R Short Course: Day 5

Using R in the classroom

Programming in R

Programming in R

I. Data types

II. operators

III. simple functions

IV. Writing functions

Data structures

I. elements: logical, integer, real, character, factor

II. vectors: ordered sets of elements of one type

III. matrices: ordered sets of vectors (all of one type)

IV. data.frames: ordered sets of vectors, may be different types

V. lists: ordered set of anything

Operators

I. arithmetical

1. +, -, *, /, ^, %%
2. a + b, a-b, a*b, a/b, a^b, a %%b

II. Logical

A. ==, !, !=, >, <, >=, <=

III. Matrix

A. %*% is matrix multiplication

B. %o% is outer product

example operations

```
> a <- 2
> b <- 3
> v <- 5:10
> w <- 6:7
> v
[1]  5  6  7  8  9 10
> w
[1] 6 7
> v ^a
[1]  25  36  49  64  81 100
> w* b
[1] 18 21
> w * v
[1] 30 42 42 56 54 70
```

Matrix operations

```
> v
[1]  5  6  7  8  9 10
> t(v)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    5    6    7    8    9   10
> t(v)%*% v
     [,1]
[1,]  355
> v %*% t(v)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   25   30   35   40   45   50
[2,]   30   36   42   48   54   60
[3,]   35   42   49   56   63   70
[4,]   40   48   56   64   72   80
[5,]   45   54   63   72   81   90
[6,]   50   60   70   80   90  100
```

Additional matrix operators

outer product

```
> v
[1]  5  6  7  8  9 10
> w
[1] 6 7
> v %o% w
     [,1] [,2]
[1,]   30   35
[2,]   36   42
[3,]   42   49
[4,]   48   56
[5,]   54   63
[6,]   60   70
```

matrix "addition" (psych)

```
>  x <- seq(4,8,2)
> x
[1] 4 6 8

> x %+% t(x)
     [,1] [,2] [,3]
[1,]    8   10   12
[2,]   10   12   14
[3,]   12   14   16
```

Review of Matrix Algebra

I. scalers, vectors, and matrices

A. scalers: simple numbers

B. vectors: ordered sets of numbers

1. V1 = {1 2 3 4 5 6 7 8 9 10 }
2. V2 = { 11 12 13 14 15 16 17 18 19 20 }
3. V2[3] = 13

C. Matrices (vectors of vectors)

See personality-project.org/r/sem.appendix.1.pdf

Matrices

$$
{}_{n}X_{m} = \left\{ \begin{matrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{matrix} \right\}
$$

R

	x1	x2	x3
x1	1.00	0.56	0.48
x2	0.56	1.00	0.42
x3	0.48	0.42	1.00

Xij

	V1	V2	V3	V4
S1	9	4	9	7
S2	9	7	1	8
S3	2	9	9	3
S4	8	2	9	6
S5	6	4	0	0
S6	5	9	5	8
S7	7	9	3	0
S8	1	1	9	2
S9	6	4	4	9
S10	7	5	8	6

Vector operations

I. addition

A. V3 <- V1 + V2
B. V3 = {12 14 16 18 20 22 24 26 28 30}

II. multiplication

A. element by element

1. V1*V2 = 11 24 39 56 75 96 119 144 171 200

B. inner product of vector (Sums of products)

C. outer product of vectors (matrix of products)

Inner and outer products

$$inner.product = \sum_{i=1}^{N} V1_i * V2_i$$

$${}_nX_1 *_1 Y_m =_n (XY)_m$$

Outer product (graphically)

```
> V1

 [1]  1  2  3  4  5  6  7  8  9 10

> V2

[1] 1 2 3 4

> outer.prod <- V1 %*% t(V2)
> outer.prod

      [,1] [,2] [,3] [,4]
 [1,]    1    2    3    4
 [2,]    2    4    6    8
 [3,]    3    6    9   12
 [4,]    4    8   12   16
 [5,]    5   10   15   20
 [6,]    6   12   18   24
 [7,]    7   14   21   28
 [8,]    8   16   24   32
 [9,]    9   18   27   36
[10,]   10   20   30   40
```

Matrix Operations

I. Addition/Subtraction

A. (element by element)

B. must be of same dimensions

II. Multiplication

A. ${}_{m}X_{n}\ {}_{n}Y_{p} = {}_{m}XY_{p}$ where the elements of XY, x_{ij} are the sums of the products of the elements of the ith row and jth column

B. $X\,Y \neq Y\,X$

Matrix multiplication

$$ {}_{m}X_{n} \; {}_{n}Y_{p} = {}_{m}XY_{p} $$

$$ xy_{ij} = \sum_{k=1}^{N} x_{ik} * y_{jk}. $$

Matrix multiplication for data

one

```
1 1 1 1 1 1 1 1 1 1
```

one %*%Xij=

	V1	V2	V3	V4
[1,]	60	54	57	49

X.means <- one %*% Xij/n

	V1	V2	V3	V4
[1,]	6	5.4	5.7	4.9

Xij

	V1	V2	V3	V4
S1	9	4	9	7
S2	9	7	1	8
S3	2	9	9	3
S4	8	2	9	6
S5	6	4	0	0
S6	5	9	5	8
S7	7	9	3	0
S8	1	1	9	2
S9	6	4	4	9
S10	7	5	8	6

