Psychology 350: Special Topics
An introduction to R for psychological research
Adventures in graphics

William Revelle
Northwestern University
Evanston, Illinois USA

https://personality-project.org/courses/350

NORTHWESTERN
UNIVERSITY

May, 2024

Outline

Core graphics

1. Graphic displays are among the most useful tools in R. Some of the basic graphic tools have been used in the 'psych' package for displays of factor/cluster structures, models of linear regressions, etc. These all take advantage of some basic graphic tools.
2. The most basic function is 'plot' which creates a graphic window with certain parameters (arguments).

```
plot {base}        R Documentation
Generic X-Y Plotting

Description

Generic function for plotting of R objects.

For simple scatter plots, plot.default will be used.
However, there are plot methods for many R objects, including functions, data.frames,
   density objects, etc.
 Use methods(plot) and the documentation for these.
 Most of these methods are implemented using traditional graphics (the graphics package),
          but this is not mandatory.

For more details about graphical parameter arguments used by traditional graphics, see par.

Usage
```

the plot function

```
Arguments to be passed to methods, such as graphical parameters (see par).
  Many methods will accept the following arguments:

type
what type of plot should be drawn. Possible types are

"p" for points,

"l" for lines,

"b" for both,

"c" for the lines part alone of "b",

"o" for both `overplotted',

"h" for `histogram' like (or `high-density') vertical lines,

"s" for stair steps,

"S" for other steps, see `Details' below,

"n" for no plotting.

All other types give a warning or an error; using, e.g., type = "punkte" being equivalent
      to type = "p" for S compatibility.
      Note that some methods, e.g. plot.factor, do not accept this.
```

Other parameters passed to plot

xlim	The size of the x axis
ylim	The size of the y axis
xlab	The label for the x axis
ylab	The label for the y axis
main	A title for the figure
axes	Show the axes (defaults to TRUE)
frame.plot	Show the frame (defaults to TRUE)
cex	The size of the characters
pch	Which plot character to use for data points

The text function

```
Add Text to a Plot

Description

text draws the strings given in the vector labels at the coordinates given by x and y.
          y may be missing since xy.coords(x, y) is used for construction of the coordinates.

Usage

text(x, ...)

## Default S3 method:
 text(x, y = NULL, labels = seq_along(x$x), adj = NULL,
     pos = NULL, offset = 0.5, vfont = NULL,
     cex = 1, col = NULL, font = NULL, ...)
Arguments

x, y
numeric vectors of coordinates where the text labels should be written.
                 If the length of x and y differs, the shorter one is recycled.

labels
a character vector or expression specifying the text to be written.
   An attempt is made to coerce other language objects (names and calls) to expressions,
    and vectors and other classed objects to character vectors by as.character.
    If labels is longer than x and y, the coordinates are recycled to the length of labels.

adj            one or two values in [0,1]
[0,1] which specify the x (and optionally y) adjustment (`justification') of the labels,
    with 0 for left/bottom, 1 for right/top, and 0.5 for centered. On most devices values outs

pos        a position specifier for the text. If specified this overrides any adj value given
          Values of 1, 2, 3 and 4, respectively indicate positions below, to the left of,
```

Two basic graphic formats

1. Diagrams (e.g. fa.diagram, iclust.diagram, lm.diagram, etc) for showing structural relations..
2. Plots (e.g. pairs.panels, error.bars, scatter.hist) for displaying data.

Psych graphics: diagrams

fa.diagram	Show factor loadings and latent variables
iclust.diagram	Show a cluster solution
lmDiagram	A linear model
omega	Hierarchical factoring
bassAckward	A multi factor plot
diagram	Calls the appropriate diagram function given the class of the object.

From the help for diagram

1. The diagram function calls fa.diagram, omega.diagram, ICLUST.diagram, lavaan.diagram or bassAckward.diagram depending upon the class of the fit input. See those functions for particular parameter values.
2. The remaining functions are the graphic primitives used by fa.diagram, structure.diagram, omega.diagram, ICLUST.diagram and het.diagram
3. They create rectangles, ellipses or triangles surrounding text, connect them to straight or curved arrows, and can draw an arrow from and to the same rectangle.
4. To speed up the plotting, dia.rect and dia.arrow can suppress the actual drawing and return the locations and values to plot. These values can then be directly called by text or rect with matrix input. This leads to an impressive increase in speed when doing many variables.
5. The functions multi.rect, multi.self, multi.arrow and multi.curved.arrow will take the saved output from the

The primitives

1. Each shape (ellipse, rectangle or triangle) has a left, right, top and bottom and center coordinate that may be used to connect the arrows.
2. Curves are double-headed arrows. By default they go from one location to another and curve either left or right (if going up or down) or up or down (going left to right). The direction of the curve may be set by dir="up" for left right curvature.
3. The helper functions were developed to get around the infelicities associated with trying to install Rgraphviz and graphviz.
4. These functions form the core of fa.diagram,het.diagram.
5. Better documentation will be added as these functions get improved. Currently the helper functions are just a work around for Rgraphviz.
6. dia.cone draws a cone with (optionally) arrows as sides and centers to show the problem of factor indeterminacy.

