

R FOR OCTAVE USERS
version 0.2
Copyright (C) 2001 Robin Hankin

=====

Permission is granted to make and distribute verbatim copies of this manual provided this permission notice is preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Free Software Foundation.

=====

This whole thing started when I couldn't figure out how to do the equivalent of the octave command "plot(sort(randn(100,10)))"---which I do quite frequently---in R. This document shows you how to do it.

The idea is to scan down until you see an octave command you wish to emulate in R. The equivalent R command is given on the right hand side (I tend to think of octave and matlab interchangeably).

I have been deliberately using octave stuff with little in the way of explanation because I'm writing for octave experts learning R, not R experts learning octave; so I'll assume that the octave commands will be understood with no further explanation.

In the lines below, the octave stuff is on the left and the equivalent R commands on the right. If the commands are too long for that, octave comes first then R. Some of the examples aren't exact matches; and where something doesn't have a vectorized equivalent I've written "nve".

It goes without saying that I would welcome comments, suggestions, improvements, etc. I *think* everything works (octave 2.1.33; R-1.5.1).

kia ora

HELP:

help -i	help.start ()
help	help(help)
help sort	help(sort)
	demo()
<matlab>	
lookfor plot	apropos('plot')
	help.search('plot')

COMPLEX NUMBERS:

3+4i	3+4i	
i	1i	%R treats "i" as a variable name
abs(3+4i)	Mod(3+4i)	%get these from help(Mod)
arg(3+4i)	Arg(3+4i)	
conj(3+4i)	Conj(3+4i)	
real(3+4i)	Re(3+4i)	
imag(3+4i)	Im(3+4i)	

VECTORS; SEQUENCES:

1:10	1:10 _or_ seq(10)
1:3:10	seq(1,10,by=3)
10:-1:1	10:1
10:-3:1	seq(from=10,to=1,by= -3)

```
linspace(1,10,7)          seq(1,10,length=7)

(1:10)+i                  1:10+1i
```

VECTORS; CONCATENATION:

```
a=[2 3 4 5];              a <- c(2,3,4,5)
a=[2 3 4 5]               (a <- c(2,3,4,5))

adash=[2 3 4 5]' ;       adash <- t(c(2,3,4,5))

[a a]                     c(a,a)
[a a*3]                   c(a,a*3)

a.*a                      a*a
a.^3                     a^3

[1:4 a]                   c(1:4,a)
```

VECTORS; CONCATENATING AND REPEATING:

```
[1:4 1:4]                rep(1:4,2)
[1:4 1:4 1:4]            rep(1:4,3)
<nve>                    rep(1:4,1:4)
<nve>                    rep(1:4,each=3)
```

VECTORS; NEGATIVE INDICES MEAN MISS THOSE ELEMENTS OUT:

```
a=1:100;                 a <- 1:100
a(2:100)                 a[-1]      %ie miss the first element.
a([1:9 11:100])         a[-10]     %ie miss the tenth element.
nve                      a[-seq(1,50,3)] %ie miss 1,4,7,...
```

VECTORS; ASSIGNMENT:

```
a(a>90)= -44;           a[a>90] <- -44
```

VECTORS; MAX AND MIN:

```
a=randn(1,4);            a <- rnorm(4)
b=randn(1,4);            b <- rnorm(4)
max(a,b)                 pmax(a,b)      %mnemonic: pairwise max.
max([a' b'])             cbind(max(a),max(b))
max([a b])               max(a,b)
[m i] = max(a)           m <- max(a) ; i <- which.max(a)
```

"min" is analogous in R and octave.

