Einführung in die Programmierung für Studierende der Physik

JOHANN WOLFGANG GOETHE UNIVERSITÄT
03.05.2022

MATTHIAS HANAUSKE

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

4. Vorlesung

Plan für die heutige Vorlesung

- Kurze Wiederholung der Vorlesung 3
- C++ Anweisungen: Auswahlanweisungen mit if und switch
- Funktionen in C++
- Anwendungsbeispiel: Nullstellensuche einer Funktion
- Übungsaufgaben: Übungsblatt Nr.4

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
```

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:

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

While_4.cpp

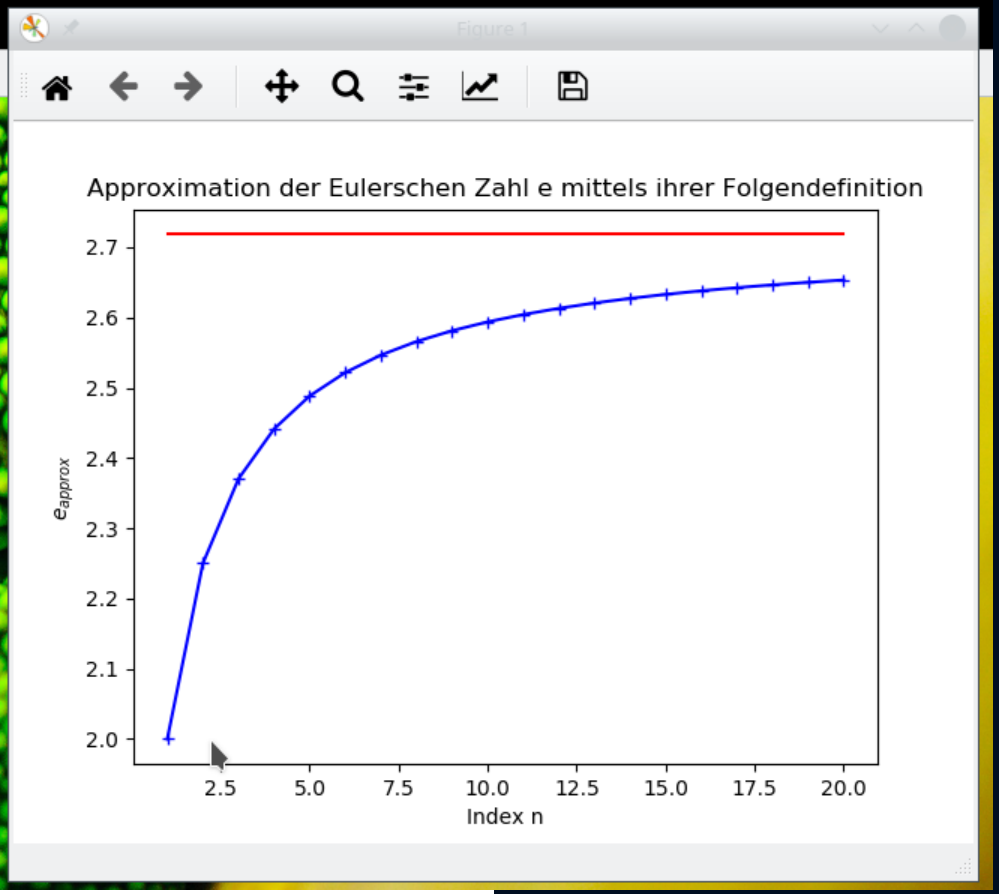
```
/* Die Eulerschen Zahl e wird mittels ihrer Folgendefinition approximiert
 * Ausgabe zum Plotten (Gnuplot oder Python) mittels: "./a.out > While_4.dat" */
#include <iostream>                                          // Ein- und Ausgabebibliothek
#include <cmath>                                  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

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
```

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

```
/* 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-7745-72G:~/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*

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

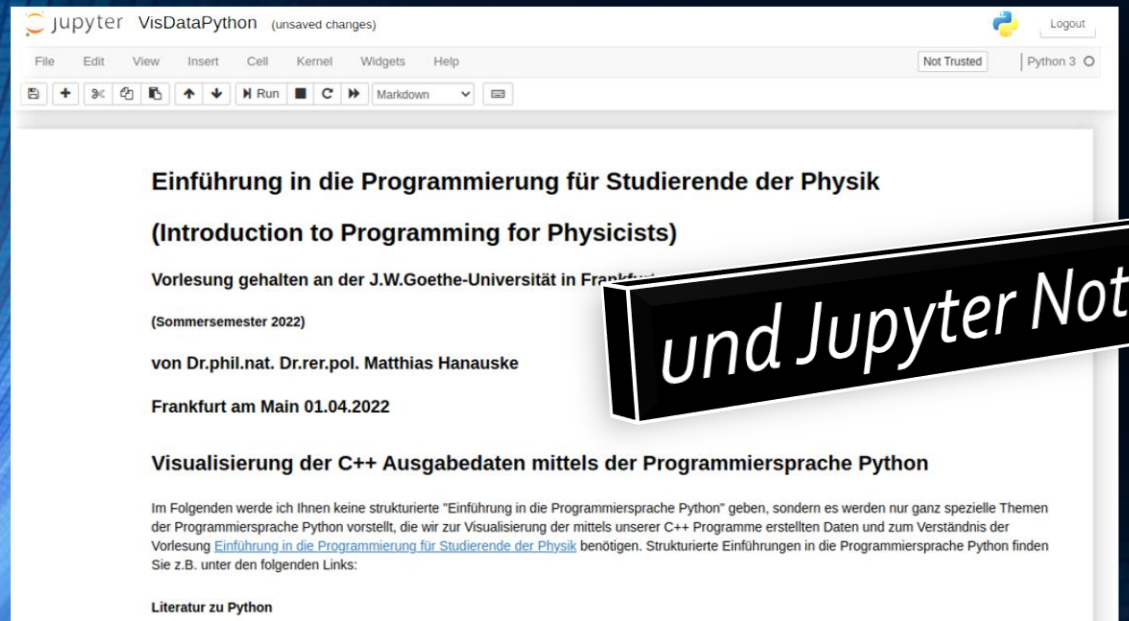
Im Folgenden werden wir das Python-Skript (Python-Skript: PythonPlot_While_4.py, siehe untere Abbildung) näher betrachten.

