

Zufallszahlen in R

Wiederholung

Diskutieren Sie die folgenden Fragen mit Ihrem Nachbarn oder in einer Kleingruppe. Wählen Sie für Erklärungen am besten ein konkretes Beispiel.

- Wann muss man, salopp gesprochen, Wahrscheinlichkeiten addieren und wann multiplizieren?
- Worin unterscheiden sich der klassische, der statistische und der subjektive Wahrscheinlichkeitsbegriff? Was sind ihre Gemeinsamkeiten?
- Was muss man sich unter einer diskreten Zufallsvariable vorstellen?

Zufallszahlen in R

Ein fairer Würfel liefert gleichverteilte Zufallszahlen: Jede der Zahlen 1 bis 6 erscheint mit gleicher Wahrscheinlichkeit. Die entsprechende Verteilung ist diskrete. Um eine diskrete gleichverteilte Zahlenfolge zu erzeugen, bietet R die Funktion `sample()`. Auf diesem Übungsblatt werden Sie ausschliesslich diskrete zufällige Ereignisse aus Laplace-Wahrscheinlichkeitsmodellen simulieren.

Die verschiedenen Verteilungen, die es in R gibt, werden wir in den nächsten Übungsserien noch kennenlernen. Um gleichförmig verteilte Zufallszahlen zu erzeugen, kann man z.B. die Funktion `runif()` verwenden. Sie erzeugt standardmässig Zahlen zwischen 0 und 1. Das Intervall lässt sich aber nach Belieben anpassen.

Die Funktion `sample()` besitzt 4 Argumente, von denen zwei fakultativ sind. Beginnen wir zunächst mit den obligatorischen Argumenten. Das erste Argument `x` enthält die Gesamtheit, aus der gezogen werden soll. Man kann für `x` auch eine natürliche Zahl `n` eingeben. In diesem Fall wird aus den natürlichen Zahlen $1, \dots, n$ gezogen. Das zweite Argument `size` gibt den Stichprobenumfang an. Der Aufruf

```
> sample(x = 6, size=1)
```

entspricht z.B. dem einmaligen Wurf eines fairen Würfels. Das Argument `x = 6` gibt an, dass wir Zahlen von 1 bis 6 *gleichverteilt* ziehen wollen. Wir können natürlich auch mehrere Zufallszahlen erzeugen. Diese sind unabhängig, wenn das Argument `replace` der Funktion `sample()` den Wert `TRUE` annimmt (entspricht dem Fall Ziehen mit Zurücklegen). Standardmässig steht es auf `FALSE`. Um zweimal zu würfeln, geben wir also ein

```
> sample(x = 6, size=2, replace=TRUE)
```

Eine Sequenz von 10 Würfelzahlen erhalten wir mit dem einzigen Funktionsaufruf

```
> sample(x = 6, size=10, replace=TRUE)
```

Probieren Sie den Befehl in der Kommandozeile von R aus: Wenn Sie ihn mehrmals nacheinander ausführen, erhalten Sie unterschiedliche Zufallszahlen! Das vierte Argument der Funktion `sample()` kann benützt werden, um z.B. einen unfairen Würfel zu simulieren. Wir gehen aber im Moment nicht näher auf diese Situation ein.

Es mag seltsam klingen, aber bei Simulationen möchten wir oft *reproduzierbare* Sequenzen von Zufallszahlen erzeugen. D.h.: Jedes Mal, wenn wir die Simulation starten, möchten wir dieselben Ergebnisse erhalten. Das mag auf den ersten Blick der Vorstellung von „Zufall“ widersprechen, macht aber spätestens angesichts der folgenden Überlegungen Sinn:


