

# Interpolation der Runge-Funktion und anderer Funktionen mit Octave

HENRY HAUSTEIN, LARS ORTSCHIEDT

28. November 2018

## Inhaltsverzeichnis

<b>1</b>	<b>Interpolation der Runge-Funktion</b>	<b>2</b>
1.1	Berechnung der Splines . . . . .	2
1.1.1	Polynomsplines aus $\mathcal{S}_1^0(\Delta)$ . . . . .	2
1.1.2	Polynomsplines aus $\mathcal{S}_3^1(\Delta)$ . . . . .	4
1.2	Fehlerbetrachtung . . . . .	5
1.3	Diskussion der Ergebnisse . . . . .	6
<b>2</b>	<b>Interpolation der anderen Funktion</b>	<b>7</b>
2.1	Berechnung der Splines . . . . .	8
2.2	Diskussion der Ergebnisse . . . . .	8

# 1 Interpolation der Runge-Funktion

$$f(x) = \frac{1}{1 + 25x^2}$$
$$f'(x) = -\frac{50x}{625x^4 + 50x^2 + 1}$$

## 1.1 Berechnung der Splines

### 1.1.1 Polynomsplines aus $\mathcal{S}_1^0(\Delta)$

Eine Polynomspline  $s \in \mathcal{S}_1^0(\Delta)$  ist eine affin lineare Funktion, das heißt er hat die Form  $s(x) = mx + n$  mit Anstieg  $m$  und  $y$ -Achsenverschiebung  $n$ .

Die Interpolationsfunktion  $g_N$ , mit  $N + 1$  Stützstellen, besteht nun also aus Splines  $s_i \in \mathcal{S}_1^0(\Delta)$ , wobei für jeden Spline gilt:

$$\text{Definitionsbereich: } [x_i, x_{i+1}]$$
$$m_i = \frac{f_{i+1} - f_i}{x_{i+1} - x_i}$$
$$n_i = f_i$$

wobei  $x_i$  die Stützstellen und  $f_i$  die Stützwerte sind. Dabei läuft  $i$  von 0 bis  $N - 1$ .

Der Quelltext für Octave sieht dann so aus:

```
1  runge = @(x) 1./(1+25*x.^2);
2  xreal = -1:0.01:1;
3
4  n = input('Anzahl der Stuetzstellen - 1 := N: ');
5
6  %Schritweite h berechnen
7  h = 2/n
8  %Stuetzstellenvektor x berechnen
9  x = -1:h:1;
10
11 for i=1:n+1
12  %Stutzwertevektor f berechnen
13  f(i) = runge(x(i));
14 endfor
15
16 for i=1:n
17  %Anstiege m_i berechnen
18  m(i) = (f(i+1)-f(i))./(x(i+1)-x(i));
19  %Achsenabschnitte n_i berechnen
```

```

20  n(i) = f(i);
21  endfor
22
23  plot(x, f, "-;Interpol.;", xreal, runge(xreal), "-;Rungefkt.;" )

```

Das Interessante hierbei ist, dass die berechneten Werte in den Arrays `m` und `n` gar nicht für die Interpolation gebraucht werden - die Funktion `plot` interpoliert automatisch linear, wenn man ihr die Stützstellen und -werte übergibt.