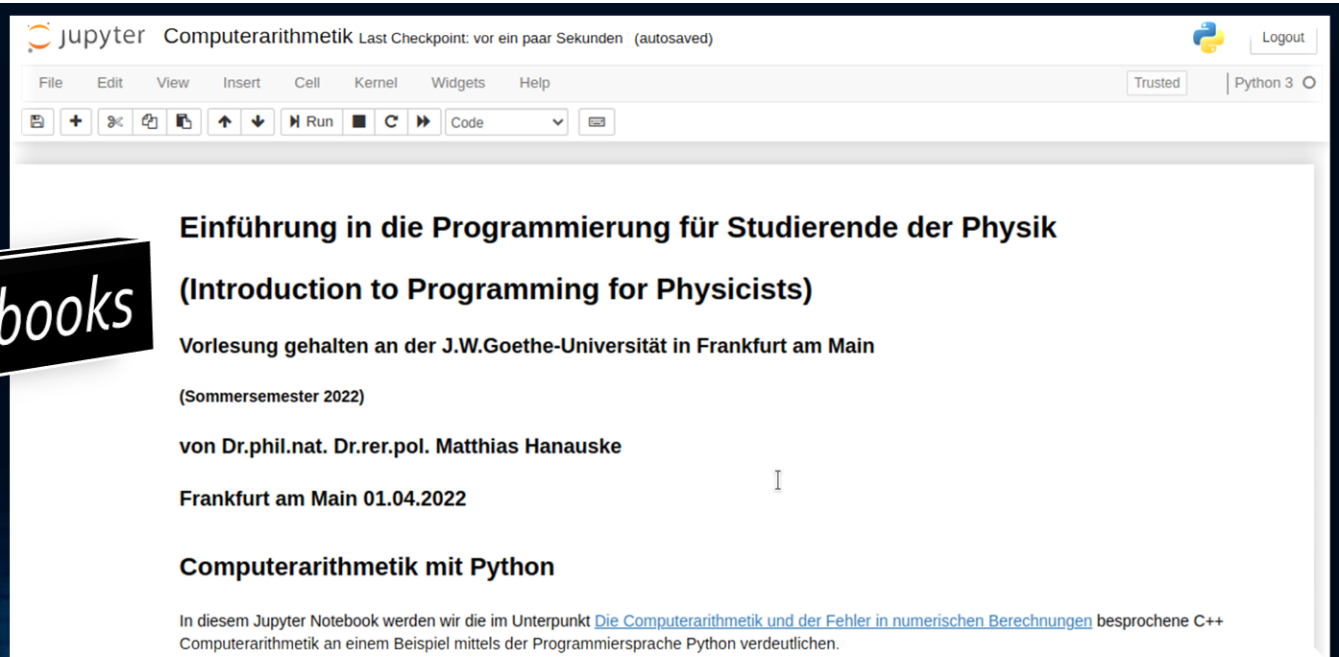
Python-Skript: PythonPlot_While_2.py

```
# Python 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





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

$$\underbrace{0}_{\text{Vorzeichen}} \quad \underbrace{1000010}_{\text{Charakteristik}} \quad \underbrace{10110011000001000000000}_{\text{Mantisse}} \; .$$

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):

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

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:

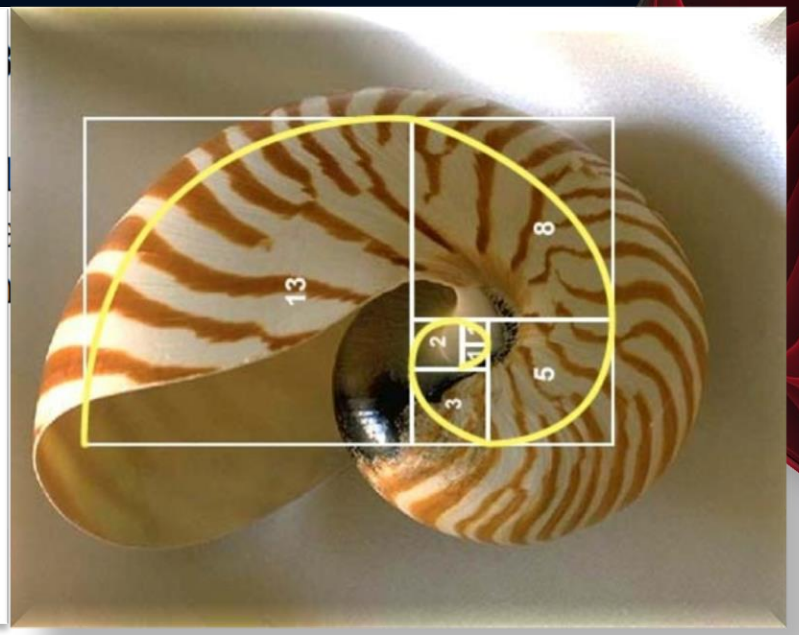
$$f_n = f_{n-1} + f_{n-2} \ , \quad \text{mit den Anfangswerten: } f_1 = f_2 = 1$$

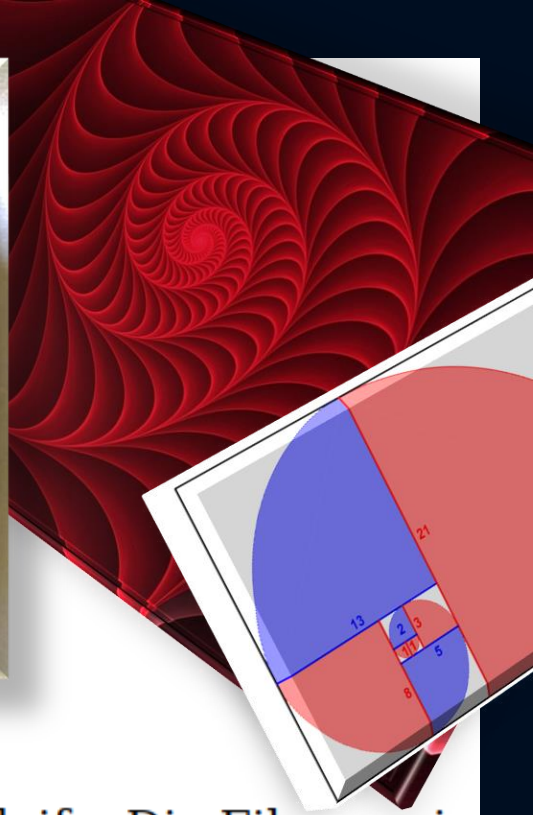
Zeigen Sie, dass das Verhältnis zweier aufeinanderfolgender Zahlen der Fibonacci-Folge $(\frac{f_n}{f_{n-1}})$ im Grenzwert $\lim_{n\to\infty}$ gegen die irrationale Zahl des Goldenen Schnitts $\Phi \approx 1.6180339887498948482045868343$ konvergiert. Bemerkung: Der goldene Schnitt ist in vielen Bereichen 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 $f_n, n = [1, 2, 3, \dots]$ 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 $\lim_{n\to\infty}$ gegen die irrationale Zahl des Goldenen Schnitts $\Phi \approx 1.6180339887498948482045868343$ konvergiert. Bemerkung: Der goldene Schnitt ist in vielen Bereichen der Mathematik, Kunst, Architektur und Biologie von Bedeutung (näheres siehe z.B. Wikipedia: Goldene Schnitt).

Vorlesung 4

In dieser Vorlesung werden wir zunächst die *Auswahlanweisungen* der Sprache C++ vorstellen und danach auf die Definition von *C++ Funktionen* eingehen. In dem Anwendungsbeispiel *Nullstellensuche einer Funktion* werden dann die erlernten Konzepte am Beispiel der *Methode der Bisektion*, dem sogenannten *Intervallhalbierungsverfahren* verdeutlicht. Zusätzlich wird am Ende die *Newton-Raphson Methode* der Nullstellenermittlung vorgestellt.

C++ Anweisungen: Auswahlanweisungen mit if und switch

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 *Auswahlanweisungen* behandeln, wobei wir in der vorigen Vorlesung die *Schleifenanweisungen* vorstellt hatten (siehe C++ Anweisungen: Die for-, while- und do-Schleifen). *Auswahlanweisungen* werden auch als Verzweigung , Tests oder *bedingte Anweisungen* bezeichnet und sind immer dann anzuwenden, wenn das Programm bei einem gewissen Ereignis bzw. unter einer gewissen Bedingung etwas Bestimmtes tun soll. Es teilt somit das Programm in unterschiedliche Anweisungspfade auf. *Auswahlanweisungen* können in Form einer **if**-, (**if-else**)- oder **switch**-Anweisung ausgedrückt werden. C++ Anweisungen: Auswahlanweisungen mit if und switch

Funktionen in C++

Die Definition einer C++ Funktion ist im Grunde nichts Anderes, als eine Code-Block (*Anweisungsblock*) mit einem Funktionsnamen zu verbinden. C++ Funktionen werden außerhalb der main()-Hauptfunktion definiert und vereinfachen somit das Verständnis und die Lesbarkeit des Quelltextes. C++ Funktionen sind ein wichtiges Werkzeug, um den Quelltext eines Programms zu ordnen und wesentliche Algorithmen und zusammenhängende Anweisungsblöcke der main()-Hauptfunktion in einer zusammenhängenden Form auszulagern. Die Definition einer C++ Funktion besteht aus einer *Deklaration* und einem *Anweisungsblock* und sie ist der formalen Definition einer mathematischen Funktion nicht unähnlich: "Eine C++ Funktion ist eine Abbildung von dem Datenraum der *Argumentenliste* in den Datenraum des *Rückgabetyps*. Die dabei benutzte Abbildungsvorschrift findet sich in dem *Anweisungsblock* der Funktion. Der 'Rückgabe Typ' kann hierbei eine der schon besprochenen Datentypen (z.B. int oder double) oder ein Daten-Array (siehe nächste Vorlesung) sein. Die *Argumentenliste* setzt sich aus einer Liste von Datentypen der formalen Argumente (Parameter) der Funktion zusammen, die jeweils mit einem Komma voneinander getrennt sind. In C++ hat das Wort Funktion eine allgemeinere Bedeutung als im Bereich der Mathematik und die in der Mathematik und Physik definierte Funktionen stellen eine echte Teilmenge der C++ Funktionen dar (näheres siehe Funktionen in C++).

Anwendungsbeispiel: Nullstellensuche einer Funktion

Vorlesung 4

Die möglichen integrierten Anweisungsbefehle innerhalb einer Programmiersprache sind die wohl wichtigsten Grundvokabeln, die man als Programmierer kennen muss. In der vorigen Vorlesung hatten wir uns bereits mit den *Schleifenanweisungen* befasst und in dieser Vorlesung werden wir uns mit den *Auswahlanweisungen* beschäftigen. Im Speziellen werden die **if**-Anweisung, die (**if-else**)-Anweisung und die **switch**-Anweisung besprochen. *Auswahlanweisungen* stellen eine Art von Programmverzweigungen dar und sind immer dann anzuwenden, wenn das Programm bei einem gewissen Ereignis etwas Bestimmtes tun soll.
Ein weiteres wichtiges Konzept bei der Erstellung eines C++ Quelltextes ist der Begriff der "Funktion". *C++ Funktionen* sind ein wichtiges Werkzeug, um den Quelltext eines Programms zu ordnen und wesentliche Algorithmen und zusammenhängende Anweisungsblöcke der main()-Hauptfunktion in einer zusammenhängenden Form auszulagern. Man könnte salopp sagen, dass Funktionen kleine Unterprogramme sind, die definierte Teilprobleme lösen. Im Grunde bedeutet die Definition einer Funktion im Programm nichts Anderes, als eine Code-Block (*Anweisungsblock*) mit einem Funktionsnamen zu verbinden. In C++ hat das Wort Funktion eine allgemeinere Bedeutung als im Bereich der Mathematik und die in der Mathematik und Physik definierte Funktionen stellen eine echte Teilmenge der C++ Funktionen dar. Wir werden in dieser Vorlesung lediglich eine erste grundsätzliche Einführung in den Themenbereich der *C++ Funktionen* geben und auf die allgemeinere Verwendung von Funktionen im Laufe der Vorlesung noch genauer eingehen. Gerade in den Unterpunkten, die sich mit der Objekt-orientierten Programmierung befassen, sind Klassen-Funktionen, Konstruktoren und das Überladen von Funktionen ein wichtiges Thema.
Am Ende dieser Vorlesung wenden wir das Erlernte an und besprechen eines der grundlegenden Probleme der numerischen Mathematik: *Die Nullstellensuche einer Funktion*. Wir nehmen dabei an, dass eine Funktion $f(x)$ im Intervall $[a, b] \in \mathbb{R}$ eine Nullstelle hat und berechnen dann diese Nullstelle numerisch, approximativ mittels der *Methode der Bisektion* und dem *Newton-Raphson Algorithmus*. Beide Methoden werden in dem Anwendungsbeispiel:

C++ Anweisungen: Auswahlanweisungen mit if und switch

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 sogennanten *Auswahlanweisungen* behandeln wobei wir in der vorigen Vorlesung die *Schleifenanweisungen* vorstellt hatten (siehe C++ Anweisungen: Die for-, while- und do-Schleifen). *Auswahlanweisungen* werden auch als Verzweigung , Tests oder *bedingte Anweisungen* bezeichnet und sind immer dann anzuwenden, wenn das Programm bei einem gewissen Ereignis bzw. unter einer gewissen Bedingung etwas bestimmtes tun soll. Es teilt somit das Programm in unterschiedliche Anweisungspfade auf. *Auswahlanweisungen* können in Form einer **if**-, (**if-else**)- oder **switch**-Anweisung ausgedrückt werden.

Die if-Anweisung

Eine **if**-Anweisung hat die folgende Struktur:

if ('Anweisungsbedingung') { 'Block von Anweisungen' }

Die 'Anweisungsbedingung' spezifiziert, in welchem Fall der 'Block von Anweisungen' ausgeführt werden soll. Ist diese Bedingung nicht erfüllt, so überspringt das Programm einfach die **if**-Anweisung.

If_0.cpp

```cpp
#include <iostream>                                          // Ein- und Ausgabebibliothek
using namespace std;                                         // Benutze den Namensraum std

int main(){                                                  // Hauptfunktion
    int zahl;                                                // Deklaration der Integer Variable 'zahl'

    cout << "Geben Sie bitte eine Fibonacci-Zahl im Bereich zwischen 100 und 200 ein: "; // Ausgabe eines Textes
    cin  >> zahl;                                            // Einlesen der Zahl mittels der Tastatur

    if( zahl == 144 ){                                       // if-Anweisung Anfang
        cout << "Richtig!" << endl;                          // Ausgabe falls if-Bedingung erfüllt ist
    }                                                        // if-Anweisung Ende
```

Das nebenstehende Programm gibt ein einfaches Beispiel für die Verwendung einer **if**-Anweisung. Das Programm fordert zunächst den Benutzer auf eine Zahl der Fibonacci-Folge im Bereich zwischen 100 und 200 einzugeben. Da es in diesem Zahlenbereich nur eine Fibonacci-Zahl gibt (144) ist es eindeutig, ob die vom Benutzer eingegebene Zahl richtig ist. Die **if**-Anweisung wird nun dafür benutzt zu entscheiden, ob die eingegebene Zahl richtig ist und dies wird mittels der 'Anweisungsbedingung' (zahl == 144) überprüft. Falls die 'Anweisungsbedingung' erfüllt ist, wird der 'Block von Anweisungen', in

Die if-Anweisung

Die 'Anweisungsbedingung' spezifiziert, in welchem Fall der 'Block von Anweisungen' ausgeführt werden soll. Ist diese Bedingung nicht erfüllt, so überspringt das Programm einfach die **if**-Anweisung.

Eine **if**-Anweisung hat die folgende Struktur:

if ('Anweisungsbedingung') { 'Block von Anweisungen' }

Das untere Programm gibt ein einfaches Beispiel für die Verwendung einer **if**-Anweisung. Das Programm fordert zunächst den Benutzer auf eine Zahl der Fibonacci-Folge im Bereich zwischen 100 und 200 einzugeben. Da es in diesem Zahlenbereich nur eine Fibonacci-Zahl gibt (144) ist es eindeutig, ob die vom Benutzer eingegebene Zahl richtig ist.

If_0.cpp

```cpp
#include <iostream>                                          // Ein- und Ausgabebibliothek
using namespace std;                                         // Benutze den Namensraum std

int main(){                                                  // Hauptfunktion
    int zahl;                                                // Deklaration der Integer Variable 'zahl'

    cout << "Geben Sie bitte eine Fibonacci-Zahl im Bereich zwischen 100 und 200 ein: "; // Ausgabe eines Textes
    cin  >> zahl;                                            // Einlesen der Zahl mittels der Tastatur

    if( zahl == 144 ){                                       // if-Anweisung Anfang
        cout << "Richtig!" << endl;                          // Ausgabe falls if-Bedingung erfüllt ist
    }                                                        // if-Anweisung Ende

    cout << "Vielen Dank für Ihre Eingabe."  << endl;        // Ausgabe eines Textes
}
```

Die if-Anweisung

Liste der ersten 40 Fibonacci Zahlen ->

If_0.cpp

```cpp
#include <iostream>                                          // Ein- und Ausgabebibliothek
using namespace std;                                         // Benutze den Namensraum std

int main(){                                                  // Hauptfunktion
    int zahl;                                                // Deklaration der Integer Variable 'zahl'

    cout << "Geben Sie bitte eine Fibonacci-Zahl im Bereich zwischen 100 und 200 ein: "; // Ausgabe eines Textes
    cin  >> zahl;                                            // Einlesen der Zahl mittels der Tastatur

    if( zahl == 144 ){                                       // if-Anweisung Anfang
        cout << "Richtig!" << endl;                          // Ausgabe falls if-Bedingung erfüllt ist
    }                                                        // if-Anweisung Ende

    cout << "Vielen Dank für Ihre Eingabe."  << endl;        // Ausgabe eines Textes
}
```

Terminalausgabe des Programms

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ g++ If_0.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ ./a.out
Geben Sie bitte eine Fibonacci-Zahl im Bereich zwischen 100 und 200 ein: 178
Vielen Dank für Ihre Eingabe.
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ ./a.out
Geben Sie bitte eine Fibonacci-Zahl im Bereich zwischen 100 und 200 ein: 144
Richtig!
Vielen Dank für Ihre Eingabe.
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~
(base) hanauske@hanauske-Aspire-A717-72G:~
# 0: Folgenindex n
# 1: Fibonacci Zahl
# 2: Goldener Schnitt
  0            1
  1            1
  3            2  2.00000000000000000
  4            3  1.50000000000000000
  5            5  1.66666666666666674
  6            8  1.60000000000000009
  7           13  1.62500000000000000
  8           21  1.61538461538461542
  9           34  1.61904761904761907
 10           55  1.61764705882352944
 11           89  1.61818181818181817
 12          144  1.61797752808988760
 13          233  1.61805555555555558
 14          377  1.61802575107296143
 15          610  1.61803713527851456
 16          987  1.61803278688524599
 17         1597  1.61803444782168193
 18         2584  1.61803381340012531
 19         4181  1.61803405572755410
 20         6765  1.61803396316670645
 21        10946  1.61803399852180330
 22        17711  1.61803398501735796
 23        28657  1.61803399017559713
 24        46368  1.61803398820532496
 25        75025  1.61803398895790207
 26       121393  1.61803398867044312
 27       196418  1.61803398878024263
 28       317811  1.61803398873830306
 29       514229  1.61803398875432247
 30       832040  1.61803398874820359
 31      1346269  1.61803398875054083
 32      2178309  1.61803398874964821
 33      3524578  1.61803398874998905
 34      5702887  1.61803398874985893
 35      9227465  1.61803398874990867
 36     14930352  1.61803398874988957
 37     24157817  1.61803398874989690
 38     39088169  1.61803398874989401
 39     63245986  1.61803398874989512
 40    102334155  1.61803398874989468
# Approximierte irrationale Zahl des Golde
# Wirklicher Zahlenwert des Goldenen Schni
```

