2} = 3 \qquad A_{1,\cdot} = (1,3)$$

```
1  > A <- matrix(c(1,2,3,4),nrow=2,nrow=2)
2  > A
3       [,1] [,2]
4  [1,]    1    3
5  [2,]    2    4
6  > A[1,2]
7  [1] 3
8  > A[,2]
9  [1] 3 4
10 > A[1,]
11 [1] 1 3
```

Matrices

- We can also perform matrix operations

```
> solve(A) #A^{-1}
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
> 10*A
     [,1] [,2]
[1,]   10   30
[2,]   20   40
> B<-diag(c(1,2))
> B
     [,1] [,2]
[1,]    1    0
[2,]    0    2
> A%*%B
     [,1] [,2]
[1,]    1    6
[2,]    2    8
```

Matrices

› Continued

```
> t(A)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> rbind(A,B)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    1    0
[4,]    0    2
> cbind(A,B)
     [,1] [,2] [,3] [,4]
[1,]    1    3    1    0
[2,]    2    4    0    2
> c(1,2,3)%x%c(1,1) #Kronecker Product
[1] 1 1 2 2 3 3
```

Matrices

› Naming things

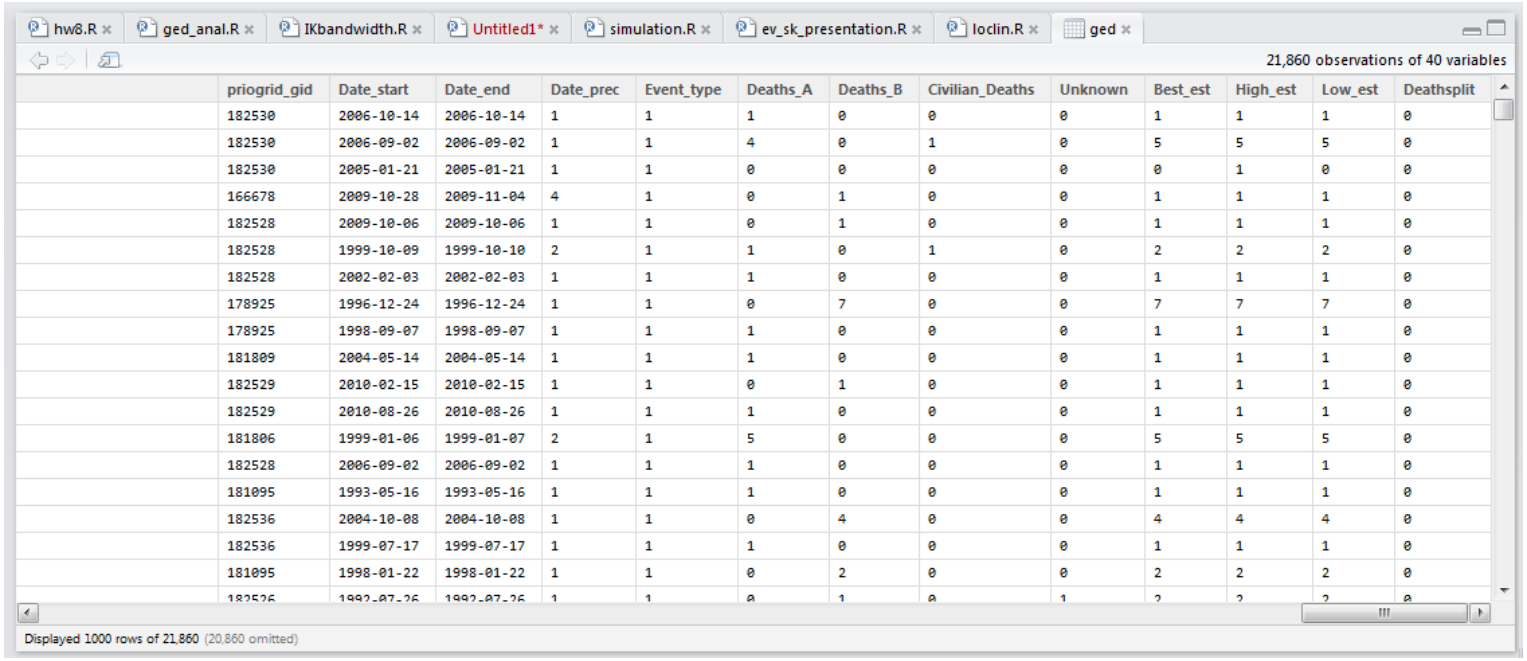
```
> rownames(A)
NULL
> rownames(A)<-c("a","b")
> colnames(A)<-c("c","d")
> A
  c d
a 1 3
b 2 4
> A[,"d"]
a b
3 4

> A[3] #Matrices are vectors
[1] 3
```

