Department of Statistics
University of California, Los Angeles

Advanced Graphics in R

Ryan Rosario

January 27, 2010

Outline

Code for Graphics

The code to produce the graphics in this presentation is available at the following URL, for you to review at your leisure.

`http://www.stat.ucla.edu/~rosario/scc/10w_agr_code.R`

A copy of these slides (big) for following along is available at the following URL.

`http://www.stat.ucla.edu/~rosario/scc/10w_agr-big.pdf`

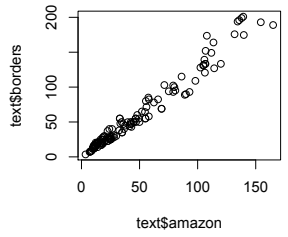
A copy of these slides (handout) for printing at home is available at the following URL.

`http://www.stat.ucla.edu/~rosario/scc/10w_agr.pdf`

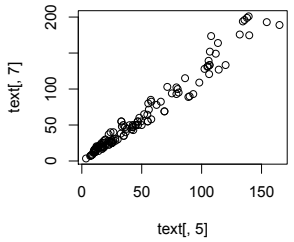
Early Days with R

Using the `plot` command is so simple, but when first getting started with R, something like the following is discouraging...

```
plot(text$Amazon.Sale.Price,
     text$Borders.Price)
```




```
plot(text[,4],text[,7])
```

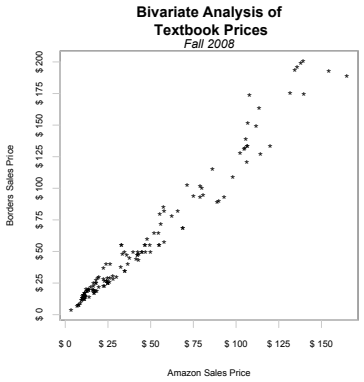


So, What's Wrong with That?

- there is no title to introduce the graphic.
- the axes refer to data frame dimensions, rather than the context of the data.
- data points are too large as displayed.
- data points are "clumped" which reduces signal to noise ratio in the plot.
- there may be multiple classes of data points.

Now What?

Of course it is possible to make beautiful graphics in R.



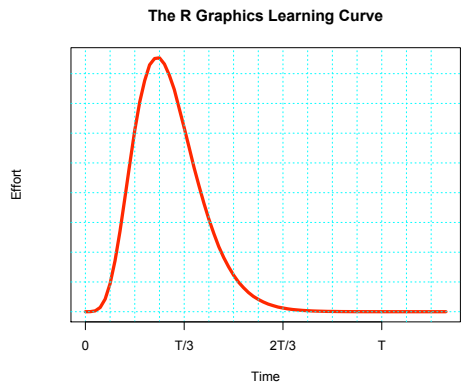
The Learning Curve

...but at first it requires a lot of work!

```
1 plot(text[,5],text[,7],main="Bivariate Analysis of
2 \n Textbook Prices",xlab="Amazon Sales Price",ylab=
      "Borders Sales Price",pch='*',xaxt="n",yaxt="n"
      ,cex.lab=0.75,box=FALSE,bg="grey")
3 mtext("Fall 2008",3,font=3)
4 axis(1,at=seq(0,200,25),paste("$",seq(0,200,25)),
      cex.axis=0.75,col="grey")
5 axis(2,at=seq(0,200,25),paste("$",seq(0,200,25)),
      cex.axis=0.75,col="grey")
6 axis(c(3:4),col="grey",tick=FALSE,labels=FALSE)
7 box("plot",col="grey")
```

Ouch...

The Learning Curve



In Intermediate Graphic in R we have already seen some ways to customize graphics:

`col`	`lines`	`add`	`wireframe`
`main`	`density`	`identify`	`drape`
`pch`	`boxplot`	`ts`	`color.palette`
`legend`	`bwplot`	`mvtsplot`	`contour`
`pie`	`levelplot`	`xyplot`	`scatterplot.matrix`
`hist`	`curve`	`map`	`persp`
`abline`	`lwd`	`points`	

We will skip most of these, and review some of them here.

The par Command

Graphics options can be passed directly to `par`, or to higher level plotting functions.

```
par(..., no.readonly = FALSE)
<highlevel plot> (..., <tag> = <value>)
```

We will stick with the second method for now. We will discuss the first method later.

The par Command

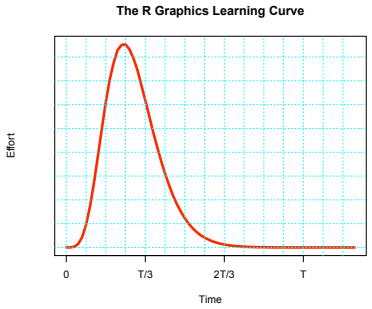
We use a graphics parameter by calling `plot`, or one of its friends (`hist`, `boxplot` etc.) with a comma separated list of *named* options.

```
plot(text[,5],text[,7],main="Bivariate Analysis of
    \n Textbook Prices",xlab="Amazon Sales Price",
    ylab="Borders Sales Price",pch='*',xaxt="n",
    yaxt="n",cex.lab=0.75)
```

The above `plot` command contains the following parameters: `xlab`, `ylab`, `pch`, `xaxt`, `yaxt`, `cex.lab`, `box`.

A Motivating Example

There is a lot of material, so let's start with an example and see where it takes us. Let's look at the Learning Curve graphic.

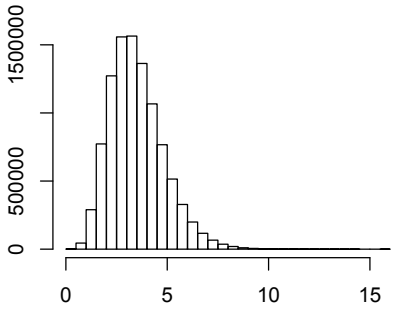


This graphic is an example of how I can express this trend using the graphical parameters in R.

