Computerphysik WS 2012/2013

Prof. Dr. Roland Netz, FU Berlin

Übungsblatt 1: Einführung und Numerische Genauigkeit

Alexander Schlaich, Matej Kanduč
17. Oktober 2013

Allgemeine Hinweise

Abgabetermin für die Lösungen ist

- **Sonntag, 27.10., 24:00 Uhr.**

Die Lösungen solltest Du in eine Textdatei kopieren, deren erste Zeile Deinen Namen, die Nummer des Übungsblattes und den Namen Deines Tutor enthält. Zur Abgabe schickst Du die Lösungsdatei und evtl. Plots oder Rechnungen im Anhang einer Email an Deinen Tutor.
Die Aufgaben sollen in Gruppen von maximal 2 Personen bearbeitet werden. Denke daran, dass aus der Lösungsdatei der Name des Mitbearbeiters ersichtlich sein muss.

Aufgabe 1.1: Fließkommastandard nach IEEE 754 (6 Punkte)

Moderne Computer verarbeiten Fließkommazahlen intern nach dem Standard IEEE 754[1]. Dabei wird eine Gleitkommazahl x dargestellt als Binärzahlen der Form

$$x = s \cdot m \cdot b^e, \tag{1}$$

mit dem Vorzeichen s (0 oder 1), der Mantisse m, der Basis $b = 2$ und dem Exponent e. Dazu definiert das *Institute of Electrical and Electronics Engineers (IEEE)* genaue Verfahren für mathematische Operationen und Rundungen, insbesondere legt es fest, wieviele Bit zum speichern der Mantisse und des Exponenten für einen bestimmten Datentyp verwendet werden.

- 1.1.1 (2 Punkte) Schreibe ein Python-Programm, das den folgenden Pseudo-Code implementiert:
  - Initialisiere $\epsilon = 1$, $\alpha = 1$.
  - Solange ($\alpha \neq \alpha + \epsilon$) halbiere ϵ.
  - Gib den Wert von ϵ aus.

  Führe das Programm aus und notiere die Werte von ϵ für verschiedene Startwerte von $\alpha = [1, 1 \cdot 10^{-5}, 1 \cdot 10^{-30}, 5 \cdot 10^{12}]$ in Deiner Lösungsdatei.
- 1.1.2 (2 Punkte) Welche Bedeutung hat der Wert von ϵ?
- 1.1.3 (2 Punkte) Kann man ϵ auch aus dem IEEE Standard bestimmen? Welcher Wert ergibt sich für eine 32 bit single precision Fließkommazahl?

[1] siehe `http://de.wikipedia.org/wiki/IEEE_754`

Aufgabe 1.2: Rundungsfehler und Gleitkommaarithmetik (6 Punkte)

Wandelt man eine n-stellige Dezimalzahl $\tilde{x}$ in eine Gleitkommazahl $x = \pm(0.z_1 z_2 \dots z_n) \cdot 2^e$ im Binärsystem um, so wird der dabei enstehende Rundungsfehler in allen weiteren Berechnungen fortgepflanzt. Um hierfür einen einheitlichen Standard zu schaffen, sind verbindliche Regeln zur internen Rechengenauigkeit, die so genannte *Gleitkommaarithmetik*, festgelegt. Addiert man jedoch mehrere Zahlen der gleichen Größenordnung zu einer deutlich größeren Zahl, kann es durch Rundungsfehler zu sogenannten Auslöschungen kommen.

- 1.2.1 (2 Punkt): Beschreibe mit eigenen Worten die Begriffe Gleitkommaarithmetik und Auslöschung. Warum kann numerisch die Reihenfolge der Summation einen Einfluss auf das Ergebnis haben?
- 1.2.2 (2 Punkt): Schreibe ein kurzes Programm, das die Leibniz-Reihe

$$s_k = \sum_{i=1}^{k} \frac{(-1)^i}{2i+1} \overset{k\to\infty}{\longrightarrow} \frac{\pi}{4} \tag{2}$$

  numerisch für $k = 500$ auswertet. Benutze dabei die untenstehende Funktion `mround(x, n)` um die Summanden der Reihe nur mit einer Genauigkeit von $n = 4$ bzw. $n = 6$ Stellen zu berechnen. Wofür wird `mround(x,n)` benötigt?
  Was ist das Ergebnis bei Verwendung von single precision? Welches Ergebnis erhält man bei Summation in umgekehrter Reihenfolge?
- 1.2.3 (2 Punkt): Schätze analytisch ab, ab welchem Wert k das Ergebnis $s_k = \sum_{i=1}^{k} \frac{1}{i^2}$ für $n = 4$ bzw. $n = 6$ stagniert. Wie lässt sich das analytisch abschätzen? Überprüfe Dein Ergebnis durch das Erweitern Deines Programmes.

```
def mround(x,N):
    if (x==0):
        return x
    return round(x, int(N - math.ceil(math.log10(abs(x)))))
```

Aufgabe 1.3: Taylorentwicklung (8 Punkte)

Für viele alltägliche Anwendungen ist es nötig, mittels eines Computers schnell und effizient transzendente Funktionen wie beispielsweise sin, cos, exp oder log auszuwerten. Um die Berechnungen zu ermöglichen, werden entweder sog. Nachschlagetabellen verwendet, in denen bereits vorberechnete Werte gespeichert werden, oder es muss ein Näherungsverfahren angewendet werden. Moderne Software-Bibliotheken greifen dafür auf effiziente Algorithmen wie beispielsweise *CORDIC* zurück, die auf Koordinatentransformationen mittels Drehmatrizen zurückgreifen.
Eine weitere Alternative, welche wir in dieser Aufgabe anwenden möchten, ist die Verwendung eines Taylorpolynoms, welches aus der Definition der Exponentialfunktion folgt:

$$\cos(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = \frac{x^0}{0!} - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots \tag{3}$$

- 1.3.1 (3 Punkte): Implementiere in Python eine Funktion `tcos(x, N)`, welche das Taylorpolynom der Sinusfunktion der Ordnung N an einem Punkt x auswertet und den berechneten Wert zurückgibt:

$$\mathrm{tcos}_N(x) = \sum_{n=0}^{N} (-1)^n \frac{x^{2n}}{(2n)!} \tag{4}$$

- 1.3.2 (3 Punkte): Benutze die Python-Bibliotheken *matplotlib* und *numpy*, um $\mathrm{tcos}_N(x)$ im Intervall $[0; 2\pi)$ für verschiedene Werte von N darzustellen. Achte auf eine aussagekräftige Ausgabe (Achsenskalierung, Beschriftung, etc.) und vergleiche außerdem mit der Methode `numpy.cos(x)`. Für welche Werte N kann die Taylorentwicklung qualitativ den Verlauf der Cosinus-funktion wiedergeben? Speichere dein Ergebnis als PNG Grafik ab.
- 1.3.3 (2 Punkte): Erweitere Dein Programm mittels `time.clock()` aus dem Python-Modul `time`, um die Zeit zu Beginn und am Ende der Methode `tcos` zu bestimmen. Lass die Laufzeit der verschiedenen Werte von N ausgeben.

  Welches Verhalten kannst Du beobachten? Wie verhält sich `numpy.cos()`? Wie kann man die Beobachtungen begründen?