Italics

John McKinlay

29 January 2015

This doc is inspired by a need for axis tick labels in *italics*, arrived at by the circuitous route of reading text files with long and/or unusual column headers, 'fixing' them, and creating expressions to capture variable units.

Make some data

First, make a csv with unusual names.

```
temp <- textConnection("
col,  a _very_ _very_ long var name,   col, 99-01,  99-02,  12map
1,                                 A,    10,       1,        4,      z
2,                                BB,    20,     2,      5,        c
3,                               CCC,    30,     3,      6,        s")

mdf <- read.csv(temp,
        header=TRUE,
        strip.white=TRUE,
        colClasses=c("integer", "character", "numeric", "numeric", "numeric", "character"))
close(temp)
mdf
```

```
##   col a._very_._very_.long.var.name col.1 X99.01 X99.02 X12map
## 1   1                             A    10      1      4      z
## 2   2                            BB    20      2      5      c
## 3   3                           CCC    30      3      6      s
```

```
str(mdf)
```

```
## 'data.frame':    3 obs. of  6 variables:
##  $ col                          : int  1 2 3
##  $ a._very_._very_.long.var.name: chr  "A" "BB" "CCC"
##  $ col.1                        : num  10 20 30
##  $ X99.01                       : num  1 2 3
##  $ X99.02                       : num  4 5 6
##  $ X12map                       : chr  "z" "c" "s"
```

Note that out-of-the-ordinary column names are simplified; this is controlled by the argument `check.names=TRUE` which fires function `make.names()` under the hood.

Fix the column names

These names are perhaps not simplied enough. I prefer to work with short names, all lower case, no underscores and no dots. Here's a function for the job, noting that conversions fire sequentially in the order they are specified as arguments.

```
fixNames <- function(yuckNames, lower=TRUE, us2dot=TRUE, rmDot=TRUE, makeShort=TRUE, n=8) {
  # Function to fix up names of an object. Takes a character vector and:
    # - replace underscores with dots
    # - remove dots (periods)
    # - force to lower case
    # - truncate to n characters, make unique
    # eg. fixNames(c("what___.A...PAIN", "..__FoR._.", "incoming..col_NAMES"))
    if(class(yuckNames)!="character") stop("Designed to work on character strings only.")
    fixedNames <- yuckNames
    if(lower) fixedNames <- tolower(fixedNames)
    if(us2dot) fixedNames <- chartr("_", ".", fixedNames)
    if(rmDot) fixedNames <- gsub("\\.", "", fixedNames)
    if(makeShort) fixedNames <- abbreviate(fixedNames, minlength=n)
    return(fixedNames)
}

fixNames(names(mdf))
```

```
##                  col averyverylongvarname                 col1
##                "col"           "avryvryl"               "col1"
##                x9901                x9902              x12map
##              "x9901"              "x9902"             "x12map"
```

```
as.data.frame(fixNames(names(mdf)))
```

```
##                       fixNames(names(mdf))
## col                                    col
## averyverylongvarname              avryvryl
## col1                                  col1
## x9901                                 x9901
## x9902                                 x9902
## x12map                               x12map
```

Now imagine we have many columns, many of which have unreasonably wordy col names. How to deal shortening them (for programming purposes) and keeping track of what they are?

My approach is to create an auxilliary `data.frame` containing the truncated variable names (using `fixNames` above), full variable names, and expressions for use in labelling axes. You could add a description/note column if important.

```
nvar <- ncol(mdf)

vn <- data.frame(
    cn=1:ncol(mdf),
    xcn=paste(c(rep("",26), rep(LETTERS[1:((nvar%/%26))], each=26)), rep(LETTERS, (nvar%/%26)+1), sep="
    shortName=fixNames(names(mdf), n=6),
    oriName=names(mdf),
    stringsAsFactors=FALSE, row.names=NULL)

vn
```

```
##   cn xcn shortName                       oriName
```

```
## 1  1   A       col                           col
## 2  2   B    avryvr a._very_._very_.long.var.name
## 3  3   C      col1                         col.1
## 4  4   D     x9901                        X99.01
## 5  5   E     x9902                        X99.02
## 6  6   F    x12map                        X12map
```

```
names(mdf) <- fixNames(names(mdf))
mdf
```

```
##   col avryvryl col1 x9901 x9902 x12map
## 1   1        A   10     1     4      z
## 2   2       BB   20     2     5      c
## 3   3      CCC   30     3     6      s
```

Column `xcn` (Excel column number) is slightly interesting: if your primary data 'viewer' is MS Excel, and you have an overwhelmning number of columns, it sometime useful to reference a column number to the A-Z system used by Excel. Note that `xcn` above is only defined up to `26 * 26 = 676` combinations (i.e. A through ZZ).

I might also define expressions as part of `vn` for the key variables, particularly those that require some kind of `plotmath` notation for units. More on expressions later, but note here that these are being stored in the `data.frame` as text, not as expressions.