VECTORS: RANKS

```
ranks(rnorm(8,1))        rank(rnorm(8))
ranks(rnorm(randn(5,6)))  apply(matrix(rnorm(30),6),2,rank)
```

MATRICES:

MATRICES: RBIND AND CBIND:

```
[1:4 ; 1:4]              rbind(1:4,1:4)
[1:4 ; 1:4]'              cbind(1:4,1:4) _or_ t(rbind(1:4,1:4))

[2 3 4 5]                c(2,3,4,5)
[2 3;4 5]                 rbind(c(2,3),c(4,5)) %rbind() binds rows while
[2 3;4 5]'                cbind(c(2,3),c(4,5)) %cbind() binds cols.

a=[5 6] ;                a <- c(5,6)
b=[a a;a a]              b <- rbind(c(a,a),c(a,a)) %see repmat below

[1:3 1:3 1:3 ; 1:9]      rbind(1:3, 1:9)
[1:3 1:3 1:3 ; 1:9]'     cbind(1:3, 1:9)
nve                      rbind(1:3, 1:8)
```

MATRICES; MATRIX AND ARRAY:

```
ones(4,7)          matrix(1,4,7)  _or_  array(1,c(4,7))
ones(4,7)*9        matrix(9,4,7)  _or_  array(9,c(4,7))
```

MATRICES; DIAGONAL

```
eye(3)             diag(1,3)
diag([4 5 6])      diag(c(4,5,6))
diag(1:10,3)        %I don't think there is a drop-in
                    %replacement but it's easy to
                    %write a little function:
```

```
di <- function(vec,n=0) {
  l <- length(vec)
  if (n >= 0) {
    return (cbind(matrix(0,l+n,n),diag(vec,l+n,1)))
  } else {
    return (t(di(vec, -n)))
  }
}
```

then di(1:10,3) works as per Octave.

MATRICES; MATRIX AND ARRAY FUNCTIONS

```
reshape(1:6,2,3)    matrix(1:6,nrow=2)  _or_  array(1:6,c(2,3))
reshape(1:6,3,2)    matrix(1:6,ncol=2)  _or_  array(1:6,c(3,2))
```

```
reshape(1:6,3,2)'    matrix(1:6,nrow=2,byrow=T)
```

```
[reshape(1:6,3,2) reshape(1:6,3,2)]
cbind( matrix(1:6,ncol=2), matrix(1:6,ncol=2))
```

```
a=reshape(1:36,6,6)    a <- matrix(1:36,c(6,6))
rem(a,5)               a %% 5
a[rem(a,5)==1]=-999    a[a%%5==1] <- -999

a(:)                   as.vector(a)
```

MATRICES; ACCESSING ELEMENTS:

```
a=reshape(1:12,3,4);  a <- matrix(1:12,nrow=3)
a(2,3)               a[2,3]
a(2,:)              a[2, ]          %spaces optional---just miss out
                               %the colon!
a(2:3,:)            a[-1,]          %Negative indices mean
a(:, [1 3 4])       a[, -2]        %leave them out.

a(:,1)              a[ ,1]
a(:,2:4)            a[ , -1]

a([1 3],[1 2 4]);nve  a[-2,-3]    %Negative indices still work in pairs
```

ASSIGNMENT:

```
a(:,1) = 99          a[ ,1] <- 99
a(:,1) = [99 98 97]' a[ ,1] <- c(99,98,97)
```

MATRICES: TRANSPOSE AND CONJ:

```
a'                  Conj(t(a))    %this can't be right...
a.'                 t(a)
```

