Übung 02: Verfahrens- und Maschinenfehler

Tobias Blesgen und Leonardo Thome

19.05.2021

Im folgenden wollen wir die 2-Punktformel und 3-Punktformel auf ihre Verfahrensfehler untersuchen und durch Verwendung verschiedener Datentypen den Maschienfehler abschätzen.

Die 2-Punktformel beschreibt die Ableitung einer Funktion $f(x)$ an der Stelle x_i mit:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h) \tag{1}$$

Die 3-Punktformel tut dies hingegen mit einem kleineren Verfahrensfehler der Ordnung $O(h^2)$

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2) \tag{2}$$

Verfahrens- und Maschinenfehler

Maschinenfehler

Da wir Zahlen am Computer nur endliche Stellen haben, geschiehen Rundungsfehler bei Berechnung von Zahlen die mehr Stellen besitzen als der benutzte Typ. Diese Genauigkeit nennen wir Maschiengenauigkeit δ_M und ist definiert über die größte reelle Zahl δ_M, so dass der Rechner noch $1 + \delta_M = 1$ berechnet (Seite 12 /cite{DeltaM}).

Verfahrensfehler

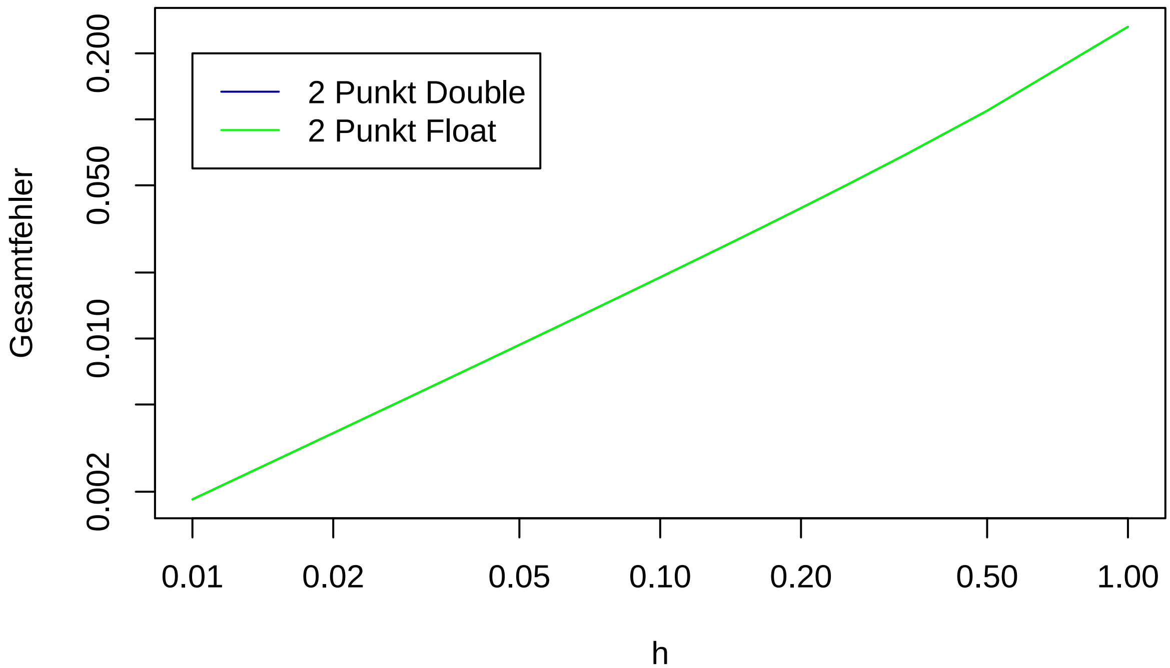
Beispiel Anhand der Differentation

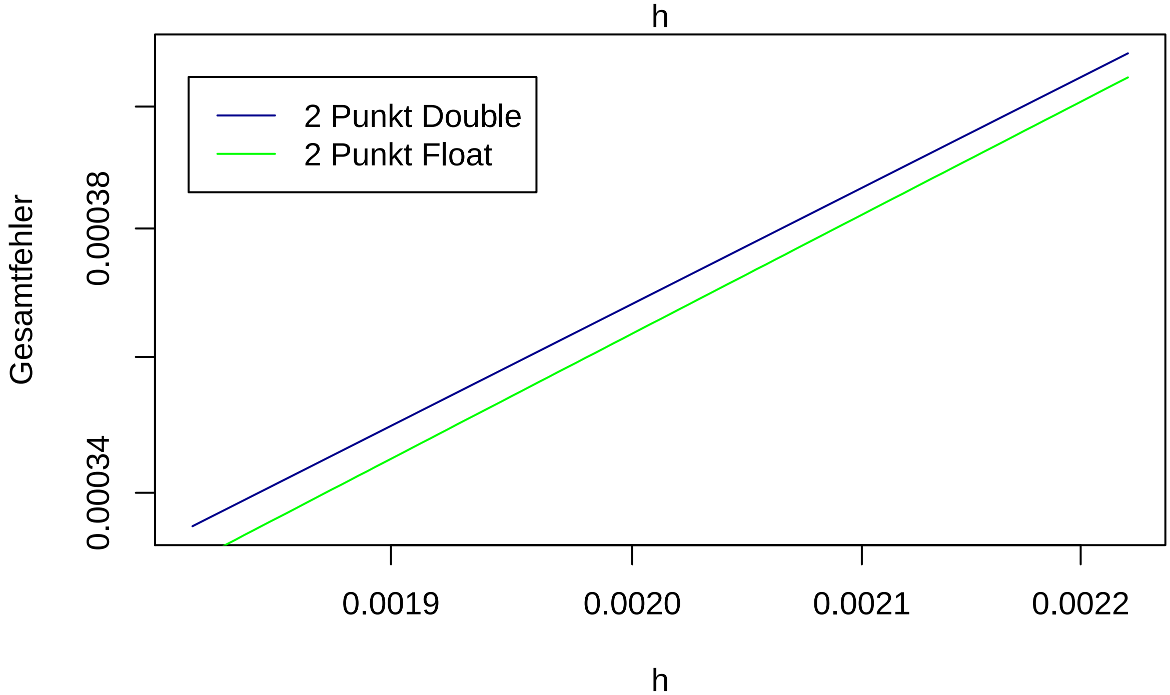
Der 2-Punkt Ansatz

Am Beispiel der Differentation der Funktion $f(x) = e^x$ an der Stelle x = -1 sollen die Fehlertypen aufgezeigt werden. Wir betrachten zuerst die 2-Punktformel und den Unterschied zwischen einem Ansatz, der `Double` und einem, der `Floats` verwendet. Da der Code bis auf den Typwechsel identisch ist, liegt hier nur der Double-Code vor:

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

using namespace Rcpp;

