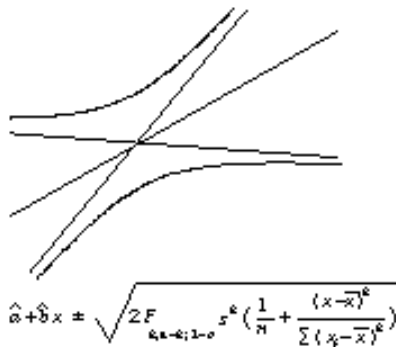


Statistical Computing: Einführung in S

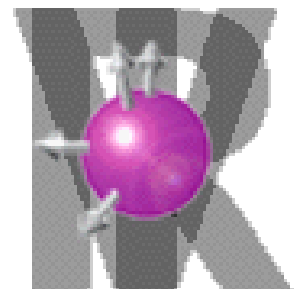
G. Sawitzki
<gs@statlab.uni-heidelberg.de>



Universität Heidelberg
<http://www.uni-heidelberg.de/>



StatLab Heidelberg
<http://www.statlab.uni-heidelberg.de/>
Stand: Tue, May 29, 2001



Virtuelle Universität Oberrhein
<http://www.viror.de/>

S als Programmiersprache: Übersicht

S ist eine interpretierte Ausdruckssprache. Ausdrücke sind zusammengesetzt aus Objekten und Operatoren.

Objekte haben zwei implizite Attribute, die erfragt werden mit `mode()` und `length()`. Die Funktion `typeof()` gibt den (internen) Speichermodus eines Objektes an.

A.1 Basis-Datentypen

<i>S Basis-Datentypen</i>			<i>Beispiele</i>
numeric	real oder integer	in R: real ist stets doppelt-genau. Einfache Genauigkeit wird für externe Aufrufe zu anderen Sprachen mit .C oder .Fortran unterstützt. Funktionen wie mode() und typedef() können je nach Implementierung auch den Speicherungsmodus (single, double...) melden.	<code>1.0</code> <code>2</code> <code>3.14E0</code>
complex	komplex, in cartesischen Koordinaten		<code>1.0+0i</code>
logical	TRUE, FALSE	in R: auch vordefinierte Variable T, F in S-Plus sind T und F Basis-Objekte.	
character	Zeichenketten	Delimiter sind alternativ " oder '.	<code>"T"</code> , <code>"abc"</code>
list			<code>list(1:10,"Hello")</code>
function	S-Funktion		<code>sin</code>
NULL	Spezialfall: leeres Objekt		<code>NULL</code>

A.1 Basis-Datentypen

Zusätzlich zu den Konstanten TRUE und FALSE gibt es drei spezielle Werte für Ausnahmesituationen:

<i>S spezielle Konstanten</i>			
TRUE	logical		alternativ: T
FALSE	logical		alternativ: F
NA	logical	“not available”	NA ist von TRUE und FALSE verschieden
NaN	numeric	“not a valid numeric value”. Implementationsabhängig. Sollte dem IEEE Standard 754 entsprechen.	Beispiel: 0/0
Inf	numeric	unendlich. Implementationsabhängig. Sollte dem IEEE Standard 754 entsprechen.	Beispiel: 1/0

Die Objekt-Attribute und weitere Eigenschaften können abgefragt oder mit Ausgaberoutinen angefordert werden. Die Ausgaberoutinen sind in der Regel polymorph, d.h. sie erscheinen in Varianten, die den jeweiligen Objekten angepaßt werden.

Die folgende Tabelle faßt die wichtigsten Informationsmöglichkeiten über Objekte sowie über die allgemeine Systemumgebung zusammen.