Use these basics with some simple functions

Must initiate plot first.

```
dia.rect(x, y = NULL, labels = NULL,  cex = 1,  xlim = c(0, 1), ylim = c(0, 1),
    draw=TRUE, ...)

dia.ellipse(x, y = NULL, labels = NULL, cex=1,e.size=.05, xlim=c(0,1),
       ylim=c(0,1),draw=TRUE,  ...)

dia.triangle(x, y = NULL, labels =NULL,  cex = 1, xlim=c(0,1),ylim=c(0,1),...)

dia.ellipse1(x,y,e.size=.05,xlim=c(0,1),ylim=c(0,1),draw=TRUE,...)

dia.shape(x, y = NULL, labels = NULL, cex = 1,
         e.size=.05, xlim=c(0,1), ylim=c(0,1), shape=1, ...)

dia.arrow(from,to,labels=NULL,scale=1,cex=1,adj=2,both=FALSE,pos=NULL,l.cex,
        gap.size,draw=TRUE,col="black",lty="solid",...)

dia.curve(from,to,labels=NULL,scale=1,...)

dia.curved.arrow(from,to,labels=NULL,scale=1,both=FALSE,dir=NULL,draw=TRUE,...)

dia.self(location,labels=NULL,scale=.8,side=2,draw=TRUE,...)

dia.cone(x=0, y=-2, theta=45, arrow=TRUE,curves=TRUE,add=FALSE,labels=NULL,
      xlim = c(-1, 1), ylim=c(-1,1),... )
```

Using the dia. functions

1. The five shape functions draw a particular shape at a specified x, y location. They each return top, bottom, left and right of the shape they draw.
2. The four connection functions (arrow, curve, curved arrow, self) draw lines between different shapes (from and to) using the top, bottom, etc values of the shape functions.
3. Each function has a draw option (defaults to TRUE) to either draw the shape or just return the information for later drawing (drawing several shapes at a time is faster).

dia.shape just calls dia.rect, dia.ellipse, dia.triangle

```
"dia.shape" <-
function (x, y = NULL, labels = NULL, cex = 1, e.size = 0.05,
    xlim = c(0, 1), ylim = c(0, 1), shape = 1, ...)
{
    if (shape == 1) {
        dia.rect(x, y = NULL, labels = NULL, cex = 1, xlim = c(0,
            1), ylim = c(0, 1), ...)
    }
    if (shape == 2) {
        dia.ellipse(x, y = NULL, labels = NULL, cex = 1, e.size = 0.05,
            xlim = c(0, 1), ylim = c(0, 1), ...)
    }
    if (shape == 3) {
        dia.triangle(x, y = NULL, labels = NULL, cex = 1, xlim = c(0,
            1), ylim = c(0, 1), ...)
    }
}
```

Examine dia.rect

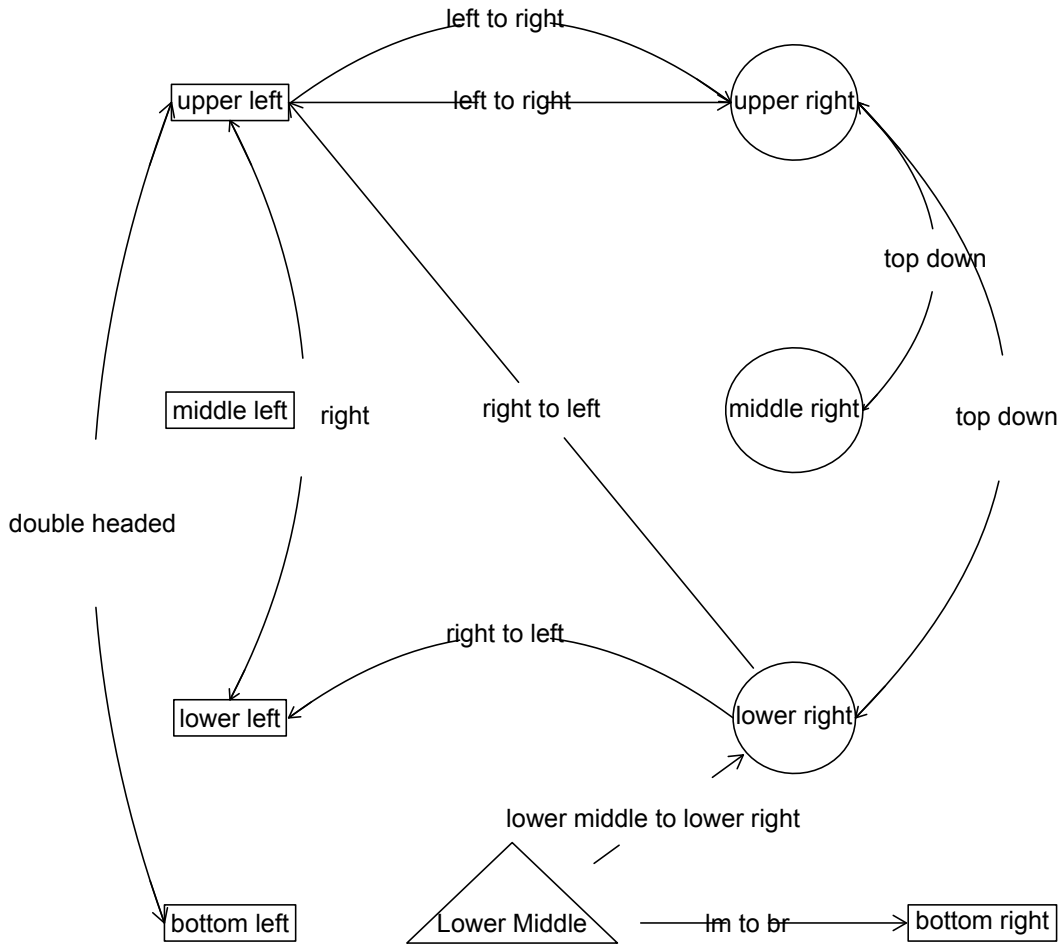
```
" dia.rect" <-
function (x, y = NULL, labels = NULL, cex = 1, xlim = c(0, 1),
    ylim = c(0, 1), draw = TRUE, ...)
{
    if (draw)
        text(x = x, y = y, labels = labels, cex = cex, ...)
    xrange = (xlim[2] - xlim[1])
    yrange = (ylim[2] - ylim[1])
    xs <- 0.1 * xrange
    ys <- 0.1 * yrange
    len <- pmax(strwidth(labels, units = "user", cex = cex, ...),
        strwidth("abc", units = "user", cex = cex, ...))/1.8
    vert <- pmax(strheight(labels, units = "user", cex = cex,
        ...), strheight("ABC", units = "user", cex = cex, ...))/1
    if (draw)
        rect(x - len, y - vert, x + len, y + vert)
    left <- c(x - len, y)
    right <- c(x + len, y)
    top <- c(x, y + vert)
    bottom <- c(x, y - vert)
    radius <- sqrt(len^2 + vert^2)
    dia.rect <- list(left = left, right = right, top = top, bottom = bottom,
        center = c(x, y), radius = radius)
}
```

