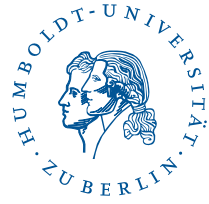


Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Mathematik



Bericht

Untersuchung der harmonischen Reihe

Arbeitsblatt 4

Chantal Gerth, Alina Apel
635838, 614787

Gruppe 22

Inhaltsverzeichnis

1	Einführung und Motivation	3
2	Theoretische Grundlagen	3
3	Experiment	4
3.1	Ablaufplan	4
3.2	Ergebnisse	6
3.3	Auswertung	8
4	Schlussfolgerung	9
5	Python-Dokumentation tools_read_save	10
5.1	Schnittstellendokumentation	10
5.2	Nutzungshinweise und Hauptprogramm	14
6	Literaturverzeichnis und Übersicht verwendeter Hilfsmittel	16
7	Selbstständigkeitserklärung	17

1 Einführung und Motivation

Im heutigen Zeitalter werden Computer in fast allen Lebensbereichen genutzt. Daher ist es besonders wichtig, dass der Computer korrekt rechnet. Leider tut er das nicht immer. Ein anschauliches Beispiel dafür ist die harmonische Reihe. Deswegen geht es in diesem Experiment darum, die Konvergenz der harmonischen Reihe mit verschiedenen Algorithmen und Datentypen zu untersuchen und dies mit dem analytischen Ergebnis zu vergleichen.

2 Theoretische Grundlagen

Die harmonische Reihe ist eine unendliche Reihe der Form

$$\sum_{n=1}^{\infty} \frac{1}{n},$$

die aus den Kehrwerten natürlicher Zahlen besteht. Mathematisch gesehen divergiert die harmonische Reihe, was bedeutet, dass die Summe der Reihe gegen Unendlich strebt, wenn die Anzahl der Terme zunimmt. Beim Berechnen der Partialsummen der harmonischen Reihe durch einen Computer kann es allerdings zu Rundungsfehlern kommen, die durch die begrenzte Genauigkeit der Fließkommazahlen verursacht werden. Je nach Datentyp wird n so groß und damit der auf die vorherige Partialsumme aufzuaddierende Summand so klein, dass die Bitmuster der entsprechenden Datentypen nicht mehr ausreichen, um diese numerisch kleine Zahl darzustellen. Der entsprechende Summand wird deshalb auf 0 gerundet. Für höhere n wird dann nur noch 0 aufaddiert, sodass die Folge der Partialsummen zu konvergieren scheint. Die Genauigkeit der Ergebnisse hängt dabei vom verwendeten Datentyp und Algorithmus ab. Je weniger Nachkommastellen das Bitmuster eines Datentyps darstellen kann, desto früher tritt das Problem auf. Schlussendlich trat in allen von uns untersuchten Beispielen diese Konvergenz auf, obwohl die Reihe selbst divergiert.

3 Experiment

3.1 Ablaufplan

Im Experiment soll die Konvergenz der harmonischen Reihe bei einer Berechnung der Folge der Partialsummen mit dem Computer beobachtet werden. Aus dem theoretischen Hintergrund lässt sich die Vermutung ableiten, dass je nach genutztem Datentyp trotz der mathematischen Divergenz der Reihe eine scheinbare Konvergenz der vom Computer berechneten Folge der Partialsummen beobachtet werden kann.

Für die Durchführung des Experiments wird das Experimentierskript `main.py` genutzt. Für die Berechnung wird ein Anfangswert und ein Endwert, sowie eine Basis benötigt. Dann kann der Nutzer wählen, wie viele Partialsummen er berechnet haben will. Die Eingabe der Parameter erfolgt über den Terminal. Dafür wird die Funktion `read_number` aus dem Modul `tools.read.save` (Abschnitt 5) genutzt. Wählt der Nutzer die vordefinierten Parameter, wird mit `start = 1`, `stop = 5`, `basis = 10`, `num = 300` fortgefahren.

