Institut für Theoretische Physik
der Universität zu Köln

Prof. Dr. Simon Trebst
Michael Becker, Peter Bröcker

Computerphysik

Übungsblatt 12

SS 2013

Website: http://www.thp.uni-koeln.de/trebst/Lectures/2013-CompPhys.html
Abgabedatum: Montag, 15. Juli 2013 vor Beginn der Vorlesung

46. Nobody is perfect

Programmiertechniken

Messungen physikalischer Observablen sind immer fehlerbehaftet, ob Sie im Labor oder durch Simulationen erlangt werden. Um diese Fehler bei der Präsentation zu berücksichtigen, benutzt man wann immer möglich **Fehlerbalken**, die Sie sicherlich schon während der einführenden Experimentalphysikpraktika kennengelernt haben. In Python ist das Zeichnen von Fehlerbalken sehr einfach, was wir Ihnen mit dieser Aufgabe demonstrieren wollen.

Beginnen wollen wir jedoch damit zu erklären, wie man Ergebnisse auf der Festplatte speichern und wieder einlesen kann. Dazu werden wir das **CSV-Format** verwenden, welches gegenüber binären Formaten den Vorteil hat, direkt lesbar zu sein. Solche Formate werden auch als *human-readable* bezeichnet. Es gibt in Python zwar ein spezielles Modul csv, jedoch sind für unsere Zwecke die Methoden savetxt und loadtxt aus dem *numpy*-Modul besser geeignet, denn in der Regel erhalten wir Daten aus einem Programm, dass *numpy* verwendet, oder wir wollen die Daten in *numpy* weiter verarbeiten. Das Benutzen der oben genannten Funktionen ist denkbar einfach. Angenommen, wir wollen einen Plot speichern, der aus x- und y-Werten, sowie Fehlern auf den y-Werten besteht, die jeweils als ein Array vorliegen. Der erste Schritt ist es, die Daten zusammenzufügen mit der Funktion vstack, um sie dann im zweiten Schritt abspeichern zu können:

```
import numpy.random as npr

# generate random data
x = npr.random(10)
y = npr.random(10)
yerr = npr.random(10)

# join data vertically -v-stack to obtain a data matrix
# also try hstack and see what happens!
data = np.vstack((x, y, yerr)).T

# then save data
np.savetxt('data.csv', data, delimiter=',')
```

Um die Daten nun wieder zu laden, müssen wir nur ausführen:

```
data = loadtxt('data.csv', delimiter=',')
```

Die Daten liegen nach dem Einlesen als Matrix vor und wir können wir gewohnt mit der Slice-Notation auf die Spalten zugreifen.

In beiden Fällen haben wir das Schlüsselwort *delimiter* angegeben. So wählen wir das Zeichen aus, welches benutzt werden soll, um zwei Werte voneinandern zu trennen. CSV stand ursprünglich für *Comma Separated Values*, aber wird nun häufig auch als *Character Separated Values* ausgeschrieben, um die Freiheit in der Wahl des Trennzeichens zu berücksichtigen.

Nun möchten wir die eingelesenen Daten inklusive Fehlerbalken graphisch darstellen. Dazu verwenden wir die *matplotlib*-Methode *errorbar* und geben als Schlüsselwort *yerr* an:

```
plt.errorbar(x, y, yerr=yerr, fmt='o')
```

Als eine der vielen möglichen Optionen, zu denen auch Fehlerbalken in x-Richtung mittels *xerr* gehören, geben wir mit *fmt* an, dass nur Punkte gezeichnet werden sollen, die nicht mit Linien verbunden werden.

Ihre Aufgabe sei es, die Funktion

$$cos(kx) + 1$$

auf dem Intervall $x \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$ mittels einfacher Monte Carlo-Integration für verschiedene k-Werte zu integrieren, d.h. mit uniformer Wahrscheinlichkeitsverteilung. Wählen Sie dazu 100 verschiedene k-Werte aus dem Intervall $(0, 10)$ und berechnen Sie jeweils die Varianz (wie in der Vorlesung gezeigt) mit N=100 gezogenen Zufallszahlen. Speichern Sie Ihre Ergebnisse in einer CSV-Datei wie oben beschrieben und lesen Sie sie wieder ein. Plotten Sie dann das Ergebnis der Integration inklusiver Fehlerbalken gegen den Parameter k. Auf dem vorherigen Übungszettel haben Sie außerdem gelernt, die Funktion *quad* zu benutzen um beliebige Funktionen numerisch zu integrieren. Verwenden Sie diese, um Referenzwerte zu erhalten und plotten Sie diese ebenfalls in das gleiche Bild um zu sehen wie stark die Abweichung ist.

