Statistical Programming Languages – Day 5

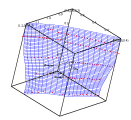
Uwe Ziegenhagen

Institut für Statistik and Ökonometrie
Humboldt-Universität zu Berlin
http://www.uweziegenhagen.de

Agenda for Today

Programming *R*

- Functions
- Loops
- Conditions
- Packages

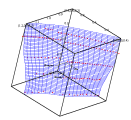


Functions

```
1 myfun <- function(x){
2 return(x*x)
3 }
4 myfun(2)
```

```
1 myfun <- function(x){
2 x*x
3 }
4 myfun(2)
```

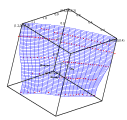
If no return() is given, the object last created is returned.



Functions

```
1 myfun <- function(x, a){
2 r <- a*sin(x)
3 return(r)
4 }
5 myfun(pi/2,2)
```

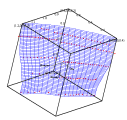
```
1 myfun1 <- function(x, a){ a*sin(x) } #short version
2 myfun1(pi/2,2)
```

Functions

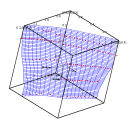
```
1 myfun2 <- function(x, a=1){ # opt. parameter with
    default=1
2 a*sin(x)
3 }
4 myfun2(pi/2,2)
5 myfun2(pi/2)
```

optional = the function can also be run without specification



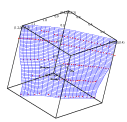
Functions

```
1 myfun3 <- function(x, a=NULL){ # opt. parameter
    without default
2 if (!is.null(a)){
3      a*sin(x)
4 }else{
5      cos(x)
6      }
7 }
8 myfun3(pi/2,2)
9 myfun3(pi/2)
```

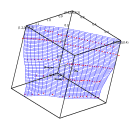
Functions

```
1 myfun4 <- function(x, a=1){
2  r1 <- a*sin(x)
3  r2 <- a*cos(x)
4 return(r1,r2)
5 # return two results
6 # don't use this, use a list
7 }
8 myfun4(pi/2)
```

Functions

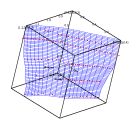
```
1 myfun5 <- function(x, a=1){
2  r1 <- a*sin(x)
3  r2 <- a*cos(x)
4 return(list(r1,r2)) # one list as result
5 }
6 myfun5(pi/2)
```

Functions

many different calls on this page

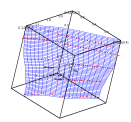
```
1  myfun6 <- function(x, a=1, b=2){
2  r1 <- a*sin(x)
3  r2 <- b*cos(x)
4  return(list(r1,r2))
5  }
6  myfun6(pi/2)      # a=1, b=2 (defaults)
7  myfun6(pi/2,1,2)  # a=1, b=2 (given explicitly)
8  myfun6(pi/2,2)    # a=2, b=2 (only a given)
9  myfun6(pi/2,a=2)  # a=2, b=2 (only a given)
10 myfun6(pi/2,b=3)  # a=1, b=2 (only b given)
```

Using Sets

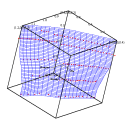
```
1  a <- 1:3
2  b <- 2:6
3  a
4  b
5  a %in% b
6  b %in% a
7  a <- c("A","B"); b <- LETTERS[2:6]
8  a
9  b
10 a %in% b
11 b %in% a
```

LETTERS is a special predefined variable, containing 'A' to 'Z', (letters='a' to 'z')



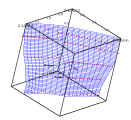
Loops and Conditions - IF

```
1 # simple if
2 x<-1
3 if (x==2){ print("x=2") }
4 # if-else
5 x<-1
6 if (x==2){
7   print("x=2")
8 } else {
9   print("x!=2")}
```

Loops and Conditions - FOR

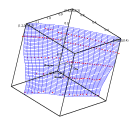
```
1 for (i in 1:4){ print(i) }
2 for (i in letters[1:4]){ print(i) }
3
4 a<-numeric(400) # generate empty a of length 400
5 for (i in 1:400){ a[i]=i } # fill a with 1:400
6 # takes much longer than a<-1:400
```

Loops and Conditions - WHILE

```
1 i<-0
2 while(i<4){
3       i <- i+1
4       print(i)
5 }
```

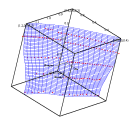
Do NOT forget the increase of the counter variable *i*!



Loops and Conditions - REPEAT

```
1 i <- 0;
2 repeat{
3  i <- i+1;
4  print(i);
5  if (i==4) break
6  }
```

If no break is given, loops runs forever!