Anschließend berechnet das Programm durch Vorwärtssummation und Rückwärtssummation mit drei verschiedenen Datentypen die Partialsummen und stellt die Ergebnisse graphisch dar. Damit alle Berechnungen im entsprechenden Datentyp erfolgen, wird folgende Implementierung der Summationsalgorithmen genutzt:

```
for k in logspace: # fuer jede der gewuenschten Partialsummen
    partialsumme = data_type(0) # Casting zum gewuenschten Datentyp
    for i in range(1, k+1): #Vorwaertsschleife
        partialsumme += data_type(1)/data_type(i)
    result.append(partialsumme)
```

Dieses Vorgehen ist analog für die Rückwärtssummation (nur dass die Schleife dort rückwärts läuft).

Für das Experiment wurden folgende drei Datentypen gewählt: `np.float16`, `np.float32` und `np.float64`. Mittels `np.iinfo([datentyp])` kann überprüft werden, welche maximale Zahl mit dem Bitmuster des jeweiligen Datentyps dargestellt werden kann.

Für `np.float16` beträgt die maximale darstellbare positive ganze Zahl 32767. Demnach ist zu erwarten, dass bei Zugriff auf größere Zahlen (z.B. im Zähler der auf die vorherige Partialsumme aufsummierten rationalen Zahlen) Rechenfehler auftreten.

Für `np.float32` beträgt die maximale darstellbare positive ganze Zahl 2147483647.

Demnach ist zu erwarten, dass bei Zugriff auf größere Zahlen (z.B. im Zähler der auf die vorherige Partialsumme aufsummierten rationalen Zahlen) Rechenfehler auftreten. Für `np.float32` sollten jedoch erst bei größeren Partialsummen als für `np.float16` Rechenfehler auftreten, die Konvergenz der Reihe sollte bei einem größeren Grenzwert eintreten. Für `np.float64` beträgt die maximale darstellbare positive ganze Zahl 9223372036854775807. Die Laufzeit der zur Vorwärts- und Rückwärtssummation verwendeten Algorithmen lässt eine Berechnung solch großer Partialsummen der harmonischen Reihe nicht zu. Dies lässt vermuten, dass in allen von uns durchgeführten Experimenten unter Verwendung von `np.float64` die Divergenz der harmonischen Reihe erkennbar sein wird.

Zu jedem der drei beschriebenen Datentypen werden Berechnungen mit folgenden Parametern durchgeführt:

- `start = 0, stop = 10, basis = 2, num = 300`
- `start = 0, stop = 15, basis = 2, num = 300`
- `start = 0, stop = 23, basis = 2, num = 300`
- `start = 0, stop = 25, basis = 2, num = 300`
- `start = 0, stop = 5, basis = 7, num = 300`
- `start = 0, stop = 8, basis = 7, num = 300`
- `start = 0, stop = 26, basis = 2, num = 300`

`num = 300` bedeutet, dass jedes Mal 300 Partialsummen berechnet werden. Die Berechnungen sollen graphisch dargestellt werden.

Für diesen Experimentierplan ergeben sich somit folgende Hypothesen:

Hypothese 1: Obwohl die harmonische Reihe divergiert, kann bei der Berechnung der Folge der Partialsummen mit dem Computer eine Konvergenz beobachtet werden.

Hypothese 2: Diese Konvergenz tritt für Datentypen mit einem größeren verfügbaren Bitmuster erst bei größeren Partialsummen auf als für Datentypen mit einem kleineren verfügbaren Bitmuster.

3.2 Ergebnisse

Nach der Durchführung des in Unterabschnitt 3.1 beschriebenen Experiments ergaben sich folgende Ergebnisse für die Basis 2 und jeweils 300 berechnete Partialsummen:

