Linguistics 582
Basics of Digital Signal Processing

Moving Average (Feedforward) Filters

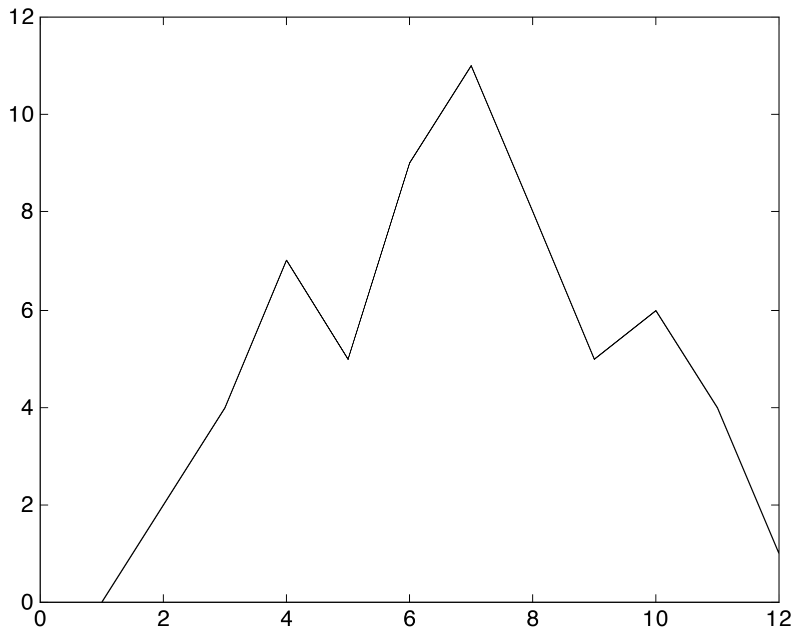
I. Simple digital filters

Suppose that we have a sequence of data points that we think should be characterizable as a smooth curve, for example, increasing in value and then decreasing. Suppose further that the data roughly follow the expected form, but there is some irregularity in the curve that we assume is some kind of "noise".

(1) The MATLAB vector X is an example of such an hypothetical data string:

```
»X = [0 2 4 7 5 9 11 8 5 6 4 1]
X =
     0     2     4     7     5     9    11     8     5     6     4     1
```

»plot (X)

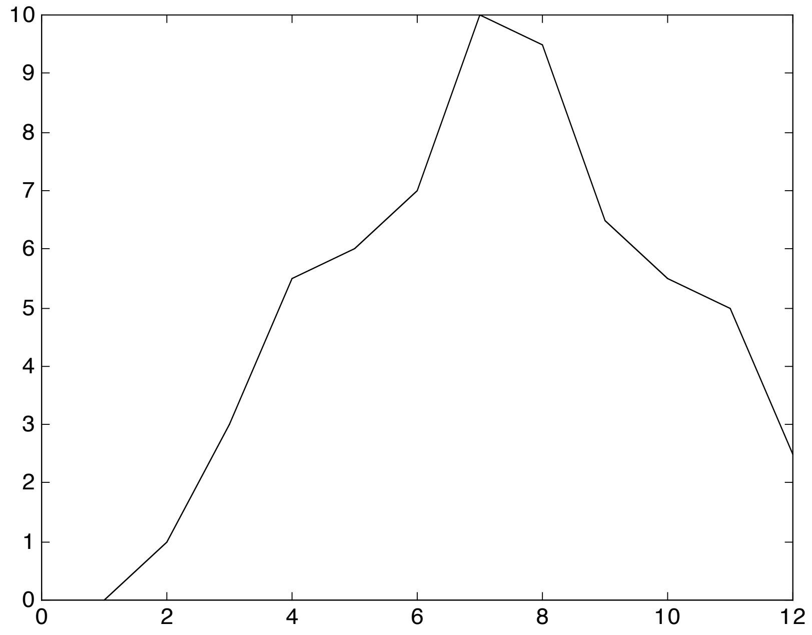


Suppose further that we want to "remove" the noise from the data, so as to obtain a closer representation to the expected underlying smooth curve. Intuitively, this could be accomplished by averaging the values of successive pairs of points in the X vector. To do this, we can generate a new output data sequence (Y), the values of which are calculated as the average of two successive points in the input (X) vector. For example, (2) is an MATLAB script `(average2pts)` that does this:

(2)
```
% average2pts
% Louis Goldstein


% Create a new signal Y, by averaging successive points in signal X

for k = 2:length(X)
   Y(k) = .5*X(k) + .5*X(k-1);
end
```

When this script is executed, it produces the new vector (Y) shown below:

```
»average2pts
»Y
Y =
  Columns 1 through 7
        0    1.0000    3.0000    5.5000    6.0000    7.0000   10.0000
  Columns 8 through 12
    9.5000    6.5000    5.5000    5.0000    2.5000

»plot (Y)
```



Note that some of the jaggedness of the curve has been leveled out.

