E E 2 7 5 Lab January 15, 2012

Lab 4. LTI Systems, the Z-Transform, and an Introduction to Filtering

Linear Time Invariant Systems

$$x[n] \longrightarrow \boxed{\quad h[n] \quad} \longrightarrow y[n]$$

A system is:

- *linear* if $y[n]$ is a linear combination of values of $x[n]$ and $y[n]$. A difference equation expressing this relation must not include constants or nonlinear functions of $x[n]$ and $y[n]$.
- *time-invariant* if $x[n-m]$ produces output $y[n-m]$ for any m. That is, the system response to an excitation is independent of when the excitation is applied. Typically, coefficients of the difference equation must not be functions of time (n).

For LTI systems, if inputs are decomposed into linear combinations of simpler signals, responses to individual inputs can be analyzed individually, and the total response can be found by superposition.

An arbitrary LTI system can be completely described by its impulse response. The impulse response $h[n]$ is the response of the system to a unit impulse $\delta[n]$. The unit impulse exists only at $n = 0$ and is zero elsewhere. (Recall that in Matlab, vectors subscripts start at 1, so the impulse response becomes the response of the system to a unit impulse at $n = 1$). By superposition, the response of the system to an arbitrary input $x[n]$ can be described as a linear combination of scaled and shifted impulse responses, giving rise to the convolution sum,

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \equiv x[n] * h[n]$$

Try a few simple convolutions:

```
h = [1 2 3];        % impulse response of a filter h
stem(h) x=1;                % input is an impulse
```

```
y=conv(x,h);
figure; stem(y)      % output is the impulse response

x=[1 1 1];
y=conv(x,h);
figure; stem(y)

x=[1 2 3];
y=conv(x,h);
figure; stem(y)
```

LTI System Representation: Difference Equations

Matlab's Signal Processing Toolbox provides several models for representing linear, time-invariant systems. Difference equations describe the input/output behavior of an LTI system directly in terms of signal values. The general form of the difference equation of a LTI system is written as,

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{m=0}^{M} b_m x[n-m] \tag{1}$$

$$y[n] = \frac{1}{a_0}\left(\sum_{m=0}^{M} b_m x[n-m] - \sum_{k=1}^{N} a_k y[n-k]\right)$$

Note the summation limits in the two equations.

A difference equation can be described in Matlab by two vectors:
`b = [b0 b1 ... bM]; a = [a0 a1 ... aN];`

When *any or all* `ai` $\neq 0$, `i=1,2,...N` (so that $y[n]$ depends on a previous values of y), the system is said to be *recursive*. It will have an *infinite impulse response* (IIR). When `ai` $= 0$, `i=1,2,...N` , the system is said to be nonrecursive ($y[n]$ does not depend on any previous value of y), It will have a *finite impulse response* (FIR).

Q1.What does any `ai` $\neq 0$, `for some i`, imply in terms of the connections in a filter block diagram, where the blocks represent delays, in contrast to the situation where `ai =0, all i=1,2,...aN`? That is, show the block diagram for the difference equation in the example below. Also show the block diagram for the non-recursive case, by setting the coefficient of y[n-2] as zero.

Example:
Represent the following difference equation by two vectors:

$$y[n] + 0.2y[n-2] = x[n] + 2x[n-1] \tag{2}$$

```
b = [1 2 0];
a = [1 0 0.2];
```

The z-Transform

Just as the Fourier transform forms the basis of signal analysis, the *z-transform* forms the basis of system analysis. If $x[n]$ is a discrete-time signal, its (two-sided) z-transform $X(z)$ is given by,

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n}$$

The z-transform maps a signal in the time domain to a power series in z. : $x[n] \rightarrow X(z)$. Note that `z = u + jv` is complex and can be considered as complex frequency (just like s in the continuous-time domain) and represents the so-called complex z-plane.

Some of the z-Transform properties are as follows:
- linearity and superposition are preserved
- $x[n-k] \rightarrow z^{-k}X(z)$
- $x[-n] \rightarrow X(1/z)$
- $a^n x[n] \rightarrow X(z/a)$
- $x[n] * y[n] \rightarrow X(z)Y(z)$

The overall result is that the algebra of system analysis becomes greatly simplified in the z domain. The only tradeoff is the necessity of taking an inverse transform to obtain time domain responses.