Plotting with the diagram functions

R code

```
#first set up the margins and initiate the plot
xlim=c(-2,10);   ylim=c(0,10)
plot(NA,xlim=xlim,ylim=ylim,main="Demonstration of diagram functions",
axes=FALSE,xlab="",ylab="")
#place and name the objects
ul <- dia.rect(1,9,labels="upper left",xlim=xlim,ylim=ylim)
ml <- dia.rect(1,6,"middle left",xlim=xlim,ylim=ylim)
ll <- dia.rect(1,3,labels="lower left",xlim=xlim,ylim=ylim)
bl <- dia.rect(1,1,"bottom left",xlim=xlim,ylim=ylim)
lr <- dia.ellipse(7,3,"lower right",xlim=xlim,ylim=ylim,e.size=.07)
ur <- dia.ellipse(7,9,"upper right",xlim=xlim,ylim=ylim,e.size=.07)
mr <- dia.ellipse(7,6,"middle right",xlim=xlim,ylim=ylim,e.size=.07)
lm <- dia.triangle(4,1,"Lower Middle",xlim=xlim,ylim=ylim)
br <- dia.rect(9,1,"bottom right",xlim=xlim,ylim=ylim)
#connect them
dia.curve(from=ul$left,to=bl$left,"double headed",scale=-1)
dia.arrow(from=lr,to=ul$right,labels="right to left")
dia.arrow(from=ul,to=ur,labels="left to right")
dia.curved.arrow(from=lr,to=ll,labels ="right to left")
dia.curved.arrow(to=ur,from=ul,labels ="left to right")
dia.curve(ll$top,ul$bottom,"right",pos=4)  #for rectangles, specify wh
dia.arrow(from=lm, to= lr, "lower middle to lower right")
dia.curve(ur$right,lr$right,"top down",scale =2) #scale is size and cu
dia.curve(ur$right,mr$right,"top down",scale =1) #scale is size and cu
```

