Institut für Theoretische Physik
der Universität zu Köln

Prof. Dr. Simon Trebst
Michael Becker, Peter Bröcker

Computerphysik

Übungsblatt 12

SS 2014

Website: http://www.thp.uni-koeln.de/trebst/Lectures/2014-CompPhys.shtml
Abgabedatum: Montag, 7. Juli 2014 vor Beginn der Vorlesung

48. Integrieren mit Python

Programmiertechniken

Mit den Bibliotheken **scipy** und **numpy** bedient Python im Grunde alle Bedürfnisse des **anwendungsorientierten Numerikers**. Dazu gehört selbstverständlich auch das **Integrieren**. Auf vorherigen Übungszetteln haben Sie dazu verschiedene Algorithmen selbst implementiert, die auf der Diskretisierung des Integrationsbereichs basierten. Genau diese Algorithmen sind auch in *scipy* enthalten und wir wollen an dieser Stelle kurz erläutern, wie man damit umgeht.

Grundlage ist das Modul scipy.integrate, welches verschiedene Funktionen zur Verfügung stellt. Es werden zwei verschiedene Typen von Funktionen unterschieden. Zum einen gibt es solche, denen man eine gegebene Anzahl von Stützstellen übergibt und die dann darauf das Integral berechnen. Dies ist zum Beispiel der Fall, wenn man mit gemessenen Daten arbeitet. Zu den verfügbaren Methoden zählen unter anderem die von Ihnen implementierte Trapez- oder Simpsonregel. In *scipy* sähe eine Aufruf so aus:

```
from scipy import integrate
import numpy as np

# set up x_values with known spacing dx
x_values = np.arange(0, 2, 0.01)
# compute function values
y_values = np.cos(np.exp(-x_values))

# trapezoidal rule
print integrate.trapz(y_values, x_values, 0.01)

# simpson rule
print integrate.simps(y_values, x_values, 0.01)
```

Eine andere Möglichkeit ist, dass die Funktion noch als Vorschrift vorliegt, wie es häufig im Rahmen analytischer Rechnungen der Fall ist. Für ein-, zwei- und dreifach-Integrale stehen die Funktionen *quad, dblquad* und *tplquad* zur Verfügung. Die Hauptargumente sind die zu integrierende Funktion sowie der Integrationsbereich (a,b). Das obige Beispiel sähe dann folgendermaßen aus:

```
from scipy import integrate
from math import cos, exp

def f(x):
    return cos(exp(-x))
```

```
integral, error = integrate.quad(f, 0, 2)
print integral
```

Neben dem Wert des Integrals erhalten Sie auch gleichzeitig eine Abschätzung für den Fehler.

Wie Sie sehen, ist es sehr einfach, die in *scipy* vorhandenen Algorithmen zu benutzen. Zum Schluss möchten wir Sie darauf hinweisen, dass in dem gleichen Paket auch Routinen enthalten sind um Differentialgleichungen zu integrieren, die ähnlich einfach zu benutzen sind. Diese sind durch die Implementierung in C oft deutlich schneller als in Python geschriebene Routinen. Bei Interesse können Sie ja einmal versuchen mit der Funktion odeint Ihren Code für das Magnetpendel zu beschleunigen.

49. Integration mit gezinkten Würfeln *5 Punkte*

In dieser Aufgabe wollen wir uns mit der **Integration** von Funktionen mithilfe von **Zufallszahlen** befassen. Die ersten Schritte in diese Richtung haben wir in der Vorlesung besprochen und wollen nun an einfachen Beispielen die verschiedenen **Sampling-Methoden** untersuchen.

Beginnen wir mit der folgenden Funktion:

$$f(x) = 2 + 0.1 \cdot \cos(x), \quad x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$$

Sie können sich durch simples plotten leicht davon überzeugen, dass die Funktion relativ flach ist und sich deshalb dazu eignet mit einer gleichförmigen Verteilung integriert zu werden. Führen Sie nun eine Integration dieser Funktion mit bis zu $N = 1000$ *gleichförmig verteilten* Zufallszahlen durch und plotten Sie das Ergebnis der Integration gegen die Anzahl gezogender Zufallszahlen N. Vergleichen Sie außerdem Ihr Ergebnis mit der numerischen Lösung, die Sie mit der Python-Funktion *quad* bestimmen können.

Als nächstes betrachten wir die Funktion

$$g(x) = \cos(x) \cdot \exp\left(-\frac{x^2}{0.01}\right), \quad x \in (-\infty, \infty)$$

Bereits auf den ersten Blick ist klar, dass das Integral dieser Funktion hauptsächlich durch einen kleinen Bereich um 0 bestimmt wird. Deshalb ist es zunächst einmal zulässig den Integrationsbereich auf ein endliches Intervall, zum Beispiel $(-20, 20)$, einzuschränken. Bestimmen Sie das Integral nun wie zuvor mithilfe einer gleichförmig verteilten Zufallsverteilung und plotten Sie das Ergebnis als Funktion der Anzahl gezogener Zufallszahlen. Benutzen Sie dann die Funktion *gauss(mu, sigma)* aus dem Paket *random* um Zufallszahlen gemäßeiner Gaussverteilung mit Mittelwert *mu* und Standardabweichung *sigma* zu erzeugen. Passen Sie diese Parameter so an, dass die Form der gesampleten Gauß-Kurve möglichst genau der Funktion $g(x)$ entspricht und führen Sie auch diese Integration durch und vergleichen Sie beide Ergebnisse mit der durch *quad* gewonnenen Lösung.

Beachten Sie, dass Sie mit der Gaußverteilung Zahlen aus $(-\infty, \infty)$ erzeugen. Aufgrund der stark um den Mittelwert zentrierten Form der Verteilung ist der Fehler, den wir gegenüber der Integration mit eingeschränktem Integrationsbereich machen vernachlässigbar. Alternative könnte man nur Zufallszahlen im Bereich (-20, 20) akzeptieren und die Verteilung neu zu skalieren.

50. 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.

In der vorherigen Aufgabe 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.

51. Ising Modell

optionale Aufgabe – 6 Punkte
Abgabe am Montag, 14. Juli

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.