Dataframes

- The workhorse.
- Basically just a matrix that allows mixing of type by column

```
1 > rstudio::viewData(ged)
```



Control Flow

- loops
- if/else
- functions
- Useful Functions

Loops

- for loops – just a way of saying “do this **for** each element of the index”
- "this" is defined in what follows the "for" expression.

```
> for(i in 1:5) {
+  cat(i*10," ")
+ }
10  20  30  40  50

> for(i in 1:length(vec)) {
+  cat(vec[i]," ")
+ }
a  b  c

> for(i in vec) {
+  cat(i," ")
+ }
a  b  c
```

If/Else

- To conditionally execute code, we use if/else statements.

```
1  > if(vec[1]=="a") print("Hello World!")
2  [1] "Hello World!"
3
4  > if(vec[1]=="b") {
5  +   print("Hello World!")
6  + } else {
7  +   print("!dlroW olleH")
8  + }
9  [1] "!dlroW olleH"
```

Vectorized If/Else

- What if we want to conditionally execute something for each element in a vector?

```
> new <- vector(length=length(vec))
> for(i in 1:length(vec)) {
+   if(vec[i]=="a") {
+     new[i]<-13
+   } else {
+     new[i]<-0
+   }
+ }
> new
[1] 13  0  0

> new <- ifelse(vec=="a",13,0)
> new
[1] 13  0  0
```

Functions

- $f : X \to Y$
- Functions in R are essentially the same.

```
> add3<-function(X) {
+ return(X+3)
+ }
> add3(2)
[1] 5

> makeGroups<-function(groups,members=1) {
+   return((1:groups)%x%rep(1,members))
+ }
> makeGroups(5)
[1] 1 2 3 4 5
> makeGroups(5,2)
[1] 1 1 2 2 3 3 4 4 5 5
```

- Drew will now get on a function soapbox

Useful functions

- Note: Most functions don't do complete case analysis by default (usually option `na.rm=TRUE`)

- `print`
- `cat`
- `paste`
- `with`
- `length`
- `sort`
- `order`
- `unique`
- `rep`
- `nrow`
- `ncol`
- `complete.cases`
- `subset`
- `merge`
- `mean`
- `sum`
- `sd`
- `var`
- `lag`
- `lm`
- `model.matrix`
- `coef`
- `vcov`
- `residuals`
- `vcovHC` (from `sandwich`)
- `ivreg` (from `AER`)
- `countrycode` (from `countrycode`)
- `summary`
- `pdf`
- `plot`
- Tools from `plm`
- Many more.

Distributional functions

- `?Distributions`
- They have a consistent naming scheme.
- `rnorm`, `dnorm`, `qnorm`, `pnorm`
- `rdist` – generate random variable from dist
- `ddist` – density function of dist
- `qdist` – quantile function of dist
- `pdist` – distribution function of dist
- look at documentation for parameterizations.
- Example: `rnorm(100)` generates 100 samples from a standard normal