The operation in (2) is a called a digital filter and it performs a smoothing function. However, our script is bad MATLAB code. It has to loop through all the points in the signal. We can write an alternate version that is not only more efficient, but teaches us something about this kind of filter.

We can take the input sequence X, scale it by .5, and add it to a version of itself that is shifted to the right , and also scaled by .5. A script for doing this is shown in (3):

(3)
```
% average_shift.m


% Create a new signal Y, by averaging successive points in signal X
% shift method

X_shifted = [0 X];
X_shifted(end) = [];

Y = .5*X + .5*X_shifted;
```

Note that if we think of a vector of this kind as representing *time*, then the shift to the right operation is a delay. This filter delays the input by one sample, and then adds it to the undelayed signal. Both are multiplied by .5. Filtering as delay is discussed in Steiglitz 4.1 and 4.2.

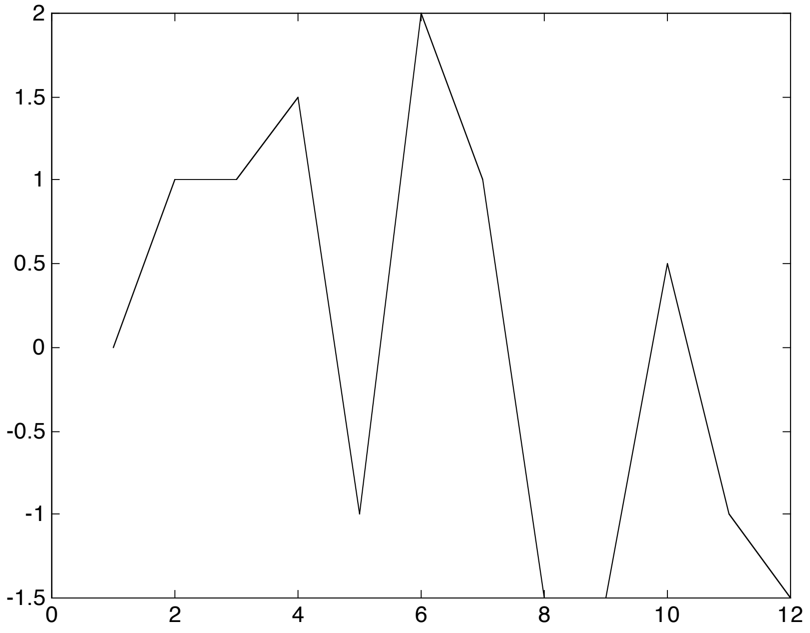
Differencing

Conversely, if we want to ignore the overall up and down trend of the data sequence X, and examine only the local irregularity, we could take the difference between successive points, rather than the average. Such a script (`diff2pts`) is shown in (4) and the result of executing it is shown in (5).

(4)
```
% diff2pts
% Louis Goldstein
% 30 March 1996

% Create a new signal Y, by subtracting successive points in signal X

for k = 2:length(X)
   Y(k) = .5*X(k) - .5*X(k-1);
end
```

(5)
```
»diff2pts
»Y
Y =
  Columns 1 through 7
         0    1.0000    1.0000    1.5000   -1.0000    2.0000    1.0000
  Columns 8 through 12
   -1.5000   -1.5000    0.5000   -1.0000   -1.5000
```

»plot(Y)



The overall up and then down trend in the data is gone--only the local irregularities remain. Note that if we treat the successive points in data sequence X as representing successive time samples, then `diff2pts` effectively differentiates the sequence X (with respect to time), and produces a signal that represents the instantaneous velocity of the original signal X.

II. Amplitude and Phase response of a simple smoothing filter

To understand what filters of this kind will do to the sound signals, we should consider what they do to phasors. Since any signal can, ultimately, be represented as a sum of phasors, and since filtering (as defined above) is a linear operation, we can determine the effect of filtering on an arbitrary signal by calculating the effect it has on all the frequency components of that signal. The output of a sum of phasors is the sum of the outputs to the individual phasors.

So we take input to be a phasor -- $X(k) = e^{j\omega k}$

where k is sample number, and w is frequency in radians per sample.

What does a filter like the one in (2) and (3) in section (1) do to a phasor input?

