Statistics and Experimental Design using R

Sessions (Jan Kreft)

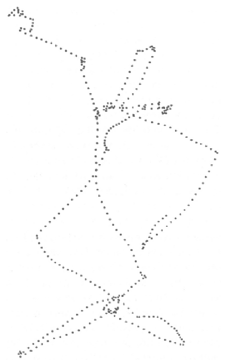
Resources

- Books
  - Michael J Crawley (2005). Statistics: an introduction using R. Wiley: Chichester, England [10 copies in Library]
    - Helmut Fritz van Emden (2008). Statistics for terrified biologists. Blackwell Publishing: Oxford [11 copies in Library]
    - Roland Ennos (2007). Statistical and data handling skills in biology. 2nd ed. Pearson Prentice Hall: Harlow, England [7 copies in Library]
    - Ruxton & Colegrave (2003). Experimental design for the Life Sciences, Oxford [20 copies in Library]
    - Fowler, Cohen & Jarvis (1998). Practical Statistics for Field Biology. 2nd. ed. Wiley [17 copies in Library]
- WebCT
  - www.weblearn.bham.ac.uk
  - Slides, data files, R scripts, Course info

Session 1-2: Fundamental statistical concepts and R primer

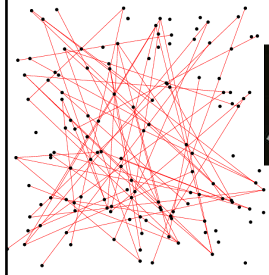
Fundamental statistical concepts

- Why statistics?
- Significance
- p-values
- Null-hypothesis
- Effect size
- Biological importance
- The power of random processes



Howard C Berg (1993) Random Walks in Biology. Princeton University Press

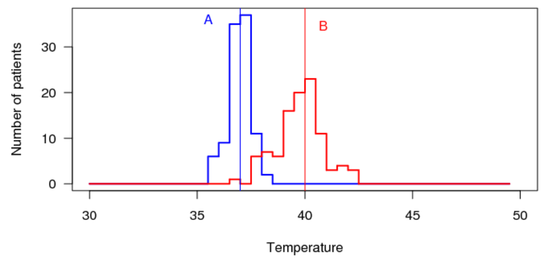
Random or not?



Why we need statistics

- Statistics is about distinguishing real from chance effects
- A typical example is the comparison of the means of two samples
  - Compare the effect of two different treatments A and B on the body temperature of patients with fever
    - Does drug A have an effect, i.e. reduce fever?
    - 'Treatment' B could be the control placebo treatment

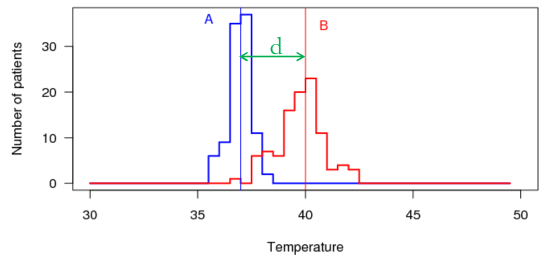
Why we need statistics



- It looks like drug A has an effect
- But randomly picking patients can also result in different samples even though there are no real differences

- Statistical analysis allows us to answer this question

Why we need statistics



- More precisely, statistical analysis gives a quantitative answer, it tells us how likely it is that we obtain this difference d between the means of samples A and B if they were from the same population
- In other words, how likely it is to obtain this difference d between the means of two samples if the null-hypothesis that the two treatments are equally effective were true

Exercise

- To illustrate this point, let's create a population with known average temperature
  - Assume temperature is normally distributed
  - Mean = 37
  - SD = 1
    - `pop <- rnorm(100, mean=37, sd=1)`
    - `mean(pop)`
- Now take some samples from this artificial population of human patients
    - `sampleA <- sample(pop, 5, replace=FALSE)`
    - `mean(sampleA)`

