EE 3054: Signals, Systems, and Transforms

Additional Notes for the MATLAB Lab
Notes on Partial Fraction Expansion in Matlab

The Matlab command residue allows one to do partial fraction expansion.

>> help residue

RESIDUE Partial-fraction expansion (residues).

[R,P,K] = RESIDUE(B,A) finds the residues, poles and direct term of
a partial fraction expansion of the ratio of two polynomials B(s)/A(s).
If there are no multiple roots,

\[
\frac{B(s)}{A(s)} = \frac{R(1)}{s - P(1)} + \frac{R(2)}{s - P(2)} + \ldots + \frac{R(n)}{s - P(n)} + K(s)
\]

Vectors B and A specify the coefficients of the numerator and
denominator polynomials in descending powers of s. The residues
are returned in the column vector R, the pole locations in column
vector P, and the direct terms in row vector K. The number of
poles is \(n = \text{length}(A)-1 = \text{length}(R) = \text{length}(P)\). The direct term
coefficient vector is empty if \(\text{length}(B) < \text{length}(A)\), otherwise
\(\text{length}(K) = \text{length}(B)-\text{length}(A)+1\).

If \(P(j) = \ldots = P(j+m-1)\) is a pole of multiplicity \(m\), then the
expansion includes terms of the form

\[
\frac{R(j)}{s - P(j)} + \frac{R(j+1)}{(s - P(j))^2} + \ldots + \frac{R(j+m-1)}{(s - P(j))^m}
\]

[B,A] = RESIDUE(R,P,K), with 3 input arguments and 2 output arguments,
converts the partial fraction expansion back to the polynomials with
coefficients in B and A.

See also POLY, ROOTS, DECONV.
To perform partial fraction expansion on $T(z)$

$$T(z) = \frac{4z + 1}{z^2 - z - 2}$$

we enter the following Matlab commands.

```matlab
>> num = [4 1];
>> den = [1 -1 -2];
>> [r,p,k] = residue(num,den)
```

```
r =
 3
 1

p =
 2
-1

k =
[]
```

This tells us that,

$$T(z) = \frac{3}{z - 2} + \frac{1}{z + 1}.$$
The `residue` command also works when some poles are complex. For example, to find the partial fraction expansion of \( X(z) \)

\[
X(z) = \frac{4z + 3}{2z^4 - 3.4z^3 + 1.98z - 0.406}
\]

we can use the following Matlab commands.

```matlab
>> num = [4 3];
>> den = [2 -3.4 1.98 -0.406];
>> [r,p,k] = residue(num,den)
```

\[
r =
36.2500
-18.1250 +13.1250i
-18.1250 -13.1250i

p =
0.7000
0.5000 + 0.2000i
0.5000 - 0.2000i

k =
[
]
```

This tells us that,

\[
X(z) = \frac{36.25}{z - 0.7} + \frac{-18.125 + 13.125i}{z - (0.5 + 0.2i)} + \frac{-18.125 - 13.125i}{z - (0.5 - 0.2i)}.
\]

Notice that the residues and poles appear in complex-conjugate pairs. In fact, this is always the case when the coefficients in \( X(z) \) are real.
Here is an example with a repeated pole. To perform partial fraction expansion on $T(z)$

\[ T(z) = \frac{2z + 1}{z^3 + 5z^2 + 8z + 4} \]

we enter the following Matlab commands.

\begin{verbatim}
>> num = [2 1];
>> den = [1 5 8 4];
>> [r,p,k] = residue(num,den)
\end{verbatim}

\begin{verbatim}
r =
    1.0000
    3.0000
   -1.0000

p =
    -2.0000
    -2.0000
    -1.0000

k =
    []
\end{verbatim}

This tells us that,

\[ T(z) = \frac{1}{z + 2} + \frac{3}{(z + 2)^2} + \frac{-1}{z + 1} \]
The `residue` command also works in the other direction. To write $G(z)$,

$$G(z) = \frac{5}{z - 3} + \frac{6}{z + 4} - \frac{7}{z + 1/5}$$

as the ratio of two polynomials we can use the following commands.