Demonstration of diagram functions

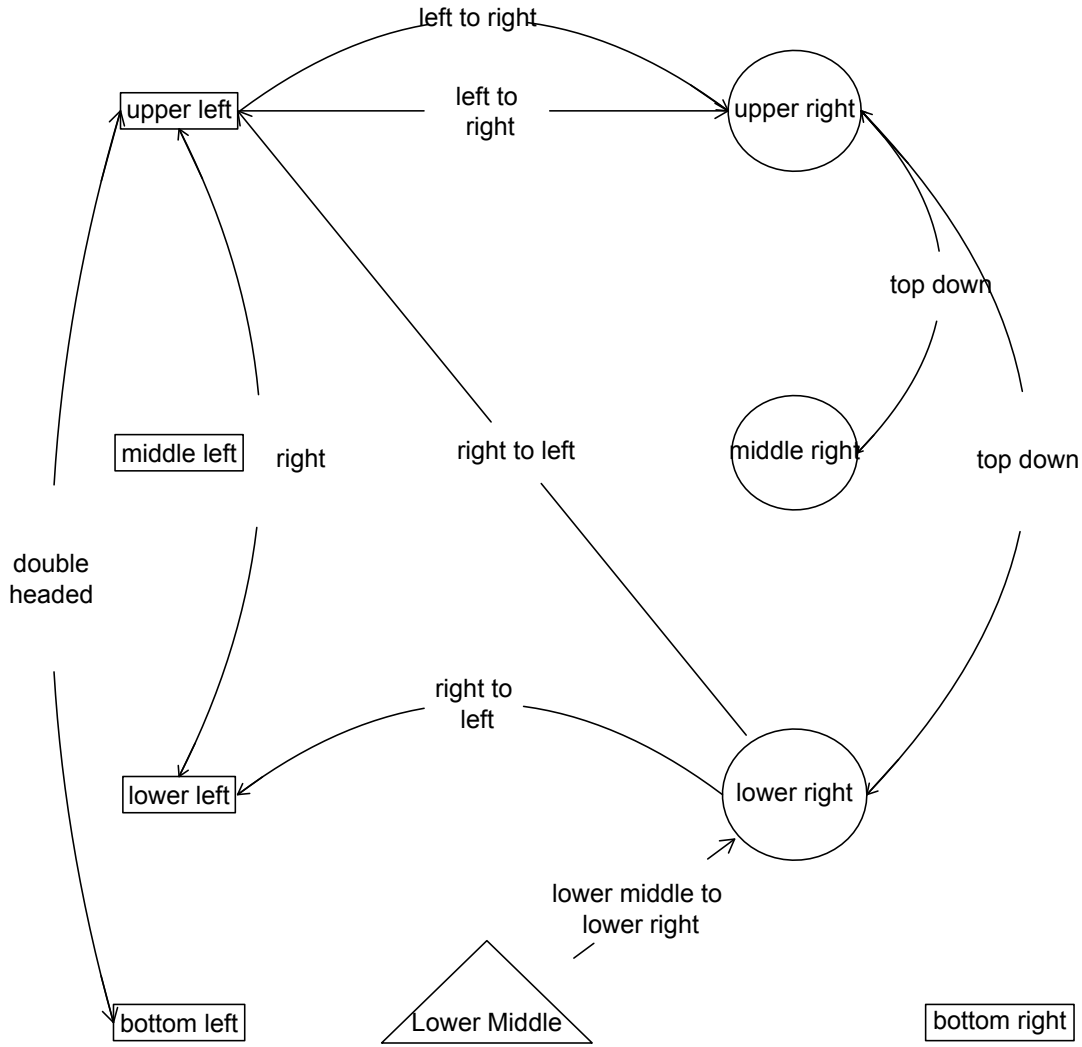


Subtle changes in calling parameters to change the size

R code

```
#first set up the margins and initiate the plot
xlim=c(-1,10);      ylim=c(1,10)
plot(NA,xlim=xlim,ylim=ylim,main="Demonstration of diagram functions",
      axes=FALSE,xlab="",ylab="")
#place and name the objects
ul <- dia.rect(1,9,labels="upper left",xlim=xlim,ylim=ylim)
ml <- dia.rect(1,6,"middle left",xlim=xlim,ylim=ylim)
ll <- dia.rect(1,3,labels="lower left",xlim=xlim,ylim=ylim)
bl <- dia.rect(1,1,"bottom left",xlim=xlim,ylim=ylim)
lr <- dia.ellipse(7,3,"lower right",xlim=xlim,ylim=ylim,e.size=.09)
ur <- dia.ellipse(7,9,"upper right",xlim=xlim,ylim=ylim,e.size=.08)
mr <- dia.ellipse(7,6,"middle right",xlim=xlim,ylim=ylim,e.size=.07)
lm <- dia.triangle(4,1,"Lower Middle",xlim=xlim,ylim=ylim)
br <- dia.rect(9,1,"bottom right",xlim=xlim,ylim=ylim)
#now connect the shapes
dia.curve(from=ul$left,to=bl$left,"double\nheaded",scale=-.8)
dia.arrow(from=lr,to=ul$right,labels="right to left")
dia.arrow(from=ul,to=ur,labels="left to\n right",gap.size=.25)
dia.curved.arrow(from=lr,to=ll,labels ="right to\n left")
dia.curved.arrow(to=ur,from=ul,labels ="left to right")
dia.curve(ll$top,ul$bottom,"right",pos=4)  #for rectangles, specify wh
dia.arrow(from=lm, to= lr, "lower middle to \nlower right")#use \n fo
dia.curve(ur$right,lr$right,"top down",scale =2) #scale is size and cu
dia.curve(ur$right,mr$right,"top down",scale =1) #scale is size and cu
```

