R: Graphics

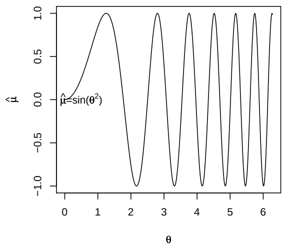
140.776 Statistical Computing

September 13, 2011

Mathematical notations

You can show mathematical symbols in texts using the function expression(). Please use help(plotmath) to learn details.

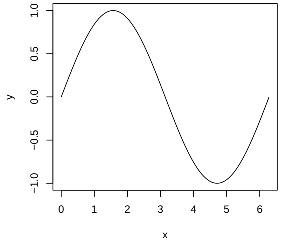
```
> x<-seq(0,2*pi,by=0.01); y<-sin(x^2)
> xtext<-expression(theta)
> ytext<-expression(hat(mu))
> mtext<-expression(paste(hat(mu),"=sin(",theta^2,")"))
> plot(x,y,type="l",xlab=xtext,ylab=ytext)
> text(0.5,0,mtext)
```

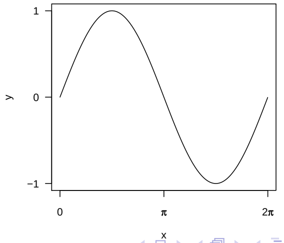


Axes

Axis() can be used to set the axis line and tick marks:

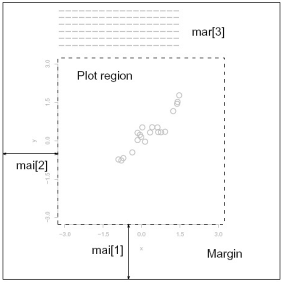
```
> x<-seq(0,2*pi,by=0.01); y<-sin(x)
> plot(x,y,type="l")
> plot(x,y,type="l",xaxt="n",yaxt="n")
> u<-2*pi*(0:2)/2
> axis(1,at=u,labels=c("0",expression(pi),
+ expression(paste("2", pi))))
> axis(2,at=c(-1,0,1),las=1)
```





Margin

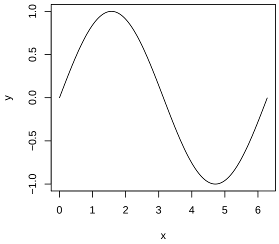
- mai sets margins measured by inches
- mar sets margins using text line as measurement unit

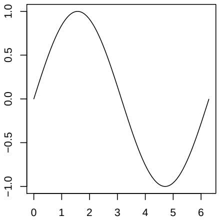


Venables and Smith, An introduction to R

Margin

```
> plot(x,y,type="l")
> oldpar<-par(mai=c(0.5,0.5,1,1))
> plot(x,y,type="l")
> par(oldpar)
```

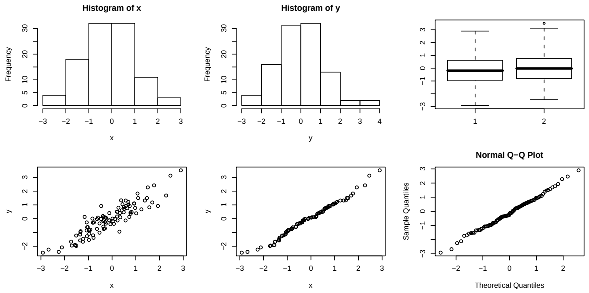




Multiple figures

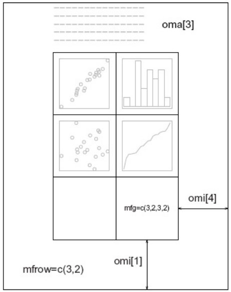
mfcol and mfrow allow you to create multiple figures on a single page. mfcol fills the subplots by column, and mfrow fills by row.

```
> x<-rnorm(100); y<-x+rnorm(100,sd=0.5)
> oldpar<-par(mfrow=c(2,3),oma=c(0,1,1,0),mar=c(4,4,3,2))
> hist(x); hist(y); boxplot(x,y)
> plot(x,y); qqplot(x,y); qqnorm(x)
> par(oldpar)
```