```matlab
>> r = [5 6 -7];
>> p = [3 -4 -1/5];
>> [num,den] = residue(r,p,[])
```

num =

4.0000  -2.8000  84.4000

den =

1.0000  1.2000 -11.8000  -2.4000

This tells us that

$$G(z) = \frac{4z^2 - 2.8z + 84.4}{z^3 + 1.2z^2 - 11.8z - 2.4}.$$  

The argument $k$ is needed only when the degree of the numerator is greater than or equal to the degree of the denominator, so it was not needed for these examples.
In this example, the degree of the numerator is greater than the degree of the denominator, so \( k \) is required.

\[
X(z) = \frac{3z^4 - 1.1z^3 + 0.88z^2 - 2.396z + 1.348}{z^3 - 0.7z^2 - 0.14z + 0.048}
\]

The partial fraction expansion of \( X(z) \) can be found using the following Matlab commands.

\[
>> \text{num} = [3 \ -1.1 \ 0.88 \ -2.396 \ 1.348]; \\
>> \text{den} = [1 \ -0.7 \ -0.14 \ 0.048]; \\
>> [r,p,k] = \text{residue(num,den)}
\]

\[
\begin{align*}
r &= \\
   &1.0000 \quad 4.0000 \quad -3.0000 \\
\end{align*}
\]

\[
\begin{align*}
p &= \\
   &0.8000 \quad -0.3000 \quad 0.2000 \\
\end{align*}
\]

\[
\begin{align*}
k &= \\
   &3.0000 \quad 1.0000 \\
\end{align*}
\]

This tells us that

\[
X(z) = 3z + 1 + \frac{1}{z - 0.8} + \frac{4}{z + 0.3} - \frac{3}{z - 0.2}
\]
Notes on Recursive Digital Filters

A recursive digital filter is an LTI system based on a difference equation of the form:

\[ y(n) = -\sum_{k=1}^{N} a(k) y(n-k) + \sum_{k=0}^{M} b(k) x(n-k) \]  

A system described by this type of difference equation is called an IIR (Infinite Impulse Response) filter, a recursive filter, or an autoregressive moving-average (ARMA) filter. The output \( y(n) \) of the filter can be written as

\[ y(n) = \sum_{k=0}^{\infty} h(k) x(n-k). \]  

The transfer function of the system \( H(z) \) can be written as

\[ H(z) = \sum_{k=0}^{\infty} h(k) z^{-k}. \]  

The frequency response is given by

\[ H_f(\omega) = H(e^{j\omega}) = \sum_{n=0}^{\infty} h(n) e^{-j\omega}. \]  

The transfer function can also be written in terms of the difference equation coefficients as

\[ H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_N z^{-N}} \]  

or

\[ H(z) = \frac{B(z)}{A(z)} \]  

where

\[ B(z) = \sum_{n=0}^{M} b(n) z^{-n} \]  

\[ A(z) = 1 + \sum_{n=1}^{N} a(n) z^{-n}. \]

**Computing \( h(n) \).** Given the difference equation coefficients \( a(k) \) and \( b(k) \), the impulse response \( h(n) \) can be obtained by taking the inverse Z-transform of \( H(z) \), but it is usually simpler to calculate \( h(n) \) numerically by running the difference equation with the input \( x(n) = \delta(n) \). The Matlab function `filter` implements a recursive difference equation. The following code fragment computes the first 30 values of \( h(n) \) from \( a(k) \) and \( b(k) \).
\[ x = [1, \text{zeros}(1,29)]; \]
\[ h = \text{filter}(b,a,x); \]

where the vectors \( b \) and \( a \) contain the filter parameters

\[
\begin{align*}
    b &= [b(0), b(1), \ldots, b(M)] \\
    a &= [a(0), a(1), \ldots, a(N)]
\end{align*}
\]

**Computing** \( H(f) \) \( [H(e^{j\omega})] \). The frequency response can be calculated with the Matlab command `freqz`, which uses the FFT (Fast Fourier Transform).

The following code, `filt1.m`, illustrates how to use these commands to display the frequency response and impulse response of a digital filter. This code is available of the class web page.

\[
\begin{align*}
    b &= [0.0931 \ 0.0960 \ 0.1801 \ 0.1801 \ 0.0960 \ 0.0931] \\
    a &= [1.0000 \ -1.5757 \ 2.2408 \ -1.5554 \ 0.8123 \ -0.1837]
\end{align*}
\]

% ------ Plot frequency response magnitude ---
subplot(3,1,1)
[H,w]=freqz(b,a,2^10);
plot(w/pi,abs(H))
axis([0 1 0 1.2])
xlabel('\omega/\pi')
ylabel('\|H(e^{j \omega})\|')

% ------ Plot frequency response in dB -------
subplot(3,1,2)
plot(w/pi,20*log10(abs(H)))
axis([0 1 -60 10])
xlabel('\omega/\pi')
ylabel('\|H(e^{j \omega})\| \text{ in dB}')

% ------ Plot impulse response h(n) ----------
subplot(3,1,3)
N = 30;
im = [1 \text{zeros}(1,N-1)];
h = \text{filter}(b,a,im);
stem(0:N-1,h,'.')
xlabel('n')
ylabel('h(n)')
axis([-1 30 -0.2 0.6])

This code produces the following figure.
