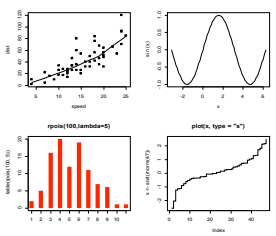
Graphics

4

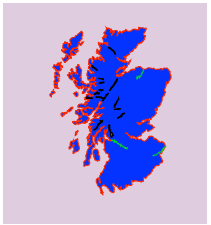
Starting out with R graphics
Graphics

- R provides several mechanisms for producing graphical output
  - Functionality depends on the level at which the user seeks interaction with R
    - graphics systems, packages, devices & engines
- High level graphics
  - Functions compute an appropriate chart based up on the information provided. Optional arguments may tailor the chart as required
    - Interaction is at traditional graphics system level. The user isn't required to know much about anything
- Low level graphics
  - The user interacts with the drawing device to build up a picture of the chart piece by piece.
    - This fine granular control is only required if you seek to do something exceptional
- R graphics produces plots using a painter's model
  - Elements of the graph are added to the canvas one layer at a time, and the picture built up in levels. Lower levels are obscured by higher levels, allowing for blending, masking and overlaying of objects.

High level vs. Low level plotting
Graphics



High level plotting
`example(plot)`

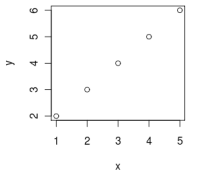


Low level plotting
(Scotland by blighty package)

Essential plotting - plot()

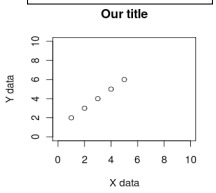
• plot() is the main function for plotting, it takes x,y values to plot and also lots of graphical parameters (see **?par** for all of them)

default plotting




```
x <- 1:5
y <- 2:6
plot(x,y)
```

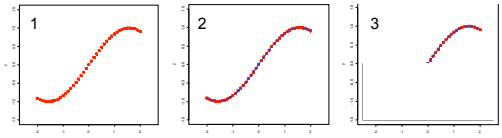
custom plotting




```
x <- 1:5
y <- 2:6
plot(x,y, xlab="X data", ylab="Y
data", xlim=c(0,10), ylim=c(0,10),
main="Our title")
```

R graphics uses a painter's model

```
x <- seq(-2, 2, 0.1)
y <- sin(x)
```




```
plot(y~x, ylim=c(-1.5,1.5),
xlim=c(-2.5,2.5),
col="red" ,pch=16, cex=1.4)
```

```
lines(y~x, ylim=c(-1.5,1.5),
xlim=c(-2.5,2.5), col="blue",
lty=1,lwd=2)
```

```
rect(-2.5,0,2.5,-1.5,
col="white", border="white")
```

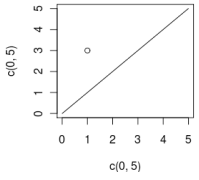
xlim, ylim = axis limits
col = line colour
pch = plotting character [`example(points)`]
cex = character expansion [scaling factor]

lty = line type
lwd = line width
rect = rectangle

Example code:
14_painterModel.R

Plotting x,y data - plot(), points(), lines()

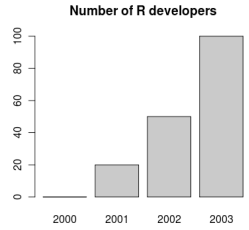
• **plot()** is used to start a new plot, accepts x,y data, but also data from some objects (like linear regression). Use the parameter **type** to draw points, lines, etc (see **?plot**)
• **points()** is used to add points to an existing plot
• **lines()** is used to add lines to an existing plot




```
plot(c(0, 5), c(0, 5), type="l") # draw as line from (0,0) to (5,5)
points(1, 3) # add a point at 1,3
```

Making bar plots - barplot()

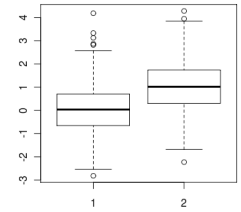
• visualizing a vector of data can be done with bar plots, using function **barplot()**




```
data <- c("2000"=0, "2001"=20, "2002"=50, "2003"=100)
barplot(data, main="Number of R developers")
```

