Abgabe 2 Verfahrens- und Maschinenfehler

Wayne Ströbel, Silas Müller

16/5/2021

Verfahrens- und Maschinenfehler

Wir wollen anhand des Beispiels der numerischen Differentiation die bei praktischen Anwendungen häufig gegenläufig auftretenden Einflüsse von Verfahrens- und Maschinenfehler empirisch untersuchen. Wir betrachte die Ableitung der folgenden Testfunktion:

$$f(x) = e^x \tag{1}$$

an der Stelle $x = -1$.

Maschinengenauigkeit

Um bei den folgenden Punkten den Maschinenfehler zu berechnen müssen wir zuerst die Maschinengenauigkeit bestimmen. Für diese gilt:

$$rd(1.0 + \delta_m) > 1.0 \tag{2}$$

Dafür haben wir dann einen Code geschrieben, der uns für die Datentypen double und float das δ_m berechnen:

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




//[[Rcpp::export]]
double Maschinenfehler_Double(){
  double k = 1;
  double x = 0;
  while (k == 1)
  {
    x += 0.00000000000000000001;
    k += x;
  }
  return x;
}


//[[Rcpp::export]]
float Maschinenfehler_Float(){
```

```
  float k = 1.0;
  float x = 0;
  for (int i = 0; k == 1.0; i++)
  {
    x += 0.000000001;
    k += x;
  }
  return x;
}
```

```
Maschinenzahl_Double <- Maschinenfehler_Double()
Maschinenzahl_Float <- Maschinenfehler_Float()
```

Mit dem Programm erhalten wir für die Maschinenzahl folgende Werte:

Double $= 1.11023 \times 10^{-16}$

Float $= 5.9999998 \times 10^{-8}$

die Literaturwerte nach IEEE 754 Standard lauten:

Double $= 2^{-53} \approx 1.1 * 10^{-16}$

Float $= 2^{-24} \approx 6 * 10^{-8}$

Also liegen unsere ermittelten Werte sehr nah an den tatsächlichen.

Zwei-Punkt-Formel

Die 2-Punkt-Formel nähert die Ableitung wie folgt, wobei ΔV hier dem Verfahrensfehler entspricht.

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h} + \Delta V \tag{3}$$

Die Herleitung der Fehler ist hier komplett anlog zur 3-Punkt-Formel (4). Mit der Näherung von $f(x) \approx f(x + h) \approx f(x - h)$ für die Fehlerfortpflanzung ergibt sich:

$$\epsilon_g = \epsilon_m + \epsilon_v = |f'(x) - \frac{f(x + h) - f(x)}{h}|$$

und

$$\epsilon_m = \frac{1}{h}\sqrt{\frac{2}{3}}|f(x)|$$

sowie

$$\epsilon_v = \epsilon_g - \epsilon_m$$

mit $x_i = -1$ und h wählbar. Die Rechnung wird für verschiedene Datentypen (`double` und `float`) durchgeführt und später im Hinblick auf die Genauigkeit verglichen. Die jeweiligen Fehler werden extrahiert und gegen h aufgetragen. Die Auftragung geschieht in einem dekadischen log-log-System.

Diese Formel können wir nun wie folgt in C implementieren:

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

```
//[[Rcpp::export]]
Rcpp::NumericVector PPAbleitung_expf(float h,float x,float g,float Maschinenfehler){

  int temp = h / g;

  Rcpp::NumericVector Werte_float(temp);

  for (int i = 0; i < temp; i += 5){
    Werte_float[i] = h - i*g;
    Werte_float[i+1] = (exp(x+(h - i*g))-exp(x))/(h - i*g);
    Werte_float[i+2] = (exp(x+(h - i*g))-exp(x))/(h - i*g)-exp(x);
    Werte_float[i+3] = 1 / ((h - i*g)) * 0.8164965809 * Maschinenfehler * exp(x);
    Werte_float[i+4] = Werte_float[i+2] - Werte_float[i+3];

  }
  return Werte_float;
}

//[[Rcpp::export]]
Rcpp::NumericVector PPAbleitung_expd(double h,double x, double g, double Maschinenfehler){

  int temp = h / g;