We can out the output by adding a delayed version of the phasor to the phasor (ignoring the .5 multipliers, for now). We know that if we delay a phasor by one sample, we get a phase shift – by whatever angle is elapsed in the delay time (which is determined by ω, the number of radians per sample). So what will the result be of adding a phasor to a shifted one?

We already know the answer to this. Since the original phasor and the shifted one have the same frequency, the question is the same as the one we asked in the first week—what is the result of adding two sinusoids of the same frequency and different amplitudes. The answer was that we get a sinusoid of the same frequency, with a different amplitude, and a different phase. We even worked out the equation for the amplitude of the sum, in terms of the individual amplitudes and the phase difference. The equation for the amplitude of this delayed sum is given in Steiglitz Chap. 4, Eq. (2.7), in which the amplitude of the unshifted phasor is 1, the amplitude of the shifted signal is a_1, and the phase difference between them is ωτ. You should check that it is equal to the expression we worked out in our `add_phases.m` script from Week 2, given these amplitudes and the phase difference.

Let's look at a particular example.

Suppose our signal is a phasor X, whose frequency is ω radians per sample, as defined in (1).
(1) $X(k) = e^{j\omega k}$

Now let us filter this phasor through a two-point moving average filter, like that described in section I.
(2) cx

Substituting for X(k):
(3) $Y(k) = .5\ e^{j\omega k} + .5\ e^{j\omega(k-1)}$

Now let us try to simplify the right side of (3):
By the rules of exponents: $.5\ e^{j\omega k} + .5\ e^{j\omega k}e^{-j\omega}$
Factoring out .5 and $e^{j\omega k}$: $.5(1+e^{-j\omega})e^{j\omega k}$

But, $e^{j\omega k}$ is just X(k), so
(4) $Y(k) = .5(1+e^{-j\omega})X(k)$

If we define the *transfer function* of a filter, H(ω), as the ratio of the output Y(k) to the input X(k), then

(5) $Y(k) = H(\omega)X(k)$

and in this case:

(6) $H(\omega) = .5(1+e^{-j\omega})$

What is H(ω) and how can we understand it? H(ω) is a complex number that is multiplied by the input phasor to get the output phasor. It is sometimes called the *frequency response* of the filter, because it describes what happens to individual frequencies in the input. What is the effect of multiplication by H?

First, it is important to understand that H(ω) is just some complex number with a magnitude and a phase, even though it is not yet clear what its magnitude and phase are, given the somewhat complicated-looking expression for it, shown in (6). However, for any value of ω, we could numerically calculate the value of H and obtain either the real and imaginary parts, or the magnitude and angle. The latter is more useful for our purposes. Given the magnitude and angle, we can express H(ω) in its complex exponential form, as in (7):

(7) $H(\omega) = |H(\omega)|\, e^{j\,\mathrm{Arg}\,H(\omega)}$

and then the expression for Y(k) becomes:

(8) $Y(k) = |H(\omega)|\, e^{j(\omega k + \mathrm{Arg}\,H(\omega))}$

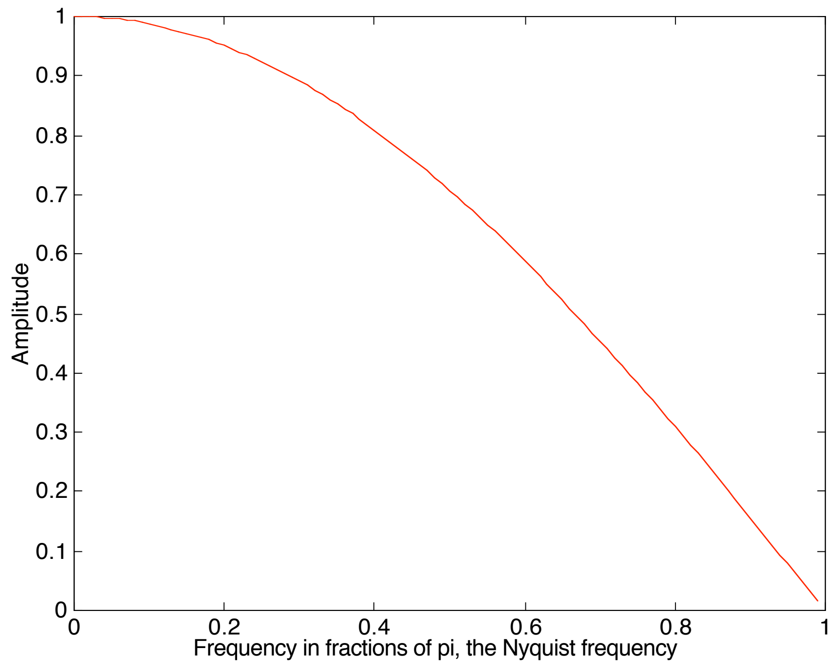
By looking at (8), we know that Y(k) is a phasor whose amplitude is $|H(\omega)|$. $|H(\omega)|$ is called the *amplitude response* of the filter, and can be calculated for any value of ω. (We will do this below).