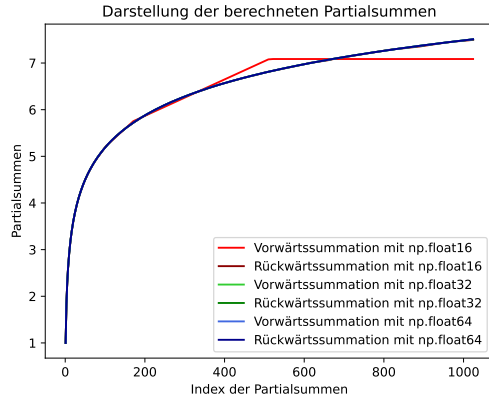


Abbildung 1: Startwert = 0, Endwert = 10, Basis = 2

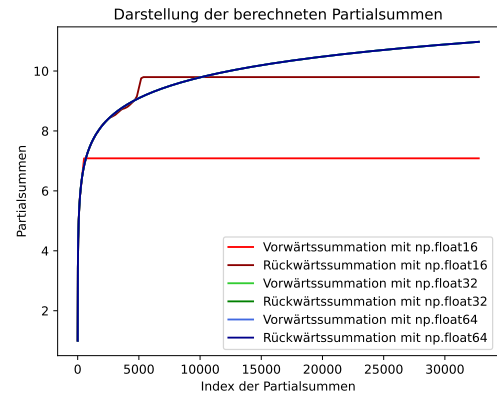


Abbildung 2: Startwert = 0, Endwert = 15, Basis = 2

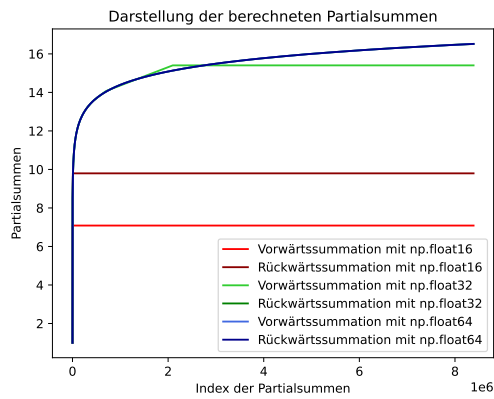


Abbildung 3: Startwert = 0, Endwert = 23, Basis = 2

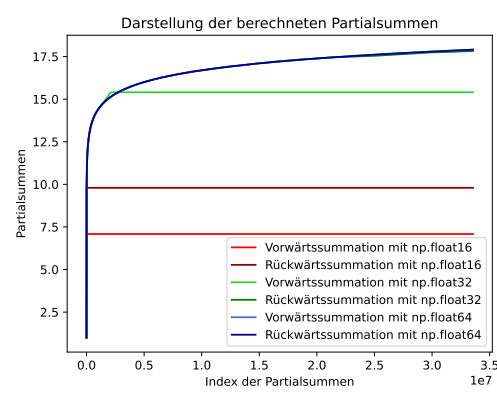


Abbildung 4: Startwert = 0, Endwert = 25, Basis = 2

Bereits in Abbildung 1 ist für den Datentyp `np.float16` in der Vorwärtssumme eine Konvergenz zu erkennen, in Abbildung 2 auch für die Rückwärtssumme. Werden noch mehr Partialsummen berechnet, zeigt sich auch für `np.float32` in der Vorwärtssumme eine Konvergenz (siehe Abbildung 3 und Abbildung 4). Den Erwartungen entsprechend ergab sich dies auch für die Experimente zur Basis 7:

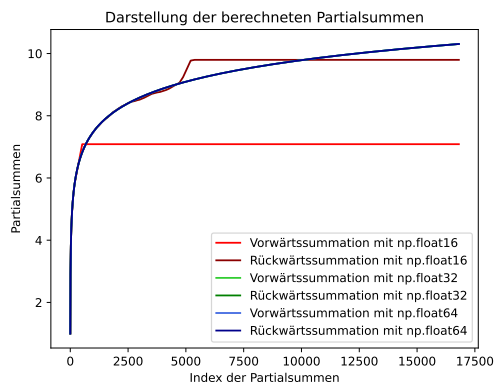


Abbildung 5: Startwert = 0, Endwert = 7, Basis = 7

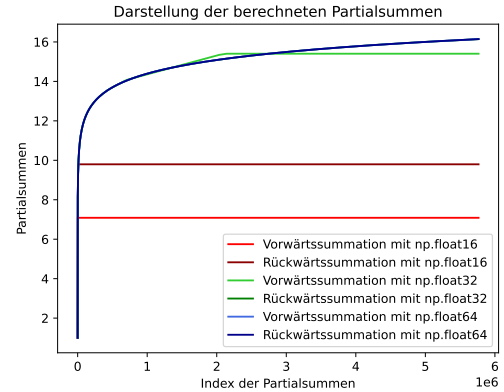


Abbildung 6: Startwert = 0, Endwert = 8, Basis = 7

Die Werte der in diesem Experiment größten berechneten Partialsumme (Index $2^{26} = 67108864$) ist in Abbildung 7 dargestellt.