  Rcpp::NumericVector Werte_double(temp);

  for (int i = 0; i < temp; i += 5){
    Werte_double[i] = h - i*g;
    Werte_double[i+1] = (exp(x+(h - i*g))-exp(x))/(h - i*g);
    Werte_double[i+2] = (exp(x+(h - i*g))-exp(x))/(h - i*g)-exp(x);
    Werte_double[i+3] = 1 / ((h - i*g)) * 0.8164965809 * Maschinenfehler * exp(x);
    Werte_double[i+4] = Werte_double[i+2] - Werte_double[i+3];

  }
  return Werte_double;
}
```

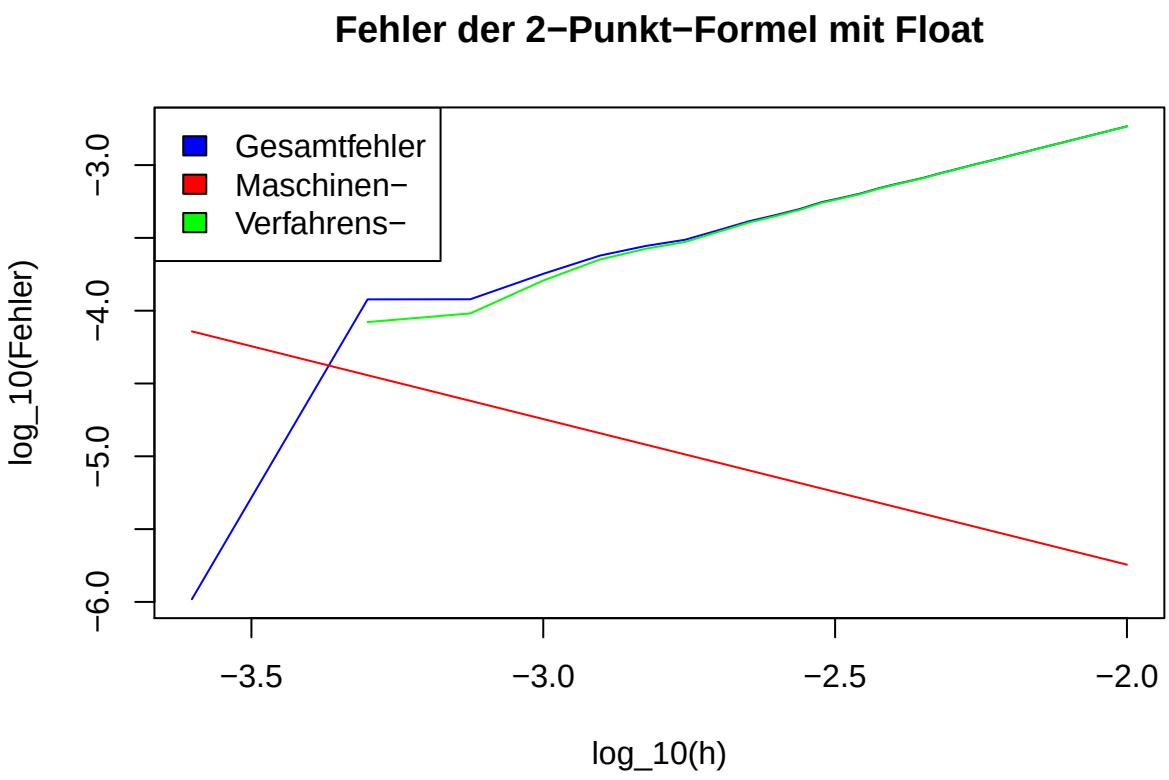
```
x <- -1

#ZweiPunktFloat
ppf_start <- 0.01
ppf_step <- 0.00005

PPAblExp_float <- PPAbleitung_expf(ppf_start,x,ppf_step,Maschinenzahl_Float)

#ZweiPunktDouble
ppd_start <- 0.00001
ppd_step <- 0.000000001

PPAblExp_double <- PPAbleitung_expd(ppd_start,x,ppd_step,Maschinenzahl_Double)
```

Plots für die 2-Punkt-Formel

```
#Funktion zur Einordnung der Werte
nth_element <- function(vector, starting_position, n){
  vector[seq(starting_position, length(vector), n)]
}

