Institut für Theoretische Physik
der Universität zu Köln

Prof. Dr. Simon Trebst
Michael Becker, Peter Bröcker

# Computerphysik

## Übungsblatt 11

*SS 2014*

**Website**: http://www.thp.uni-koeln.de/trebst/Lectures/2014-CompPhys.shtml
**Abgabedatum:** Montag, 30. Juni 2014 vor Beginn der Vorlesung

## 44. Würfeln mit Python

*Programmiertechniken*

Die **Erzeugung von Zufallszahlen** ist ein elementarer Teilschritt vieler Programme. Aus diesem Grund gibt es natürlich auch in Python entsprechende Routinen, um Zufallszahlen mit verschiedensten Verteilungen zu generieren. Diese wollen wir in dieser Aufgabe untersuchen.

Das relevante **Modul** heißt **random**. Beginnen wir mit der Erzeugung **gleichförmig verteilter** Zufallszahlen. Um eine Zufallszahl aus dem halb-offenen Intervall [0, 1) zu ziehen, benutzen Sie die Funktion *random()*. Prinzipiell können Sie durch multiplizieren und addieren der Funktion Zufallszahlen aus jedem beliebigen Bereich erzeugen. Um dies zu erleichtern gibt es die Funktion *uniform(a, b)*, die Zahlen aus dem Intervall [a, b) zurückgibt. Möchten Sie statt Gleitkommazahlen **ganze Zahlen** verwenden, benutzen Sie die Funktion *randint(a, b)*, wobei zu beachten ist, dass nun auch der Randpunkt b in der Menge enthalten ist. In Codeform sieht das ganze dann folgendermaßen aus:

```
import random

print random.random() # random number in [0, 1)
print random.uniform(1, 5) # random number in [1, 5)
print random.randint(1, 5) # random integer number in [1, 5]
```

Verschiedene Anwendungen erfordern Zufallszahlen die nicht gleichförmig verteilt sind, sondern einer gewissen Funktion folgen. Einige der wichtigsten Verteilungen sind im *random*-Modul enthalten und lassen sich genauso einfach benutzen wie die oben beschriebenen Verteilungen. Um zum Beispiel Zufallszahlen gemäß einer **Gaußverteilung** zu erzeugen, benutzen Sie den Befehl *gauss(mu, sigma)*. Die Parameter *mu* und *sigma* geben den Erwartungswert sowie die Standardabweichung an. Zugriff auf eine **Exponentialverteilung** haben Sie über *expovariate(a)*, deren Mittelwert bei $\ln 2/a$ liegt.

Um zu überprüfen, ob die Verteilungen tatsächlich den gewünschten Gesetzen gehorchen, wollen wir **Histogramme** erstellen und sie mit den analytischen Funktionen vergleichen. Histogramme können Sie mit dem *matplotlib*-Befehler *hist()* erstellen:

```
import matplotlib.pylab as plt

data = [0, 1, 1, 2, 2, 2, 3, 3, 3, 3]
plt.hist(data, bins=4, normed=True)
plt.show()
```

Als Parameter müssen Sie zunächst nur die darzustellenden Daten übergeben. Optional sind die Angabe der *bins* sowie ob die Verteilung normiert werden soll oder nicht (*normed*)

Erzeugen Sie Zufallszahlen mit den Befehlen *gauss* und *expvariate* und stellen Sie diese in einem Histogramm dar. Plotten Sie außerdem die analytischen Verteilungen, deren Formeln wir zur Erinnerung noch einmal aufführen:

$$f_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(x-\mu)^2}{2\sigma^2} \right\}$$

$$f_a(x) = \begin{cases} a\mathrm{e}^{-ax} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

# 45. Pseudo-Zufallszahlen

*5 Punkte*

In dieser Aufgabe wollen wir uns nun selbst daran machen, Zufallszahlen mit dem Computer zu erzeugen und wollen dazu zwei algorithmische **Pseudo-Zufallszahlengeneratoren** schreiben und deren Qualität untersuchen.