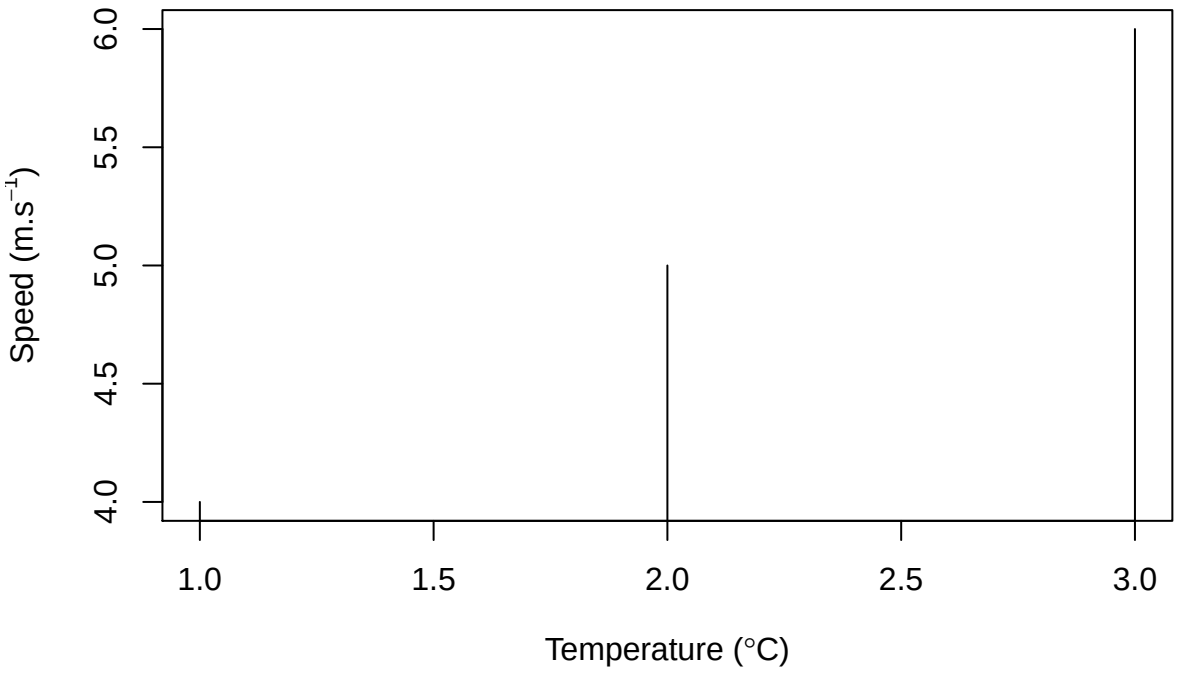
```
vn$expre <- NA
vn$expre[4]  <- 'expression({"Temperature ("}*degree*{"C)"})'
vn$expre[5]  <- 'expression({"Speed (m."}*s^-1*{")"})'

vn
```

```
##   cn xcn shortName                       oriName
## 1  1   A       col                           col
## 2  2   B    avryvr a._very_._very_.long.var.name
## 3  3   C      col1                         col.1
## 4  4   D     x9901                        X99.01
## 5  5   E     x9902                        X99.02
## 6  6   F    x12map                        X12map
##                                       expre
## 1                                      <NA>
## 2                                      <NA>
## 3                                      <NA>
## 4 expression({"Temperature ("}*degree*{"C)"})
## 5        expression({"Speed (m."}*s^-1*{")"})
## 6                                      <NA>
```

Plots and labels

Here we'll look at some of the anatomy of a plot in base graphics, adding our fancy axis labels. Most of the functionality to be discussed can be found in the help pages for `par`, `axis` and `plotmath`. We use `eval(parse())` to turn our fancy labels from pure text to their R construct (`parse`), then their evaluated form `eval` for use by plot.

```
plot(x9902 ~ x9901, type=c('h'), xlab=eval(parse(text=vn$expre[4])), ylab=eval(parse(text=vn$expre[5])))
```



Digression begin: eval and parse?

There's been plenty of R-Help mail over the years about the undesirablity of using `parse()` constructs.

```
fortunes::fortune(106) # among several
```

```
##
## If the answer is parse() you should usually rethink the question.
##    -- Thomas Lumley
##       R-help (February 2005)
```

The main reasons for this, I think, are three potential downsides:

1. If others use your code, you don't know what they might pass in to `parse` (e.g. `text="format C:"`)
2. Errors won't be picked up when code is sourced, only at runtime.
3. It's slow.

In this instance, I don't think these are terribly important and, anyway, I couldn't think of an easy way to pass in the expression without `eval(parse())`.

Examples of points ii) and iii) are:

```
f <- function() { # this will error on source
  mean(1::5)
}

f <- function() { # this will only error at runtime
  eval(parse(text = "1::5"))
}
```

```
system.time(for(i in seq_len(1000)) mean(rnorm(i)))
```

```
##    user  system elapsed
##    0.07    0.00    0.07
```

```
system.time(for(i in seq_len(1000)) eval(parse(text = "mean(rnorm(i))")))
```

```
##    user  system elapsed
##    0.11    0.00    0.11
```