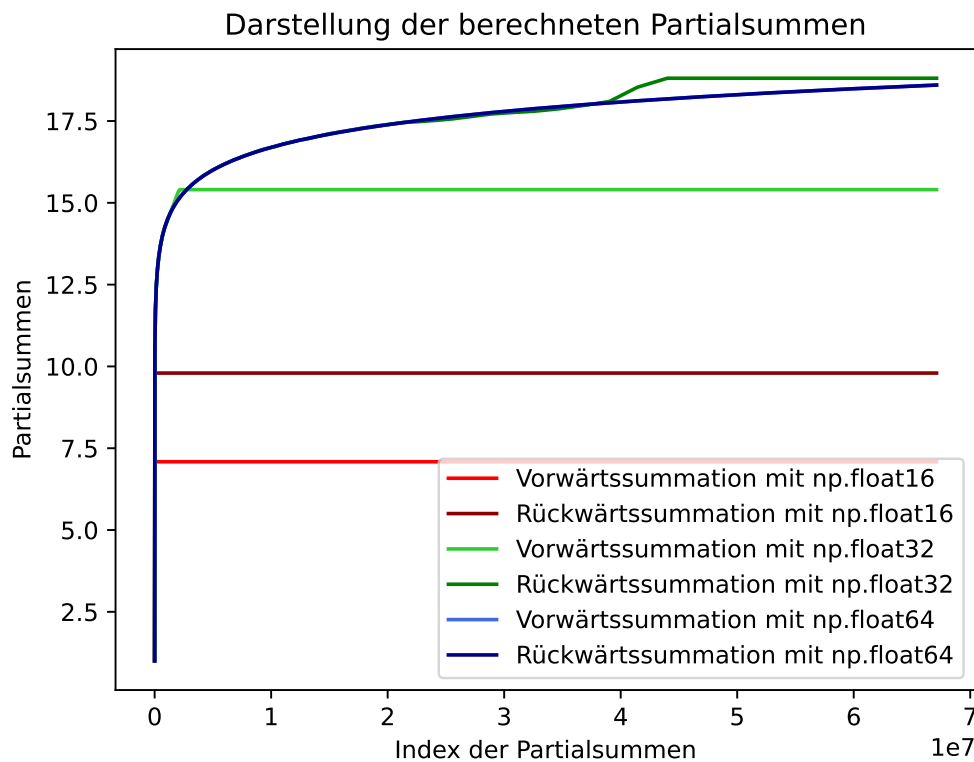


Abbildung 7: Startwert = 0, Endwert = 26, Basis = 2

Für `np.float16` und Vorwärtssummation konvergierte die Reihe gegen 7.086. Für `np.float16` und Rückwärtssummation konvergierte die Reihe gegen 9.797. Für `np.float32` und Vorwärtssummation konvergierte die Reihe gegen 14.404. Für `np.float32` und Rückwärtssummation konvergierte die Reihe gegen 18.808. Für `np.float64` konnte keine Konvergenz beobachtet werden.

Bei jeder Berechnung, die zu einer Konvergenz der Reihe führte, trat eine Warnmeldung auf: `RuntimeWarning: overflow encountered in cast partialsumme += data_type(1)/data_type(i).`

3.3 Auswertung

In Unterabschnitt 3.2 wurde in mehreren Beispielen gezeigt, dass bei der Berechnung der Partialsummen der harmonischen Reihe mit dem Computer eine Konvergenz beobachtet werden kann, obwohl die harmonische Reihe mathematisch belegbar divergiert. Hypothese 1 kann somit angenommen werden.

