Laboratory

LTI System Modeling Using Matlab

Prof. Dr. Martin J. W. Schubert

Electronics Laboratory

Regensburg University of Applied Sciences

Regensburg

Abstract. After a short introduction to the required Matlab commands linear and time-invariant (LTI) Matlab models are introduced and examples given.

1 Introduction

Matlab (Matrix Laboratory) [1], [2], [3] is a useful tool for many technical applications and also for the investigation of linear and time-invariant (LTI) systems, typically modeled in the s and z domain.

Availability of the software. Matlab is not free. Open source freeware with same functionality (except some toolboxes) can be obtained from Scilab (Scientific Laboratory) [4] or Octave [5], [6], [7]. As graphics software gnuplot [8] or jhandles [9] may be added. The latter is based on Java and more similar to Matlab, gnuplot is faster (and some people think more beautiful).

Digital filters should – with respect to filter quality – be directly designed in the z domain instead of being translated from s to z. The s $\rightarrow$ z translation technique, demonstrated in section 4.3, is recommended for control systems only.

The organization of this laboratory is as follows:

Chapter 1 introduction.

Chapter 2 introduce some basic theory about the coherence of Laplace s-domain and z-domain models.

Chapter 3 introduces the required Matlab commands.

Chapter 4 demonstrates how to write self-made Bode diagrams in *s* and *z*.

Chapter 5 is an extraction of the “LTI Systems” chapter of the Matlab book [3].

Chapter 6 demonstrates how the functionality of the Bode-command can be realized with some self-made Matlab statements.

Chapter 7 draws relevant conclusion,

Chapter 8 offers some references.

2 Theory

2.1 Application Fields for Laplace and z-Transformation

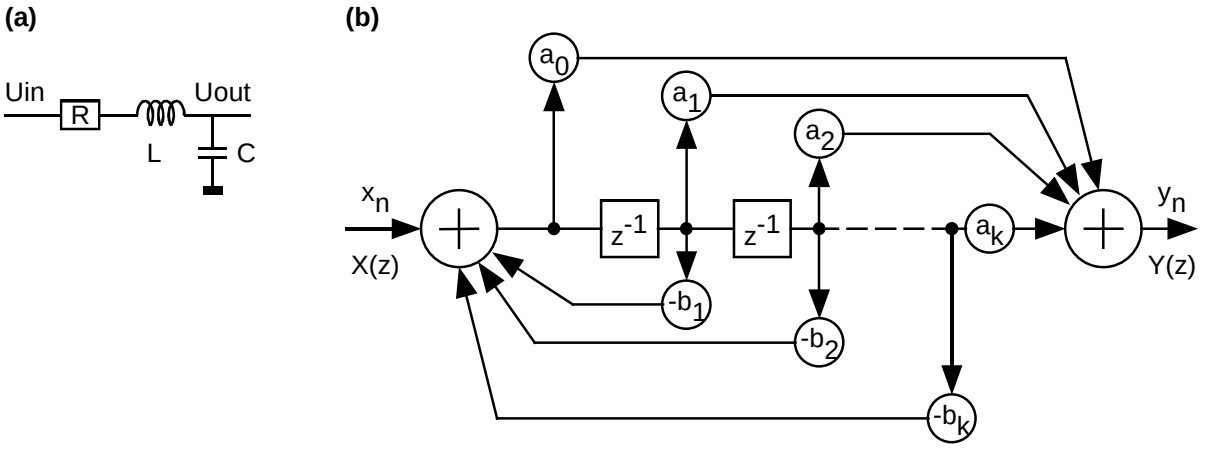

Fig. 2.1-1: (a) Time-continuous filter, **(b)** time-discrete filter

Fig. 2.1-1(a) illustrates a time-continuous and a time-discrete filter typically modeled with Laplace transformation using the impedances *sL* for an inductor and *1/sC* for a capacitor. The transfer function is obtained with $s=j\omega$.

Fig. 2.1-1 (b) illustrates a filter type using boxes with transfer function z^{-1}. These boxes do nothing else than delaying all frequencies by the same amount of time, typically termed *T*. Seems simple, doesn't it?

- Can you construct an analog circuit delaying all frequencies by the same amount of time?

An approximation for the z^{-1} elements in Fig. 2.1-1(b) could be wires of same length. In the digital world delay elements are realized by memory as illustrated in Fig. 2.1-2.

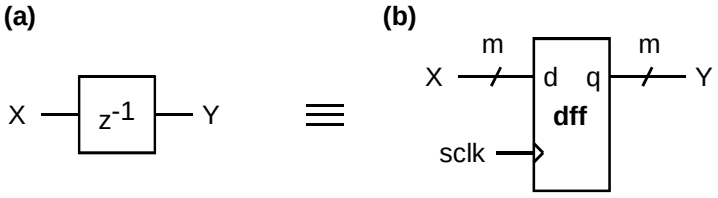

Fig. 2.1-2: (a) Delay element and **(b)** digital realization

z^{-1} in the figure above delays the data samples by integral multiples of the sampling clock interval, $T=1/f_s$, with f_s being the sampling frequency.

Although z^{-1} is not bound to time-discrete systems the general rule of thumb applies in at least 99% of all cases:

Describe time-continuous models using *s* and time-discrete models using *z*.

2.2 The Relationship Between *s* and *z*

Let $X(j\omega)$ be the Fourier transformed of the time domain signal $x(t)$: $X(j\omega)=F\{x(t)\}$.

- What happens to $X(j\omega)$ when $x(t)$ is delayed by T becoming $x(t-T)$?