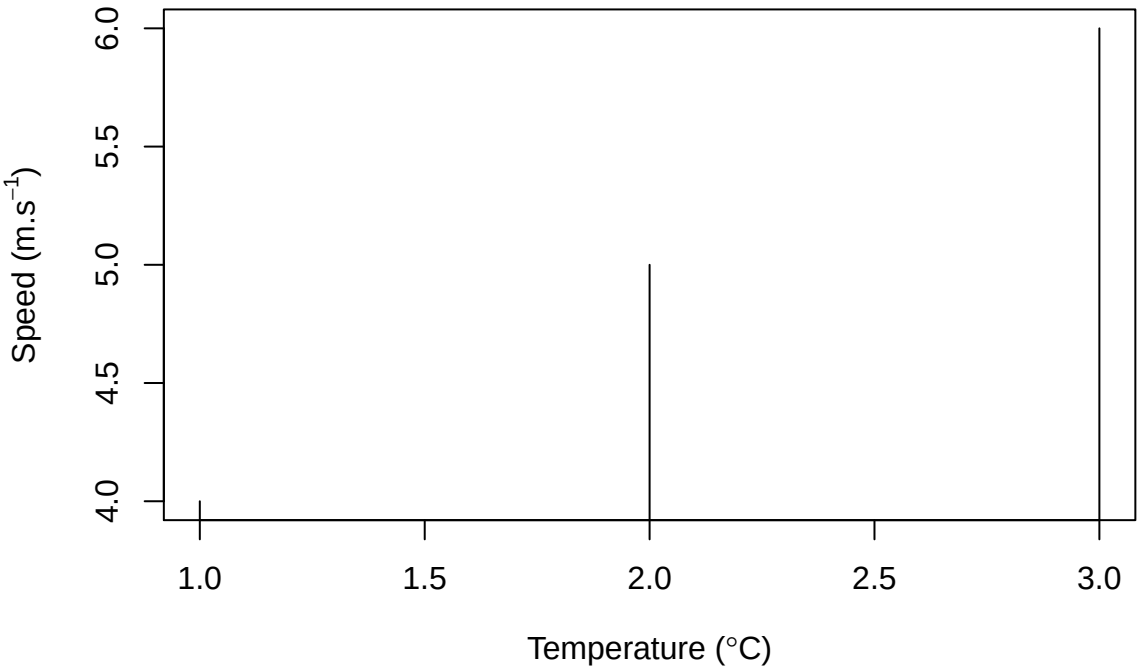
Digression end

We saw on the previous plot that part of the y-axis label was outside the printable area because of the superscript. We can fix this by expanding the margin space on the LHS.

```
par("mar") # 4.1 not enough
```

```
## [1] 5.1 4.1 4.1 2.1
```

```
op <- par(mar=c(5.1,5.1,4.1,2.1))
plot(x9902 ~ x9901, type=c('h'), xlab=eval(parse(text=vn$expre[4])), ylab=eval(parse(text=vn$expre[5])))
```

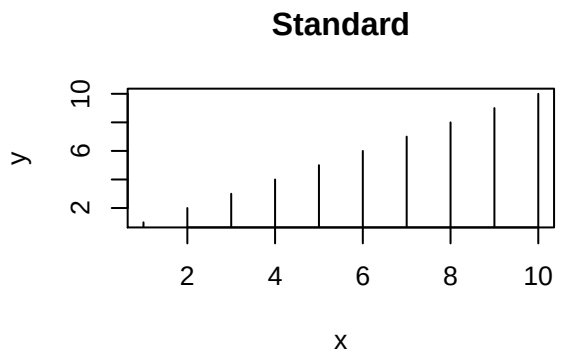


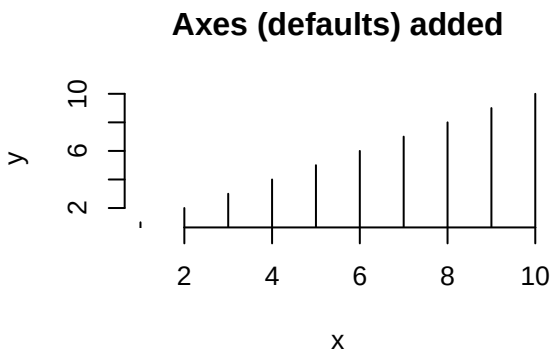

```
par(op)
```

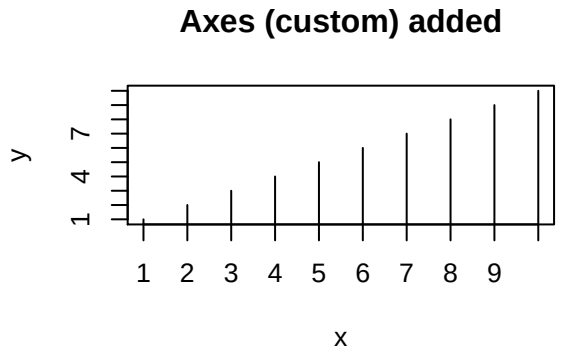
Finally, italics

Base graphics allows you to create most high-level graphics by using a series of low-level calls that build constituent parts. Compare the following three plots:

```
x <- 1:10
y <- 1:10

op <- par(mfrow=c(2,2))
plot(y~x, type='h', main="Standard")

plot(y ~ x, type='h', axes = FALSE, main="Axes (defaults) added")
axis(1)
axis(2)

plot(y ~ x, type='h', axes = FALSE, main="Axes (custom) added")
axis(1, at=seq(1,10,1))
axis(2, at=seq(1,10,1))
box()
par(op)
```

