Interpolation der Runge-Funktion und anderer Funktionen mit Octave

Henry Haustein, Lars Ortscheidt

14. November 2018

Inhaltsverzeichnis

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('Anzahhl 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.

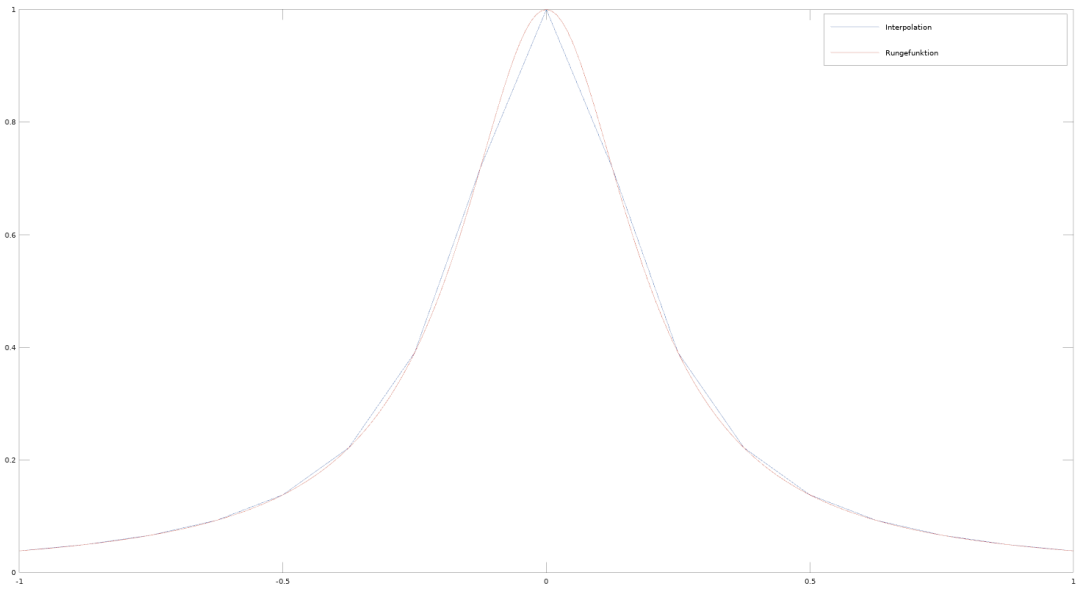

Abbildung 1: lineare Splineinterpolation mit $N = 16$

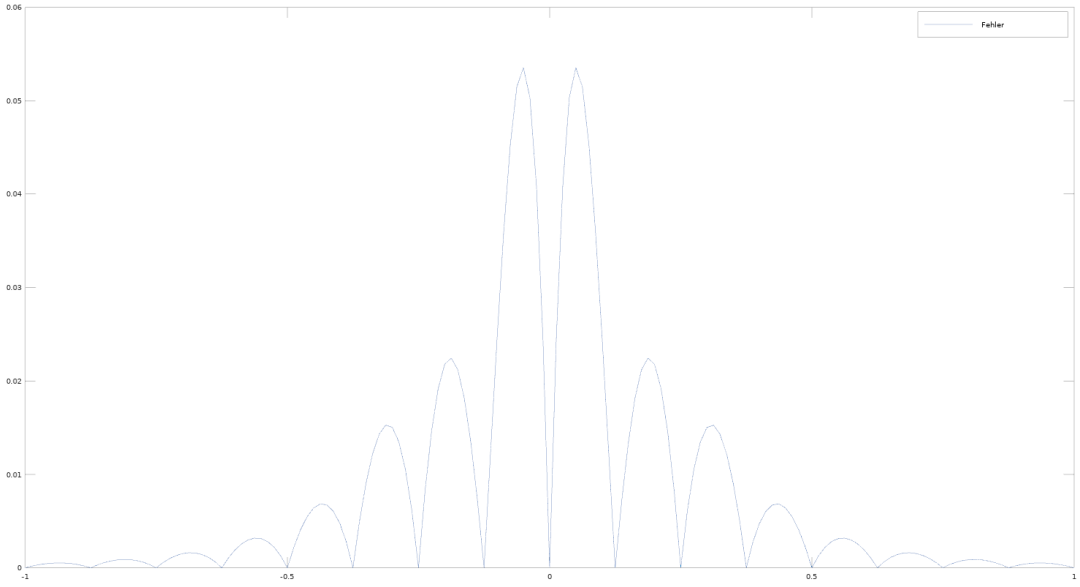

Abbildung 2: Fehler bei linearer Splineinterpolation mit $N = 16$

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

1.2 Fehlerbetrachtung

Da Δ_M zehnmal so fein wie Δ_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 Δ_M durchläuft. Da die Fehlerfunktion laut Aufgabenstellung an den Stützstellen der Zerlegung Δ_M zu berechnen ist, lässt sich der nachfolgende Code auch für die Abschätzung des Fehlers (der auch an den Stützstellen von Δ_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 Durchlauufe 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)
```

1.3 Diskussion der Ergebnisse

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

k	N	$E(h_N)$
0	4	0.17872
1	8	0.063128
2	16	0.053536
3	32	0.020652
4	64	0.0058496

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

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

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

2.2 Fehlerbetrachtung

2.3 Diskussion der Ergebnisse