Signaltransformationen mit MATLAB

Numerische Überprüfung von Ergebnissen

Stefan L. Hölzl
Graz, 16. April 2020

MATLAB kann ein gutes Hilfsmittel im Umgang mit Signaltransformationen sein. Es hilft einem zwar nicht die Transformationen zu finden, doch kann man damit numerisch überprüfen, ob die Ergebnisse richtig sind. Im folgenden sind ein paar Hilfestellungen dazu gegeben; für die meisten Beispiele benötigt man die Control System Toolbox[1].

1 LAPLACE-Transformation

Zur Überprüfung, ob eine Funktion $\bar{f}(s)$ wirklich die LAPLACE-Transformierte einer Funktion $f(t)$ ist, können die zeitlichen Verläufe der beiden dargestellt und verglichen werden. Als Beispiel diene uns die Funktion $f(t) = te^{at}$; wir wollen überprüfen, ob

$$\bar{f}(s) = \frac{1}{(s-a)^2}$$

die zugehörige Transformierte ist.

MATLAB-Code 1: Darstellung des zeitlichen Verlaufs.

```
% Parameter a
a = -3; % Was passiert, wenn a > 0? ;-)

% Anlegen der Transformierten: 2 Möglichkeiten
%   (a) tf(b,a) mit Zähler- und Nennerpolynom
%   (b) zpk(z,p,k) mit Null- und Polstellen, Verstärkung
%
% Polynome: Vektoren mit Koeffizienten:
%   s^2 - 2*a*s + a^2 => [1 -2*a a^2]
fs = tf(1,[1 -2*a a^2]);
% oder
```

[1] Mit der freien Software GNU Octave sollten die Beispiele ebenfalls funktionieren; dies habe ich allerdings nicht getestet.

```
fs = zpk([],[a a],1);

% Umrechnung der Darstellungen
tf(fs)
zpk(fs)

% Ermitteln des zeitlichen Verlaufs y=f(t)
[y,t] = impulse(fs);

% Analytische Berechnung von f(t)
ft = t.*exp(a*t);

% Darstellung: (liegen die Verläufe übereinander?)
plot(t,y,t,ft,'r--');
```

Damit kann man natürlich auch den Grenzwert für $t \to \infty$ numerisch ermitteln.

MATLAB-Code 2: Endwert von Funktionen.

```
% Bsp 1: Der Grenzwert verschwindet
%   (Matlab führt die Kürzung selber durch!)
fs1 = zpk([3 -1],[-2 3 -4],1/3)
impulse(fs1); % ohne Wertrückgabe: öffnet neues Fenster

% Bsp 2: Die Funktion strebt gegen den Wert 2
%   (woran sieht man das ohne Matlab?)
fs2 = tf([1.5 4],[1 2 0])
impulse(fs2);

% Bsp 3: Grenzwert wächst über alle Schranken
%   (existiert also nicht)
fs3 = tf(1,[1 -0.00001])
impulse(fs3);
```

Hier gilt es zu beachten, daß bei `impulse` Anfangswerte stets zu null gesetzt werden. Sind sie verschieden von null, muß anstelle von `impulse` die Funktion `lsim` verwendet werden; dieser können Anfangswerte übergeben werden (siehe folgenden Abschnitt).

Mit MATLAB können auch Differentialgleichungen numerisch gelöst werden. Man betrachte die Gleichung (aus dem Skriptum)

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -x + u$$

mit Anfangswert $x(t) = x_0$ und Eingangsfunktion $u(t) = \sin \omega_0 t$. Hier müssen wir `lsim` aus zwei Gründen verwenden: zum einen wegen des Anfangswertes, zum anderen wegen der Eingangsfunktion. Die LAPLACE-Transformierte lautet (siehe Skriptum)

$$\bar{x}(s) = \frac{1}{s+1} x_0 + \frac{1}{s+1} \cdot \frac{\omega_0}{s^2 + \omega_0^2} .$$

MATLAB-Code 3: Lösen einer Differentialgleichung.

```
% Eingabe mittels ss(a,b,c,d)
%   ss erwartet: dx/dt = a*x + b*u
a = -1; b = 1;
xsys = ss(a,b,1,0); % warum c=1, d=0? -> andere LVs

% Anfangswert
x0 = 3;

% lsim benötigt u und den Zeitvektor t
t = (0:0.001:25)'; % Endzeitpunkt evtl. anpassen!
w0 = 1;
u = sin(w0*t);

xt = lsim(xsys,u,t,x0);

% Vergleich mit x(s)
xs = tf(1,[1 1])*x0 + tf(1,[1 1])*tf(w0,[1 0 w0^2]);
y = impulse(xs,t); % denselben Zeitvektor wie oben verwenden!

% Darstellung
plot(t,xt,t,y,'r--');
```

Der Befehl `ss` funktioniert auch für Differentialgleichungssysteme. Dann ist `x` ein Vektor und `a`, `b` Matrizen; die `1` muß durch die Einheitsmatrix ersetzt werden.

