

---

## Einfache Laplace-Versuche

### Zufallszahlen erzeugen

#### ■ Zufall oder Pseudozufall

Echte Zufallszahlen erzeugt man am besten mit dafür vorgesehenen Laplace-Geräten (Münze, Würfel, Roulette, etc.) oder mit authentischem Ziffernmaterial [vgl. das 1955 in The Free Press (New York) erschienene Buch *A Million Random Digits* bzw. den Auszug daraus in Arthur Engel: *Wahrscheinlichkeitsrechnung und Statistik*, Band 1, Klett Verlag: Stuttgart 1973, S. 154-155].

Ein Computer ist eine deterministische Maschine. Die Werte einer berechenbaren Funktion (bzw. die Ausgaben eines Computer-Programms) können nicht wirklich zufällig sein. Mit speziellen Algorithmen lassen sich aber sog. Pseudozufallszahlen erzeugen. Diese entstehen aus einer Startzahl durch wiederholte Ausführung (Iteration) einer geeigneten Funktion. Bei der Wahl des Iterationsalgorithmus kommt es darauf an, dass die erzeugten Zahlen "regellos" aufeinander zu folgen scheinen (und sich insbesondere nicht in zu kurzen Perioden wiederholen). Man vgl. zu dieser interessanten Problemstellung Kapitel 6 des oben erwähnten Buchs von A. Engel sowie [A. Engel: *Mathematisches Experimentieren mit dem PC*: Klett Verlag: Stuttgart 1991, S. 91-114]. Eine eingehende Erörterung des Themas liefert [Donald E. Knuth: *The Art of Computer Programming*, Vol. 2, Chapter 3, das den Umfang eines Buches besitzt].

#### ■ Rückgriff auf eingebaute Funktionen

In den meisten Programmiersprachen sind brauchbare Zufallszahlen-Generatoren eingebaut. Am weitesten verbreitet ist eine Funktion (zumeist **Random** genannt), die eine reelle Pseudozufallszahl im Intervall  $[0,1)$  ausgibt. Die Ergebnisse sind gleichverteilt, d.h. jede Zahl wird mit derselben Wahrscheinlichkeit gewählt.

Aufruf von **Random** in *Mathematica*:

```
Random[ ]
```

```
0.550926
```

In JavaScript ist **random** eine Methode des **Math**-Objekts. Mit ihr lässt sich leicht eine selbstdefinierte Funktion hinschreiben, die gleichverteilte ganze Zufallszahlen aus  $\{0, 1, \dots, m - 1\}$  erzeugt:

```
function RandomInt(m)
{
  return parseInt( m * Math.random() )
}
```

## Elementare Zufallsgeneratoren

Im Folgenden wollen wir einige Funktionen bereitstellen, die einfache (diskrete) Laplace-Versuche und zugehörige Versuchsreihen simulieren, dabei aber von den komfortableren Möglichkeiten von *Mathematica* Gebrauch machen.

### ■ Ganze Zufallszahl zwischen zwei Grenzen

Die Funktion **Random[Integer, {m,n}]** erzeugt eine ganze Pseudozufallszahl zwischen den ganzen Zahlen **m** und **n** (einschließlich):

```
Random[Integer, {100, 120}]
```

```
115
```

Denselben Effekt erzielt man mit folgender Zeile:

```
100 + Floor[21 * Random[]]
```

```
113
```

Die dementsprechende allgemeine Funktion:

```
RandomInteger[m_, n_] := m + Floor[(n - m + 1) * Random[]]
```

Eine Versuchsreihe dazu:

```
Table[RandomInteger[100, 120], {30}]
```

```
{117, 106, 114, 115, 107, 103, 101, 100, 101, 109, 101, 114, 118, 105, 105,
 108, 100, 102, 115, 101, 104, 102, 105, 108, 107, 116, 112, 113, 100, 113}
```

## ■ Urnenmodell I: Ziehen mit Zurücklegen

Die Funktion **ZieheMitZuruecklegen** (aus dem Paket **Zufallsversuche.m**) erzeugt eine gewünschte Anzahl zufällig einer beliebigen Menge (Merkmalraum) entnommener Elemente, genauer (im Sinne der Kombinatorik): eine zufällige Permutation mit Wiederholung.

```
ZieheMitZuruecklegen[m_, anzahl_] := Module[{mr = Union[m]},
  Table[Part[mr, Random[Integer, {1, Length[mr]}]], {anzahl}]
]
```

```
ZieheMitZuruecklegen[{"Otto", 7, "€"}, 10]
```

```
{€, Otto, €, Otto, Otto, 7, Otto, Otto, €, Otto}
```

Die Funktion simuliert das Urnenmodell der Stochastik: Aus einem Kasten mit irgendwelchen Objekten (Kugeln) wird (verdeckt) eines gewählt, jedoch nicht entfernt ("Ziehen mit Zurücklegen"). Die als erstes Argument übergebene Liste steht für den Kasten bzw. die Urne; das zweite Argument gibt die Anzahl der Ziehungen an.

Anmerkung: **ZieheMitZuruecklegen** ist funktionell identisch mit **Stichprobe** aus dem *Mathematica*-Paket **Verteilungen.m**.

## ■ Urnenmodell II: Ziehen ohne Zurücklegen

Das Gegenstück **ZiehenOhneZuruecklegen** (vom bekannten Zahlenlotto "6 aus 49" her bekannt) erfordert, dass die Urne das gezogene Element anschließend nicht mehr enthält. Dank leistungsfähiger Listenoperationen lässt sich auch diese Funktion in *Mathematica* auf natürliche Weise formulieren:

```
ZieheOhneZuruecklegen[m_, anzahl_] :=
Module[{mr = Union[m], sprobe = {}, elem, j = 0},
  While[j < anzahl && Length[mr] > 0,
    elem = Part[mr, Random[Integer, {1, Length[mr]}]];
    AppendTo[sprobe, elem];
    mr = Drop[mr, Position[mr, elem][[1]]];
    j++;
  ]
sprobe
```

Die While-Schleife erledigt nacheinander 1) die Entnahme einer Kugel (Element), 2) ihre Hinzügung zur Stichprobe und 3) ihre Entfernung aus der Urne. Dies geschieht höchstens so oft wie im zweiten Argument (**anzahl**) angegeben; sobald die Urne keine Kugel mehr enthält, wird abgebrochen.

```
urne = Table[k, {k, 1, 49}];  
ZieheOhneZuruecklegen[urne, 6]
```

```
{38, 22, 42, 47, 5, 20}
```

In dieser Form realisiert der Zufallsversuch eine zufällige 6-Permutation von 49 Elementen ohne Wiederholung. Wenn wir die Stichprobenelemente anordnen (weil es, wie beim Zahlenlotto, auf ihre Reihenfolge nicht ankommt), so erhalten wir 6-Kombinationen:

```
Sort[%]
```

```
{5, 20, 22, 38, 42, 47}
```

Beschränkt man sich auf herkömmliche Programmiersprachen (Pascal, C, etc.), sind die algorithmischen Lösungen dieses Problems ein wenig schwieriger zu verstehen. Vgl. das in HTML und JavaScript entwickelte Beispiel-Formular *Simulierte Lottoziehung "6 aus 49"*.