Simulation zähmt Krokodil

OLAF LOTTER, STRALSUND

Zusammenfassung: *Ein Kinderspielzeug für Vierjährige kann am Computer simuliert werden, um theoretische Erkenntnisse zu bestätigen und sogar auf neue zu stoßen.*

1 Einleitung

In Lotter (2020) wurde vorgestellt, wie man die Funktionsweise des Kinderspielzeugs KROKO DOC des Herstellers Hasbro spielerisch entdecken kann. Dazu wurden Hypothesen aufgestellt und getestet sowie optimale Drückstrategien und Gewinnwahrscheinlichkeiten *analytisch* hergeleitet.

In dem Spiel geht es darum herauszufinden, welcher Zahn dem Krokodil wehtut. Für zwei Spielende wird bei geöffnetem Maul abwechselnd je ein Zahn gedrückt. Erwischt man den kranken, schnappt das Krokodil zu, und man verliert. Eingebaut ist ein mechanischer Zufallsgenerator.

Nun soll gezeigt werden, wie man die Gewinnwahrscheinlichkeiten durch *Simulation* mithilfe eines kurzen Computerprogramms erhalten kann. Daran lässt sich gut beobachten, dass die relativen Häufigkeiten mit wachsender Anzahl von Simulationen gegen die theoretischen Werte konvergieren. Außerdem können neue Drückstrategien simuliert werden.

2 Simulationsprogramm

Als Programmiersprache wählte ich Python, deren Popularität rasant wächst und die mittlerweile von einigen Lehrkräften in der Schule eingesetzt wird. Python-Code lässt sich auch ohne Installation sofort im Browser ausführen.[1]

```
# ausloesende Zaehne
a=[[1,10],[2,0],[3,0],[4,0],[5,0],
   [1,6],[2,7],[3,8],[4,9],[5,10],
   [6,0],[7,0],[8,0],[9,0]]
# optimale Drueckstrategie
o=[1,6,10,5,2,7,3,8,4,9]
zuege=0
gew=0
n=int(input('n='))
import random
for j in range(n):
  # Rotorposition
  rp=random.randrange(14)
  # bereitstehende Zaehne
  b=list(range(1,11))
  maulauf=True
  zug=0
  while maulauf:
    # Drueckstrategie: von den
    # folgenden 4 Zeilen sind die
    # nicht gewuenschten 3 mit #
    # auszukommentieren
    i=zug      # links -> rechts
    i=9-zug    # rechts -> links
    i=o[zug]-1# optimal
    i=random.randrange(10-zug)#Zuf
    if b[i]==a[rp][0] \
    or b[i]==a[rp][1]:
      maulauf=False
    # letzte Nummer umkopieren,
    # nur bei Zufall relevant
    b[i]=b[9-zug]
    zug+=1
  zuege+=zug
  # Gegner beginnt, macht also
  # ungerade Zuege
  if divmod(zug,2)[1]==1: gew+=1
print(f'gew  ={gew:7}',end='')
print(f' Anteil={gew/n:5.3f}')
print(f'zuege={zuege:7}',end='')
print(f' E(X)  ={zuege/n:4.2f}')
```

[1] www.online-python.com

Relativ gesehen am aufwendigsten ist die Implementierung der auslösenden Zähne (Tab. 1, Programmzeilen 1 bis 4) und der optimalen Drückstrategie 1,6,10,5,2,7,3,8,4,9 (vgl. Abschnitt 4.3 in Lotter (2020), Programmzeilen 5 und 6). Es sei daran erinnert, dass bei der optimalen Drückstrategie der Erwartungswert der Anzahl der Spielzüge am größten ist.

Einzige Programmeingabe ist die Anzahl n der Simulationen (Zeile 9). In der von mir genutzten Online-Programmierumgebung sollte $n < 10^6$ gelten, da sonst mit der Fehlermeldung „Session Killed due to Timeout.“ gerechnet werden muss. Mithilfe der Anweisung `random.randrange(14)` (Zeile 13) wird die Rotorposition gemäß einer Laplace-Verteilung (diskrete Gleichverteilung) aus der Menge $\{0,1,\ldots,13\} \subset \mathbb{N}$ bestimmt. Wegen der in Python

Tab. 1: Auslösende Zähne, vgl. Tab. 5 in Lotter (2020)

Rotorposition	auslösend
1	1, 10
2	2
3	3
4	4
5	5
6	1, 6
7	2, 7
8	3, 8
9	4, 9
10	5, 10
11	6
12	7
13	8
14	9

üblichen Indizierung ab 0 ist dies praktischer als aus der Menge $\{1,2,\ldots,14\}$, vgl. Zeilen 27 und 28. Bei jedem Programmaufruf wird der Zufallsgenerator neu initialisiert, sodass nicht alle Simulationen gleich laufen. Will man genau das Gegenteil erreichen, kann nach Zeile 10 der Befehl `random.seed(k)` mit festem $k \in \mathbb{N}$ eingefügt werden.