Berechnet man den Verlauf von $x(t)$ für mehrere (verschiedene) Anfangswerte und stellt diese nebeneinander dar, so erkennt man auch schnell die stationäre Lösung.

MATLAB-Code 4: Stationäre Lösung.

```
% erster Teil: siehe oben.
a = -1; b = 1;
xsys = ss(a,b,1,0);
```

```
t = (0:0.001:25)';
w0 = 1;
u = sin(w0*t);

% verschiedene Anfangswerte
x0a = 3;
x0b = -2;
x0c = 1;

xta = lsim(xsys,u,t,x0a);
xtb = lsim(xsys,u,t,x0b);
xtc = lsim(xsys,u,t,x0c);

% Analytische Lösung laut Skriptum
xst = 1/sqrt(1+w0^2)*sin(w0*t - atan(w0));

% Darstellen
plot(t,[xta xtb xtc],t,xst,'k--');
legend('x0,a','x0,b','x0,c','xst');
```

2 z-Transformation

Für die z-Transformation kann vieles aus dem vorigen Kapitel übernommen werden, nur daß man zusätzlich die Diskretisierungszeit T_d (also die Zeitdauer zwischen den äquidistanten Abtastzeitpunkten) angeben muß. Einen kleinen „Fallstrick“ gibt es jedoch: abhängig von der MATLAB-Version macht der Befehl `impulse` etwas unterschiedliches; daher empfehle ich generell die Verwendung von `lsim`.

Beispielhaft werden wir folgendes Transformationspaar überprüfen:

$$f_i = \sin i\omega T_\mathrm{d} \qquad \square\!\!-\!\!\blacksquare \qquad \bar{f}(z) = z\,\frac{\sin \omega T_\mathrm{d}}{z^2 - 2z\cos \omega T_\mathrm{d} + 1}\,.$$

MATLAB-Code 5: Verläufe für z-Transformation.

```
% Parameter
Td = 0.1;
w = 3;

% z-Bereich: entweder tf(b,a,Td) oder zpk(z,p,k,Td)
fz = tf([sin(w*Td) 0],[1 -2*cos(w*Td) 1],Td);
```

```
7
8  i = (0:30)'; % Endindex evtl. anpassen!
9  % statt impulse: lsim mit zeitdiskretem delta-Impuls
10 deltai = zeros(size(i)); deltai(1) = 1;
11
12 y = lsim(fz,deltai,i*Td); % t=i*Td!
13
14 % Zum Vergleich
15 fi = sin(i*w*Td);
16
17 % Darstellung
18 plot(i,fi,i,y,'r--');
19 % statt plot kann man auch stem verwenden:
20 %   (man sieht nur eine Funktion, da beide gleich)
21 stem(i,[fi y]);
```

Damit sollte man alle Aufgaben der Signaltransformationen überprüfen können. Manchmal kann es etwas trickreich sein f_i zu berechnen (insbesondere bei Rekursionsgleichungen und Verschiebungen[2]) – hier ist also Obacht geboten! Daher betrachten wir als abschließendes Beispiel die Rekursionsgleichung

$$f_{i+1} = -\frac{2}{3} f_i\,, \qquad i = 0, 1, \ldots\,,$$

mit $f_0 = 3$. Die Ermittlung der expliziten Darstellung und der z-Transformierten (und deren Überprüfung) lasse ich aus didaktischen Gründen hier aus.

MATLAB-Code 6: Rekursionsgleichung.

```
1 i = (0:30)';
2 fi = zeros(size(i));
3 fi(1) = 3; % Achtung: in Matlab beginnen Indizes bei 1!
4 for j=2:length(i)
5     fi(j) = -2/3*fi(j-1);
6 end
7
8 stem(i,fi);
```

[2] Für Verschiebungen mag es sinnvoll sein die Funktion σ_i als eigene MATLAB-Funktion zu definieren, bspw. so: `sigmai = @(i) max(0,min(i+1,1));` diese kann dann wie eine normale MATLAB-Funktion verwendet werden; σ_3 erhielte man bspw. mit `sigmai(3)`.