<i>S Inspektion</i>			
class()	Objekt-Klasse		
mode()	Speichermodus eines Objekts		
typeof()	Modus eines Objekts	Kann vom Speichermodus abweichen. Je nach Implementierung kann etwa eine numerische Variable standardmässig doppelt- oder einfach genau abgespeichert werden.	
length()	Länge = Anzahl der Elemente		
attributes()	Liest/setzt Attribute eines Objekts		
names()	Namen-Attribut für Elemente eines Objekts, z.B. eines Vektors.	<code>names(obj)</code> gibt das Namen-Attribut von obj <code>names(obj)<- charvec</code> setzt es	Beispiel: Benennung von Beobachtungen <code>x<- values</code> <code>names(x)<- charvec</code>
print()	Standard-Ausgabe		
summary()	Standard-Ausgabe als Übersicht, insbesondere für Modellanpassungen		
plot()	Standard-Graphikausgabe		
ls()	aktuelle Objekte		
methods()	generische Methoden	<code>methods(fun)</code> zeigt spezialisierte Funktionen zu fun, <code>methods(class=c)</code> die klassenspezifischen Funktionen zu class c.	<code>methods(plot)</code> <code>methods(class=lm)</code>
data()	zugreifbare Daten		
library()	zugreifbare Bibliotheken		
help()	allgemeines Hilfe-System		

A.2 Zugriff auf Komponenten

S ist vektor-basiert. Einzelne Konstanten oder Werte sind nur Vektoren der speziellen Länge 1. Sie genießen keine Sonderbehandlung.

Die Länge von Vektoren ist ein dynamisches Attribut. Sie wird bei Bedarf erweitert und gekürzt. Insbesondere gilt implizit eine "Recycling-Regel": Hat ein Vektor nicht die erforderliche Länge für eine Operation, so wird er periodisch bis zur erforderlichen Länge wiederholt.

Auf Vektor-Komponenten kann über Indizes zugegriffen werden. Die Indizes können explizit oder als Regel-Ausdruck angegeben werden.

<i>S Index-Zugriff</i>			<i>Beispiele</i>
x[indices]	Indizierte Komponenten von x		<code>x[1:3]</code>
x[-indices]	x ohne indizierte Komponenten		<code>x[-3]</code>
x[condition]	Komponenten von x, für die <i>condition</i> gilt.		<code>x[x<0.5]</code>

Vektoren (und andere Objekte) können auf höherdimensionale Konstrukte abgebildet werden. Die Abbildung wird durch zusätzliche Dimensions-Attribute beschrieben. Nach Konvention erfolgt eine spaltenweise Einbettung, d.h. der höchste Index variiert zuerst (FORTRAN-Konvention). Operatoren und Funktionen können die Dimensions-Attribute auswerten.

<i>S Index-Zugriff</i>			<i>Beispiele</i>
dim()	Setzt oder liest die Dimensionen eines Objekts		<code>x <- 1:12 ; dim(x) <- c(3,4)</code>
dimnames()	Setzt oder liest Namen für die Dimensionen eines Objekts		
nrow()	Gibt die Anzahl der Zeilen= Dimension 1		
ncol()	Gibt die Anzahl der Spalten= Dimension 2		
matrix()	Erzeugt eine Matrix mit vorgegebenen Spezifikationen	<code>matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)</code>	
array()	Erzeugt eine evtl. höherdimensionale Matrix	<code>array(x, dim=length(x), dimnames=NULL)</code>	

<i>S Array-Zugriffe</i>			<i>Beispiele</i>
cbind() rbind()	Verkettet Zeilen oder Spalten		
split()	Teilt einen Vektor nach Faktoren auf		
table()	Erzeugt eine Tabelle von Besetzungszahlen		

<i>S Iteratoren</i>			<i>Beispiele</i>
apply()	wendet eine Funktion auf die Zeilen oder Spalten einer Matrix an	<code>apply(x, MARGIN, FUNCTION, ...)</code> Margin=1: Zeilen, =2:Spalten	
tapply()	wendet eine Funktion auf Komponenten eines Objekts in Abhängigkeit von einer Liste von kontrollierende Faktor an.		
outer()	erzeugt eine Matrix mit allen Paar-Kombinationen aus zwei Vektoren, und wendet eine Funktion auf jedes Paar an.	<code>outer(vec1,vec2, FUNCTION,...)</code>	

