BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Prof. Dr.-Ing. Kristian Hildebrand

Visual and Scientific Computing
Gleitkommazahlen und Lineare Algebra

WiSe 2019/20

Lesen Sie bitte folgende wichtige Standardveröffentlichung zu diesem Thema mit dem Titel: *What Every Computer Scientist Should Know About Floating-Point Arithmetic* http://www.lsi.upc.edu/~robert/teaching/master/material/p5-goldberg.pdf.

Das Lernziel dieser Übungsaufgabe ist, die Problematik der Gleitkommazahlendarstellung zu rekapitulieren und Ihre Kenntnisse der Linearen Algebra aufzufrischen und in Verbindung mit Numpy anzuwenden.

Aufgabe 1: Gleitkommazahlen (6 Punkte)

a) Schreiben Sie ein Python Programm, welches die Maschinengenauigkeit der von Python verwendeten Gleitkommazahlen ermittelt. Die Maschinengenauigkeit ist definiert als die kleinste positive Gleitkommazahl ϵ, so dass $1 + \epsilon = 1$ ist. Wiederholen Sie das Experiment unter Verwendung der numpy Datentypen float32 und float64. Diese lassen sich mittels der Funktionen *numpy.float32(x)* und *numpy.float64(x)* erzeugen, z.B.:

```
1 import numpy as np
2 eps32 = np.float32(1)
3 eps64 = np.float64(1)
```

Nutzen Sie die Funktion *type*, um zu überprüfen, dass alle Berechnungen mit dem jeweiligen numpy Datentypen durchgeführt werden. **Hinweise:** In jeder Iteration soll ein ϵ auf 1.0 addiert werden und mit der Floating-Point Darstellung von np.float64(1) bzw. np.float32(1) verglichen werden. Starten Sie dabei mit $\epsilon = 1.0$ und halbieren Sie den Wert in jeder Iteration (siehe Source Code). Stoppen Sie die Iteration, wenn np.float32(1) + epsi == np.float32(1) ist (bzw. float64).

b) Das wohl älteste bekannte Verfahren zur Bestimmung des Kreisumfanges geht auf Archimedes zurück (ca. 250 v.Chr) und approximiert den Kreisumfang mittels einbeschriebener regelmäßiger Polygone. Durch Verdopplung der Kanten wird die Approximation sukzessive verbessert (siehe Abbildung 2). In dieser Aufgabe wollen wir untersuchen, ob sich das Verfahren für Gleitkommazahlen geeignet auf einem Computer implementieren lässt.

Der Umfang eines Kreise ist bekanntlich $U = 2\pi r$ und für den Einheitskreis damit 2π.

Gegeben ein Einheitskreis lässt sich zunächst der Umfang eines Quadrates nach dem Satz des Pythagoras wie folgt berechnen (siehe Abbildung 2(a)): $s_2^2 = r^2 + r^2 = 1 + 1 \Rightarrow s_2 = \sqrt{2}$. s_n stellt dabei das 2^n-Eck dar. 2^3 ist dann ein Achteck (siehe Abb. 2(b)), 2^4 ein Sechzehneck und so weiter. Die Seitenlänge eines s_{n+1}-Ecks lässt sich daher aus der Seitenlänge eines s_n-Ecks wie folgt herleiten (siehe Abb. 2(c)):

$$(\frac{s_n}{2})^2 + \overline{BE}^2 = \overline{BA}^2 = 1 \Rightarrow \overline{BE} = \sqrt{1 - (\frac{s_n}{2})^2}$$

$$s_{n+1}^2 = \overline{ED}^2 + (\frac{s_n}{2})^2 = (r - \overline{BE})^2 + (\frac{s_n}{2})^2$$

$$= \left(1 - \sqrt{1 - (\frac{s_n}{2})^2}\right)^2 + (\frac{s_n}{2})^2$$

$$= 1 - 2\sqrt{1 - (\frac{s_n}{2})^2} + 1 + (\frac{s_n}{2})^2 + (\frac{s_n}{2})^2$$

Nach einigem Umformen erhält man:

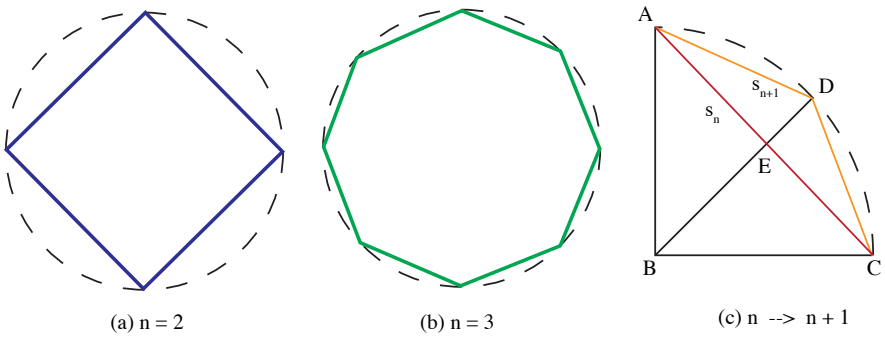
$$s_{n+1} = \sqrt{2 - \sqrt{4 - s_n^2}}$$

