Abgabe 1 Zetafunktion

Wayne Ströbel, Silas Müller

29/4/2021

Riemannsche Zeta Funktion

Die Riemannsche Zeta Funktion spielt eine wichtige Rolle für viele Anwendungen in der Physik und Mathematik. Sie ist wie folgt definiert:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s}, \qquad s \in \mathbb{C}\backslash\{0\} \tag{1}$$

Wir wollen nun im folgenden $\zeta(2)$ numerisch auf die 15. Dezimalstelle genau berechnen. Hierfür haben wir zwei verscheidene Methoden benutzt. Der Analytisch bestimmbare Konvergenzwert liegt hierfür bei $\frac{\pi^2}{6}$.

Naiver Ansatz

Zuerst hatten wir es auf die Naive Methode versucht. Dabei haben wir die Formel einfach in C programmiert und die Schrittzahl variiert:

```
  //gcc -Wall -pedantic Zeta_slow.c -o Zeta_slow
 //./Zeta_slow
//Silas Mueller, Wayne Stroebel
#include <stdio.h>
#include <math.h>
#include <stdlib.h>


double zetafunktion(double s,int genauigkeit, double* z)
{

    double x = 0;
    for (long int i = genauigkeit;(i > 0); i--)
    {
        x += 1/(pow(i, s));
        if (i < 100)
        {
            z[i] = x;
        }
    }
    return x;
}

int main(int argc, char const *argv[])
```

```
{
    //Wert fuer Zeta(s)
    double s = 2;
    //Genauigkeit if Form von 10^genauigkeit
    int genauigkeit = 9;
    int N = pow(10, genauigkeit);
    double z[100];
    int y[100];
    //Laufwert fuer Plot
    for (int i = 1; i < 100; i++)
    {
        y[i] = i;
    }

    printf("Zeta(2) = %.15f\n",zetafunktion(s,N,z));


    //oeffne die Datei Zeta_slow_data.txt
    FILE* fp;
    fp = fopen("Zeta_slow_data.txt", "w");
    if(fp == NULL) {
        printf("Datei konnte nicht geoeffnet werden!\n");
        return 1;
    }

    //schreibe zwischenergebnisse in Zeta_Slow.txt
    for (unsigned int i = 1; i < 100; i++) {
        fprintf(fp, "%i %.16f \n",y[i],z[i]);
    }
    fclose(fp);

    return 0;
}
```

Unser Programm berechnet für jeden Schritt den N-ten Wert der Summe, und addiert diesen auf einen Gesamtwert. Die Werte nach jedem Schritt speichern wir für die ersten 100 Schritte in einer externen .txt Datei, damit das Konvergenzverhalten später geplottet werden kann. Die berechneten Werte sind in der Zeta_slow_data.txt Datei wiederzufinden. Dieser Ansatz liefert uns nach einer Rechenzeit von ca. 1 Minute eine Genauigeikt von 8 Nachkommastellen mit 10^9 Schritten. Diese Methode ist also nicht ausreichend, um auf das gewollte Ergebnis zu kommen.

Borwein Methode

Die Borwein Methode nutzt einen anderen Ansatz. Hierbei verwenden wir abgebrochene alternierende Summen. Die Formel dafür sieht wie folgt aus:

$$\zeta(s) = \frac{1}{1-2^{1-s}} \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k^s} \tag{2}$$

Die Implementierung in C sieht bei uns folgendermaßen aus:

```
//gcc -Wall -pedantic Zeta_Borwein.c -o Zeta_Borwein -lm
```

```
 //./Zeta_Borwein
//Silas Mueller, Wayne Stroebel
#include <stdio.h>
#include <math.h>

double zetafunktion(double s,long int genauigkeit, double* z)
{

    double x;
    double a = 1 / (1-pow(2, 1 - s));
    for (size_t i = genauigkeit;(i > 0); i--)
    {
        x += pow(-1, i-1)/(pow(i, s));
        if (i < 100)
        {
            z[i] = a*x;
        }
    }
    x = a*x;
    return x;
}

int main(int argc, char const *argv[])
{
    //Wert fuer Zeta(s)
    double s = 2;
    //Genauigkeit if Form von 10^genauigkeit
    int genauigkeit = 8;
    long int N = pow(10, genauigkeit);
    double z[100];
    int y[100];
    //Laufwert fuer Plot
    for (int i = 1; i < 100; i++)
    {
        y[i] = i;
    }

    printf("Zeta(2) = %.15f\n",zetafunktion(s,N,z));

    //oeffne die Datei Zeta_Borwein_data.txt
    FILE* fp;
    fp = fopen("Zeta_Borwein_data.txt", "w");
    if(fp == NULL) {
        printf("Datei konnte nicht geoeffnet werden!\n");
        return 1;
    }

    //schreibe zwischenergebnisse in Zeta_Slow.txt
    for (unsigned int i = 1; i < 100; i++) {
        fprintf(fp, "%i %.16f \n",y[i],z[i]);
    }
    fclose(fp);
    return 0;
}
```

