GF Informatik: Simulationen

Gustavo Aeppli, Maurus Gubser, Christian Haas

23. Juni 2018

Simulationen

Worum geht es?

In diesem Kapitel betrachten wir Grundkonzepte der Simulation und einige Anwendungen. Dabei werden wir selber einige Simulation mithilfe von Python (Tigerjython) durchführen.

Was ist eine Simulation?

Die Simulation ist eine Arbeitsweise zur Analyse von komplexen Systemen und Zusammenhängen. Typischerweise sind das Fragestellungen, bei welchen man mit theoretischer und formelmässiger Behandlung an Grenzen stösst.
In der Forschung und Industrie, aber auch in der Finanzwelt spielen Computersimulationen eine wichtige Rolle. Computersimulationen haben gegenüber realen Experimenten und Untersuchungen den Vorteil, dass sie kostengünstig und umweltschonend durchführbar sowie ungefährlich sind. Allerdings können sie die Wirklichkeit meist nie exakt wiedergeben. Die Gründe dafür sind vielfältig:

- Die Wirklichkeit kann wegen Messfehlern nie exakt in Zahlen gefasst werden (ausser bei Abzählungen).
- Das Zusammenwirken der Komponenten ist oft nicht exakt bekannt, da die zu Grunde liegenden Gesetze nicht exakt sind oder nicht alle Einflüsse berücksichtigt werden.

Immerhin werden Computersimulationen mit steigender Rechenleistung der Computer immer präziser, man denke etwa an die Wetterprognosen für die nächsten Tage.

Monte-Carlo Simulationen

Eine Untergruppe der angewandten Techniken bilden die *Monte-Carlo Simulationen*. Dabei wird eine sehr grosse Zahl gleichartiger Zufallsexperimente durchgeführt, um eine Wahrscheinlichkeit näherungsweise zu bestimmen.
Alle Beispiele und Aufgaben in diesem Skript gehören zu dieser Gruppe.

1 Würfeln simulieren

Deine Kameradin schlägt dir folgendes Spiel vor:

> "Du darfst drei Würfel werfen. Treten dabei Sechser auf, so hast du gewonnen und ich gebe dir eine Murmel. Wenn keine Sechser vorkommen, habe ich gewonnen und du gibst mir eine Murmel".

Aufgabe 1

Findest du dieses Spiel fair?
Diskutiert zu zweit!

Mit dem Computer und deinen Programmierkenntnissen kannst du deine Überlegungen nachprüfen. Dabei gehst du davon aus, dass es gleichgültig ist, ob du die 3 Würfe hintereinander oder miteinander machst. Oder anders gesagt ist die Wahrscheinlichkeit, mit einem Würfel eine bestimmte Zahl zu erhalten, unabhängig von den anderen Würfeln immer $\frac{1}{6}$.
Es gibt zwei Möglichkeiten, das Problem anzupacken, entweder **statistisch** oder **kombinatorisch**. Die statistische Lösung entspricht dem realen Spiel.
Du simulierst das Werfen der Würfel, indem du im Programm oftmals 3 Zufallszahlen zwischen 1 und 6 erzeugst und die Gewinnfälle zählst.

RANDOM

Wir brauchen neu Zufallszahlen. Dazu können wir das Modul `random` mit der Funktion `randint` importieren. Es benötigt als *Parameter* zwei Zahlen. Die erste gibt die tiefste mögliche Zahl an, die zweite die höchste. Das folgende einfache Programm erzeugt eine Zufallszahl zwischen 1 und 100.

Zufallszahl

```
1 from random import randint
2 zufallszahl = randint(1,100)
3 print(zufallszahl)
```

zufallszahl.py

Aufgabe 2

Schreibe ein Programm, welches 20 Zufallszahlen zwischen 1 und 1000 erzeugt und diese jeweils ausgibt.