A.3 Zusammengesetzte Datentypen

<i>Zusammengesetzte Objekttypen</i>		<i>Spezielle Funktionen</i>	<i>Beispiele</i>
Vektoren			
Matrizen	Vektoren mit zwei-dimensionalem Layout		
Arrays	Vektoren mit höher-dimensionalem Layout	<code>dim()</code> definiert Dimensionsvektor <code>array()</code> konstruiert neuen Vektor mit gegebener Dimensionsstruktur <code>cbind()</code> kettet Spalten an <code>rbind()</code> kettet Reihen an	<pre>x<- runif(100) dim(x) <- c(5,5,4) z <- array(0,c(4,3,2))</pre>
Faktoren	Sonderfall für kategoriale Daten	<code>factor()</code> wandelt Vektor in Faktor um <code>ordered()</code> analog Vektor für geordnete Kategorien <code>levels()</code> <code>tapply()</code> wendet eine Funktion getrennt für alle Stufen von Faktoren einer Faktorliste an <code>table()</code>	<pre>> x<- c("a", "b", "a", "c", "a") > xf<- factor(x) > levels(xf) [1] "a" "b" "c"</pre>
Listen	Analog Vektoren, mit Elementen auch unterschiedlichen Typs	<code>list(<Komponenten>)</code> <code>[[]]</code> Indexweiser Zugriff auf Komponenten <code>Liste\$Komponente</code> Zugriff nach Namen	<pre>l <- list(name="xyz", age=22, fak="math") >l[[2]] 22 >l\$age 22</pre>
Datenrahmen data frames	Analog Arrays bzw. Listen, mit spaltenweise einheitlichem Typ und einheitlicher Spaltenlänge	<code>data.frame()</code> analog <code>list()</code> , aber Restriktionen müssen erfüllt sein. <code>attach()</code> fügt Datenrahmen in die aktuelle Suchliste ein, d.h. für Komponenten reicht der Komponentenname. <code>detach()</code>	

A.4 Operatoren

Ausdrücke in S können aus Objekten und Operatoren zusammengesetzt sein. Die folgende Tabelle ist nach Vorrang geordnet (höchster Rang oben).

S Basisoperatoren			Beispiele
\$	Komponenten-Selektion		<code>list\$item</code>
[[[Indizierung, Elementzugriff		<code>x[i]</code>
^	Potenzierung		<code>x^3</code>
-	unitäres Minus		
:	Folge-Generierung		<code>1:5</code>
%<name>%	spezielle Operatoren	können auch benutzer-definiert sein	
* /	Multiplikation, Division		
+ -	Addition, Subtraktion		
< > <= >= == !=	Vergleichsoperatoren		
!	Negation		
& &&	und, oder	&&, sind "Shortcut"-Operatoren	
<- ->	Zuweisung	varaltet: _ anstelle von <-	

Operatoren der Form %<name>% können vom Benutzer definiert werden. Die Definition folgt den Regeln für Funktionen.

Ausdrücke können als Folge mit trennendem Semikolon geschrieben werden.
Ausdrucksgruppen können durch { ... } zusammengefasst werden.

A.5 Funktionen

Funktionen sind spezielle Objekte. Funktionen können Resultat-Objekte übergeben.