Since the response $y[n]$ of an LTI system to input $u[n]$ is given by the convolution $u[n] * h[n]$, where $h[n]$ is the impulse response, it is easily seen that,

$$y[n] = u[n] * h[n] \rightarrow Y(z) = U(z)H(z)$$

where $H(z)$ is the z-Transform of $h[n]$.

The ratio $H(z) = Y(z)/U(z)$ is referred to as the *transfer function* of the system.

LTI System Representation: Transfer Functions

The z-transform transforms a difference equation into a transfer function. The transfer function of the system in equation (1) above, can be written either in **discrete filter form**:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_M z^{-M}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + ... + a_N z^{-N}}$$

or, by multiplying the numerator and denominator by the highest power of z, in **proper form**:

$$H(z) = z^{N-M} \frac{b_0 z^M + b_1 z^{M-1} + b_2 z^{M-2} + ... + b_M}{a_0 z^N + a_1 z^{N-1} + a_2 z^{N-2} + ... + a_N}$$

Either form can be represented in Matlab by a vector of coefficients for the numerator and denominator (as described earlier),

```
b = [b0 b1 ...  bM]; a = [a0,a1 ...  aN];
```

$$y[n] + 0.2y[n-2] = u[n] + 2u[n-1]$$ Original difference equation

$$Y(z) + 0.2z^{-2}Y(z) = U(z) + 2z^{-1}U(z)$$ z-transform

$$H(z) = Y(z)/U(z) = \frac{1 + 2z^{-1}}{1 + 0.2z^{-2}}$$ Discrete filter form (inverse powers)

$$= \frac{z^2 + 2z}{z^2 + 0.2}$$ Proper form (positive powers)

Matlab representation

```
b = [1 2 0];
a = [1 0 0.2];
```

Try the same example, equation (2) above: $y[n] + 0.2y[n-2] = u[n] + 2u[n-1]$

```
b = [1 2];
a = [1 0 0.2];
[beq aeq] = eqtflength(b,a)
```

Q2. What does the function eqtflength do?

LTI System Representation: Zero-Pole-Gain

If the numerator and denominator of the proper form of a transfer function are factored, the *zeros* (roots of the numerator polynomial) and *poles* (roots of the denominator polynomial) become apparent:

$$H(z) = k\frac{(z-q_0)(z-q_1)...(z-q_{N-1})}{(z-p_0)(z-p_1)...(z-p_{N-1})}$$

The leading coefficient in the numerator, k, is called the gain.

Example:

$$H(z) = \frac{z^2+2z}{z^2+0.2}$$

Zeros:

$$\begin{aligned} &z^2+2z=0 \\ &z(z+2)=0 \\ &z=0 \quad \text{and} \quad z=-2 \end{aligned}$$

Poles:

$$\begin{aligned} &z^2+0.2=0 \\ &z=\pm\sqrt{0.2}j \end{aligned}$$

Gain:

$$k=1$$

The location of the zeros and poles of the transfer function determines the response of an LTI system.

Matlab's Signal Processing Toolbox provides a number of functions to assist in the zero-pole-gain analysis of a system.

```
[z p k] = tf2zpk(b,a)
```

finds the zeros z, poles p, and gain k from the coefficients b and a of a transfer function $H(z)$. Do it for the example given.

Q3. For the example above, write the transfer function $H(z)$ in discrete filter form and then *calculate* values of the zeros and poles. How does this compare with Matlab results ?

```
zplane(z,p)
zplane(b,a)
```

both display the zeros and poles of a system. A **o** marker is used for zeros and an **x** marker is used for poles. The unit circle ($z = e^{j\omega}$)is also plotted for reference. We will see later in class that $H(z)|_{z=e^{j\omega}} = H(e^{j\omega})$ which of course is the DTFT of a filter `h[n]`, (assuming that it exists). As we shall see, the unit circle has implications for stability and frequency analysis of the digital filter H(z).

Try:

```
b=[1 2 0];
a=[1 0 0.2];
[z p k] = tf2zpk(b,a)
zplane(z,p)
```

Q4. Explain the display.

Zero-Pole Placement in Filter Design

LTI systems, particularly digital filters, are often designed by positioning zeros and poles in the z-plane. Since $H(z)|_{z=e^{j\omega}} = H(e^{j\omega})$, we observe that traversing the unit circle anti-clockwise in the z-plane, starting from z=1, traces frequency ω from `0 to` 2π.

For filter design, the strategy often is to identify passband and stopband frequencies on the unit circle and then position zeros and poles by considering the following.