Definitions: statistical significance

- Statistical significance: An effect is called significant if it is unlikely to have arisen by chance
  - More precisely, a result is statistically significant if it is unlikely to have occurred by chance if the null hypothesis were true
  - By unlikely we usually mean a probability of less than 5%

Definitions: null-hypothesis

- Null-hypothesis: no effect, nothing is happening, the differences or patterns we observe are simply due to random fluctuations. For example:
  - If we compare two samples, the null hypothesis could be that the means of the two samples are the same
    - or that the variances are the same
    - or more generally that the two samples have the same distribution
  - If we compare two treatments, the null hypothesis is that there is no difference between the two treatments
  - Like any good hypothesis, it must be possible to falsify it

Definitions: p value

- A *p* value is an estimate of the probability that a given result (or any more extreme result) could have occurred by chance if the null hypothesis were true
- If the *p* value is sufficiently low, e.g. $p < 0.05$, then the
  - result is unlikely to have occurred by chance
  - result is significant
  - we reject the null hypothesis because it is unlikely to be true
- Statistics gives us a quantitative answer...
- ...but no guarantees, only an estimated probability that the null hypothesis is correct, the *p*-value

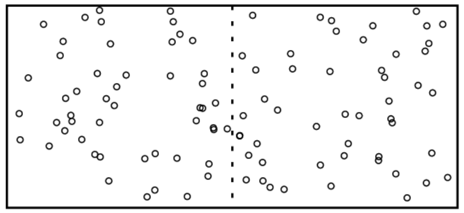
Statistical versus biological 'significance'

- A statistically significant difference can be insignificant
  - because a significant difference may be very small
    - this is about the effect size
  - because a significant difference may be of no (fitness) consequence
    - For example, the mean difference in height between men and women is 13 cm (UK population, age group 16-24)
    - This difference is statistically significant (as you can work out if you know sample size and variability)
    - And the effect is large (13 cm), but still...
    - ...is this difference biologically important?

Exercise

- Explain the meaning of statistical significance to your neighbour
- Explain what it means when someone says "Drug A had a significant effect ($p = 0.022$)" and what it does not mean

How random walks lead to directed change



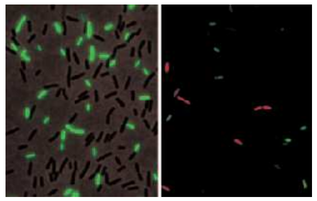
- The random walk of many particles on the microscopic scale leads to diffusion on the macroscale
  - Diffusion is directed, flux is down the concentration gradient

Appreciating the power of random processes

- Noise in gene regulation, say a repressor falling off by chance, can enable a cell to switch from downregulated to upregulated
  - Such random events will only happen in some of the cells, creating heterogeneity in the population
  - Need positive feedback (or double negative feedback) for such bistability

upregulated

downregulated



Stable state A

Stable state B

Exercise

- Think of biological processes or phenomena that are driven by chance and write your random thoughts down here now

R primer

What is R?

- R is a powerful software package for statistical analysis
- Originally written for teaching statistics
  - Basically an implementation of S, the standard statistical programming language (used in R and S-PLUS)
  - R comes before S
  - R was initially written by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland
- Has become a mainstream research tool
  - Users have contributed hundreds of add-on packages
  - It is very unlikely that you will encounter a statistical problem others haven't already solved in R
  - Large and active user community, so there are many to help you

What is R?

- And it's open-source, available for free for Win, Mac, Linux, etc.
- R is also a high-level programming language, particularly useful for numerical, technical, mathematical computing as it comes with a lot of matrix algebra functions
- Home page http://www.r-project.org/index.html
  - Download the latest release
  - Source code or (easier) a precompiled binary of the base system with contributed packages for a particular OS
  - E.g. for Windows, download R-2.11.1-win32.exe from a suitable mirror
  - Choose full installation to get help files in various formats (needs 65 MB disk space)
  - Otherwise you can accept the default settings

Some general advice

- A computer is stupid: it will do exactly what you tell it to do and you need to tell it every little step, in the right order
  - Imagine writing instructions that someone can execute without having to think
  - Like describing an experimental protocol in the methods section of a paper
  - Be perfect! (Or rather learn from your errors)

Some general advice

- Problem solving
  - You will run into problems when you are programming
  - Solve them by
    - reading the error message
    - reading the help file
    - make a list of what could possibly be wrong
      - check these possibilities one by one
    - break down a complex step into smaller, simpler steps
      - Divide and conquer!