<i>S Funktionen</i>			<i>Beispiele</i>
Deklaration	function (<Formale Parameterliste>) <Ausdruck>		<code>fak <- function(n) prod(1:n)</code>
Formale Parameterliste	Liste von formalen Parametern, durch Komma getrennt		<code>n, mean=0, sd=1</code>
Formale Parameter	<Parametername> <Parametername> = <Default- Wert>		
...	...	Variable Parameterliste. Variable Parameter- listen können innerhalb von Prozeduren weiter- gegeben werden.	<code>mean.of.all <- function(...) mean(c(...))</code>
Funktions- Resultate	<code>return</code> <Wert>	bricht Funktionsaus- wertung ab und übergibt Wert	
Funktions- Resultate	<Wert>	als letzter Ausdruck in einer Funktionsde- klaration: übergibt Wert	
Funktions- Resultate	<Variable> <code><<-</code> <Wert>	übergibt Wert. Nor- malerweise wirken Zu- weisungen nur auf lokale Kopien der Va- riablen. Die Zuweisung mit <code><<-</code> jedoch sucht die Zielvariable in der gesamten Umgebungs- hierarchie.	
Funktionsaufruf	<Name>(<Aktuelle Parameterliste>)		<code>fak(3)</code>
Aktuelle Parameterliste	<Werteliste> <Parametername> = <Werte>	Werte werden zunächst der Position nach zuge- ordnet. Abweichend davon können Namen benutzt werden, um Werte gezielt zuzu- ordnen. Dabei reichen die Anfangsteile der Namen	<code>rnorm(10, sd=2)</code>

Spezialfall: Funktionen mit Namen der Form `xxx<-` erweitern die Zuweisungsfunktion. Beispiel:

```
"inc<-" <- function (x,value) x+value
```

```
> x <- 10  
> inc(x)<- 3  
> x  
13
```

In R **muss** das Wert-Argument “value” heissen.

A.6 Kontrollstrukturen

Kontrollstrukturen wie Schleifen etc. werden in S weitgehend vermieden. Anstelle dessen werden nach Möglichkeit vektor-orientierte Prozeduren eingesetzt. Explizite Kontrollstrukturen sind die Ausnahme.

<i>S Kontrollstrukturen</i>			<i>Beispiele</i>
if	<code>if</code> (log. Ausdruck1) .Ausdruck2	Der logische Ausdruck 1 darf nur einen logischen Wert ergeben.	
	<code>if</code> (log. Ausdruck1) .Ausdruck2 <code>else</code> Ausdruck3		
ifelse	<code>ifelse</code> (log. Ausdruck1, Ausdruck2, Ausdruck3)	Wertet den logischen Ausdruck 1 elementweise auf einen Vektor an, und übergibt bei wahrem Resultat den elementweisen Wert von Ausdruck2, sonst von Ausdruck3)	<code>trimmedX <- ifelse(abs(x)<2, X, 2)</code>
switch	<code>switch</code> (Ausdruck1, ...)	Ausdruck1 muss einen numerischen Wert oder eine Zeichenkette ergeben. ... ist eine explizite Liste der Alternativen.	<code>centre <- function(x, type) {switch(type, mean = mean(x), median = median(x), trimmed = mean(x, trim = .1))}</code>
for	<code>for</code> (name <code>in</code> Ausdruck1) Ausdruck2		
repeat	<code>repeat</code> Ausdruck		<code>pars<-init repeat { res <- get.resid (data,pars) if (converged(res)) break pars<- new.fit(data,pars)}</code>
while	<code>while</code> (log. Ausdruck) Ausdruck		<code>pars<-init; res <- get.resid (data,pars) while (!converged(res)) { pars<- new.fit(data,pars) res <- get.resid}</code>
break		verlässt die aktuelle Schleife	
next		verlässt einen Schleifenzyklus und springt zum nächsten	

A.7 Verwaltung und Anpassung

			<i>Beispiele</i>
objects() ls()	Liste der aktuellen Objekte		
rm()	<code>rm(<Objektliste>)</code>	Löscht die angegebenen Objekte	
source()	<code>source(" <Dateiname>")</code>	Führt die R-Anweisungen aus der angegebene Datei aus.	<code>source("cmds.R")</code>
sink()	<code>sink(" <Dateiname>")</code>	Lenkt Ausgaben in die angegebene Datei.	<code>sink()</code> lenkt die Ausgabe wieder auf die Konsole.