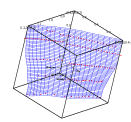


Loops and Conditions - IFELSE

`ifelse` allows shorter if-else syntax

```
ifelse(boolean check, if-case, else-case)
```

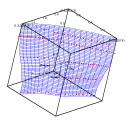
```
1 x <- c(6:-4)
2 sqrt(x) # gives warning
3 sqrt(ifelse(x >= 0, x, NA))# no warning
```

Loops and Conditions - SWITCH

switch avoids the use of nested if-else constructs, however it looks tricky. If type is numeric, switch uses the *i*-th item.

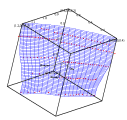
```
1 rootsquare <- function(x, type) {
2   switch(type,
3          square = x*x,
4          root = sqrt(x))
5 }
6 rootsquare(10,1)
7 rootsquare(10,2)
```

Loops and Conditions - SWITCH

If type is a string, R matches the string.

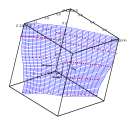
```
1 rootsquare <- function(x, type) {
2   switch(type,
3          square = x*x,
4          root = sqrt(x))
5 }
6 rootsquare(10,"square") # ok
7 rootsquare(10,"root")   # ok
8 rootsquare(10,"ROOT")   # returns NULL
```

Loops and Conditions - SWITCH

another example for switch

```
centre <- function(x, type) {
  switch(type,
         mean = mean(x),
         median = median(x),
         trimmed = mean(x, trim = .1))
}
x <- rcauchy(10)
centre(x, "mean")
centre(x, "median")
centre(x, "trimmed")
```

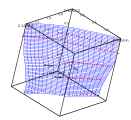



Loops and Conditions

`system.time()` measures the time necessary to run a command.

```
> x <- c(1:50000)
> system.time(for (i in 1:50000){x[i]<-rnorm(1)})
      User      System verstrichen
      0.67        0.01        0.69
> system.time(x<-rnorm(50000))
      User      System verstrichen
      0.01        0.00        0.01
```

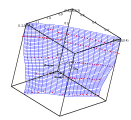
$\Rightarrow$ use matrix functions



The `apply()` commands

these commands allow functions to be run on matrices.

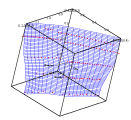
`apply()`	Function used on matrix
`tapply()`	table grouped by factors
`lapply()`	on lists and vectors; returns a list
`sapply()`	like `lapply()`, returns vector/matrix
`mapply()`	multivariate `sapply()`



apply()

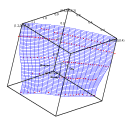
apply(data, margin, function)

```
1 > matrix(1:10,nrow=2)->a
2 > apply(a,1,mean)
3 [1] 5 6
4 > apply(a,2,mean)
5 [1] 1.5 3.5 5.5 7.5 9.5
```

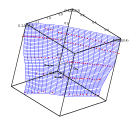
lapply()

```
> matrix(2:11,nrow=2)->a
> matrix(1:10,nrow=2)->b
> c= list(a,b)
> lapply(c,mean)
[[1]]
[1] 6.5

[[2]]
[1] 5.5
```

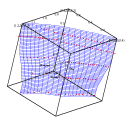
sapply()

```
1 > matrix(2:11,nrow=2)->a
2 > matrix(1:10,nrow=2)->b
3 > c= list(a,b)
4 > sapply(c,mean)
5 [1] 6.5 5.5
```

mapply()

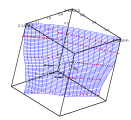
```
> mapply(rep, pi, 3:1)
[[1]]
[1] 3.141593 3.141593 3.141593

[[2]]
[1] 3.141593 3.141593

[[3]]
[1] 3.141593
```

tapply()

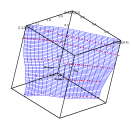
requires variable, Factor and function

```
> data(iris)
> attach iris
> tapply(Sepal.Width, Species, mean)
    setosa versicolor  virginica
     3.428      2.770      2.974
```

User Input

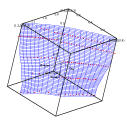
```
1 a<-scan()
2 menu(c("abc","def"),title="Enter value")
3 menu(c("abc","def"),graphics=TRUE, title="Enter
    value")
```

GUIs with Tcl/Tk

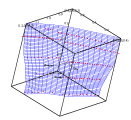
examples taken from http://bioinf.wehi.edu.au/~wettenhall/RTclTkExamples/mb.html

```
1 require(tcltk)
2 ReturnVal <- tkmessageBox(title="Greetings from R
    TclTk",
3 message="Hello, world!",icon="info",type="ok")
```

GUIs with Tcl/Tk - MessageBoxes

```
1 require(tcltk)
2 tkmessageBox(message="Do you want to save before
    quitting?",
3 icon="question",type="yesnocancel",default="yes")
```

Tips & Hints

Some useful stuff:

- `system()` runs OS commands
- `xtable()` generates LaTeX code (package **xtable**)
- `latex()` generates LaTeX code (package **Hmisc**)
- `eval()` evaluate string as command
- `parse()` evaluate string as command