A Motivating Example

First, I generated 100,000 random numbers from a gamma distribution with $k = 7$ and $\theta = 2$ to construct this "trend." The histogram below displays the gamma distribution.

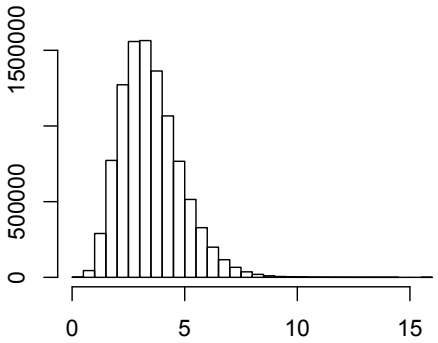
```
my.gamma <- rgamma(100000,7,2)
```



A Motivating Example

I suppress the default axis labels and default plot title by passing an empty string "" to some graphical parameters.

```
hist(my.gamma,xlab="",ylab="",main="")
```



A Motivating Example

But this is not the type of graphic I want. Instead, I want to plot the `density` using a curve. We could use `density` or `dgamma`, but let's work with the histogram. A histogram is an *object* of type `hist`. We can see what goodies this object contains using the `attributes` function.

```
attributes(hist(my.gamma))
```

```
$names
[1] "breaks"      "counts"      "intensities"
    "density"     "mids"        "xname"       "equidist"

$class
[1] "histogram"
```

Extracting Information from `hist`

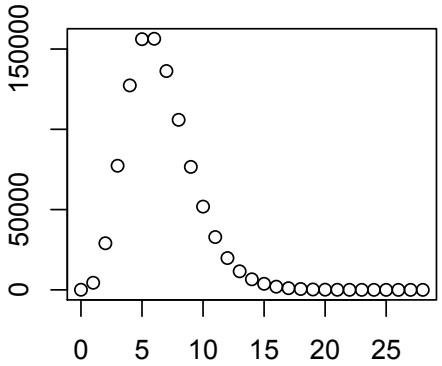
`hist(my.gamma,plot=FALSE)$counts` returns a vector of counts for each *bin* in the histogram, and I use this as the y axis. The number of bins in the histogram can be modified by adding the parameter `br` to the `hist` call.

```
y <- hist(my.gamma,plot=FALSE)$counts
#plot=FALSE suppresses plot, constructs object
x <- seq(0, length(y)-1)
#I used seq to create a dummy axis.
plot(x,y,xlab="",ylab="", main="")
```

The number of bins (or *breaks*) can be controlled using the `br` parameter in the `hist` call.

An Aside: Extracting Information from `hist`

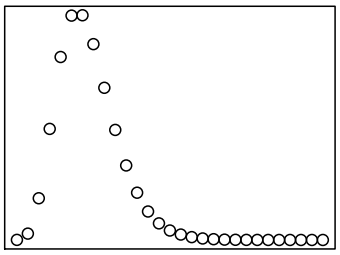
Let's check our progress...



Manipulating Axis

Recall that my x axis has no units and currently does not make sense, so let's replace it with something more appropriate. First we must remove it. Also, my y axis really has no practical meaning, so let's just remove it altogether.

```
plot(x,y,xlab="",ylab="", main="", xaxt="n",yaxt="n
    ")
```



Manipulating Axis

We can then add back new axes that look how we want using `axis`.

1. can put labels at specific places on the x axis using the `at` parameter can
2. can give these tick marks labels given in parameter `labels`.
3. the first parameter in `axis` indicates where to put the axis.

The `side` Argument (first argument of `axis`)

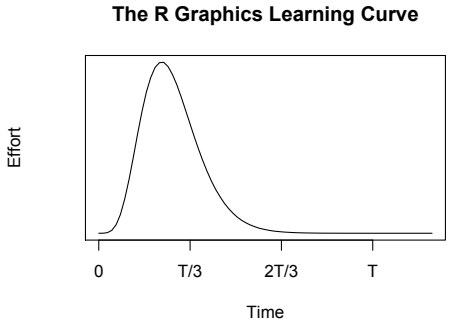
1 = bottom (x), 2 = left (y), 3 = above, 4 = right works for other functions such as `mtext`.

Manipulating Axis

```
1 #Learning Curve graphic
2 y <- hist(my.gamma,br=100,plot=FALSE)$counts
3 x <- seq(0,max(my.gamma),length=length(y))
4 plot(y~x,xlab="Time",ylab="Effort",xaxt="n",
    yaxt="n", main="The R Graphics Learning
    Curve")
5 axis(1,at=seq(0,max(x),length=4), labels=c(0,"
    T/3","2T/3","T"))
6 #Cyan grid.
7 abline(v=seq(0,max(x),length=15),lty=3,col="
    cyan")
8 abline(h=seq(0,max(y),length=15),lty=3,col="
    cyan")
```

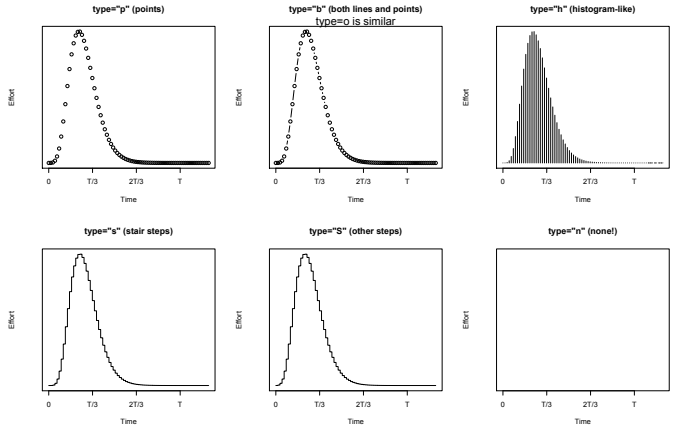
Some More Tinkering: Plot Types

We can create a continuous curve by simply changing the plot type to `type="l"`.