Cluster Robust SEs


```
robust.se <- function(model,
                      cluster=1:length(model$residuals)) {
  require(sandwich)
  require(lmtest)
  M <- length(unique(cluster))
  N <- length(cluster)
  K <- model$rank
  dfc <- (M/(M - 1)) * ((N - 1)/(N - K))
  uj  <- apply(
        estfun(model),
        2,
        function(x) tapply(x, cluster, sum)
      )
  rcse.cov <- dfc * sandwich(model,meat = crossprod(uj)/N)
  rcse.se <- coeftest(model, rcse.cov)
  return(list(rcse.cov, rcse.se))
}
```

Functions and Objects

- Sometimes, passing an object to a function will do something you might not expect.

```
> summary(vec)
   Length     Class      Mode
        3 character character

> summary(c(1,2,3,4))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.75    2.50    2.50    3.25    4.00

> summary(data.frame(A))
       c               d
 Min.   :1.00   Min.   :3.00
 1st Qu.:1.25   1st Qu.:3.25
 Median :1.50   Median :3.50
 Mean   :1.50   Mean   :3.50
 3rd Qu.:1.75   3rd Qu.:3.75
 Max.   :2.00   Max.   :4.00
```

- Why? `?summary`, `?summary.lm`

The *apply family

- These functions allow one to efficiently perform a large number of actions on data.
- `apply` – performs actions on the rows or columns of a matrix (1 for rows, 2 for columns)
- `sapply` – performs actions on every element of a vector
- `tapply` – performs actions on a vector by group
- `replicate` – performs the same action a given number of times

apply

- `apply` – performs actions on the rows or columns of a matrix (1 for rows, 2 for columns)

```
> A
  c d
a 1 3
b 2 4
> apply(A,1,sum)
[1] 4 6
> apply(A,2,mean)
[1] 1.5 3.5
```

sapply

- `sapply` – performs actions on every element of a vector

```
> vec
[1] "a" "b" "c"
> sapply(vec,function(x) paste0(x,".vec"))
      a       b       c
"a.vec" "b.vec" "c.vec"

#Bad example, though
> paste0(vec,".vec")
[1] "a.vec" "b.vec" "c.vec"
```

- `replicate` is basically just `sapply(1:N,funct)` where `funct` never uses the index.

tapply

- `tapply` – performs actions on a vector by group

```
> tapply(1:10,
+   makeGroups(5,2),
+   function(x) mean(x)
+ )
  1   2   3   4   5
1.5 3.5 5.5 7.5 9.5
```

Working with data

- Input
- Output

Input


```
1 setwd("data")
2 list.files()
3 dat<-read.csv("dataset.csv")
4
5 library("foreign")
6 dat<-read.dta("dataset.dta")
```

- Useful options:
  - `sep`
  - `na.strings`
- If you have a different format, Google it.

Simulate some fake data

- We will now just simulate some data for demonstration

```
> set.seed(1023)
> X<-rnorm(1000,0,5)
> Y<-sin(5*X)*exp(abs(X))+rnorm(1000)
> dat<-data.frame(X,Y)

#Simple Linear Regression
> dat.lm<-lm(Y~X,data=dat)

> summary(dat.lm)
```

