Übungen zu Physik auf dem Computer SS 2012

Übungsblatt 4: Fouriertransformationen

8. Mai 2012

Allgemeine Hinweise

- Abgabetermin ist **Montag, 14.5.2012, 13:00**
- Zur Abgabe schickst Du die Lösungsdatei(en) im Anhang einer Email an Deinen Tutor:
  - Florian (`floh@icp.uni-stuttgart.de`; Dienstag, 15:45–17:15)
  - Dominic (`dominic@icp.uni-stuttgart.de`; Dienstag, 15:45–17:15)
  - Olaf (`olenz@icp.uni-stuttgart.de`; Mittwoch, 15:45–17:15)
- Die Übungen werden in Gruppen von jeweils zwei oder drei Leuten bearbeitet. Diese dürfen sich gerne von Blatt zu Blatt unterscheiden. Aus formalen Gründen muss allerdings jeder von Euch eine eigene Lösung abgeben. Schreibt bitte auf die Lösungen, mit wem Ihr zusammengearbeitet habt, um uns das Korrigieren zu erleichtern.
- Die Übungen finden statt im CIP-Pool des Instituts für Computerphysik (ICP) im Pfaffenwaldring 27.

Aufgabe 4.1 (4 Punkte): Fouriertransformation in NumPy

Im folgenden betrachten wir die periodische Rungefunktion $f(x) = \frac{1}{1+\cos^2(x)}$ im Wertebereich $[0, 2\pi]$ und die Lennard-Jones-Funktion $g(x) = x^{-12} - x^{-6}$ im Wertebereich $[1, 5]$.

- 4.1.1 (2 Punkte) Schreibe ein Pythonskript, das folgendes für die beiden Funktionen tut:
  1. Es erzeugt eine Datenreihe von 2048 äquidistanten Punkten im jeweiligen Wertebereich und eine Reihe von Funktionswerten an diesen Stützstellen.
  2. Es verwendet die Pythonfunktion `numpy.fft.rfft()`, um die Fourierkoeffizienten dieser Datenreihe zu berechnen.
  3. Es schneidet die Koeffizientenreihe nach den ersten 5, 10 bzw. 50 Koeffizienten ab.
  4. Es transformiert die abgeschnittene Fourierreihe mit Hilfe der Pythonfunktion `numpy.fft.irfft()` in den Normalraum zurück. Dabei wird als Länge der Ausgabereihe wieder 2048 verwendet.

  Gib das Pythonskript als Lösung ab.
- 4.1.2 (1 Punkt) Erzeuge für jede der Funktionen einen Plot, der die Funktion und die aus der abgeschnittenen Fourierreihe rekonstruierten Funktionen im angegebenen Wertebereich zeigt. Gib die Plots im PDF-Format als Lösung ab.
- 4.1.3 (1 Punkt) Wieso zeigt die Rekonstruktion der Funktion $h(x)$ am rechten Rand selbst bei einem Grad von 50 noch Artefakte, während die Funktion $g(x)$ das selbst beim Grad 10 nicht tut? Schreibe die Antwort als Lösung in die Lösungsemail.

Aufgabe 4.2 (3 Punkte): Diskrete Fouriertransformation (DFT)

- 4.2.1 (2 Punkte) Implementiere eine Pythonfunktion `dft_forw()` die eine diskrete Fouriertransformation einer Datenreihe durchführt, sowie eine Pythonfunktion `dft_back()` zur Rücktransformation. Diese Funktionen sollten identische Ergebnisse zu den Funktionen `numpy.fft.fft` und `numpy.fft.ifft` ergeben.

  Gib ein Pythonskript mit den funktionierenden Pythonfunktionen als Lösung ab.

  Hinweis In den Pythonfunktionen kannst Du den Datentyp `numpy.complex` verwenden. Die komplexe Zahl $x = 1 + 2i$ kannst Du dann in Python als `x=1.0+2.0j` schreiben.

- 4.2.2 (1 Punkt) Implementiere zwei weitere Pythonfunktionen `rdft_forw()` und `rdft_back()`, die speziell mit rein reellen Datenreihen umgehen können und dann nur die Hälfte der Koeffizienten benötigen. `rdft_back(a, n)` erhält dabei einen zweiten, optionalen Parameter, mit dem man (wie in der Funktion `numpy.fft.irfft()`) die Länge der Ausgabereihe angeben können soll.

  Gib ein Pythonskript mit den funktionierenden Pythonfunktionen als Lösung ab.

  Hinweis Diese Pythonfunktionen sollten dasselbe tun, wie die Pythonfunktionen `numpy.fft.rfft()` und `numpy.fft.irfft()`. Insbesondere sollte es damit möglich sein, das Skript aus der vorigen Aufgabe zu verwenden.

Aufgabe 4.2 (3 Punkte): Schnelle Fouriertransformation (FFT)

- 4.3.1 (2 Punkte) Implementiere die Pythonfunktionen `fft_forw()` und `fft_back()`, die dasselbe tun, wie die Pythonfunktionen `dft_forw()` und `dft_back()` aus der vorigen Aufgabe, dabei aber den Algorithmus der schnellen Fouriertransformation verwendet.

  Gib das Skript mit den funktionierenden Pythonfunktionen als Lösung ab.

- 4.3.2 (1 Punkt) Messe die Laufzeiten der in Aufgabe 4.2 implementierten DFT-Funktionen, der in 4.3.1 implementierten FFT-Funktionen und den FFT Funktionen von NumPy für $N \in 16, 64, 256, 1024, 4096$ mittels der Pythonfunktionen `timeit.timeit`. Erzeuge einen doppeltlogarithmischen Plot mit den Laufzeiten über N.

  Gib den Plot im PDF-Format als Lösung ab.