#Plot für 2PunktFloat
ppf_h <- nth_element(PPAblExp_float, 1, 5)
ppf_ges <- nth_element(PPAblExp_float, 3, 5)
ppf_masch <- nth_element(PPAblExp_float, 4, 5)
ppf_verf <- nth_element(PPAblExp_float, 5, 5)

plot(log10(ppf_h), log10(ppf_ges),
     main="Fehler der 2-Punkt-Formel mit Float",
     ylab="log_10(Fehler)",
     xlab="log_10(h)",
     type="l",
     col="blue")
lines(log10(ppf_h), log10(ppf_masch), col="red")
lines(log10(ppf_h), log10(ppf_verf), col="green")
```

```
## Warning in xy.coords(x, y): NaNs produced
```

```
legend("topleft", c("Gesamtfehler","Maschinen-","Verfahrens-"),
       fill=c("blue","red","green"))
```



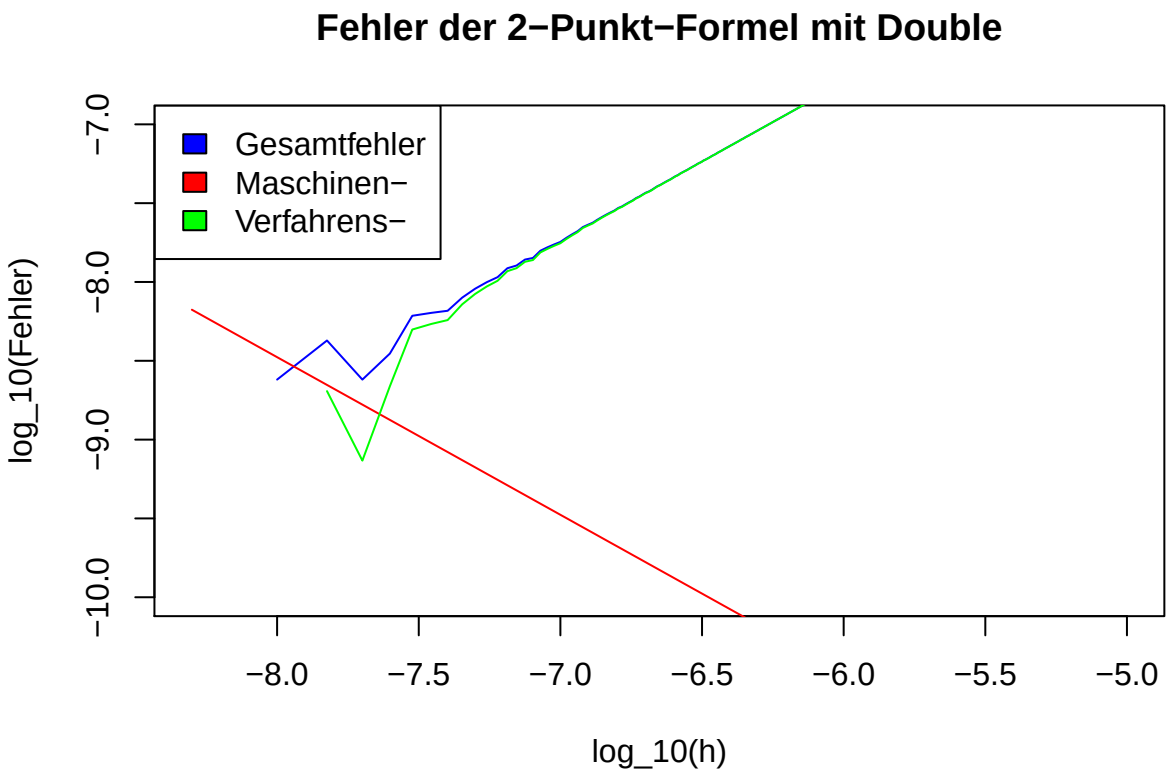

```
#Plot für 2PunktDouble
ppd_h <- nth_element(PPAblExp_double, 1, 5)
ppd_ges <- nth_element(PPAblExp_double, 3, 5)
ppd_masch <- nth_element(PPAblExp_double, 4, 5)
ppd_verf <- nth_element(PPAblExp_double, 5, 5)

plot(log10(ppd_h), log10(ppd_ges),
     main="Fehler der 2-Punkt-Formel mit Double",
     ylab="log_10(Fehler)",
     xlab="log_10(h)",
     type="l",
     col="blue",
     ylim=c(-10,-7))
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): NaNs produced
```