Because the ideal low-pass filter can not be implemented, it is necessary to settle for an approximation to it. The problem of digital filter design is choose coefficients $a(k), b(k)$, so that the frequency response of the filter $H^f(\omega)$ matches a desired target function.

A good approximation to the ideal low-pass filter is the elliptic filter, also called the Cauer filter. The coefficients $a(k), b(k)$ of elliptic filters can be obtained with the Matlab command `ellip`. (It is part of the signal processing toolbox.) Here is the beginning of the help file for this command.

ELLIP Elliptic or Cauer digital and analog filter design.

$[B,A] = ELLIP(N,Rp,Rs,Wn)$ designs an $N$th order lowpass digital elliptic filter with $R_p$ decibels of ripple in the passband and a stopband $R_s$ decibels down. ELLIP returns the filter coefficients in length $N+1$ vectors B (numerator) and A (denominator).

The cut-off frequency $W_n$ must be $0.0 < W_n < 1.0$, with 1.0 corresponding to half the sample rate. Use $R_p = 0.5$ and $R_s = 20$ as starting points, if you are unsure about choosing them.

The meaning of $R_p$ and $R_s$ is given by

\[
\delta_p = 1 - 10^{-R_p/20} \tag{11}
\]

\[
\delta_s = 10^{-R_s/20} \tag{12}
\]
and

\[ R_p = -20 \log_{10}(1 - \delta_p) \]  \hspace{1cm} (13)
\[ R_s = -20 \log_{10}(\delta_s) \]  \hspace{1cm} (14)

where \( \delta_p \) represents the size of the ripple in the pass-band, and \( \delta_s \) represents the size of the ripple in the stop-band. \( R_p \) and \( R_s \) are just \( \delta_p \) and \( \delta_s \) in decibels. The pass-band ripple size \( \delta_p \) is the maximum deviation of the actual frequency response from 1 in the pass-band. The stop-band ripple size \( \delta_s \) is the maximum deviation of the actual frequency response from 0 in the stop-band.

The Matlab filter design commands use a somewhat non-standard convention for specifying the cut-off frequency \( \omega_c \). For example, if you want to design a discrete-time filter with cut-off frequency at \( \omega_c = 0.4 \pi \), then you should set \( \text{Wn} = 0.4 \) when using the \texttt{ellip} command. The same is true for the other Matlab digital filter design commands.

By the way, this assumes that the sampling period is 1 second. That is, the values \( x(n) \) are obtained from sampling a continuous-time signal once each second. If the sampling period (call it \( T_s \)) is different from 1 second, and you want to design a digital filter with cut-off frequency at \( F_o \) Hz. Then you should take \( \text{Wn} = 2*F_o/F_s \). Here \( F_s \) is the sampling frequency in Herz, \( F_s = 1/T_s \).

For example, suppose we sample an analog signal with a sampling rate of 100 Hz and we want to design a digital filter with a cut-off frequency of 30 Hz. Here are the Matlab commands to set up the design parameters, call the function, and plot the frequency response versus frequency in Herz.

\begin{verbatim}
Ts = 0.01; % Sampling Period Ts = 0.01 seconds
Fs = 1/Ts % Sampling frequency Fs = 100 Hz
Fo = 30; % Cut-Off frequency Fo = 30 Hz

Wn = 2*Fo/Fs; % Normalize the cut-off frequency

dp = 0.01;
ds = 0.01;
N = 5;
Rp = -20*log10(1-dp);
Rs = -20*log10(ds);

[b,a] = ellip(N,Rp,Rs,Wn);
figure(1), clf

% Plot frequency response magnitude
[H,w] = freqz(b,a);
plot(w/pi*Fs/2,abs(H)) % rescale frequency axis appropriately.
axis([0 Fs/2 0 1.2])
xlabel('FREQUENCY (Hz)')
ylabel('FREQUENCY RESPONSE')
grid
orient portrait
\end{verbatim}
print -deps filt2
Notes on Stable Inverses of Discrete-Time LTI Systems

Suppose the signal $x(n)$ represents binary data that is to be transmitted.

However, the channel over which $x(n)$ is transmitted introduces some distortion. The signal $r(n)$ that is received is changed.

In this example, the channel can be modeled as an LTI system described by the difference equation

$$r(n) - \frac{11}{20} r(n-1) - \frac{1}{2} r(n-2) = x(n)/10 + \frac{7}{3} x(n-1)/30 + \frac{2}{3} x(n-2)/30$$

where $r(n)$ represents the received signal. Equivalently, the channel can be modeled by the transfer function