We also know from looking at exponent of *e* that Y(k) is a phasor with frequency ω radians per sample (same as X(k)). The only difference between Y and X is that the complex exponent for Y(k) has *Arg H(ω)* added to it, compared to X(k). The effect of adding this angle is that every point in the phasor is shifted around the circle by *Arg H(ω)* radians, compared to X(k). This means that the phase of the phasor is shifted by this many radians. Put another way, (see `shift_demo` discussed in section IV), the starting point for rotation in the complex plane is shifted by *Arg H(ω)* radians from zero (which was its value for X(k)). After that starting point, everything is the same.

The script `AVERAGE2PTS_RESP` calculates the amplitude and phase response of our simple two point averaging filter, by creating a vector of ω with 100 equally spaced frequencies (between 0 and π radians per sample), substituting the value of (each) ω into equation (6) to get the value of the complex transfer function (H) , and then taking the magnitude and angle of H to get the amplitude and phase response.

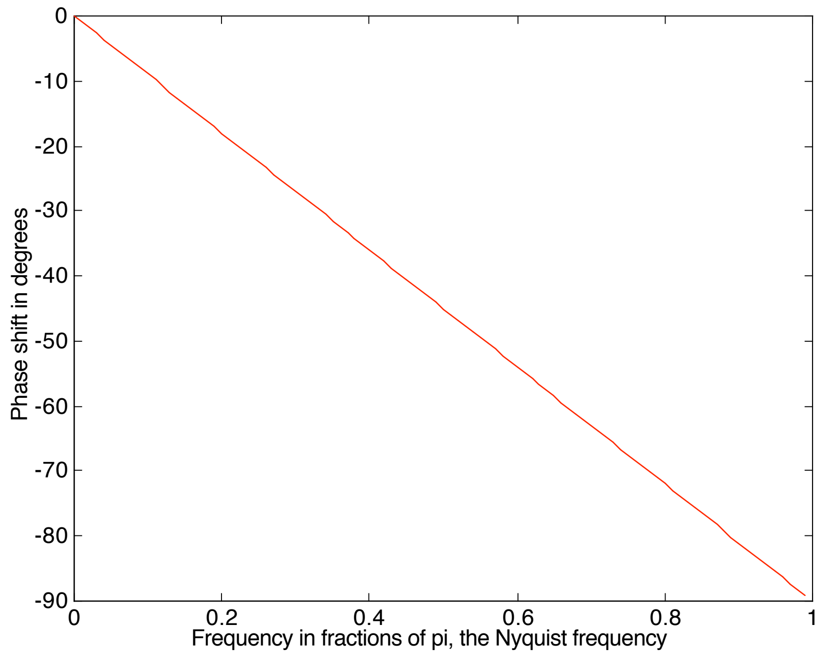
```
(9)
% AVERAGE2PTS_RESP
% Louis Goldstein
% 31 March 1996
% Use MATLAB to sketch magnitude and phase of H(w)
% for the simple average2pts filter.
%
% Set up a vector of test frequencies to plot, from
% zero to the Nyquist frequency
% Frequencies are represented in radians per sample
nfreqs = 100;
w = linspace (0, pi, nfreqs);
%
% Give the expression for the value of the transfer function
% H as a function of frequency
% This will compute a vector of H values, one for each value of w.
% This is equation (6) above
H = .5 * (1 + exp(-j*w));
%
% Plot the Amplitude of H as a function of frequency:
plot (w/pi,abs(H),'r')
xlabel ('Frequency in fractions of pi, the Nyquist frequency')
ylabel ('Amplitude')
pause
%
% Plot the phase of H as a function of frequency
plot (w/pi,angle(H)*180/pi, 'r')
xlabel ('Frequency in fractions of pi, the Nyquist frequency')
ylabel ('Phase shift in degrees')
```

The amplitude response function that results from running is `AVERAGE2PTS_RESP` shown in (10):
(10)



Note that the low frequencies are passed through the filter with little attenuation of amplitude, while the high frequencies are severely attenuated. This means that the filter is a *low-pass* filter.
The phase response is:

(11)



Note that phase of the output is linearly related to the frequency of the input (this is called a *linear phase* filter). In fact, we can see that the phase shift is equal to half the frequency—a frequency of π radians per sample, is shifted by 90 degrees (π/2 radians). Why is this so? Recall that ω is measured in radians per sample. What this means is that the phase shift is always equal to one half the angle traversed by the phasor in one sample. Since the filter averages the input signal and the input signal delayed by one sample, it makes sense that the resulting phase is delayed by one half a sample.