- Look at `?summary.lm`

Output

- Multiple options: `xtable`, `apsrtable`, `stargazer`
- Can be used with, for instance, `booktabs` for prettier LaTeX

```
1  > install.packages("xtable")
2  > library(xtable)
3  > xtable(dat.lm) #shown below
4
5  #This is the same as
6  > xtable(summary(dat.lm)$coefficients)
7
8  #Look at the following:
9  > summary(dat.lm)$coefficients
10 > class(summary(dat.lm)$coefficients)
11 [1] "matrix"
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	7.8851	4073.7725	0.00	0.9985
X	2989.0614	828.6452	3.61	0.0003

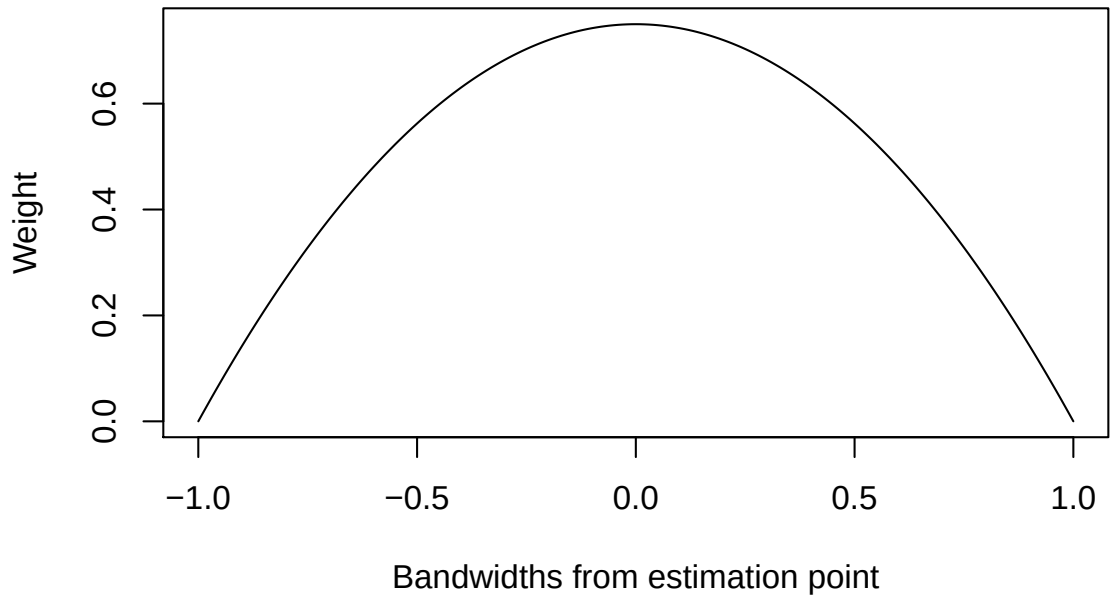
Plotting

- Plotting is just all about coordinate pairs.
- `plot(x,y)` plots the pairs of `x` and `y` in order
- Notable options:
  - `type` – determines whether you plot points, lines, or whatnot
  - `pch` – determines plotting character
  - `xlim` – x limits of the plot (similar for Y)
  - `xlab` – label on X-axis
  - `main` – main title
  - `col` – color
- A huge number of other options. Read the manual.
- Some objects will use a special plotting function of their own.
- Example `plot(dat.lm)`

Tying it all together

› We will now create a function for local linear regression.