Deviation scores as matrix differences

```
X.diff <- Xij - one %*% X.means
```

	V1	V2	V3	V4
S1	3	-1.4	3.3	2.1
S2	3	1.6	-4.7	3.1
S3	-4	3.6	3.3	-1.9
S4	2	-3.4	3.3	1.1
S5	0	-1.4	-5.7	-4.9
S6	-1	3.6	-0.7	3.1
S7	1	3.6	-2.7	-4.9
S8	-5	-4.4	3.3	-2.9
S9	0	-1.4	-1.7	4.1
S10	1	-0.4	2.3	1.1

Covariance as matrix product

```
X.cov <- t(X.diff) %*% X.diff/(n - 1)
```

X.cov

	V1	V2	V3	V4
V1	7.33	0.11	-3.00	3.67
V2	0.11	8.71	-3.20	-0.18
V3	-3.00	-3.20	12.68	1.63
V4	3.67	-0.18	1.63	11.43

diag(X.cov)

V1	V2	V3	V4
7.33	8.71	12.68	11.43

Correlation = standardized covariance

	V1	V2	V3	V4
V1	0.37	0.00	0.00	0.0
V2	0.00	0.34	0.00	0.0
V3	0.00	0.00	0.28	0.0
V4	0.00	0.00	0.00	0.3

```
sdi <-
diag(1/sqrt(diag(X.cov)))
```

	V1	V2	V3	V4
V1	1.00	0.01	-0.31	0.40
V2	0.01	1.00	-0.30	-0.02
V3	-0.31	-0.30	1.00	0.14
V4	0.40	-0.02	0.14	1.00

```
X.cor <-
sdi %*% X.cov %*% sdi
```

The identity matrix

```
I <- diag(1,nrow=4)

     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
```

Matrix Inverse

$X'X^{-1} = X^{-1}X = I$

	V1	V2	V3	V4
V1	1.00	0.01	-0.31	0.40
V2	0.01	1.00	-0.30	-0.02
V3	-0.31	-0.30	1.00	0.14
V4	0.40	-0.02	0.14	1.00

X.cor

	V1	V2	V3	V4
V1	1.44	0.15	0.58	-0.65
V2	0.15	1.12	0.40	-0.09
V3	0.58	0.40	1.36	-0.41
V4	-0.65	-0.09	-0.41	1.32

$X.cor^{-1}$

$X^{-1}X = XX^{-1} = I$

X.inv %*% X.cor

	V1	V2	V3	V4
V1	1	0	0	0
V2	0	1	0	0
V3	0	0	1	0
V4	0	0	0	1

X.cor %*% X.inv

	V1	V2	V3	V4
V1	1	0	0	0
V2	0	1	0	0
V3	0	0	1	0
V4	0	0	0	1

Matrix algebra is your friend

Functions

I. Operate on an object and provide a new object

II. e.g., f <- function(x) {x * 2}

Simple functions

```
> f <- function(x) {x * 2}
> f(43)
[1] 86
> x
[1] 4 6 8
> f(x)
[1]  8 12 16
> f( v %o% w)
     [,1] [,2]
[1,]   60   70
[2,]   72   84
[3,]   84   98
[4,]   96  112
[5,]  108  126
[6,]  120  140
```

a subset of useful functions

I. is.na(), is.null(), is.vector(), is.matrix(), is.list()

II. sum(), rowSums(), colSums(), mean(x), rowMeans(), colMeans(), max, min, median (these work on the entire matrix)

III. var, cov, cor, sd (these work on the columns of the matrix/data.frame)

IV. help.start() brings up a web page of manuals

More useful functions

I. rep(x,n) (repeats the value x n times)

II. c(x,y) (combines x with y)

III. cbind(x,y) combines column wise

IV. rbind(x,y) combines rowwise

V. seq(a,b,c) sequence from a to b stepping by c

sums on matrices and data.frames

```
> z <- f( v %o% w)
> z
     [,1] [,2]
[1,]   60   70
[2,]   72   84
[3,]   84   98
[4,]   96  112
[5,]  108  126
[6,]  120  140
```

```
> sum(z)
[1] 1170
> min(z)
[1] 60
> max(z)
[1] 140
> median(z)
[1] 97
```

```
> rowSums(z)
[1] 130 156 182 208 234 260
> colSums(z)
[1] 540 630
> mean(z)
[1] 97.5
> rowMeans(z)
[1]  65  78  91 104 117 130
```

Basic stats functions, part 2

```
> var(z)
     [,1] [,2]
[1,]  504  588
[2,]  588  686
> cov(z)
     [,1] [,2]
[1,]  504  588
[2,]  588  686
> cor(z)
     [,1] [,2]
[1,]    1    1
[2,]    1    1
> sd(z)
[1] 22.44994 26.19160
> z
     [,1] [,2]
[1,]   60   70
[2,]   72   84
[3,]   84   98
[4,]   96  112
[5,]  108  126
[6,]  120  140
```

?cor

```
 var(x, y = NULL, na.rm = FALSE, use)

cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall",
"spearman"))

cor(x, y = NULL, use = "everything",
     method = c("pearson", "kendall",
"spearman"))

cov2cor(V)
```

More on cor

x

a numeric vector, matrix or data frame.

y

NULL (default) or a vector, matrix or data frame with compatible dimensions to x. The default is equivalent to y = x (but more efficient).

na.rm

logical. Should missing values be removed?

use

an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".

method

a character string indicating which correlation coefficient (or covariance) is to be computed. One of "pearson" (default), "kendall", or "spearman", can be abbreviated.

