Signaltransformationen mit Matlab

Numerische Überprüfung von Ergebnissen

Stefan L. Hölzl
Graz, 27. April 2023

Matlab ist ein gutes Hilfsmittel im Umgang mit den Signaltransformationen. Es hilft einem zwar nicht, Transformationen zu finden, aber man kann 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 $f(t) = t\mathrm{e}^{at}$; 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
%     i) tf(b,a) mit Zähler- und Nennerpolynom
%    ii) 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]);
```

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

```
% oder
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 $t \longrightarrow \infty$ 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 die Anfangswerte bei `impulse` stets zu null gesetzt werden. Sind sie verschieden von null, bietet sich die Funktion `lsim` an; dieser können Anfangswerte übergeben werden (siehe den folgenden Abschnitt).

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

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -x + u\,, \qquad x(t=0) = x_0\,, \qquad u(t) = \sin\omega_0 t\,.$$

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)'; % 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 sind `x` und `b` durch Vektoren und `a` durch eine Matrix zu ersetzen; die `1` muß durch die Einheitsmatrix (`eye`) ersetzt werden.

Berechnet man den Verlauf von $x(t)$ für mehrere (verschiedene) Anfangswerte und stellt sie nebeneinander dar, so erkennt man 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, wobei zusätzlich die Diskretisierungszeit T_d (Zeit zwischen den Abtastzeitpunkten) angeben werden muß. Einen kleinen „Fallstrick“ gibt es jedoch: abhängig von der Matlab-Version verhält sich der Befehl `impulse` unterschiedlich; daher empfehle ich generell die Verwendung von `lsim`.

Beispielhaft werden wir folgendes Transformationspaar überprüfen:

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

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);

i = (0:30)'; % Endindex evtl. anpassen!
% statt impulse: lsim mit zeitdiskretem delta-Impuls
deltai = zeros(size(i)); deltai(1) = 1;

y = lsim(fz,deltai,i*Td); % t=i*Td!

% Zum Vergleich
fi = sin(i*w*Td);

% Darstellung
plot(i,fi,i,y,'r--');
% statt plot kann man auch stem verwenden:
%   (man sieht nur eine Funktion, da beide gleich)
stem(i,[fi y]);
```

Damit sollte man alle Aufgaben der Signaltransformationen überprüfen können. Bei Rekursionsgleichungen und Verschiebungen[2] kann es etwas trickreich sein, f_i zu berechnen. Dazu ein abschließendes Beispiel:

$$f_{i+1} = -\frac{2}{3} f_i \,, \qquad f_0 = 3 \,, \qquad i = 0, 1, \ldots \,.$$

Die Ermittlung der z-Transformierten und der expliziten Darstellung von f_i (und deren Überprüfung) lasse ich aus didaktischen Gründen weg.

Matlab-Code 6: Rekursionsgleichung.

```
i = (0:30)';
fi = zeros(size(i));
fi(1) = 3; % Achtung: in Matlab beginnen Indizes bei 1!
for k=2:length(i)
    fi(k) = -2/3*fi(k-1);
end

stem(i,fi);
```

[2] Für Verschiebungen ist es sinnvoll, die Funktion σ_i als Matlab-Funktion zu definieren, z. B. so: `sigma = @(i) double(i>=0);` σ_{23} erhält man dann mit `sigma(23)`.