Betrachten wir nun die zu Beginn gestellte Frage mit den drei Würfeln. Studiere dazu den folgenden Programmcode.
(Darin kommt eine nützliche Kurzschreibweise vor: `anzahlGewonnen+=1`. Sie steht für `anzahlGewonnen=anzahlGewonnen+1`, erhöht also den Wert der *Variablen* `anzahlGewonnen` um 1.)

Aufgabe 3

Betrachte den folgenden Python-Code und führe ihn selber aus.

a) Wozu dient die Variable `anzahlGewonnen`?

b) Erkläre es deinem Sitznachbar .

Würfelsimulation: Drei Würfe

```
1  from random import randint
2
3  anzahlDurchgaenge = 100
4  anzahlGewonnen = 0
5  repeat anzahlDurchgaenge:
6      a = randint(1, 6)
7      b = randint(1, 6)
8      c = randint(1, 6)
9      if a == 6 or b == 6 or c == 6:
10         anzahlGewonnen += 1
11
12 print "Gewonnen in:", anzahlGewonnen, " von",
       anzahlDurchgaenge, " Spielen"
13 print "Meine Gewinnwahrscheinlichkeit:",
       anzahlGewonnen / anzahlDurchgaenge
```

dreiWuerfe.py

Aufgabe 4

Führe einige Durchgänge dieser Simulation durch.

a) Was beobachtest du?

b) Erhöhe die Zahl der Spieldurchführungen auf 10000. Wie gross ist die Gewinnwahrscheinlichkeit gemäss deiner Simulation?

Wie du siehst, kommt nicht bei jedem Durchgang das genau gleiche Resultat heraus. Er ist **statistischen Schwankungen** unterworfen. **Je mehr** Versuche, **desto**

genauer wird das Resultat.

Wir sehen, dass in etwa 42% der Durchgänge mindestens eine 6 vorkommt.

Aufgabe 5

Kannst du auch kombinatorisch nachvollziehen, dass es etwa 42% sind? Tipp: Überlege dir, wie viele Möglichkeiten es insgesamt für einen Wurf mit 3 Würfeln gibt. Bei wie vielen dieser Würfe kommt keine 6 vor?

2 Simulationsfunktion benutzen

Das vorherige Problem lässt sich relativ einfach auch exakt berechnen. Wir betrachten nun ein Beispiel, bei welchem das nicht so einfach möglich ist.

Mit welcher Wahrscheinlichkeit treten bei zehnmal Würfeln alle Zahlen mindestens einmal auf?

Aufgabe 6

Schätze die Wahrscheinlichkeit!

Wir müssen irgendwie die simulierten Augenzahlen eines Wurfes speichern. Dazu können wir in Python Listen verwenden.

LISTEN

Verwendung von Listen in Python.

`meineListe=[]`	Erzeugt eine leere Liste.
`meineListe.append(e)`	Fügt `e` zu `meineListe` hinzu.
`meineListe.count(e)`	Gibt die Anzahl `e` in `meineListe` hinzu.
`x in (listenname)`	gibt `true` zurück, falls `x` in der Liste vorkommt

Beispiel zu Listen

```
1 meineListe=[]
2 meineListe.append(1)
3 meineListe.append(4)
4 meineListe.append(4)
5 print(meineListe) # Ausgabe [1,4,4]
6 print(meineListe.count(4)) # Gibt 2 aus.
```

listenBeispiel.py

Das folgende Programm erzeugt zuerst 10 Zufallswürfe und speichert sie mit `append` in einer Liste ab. Anschliessend wird die Anzahl verschiedener Zahlen gezählt. Dies wird mit einer **for**-Schleife[1] erzielt. Jede Zahl zwischen 1 und 6 wird getestet und falls sie in der Liste vorkommt, wird die Anzahl verschiedener Zahlen um 1 erhöht.[2] Falls 6 verschiedene Zahlen vorkommen, wird anschliessend `6 verschiedene` ausgegeben.