Complex inputs

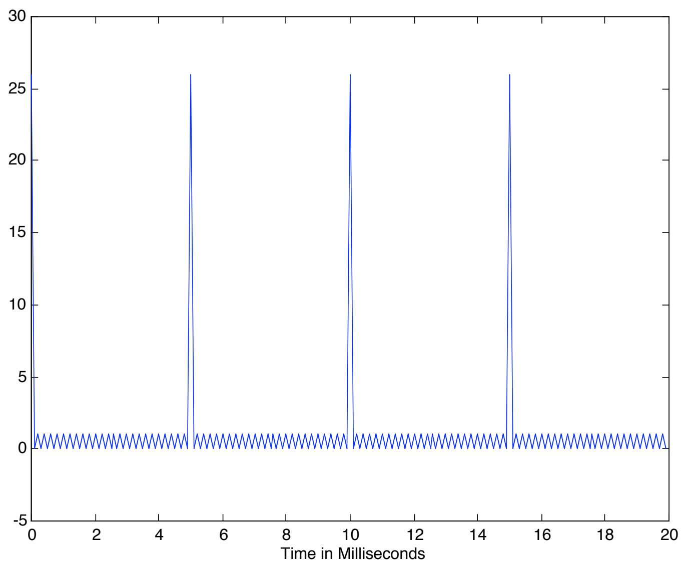
Given the definition of H above, it is possible to show that filtering is distributive:

If:
$X \rightarrow H \rightarrow Y$
$W \rightarrow G \rightarrow Z$

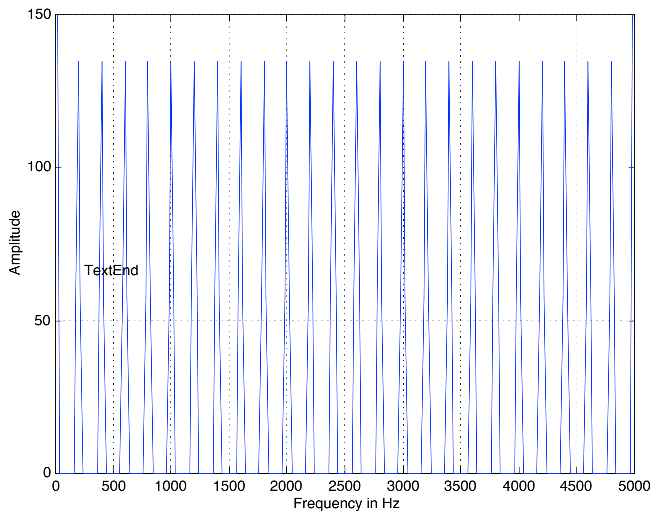
Then:
$X+W \rightarrow H \rightarrow Y+Z$

So let is look at what happens when a signal containing sinusoids at 200 Hz intervals from 200 to 5000 Hz is filtered through our filter?

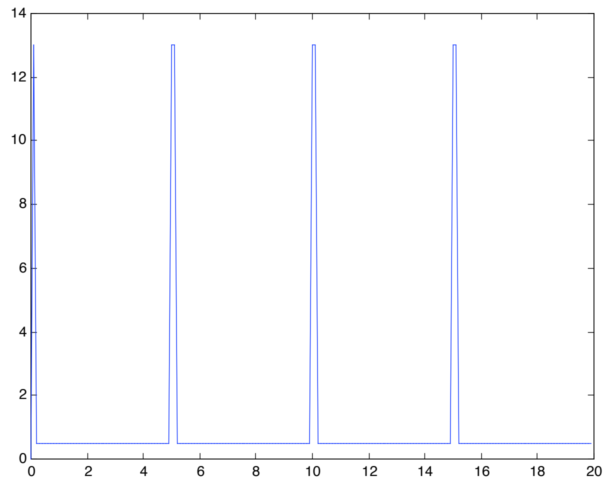
original signal (buzz):