Beispiel: Eingeschränkte Doppelsumme

Ein einfaches mathematisches Beispiel der Verwendung einer **if**-Anweisung ist z.B. die neben abgebildete Doppelsumme. In der zweiten Summe, besteht der Zusatz, dass der Summenindex i niemals den Wert des Summenindex k annehmen darf.

$$\sum_{k=1}^{4} \sum_{\substack{i=1 \\ i \neq k}}^{20} \frac{i^k}{(k-i)}$$

Doppelsumme_If_0.cpp

```cpp
#include <iostream>                                   // Ein- und Ausgabebibliothek
#include <cmath>                                      // Bibliothek für mathematisches (e-Funktion, Betrag, ...)
using namespace std;                                  // Benutze den Namensraum std

int main(){                                           // Hauptfunktion
    int k, i;                                         // Deklaration der Summenindexe k und i
    double Ergebnis = 0;                              // Deklaration und Initialisierung der Ergebnis-Variable

    for(k=1; k<=4; ++k){                              // Schleifen Anfang der 1.Summe
        for(i=1; i<=20; ++i){                          // Schleifen Anfang der 2.Summe
            if( i != k ){                             // if-Anweisung Anfang
                Ergebnis = Ergebnis + pow(i,k)/(k - i);   //Innerer, math. Teil der Doppelsumme
                }                                     // if-Anweisung Ende
        }                                             // Schleifen Ende der 2.Summe
    }                                                 // Schleifen Ende der 1.Summe

    cout << "Das Ergebnis der Doppelsumme beträgt: " << Ergebnis << endl; // Ausgabe des Ergebnisses
}
```

Das links abgebildete C++ Programm zeigt den Quelltext des Programmes zur Berechnung der angegebenen Doppelsumme. Der Anweisungsblock der 2. **for**-Schleife wird nur ausgeführt, falls der Wert des Index i ungleich dem Index k ist (**if**(i != k){ ... }). Das Ergebnis ergibt sich zu "Ergebnis = -64344".

(**if-else**)-Anweisung

Eine (**if-else**)-Anweisung hat die folgende Struktur:

if ('Anweisungsbedingung') { '1. Block von Anweisungen' } **else** { '2. Block von Anweisungen' }

Die 'Anweisungsbedingung' spezifiziert, in welchem Fall der '1. Block von Anweisungen' ausgeführt werden soll. Ist diese Bedingung nicht erfüllt, so führt das Programm den '2. Block von Anweisungen' aus.

If_Else_0.cpp

```cpp
#include <iostream>                                          // Ein- und Ausgabebibliothek
using namespace std;                                         // Benutze den Namensraum std

int main(){                                                  // Hauptfunktion
    int zahl;                                                // Deklaration der Integer Variable 'zahl'

    cout << "Geben Sie bitte eine Fibonacci-Zahl im Bereich zwischen 100 und 500 ein: "; // Ausgabe eines Textes
    cin  >> zahl;                                            // Einlesen der Zahl mittels der Tastatur

    if( zahl == 144 || zahl == 233 || zahl == 377 ){         // if-Anweisung Anfang mit logischem 'oder'
        cout << "Richtig!" << endl;                          // Ausgabe falls if-Bedingung erfüllt ist
    } else{                                                  // if-Anweisung Ende, else-Anweisung Anfang
        cout << "Leider Falsch :-(" << endl;                 // Ausgabe falls if-Bedingung nicht erfüllt ist
    }                                                        // else-Anweisung Ende

    cout << "Vielen Dank für Ihre Eingabe."  << endl;        // Ausgabe eines Textes
}
```

(**if-else**)-Anweisung

Das Programm gibt nun dem Benutzer eine Rückmeldung ob seine Zahl falsch war. Zusätzlich ist die Verwendung von Operatoren der Aussagenlogik bei mehrkomponentigen 'Anweisungsbedingungen' aufgezeigt. Das Programm fordert zunächst den Benutzer auf eine Zahl der Fibonacci-Folge im Bereich zwischen 100 und 500 einzugeben. Da es in diesem Zahlenbereich drei Fibonacci-Zahl gibt (144, 233 und 377) wurden die drei richtigen Fälle unter Verwendung des logischen Oder-Operators '||' in die 'Anweisungsbedingungen' der (**if-else**)-Anweisung (**if**(zahl == 144 || zahl == 233 || zahl == 377){ ... } **else**{ ...}) eingefügt. Um mehrere 'Anweisungsbedingungen' logisch miteinander zu verknüpfen steht einem neben dem Oder-Operator '||' noch der logische Und-Operator '&&' zu Verfügung.

If_Else_0.cpp

```cpp
#include <iostream>                                          // Ein- und Ausgabebibliothek
using namespace std;                                         // Benutze den Namensraum std

int main(){                                                  // Hauptfunktion
    int zahl;                                                // Deklaration der Integer Variable 'zahl'

    cout << "Geben Sie bitte eine Fibonacci-Zahl im Bereich zwischen 100 und 500 ein: "; // Ausgabe eines Textes
    cin  >> zahl;                                            // Einlesen der Zahl mittels der Tastatur

    if( zahl == 144 || zahl == 233 || zahl == 377 ){         // if-Anweisung Anfang mit logischem 'oder'
        cout << "Richtig!" << endl;                          // Ausgabe falls if-Bedingung erfüllt ist
    } else{                                                  // if-Anweisung Ende, else-Anweisung Anfang
        cout << "Leider Falsch :-(" << endl;                 // Ausgabe falls if-Bedingung nicht erfüllt ist
    }                                                        // else-Anweisung Ende

    cout << "Vielen Dank für Ihre Eingabe."  << endl;        // Ausgabe eines Textes
}
```

(**if-else**)-Anweisung

If_Else_0.cpp

```cpp
#include <iostream>                                          // Ein- und Ausgabebibliothek
using namespace std;                                         // Benutze den Namensraum std

int main(){                                                  // Hauptfunktion
    int zahl;                                                // Deklaration der Integer Variable 'zahl'

    cout << "Geben Sie bitte eine Fibonacci-Zahl im Bereich zwischen 100 und 500 ein: "; // Ausgabe eines Textes
    cin  >> zahl;                                            // Einlesen der Zahl mittels der Tastatur

    if( zahl == 144 || zahl == 233 || zahl == 377 ){         // if-Anweisung Anfang mit logischem 'oder'
        cout << "Richtig!" << endl;                          // Ausgabe falls if-Bedingung erfüllt ist
    } else{                                                  // if-Anweisung Ende, else-Anweisung Anfang
        cout << "Leider Falsch :-(" << endl;                 // Ausgabe falls if-Bedingung nicht erfüllt ist
    }                                                        // else-Anweisung Ende

    cout << "Vielen Dank für Ihre Eingabe."  << endl;        // Ausgabe eines Textes
}
```

Die untere Abbildung zeigt die Terminalausgabe, wobei beim ersten Versuch ein falscher Zahlenwert (311) eingegeben wurde. Danach wurde das Programm nochmals gestartet und ein richtiger Wert (377) eingegeben.

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ g++ If_Else_0.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ ./a.out
Geben Sie bitte eine Fibonacci-Zahl im Bereich zwischen 100 und 500 ein: 311
Leider Falsch :-(
Vielen Dank für Ihre Eingabe.
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ ./a.out
Geben Sie bitte eine Fibonacci-Zahl im Bereich zwischen 100 und 500 ein: 377
Richtig!
Vielen Dank für Ihre Eingabe.
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$
```

Die **switch**-Anweisung

Formale Struktur einer **switch**-Anweisung

```
switch ('Ganzzahlige Variable') {
   case Ganze Zahl Nr.1 :
      'Block von Anweisungen' ;
      break ;
   case Ganze Zahl Nr.2 :
      'Block von Anweisungen' ;
      break ;
    ....
   default :
      'Block von Anweisungen' ;
      break ;
}
```

Eine **switch**-Anweisung wählt unter einem Satz von Alternativen (**case**-Marken) aus. Der Ausdruck welcher direkt hinter dem **switch**-Befehl in runden Klammern steht, ist die 'Auswahl-Variable' der **switch**-Anweisung und der aktuelle Wert dieser Variable spezifiziert welche der **case**-Marken ausgewählt wird.

Switch_0.cpp

C++ Beispielprogramm

```
#include <iostream>                                                    // Ein- und Ausgabebibliothek
#include <cmath>                                                       // Bibliothek für mathematisches (e-Funktion, Betrag, ...)
using namespace std;                                                   // Benutze den Namensraum std

int main(){                                                            // Hauptfunktion
    double zahl;                                                       // Deklaration der Double Variable 'zahl'
    int auswahl;                                                       // Deklaration der Integer Variable 'auswahl' für switch

    cout << "Geben Sie bitte eine Gleitkommazahl ein: ";               // Ausgabe eines Textes
    cin  >> zahl;                                                      // Einlesen der Zahl mittels der Tastatur
    cout << "Möchten Sie die Zahl ..." << endl; ;                      // Ausgabe eines Textes
    cout << "... mal 25 nehmen, dann geben Sie 1 ein." << endl;                 // Ausgabe eines Textes
    cout << "... die Zahl hoch 5 nehmen, dann geben Sie 2 ein." << endl;        // Ausgabe eines Textes
    cout << "... den Sinus dieser Zahl berechnen, dann geben Sie 3 ein." << endl; // Ausgabe eines Textes
    cin  >> auswahl;                                                           // Einlesen der Variable 'auswahl'

    switch( auswahl ){                                                         // switch-Anweisung Anfang
        case 1:                                                                // switch-Fall Nr.1
            cout << "Der berechnete Wert beträgt: " << 25*zahl << endl;        // Anweisung Nr.1
            break;                                                             // switch-Fall Nr.1 Ende
        case 2:                                                                // switch-Fall Nr.2
            cout << "Der berechnete Wert beträgt: " << pow(zahl,5) << endl;    // Anweisung Nr.2
            break;                                                             // switch-Fall Nr.2 Ende
        case 3:                                                                // switch-Fall Nr.3
            cout << "Der berechnete Wert beträgt: " << sin(zahl) << endl;      // Anweisung Nr.3
            break;                                                             // switch-Fall Nr.3 Ende
        default:                                                               // switch-default
            cout << "Leider haben Sie die falsche Auswahl getroffen :-(" << endl; // Anweisung default
            break;                                                             // switch-default Ende
    }                                                                          // switch-Anweisung Ende

    cout << "Vielen Dank für Ihre Eingabe."  << endl;                          // Ausgabe eines Textes
}
```

Die 'Auswahl-Variable' sollte eine ganzzahlige Integer-Variable, oder ein char-Zeichen sein. Die einzelnen Werte der **case**-Marken legen die jeweilige Alternative und den auszuführenden 'Block von Anweisungen' fest. Am Ende einer jeden **case**-Marke steht ein **break**-Befehl, der ein Verlassen der **switch**-Auswahl bewirkt. Am Ende einer **switch**-Anweisung sollte zusätzlich eine **default**-Marke stehen, sodass nicht explizit betrachtete, potentiell mögliche Werte der 'Auswahl-Variable' auch behandelt werden können.

Die **switch**-Anweisung

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ g++ Switch_0.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ ./a.out
Geben Sie bitte eine Gleitkommazahl ein: 1.257
Möchten Sie die Zahl ...
... mal 25 nehmen, dann geben Sie 1 ein.
... die Zahl hoch 5 nehmen, dann geben Sie 2 ein.
... den Sinus dieser Zahl berechnen, dann geben Sie 3 ein.
4
Leider haben Sie die falsche Auswahl getroffen :-(
Vielen Dank für Ihre Eingabe.
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ ./a.out
Geben Sie bitte eine Gleitkommazahl ein: 1.257
Möchten Sie die Zahl ...
... mal 25 nehmen, dann geben Sie 1 ein.
... die Zahl hoch 5 nehmen, dann geben Sie 2 ein.
... den Sinus dieser Zahl berechnen, dann geben Sie 3 ein.
2
Der berechnete Wert beträgt: 3.13817
Vielen Dank für Ihre Eingabe.
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ ./a.out
Geben Sie bitte eine Gleitkommazahl ein: 1.257
Möchten Sie die Zahl ...
... mal 25 nehmen, dann geben Sie 1 ein.
... die Zahl hoch 5 nehmen, dann geben Sie 2 ein.
... den Sinus dieser Zahl berechnen, dann geben Sie 3 ein.
3
Der berechnete Wert beträgt: 0.951169
Vielen Dank für Ihre Eingabe.
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$
```