In den Zeilen 23 bis 26 sind die vier unterschiedlichen Drückstrategien implementiert. Vor dem Ausführen kommentiere man die nicht gewünschten drei Varianten mittels eines vorangestellten Rautenzeichens # aus. Es sei daran erinnert, dass man besser *nicht* beginnt, d. h. ein Gewinn wird verbucht, wenn das Maul in einem *un*geraden Spielzug zuschnappt (Zeilen 35 bis 37).

Am Ende werden Anzahl (Zeile 38) und Anteil (Zeile 39) der gewonnenen Spiele (Tab. 2) sowie Anzahl (Zeile 40) und mittlere Anzahl (Zeile 41) der Spielzüge (Tab. 3) ausgegeben. Dabei legen die Zusätze `:5.3f` bzw. `:4.2f` die Anzahl der Gesamt- und Nachkommastellen fest.

Alle theoretischen Werte stammen aus Lotter (2020) – bis auf $\frac{30}{7}$, das der Vollständigkeit halber eingetragen wurde.[2] Selbst für $n = 10^5$ fällt die Rechenzeit nicht ins Gewicht: Sie beträgt in meiner Programmierumgebung knapp unter zwei Sekunden für die optimale und gut zweieinhalb Sekunden für die zufällige Drückstrategie – Zeiten, in denen Spielende normalerweise kein einziges echtes Spiel schaffen. Man sieht, dass der `randrange`-Befehl (Zeile 26) mehr Zeit benötigt als die bloße Auswahl des optimalen Spielzugs (Zeile 25), obwohl pro Runde im Mittel sogar fast ein Spielzug weniger ausgeführt wird ($5{,}50 - 4{,}71 = 0{,}79$).

Eine Vergleichssimulation mit Free Pascal – einem schlanken Open-Source-Compiler,[3] der auf allen gängigen Betriebssystemen leicht zu installieren ist, – ergab für $n = 10^8$ auf meinem Tower-Büro-PC sieben Sekunden für die optimale und etwas unter zehn Sekunden für die zufällige Drückstrategie, d. h. er ist um mehr als den Faktor 250 schneller.

3 Vorteile der Simulation

Theoretische Herleitungen bereiten Lernenden üblicherweise Schwierigkeiten, da ein Fehler nicht zwangsläufig auffällt: Eine Probe ist nicht immer möglich bzw. kostet viel Zeit. Hingegen überprüft der Computer unverzüglich, ob der Programmcode syntaktisch korrekt ist, sodass ein unmittelbares Feedback erfolgt. Natürlich ist dadurch noch nicht klar, ob die Resultate sinnvoll sind, doch es lassen sich schneller Plausibilitätsprüfungen durchführen.

Nach Definition muss die optimale Drückstrategie größere Werte für die mittlere Anzahl $\bar{z}_n$ der Spielzüge liefern als andere Drückstrategien. Von den $10! = 3\,628\,800$ denkbaren Drückstrategien sind viele optimal, wie in Lotter (2020) ausgeführt. Dies kann mithilfe der Simulation einfach überprüft werden.

Gibt man z. B. $10,5,1,6,8,3,7,2,9,4$ ein, konvergiert $\bar{z}_n$ ebenfalls gegen $\frac{11}{2}$, ebenso die relative Gewinn-Häufigkeit g_n gegen $\frac{9}{14}$. Durch Experimentieren mit der Reihenfolge stieß ich auf folgende in-

[2] $E(X) = \sum_{i=1}^{10} i \cdot p_i = (1+2+3+4+5) \cdot \frac{2}{14} + (6+7+8+9) \cdot \frac{1}{14} + 10 \cdot 0 = \frac{60}{14} = \frac{30}{7}$