Als erstes sollten Sie den (immer noch) vielfach verwendeten multiplikativen **linearen kongruenten Generator** GGL implementieren, der Zufallszahlen gemäß folgender Vorschrift erzeugt:

$$x_n = (ax_{n-1} + c) \mod m. \tag{1}$$

Als Parameter nutzt GGL die folgenden Werte:

$$a = 16807, \quad c = 0, \quad m = 2^{31} - 1. \tag{2}$$

Den *Seed* $x_0$ können Sie beliebig wählen.

Als zweites sollen Sie einen additiven **lagged Fibonacci Generator** programmieren. Dieser erzeugt Zufallszahlen basierend auf *zwei* vorherigen Zufallszahlen:

$$x_n = \left(x_{n-p} + x_{n-q}\right) \mod m, \tag{3}$$

mit $n \geq p > q > 0$. Die Seed-Sequenz $x_0, x_1, \ldots, x_{p-1}$ können Sie dabei mit dem linearen kongruenten Generator aus dem ersten Teil erzeugt werden. Wir wählen folgende Parameter:

$$p = 2281, \quad q = 1252, \quad m = 2^{31} - 1. \tag{4}$$

*Hinweis:* Da unsere Zufallszahlen alle auf das Intervall $[0,1)$ skaliert werden sollen, sollten Sie ihre Zahlen am Ende noch durch $m$ teilen.

Nun wollen wir die Qualität der mit den beiden Generatoren erzeugten Zufallszahlen untersuchen. Erzeugen Sie dazu mit beiden Generatoren 10.000.000 Zufallszahlen im Intervall $[0,1)$ und tragen diese in Histogrammen mit 100 Bins auf. Schätzen Sie die **Korrelationen** der erzeugten Zufallszahlen ab, indem Sie wie in der Vorlesung besprochen

$$\chi = \langle x_i x_{i+1} \rangle - \langle x_i \rangle^2$$

berechnen. Können Sie mit diesem Maß die Qualität der beiden Generatoren unterscheiden?

Führen Sie nun einen sogenannten **Spektraltest** durch. Fassen Sie dazu 3er-Tupel von generierten Zufallszahlen

$$p_i = (x_i, x_{i+1}, x_{i+2}), \qquad i = 0, 1, 2, \ldots, n-2$$

zusammen und stellen Sie die Tupel $p_i$ als Punkte im $\mathbb{R}^3$ dar, indem Sie einen 3D-Plot mit *matplotlib* erzeugen. Untersuchen Sie die beiden erzeugten Datensets von unterschiedlichen Blickwinkeln. Zoomen Sie sich in den Datensatz herein und betrachten lediglich den Ausschnitt

$$0 < x < 0.001, \qquad 0 < y < 1, \qquad 0 < z < 1.$$

Welche Unterschiede lassen Sie schließen, daß die Daten des GGL erheblich schlechtere Qualität besitzen als die des lagged Fibonacci Generators? Untersuchen Sie abschließend auch die Qualitäten der Zufallszahlengeneratoren im Python-Modul *random*.

*Hinweis:* Wer sich für weitergehende Tests zur Qualität von Pseudo-Zufallszahlengeneratoren interessiert, der sei auf die sogenannten **diehard-Tests** verwiesen.

## 46. Viele Wege führen nach $\pi$ *5 Punkte*

Die **Zahl $\pi$** fasziniert die Menschen seit geraumer Zeit. Ihre anschaulichste Bedeutung ist wohl, dass sie das Verhältnis des Flächeninhalts eines Quadrats zu dem eines Kreises mit gleichem Durchmesser wie die Seitenlänge des Quadrats angibt. Analytisch berechnen lässt Sie sich auf verschiedenen Wegen im Prinzip beliebig genau.

Wir wollen in dieser Aufgabe zwei Methoden kennenlernen, mit denen Sie $\pi$ auch dann berechnen könnten, wenn Sie einmal auf einer einsamen Insel stranden, alles über $\pi$ vergessen, aber unbedingt den Flächeninhalt eines Kreises berechnen müssen.

In den Sand zeichnen Sie ein Quadrat und darin einen Kreis. Sie werfen nun **Kieselsteine** in das Quadrat und schauen, wie oft Sie in den Kreis treffen (und wie oft nicht). Angenommen Sie könnten so werfen, dass Sie jeden Ort im Quadrat mit gleicher Wahrscheinlichkeit treffen, so ist es leicht nachzuvollziehen, dass das Verhältnis aus Kieseln im Kreis zu den insgesamt Geworfen proportional zu $\pi$ ist. Berechnen Sie $\pi$ auf diesem Weg.

