5 Wahrscheinlichkeitsverteilungen und Pseudo-Zufallszahlen

R hat bereits in der "base distribution" für viele Wahrscheinlichkeitsverteilungen sowohl die jeweilige Dichtefunktion (im Fall einer stetigen Verteilung) bzw. Wahrscheinlichkeitsfunktion (im diskreten Fall) als auch Verteilungs- und Quantilfunktion implementiert; fallweise natürlich nur approximativ. Ebenso sind Generatoren für Pseudo-Zufallszahlen implementiert, mit denen Stichproben aus diesen Verteilungen simuliert werden können. Wenn wir im Folgenden (zur Abkürzung) von der Erzeugung von Zufallszahlen sprechen, meinen wir stets *Pseudo*-Zufallszahlen. (Beachte: In der **R**-Terminologie wird nicht zwischen Dichtefunktion einer stetigen Verteilung und Wahrscheinlichkeitsfunktion einer diskreten Verteilung unterschieden, sondern beides als "density function" bezeichnet.)

5.1 Die „eingebauten" Verteilungen

Die **R**-Namen der obigen vier Funktions*typen* setzen sich wie folgt zusammen: Ihr erster Buchstabe gibt den Funktionstyp an, der Rest des Namens identifiziert die Verteilung, für die dieser Typ zu realisieren ist. Der Aufruf eines Funktionstyps einer Verteilung benötigt (neben eventuell anzugebenden spezifischen Verteilungsparametern) die Angabe der Stelle, an der eine Funktion ausgewertet werden soll, bzw. die Anzahl der zu generierenden Zufallszahlen. Etwas formaler gilt allgemein für eine beliebige, implementierte Verteilung *dist* (mit möglichen Parametern „. . . .") mit Verteilungsfunktion (VF) $F_{....}$ und Dichte- bzw. Wahrscheinlichkeitsfunktion $f_{....}$:

`d`*`dist`*`( x, ....)`	$= f_{....}(x)$	: Dichte-/Wahrscheinlichkeitsfkt.
`p`*`dist`*`( x, ....)`	$= F_{....}(x)$	: VF (also $\mathbb{P}(X \leq x)$ für $X \sim F$)
`q`*`dist`*`( y, ....)`	$= F^{-1}_{....}(y)$	: Quantilfunktion
`r`*`dist`*`( n, ....)`	liefert n Zufallszahlen aus	

} der Verteilung *dist*

Beispiele: Für den stetigen Fall betrachten wir die (Standard-)Normalverteilung:

`dnorm( x)`	$= \phi(x)$	: Dichte
`pnorm( x)`	$= \Phi(x)$	: VF
`qnorm( y)`	$= \Phi^{-1}(y)$	: Quantilfunktion
`rnorm( n)`	liefert n Zufallszahlen aus	

} der Standardnormalverteilung

Gemäß der Voreinstellung liefern Aufrufe der Funktionstypen aus der Familie der Normalverteilung immer Resultate für die *Standard*normalverteilung, wenn die Parameter `mean` und `sd` nicht explizit mit Werten versehen werden:

Die Normalverteilung	
`> dnorm( c( 7, 8), mean = 10)` `[1] 0.004431848 0.053990967`	Werte der Dichte der $\mathcal{N}(10, 1)$-Verteilung an den Stellen 7 und 8.
`> pnorm( c( 1.5, 1.96))` `[1] 0.9331928 0.9750021`	Werte der Standardnormalverteilungsfunktion Φ an den Stellen 1.5 und 1.96.
`> qnorm( c( 0.05, 0.975))` `[1] -1.644854  1.959964`	Das 0.05- und das 0.975-Quantil der Standardnormalverteilung, also $\Phi^{-1}(0.05)$ und $\Phi^{-1}(0.975)$.

`> rnorm( 6, mean = 5, sd = 2)`
`[1] 5.512648 8.121941 5.672748`
`[4] 7.665314 4.081352 4.632989`
	Sechs Zufallszahlen aus der $\mathcal{N}(5, 2^2)$-Verteilung.