Würfelsimulation: Alles verschiedene, ein Durchgang

```
1  from random import randint
2  zahlen=[]
3  repeat 10:
4      eineZahl=randint(1,6)
5      zahlen.append(eineZahl)
6  print zahlen
7  anzahlVerschiedene=0
8  for i in range(1,7):
9      if i in zahlen:
10         anzahlVerschiedene+=1
11 if anzahlVerschiedene==6:
12     print "6 verschiedene"
```

allesVerschiedeneEinDurchgang.py

Will man nun die Wahrscheinlichkeit für alle Zahlen bei zehnmal Würfeln mittels Simulation bestimmen, muss obiges Programm sehr oft ausgeführt werden. Man könnte dies erreichen, indem der Programmcode mit einer grossen `repeat` Schleife umgeben wird. Wir müssten dabei die Versuche, bei welchen alle sechs Zahlen aufgetreten sind, zählen und könnten diese Anzahl am Ende durch die Anzahl Versuche teilen.
Das Programm wird dadurch jedoch unübersichtlich mit mehreren verschachtelten Schleifen. Viel übersichtlicher wird es mit einer Funktion, welche einen Durchgang der Simulation durchführt.

SIM()

Wir bezeichnen diese Funktion mit `sim()`. Diese Funktion `sim()` gibt 0 oder 1 zurück, je nach Ergebnis der Simulation.
Betrachte dazu den folgenden Programmcode:

[1] Mit `for i in range(1,7):` wird der folgende Block sechs Mal durchlaufen, zuerst mit `i=1`, dann `i=2`, bis `i=6` (und nicht etwa bis 7).
[2] `anzahlVerschiedene+=1` ist eine Kurzschreibweise für `anzahlVerschiedene=anzahlVerschiedene+1`.

Würfelsimulation: Alles verschiedene

```
1  from random import randint
2  def sim():
3      zahlen=[] #leere Liste
4      repeat 10:
5          eineZahl=randint(1,6) #Augenzahl eines
               Wurfs
6          zahlen.append(eineZahl)
7      print zahlen
8      anzahlVerschiedene=0
9      for i in range(1,7):
10         if i in zahlen:
11             anzahlVerschiedene+=1
12     if anzahlVerschiedene==6:
13         print "6 verschiedene"
14         return 1
15     else:
16         return 0
17
18 anzahlDurchgaenge=100
19 anzahlErfolge=0
20 repeat anzahlDurchgaenge:
21     x=sim()
22     anzahlErfolge+=x
23 print "Die Wahrscheinlichkeit betraegt ca." , 100
     * anzahlErfolge / anzahlDurchgaenge ,"Prozent"
```

allesVerschiedene.py

3 Finden und Lösen eigener Probleme

Wir haben nun zwei Probleme mit Hilfe von Simulationen gelöst. Es gibt viele weitere einfache und schwierige Probleme, welche man mit einer Simulation angehen kann.

Aufgabe 7

Ihr sollt zu zweit selber ein Problem finden, welches ihr mit Hilfe einer Simulation in Python lösen könnt. Anschliessend tauscht ihr eure Probleme mit anderen Gruppen aus, löst diese und vergleicht eure Lösungsvorschläge.
Achtet bei der Auswahl eures Problems auf folgende Punkte:

- Geeignete Probleme findet man in den Gebieten der Kombinatorik und Wahrscheinlichkeitsrechnung.
- Interessante Probleme könnten Fragestellungen in Zusammenhang mit Würfeln, Jasskarten, Poker, ... sein.

Eure Abgabe umfasst folgendes:

- Eine Aufgabenstellung mit detaillierter Beschreibung des Problems.
- Eine Lösung, in welcher ihr das Problem analysiert und eure Ideen zur Lösung zusammenfasst.
- Ein Pythonprogramm, welches euer Problem simuliert und löst.