Running R

- Once you started R (the R GUI) you will find the
  - Console (this is the command window)
  - Graphics device
    - will appear if you have created a plot
  - Script editor window
    - will appear when you create a new script or open an old one
    - here you can write your program line by line(a number of command lines to be executed in sequence)
    - it's just a text file, extension .R
  - Some terms and their meanings
    - Workspace: set of currently existing variables (objects)
      - Don't use "Save workspace..." to save your R session (commands and output), use "Save to file..."
    - History: list of previously executed commands, great for the lazy

R documentation and help

- `> help.start()`
  - Most useful if you don't know which command to use
  - Starts a Search Engine in your favourite browser
  - Can get this also from the R GUI Help Menu
- `> ?plot`
  - If you do know which command to use but need to check details
  - Brings up the help page on plot
- From the R GUI Help menu, you can get Manuals (in PDF)
  - An Introduction to R
  - R Reference manual

R documentation and help

- Further reading
  - Michael J Crawley (2005) Statistics: an introduction using R. Wiley: Chichester, England
    - Particularly chapter 2 and appendix and web page
    - http://www.imperial.ac.uk/bio/research/crawley/statistics/
    - Example data frames and further introduction to R
  - Other books
  - Various material on the R web site
  - FAQ
  - R-help mailing list

R basics: command line

Getting used to the command line (in red):

Prompt [Whitespace] Command [Whitespace] Return

```
>                    1+2
[1] 3
```

Results are output underneath (in blue)

The prompt tells you that R is waiting for you to enter a command, after you have entered the command, press return or enter to send it off

R basics: assignment

Assigning these results (output from command) to a variable

Variable name   Assignment operator (gets)   value

```
> x                <-             1
> X
Error: object "X" not found
```

R is case sensitive so x and X are different variables

Variable names should start with a letter followed by letters and numbers

You can use "." or "_" instead of whitespace – but do not use whitespace since this is used to detect the end of a token ('word')

salt_concn instead of salt concn

R as a calculator

Arithmetic operators

`+`	add
`-`	subtract
`*`	multiply
`/`	divide
`^`	raise to the power

R as a calculator

How to enter numbers

```
> 10^3 # this computes 10 * 10 * 10
[1] 1000
> 1e3 # this just enters “1000” in scientific notation
[1] 1000
> 10e2
[1] 1000
> -5e3
[1] -5000
> -5e-2
[1] -0.05
> -5e-10
[1] -5e-10
```

The “#” starts a comment, any text after ‘#’ is ignored by R

R as a calculator

Special numbers that are NOT numbers

Missing values are NA, Not Available

```
> x <- c(0,2,4,NA)
> x
[1]  0  2  4 NA
> mean(x)
[1] NA
> mean(x, na.rm=T) # na remove true
[1] 2
```

R as a calculator

Something totally different is NaN, Not any Number

Results from errors

```
> 0/0 # division by zero
[1] NaN
```

Like infinity, Inf

```
> 1/0
[1] Inf
> exp(1e200)
[1] Inf
```

R as a calculator

Some powers

```
> (1/27)^(1/3)
[1] 0.3333333
```

Let's do something useful: body mass index

```
> bmi <- 75/1.75^2
> bmi
[1] 24.48980
```

$$bmi = \frac{weight}{height^2}, \text{units} \left[\frac{kg}{m^2}\right]$$

33

R as a calculator

- All the usual functions exist (log, exp, sin, cos, tan, sqrt, etc.)
- `> log(10)`
- `[1] 2.302585 #so log() is the natural logarithm`
- `> exp(1)`
- `[1] 2.718282 # this is e`
- `> pi`
- `[1] 3.141593 # π is known as pi`
- `> sin(pi)`
- `[1] 1.224606e-16 # should be zero and is fairly close`
- `> cos(pi)`
- `[1] -1`