MATRICES: R EQUIVALENTS TO "SUM" AND "CUMSUM" ETC

```
a=ones(6,7)         a <- matrix(1,6,7)
sum(a)              apply(a,2,sum)
sum(a')             apply(a,1,sum)
sum(sum(a))         sum(a)
```

```

cumsum(a)          apply(a,2,cumsum)
cumsum(a')         apply(a,1,cumsum)

a=rand(3,4);       a <- matrix(runif(12),c(3,4))
sort(a(:))         sort(a)
sort(a)            apply(a,2,sort)
sort(a')           apply(a,1,sort)
cummax(a)          apply(a,2,cummax)

MATRICES; MAX AND MIN:

a=randn(100,4)     a <- matrix(rnorm(400),4)
max(a)             apply(a,1,max)
[v i] = max(a)     v <- apply(a,1,max) ; i <- apply(a,1,which.max)

b=randn(4,4);      b <-matrix(rnorm(16),4)
c=randn(4,4);      c <-matrix(rnorm(16),4)
max(b,c)           %NB cf max(b,c) ~ max(rnorm(32))

OTHER MATRIX MANIPULATION

a=rand(3,4);       a <- matrix(runif(12),c(3,4))
fliplr(a)          a[,4:1] (improvements anyone?)
flipud(a)          a[3:1,]

rot90(a)           %no builtin but it's easy to write a little function:
rot90 <- function(a,n=1) {
  n <- n %% 4
  if (n > 0) {
    return (rot90( t(a)[nrow(a):1,,n-1] )
  } else {
    return (a)
  }
}

a=reshape(1:9,3,3) a <- matrix(1:9,3)
vec(a)             as.vector(a)
vech(a)            a[row(a) <= col(a)]

EQUIVALENTS TO "SIZE" ETC

size(a)            dim(a)

MATRICES: MATRIX- AND ELEMENTWISE- MULTIPLICATION

a=reshape(1:6,2,3); a <- matrix(1:6,2,3)
b=reshape(1:6,3,2); b <- matrix(1:6,3,2)
c=reshape(1:4,2,2); c <- matrix(1:4,2,2)
v=[10 11];         v <- c(10,11)
w=[100 101 102];   w <- c(100,101,102)
x=[4 5]';          x <- t(c(4,5))

a*b               a %*% b
v*a              v %*% a
a*w'             a %*% w
b*v'            b %*% v
v*x             x %*% v _or_ v %*% t(x)
x*v             t(x) %*% v

v*a*w'          v %*% a %*% w

v .* x'         v*x _or_ x*v
a .* [w ;w]     w * a
a .* [x x x]    a * t(rbind(x,x,x)) (er, any improvements anyone?)

%NB: R treats v and w as _column_ vectors by default (if there is a
choice), eg

v*c             v %*% c
c*v'           c %*% v

```

MESHGRID:

```
[x y]=meshgrid(1:5,10:12);
```

R has no builtin meshgrid() function but you can write one:

```
meshgrid <- function(a,b) {  
  list(  
    x=outer(b*0,a,FUN="+"),  
    y=outer(b,a*0,FUN="+")  
  )  
}
```

```
R> meshgrid(1:5,10:12)
```

octave:

```
meshgrid(1:3,1:8)' .^ meshgrid(1:8,1:3)  
_or_ [x y]=meshgrid(1:8,1:3); x.^y
```

R:

```
outer(1:3,1:8,"^") _or_ t(meshgrid(1:3,1:8)$x^(1:8)) <cringe>
```

REPMAT

I don't think octave has an equivalent to matlab's repmat; and neither does R. Instead, R uses the much more flexible and wonderful kronecker(). With this, repmat could be:

```
repmat <- function(a,n,m) {kronecker(matrix(1,n,m),a)}
```

then

<Matlab>

```
a=[1 2 ; 3 4];  
repmat(a,2,3)
```

<R>

```
a <- matrix(1:4,2,byrow=T)  
repmat(a,2,3)
```

FIND:

```
find(1:10 > 5.5)
```

```
which(1:10 > 5.5)
```

```
a=diag([4 5 6])
```

```
a <- diag(c(4,5,6))
```

```
find(a)
```

```
which(a != 0) %which() needs a Boolean argument.
```

```
[i j]= find(a)
```

```
which(a != 0,arr.ind=T)
```

```
[i j k]=find(a)
```

```
ij <- which(a != 0,arr.ind=T); k <- a[ij]
```

READING FROM A FILE:

```
localhost:~% cat foo.txt
```

```
1 2
```

```
3 4
```

```
load foo.txt
```

```
f <- read.table("~/foo.txt")
```

```
f <- as.matrix(f)
```