[3] `www.freepascal.org`

Tab. 2: Relative Gewinn-Häufigkeiten g_n bei n Simulationen mit unterschiedlichen Drückstrategien

	Drückstrategie			
n	optimal	links	rechts	Zufall
250	0,576	0,572	0,588	0,520
1 000	0,666	0,568	0,571	0,551
10 000	0,640	0,564	0,578	0,510
100 000	0,641	0,572	0,570	0,526
Theorie	$\frac{9}{14} \approx 0{,}643$	$\frac{8}{14} \approx 0{,}571$	$\frac{8}{14} \approx 0{,}571$	$\frac{11}{21} \approx 0{,}524$

Tab. 3: Mittlere Anzahl $\bar{z}_n$ der Spielzüge bei n Simulationen mit unterschiedlichen Drückstrategien

	Drückstrategie			
n	optimal	links	rechts	Zufall
250	5,64	4,32	4,16	4,82
1 000	5,49	4,40	4,28	4,66
10 000	5,46	4,29	4,28	4,72
100 000	5,50	4,28	4,29	4,72
Theorie	$\frac{11}{2} = 5{,}50$	$\frac{30}{7} \approx 4{,}29$	$\frac{30}{7} \approx 4{,}29$	$\frac{33}{7} \approx 4{,}71$

teressante Erkenntnis: Werden die Werte 1 und 6 vertauscht, reduziert sich erwartungsgemäß $\bar{z}_n$ (und zwar auf 5,43), aber g_n erhöht sich auf 0,714, was $\frac{10}{14}$ statt $\frac{9}{14}$ entspricht. Im Nachhinein leuchtet das ein: Wenn Zahn 10 nicht auslöst, fallen die Rotorpositionen 1 und 10 weg (s. Tab. 1), sodass es optimal ist, entweder Zahn 1 oder Zahn 5 zu drücken. Entscheidet man sich im zweiten Zug für Zahn 5, sollte im *dritten* Zug Zahn 1 gewählt werden. Ignoriert man dies und drückt stattdessen Zahn 6, kann im *vierten* Zug mit Zahn 1 nicht verloren werden: Neben der Rotorposition 1 ist nämlich jetzt auch die Rotorposition 6 ausgeschlossen, sodass Zahn 1 nicht mehr scharf sein kann. Dadurch erhöht sich g_n um $\frac{1}{14}$.

4 Einsatzszenarien in der Lehre

Idealerweise ist mindestens ein KROKO DOC vorhanden, damit das Spielzeug vorgeführt werden kann. Anschließend könnte das Simulationsprogramm verteilt werden, sodass alle experimentieren können. Alternativ ließe sich der Code im Informatikunterricht selbst programmieren, nachdem die Funktionsweise von KROKO DOC besprochen worden ist. Beim Experimentieren könnten parallel mehrere Drückstrategien und unterschiedliche Werte für n ausprobiert werden. Dann ließen sich sicherlich mehrere Phänomene wie in Tab. 2 für $n = 250$ beobachten, wo die optimale Drückstrategie einen *kleineren* Wert für g_n ergeben hat (0,576) als diejenige von rechts nach links (0,588), obwohl die Theorie einen *Vorsprung* von $\frac{1}{14} = 0{,}071$ voraussagt.

Ein zweites lohnenswertes Experimentierfeld stellt die Suche nach weiteren optimalen Drückstrategien oder nach hohen bzw. niedrigen Gewinnwahrscheinlichkeiten dar. Durch Ausprobieren lässt sich bestätigen, dass optimale Drückstrategien nur mit 1, 5, 6 oder 10 beginnen können, weil Zähne 1 und 10 stets in Kombination auslösen (u. a. mit 6 bzw. 5). Wer weiß, welche Erkenntnisse noch erhalten werden, wenn eine ganze Schulklasse mit dem Programm experimentiert? Simulieren schont Nerven und Finger, weil das Zuschnappen ziemlich laut ist und schmerzhaft sein kann.

Literatur

Lotter, Olaf (2020): Krokodil hält Einzug in die Hochschule. Stochastik in der Schule, 40(3):28–33.

Anschrift des Verfassers
Prof. Dr.-Ing. Dipl.-Math. Olaf Lotter
Hochschule Stralsund
Fakultät für Maschinenbau
Zur Schwedenschanze 15
18435 Stralsund
olaf.lotter@hochschule-stralsund.de