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...

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

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

![The R Graphics Learning Curve](image)

The R Graphics Learning Curve

Time

Effort

0 T/3 2T/3 T

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

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

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.

```r
plot(text[5], text[7], main="Bivariate Analysis of 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.

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

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

```r
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.

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

$names

[1] "breaks"  "counts"  "intensities"
"density"  "mids"  "xname"  "equidist"

$class

[1] "histogram"

Histories: Not Just a Plot!

Extracting Information from hist

```r
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.

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

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

Manipulating Axes

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.

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

An Aside: Extracting Information from hist

Let’s check our progress...
Manipulating Axes

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.

Intro

Intro Customization

Plot Windows

Math/Movies

Conclusion

Manipulating Axis

Some More Tinkering: Plot Types

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

```
# Learning Curve graphic
y <- hist(my.gama, br=100, plot=FALSE)$counts
x <- seq(0, max(my.gama), length=length(y))
plot(y~x, 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")
```
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.

![The R Graphics Learning Curve](image)

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/](http://research.stowers-institute.org/efg/R/Color/Chart/). You can create your own index by using the following command:

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

Color - Palette Codes

![R colors — Sorted by Hue, Saturation, Value](image)

Color - Sample Index

![Color - Sample Index](image)

---

1 Image from Earl F. Glynn, Stowers Institute for Medical Research [http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf]
After some last few touches using `abline` covered in the previous minicourse, we get the following code:

```r
# 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")
```

---

**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.

```r
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", xaxt="n", xlab="z", ylab="", main="My Normal Distribution")
# With palette code
plot(y~x, type="l", lwd=4, col=139, xaxt="n", xlab="z", ylab="", main="My Normal Distribution")
# Or with RGB color content
plot(y~x, type="l", lwd=4, col="#228B22", 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.

```r
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,

```r
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.

```r
2  group.2 <- read.csv("http://www.stat.ucla.edu/~rosario/scc/facebook-2009.csv", header=TRUE)
```
Using Multiple Plotting Windows

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

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

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.

1. `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.

\[
\text{points}(x, y = \text{NULL}, \text{type} = "p", \ldots)
\]

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

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...

```
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")
```

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")
```
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

```r
points(Barnes.Noble.Price ~ Amazon.List.Price, pch='o')
abline(a=0,b=1,lty=2,col="grey")
good.deals <- which(abs((Amazon.List.Price-Amazon.Sale.Price)/Amazon.List.Price) >= 0.25)
points(Amazon.Sale.Price[good.deals] ~ Amazon.List.Price[good.deals], pch='o', col="green")
```
Solution Exercise 2

![Comparison of UCLA Textbook Prices: Amazon vs. BN](image)

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

\[
\int_{-\pi}^{\pi} \cos^3 \theta d\theta
\]

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.

```r
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="")
```

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.

```r
\textbf{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.

\[
\text{curve}(\text{expr}, \text{from} = \text{NULL}, \text{to} = \text{NULL}, n = 101, \text{add} = \text{FALSE}, \text{type} = \text{"l"}, \text{ylab} = \text{NULL}, \text{log} = \text{NULL}, \text{xlim} = \text{NULL}, \ldots)
\]

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

Math Typesetting in Graphics: Some Functions

Our title will be:

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

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

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 \(\frac{\pi}{2}\).
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`.

```r
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`.

```r
1 x.label <- expression(symbol(theta))
2 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

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

Computing the Integral \( \int_{-\pi/2}^{\pi/2} \cos^3(\theta) \, d\theta \)

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

1. \texttt{x.tick.locations \leftarrow \texttt{seq(-pi,pi/pi,pi/2)}}
2. \texttt{x.tick.labels \leftarrow \texttt{c(expression(-pi),expression(-frac(symbol(pi),2)),0,expression(frac(symbol(pi),2)),expression(symbol(pi))}}
3. \texttt{y.tick.locations \leftarrow \texttt{c(-1,0,1)}}
4. \texttt{y.tick.labels \leftarrow \texttt{y.tick.locations}}

Now, we need to think about adding back the axes, but let's use

\begin{array}{c|c|c|c|c|c}
\text{at value} & -\pi & -\pi/2 & 0 & \pi/2 & \pi \\
\text{labels value} & -\pi & -\pi/2 & 0 & \pi/2 & \pi \\
\end{array}

Using the \texttt{axis} function:

1. \texttt{axis(1,at=x.tick.locations,labels=x.tick.labels,}
\texttt{cex.axis=0.5)}
2. \texttt{axis(2,at=y.tick.locations,labels=y.tick.labels,}
\texttt{cex.axis=0.5)}

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

Other Variants of \texttt{cex}

c\texttt{ex} controls all text and symbols. \texttt{cex.lab} controls axis labels (xlab and ylab), \texttt{cex.main} controls the size of the title of the plot, and \texttt{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 \((x_1, y_1)\).

\[
\text{segments}(x0, y0, x1, y1,\ldots)
\]

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

\[
\text{segments}(\text{-pi}/2,-1,-\text{pi}/2,0,\text{lty}=2,\text{col}="\text{grey}"
\]
\[
\text{segments}(\text{pi}/2,-1,\text{pi}/2,0,\text{lty}=2,\text{col}="\text{grey}"
\]

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

The Final Integral Graphic

Computing the Integral \[\int_{-\pi/2}^{\pi/2} \cos^3(\theta) \, d\theta\]

\[
\begin{align*}
\text{curve(f, from=-\text{pi}, to=\text{pi}, n=10000, lwd=4, col="\text{green}"}, \\
xaxt="n", yaxt="n", main=my.\text{title}, xlab=x.\text{label}, \\
ylab=y.\text{label})
\end{align*}
\]

\[
\begin{align*}
\text{axis}(1, at=x.\text{tick}.\text{locations}, labels=x.\text{tick}.\text{labels}, \\
cex.\text{axis}=0.5)
\end{align*}
\]

\[
\begin{align*}
\text{axis}(2, at=y.\text{tick}.\text{locations}, labels=y.\text{tick}.\text{labels}, \\
cex.\text{axis}=0.5)
\end{align*}
\]

\[
\begin{align*}
\text{segments}(\text{-pi}/2,-1,-\text{pi}/2,0,\text{lty}=2,\text{col}="\text{grey}"
\end{align*}
\]

\[
\begin{align*}
\text{segments}(\text{pi}/2,-1,\text{pi}/2,0,\text{lty}=2,\text{col}="\text{grey}"
\end{align*}
\]
First, wrap the mega-code for my plot into a function called `my.plot`. It takes a parameter `i` representing the current iteration.

```r
my.plot <- function(i) {
  curve(f, from=-pi, to=pi, n=1000, 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*pi/2+e*seq(0,(2*pi/2+e*seq(1,(2*pi/2+e*seq(1,(2*(i+2)))))))))
  col="black")
}
```

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

```r
setwd(tempdir())
for (i in 1:5) {
  filename <- paste("plot", i, "jpg", sep="")
  jpeg(file = filename)
  my.plot(i)
  dev.off()
}
```

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

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

```r
make.mov <- function() {
  unlink("plot.mp4")
  system("convert -delay 0.25 plot*.jpg plot.mp4")
}
```

This function deletes (`unlink`) file `plot.mp4` 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.

- `write.gif` in package `caTools`
- `animation` package on CRAN.
- `EBImage` package in BioConductor

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

Other Resources for R Graphics

R Graphics Gallery
http://research.stowers-institute.org/efg/R/

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

Other Resources for R Graphics

StackOverflow
http://www.stackoverflow.com

Thank you for your attention!