Dies ist darauf zurückzuführen, dass je nach verwendetem Datentyp nur Zahlen in gewisser Größe durch das verfügbare Bitmuster dargestellt werden können. Wird eine Zahl größer als im entsprechenden Datentyp darstellbar, kommt es zu einem sogenannten *Overflow*, welcher in Form der beobachteten Warnmeldung ersichtlich war. Dieser Fehler führt jedoch nicht zum Programmabbruch, sondern stattdessen wird so gerundet, dass der auf die vorherige Partialsumme aufzuaddierende Summand 0 beträgt.

Die Genauigkeit der Ergebnisse hängt dabei vom verwendeten Datentyp und Algorithmus ab. Je kleiner die maximale Zahl, die das Bitmuster eines Datentyps darstellen kann, desto früher tritt das Problem auf. Dies erklärt die Beobachtung, dass die Konvergenz für `np.float16` früher eintritt als für `np.float32`. Somit kann auch Hypothese 2 angenommen werden: Die mit dem Rundungsfehler verbundene Konvergenz tritt für Datentypen mit einem größeren verfügbaren Bitmuster erst bei größeren Partialsummen auf als für Datentypen mit einem kleineren verfügbaren Bitmuster.

Für die Untersuchung der Fragestellung bezüglich `np.float64` ist eine Wahl höherer Parameter erforderlich, was jedoch mit einer erheblich längeren Laufzeit verbunden wäre.

4 Schlussfolgerung

Basierend auf den Ergebnissen unseres Experiments lässt sich feststellen, dass die harmonische Reihe aufgrund von Rundungsfehlern und begrenzter Genauigkeit der Fließkommazahlen auf Computern scheinbar konvergiert, obwohl sie theoretisch divergiert. Insbesondere zeigt sich, dass die Genauigkeit der Berechnung von der Wahl des Datentyps und des verwendeten Algorithmus abhängt. Während die Vorwärtssummentation mit dem Datentyp `np.float32` die geringste Genauigkeit aufweist, liefert die Rückwärtssummentation mit dem Datentyp `np.float64` das genaueste Ergebnis. Diese Erkenntnisse regen dazu an, weitere Optimierungen zu untersuchen, um möglicherweise eine Divergenz der harmonischen Reihe auf dem Computer nachzuweisen.

5 Python-Dokumentation tools_read_save

Das Modul `tools_read_save.py` dient der Modularisierung häufig verwendeter Funktionen zur Nutzerinteraktion. Das Programm implementiert drei Funktionen: `read_number` zum Einlesen von Nutzereingaben, `save_data` zum Abspeichern von Listen in `.csv`-Dateien, und `load_data` zum Einlesen von Daten aus `.csv`-Dateien.

5.1 Schnittstellendokumentation

`read_number`

Liest eine Zahl vom Benutzer ein, die innerhalb der angegebenen Grenzen liegt und in den gewünschten Datentyp konvertiert werden kann.

Parameter:

- `question` (str): Eingabeaufforderung an den Benutzer.
- `data_type` (type): Gewünschter Datentyp der Eingabe.
- `lower_limit` (float): Untere Grenze der gültigen Eingabe (Standardwert: $-\infty$).
- `upper_limit` (float): Obere Grenze der gültigen Eingabe (Standardwert: ∞).

Rückgabewert:

- Die eingegebene Zahl im gewünschten Datentyp.

Ausnahmen:

- `ValueError`: Wenn eine leere Eingabe gemacht wird.

Anwendung:

Mit dem Parameter `question` spezifizieren Sie die Nachricht, mit der der Nutzer im Terminal dazu aufgefordert wird, eine Eingabe zu machen. Sie sollte möglichst informativ sein, sodass der Nutzer daraus ableiten kann, welche Form der Eingabe (z.B. eine ganze Zahl) erwartet wird.

Die Funktion überprüft anschließend, ob die Eingabe in den mit `data_type` spezifizierte Datentyp umgewandelt werden kann, und ob die Eingabe innerhalb der Grenzen `lower_limit` und `upper_limit` liegt. Sind die Bedingungen nicht erfüllt, wird der Nutzer automatisch dazu aufgefordert, seine Eingabe zu korrigieren.