34

Exercise: R as a calculator

- Calculate and assign result into variable with name z
  - Square root of 2
  - Sine of 2π
  - Natural logarithm of 10 and 10^{1000}
  - 2^{10}
  - e^{10}
  - Calculate the values of the following functions

$$\frac{1}{9}, \quad \left(\frac{1}{9}\right)^2, \quad \sqrt[3]{27}, \quad \left(\frac{1}{9}\right)^{\frac{1}{2}}, \quad \sqrt{\frac{1}{9}}$$

35

R basics: writing functions

- Writing your own functions and understand what a function is
- Example

name of the function

input arguments
(here we need 2)

```
add <- function(arg1, arg2) {
result <- arg1 + arg2
result
}
```

output of function

expression block
enclosed by {...}

  - NB: Write this as a script in the script editor, then execute
  - Use like this

```
> s <- add(1,2)
> s
[1] 3
```

36

R basics: data structures

- Scalar
  - single value of a physical quantity (product of numerical value and physical unit)
  - e.g. 250 µM oxygen concentration in lake water
- Vector
  - series of scalars
  - e.g. oxygen concentration gradient along a transect
  - e.g. oxygen concentration in a number of lakes
- Matrix (2D Array)
  - table of scalars (e.g. a field of scalars, spatial)
  - e.g. oxygen concentration field over lake surface (2D) or including depth (3D matrix/array)

R basics: data structures

- Dataframes
  - O_2 concn as response variable, lake depth, pH, season, nutrient availability as (potential) explanatory variables
  - This is how you should organize most of your results
- Lists
- Tables
- Timeseries

R basics: constructing vectors

Many ways to make vectors

Simplest is the colon “:”

```
> x <- 5:1
[1] 5 4 3 2 1
> x <- 0:4
[1] 0 1 2 3 4
```

Versatile is the concatenate function “c”

```
> x <- c(0,2,-1,pi,10)
[1]  0.000000  2.000000 -1.000000  3.141593 10.000000
> x <- c(0:5,10:15)
 [1]  0  1  2  3  4  5 10 11 12 13 14 15
```

R basics: constructing vectors

Sequences with finer control than “:” (colon operator) with seq

```
> x <- seq(0,2,.25)
[1] 0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
> length(x)
[1] 9
> x <- seq(0,2,length=11)
[1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0
```

Repeating elements with rep

```
> rep(1:2,4)
[1] 1 2 1 2 1 2 1 2
```

R basics: working with vectors

- Functions also work on vectors (element-wise)!

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x2 <- x^2
> x2
 [1]   1   4   9  16  25  36  49  64  81 100
> sqrt(x2)
 [1]  1  2  3  4  5  6  7  8  9 10
```

R basics: working with vectors

```
> p <- seq(-2*pi,2*pi,pi/2)
> p
[1] -6.283185 -4.712389 -3.141593 -1.570796
0.000000  1.570796  3.141593  4.712389
6.283185
> sin(p)
[1]  2.449213e-16  1.000000e+00 -1.224606e-16
-1.000000e+00  0.000000e+00  1.000000e+00
1.224606e-16 -1.000000e+00 -2.449213e-16
```

Exercise: vectors

- Make these vectors
  - 0.00 0.25 0.50 0.75 1.00
  - 0 7
  - 1 2 3 1 2 3 1 2 3
  - Make a vector x containing numbers 0,1,2,...,10
  - Calculate vectors y according to these equations

$$y = x^2, \quad y = \sqrt{x}, \quad y = \frac{1}{x}, \quad y = \frac{1}{x^2}$$

  - Make a vector x containing 100 elements from -π to π (equally spaced)
  - Calculate vectors y according to these equations

$$y = 2\sin(x), \quad y = \cos(2x)$$

R basics: vector subscripts (indices)

- Accessing blocks of vectors with subscripts
  - Distinguish index from value of a vector element
  - You access some elements of vectors rather than all using `[]`