//[[Rcpp::export]]
Rcpp::List diff2PunktDouble(const double x, const int r){
```

```
  // Array der ersten 100 Werte:
    Rcpp::NumericVector xValue(100);
    Rcpp::NumericVector yValue(100);
  // Quelltext
    for (int i = r; i<=99+r; i++){
      xValue[i-r] = 1./i;
      yValue[i-r] = (exp(x+1./i) - exp(x))/(1./i);
    }
  // Rückgabe für eine grafische Wiedergabe
    return List::create(Named("x") = xValue, Named("y") = yValue);
}
```





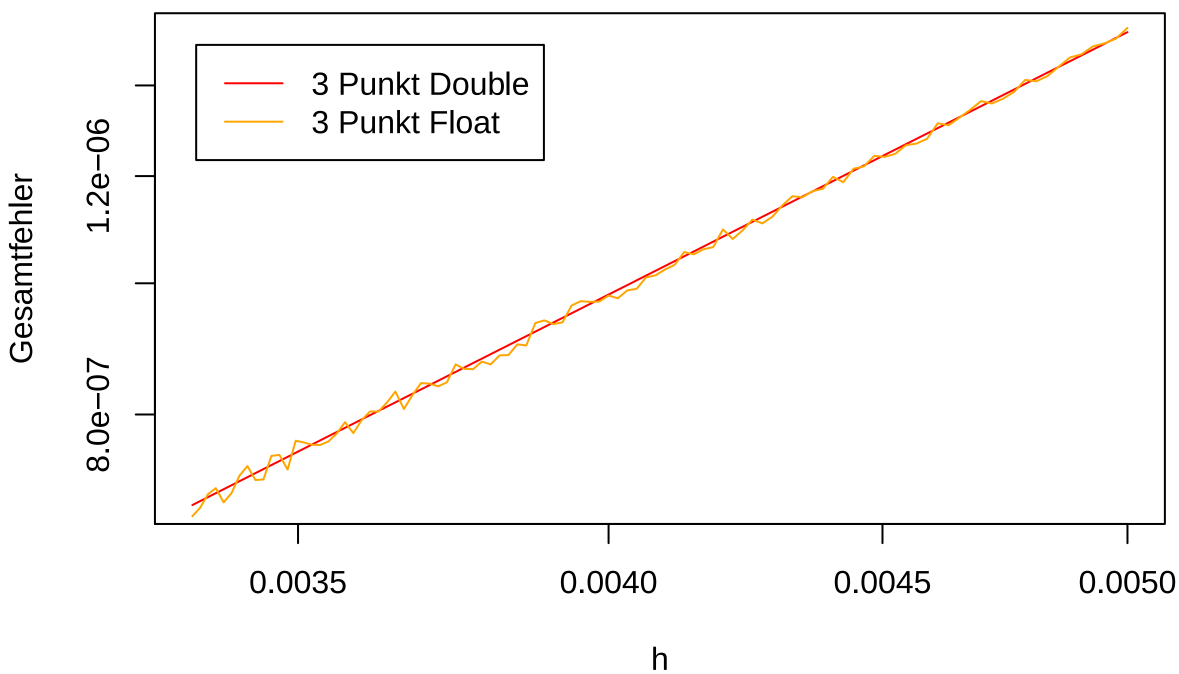
