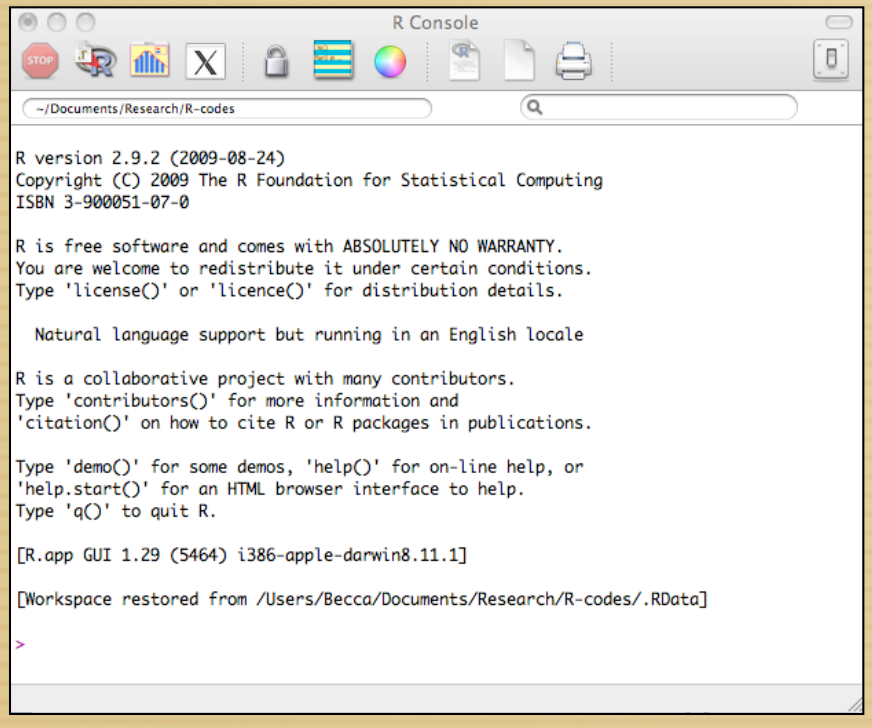
The very Basics

The R Command-line




```
> 5          5
> 5+5        10
```

Assign Information to Variables

```
> x <- 5
> x          5
> x*5        25
> x <- y <- 5
> x <- "Krat"
> x <- 'Krat'
```

```
> x <- 5
> x = 5
> 5 -> x
```

Typically interchangeable

Variables are CASE SENSITIVE (Krat ≠ krat)

Text Editor Programs

Make Life So Much Easier!!!

Lets you save your codes - record of what you have done

Color coded to show you types of objects in your code

Shows you when you brackets or parentheses are unmatched

```
72  ##################################################################
73  ## How I do Rarefacation Curves
74  ##################################################################
75  ## Call the plot space - No Axes
76  ####################################
77  plot(x07rare$N, x07rare$ES,type="n", xlab="Number of Individuals", ylab="Estimated Richness
    ...     +/- 95% CI", main="Grasshopper Rarefaction", axes=FALSE)
78
79  ## Add in the axes
80  ##################
81  axis(1,lwd=1.5, tcl=-0.3, mgp=c(2,0.4,0))
82  axis(2, lwd=1.5, tcl=-0.3, mgp=c(2,0.5,0),las=1)
83  box(which="plot",lwd=2,bty="l")
84
```

The R Command-line

Action	Symbol	Example
Arithmetic	+ - * / ^	4*(2+2)^3
Grouping	{ [(	[3,5]
Assignment	<- = ->	x <- 8

Types of Variables

Variable Type	Example
Numeric	5, 30.5, -0.362
Character	“Ariel, Lauren, Allison, Cara”
Logical	TRUE, FALSE, T, F
Factor	{for categorical data}
Complex	2 + 3i

Functions

Functions take information, do something with it, and return a result

```
> sqrt(9)        3
```

Functions can be nested

```
> mean(c(9,5,2))     5.33
```

Functions take arguments, which specify their behavior:

```
> seq()
> seq(1,5)     1 2 3 4 5
> 1:5          1 2 3 4 5

> seq(1,5,0.5)     0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
```

```
> seq(from = 1,
      to = 5,
      by = 0.5)
```

Must use '=' for specifying arguments

Functions—Using help pages

How do we know what arguments a function takes?