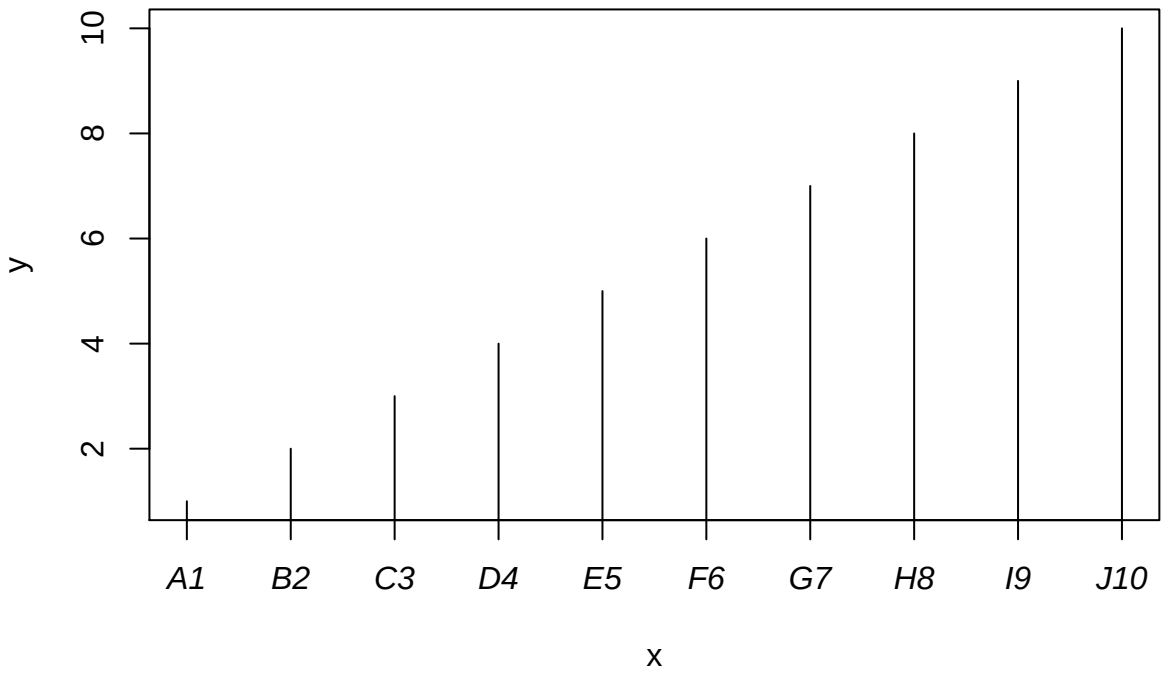






Some/many parameters described in `par()` can be provided to `plot`: for example, `xant` ('x axis type') lets us turn off just the x axis. Italics in axis labels is, in fact, relatively easy.

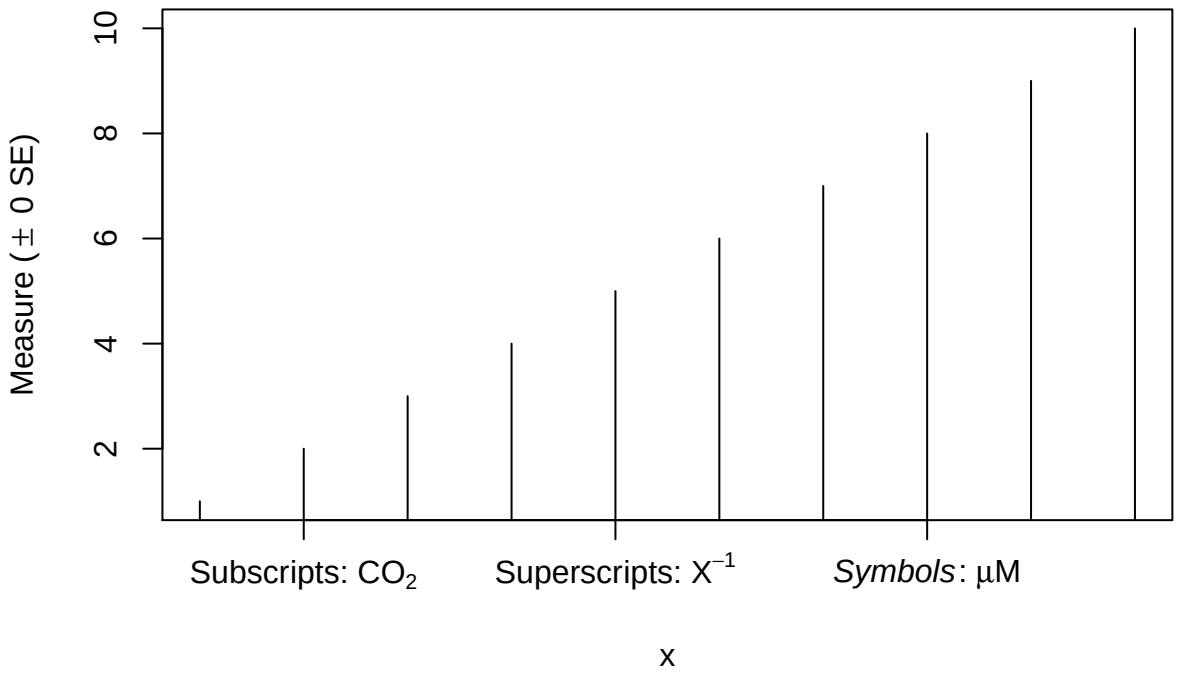
```
plot(y ~ x, type=c('h'), xaxt = 'n')
axis(1, at=1:10, labels=paste0(LETTERS[1:10], 1:10), font=3)
```




```
# 1 is plain text (default), 2 bold, 3 italic & 4 bold italic
```