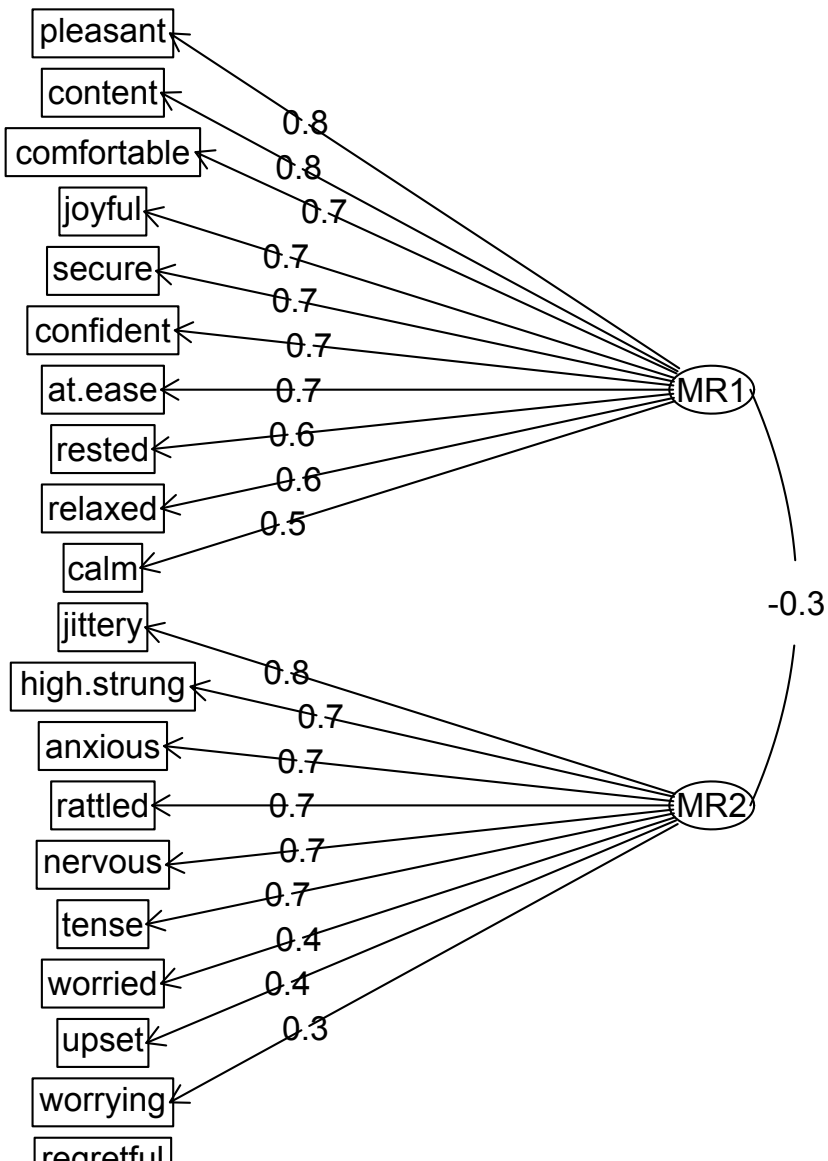
Demonstration of diagram functions



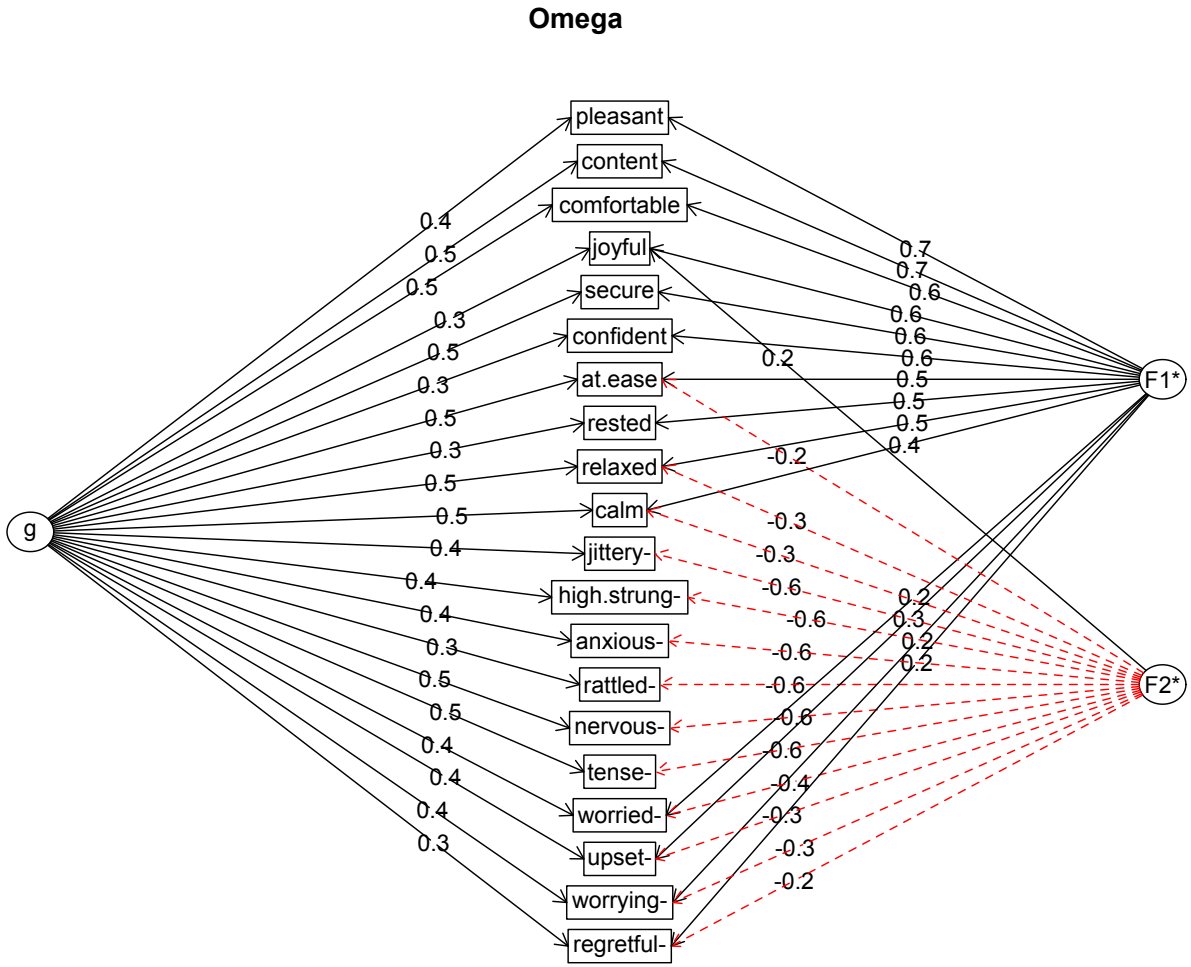
Functions that use these primitives

1. `fa.diagram`
2. `omega.diagram`
3. `lmDiagram`
4. `structure.diagram`
5. `diagram` calls the appropriate function from the psych class.