Some More Tinkering: Plot Types

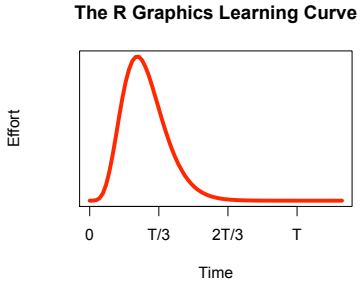
Note: plot types are part of the `plot` function, not `par`.



Some More Tinkering: Line Width and Color

We can change the line width using the `lwd` parameter. `lwd` defaults to 1, and larger integer values provide thicker lines. I use `lwd=4`.

We can also change the color of the line using the `col` parameter. I use `col="red"`, a named color.



Color

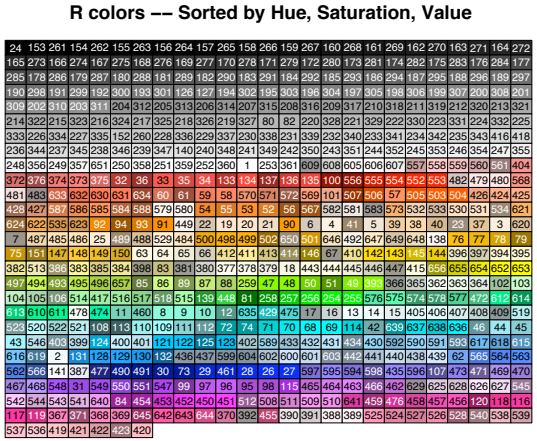
Colors can be specified by name (i.e. `"red"`), by palette code (i.e. 10), or by RGB content `#10AF09`.

A full list of color names is available using the `colors()` command. We can also convert a color name into its corresponding RGB value using the `col2rgb` function.

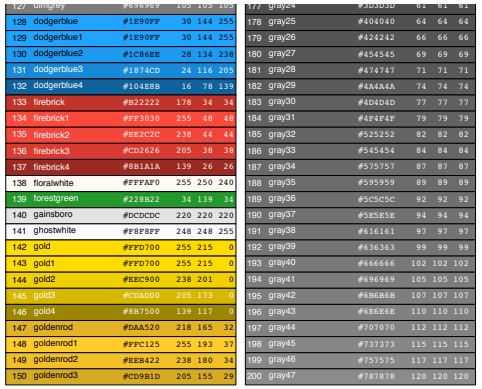
A full index of R colors can be found at `http://research.stowers-institute.org/efg/R/Color/Chart/`. You can create your own index by using the following command:

```
source("http://research.stowers-institute.org
    /efg/R/Color/Chart/ColorChart.R")
```


Color - Palette Codes



Color - Sample Index[1]



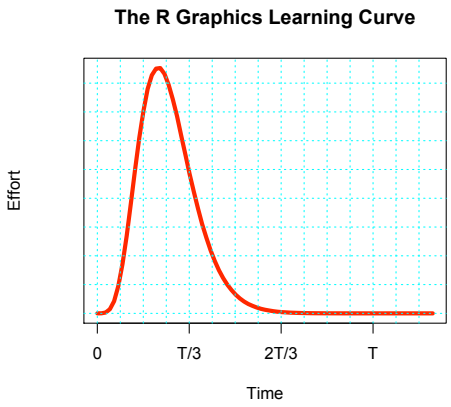
[1] Image from Earl F. Glynn, Stowers Institute for Medical Research
`http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf`

Final Result

After some last few touches using `abline` covered in the previous minicourse, we get the following code:

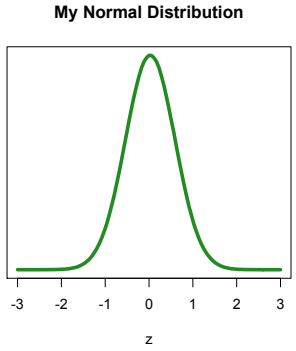
```
#Learning Curve graphic
y <- hist(my.gamma,br=100,plot=FALSE)$counts
x <- seq(0,max(my.gamma),length=length(y))
plot(y~x,type="l",lwd=4,col="red",xlab="Time",ylab="
    Effort",xaxt="n",yaxt="n", main="The R Graphics
    Learning Curve")
axis(1,at=seq(0,max(x),length=4), labels=c(0,"T/3","2T/3"
    ,"T"))
#Cyan grid.
abline(v=seq(0,max(x),length=15),lty=3,col="cyan")
abline(h=seq(0,max(y),length=15),lty=3,col="cyan")
```

Final Result



Exercise 1

Create a plot of the normal distribution. The curve should be thicker than the default, should be colored *Forest Green* (Hint: go 3 slides back). The x axis should represent z scores, and the y axis should be blank. Add the title "My Normal Distribution", add a label to the x axis "z" and leave the y axis blank.



Exercise 1 Solution

```
my.norm <- rnorm(10000000,0,1)
#easiest to use standard normal!
#large number makes the curve smooth.
y <- hist(my.norm,br=100,plot=FALSE)$counts
#dummy x axis.
x <- seq(-3,3,length=length(y))
x.ticks <- seq(-3,3,1)
#Can specify the color in many different ways:
#With color name string
plot(y~x,type="l",lwd=4,col="forestgreen",yaxt="n",xaxt="
    n",xlab="z",ylab="",main="My Normal Distribution")
#With palette code
plot(y~x,type="l",lwd=4,col=139,yaxt="n",xaxt="n",xlab="z
    ",ylab="",main="My Normal Distribution")
#Or with RGB color content
plot(y~x,type="l",lwd=4,col="#228B22",yaxt="n",xaxt="n",
    xlab="z",ylab="",main="My Normal Distribution")
axis(1,at=x.ticks,labels=x.ticks)
```

Exercise 1 Solution: Is there a Better Way?