```
1  pdf("locallin.pdf",height=4,width=6)
2  plot(
3    seq(-1,1,.01),
4    3/4*(1-seq(-1,1,.01)^2),
5    type="l",
6    xlab="Bandwidths from estimation point",
7    ylab="Weight"
8  )
9  dev.off()
```



Tying it all together

- W is an $n \times p$ diagonal weighting matrix.
- Diagonal entries are $\frac{3}{4} \cdot (1 - d^2) \cdot 1_{\{|d| \leq 1\}}$ where $d = \frac{(X-c)}{\text{bandwidth}}$
- $\hat{\beta}_c = (X'WX)^{-1}X'WY$
- Covariance matrix: $s^2(X'WX)^{-1}$

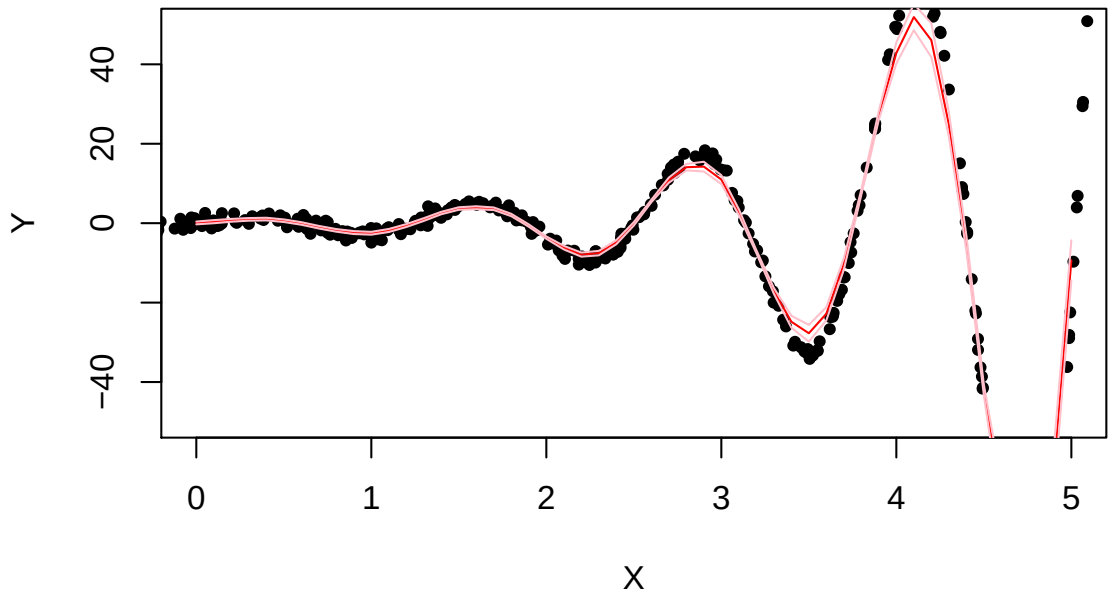
```
1  loc.lin<-function(Y,X,c=0,bw=sd(X)/2){
2    d<-(X-c)/bw
3    W<-3/4*(1-d^2)*(abs(d)<1)
4    W<-diag(W)
5    X<-cbind(1,d)
6    b<-solve(t(X)%*%W%*%X)%*%t(X)%*%W%*%Y
7    sigma<-t(Y-X%*%b)%*%W%*%(Y-X%*%b)/(sum(diag(W)>0)-2)
8    sigma<-solve(t(X)%*%W%*%X)*c(sigma)
9    return(c(est=b[1],se=sqrt(diag(sigma))[1]))
10 }
```

Testing it out

- We generate some random data, estimate our regressions at various points, and then plot it all.
- $X \sim N(0,5)$
- $Y_i = \sin(5 \cdot X_i) \cdot e^{|X_i|} + \epsilon_i, i \in 1, \ldots, n$
- $\epsilon_i \sim N(0,1)$

```
1  X<-rnorm(1000,0,5)
2  Y<-sin(5*X)*exp(abs(X))+rnorm(1000)
3  loc.lin(Y,X,c=2,bw=.25)
4
5  X.est<-seq(0,5,.1)
6  dat.llm<-sapply(
7    X.est,
8    function(x)loc.lin(Y,X,c=x,bw=.25)
9  )
10
11 plot(X,Y,xlim=c(0,5),ylim=c(-50,50),pch=20)
12 lines(X.est,dat.llm[1,],col="red")
13 lines(X.est,dat.llm[1,]+1.96*dat.llm[2,],col="pink")
14 lines(X.est,dat.llm[1,]-1.96*dat.llm[2,],col="pink")
```

Tying it all Together




```
1 pdf("locallin2.pdf",height=4,width=6)
2 plot(X,Y,xlim=c(0,5),ylim=c(-50,50),pch=20)
3 lines(X.est,df[1,],col="red")
4 lines(X.est,df[1,]+1.96*df[2,],col="pink")
5 lines(X.est,df[1,]-1.96*df[2,],col="pink")
6 dev.off()
```

Conventions and Good Habits

- Comment your code. You won't remember what you did in 2 months.
- Make your variables mean something. `tmp` is a horrible name. Don't use it.
- Always be making functions.
- Be consistent in style – brackets on the same line as `for`
- Indent your code – but not too much.
- If you're just clever enough to write it, you aren't clever enough to debug it.
- Fiddle with things. That's how you learn.
- Save your work (and consider offsite backups via Dropbox/git/etc)
- Break rules when it makes for cleaner, easier to understand code.