Übung 01: Riemannsche Zeta Funktion

Tobias Blesgen und Leonardo Thome

4/14/2021

Riemannsche Zeta Funktion

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

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

//[[Rcpp::export]]
long double zetafunktion(int genauigkeit){
    long double sum = 0;
    for (long int i = 10000000; i > 0; i--)
    {
        sum += 1.0/pow(i,2);
    }
    Rprintf("Die Zahl ergibt sich als %.15Lf .\n", sum);
    return sum;

}
```

```
## Die Zahl ergibt sich als 1.644933966848231 .
```

Der eigentliche Wert sollte: `1.644934066848226` sein. Während VSC `1.644934035302976` ausgibt.

Borwein:

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

using namespace Rcpp;

//[[Rcpp::export]]
Rcpp::List borwein(int s, int N){
    // Werte vector
    Rcpp::NumericVector Werte(N*2+1);
    // Algorithmus

    Werte[0] = 0.0;
```

```
    for (int k = 1; k < N; k+=2)
    {
        Werte[k+1] = Werte[k]+pow(k,-s);

    }
    for (int k = 2; k < N; k+=2)
    {
        Werte[N+k+1] = Werte[N+k] - pow(k,-s);
    }

    int skalierung = 1-pow(2,1-s);
    for (int i = 0; i< 2*N+1; i++){
        Werte[i] = Werte[i]/skalierung;
    }
    Rprintf("Das Verfahren nach Borwein ergibt: %.16f \n",Werte[2*N]);
    Rprintf("Fuck\n");
    return List::create(Named("Werte") = Werte);
}
```

funktionierend:

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

//[[Rcpp::export]]
long double borwein2(int s, int N){
    long double summe1 = 0;
    for (int k = 1; k < N; k+=2)
    {
        summe1 += pow(k,-s);
    }
    for (int k = 2; k < N; k+=2)
    {
        summe1 -= pow(k,-s);
    }
    summe1/= (1-pow(2,1-s));
    Rprintf("Das Verfahren nach Borwein scs: %.16Lf \n",summe1);
    return summe1;
}
```

```
## Das Verfahren nach Borwein scs: 1.6449340668482266
```

```
## [1] 1.644934
```