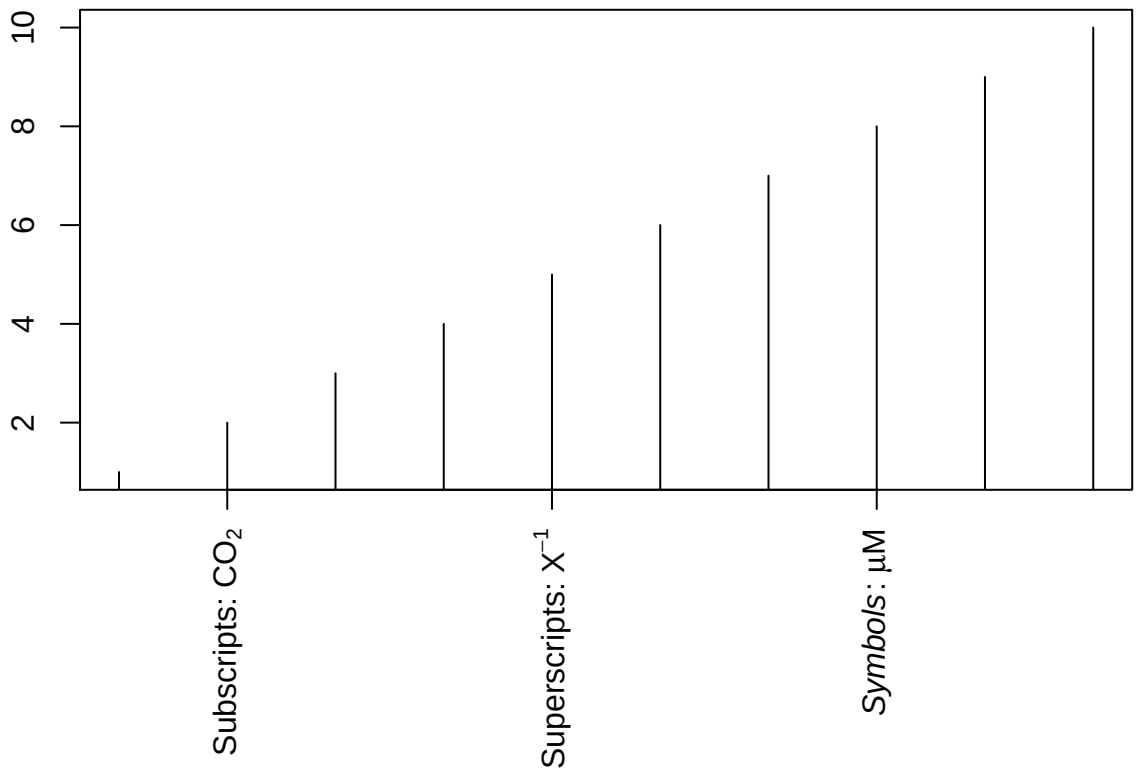
Mixing italics and regular text is also straighforward, but you have to dive into 'plotmath' to get there.

```
exp1 <- expression("Measure (" %+-% " 0 SE)")
exp3 <- c(expression({"Subscripts: "}*CO[2]), expression({"Superscripts: X"}^{-1}), expression({italic(

plot(y ~ x, type=c('h'), xaxt = 'n', ylab=exp1)
axis(1, at=c(2,5,8), labels=exp3)
```



Can we rotate the labels? Yes. `axis` will work for horizontal/vertical rotation.

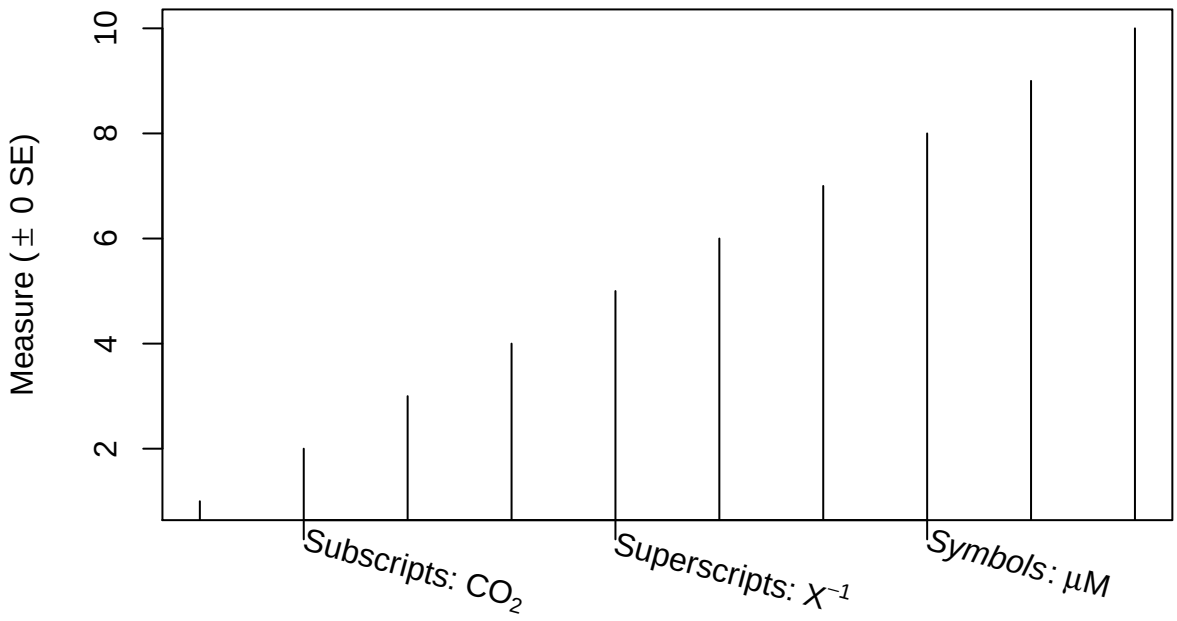
```
op <- par(mar=c(8,3.1,2,2))
plot(y ~ x, type=c('h'), xaxt = 'n', ylab=exp1, xlab="")
axis(1, at=c(2,5,8), labels=exp3, las=3) # 3 = vertical
```




```
par(op)
```

Annoyingly, for other arbitrary rotations you need to suppress the labels in `axis` and add them separately using `text`.