Für den diskreten Fall betrachten wir die Binomialverteilung:

`dbinom( k, size =` m`, prob =` p`)`	$= \mathbb{P}(X = k)$	: W.-Funktion	} ...
`pbinom( k, size =` m`, prob =` p`)`	$= F(k) := \mathbb{P}(X \leq k)$	: VF	
`qbinom( y, size =` m`, prob =` p`)`	$= F^{-1}(y)$	: Quantilfunktion	
`rbinom( n, size =` m`, prob =` p`)`	liefert n Zufallszahlen aus		

... der Binomial(m, p)-Verteilung (d. h. für $X \sim \text{Bin}(m, p)$).

Hier zwei Tabellen vieler der in **R** zur Verfügung stehenden Verteilungen, ihrer jeweiligen Funktionsnamen und Parameter (samt Voreinstellungen, sofern gegeben; beachte auch die Ergänzung auf Seite 89 oben):

Diskrete Verteilungen		
... (-)Verteilung	**R**-Name	Verteilungsparameter
Binomial	`binom`	`size, prob`
Geometrische	`geom`	`prob`
Hypergeometrische	`hyper`	`m, n, k`
Multinomial	`multinom`	`size, prob` (nur `r....` und `d....`)
Negative Binomial	`nbinom`	`size, prob`
Poisson	`pois`	`lambda`
Wilcoxons Vorzeichen-Rangsummen	`signrank`	`n`
Wilcoxons Rangsummen	`wilcox`	`m, n`

Stetige Verteilungen		
... (-)Verteilung	**R**-Name	Verteilungsparameter
Beta	`beta`	`shape1, shape2, ncp = 0`
Cauchy	`cauchy`	`location = 0, scale = 1`
χ^2	`chisq`	`df, ncp = 0`
Exponential	`exp`	`rate = 1`
F	`f`	`df1, df2 (ncp = 0)`
Gamma	`gamma`	`shape, rate = 1`
Log-Normal	`lnorm`	`meanlog = 0, sdlog = 1`
Logistische	`logis`	`location = 0, scale = 1`
Multivariate Normal (im package `mvtnorm`)	`mvnorm`	`mean = rep(0,`*`d`*`), sigma = diag(`*`d`*`)` (mit *`d`* = Dimension)
Multivariate t (im package `mvtnorm`)	`mvt`	(etwas komplizierter; siehe seine Online-Hilfe)
Normal	`norm`	`mean = 0, sd = 1`
Students t	`t`	`df, ncp = 0`
Uniforme	`unif`	`min = 0, max = 1`
Weibull	`weibull`	`shape, scale = 1`

Hinweise: Mehr Informationen über die einzelnen Verteilungen – wie z. B. die Beziehung zwischen den obigen **R**-Funktionsargumenten und der mathematischen Parametrisierung der Dichten – liefert die Online-Hilfe, die etwa mit dem Kommando `?d`*dist* konsultiert werden kann, wenn etwas über die Verteilung namens *dist* (vgl. obige Tabelle) in Erfahrung gebracht werden soll. Beachte: `?`*dist* funktioniert *nicht* oder liefert nicht das Gewünschte! Aber `?distribution` liefert eine Übersicht über alle Verteilungen im Paket `stats` mit Links auf deren Hilfe-Seiten. (Zusätzliche Möglichkeit: Mittels `help.search( "distribution")` erhält man u. A. Hinweise

auf alle Hilfedateien, in denen etwas zum Stichwort "distribution" steht.) Ist man an einem umfangreichen Überblick über die in "base-**R**" und anderen **R**-Paketen zur Verfügung gestellten Verteilungen interessiert, lohnt sich ein Blick auf den Task View "Probability Distributions" auf CRAN unter `http://cran.r-project.org` → Task Views → Distributions.