1. **Conjugate symmetry**
   All poles and zeros must be paired with their complex conjugates. (Do you see why?)
2. **Causality**
   To ensure that the system does not depend on future values, the number of zeros must be less than or equal to the number of poles. (Do you see why?)
3. **Origin**
   Poles and zeros at the origin do not affect the magnitude response.
4. **Stability**
   For a stable system, poles must be inside the unit circle. (Do you see why?) Pole radius is proportional to the gain and inversely proportional to the bandwidth. Passbands should contain poles near the unit circle for larger gains.
5. **Minimum phase**
   Zeros can be placed anywhere in the z-plane. Zeros inside the unit circle ensure "minimum phase". Zeros on the unit circle give a null response. Stopbands should contain zeros on or near the unit circle.

6. **Transition band**
   A steep transition from passband to stopband can be achieved when stopband zeros are paired with poles along (or near) the same radial line and close to the unit circle.

7. **Zero-pole interaction**
   Zeros and poles interact to produce a composite response that might not match design goals. Poles closer to the unit circle or farther from one another produce smaller interactions. Zeros and poles might have to be repositioned or added, leading to a higher filter order.

(We will see more of this when we cover filter design issues in class.)

Zero-Pole-Gain Editing in SPTool

Run:
`sptool`

To access the Pole/Zero Editor in SPTool, do the following:
1. Click the **New** button under the **Filters** list in SPTool.
2. You get to the Filter Design and Analysis Tool. (You can also get here by typing `fdatool` in the command window).
3. Design a very simple FIR Equiripple low-pass filter, order 3, sampling frequency 1000 hz, Fpass =0, Fstop =200, Wpass=1, Wstop=1, Density Factor=20.
4. Design Filter.
5. Now go through the various icons on the top, starting with Full View Analysis and see all the filter characteristics that are available from FDA Tool.
6. Select the **Pole/Zero Editor** in the left hand pane. (Sort of hard to see. It is the bottom left. Shows up with your cursor.) You can change position of the poles and zeros and watch any chosen filter characteristics change.
6. Invoking *i* in the icons on top shows you the implementation cost.
7. Going to Help → Show Filter Structure, shows you general filter implementations.

(We'll see this tool again later in the lab.)

Linear System Transformations

The Signal Processing Toolbox (we are not speaking of *sptool*) provides a number of command window functions for converting amongst several system representations.

Note: Representation conversions can produce systems with slightly different characteristics. This difference is due to roundoff error in the floating point computations performed during conversion.

	Transfer Function	*State Space*	*Zero Pole Gain*	*Partial Fraction*	*Lattice Filter*	*Second Order Sections*	*Convolution Matrix*
Transfer Function		`tf2ss`	`tf2zp` `tf2zpk` `roots`	`residuez`	`tf2latc`	`tf2sos`	`convmtx`
State Space	`ss2tf`		`ss2zp`			`ss2sos`	
Zero-Pole-Gain	`zp2tf` `poly`	`zp2ss`				`zp2sos`	
Partial Fraction	`residuez`						
Lattice Filter	`latc2tf`						
Second Order Sections	`sos2tf`	`sos2ss`	`sos2zp`				

Try the same example of equation (2):

```
b = [1 2 0];
a = [1 0 0.2];
[A B C D] = tf2ss(b,a)
```

where the so-called *state equations* of the system are:

```
x[n + 1] = Ax[n] + Bu[n]
y[n] = Cx[n] + Du[n]
```

Q5. Convert the state space vector first-order difference equation that you get above, back into the higher order difference equation, equation (2). It is easiest using z-Transforms, where $Z\{x[n]\} = X(z), Z\{x[n+1]\} = z^1 X(z)$ etc. {We cover z-Transforms later on in class}.

Impulse Reponse

The impulse response $h[n]$ and its z-transform (the transfer function $H(z)$) completely characterize the response of an LTI system. For input $x[n]$ in the time domain ($X(z)$ in the frequency domain), the output of the system $y[n]$ in the time domain ($Y(z)$ in the frequency domain) is given by:

$y[n] = x[n] * h[n]$	(convolution)	Time domain
$Y(z) = X(z)Y(z)$	(product)	Frequency domain

The Signal Processing Toolbox function

```
[h t]=impz(b,a,n,fs)
```

computes the impulse response of the LTI system with transfer function coefficients b and a. It computes n samples and produces a vector t of length n so that the samples are spaced $1/fs$ units apart. It returns the response in the column vector h and sample times in the column vector t. When called with no outputs, it plots the response.