Yes! Instead of using the features of the `hist` object, we can construct the normal distribution directly using the `dnorm` function

```
x <- seq(-3,3,by=0.01)
plot(dnorm(x)~x,type="l",....)
```

and then we do not need to fudge the x axis. Or even better,

```
curve(dnorm(x),....)
```

Data for this Example

These two datasets come from Facebook. Each dataset contains information about users during two time periods: 2007 and 2009.

```
group.1 <- read.csv("http://www.stat.ucla.edu
    /~rosario/scc/facebook-2007.csv",
    header=TRUE)
group.2 <- read.csv("http://www.stat.ucla.edu
    /~rosario/scc/facebook-2009.csv",
    header=TRUE)
```

Monitors/Plotting Windows

When we execute `plot` (or similar), a new graphics window pops up. If we execute `plot` again, the current graphic is replaced with a new graphic. We can also *open a new window* for the new plot instead, using `dev.new`

We can can specify the `height` and `width` in inches of the new plotting window. This is good when we want to produce several graphics with identical dimensions.

Using Multiple Plotting Windows

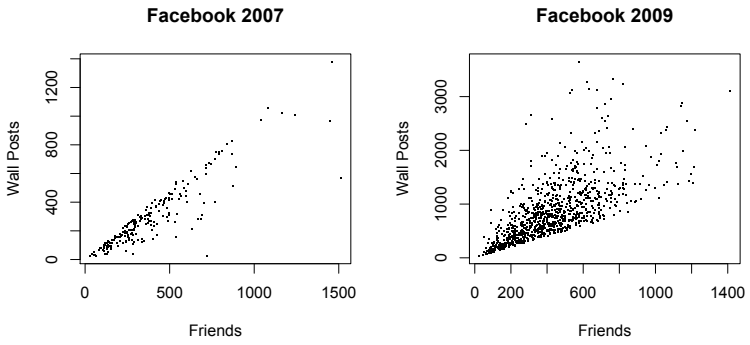
```
dev.new(height=4,width=4)
plot(group.1$Wall.Posts~group.1$Friends,pch='.',
    main="Facebook 2007",xlab="Friends",ylab="Wall
    Posts")
dev.new(height=4,width=4)
plot(group.2$Wall.Posts~group.2$Friends,pch='.',
    main="Facebook 2009",xlab="Friends",ylab="Wall
    Posts")
```

Multiple Plots in one Plotting Window

Calling `par(mfrow(c(m,n))` will produce a display containing m rows and n columns, and plots will appear row-wise, from left to right. (`mfcol` is similar except plots appear column-wise, from top to bottom, and then right.)

```
par(mfrow=c(1,2))
plot(group.1$Wall.Posts~group.1$Friends,pch='.',
    main="Facebook 2007",xlab="Friends",ylab="Wall
    Posts")
plot(group.2$Wall.Posts~group.2$Friends,pch='.',
    main="Facebook 2009",xlab="Friends",ylab="Wall
    Posts")
```

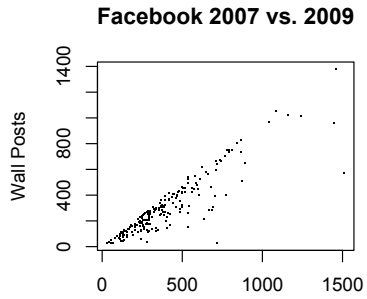
Multiple Plots in one Plotting Window



Two Plots for the Price of One

We can also plot both datasets on the same plot. First, construct a `plot` for the first graphic.

```
plot(group.1$Wall.Posts~group.1$Friends,pch='.',
    main="Facebook 2007 vs. 2009",xlab="Friends",
    ylab="Wall Posts")
```



Two Plots for the Price of One

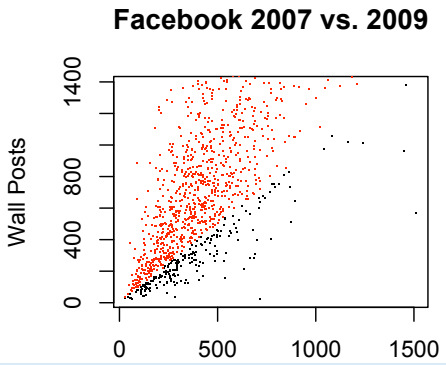
Recall that once a `plot` is constructed, the plot will be replaced if we construct another one. Instead, we need to *overlay* another plot on top of this one. To add more data onto this plot, use the `points` function.

```
points(x, y = NULL, type = "p", ...)
```

x and y are vectors containing the x and y coordinates of the values to overlay.

Two Plots for the Price of One

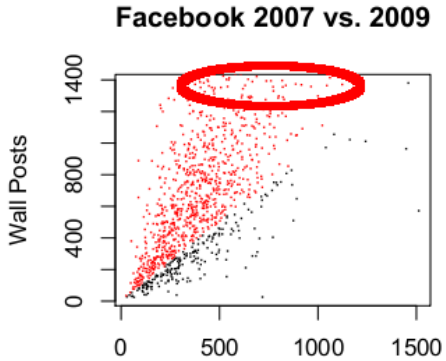
```
1 plot(group.1$Wall.Posts~group.1$Friends,pch='.',main="
    Facebook 2007 vs. 2009",xlab="Friends",ylab="Wall
    Posts")
2 points(group.2$Wall.Posts~group.2$Friends,pch='.',col="
    red")
```



Two Plots for the Price of One

But, there's a problem...