Für die zweite Methode benötigen Sie keine Kiesel, sondern **Stöcke**. Dieses Experiment geht zurück auf Georges Louis Leclerc, Graf von Buffon. Sie stutzen einen Stock auf die Länge $a$ und zeichnen zwei Linien im Abstand $b$ voneinander in den Sand. Nun werfen Sie den Stock mehrere Male auf dieses Muster und zählen wie oft er auf einer Linie liegt. Leclerc hat dieses Verhältnis zu $2a/(\pi b)$ bestimmt. Schreiben Sie auch für dieses Problem ein Programm. Am einfachsten ist es, wenn Sie $a$ und $b$ gleich 1 setzen.

# 47. Dendriten-Wachstum

*optionale Aufgabe – 6 Punkte*
*Abgabe am Montag, 7. Juli*

Verästelte Kristallstrukturen – sogenannte Dendriten – entstehen durch die **zufällige Anlagerung diffusiver Teilchen**. Den Wachstumsprozess eines solchen Dendriten wollen wir in dieser Aufgabe illustrieren, indem wir den Zufallslauf einzelner diffusiver Teilchen simulieren.

Dazu gehen wir wie folgt vor: Gegeben sei eine Experimentierscheibe von Radius $R$, in deren Mitte sich ein *Seed*, ein erstes unbewegliches Teilchen, befinde. Wir starten nun ein weiteres Teilchen an beliebiger Position auf dem Rand der Experimentierscheibe. In jedem Simulationsschritt kann sich das Teilchen mit gleicher Wahrscheinlichkeit in eine von vier Richtungen (oben/unten/links/rechts) bewegen. Diese diffusive Bewegung simulieren wir so lange bis das Teilchen an den bestehenden Kristall andockt, d.h. es erreicht eine Position in deren unmittelbaren Nachbarschaft (oben/unten/links/rechts) sich ein bereits fixiertes, unbewegliches Teilchen befindet. In einem solchen Fall fixieren wir die Position des Teilchens und starten das nächste Teilchen vom Rand. Sollte das Teilchen während der Simulation von der Experimentierscheibe wandern – also einen Abstand grösser als R vom Mittelpunkt erreichen – dann verwerfen wir das aktuelle Teilchen und starten ein neues Teilchen auf dem Rand.

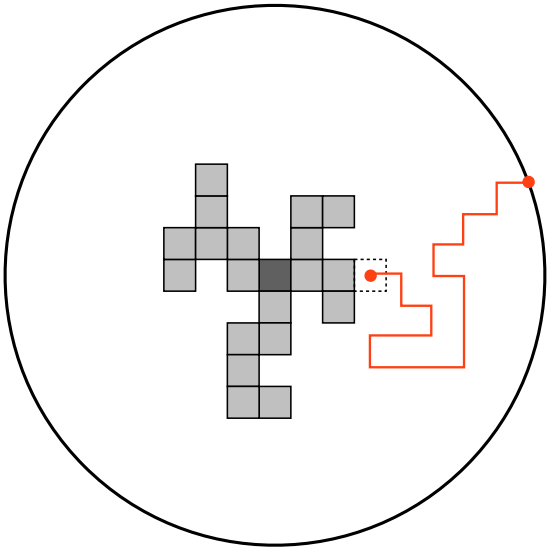

Abbildung 1: Diffusiver Zufallslauf eines Teilchens vom Rand zum Dendriten

Simulieren Sie den Wachstum eines derartigen Dendriten auf einem Raster von $128 \times 128$ Positionen und einer Experimentierscheibe vom Radius $R = 64$. Stellen Sie den Wachstumsprozess graphisch dar. Woran erinnern Sie die erhaltenen Strukturen?

Wer sich ein wenig über die historische Entwicklung dieses im Englischen als *diffusion limited aggregation* bezeichneten Phänomens informieren möchte, der sei auf diesen **Bericht** seines Entdeckers Thomas Witten von der University of Chicago verwiesen.