Einführung in die Programmierung für Studierende der Physik

JOHANN WOLFGANG GOETHE UNIVERSITÄT
26.04.2022

MATTHIAS HANAUSKE

FRANKFURT INSTITUTE FOR ADVANCED STUDIES
JOHANN WOLFGANG GOETHE UNIVERSITÄT
INSTITUT FÜR THEORETISCHE PHYSIK
ARBEITSGRUPPE RELATIVISTISCHE ASTROPHYSIK
D-60438 FRANKFURT AM MAIN
GERMANY

3. Vorlesung

Plan für die heutige Vorlesung

- Kurze Wiederholung der Vorlesung 2
- C++ Anweisungen: Die for-, while- und do-Schleifen
- Anwendungsbeispiel: Folgen und Reihen
- Eine kleine Einführung in die Programmiersprache Python
- Die Computerarithmetik und der Fehler in numerischen Berechnungen
- Übungsaufgaben: Übungsblatt Nr.3

Kurze Wiederholung der Vorlesung 2

- Das erste C++ Programm (Hello World)
- Der numerische Zahlenraum eines C++ Programms
- Datentypen und Variablen
- Die Standardbibliothek <cmath>
- Arithmetik und Operatoren
- Die Ein- und Ausgabe
- Übungsaufgabe Nr.2

Vorlesung 3

C++ Anweisungen: Die for-, while- und do-Schleifen

Möchte man als Programmierer ein Problem mittels eines C++ Programms lösen, so muss man dem Computer in Form von Anweisungen sagen, was er zu erledigen hat. In diesem Unterpunkt behandeln wir eine der wichtigsten Anweisungsarten, die sogenannten *Schleifenanweisungen*. Im Prinzip ist jede Programmzeile, die mit einem Semikolon endet, eine Anweisung an den Computer, jedoch stellen die *Schleifenanweisungen* eine besondere Art von iterativen Anweisungsprozessen dar, und sind ein oft verwendetes Hilfsmittel der prozeduralen Programmierung. Eine Schleifenanweisung kann als eine **for**-, **while**- oder **do**-Anweisung ausgedrückt werden (näheres siehe C++ Anweisungen: Die for-, while- und do-Schleifen).

Anwendungsbeispiel: Folgen und Reihen

Die im vorigen Unterpunkt besprochenen *Schleifenanweisungen* finden in vielen C++ Programmen ihre Anwendung. In diesem Unterpunkt wird ihre Anwendung im Bereich der mathematischen Folgen und Reihen diskutiert. Am Beispiel der konvergenten Folge der Eulersche Zahl e und der Leibniz-Reihe zur Berechnung der Kreiszahl π wird die Verwendung der while-Schleife vorgestellt. Das Umschreiben der Programme unter Verwendung einer for-Schleife ist Teil der Aufgabe 2 des (siehe Übungsblattes Nr. 3. Es wird unter anderem das Konvergenzverhalten der Folge für die ersten Folgenglieder untersucht und die von vom Programm ausgegebenen Werte visualisiert. Hierzu werden die ausgegebenen Daten in eine separate Datei umgeleitet und dann mittels Gnuplot bzw. Python-Matplotlib dargestellt (näheres siehe Anwendungsbeispiel: Folgen und Reihen).

Eine kleine Einführung in die Programmiersprache Python

Die Programmiersprache Python ist auch eine sehr gute Objekt-orientierte Programmiersprache und im Prinzip hätten wir die gesamte Vorlesung nur mittels Python gestalten können. Die in dieser Vorlesung behandelten Python-Skripte und Python Jupyter Notebooks werden jedoch lediglich zur Visualisierung von Daten und im Bereich der Illustration von mathematisch/physikalischen Gleichungen benutzt. Mittels des Python-Moduls "matplotlib" (siehe Matplotlib: Visualization with Python) können auf einem einfachen Weg Bilder and Animationen des zuvor mit C++ simulierten Systems erzeugt werden. Zusätzlich werden wir, die im nächsten Unterpunkt besprochene C++ Computerarithmetik mittels eines Python Jupyter Notebooks verdeutlichen und dabei die Verwendung von Listen, Arrays und for-Schleifen in der Programmiersprache Python kennenlernen (näheres siehe Eine kleine Einführung in die Programmiersprache Python).

Die Computerarithmetik und der Fehler in numerischen Berechnungen

In diesem Unterpunkt werden wir zwei Klassen von Fehlerquellen in numerischen Berechnungen besprechen, den sogenannten *Rundungsfehler*, der aufgrund der Computerarithmetik entsteht und der sogenannte *Approximierungsfehler* (*Abschneidefehler* bzw.'*Truncation error*'), der immer dann auftritt, wenn der Programmierer eine exakte mathematische Gleichung mittels approximativer Ausdrücke annähert (näheres siehe

Vorlesung 3

In dieser Vorlesung werden wir die wohl wichtigste Form von C++ Anweisungen, die sogenannten *Schleifenanweisungen*, kennenlernen. Die *Schleifenanweisungen* stellen einen iterativen Anweisungsprozess dar und können in Form von **for**-, **while**- oder **do**-Anweisung ausgedrückt werden. Möchte man z.B. die natürlichen Zahlen von Null bis 100 im Terminal ausgeben lassen, so kann man dies in einer einfachen Weise mittels einer Schleifenanweisung dem Computer sagen. Die Anwendung von Schleifenanweisung wird danach am Beispiel einer mathematischen Folge und Reihe diskutiert.
Die Visualisierung von Daten, die mittels C++ Programmen erstellt wurden, ist ein wichtiges Teilgebiet eines Programmierers im Bereich der Physik. In dieser Vorlesung werden diverse Python-Skripte und Python Jupyter Notebooks vorgestellt, die zur Visualisierung von Daten und im Bereich der Illustration von mathematisch/physikalischen Gleichungen benutzt werden. Mittels des Python-Moduls "matplotlib" (siehe Matplotlib: Visualization with Python) können auf einem einfachen Weg Bilder and Animationen des zuvor mit C++ simulierten Systems erzeugt werden.

Am Ende der Vorlesung kommen wir nochmals auf den, bereits im Unterkapitel Datentypen und Variablen angesprochenen Zahlenraum des Computers ($\mathbb{R}_C$) zurück und diskutieren die zwei wichtigsten Fehlerquellen bei nummerischen Berechnungen (*Rundungsfehler* und *Approximierungsfehler*). Der *Rundungsfehler* ist darin begründet, dass der Zahlenraum des Computers ($\mathbb{R}_C$), nur eine relativ kleine Teilmenge der reellen Zahlen benutzt und somit reellwertige Zahlen mit einer unendlichen Anzahl von Nachkommastellen nicht speichern kann. Diese Teilmenge $\mathbb{R}_C$ umfasst nur rationale Zahlen und speichert den gebrochenen Teil, der mit *Mantisse* bezeichnet wird, zusammen mit dem exponentiellen Teil, welchen man *Charakteristik* nennt als eine binäre Liste von Nullen und Einsen. Die zweite Klasse von nummerischen Fehlern, die *Approximierungsfehler* werden am Beispiel der approximativen Annäherung an die Kreiszahl π mittels der alternierenden Leibniz-Reihe besprochen. Bei dieser Art von Fehlern hat es der Programmierer selbst in der Hand, wie genau er in seinem Programm die Berechnung durchführen möchte (näheres siehe Die Computerarithmetik und der Fehler in numerischen Berechnungen).

C++ Anweisungen: Die for-, while- und do-Schleifen

Die Programmiersprache C++ bietet einen konventionellen und flexiblen Satz von Anweisungen. Im Prinzip ist jede Programmzeile, die mit einem Semikolon endet, eine Anweisung. In diesem Unterpunkt werden wir die sogenannten *Schleifenanweisungen* behandeln und in der nächsten Vorlesung die *Auswahlanweisungen* vorstellen (siehe C++ Anweisungen: Auswahlanweisungen mit if und switch). Um die Sinnhaftigkeit von Schleifenanweisungen zu verdeutlichen, betrachten wir die (leicht abgeänderte) Programmierungsaufgabe 3 aus dem vorigen Übungsblatt (siehe Übungsblatt Nr.2, Aufgabe 3). Die Aufgabe soll jedoch nun sein, die Zahlenwerte der Folge $(a_n)_{n\in\mathbb{N}}$ mit $a_n := n^2$ für $n \in [0, 1, 2, .., 10]$ in einem C++ Programm auszugeben. Natürlich könnte man die Inkrementierung der Variable n und die cout-Ausgabe noch sieben zusätzliche Male am Ende der main()-Funktion hinzufügen und hätte die Programmieraufgabe auch gelöst. Was würde man jedoch machen, wenn $n \in [0, 1, 2, .., N]$ mit $N = 10000$ wäre, oder die natürliche Zahl N vom Benutzer des ausführbaren Programms selbst eingegeben werden sollte? Für solche und viele weitere, auf iterativen Anweisungen basierenden Problemen, sind Schleifen hilfreich und oft sogar notwendig.

Eine Schleife kann als eine **for**-, **while**- oder **do**-Anweisung ausgedrückt werden und die folgenden Programme zeigen die Verwendung dieser drei Schleifenanweisungen an unserem einfachen Beispiel, wobei die rechte untere Abbildung die Terminalausgabe nach dem Ausführen der Programme zeigt, die bei allen das gleiche Ergebnis liefert:

For_0.cpp

```cpp
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N */
#include <iostream>                 // Ein- und Ausgabebibliothek

int main(){                         // Hauptfunktion
    unsigned int n;                 // Deklaration des Folgenindex n
    const unsigned int N = 10;      // Definition des maximal ausgegebenen Folgenindex

    printf("%5s %5s \n","n","a_n"); // Ausgabe Beschreibung

    for(n=0; n<=N; ++n){            // Schleifen Anfang
        printf("%5i %5i \n",n,n*n); // Ausgabe des Folgenindex n und des Wertes des Folgengliedes
    }                               // Ende der Schleife
}
```

While_0.cpp

```cpp
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N */
#include <iostream>                 // Ein- und Ausgabebibliothek

int main(){                         // Hauptfunktion
    unsigned int n = 0;             // Definition des Folgenindex n und gleichzeitige Initialisierung
    const unsigned int N = 10;      // Definition des maximal ausgegebenen Folgenindex

    printf("%5s %5s \n","n","a_n"); // Ausgabe Beschreibung

    while( n <= N ){                // Schleifen Anfang
        printf("%5i %5i \n",n,n*n); // Ausgabe des Folgenindex n und des Wertes des Folgengliedes
        ++n;                        // Folgenindex wird um eins erhöht
    }                               // Abruchbedingung der Schleife
}
```

Do_0.cpp

```cpp
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N */
#include <iostream>                 // Ein- und Ausgabebibliothek
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$ g++ For_0.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$ ./a.out
    n   a_n
    0     0
```