V

symmetric numeric matrix, usually positive definite such as a covariance matrix.

row and col as functions

```
> r <- .8
> R <- diag(1,8)
> R
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    0    0    0    0    0    0    0
[2,]    0    1    0    0    0    0    0    0
[3,]    0    0    1    0    0    0    0    0
[4,]    0    0    0    1    0    0    0    0
[5,]    0    0    0    0    1    0    0    0
[6,]    0    0    0    0    0    1    0    0
[7,]    0    0    0    0    0    0    1    0
[8,]    0    0    0    0    0    0    0    1
> R <- r^(abs(row(R)-col(R)))
> round(R,2)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] 1.00 0.80 0.64 0.51 0.41 0.33 0.26 0.21
[2,] 0.80 1.00 0.80 0.64 0.51 0.41 0.33 0.26
[3,] 0.64 0.80 1.00 0.80 0.64 0.51 0.41 0.33
[4,] 0.51 0.64 0.80 1.00 0.80 0.64 0.51 0.41
[5,] 0.41 0.51 0.64 0.80 1.00 0.80 0.64 0.51
[6,] 0.33 0.41 0.51 0.64 0.80 1.00 0.80 0.64
[7,] 0.26 0.33 0.41 0.51 0.64 0.80 1.00 0.80
[8,] 0.21 0.26 0.33 0.41 0.51 0.64 0.80 1.00
```

Yet more stats functions

I. sample(n, N, replace=TRUE)

II. eigen(X) (eigen value decomposition of X)

III. solve(X) (inverse of X)

IV. solve (X,Y) Regression of Y on X

Creating a matrix

```
> x <- matrix(sample(10,50,replace=TRUE),ncol=5)
> x
      [,1] [,2] [,3] [,4] [,5]
 [1,]   10    3    4    4    6
 [2,]    3   10    8    8    9
 [3,]    1    6    5    8    5
 [4,]    9    1    3    5    3
 [5,]    6    8    3    5    1
 [6,]    8    6   10    1   10
 [7,]   10    5   10    2    1
 [8,]    9    3    2    2    9
 [9,]    6   10    2    9    4
[10,]    1    8    8    2    6
```

standardize it

```
> z <- scale(x)
> z
             [,1]       [,2]       [,3]       [,4]       [,5]
 [1,]  1.04838349 -0.9819805 -0.4678877 -0.2059329  0.1852621
 [2,] -0.93504473  1.3093073  0.7798129  1.1669533  1.1115724
 [3,] -1.50173851  0.0000000 -0.1559626  1.1669533 -0.1235080
 ...
 [9,] -0.08500407  1.3093073 -1.0917380  1.5101749 -0.4322782
[10,] -1.50173851  0.6546537  0.7798129 -0.8923761  0.1852621
attr(,"scaled:center")
[1] 6.3 6.0 5.5 4.6 5.4
attr(,"scaled:scale")
[1] 3.529243 3.055050 3.205897 2.913570 3.238655
```

Just center it

```
> c <- scale(x,scale=FALSE)

> c
      [,1] [,2] [,3] [,4] [,5]
 [1,]  3.7   -3 -1.5 -0.6  0.6
 [2,] -3.3    4  2.5  3.4  3.6
 [3,] -5.3    0 -0.5  3.4 -0.4
 [4,]  2.7   -5 -2.5  0.4 -2.4
 [5,] -0.3    2 -2.5  0.4 -4.4
 [6,]  1.7    0  4.5 -3.6  4.6
 [7,]  3.7   -1  4.5 -2.6 -4.4
 [8,]  2.7   -3 -3.5 -2.6  3.6
 [9,] -0.3    4 -3.5  4.4 -1.4
[10,] -5.3    2  2.5 -2.6  0.6
attr(,"scaled:center")
[1] 6.3 6.0 5.5 4.6 5.4
```

Find the covariance and inverse

```
>  c <- cov(x)
> round(c,2)
      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] 12.46 -6.89 -1.61 -4.53 -1.58
[2,] -6.89  9.33  2.11  4.11  0.56
[3,] -1.61  2.11 10.28 -3.89  2.22
[4,] -4.53  4.11 -3.89  8.49 -1.60
[5,] -1.58  0.56  2.22 -1.60 10.49
```

```
> round(solve(c),2)
     [,1]  [,2]  [,3]  [,4]  [,5]
[1,] 0.15  0.08  0.02  0.06  0.02
[2,] 0.08  0.23 -0.07 -0.10  0.00
[3,] 0.02 -0.07  0.16  0.12 -0.01
[4,] 0.06 -0.10  0.12  0.26  0.03
[5,] 0.02  0.00 -0.01  0.03  0.10
```