```
v <- -50:50 # make simple example vector
mean(v) # this uses all elements of v
v[1] # first element
v[40:60] # a section in the middle
v[length(v)] # last element since length() gives number of elements
v[(length(v)-9):length(v)] # use the last 10 elements (maybe you want to select the final stage)
```

R basics: vector subscripts (indices)

- Something special: you can select elements of a vector that fulfil a logical condition
  - `v[v > 0] # all elements for which condition is true will be selected`
  - `v[v == 50] # exactly equal, may give no hit`
  - `v[v == mean(v)] # also unlikely to get a hit`

Exercise: vector subscripts (indices)

- Make a vector from 0 to 99
  - Select these elements
    - first
    - last
    - 11 in the middle
    - the last 50
  - Select elements based on logical conditions
    - all elements smaller than 10
    - all elements smaller and equal to 10
    - all elements equal to 10
    - all elements equal to 100
    - all elements equal to 0.1

R basics: matrix subscripts (indices)

- Accessing blocks of matrices with subscripts using `[]`
  - Arrays with 2 dimensions (2D array or 2D matrix)
    - 1st dimension: rows
    - 2nd dimension: columns
      - That's how we read newspapers
  - Accessing blocks of elements in an 2D array
    - A[1,2]
    - Read like this: [rows , columns]
    - A[,1]
    - Empty means everything (all rows, or all columns)

R basics: matrix subscripts (indices)

- Construct our example array from a vector