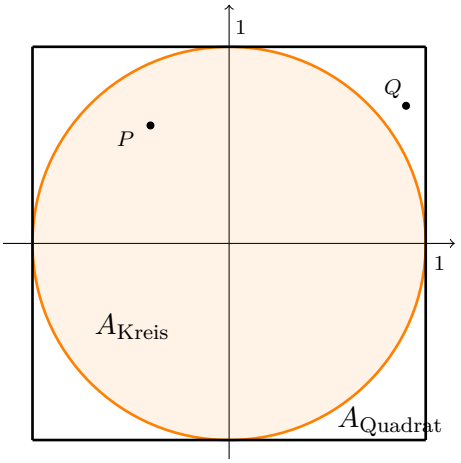
4 Monte-Carlo-Simulation für die Kreiszahl π

Monte-Carlo-Simulationen sind wichtige Verfahren in der Mathematik und ihren Anwendungen. Wir studieren folgendes Beispiel einer Monte-Carlo-Simulation:

> Wie kann man die Zahl π näherungsweise mittels einer Simulation berechnen?

Die Grundidee zur Berechnung stammt aus der Geometrie. Die Fläche eines Kreises mit Radius $r = 1$ beträgt $A_{\text{Kreis}} = \pi \cdot r^2 = \pi$. Nun bettet man diesen Kreis in ein Quadrat der Seitenlänge 2 ein, welches die Fläche $A_{\text{Quadrat}} = 2^2 = 4$ hat. Der Anteil der Punkte im Quadrat, welche auch innerhalb der Kreisfläche liegen, beträgt dann

$$p_{\text{imKreis}} = \frac{A_{\text{Kreis}}}{A_{\text{Quadrat}}} = \frac{\pi}{4}$$



Wir können nun durch eine Simulation diesen Anteil p_{imKreis} näherungsweise bestimmen, indem wir zufällig Punkte im Quadrat wählen und jeweils überprüfen, ob der Punkt im Kreis liegt oder nicht. Formal machen wir das folgendermassen:

- Wir definieren einen zufälligen Punkt $P = (x_P|y_P)$ im Quadrat. D.h. P hat zufällige Koordinaten $-1 \leq x_P \leq 1$ und $-1 \leq y_P \leq 1$.
- Wir testen, ob $P = (x_P|y_P)$ im Kreis liegt. D.h. wir testen, ob $x_P^2 + y_P^2 \leq 1$. Falls der Punkt im Kreis liegt, so erhöhen wir unseren Punktezähler `punkteImKreis` um 1.
- Nachdem wir für 500'000 Zufallspunkte getestet haben, können wir den Anteil der Punkte innerhalb des Kreises ausrechnen:

  $$\frac{\texttt{punkteImKreis}}{500000} \approx p_{\text{imKreis}} = \frac{\pi}{4}$$

  Damit lässt sich nun π näherungsweise angeben.

Die benötigten Zufallszahlen sind aber nicht mehr ganze Zahlen, sondern beliebige Zahlen aus dem Intervall $[-1, 1]$. Diese können wir in Python mit dem Befehl `uniform(-1, 1)` erzeugen.

Monte-Carlo-Simulation: Kreiszahl π

```
from random import uniform

anzahlPunkte=500000
punkteImKreis=0
repeat anzahlPunkte:
    xP=uniform(-1,1)
    yP=uniform(-1,1)
    if xP**2+yP**2 <=1:
        punkteImKreis+=1

pImKreis=punkteImKreis/anzahlPunkte
naeherungPi=4*pImKreis
print "Mit einer Monte-Carlo-Simulation von",
   anzahlPunkte, "Punkten ergibt sich eine
   Naeherung fuer pi von", naeherungPi
```

MonteCarloSimulationPi.py

Aufgabe 8

Schreibe mit Hilfe des obigen Codes eine Funktion `sim(anzahlPunkte)`, welche für eine vorgegebene Anzahl Punkte jeweils eine Näherung für π berechnet. Wie gross muss `anzahlPunkte` sein, damit die Näherung ungefähr gleich 3.14 ist?

Aufgabe 9

Ändere den obigen Programmcode so ab, dass du die Näherung von π nicht mit Hilfe eines Vollkreises durchführst, sondern einen Viertelkreis benutzt.