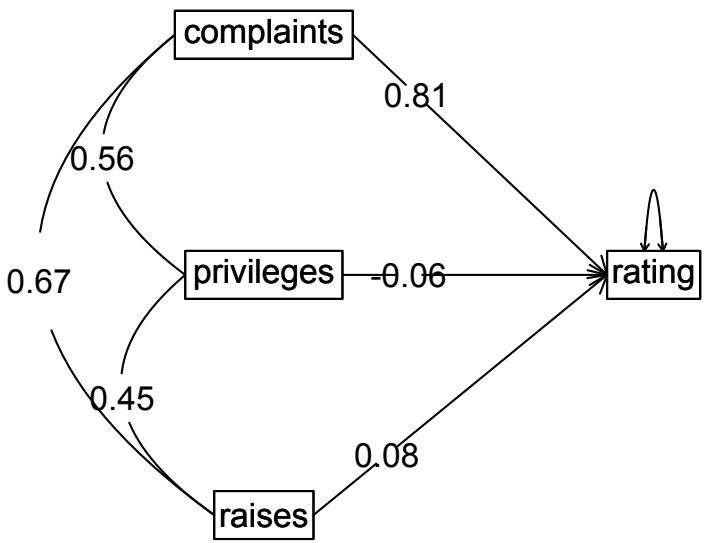
factor analysis of 20 State Anxiety items



Omega represents higher order and bifactor structure



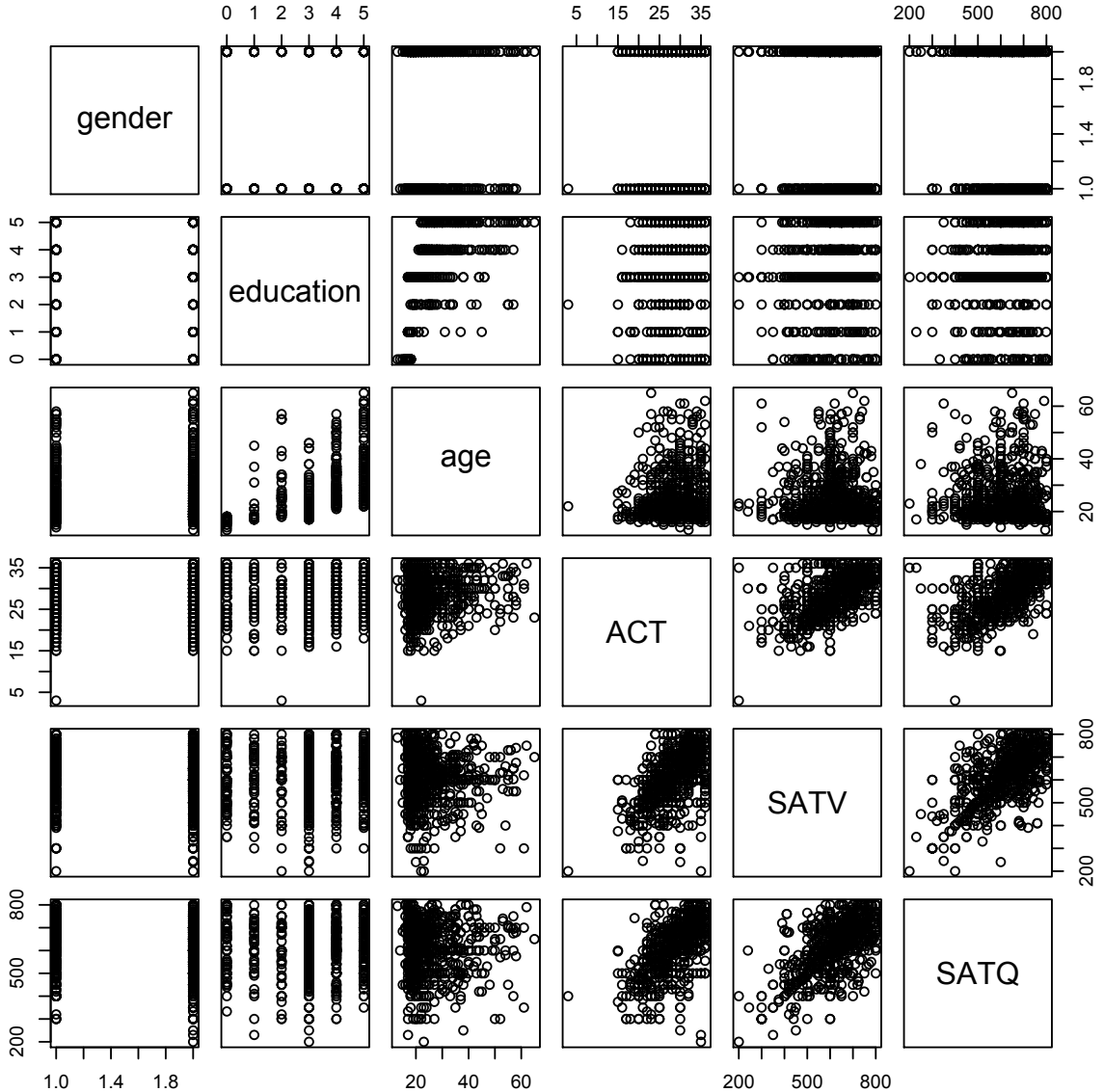
Regression Models



Function development

1. Specify a need (e.g., graphically display scatter plot matrices SPLOMs)
2. Is there a function that almost does what you want to do?
3. Do just want to use that function with new options?
4. Examine the code of that function and modify it to do what you want.
5. The example of `pairs.panels`
6. Developed from the help pages of `pairs`