Die for-Schleife

Die for-Schleife

Die im Programm For_0.cpp verwendete **for**-Schleife (**for** (n=0; n<=N; ++n){ ... }) basiert auf der allgemeinen traditionellen Schreibweise einer **for**-Schleife, die die folgende Struktur besitzt:

for ('Initialisierungsteil'; 'Anweisungsbedingung'; 'Anweisung am Ende eines jeden Schleifendurchlaufs') { 'Block von Anweisungen' }

```
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N */
#include <iostream>                    // Ein- und Ausgabebibliothek

int main(){                            // Hauptfunktion
    unsigned int n;                    // Deklaration des Folgenindex n
    const unsigned int N = 10;         // Definition des maximal ausgegebenen Folgenindex

    printf("%5s %5s \n","n","a_n"); // Ausgabe Beschreibung

    for(n=0; n<=N; ++n){               // Schleifen Anfang
        printf("%5i %5i \n",n,n*n); // Ausgabe des Folgenindex n und des Wertes des Folgengliedes
    }                                  // Ende der Schleife
}
```

```
(base) hanauske@hanauske
(base) hanauske@hanauske
    n   a_n
    0     0
    1     1
    2     4
    3     9
    4    16
    5    25
    6    36
    7    49
    8    64
    9    81
   10   100
(base) hanauske@hanauske
```

Die while-Schleife

Die while-Schleife

Die im Programm While_0.cpp verwendete **while**-Schleife (**while** (n <= N){ ... }) basiert auf der allgemeinen Schreibweise einer **while**-Schleife, die die folgende Struktur besitzt:

while ('Anweisungsbedingung der Schleife') { 'Block von Anweisungen' }

```
(base) hanauske@hana
(base) hanauske@hana
    n    a_n
    0      0
    1      1
    2      4
    3      9
    4     16
    5     25
    6     36
    7     49
    8     64
    9     81
   10    100
(base) hanauske@hana
```

```
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N */
#include <iostream>                 // Ein- und Ausgabebibliothek

int main(){                         // Hauptfunktion
    unsigned int n = 0;             // Definition des Folgenindex n und gleichzeitige Initialisierung mit 0
    const unsigned int N = 10;      // Definition des maximal ausgegebenen Folgenindex

    printf("%5s %5s \n","n","a_n"); // Ausgabe Beschreibung

    while( n <= N ){                // Schleifen Anfang
        printf("%5i %5i \n",n,n*n); // Ausgabe des Folgenindex n und des Wertes des Folgengliedes
        ++n;                        // Folgenindex wird um eins erhöht
    }                               // Abruchbedingung der Schleife
}
```

Die do-Schleife

Die do-Schleife

Die im Programm Do_0.cpp verwendete **do**-Schleife (**do** { ... } **while**(n <= N);) basiert auf der allgemeinen Schreibweise einer **do**-Schleife, die die folgende Struktur besitzt:

do { 'Block von Anweisungen' } **while** ('Anweisungsbedingung der Schleife');

```
(base) hanauske@hanau
(base) hanauske@hanau
    n   a_n
    0     0
    1     1
    2     4
    3     9
    4    16
    5    25
    6    36
    7    49
    8    64
    9    81
   10   100
(base) hanauske@hanau
```

```
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N */
#include <iostream>                    // Ein- und Ausgabebibliothek

int main(){                            // Hauptfunktion
    unsigned int n = 0;                // Definition des Folgenindex n und gleichzeitige Initialisierung mit 0
    const unsigned int N = 10;         // Definition des maximal ausgegebenen Folgenindex

    printf("%5s %5s \n","n","a_n"); // Ausgabe Beschreibung

    do{                                // Schleifen Anfang
        printf("%5i %5i \n",n,n*n); // Ausgabe des Folgenindex n und des Wertes des Folgengliedes
        ++n;                           // Folgenindex wird um eins erhöht
    } while( n <= N );                 // Abruchbedingung der Schleife
}
```

C++ Schleifen-Anweisungen

For_0.cpp

```cpp
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N */
#include <iostream>                 // Ein- und Ausgabebibliothek

int main(){                         // Hauptfunktion
    unsigned int n;                 // Deklaration des Folgenindex n
    const unsigned int N = 10;      // Definition des maximal ausgegebenen Folgenindex

    printf("%5s %5s \n","n","a_n"); // Ausgabe Beschreibung

    for(n=0; n<=N; ++n){            // Schleifen Anfang
        printf("%5i %5i \n",n,n*n); // Ausgabe des Folgenindex n und des Wertes des Folgengliedes
    }                               // Ende der Schleife
}
```

While_0.cpp

```cpp
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N */
#include <iostream>                 // Ein- und Ausgabebibliothek

int main(){                         // Hauptfunktion
    unsigned int n = 0;             // Definition des Folgenindex n und gleichzeitige Initialisierung
    const unsigned int N = 10;      // Definition des maximal ausgegebenen Folgenindex

    printf("%5s %5s \n","n","a_n"); // Ausgabe Beschreibung

    while( n <= N ){                // Schleifen Anfang
        printf("%5i %5i \n",n,n*n); // Ausgabe des Folgenindex n und des Wertes des Folgengliedes
        ++n;                        // Folgenindex wird um eins erhöht
    }                               // Abruchbedingung der Schleife
}
```

Do_0.cpp

```cpp
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N */
#include <iostream>                 // Ein- und Ausgabebibliothek

int main(){                         // Hauptfunktion
    unsigned int n = 0;             // Definition des Folgenindex n und gleichzeitige Initialisierung
    const unsigned int N = 10;      // Definition des maximal ausgegebenen Folgenindex

    printf("%5s %5s \n","n","a_n"); // Ausgabe Beschreibung

    do{                             // Schleifen Anfang
        printf("%5i %5i \n",n,n*n); // Ausgabe des Folgenindex n und des Wertes des Folgengliedes
        ++n;                        // Folgenindex wird um eins erhöht
    } while( n <= N );              // Abruchbedingung der Schleife
}
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$ g++ For_0.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$ ./a.out
    n   a_n
    0     0
    1     1
    2     4
    3     9
    4    16
    5    25
    6    36
    7    49
    8    64
    9    81
   10   100
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$
```

C++ Schleifen-Anweisungen

Schleifenanweisungen finden ihre Verwendung in den unterschiedlichsten Anwendungsbereichen und wir werden im nächsten Unterpunkt die Anwendung auf mathematische Folgen vertiefen und die Anwendung auf mathematisch definierte Reihen vorstellen (siehe Unterkapitel Anwendungsbeispiel: Folgen und Reihen).

Verwendung von for-, while- und do-Schleifen

while-Schleifen werden oft benutzt, wenn es keine offensichtliche Schleifenvariable (in unserem Beispiel die Variable 'n') gibt, oder wenn die Aktualisierung der Schleifenvariable nicht am Ende des 'Anweisungsblockes' gemacht wird. In unserem Beispiel ist dies jedoch nicht der Fall und somit ist eine Formulierung mittels einer **for**-Schleife geeigneter, da man direkter den iterativen Ablauf der Schleife sieht.

```
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N */
#include <iostream>                  // Ein- und Ausgabebibliothek

int main(){                          // Hauptfunktion
    unsigned int n = 0;              // Definition des Folgenindex n und gleichzeitige Initialisierung mit 0
    const unsigned int N = 10;       // Definition des maximal ausgegebenen Folgenindex

    printf("%5s %5s \n","n","a_n"); // Ausgabe Beschreibung

    while( n <= N ){                 // Schleifen Anfang
        printf("%5i %5i \n",n,n*n); // Ausgabe des Folgenindex n und des Wertes des Folgengliedes
        ++n;                         // Folgenindex wird um eins erhöht
    }                                // Abruchbedingung der Schleife
}
```

C++ Schleifen-Anweisungen

Es sei hierbei noch erwähnt, dass es mehrere unterschiedliche Schreibweisen von **for**-Schleifen in C++ gibt, die sich von der vorgestellten traditionellen Schreibweise unterscheiden. So kann man z.B. mittels der sogenannten *Bereichsbasierten for-Anweisungsbedingungen* über eine zuvor definierte Zahlenmenge die Schleife laufen lassen (in unserem Beispiel z.B. '**for** (**auto** n : n_array){ … }'), wobei die Zahlenmenge über die Schleife laufen soll in dem Zahlenarray 'n_array' zuvor mittels 'unsigned int n_array[N+1] = {0,1,2,3,4,5,6,7,8,9,10};' definiert werden muss (näheres zur Definition von Arrays unterschiedlicher Datentypen in der Vorlesung Nr.5).

Bereichsbasierten for-Anweisungsbedingungen

For_0b.cpp

```
/* Berechnung und Ausgabe der Folgenglieder
 * der Folge a_n=n^2 bis n=N
 * mittels einer Bereichsbasierten for-Anweisungsbedingung */
#include <iostream>                 // Ein- und Ausgabebibliothek

int main(){                         // Hauptfunktion
    const unsigned int N = 10;      // Definition des maximal ausgegebenen Folgenindex
    unsigned int n_array[N+1] = {0,1,2,3,4,5,6,7,8,9,10}; // Array von Zahlenwerten

    printf("%5s %5s \n","n","a_n"); // Ausgabe Beschreibung

    for(auto n : n_array){          // Schleifen Anfang
        printf("%5i %5i \n",n,n*n); // Ausgabe des Folgenindex n und des Wertes des Folgengliedes
    }                               // Ende der Schleife
}
```

Folgen und Reihen
als C++ Schleifen-Anweisungen

Anwendungsbeispiel: Folgen und Reihen

In diesem Unterpunkt sollen einige Anwendungsbeispiele der for-, while- und do-Schleifen (siehe C++ Anweisungen: Die for-, while- und do-Schleifen) diskutiert werden. Es wird im Speziellen die konvergente Folge der Eulersche Zahl e und die Leibniz-Reihe zur Berechnung der Kreiszahl π vorgestellt. Obwohl es bei der nummerischen Berechnung von Folgen und Reihen besser ist, eine **for**-Schleife zu verwenden (offensichtliche Schleifenvariable, Aktualisierung der Schleifenvariable am Ende des 'Anweisungsblockes'), wird in den folgenden Anwendungsbeispielen stets eine **while**-Schleife verwendet. Das Umschreiben der Programme unter Verwendung einer **for**-Schleife ist den Studierenden überlassen (siehe Übungsblatt Nr. 3 , Aufgabe 2).