Making box plots - boxplot()

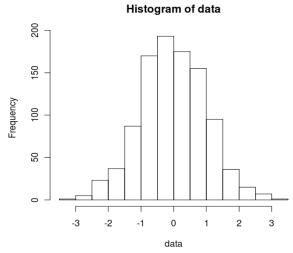
• when a spread of data needs to be visualised, we can use boxplots with function **boxplot()**




```
data1 <- rnorm(1000, mean=0)
data2 <- rnorm(1000, mean=1)
boxplot(data1, data2)
```

Making histograms - hist()

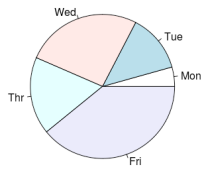
• when we need to look at the distribution of data, we can visualize it using histograms with function hist()




```
data <- rnorm(1000)
hist(data)
```

Pie charts - pie()

• to visualise percentages or parts of a whole we can use pie charts with function **pie()**




```
data <- c("Mon"=1, "Tue"=3, "Wed"=6, "Thr"=4, "Fri"=9)
pie(data)
```

Typical plotting workflow

- Set the plot layout and style - `par()`
  - Set the number of plots you want per page
  - Set the outer margins of the figure region
    - The distance between the edge of the page and the figure region, or between adjacent plots if there are multiple figures per page
  - Set the inner margins of the plot
    - The distance between the plot axes and the labels & titles
  - Set the styles for the plot
    - Colours, fonts, line styles and weights
- Draw the plot - `plot(x,y, …)`

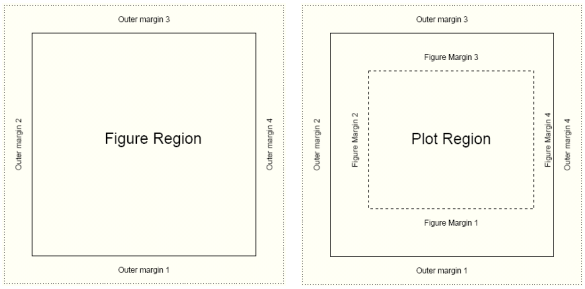
Setting graphics layout and style - par()

`par()` Top level graphics function

- parameter specifies various page settings. These are inherited by subordinate functions, if no other styles are set.
  - Specific colours and styles may be set globally with `par`, but changed ad hoc in plotting commands
  - The global setting will remain unchanged, and reused in future plotting calls.
- `par` sets the size of page and figure margins
  - Margin spacing is in 'lines'
- `par` is responsible for controlling the number of figures that are plotted on a page
- `par` may set global colouring of axes, text, background, foreground, line styles (solid/dashed), if figures should be boxed or open etc. etc.

type `par()` to get a list of top down settings which may be set globally

Page settings with `par`
Graphics



`par(mfrow=c(1,1))`
one figure on page
`par(oma=c(2,2,2,2))`
equal outer margins

`par(mar=c(5,4,4,2))`
Sets space for x & y labels, a main title, and a thin margin on the right

Numbering: bottom, left, top, right

Page layout plot exercise
Graphics

`par(mfrow=c(2,2))`
- 2 x 2 figures per page

`par(oma=c(1,0,1,0))`
- 1 line spacing top and bottom

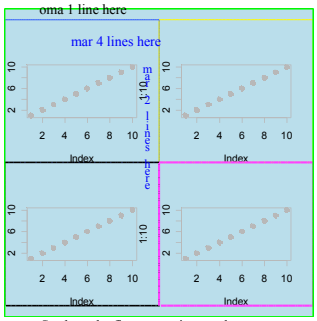
`par(mar=c(4,2,4,2))`
- 4 lines at bottom & top
- 2 lines left & right

`par(bg="lightblue",fg="darkgrey")`
- light blue background
- dark grey spots

`par(pch=16,cex=1.4)`
- Large circles for spots
- Execute 4 times with different colors:

```
plot(1:10)
box("figure",lty=3,col="blue")
```
- Draw a blue dashed line around plot

`box("outer",lty=1,lwd=3, col="green")`
- Draw a green solid line around figure



See how the figure margins overlap
Using painter's model

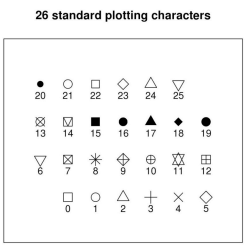
15_parExample.R

Plotting characters for plot()
size and orientation

`pch= ...`
Sets one of the 26 standard plotting character used.
Can also use characters, such as "."