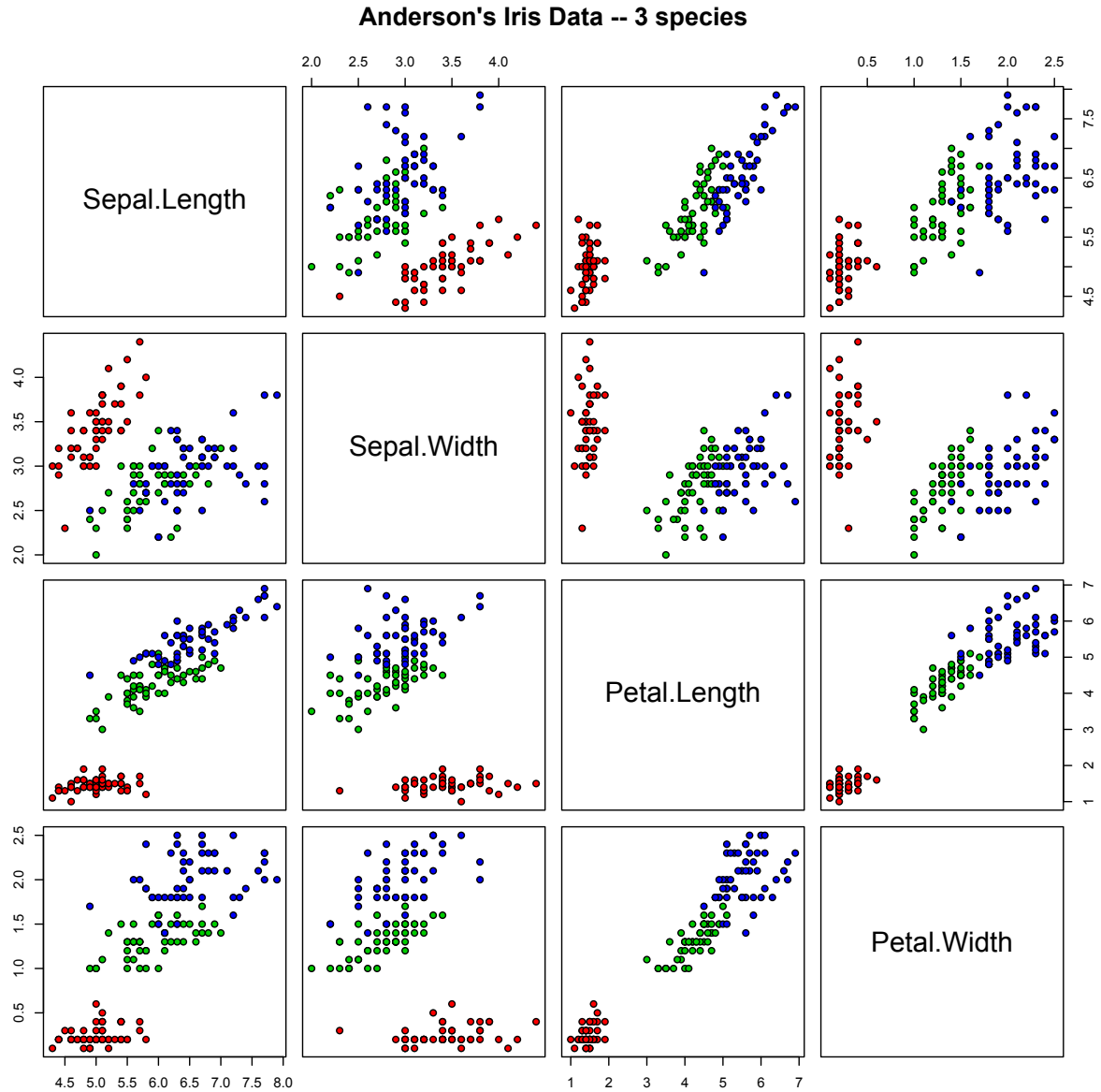
The basic 'pairs' function



pairs(sat.act)

A somewhat better use of `pairs`

```
pairs(iris[1:4], main = "Anderson's Iris Data -- 3 species",
      pch = 21, bg = c("red", "green3", "blue")[unclass(iris$Species)])
```



`pairs` is a perfectly good function

The help menu shows nice options. Can we create a new function with those options.

We find and then look at the code..

R code

```
pairs   #to look at the code
methods(pairs)  # it is a method, so we need to find
                the various methods
getAnywhere(pairs.default)  #the code for pairs
```