Anwendung: Mathematische Folgen

Im Übungsblatt Nr. 2 in der Aufgabe 1 wurde bereits die Folge $(a_n)_{n\in\mathbb{N}}$ mit $a_n := \left(1+\frac{1}{n}\right)^n$ vorgestellt, die als Grenzwert die Eulersche Zahl e hat:

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

Wir sind nun zunächst nicht an dem genauen Wert dieses Grenzwertes interessiert, sondern möchten das Konvergenzverhalten der Folge a_n studieren und uns die ersten N Folgenglieder mittels eines C++ Programms ausgeben lassen. Das folgende Programm gibt z.B. die ersten 20 ($N = 20$) Folgenglieder auf 15 Nachkommastellen genau aus

While_4.cpp

```
/* Die Eulerschen Zahl e wird mittels ihrer Folgendefinition approximiert
 * Ausgabe zum Plotten (Gnuplot oder Python) mittels: "./a.out > While_4.dat" */
#include <iostream>                                          // Ein- und Ausgabebibliothek
```

Folgen

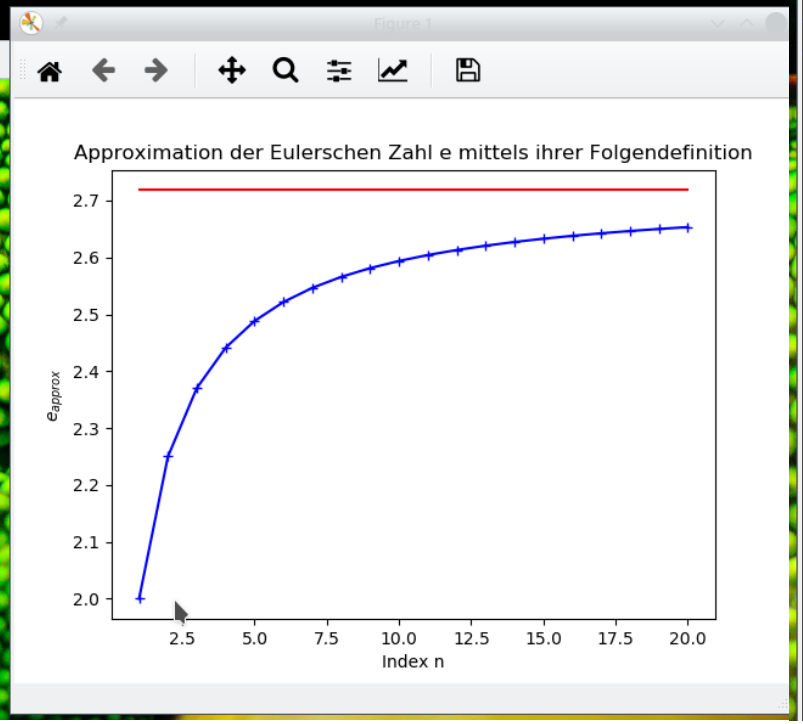
als C++ Schleifenanweisungen

Im Folgenden wird das Konvergenzverhalten einer Folge mittels einer while-Schleife in einem C++ Programm implementiert.

Die Eulersche Zahl e kann mittels des folgenden Grenzwertes der Folge $(a_n)_{n\in\mathbb{N}}$ mit $a_n := \left(1+\frac{1}{n}\right)^n$ definiert werden:

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$ python3 PythonPlot_While_4.py
```




```
(base) hanauske@hanauske-Aspire-A717-72G:
# 0: Folgenindex n
# 1: Approximierter Wert von e
  1    2.000000000000000
  2    2.250000000000000
  3    2.370370370370370
  4    2.441406250000000
  5    2.488319999999999
  6    2.521626371742113
  7    2.546499697040712
  8    2.565784513950348
  9    2.581174791713198
 10    2.593742460100002
 11    2.604199011897529
 12    2.613035290224676
 13    2.620600887885731
 14    2.627151556300868
 15    2.632878717727919
 16    2.637928497366600
 17    2.642414375183110
 18    2.646425821097687
 19    2.650034326640442
 20    2.653297705144422
(base) hanauske@hanauske-Aspire-A717-72G:
```

While_4.cpp

```cpp
/* Die Eulerschen Zahl e wird mittels ihrer Folgendefinition approximiert
 * Ausgabe zum Plotten (Gnuplot oder Python) mittels: "./a.out > While_4.dat" */
#include <iostream>                                          // Ein- und Ausgabebibliothek
#include <cmath>                                             // Bibliothek für mathematisches (e-Funktion, Betrag, ...)

int main(){                                                  // Hauptfunktion
    unsigned int n = 1;                                      // Deklaration und Initialisierung des Folgenindex als naturliche Zahl
    const unsigned int N=20;                                 // Deklaration und Initialisierung des maximalen Folgengliedes
    double e_Approx;                                         // Deklaration der approximierten Gleitkommazahl von e

    printf("# 0: Folgenindex n \n# 1: Approximierter Wert von e \n"); // Beschreibung der ausgegebenen Groessen

    while( n <= N ){                                         // Schleife zur Berechnung von e mit Abbruchbedingung
        e_Approx = pow(1 + 1.0/n,n);                         // Mathematische Folgendefinition fuer die Eulerschen Zahl e
        printf("%3i %20.15f \n",n, e_Approx);                // Ausgabe des approximierten Wertes von e
        n++;                                                 // Index der Folge wird um eins erhöht (entspricht n=n+1)
    }                                                        // Ende der Schleife
}
```

Folgen

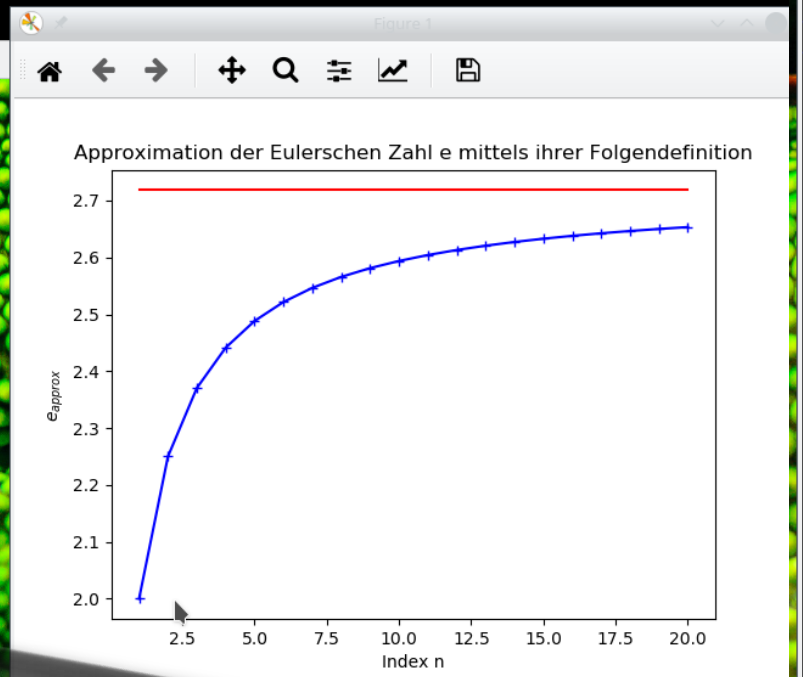
als C++ Schleifenanweisungen

Im Folgenden wird das Konvergenzverhalten einer Folge mittels einer while-Schleife in einem C++ Programm implementiert.

Die Eulersche Zahl e kann mittels des folgenden Grenzwertes der Folge $(a_n)_{n\in\mathbb{N}}$ mit $a_n := \left(1+\frac{1}{n}\right)^n$ definiert werden:

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$ python3 PythonPlot_While_4.py
```




```
(base) hanauske@hanauske-Aspire-A717-72G:
# 0: Folgenindex n
# 1: Approximierter Wert von e
  1    2.000000000000000
  2    2.250000000000000
  3    2.370370370370370
  4    2.441406250000000
  5    2.488319999999999
  6    2.521626371742113
  7    2.546499697040712
  8    2.565784513950348
  9    2.581174791713198
 10    2.593742460100002
 11    2.604199011897529
 12    2.613035290224676
 13    2.620600887885731
 14    2.627151556300868
 15    2.632878717727919
 16    2.637928497366600
 17    2.642414375183110
 18    2.646425821097687
 19    2.650034326640442
 20    2.653297705144422
(base) hanauske@hanauske-Aspire-A717-72G:
```

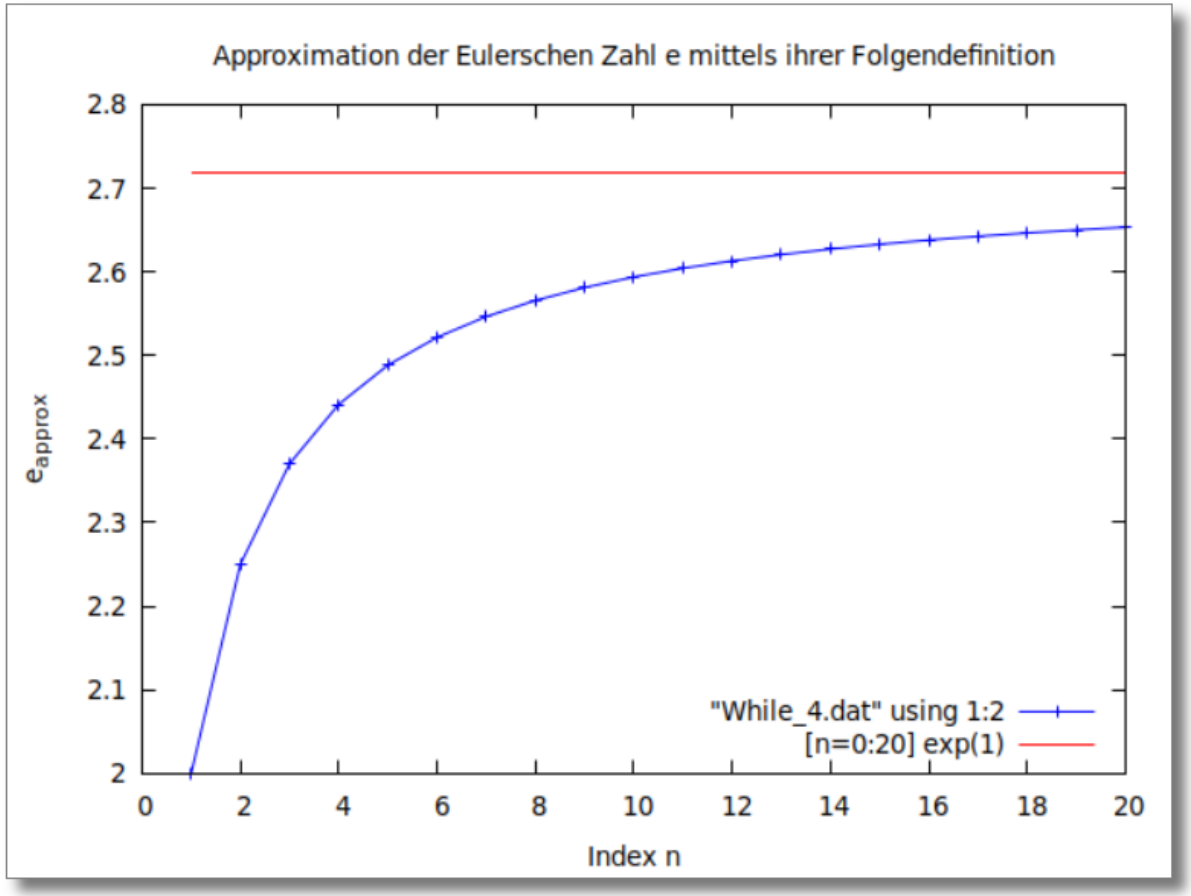
*Bei nummerischen Berechnung von Folgen(Reihen) ist es jedoch besser eine **for**-Schleife, anstatt einer **while**-Schleife zu verwenden, da der Folgenindex(Summenindex) eine offensichtliche Schleifenvariable darstellt und die Aktualisierung der Schleifenvariable am Ende des 'Anweisungsblockes' stattfindet -> Siehe Übungsaufgabe 2*