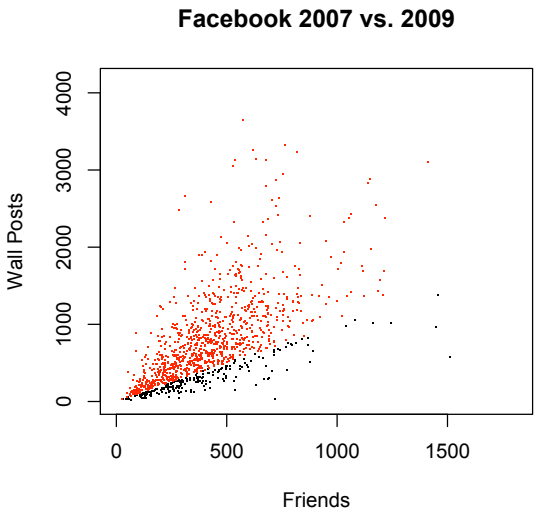
Data points are truncated on the plot, because the new points were laid on top of the *existing* coordinate system...



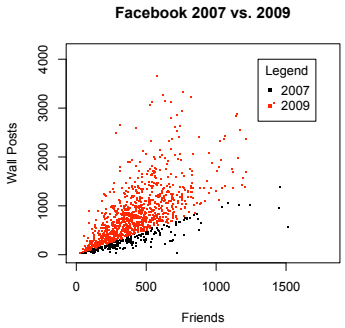
Two Plots for the Price of One

We can fix this problem by expanding the axes using `xlim` and/or `ylim`, using something like the following:

```
1 plot(group.1$Wall.Posts~group.1$Friends,pch='.',
    main="Facebook 2007 vs. 2009",xlab="Friends",
    ylab="Wall Posts",xlim=c(0,max(group.2$Friends)
    +400),ylim=c(0,max(group.2$Wall.Posts)+500))
2 #Set the coordinate system w/r/t the dataset that
    exceeds the bound.
3 points(group.2$Wall.Posts~group.2$Friends,pch='.',
    col="red")
```



We can also add a legend to the plot. The first two parameters are the x and y locations of the legend *with respect to the data*. The third parameter is a vector containing the text labels in the legend. `col` indicates the colors of the items in the key, `pch` indicates the character to use for the key and `pt.cex` blows up the points by a factor of 5, so they are visible. `inset` pushes the legend box a bit further into the plot.




```
1 legend(1300, 4000, c(2007,2009),
      col=c("black","red"),
      pch=c('.','.'), inset=1,
      title="Legend", pt.cex=5)
```

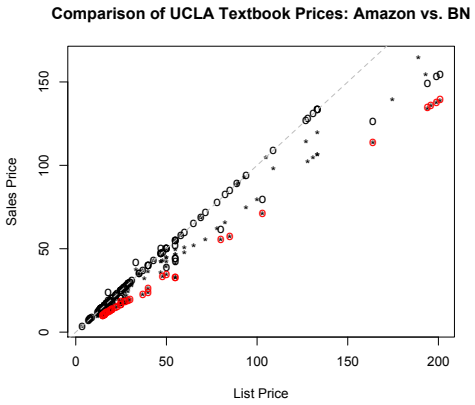

Exercise 2

Load in the UCLA textbook price comparison data from `http://www.stat.ucla.edu/~rosario/scc/textbooks.csv` It is a CSV file with a header. Plot the Amazon list price vs. the Amazon sales price and Amazon list price vs. Barnes & Noble price on the same plot. Use different plotting symbols for Amazon and Barnes and Noble. Add a grey dashed line (Hint: `?abline`) representing the location on the plot where the sales price of a book is equal to the list price. Highlight in red those Amazon books that differ from the Amazon list price by more than 25%. Label the plot and the axes.

Solution Exercise 2

```
1 plot(Amazon.Sale.Price~Amazon.List.Price,pch='*',
      main="Comparison of UCLA Textbook Prices:
      Amazon vs. BN",xlab="List Price",ylab="Sales
      Price")
2 points(Barnes.Noble.Price~Amazon.List.Price,pch='o'
      )
3 abline(a=0,b=1,lty=2,col="grey")
4 good.deals <- which(abs((Amazon.List.Price-Amazon.
      Sale.Price)/Amazon.List.Price) >= 0.25)
5 points(Amazon.Sale.Price[good.deals]~Amazon.List.
      Price[good.deals],pch='o',col="green")
```

Solution Exercise 2



Integrals

In this section we will take a look at using **math typesetting** in graphics as well as constructing a movie displaying changes over time graphically.

Task: Consider the Riemann integral, or definite integral, of the function $f(\theta) = \cos^3\theta d\theta$ can be defined as

$$\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos^3\theta d\theta$$

That is, we can fill attempt to fill the area under the curve with a bunch of rectangles of some width. As the width of these little rectangles goes to zero, we have can fill the area under the curve up to the curve.

The curve Function

First, we need to plot the curve $\cos^3\theta d\theta$. To do this, we use the `curve` function. Let's exclude all plotting options for now.

```
f <- function(x) { return(cos(x)^3) }
curve(f,from=-pi,to=pi,n=10000,lwd=4,col="green",
    xaxt="n",yaxt="n",main="",xlab="",ylab="")
```

curve and plot

`curve` acts like `plot`. It generates a **new** plotting window. To overlay a `curve` on an existing `plot`, we need to add the parameter `add=TRUE` to the call to `curve`.