```
> v <- 0:15
> v
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12
13 14 15
> A <- matrix(v,4,4)
> A
     [,1] [,2] [,3] [,4]
[1,]    0    4    8   12
[2,]    1    5    9   13
[3,]    2    6   10   14
[4,]    3    7   11   15
```

Exercise: matrix subscripts

- Now play with this matrix, selecting
  - the entire first row
  - the entire first column
  - the block in the middle
  - the last two entire columns
  - all values larger than 10

R basics: dataframes

You can make such dataframes in Excel!

Response variable

Explanatory variables

Field.Name	Area	Slope	Vegetation	Soil.pH	Damp	Worm.density
Nashs.Field	3.6	11	Grassland	4.1	F	4
Silwood.Bottom	5.1	2	Arable	5.2	F	7
Nursery.Field	2.8	3	Grassland	4.3	F	2
Rush.Meadow	2.4	5	Meadow	4.9	T	5
Gunness.Thicket	3.8	0	Scrub	4.2	F	6
Oak.Mead	3.1	2	Grassland	3.9	F	2
Church.Field						
Ashurst						
The.Orchard						
Rookery.Slope						
Garden.Wood						
North.Gravel	3.3	1	Grassland	4.1	F	1
South.Gravel	3.7	2	Grassland	4	F	2
Observatory.Ridge	1.8	6	Grassland	3.8	F	0
Pond.Field	4.1	0	Meadow	5	T	6
Water.Meadow	3.9	0	Meadow	4.9	T	8
Cheapside	2.2	8	Scrub	4.7	T	4
Pound.Hill	4.4	2	Arable	4.5	F	5
Gravel.Pit	2.9	1	Grassland	3.5	F	1
Farm.Wood	0.8	10	Scrub	5.1	T	3

All values of the same variable must be in the same column!

R basics: dataframes

- Dataframes: THE data structure for your results
- Create table of results in Excel (that's easier)
  - "Save as" "Text file (Tab delimited) (*.txt)"
  - Make sure you know where you saved the file, and its name
  - Let's call the file "worms.txt"
- In R, you can use the menu (File > Change dir…) to select the directory where you have saved the file, makes entering the file name easier
  - Read this txt file with read.table() like this
  - `worms <- read.table("worms.txt",header=TRUE,row.names=1)`
    - File name must be correct and exist where R is looking & put in quotes
    - The first row is the header, containing the names of the variables
    - The first column contains the row names (don't treat Field.Name as variable)
    - Variable names must be ONE word (no blanks)
    - Missing values should be denoted NA (no blanks)

This is a string
Always put in "quotes"

R basics: dataframes

- Assigning the data to the variable "worms", the dataframe will be known by this name
  - `worms <- read.table("worms.txt",header=TRUE)`
  - `worms`
  - `worms <- read.table("worms.txt",header=TRUE,row.names=1)`
  - `# now the first column is treated as row label, not explanatory variable`
  - `worms`
  - `class(worms) # yes, it is a dataframe`
  - `names(worms) # lists the names of the variables in the dataframe`
  - `plot(worms) # plots all variables against all others, pH seems to be the most interesting`

R basics: dataframes

- `plot(Soil.pH,Worm.density) # error message, the variables are not known!`
- `attach(worms) # this makes the variables inside the dataframe visible to the outside world`
- `plot(Soil.pH,Worm.density) # now it works`

- `summary(worms)`
- `# now let’s try subscripting to select parts of the dataframe`
- `worms[,4] # all rows, column 4`
- `worms[Soil.pH >= 5,] # those rows with pH >= 5, all columns`

R basics: dataframes

- Sorting and ordering (use order() for dataframes)
  - `sort(Soil.pH) # don't use sort on dataframes, it will only sort one column, leaving the rest of the row as it was`

  - `i <- order(Soil.pH) # use order instead, order returns subscripts (indices)`
  - `i # let’s have a look at the indices i`
  - `Soil.pH # unordered`
  - `Soil.pH[i] # ordered`
  - `worms[i,] # now the whole dataframe in order`

R basics: dataframes

- `# these commands do the same but are less easy to read`
- `ws <- worms[order(Soil.pH),] # all in one line`
- `ws <- worms[order(worms[,4]),] # the same`

- Remove variables so we can use the same names again without creating confusion
  - `detach(worms) # undo attach`
  - `rm(worms) # rm for remove`

Exercise: dataframes

- Example: Enzyme activity as a function of substrate concn
  - Download file KMINDOL.xls from WebCT
  - Open file in Excel
    - Inspect data: are there variable names in the header, are there row names, and if so, in which column?
    - Save as tab-delimited text file KMINDOL.txt
  - Read text file into R dataframe
  - Examine variable names and dataframe content
  - Attach dataframe
  - Plot the response variable versus the explanatory variable

R: plotting

- Plot a function to see it’s shape
  - `x <- seq(-2*pi,2*pi,length=101) # make vector x`
  - `y <- sin(x) # calculate vector y as a function of x`
  - `plot(x,y)`
  - `plot(x,sin(x)) # another way of doing the same`
- The result is OK, but for publication quality this default plot is not good enough, we want lines not points (type=“l”), y-axis labels horizontal (las=1), etc.
  - `plot(x,y, las=1, xlab="x", ylab="sin(x)", type="l", main="Trigonometric functions")`
- Many more parameters can be set within the plot() call

R: plotting

- Distinguish high-level from low-level plotting functions
  - High-level plotting functions create a new plot, complete with axes, ticks, labels etc.
    - These are difficult to change afterwards!
    - Make plot behave by setting graphics parameters before the call to plot, e.g.
      - `par(las=1)`
    - Or set them within the high-level plotting function, e.g.
      - `plot(x,y, las=1, ...)`
    - Example high-level plotting functions
      - plot(), qqplot(), hist(), image(), contour(), wireframe(), cloud(), etc.

R: plotting

- Low-level plotting functions add to an existing plot
- Let’s plot sin(x) again
  - `plot(x,y, las=1, xlab="x", ylab="f(x)", type="l")`
- Now that we have a plot, we can add
  - `lines(x,cos(x),col="red")`
  - `points(x, cos(x)+rnorm(length(x), mean=0, sd=0.1), col="blue") # cos(x) with some noise added`
  - `title("Trigonometric functions")`
  - `text(0.5*pi+pi/4, sin(0.5*pi), "sin(x)")`
  - Example low-level plotting functions: lines, points, text, title, legend, grid, arrows, etc.

R: plotting

- Here is a list of options for plot() that are commonly used (there are far more). Many can also be used in other plotting functions