A.8 Ein- und Ausgabe

<i>S Ein/Ausgabe</i>			<i>Beispiele</i>
write		<code>write(val,file)</code>	<code>write(x,file="data")</code>
source			
sink			
dump	Schreibt für ein Objekt die definierenden Kommandos. Mit <code>source()</code> kann aus der Ausgabe das Objekt regeneriert werden	<code>dump(list, file="dumpdata.R", append=FALSE)</code>	<code>"T", "abc"</code>
list			<code>list(1:10,"Hello")</code>
function	S-Funktion		<code>sin</code>
null	Spezialfall: leeres Objekt		<code>NULL</code>

A.9 Externe Daten, Libraries, Packages

Externe Information kann in (Text)-Dateien und Paketen(Packages) gespeichert sein. Bibliotheken und Pakete sind dabei nach speziellen R-Konventionen strukturiert. "Bibliotheken" sind Sammlungen von "Paketen".

Vorbereitete Daten werden üblicherweise als data frames bereitgestellt; zusätzliche Objekte können aber auch damit verbunden sein. Die online-help-Funktion gibt in der Regel weitere Auskunft.

Daten werden mit der Funktion

```
data()
```

geladen.

Syntax:

```
data(..., list = character(0), package = c(.packages(),.Autoloaded),
      lib.loc = .lib.loc)
```

Beispiel:

```
> data(crimes)           # lädt den Datensatz 'crimes'
```

Zusätzliche Funktionen werden in der Regel als Pakete bereitgestellt. Pakete werden mit

```
library()
```

geladen. Im Paket enthaltene Datensätze sind dann direkt auffindbar und werden mit

```
data()
```

(ohne Argument) aufgelistet.

Beispiel:

```
library(nls)
data()
data(Puromycin)
```

Externe Daten werden in der Regel als Text-Dateien bereitgestellt, nach Möglichkeit

in Tabellenform,

nur ASCII-Zeichen (z.B. keine Umlaute!)

Variablen spaltenweise angeordnet

Spalten durch Tabulator-Sprünge getrennt.

evtl. Spaltenüberschriften in Zeile 1

evtl. Zeilennr. in Spalte 1

Dafür wird die Funktion

```
read.table()
```

bereitgestellt. Wichtige Spezialfälle:

```
read.table(<Dateiname>)           # Überschriften in Zeile 1,
                                   # Zeilennr. in Spalte 1

read.table(<Dateiname>, header=TRUE) # keine Zeilennr.,
                                   # Überschriften in Zeile 1,
```

Zum editieren nach Spreadsheet-Art gibt es `edit()` (früherer Name: `data.entry()`).

Für sequentielles Lesen steht `scan()` zur Verfügung.

Dateien mit stengenau fest vorgegebenem Format können mit `read.fwf()` gelesen werden.

Zum Import aus Datenbanken und anderen Paketen steht je nach Implementierung eine Reihe von Bibliotheken zu Verfügung, z.B. `stataread` für Stata, `foreign` für SAS, Minitab und SPSS, `RODBC` für SQL. Weitere Information findet sich im Manual "R Data Import/Export".

A.10 Modell-Beschreibungen

Lineare statistische Modelle können durch Angabe einer Design-Matrix X spezifiziert werden und in der allgemeinen Form

$$Y = X \cdot \beta + \text{err}$$

dargestellt werden, wobei die Matrix X jeweils genauer bestimmt werden muß.

S erlaubt es, Modelle auch dadurch zu spezifizieren, dass die Regeln angegeben werden, nach denen die Design-Matrix gebildet wird.

Die Modell-Spezifikation ist auch für allgemeinere, nicht lineare Modelle möglich.