Während für h in der Größenortnung 1 bis 0,01 keine sichtbaren Differenzen zwischen den Gesammtfehlern hervorruft, kann in der nächst kleineren Größerenordnung eine Divergenz der beiden Fehlergrößen beobachtet werden.

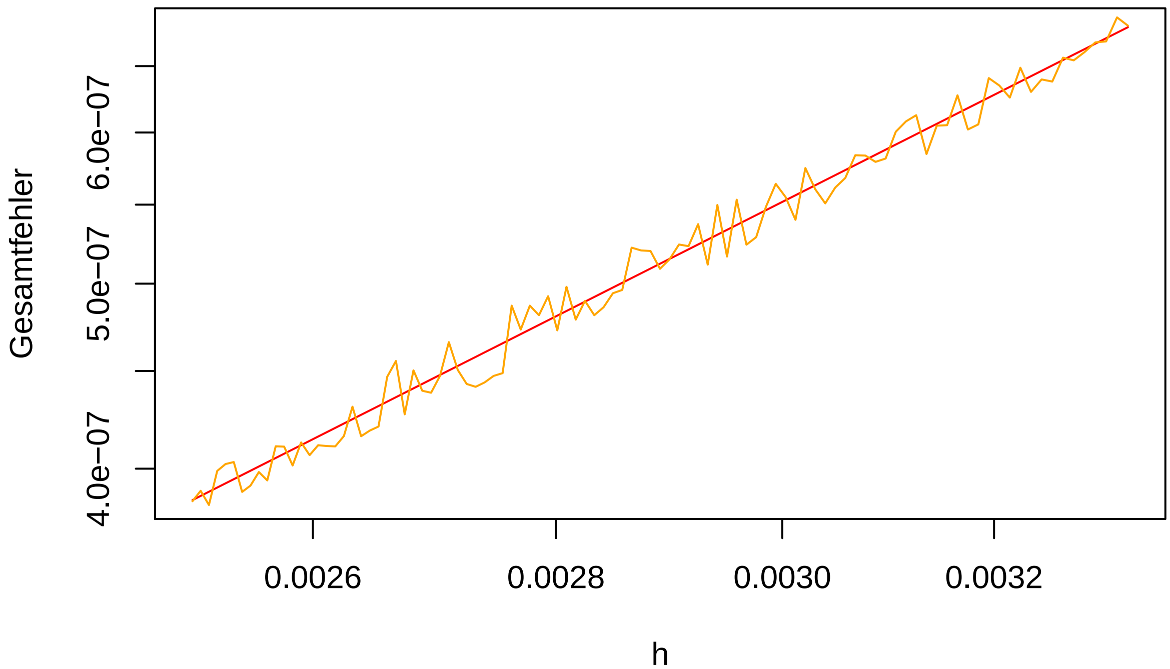
Der 3-Punkt Ansatz

Um das 2-Punktverfahren auch mit einem anderen Verfahren vergleichen zu können, verwenden wir das 3-Punktverfahren. Erneut implementieren wir es einmal mit `Double` und einmal mit `Float`-Precision.

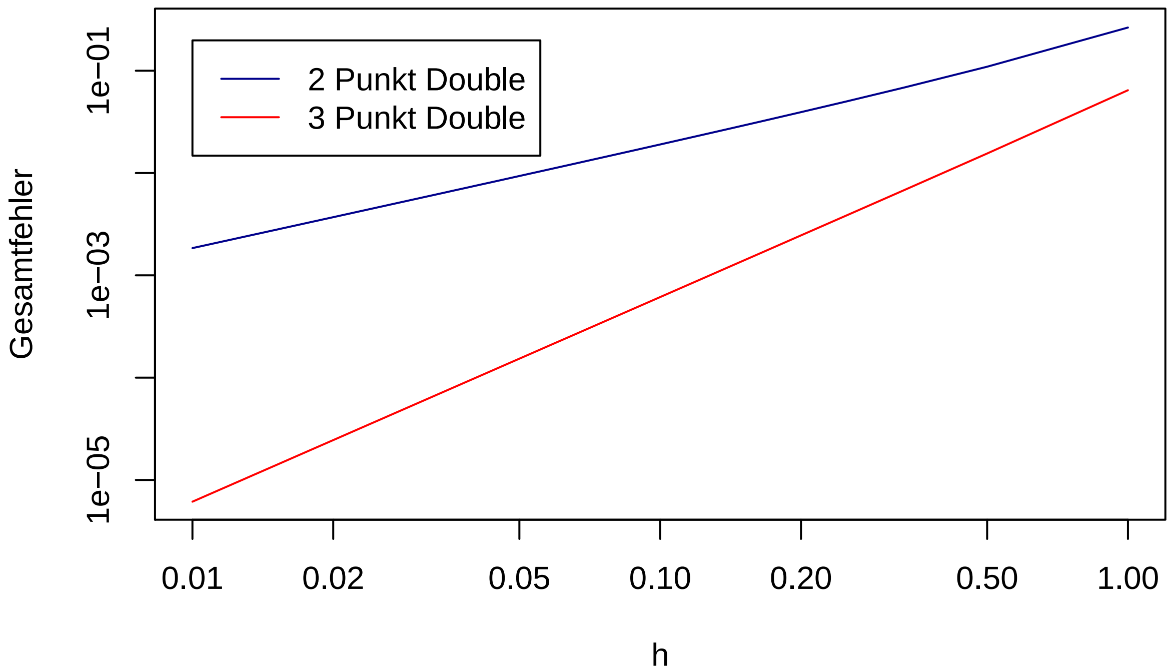
```
#include <Rcpp.h>
#include <math.h>