$$H(z) = \frac{1}{10} \cdot \frac{1 + \frac{7}{3} z^{-1} + \frac{2}{3} z^{-2}}{1 - \frac{11}{20} z^{-1} - \frac{1}{5} z^{-2}}, \tag{15}$$

$$H(z) = \frac{1}{10} \cdot \frac{z^2 + \frac{7}{3} z + \frac{2}{3}}{z^2 - \frac{11}{20} z - \frac{1}{5}}, \tag{16}$$

or equivalently by its impulse response:

Define

$$G(z) := \frac{1}{H(z)}. \tag{17}$$
Then

\[ G(z) = 10 \cdot \frac{z^2 - \frac{11}{20}z - \frac{1}{5}}{z^2 + \frac{7}{3}z + \frac{2}{3}}. \] (18)

A causal system with the transfer function \( G(z) \) can be implemented by the difference equation

\[
y(n) / 10 + 7 y(n - 1) / 30 + 2 y(n - 2) / 30 = r(n) - 11 r(n - 1) / 20 - r(n - 2) / 5
\]

where \( r(n) \) represents the input to the system and \( y(n) \) represents the output of the system. This system can be implemented in a causal fashion by writing the difference equation as

\[
y(n) = 10 r(n) - 11 r(n - 1) / 2 - 2 r(n - 2) - 7 y(n - 1) / 3 - 2 y(n - 2) / 3.
\]

However, when this is done, the signal \( y(n) \) that you obtain is:

![Graph showing signal y(n)](image)

Because the causal implementation of this difference is **unstable**, the signal blows up after a short while. In order to obtain a stable inverse for the system \( H(z) \), we need to be more careful. Let us find a partial fraction expansion of \( G(z)/z \)

\[
\frac{G(z)}{z} = \frac{10z^2 - 11/2z - 2}{z^3 + 7/3z^2 + 2/3z}. \] (19)

We can use Matlab to help with the partial fraction expansion.

\[
>> [r, p, k] = \text{residue([10 -11/2 -2], [1 7/3 2/3 0])}
\]

\[
r =
\begin{align*}
14.7000 \\
-1.7000 \\
-3.0000
\end{align*}
\]

\[
p =
\begin{align*}
-2.0000 \\
-0.3333 \\
0
\end{align*}
\]

\[
k = []
\]
Therefore, we have

\[
\frac{G(z)}{z} = 14.7 \frac{z}{z+2} - 1.7 \frac{z}{z+1/3} - 3 \tag{20}
\]

or

\[
G(z) = \frac{14.7z}{z+2} + \frac{-1.7z}{z+1/3} - 3 \tag{21}
\]

and

\[
g(n) = \begin{cases} 
14.7 (-2)^n \ u(n) \quad \text{or} \\
-14.7 (-2)^n \ u(-n-1)
\end{cases} + \begin{cases} 
-1.7 (-1/3)^n \ u(n) \quad \text{or} \\
1.7 (-1/3)^n \ u(-n-1)
\end{cases} - 3 \delta(n).
\]

Consider the first component — the causal signal \(14.7 (-2)^n \ u(n)\) grows without bound:

On the other hand, the noncausal signal \(-14.7 (-2)^n \ u(-n-1)\) decays:

To obtain a stable system, we must choose the noncausal form for this component.

Consider the second component — the causal signal \(-1.7 (-1/3)^n \ u(n)\) decays:

On the other hand, the noncausal signal \(1.7 (-1/3)^n \ u(-n-1)\) grows without bound:
To obtain a stable system, we must choose the causal form for this component. Therefore, to obtain a stable system, the impulse response \( g(n) \) must be given by

\[
g(n) = -14.7 (-2)^n u(-n - 1) - 1.7 (-1/3)^n u(n) - 3 \delta(n).
\]  

This impulse response \( g(n) \) is shown in the figure.

For an inverse system, it is required that \( h(n) \ast g(n) = \delta(n) \). We can verify this by numerically convolving the two functions.

We can now recover \( x(n) \) from \( r(n) \) by using the LTI system with impulse response \( g(n) \). Note that even though \( g(n) \) is noncausal, for \( n < -10 \), the values of \( g(n) \) are very small and for practical purposes can be set to zero. If they are set to zero, then the shifted signal \( g(n - 10) \) will be causal. So you can define a causal system with impulse response \( g_c(n) \):

\[
g_c(n) := \begin{cases} 
  g(n - 10) & n \geq 0 \\
  0 & n < 0 
\end{cases}
\]  

or more compactly as

\[
g_c(n) := g(n - 10) u(n).
\]
This is shown in the figure:

![Figure showing the signal and its approximation](image)

The signal $g_c(n)$ is a good approximation to $g(n)$ because the values of $g(n)$ that are lost due to truncation are close to zero.

Using the causal stable system $g_c(n)$ to filter the received signal $r(n)$ gives:

![Filtered signals](image)

The top signal is $x(n)$. The lower signal is $y_2(n)$, obtained by convolving $r(n)$ with $g_c(n)$. Notice that $y_2(n)$ is almost the same as $x(n)$ except for a delay of 10 samples.

**Therefore, we can get a stable near-inverse of the channel if we allow for some delay.**

The following Matlab code, used for this problem, can be found on the course webpage.