While_4.cpp

```cpp
/* Die Eulerschen Zahl e wird mittels ihrer Folgendefinition approximiert
 * Ausgabe zum Plotten (Gnuplot oder Python) ./a.out > "1.dat" */
#include <iostream>                                   // Standard Ausgabebibliothek
#include <cmath>                                      // Bibliothek mathematisches (z.B. Betrag, ...)

int main(){                                           // Hauptfunktion
    unsigned int n = 1;                               // Deklaration und Initialisierung des Index als natürli
    const unsigned int N=20;                          // Deklaration und Initialisierung des maximalen Index des
    double e_Approx;                                  // Deklaration der approximierten Kommazahl von

    printf("# 0: Folgenindex n \n# 1: Approximierter Wert von e \n"); // Beschreibung der ausgegebenen Groessen

    while( n <= N ){                                  // Schleife zur Berechnung von e mit Abbruchbedingung
        e_Approx = pow(1 + 1.0/n,n);                  // Mathematische Folgendefinition fuer die Eulerschen Zahl e
        printf("%3i %20.15f \n",n, e_Approx);         // Ausgabe des approximierten Wertes von e
        n++;                                          // Index der Folge wird um eins erhöht (entspricht n=n+1)
    }                                                 // Ende der Schleife
}
```

Visualisierung der berechneten Daten

mittels eines Gnuplot-Shellskripts



Gnuplot-Shellskript: GnuPlot_While_4.sh

```
# Shell Script zum Plotten der berechneten Daten von (While_4.cpp) mittels Gnuplot
FILE=While_4.dat
echo "Starte Plotten mit Gnuplot"
gnuplot -persist << PLOT
reset
set terminal pngcairo size 600,450 enhanced font 'Verdana,10'
##############################################
set title "Approximation der Eulerschen Zahl e mittels ihrer Folgendefinition"
set ylabel "e_{approx}"
set xlabel "Index n"
set key right bottom
set output sprintf('GnuPlot_While_4.png')
plot "$FILE" using 1:2 with linespoints lt rgb "blue", [n=0:20] exp(1) with lines lt rgb "red"
##############################################
quit
PLOT
echo "Fertig"
```

Im abgebildeten Gnuplot-Shellskript wird zunächst die Datei definiert, in der sich die zu plottenden Daten befinden (FILE=While_4.dat) und danach wird die eine Terminalausgabe mittels **echo** "Starte Plotten mit Gnuplot" gemacht. Es sei hier noch erwähnt, dass sowohl in Gnuplot als auch in Python, die Zeilen welche mit dem Rautesymbol '#' beginnen ignoriert werden und somit wie Komentarzeilen fungieren. Diese Eigenschaft wurde bei der 'Beschreibung der ausgegebenen Größen' im C++ Programm verwendet (printf("# 0: Folgenindex n \n# 1: Approximierter Wert von e \n");). Das Plot-Programm 'Gnuplot' startet man im Terminal mit dem Befehl 'gnuplot'. Es werden dann mit den Befehlen 'set ...' mehrere Plot-Eigenschaften spezifiziert (z.B. der Titel des Diagramms: set title "...", die Bezeichnung der x- und y-Achsen, die Position der Plotlegende: set key,

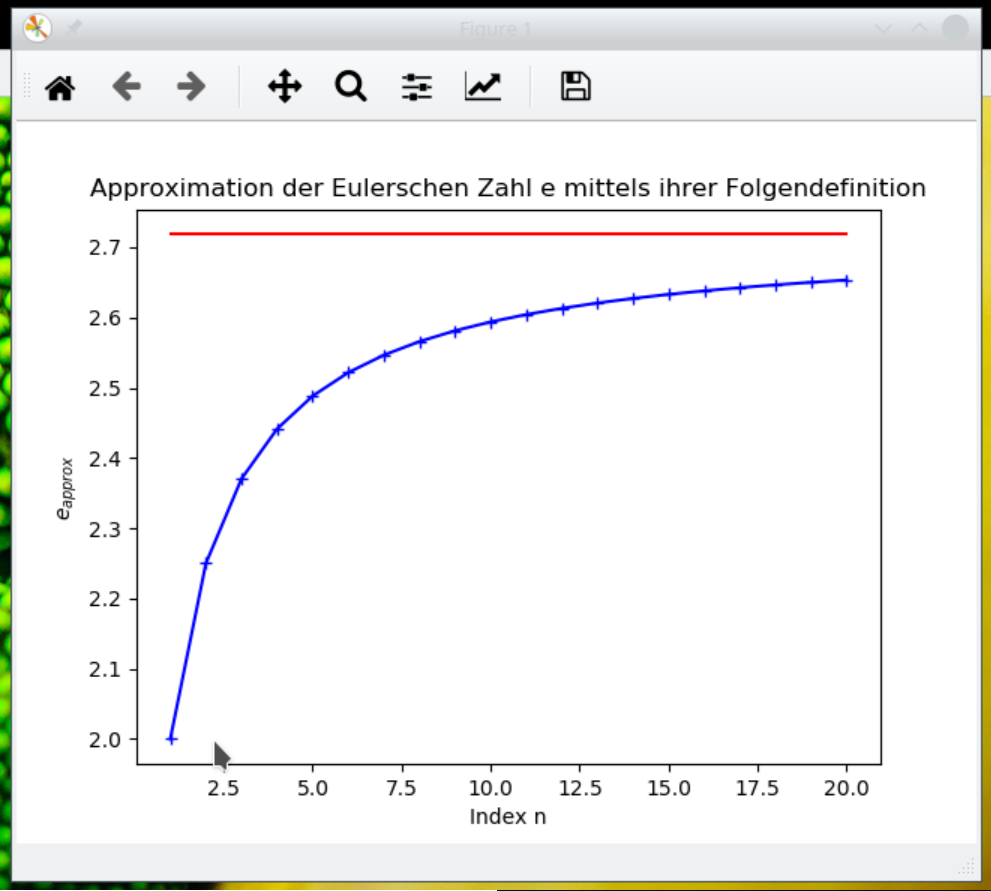
```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$ ./a.out > While_4.dat
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$ ./GnuPlot_While_4.sh
Starte Plotten mit Gnuplot
Fertig
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$
```

Visualisierung der berechneten Daten

mittels eines Python Programms (Python-Skript)

In dem interaktiven Diagrammfenster (rechte Abbildung) kann der Benutzer z.B., indem er zuvor auf das Symbol mit der Lupe klickt, gewisse Teilbereiche der Abbildung vergrößert darstellen. Um die Linux-Shell wieder benutzen zu können, muss man zuvor das interaktive Fenster schließen. In der unteren Abbildung ist das entsprechende Python-Skript dargestellt. Dieses wird in dem nächsten Unterpunkt der Vorlesung besprochen (siehe Eine kleine Einführung in die Programmiersprache Python).

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$ python3 PythonPlot_While_4.py
```

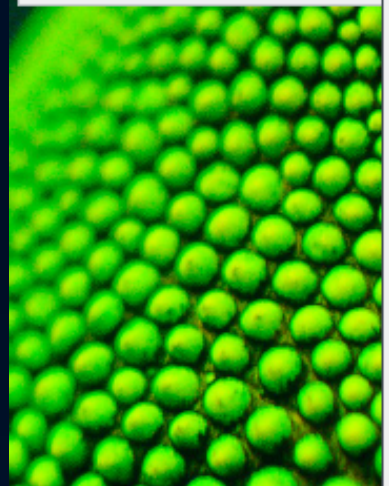


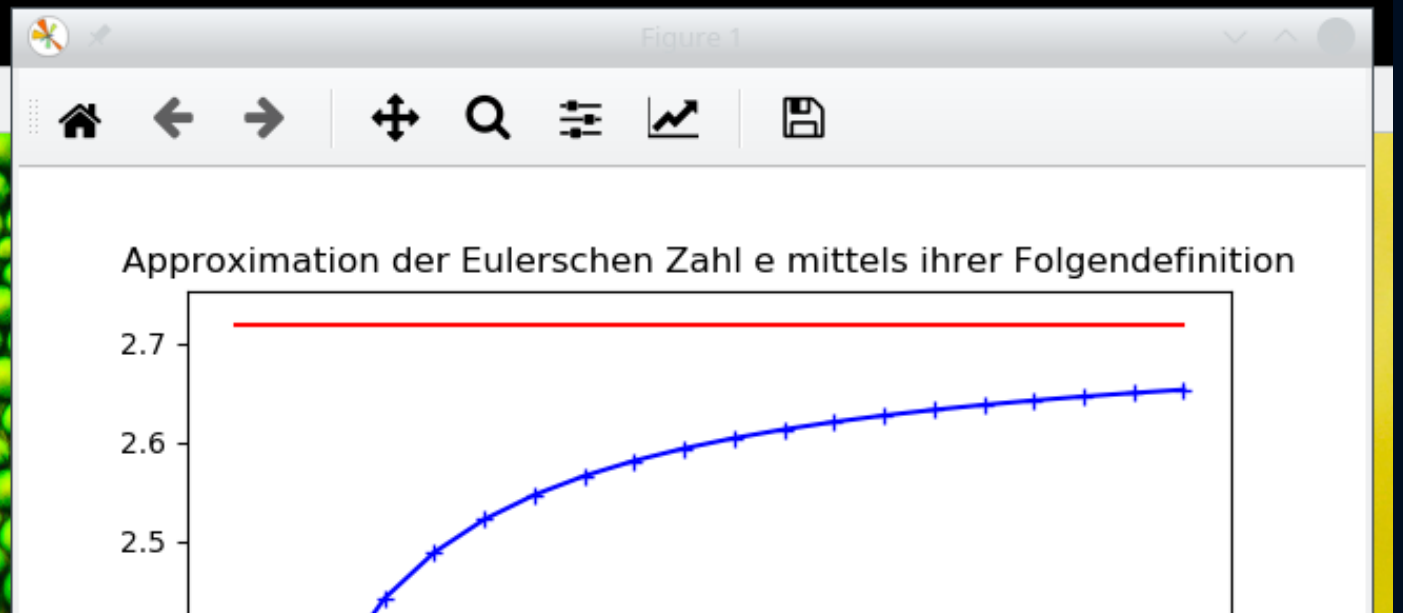
Python-Skript: PythonPlot_While_4.py