using namespace Rcpp;

//[[Rcpp::export]]
Rcpp::List diff3PunktDouble(const double x, const int r){
  // Array der ersten 100 Werte:
    Rcpp::NumericVector xValue(100);
    Rcpp::NumericVector yValue(100);
  // Quelltext
    for (int i = r; i<=99+r; i++){
      xValue[i-r] = 1./i;
      yValue[i-r] = (exp(x+1./i) - exp(x-1./i))/(2./i);
    }
  // Rückgabe für eine grafische Wiedergabe
    return List::create(Named("x") = xValue, Named("y") = yValue);
}
```

Trägt man nun die beiden Graphen grafisch auf so findet man für h's bei um die 0,005 die ersten Auffälligen Unterschiede:





Während der `Double`-Graph eine glatte Gerade beschreibt, springt der `Float`-Graph wahllos wirkend über und unter diese Gerade. Dieses Zittern wird mit sinkenden h immer extremer und verdeutlicht, dass die Unterschiede der Näherungswerte die Größenordnung des Maschinenfehlers erreichen. Um diese Zitterbewegung auch am `Float`-Graphen zu entdecken, muss das h in der Größenordnung von 10^{-4} betrachtet werden.



References

[1] C. Urbach, *Vorlesungsskript Computerphysiks*,Seite 12,2021.