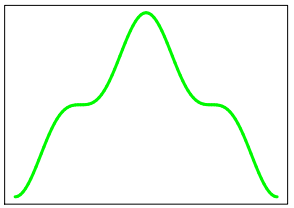
The curve Function

curve has a few different options than plot.

```
curve(expr, from = NULL, to = NULL, n = 101, add = FALSE,
      type = "l", ylab = NULL, log = NULL, xlim = NULL, ...)
```

- expr is an expression in terms of x, OR, a function f
- from is a the minimum value of x to be plotted.
- to is the maximum value of x to be plotted.
- n is the number of data points to plot, defaults to 101.
- the other options have been covered.

What we Need to Do...



1. add a title
2. add axes
3. add axis labels
4. overlay bounds for the definite integral, $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.

Math Typesetting in Graphics: Some Functions

Our title will be:

Computing the Integral $\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos^3\theta d\theta$

Enclose *anything* that may contain math text in the expression function. To concatenate text with a math object, use the paste function.

Math Typesetting in Graphics: Some Functions

The symbol function takes its parameter and prints it as a symbol (i.e. symbol(theta) displays as θ).

The plain function takes its parameter and prints it as standard text.

Some valid expressions are on the next slide, and can be generated using demo(plotmath).

Math Typesetting in Graphics: Options

Arithmetic Operators	
x + y	x + y
x - y	x − y
x * y	xy
x/y	x/y
x %+% y	x ± y
x%%y	x + y
x %*% y	x × y
x %.% y	x · y
-x	−x
+x	+x
Sub/Superscripts	
x[i]	x_i
x^2	x^2
Juxtaposition	
x * y	xy
paste(x, y, z)	xyz
Lists	
list(x, y, z)	x, y, z

Radicals	
sqrt(x)	$\sqrt{x}$
sqrt(x, y)	$\sqrt[y]{x}$
Relations	
x == y	x = y
x != y	x ≠ y
x < y	x < y
x <= y	x ≤ y
x > y	x > y
x >= y	x ≥ y
x %~~% y	x ≈ y
x %=~% y	x ≅ y
x %==% y	x ≡ y
x %prop% y	x ∝ y
Typeface	
plain(x)	x
italic(x)	*x*
bold(x)	**x**
bolditalic(x)	***x***
underline(x)	$\underline{x}$

(a)

Ellipsis	
list(x[1], ..., x[n])	$x_1, \ldots, x_n$
x[1] + ... + x[n]	$x_1 + \cdots + x_n$
list(x[1], cdots, x[n])	$x_1, \cdots, x_n$
x[1] + ldots + x[n]	$x_1 + \ldots + x_n$
Set Relations	
x %subset% y	$x \subset y$
x %subseteq% y	$x \subseteq y$
x %supset% y	$x \supset y$
x %supseteq% y	$x \supseteq y$
x %notsubset% y	$x \not\subset y$
x %in% y	$x \in y$
x %notin% y	$x \notin y$
Accents	
hat(x)	$\hat{x}$
tilde(x)	$\tilde{x}$
ring(x)	$\mathring{x}$
bar(xy)	$\overline{xy}$
widehat(xy)	$\widehat{xy}$
widetilde(xy)	$\widetilde{xy}$

Arrows	
x %<->% y	$x \leftrightarrow y$
x %->% y	$x \rightarrow y$
x %<-% y	$x \leftarrow y$
x %up% y	$x \uparrow y$
x %down% y	$x \downarrow y$
x %<=>% y	$x \Leftrightarrow y$
x %=>% y	$x \Rightarrow y$
x %<=% y	$x \Leftarrow y$
x %dblup% y	$x \Uparrow y$
x %dbldown% y	$x \Downarrow y$
Symbolic Names	
Alpha - Omega	$A - \Omega$
alpha - omega	$\alpha - \omega$
phi1 + sigma1	$\varphi + \varsigma$
Upsilon1	Υ
infinity	∞
32 * degree	32°
60 * minute	60′
30 * second	30″

(b)

Math Typesetting in Graphics: Options

Style
displaystyle(x)	x
textstyle(x)	x
scriptstyle(x)	x
scriptscriptstyle(x)	x
Spacing
x ~ ~y	x y

x + phantom(0) + y	x + + y
x + over(1, phantom(0))	$x + \frac{1}{\ }$
Fractions
frac(x, y)	$\frac{x}{y}$
over(x, y)	$\frac{x}{y}$
atop(x, y)	$\begin{matrix} x \\ y \end{matrix}$

(c)

Big Operators	
sum(x[i], i = 1, n)	$\sum_{i}^{n} x_i$
prod(plain(P)(X == x), x)	$\prod_{x} P(X = x)$
integral(f(x) * dx, a, b)	$\int_{a}^{b} f(x)dx$
union(A[i], i == 1, n)	$\bigcup_{i=1}^{n} A_i$
intersect(A[i], i == 1, n)	$\bigcap_{i=1}^{n} A_i$
lim(f(x), x %->% 0)	$\lim_{x \to 0} f(x)$
min(g(x), x >= 0)	$\min_{x \geq 0} g(x)$
inf(S)	inf S
sup(S)	sup S

(d)

Math Typesetting in Graphics: Options

Grouping	
(x + y) * z	(x + y)z
x^y + z	$x^{y} + z$
x^(y + z)	$x^{(y+z)}$
x^{y + z}	x^{y+z}
group("(", list(a, b), "]")	(a, b]
bgroup("(", atop(x, y), "]")	$\binom{x}{y}$
group(lceil, x, rceil)	⌈x⌉
group(lfloor, x, rfloor)	⌊x⌋
group("|", x, "|")	|x|

(e)

Using the tables from the previous slides, we can produce the plot title, using the `main` parameter. Here I store the syntax for the title in a variable called `my.title`.

```
my.title <- expression(paste("Computing the
  Integral ",integral(plain(cos)^3*symbol(theta)
  *d*symbol(theta), -symbol(pi)/2, symbol(pi)/2))
  )
```

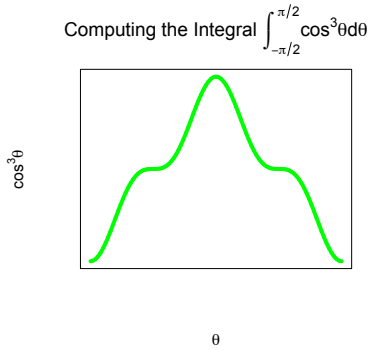
Producing axis labels is easier. I store axis labels in the variables `x.label` and `y.label`.

```
x.label <- expression(symbol(theta))
y.label <- expression(plain(cos)^3*symbol(theta))
```

Now I can pass the variables `my.title`, `x.label`, `y.label` as the values for the `main`, `xlab` and `ylab` parameters respectively.