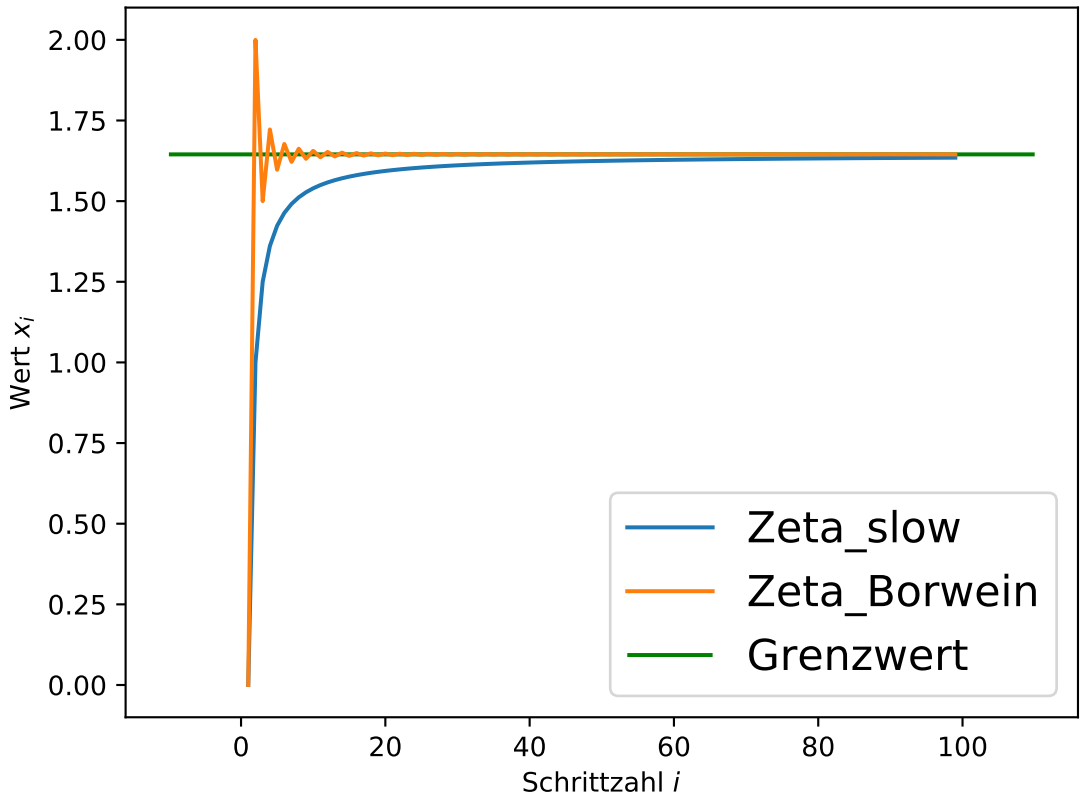
Wir berechnen wieder die einzelnen Summen und multiplizieren das Ergebnis mit dem Vorfaktor $a = \frac{1}{1-2^{1-s}}$, um auf unseren Wert zu kommen. Mit dieser Methode erhalten wir nach nur wenigen Sekunden und wieder einer Schrittzahl von 10^9 das Ergebnis genau auf 15 Nachkommastellen. Hierbei ist wichtig zu erwähnen, dass wir beide Methoden Rückwärts laufen lassen haben, also von $N, N-1, N-2, ..., 2, 1$. Dies vermindert floating-point-fehler und reduziert die notwendige Rechenzeit drastisch.

Konvergenz Plotten

Um das Konvergenzverhalten zu plotten haben wir im Code der beiden Methoden die ersten (bzw letzten, da wir von hinten anfangen) 100 Werte in eine .txt ausgeben lassen, und haben diese mithilfe von Python plotten lassen:

```
##python3 convergenzplot.py
##Silas Mueller, Wayne Stroebel
##bitte zuerst Zeta_Borwein.c und Zeta_slow.c ausführen um die Daten.txt files zu erstellen

import matplotlib.pyplot as plt
import numpy as np

X1, Y1 = [], []
X2, Y2 = [], []
for line in open('Zeta_slow_data.txt', 'r'):
  values1 = [float(s) for s in line.split()]
  X1.append(values1[0])
  Y1.append(np.pi**2 / 6 - values1[1])

for line in open('Zeta_Borwein_data.txt', 'r'):
    values2 = [float(s) for s in line.split()]
    X2.append(values2[0])
    Y2.append(np.pi**2 / 6 - values2[1])

plt.figure()
plt.xlabel("Schrittzahl $i$")
plt.ylabel("Wert $x_{i}$")
plt.plot(X1, Y1, label=r'Zeta_slow')
plt.plot(X2, Y2, label=r'Zeta_Borwein')
plt.hlines(np.pi**2 / 6, xmin=-10, xmax=110, label=r'Grenzwert',colors='green')
plt.legend(loc='best',prop={'size':16})
plt.savefig("Konvergenzverhalten.pdf")
plt.show()
```

Dies liefert uns den folgenden Plot:



Da wir die letzten hundert Ergebnisse genommen haben, mussten wir den Graphen etwas anpassen, hierzu haben wir die Werte des Codes von dem Konvergenzwert abgezogen.

Konvergenzordnung der Reihengleichung (1)

Mögliche bessere Methoden zur Berechnung von $\zeta(s)$

Es ist durchaus sinnvoll, auch noch bessere Methoden der Berechnung anzugucken, allerdings führen wir diese nicht genauer auf, da die Genauigkeit unserer zweiten Methode das Ziel der Aufgabe erreicht.

Da wir mit der zweiten Methode eine alternierende Summe betrachten, ist es mögliche diese mit konvergenzbeschleunigenden Transformationen zu verbessern. Diese würde aus der Gleichung (2) die folgende ergeben:

$$\zeta(s) = \frac{1}{1-2^{1-s}} \left(\sum_{k=1}^{N} \frac{(-1)^{k-1}}{k^s} + \frac{1}{2^N} \sum_{k=N+1}^{2N} \frac{(-1)^{k-1} e_{k-N}}{k^s} \right) + E_N(s) \tag{3}$$

mit $e_k = \sum_{j=k}^{N} \binom{N}{j}$ und $|E| \leq \frac{1}{8^N} \frac{(1+|t/\sigma|)e^{|t|\pi/2}}{|1-2^{1-s}|}$, für $s = \sigma + it$ und $\sigma > 0$.