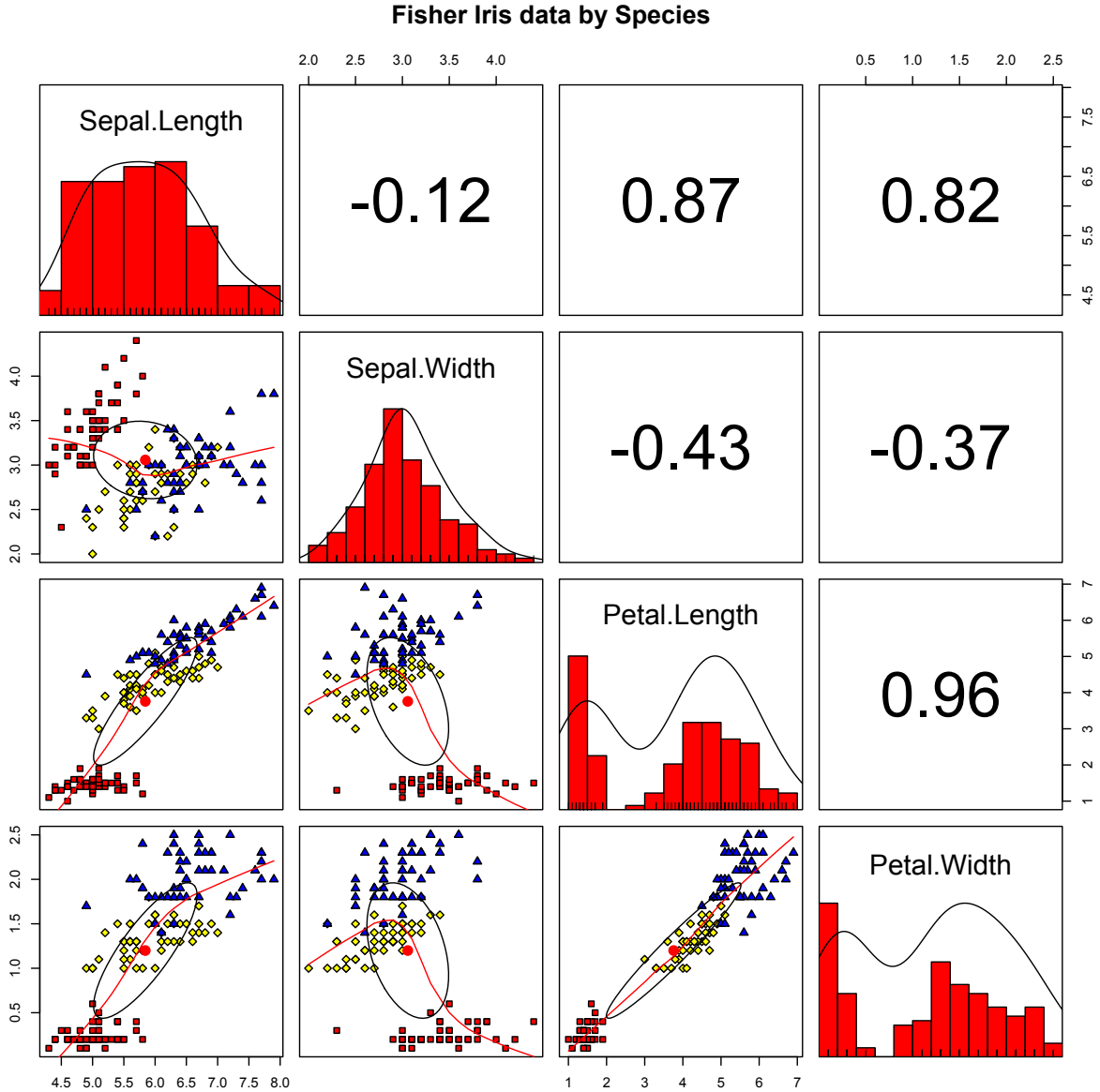
```
pairs
function (x, ...)
UseMethod("pairs")
<bytecode: 0x14b269ad0>
<environment: namespace:graphics>

methods(pairs)
[1] pairs.compareFits* pairs.default*     pairs.formula*     pairs.lme*         pairs.lmList*
[6] pairs.panels*
see '?methods' for accessing help and source code
```

Now edit pairs.default function to do what you want.

The pairs.panels function

```
pairs.panels(iris[1:4],bg=c("red","yellow","blue")[iris$Species],
  pch=21+as.numeric(iris$Species),main="Fisher Iris data by Species",hist.col="red")
```



`lm` and `lmCor`

1. `lm` does linear regression from complete data
2. Because the regression slopes (β) are found from the covariance matrix, most of the statistics do not require the raw data.
3. The fit of the residuals require the raw data, but the tests of slopes do not.
4. Can do the same work from the correlation matrix. Hence `lmCor`.
5. Developed from the problem, not modifying the code. Just following the math.
6. Although based upon correlations, was extended to work with raw data.
7. Originally developed to do regressions from correlation matrices (mat.regress), it eventually was expanded to handle raw data and to use formula input.

lm for linear regression from data

R code

```
mod.lm <- lm(rating ~ complaints + privileges, data = attitude)
mod.lm
summary(mod.lm)
```

```
mod.lm <- lm(rating ~ complaints + privileges, data = attitude)
Call:
lm(formula = rating ~ complaints + privileges, data = attitude)

Coefficients:
(Intercept)   complaints   privileges
   15.32762      0.78034     -0.05016

> summary(mod.lm)\
Call:
lm(formula = rating ~ complaints + privileges, data = attitude)

Residuals:
     Min       1Q   Median       3Q      Max
-12.7887  -5.6893  -0.0284   6.2745   9.9726

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 15.32762    7.16023   2.141   0.0415 *
complaints   0.78034    0.11939   6.536 5.22e-07 ***
privileges  -0.05016    0.12992  -0.386   0.7025
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.102 on 27 degrees of freedom
Multiple R-squared:  0.6831,        Adjusted R-squared:  0.6596
F-statistic:  29.1 on 2 and 27 DF,  p-value: 1.833e-07
```

lmCor on raw data

R code

```
mod.lmCor <- lmCor(rating ~ complaints + privileges, data = attitude,
           std=FALSE)
```

```
mod.lmCor
Call: lmCor(y = rating ~ complaints + privileges, data = attitude,
    std = FALSE)

Multiple Regression from raw data

 DV =  rating
            slope   se     t       p lower.ci upper.ci  VIF  Vy.x
(Intercept) 15.33 7.16  2.14 4.1e-02     0.64    30.02 1.00  0.00
complaints   0.78 0.12  6.54 5.2e-07     0.54     1.03 1.45  0.70
privileges  -0.05 0.13 -0.39 7.0e-01    -0.32     0.22 1.45 -0.02

Residual Standard Error =  7.1  with  27  degrees of freedom

 Multiple Regression
          R   R2  Ruw R2uw Shrunken R2 SE of R2 overall F df1 df2        p
rating 0.83 0.68 0.62 0.38        0.66     0.08       29.1   2  27 1.83e-07
```

Values are identical