WRITING TO A FILE:

```
save -ascii bar.txt f
```

```
write(f,file="bar.txt")
```

POSTSCRIPT OUTPUT

```
plot(1:10)
```

```
print -deps foo.eps
```

<matlab>

```

gset output "foo.eps"
gset terminal postscript eps
plot(1:10)                                <octave>

postscript(file="foo.eps")
plot(1:10)
dev.off ()                                <R>

EVAL

string="a=234";                          string <- "a <- 234"
eval(string)                              eval(parse(text=string))

GENERATE RANDOM NUMBERS FROM DIFFERENT DISTRIBUTIONS:

UNIFORM:

rand(10,1)                                runif(10)
2+5*rand(10,1)                            runif(10,min=2,max=7) _or_ runif(10,2,7)
rand(10)                                  matrix(runif(100),10)

NORMAL:

randn(10,1)                               rnorm(10)
2+5*randn(10,1)                           rnorm(10,2,5)
rand(10)                                   matrix(rnorm(100),10)

BETA:

hist(beta_rnd(4,2,1000,1))
hist(rbeta(1000,shape1=4,shape2=10)) _or_ hist(rbeta(1000,4,10))

PLOTING IID RANDOM VARIABLES:

hist(mean(binomial_rnd(10,0.4,100,500)))
hist(apply(matrix(rbinom(50000,10,0.4),nr=100),2,mean))

a=randn(100,10);                          a <- matrix(rnorm(1000),nr=10)
plot(sort(a))                             matplot(apply(a,1,sort),type="l")
plot(sort(mean(a)))                       plot(sort(apply(a,1,mean)))

LOOPS; FOR:

for i=1:5 ; disp(i) ; endfor
for(i in 1:5) print(i)

MULTILINE FOR STATEMENTS:

for i=1:5
    disp(i)
    disp(i+100)
endfor

for(i in 1:5)
{
    print(i)
    print(i+100)
}

LOOPS; WHILE:

i=0;
while i < 10
    disp(i*i)
    i++ ;
endwhile

i <- 0
while (i < 10) {

```

```

    print(i*i)
    i <- i+1
}

```

CONDITIONALS; IF:

```
if 1>0 a=100; endif      if (1>0) a <- 100
```

SWITCH:

```

switch i              a <- switch(as.character(i),"1"=66, "5"=77, -99)
  case 1
    a=66;
  case 5
    a=77;
  otherwise
    a=-99;
endswitch

```

POLYNOMIALS

ROOT FINDING:

```
roots([1 2 1])          polyroot(c(1,2,1))
```

```
polyval([1 2 1 2],1:10)
```

there's no direct equivalent of this in R but it's quite simple to write one:

```

polyval <- function(c,x) {
  n <- length(c)
  y <- x*0+c[1];
  for (i in 2:n) {
    y <- c[i] +x*y
  }
  y
}

```

so then

```
R> polyval(c(1,2,1,2),1:10)    should work.
```

SET THEORY

I'm not sure whether this lot works in Matlab or just Octave.

```

a = create_set([1 2 2 99 2 ])
b = create_set([2 3 4 ])
intersection(a,b)
union(a,b)
complement(a,b)
any(a == 2)

```

```

a <- sort(unique(c(1,2,2,99,2)))
b <- sort(unique(c(2,3,4)))
intersect(a,b)              %note that intersect() etc call
union(a,b)                  %unique() directly. So the four
setdiff(b,a)                %examples here would work with
is.element(2,a)              %a <- c(1,2,2,99,2).