For a system identification application, the impulse response of an unknown system can be compared with the impulse response of a model of that system, and model parameters (such as zeros and poles) can be tuned to match the response of the unknown system.

Try:
```
b=[1 2 0];
a=[1 0 0.2];
impz(b,a,10,1e3)
```

Q6. Design a simple unstable filter and show its impulse response.

Frequency Response

The Signal Processing Toolbox function

```
freqz(b,a)
```

when called with no output arguments, plots the magnitude and *unwrapped* phase of the filter in the current figure window.

Try:
```
b=[1 2 0];
```

```
a=[1 0 0.2];
freqz(b,a)
```

The following form of the function

```
[h w]=freqz(b,a)
```

computes the frequency response of the LTI system with transfer function coefficients b and a. It returns the frequency response vector h and the corresponding angular frequency vector w. The vector w has values ranging from 0 to the Nyquist rate, which is normalized to π radians per sample by default. The response is calculate using a default of 512 samples.

Try:

```
[h w]=freqz(b,a);
subplot(211), plot(w/pi, 20*log10(abs(h)))
subplot(212), plot(w/pi, (180/pi)*angle(h))
```

```
h=freqz(b,a,w)
```

returns the frequency response vector h calculated at the frequencies (in radians per sample) supplied by the vector w.

Q7. What does "1" on the horizontal axis correspond to?

Frequency Response and the Transfer Function

A z-plane plot of the magnitude of the transfer function H(z) has a height of zero at each of the zeros and a singularity at each of the poles. The curve on this surface above the unit circle gives the frequency response.

$$H(z) = 8\frac{z-1}{z^2 - 1.2z + 0.72}$$

You can take a look at the function (it's in *sg01*) which you should have:

```
edit transferplot
```

Run it:

```
transferplot
```

Then try this, to see a "birds-eye view":

```
view(0,90)
```

Filter Visualization Tool

For an arbitrary LTI system, the Filter Visualization Tool (fvtool) displays:
• Magnitude response
• Phase response
• Group delay
• Impulse response
• Step response
• Pole-zero plot
• Filter coefficients

You can:
• Click the plot to display a value at a point
• Change axis units by right-clicking an axis label or by right-clicking the plot and selecting **Analysis Parameters**
• Export the display to a file with **Export** on the **File** menu

Try this:
```
a=[1 0 .2];
b=[1 2 0];
fvtool(b,a)
```

Can you figure out how to change the display to show the magnitude plot as dB vs. $log(f)$**.**

Filtering a Signal

LTI systems are commonly called filters, and the process of generating the output $y[n]$ from an input $x[n]$ is called filtering. To filter a signal using a system described by transfer function coefficients b and a, use the Matlab filter function.

```
y=filter(b,a,x)
```

filters the data in vector x with the filter described by numerator coefficient vector b and denominator coefficient vector a. If $a(1)$ is not equal to 1, $filter$ normalizes the filter coefficients by $a(1)$. If $a(1)$ equals 0, $filter$ returns an error.

The *filter* function is implemented in a "direct form II transposed filter" architecture.

The difference equation is,

$$y[n] = b(1)*x[n]+b(2)*x[n-1]+...+b(nb+1)*x[n-nb]...-a(2)*y[n-1]-...-a(na+1)*y[n-na]$$

where $n-1$ is the filter order (the highest degree in the proper form of the transfer function).

(You have noisyC.m in sg01. You can look at the file *edit noisyC*. You can also run it by typing "*noisyC*" in the Command window. Helps if you can hear it.)

Run *noisyC* script to generate a noisy sine wave: `fs = 1e4;`

```
t = 0:1/fs:5;
sw = sin(2*pi*262.62*t); % Middle C
n = 0.1*randn(size(sw));
swn = sw + n;
```

Use a simple lowpass (averaging) filter:

```
b=[.25 .25 .25 .25];
a=[1 0 0 0];
y=filter(b,a,swn);
figure, plot(t,y), axis([0 0.04 -1.1 1.1])
h=impz(b,a);
y2=conv(swn,h);
figure, plot(t,y2(1:end-3)), axis([0 0.04 -1.1 1.1])
```

Q8. How do the two outputs (y and y2) compare?
Q9. How do the methods (filter and conv) differ?