Switch_0.cpp

```cpp
#include <iostream>                                                    // Ein- und Ausgabebibliothek
#include <cmath>                                                       // Bibliothek für mathematisches (e-Funktion, Betrag, ...)
using namespace std;                                                   // Benutze den Namensraum std

int main(){                                                            // Hauptfunktion
    double zahl;                                                       // Deklaration der Double Variable 'zahl'
    int auswahl;                                                       // Deklaration der Integer Variable 'auswahl' für switch

    cout << "Geben Sie bitte eine Gleitkommazahl ein: ";               // Ausgabe eines Textes
    cin  >> zahl;                                                      // Einlesen der Zahl mittels der Tastatur
    cout << "Möchten Sie die Zahl ..." << endl; ;                      // Ausgabe eines Textes
    cout << "... mal 25 nehmen, dann geben Sie 1 ein." << endl;                 // Ausgabe eines Textes
    cout << "... die Zahl hoch 5 nehmen, dann geben Sie 2 ein." << endl;        // Ausgabe eines Textes
    cout << "... den Sinus dieser Zahl berechnen, dann geben Sie 3 ein." << endl; // Ausgabe eines Textes
    cin  >> auswahl;                                                           // Einlesen der Variable 'auswahl'

    switch( auswahl ){                                                         // switch-Anweisung Anfang
        case 1:                                                                // switch-Fall Nr.1
            cout << "Der berechnete Wert beträgt: " << 25*zahl << endl;        // Anweisung Nr.1
            break;                                                             // switch-Fall Nr.1 Ende
        case 2:                                                                // switch-Fall Nr.2
            cout << "Der berechnete Wert beträgt: " << pow(zahl,5) << endl;    // Anweisung Nr.2
            break;                                                             // switch-Fall Nr.2 Ende
        case 3:                                                                // switch-Fall Nr.3
            cout << "Der berechnete Wert beträgt: " << sin(zahl) << endl;      // Anweisung Nr.3
            break;                                                             // switch-Fall Nr.3 Ende
        default:                                                               // switch-default
            cout << "Leider haben Sie die falsche Auswahl getroffen :-(" << endl; // Anweisung default
            break;                                                             // switch-default Ende
    }                                                                          // switch-Anweisung Ende

    cout << "Vielen Dank für Ihre Eingabe."  << endl;                          // Ausgabe eines Textes
}
```

In dem Beispielprogramm der **switch**-Anweisung (rechte obere Abbildung) wird zunächst eine Gleitkommazahl vom Benutzer eingegeben und danach wird der Benutzer gefragt, welche mathematische Operation das Programm mit dieser Zahl machen soll. Der Benutzer kann dabei aus drei Alternativen auswählen, die als drei separate **case**-Marken in der **switch**-Anweisung implementiert sind. Mittels der **default**-Marke wird dann gewährleistet, dass eine falsch eingegebene Auswahl des Benutzers auch berücksichtigt wird.
Die untere Abbildung zeigt die Terminalausgabe, wobei beim ersten Versuch ein falscher 'Auswahl-Wert' (4) eingegeben wurde und die danach folgenden zwei Aufrufe des Programms zwei der drei **case**-Marken ausgewählt wurden.

C++ Funktionen

Funktionen in C++

C++ Funktionen sind kleine Unterprogramme, die Teilprobleme lösen. Funktionen sind ein wichtiges Werkzeug um den Quelltext eines Programms zu ordnen und wesentliche Algorithmen und zusammenhängende Anweisungsblöcke der main()-Hauptfunktion in einer zusammenhängenden Form auszulagern. C++ Funktionen werden ausserhalb der main()-Hauptfunktion definiert und vereinfachen somit das Verständnis und die Lesbarkeit des Quelltextes.

Deklaration und Definition von Funktionen

Im Grunde bedeutet die Definition einer Funktion nichts Anderes, als eine Code-Block (*Anweisungsblock*) mit einem Funktionsnamen zu verbinden. Die Definition einer C++ Funktion besteht aus einer *Deklaration* und einem *Anweisungsblock*.

- **Der Name der Funktion**:
  Hier sollte der Programmierer einen Namen wählen, der die im *Anweisungsblock* definierten Anweisungen präzise und kurz in einem Wort zusammen fasst.
- **Rückgabe Typ**:
  Welchen Datentyp gibt die Funktion an das Hauptprogramm zurück? Der "Rückgabe Typ" steht vor dem Funktionsnamen.
- **Argumentenliste**:
  Die "Argumentenliste" steht direkt hinter dem Funktionsnamen, im Aufrufoperator "(...)". Sie spezifiziert die Datentypen der Variablen, die die Funktion zur Berechnung ihrer Aufgabe benötigt.

Für die *Deklaration* einer C++ Funktion sind gewisse Angaben erforderlich (siehe nebenstehendes Frame) und gewisse zusätzliche Spezifikationen möglich (z.B. '**inline**' oder '**virtual**'). Um eine Funktion schliesslich zu definieren, muss man zusätzlich noch die zu erledigenden Anweisungen in einem Anweisungsblock zusammenfassen. Die allgemeine Definition einer C++ Funktion ist der formalen Definition einer mathematischen Funktion nicht unähnlich und man könnte sie auch wie folgt definieren: "Eine C++ Funktion ist eine Abbildung von dem Datenraum der Argumentenliste in den Datenraum des Rückgabetyps. Die dabei benutzte Abbildungsvorschrift findet sich in ihrem Anweisungsblock. Formal besitzt sie somit die folgende Struktur:"

'Rückgabe Typ' **Funktionsname** ('Argumentenliste') { 'Block von Anweisungen' }

C++ Funktionen

Die Definition einer C++ Funktion besteht aus einer *Deklaration* und einem *Anweisungsblock*.

```
'Rückgabe Typ' Funktionsname ('Argumentenliste') { 'Block von Anweisungen' }
```

Die allgemeine Definition einer C++ Funktion ist der formalen Definition einer mathematischen Funktion nicht unähnlich und man könnte sie auch wie folgt definieren: "Eine C++ Funktion ist eine Abbildung von dem Datenraum der Argumentenliste in den Datenraum des Rückgabetyps. Die dabei benutzte Abbildungsvorschrift findet sich in ihrem Anweisungsblock.

- **Der Name der Funktion**:
  Hier sollte der Programmierer einen Namen wählen, der die im *Anweisungsblock* definierten Anweisungen präzise und kurz in einem Wort zusammen fasst.
- **Rückgabe Typ**:
  Welchen Datentyp gibt die Funktion an das Hauptprogramm zurück? Der "Rückgabe Typ" steht vor dem Funktionsnamen.
- **Argumentenliste**:
  Die "Argumentenliste" steht direkt hinter dem Funktionsnamen, im Aufrufoperator "(...)". Sie spezifiziert die Datentypen der Variablen, die die Funktion zur Berechnung ihrer Aufgabe benötigt.

Für die *Deklaration* einer C++ Funktion sind gewisse Angaben erforderlich (siehe nebenstehende Abbildung) und gewisse zusätzliche Spezifikationen möglich (z.B. '**inline**' oder '**virtual**'). Um eine Funktion schließlich zu definieren, muss man zusätzlich noch die zu erledigenden Anweisungen in einem Anweisungsblock zusammenfassen.

Das untere Programm zeigt den Quelltext mit ausgelagerter 'print_wert' Funktion. Bei der Definition der Funktion wurden in der 'Argumentenliste' die Variablen 'arg_1' und 'arg_2' verwendet und als double- bzw. int-Datentyp deklariert. Im Anweisungsblock der Funktion wurde nun der gesamte Code-Block der **switch**-Anweisung mit den Textausgaben eingefügt ...

Switch_0_Funktion.cpp

```cpp
#include <iostream>                                     // Ein- und Ausgabebibliothek
#include <cmath>                                        // Bibliothek für mathematisches (e-Funktion, Be
using namespace std;                                    // Benutze den Namensraum std

void print_wert (double arg_1, int arg_2){              // Definition der Funktion 'print_wert'
    switch( arg_2 ){                                                    // switch-Anweisung Anfang
        case 1:                                                         // switch-Fall Nr.1
            cout << "Der berechnete Wert beträgt: " << 25*arg_1 << endl;// Anweisung Nr.1
            break;                                                      // switch-Fall Nr.1 Ende
        case 2:                                                         // switch-Fall Nr.2
            cout << "Der berechnete Wert beträgt: " << pow(arg_1,5) << endl;// Anweisung Nr.2
            break;                                                      // switch-Fall Nr.2 Ende
        case 3:                                                         // switch-Fall Nr.3
            cout << "Der berechnete Wert beträgt: " << sin(arg_1) << endl;// Anweisung Nr.3
            break;                                                      // switch-Fall Nr.3 Ende
        default:                                                        // switch-default
            cout << "Leider haben Sie die falsche Auswahl getroffen :-(" << endl; // Anweisung default
            break;                                                      // switch-default Ende
    }                                                                   // switch-Anweisung Ende
    cout << "Vielen Dank für Ihre Eingabe."  << endl;                   // Ausgabe eines Textes
}                                                       // Ende des Anweisungsblocks der Funktion 'print_wert'

int main(){                                             // Hauptfunktion
    double zahl;                                        // Deklaration der Double Variable 'zahl'
    int auswahl;                                        // Deklaration der Integer Variable 'auswahl' für switch

    cout << "Geben Sie bitte eine Gleitkommazahl ein: ";// Ausgabe eines Textes
    cin  >> zahl;                                       // Einlesen der Zahl mittels der Tastatur
    cout << "Möchten Sie die Zahl ..." << endl; ;       // Ausgabe eines Textes
    cout << "... mal 25 nehmen, dann geben Sie 1 ein." << endl;                  // Ausgabe eines Textes
    cout << "... die Zahl hoch 5 nehmen, dann geben Sie 2 ein." << endl;         // Ausgabe eines Textes
    cout << "... den Sinus dieser Zahl berechnen, dann geben Sie 3 ein." << endl;// Ausgabe eines Textes
    cin  >> auswahl;                                                            // Einlesen der Variable 'auswahl'

    print_wert(zahl, auswahl);                          // Aufruf der oben definierten Funktion
}
```

Switch_0.cpp

```cpp
#include <iostream>                                     // Ein- und Ausgabebibliothek
#include <cmath>                                        // Bibliothek für mathematisches (e-Funktion, Betrag, ...)
using namespace std;                                    // Benutze den Namensraum std

int main(){                                             // Hauptfunktion
    double zahl;                                        // Deklaration der Double Variable 'zahl'
    int auswahl;                                        // Deklaration der Integer Variable 'auswahl' für switch

    cout << "Geben Sie bitte eine Gleitkommazahl ein: ";// Ausgabe eines Textes
    cin  >> zahl;                                       // Einlesen der Zahl mittels der Tastatur
    cout << "Möchten Sie die Zahl ..." << endl; ;       // Ausgabe eines Textes
    cout << "... mal 25 nehmen, dann geben Sie 1 ein." << endl;                  // Ausgabe eines Textes
    cout << "... die Zahl hoch 5 nehmen, dann geben Sie 2 ein." << endl;         // Ausgabe eines Textes
    cout << "... den Sinus dieser Zahl berechnen, dann geben Sie 3 ein." << endl;// Ausgabe eines Textes
    cin  >> auswahl;                                                            // Einlesen der Variable 'auswahl'

    switch( auswahl ){                                                  // switch-Anweisung Anfang
        case 1:                                                         // switch-Fall Nr.1
            cout << "Der berechnete Wert beträgt: " << 25*zahl << endl; // Anweisung Nr.1
            break;                                                      // switch-Fall Nr.1 Ende
        case 2:                                                         // switch-Fall Nr.2
            cout << "Der berechnete Wert beträgt: " << pow(zahl,5) << endl;// Anweisung Nr.2
            break;                                                      // switch-Fall Nr.2 Ende
        case 3:                                                         // switch-Fall Nr.3
            cout << "Der berechnete Wert beträgt: " << sin(zahl) << endl;// Anweisung Nr.3
            break;                                                      // switch-Fall Nr.3 Ende
        default:                                                        // switch-default
            cout << "Leider haben Sie die falsche Auswahl getroffen :-(" << endl; // Anweisung default
            break;                                                      // switch-default Ende
    }                                                                   // switch-Anweisung Ende

    cout << "Vielen Dank für Ihre Eingabe."  << endl;                   // Ausgabe eines Textes
}
```

... und die Variablen auf 'arg_1' und 'arg_2' umbenannt (man hätte hier auch bei den Variablen 'zahl' und 'auswahl' bleiben können). Die Variablen 'arg_1' und 'arg_2' werden auch als *Lokale Variablen* bezeichnet, da ihr Gültigkeitsbereich auf die Funktion beschränkt ist.