Im beigefügten Fließdiagramm (Abbildung 8) ist dargestellt, wie die Funktion sicherstellt, dass die Eingabe des Nutzers den gewünschten Anforderungen entspricht, bevor der eingelesene Wert zurückgegeben wird.

Beispiel:

Nachfolgend wird die Verwendung der Funktion im Experimentierskript zur Konvergenz der harmonischen Reihe (Abschnitt 2) gezeigt:

```
num = read_number(question = "Anzahl der zu berechnenden  
                             Partialsummen: ",  
                  data_type = int,  
                  lower_limit = 2)
```

Dabei soll eine ganze Zahl eingelesen werden (`data_type = int`), die größer gleich 2 ist (`lower_limit = 2`).

Dabei werden falsche Eingabeformate und Zahlen kleiner als 2 nicht akzeptiert:

```
Anzahl der zu berechnenden Partialsummen: abc  
Bitte geben Sie eine Zahl vom Typ <class 'int'> ein: 1  
Bitte geben Sie eine Zahl >= 2 ein: 2
```

Da die Zahl 2 den Anforderungen entspricht, wird sie `num` zugewiesen, sodass zum Beispiel der Aufruf `print(num)` die Zahl 2 ausgibt.

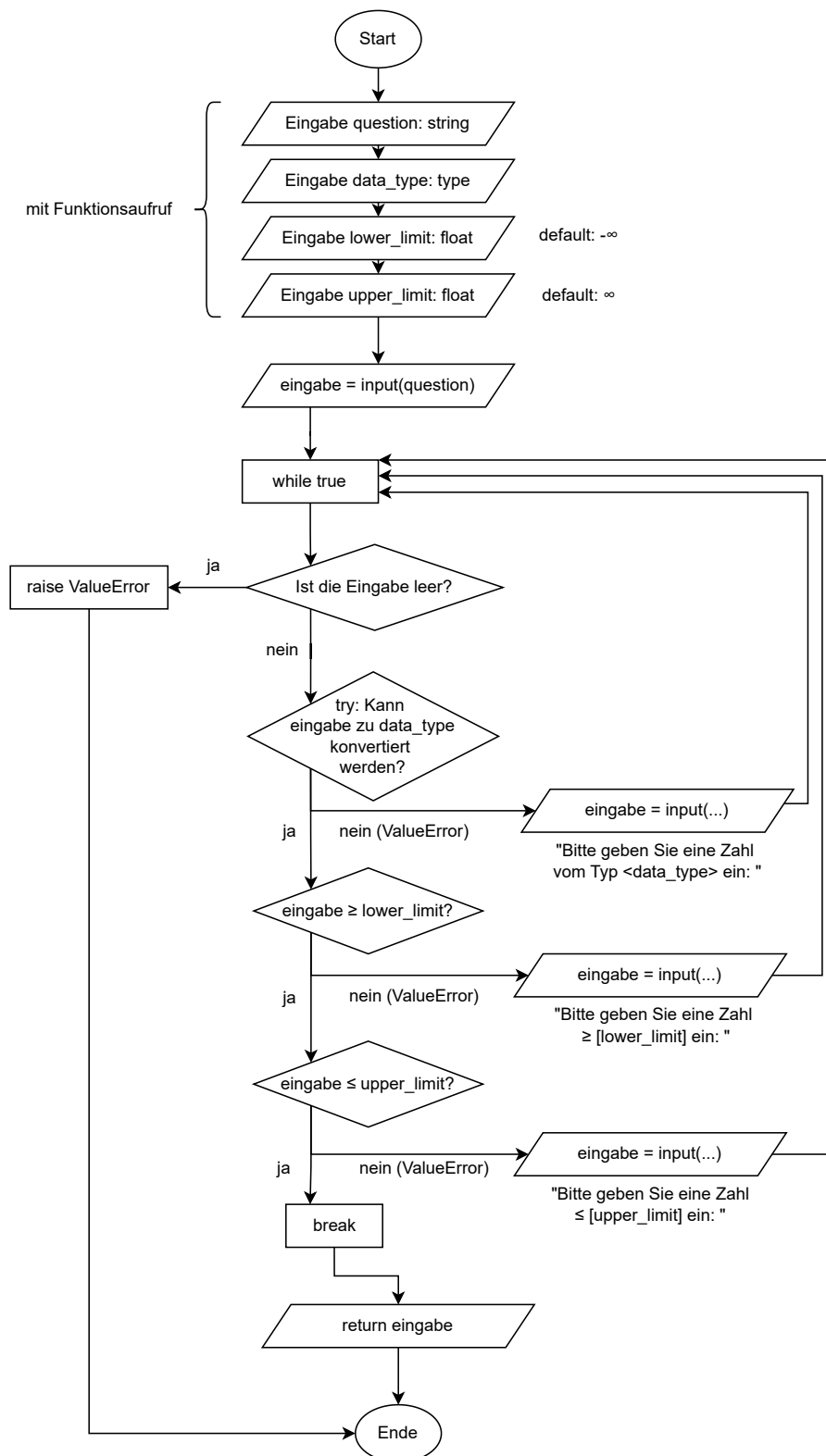


Abbildung 8: Fließdiagramm zum Ablauf von `read_number`

save_data

Speichert eine Liste von Zahlen in eine CSV-Datei.

Parameter:

- `data` (list): Eine Liste von Zahlen.
- `filepath` (str): Pfad der zu speichernden Datei.

Ausnahmen:

- `FileNotFoundError`: Wenn das Speichern fehlschlägt.

Beispiel:

Nachfolgend wird die Verwendung der Funktion im Experimentierskript zur Konvergenz der harmonischen Reihe (Abschnitt 2) gezeigt:

```
save_data(result_vorwaerts_float16 ,  
          "convergence_data_export/result_vorwaerts_float16.csv")
```