47. Integration auf Irrwegen *5 Punkte*

In dieser Aufgabe wollen wir uns noch einmal der **Monte Carlo-Integration** einer Funktion widmen. Dazu wollen wir die zwei-dimensionale Funktion

$$f(x,y) = \cos(x^2 + y^2)e^{-x^2-y^2}$$

betrachten, welche über den gesamten Bereich $\mathbb{R}^2$ integriert werden soll, d.h. wir wollen das Integral

$$I = \int_{\mathbb{R}^2} f(x,y)\,\mathrm{d}x\,\mathrm{d}y \tag{1}$$

berechnen.

Auf dem letzten Übungsblatt haben wir bereits gesehen, dass wir das Integral besonders gut annähern können, indem wir ein sogenanntes **importance sampling** durchführen. Dabei generieren wir zufällige Koordinaten (x,y) nicht gleichverteilt, sondern gemäss einer Verteilung $P(x,y)$, die der zu integrierenden Funktion ähnelt (und welche wir leicht erzeugen können). Dann gilt:

$$I = \int_{\mathbb{R}^2} f(x,y)\,\mathrm{d}x\,\mathrm{d}y = \int_{\mathbb{R}^2} \frac{f(x,y)}{P(x,y)} P(x,y)\,\mathrm{d}x\,\mathrm{d}y \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i,y_i)}{P(x_i,y_i)}, \tag{2}$$

wobei die (x_i, y_i) in der Annäherung im letzten Term P-verteilte Koordinaten seien.

Als Verteilung $P(x,y)$ für die zu integrierende Funktion $f(x,y)$ können wir etwa eine Normalverteilung der Form

$$P(x,y) = \frac{1}{\pi} e^{-x^2-y^2},$$

wählen. Um eine **Markov-Kette** von Koordinaten (x_i, y_i) entsprechend dieser Verteilung zu generieren, wollen wir den **Metropolis-Algorithmus** nutzen. Impementieren Sie dazu zunächst diesen Algorithmus.

Visualisieren Sie die Markov-Kette indem Sie 10^5 zufällige Koordinaten (x_i, y_i) erzeugen und damit den "Pfad" der Markov-Kette nachzeichnen. Zeichnen Sie die dabei die einzelnen Wegsegmente mit einer starken Transparenz, damit deutlich wird, wo besonders viele Koordinaten erzeugt werden. Mit `pyplot` können Sie die Transparenz (den sogenannten *alpha*-Wert) etwa wie folgt einstellen:

```
plt.plot(x_values, y_values, marker='o', linestyle='', alpha=0.02).
```

Nachdem wir uns nun visuell vergewissern konnten, dass unsere Zufallszahlen sinnvoll erzeugt werden, soll das eigentliche Integral mittels Gleichung (2) berechnet werden.

Der exakte Wert des Integrals in (1) ist $I = \pi/2$. Berechnen Sie das Integral jeweils mit $N = 10^4, 10^5, 10^6, 5 \times 10^6$ Zufallszahlen und plotten Sie den absoluten Fehler als Funktion von N.

48. Ising Modell

5 Punkte

Ursprünglich von Ernst Ising (1910 in Köln geboren) zur Beschreibung von **Ferromagnetismus** eingeführt, hat sich das **Ising-Modell** zur Drosophila der Statistischen Physik entwickelt. Doch selbst weit über die Physik hinaus hat es Anwendung gefunden in einer Reihe anderer Disziplinen, etwa in den Neurowissenschaften um die Aktivität von Neuronen zu simulieren oder selbst in den Sozialwissenschaften. Grund genug also, ein paar grundlegende Eigenschaften an dieser Stelle zu studieren.