Now we have

```
curve(f,from=-pi,to=pi,n=10000,lwd=4,col="green",
    xaxt="n",yaxt="n",main=my.title,xlab=x.label,
    ylab=y.label)
```



Now, we need to think about adding back the axes, but let's use the common values of θ for trigonometric functions. For this function, the y axis does not have any special values (but if you're ambitious, you can add the more complicated values to the y axis).

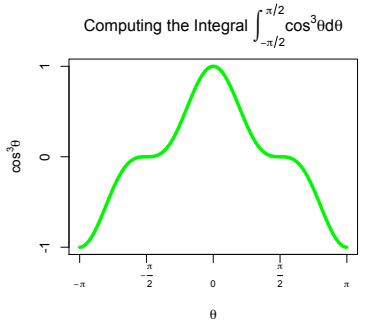
at value	-pi	-pi/2	0	pi/2	pi
labels value	$-\pi$	$-\frac{\pi}{2}$	0	$\frac{\pi}{2}$	π

Let's store the ticks we want to use for the x and y axes:

```
x.tick.locations <- seq(-pi,pi,pi/2)
x.tick.labels <- c(expression(-pi),expression(-frac
    (symbol(pi),2)),0,expression(frac(symbol(pi),2)
    ),expression(symbol(pi)))
#y axis is even easier.
y.tick.locations <- c(-1,0,1)
y.tick.labels <- y.tick.locations
```


Now, we can add back the axes:

```
axis(1,at=x.tick.locations,labels=x.tick.labels,
    cex.axis=0.5)
axis(2,at=y.tick.locations,labels=y.tick.labels,
    cex.axis=0.5)
```



Using cex and Friends

If the tick labels are too large for your liking, you can shrink them using the `cex.axis` parameter. The value of this parameter is the percentage of the current object size. To shrink, set `cex.axis` less than 1, and to enlarge, set greater than 1.

Other Variants of cex

`cex` controls all text and symbols. `cex.lab` controls axis labels (`xlab` and `ylab`), `cex.main` controls the size of the title of the plot, and `cex.sub` controls the size of the subtitle of the plot.

The segments Function

Using `segments`, we can draw a line segments from a point (x_0, y_0) to another point (x_1, y_1).

`segments(x0, y0, x1, y1,...)`

I added dashed lines in grey for to denote the limits of integration.

```
segments(-pi/2,-1,-pi/2,0,lty=2,col="grey")
segments(pi/2,-1,pi/2,0,lty=2,col="grey")
```

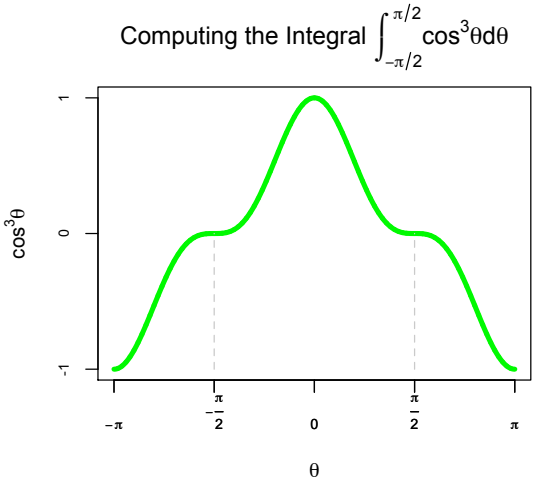
`lty` controls the line type. Note that `segments` is similar to `lines`.

Line Types: lty

The `lty` parameter is a number or string representing the type of line to draw.

- 0, "blank"
- 1, "solid"
- 2, "dashed"
- 3, "dotted"
- 4, "dotdash"
- 5, "longdash"
- 6, "twodash"

The Final Integral Graphic



The Final Integral Graphic

```
curve(f,from=-pi,to=pi,n=10000,lwd=4,col="green",
    xaxt="n",yaxt="n",main=my.title,xlab=x.label,
    ylab=y.label)
axis(1,at=x.tick.locations,labels=x.tick.labels,
    cex.axis=0.5)
axis(2,at=y.tick.locations,labels=y.tick.labels,
    cex.axis=0.5)
segments(-pi/2,-1,-pi/2,0,lty=2,col="grey")
segments(pi/2,-1,pi/2,0,lty=2,col="grey")
```

Integration: The Movie

First, wrap the mega-code for my plot into a function called `my.plot`. It takes a parameter `i` representing the current iteration.

```
my.plot <- function(i) {
curve(f,from=-pi,to=pi,n=10000,lwd=4,col="green",xaxt="n",yaxt="n",main=my.title,xlab=x.label,ylab=y.label)
axis(1,at=x.tick.locations,labels=x.tick.labels,  cex.axis=0.5)
axis(2,at=y.tick.locations,labels=y.tick.labels, cex.axis=0.5)
segments(-pi/2,-1,-pi/2,0,lty=2,col="grey")
segments(pi/2,-1,pi/2,0,lty=2,col="grey")
e <- (pi/4)*(1/(2**i))
rect(-pi/2+e*seq(0,(2**(i+2))-1),f(-pi),-pi/2+e*seq(1,(2**(i+2))),f(-pi/2+e*seq(1,(2**(i+2)))),col="black")
}
```

Integration: The Movie

Next, I will call this function in a loop...

```
setwd(tempdir()) #set working dir to a temp directory.
for (i in 1:5) {
  filename <- paste("plot",i,".jpg",sep="")
  #Create a JPEG with name "filename"
  jpeg(file=filename)
  #make a "frame" (one plot)
  my.plot(i)
  #turn the device off
  dev.off()
}
```

Instead of `jpeg`, you can also use `png`, `tiff`, `gif` etc. They all have similar options. `pdf` is a bit different...