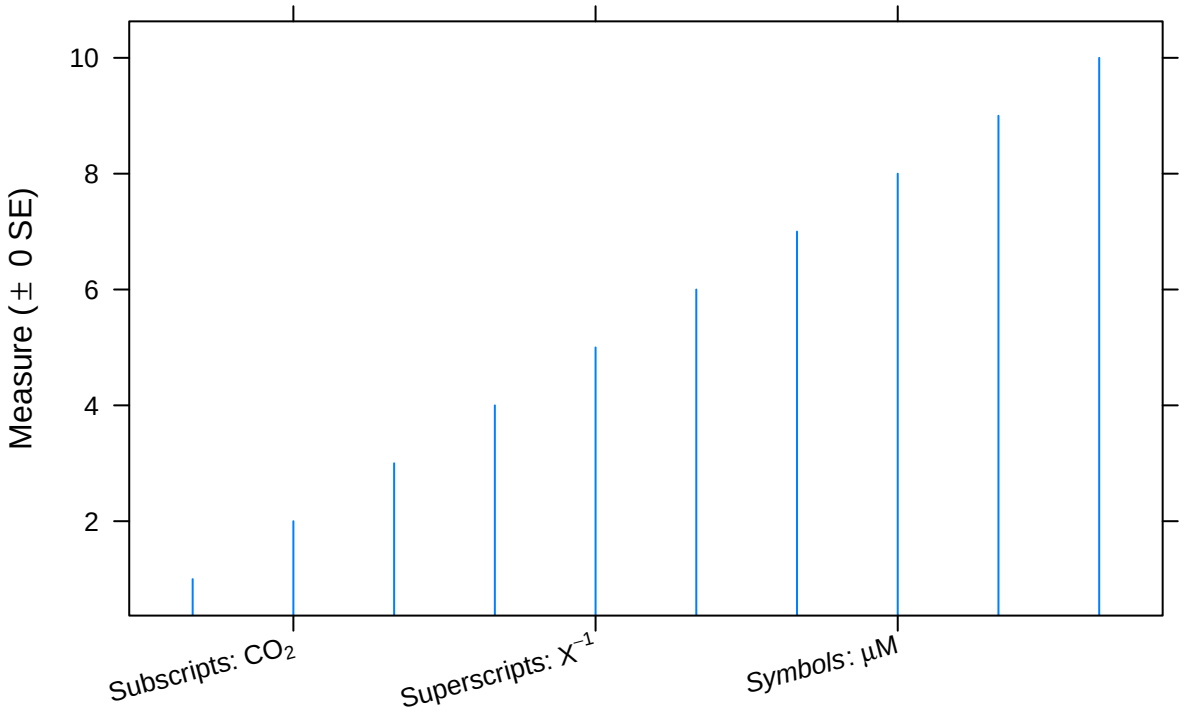
```
plot(y ~ x, type=c('h'), xaxt = 'n', ylab=exp1, xlab="")
axis(1, at=c(2,5,8), labels=FALSE)
text(c(2,5,8), par("usr")[3] - 0.4, labels = exp3, srt = -15, adj = 0, xpd = TRUE)
```




```
# xpd=T clips to the figure region, not plot region
par(op)
```

Lattice offers some relief but is still not perfect.

```
library(lattice)
xyplot(y~x, type='h', xlab="", ylab=exp1, scales=list(x=list(at=c(2,5,8), labels=exp3, rot=15)))
```



Finally, I am no expert on `plotmath`. Most things I have gleaned over the years have been on a "need to know" basis - when I need to know it, I figure it out, usually from the 'plotmat' demo.

```
demo(plotmath)
```

```
##
##
##  demo(plotmath)
##  ---- ~~~~~~~~
##
## > #  Copyright (C) 2002-2009 The R Core Team
## >
## > require(datasets)
##
## > require(grDevices); require(graphics)
##
## > ## --- "math annotation" in plots :
## >
## > ######
## > # create tables of mathematical annotation functionality
## > ######
## > make.table <- function(nr, nc) {
## +     savepar <- par(mar=rep(0, 4), pty="s")
## +     plot(c(0, nc*2 + 1), c(0, -(nr + 1)),
## +          type="n", xlab="", ylab="", axes=FALSE)
## +     savepar
```

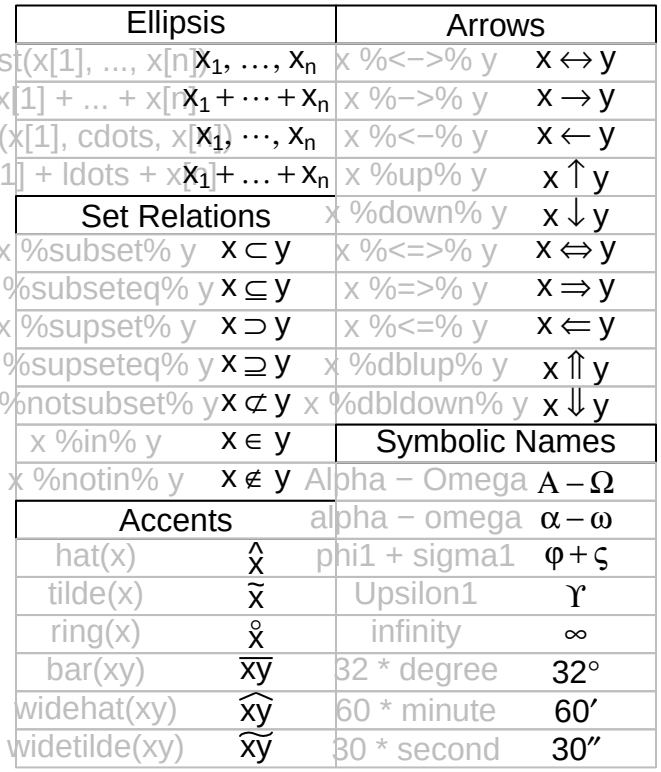
```
## + }
##
## > get.r <- function(i, nr) {
## +     i %% nr + 1
## + }
##
## > get.c <- function(i, nr) {
## +     i %/% nr + 1
## + }
##
## > draw.title.cell <- function(title, i, nr) {
## +     r <- get.r(i, nr)
## +     c <- get.c(i, nr)
## +     text(2*c - .5, -r, title)
## +     rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5))
## + }
##
## > draw.plotmath.cell <- function(expr, i, nr, string = NULL) {
## +     r <- get.r(i, nr)
## +     c <- get.c(i, nr)
## +     if (is.null(string)) {
## +         string <- deparse(expr)
## +         string <- substr(string, 12, nchar(string) - 1)
## +     }
## +     text((2*(c - 1) + 1), -r, string, col="grey")
## +     text((2*c), -r, expr, adj=c(.5,.5))
## +     rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5), border="grey")
## + }
##
## > nr <- 20
##
## > nc <- 2
##
## > oldpar <- make.table(nr, nc)
```