Beispiele

$y \sim 1 + x$	entspricht $y_i = (1 \ x_i) \begin{pmatrix} \beta_1 & \beta_2 \end{pmatrix}' + \text{err}$
$y \sim x$	Kurzschreibweise für $y \sim 1 + x$ (Konstanter Term wird implizit angenommen)
$y \sim 0 + x$	entspricht $y_i = x_i \beta_1 + \text{err}$
$\log(y) \sim x_1 + x_2$	entspricht $\log(y_i) = (1 \ x_{i1} \ x_{i2}) \begin{pmatrix} \beta_1 & \beta_2 & \beta_3 \end{pmatrix}' + \text{err}$ (Konstanter Term wird implizit angenommen)
$y \sim A$	Einweg-Varianzanalyse mit Faktor A
$y \sim A + x$	Covarianzanalyse mit Faktor A und Covariable x
$y \sim A*B$	Zwei-Faktor-Kreuzklassifikation mit Faktoren A und B
$y \sim A/B$	Zwei-Faktor hierarchische Klassifikation mit Faktoren A und Subfaktor B

Um zwischen verschiedenen Modellen ökonomisch wechseln zu können, steht die Funktion `update()` zur Verfügung.

Anwendungsbeispiel:

```
lm(y ~ poly(x,4), data = experiment)
```

analysiert den Datensatz "experiment" mit einem linearen Modell für polynomiale Regression vom Grade 4.

<i>Standard-Analysen</i>			
lm()	lineares Modell		
glm()	generalisiertes lineares Modell		
nls()	nicht-lineare kleinste Quadrate		
nlm()	Maximum likelihood		
update()	Wechsel zwischen Modellen		
anova()	Varianz-Analyse		

A.11 Grafik-Funktionen

Grafik-Funktionen fallen im wesentlichen in drei Gruppen:

- "high level"-Funktionen. Diese definieren eine neue Ausgabe.
- "low level"-Funktionen. Diese ergänzen oder modifizieren eine "high level"-Ausgabe.
- Parametrisierungen. Diese modifizieren die Voreinstellungen des Grafik-Systems.

<i>"high level"</i>			<i>Beispiele</i>
plot()	Generische Grafikfunktion		
pairs()	paarweise Scatterplots		
coplot()	Scatterplots, bedingt auf Covariable		
qqplot()			
qqline()			
qqnorm()			
hist()	Histogramm		
boxplot()			
dotplot()			
image()	farbcodiertes z gegen x,y		
contour()	Contourplot von z gegen x,y		
persp()	3D-Fläche		

<i>"low level"</i>			<i>Beispiele</i>
points()		<code>points(x, ...)</code>	
lines()		<code>lines(x, ...)</code>	
abline()		<code>abline(a, b, ...)</code>	
text()		<code>text (x, ...)</code>	
polygon()			
legend()	Fügt eine Legende hinzu.		
title()	Setz Überschrift.		
axis()	Fügt Achsen hinzu.		
mtext()	Fügt Seitenbeschriftung hinzu.		

Daneben hat S rudimentäre Möglichkeiten für Interaktion mit Grafik.

<i>Interaktionen</i>			<i>Beispiele</i>
locator()	bestimmt die Position von Mausclicks	eine aktuelle Graphik muss definiert sein, bevor <code>locator()</code> benutzt wird.	<code>plot(runif(19)); locator(n = 3,type = "l")</code>

`Text()` gibt auch eingeschränkte Möglichkeiten zum Formelsatz (implementierungsabhängig). Falls implementiert, erhält man eine Übersicht mit `help(plotmath)`

Beispiel:

```
text(x,y, expression(paste(bgroup("(", atop(n,x),"),"), p^x, q^{n-x})))
```

<i>Parametrisierungen</i>			<i>Beispiele</i>
par()			

A.12 Einfache Statistische Funktionen

<i>Statistik-Funktionen</i>			<i>Beispiele</i>
sum()	summiert Komponenten eines Vektors		
cumsum()	bildet kumulierte Summen		
prod()	multipliziert Komponenten eines Vektors		
cumprod()	bildet kumulierte Produkte		
length()	Länge eines Objekts, z.B. Vektors		
max() min()	Maximum, Minimum	Siehe auch pmax, pmin	
range()	Minimum und Maximum		
cummax() cummin()	Kumulatives Maximum, Minimum		
quantile()	Stichprobenquantile	Für theoretische Verteilungen: qxxxx, z.B. qnorm	
median()	Median		
mean()	Mittelwert	auch getrimmte Mittel	
var()	Varianz, Varianz/Covarianzmatrix		
sort() rev()	Sortierung		
order()	Sortierung nach Leit-Element		
rank()	Stichprobenränge		

A.13 Verteilungen, Zufallszahlen, Dichten...

