Übungen zu Computergrundlagen WS 2012/2013

Übungsblatt 10: Wiederholung Python+Gnuplot

21. 12. 2012

Allgemeine Hinweise

- Abgabetermin für die Lösungen ist **Freitag, 11. 1. 2013, 13:30**, also im neuen Jahr!
- Abgabe wie immer als Email an Deinen Tutor.
- Vergiss nicht, Deinen Namen als Kommentar an den Anfang der Dateien zu schreiben.
- Kopierte Lösungen bringen keine Punkte und bereiten auch nicht auf die Klausur vor.

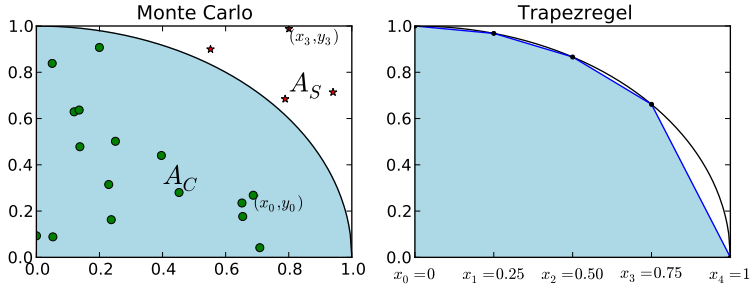
Aufgabe 10.1: Verständnisfragen (2 Punkte)

10.1.1: Was macht das folgende Codestück? Ist das sinnvoll? Wie ginge es einfacher? *Begründe* Deine Antwort! (1 Punkt)

```
l = []
for i in range(10): l.append(i)
```

10.1.2: Was macht folgendes Beispiel? Wie würdest Du diesen Code korrigieren? Beschreibe und *erkläre* die Ausgabe dieses Codes und gib eine korrigierte Version ab! (1 Punkt)

```
liste = [ "Hase", "Igel", "Hamster" ]
for i in liste: print liste
```



Aufgabe 10.2: Πthon I (4 Punkte)

Eine einfache Methode, um eine Abschätzung für die Kreiszahl π zu erhalten, ist, zufällig N Punkte (x_i, y_i) aus dem Einheitsquadrat zu ziehen ($0 \leq x_i \leq 1$, $0 \leq y_i \leq 1$). N_c sei die Anzahl der Punkte davon, die in einem Viertel des Einheitskreises (Kreis mit Radius 1) liegen (also bei denen $x^2+y^2 < 1$ ist). Da die Fläche des Einheitsquadrats $A_s = 1$ und die Fläche des Einheitskreises $A_c = \pi$ ist, gilt

$$\frac{N_c}{N} \approx \frac{\frac{1}{4}A_c}{A_s} = \frac{1}{4}\pi \implies \pi \approx 4\frac{N_c}{N}.$$

Wir können π also durch folgende Formel annähern:

$$\pi \approx 4\frac{1}{N}\sum_{i=1}^{N} \chi_{x^2+y^2<1}(x_i, y_i) \quad \text{wobei} \quad \chi_{x^2+y^2<1}(x,y) = \begin{cases} 1, & \text{falls } x^2+y^2 < 1 \\ 0, & \text{sonst.} \end{cases}$$

Dabei gilt: je größer N, desto genauer ist die Abschätzung.

```
# Fuer den Befehl random.random(), der Zufallszahlen in [0,1] erzeugt.
import random
MAX_STEPS = 1000000
step = 0
sum = 0
while step < MAX_STEPS
    x, y = random.random(), random.random()
    if x*x + y*y < 1.0 then:
        Sum = Sum + 1
step = step + 1

estimate = 4 * sum / MAX_STEPS
print "Abschätzung für pi nach $step Schritten: $estimate"
```

10.2.1: Das obige Python-Skript, das Du auch unter `~arnolda/computergrundlagen/10/compute_pi.py` findest, soll die Zahl π nach obiger Methode annähern. Leider ist es fehlerhaft. Korrigiere das Skript und gib die korrigierte Version ab. (2 Punkte)

10.2.2: Seien nun $p(N)$ die Näherungen als Funktion der Anzahl Punkte N. Erweitere das Programm so, dass es $p(N)$ tabellarisch für $N = 1, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7$ ausgibt. Erstelle ein gnuplot-Skript, um die Genauigkeit $|p(N) - \pi|$ als Funktion der Anzahl der Punkte N zu zeichnen, und gib dieses ab. Welchem Potenzgesetz folgt die Genauigkeit etwa? (2 Punkte)

Hinweis: In gnuplot ist die Lösung π als Konstante `pi` definiert. Du kannst das Potenzgesetz cN^e mit Exponent e und Vorfaktor c entweder durch Probieren erraten oder mit Hilfe der `fit`-Funktion fitten. Dabei geht es um *große* N, fange daher erst bei $N = 100$ an. Benutze eine geeignete Achsenskala, damit der Fehler für alle N gut ablesbar ist!

Aufgabe 10.3: Πthon II (4 Punkte)

Die in Aufgabe 10.2 beschriebene Methode ist nicht besonders effizient. Nun wollen wir die Genauigkeit (und damit die Effizienz) der Methode erhöhen. Dazu betrachten wir nun die Kreislinie als Funktion

$$y = f(x) = \sqrt{1 - x^2}$$

Wieder wollen wir die Fläche des Kreissegments (also die Fläche unter der Funktion) annähern, d.h. wir *integrieren* die Funktion $f(x)$ numerisch. Ein bekanntes Verfahren dazu ist die *Trapezregel*, bei der die x-Achse in gleich große Abschnitte $[x_i, x_{i+1})$ unterteilt wird, und die Fläche unter der Funktion in diesen Abschnitten durch ein Trapez angenährt wird (vgl. Abbildung oben rechts).

10.3.1: Die Funktion `numpy.trapz(y, x)` berechnet bei gegebenen Punkten `x`=(x_i) und `y=f(x)` den näherungsweisen Flächeninhalt unter $f(x)$ (Achtung, gegenüber der intuitiven Form sind die Parameter vertaucht, `y` kommt *vor* `x`). Schreibe eine Funktion `est_pi(N)`, die mit Hilfe von `numpy.trapz` den Flächeninhalt unter $f(x)$ berechnet, wobei die Punkte `x` den Bereich [0,1] gleichförmig in N Abschnitte unterteilen sollen. Hierzu kannst Du die Funktion `numpy.linspace` benutzen. (2 Punkte)

10.3.2: Benutze nun matplotlib in dem Skript, um wie in Aufgabe 10.2.2 die Genauigkeit als Funktion der Zahl Punkte N zu zeichen. Welches Potenzgesetz erhälst Du nun? Welches Verfahren ist besser? (2 Punkte)

Hinweis: Zum Zeichnen musst Du (z.B. mit Hilfe von `append`) eine Liste mit den gewünschten N erstellen, und eine zweite, gleich lange Liste mit den zugehörigen Schätzungen. Fitten geht nicht ganz so leicht mit NumPy/matplotlib, daher musst Du hier das Gesetz raten oder nochmal gnuplot bemühen (nur zum Fitten, nicht zur Ausgabe!). Fange wieder bei $N = 100$ mit dem Fitten an!