Flow control

I. if(condition) {then do this} else {do this}

II. for (condition) do {expression}

  A. for (i in 1:n} do {x <- x + 1}

III. while (condition) {expression}

conditionals

I. (a & b) vs. (a && b)

II. (a | b) vs. (a || b)

```
a <- 1
> if (a & b) {print ("hello")} else {print("goodby")}
Error: object 'b' not found
> if (a && b ) {print ("hello")} else {print("goodby")}
[1] "goodby"
> if (a | b) {print ("hello")} else {print("goodby")}
Error: object 'b' not found

> if (a || b) {print ("hello")} else {print("goodby")}
Error: object 'b' not found
> a <- 1
> if (a || b) {print ("hello")} else {print("goodby")}
[1] "hello"
>
```

simple control

```
> a <- 1
> b <- 2
> c <- 3
> k <- 10
> x <-1
> if(x == a) {print("x is the same as a and has
value",x)} else {print ("x is not equal to a")}

> x <- 3
> if(x == a) {print("x is the same as a and has
value",x)} else {print ("x is not equal to a")}
[1] "x is not equal to a"
>
```

Make that a function

```
> f1 <- function(x,y) {if(x == y)
{print("x is the same as y and has
value",x)} else {print ("x is not equal
to y")}}
> f1(3,4)
[1] "x is not equal to y"
> f1(5,5)
[1] "x is the same as y and has value"
```

Simple functions:part 2

I. Find the squared multiple correlation of a variable with all the other variables in a matrix.

II. The R^2 is 1- the residual variance

The essence of the function

```
SMC <- function(R) {
R.inv <- solve(R)
SMC <- 1 - 1/diag(R.inv)}

> S <-cor(attitude)
> SMC(S)      #does not show anything

> round(SMC(S),2) #but this does
    rating complaints privileges   learning     raises
critical    advance
      0.73       0.77       0.38       0.62       0.68
0.19       0.52
```

Add a return

```
SMC <- function(R) {
R.inv <- solve(R)
SMC <- 1 - 1/diag(R.inv)
return(SMC)}

> SMC(S)
    rating complaints privileges   learning     raises
critical    advance
 0.7326020  0.7700868  0.3831176  0.6194561  0.6770498
0.1881465  0.5186447
```

Allow it to find R

```
SMC <- function(R) {
if(dim(R)[1] !=dim(R)[2]) {R <-cor(R)}
R.inv <- solve(R)
SMC <- 1 - 1/diag(R.inv)
return(SMC)}
```

```
> SMC(attitude)
    rating complaints privileges   learning     raises
critical    advance
 0.7326020  0.7700868  0.3831176  0.6194561  0.6770498
0.1881465  0.5186447
```

Clean up the output

```
SMC <- function(R,digits=2) {
if(dim(R)[1] !=dim(R)[2]) {R <-cor(R)}
R.inv <- solve(R)
SMC <- 1 - 1/diag(R.inv)
return(round(SMC,digits))}

> SMC(attitude)
    rating complaints privileges   learning     raises   critical
advance
      0.73       0.77       0.38       0.62       0.68       0.19
0.52
>
```

Check for poor input

```
>  att <- data.frame(attitude[1:3],attitude[1:3])
> SMC(att)
Error in solve.default(R) :
  Lapack routine dgesv: system is exactly singular
```

Add checks for weird matrices

```
SMC <- function(R,digits=2) {
 p <- dim(R)[2]
 if (dim(R)[1] != p) {R <-cor(R)}
R.inv <- try(solve(R),TRUE)
if(class(R.inv)== as.character("try-error")) {SMC <- rep(1,p)
 warning("Correlation matrix not invertible, smc's returned as 1s")}
else  {smc <- 1 -1/diag(R.inv)
SMC <- 1 - 1/diag(R.inv)}
return(round(SMC,digits))}

> SMC(att)
[1] 1 1 1 1 1 1
Warning message:
In SMC(att) : Correlation matrix not invertible, smc's returned as
1s

> SMC(attitude)
    rating complaints privileges   learning     raises   critical
      0.73       0.77       0.38       0.62       0.68       0.19
```

Further checks

Input is a covariance matrix

```
> SMC(cov(attitude))
    rating complaints privileges   learning     raises   critical
advance
    -38.62     -39.76     -91.35     -51.42     -33.91     -78.49
-49.96
```

Input is raw data or correlations

```
> SMC(cor(attitude))
    rating complaints privileges   learning     raises   critical
advance
      0.73       0.77       0.38       0.62       0.68       0.19
0.52
> SMC(attitude)
    rating complaints privileges   learning     raises   critical
advance
      0.73       0.77       0.38       0.62       0.68       0.19
0.52
```