> ?seq

seq {base} R Documentation

Sequence Generation

Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is an internal generic which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

Usage

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq_along(along.with)
seq_len(length.out)
```

Arguments

`...`	arguments passed to or from methods.
`from, to`	the starting and (maximal) end value of the sequence.
`by`	number: increment of the sequence.
`length.out`	desired length of the sequence. A non-negative number, which for `seq` and `seq.int` will be rounded up if fractional.
`along.with`	take the length from the length of this argument.

Functions—Using help pages

How do we know what arguments a function takes?

> ?seq

Details

The interpretation of the unnamed arguments of `seq` and `seq.int` is *not* standard, and it is recommended always to name the arguments when programming.

Both `seq` are `seq.int` are generic, and only the default method is described here. Typical usages are

Value

Currently, the default method returns a result of type `"integer"` if `from` is (numerically equal to an) integer and, e.g., only `to` is specified, or also if only `length` or only `along.with` is specified. **Note:** this may change in the future and programmers should not rely on it.

See Also

The methods `seq.Date` and `seq.POSIXt`.

How do we know which function to use?

> ??"sequence"

> help.search"sequence"

Use the Google: type in "*R create sequence*"

Data Structures

To make a **Vector:**

```
> x <- c(5, 10, 15, 20)          # Concatenate fxn
```

1st 2nd 3rd 4th Elements in the vector

```
> length(x)          4
```

Extracting/Subsetting Elements from a Vector Using **[ ]**

```
> x[2]          10
> i=2;  x[i]     10
> x[2:3]        10  15
```

Note that:

```
> x[c(2,3)]  =  x[2:3]
> x[c(1,3)]  ≠  x[1:3]
```

Why?

Data Structures

Array

```
> X <- array(NA, dim=c(3,3,3))
```

1D Array = Vector

2D Array = Matrix

```
> Y <- matrix(1:6, nrow=3)
```

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
> Y[1,2]    # element in 1st row 2nd column
> Y[1,]     # entire 1st row
> Y[,2]     # entire 2nd column
```

Other useful functions

```
> dim(X)      3 3 3
> nrow(Y)     3    # number of rows
> ncol(Y)     2    # number of columns
```

For Loops

Iterative creation of vectors, matrices, arrays...

```
> X <- array(NA,dim=5)                    NA NA NA NA
> for(i in 1:4){ X[i] <- i^2 }             1 4 9 16 NA

> X <- array(NA,dim=5); X[1]<-2           2 NA NA NA
> for(i in 1:4){ X[i+1] <- X[i]^2 }        2 4 16 246 65336
```

Functions / Operations on Arrays

Can apply functions or operations to numbers, vectors, and matrices

Operations are *usually* done element-wise

```
> x <- 1:4        1 2 3 4
> x+5             6 7 8 9
> sqrt(x)         1 1.4 1.7 2
> mean(x)         2.5
```

Logical and Comparison Operators

Testing Relationships

Greater than, less than	>, <
Greater or equal to, less than or equal to	>=, <=
Equal, not equal	==, !=
AND, OR	&, |

```
> x <- 4
> x > 10        FALSE

> x <- c(4, 8, 30, 52)
> x > 10        FALSE FALSE TRUE TRUE

> x <- c(4, 8, 30, 52)
> x >10 & x < 50        FALSE FALSE TRUE FALSE
```

Extracting Subsets of Data

```
> x <- c(4, 8, 30, 52)
```

What if only want values greater than 10?

A) Choose by their indices

```
> x[c(3,4)]        30  52
```

! "bangs" = opposite

```
> !FALSE      TRUE
```

B) Using logical T/F vectors

```
> x > 10        FALSE FALSE TRUE TRUE
> x[x > 10]     30  52
```

C) Which commands

```
> which(x > 10)       3   4
```

Vector indices

```
> y <- which(x > 10)       3   4
> x[y]            30   52
> x[which(x > 10)]          # shortcut
```

3) Summarizing and Reordering Data

Useful Commands

```
> x <- c(2, 569, 7, 3, 45, 9, 10, 32, 56, 2210)
> rank(x)   1  9  3  2  7  4  5  6  8  10        # returns vector positions

> order(x, decreasing=T)   10  2  9  5  8  7  6  3  4  1
> sort(x, decreasing=T)    2210 569 56 45 32 10 9 7 3 2
```