Producing Graphics on Disk rather than Screen

In the previous slide we construct JPEGs for each frame in the movie using the `jpeg` function. We can also print a graphic to disk rather than to the screen. This is commonly done to create PDFs.

```
pdf(file="mypdf.pdf",height=7,width=7,onefile=TRUE)
#height and width specify size of graphic, in inches.
#onefile=TRUE - all plots will be in same file instead of multiple files.
#execute your plotting commands here...
plot(1:10,1:10)
#turn the plotting "device" off.
dev.off()
#like closing a file in a programming language.
```

Then, to stitch together the plot frames into a movie, I use the following code from the `rgl` library help.

```
make.mov <- function(){
      unlink("plot.mpg")
      system("convert -delay 0.25 plot*.jpg plot.mpg")
}
```

This function deletes (`unlink`) file `plot.mpg` if it exists. Then `system` executes the string passed to it, as if it were typed at the command line.

Caveat

This is only known to work on Unix, Linux and MacOS X systems containing the ImageMagick package as well as the ffmpeg package.

It's Show Time!

We can call `make.mov` to create the movie on disk. Then, open it using the OS.

```
make.mov()
```

Alternatives for Movies

My solution is rather primitive. There are packages that can produce animations or movies in R, that may be cross-platform.

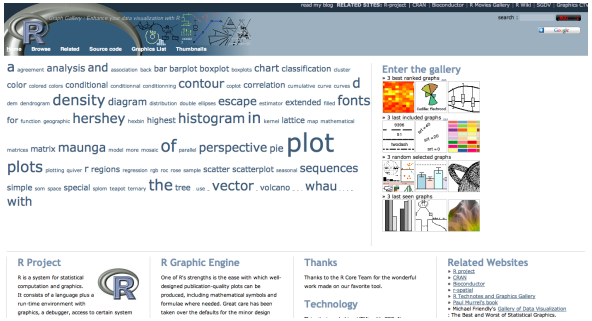
1. `write.gif` in package `caTools`
2. `animation` package on CRAN.
3. `EBImage` pacakge in BioConductor

For more information, check out a related question on **StackOverflow.com**:

```
http://stackoverflow.com/questions/
    1298100/creating-a-movie-from-a-series-of-plots-in-r
```

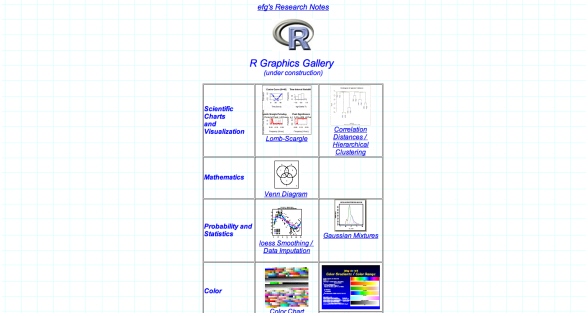
Other Resources for R Graphics

R Graph Gallery
`http://addictedtor.free.fr/graphiques/`



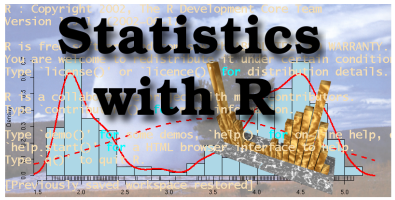
Other Resources for R Graphics

R Graph*ics* Gallery
`http://research.stowers-institute.org/efg/R/`



Other Resources for R Graphics

Statistics with R
`http://zoonek2.free.fr/UNIX/48_R/all.html`



Warning
Here are the notes I took while discovering and using the statistical environment R. However, I do not claim any competence in the domains I tackle: I hope you will find those notes useful, but keep you eyes open – errors and bad advice are still lurking in those pages...

Should you want it, I have prepared a quick-and-dirty PDF version of this document.

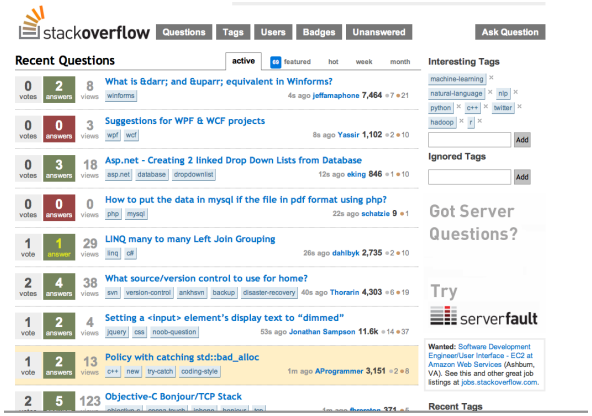
The old, French version is still available, in HTML or as a single file.

You may also want all the code in this document.

1. Introduction to R
2. Programming in R
3. From Data to Graphics
4. Customizing graphics
5. Factorial methods: Around Principal Component Analysis (PCA)
6. Clustering
7. Probability Distributions
8. Estimators and Statistical Tests
9. Regression

Other Resources for R Graphics

StackOverflow
`http://www.stackoverflow.com`



Thank you for your attention!