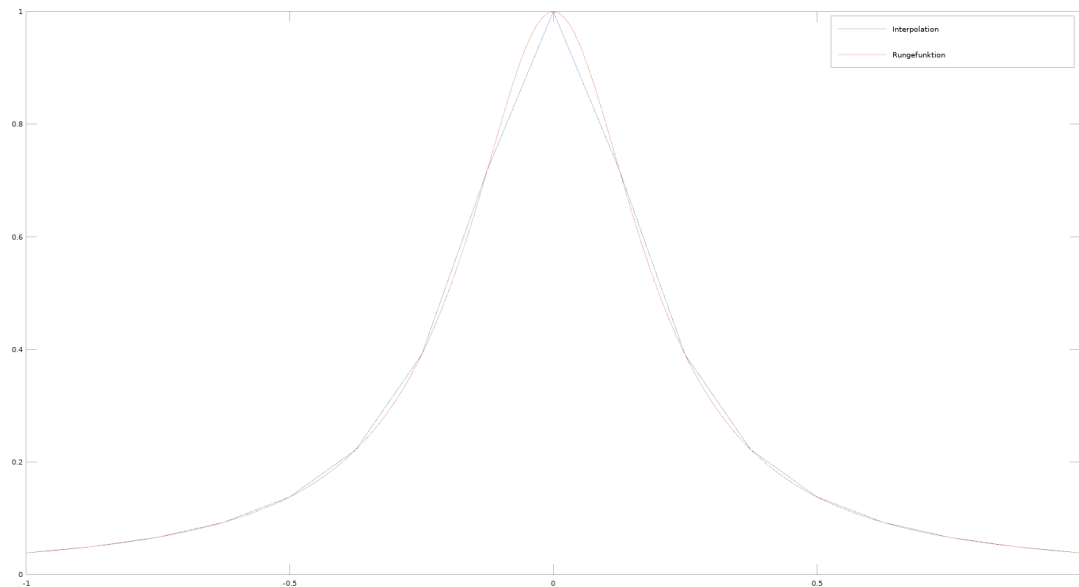


Abbildung 1: lineare Splineinterpolation mit  $N = 16$

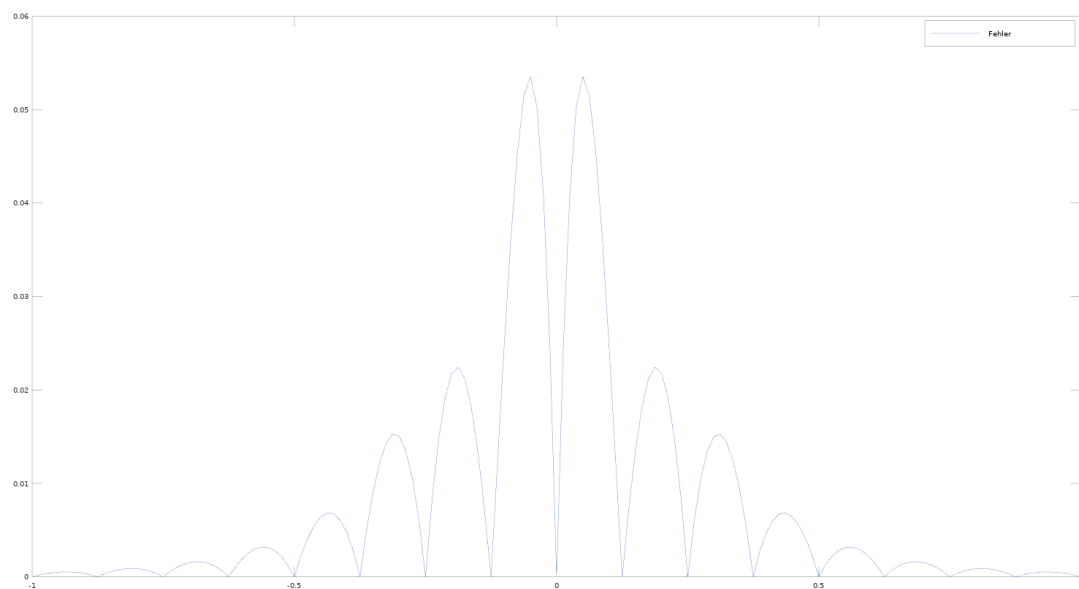


Abbildung 2: Fehler bei linearer Splineinterpolation mit  $N = 16$

### 1.1.2 Polynomsplines aus $\mathcal{S}_3^1(\Delta)$

Da die Interpolationssplines Polynome dritten Grades und einmal stetig differenzierbar sein sollen, nehmen wir aus der Vorlesung den Ansatz (1.7):

$$s_k(x) = a_k(x - x_k)^3 + b_k(x - x_k)^2 + c_k(x - x_k) + d_k$$

mit  $x \in [x_k, x_{k+1}]$ . Die Vorfaktoren  $a_k$ ,  $b_k$ ,  $c_k$  und  $d_k$  ergeben sich aus (1.9) und (1.10) in der Vorlesung.