Bei dem übergebenen Objekt `result_vorwaerts_float16` handelt es sich um eine Liste, in der Partialsummen der harmonischen Reihe enthalten sind. Diese werden als `result_vorwaerts_float16.csv` im Ordner `convergence_data_export` im aktuellen Arbeitsverzeichnis gespeichert.

read_data

Liest eine Liste von Zahlen aus einer CSV-Datei ein.

Parameter:

- `filepath` (str): Pfad der zu lesenden Datei.

Rückgabewert:

- Eine Liste von Zahlen.

Ausnahmen:

- `FileNotFoundError`: Wenn das Einlesen fehlschlägt.

Beispiel:

Nachfolgend wird die Verwendung der Funktion im Experimentierskript zur Konvergenz der harmonischen Reihe (Abschnitt 2) gezeigt:

```
result_vorwaerts_float16 = load_data("convergence_data_import/  
                                     result_vorwaerts_float16.csv")
```

Die Funktion liest die Datei `result_vorwaerts_float16.csv` im Ordner `convergence_data_import` des aktuellen Arbeitsverzeichnisses ein. Darin sind spaltenweise Fließkommazahlen enthalten, die Partialsummen der harmonischen Reihe entsprechen. Zurückgegeben wird eine Liste.

5.2 Nutzungshinweise und Hauptprogramm

Das Abspeichern von Daten mittels `save_data` überschreibt bereits vorhandene Dateien mit dem selben Dateinamen. Stellen Sie deshalb sicher, dass Ergebnisse aus vorherigen exportierten Experimenten ordnungsgemäß in einem anderen Ordner abgelegt werden, bevor Sie `save_data` aufrufen.

In `tools_read_save.py` ist eine `main()` Funktion implementiert. Sie dient der Demonstration der enthaltenen Funktionen und wird nur ausgeführt, wenn `tools_read_save.py` direkt mittels `python3 tools_read_save.py` gestartet wird.