- Der Computer produziert ja ohnehin keine „echten“ Zufallszahlen, sondern sogenannte *Pseudozufallszahlen*: Er berechnet mit Hilfe eines Algorithmus eine Sequenz von Zahlen, die in vielen Aspekten „zufällig“ wirkt. Bei Würfelzahlen z.B. heisst das: Er generiert eine Sequenz von Zahlen, in denen jede der Zahlen 1 bis 6 im Schnitt gleich häufig vorkommt; auch jedes mögliche Paar, Tripel etc. aufeinanderfolgender Zahlen kommt über lange Sequenzen gleich häufig vor. Er generiert also eine Sequenz, von der man nicht sagen kann, ob sie tatsächlich von einem fairen Würfel produziert wurde oder von einem Computer.^a
- Stellen Sie sich vor, Sie programmieren für Ihre Bachelor-Arbeit eine Simulation, die auf Zufallszahlen basiert. Sie finden dabei ein überraschendes Ergebnis, welches den theoretischen Überlegungen ihres Dozenten widerspricht. Verunsichert wollen Sie das Beobachtete mit ihm besprechen – doch wenn Sie die Simulation erneut laufen lassen, ist der Effekt plötzlich nicht mehr zu sehen. Spätestens dann werden Sie sich *reproduzierbare Pseudozufallszahlen* wünschen.

Um sicherzustellen, dass eine Simulation immer dieselbe Sequenz von Zufallszahlen liefert, können wir den Zufallszahlen-Generator zu Beginn der Simulation *initialisieren*. Das geschieht mit Hilfe der Funktion `set.seed()`; sie übernimmt als Argument eine ganze Zahl, einen sogenannten *Seed*:

```
>> set.seed(13)
```


Verschiedene Seeds produzieren verschiedene Folgen von Zufallszahlen, aber wenn Sie mehrmals den gleichen Wert als Seed setzen, werden Sie immer wieder dieselbe Folge von Zufallszahlen erhalten. Falls Sie den Seed nicht explizite setzen, wird in R beim Neustart ein Seed aus der entsprechenden Startzeit und der Prozess ID generiert. So können wir möglichst „echten“ Zufall simulieren. Damit wird der aktuelle Zustand der rechnerinternen Uhr als Seed gesetzt; dieser ist mit an Sicherheit grenzender Wahrscheinlichkeit jedesmal anders, wenn wir diesen Befehl verwenden.

^aR und Matlab, beide verwenden zur Generierung von Zufallszahlen den sogenannten Mersenne-Twister.

Aufgabe S6.1  Zum Aufwärmen wollen wir in dieser Aufgabe einen fairen Würfel simulieren.

- Generieren Sie eine Sequenz von 1000 Würfelzahlen (Zufallszahlen zwischen 1 und 6). Spielen Sie mit dem Seed für `set.seed` und überprüfen Sie, dass Sie jedes Mal dieselbe Sequenz kriegen, wenn Sie denselben Seed verwenden.
- Berechnen Sie für $n = 1, 2, 3, \dots, 1000$ die absolute sowie die relative Häufigkeit $H_n(A)$ bzw. $h_n(A)$ des Ereignisses $A = \text{„es wurde eine 6 gewürfelt“}$ (vgl. Übungsblatt S3). Plotten Sie den Verlauf von $h_n(A)$ als Funktion von n . Finden Sie ein ähnliches Verhalten wie in den Plots auf Blatt S3?

Hinweise: In Bearbeitung.

Aufgabe S6.2  Einige von Ihnen mögen dem Resultat aus Aufgabe S2.12 immer noch nicht so recht trauen! Überprüfen Sie die berechnete Wahrscheinlichkeit nun numerisch: Generieren Sie 23 Zufallszahlen aus $\{1, 2, \dots, 365\}$ und schauen Sie, ob Sie mindestens 2 gleiche Zahlen erhalten. Wiederholen Sie dieses Zufallsexperiment 100'000 Mal und approximieren Sie so die gesuchte Wahrscheinlichkeit.

Hinweis: In Bearbeitung.

Lösungen

Lösung S6.1 In Bearbeitung.

Lösung S6.2 In Bearbeitung.