Final version

```
#modified Dec 10, 2008 to return 1 on diagonal if non-invertible
#modifed March 20, 2009 to return smcs * variance if covariance matrix
is desired
#modified April 8, 2009 to remove bug introduced March 10 when using
covar from data
"smc" <-
function(R,covar =FALSE) {
failed=FALSE
 p <- dim(R)[2]
 if (dim(R)[1] != p) {if(covar) {C <- cov(R, use="pairwise")
                                 vari <- diag(C)
                                 R <- cov2cor(C)
                                 } else {R <- cor(R,use="pairwise")}}
else {vari <- diag(R)
                                 R <- cov2cor(R)
      if (!is.matrix(R)) R <- as.matrix(R)}
 R.inv <- try(solve(R),TRUE)
 if(class(R.inv)== as.character("try-error")) {smc <- rep(1,p)
 warning("Correlation matrix not invertible, smc's returned as 1s")}
else  {smc <- 1 -1/diag(R.inv)
 if(covar) {smc <- smc * vari}}
 return(smc)
 }
```

Creating a new function

I. Is there a base function to modify?

II. Consider the case of modifying Promax rotation to allow for any target matrix

III. original promax (inside the factanal package) had been modified to report the factor correlation.

IV. This version was created with the assistance of Pat Shrout and Steve Miller

promax

```
> promax
function (x, m = 4)
{
    if (ncol(x) < 2)
        return(x)
    dn <- dimnames(x)
    xx <- varimax(x)
    x <- xx$loadings
    Q <- x * abs(x)^(m - 1)
    U <- lm.fit(x, Q)$coefficients
    d <- diag(solve(t(U) %*% U))
    U <- U %*% diag(sqrt(d))
    dimnames(U) <- NULL
    z <- x %*% U
    U <- xx$rotmat %*% U
    dimnames(z) <- dn
    class(z) <- "loadings"
    list(loadings = z, rotmat = U)
}
<environment: namespace:stats>
```

Promax

```
> Promax
function (x, m = 4)
{
    if (!is.matrix(x) & !is.data.frame(x)) {
        if (!is.null(x$loadings))
            x <- as.matrix(x$loadings)
    }
    else {
        x <- x
    }
    if (ncol(x) < 2)
        return(x)
    dn <- dimnames(x)
    xx <- varimax(x)
    x <- xx$loadings
    Q <- x * abs(x)^(m - 1)
    U <- lm.fit(x, Q)$coefficients
    d <- diag(solve(t(U) %*% U))
    U <- U %*% diag(sqrt(d))
    dimnames(U) <- NULL
    z <- x %*% U
    U <- xx$rotmat %*% U
    ui <- solve(U)
    Phi <- ui %*% t(ui)
    dimnames(z) <- dn
    class(z) <- "loadings"
    result <- list(loadings = z, rotmat = U, Phi = Phi)
    class(result) <- c("psych", "fa")
    return(result)}
```

target.rot

```
"target.rot" <-
function (x, keys=NULL,m = 4)
{
 if(!is.matrix(x) & !is.data.frame(x) )  {
        if(!is.null(x$loadings)) x <- as.matrix(x$loadings)
      } else {x <- x}
    if (ncol(x) < 2)
        return(x)
    dn <- dimnames(x)
    if(is.null(keys)) {xx <- varimax(x)
    x <- xx$loadings
    Q <- x * abs(x)^(m - 1)} else {Q <- keys}
    U <- lm.fit(x, Q)$coefficients
    d <- diag(solve(t(U) %*% U))
    U <- U %*% diag(sqrt(d))
    dimnames(U) <- NULL
    z <- x %*% U
    if (is.null(keys)) {U <- xx$rotmat %*% U } else {U <- U}
    ui <- solve(U)
    Phi <- ui %*% t(ui)
    dimnames(z) <- dn
    class(z) <- "loadings"
    result <- list(loadings = z, rotmat = U,Phi = Phi)
    class(result) <- c("psych","fa")
    return(result)
}
```