```

DEBUGGING

```

keyboard          debug("function_name")
ans                .Last.value
disp(44)           print(44)

```

DEFINITION OF FUNCTIONS:

```

function out=h(n); out=1./(meshgrid(1:n)+ meshgrid(1:n)' -1) ;endfunction
h <- function (n) 1/(col(diag(n))+row(diag(n))-1)
_or_
h <- function (n) { 1/outer(1:n,0:(n-1),"+") }

```

MISC PLOTTING:

```
a=rand(10);          a <- array(runif(100),c(10,10))
help plot            help (plot) _and_ methods(plot)

plot(a)              matplot(a,type="l",lty=1)

plot(a,'r')          matplot(a,type="l",lty=1,col="red")
plot(a,'x')          matplot(a,pch=4)
plot(a,'--')         matplot(a,type="l",lty=2)

plot(a,'x-')         matplot(a,pch=4,type="b",lty=1)
plot(a,'x--')        matplot(a,pch=4,type="b",lty=2)

semilogy(a)          matplot(a,type="l",lty=1,log="y")
semilogx(a)          matplot(a,type="l",lty=1,log="x")
loglog(a)            matplot(a,type="l",lty=1,log="xy")

plot(1:10,'r')        plot(1:10,col="red",type="l")
hold on              matplot(10:1,col="blue",type="l",add=T)
plot(10:-1:1,'b')

grid                  grid ()

<Matlab>
plot([1:10 10:-1:1])
axis equal

<Octave>
plot([1:10 10:-1:1])
axis('equal')
replot

<R>
plot(c(1:10,10:1),asp=1)
```

REORDERING VECTORS:

```
x=randn(1,10); y=randn(1,10);
plot(x,y)
[x_sort index]=sort(x);
plot(x_sort,y(index))

x <- rnorm(10) ; y <- rnorm(10)
plot(x,y,type="l")
plot(sort(x),y[order(x)],type="l")
```

STRAIGHT LINE FITTING:

```
a=randn(1,10); x=1:10;
plot(x,a,'o',x,polyval(polyfit(x,a,1),x) , '-')

a <- rnorm(10)      # generate the data
x <- 1:10           # create the x-axis
z <- lm(a~x)        # z becomes a linear model of "a" depending on "x"
z                  # see that z is just an intercept and slope
plot(a)             # er, plot(a)
abline(z)           # plot the best-fit line above.
```

AXES AND TITLES:

```
plot(1:10);xlabel("foo");ylabel("bar");title("FooBar")
matplot(1:10,type="l",xlab="foo",ylab="bar",main="FooBar",lty=1)

hist(randn(1000,1))      hist(rnorm(1000))
hist(randn(1000,1), -4:4) hist(rnorm(1000), breaks= -4:4)
????????????            hist(rnorm(1000), breaks= c(seq(-5,0,0.25),seq(0.5,5,0.5)),freq=F)
```


CONTOUR PLOTS:

```
a=randn(10);                a <- matrix(rnorm(100),nr=10)

contour(a)                   contour(a)
contour(a,77)                contour(a,nlevels=77)
????????????????           filled.contour(a)
```

MESH PLOTS:

```
mesh(rand(10))
persp(matrix(runif(100),10),theta=30,phi=30,d=1e9)
```

FILES AND OS

```
system("ls")                 system("ls")
pwd                           getwd()
cd                             setwd()
```