```
lines(log10(ppd_h), log10(ppd_masch), col="red")
lines(log10(ppd_h), log10(ppd_verf), col="green")
```

```
## Warning in xy.coords(x, y): NaNs produced
```

```
legend("topleft", c("Gesamtfehler","Maschinen-","Verfahrens-"),
       fill=c("blue","red","green"))
```



Erläuterung der Auftragung

Es wurden die bereits genannten Fehler doppellogarithmisch gegen h aufgetragen. Es ist gut zu erkennen, dass für große h der Verfahrensfehler ansteigt und dieser praktisch dem Gesamtfehler entspricht. Erst für kleine h weicht der Verfahrensfehler stärker von dem Gesamtfehler ab und der Maschinenfehler steigt signifikant. So ist der erwähnte "ideale Punkt" mit dem kleinsten Fehler zu erkennen. Es ist außerdem zu erwähnen, dass eben der Maschinenfehler für kleine h dem Gesamtfehler entspricht. Nähert man sich diesem Punkt, an dem der Maschinenfehler der signifikante Fehler wird, so entstehen auch Probleme in der Berechnung des Gesamt- und Verfahrenfehlers, sodass diese instabil werden. Dies ist durch das "Springen" der Graphen in diesem Bereich erkennbar. Des Weiteren ist ein deutlicher Unterschied zwischen double und float erkennbar. Die double Rechnung liefert deutlich geringere Fehler, und ließ sich dem entsprechend bei kleineren h deutlich genauer durchführen.

Drei-Punkt-Formel

Die 3-Punkt-Formel nähert die Ableitung wie folgt, wobei ΔV dem Verfahrensfehler entspricht.

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \Delta V \tag{4}$$

Betrachtet man nun auch den Maschinenfehler so stellt man fest, dass $f(x+h)$ auf

$$[(1-\delta_m)f(x+h), (1+\delta_m)f(x+h)]$$

abgebildet wird (analog für $f(x)$ und $f(x+h)$). Nimmt man nun eine gleichverteilte Wahrscheinlichkeitsverteilung an, so ergibt sich ein Fehler von $\sigma = \frac{\delta_m}{\sqrt{3}}$ (siehe Vorlesung). Berechnet man nun den Maschinenfehler so ergibt sich:

$$\Delta M(f(x+h) - f(x-h)) = \sqrt{2}\sigma|f(x)|$$

Betrachtet man nun den Gesamtfehler ϵ_g mit der wahren Ableitung $f'(x)$:

$$\epsilon_g = \epsilon_m + \epsilon_v = |f'(x) - \frac{f(x+h) - f(x-h)}{2h}|$$

Also ergibt sich für den Maschinenfehler für die 3-Punkt-Formel:

$$\epsilon_m = \frac{1}{2h}\sqrt{\frac{2}{3}}|f(x)|$$

und für den Verfahrensfehler:

$$\epsilon_v = \epsilon_g - \epsilon_m$$

Für die genannte Testfunktion soll nun die 3-Punkt-Formel angwendet und die jeweiligen Fehler extrahiert und gegen h aufgetragen werden. Außerdem wird die Rechnung für verschiedene Datentypen (double und float) durchgeführt und später im Hinblick auf die Genauigkeit verglichen. Dies geschieht ebenfalls in einem dekadischen log-log-System. Es folgt die Implementierung in C und späteres Plotten:

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


//[[Rcpp::export]]
Rcpp::NumericVector PPPAbleitung_expf(float h,float x,float g,float Maschinenfehler){

  int temp = h / g;

  Rcpp::NumericVector Werte_float(temp);

  for (int i = 0; i < temp; i += 5){
    Werte_float[i] = h - i*g;
    Werte_float[i+1] = (exp(x+(h - i*g))-exp(x-(h - i*g)))/(2*(h - i*g));
    Werte_float[i+2] = (exp(x+(h - i*g))-exp(x-(h - i*g)))/(2*(h - i*g))-exp(x);
    Werte_float[i+3] = 1 / (2*(h - i*g)) * 0.8164965809 * Maschinenfehler * exp(x);
    Werte_float[i+4] = Werte_float[i+2] - Werte_float[i+3];

  }
  return Werte_float;
}

//[[Rcpp::export]]
Rcpp::NumericVector PPPAbleitung_expd(double h,double x,double g,double Maschinenfehler){

  int temp = h / g;

  Rcpp::NumericVector Werte_double(temp);

  for (int i = 0; i < temp; i += 5){
```

```
    Werte_double[i] = h - i*g;
    Werte_double[i+1] = (exp(x+(h - i*g))-exp(x-(h - i*g)))/(2*(h - i*g));
    Werte_double[i+2] = (exp(x+(h - i*g))-exp(x-(h - i*g)))/(2*(h - i*g))-exp(x);
    Werte_double[i+3] = 1 / (2*(h - i*g)) * 0.8164965809 * Maschinenfehler * exp(x);
    Werte_double[i+4] = Werte_double[i+2] - Werte_double[i+3];

  }
  return Werte_double;
}
```