Schreiben Sie ein Python Programm, welches unter Verwendung der hergeleiteten Formel den Kreisumfang möglichst genau approximiert. Schauen Sie sich dafür den vorgegebenen SourceCode in *floating_point.py* an. Geben Sie die jeweilige Approximation und den Fehler auf der Console für aufsteigende n aus. Erzeugen Sie außerdem unter Verwendung von matplotlib einen Graph des Fehlers in Abhängigkeit von n. Welche Genauigkeit lässt sich maximal erreichen?

c) Problematisch an der obigen Gleichung ist die Subtraktion unter der ersten Wurzel. Für kleine s_n wird hier die Differenz von fast zwei gleich großen Zahlen gebildet, was zur Verstärkung des relativen Fehlers führt. Um diese Art der Auslöschung zu verhindern, kann man die hergeleitete Formel erweitern und das Problem vermeiden. Hier wird mal wieder ein kleiner mathematischer Trick angewandt:

$$\begin{aligned} s_{n+1} &= \sqrt{2 - \sqrt{4 - s_n^2}} \\ &= \sqrt{2 - \sqrt{4 - s_n^2}} \cdot \frac{\sqrt{2 + \sqrt{4 - s_n^2}}}{\sqrt{2 + \sqrt{4 - s_n^2}}} \\ &= \frac{\sqrt{2^2 - \sqrt{4 - s_n^2}^2}}{\sqrt{2 + \sqrt{4 - s_n^2}}} \\ &= \frac{s_n}{\sqrt{2 + \sqrt{4 - s_n^2}}} \end{aligned}$$

Implementieren Sie auch diese Funktion und plotten Sie genau wie oben einen Fehlergraphen und den Fehler auf die Console. **Was sehen Sie in den Plots? Schreiben Sie Ihre Interpretation in eine kleine Textdatei oder im Sourcecode auf.**



(a) n = 2 (b) n = 3 (c) n --> n + 1

Abbildung 1: Approximation des Kreisumfangs nach Archimedes.

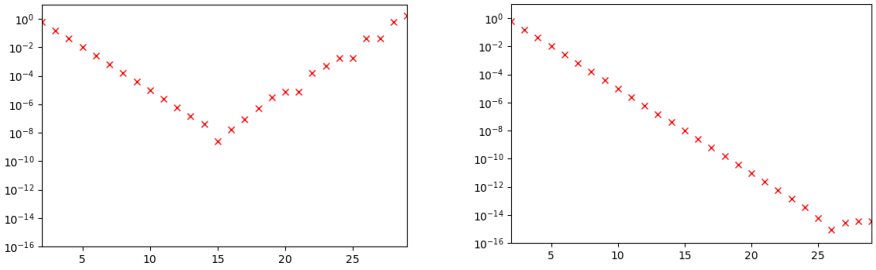


Abbildung 2: Ergebnisse der Aufgabe 1(b) links und 1(c) rechts.

Aufgabe 2: Einführung in die Lineare Algebra (4 Punkte)

Implementieren Sie folgende Punkte mit numpy:

a) Berechnen Sie den Winkel α in Grad zwischen den folgenden beiden Vektoren $a = [1., 1.77]$ und $b = [1.5, 1.5]$

b) Gegeben ist die quadratische reguläre Matrix A und ein Ergbnisvektor b

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & 4 \\ 3 & -2 & -1 \\ 5 & 4 & 3 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1.4 \\ 1.2 \\ 1.4 \end{bmatrix}$$

Rechnen Sie über unter Nutzung der Inversen die Lösung x des Gleichungssystems $Ax = b$ aus.

c) Schreiben Sie eine Funktion die das Matrixprodukt berechnet. Nutzen Sie dafür nicht die Numpy Implementierung.

Abgabe Die Bearbeitungszeit der Teilaufgabe ist für ca. eine Woche ausgelegt, kann aber aufgrund der Anfangshürden etwas mehr Zeit benötigen. Die Abgabe soll via Moodle bis zu dem dort angegebenen Termin erfolgen. Verspätete Abgaben werden mit einem Abschlag von 3 Punkten je angefangener Woche Verspätung belegt. Geben Sie bitte jeweils nur eine einzige .zip-Datei mit den Quellen Ihrer Lösung ab.

Ich werde mich stichprobenartig Ihre Lösungen in der Übung demonstrieren und erläutern lassen. Die Qualität Ihrer Demonstration ist, neben dem abgegebenen Code, ausschlaggebend für die Bewertung! Ich behalte mir vor, einzelne hochgeladene Aufgaben zusätzlich anzuschauen.