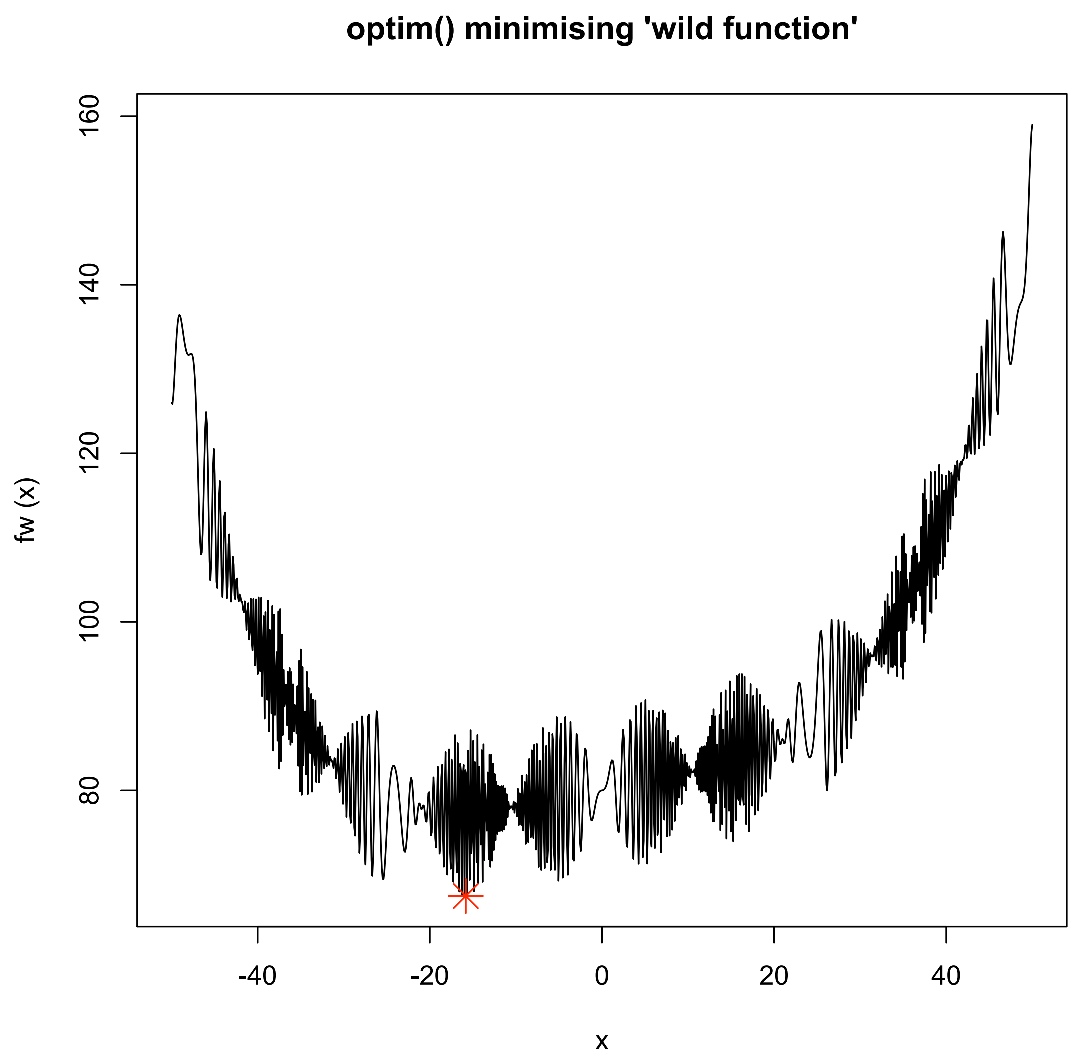
optim as “solver” for R

I. Many statistical functions are not closed form but rather are solved iteratively.

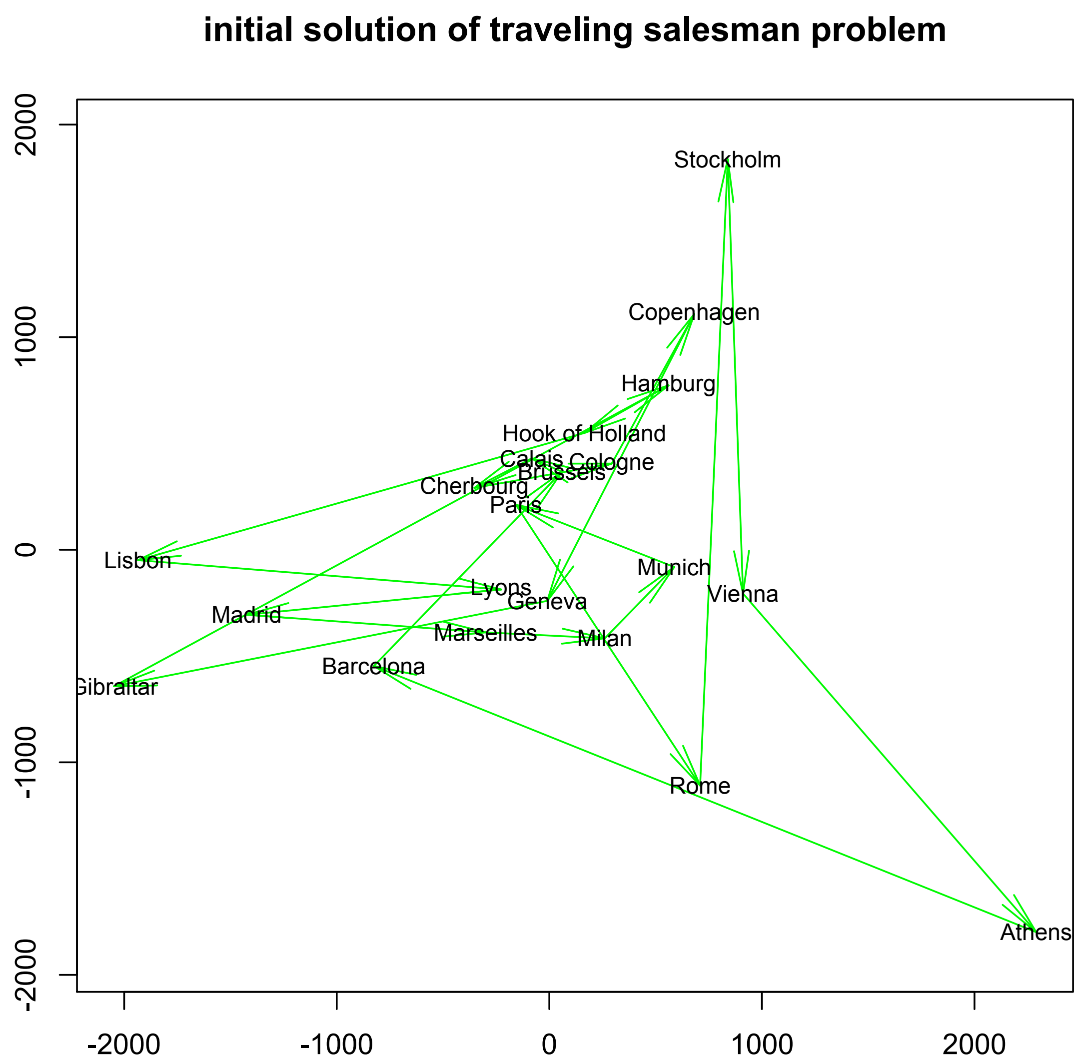
II. We start with a good guess and then minimize the function