Mathematische Funktionen in C++

Definition von mathematischen Funktionen in C++

In C++ hat das Wort Funktion eine allgemeinere Bedeutung als im Bereich der Mathematik und die in der Mathematik und Physik definierte Funktionen stellen eine echte Teilmenge der C++ Funktionen dar. In diesem Teilkapitel möchten wir weiter in den Themenbereich der C++ Funktionen einführen und uns hauptsächlich mit der Definition von mathematischen Funktionen in C++ befassen. Betrachten wir uns z.B. die Funktion $f(x) = x^2$. In der Mathematik definiert man diese Funktion als eine Abbildung von der Menge der reellen Zahlen $\mathbb{R}$ in die Menge der positiv reellen Zahlen $\mathbb{R}^+$ und man schreibt formal:

$$f: \underbrace{\mathbb{R}}_{\text{Definitionsmenge}} \rightarrow \underbrace{\mathbb{R}^+}_{\text{Bildmenge}} \quad \text{mit:} \quad f(x) = x^2 \ , \ x \in \mathbb{R}$$

Wir möchten nun diese mathematische Funktion in C++ formulieren und z.B. ihre Funktionswerte in einem Teilintervall ihrer Definitionsmenge $[a, b] \in \mathbb{R}$ ausgeben lassen. Die einfachste C++ Version der oben definierten mathematischen Funktion lautet:

```
double f (double x) { return x*x; }
```

Mathematische Funktionen in C++

Beispiel $f(x)=x^2$

Die Definitionsmenge der Funktion spiegelt sich in dem Datentyp der 'Argumentenliste' wider (hier nur ein Argument: double x). Die Bildmenge wird durch den 'Rückgabe Typ' der Funktion ausgedrückt (double f, eigentlich unsigned double, das gibt es aber nicht). Die Rückgabe des y-Wertes der Funktion wird mittels der Anweisung 'return x*x;' im Anweisungensblock der Funktion erreicht.

Funktion_Math_0.cpp

```
/* Definition der Funktion f(x)=x^2
 * und Ausgabe ihrer Funktionswerte im Telintervall [a,b]=[-3,3]
 * Ausgabe zum Plotten (Gnuplot oder Python) mittels: "./a.out > Funktion_Math_0.dat" */
#include <iostream>                                   // Ein- und Ausgabebibliothek

double f (double x) {                                 // Deklaration und Definition der Funktion f(x)
    double wert;                                      // Lokale double-Variable (nur im Bereich der Funktion gültig)
    wert = x*x;                                       // Eigentliche Definition der Funktion
    return wert;                                      // Rueckgabewert der Funktion f(x)
}                                                     // Ende der Funktion f(x)

int main(){                                           // Hauptfunktion
    int i;                                            // Deklaration des Laufindex als ganze Zahl
    const double a = -3;                              // Untergrenze des x-Intervalls [a,b]
    const double b = 3;                               // Obergrenze des x-Intervalls [a,b]
    const int N_xp=300;                               // Anzahl der Punkte in die das x-Intervall aufgeteilt wird
    const double dx = (b - a)/N_xp;                   // Abstand dx zwischen den aequidistanten Punkten des x-Intervalls
    double p, fp;                                     // Deklaration der Variablen des x- und f(x)-Wertes

    printf("# 0: Index i \n# 1: x-Wert \n# 2: y-Wert\n"); // Beschreibung der ausgegebenen Groessen

    for(i=0; i<=N_xp; ++i){                           // Schleifen Anfang
        p = a + i*dx;                                 // Festlegung des aktuellen x-Wertes
        fp = f(p);                                    // Berechnung des f(x)-Wertes
        printf("%4d %14.10f %14.10f \n",i, p, fp);    // Ausgabe der berechneten Werte
    }                                                 // Ende der for-Schleife
}
```

Bei komplizierteren Funktionen ist die Verwendung einer zusätzlichen lokalen Variable sinnvoll.
Im nebenstehenden Programm wird die gleiche Funktion definiert, der Allgemeinheit halber ist jedoch im Anweisungsblock der Funktion die lokale double Variable 'wert' verwendet worden, der dann der aktuelle Funktionswert übergeben wird. Das Programm gibt mittels einer **for**-Schleife 300 äquidistante Funktionswerte des Teilintervalls [−3,3] aus, die man sich dann z.B. mit einem Python Skript darstellen kann.

Mathematische Funktionen in C++

Beispiel f(x,y,z) = $x\ y^2\ z^2$

In gleicher Weise können auch mathematische Funktionen dargestellt werden, die eine kompliziertere Abbildungsstruktur aufweisen. Nehmen wir z.B. eine Funktion, die eine Abbildung von dem $\mathbb{R}^3$ in den reelwertigen Raum $\mathbb{R}$ darstellt:

$$f: \underbrace{\mathbb{R}^3}_{\text{Argumentenliste}} \rightarrow \underbrace{\mathbb{R}}_{\text{Rückgabetyp}} \quad \text{mit:} \quad f(x,y,z) = x \cdot y^2 \cdot z^2 \ , \ x,y,z \in \mathbb{R}$$

Die entsprechende C++ Funktion hätte dann das folgende Aussehen:

```
double f (double x, double y, double z) { return x*y*y*z*z; }
```

Wir werden in einer späteren Vorlesung auf die allgemeinere Verwendung von Funktionen noch genauer eingehen und gerade in den Unterpunkten, die sich mit der Objekt-orientierten Programmierung befassen, sind Klassen-Funktionen, Konstruktoren und das Überladen von Funktionen ein wichtiges Thema.

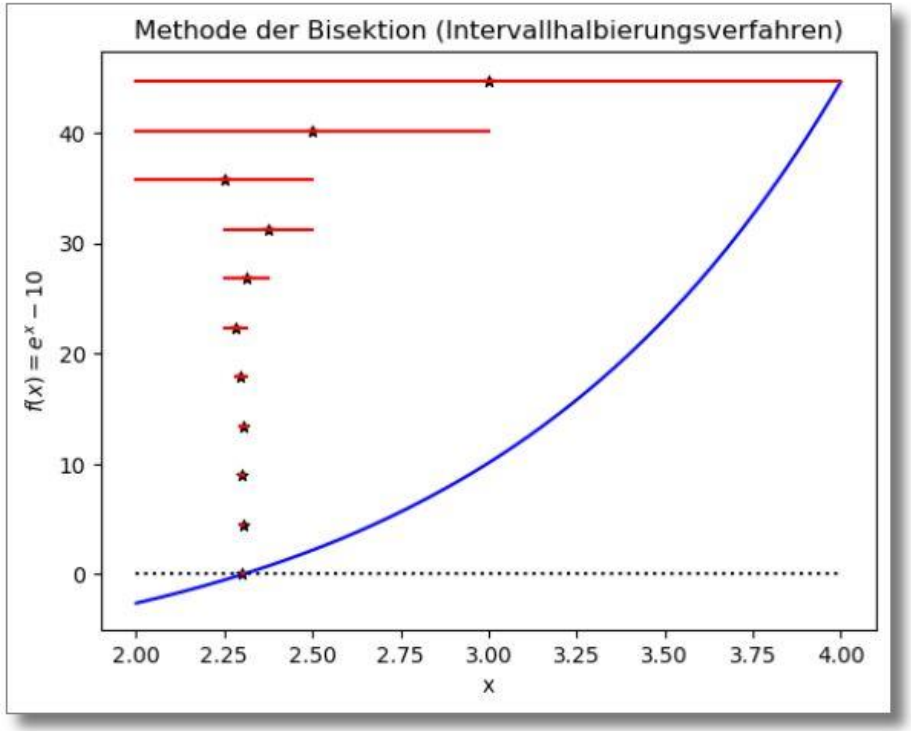
In der Vorlesung 5 werden wir auch auf die Übergabe von C++ Arrays (Vektoren) an Funktionen eingehen.

Nullstellensuche einer Funktion

Anwendungsbeispiel: Nullstellensuche einer Funktion

In diesem Unterpunkt möchten wir ein grundlegendes Problem der numerischen Mathematik behandeln, die Nullstellensuche einer Funktion. Wir betrachten im speziellen die Funktion $f(x) = e^x - 10$ und wollen berechnen, bei welchem x-Wert die Funktion Null wird ($f(x) = 0$). Bei der angegebenen Funktion ist das nicht schwierig und man erhält mittels einfacher analytischer Umformungen das Ergebnis $p = \ln(10) \approx 2.302585...$. Eine solche analytische Umformung ist jedoch nicht immer möglich und man kann dann eine Nullstelle auf numerischem Weg mittels unterschiedlicher Methoden berechnen. In diesem Unterpunkt betrachten wir zunächst die *Methode der Bisektion* (das *Intervallhalbierungsverfahren*) und danach die *Newton-Raphson Methode* zur numerischen Ermittlung der Nullstelle.

Der Algorithmus der Bisektion (Intervallhalbierungsverfahren)



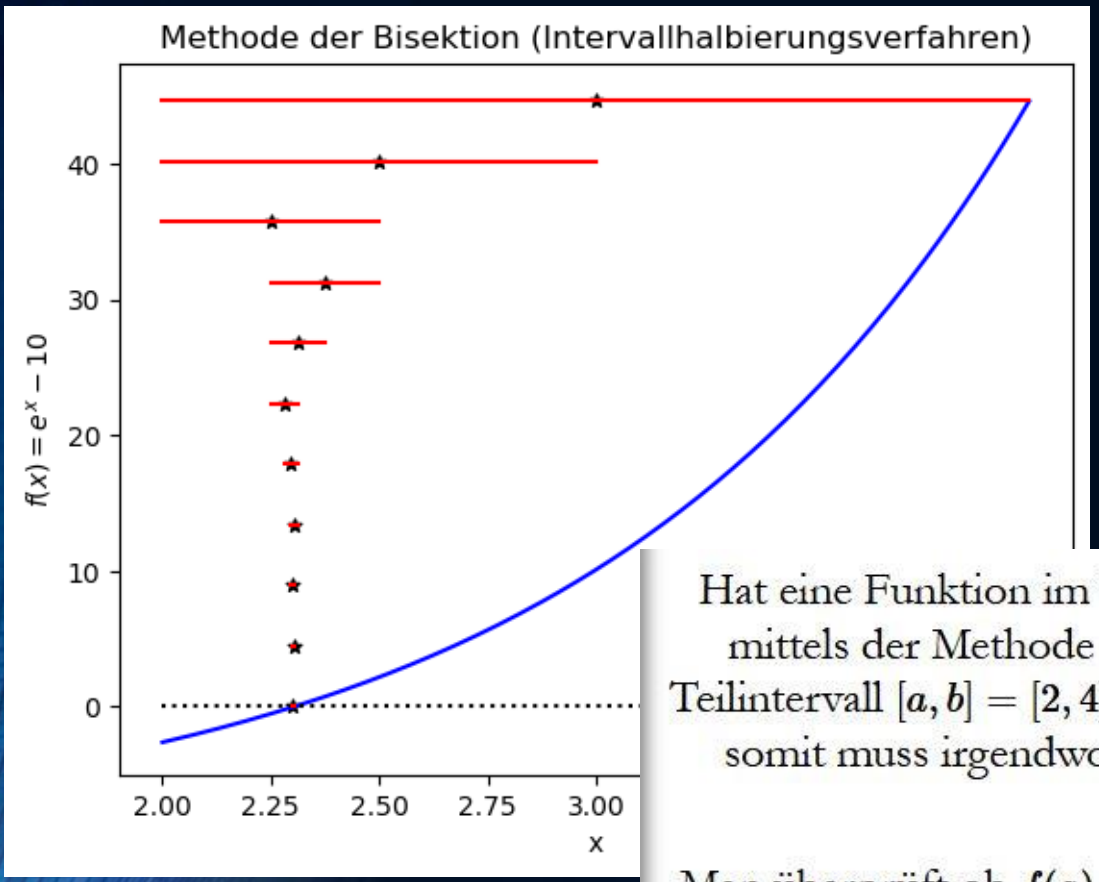
Hat eine Funktion im Teilintervall $[a, b] \in \mathbb{R}$ eine Nullstelle, so kann man diese Nullstelle numerisch mittels der Methode der Bisektion ermitteln. Betrachten wir z.B. die Funktion $f(x) = e^x - 10$ im Teilintervall $[a, b] = [2, 4]$. Es gilt $f(2) = e^2 - 10 \approx -2.61094 < 0$ und $f(4) = e^4 - 10 \approx 44.59815 > 0$ und somit muss irgendwo zwischen 2 und 4 eine Nullstelle existent sein. Man geht nun wie folgt vor:

Man überprüft ob $f(a) \cdot f(a + \frac{(b-a)}{2})$ größer oder kleiner Null ist. Ist der Wert größer Null, so ist man sicher, dass sich die Nullstelle nicht im Teilintervall $[a, a + \frac{(b-a)}{2}]$ befindet und kann das neue Teilintervall auf $[a + \frac{(b-a)}{2}, b]$ setzen. Ist der Wert hingegen kleiner Null, so befindet sich die Nullstelle im Teilintervall $[a, a + \frac{(b-a)}{2}]$, welches man dann als neues Teilintervall verwendet. Der spezielle Fall $f(a) \cdot f(a + \frac{(b-a)}{2}) = 0$ bedeutet, dass die Nullstelle gerade bei $p_0 = p = a + \frac{(b-a)}{2}$ ist und man kann den Algorithmus des Intervallhalbierungsverfahrens abrechen.