Margin

- mai sets margins measured by inches
- mar sets margins using text line as measurement unit

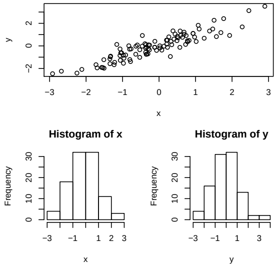


Venables and Smith, An introduction to R

Multiple figures

layout() is another way to organize multiple figures:

```
> layout(matrix(c(1,1,2,3),2,2,byrow=TRUE))
> oldpar<-par(oma=c(0,1,1,0),mar=c(4,4,3,2))
> plot(x,y)
> hist(x); hist(y)
> par(oldpar)
```



Save plots

You can save plots to files using pdf(), postscript(), png(), jpeg(), bmp(), tiff(), bitmap(). For example, the plot in the previous slide can be saved to a pdf file using the commands below:

```
> pdf("testplot.pdf",width=4, height=4, pointsize=10)
> layout(matrix(c(1,1,2,3),2,2,byrow=TRUE))
> oldpar<-par(oma=c(0,1,1,0),mar=c(4,4,3,2))
> plot(x,y)
> hist(x); hist(y)
> par(oldpar)
> dev.off()
```

Graphics devices

R can generate graphics on many types of devices:

- X11(), windows(), quartz() are default for UNIX, Windows, and Mac OS X respectively.
- pdf(), png(), jpeg(), etc.

Graphics devices

To plot to a specific device, usually you need to go through the following steps:

1. Open a device driver, e.g. pdf()
2. Make a plot
3. Close the device driver, e.g. dev.off()

Graphics devices

Sometimes you want to work with multiple devices and copy figures from one device to the others. To do this, read help documents for these functions:

- dev.list()
- dev.prev(), dev.next()
- dev.set()
- dev.copy(), dev.print()
- graphics.off()
- ...

Plotting packages

There are several R packages for plotting:

- **graphics**: contains functions for the "base" graphing systems, including plot, hist, etc. This is the package we've talked about so far.
- **lattice**: contains code for producing Trellis graphics, including xyplot, bwplot, levelplot etc.
- **grid**: implements a different graphing system independent of "base"; the **lattice** package builds on top of **grid**; we seldom call functions from the **grid** package directly.
- **grDevices**: contains all the code implementing the various graphics devices, including X11, PDF, PostScript, PNG, etc.

Lattice functions

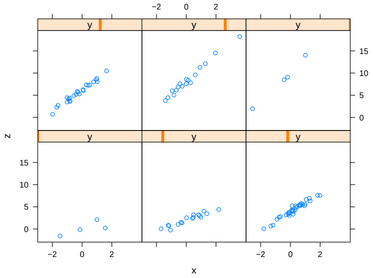
This package contains a lot of functions:

- xyplot: scatterplots
- bwplot: box-and-whisker plots
- histogram
- stripplot: like a boxplot but with acutal points
- ...

Lattice functions

Lattice functions usually take a formula as the first argument: $z \sim x \mid y$. It means plotting z vs. x conditional on y.

```
> library(lattice)
> x<-rnorm(100)
> y<-rbinom(100,5,0.5)
> z<-x+2*y+0.5*x*y+rnorm(100,sd=0.5)
> xyplot(z~x | y)
```



Reading data from files

read.table() is one of the most commonly used functions for reading data.

```
> x1<-read.table("data1.txt")
> x2<-read.table("data2.txt")
> x3<-read.table("data3.txt")
```

Reading data from files

```
> x1
   V1   V2       V3 V4       V5        V6
1   1 chr1 11963092  + 5.160150 TGGGTGGTC
2   2 chr1 12677790  + 3.799299 TGTGTGGTC

> x2
     V1         V2       V3     V4                       V5        V6
1  seq_id chromosome    start strand log10(Likelihood_Ratio)      site
2       1       chr1 11963092      +                  5.16015 TGGGTGGTC
3       2       chr1 12677790      +                 3.799299 TGTGTGGTC

> x3
   chromosome    start strand log10.Likelihood_Ratio.      site
1        chr1 11963092      +                5.160150 TGGGTGGTC
2        chr1 12677790      +                3.799299 TGTGTGGTC
```