It is shown in the appendix that the time-domain delay T corresponds to a frequency-domain phase shift $\varphi = e^{-j\omega T}$. Using $z=e^{j\omega T}$ we can model that as multiplication of $X(j\omega)$ by z^{-1}:

When	$F\{x(t)\}$	$= X(j\omega)$	**with F being the Fourier Transformation,**
then	$F\{x(t-T)\}$	$= X(j\omega) \cdot z^{-1}$	**with $z = e^{j\omega T}$.**

The relationship between Laplace variable $\boldsymbol{z = e^{j\omega T}}$ is illustrated in Fig. 2.2. We define the relative frequencies $F=f/f_s=fT$ and $\Omega=\omega/f_s=\omega T$. Some observations:

- Frequency f=0 corresponds to z=1, frequency $f=½f_s$ ⇔ F=½ ⇔ Ω=π corresponds to z=-1.
- The behavior of $|H(z)|$ becomes periodic for $F>½$.
- The $j\omega$-axis becomes the unit circle.
- Stable systems in the Laplace domain s have all their poles s_p in the left half-plane: σ<0.
- Stable systems in the z domain have all their poles z_p within the unit circle: $|z_p|<1$.

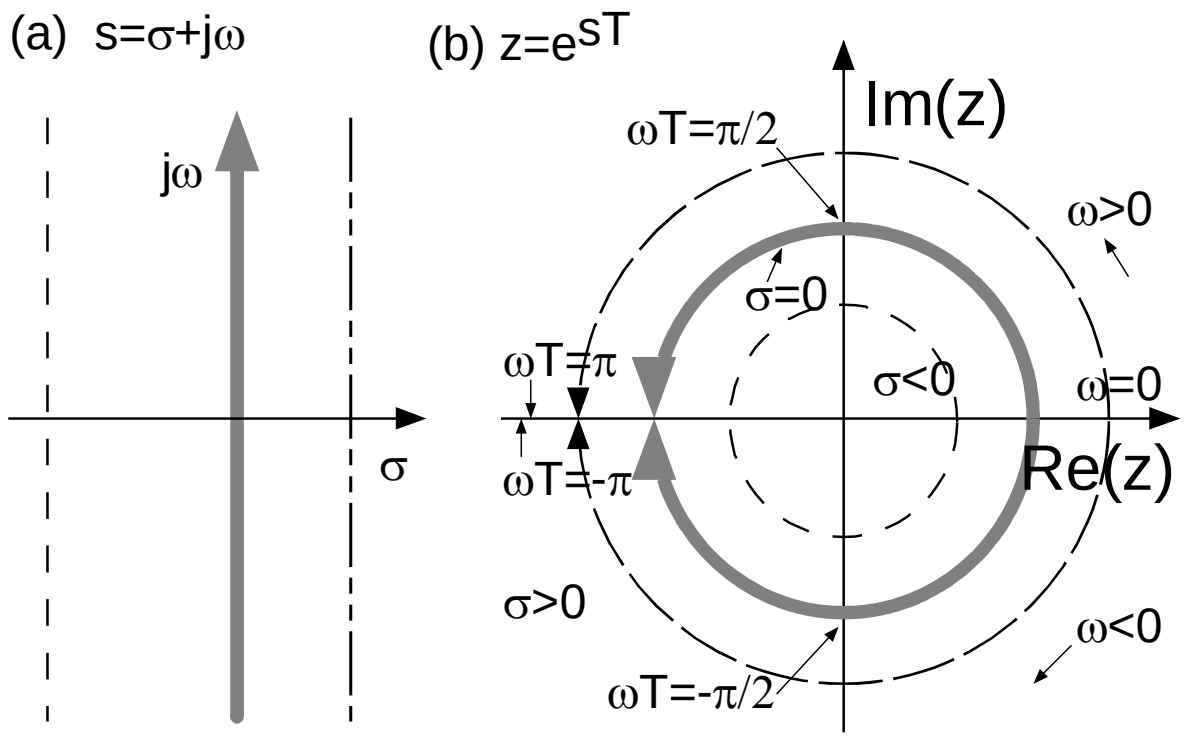

Fig. 2.2: (a) Laplace domain s and **(b)** corresponding z-domain.

3 Using Matlab

3.1 Matlab Basics

Getting Help:

To get help e.g. for the command sqrt type:

```
> help sqrt
```

Scalars:

Start Matlab on your computer. Type into the Matlab Command Window:

```
> a=3
> A = a*a
> a
```

Note: Matlab is case sensitive: **a ≠ A**.
A **;** (semicolon) after a command suppresses echo on screen.
A **%** (percent) sign comments the rest of the line.

Predefined constants: to check for **pi=π** and **i=**$\sqrt{-1}$ type

```
> pi
> i^2
```

Its easy to redefine them:

```
> i=0:10
> j=sqrt(-1)  % sqrt = square root
> j, j^2
```

Directories and Files:

Type **ls** to list directory. "**.**" is the actual working directory that can be resolved with **pwd** and "**..**" is the actual parent directory in the hierarchy:

```
> pwd  % treename of actual working directory
> ls   % list directory
```

Go to a directory of your choice using change directory (cd):

```
> cd <treename>   % use a valid treename for <treename>
```

Open an editor window:

```
> edit % an editor window will open
```

Write into the editor window

```
function y = square(x)
y = x*x;
```

Save it with filename *square.m*. Using command **ls** you should be able to observe this file. Type into the Matlab Command Window:

```
> square(5)
```

You can save all the following commands in a command file. There is no need to begin it as function.