```python
# Python Programm zum Plotten der berechneten Daten von (While_4.cpp)
import matplotlib.pyplot as plt                          # Python Bibliothek zu
import numpy as np                                       # Python Bibliothek fu

data = np.genfromtxt("./While_4.dat")                    # Einlesen der berechn

plt.title(r'Approximation der Eulerschen Zahl e mittels ihrer Folgendefinition')
plt.ylabel(r'$e_{approx}$')                              # Beschriftung y-Achse
plt.xlabel('Index n')                                    # Beschriftung x-Achse
plt.plot(data[:,0],data[:,1], marker='+', color="blue")  # Plotten der Daten
plt.plot([data[0,0],data[-1,0]],[np.e,np.e], color="red") # Horizontale Linie, die den exakten Wert von e markiert

plt.savefig("PythonPlot_While_4.png", bbox_inches="tight") # Speichern der Abbildung als Bild
plt.show()                                               # Zusaetzliches Darstellen der Abbildung in einem separaten Fenster
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$ python3 PythonPlot_While_4.py
```

V2 : python3



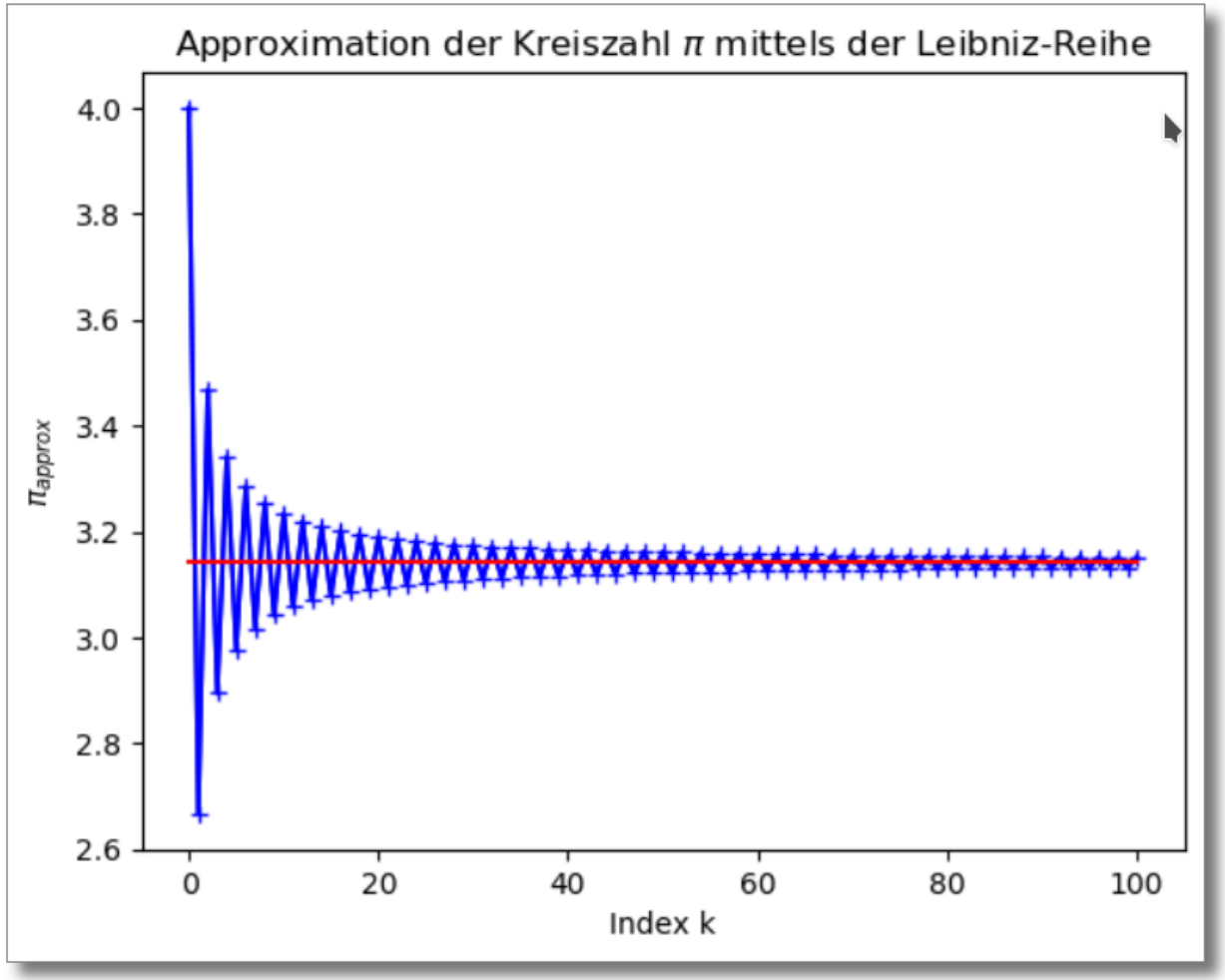


Python-Skript: PythonPlot_While_4.py

```python
# Python Programm zum Plotten der berechneten Daten von (While_4.cpp)
import matplotlib.pyplot as plt                          # Python Bibliothek zum Plotten (siehe https://matplotlib.org/ )
import numpy as np                                       # Python Bibliothek fuer Mathematisches (siehe https://numpy.org/ )

data = np.genfromtxt("./While_4.dat")                    # Einlesen der berechneten Daten von While_4.cpp

plt.title(r'Approximation der Eulerschen Zahl e mittels ihrer Folgendefinition') # Titel der Abbildung
plt.ylabel(r'$e_{approx}$')                              # Beschriftung y-Achse
plt.xlabel('Index n')                                    # Beschriftung x-Achse
plt.plot(data[:,0],data[:,1], marker='+', color="blue")  # Plotten der Daten
plt.plot([data[0,0],data[-1,0]],[np.e,np.e], color="red") # Horizontale Linie, die den exakten Wert von e markiert

plt.savefig("PythonPlot_While_4.png", bbox_inches="tight") # Speichern der Abbildung als Bild
plt.show()                                               # Zusaetzliches Darstellen der Abbildung in einem separaten Fenster
```

Mathematische Reihen

als C++ Schleifenanweisungen




```
  94    3.152118677831945
  95    3.131176269454982
  96    3.151901658056018
  97    3.131388837543198
  98    3.151693406071117
  99    3.131592903558554
 100    3.151493401070991
# Approximierter Wert von Pi:        3.151493401070991
# Wirklicher Wert von Pi:            3.141592653589793
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$
```

Die Terminalausgabe der **while**-Schleife endet bei $k = 100$ und die letzten beiden Zeilen stellen die beiden printf(...)-Befehle dar, die sich außerhalb der **while**-Schleife befinden. Es wird hier der letzte approximierte Zahlenwert von π ($\pi_{approx} = 4\cdot$ 'Pi_Approx') mit dem wirklichen Wert von π verglichen, wobei der "wirkliche Wert" durch die in <cmath> vordefinierte long double Zahl 'M_PIl' gegeben ist. Man erkennt hierbei, dass der mit $N = 100$ erhaltene approximierte Wert von π lediglich auf drei Nachkommastellen genau ist. Dies liegt unter anderem an dem alternierenden Verhalten der Leibniz-Reihe, welches man gut in der linken Abbildung erkennt. Die Abbildung stellt die Zahlenwerte der printf(...)-Befehle des inneren Teils der **while**-Schleife dar und wurde mithilfe des unten abgebildeten Python-Skripts erzeugt, wobei zuvor die ausgegebenen Daten wieder mit dem Trick '**./a.out > While_2.dat**' in eine Datei 'While_2.dat' umgeleitet wurden. Das Python-Skript: PythonPlot_While_2.py ist dem Skript Python-Skript: PythonPlot_While_4.py von seiner Struktur her sehr ähnlich und es wurden lediglich die eingelesene Datei, der Titel, die Beschreibungen der x- und y-Achsen, die Position der roten horizontalen Linie und der Name der zu speichernden Bilddatei verändert.

Reihen

als C++ Schleifenanweisungen

Im Folgenden wird das Konvergenzverhalten einer Reihe mittels einer while-Schleife in einem C++ Programm implementiert.

Anwendung: Mathematische Reihen

Wir betrachten im Folgenden das mathematische Problem der Berechnung der Kreiszahl $\pi = 3.141592653589793...$ und verwenden dafür die im Jahre 1682 von Gottfried Wilhelm Leibniz vorgestellte iterative Annäherung. Die sogenannte alternierende Leibniz-Reihe konvergiert im Limes $\lim\limits_{N\to\infty}$ zu dem Wert der irrationalen Zahl $\frac{\pi}{4}$:

$$\sum_{k=0}^{N} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \ldots + \frac{(-1)^N}{2N+1}, \quad \text{mit:} \quad \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{\pi}{4}$$

While_2.cpp

```cpp
/* Mittels der Leibniz-Reihe
 * (Gottfried Wilhelm Leibniz, Zeitschrift Acta Eruditorum, 1682)
 * wird die Kreiszahl Pi approximiert
 * (Anzahl der Summanden = N)
 * Ausgabe zum Plotten (Gnuplot oder Python) mittels: "./a.out >> While_2.dat" */
#include <iostream>                                          // Ein- und Ausgabebibliothek
#include <cmath>                                             // Bibliothek für mathematisches (e-Funktion, Betrag, ...)

int main(){                                                  // Hauptfunktion
    unsigned int k = 0;                                      // Definition des Summenindex als ganze Zahl
    const unsigned int N = 100;                              // Definition der Anzahl der mitgenommenen Summenglieder als konstante ganze Zahl
    double Pi_Approx = 0;                                    // Definition der approximierten Gleitkommazahl von Pi (Datentyp 'double')

    printf("# 0: Summenindex k \n# 1: Approximierter Wert von Pi \n"); // Beschreibung der ausgegebenen Groessen

    while( k <= N ){                                         // Schleife zur Berechnung von Pi/4 mit Abbruchbedingung
        Pi_Approx = Pi_Approx + pow(-1,k)/(2*k + 1);         // Innerer Teil der Leibniz-Reihe
        printf("%4i %20.15f \n",k, 4*Pi_Approx);             // Ausgabe des approximierten Wertes von Pi
        ++k;                                                 // Index der Summe wird um eins erhöht
    }                                                        // Ende der Schleife