Arithmetic Operators		Lists	
x + y	$x + y$	list(x, y, z)	x, y, z
x − y	$x - y$	**Relations**	
x * y	xy	x == y	$x = y$
x/y	x/y	x != y	$x \neq y$
x %+−% y	$x \pm y$	x < y	$x < y$
x%/%y	$x \div y$	x <= y	$x \leq y$
x %*% y	$x \times y$	x > y	$x > y$
x %.% y	$x \cdot y$	x >= y	$x \geq y$
−x	$-x$	x %~~% y	$x \approx y$
+x	$+x$	x %=~% y	$x \cong y$
Sub/Superscripts		x %==% y	$x \equiv y$
x[i]	x_i	x %prop% y	$x \propto y$
x^2	x^2	x %~% y	$x \sim y$
Juxtaposition		**Typeface**	
x * y	xy	plain(x)	x
paste(x, y, z)	xyz	italic(x)	*x*
Radicals		bold(x)	**x**
sqrt(x)	$\sqrt{x}$	bolditalic(x)	***x***
sqrt(x, y)	$\sqrt[y]{x}$	underline(x)	$\underline{x}$

```
##
## > i <- 0
##
## > draw.title.cell("Arithmetic Operators", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x + y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x - y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x * y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x / y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %+-% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %/% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %*% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %.% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(-x), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(+x), i, nr); i <- i + 1
##
## > draw.title.cell("Sub/Superscripts", i, nr); i <- i + 1
```

```
##
## > draw.plotmath.cell(expression(x[i]), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x^2), i, nr); i <- i + 1
##
## > draw.title.cell("Juxtaposition", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x * y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(paste(x, y, z)), i, nr); i <- i + 1
##
## > draw.title.cell("Radicals", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(sqrt(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(sqrt(x, y)), i, nr); i <- i + 1
##
## > draw.title.cell("Lists", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(list(x, y, z)), i, nr); i <- i + 1
##
## > draw.title.cell("Relations", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x == y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x != y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x < y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x <= y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x > y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x >= y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %~~% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %=~% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %==% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %prop% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %~% y), i, nr); i <- i + 1
##
## > draw.title.cell("Typeface", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(plain(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(italic(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(bold(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(bolditalic(x)), i, nr); i <- i + 1
```

```
##
## > draw.plotmath.cell(expression(underline(x)), i, nr); i <- i + 1
##
## > # Need fewer, wider columns for ellipsis ...
## > nr <- 20
##
## > nc <- 2
##
## > make.table(nr, nc)
```

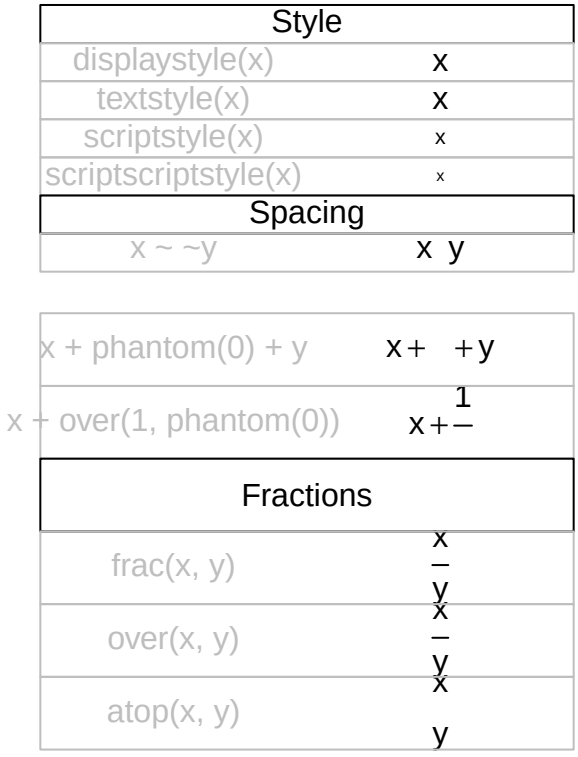



```
## $mar
## [1] 0 0 0 0
##
## $pty
## [1] "s"
##
##
## > i <- 0
##
## > draw.title.cell("Ellipsis", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(list(x[1], ..., x[n])), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x[1] + ... + x[n]), i, nr); i <- i + 1
##
```

```
## > draw.plotmath.cell(expression(list(x[1], cdots, x[n])), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x[1] + ldots + x[n]), i, nr); i <- i + 1
##
## > draw.title.cell("Set Relations", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %subset% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %subseteq% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %supset% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %supseteq% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %notsubset% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %in% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %notin% y), i, nr); i <- i + 1
##
## > draw.title.cell("Accents", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(hat(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(tilde(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(ring(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(bar(xy)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(widehat(xy)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(widetilde(xy)), i, nr); i <- i + 1
##
## > draw.title.cell("Arrows", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %<->% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %->% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %<-% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %up% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %down% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %<=>% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %=>% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %<=% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %dblup% y), i, nr); i <- i + 1
##
```