Vectors:

Define a vector using start:step:stop.

```
> vec1 = -5:5
```

Step=1 is default and can be omitted.

```
> vec2 = -10:2:10
```

Use brackets to create a composed data object (corresponds to a C struct):

```
> vec3 = [-3 5 3 2 2 2 6]
```

Define a frequency axis:

```
> vec3 = [-3 5 3 2 2 2 6]
```

Operations with Vectors:

Try the following commands and explain them:

```
> vec = 1:3
> vec
> vec'
> vec*vec
> vec*vec'
> vec.*vec
> x = -10:0.5:10;
> y1=x.*x.*x;
> y2=x^3;
> y2=x.^3;
```

Plot Commands:

```
> plot(x,y1)
> grid on
> plot(x,y1,x,y2+100); grid on
```

If the plot command has only one argument vector of real elements only, Matlab uses the index as abscissa. A Matlab vector does always begin with index 1:

```
> plot(y1)
> stem(y1), grid on
> hold on; plot(y1); hold off;
```

If the plot command has only one argument vector with complex elements, Matlab uses abscissa for the real and the ordinate for the imaginary part of the numbers. By default letter i is the square root of –1:

```
> F=[0:8]/8
> z=exp(i*2*pi*F)  % you will get complex numbers now
> plot(z)
```

4 Computing Frequency Responses of LTI Models

4.1 Time-Continuous Modeling

4.1.1 Given Time-Continuous System of 2nd Order

IN the time-continuous domain the Fourier transformation

$$X(j\omega) = F\{x(t)\} = \int_{-\infty}^{\infty} x(t) \cdot e^{-j\omega t} dt \qquad (4.1\text{-}1)$$

is useful but has some problems, amongst others with convergence. Such problems are ameliorated by Laplace transformation

$$X(s) = L\{x(t)\} = \int_{-\infty}^{\infty} x(t) \cdot e^{-st} dt \quad \text{with} \quad s = \alpha + j\omega. \qquad (4.1\text{-}2)$$

Using the so-called Doetsch symbol we write $x(t) \circ\!\!-\!\!\bullet\, X(s)$ and $x'(t) \circ\!\!-\!\!\bullet\, s \cdot X(s)$. In the s-domain symbol the impedance of a capacitor C is expressed as $1/sC$ and the impedance of an inductor L as sL.

Application example:
Bode diagram of a 2nd order transfer function, A_0: DC-amp., f_0 cut-off freq., D damping par.:

$$H(s) = \frac{Y(s)}{X(s)} = \frac{A_0 \omega_0^2}{s^2 + 2D\omega_0 s + \omega_0^2} \qquad (4.3)$$

Listing 4.1-1: Function *f_dB* in file *f_dB.m*:

```
function dB = f_dB(x)
dB = 20*log10(x);
```

Listing 4.1-2: Generate Bode diag. of *H(s)*

```
% Bode diagram
f  = 0:1:10000;
A0 = 1;
D  = 0.1;
f0 = 100;
j  = sqrt(-1);
s  = j*2*pi*f;
omega0 = 2*pi*f0;
Hs = A0*omega0^2./(s.^2  +2*D*omega0*s  +omega0^2);

subplot(211); semilogx(f,20*log10(abs(Hs)));
grid on; ylabel('Amplitude [dB]');

subplot(212); semilogx(f,angle(Hs)*180/pi);
grid on; ylabel('Phase [°]');
```

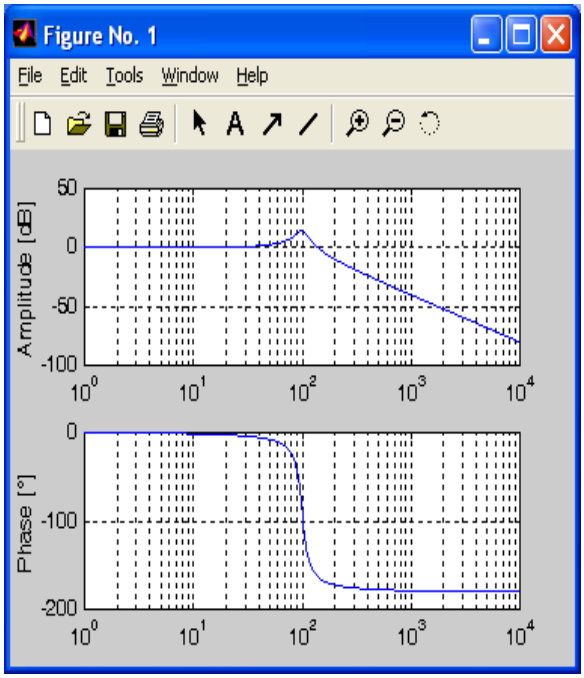

Fig. 4.1: Bode diagram generated with listings 4.1-1 and 4.1.2.

Applying a Matlab operator element wise: Let x =[1 2 3], then 1./x=[1 1/2 1/3], x.^2=[1 4 9].

4.1.2 Time-Continuous System of 2nd Order

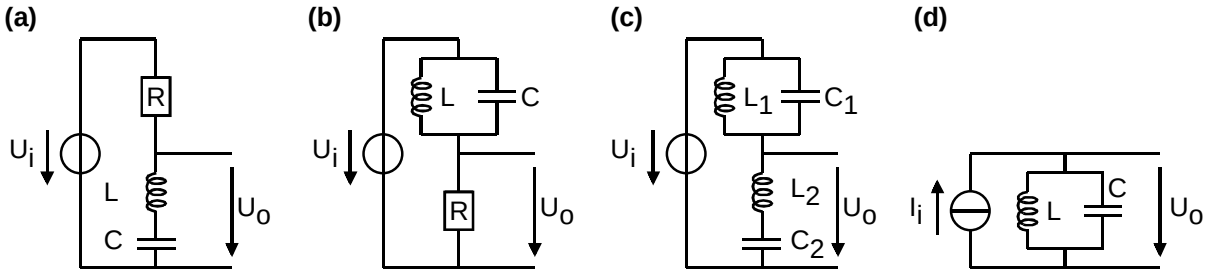

Fig. 4.1.2: Time-continuous circuits with zeros and poles on the jω axis.