`cex= ...`
Character expansion. Sets the scaling factor of the printing character

`las= ...`
Axes label style. 1 normal, 2 rotated 90º
4 styles (0-3)



Plotting characters exercise
Graphics

16_plottingChars.R

```
xCounter<-1
yCounter<-1
plotChar<-0

plot(NULL, xlim=c(0,8),
ylim=c(0,5),xaxt="n",
    yaxt="n",ylab="",xlab=""
    ,main="26 standard plotting
characters")

while (plotChar < 26){
  if(xCounter < 7){
    xCounter <- xCounter+1
  } else {
    xCounter <- 1
    yCounter <- yCounter+1
  }

  points(xCounter,yCounter,pch=plotChar,
cex=2)
  text(xCounter,(yCounter-0.3),plotChar)
  plotChar <- plotChar+1
}
```

X-Y coordinates,
Plotting character index counter

Sets up an empty plotting area.
Axis scale limits, xlim, ylim
Don't draw axis ticks, xaxt, yaxt="n"
Don't annotate axis, xlab, ylab=""
Set a main title, main

We want to print the characters in a 7 x 4 grid. The if statement sets up the character plotting coordinates such that each time x =7, make it 1 again and increment the y axis by 1 at the same time

While loop counts up to 25
(0 to 25 = 26 iterations)
And cycles through each pch available

Annotating the plot

- plot accepts main title, subtitle, X label, Y label as standard arguments

```
plot(x, y, main="...", sub="...", xlab="...", ylab="...")
```

`mtext(text="…", side= …)`
- allows text to be written directly into the margin of a plot

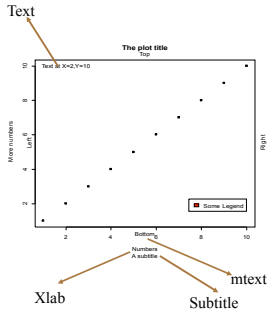
`text(x,y,labels="…")`
- allows text to be written in the plot at x,y

`legend(x,y, legend=…)`
- produces a legend for the plot

Appreciating drawing coordinates

- How do we know where to place items within the plot region when building up our customized graphs?
- Most of the time we can specify X,Y coordinates.
  - R calculates sensible pixel coordinates of plots from the data we provide. We don't need to worry about pixels, centimetre distances etc.
- `locator(…)`
  - Returns x,y coordinates from a mouse click within a plot
  - good for working out where to place legend items
- `identify(…)`
  - provides an id tag for the closest plotted point to a mouse click
  - useful if you want to label points on a chart
- `xy.coords(…)`
  - translates x,y coordinates into pixel coordinates
- Margin spacing is in lines
  - The exact distance is a factor of font family, style and size
    - Text may appear bunched or squashed if sufficient distance is not left between the axes and the caption

Building up a plot
Graphics



17_buildingAplot.R

R code

```
par(mfrow=c(1,1))
par(bg="white",fg="black",cex=1)
par(oma=c(1,1,1,1))
par(mar=c(5,4,4,2)+0.1)

plot(1:10,main="The plot title",
sub="A subtitle", xlab="Numbers",
        ylab="More numbers")

mtext(c("Bottom", "Left", "Top",
"Right"), c(1,2,3,4),     line=.5)

text(2,10,"Text at X=2,Y=10")
legend(locator(1),"Some
Legend",fill="red")
```

Adding legend …
Don't forget to mouse click!