```
xlim=c(0,80), ylim=c(0,1) # set x and y axis limits
log=“x”, log=“y”, log=“xy” # logarithmic axes
asp=1 # aspect ratio of axes (y/x), use 1 for square axes
type=“whatever” # l for lines, p for points, b for both, n
  for nothing, o for both overplotted, s for stairs
col=“yellow” # never use yellow, it is very hard to see
lwd=1 # line width
lty=1 # line type: 1 solid, 2 dashed, etc up to 6, more
  control with “8414”
pch=1 # plot character, can use integers in range 0:25 or
  (easier) a single character in quotes, e.g. pch=“+”,
  pch=“*”, pch=“?”
```

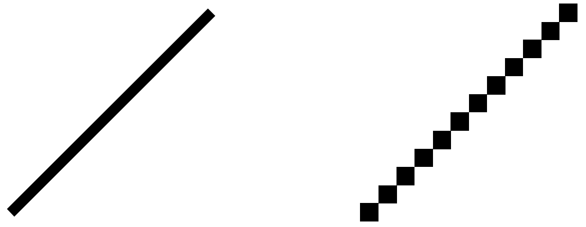
Exercise: plotting

- Plot all functions into one plot
  - Make vector for x axis, from 0 to 2 (make 100 points for smooth curve)
  - Plot $y=x^2$ versus x as a red line
  - Plot $y=\sqrt{x}$ versus x as a blue line
  - Plot y=x versus x as a black line

From plot window to publication

- eps: encapsulated postscript
- postscript is a page description language, a programming language high end printers can understand (interpret)
  - basically a text file containing commands for the printer
    - put this there
- encapsulated postscript is for single objects rather than a complete printed page/document
- a bit like pdf, also from Adobe
- used by publishers
- ideal for submission of vector graphics to journals
- (for bitmaps, use tiff)

Vector graphics versus pixel graphics (bitmaps)



From plot window to PowerPoint

- Right click on R graphics window and select
  - Save as postscript...
    - Select encapsulated postscript as file type
    - Choose a file name, extension eps
- Open file in GSview (GhostScript view) to display eps file
  - GSview can convert to pdf, print to printer, etc.
  - You need to install GhostScript and GSview
  - http://pages.cs.wisc.edu/~ghost/
- Zoom to the final size you want to use in PowerPoint
  - Do not change the size in PowerPoint, quality will suffer
- Copy the image window (Control-C)
- Paste into PowerPoint

From plot window to Word

- Save eps file as before
- Insert eps file in Word
  - Insert picture from file
  - Word cannot display eps, instead it displays a crude bitmap of the eps, a sort of thumbnail tiff image stored inside the eps file
  - You can print this file to a postscript printer no problem
  - You cannot print to normal cheap printers
  - But you can save as pdf and print that

From plot window to Vector graphics software

- For further editing of the figure such as annotations with arrows, text, equations
- Save eps file as before
- Import or open the eps file in a good Vector graphics program that can interpret the eps format as vector graphics (rather than producing a bitmap from it, which will not be editable)
  - Corel Draw can import and export eps
  - Inkscape can import pdf (R can export figure as pdf)

List of some useful R functions

- Central tendency
  - mean()
  - median()
- Other useful functions
  - sum()
  - prod()
  - length()
  - sort()
  - order()
  - min()
  - max()
  - abs()
- Measures of variation
  - range()
  - quantile() (percentiles)
  - boxplot()
  - var()
  - sd()
  - mad() (median absolute deviation)