Prepare the formulae at home. In the laboratory spend maximal a ½ hour on this sub-chapter.

Compute poles and zeros for the 4 circuits in the Fig. above. Start with R=1KΩ, L=1μH, C=1μF. Then cou can modifiy the values.

Simulate the circuit and demonstrate with Spice or Matlab:

- A zero on the jω axis is a notch in the zero frequency.
- A pole on the jω axis is an oscillator in the pole frequency.

(PS: The author had problems with LTspice, most probably due to round-off errors.)

4.2 Time-Discrete Modeling

4.2.1 Using the Discrete Fourier Transformation

Let $x[n]$ be a time-discrete function $x[n]=x(t_n)$ with $t_n=n\cdot T_s$ where n=0...N-1 and T_s sampling interval, $f_s=1/T_s$ sampling frequency. In this case we have to apply the Discrete Fourier Transformation (DFT) defined by

$$X[f_k]=F\{x[n]\}=\sum_{n=0}^{N-1}x[n]\,e^{-j2\pi kn/N} \quad \text{with} \quad f_k=\frac{k}{N}f_s\ , \quad k=0...N\text{-}1. \tag{4.2.1}$$

A particular form of the DFT is the numerically very efficient Fast Fourier Transform (FFT) used as function *fft* in the listing below.

Listing 4.2.1: Matlab code of figure right. Top-down: Lowpass impulse response *hlp*, linear and logarithmic transfer function *fft(hlp)*, test-input signal *x* and filtered output signal *y*.

```
% Bode plot by FFT
Order=20;
Fg=0.2;
% Lowpass without window function:
hlp=sinc(2*Fg*[-Order/2:Order/2]);
hlp=hlp/sum(hlp);

figure(1)
subplot(511)
stem(hlp); grid on;
hold on; plot(zeros(1,Order+1));
hold off; ylabel('hlp');
axis([1 Order+1 -0.2 0.5]);

%------- computation of FFT ----------
F  = (0:Order)/Order;
Hlp = fft(hlp);
%---------------------------------------

subplot(512)
plot(F,abs(Hlp)); grid on;
axis([0 1 -0.1 1.2]); ylabel('|Hlp|');

subplot(513)
plot(F,f_dB(abs(Hlp))); grid on;
axis([0 1 -60 10]); hold off;
ylabel('|Hlp| [dB]');

t=0:100; F_low=0.1; F_high=0.4321;
x=sin(2*pi*F_low*t)+sin(2*pi*F_high*t);
y=conv(x,hlp);

subplot(514); plot(t,x);
grid on; ylabel('FilterIn');

subplot(515); plot(t,y(1:length(t)));
grid on; ylabel('FilterOut');
```

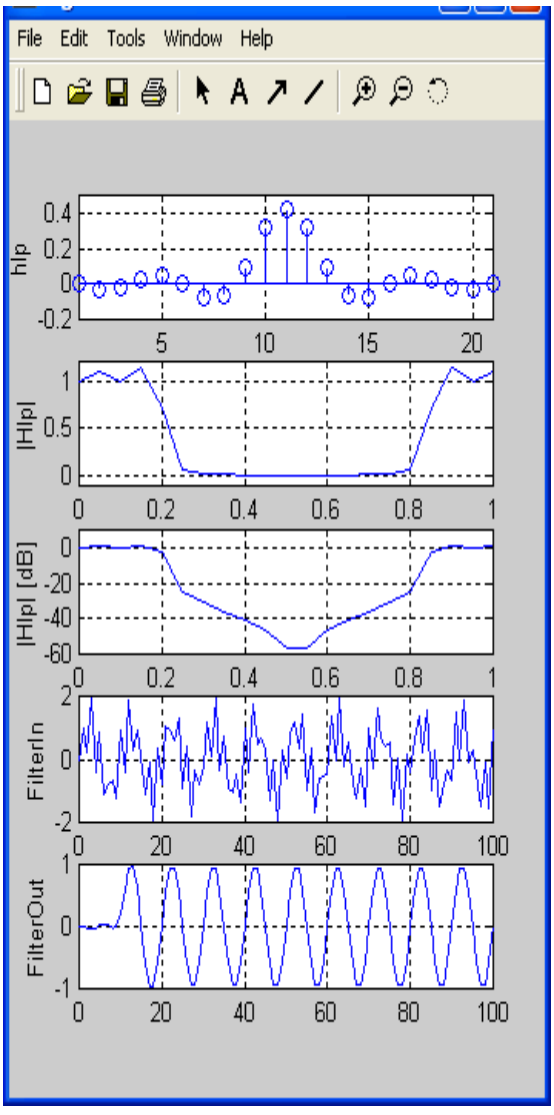

Fig. 4.2.1: Matlab response to listing 4.2.1: Top - down: Lowpass time-domain impulse response, frequency domain responses linear and in dB, input signal, filtered output.

