

Einführung

Dieses Tutorial vermittelt innert kürzester Zeit ein Basiswissen über *R*. Es wird Sie auch mit einer geeigneten Arbeitsumgebung, um mit *R* zu arbeiten, vertraut machen. Diese Umgebung ist in allen Computerräumen des ETH Hauptgebäudes verfügbar.

Über *R*

R ist eine freie Software (copyright: GNU public license) und verfügbar unter <http://stat.ethz.ch/CRAN/>. Dort finden Sie auch umfangreiche **Dokumentationen**, [Manual](#), “An Introduction to *R*” (zirka 100 Seiten als pdf) und eine etwas kürzere Einführung [Contributed](#), “*R* for Beginners / *R* pour les débutants” (31 Seiten, English/French).

R-Umgebungen

Wer “professionelle” mit *R* arbeiten will, sollte dies mit *R*-Skripten und einem geeigneten Editor tun, welcher es erlaubt, *R*-Code direkt an einen laufenden *R*-Prozess zu schicken. Es gibt zahlreiche Editoren, auf allen Betriebssystemen, welche dies erlauben. Wir empfehlen die Benützung von *R Studio*, welche für all gängigen Betriebssysteme frei verfügbar ist (<http://rstudio.org>).

Als Alternativen stehen Ihnen die Editoren zur Verfügung, welche in den jeweiligen Betriebssystemen zusammen mit *R* installiert werden, *Emacs* mit dem Add-On *Emacs Speaks Statistics* (<http://stat.ethz.ch/ESS/>), *TinnR* (<http://www.sciviews.org/Tinn-R/>) und *WinEdt* für Windows (<http://www.winedt.com/>). Dieses Tutorial ist für *R Studio* geschrieben.

R Studio mit *R* starten

R wird direkt aus *R Studio* aufgerufen. Um *R Studio* zu starten, wählen Sie es im Startmenü unter Programme/Applications aus oder geben Sie `rstudio` in der Konsole ein.

R Studio vereinigt alle Ressourcen, welche Sie für das Programmieren mit *R* benötigen, in einem einzelnen aufgeräumten Fenster (Abb. 1). Im Panel *console* ist ein laufender *R*-Prozess. Es ist **nicht** nötig *R* separat zu starten.

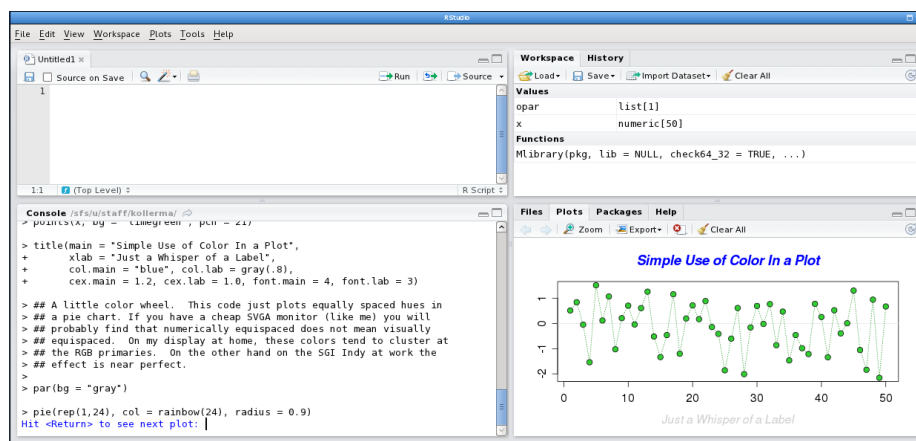


Figure 1: Die Arbeitsumgebung von *R Studio*. Die Standardanordnung der Panele besteht aus (im Uhrzeigersinn von oben links) der Editor; Objekte (workspace) und Verlauf (history); ein Panel mit Dateien, Grafik Browser (plots), verwendeten Packages und der *R*-Hilfe; und der Konsole mit dem laufenden *R*-Prozess.

R-Grundlagen

Geben Sie folgendes in der *R*-Konsole ein:

```
> x <- 2 (drücken Sie <Enter>)
```

```
> x (drücken Sie <Enter>)
```

Resultat: [1] 2

Der Zuweisungsoperator `<-` hat ein Objekt `x` generiert. *R* ist vektororientiert und somit ist `x` ein Vektor mit einem Eintrag, welcher den Wert 2 hat.

Als nächstes probieren Sie folgendes aus (die Kommando sind jeweils gefolgt vom Drücken der `<Enter>`-Taste, dies ist ab jetzt nicht mehr eingblendet):

```
> y <- c(3,5) (c steht für combine)
```

```
> y
```

Resultat: [1] 3 5, ein Vektor mit zwei Einträgen.






`ls()` zeigt alle Objekte, die Sie bisher generiert haben. Um `x` zu löschen, geben Sie `rm(x)` ein. *R* beinhaltet Beispiele für viele Funktionen. Schauen Sie sich die Beispiele für *image* an: `example(image)`. Dadurch generiert *R* eine Auswahl von Plots. Mit der `<Enter>`-Taste können Sie jeweils zum nächsten Plot wechseln.