align text left, right & centre with adj=(i,j) i.e centre is adj=(0.5,0.5), left is adj=(1,0) and right is adj=(0,1)

Plots with custom axes
Graphics

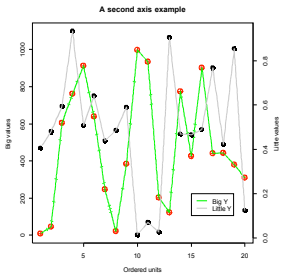
- R `plot` doesn't support multiple Y axis by default
  - You have to make additional axes yourself!
- Adding custom axis

```
axis(side=, at=, labels=, …)
```

- If you want to specify custom axes, make sure you turn off the automatic axes in the plot / points call

```
plot( …, axes=F)
```

Adding a second Y axis
Graphics



The trick

1. plot first Y series
2. use `par(new=T)` to overlay a second figure region
3. plot second series without axes
4. `axis(side=4, …)` to add second Y axis
5. `mtext(side=4, …)` to label second Y

Example: The second Y series
Graphics

18_secondYaxis.R

```
x1<-1:20
y1<-sample(1000,20)
y2<-runif(20)
y2axis<-seq(0,1,.2)
```
Demo data

```
par(mar=c(4,4,4,4))
```
Set up equivalent figure margins

```
plot(x1,y1,type="p",pch=10,cex=2,col="red",
     main="A second axis example",
     ylab="Big values",ylim=c(0,1100),
     xlab="Ordered units")
```
Plot and label first Y series

```
points(x1,y1,type="l",lty=3,lwd=2,col="green")
```
Connect dots with a line

```
par(new=T)
```
Overlay a second plot region

```
plot(x1,y2,type="p",pch=20,cex=2,col="black",axes=FALSE,bty="n",xlab="",ylab="")
points(x1,y2,type="l",lty=2,lwd=2,col="grey")
```
Plot second Y series, but suppress labels

```
axis(side=4,at=pretty(y2axis))
mtext("Little values",side=4,line=2.5)
```
Anotate second Y axis

```
legend(15,0.2,c("Big Y","Little Y"),lty=1,lwd=2,col=c("green","grey"))
```
Add legend, note X,Y is on second Y axis scale

Use of colour in R
Graphics

- Colour is usually expressed as a hexadecimal code of Red, Green, and Blue counterparts
  - No good for humans.
- R supports numerous colour palettes which are available through several "colour" functions.
  - `colours()` # get inbuilt names of known colours
    - RGB primaries may take on a decimal intensity value of 0 to 255
      - 255 is #FF in hexidecimal
        - White is #FF FF FF
        - Black is #00 00 00
  - `rgb()` # converts red green blue intensities to colour
    - Strangely, likes decimalized intensities (ie. 0 is black, 1 is white)

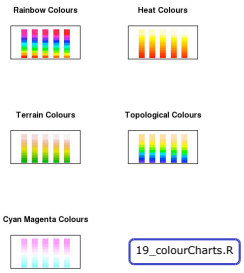
```
> rgb(1,1,1)
[1] "#FFFFFF"

> par(mfrow=c(2,2))
> plot(1:10,col="#FF00FF")
> plot(1:10,col=rgb(1,0,1))
> plot(1:10,col="magenta")
```

Colour Ramps & Palettes
Graphics

•Heatmaps use colour depth to convey data values. Cold colours are typically low values, and light colours are high state values. This is a colour ramp.

•R supports numerous graded colour charts. Specify *n*, to set the number of gradations required in the palette

```
rainbow(n)
heat.colors(n)
terrain.colors(n)
topo.colors(n)
cm.colors(n)
```



19_colourCharts.R

You can specify a user defined palette of indexed colours:

```
palette(rainbow(7)) # creates 7 indexed colours (1:7) based on
                    # rainbow palette  R O Y G B I V !!!
```

Colour packages: RColorBrewer
Graphics

- This add on package provides a series of well defined colour palettes. The colours in these palettes are selected to permit maximum visual discrimination
- Access the RColorBrewer library functions ...

```
library("RColorBrewer")
```