Explanations on the Matlab code used:

- Matlab function *conv(x,hlp)* performs a convolution *x*y*, which has the length of *length(x)+length(y)*-1.
- Function *f_dB* was declared in listing 4.1.1.
- Matlab function *fft(x)* performs a fast Fourier transformation delivering complex numbers.
- Matlab function *abs(fft(x))* delivers the amplitude curve within Bode diagram.
- Matlab function *angle(fft(x))* will deliver the phase information of the Bode diagram.
- The filter is obviously poor but it separates *F_low<Fg* from *F_high>Fg* quite well as illustrated in the two lowest subplots.
- *plot(x,y)* plots a line connecting points (x_i,y_i) of the two vectors *x, y* of same length.
- *plot(Ax,Ay,Bx,By,Cx,Cy)* plots three lines connecting points (Ax_i,Ay_i), (Bx_i,By_i), (Cx_i,Cy_i).
- *plot(r)* with *r* being a vector of real numbers uses the indices of *r*, 1,2,...*length(r)*, as abscissa.
- *plot(c)* with *c* being a vector of complex numbers plots in the complex plane (Re{*c*},Im{*c*}).

Problems:

- The DFT is defined for periodic functions only. This problem can be overcome by assuming that the actual data vector is repeated periodically. However, this bears impurities if such a periodic repetition of the data vector disturbs the harmonic behavior of the represented frequencies. It is nearly impossible to avoid this problem, since the recorded frequencies may be unknown before application of the DFT. A possible amelioration of this problem can be obtained by the application of window functions. Furthermore, the DFT requires high computational effort.

- The FFT requires that the data vector consists of $N=2^M$ samples with *M* being an integer. If this condition is not fulfilled, the rest of the data vector has to be filled, e.g. with zeros.

- Note that *N* time-domain values deliver *N* frequency-domain values. Consequently the resolution of the transfer function in Fig. 4.2.1 is quite rough and cannot be improved.

4.2.2 Using the *z* Transformation

Let $h(n) = a_0, a_1, a_2, \ldots a_k$ be the impulse response of a time-discrete filter and $x(n)=x(t_n)$ a sampled waveform, whereat $t_n=n \cdot T$ with T sampling interval. Then y(n) computes as convolution

$$y(n) = h(n) * x(n) = a_0 x(n) + a_1 x(n-1) + a_2 x(n-2) + \ldots + a_k x(n-k) = \sum_{i=0}^{k} a_i x(n-i) .$$

Convolution in one domain (here time) corresponds to a multiplication in the other domain, here frequency represented by $z=e^{sT}$:

$$Y(z) = a_0 X(z) + a_1 z^{-1} X(z) + a_2 z^{-2} X(z) + \ldots + a_k z^{-k} X(z) = X(z) \sum_{i=0}^{k} a_i z^{-i} = H(z) \cdot X(z)$$

and consequently $H(z) = \sum_{i=0}^{k} a_i z^{-i}$. (4.2.2)

The only differences between listings 4.2.1 and 4.2.2 is (except the 1st comment line) in the computation of *Hlp* and the relative frequency *F*, which has significantly more points for the *z* transformation. Figures 4.2.1 ad 4.2.2 differ only in the 2nd and 3rd subplot. Here the higher resolution of the *z* transformation becomes visible. Furthermore a need to fill the *hlp* vector with zeros to 2^M taps as done for the FFT is not given for the *z* transformation.

For recursive filters we have

$$H(z) = \frac{\sum_{i=0}^{k} a_i z^{-i}}{\sum_{j=1}^{k} b_i z^{-j}} = \frac{A(z)}{B(z)}$$

In this case compute the polynomials *A(z)* and *B(z)* and divide them.

Listing 4.2.2: Matlab code of figure right. Top-down: Lowpass impulse response *hlp*, linear and logarithmic transfer function *Hz=Z{hlp}*, test-input signal *x* and filtered output signal *y*.

```
% Bode plot by z Transform
Order=20;
Fg=0.2;
% Lowpass without window function:
hlp=sinc(2*Fg*[-Order/2:Order/2]);
hlp=hlp/sum(hlp);

figure(2)
subplot(511)
stem(hlp); grid on;
hold on; plot(zeros(1,Order+1));
hold off; ylabel('hlp');
axis([1 Order+1 -0.2 0.5]);

%---computation of z Transformation---
F=0:1e-5:1;
j=sqrt(-1);
z=exp(j*2*pi*F);
Hz = zeros(1,length(F));
for i=0:Order;
   Hz = Hz + hlp(i+1)*z.^-i;
end;
Hlp = Hz;
%-------------------------------------

subplot(512)
f=(0:Order)/Order;
plot(F,abs(Hlp)); grid on;
axis([0 1 -0.1 1.2]); ylabel('|Hlp|');

subplot(513)
plot(F,f_dB(abs(Hlp))); grid on;
axis([0 1 -60 10]); hold off;
ylabel('|Hlp| [dB]');

t=0:100; F_low=0.1; F_high=0.4321;
x=sin(2*pi*F_low*t)+sin(2*pi*F_high*t);
y=conv(x,hlp);

subplot(514); plot(t,x);
grid on; ylabel('FilterIn');

subplot(515); plot(t,y(1:length(t)));
grid on; ylabel('FilterOut');
```