```matlab
% generate a data signal (don't worry about understanding this command)
x = kron(round(rand(1,40)),ones(1,5));

% ------ Plot data signal ------
figure(1)
subplot(4,1,1), plot(x), ylabel('x(n)')
axis([0 200 -1 2])

% ------ Define System I ------
b = [1 +7/3 2/3]*(1/10);
a = [1 -11/20 -4/20];
r = filter(b,a,x);
```

17
% r(n) is the distorted version of x(n)

subplot(4,1,2), plot(r), ylabel('r(n)')

y = filter(a,b,r);

% y(n) is what you get when you try the "simple" thing.
% The "simple" thing is to reverse the difference equation.

subplot(4,1,3), plot(y), ylabel('y(n)')

% --> the "simple" thing is unstable!

im = [1 zeros(1,100)]; % define impulse signal
h = filter(b,a,im); % impulse response of System I

% ------ Let G(z) = 1/H(z) ------

% ------ Compute partial fraction expansion of G(z)/z ------

[R,P,K] = residue([10 -11/2 -2],[1 7/3 2/3 0]);

% R =
% 14.7000
% -1.7000
% -3.0000
%
% P =
% -2.0000
% -0.3333
% 0
%
% K =
% []

% so g(n) = -14.7 (-2)^n u(-n-1) - 1.7 (-1/3)^n u(n) - 3 delta(n)

% ------ create stable signal g(n) corresponding to G(z) ------

n = -10:10;
g1 = -14.7 .* (-2).^n .* (-n-1>=0);
g2 = -1.7 .* (-1/3).^n .* (n>=0);
g3 = -3*(n==0);
g = g1 + g2 + g3;

% ------ plot h(n) and stable g(n) ------
% ------ also plot h(n) convolve g(n) ------

figure(2)
subplot(4,1,1), stem(h,'.'), ylabel('h(n)')
subplot(4,1,2), stem(g,'.'), ylabel('stable g(n)')
check = conv(h,g);
subplot(4,1,3), stem(check,'.'), ylabel('h(n) conv g(n)')

% ------ send r(n) through the system G(z) ------
y2 = conv(r,g);
figure(1)
subplot(4,1,4), plot(y2); ylabel('y_2(n)')
axis([0 200 -1 2])

% ------ plot x(n) and recovered signal y2(n) ----
% ------ on same graph ------

figure(3)
k = 1:200;
plot(k,x(k),k,y2(k)-2)
axis([0 200 -3 2])
ylabel('x(n) and y_2(n)')

% notice that y2(n) is a delayed version of x(n)
% ---> with a delay we can get x(n) back from y(n) using
% a STABLE system.