Übungen zu Physik auf dem Computer SS 2012

Übungsblatt 9: Zufallszahlen

20. Juni 2012

Allgemeine Hinweise

- Abgabetermin ist **Montag, 25.6.2012, 13:00**
- Zur Abgabe schickst Du die Lösungsdatei(en) im Anhang einer Email an Deinen Tutor:
  - Florian (`floh@icp.uni-stuttgart.de`; Dienstag, 15:45–17:15)
  - Dominic (`dominic@icp.uni-stuttgart.de`; Dienstag, 15:45–17:15)
  - Olaf (`olenz@icp.uni-stuttgart.de`; Mittwoch, 15:45–17:15)
- Die Übungen werden in Gruppen von jeweils zwei oder drei Leuten bearbeitet. Diese dürfen sich gerne von Blatt zu Blatt unterscheiden. Aus formalen Gründen muss allerdings jeder von Euch eine eigene Lösung abgeben. Schreibt bitte auf die Lösungen, mit wem Ihr zusammengearbeitet habt, um uns das Korrigieren zu erleichtern.
- Die Übungen finden statt im CIP-Pool des Instituts für Computerphysik (ICP) im Pfaffenwaldring 27.

Aufgabe 9.1 (6 Punkte): Qualitätsanalyse von Zufallsfolgen

Die Datei `/share/Courses/PC2012/09/random.npy` enthält 5 Datenreihen von jeweils 100000 scheinbar zufälligen Zahlen im Intervall $[0, 1]$. Kopiere die Datei in Dein Heimatverzeichnis. Tatsächlich ist nur eine der Datenreihen durch einen guten Zufallszahlengenerator entstanden, die anderen sind in irgendeiner Weise auffällig.

Die Daten aus der Datei kannst Du in Python mittels des folgenden Befehls lesen:

```
r1, r2, r3, r4, r5 = numpy.load('random.npy')
```

- 9.1.1 (1 Punkt) Erzeuge mit Hilfe des Befehls `matplotlib.pyplot.hist` Histogramme der Auftretenswahrscheinlichkeit der Zahlen in den verschiedenen Folgen für 100 *bins*. Gib ein Pythonskript als Lösung ab, das die Plots erzeugt.
- 9.1.2 (1 Punkt) Berechne für Blöcke von jeweils 10 aufeinanderfolgenden Zahlen den Mittelwert und erstelle Histogramme der Mittelwerte (χ^2-Test). Plotte dazu zusätzlich die zu erwartetende Verteilung für eine echte Zufallsfolge. Gib ein Pythonskript als Lösung ab, das die Plots erzeugt.
- 9.1.3 (1 Punkt) Fouriertransformiere die Zahlenreihen mit Hilfe der Funktion `numpt.fft.rfft` und plotte den Betrag der Fourierkoeffizienten im semilogarithmischen Plot (`semilogy`). Gib ein Pythonskript als Lösung ab, das die Plots erzeugt.
- 9.1.4 (1 Punkt) Erzeuge selbst eine Datenreihe von 100000 Zahlen, die sich im χ^2-Test *nicht* an die erwartete Verteilung hält, die sich aber ansonsten in den oben beschriebenen Tests unauffällig verhält. Gib ein Pythonskript als Lösung ab, in dem die Folge erzeugt wird.

- 9.1.5 (1 Punkt) Zur Identifikation, welche der beiden bislang unauffällig verbleibenden Zahlenreihen von einem schlechten Zufallszahlengenerator stammt, müssen *Tripletkorrelationen* der Zufallszahlen berechnet werden. Diese berechnen sich als $C(p,k) = \sum_{i=0}^{N-p} r_i r_{i-p} r_{i-k}$. Für eine Reihe von statistisch unabhängigen Zufallszahlen sollte gelten $C(p,k) = \frac{1}{2}^3$. Berechne die Tripletkorrelationen der Datenreihen für $p = 250$ und $k \in [1, 249]$, und dessen Fehler $|C(p,k) - \frac{1}{2}^3|$. Plotte den Fehler der verschiedenen Datenreihen über k. Gib ein Pythonskript als Lösung ab, das die Plots erzeugt.

  Hinweis Dieser scheinbar willkürlich ausgewählte Test zeigt, dass auch in Folgen von guten Zufallszahlen ungewollte statistische Zusammenhänge zwischen den Zahlen auftauchen können. Eine der Datenreihen wurde vom Zufallszahlengenerator *r250* erzeugt, der über lange Zeit hinweg als guter Generator bekannt war. Im Jahre 1995 zeigten Schmid und Wilding[1], dass der Generator im Zusammenhang mit bestimmten Arten von Simulationen zu großen Fehlern führen kann (siehe z.B. Abbildung 2 in [1]).

Aufgabe 9.2 (5 Punkte): Random Walks

- 9.2.1 (1 Punkte) Schreibe ein Pythonskript, in dem Du $m = 10000$ eindimensionale *Random Walker* simulierst. Jeder Walker macht in einem Zeitschritt zufällig einen Schritt nach vorne oder nach hinten. Gib ein Pythonskript als Lösung ab, das die Walker simuliert.

  Hinweis Speichere die Positionen der Walker zu einem Zeitpunkt in einem ganzzahligen NumPy-Array, den Du wie folgt erzeugen kannst: `x = numpy.zeros(m, dtype=int)`. Auch die Schritte kannst Du alle Walker auf einmal machen lassen (`numpy.random.random_integers`).

- 9.2.2 (2 Punkte) Erweitere das Pythonskript aus der vorigen Aufgabe so, dass gemessen wird, wie wahrscheinlich es ist, daß ein Walker irgendwann während $N = 2^i$ mit $i \in [0, 14]$ Schritten wieder am Ausgangsort vorbeikommt (*Rückkehrwahrscheinlichkeit*). Erzeuge einen semilogarithmischen Plot (`semilogx`), der die Rückkehrwahrscheinlichkeit p über der Anzahl der Schritte N zeigt. Gib das erweiterte Pythonskript, welches den Plot erzeugt, als Lösung ab.

  Hinweis Folgende NumPy-Funktionen können hierbei nützlich sein:
  `all, any, logical_and, logical_or, compress, count_nonzero`

- 9.2.3 (2 Punkte) Erweitere das Pythonskript aus der vorigen Aufgabe so, dass es die Rückkehrwahrscheinlichkeit von *random walks* in $d \in 1, 2, 3, 4$ Dimensionen misst. Erzeuge einen doppeltlogarithmischen Plot, der die Rückkehrwahrscheinlichkeit p über der Anzahl der Schritte N für die verschiedenen Dimensionen d zeigt. Gib das erweiterte Pythonskript, welches den Plot erzeugt, als Lösung ab. Sieht Du einen qualitativen Unterschied bei den verschiedenen Dimensionalitäten?

Literatur

[1] F. Schmid and N. B. Wilding. Errors in Monte Carlo Simulations Using Shift Register Random Number Generators. *International Journal of Modern Physics C*, 6:781–787, 1995. `http://arxiv.org/abs/cond-mat/9512135`.