$$\begin{aligned} d_k &= f_k \\ c_k &= m_k = s'(x_k) = f'(x_k) \\ \begin{pmatrix} h_k^3 & h_k^2 \\ 3h_k^2 & 2h_k \end{pmatrix} \begin{pmatrix} a_k \\ b_k \end{pmatrix} &= \begin{pmatrix} f_{k+1} - f_k - m_k h_k \\ m_{k+1} - m_k \end{pmatrix} \end{aligned}$$

wobei  $h_k$  mit  $\frac{2}{N}$  gegeben war. Der Quelltext sieht dann folgendermaßen aus:

```
1  runge = @(x) 1./(1+25*x.^2);
2  runge_abl = @(x) (-50*x)/(1+25*x^2)^2;
3  xreal = -1:0.01:1;
4
5  N = input('Anzahl der Stuetzstellen -1 :=N : ');
6
7  %Abstand Stuetzstellen h
8  h = 2./N;
9
10 %Stuetzstellen x
11 x = -1:h:1;
12
13 for i = 1:N+1
14     %Stuetzwerte f
15     f(i) = runge(x(i));
16     %Ableitungen
17     m(i) = runge_abl(x(i));
18 endfor
19
20 %Berechnung a_k, b_k nach 1.10
21 H = [h^3 , h^2 ; 3*h^2 , 2*h];
22 for i = 1:N
23     r = H\[f(i+1)-f(i)-m(i)*h ; m(i+1)-m(i)];
24     a(i) = r(1);
25     b(i) = r(2);
26     %c(i) = m(i)
27     %d(i) = f(i)
28 endfor
```

```

29
30 %Interpolierende und Runge plotten auf Zerlegung M
31 M = 10 * N;
32 h_fein = 2/M;
33 x_fein = -1:h_fein:1;
34 k = 1;
35 for i=1:N
36     %in jedem dieser Durchlaufe ist der Spline-Abschnitt der Selbe
37     for j=1:10
38         s(k) = a(i)*(x_fein(k)-x(i))^3 + b(i)*(x_fein(k)-x(i))^2 + ...
39             m(i)*(x_fein(k)-x(i))+f(i);
40         k = k + 1;
41     endfor
42 endfor
43
44 s(k) = f(N+1);
45
46 figure(1);
47 plot(x_fein, runge(x_fein), "-;Rungefunktion;", x_fein, ...
48 s, "-;Interpolation;")

```

## 1.2 Fehlerbetrachtung

Da  $\Delta_M$  zehnmal so fein wie  $\Delta_N$  ist, bedeutet das, dass man für jeden Spline den Fehler in 10 Punkten in seinem Definitionsbereich berechnet.

Bei linearer Interpolation kann man also deswegen den Fehler nach folgendem Muster ausrechnen:

$$\text{Fehler} = |f(x) - (n + \text{Abstand zur nächsten Stützstelle} \cdot m)|$$

wobei  $n$  und  $m$  zum jeweiligen Spline gehören und  $x$  die Werte in  $\Delta_M$  durchläuft. Da die Fehlerfunktion laut Aufgabenstellung an den Stützstellen der Zerlegung  $\Delta_M$  zu berechnen ist, lässt sich der nachfolgende Code auch für die Abschätzung des Fehlers (der auch an den Stützstellen von  $\Delta_M$  gesucht ist) wiederverwenden. Der Quelltext dazu sieht folgendermaßen aus:

```

1 M = 10 * N
2 h_neu = 2/M
3 x_Fehler = -1:h_neu:1;
4
5 k = 1;
6 for i=1:N
7     %in jedem dieser Durchlaufe ist der Spline-Abschnitt der Selbe
8     for j=1:10
9         y_Fehler(k) = abs(runge(x_Fehler(k)) - ...
10             (n(i) + abs(abs(x_Fehler(k)) - abs(x(i))) * m(i)));