Die nebenstehende Abbildung veranschaulicht die Methode der Bisektion an unserem Beispiel. Die blaue Kurve zeigt die Funktion $f(x) = e^x - 10$ im Teilintervall $x \in [2, 4]$. Die roten, horizontalen Linien kennzeichnen die jeweiligen Teilintervalle, wobei die oberste Linie das ursprüngliche Teilintervall $[2, 4]$ kennzeichnet. Die Mitte des Intervalls ist mit einem schwarzen Stern markiert und dieser Wert stellt auch gleichzeitig den jeweiligen approximierten Wert p_i der Nullstelle dar. Beim Anfangsintervall $[a, b] = [2, 4]$ erhält man $p_0 = a + \frac{(b-a)}{2} = 2 + \frac{(4-2)}{2} = 3$ und da $f(a) \cdot f(a + \frac{(b-a)}{2}) = f(2) \cdot f(3) < 0$ ist, befindet sich die wirkliche Nullstelle im Teilintervall $[2, 3]$, welches wir bei dann bei der nächsten Iteration als neues Teilintervall $[a, b]$ definieren. Wie wir in der nebenstehenden Abbildung sehen, tastet sich das Intervallhalbierungsverfahren bei fortlaufender Iteration immer näher an den wirklichen Wert der Nullstelle heran.

Das entsprechende C++ Programm des betrachteten Beispiels der Methode der Bisektion (siehe Nullstelle_Bisektion_0.cpp) ist in dem folgenden Frame abgebildet:

Nullstellensuche einer Funktion



Der Algorithmus der Bisektion

Intervallhalbierungsverfahren

In diesem Anwendungsbeispiel möchten wir ein grundlegendes Problem der numerischen Mathematik behandeln, die Nullstellensuche einer Funktion. Wir betrachten eine Funktion f(x) und wollen berechnen, bei welchem x-Wert die Funktion Null wird (f(x)=o).

Hat eine Funktion im Teilintervall $[a, b] \in \mathbb{R}$ eine Nullstelle, so kann man diese Nullstelle numerisch mittels der Methode der Bisektion ermitteln. Betrachten wir z.B. die Funktion $f(x) = e^x - 10$ im Teilintervall $[a, b] = [2, 4]$. Es gilt $f(2) = e^2 - 10 \approx -2.61094 < 0$ und $f(4) = e^4 - 10 \approx 44.59815 > 0$ und somit muss irgendwo zwischen 2 und 4 eine Nullstelle existent sein. Man geht nun wie folgt vor:

Man überprüft ob $f(a) \cdot f(a + \frac{(b-a)}{2})$ größer oder kleiner Null ist. Ist der Wert größer Null, so ist man sicher, dass sich die Nullstelle nicht im Teilintervall $[a, a + \frac{(b-a)}{2}]$ befindet und kann das neue Teilintervall auf $[a + \frac{(b-a)}{2}, b]$ setzen. Ist der Wert hingegen kleiner Null, so befindet sich die Nullstelle im Teilintervall $[a, a + \frac{(b-a)}{2}]$, welches man dann als neues Teilintervall verwendet. Der spezielle Fall $f(a) \cdot f(a + \frac{(b-a)}{2}) = 0$ bedeutet, dass die Nullstelle gerade bei $p_0 = p = a + \frac{(b-a)}{2}$ ist und man kann den Algorithmus des Intervallhalbierungsverfahrens abrechen.

Nullstelle_Bisektion_0.cpp

C++ Programm zur Berechnung der Nullstelle einer Funktion f(x) mittels des Intervallhalbierungsverfahrens

```cpp
/* Mittels der Methode der Bisektion (Intervallhalbierungsverfahren)
 * wird die Nullstelle einer Funktion f(x)=0 approximiert
 * (hier speziell f(x)=exp(x)-10=0)
 * Ausgabe zum Plotten (Gnuplot oder Python) mittels: "./a.out > Nullstelle_Bisektion_0.dat" */
#include <iostream>                                     // Ein- und Ausgabebibliothek
#include <cmath>                                        // Bibliothek für mathematisches (e-Funktion, Betrag, ...)

double f (double x) {                                   // Deklaration und Definition der Funktion f(x)
    double wert;                                        // Lokale double-Variable (nur im Bereich der Funktion gültig)
    wert = exp(x) - 10;                                 // Eigentliche Definition der Funktion
    return wert;                                        // Rueckgabewert der Funktion f(x)
}                                                       // Ende der Funktion f(x)

int main(){                                             // Hauptfunktion
    int i;                                              // Deklaration des Laufindex als ganze Zahl
    double a = 2;                                       // Untergrenze des Intervalls [a,b] in dem f(x)=0 gelten soll
    double b = 4;                                       // Obergrenze des Intervalls [a,b] in dem f(x)=0 gelten soll
    const int N = 10;                                   // Anzahl der Iterationen im Intervallhalbierungsverfahren
    double p, fp;                                       // Deklaration der Variablen des approximierten x- und f(x)-Wertes der Nullstelle
    double fa=f(a);                                     // Deklaration und Initialisierung des Funktionswertes f(a) an der Untergrenze

    printf("# 0: Index i der Iteration \n# 1: Approximierter Wert der Nullstelle p_i \n");          // Beschreibung der ausgegebenen Groessen
    printf("# 2: Funktionswert f(p_i) \n# 3: Relativer Fehler zum wirklichen Wert |(p-p_i)/p| \n"); // Beschreibung der ausgegebenen Groessen
    printf("# 4: Untergrenze a \n# 5: Obergrenze b \n");                                            // Beschreibung der ausgegebenen Groessen

    for(i=0; i<=N; ++i){                                // Schleifen Anfang
        p = a + (b-a)/2;                                // Festlegung des x-Wertes in der Mitte des Intervalls [a,b] (x=p)
        fp = f(p);                                      // Berechnung des f(p)-Wertes in der Mitte des Intervalls [a,b]

        printf("%3d %14.10f %14.10f %14.10f %14.10f %14.10f \n",i, p, fp, fabs((log(10)-p)/log(10)), a, b); // Ausgabe der berechneten Werte

        if( fa*fp > 0 ){                                // Falls der berechnete Wert f(p) das gleiche Vorzeichen hat wie f(a), dann ...
            a = p;                                      // ... setze die Untergrenze auf a=p und ...
            fa = fp;                                    // ... lege den neuen Funktionswert an der Untergrenze fest: f(a)=f(p)
        } else if( fa*fp < 0 ){                         // Falls der berechnete Wert f(p) ein unterschiedliches Vorzeichen hat wie f(a), dann ...
            b = p;                                      // ... setze die Obergrenze auf b=p
        } else{                                         // Falls durch der berechnete Funktionswert gerade Null ist (f(p)=0 -> f(a)*f(p)=0), dann ...
            i = N;                                      // ... beende das Bisektion-Verfahren vorzeitig
        }                                               // else Ende
    }                                                   // Ende der for-Schleife des Bisektion-Verfahres
}
```

Nullstelle_Bisektion_0.cpp

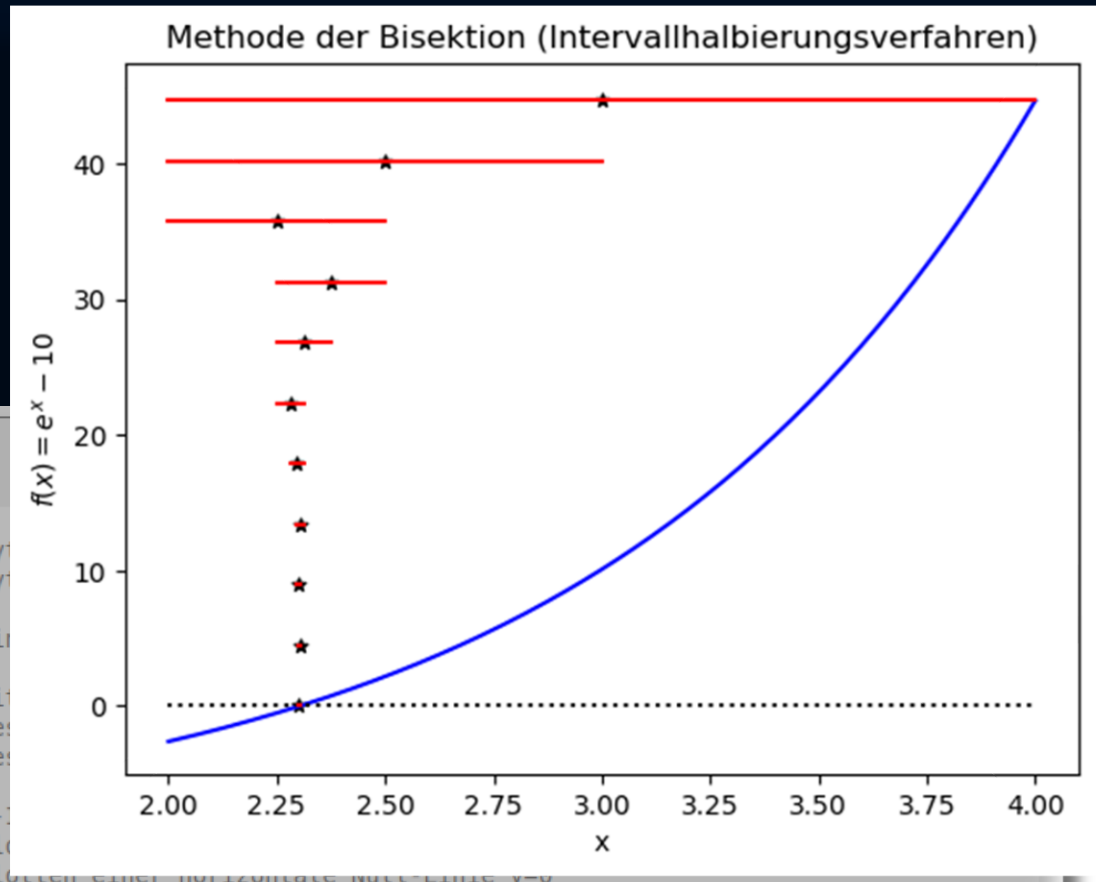
```
/* Mittels der Methode der Bisektion (Intervallhalb
 * wird die Nullstelle einer Funktion f(x)=0 approx
 * (hier speziell f(x)=exp(x)-10=0)
 * Ausgabe zum Plotten (Gnuplot oder Python) mittel
#include <iostream>
#include <cmath>

double f (double x) {
    double wert;
    wert = exp(x) - 10;
    return wert;
}

int main(){
    int i;
    double a = 2;
    double b = 4;
    const int N = 10;
    double p, fp;
    double fa=f(a);

    printf("# 0: Index i der Iteration \n# 1: Approximierter Wert der Nullstelle p_i \n");          // Beschreibung der ausgegebenen Groessen
    printf("# 2: Funktionswert f(p_i) \n# 3: Relativer Fehler zum wirklichen Wert |(p-p_i)/p| \n"); // Beschreibung der ausgegebenen Groessen
    printf("# 4: Untergrenze a \n# 5: Obergrenze b \n");                                           // Beschreibung der ausgegebenen Groessen

    for(i=0; i<=N; ++i){                                    // Schleifen Anfang
        p = a + (b-a)/2;                                    // Festlegung des x-Wertes in der Mitte des Intervalls [a,b] (x=p)
        fp = f(p);                                          // Berechnung des f(p)-Wertes in der Mitte des Intervalls [a,b]

        printf("%3d %14.10f %14.10f %14.10f %14.10f %14.10f \n",i, p, fp, fabs((log(10)-p)/log(10)), a, b); // Ausgabe der berechneten Werte

        if( fa*fp > 0 ){                                    // Falls der berechnete Wert f(p) das gleiche Vorzeichen hat wie f(a), dann ...
            a = p;                                          // ... setze die Untergrenze auf a=p und ...
            fa = fp;                                        // ... lege den neuen Funktionswert an der Untergrenze fest: f(a)=f(p)
        } else if( fa*fp < 0 ){                             // Falls der berechnete Wert f(p) ein unterschiedliches Vorzeichen hat wie f(a), dann ...
            b = p;                                          // ... setze die Obergrenze auf b=p
        } else{                                             // Falls durch der berechnete Funktionswert gerade Null ist (f(p)=0 -> f(a)*f(p)=0), dann ...
            i = N;                                          // ... beende das Bisektion-Verfahren vorzeitig
        }                                                   // else Ende
    }                                                       // Ende der for-Schleife des Bisektion-Verfahres
}
```