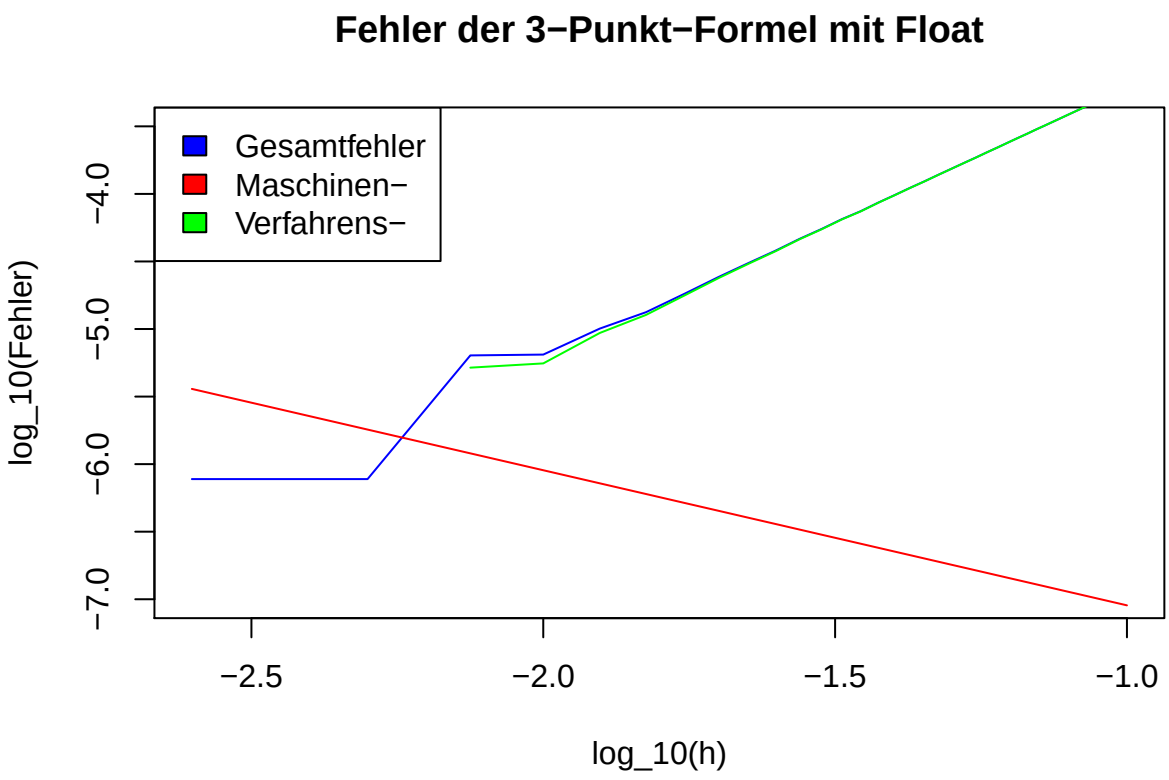
```
x <- -1

#DreiPunktFloat
pppf_start <- 0.1
pppf_step <- 0.0005

PPPAblExp_float <- PPPAbleitung_expf(pppf_start,x,pppf_step,Maschinenzahl_Float)

#DreiPunktDouble
pppd_start <- 0.0001
pppd_step <- 0.0000005

PPPAblExp_double <- PPPAbleitung_expd(pppd_start,x,pppd_step,Maschinenzahl_Double)
```