Mit einer .R (Skript-)datei arbeiten


Erstellen Sie eine neue Skriptdatei via **File** → **New** → **R Script**. Sie sollten jetzt vier Paneele wie in Abb. 1 sehen. Speichern Sie die Datei als *tutorial.R* via **File** → **Save**. Von nun an sollten Sie alle *R*-Funktionsaufrufe in die Skriptdatei schreiben. Kommentieren Sie immer Ihren Code mittels dem Symbol `#`, damit Sie oder jemand anderes zu einem späteren Zeitpunkt nachvollziehen kann, was der Code macht.

Geben Sie im Editor-Panel *tutorial.R* als erste Zeile `z <- c(8,13,21)` und als zweite Zeile `2*z` ein.

Sie haben verschiedene Möglichkeiten *R*-Code an den *R*-Prozess zu senden:

1. Klicken Sie auf  **Source**. Der gesamte Code ihres Skriptes wird an die *R*-Konsole geschickt.
2. Bewegen Sie den Cursor zur ersten Zeile. Nun klicken Sie auf  **Run**. Dadurch wird nur die entsprechende Zeile an die *R*-Konsole geschickt und der Cursor springt zur nächsten Zeile. Wiederholen Sie den Klick auf  **Run** um die zweite Zeile an die *R*-Konsole zu schicken, etc.
3. Sie können auch bestimmte Teile des Codes mit der Maus auswählen und mit  **Run** an die *R*-Konsole senden.
4. Die Tastaturkombination für  **Run** ist “`<Ctrl>+<Enter>`” (d.h. `<Ctrl>` und `<Enter>` gleichzeitig drücken).

Bemerkung:

Es kann vorkommen, dass die Ausführung eines *R*-Skriptes zu lange dauert. Dies passiert oft, wenn man einen Fehler in einer Schleife hat. Sie können die Evaluation jederzeit abbrechen, indem Sie auf  klicken (dieser Button wird nur sichtbar, wenn *R* etwas am Berechnen ist). Alternativ kann man die `<Esc>`-Taste drücken, wenn man in der *R*-Konsole ist.

Arbeiten mit Vektoren

Geben Sie in der nächsten Zeile von *tutorial.R* `fib <- c(1,1,2,3,5,z)` ein (das ergibt die ersten acht Fibonacci-Zahlen). Schicken Sie diese Zeile nun an die *R*-Konsole und schauen Sie sich das Objekt `fib` an. Geben Sie in den folgenden Zeilen von *tutorial.R* `2*fib+1`, `fib*fib` und `log(fib)` ein. Wählen Sie die drei Zeilen mit der Maus aus und senden sie Sie an die *R*-Konsole. Dies wertet die drei Zeilen nacheinander aus. Schauen Sie sich die Resultate an. Alles klar?

Erstellen Sie nun die Folge 2, 4, 6 als Objekt `s`: `s<-2*(1:3)`, alternativ geht es auch mit `s<-seq(2,6,by=2)`. Schauen Sie sich verschiedene Elemente von `fib` an: `fib[3]`, `fib[4:7]`, `fib[s]`, `fib[c(3,5)]` und `fib[-c(3,5)]`.

Generieren Sie nun einen beliebigen Vektor `x` mit 8 Einträgen, von denen ein paar positiv und ein paar negativ sind. Jetzt schauen Sie sich `x > 0` und `fib[x > 0]` an.

Vergessen Sie nicht, Ihren Code zu kommentieren. Ihre Skriptdatei könnte jetzt etwa so aussehen:

```
## Mathematik IV: Statistik -- R tutorial
## Author: Hans Muster
## Date: 15 March 2012

## getting started
z <- c(8,13,21)
2*z

## computing with vectors
fib <- c(1,1,2,3,5,z)      # vector with first 8 Fibonacci numbers
fib
2*fib + 1                # element-wise operations
fib*fib                   # element-wise multiplication
log(fib)                  # takes the log of each element
s <- 2*(1:3)              # vector holding 2, 4, 6
s1 <- seq(2,6,by=2)       # same vector as s
fib[3]                    # 3rd element of vector fib
fib[4:7]                  # 4th, 5th, 6th and 7th element of fib
fib[s]                    # 2nd, 4th and 6th element of fib
fib[c(3,5)]               # elements 3 and 5 of fib
fib[-c(3,5)]              # vector fib without elements 3 and 5
x <- c(1,-3,5,-1,8,9,-2,1) # new vector x
x > 0                      # elements 1, 3, 5, 6 and 8 of x are > 0
fib[x > 0]                 # elements 1, 3, 5, 6 and 8 of fib
```

Matrizen: Erstellen und rechnen

Erstellen Sie zwei Vektoren `x <- 1:4` und `y <- 5:8`. Die Funktionsaufrufe `mat1 <- cbind(x,y)` und `mat2 <- rbind(x,y,x+y)` generieren zwei Matrizen (`cbind` bedeutet column-bind, `rbind` bedeutet row-bind). Schauen Sie sich die Matrizen mit `mat1` und `mat2` an. Was passiert, wenn Sie `mat2[3,2]`, `mat2[2,]` und `mat2[,1]` eingeben?