Terminalausgabe

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ g++ Nullstelle_Bisektion_0.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ ./a.out
# 0: Index i der Iteration
# 1: Approximierter Wert der Nullstelle p_i
# 2: Funktionswert f(p_i)
# 3: Relativer Fehler zum wirklichen Wert |(p-p_i)/p|
# 4: Untergrenze a
# 5: Obergrenze b
  0   3.0000000000  10.0855369232   0.3028834457   2.0000000000   4.0000000000
  1   2.5000000000   2.1824939607   0.0857362048   2.0000000000   3.0000000000
  2   2.2500000000  -0.5122641636   0.0228374157   2.0000000000   2.5000000000
  3   2.3750000000   0.7510131861   0.0314493945   2.2500000000   2.5000000000
  4   2.3125000000   0.0996422255   0.0043059894   2.2500000000   2.3750000000
  5   2.2812500000  -0.2110910987   0.0092657132   2.2500000000   2.3125000000
  6   2.2968750000  -0.0569382140   0.0024798619   2.2812500000   2.3125000000
  7   2.3046875000   0.0210461861   0.0009130638   2.2968750000   2.3125000000
  8   2.3007812500  -0.0180221705   0.0007833991   2.2968750000   2.3046875000
  9   2.3027343750   0.0014929315   0.0000648324   2.3007812500   2.3046875000
 10   2.3017578125  -0.0082693839   0.0003592834   2.3007812500   2.3027343750
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$
```

Visualisierung mit einem Python Skript

Um die jeweiligen Teilintervalle als diskrete horizontale Linien darstellen zu können, definieren wir die Liste 'y_interval', welche gerade 'N+1' äquidistante Werte im Bereich [0,f(4)] beinhaltet. Die einzelnen Resultate der Bisektions-Iteration werden schließlich mittels einer Python **for**-Schleife realisiert und die einzelnen Teilintervalle als horizontale rote Linien und der approximierte Nullstellenwert p_i als Stern mittels der Funktion 'plt.scatter(...)' geplottet.



PythonPlot_Bisektion_0.py

```python
# Python Programm zum Plotten der berechneten Daten von (Nullstelle_Bisektion_0.cpp)
import matplotlib.pyplot as plt
import numpy as np

data = np.genfromtxt("./Nullstelle_Bisektion_0.dat")

plt.title(r'Methode der Bisektion (Intervallhalbierungsverfahren)')
plt.ylabel(r'$f(x)=e^x - 10$')
plt.xlabel('x')

x_interval=np.linspace(data[0,4], data[0,5], 200)
plt.plot(x_interval,np.exp(x_interval)-10, color="blue")
plt.plot([data[0,4],data[0,5]],[0,0], color="black", linestyle=":")

y_interval=np.linspace(np.exp(x_interval[-1])-10, 0, int(data[-1,0])+1)

for i in data[:,0]:
    index=int(i)
    plt.plot([data[index,4],data[index,5]],[y_interval[index],y_interval[index]], color="red")
    plt.scatter(data[index,1],y_interval[index], marker='*', color="black", s=25)

plt.savefig("Nullstelle_Bisektion_0.png", bbox_inches="tight")
plt.show()
```

Die Newton-Raphson Methode der Nullstellensuche

Die Newton-Raphson Methode

Die *Newton-Raphson Methode* der Nullstellenermittlung ist ein sehr effektives Verfahren zur Ermittlung einer Nullstelle. Im Gegensatz zur Methode der Bisektion hat dieses Verfahren jedoch den Nachteil, dass man neben der Funktion $f(x)$ zusätzlich auch noch die erste Ableitung der Funktion ($f'(x) = \frac{df(x)}{dx}$) im Teilintervall $[a, b]$ benötigt. Die *Newton-Raphson Methode* basiert auf dem Prinzip der Taylorreihenentwicklung der Funktion, wobei man nur die ersten beiden Terme der Reihenentwicklung berücksichtigt und alle nicht-linearen Terme vernachlässigt. Man nimmt hierbei an, dass die Funktion im Teilintervall $[a, b]$ eine Nullstelle bei $x = p \in [a, b]$ hat ($f(p) = 0$) und rät im 0-ten Schritt des Algorithmus einen x-Wert, der nahe dieser Nullstelle liegen soll. Wir bezeichnen diesen geratenen Wert im Folgenden mit p_0 und entwickeln unsere Funktion um diesen Wert in eine Taylorreihe

$$f(x) = f(p_0) + (x - p_0) \cdot f'(p_0) + \frac{(x - p_0)^2}{2} \cdot f''(p_0) + \dots \quad .$$

Betrachtetet man nun die taylorentwickelte Funktion bei $x = p$ und nimmt an, dass der Wert von p_0 nahe der wirklichen Nullstelle ist, so erhält man:

$$0 = f(p) = f(p_0) + (p - p_0) \cdot f'(p_0) \underbrace{+ \frac{(p - p_0)^2}{2} \cdot f''(p_0) + \dots}_{\approx 0} \approx f(p_0) + (p - p_0) \cdot f'(p_0)$$

$$\Longrightarrow p \approx p_0 - \frac{f(p_0)}{f'(p_0)}$$

In der *Newton-Raphson Methode* benutzt man nun diesen neu approximierten Wert der Nullstelle und nimmt diesen für eine neue Iteration. Der Algorithmus benutzt somit die folgende Gleichung um die n-te Approximation der Nullstelle p_n mithilfe der (n-1)-ten Approximation zu berechnen:

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \quad \text{mit: } n \geq 1$$

Die Newton-Raphson Methode der Nullstellensuche

Die *Newton-Raphson Methode* der Nullstellenermittlung ist ein sehr effektives Verfahren zur Ermittlung einer Nullstelle. Im Gegensatz zur Methode der Bisektion hat dieses Verfahren jedoch den Nachteil, dass man neben der Funktion $f(x)$ zusätzlich auch noch die erste Ableitung der Funktion ($f'(x) = \frac{df(x)}{dx}$) im Teilintervall $[a, b]$ benötigt. Die *Newton-Raphson Methode* basiert auf dem Prinzip der Taylorreihenentwicklung der Funktion, wobei man nur die ersten beiden Terme der Reihenentwicklung berücksichtigt und alle nicht-linearen Terme vernachlässigt. Man nimmt hierbei an, dass die Funktion im Teilintervall $[a, b]$ eine Nullstelle bei $x = p \in [a, b]$ hat ($f(p) = 0$) und rät im 0-ten Schritt des Algorithmus einen x-Wert, der nahe dieser Nullstelle liegen soll. Wir bezeichnen diesen geratenen Wert im Folgenden mit p_0 und entwickeln unsere Funktion um diesen Wert in eine Taylorreihe

$$f(x) = f(p_0) + (x - p_0) \cdot f'(p_0) + \frac{(x - p_0)^2}{2} \cdot f''(p_0) + \dots \quad .$$

Betrachtetet man nun die taylorentwickelte Funktion bei $x = p$ und nimmt an, dass der Wert von p_0 nahe der wirklichen Nullstelle ist, so erhält man:

$$0 = f(p) = f(p_0) + (p - p_0) \cdot f'(p_0) \underbrace{+ \frac{(p - p_0)^2}{2} \cdot f''(p_0) + \dots}_{\approx 0} \approx f(p_0) + (p - p_0) \cdot f'(p_0)$$

$$\Longrightarrow p \approx p_0 - \frac{f(p_0)}{f'(p_0)}$$

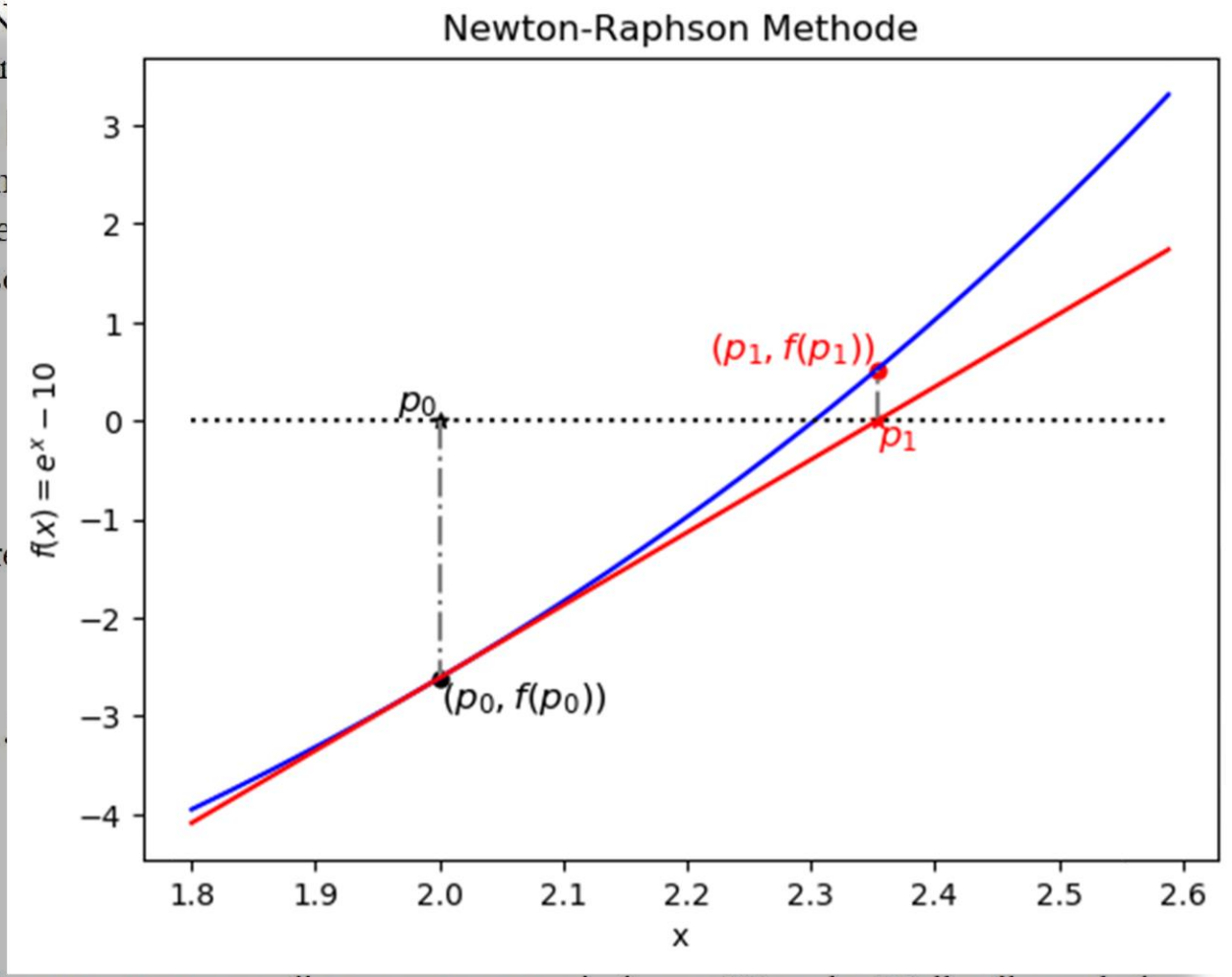
In der *Newton-Raphson Methode* benutzt man nun diesen neu approximierten Wert der Nullstelle und nimmt diesen für eine neue Iteration. Der Algorithmus benutzt somit die folgende Gleichung um die n-te Approximation der Nullstelle p_n mithilfe der (n-1)-ten Approximation zu berechnen:

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \quad \text{mit: } n \geq 1$$

Die Newton-Raphson Methode der Nullstellensuche

Die *Newton-Raphson Methode* der N
der Bisektion hat dieses Verfahre
$(f'(x) = \frac{df(x)}{dx})$ im Teilintervall
wobei man nur die ersten beiden
an, dass die Funktion im Teilinte
der nahe dieser Nullstelle liegen s

ullstelle. Im Gegensatz zur Methode
och die erste Ableitung der Funktion
rreihenentwicklung der Funktion,
vernachlässigt. Man nimmt hierbei
hritt des Algorithmus einen x-Wert,
eln unsere Funktion um diesen Wert



Betrachtetet man nun die taylor

r wirklichen Nullstelle ist, so erhält

$0 =$ $\cdot f'(p_0)$

In der *Newton-Raphson Methode* benutzt man nun diesen neu approximierten Wert der Nullstelle und nimmt diesen für eine neue Iteration. Der Algorithmus benutzt somit die folgende Gleichung um die n-te Approximation der Nullstelle p_n mithilfe der (n-1)-ten Approximation zu berechnen:

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \quad \text{mit: } n \geq 1$$

C++ Programm zur Berechnung der Nullstelle einer Funktion f(x) mittels der Newton-Raphson Methode

Nullstelle_NewtonRaphson_1.cpp