Ergänzung: Urnenmodelle sind in **R** ebenfalls realisierbar, und zwar mithilfe der Funktion `sample()`. Mit ihr kann das Ziehen von Elementen aus einer (endlichen) Grundmenge mit oder ohne Zurücklegen simuliert werden. Es kann auch festgelegt werden, mit welcher Wahrscheinlichkeit die einzelnen Elemente der Grundmenge jeweils gezogen werden sollen, wobei die Bedeutung dieser Festlegung beim Ziehen mit Zurücklegen eine andere ist als beim Ziehen ohne Zurücklegen. Die Voreinstellung von `sample()` produziert eine zufällige Permutation der Elemente der Grundmenge. In Abschnitt 9.1 „Bernoulli-Experimente mit `sample()`" gehen wir etwas näher darauf ein; vorerst verweisen wir für Details auf die Online-Hilfe.

5.2 Bemerkungen zu Pseudo-Zufallszahlen in R

Die Generierung von Pseudo-Zufallszahlen aus den verschiedenen Verteilungen basiert auf uniformen Pseudo-Zufallszahlen, welche mit einem Zufallszahlengenerator (Engl.: "random number generator" = RNG) erzeugt werden. In **R** stehen im Prinzip mehrere verschiedene RNGs zur Verfügung. Per Voreinstellung ist es der „Mersenne-Twister", ein uniformer RNG, der eine sehr lange, aber letztendlich doch periodische Zahlensequenz (der Länge $2^{19937} - 1$) im offenen Intervall $(0, 1)$ erzeugt. Diesbzgl. wird in der Online-Hilfe die Publikation [35, Matsumoto und Nishimura (1998)] zitiert. Weitere Details und zahlreiche Literaturverweise liefert `?RNG`.

Der Zustand des RNGs wird von **R** in dem Objekt `.Random.seed` im workspace (gewissermaßen unsichtbar) gespeichert. Vor dem **aller**ersten Aufruf des RNGs existiert `.Random.seed` jedoch noch nicht, z. B. wenn **R** in einem neuen Verzeichnis zum ersten Mal gestartet wird. Den Startzustand des RNGs leitet **R** dann aus der aktuellen Uhrzeit ab, zu der der RNG zum ersten Mal aufgerufen wird. Bei der Erzeugung von Zufallszahlen ändert sich der Zustand des RNGs und der jeweils aktuelle wird in `.Random.seed` dokumentiert. Daher führen wiederholte Aufrufe des RNGs, insbesondere in verschiedenen **R**-Sitzungen, zu verschiedenen Zufallszahlen(folgen).

Für die Überprüfung und den Vergleich von Simulationen ist es jedoch notwendig, Folgen von Zufallszahlen reproduzieren (!) zu können. Um dies zu erreichen, kann der Zustand des RNGs von der Benutzerin oder dem Benutzer gewählt werden, wozu die Funktion `set.seed()` dient. Wird sie vor dem Aufruf einer der obigen `r....`-Funktionen (und unter Verwendung desselben RNGs) jedesmal mit demselben Argument (einem `integer`-Wert) aufgerufen, so erhält man stets die gleiche Folge an Zufallszahlen. Beispiel:

```
> runif( 3)                          # Drei uniforme Zufallszahlen (aus dem
[1] 0.1380366 0.8974895 0.6577632    # RNG mit unbekanntem Startzustand).

> set.seed( 42)                      # Wahl eines Startzustandes des RNGs.
> runif( 3)                          # Drei weitere uniforme Zufallszahlen.
[1] 0.9148060 0.9370754 0.2861395

> set.seed( 42)                      # Wahl *desselben* RNG-Startzustandes
> runif( 3)                          # wie eben => Replizierung der drei
[1] 0.9148060 0.9370754 0.2861395    # uniformen Zufallszahlen von eben.

> runif( 3)                          # Ausgehend vom aktuellen (uns "unbe-
[1] 0.8304476 0.6417455 0.5190959    # kannten") Zustand liefert der RNG
                                     # drei andere uniforme Zufallszahlen.
```