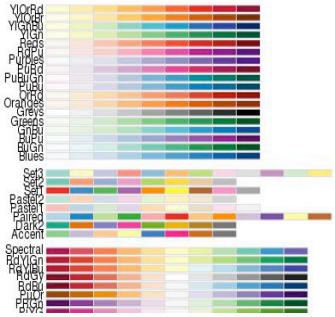
- Check out the available palettes

```
display.brewer.all(n=NULL, type="all", select=NULL, exact.n=TRUE)
```

- Define your own palette based on one of RColorBrewers'

```
myCol<-brewer.pal(n,"...") # n=number of colours, "..." is the palette name
```

RColorBrewer named palettes
Graphics



Saving plots to files

- Unless specified, R plots all graphics to the screen
- To send plots to a file, you need to set up an appropriate graphics device …

```
postscript(file="a_name.ps", …)
pdf(file="…pdf", …)
jpeg(file=" …jpg", …)
png(file=" ….png", …)
```

- Each graphics device will have a specific set of arguments that dictate characteristics of the outputted file
  - `height=, width=, horizontal=, res=, paper=`
    - Top tip: jpg, A4 @ 300 dpi, portrait, size in pixels
    - `jpg(file="my_Figure.jpg", height=3510, width=2490, res=300)`
    - Postscript & pdf work in inches by default, A4 = 8.3" x 11.7"
- Graphics devices need closing when printing is finished

```
dev.off()
```

for example:

```
png("tenPoints.png", width=300, height=300)
plot(1:10)
dev.off()
```

Thoughts when plotting to a file
Graphics

- Its very tempting to send all graphical output to a pdf file. Caution!
  - For high resolution publication quality images you need postscript. Set up postscript file capture with the following function

```
postscript("a_file.ps",paper="a4")
```

    - postscript images can be converted to JPEG using ghostscript (free to download) for low resolution lab book photos and talks
  - PDF images will grow too large for acrobat to render if plots contain many data points (e.g. Affymetrix MA plots)
  - Automatically send multiple page outputs to separate image files using …`file="somename%02d.jpg"`
  - Don't forget to close graphics devices (i.e. the file) by using
    - `dev.off()`

Plotting exercise
Graphics

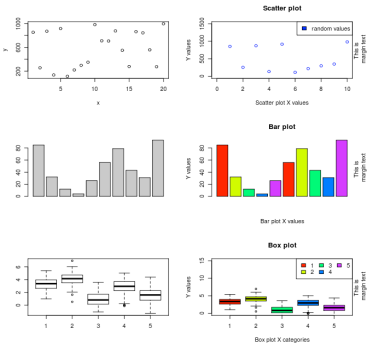
- Exercise:
  - Make a full A4 page figure comprising of 6 plots: 2 each of XY plot (`plot()`), barchart (`barplot()`) and box plots (`boxplot()`)
  - The two version of each plots should consistent of: the default plot and a customised plot (change for instance colours, range, captions...)
  - Output the completed 6-panel figure to: screen, jpeg, postscript and pdf file

- Suggested route to solution:

  1. Generate some plotting data appropriate for each type of plot
  2. Write the code to produce the six plots, once plotting the data by using default plotting, one with some customisations you want
  3. To output the plot to screen, jpeg, postscript and pdf you will need to redo the plot multiple times - create a function to do a plotting and call it by redirecting graphical output to screen, jpeg file, poscript file and pdf file

20_6PanelPlotScript.R

6 Panel plots exercise
Graphics



References

- Official documentation on:
  - http://cran.r-project.org/manuals.html
- A good repository of R recipes:
  - Quick-R: http://www.statmethods.net/
- Don't forget that many packages come with tutorials (vignettes)
- Website of this course:
  - http://logic.sysbiol.cam.ac.uk/teaching/Rcourse/
- R forums (stackoverflow & official):
  - http://stackoverflow.com/questions/tagged/r
  - http://news.gmane.org/gmane.comp.lang.r.general
- Plenty of textbooks to choose from, comprehensive list + reviews:
  - http://www.r-project.org/doc/bib/R-books.html