```
/* Mittels der Newton-Raphson Methode
 * wird die Nullstelle einer Funktion f(x)=0 approximiert
 * (hier speziell f(x)=exp(x)-10 = 0)
 * Ausgabe zum Plotten (Gnuplot oder Python) mittels: "./a.out > Nullstelle_NewtonRaphson_1.dat" */
#include <iostream>                                     // Ein- und Ausgabebibliothek
#include <cmath>                                        // Bibliothek für mathematisches (e-Funktion, Betrag, ...)

int main(){                                             // Hauptfunktion
    int i;                                              // Deklaration des Laufindex als ganze Zahl
    double p=2;                                         // Deklaration des approximierten x-Wertes der Nullstelle und Start-Initialisierung
    const int N=10;                                     // Anzahl der Iterationen in der Newton-Raphson Methode
    double fp, fp_s;                                    // Deklaration der Variablen des approximierten f(x)-Wertes der Nullstelle und dessen Ableitung f'(x)

    printf("# 0: Index i der Iteration i \n# 1: Approximierter Wert der Nullstelle p_i \n");        // Beschreibung der ausgegebenen Groessen
    printf("# 2: Funktionswert f(p_i) \n# 3: Relativer Fehler zum wirklichen Wert |(p-p_i)/p| \n"); // Beschreibung der ausgegebenen Groessen
    printf("# 4: Ableitung von f: f'(p_i) \n");                                                     // Beschreibung der ausgegebenen Groessen

    for(i=0; i<=N; ++i){                                // For-Schleife der Newton-Raphson Methode
        fp=exp(p)-10;                                   // Berechnung des f(p)-Wertes
        fp_s=exp(p);                                    // Berechnung der Ableitung (f'(p)-Wert)

        printf("%3d %20.14f %20.14f %20.14f %20.14f \n",i, p, fp, fabs((log(10)-p)/log(10)), fp_s); // Ausgabe der berechneten Werte

        p = p - fp/fp_s;                                // Berechnung des neuen p-Wertes mittels der Newton-Raphson Methode
    }                                                   // Ende der For-Schleife der Newton-Raphson Methode
}
```

C++ Programm zur Berechnung der Nullstelle einer Funktion f(x) mittels der Newton-Raphson Methode

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ g++ Nullstelle_NewtonRaphson_1.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$ ./a.out
# 0: Index i der Iteration i
# 1: Approximierter Wert der Nullstelle p_i
# 2: Funktionswert f(p_i)
# 3: Relativer Fehler zum wirklichen Wert |(p-p_i)/p|
# 4: Ableitung von f: f'(p_i)
  0     2.00000000000000    -2.61094390106935     0.13141103619350     7.38905609893065
  1     2.35335283236613     0.52078508488818     0.02204814906800    10.52078508488818
  2     2.30385224085264     0.01267951029657     0.00055031532275    10.01267951029657
  3     2.30258589548690     0.00000802493175     0.00000034851824    10.00000802493175
  4     2.30258509299437     0.00000000000322     0.00000000000014    10.00000000000322
  5     2.30258509299405    -0.00000000000000     0.00000000000000    10.00000000000000
  6     2.30258509299405    -0.00000000000000     0.00000000000000    10.00000000000000
  7     2.30258509299405    -0.00000000000000     0.00000000000000    10.00000000000000
  8     2.30258509299405    -0.00000000000000     0.00000000000000    10.00000000000000
  9     2.30258509299405    -0.00000000000000     0.00000000000000    10.00000000000000
 10     2.30258509299405    -0.00000000000000     0.00000000000000    10.00000000000000
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V3$
```

Terminalausgabe

Nullstelle_NewtonRaphson_1.cpp

```cpp
/* Mittels der Newton-Raphson Methode
 * wird die Nullstelle einer Funktion f(x)=0 appro
 * (hier speziell f(x)=exp(x)-10 = 0)
 * Ausgabe zum Plotten (Gnuplot oder Python) mitte
#include <iostream>
#include <cmath>

int main(){                                          // Hauptfunktion
    int i;                                           // Deklaration des Laufindex als ganze Zahl
    double p=2;                                      // Deklaration des approximierten x-Wertes der Nullstell
    const int N=10;                                  // Anzahl der Iterationen in der Newton-Raphson Methode
    double fp, fp_s;                                 // Deklaration der Variablen des approximierten f(x)-Wer

    printf("# 0: Index i der Iteration i \n# 1: Approximierter Wert der Nullstelle p_i \n");        // Beschreibung d
    printf("# 2: Funktionswert f(p_i) \n# 3: Relativer Fehler zum wirklichen Wert |(p-p_i)/p| \n"); // Beschreibung der ausgegebenen Groessen
    printf("# 4: Ableitung von f: f'(p_i) \n");                                                     // Beschreibung der ausgegebenen Groessen

    for(i=0; i<=N; ++i){                             // For-Schleife der Newton-Raphson Methode
        fp=exp(p)-10;                                // Berechnung des f(p)-Wertes
        fp_s=exp(p);                                 // Berechnung der Ableitung (f'(p)-Wert)

        printf("%3d %20.14f %20.14f %20.14f %20.14f \n",i, p, fp, fabs((log(10)-p)/log(10)), fp_s); // Ausgabe der berechneten Werte

        p = p - fp/fp_s;                             // Berechnung des neuen p-Wertes mittels der Newton-Raphson Methode
    }                                                // Ende der For-Schleife der Newton-Raphson Methode
}
```

Man erkennt deutlich die schnellere Konvergenz des Verfahrens im Vergleich zum Intervallhalbierungsverfahren!

Übungsblatt Nr. 4

Aufgabe 1 *(7.5 Punkte)*

Im Intervall $x \in [0.25, 2.5]$ hat die Funktion $g(x) = x^3$ mit der Funktion $h(x) = \ln(x) + 5$ einen Schnittpunkt. Berechnen Sie diesen Schnittpunkt auf nummerischem Wege mittels der Methode der Bisektion und stellen sie die ersten 11 Schritte des Intervallhalbierungsverfahrens grafisch mittels eines Python Skriptes dar. Bemerkung: Der Literaturwert des x-Wertes des Schnittpunktes beträgt $x = 1.7729093296693715...$

Aufgabe 2 *(7.5 Punkte)*

Erstellen Sie ein Plot-Programm für die folgenden Funktionen

$$f_1(x) = e^x - 10$$

$$f_2(x) = x^2 - 10\,x + 8$$

$$f_3(x) = x^3 - 8\,x^2 + 2\,x + 3$$

$$f_4(x) = 10\,e^{x/5} \cdot \sin(3\,x)$$

Erstellen Sie dafür ein C++ Programm, welches zunächst mittels einer Benutzereingabe (1,2,3 oder 4) abfragt, welche Funktion geplottet werden soll und dann die (x-y)-Wertetabelle (200 Punkte) im Intervall $x \in [0, 10]$ im Terminal ausgibt. Benutzen Sie bitte bei der Plot-Auswahl eine **switch**-Anweisung, wobei bei einer falschen Eingabe einfach die Funktion $f(x) = 0$ geplottet werden soll. Die Datentabelle leiten Sie dann in eine Textdatei um und lesen diese Daten danach in ein Python Skript ein, mit welchem Sie dann die Datenpunkte visualisieren und als Bild ausgeben lassen. Bemerkung: Gerne können Sie auch die Ausführung des C++ und Python Programms zusammengefügt in einem Shell Skript (z.B. A4_2.sh) ausführen. (Man hätte natürlich das gesamte Plot-Programm auch ausschließlich mittels eines Python Skriptes schreiben können. Bitte trennen Sie jedoch die Berechnung der (x-y)-Wertetabelle und die Visualisierung dieser Daten wie oben angegeben).

Aufgabe 3 *(5 Punkte)*

Berechnen Sie die folgenden Ausdrücke mittels eines C++ Programms und lassen Sie sich die berechneten Werte (falls diese Gleitkommazahlen sind) auf 15 Stellen genau ausgeben.

$$\prod_{k=1}^{11} k^2 \quad , \quad \sum_{k=0}^{20} \sum_{\substack{i=0 \\ i \neq k}}^{10} \frac{k + i^3}{(k-i)^2} \quad , \quad \text{Bestimmen Sie N} \quad : \quad \sum_{k=0}^{N} \sum_{\substack{i=0 \\ i \neq k}}^{250} \left(k + i^2\right) = 672013120$$

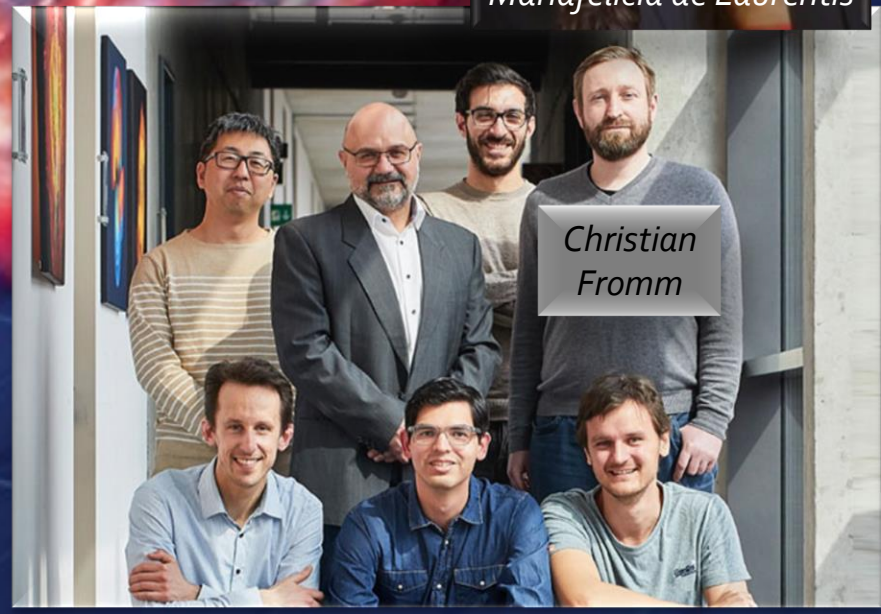
Pressekonferenz am 12. Mai 2022, 15.00 Uhr

Neue Erkenntnisse über das Schwarze Loch in unserer Milchstraße

Mariafelicia de Laurentis

Die Europäische Südsternwarte (ESO) und das Event Horizon Telescope (EHT) Projekt halten eine Pressekonferenz ab, bei der neue Ergebnisse vom EHT zur Milchstraße präsentiert werden.

- **Wann:** Am 12. Mai um 15:00 Uhr MESZ
- **Wo:** Eridanus-Auditorium, ESO-Hauptsitz, Garching bei München, und online
- **Was:** Eine Pressekonferenz, auf der bahnbrechende Ergebnisse des EHT zur Milchstraße präsentiert werden
- **Wer:** Der ESO-Generaldirektor wird die einleitenden Worte sprechen. EHT-Projektleiter Huib Jan van Langevelde und Anton Zensus, Gründungs-Vorsitzender der EHT-Kollaboration, werden ebenfalls sprechen. Eine Runde von EHT-Forschenden werden die Ergebnisse erläutern und Fragen beantworten. Diese Runde besteht aus
  - Thomas Krichbaum, Max-Planck-Institut für Radioastronomie, Deutschland
  - Sara Issaoun, Center for Astrophysics | Harvard & Smithsonian, US und Radboud University, Niederlande
  - José L. Gómez, Instituto de Astrofísica de Andalucía (CSIC), Spanien
  - Christian Fromm, Universität Würzburg, Deutschland
  - Mariafelicia de Laurentis, University of Naples "Federico II" und the National Institute for Nuclear Physics (INFN), Italien



Christian Fromm