Plots für die 3-Punkt-Formel

```
#Plot für 3PunktFloat
pppf_h <- nth_element(PPPAblExp_float, 1, 5)
pppf_ges <- nth_element(PPPAblExp_float, 3, 5)
pppf_masch <- nth_element(PPPAblExp_float, 4, 5)
pppf_verf <- nth_element(PPPAblExp_float, 5, 5)

plot(log10(pppf_h), log10(pppf_ges),
     main="Fehler der 3-Punkt-Formel mit Float",
     ylab="log_10(Fehler)",
     xlab="log_10(h)",
     type="l",
     col="blue",
     ylim=c(-7,-3.5))
lines(log10(pppf_h), log10(pppf_masch), col="red")
lines(log10(pppf_h), log10(pppf_verf), col="green")
```

```
## Warning in xy.coords(x, y): NaNs produced
```

```
legend("topleft", c("Gesamtfehler","Maschinen-","Verfahrens-"),
       fill=c("blue","red","green"))
```



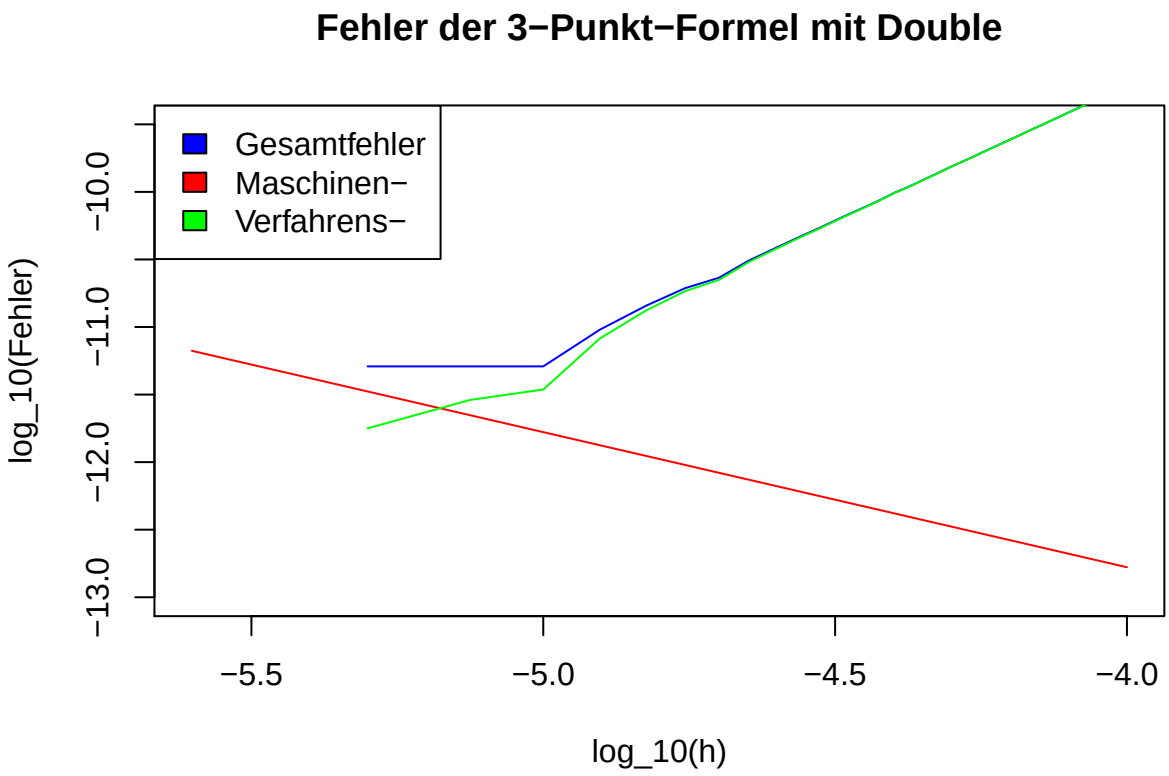

```
#Plot für 3PunktDouble
pppd_h <- nth_element(PPPAblExp_double, 1, 5)
pppd_ges <- nth_element(PPPAblExp_double, 3, 5)
pppd_masch <- nth_element(PPPAblExp_double, 4, 5)
pppd_verf <- nth_element(PPPAblExp_double, 5, 5)

plot(log10(pppd_h), log10(pppd_ges),
     main="Fehler der 3-Punkt-Formel mit Double",
     ylab="log_10(Fehler)",
     xlab="log_10(h)",
     type="l",
     col="blue",
     ylim=c(-13,-9.5))
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): NaNs produced
```

```
lines(log10(pppd_h), log10(pppd_masch), col="red")
lines(log10(pppd_h), log10(pppd_verf), col="green")
```

```
## Warning in xy.coords(x, y): NaNs produced
```

```
legend("topleft", c("Gesamtfehler","Maschinen-","Verfahrens-"),
       fill=c("blue","red","green"))
```