    Pi_Approx = Pi_Approx*4;                                 // Leibniz-Reihe liefert Pi/4, deshalb hier mal 4

    printf("# Approximierter Wert von Pi:     %20.15f \n", Pi_Approx); // Ausgabe des approximierten Wertes
    printf("# Wirklicher Wert von Pi:         %20.15Lf \n", M_PIl);    // Ausgabe des wirklichen Wertes
}
```

Reihen

als C++ Schleifenanweisungen

Im Folgenden wird das Konvergenzverhalten einer Reihe mittels einer while-Schleife in einem C++ Programm implementiert.

Anwendung: Mathematische Reihen

Wir betrachten im Folgenden das mathematische Problem der Berechnung der Kreiszahl $\pi = 3.141592653589793...$ und verwenden dafür die im Jahre 1682 von Gottfried Wilhelm Leibniz vorgestellte iterative Annäherung. Die sogenannte alternierende Leibniz-Reihe konvergiert im Limes $\lim_{N\to\infty}$ zu dem Wert der irrationalen Zahl $\frac{\pi}{4}$:

$$\sum_{k=0}^{N} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \ldots + \frac{(-1)^N}{2N+1}, \quad \text{mit:} \quad \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{\pi}{4}$$

While_2.cpp

```
/* Mittels der Leibniz-Reihe
 * (Gottfried Wilhelm Leibniz, Zeitschrift Acta Eruditorum, 1682)
 * wird die Kreiszahl Pi approximiert
 * (Anzahl der Summanden = N)
 * Ausgabe zum Plotten (Gnuplot oder Python) mittels: "./a.out >>
#include <iostream>
#include <cmath>

int main(){
    unsigned int k = 0;
    const unsigned int N = 100;
    double Pi_Approx = 0;

    printf("# 0: Summenindex k \n# 1: Approximierter Wert von Pi \n"); // Beschreibung der ausgegebenen Groessen

    while( k <= N ){                                                   // Schleife zur Berechnung von Pi/4 mit Abbruchbedingung
        Pi_Approx = Pi_Approx + pow(-1,k)/(2*k + 1);                   // Innerer Teil der Leibniz-Reihe
        printf("%4i %20.15f \n",k, 4*Pi_Approx);                       // Ausgabe des approximierten Wertes von Pi
        ++k;                                                           // Index der Summe wird um eins erhöht
    }                                                                  // Ende der Schleife

    Pi_Approx = Pi_Approx*4;                                           // Leibniz-Reihe liefert Pi/4, deshalb hier mal 4

    printf("# Approximierter Wert von Pi:     %20.15f \n", Pi_Approx); // Ausgabe des approximierten Wertes
    printf("# Wirklicher Wert von Pi:         %20.15Lf \n", M_PIl);    // Ausgabe des wirklichen Wertes
}
```

```
  94    3.152118677831945
  95    3.131176269454982
  96    3.151901658056018
  97    3.131388837543198
  98    3.151693406071117
  99    3.131592903558554
 100    3.151493401070991
# Approximierter Wert von Pi:        3.151493401070991
# Wirklicher Wert von Pi:            3.141592653589793
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$
```

Reihen

als C++ Schleifenanweisungen

Im Folgenden wird das Konvergenzverhalten einer Reihe mittels einer while-Schleife in einem C++ Programm implementiert.

Anwendung: Mathematische Reihen

Wir betrachten im Folgenden das mathematische Problem der Berechnung der Kreiszahl $\pi = 3.141592653589793...$ und verwenden dafür die im Jahre 1682 von Gottfried Wilhelm Leibniz vorgestellte iterative Annäherung. Die sogenannte alternierende Leibniz-Reihe konvergiert im Limes $\lim_{N\to\infty}$ zu dem Wert der irrationalen Zahl $\frac{\pi}{4}$:

$$\sum_{k=0}^{N} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \ldots + \frac{(-1)^N}{2N+1}, \quad \text{mit:} \quad \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{\pi}{4}$$

While_2.cpp

```cpp
/* Mittels der Leibniz-Reihe
 * (Gottfried Wilhelm Leibniz, Zeitschrift Acta Eruditorum, 1682)
 * wird die Kreiszahl Pi approximiert
 * (Anzahl der Summanden = N)
 * Ausgabe zum Plotten (Gnuplot oder Python) ./a.out >>
#include <iostream>
#include <cmath>

int main(){
    unsigned int k = 0;
    const unsigned int N = 100;
    double Pi_Approx = 0;

    printf("# 0: Summenindex k \n# 1: Approximierter Wert von Pi \n");

    while( k <= N ){
        Pi_Approx = Pi_Approx + pow(-1,k)/(2*k + 1);
        printf("%4i %20.15f \n",k, 4*Pi_Approx);
        ++k;
    }

    Pi_Approx = Pi_Approx*4;

    printf("# Approximierter Wert von Pi:     %20.15f \n", Pi_Approx);
    printf("# Wirklicher Wert von Pi:         %20.15Lf \n", M_PIl);
}
```

```
94    3.152118677831945
95    3.131176269454982
96    3.151901658056018
97    3.131388837543198
98    3.151693406071117
99    3.131592903558554
      3.151493401070991
# Approximierter Wert von Pi:     3.151493401070991
# Wirklicher Wert von Pi:         3.141592653589793
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V2$
```

Bei nummerischen Berechnung von Folgen(Reihen) ist es jedoch besser eine **for**-Schleife, anstatt einer **while**-Schleife zu verwenden, da der Folgenindex(Summenindex) eine offensichtliche Schleifenvariable darstellt und die Aktualisierung der Schleifenvariable am Ende des 'Anweisungsblockes' stattfindet -> Siehe Übungsaufgabe 2

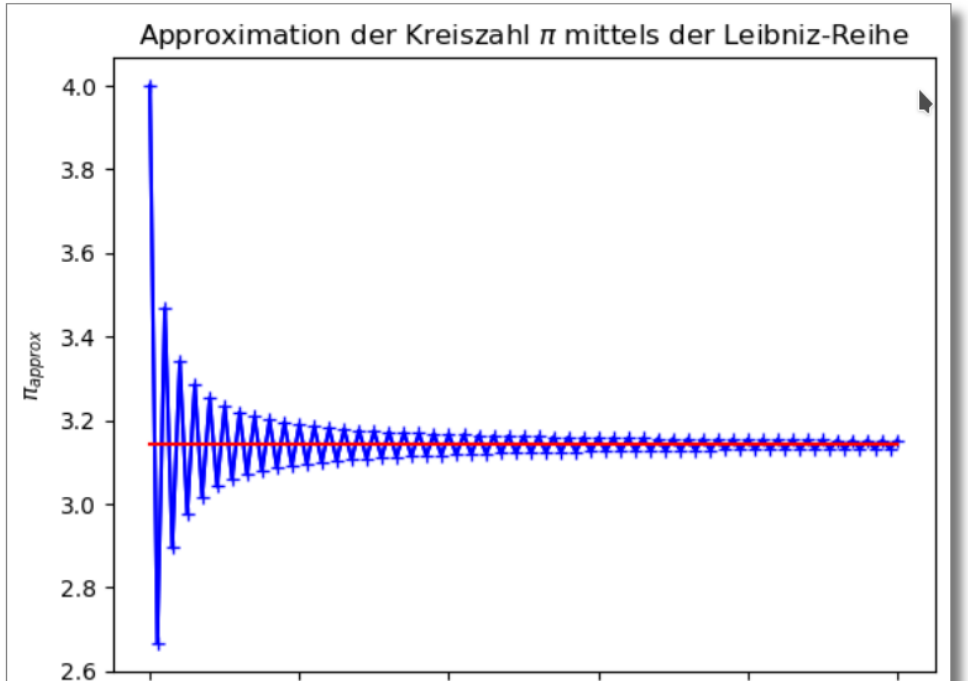
Visualisierung der berechneten Daten

mittels eines Python Programms (Python-Skript)

Anwendung: Mathematische Reihen

Wir betrachten im Folgenden das mathematische Problem der Berechnung der Kreiszahl $\pi = 3.141592653589793...$ und verwenden dafür die im Jahre 1682 von Gottfried Wilhelm Leibniz vorgestellte iterative Annäherung. Die sogenannte alternierende Leibniz-Reihe konvergiert im Limes $\lim\limits_{N\to\infty}$ zu dem Wert der irrationalen Zahl $\frac{\pi}{4}$:

$$\sum_{k=0}^{N} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \ldots + \frac{(-1)^N}{2N+1}, \quad \text{mit:} \quad \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{\pi}{4}$$



Python-Skript: PythonPlot_While_2.py

```
# Python Programm zum Plotten der berechneten Daten von (While_2.cpp)
import matplotlib.pyplot as plt                                  # Python Bibliothek zum Plotten (siehe https://matplotlib.org/ )
import numpy as np                                               # Python Bibliothek fuer Mathematisches (siehe https://numpy.org/ )

data = np.genfromtxt("./While_2.dat")                            # Einlesen der berechneten Daten von While_2.cpp

plt.title(r'Approximation der Kreiszahl $\pi$ mittels der Leibniz-Reihe') # Titel der Abbildung
plt.ylabel(r'$\pi_{approx}$')                                    # Beschriftung y-Achse
plt.xlabel('Index k')                                            # Beschriftung x-Achse
plt.plot(data[:,0],data[:,1], marker='+', color="blue")          # Plotten der Daten
plt.plot([data[0,0],data[-1,0]],[np.pi,np.pi], color="red")      # Horizontale Linie, die den exakten Wert von Pi markiert

plt.savefig("PythonPlot_While_2.png", bbox_inches="tight")       # Speichern der Abbildung als Bild
plt.show()                                                       # Zusaetzliches Darstellen der Abbildung in einem separaten Fenster
```

Die Programmiersprache Python

Python-Skripts und Jupyter Notebooks

Eine kleine Einführung in die Programmiersprache Python

Die Programmiersprache Python ist auch eine sehr gute Objekt-orientierte Programmiersprache und im Prinzip hätten wir die gesamte Vorlesung nur mittels Python gestalten können. Jedoch finden viele erfahrene Programmierer die Programmiersprache Python ein wenig unstrukturiert und auch, hinsichtlich der Performance der erstellten Programme ist die Programmiersprache C++ ein wenig besser und große, aufwendige Simulationsprogramme werden oft nicht mittels Python programmiert. Im Bereich der Visualisierung von Daten und im Bereich der Illustration von mathematisch/physikalischen Gleichungen hat Python jedoch sicherlich Vorteile gegenüber C++ und bietet mittels des Python-Moduls "matplotlib" (siehe Matplotlib: Visualization with Python) einen einfachen Weg Bilder and Animationen des simulierten Systems zu erzeugen.

Im Folgenden werde ich Ihnen keine strukturierte "Einführung in die Programmiersprache Python" geben, sondern es werden nur ganz spezielle Themen der Programmiersprache Python vorstellt, die wir zur Visualisierung der mittels unserer C++ Programme erstellten Daten und zum Verständnis der Vorlesung Einführung in die Programmierung für Studierende der Physik benötigen. Strukturierte Einführungen in die Programmiersprache Python finden Sie z.B. unter den folgenden Links:

Literatur zu Python

- Python-Onlinekurs auf Deutsch
- Python 3 documentation
- Hans Petter Langtangen: A Primer on Scientific Programming with Python
- David M. Beazley: Python - Essential Reference
- B. Slatkin: Effective Python

Im Speziellen werden wir in diesem Unterpunkt das Python-Skript (Python-Skript: PythonPlot_While_4.py) näher betrachten, welches im vorigen Unterpunkt Anwendungsbeispiel: Folgen und Reihen zur Datenvisualisierung verwendet wurde. Zusätzlich werden wir die im nächsten Unterpunkt Die Computerarithmetik und der Fehler in numerischen Berechnungen besprochene C++ Computerarithmetik mittels eines Python Jupyter Notebooks verdeutlichen und dabei die Verwendung von Listen, Arrays und for-Schleifen in der Programmiersprache Python kennenlernen.

Visualisierung der C++ Ausgabedaten mittels der Programmiersprache Python

Im Folgenden werden wir das Python-Skript (Python-Skript: PythonPlot_While_4.py, siehe untere Abbildung) näher betrachten.

Python-Skript: PythonPlot_While_2.py

