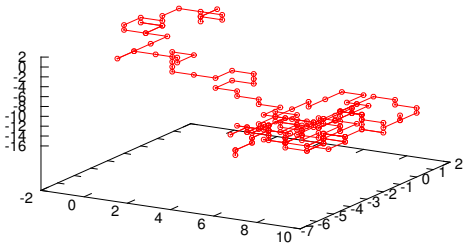
Übungen zu Computergrundlagen WS 2012/2013

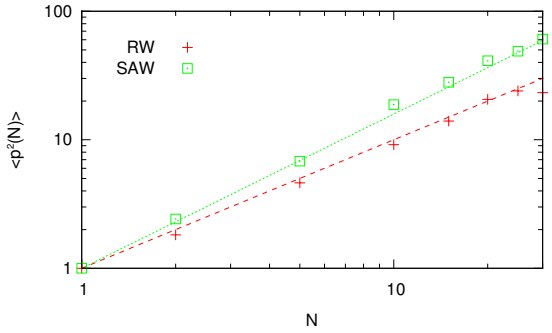
Übungsblatt 8: Python: Random Walks

7. 12. 2012

Allgemeine Hinweise

- Abgabetermin für die Lösungen ist **Freitag, 14. 12., 13:30**.
- Die Abgabe sind diesmal mehrere Python- und Gnuplot-Skripte, die Du im Anhang einer Email an Deinen Tutor schickst.
- Vergiss nicht, Deinen Namen als Kommentar an den Anfang der Python-Dateien zu schreiben.
- Denke auch daran, Deine Skripte so zu kommentieren, dass wir verstehen, was passiert.





Aufgabe 8.1: Random Walks (5 Punkte)

Ein einfaches Modell für Brownsche Bewegung ist der Random Walk (RW). Dabei beginnt man an einem Punkt im Raum (o.B.d.A dem Nullpunkt) und führt von dort aus wiederholt Schritte in einer zufälligen Richtung aus. Diese Schritte modellieren die Zusammenstöße mit den zahllosen Wassermolekülen. Der Einfachheit halber sollen diese Schritte stets Länge 1 haben und entlang einer der Koordinatenachsen führen. In drei Dimensionen gibt es also zu jedem Zeitpunkt sechs mögliche Schritte (links, rechts, hoch, runter, nach vorne, nach hinten). Ein möglicher RW ist in der Abbildung oben links gezeigt. Die ersten sechs Punkte dieses RWs sind:

$$(0,0,0) \to (0,-1,0) \to (0,-1,1) \to (1,-1,1) \to (1,0,1) \to (2,0,1) \to (2,0,2).$$

Man kann analytisch zeigen, dass der Erwartungswert für den quadratischen Abstand $p^2(N)$ des Teilchens von seiner ursprünglichen Position nach N Schritten $\langle p^2(N)\rangle = N$ ist. Dies wollen wir nun numerisch überprüfen, in dem wir den Erwartungswert in einem Computerexperiment abschätzen.

8.1.1: Ergänze die Funktion `rw(N)` so, dass sie einen dreidimensionalen RW der Länge `N` erzeugt:

```
def rw(N):
    p = [0, 0, 0]
    randomwalk = [p[:]]
    for i in range(N):
        # hier den Punkt p zufaellig um 1 versetzen
        randomwalk.append(p[:])
    return randomwalk
```

Die Rückgabe ist eine Liste der Punkte des Random Walks, die wiederum als dreielementige Listen dargestellt sind. (2 Punkte)

Hinweis: Die Routine erzeugt die Punkte nacheinander und fügt sie der Liste `randomwalk` hinzu. Zur Änderung der Koordinaten kannst Du die Funktion `randint(a,b)` des Moduls `random` benutzen.

8.1.2: Benutze nun die Funktion `rw(n)`, um ein Programm in Python zu schreiben, dass einen RW der Länge n=200 in einer für Gnuplot lesbaren Datei `rw.dat` speichert. Dazu soll jeder Punkt in einer eigenen Zeile stehen, und die drei Koordinaten durch Leerzeichen getrennt sein. Gib das Programm, das `rw.dat` erzeugt, zusammen mit der Routine `rw(n)` in einem Python-Skript ab. (2 Punkte)

Hinweis: Benutze `open`, um die Datei zum Schreiben zu öffnen, und `write`, um die Zeilen anzufügen. Diese kannst Du mit Hilfe des %-Operator formatieren. Vergiss nicht, die auszugebenden Zeilen mit „\n“ abzuschließen.

8.1.3: Stelle den RW mit Hilfe des `splot`-Befehls von Gnuplot dar. Erstelle dazu ein Gnuplot-Skript, dass den RW wie in der Abbildung links oben darstellt, und gib dieses ab. (1 Punkt)

Hinweis: Falls Du die Datei `rw.dat` (noch) nicht selber erzeugt hast, findest Du eine Ersatzdatei unter `~arnolda/computergrundlagen/08`.

Aufgabe 8.2: Self-Avoiding Walks (5 Punkte)

Der RW ist unter bestimmten Umständen auch ein Polymermodell. In einem guten Lösungsmittel benutzt man allerdings ein leicht modifiziertes Modell, den Self-avoiding Walk (SAW). Dieser ist ein Random Walk, der sich selber nicht kreuzt. Dadurch ist die mittlere Ausdehnung eines SAWs größer: es gilt nun $\langle p^2(N)\rangle \approx N^{1.2}$, wobei der exakte Exponent nur numerisch zu bestimmen ist.

8.2.1: Schreibe eine Routine `saw(n)`, die analog `rw(n)` einen SAW erzeugt. Füge diese Routine in Dein Python-Skript zur Abgabe ein! (3 Punkte)

Hinweis: Am besten kopierst Du die Funktion `rw(n)` und veränderst sie zunächst so, dass beim Hinzufügen eines neuen Punkts sofort überprüft wird, ob dieser schon im Pfad vorhanden ist. Breche in diesem Fall ab. Diese veränderte Routine erzeugt also einen SAW mit einer maximalen Länge `n`. Umgebe diese Routine nun mit einer **`while`**-Schleife, die solange durchlaufen wird, bis zufällig ein SAW der gewünschten Länge entstanden ist. Da die Bedingung einer **`while`**-Schleife stets vor dem Schleifenkörper überprüft wird, initialisiere den aktuellen Pfad als leere Liste. Bei größeren n wird diese Schleife sehr oft durchlaufen, daher teste etwa mit n=10.

8.2.2: Benutze ein Programm der Form

```
def distance(p): return sum([x**2 for x in p])
lens = [1,2,5,10,15,20,25,30]
samples = 100
for n in lens:
    srw,ssaw = 0.0, 0.0
    for r in range(samples):
        srw  += distance(rw(n)[-1])
        ssaw += distance(saw(n)[-1])
    sys.stdout.write("%d %f %f\n" % (n, srw/samples, ssaw/samples))
```

um für $n = \{1, 2, 5, 10, 15, 20, 25, 30\}$ die mittlere Distanz zu bestimmen, die ein Random Walk bzw. ein Self-Avoiding Walk in n Schritten zurücklegt. Leite die Ausgabe in eine Datei um. Schreibe nun ein Gnuplot-Skript, dass diese Datei wie auf der Vorderseite rechts oben abgebildet darstellt. Die beiden gestrichelten Linien sind N und $N^{1,2}$. (2 Punkte)

Hinweis: Auch hier gibt es eine Ersatzdatei `scaling.dat` mit vorbereiteten Daten. Es ist nicht notwendig, den Graphen exakt zu reproduzieren, aber die Skalengesetze müssen erkennbar sein.