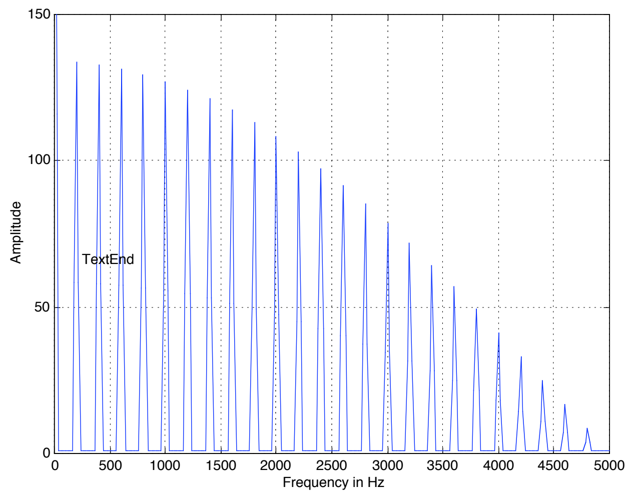
spectrum (buzz, srate)



filtered signal



filtered signal spectrum



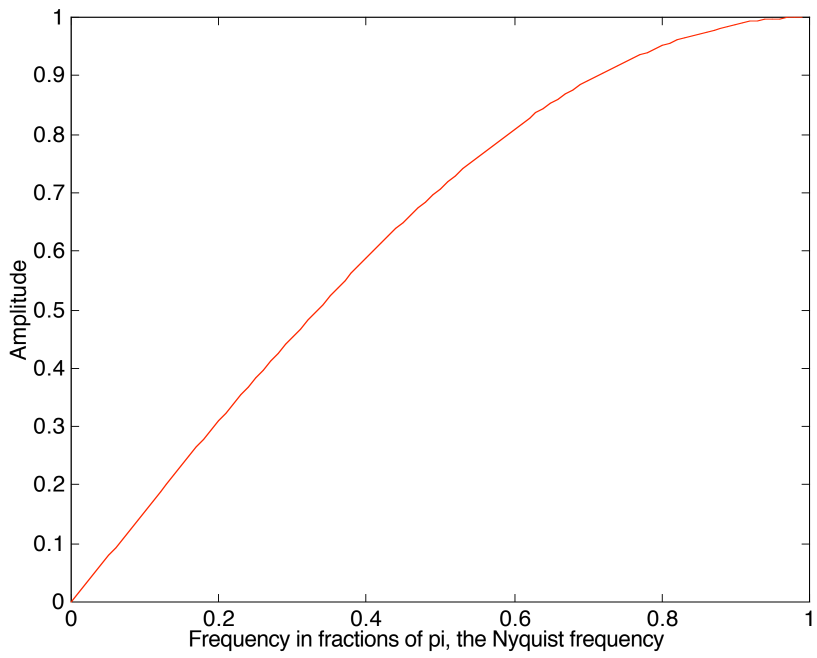
The comparable script for our two-point differencing filter is shown in (12). Note that it is different from the one for adding two points only in one line, that gives the value of H.

(12)

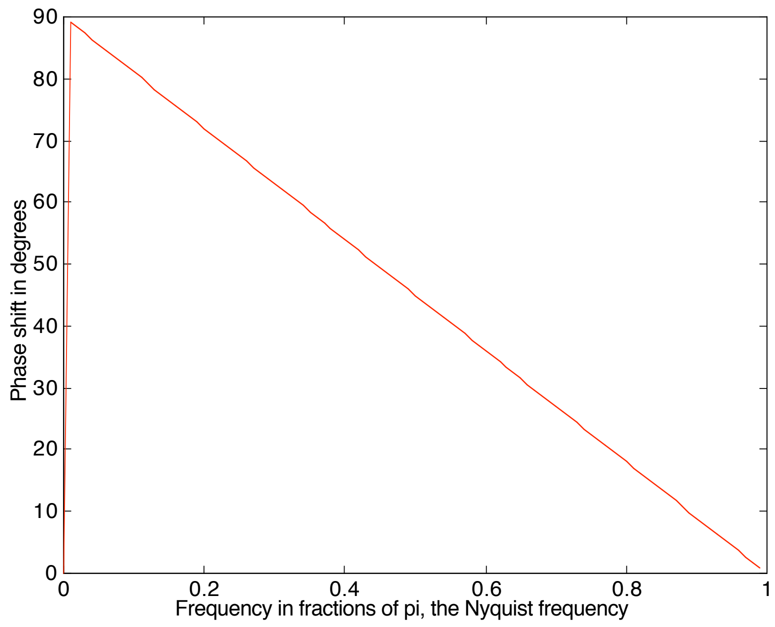
```
% DIFF2PTS_RESP
% Louis Goldstein
% 31 March 1996
% Use MATLAB to sketch magnitude and phase of H(w)
% for the simple diff2pts filter.
%
% Set up a vector of test frequencies to plot, from
% zero to the Nyquist frequency
% Frequencies are represented in radians per sample
nfreqs = 100;
w = linspace (0, pi, nfreqs);
%
% Give the expression for the value of the transfer function
% H as a function of frequency
% This will compute a vector of H values, one for each value of w.
% This is the only line that differs from average2pts, see (9) above
H = .5 * (1 - exp(-j*w));
%
% Plot the Amplitude of H as a function of frequency:
plot (w/pi,abs(H),'r')
xlabel ('Frequency in fractions of pi, the Nyquist frequency')
ylabel ('Amplitude')
pause
%
% Plot the phase of H as a function of frequency
plot (w/pi,angle(H)*180/pi, 'r')
xlabel ('Frequency in fractions of pi, the Nyquist frequency')
ylabel ('Phase shift in degrees')
```