```python
# Python Programm zum Plotten der berechneten Daten von (While_2.cpp)
import matplotlib.pyplot as plt                                  # Python Bibliothek zum Plotten (siehe https://matplotlib.org/ )
import numpy as np                                               # Python Bibliothek fuer Mathematisches (siehe https://numpy.org/ )

data = np.genfromtxt("./While_2.dat")                            # Einlesen der berechneten Daten von While_2.cpp

plt.title(r'Approximation der Kreiszahl $\pi$ mittels der Leibniz-Reihe') # Titel der Abbildung
plt.ylabel(r'$\pi_{approx}$')                                    # Beschriftung y-Achse
plt.xlabel('Index k')                                            # Beschriftung x-Achse
plt.plot(data[:,0],data[:,1], marker='+', color="blue")          # Plotten der Daten
plt.plot([data[0,0],data[-1,0]],[np.pi,np.pi], color="red")      # Horizontale Linie, die den exakten Wert von Pi markiert

plt.savefig("PythonPlot_While_2.png", bbox_inches="tight")       # Speichern der Abbildung als Bild
plt.show()                                                       # Zusaetzliches Darstellen der Abbildung in einem separaten Fenster
```

Python-Skripts

und Jupyter Notebooks

jupyter VisDataPython (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Einführung in die Programmierung für Studierende der Physik

(Introduction to Programming for Physicists)

Vorlesung gehalten an der J.W.Goethe-Universität in Frankfurt

(Sommersemester 2022)

von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske

Frankfurt am Main 01.04.2022

Visualisierung der C++ Ausgabedaten mittels der Programmiersprache Python

Im Folgenden werde ich Ihnen keine strukturierte "Einführung in die Programmiersprache Python" geben, sondern es werden nur ganz spezielle Themen der Programmiersprache Python vorstellt, die wir zur Visualisierung der mittels unserer C++ Programme erstellten Daten und zum Verständnis der Vorlesung Einführung in die Programmierung für Studierende der Physik benötigen. Strukturierte Einführungen in die Programmiersprache Python finden Sie z.B. unter den folgenden Links:

Literatur zu Python

jupyter Computerarithmetik Last Checkpoint: vor ein paar Sekunden (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Einführung in die Programmierung für Studierende der Physik

(Introduction to Programming for Physicists)

Vorlesung gehalten an der J.W.Goethe-Universität in Frankfurt am Main

(Sommersemester 2022)

von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske

Frankfurt am Main 01.04.2022

Computerarithmetik mit Python

In diesem Jupyter Notebook werden wir die im Unterpunkt Die Computerarithmetik und der Fehler in numerischen Berechnungen besprochene C++ Computerarithmetik an einem Beispiel mittels der Programmiersprache Python verdeutlichen.

Über den Fehler in nummerischen Berechnungen

der Rundungsfehler und der Approximierungsfehler

Die Computerarithmetik und der Fehler in numerischen Berechnungen

In diesem Unterpunkt werden wir zwei Klassen von Fehlerquellen in numerischen Berechnungen besprechen, den sogenannten *Rundungsfehler* der aufgrund der Computerarithmetik entsteht und der sogenannte *Approximierungsfehler* (*Abschneidefehler* bzw.'*Truncation error*'), der immer dann auftritt, wenn der Programmierer eine exakte mathematische Gleichung mittels approximativer Ausdrücke annähert.

Die Computerarithmetik und der Rundungsfehler

In der Mathematik werden Zahlen mit einer unendlichen Anzahl von Nachkommastellen zugelassen und die definierte Arithmetik auf diesem Zahlenraum $\mathbb{R}$ ist exakt (siehe Unterpunkt Datentypen und Variablen). Der Zahlenraum in dem der Computer rechnet ($\mathbb{R}_C$), benutzt jedoch Zahlen, die nur eine endliche Anzahl von Nachkommastellen haben und es wird hierbei nur eine relativ kleine Teilmenge der reellen Zahlen benutzt. Diese Teilmenge $\mathbb{R}_C$ umfasst nur rationale Zahlen und speichert den gebrochenen Teil, der mit *Mantisse* bezeichnet wird, zusammen mit dem exponentiellen Teil, welchen man *Charakteristik* nennt. Hierbei speichert der Computer die Zahl als eine binäre Liste von Nullen und Einsen. Im Folgenden wird das Beispiel der Speicherung einer Gleitkommazahl einfacher Genauigkeit besprochen, wie sie in IBM-Großrechnern der 3000er Serie benutzt wurde (Details findet man in R.L. Burden und J.D. Faires 2000: Numerische Methoden, Kapitel 1.3). In diesen Rechnern besteht eine Zahl mit einfacher Genauigkeit aus einem Vorzeichen-*bit*, einem 7-*bit* Exponenten der Basis 16 und einer 24-*bit*-Mantisse. Wir betrachten z.B. die binäre Computerzahl

$$\underbrace{0}_{\text{Vorzeichen}}\quad\underbrace{1000010}_{\text{Charakteristik}}\quad\underbrace{101100110000010000000000}_{\text{Mantisse}}\ .$$

Das Vorzeichen-*bit* ist 0, was bedeutet, dass die Zahl positiv ist. Die darauf folgenden 7-*bit* Zahlen beschreiben die Charakteristik, und diese berechnet sich wie folgt:

$$1\cdot 2^6+0\cdot 2^5+0\cdot 2^4+0\cdot 2^3+0\cdot 2^2+1\cdot 2^1+0\cdot 2^0=66$$

Um auch kleine Zahlenwerte abbilden zu können, wird von diesem Wert 64 abgezogen, sodass man als Exponenten der Zahl $16^{66-64}=16^2$ erhält. Der Zahlenwert der Mantisse ergibt sich wie folgt aus der am Ende stehenden 24-*bit* Zahlenfolge (die Null-Einträge sind hierbei nicht aufgelistet):

$$1\cdot\left(\frac{1}{2}\right)^1+1\cdot\left(\frac{1}{2}\right)^3+1\cdot\left(\frac{1}{2}\right)^4+1\cdot\left(\frac{1}{2}\right)^7+1\cdot\left(\frac{1}{2}\right)^8+1\cdot\left(\frac{1}{2}\right)^{14}=0.69927978515625$$

Die Computerarithmetik und der Rundungsfehler

In der Mathematik werden Zahlen mit einer unendlichen Anzahl von Nachkommastellen zugelassen und die definierte Arithmetik auf diesem Zahlenraum $\mathbb{R}$ ist exakt (siehe Unterpunkt Datentypen und Variablen). Der Zahlenraum in dem der Computer rechnet ($\mathbb{R}_C$), benutzt jedoch Zahlen, die nur eine endliche Anzahl von Nachkommastellen haben und es wird hierbei nur eine relativ kleine Teilmenge der reellen Zahlen benutzt. Diese Teilmenge $\mathbb{R}_C$ umfasst nur rationale Zahlen und speichert den gebrochenen Teil, der mit *Mantisse* bezeichnet wird, zusammen mit dem exponentiellen Teil, welchen man *Charakteristik* nennt. Hierbei speichert der Computer die Zahl als eine binäre Liste von Nullen und Einsen. Im Folgenden wird das Beispiel der Speicherung einer Gleitkommazahl einfacher Genauigkeit besprochen, wie sie in IBM-Großrechnern der 3000er Serie benutzt wurde (Details findet man in R.L. Burden und J.D. Faires 2000: Numerische Methoden, Kapitel 1.3). In diesen Rechnern besteht eine Zahl mit einfacher Genauigkeit aus einem Vorzeichen-*bit*, einem 7-*bit* Exponenten der Basis 16 und einer 24-*bit*-Mantisse. Wir betrachten z.B. die binäre Computerzahl

$$\underbrace{0}_{\text{Vorzeichen}}\quad\underbrace{1000010}_{\text{Charakteristik}}\quad\underbrace{101100110000010000000000}_{\text{Mantisse}}\quad .$$

Numerische Methoden, Kapitel 1.3). In diesen Rechnern besteht eine Zahl mit einfacher Genauigkeit aus einem Vorzeichen-*bit*, einem 7-*bit* Exponenten der Basis 16 und einer 24-*bit*-Mantisse. Wir betrachten z.B. die binäre Computerzahl

$$\underbrace{0}_{\text{Vorzeichen}}\quad\underbrace{1000010}_{\text{Charakteristik}}\quad\underbrace{101100110000010000000000}_{\text{Mantisse}}\quad .$$