READING OCTAVE FILES IN R

(thanks to Stephen Eglen for making this available).

```
read.oct.file <- function(filename) {

  ## Read in an Octave ASCII data file (as created by "save -ascii" in
  ## Octave 2.x) and return a list of the objects (matrix, string
  ## array or scalar) returned. Any "_" in the octave name is
  ## converted into "." to avoid confusion in R with __.

  ## e.g. create two variables in Octave
  ## octave> ident_mat = eye(3);
  ## octave> twopi = 2 *pi;
  ## octave> save -ascii 'octfile.dat'

  ## then load this file into R:
  ## > o <- read.oct.file("octfile.dat")
  ## > o
  ## $twopi
  ## [1] 6.283185
  ## $ident.mat
  ##      [,1] [,2] [,3]
  ## [1,]    1    0    0
  ## [2,]    0    1    0
  ## [3,]    0    0    1

  read.oct.matrix <- function(con) {
    ## Helper function: read in a matrix.
    nrows <- as.numeric(substring(readLines(con,1), 9))
    ncols <- as.numeric(substring(readLines(con,1), 12))
    data <- scan(con, nlines=nrows, quiet=T)
    d <- matrix(data, nrow=nrows, ncol=ncols, byrow=T)
    d
  }

  read.oct.matrix.i <- function(con) {
    ## Helper function: read in a matrix.
    nrows <- as.numeric(substring(readLines(con,1), 9))
    ncols <- as.numeric(substring(readLines(con,1), 12))
    data <- readLines(con, n=nrows)

    c1 <- paste(data,sep=" ",collapse=" ")
    c1 <- gsub("\\\\(", " ", c1)
    c1 <- gsub("\\\\)", " ", c1)
    c1 <- gsub(",", " ", c1)
    s <- unlist(strsplit(c1, " "))
    nums <- as.numeric(s[-1])           #remove initial space.
    reals <- nums[ seq(from=1, by=2, length=length(nums)/2)]
    ims <- nums[ seq(from=2, by=2, length=length(nums)/2)]
    complexes <- complex(real=reals, imaginary=ims)
    d <- matrix(data=complexes, nrow=nrows, ncol=ncols, byrow=T)
    d
  }
}
```

```

}

read.oct.stringarr <- function(con) {
  ## Helper function: read in a string array.
  elements <- as.numeric(substring(readLines(con,1), 13))
  d <- readLines(con, n=elements*2)
  ## remove the odd-numbered lines, they just store "length".
  d <- d[ seq(from=2, by=2, length=length(d)/2)]
}

read.oct.scalar <- function(con) {
  ## Helper function: read in a scalar.
  d <- as.numeric(scan(con, nlines=1, quiet=T))
  d
}

read.oct.scalar.i <- function(con) {
  ## Helper function: read in a complex scalar.
  d <- readLines(con, n=1)
  ## remove parens then split.
  str <- gsub("\\(", "", d)
  str <- gsub(")", "", str)
  nums <- as.numeric(unlist(strsplit(str,",")))
  stopifnot(length(nums)==2)
  complex(real=nums[1], imag=nums[2])
}

read.oct.range <- function(con) {
  ## Helper function: read in a range.
  d <- readLines(con, n=1) #skip over "# base, limit, increment"
  d <- as.numeric(scan(con, nlines=1, quiet=T))
  stopifnot(length(d)==3)
  d <- seq(from=d[1], to=d[2], by=d[3])
}

zz <- file(filename, "r")
readLines(zz, n=1) #skip over the header line.

## Build up a return list of items -- separately store the return values
## and the names.
ret.list <- list()
ret.names <- list()
reading <- TRUE
while(reading) {
  name.line <- readLines(zz, 1, ok=TRUE)
  if (length(name.line) == 0) {
    reading <- FALSE
  }
  else {
    name <- substring(name.line, 9)
    name <- gsub("_", ".", name) #octave vars often have _ in them.
    type <- substring(readLines(zz,1), 9)
    res <- switch(type,
      "matrix" = read.oct.matrix(zz),
      "scalar" = read.oct.scalar(zz),
      "string array" = read.oct.stringarr(zz),
      "range" = read.oct.range(zz),
      "complex scalar" = read.oct.scalar.i(zz),
      "complex matrix" = read.oct.matrix.i(zz),
      stop(paste("data type",type,"not recognised")))
    ##assign(name, res, env = .GlobalEnv)
    ret.list <- c( ret.list, list(res))
    ret.names <- c( ret.names, name)
  }
}
close(zz)
names(ret.list) <- ret.names
ret.list
}

```

--

Robin Hankin, Lecturer,
School of Environmental and Marine Science
Private Bag 92019 Auckland
New Zealand

r.hankin_AT_auckland.ac.nz [edit in the obvious way; spam precaution]
tel 0064-9-373-7599 x6820; FAX 0064-9-373-7042