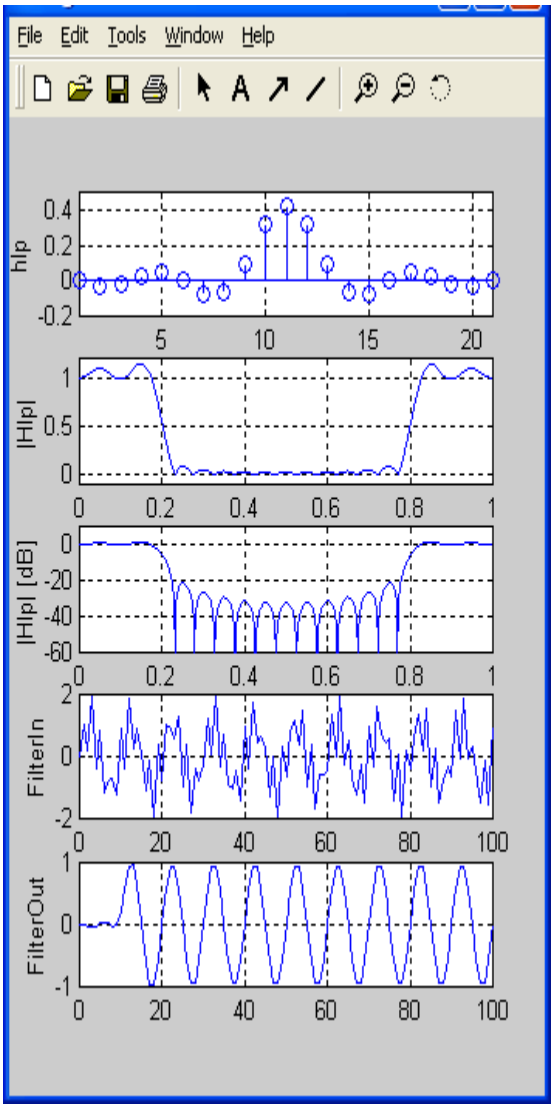

Fig. 4.2.2: Matlab response to listing 4.2.1: Top - down: Lowpass time-domain impulse response, frequency domain responses linear and in dB, input signal, filtered output.

5 Matlab's Linear and Time-Invariant (LTI) Systems

5.1 Time-Continuous Systems

Table 4.1: Representation of time-continuous LTI systems:

Command		**description**	
TF	**tf(***num***,***den***)**	Transfer Function	Polinomials in the Laplace variable *s*
ZPK	**zpk(***z***,***p***,***k***)**	Zero-Pole-Gain	Pole-Zero-Diagram

5.1.1 Laplace Transfer Function Representation: `tf(num,denom)`

General description in the Laplace domain: $H(s) = \frac{num(s)}{den(s)} = \frac{a_m s^m + ... + a_1 s^1 + a_0}{b_n s^n + ... + b_1 s^1 + b_0}$

num: numerator (deutsch: Zähler), *den*: denominator (deutsch: Nenner)

```
> num_vector = [am ... a1 a0]
> den_vector = [bn ... b1 b0]
> Hs = tf(num_vector, den_vector)
```

Example $H_1(s) = \frac{s+10}{s^3 + 2s^2 + 2s + 1}$

```
> Hs1 = tf([1 10], [1 2 2 1])

    Transfer function:
           s + 10
    ---------------------
    s^3 + 2 s^2 + 2 s + 1
```

Alternatively:

```
> s = tf('s')          % declare s to be the Laplace variable

    Tansfer function
         s

> Hs2 = 20/((s^2 + 0.5*s + 1)*(s+1))

    transfer function:
              20
    -------------------------
    s^3 + 1.5 s^2 + 1.5 s + 1
```

Observe Bode diagram, step and impulse response of these functions:

```
> bode(Hs1,Hs2)
> step(Hs1,Hs2)
> impulse(Hs1,Hs2)
```

5.1.2 Laplace Pole-Zero-Gain Representation: `zpk(z,p,k)`

General pole-zero-gain representation in s: $H(s) = k\frac{(s-s_{n,1})\cdot(s-s_{n,2})\cdot\ldots\cdot(s-s_{n,m})}{(s-s_{p,1})\cdot(s-s_{p,2})\cdot\ldots\cdot(s-s_{p,n})}$

Example: $H_{TF}(s) = \frac{3s-3}{s^2+1.1s+0.3}$ $\Leftrightarrow$ $H_{ZPK}(s) = 3\frac{s-1}{(s+0.5)(s+0.6)}$

Matlab input:

```
> Htf=tf([3 -3], [1 11 30]);
> Hzpk=zpk([1], [-5 -6], 3);
```

Alternatively: declare **s** as Laplace-Variable and write polynomial using +, -, *, /, ^:

```
> s = tf('s')    % s als Laplace-Variable deklarieren
> Hs1 = (3*s-3) / (s^2 + 11*s + 30)
> Hs2 = 3*(s-1) / ((s+5)*(s+6))
```

5.1.3 Switching the Representation: `zpk(tf)` the `tf(zpk)`:

```
> Htf_of_Hzpk = tf(Hzpk)
> Hzpk_of_Htf = zpk(Htf)
```

Note: The latter example illustrates a method to find the nulls of a polynomial.

A conjugate-complex pole-pair comes as 2nd order polynomial. Examples for 2nd and 3rd order Butterworth lowpasses. Try:

```
> Hbw2 = tf([1],[1 sqrt(2) 1]), bode(Hbw2)
> Hbw2_zpk = zpk(Hbw2), bode(Hbw2_zpk)
> Hbw3 = tf([1],[1 2 2 1]), bode(Hbw3)
> Hbw3_zpk = zpk(Hbw3), bode(Hbw3_zpk)
> bode(Hbw2,Hbw2_zpk,Hbw3,Hbw3_zpk)
```