Element names can also be used for ordering

```
> x <- c(10, 3, 5)
> names(x) <- c("b", "c", "a")
> x[c("a", "b", "c")]          5  10  3
```

Specify the order of the names you want

Applying Functions

To calculate the mean of a column

```
> x<-matrix(c(1:3,NA),nrow=2)
> mean(x[,1])        1.5
```

Time-consuming for large datasets

Apply command = for all rows or columns simultaneously

```
> apply(x, 2, mean)        1.5  NA
```

By Columns
(1 = rows)

Function

What to do about **NA's (blanks in the dataset)**?

To ignore:

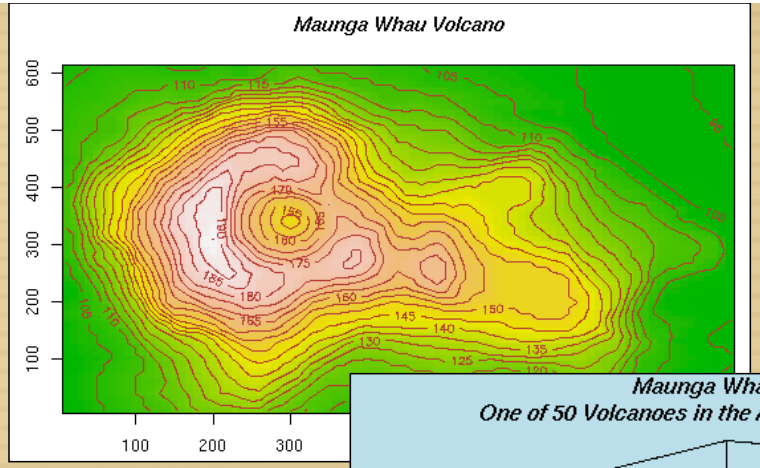
```
> apply(x, 2, mean, na.rm=T)   1.5  3
```