Using SPTool

Choose **File** → **Import** from the SPTool main menu.
Try importing the filter just created. (Recall that the filter has a numerator and a denominator which is 1.)

To apply a Filter in SPTool:
1. Select the signal to be filtered.
2. Select the filter to apply.
3. Click the Apply button under the Filters list.

Try filtering the noisy middle C signal **swn** with the simple lowpass filter created earlier.

Exercises:
1. Plot the frequency responses of the two filters with the following difference equations. Sampling frequency: 2kHz.

Low-pass: $y[n] = 0.5x[n] + 0.5x[n-1]$ (averaging)

High-pass: $y[n] = 0.5x[n] - 0.5x[n-1]$ (finite differencing)

2. Filter the signal x from Exercise 1 with each of the filters. (Run *hilo* to recreate the signal.) View and listen to the outputs. Note that the two components of x are at 5% and 95% of the Nyquist frequency, respectively.

3. Open each filter in the Pole/Zero editor in SPTool. Adjust the locations of the poles and zeros and observe the effects on both the response and the filtered signal.

Cepstral Analysis

Certain signals, such as speech signals, are naturally modeled as the output of an LTI system; i.e., as the convolution of an input and an impulse response. To analyze such signals in terms of system parameters, a deconvolution must be performed. Cepstral analysis is a common technique for such deconvolution.

The technique is based on two facts: a convolution in the time domain becomes a product in the z-domain, and logarithms of products become sums. If:

$$x[n] = x_1[n] * x_2[n] \leftrightarrow X(z) = X_1(z)X_2(z)$$

then:

$$\hat{X}(z) = log(X(z)) = log(X_1(z)) + log(X_2(z)) \leftrightarrow \hat{x}[n] = \hat{x}_1[n] + \hat{x}_2[n]$$

Convolved signals become additive in the new cepstral domain.
(The word "cepstral" reverses the first letters in the word "spectral" to reflect the back-and-forth between the time and frequency domains.)

The complex cepstrum $\hat{x}[n]$ of a sequence $x[n]$ is calculated by finding the complex natural logarithm of the Fourier transform of x, then the inverse Fourier transform of the resulting sequence:

$$\hat{x}[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} log[X(e^{j\omega})]e^{j\omega n}d\omega$$

The Signal Processing Toolbox function *cceps* perfoms this operation and the function *icceps* performs the inverse.

The *real cepstrum* (or just the *cepstrum*) of $x[n]$ is found as above, using only the magnitude of the Fourier transform. It is computed by the Signal Processing Toolbox function *rceps*. The original sequence cannot be reconstructed from only its real cepstrum, but you can reconstruct a minimum-phase version of the sequence by applying a windowing function in the cepstral domain. The minimum phase reconstruction is returned by *rceps* in a second output argument.

Read the *echodetect.m* script (shown below), then try running it. (You might need to disable the "sound" line. Also, helloGC.mat is in *sg*01.)

```
% ECHODETECT Demonstrates cepstral analysis for echo detection.
% Load stereo sound sample HGC and sample frequency fs:
load helloGC
% Delays for echoes (in samples):
d1 = 10000;
d2 = 20000;
d3 = 30000;
% Attenuation for echoes (percent):
a = 0.6;
% Add echoes to sound sample:
s0 = [HGC; zeros(d1+d2+d3,2)];
s1 = [zeros(d1,2); HGC; zeros(d2+d3,2)];
s2 = [zeros(d2,2); HGC; zeros(d1+d3,2)];
s3 = [zeros(d3,2); HGC; zeros(d1+d2,2)];
s = s0 + a*s1 + a^2*s2 + a^3*s3;
% Play sound with echoes:
%sound(s,fs)
% Cepstral analysis.
% Plot cepstrum of sound with echoes (2nd channel) in red:
c = cceps(s(:,2));
plot(c,'r')
hold on
% Overlay cepstrum of original sound (2nd channel) in blue:
c0 = cceps(s0(:,2));
plot(c0,'b')
axis([0 1e5 -1 1])
xlabel('Sample Number')
ylabel('Complex Cepstrum')
title(['Echoes at ',num2str(d1),' ',num2str(d2),' ',num2str(d3)])
% Note red peaks at delays.
```

(The material in this lab handout derives generally from the MathWorks training document "MAT-

LAB for Signal Processing", 2006. Revised 2-16-2011)