Berechnungen mit `+`, `*` etc. funktionieren für Matrizen genau gleich wie für Vektoren eintragweise. Falls Sie ein Matrixprodukt wollen, dann verwenden Sie `%*%`, z.B. `mat2 %*% mat1`.

Data Frames

Ein Data Frame ist eine verallgemeinerte Matrix. Der Hauptunterschied besteht darin, dass in einer Matrix alle Einträge vom gleichen Typ sein müssen (d.h. numerisch, kategoriell, ...) und bei einem Data Frame jede Spalte ein andere Typ sein kann.

Datensätze einlesen und anschauen

ASCII-Dateien werden am einfachsten mit der Funktion `read.table()` eingelesen. `read.table()` funktioniert auch für Datensätze vom Netz.

Zum Beispiel versuchen Sie folgendes:

```
no2 <- read.table("http://stat.ethz.ch/Teaching/Datasets/no2Basel.dat", header=TRUE)
```

Sie können nun den Datensatz direkt anschauen, indem Sie `no2` in der `R`-Konsole eingeben. Einzelne Spalten können mit `no2[, "NO2"]` (oder `no2$NO2`) aufgerufen werden. Wenn Sie sich die Originaldatei in einem Browser, z.B. Firefox, anschauen, dann sehen Sie, dass `R` aus der

ersten Zeile die Namen für die einzelnen Spalten extrahiert hat. Das Argument `header=TRUE` von `read.table()` sagt *R*, dass die Spaltennamen in der ersten Zeile der Originaldatei stehen. `no2` ist gerade noch klein genug, dass man es direkt in der Konsole aufrufen kann. Grundsätzlich ist es aber sinnvoll, Datensätze zuerst mit der Funktion `str()` anzuschauen, welche die Struktur des Objekt wiedergibt und nicht jedes einzelne Element anzeigt: `str(no2)`. `summary(no2)` zeigt Ihnen Informationen über die Spalten von `no2` an. `summary()` kann die wichtigsten Informationen aus einer Menge von verschiedenen *R*-Objekten extrahieren, z.B. die Resultate eines statistischen Tests oder einer Regression.

Grafiken

Zeichnen Sie ein Histogramm der `NO2` Daten in `no2`:

```
par(mfrow = c(1,2))      # Number of pictures one below the
                          # other [1] or side by side [2]
                          # Important to save paper!
hist(no2[, "NO2"])       # draw histogram.
```

Zeichnen Sie nun die Regressionsgerade des `NO2`-Gehalts gegen die Temperatur und illustrieren Sie dies neben dem Histogramm:

```
lm.T <- lm(NO2 ~ Temp, data = no2) # fits regression.
plot(NO2 ~ Temp, data = no2)
abline(lm.T, col = 4, lty = 2)     # col: colour; lty=2: dashed line
summary(lm.T)                     # regression summary
```

`title("Titel xy")` fügt Ihrer Grafik einen Titel hinzu. Mit dem Button  können Sie Ihre Grafik anschliessend als Bild oder PDF abspeichern.

Hilfe holen

Die *R*-Hilfe ermöglicht es Ihnen, Details über Funktionen herauszufinden. Schauen Sie sich die Hilfe zu der `plot()` Funktion an, indem Sie `help(plot)` oder `?plot` eingeben. Die Beispiele am Ende der Hilfe können Sie direkt mit der Funktion `example(plot)` ausführen lassen.

Wenn Sie Hilfe zu einem bestimmten Thema (z.B. Histogramme) suchen, aber nicht wissen, wie die entsprechende Funktion heisst, dann erhalten Sie mit `help.search("histogram")` eine Liste mit Funktionen, welche das entsprechende Schlüsselwort beinhalten.

R beenden

Sie können Ihre Arbeit sichern, indem Sie die Skriptdatei `tutorial.R` abspeichern. Wenn Sie erneut an Ihrem Skript arbeiten, müssen alle Objekte neu generiert werden, d.h. Sie müssen den gesamten Skript einmal an die *R*-Konsole senden. *R*-Objekte können auch mittel Funktionen wie `save()` und `write.table()` separat abgespeichert werden.

Die Funktion `q()` schliesst die *R*-Sitzung (genau gleich wie `File` → `Quit R`). Beantworten Sie die Frage *Save workspace image?* mit `n`.

Es gibt noch viel mehr...

R kann für komplexe Programme und Funktionen verwendet werden. Vielleicht schauen Sie sich die Funktionen `help(for)` und `help(function)` an, um einen kleinen Einblick zu gewinnen.