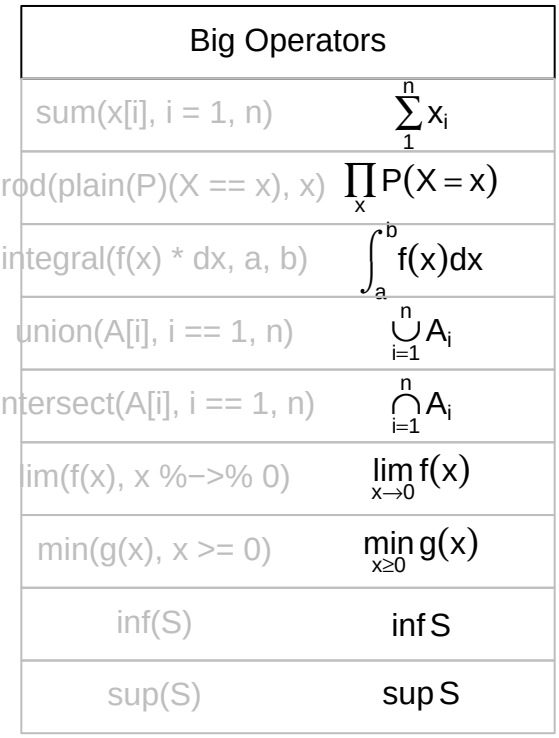
```
## > draw.plotmath.cell(expression(x %dbldown% y), i, nr); i <- i + 1
##
## > draw.title.cell("Symbolic Names", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(Alpha - Omega), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(alpha - omega), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(phi1 + sigma1), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(Upsilon1), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(infinity), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(32 * degree), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(60 * minute), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(30 * second), i, nr); i <- i + 1
##
## > # Need even fewer, wider columns for typeface and style ...
## > nr <- 20
##
## > nc <- 1
##
## > make.table(nr, nc)

## $mar
## [1] 0 0 0 0
##
## $pty
## [1] "s"
##
##
## > i <- 0
##
## > draw.title.cell("Style", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(displaystyle(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(textstyle(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(scriptstyle(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(scriptscriptstyle(x)), i, nr); i <- i + 1
##
## > draw.title.cell("Spacing", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x ~~ y), i, nr); i <- i + 1
##
## > # Need fewer, taller rows for fractions ...
## > # cheat a bit to save pages
## > par(new = TRUE)
##
```

```
## > nr <- 10
##
## > nc <- 1
##
## > make.table(nr, nc)
```



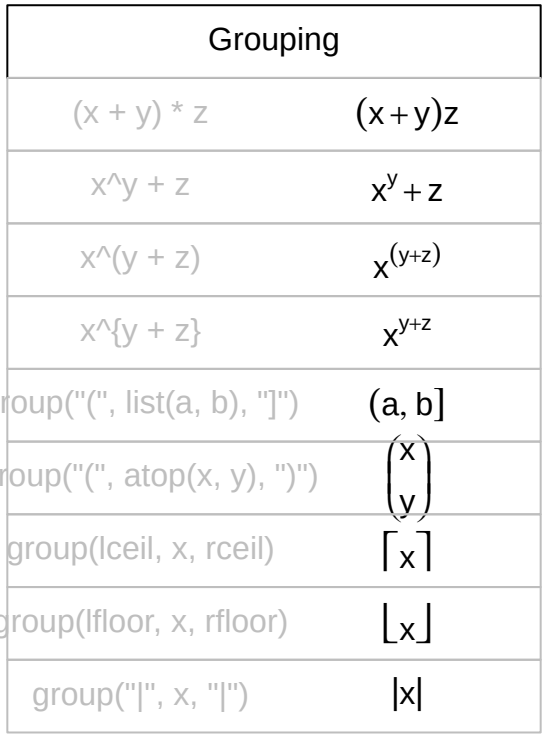

```
## $mar
## [1] 0 0 0 0
##
## $pty
## [1] "s"
##
##
## > i <- 4
##
## > draw.plotmath.cell(expression(x + phantom(0) + y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x + over(1, phantom(0))), i, nr); i <- i + 1
##
## > draw.title.cell("Fractions", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(frac(x, y)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(over(x, y)), i, nr); i <- i + 1
##
```

```
## > draw.plotmath.cell(expression(atop(x, y)), i, nr); i <- i + 1
##
## > # Need fewer, taller rows and fewer, wider columns for big operators ...
## > nr <- 10
##
## > nc <- 1
##
## > make.table(nr, nc)
```




```
## $mar
## [1] 0 0 0 0
##
## $pty
## [1] "s"
##
##
## > i <- 0
##
## > draw.title.cell("Big Operators", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(sum(x[i], i=1, n)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(prod(plain(P)(X == x), x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(integral(f(x) * dx, a, b)), i, nr); i <- i + 1
```

```
##
## > draw.plotmath.cell(expression(union(A[i], i==1, n)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(intersect(A[i], i==1, n)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(lim(f(x), x %->% 0)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(min(g(x), x >= 0)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(inf(S)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(sup(S)), i, nr); i <- i + 1
##
## > make.table(nr, nc)
```




```
## $mar
## [1] 0 0 0 0
##
## $pty
## [1] "s"
##
##
## > i <- 0
##
## > draw.title.cell("Grouping", i, nr); i <- i + 1
```

```
##
## > draw.plotmath.cell(expression((x + y)*z), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x^y + z), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x^(y + z)), i, nr); i <- i + 1
##
## > # have to do this one by hand
## > draw.plotmath.cell(expression(x^{y + z}), i, nr, string="x^{y + z}"); i <- i + 1
##
## > draw.plotmath.cell(expression(group("(", list(a, b), "]")), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(bgroup("(", atop(x, y), ")")), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(group(lceil, x, rceil)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(group(lfloor, x, rfloor)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(group("|", x, "|")), i, nr); i <- i + 1
##
## > par(oldpar)
```