Der Basis-Generator für uniforme Zufallszahlen wird von *Random* verwaltet. Verschiedene Basis-Generatoren stehen zur Verfügung. **Für ernsthafte Simulation wird eine Lektüre der Empfehlungen von Marsaglia et al. dringend empfohlen.** Siehe *help(Random)*. Alle nicht-uniformen Zufallszahlengeneratoren sind vom aktuellen Basisgenerator abgeleitet. Eine Übersicht über die wichtigsten nicht-uniformen Zufallszahlengeneratoren, ihre Verteilungsfunktionen und ihre Quantile findet sich unten.

<i>S Zufallszahlen</i>			<i>Beispiele</i>
RngKind	<i>RngKind()</i> <i>RngKind(<name>)</i>	RngKind() gibt den Namen des aktuellen Basisgenerators. RngKind(<name>) setzt einen Basis-generator.	<i>RngKind("Wichmann-Hill")</i> <i>RngKind("Marsaglia-Multicarry")</i> <i>RngKind("Super-Duper")</i>
sample	<i>sample(x, size, replace=FALSE, prob)</i>	<i>sample</i> zieht eine Zufallsstichprobe aus den im Vektor x angegebenen Werten, mit oder ohne Zurücklegen (je nach Wert von replace). Size ist defaultmäßig die Länge von x. Optional kann prob ein Vektor von Wahrscheinlichkeiten für die Werte von x sein.	<i>sample(x,10,T)</i> Zufällige Permutation: <i>sample(x)</i> <i>val<-c("H","T")</i> <i>prob<-c(0.3,0.7)</i> <i>sample(val,10,replace=T,prob)</i>

Die einzelnen Funktionsnamen für die wichtigsten nicht-uniformen Generatoren und Funktionen setzen sich aus einem Präfix und dem Kurznamen zusammen. Allgemeiner Schlüssel: *xxxx* ist der Kurzname

<i>xxxx</i>	erzeugt Zufallszahlen
<i>dx.xx</i>	Dichte oder Wahrscheinlichkeit
<i>px.xx</i>	Verteilungsfunktion
<i>qx.xx</i>	Quantile

Beispiel:

x<-runif(100) erzeugt 100 U(0,1)-verteilte Zufallsvariable
qf(0.95,10,2) berechnet das 95%-Quantil der F(10, 2)-Verteilung.

Verteilungen	Kurzname	Parameter & Defaults
Beta	beta	<i>shape1, shape2, ncp=0</i>
Binomial	binom	<i>size, prob</i>
Cauchy	cauchy	<i>location = 0, scale = 1</i>
2	chisq	<i>df, ncp=0</i>
Exponential	exp	<i>rate = 1</i>
F	f	<i>df1, df2 (ncp=0)</i>
Gamma	gamma	<i>shape, scale=1</i>
Gauss	norm	<i>mean=0, sd=1</i>
Geometrisch	geom	<i>prob</i>
Hypergeometrisch	hyper	<i>m, n, k</i>
Lognormal	lnorm	<i>meanlog = 0, sdlog = 1</i>
Logistisch	logis	<i>location = 0, scale = 1</i>
Negativ-Binomial	nbinom	<i>size, prob</i>
Poisson	pois	<i>lambda</i>
Student's t	t	<i>df</i>
Tukey Studentised Range	tukey	
Uniform	unif	<i>min=0, max=1</i>
Wilcoxon Signed Rank	signrank	<i>n</i>
Wilcoxon Rank Sum	wilcox	<i>m, n</i>
Weibull	weibull	<i>shape, scale = 1</i>