5.2 Time-Discrete Systems

5.2.1 Declaring LTI Systems in the z-Plane

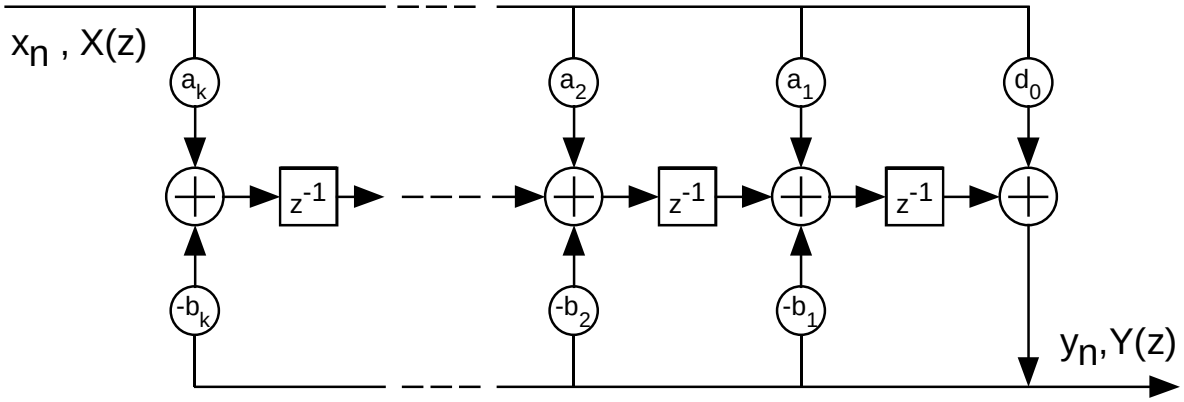

Fig. 4.2.1: Time-discrete filter in the first canonical direct structure

Figs. 2.1-1(b) shows the first and Fig. 4.2.1 the second canonical direct structures of the time-discrete filter. The coefficients are directly visible in the polynomial representation as shown on the left hand side of the example below: On the right hand side we see the polynomials factorized into their poles and nulls.

Table 4.2: Representation of time-discrete LTI systems:

Model Example:	$H_{TF}(z) = \frac{3z+3}{z^2+0.11z+0.30}$	$H_{ZPK}(z) = 3\frac{z+1}{(z+0.5)(z+0.6)}$
Matlab:	`Htf = tf([3 3],[1 0.11 0.30],Ts)`	`Hzpk=zpk([-1],[-0.5 –0.6],3,Ts)`

The time-discrete model is indicated by the additional delay T_s caused by a delay element z^{-1}. If you do not want define a particular time set T_s=-1. Matlab input examples:

```
> % generate transfer function using tf and zpk:
> Htf  = tf([3 3], [1 0.11 0.30], -1)    % Ts undefined
> Hzpk = zpk([-1], [-0.5 –0.6], 3, 1e-4) % Ts=1/10KHz

> % Translation into the other model type:
> Htf_from_Hzpk = tf(Hzpk)
> Hzpk_from_Htf = zpk(Htf)
```

5.3 System Translation Between s- and z-Planes

Table 4.3: Matlab commands for changing the domain of an LTI system:

c2d	continuous to discrete	z-domain approximation from s-domain TF
d2c	discrete to continuous	s-domain approximation from z-domain TF
d2d	discrete to discrete	Sampling-rate change within z-domain

The general model of a second-order system is $STF_{2,general} = \frac{A_0}{s'^2+2Ds'+1} = \frac{A_0\omega_0^2}{s^2+2D\omega_0 s+\omega_0^2}$ with $s'=s/\omega_0$. To observe the impact of DC amplification A_0, cutoff frequency ω_0 and stability parameter D we observe the system for different cases of D (you'd better write it into a file):

```
A0 = 10;            % DC amplification
w0 = 1000;          % cutoff frequency in rad
D_osc=0.1;          % oscillating case
D_pm45=0.5;         % 45° phase-margin case
D_bw = sqrt(2);     % Butterworth case
D_dblim=1;          % dead-beat (aperiodic) limit case
D_creep=10;         % creep case

% compute Laplace-domain transfer functions
Hs_osc = tf([A0*w0^2], [1 2*D_osc*w0 w0^2]);
Hs_pm45 = tf([A0*w0^2], [1 2*D_pm45*w0 w0^2]);
Hs_bw = tf([A0*w0^2], [1 2*D_bw*w0 w0^2]);
Hs_dblim = tf([A0*w0^2], [1 2*D_dblim*w0 w0^2]);
Hs_creep = tf([A0*w0^2], [1 2*D_creep*w0 w0^2]);
% display Laplace-domain transfer functions
bode(Hs_osc,Hs_pm45,Hs_bw,Hs_dblim,Hs_creep);
```

c2d: Translate the transfer functions (TF) above into the time-discrete domain. (We'll now get several graphics windows over each other.):

```
% compute and display Laplace- and z-domain transfer functions
% "figure(x)" clears / creates plot sheet x
Hz_osc   = c2d(Hs_osc,1e-4);   figure(2); bode(Hs_osc,Hz_osc);
Hz_pm45  = c2d(Hs_pm45,1e-4);  figure(3); bode(Hs_pm45,Hz_pm45);
Hz_bw    = c2d(Hs_bw,1e-4);    figure(4); bode(Hs_bw,Hz_bw);
Hz_dblim = c2d(Hs_bw,1e-4);    figure(6); bode(Hs_dblim,Hz_dblim);
% "figure" creates a new plot sheet with the next free number ("handle")
Hz_creep     = c2d(Hs_creep,1e-4); figure   ; bode(Hs_creep,Hz_creep);
figure(6), step(Hs_osc,Hz_osc); figure(7), impulse(Hs_osc,Hz_osc);
```

d2c: Translate **Hz_osc** back from z- to s-domain:

```
% Translate Hz_osc back from z- to s-domain:
Hs_osc_back = d2c(Hz_osc), figure(8), bode(Hz_osc,Hs_osc_back);
```

d2d: Change sampling rate. The numerator (deutsch: Zähler) must not contain real numbers. Example for a digital integrator model:

```
% increase sampling rate x 10 using d2d: (no real numbers in numerator!)
Hz_int1 = tf([1 0],[1 -1],1e-1), figure(9), bode(Hz_int1) % dig. integrator
Hz_int2 = d2d(Hz_int1,1e-2),     figure(10), bode(Hz_int1,Hz_int2)
```

6 Self-Made Bode Diagram

This part is for particularly interested students only. There is no need to work it through or understand it.

