

# 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:

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

Wir wollen nun im folgenden  $\zeta(2)$  numerisch auf die 15. Dezimalstelle genau berechnen. Hierfür haben wir zwei verschiedene Methoden benutzt,

### Naiver Ansatz

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

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double zetafunktion(double s,int genauigkeit, double* z)
{
    int N = pow(10, genauigkeit);

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

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

```

double s = 2; //Wert fuer Zeta(s)
int genauigkeit = 4; //Genauigkeit in Form von 10^genauigkeit
int N = pow(10, genauigkeit);
double z[100];
int y[N];
for (int i = 1; i < 100; i++) //Laufwert fuer Plot
{
    y[i] = i;
}

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

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

for (unsigned int i = 1; i < 100; i++) { //schreibt Zwischenergebnisse
    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 aus, 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 können. Die berechneten Werte sind in der Zeta\_slow\_data.txt Datei wiederzufinden. Dieser Ansatz liefert uns nach einer Rechenzeit von 5 Minuten eine Genauigkeit von 9 Nachkommastellen mit  $10^9$  Schritten. Diese Methode ist also nicht ausreichend, um auf das gewollte Ergebnis zu kommen.

KONVERGENZ KONVERGENZ KONVERGENZ KONVERGENZ KONVERGENZ

## Borwein Methode

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

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

Die Implementierung in C sieht bei uns wie folgt aus:

```

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

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

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

    for (unsigned int i = 1; i < 100; i++) { //schreibt zwischenergebnisse
        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.