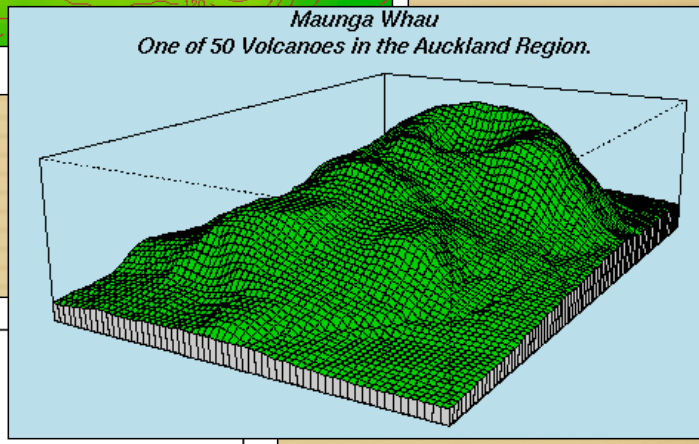
Graphing

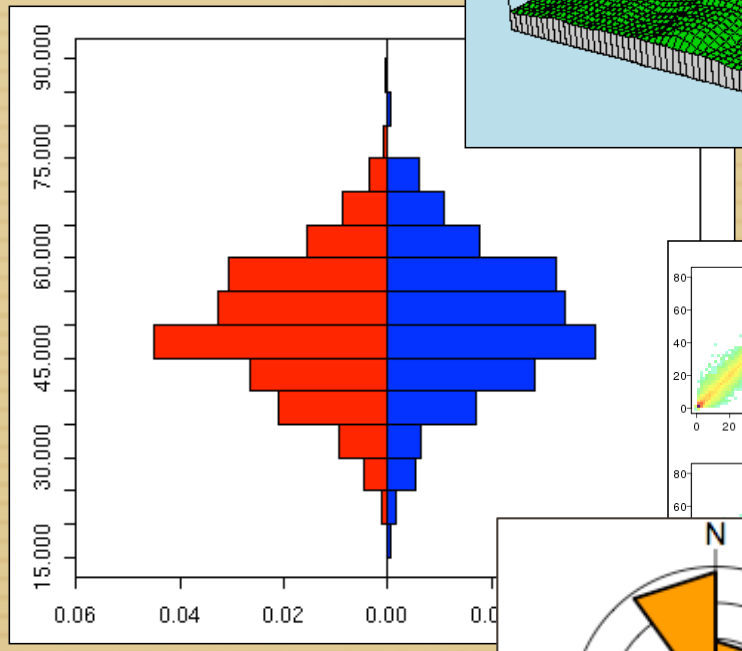
Powerful graphing capabilities

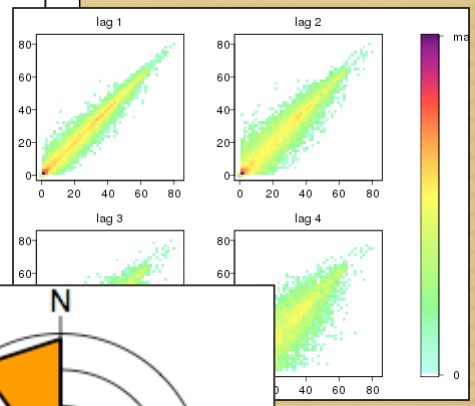
Can save plots as PDF, postscript,…

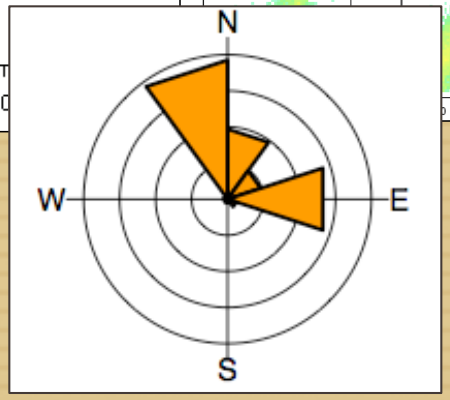
Can add to a plot, but can't edit what is already there (not clickable, *typically*)



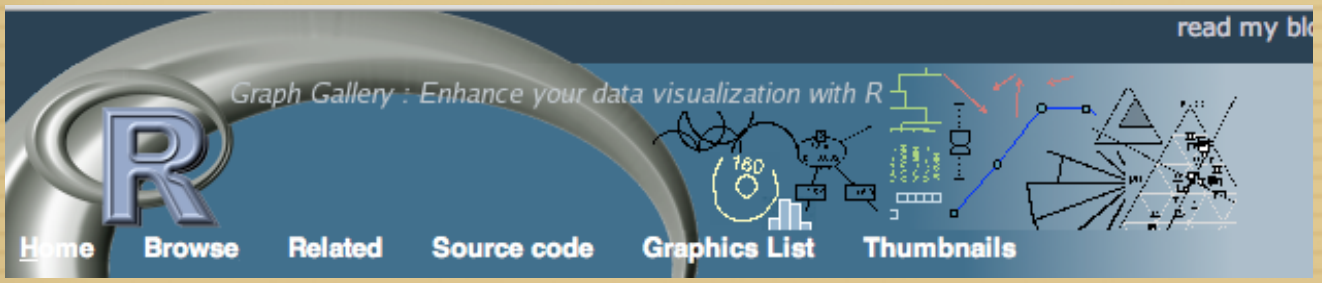








Fun site for example graphs with source codes:
http://addictedtor.free.fr/graphiques/



1) Types of Graphs – XY Plots

Generate some fake data

```
> X <- rnorm(n=20, mean=0, sd=1)      # random normal numbers
                                      # same as rnorm(20, 0, 1)
                                      # same as rnorm(20)

> y <- rnorm(n=20, mean=100, sd=10)
```