Das Vorzeichen-*bit* ist 0, was bedeutet, dass die Zahl positiv ist. Die darauf folgenden 7-*bit* Zahlen beschreiben die Charakteristik, und diese berechnet sich wie folgt:

$$1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 66$$

Um auch kleine Zahlenwerte abbilden zu können, wird von diesem Wert 64 abgezogen, sodass man als Exponenten der Zahl $16^{66-64} = 16^2$ erhält. Der Zahlenwert der Mantisse ergibt sich wie folgt aus der am Ende stehenden 24-*bit* Zahlenfolge (die Null-Einträge sind hierbei nicht aufgelistet):

Numerische Methoden, Kapitel 1.3). In diesen Rechnern besteht eine Zahl mit einfacher Genauigkeit aus einem Vorzeichen-*bit*, einem 7-*bit* Exponenten der Basis 16 und einer 24-*bit*-Mantisse. Wir betrachten z.B. die binäre Computerzahl

$$\underbrace{0}_{\text{Vorzeichen}} \quad \underbrace{1000010}_{\text{Charakteristik}} \quad \underbrace{101100110000010000000000}_{\text{Mantisse}} \quad .$$

Das Vorzeichen-*bit* ist 0, was bedeutet, dass die Zahl positiv ist. Die darauf folgenden 7-*bit* Zahlen beschreiben die Charakteristik, und diese berechnet sich wie folgt:

$$1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 66$$

Um auch kleine Zahlenwerte abbilden zu können, wird von diesem Wert 64 abgezogen, sodass man als Exponenten der Zahl $16^{66-64} = 16^2$ erhält. Der Zahlenwert der Mantisse ergibt sich wie folgt aus der am Ende stehenden 24-*bit* Zahlenfolge (die Null-Einträge sind hierbei nicht aufgelistet):

$$1 \cdot \left(\frac{1}{2}\right)^1 + 1 \cdot \left(\frac{1}{2}\right)^3 + 1 \cdot \left(\frac{1}{2}\right)^4 + 1 \cdot \left(\frac{1}{2}\right)^7 + 1 \cdot \left(\frac{1}{2}\right)^8 + 1 \cdot \left(\frac{1}{2}\right)^{14} = 0.69927978515625$$

Die eigentliche Gleitkommazahl ergibt sich nun mittels der folgenden Formel Zahl:= $M \cdot 16^{C-64}$, wobei M den Zahlenwert der Mantisse und C den Wert der Charakteristik bezeichnet. Die entsprechende Gleitkommazahl unserer gesamten binären Zahlenliste 01000010101100110000010000000000 berechnet sich somit schließlich zu folgendem Zahlenwert:

$$01000010101100110000010000000000 \quad \Longleftrightarrow \quad 0.69927978515625 \cdot 16^2 = 179.015625$$

Über den Fehler in nummerischen Berechnungen

Der Approximierungsfehler 'Truncation error'

Der '*Truncation error*' (*Abschneidefehler* bzw. *Approximierungsfehler*) ist ein Fehler, dessen Größe der Programmierer, zumindest in einem gewissen Bereich, unter seiner eigenen Kontrolle hat. Er tritt immer dann auf, wenn exakte mathematische Gleichungen mittels approximativer Ausdrücke annähert.
Betrachten wir z.B. die im Unterpunkt Anwendungsbeispiel: Folgen und Reihen besprochene approximative Annäherung an die Kreiszahl $\pi = 3.141592653589793...$ mittels der alternierenden Leibniz-Reihe und nehmen an, dass wir dabei lediglich $N = 10$ Folgenglieder der Partialsumme bei der Approximation berücksichtigen:

$$\pi_{approx} = 4 \cdot \sum_{k=0}^{10} \frac{(-1)^k}{2k+1}, \quad \text{dann gilt:} \quad \pi = 4 \cdot \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \pi_{approx} + \underbrace{4 \cdot \sum_{k=11}^{\infty} \frac{(-1)^k}{2k+1}}_{\text{Truncation error}}$$

Der Programmierer hat es dabei selbst in der Hand, wie viele Folgenglieder der Partialsumme er mitnimmt und somit wie groß der *Abschneidefehler* seiner Approximation ist. Im Laufe der Vorlesung werden wir in vielen Anwendungen Approximierungen von exakten mathematische Ausdrücken machen und der dabei entstehende *Truncation error* ist hierbei oft die größte Fehlerquelle.

Aufgaben

Übungsblatt Nr. 3

Aufgabe 1 *(5 Punkte)*

Stellen Sie für die folgende Reihe das Konvergenzverhalten für die ersten 100 ($N = 100$) Folgenglieder der Partialsumme grafisch dar

$$s_n = \sum_{k=0}^{n} q^k, \quad \text{mit:} \quad q = \frac{8}{9}$$

Erstellen Sie dafür ein C++ Programm, welches mittels einer for-Schleife die einzelnen Folgenglieder der Partialsumme formatiert auf 15 Stellen genau ausgibt und leiten Sie die berechneten Daten in eine Datei um. Die eigentliche Visualisierung machen Sie dann bitte unter Verwendung eines Python-Skriptes. Halten Sie bitte das Programm so allgemein, das man die konstante Variable q direkt bei ihrer Deklaration zum Wert $q = \frac{8}{9}$ initialisiert. Stellen Sie die Folge der Partialsummen der Reihe gegen den Index n grafisch dar und vergleichen Sie das Konvergenzverhalten mit dem Grenzwert der unendlich geometrischen Reihe

$$\sum_{k=0}^{\infty} q^k = \frac{1}{1-q}, \quad \text{mit:} \quad q \in \mathbb{R}\ ,\ |q| < 1 \quad .$$

Aufgabe 2 *(5 Punkte)*

Obwohl es bei der nummerischen Berechnung von Folgen und Reihen besser ist, eine **for**-Schleife zu verwenden (offensichtliche Schleifenvariable, Aktualisierung der Schleifenvariable am Ende des 'Anweisungsblockes') wurden die im Unterpunkt Anwendungsbeispiel: Folgen und Reihen dargestellten Folgen und Reihen mit einer **while**-Schleife programmiert. Schreiben Sie bitte das im Teilkapitel "Anwendung: Mathematische Reihen" vorgestellte C++ Programm (While_2.cpp) unter Verwendung einer for-Schleife um.

Aufgabe 3 *(5 Punkte)*

Berechnen Sie die folgenden Ausdrücke mittels eines C++ Programms und lassen Sie sich die berechneten Werte auf 15 Stellen genau ausgeben (benutzen Sie bei der Berechnung doppelte Maschienengenauigkeit und verwenden Sie eine geeignete *Schleifenanweisung*).

$$\sum_{k=0}^{2536} \left(\frac{1}{2}\right)^k \quad , \quad \sum_{k=3}^{45} k^{\frac{3}{5}} \quad , \quad \sum_{k=-5}^{20} \frac{k^5}{e^{-k}+1}$$

Aufgabe 4 *(5 Punkte)*

Berechnen Sie die ersten 40 Zahlenwerte der Fibonacci-Folge mittels einer for-Schleife. Die Fibonacci-Folge $f_n, n = [1, 2, 3, \dots]$ ist durch folgendes rekursives Bildungsgesetzt definiert:

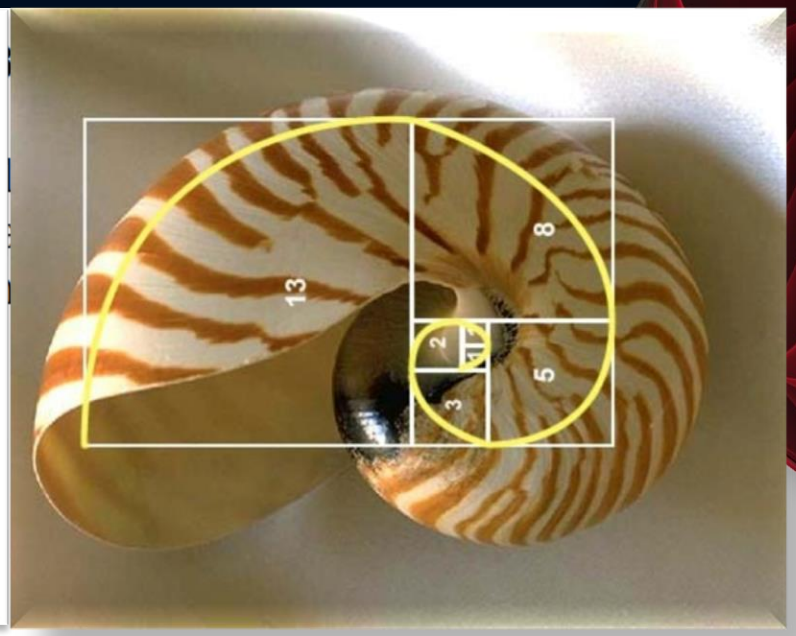
$$f_n = f_{n-1} + f_{n-2} \ , \quad \text{mit den Anfangswerten: } f_1 = f_2 = 1$$

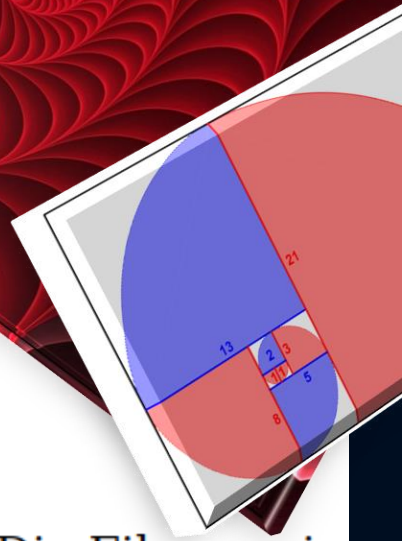
Zeigen Sie, dass das Verhältnis zweier aufeinanderfolgender Zahlen der Fibonacci-Folge ($\frac{f_n}{f_{n-1}}$) im Grenzwert $\lim\limits_{n\to\infty}$ gegen die irrationale Zahl des Goldenen Schnitts $\Phi \approx 1.6180339887498948482045868 34$ konvergiert. Bemerkung: Der goldene Schnitt ist in vielen Bereichen der Mathematik, Kunst, Architektur und Biologie von Bedeutung (näheres siehe z.B. Wikipedia: Goldene Schnitt).

Der Goldene Schnitt

Als **Goldener Schnitt** (lateinisch *sectio aurea*, *proportio divina*) wird das Teilungsverhältnis (M/m) einer Strecke bezeichnet, bei dem das Verhältnis des Ganzen (M+m) zu seinem größeren Teil (M) dem Verhältnis des größeren zum kleineren Teil gleich ist:

$$M/m = (M+m)/M$$





Aufgabe 4 *(5 Punkte)*

Berechnen Sie die ersten 40 Zahlenwerte der Fibonacci-Folge mittels einer for-Schleife. Die Fibonacci-Folge $f_n, n = [1, 2, 3, \ldots]$ ist durch folgendes rekursives Bildungsgesetzt definiert:

$$f_n = f_{n-1} + f_{n-2}\ , \quad \text{mit den Anfangswerten: } f_1 = f_2 = 1$$

Zeigen Sie, dass das Verhältnis zweier aufeinanderfolgender Zahlen der Fibonacci-Folge $(\frac{f_n}{f_{n-1}})$ im Grenzwert $\lim\limits_{n\to\infty}$ gegen die irrationale Zahl des Goldenen Schnitts $\Phi \approx 1.6180339887498948482045868 34$ konvergiert. Bemerkung: Der goldene Schnitt ist in vielen Bereichen der Mathematik, Kunst, Architektur und Biologie von Bedeutung (näheres siehe z.B. Wikipedia: Goldene Schnitt).