Reading data from files

A few important argument of read.table():

- **file**: the name of a file.
- **header**: a logical value indicating whether the file contains the names of the variables as its first line. If missing, header is set to TRUE if and only if the first row contains one fewer field than the number of columns.
- **sep**: the field separator character.
- **colClasses**: a vector of classes to be assumed for the columns.
- **nrows**: the maximum number of rows to read in. Negative and other invalid values are ignored.
- **skip**: the number of lines of the data file to skip before beginning to read data.
- **comment.char**: a character string to indicate the comment character.

Reading data from files

```
> x2
      V1         V2       V3     V4                      V5        V6
1 seq_id chromosome    start strand log10(Likelihood_Ratio)      site
2      1       chr1 11963092      +                 5.16015 TGGGTGGTC
3      2       chr1 12677790      +                3.799299 TGTGTGGTC

> x2<-read.table("data2.txt",header=TRUE)
> x2
  seq_id chromosome    start strand log10.Likelihood_Ratio.      site
1      1       chr1 11963092      +                5.160150 TGGGTGGTC
2      2       chr1 12677790      +                3.799299 TGTGTGGTC
```

Reading data from files

Example:

```
First two lines of data3.txt:

      chromosome     start strand log10.Likelihood_Ratio.      site
1          chr1  11963092      +                5.160150 TGGGTGGTC

Read the file:

> x3<-read.table("data3.txt")
> class(x3[,1])
[1] "factor"

> colClasses<-c("character","character",
+"integer","character","numeric","character")

> x3<-read.table("data3.txt",colClasses=colClasses)
>class(x3[,1])
[1] "character"
```

Reading data from files

For small to moderately sized datasets, you can usually call read.table without specifying any other arguments:

```
  > data<-read.table("foo.txt")
```

R will automatically

- skip lines that begin with a #.
- figure out how many rows there are.
- figure out what type of variable is in each column.

Telling R all these things directly makes R run faster.

- read.csv is identical to read.table except that the default separator is comma.
- read.delim uses tab as the separator.

Reading data from files

Specifying *colClasses* instead of using default option makes read.table run much faster, often twice as fast. Here is an example showing you how to figure out the classes of each column automatically:

```
> initial<-read.table("data1.txt",nrows=3)
> classes<-sapply(initial,class)
> tabAll<-read.table("data1.txt",colClasses=classes)
```

Writing data to files

The most commonly used function is write.table(). A few important parameters include:

- **x**: the object to be written, preferably a matrix or data frame. If not, it is attempted to coerce x to a data frame.
- **file**: a file name.
- **sep**: the field separator character.
- **append**: logical, only relevant if file is a character string. If TRUE, the output is appended to the file. If FALSE, any existing file of the name is destroyed.
- **row.names**: either a logical value indicating whether the row names of x are to be written along with x, or a character vector of row names to be written.
- **col.names**: similar to row.names but now for columns.

cat

There are other ways to import or export data. For example, cat() is another way to export data:

```
> cat("Hello!","\n")
Hello!

> cat(file="test.txt","123","987",sep="\n")
```

Reading fixed-width-format files

Data files may have no fixed field separators but have fields in pre-specified columns. The function read.fwf provides a simple way to read such files:

```
> ff<-tempfile()
> cat(file=ff, "123456","987654",sep="\n")
> read.fwf(ff,width=c(1,2,3))
  V1 V2  V3
1  1 23 456
2  9 87 654
> unlink(ff)
```

Other functions

Other useful functions for data import/export are:

- dump(),source()
- save(),load()
- dput(),dget()
- serialize(),unserialize()
- writeLines(),readLines()
- scan()
- ...