The **Bode** command demonstrated above is powerful but the user might prefer the frequency axis in Hz rather than in rad and may want to label and rescale the axis. For this purpose, a selfmade Bode diagram will be presented in this section.

Assuming a time-domain model without feedback in the form

$$y_n = a_1\, x_{n-1} + a_2\, x_{n-2} + a_3\, x_{n-3} + \ldots + a_{c-2}\, x_{n-(c-2)} + a_{n-1}\, x_{n-(c-1)}\, a_c\, x_{n-c}$$

It has the z-domain transfer function

$$H(z) = a_1 \cdot z^{-1} + a_2 \cdot z^{-2} + \; a_3 \cdot z^{-3} + \; \ldots\, a_{c-2} \cdot z^{-c+2} + \; + a_{c-1} \cdot z^{-c+1} + \; a_c \cdot z^{-c}$$

In the Matlab model below the impulse response of the model to be investigated is defined by hn. In the example below it is a moving averager computing the average of the last 50 input samples.

```
% function y=f_wv(hn);          % function version

% hn: time-domain impulse response: moving averager
hn=ones(1,50); hn=hn/sum(hn);   % comment this line for other versions

% hn: time-domain impulse response: comb-filter with alpha=1
hn=zeros(1,50); hn(1)=1; hn(50)=1;   % comment this line for other versions

% hn: time-domain impulse response: lowpass
Order=50; Fg=0.05; tau=-Order/2:Order/2;     % comment this line if necess.
hn=sinc(2*Fg*tau); hn=hn.*blackman(Order+1)'; hn=hn/sum(hn); % comment if n

cTaps=length(hn);
Fstep=0.0001;
Fmax=0.5;
F=0:Fstep:Fmax;
Hz=zeros(1,length(F));
zn=exp(-i*2*pi*F); % = 1/z
for j=1:length(hn);
  Hz=Hz+hn(j)*(zn.^j);
end;
subplot(411); stem(hn); axis([-5,55,min(hn),max(hn)*1.1])
grid on; ylabel('imp. resp. h(i)'); % xlabel('linear n');
subplot(412); semilogx(F,abs(Hz));
     axis([0,Fmax,min(abs(Hz)),max(abs(Hz))*1.1])
grid on; ylabel('linear H(z)'); % xlabel('log F ');
subplot(413); semilogx(F,20*log10(abs(Hz))); axis([0,Fmax,-100,10])
grid on; ylabel('dB(H(z))'); % xlabel('log F ');
subplot(414); semilogx(F,angle(Hz));
grid on; ylabel('angle(H(z))'); % xlabel('log F ');
```

Some more Matlab commands used above:

- **`exp(x) :`** = e^x.
- **`subplot(x,y,z)`** or **`Subplot(xyz)`** activates field z after dividing the plot sheet into x columns and y rows.
- **`grid on, grid off:`** switches a axis-oriented grid on in the plot field
- **`semilogx(…), semology(…):`** like plot, but with lagarithmic x-/y-axis
- **`title(…) :`** write a title over the plot
- **`xlabel(…), ylabel(…) :`** writes a label to x-/y-axis
- **`axis([xmin,xmax,ymin,ymax]) :`** user defined axis ranges

7 Conclusion

Assuming Laplace transform to be known the coherence of s and z was details. Some fundamental Matlab commands were introduced for beginners. Matlab LTI models were introduced. A proposal for something like a selfmade Bode diagram was proposed for interested, advanced students.

8 References

[1] Available: http://www.mathworks.com/, http://www.mathworks.de/.

[2] M. Schubert, "Zusammenfassung von Matlab-Anweisungen", Available: http://homepages.hs-regensburg.de/~scm39115/homepage/education/education.htm.

[3] A. Angermann, M. Beuschel, M. Rau, U. Wohlfarth: „Matlab – Simulink – Stateflow, Grundlagen, Toolboxen, Beispiele“, Oldenbourg Verlag, ISBN 3-486-57719-0, 4. Auflage (für Matlab Version 7.0.1, Release 14 mit Service Pack 1). URL: http://www.matlabbuch.de/

[4] Available: http://www.scilab.org/.

[5] Available: http://www.octave.org.

[6] Available http://www.gnu.org/software/octave/.

[7] Windows installer, available: http://octave.sourceforge.net/.

[8] Official gnuplot documentation. Avail.: http://www.gnuplot.info/documentation.html.

[9] Available: http://www.jhandles.net/.

9 Appendix A

Claim: When $F\{x(t)\} = X(j\omega)$ with $F\{\}$ being the Fourier Transformation,
Then $F\{x(t-T)\} = X(j\omega) \cdot z^{-1}$ with $z = e^{j\omega T}$.

Proof:

Fourier is defined as: $X(j\omega) = F\{x(t)\} = \int_{t=-\infty}^{+\infty} x(t) \cdot e^{-j\omega t} dt$

Using the substitution $t'=t-T \rightarrow t=t'+T \rightarrow dt=dt'$ delivers

$$F\{x(t-T)\} = \int_{t=-\infty}^{+\infty} x(t-T) \cdot e^{-j\omega t} dt = \int_{t'=-\infty+T}^{+\infty+T} x(t') \cdot e^{-j\omega(t'+T)} dt' = \int_{t'=-\infty}^{+\infty} x(t') \cdot e^{-j\omega t'} e^{-j\omega T} dt' = X(j\omega) \cdot z^{-1}$$