III. optim will do this for functions where you manipulate one vector (which can of course actually be a matrix)

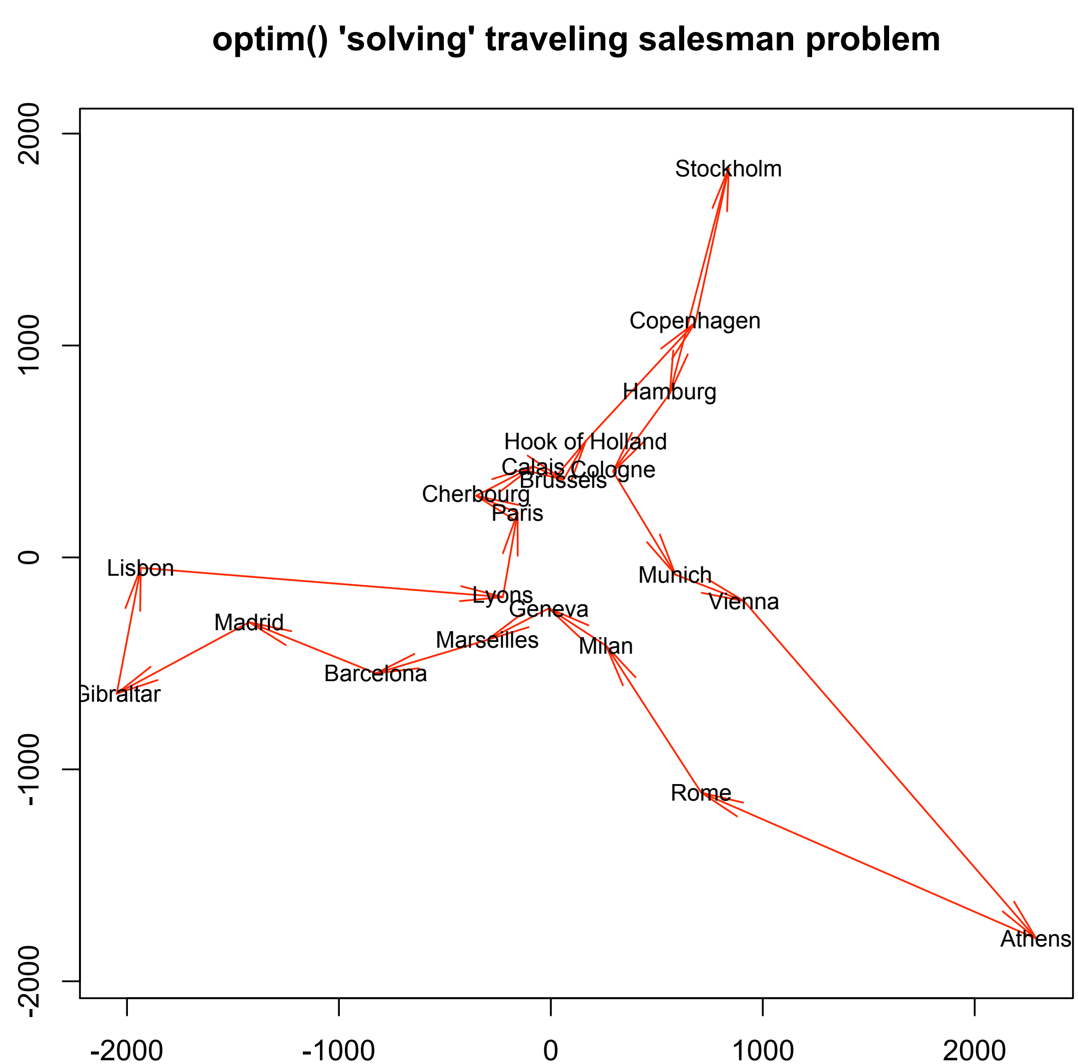
optim



optim() minimising 'wild function'
fw (x)
160
140
120
100
80
-40
-20
0
20
40
x



initial solution of traveling salesman problem
Stockholm
Copenhagen
Hamburg
Hook of Holland
Calais
Cologne
Brussels
Cherbourg
Paris
Lisbon
Munich
Lyons
Geneva
Vienna
Madrid
Marseilles
Milan
Barcelona
Gibraltar
Rome
Athens
2000
1000
0
-1000
-2000
-2000
-1000
0
1000
2000