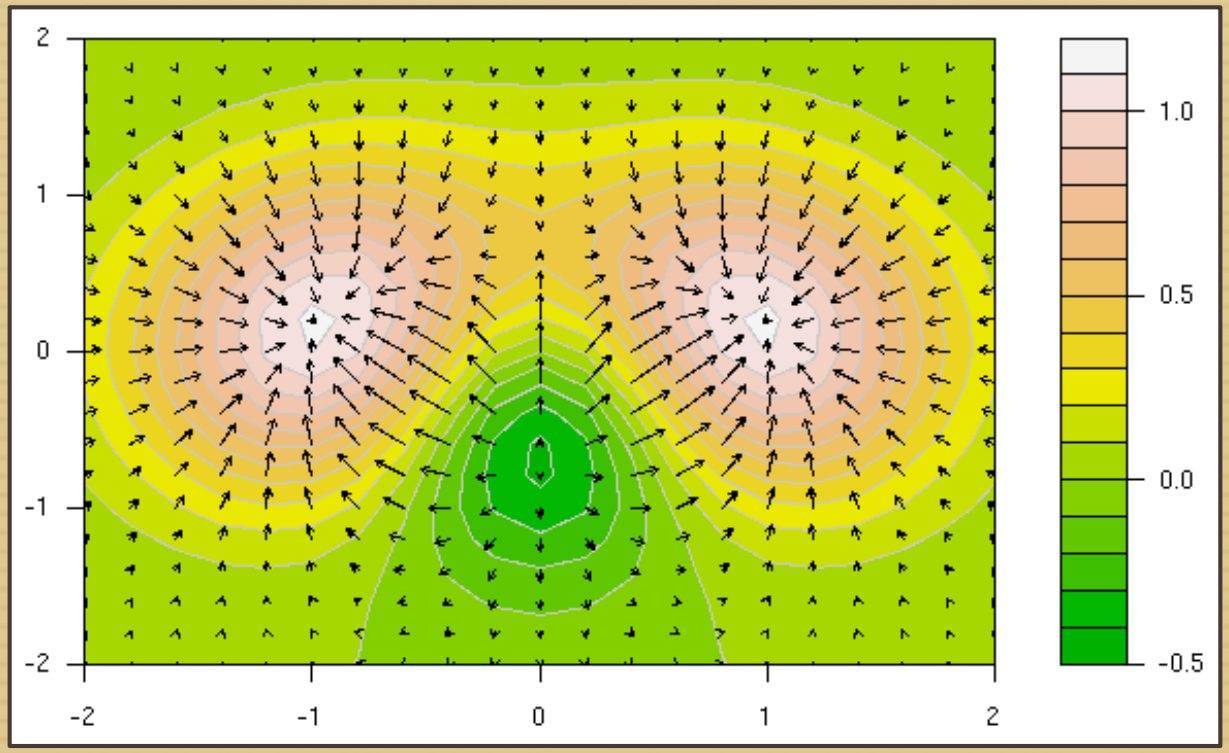
Try some plots….

```
> plot(x)
> plot(x, type="l")
> plot(x, type="b")
```

```
> plot(x, y)
> plot(x, y, pch=19, cex=2)
> plot(x, y, pch=2, col="red")
> plot(x, y, pch=19, col=seq(1,length(x)))
```

1) Types of Graphs – Other Common Graphing Functions

Type of Plot	Function
Histogram	hist()
Barplot	barplot()
Boxplot	boxplot()
Pie Chart	pie()
Contour Plot	contour()



2) Common Arguments to Graphing Functions

Graph Feature	Argument	Example
Color of Points	col	col="red", col=2
Character Size	cex	cex=2 # twice as big
Plotting Symbol	pch	pch=5 # diamonds
Line Type	lty	lty="dashed", lty=2
Log-scale Axes	log	log="x", log="xy"

Par Statement - A very important graphics tool > ?par

par {graphics} R Documentation

Set or Query Graphical Parameters

Description

`par` can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to `par` in `tag = value` form, or by passing them as a list of tagged values.

Graphical Parameters

`adj`

The value of `adj` determines the way in which text strings are justified in `text`, `mtext` and `title`. A value of `0` produces left-justified text, `0.5` (the default) centered text and `1` right-justified text. (Any value in *[0, 1]* is allowed, and on most devices values outside that interval will also work.) Note that the `adj` argument of `text` also allows `adj = c(x, y)` for different adjustment in x- and y- directions. Note that whereas for `text` it refers to positioning of text about a point, for `mtext` and `title` it controls placement within the plot or device region.

3) Adding to Existing Plots

To add:	Argument
Title	title()
Line	abline()
More Points	points()
Text	text()
Polygon	polygon()
Legend	legend()
Arrows/Error Bars	arrows()
Line Segments	segments()
Rectangles	rect()
Symbols	symbols()