Erläuterung der Auftragung

Die Erläuterung ist hier praktisch analog zum 2-Punkt-Verfahren. Jedoch ist im Vergleich ein deutlich geringerer Fehler zu erkennen. Dies ist hauptsächlich auf geringere Verfahrensfehler des 3-Punkt-Verfahrens zurückzuführen. Das Gesamtverhalten ist jedoch analog zur 2-Punkt-Formel.

Vergleich 3-Punkt- und 2-Punkt-Formel

Der Verfahrensfehler bei der 3-Punkt-Formel ist proportional zu h^2, während der Verfahrensfehler bei der 2-Punkt-Formel proportional zu h ist. Für beide Verfahren nimmt also der Verfahrensfehler für kleine h ab, bei der 3-Punkt-Formel jedoch deutlich schneller. Gleichzeitig ist anhand des Maschinenfehlers zu erkennen, dass dieser mit kleinerem h anwächst. Es sollte also ein "idealer Punkt" existieren, bei dem der Gesamtfehler am geringsten ist. Vergleiche hierzu auch die Auftragung der jeweiligen Ergebnisse. Bei beiden Verfahren stellt sich die double Rechnung als deutlich genauer heraus als die float Rechnung, erklärbar durch die berechneten Maschinengenauigkeiten und Mantissenlängen der Datentypen.

Float und Double, Mantissenlänge

Es werden die 2-Punkt-Formel und 3-Punkt-Formel mit verschiedenen Genauigkeitsstufen durchgeführt (double und float). Anhand der Resultate sollen die Maschinengenauigkeit und Mantissenlängen der einzelnen Datenobjekte abgeschätzt und mit bekannten Definitionen verglichen werden, mit der Maschinengenauigkeit δ_m für die das Folgende gilt und der Mantissenlänge k und Basis E:

$$\delta_m = \frac{E}{2} E^{-k} \tag{5}$$

Diese Definition können wir nun mit $E = 2$ umstellen zu:

$$|\log_2(\delta_m)| = k \tag{6}$$

```
k_double <- abs(log2(Maschinenzahl_Double))
k_float <- abs(log2(Maschinenzahl_Float))
```

Damit erhalten wir:

Double = 52.9999909

Float = 23.9904623

Wie zu erwarten besitzt der float-Datentyp eine geringere Mantissenlänge und Genauigkeit als der double Datentyp (mit der Basis E=2). Ein Vergleich mit der Literatur bestätigt dies (nach IEEE 754: float 23 bit, double 52 bit).

Fazit

Bei den Verfahren stellte sich die Herausforderung, einen guten Wertebereich zu finden, in dem man die Relation zwischen Verfahrens- und Maschinenfehler, sowie der Instabilität bei kleiner werdenden h gut zeigen kann.

Letzendlich können wir sagen, dass wir mit der Drei-Punkt-Formel bei gleichen h-Werten deutlich geringere Verfahrensfehler sehen, als bei der der Zwei-Punkt-Formel. In den einzelnen Verfahren erkennt man außerdem, dass doubles eine deutlich kleinere Maschinenzahl besitzen als floats, es also für genauere Rechnungen besser ist, in doubles zu rechnen, als in floats.

Quelle

Vorlesung vom 23.4 (Fehler und 2 bzw. 3-Punkt-Formel) und 14.4 (Maschinengenauigkeit,Mantissenlänge)

IEEE 754; Wikipedia, https://de.wikipedia.org/wiki/IEEE_754