```

```

11     k = k + 1;
12 endfor
13 endfor
14
15 %Fehler an letzter Stuetzstelle ist 0
16 y_Fehler(k) = 0;
17
18 plot(x_Fehler, y_Fehler, "-; Fehler;")
19
20 % maximaler Fehler E
21 E = max(y_Fehler)

```

Für die Fehlerberechnung bei kubischer Interpolation haben wir wieder den Ansatz  $s_k(x) = a_k(x - x_k)^3 + b_k(x - x_k)^2 + c_k(x - x_k) + d_k$  verwendet.

```

1  %Fehlerfunktion
2
3  k = 1;
4  for i=1:N
5      %in jedem dieser Durchlaufe ist der Spline-Abschnitt der Selbe
6      for j=1:10
7          y_Fehler(k) = abs(runge(x_fein(k)) - ...
8              (a(i)*(x_fein(k)-x(i))^3 + b(i)*(x_fein(k)-x(i))^2 + ...
9              m(i)*(x_fein(k)-x(i))+f(i)));
10         k = k + 1;
11     endfor
12 endfor
13
14 %Fehler an letzter Stuetzstelle ist 0
15 y_Fehler(k) = 0;
16
17 figure(2);
18 plot(x_fein, y_Fehler, "-; Fehler;")
19
20 %Maximaler Fehler
21 E = max(y_Fehler);

```

### 1.3 Diskussion der Ergebnisse

Der maximale Fehler  $E(h_N)$  für  $N = N_k = 4 \cdot 2^k$  mit  $k = 0, \dots, 4$  beträgt:

$k$	$N$	$E(h_N)$ linear	$E(h_N)$ kubisch
0	4	0.17872	0.21938
1	8	0.063128	0.035509
2	16	0.053536	0.0016935
3	32	0.020652	0.00038860
4	64	0.0058496	0.000033560

Man sieht also, dass bei großen  $N$  der Fehler sehr klein wird und die kubische Splineinterpolation besser als die lineare Interpolation ist.

Die exponentielle Konvergenzordnung ist

$k$	$N$	$EOC(h_{N_k}, h_{N_{k+1}})$ linear	$EOC(h_{N_k}, h_{N_{k+1}})$ kubisch
0	4		2.6272
1	8		4.3901
2	16		2.1237
3	32		3.5334
4	64		3.8869
5	128		3.9719
6	256		3.9930
7	512		3.9982
8	1024		3.9996
9	2048		3.9999
10	4096		4.0000

Mit  $k = 11$ ,  $N = 8192$  ist  $h_{N_k} = \frac{2}{8192}$

## 2 Interpolation der anderen Funktion

$$f(x) = \left(1 + \cos\left(\frac{3}{2}\pi x\right)\right)^{2/3}$$

$$f'(x) = -\frac{\pi \sin\left(\frac{3}{2}\pi x\right)}{\sqrt[3]{1 + \cos\left(\frac{3}{2}\pi x\right)}}$$

## 2.1 Berechnung der Splines

Für Rechenvorschrift siehe [1.1](#)

## 2.2 Diskussion der Ergebnisse

Der maximale Fehler beträgt

$k$	$N$	$E(h_N)$ linear	$E(h_N)$ kubisch
0	4		0.19577
1	8		0.070736
2	16		0.027316
3	32		0.010764
4	64		0.0042640

Die exponentielle Konvergenzordnung ist

$k$	$N$	$EOC(h_{N_k}, h_{N_{k+1}})$ linear	$EOC(h_{N_k}, h_{N_{k+1}})$ kubisch
0	4		1.4686
1	8		1.3727
2	16		1.3436
3	32		1.3359
4	64		1.3340
5	128		1.3335
6	256		1.3334
7	512		1.3333
8	1024		1.3333
9	2048		1.3333
10	4096		1.3333

Mit  $k = 11$ ,  $N = 8192$  ist  $h_{N_k} = \frac{2}{8192}$ .

Offensichtlich ist diese Funktion nicht so gut für eine Splineinterpolation geeignet wie die RUNGE-Funktion, weil die maximalen Fehler größer sind und die EOC kleiner ist.