Dabei wird der Nutzer aufgefordert, eine ganze Zahl zwischen 3 und 7 in den Terminal einzugeben, um `read_number` zu testen. Eine leere Eingabe beendet das gesamte Testprogramm. Eine beispielhafte Nutzerinteraktion mit dem main-Programm könnte so aussehen:

```
Zunaechst wird die Funktion read_number() getestet.  
Eine leere Eingabe ermoeeglicht den Abbruch und  
fuehrt zum Test der naechsten Funktion.
```

```
Bitte geben Sie eine ganze Zahl x mit 3 <= x <= 7 ein. abc
```

Die Eingabe wird nicht akzeptiert, da `abc` keine Zahl ist:

```
Bitte geben Sie eine Zahl vom Typ <class 'int'> ein: 22
```

Die Eingabe wird nicht akzeptiert, da 22 größer als 7 ist:

```
Bitte geben Sie eine Zahl <= 7 ein: 1
```

Die Eingabe wird nicht akzeptiert, da 1 kleiner als 2 ist. Auch Fließkommazahlen werden nicht akzeptiert, da eine ganze Zahl gefordert wurde:

```
Bitte geben Sie eine Zahl >= 2 ein: 2,5
```

```
Bitte geben Sie eine Zahl vom Typ <class 'int'> ein: 2.5
```

```
Bitte geben Sie eine Zahl vom Typ <class 'int'> ein: 2
```

Die Zahl 2 entspricht den Anforderungen und wird vom Programm akzeptiert.

```
Die Funktion gibt zur ck: 2, Datentyp: <class 'int'>
```

Abschließend wird das Exportieren und Importieren von Listen anhand der Beispielliste [1.1117634, 2.55, 3.3, 144.0] demonstriert:

```
Nun wird die Beispielliste [1.1117634, 2.55, 3.3, 144.0]  
erstellt und mit save_data() exportiert.
```

```
test.csv erfolgreich gespeichert.
```

```
Dieselbe Liste wird nun mit load_data() wieder eingelesen und  
ausgegeben.
```

```
test.csv erfolgreich eingelesen.
```

```
[1.1117634, 2.55, 3.3, 144.0]
```

6 Literaturverzeichnis und Übersicht verwendeter Hilfsmittel

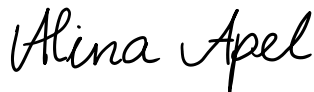
- NumPy Developers, *NumPy documentation* (2022). <https://numpy.org/doc/1.26/>
- *Harmonische Reihe*. Wikipedia [online] https://de.wikipedia.org/wiki/Harmonische_Reihe
- Scott Pakin, *The Comprehensive L^AT_EX Symbol List* (2024). <https://tug.ctan.org/info/symbols/comprehensive/symbols-a4.pdf>
- Hella Rabus, *Einführung in das Wissenschaftliche Rechnen, Skript zur Vorlesung* (2024). <https://moodle.hu-berlin.de/mod/folder/view.php?id=4331460>

Hiermit versichern wir, dass wir bei der Erstellung dieser Studienarbeit keinerlei IT-/KI-gestützte Schreibwerkzeuge verwendet haben.

7 Selbstständigkeitserklärung

Ich versichere, dass ich in dieser schriftlichen Studienarbeit alle von anderen Autor:innen wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren:innen eng anlehnenden Ausführungen meiner Arbeit besonders gekennzeichnet und die entsprechenden Quellen angegeben habe. Zusätzlich führe ich den Einsatz von IT-/KI-gestützten Schreibwerkzeugen zur Anfertigung dieser Arbeit im Abschnitt „Übersicht verwendeter Hilfsmittel“ im Anhang des eingereichten pdf-Dokumentes vollständig auf. Hierin habe ich die Verwendung solcher Tools vollständig durch Angabe ihres Produktnamens, meiner Bezugsquelle (z.B. URL) und Angaben zu genutzten Funktionen der Software sowie zum Nutzungsumfang dokumentiert. Bei der Erstellung dieser Studienarbeit habe ich durchgehend eigenständig und beim Einsatz IT-/KI-gestützter Schreibwerkzeuge steuernd gearbeitet.

Chantal Gerth, Berlin, 05.06.2024

A handwritten signature in black ink that reads "Alina Apel". The script is cursive and fluid.

Alina Apel, Berlin, 05.06.2024