optim() 'solving' traveling salesman problem
Stockholm
Copenhagen
Hamburg
Hook of Holland
Calais
Cologne
Brussels
Cherbourg
Paris
Lisbon
Lyons
Geneva
Munich
Vienna
Madrid
Marseilles
Milan
Barcelona
Gibraltar
Rome
Athens
2000
1000
0
-1000
-2000
-2000
-1000
0
1000
2000

Trying to make a new function to do OLS FA

I. First, look at current ML FA function

   A. factanal

   B. It turns out that the critical optimization is done in factanal.fit.mle, but where is that?

II. getAnywhere(factanal.fit.mle)

   A. then look at the code

   B. scratch your head and try running it

R in the classroom

I. Undergraduate statistics and research methods

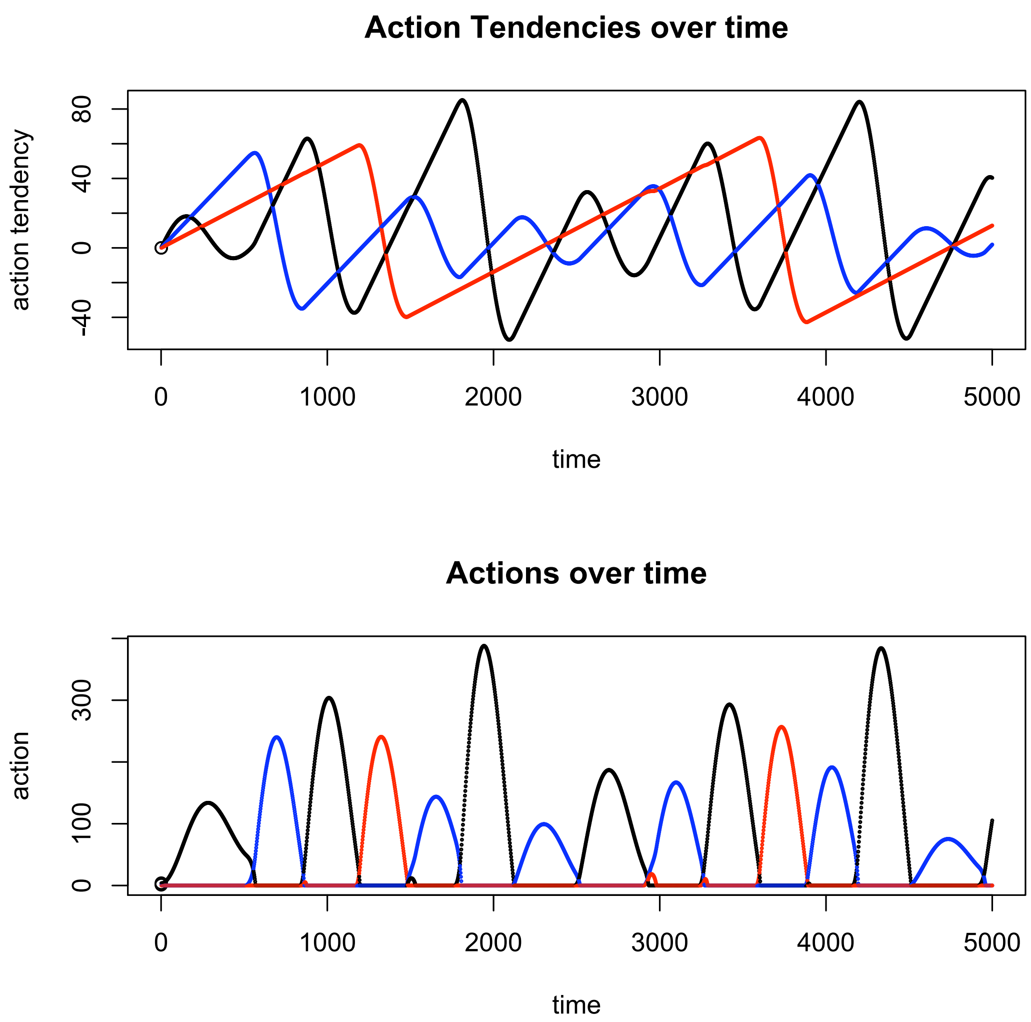
   A. describe, pairs.panels, anova, lm

   B. plot, curve, etc.

   C. see tutorials for 205 and 371

   D. simulations of data for simulated studies

   E. Examples of complex models


Action Tendencies over time
action tendency
-40
0
40
80
0
1000
2000
3000
4000
5000
time
Actions over time
action
0
100
300
0
1000
2000
3000
4000
5000
time

R in the classroom

I. Graduate
   A. data simulations
   B. data analysis
   C. longer tutorial