Institut für Theoretische Physik
der Universität zu Köln

Prof. Dr. Simon Trebst
Michael Becker, Peter Bröcker

Computerphysik

Übungsblatt 11

SS 2013

Website: http://www.thp.uni-koeln.de/trebst/Lectures/2013-CompPhys.html
Abgabedatum: Montag, 8. Juli 2013 vor Beginn der Vorlesung

41. 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.

42. Zufallszahlen-Verteilungen *5 Punkte*

Nachdem wir auf dem letzten Übungsblatt unsere eigenen Zufallszahlen-Generatoren geschrieben haben, um gleichverteilte Pseudo-Zufallszahlen im Intervall $[0,1)$ zu erzeugen, wollen wir uns nun daran machen, Zufallszahlen gemäß einer gegebenen Verteilung zu generieren. Dazu implementieren wir die **von Neumann'sche Verwerfungsmethode** (engl.: *rejection sampling*). In dieser Aufgabe wollen wir damit Zufallszahlen zu erzeugen, die einer in der Physik extrem wichtigen Verteilung, nämlich der **Fermi-Dirac-Verteilung**, folgen. Diese hängt ab von der Temperatur T und ist in normierter Form gegeben als:

$$f(E,T) = \frac{1}{T\log\left(1+\exp\left(\frac{\mu}{T}\right)\right)\left[\exp\left(\frac{E-\mu}{T}\right)+1\right]}, \tag{1}$$

wobei wir das chemische Potential auf $\mu = 1$ setzen.

Implementieren Sie nun die Verwerfungsmethode, indem Sie eine Exponentialverteilung $g(E,T)$ mit dem Erwartungswert gleich der Temperatur T als obere Schranke verwenden:

$$g(E,T) = \frac{1}{T}e^{-E/T}, \tag{2}$$

die in Python etwa durch die Routine *random.expovariate(1.0/T)* zur Verfügung gestellt wird.

Da sowohl die Fermi-Verteilung f, wie auch die Exponentialverteilung g normiert sind, müssen wir noch eine Proportionalitätskonstante $\lambda > 1$ festlegen, damit für eine gegebene Temperatur T folgende Bedingung erfüllt ist:

$$\forall E \in \mathbb{R}^+ : \quad f(E,T) < \lambda g(E,T). \tag{3}$$

Dieses λ muss groß genug, aber aus Performancegründen nicht viel zu groß, gewählt werden.Warum wird die Performance bei zu großen λ immer schlechter? Finden Sie für jede unten angegebene Temperatur einen geeigneten Wert für λ, indem Sie zunächst die beiden Verteilungen (Fermi-Dirac und Exponential-Verteilung) plotten und vergleichen.

Wir wollen nun jeweils 5.000.000 Fermi-verteilte Zufallszahlen $\{E_i\}$ für die Temperaturen $T = 0.3, 0.5, 0.8, 1.0$ erzeugen. Generieren Sie schließlich aus den erzeugten Zufallszahlen $\{E_i\}$ für jede Temperatur ein Histogramm mit 100 Bins und stellen Sie dieses jeweils gemeinsam mit

der entsprechenden Fermi-Dirac-Funktion in einem Plot dar. Untersuchen Sie, ob und wie gut Ihre Zufallszahlen der vorgegebenen Verteilung folgen.

43. 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 jedoch vernachlässigbar. Eine Alternative wäre, nur Zufallszahlen zu akzeptieren, die im Bereich (-20, 20) liegen und die Verteilung neu zu skalieren.

44. Dendriten-Wachstum

optionale Aufgabe – 9 Punkte

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.

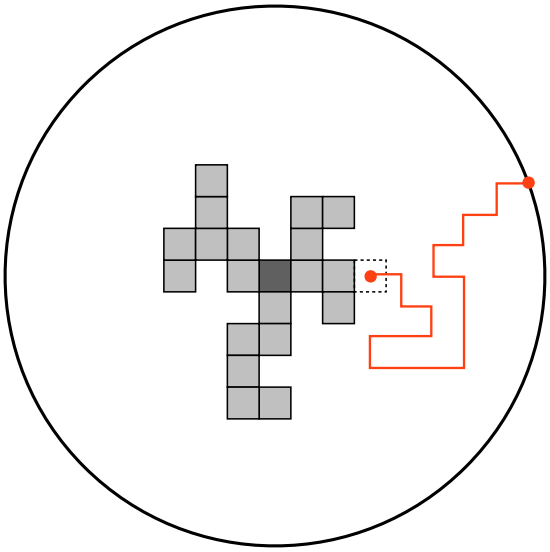

Abbildung 1: Diffusiver Zufallslauf eines Teilchens vom Rand zum Dendriten

Simulieren Sie den Wachstum eines derartigen Dendriten auf einem Raster von 128×128 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.