Wir betrachten ein System elementarer Magnete, sogenannter Spins, deren Wechselwirkung beschrieben sei durch den Hamiltonian

$$H = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j$$

wobei jeder der (klassischen) Spins σ einen der beiden Werte {+1, -1} annehmen kann; die Notation $\langle i, j \rangle$ bedeutet, dass nur Paare von direkt benachbarten Koordinaten i, j zur Summe beitragen sollen. Die Kopplungskonstante J im Hamiltonian setzen wir positiv, $J = +1$, und erhalten damit *ferromagnetische* Wechselwirkungen, die parallel ausgerichtete Spins bevorzugen.

Um das Verhalten dieses Spin-Modells bei einer gegebenen Temperaturen T zu untersuchen, müssen wir eine gegebene Spin-Konfigurationen $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ gemäss der **Boltzmann-Verteilung**

$$p(\sigma) = \frac{1}{Z} \exp(-\beta H)$$

gewichten, wobei Z die sogenannte Zustandssumme und $\beta = 1/k_B T$ die inverse Temperatur ist. Die Idee ist nun, eine ganze Sequenz von Spin-Konfigurationen entsprechend dieser Verteilung zu erzeugen und damit repräsentative Konfigurationen zu "samplen". Entsprechend können wir

Observablen wie etwa die Magnetisierung für alle gesampleten Konfigurationen messen und damit ein thermisches Mittel berechnen. Wie wir eine Sequenz repräsentativer Konfigurationen gemäss einer gegebenen Verteilung erzeugen können, wissen wir bereits – genau dies tut der Metropolis-Algorithmus zur Erzeugung einer Markov-Kette.

An dieser Stelle möchten wir noch einmal kurz die wesentlichen Punkte einer solchen **Monte Carlo Simulation** des Ising Modells zusammenfassen: Wir beginnen mit einer zufällig gewählten Konfiguration von Spins mit Energie E und schlagen vor einen zufällig ausgewählten Spin zu flippen, d.h. von -1 auf +1 bzw. von +1 auf -1 zu ändern. Diese Konfiguration mit geflipptem Spin habe nun die neue Energie E', welche grösser oder kleiner als die bisherige Energie E sein kann. Ob wir die Änderung akzeptieren, hängt ab vom Verhältnis der jeweiligen Boltzmann-Gewichte ab, gemäß

$$p_{\text{acc}} = \exp\left(-\beta(E' - E)\right).$$

Ist die Energie E' kleiner oder gleich E, d.h. $p_{\text{acc}} \geq 1$ so akzeptieren wir die Konfiguration mit dem geflippten Spin. Ist $p_{\text{acc}} < 1$, so ziehen wir eine Zufallszahl r zwischen aus dem Bereich $[0,1)$ und übernehmen die vorgeschlagene Änderung nur dann, wenn $r < p_{\text{acc}}$, anderenfalls verwerfen wir sie.

Ihre Aufgabe sei es nun, genau eine solche Monte Carlo Simulation des Ising Modells zu implementieren. Dazu stellen wir Ihnen das Grundgerüst einer solchen Simulation in der Datei **ising_skeleton.py** zur Verfügung. In diesem Code werden vorgeschlagene Änderungen *immer* akzeptiert, egal ob die Energie größer oder kleiner wird. Dies entspricht dem Limes unendlich hoher Temperatur $T \to \infty$, oder $\beta \to 0$. Implementieren Sie den oben beschriebenen Update-Algorithmus, so daß endliche Temperaturen untersucht werden können. Stellen Sie die Temperatur nacheinander auf die Werte {0.1, 1.0, 2.0, 2.1, 2.15, 2.2, 2.25, 2.3, 2.35, 3.0, 4.0} ein und speichern Sie die am Ende der Simulation angezeigte Darstellung der Spin-Konfiguration. Was können Sie dieser Sequenz entnehmen?

Betrachten Sie außerdem die Magnetisierung $m = \sum_i \sigma_i$ und die Energie $E = -J\sum_{\langle i,j\rangle} \sigma_i\sigma_j$, welche als Observablen über die gesampleten Spin-Konfigurationen gemessen werden sollen. Im bereitgestellten Code sind die relevanten Stellen bereits durch Kommentare gekennzeichnet. Ergänzen Sie die Messungen. Machen Sie eine neue Simulation und plotten Sie die Magnetisierung und Energie gegen die Temperatur.

Beachten Sie bei Ihren Implementierungen, dass das System unter *periodischen Randbedingungen* simuliert werden soll. Wir haben eine Systemgröße von 64×64 gewählt, die relativ schnell berechnet werden kann.