The amplitude response, shown in (13) shows that this filter is a *high-pass* filter.

(13)



And the phase response in (14) shows that it is also a linear phase filter.
(14)



It is possible to simplify the expression for H(ω) algebraically, and show the magnitude and angles of H, as real functions of ω. (I don't expect you to be able to do this). In the case of the `average2pts` filter, the magnitude and angle of H(w) are as follows:

(15) $|H(\omega)| = |\cos\frac{\omega}{2}|$

(16) $\text{Arg } H(\omega) = \frac{\omega}{2}$

Given the nature of the cosine function, you can see that the filter will be low-pass. The comparable values for the `diff2pts` filter are as follows:

(17) $|H(\omega)| = |\sin\frac{\omega}{2}|$

(19) Arg $H(\omega) = \frac{\pi}{2} - \frac{\omega}{2}$

Note that all of the information here was calculated directly using the scripts above.

III. Generalized moving average filter

In general, moving average filters may be constructed to make a given output (Y(k)) point equal to the weighted sum of the current input point (X(k)) and some arbitrary number (M-1) of previous input points (X(k-1), X(k-2), etc). An equation for this generalized moving average filter is given in (1). (Note that the weighting coefficients (b(1) through b(M)) may be either positive or negative, resulting in "truly" averaging or in differencing type filters, respectively. Also the value of a given coefficient may be 0, which means that particular input sample will not contribute to the output sample. In general, the more (positive) coefficients are employed, the smoother the output will be.

(20) $Y(k) = b(1)\cdot X(k) + b(2)\cdot X(k-1) + \ldots + b(M)\cdot X(k-M+1)$

A MATLAB function filtering an input signal (x) by an arbitrary set of moving average filter coefficients (b) is given in (2)

```
(21)
function y = gen_move_aver (b,x)
% GEN_MOVE_AVER
% Louis Goldstein
% 31 March 1996
% Filter an input sequence x by a moving average filter
% whose coefficients are in b.
% Undefined values of y (from k = 1 to M-1) are left as zero.
M = length(b);
N = length(x);
y = zeros (1,N);        %this create a vector with N zeros
for k = M:N
      for j = 1:M
         y(k) = y(k) + b(j)*x(k-j+1);
      end
end
```

To use this function, first a vector of coefficients must be defined. To duplicate the effect of the filter in `average2pts`, set the coefficients to [.5 .5] as in (22) below. Applying the `gen_move_aver` function to our input sequence (X), then yields the same result as `average2pts`, as shown in (22).

(22)
```
»b = [.5 .5];

»Y = gen_move_aver (b,X)
Y =
  Columns 1 through 7
       0    1.0000    3.0000    5.5000    6.0000    7.0000   10.0000
  Columns 8 through 12
    9.5000    6.5000    5.5000    5.0000    2.5000
```

Likewise, setting the coefficients to [.5 -.5] duplicates the effect of `diff2pts`, as is shown in (23).

(23)
```
»b = [.5 -.5];

»Y = gen_move_aver (b,X)
Y =
  Columns 1 through 7
       0    1.0000    1.0000    1.5000   -1.0000    2.0000    1.0000
  Columns 8 through 12
   -1.5000   -1.5000    0.5000   -1.0000   -1.5000
```

In MATLAB, there is a built-in function `filter` that performs filtering like `gen_move_aver` (but works more efficiently). It takes three arguments:

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

b     is the vector of coefficients
a     we will discuss in the future, for now, set it to 1
x     input vector
y     output vector
```

IV. Amplitude and Phase response of a generalized moving average filter

The general expression for a moving average filter is repeated here in (24):
(24) $Y(k) = b(1)\cdot X(k) + b(2)\cdot X(k-1) + ... + b(M)\cdot X(k-M+1)$

We can derive an expression for the frequency response of this generalized filter, as follows. First, substitute the phasor expression for X(k):
(25) $X(k) = e^{j\omega k}$

This yields:
(26) $Y(k) = b(1)e^{j\omega k} + b(2)e^{j\omega(k-1)} + b(3)e^{j\omega(k-2)} +... + b(M)e^{j\omega(k-M+1)}$

By rules of exponentiation, this is:
(27) $Y(k) = b(1)e^{j\omega k} + b(2)e^{j\omega k}e^{-j\omega} + b(3)e^{j\omega k}e^{-2j\omega} + \ldots + b(M)e^{j\omega k}e^{-(M-1)j\omega}$

Now factoring out $e^{j\omega k}$:
(28) $Y(k) = [b(1) + b(2)e^{-j\omega} + b(3)e^{-2j\omega} + \ldots + b(M)e^{-(M-1)j\omega}]e^{i\omega k}$

Of course, $e^{j\omega k}$ is just X(k):
(29) $Y(k) = [b(1) + b(2)e^{-j\omega} + b(3)e^{-2j\omega} + \ldots + b(M)e^{-(M-1)j\omega}]X(k)$

Thus, the transfer function is:
(30) $H(\omega) = b(1) + b(2)e^{-j\omega} + b(3)e^{-2j\omega} + \ldots + b(M)e^{-(M-1)j\omega}$

Let us define a complex variable z:
(31) $z = e^{j\omega}$

This lets us re-write (30) in simpler form, using z.

(32) $H(z) = b(1) + b(2)z^{-1} + b(3)z^{-2} + \ldots + b(M)z^{-(M-1)}$

Thus, the expression for the transfer function of an arbitrary moving average filter is a polynomial in negative powers of z. Each power corresponds to one of the previous samples included in the filter definition (Eq.24), and the coefficient of each polynomial term is the corresponding filter coefficient. Given equation (32), it would be very easy to write a MATLAB function to calculate the frequency response of an arbitrary filter. It would take an arbitrary b vector as input, and instead of the single line used to calculate H (as is done in `average2pts_resp and diff2pts_resp)`, a loop could be constructed to add up the weighted powers of z. In addition, the polynomial representation is useful because the roots of a polynomial can be determined (and there are functions in MATLAB for doing so), and the roots of the polynomial in (32) will tell us the values of ω at which the amplitude response is minimal (so-called "zeroes"). We will discuss this in the future.

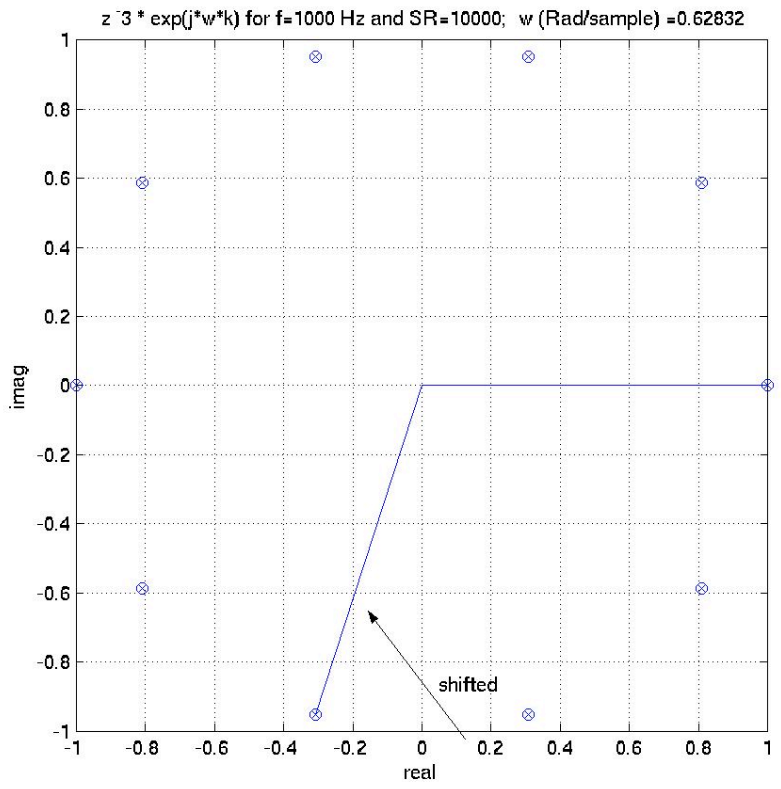
What is the meaning of these powers of *z* in the transfer function? Multiplying a phasor by z^N represents a time shift of the phasor of N samples . Why? This shift is demonstrated by the script in (33).

```
(33)
% SHIFT_DEMO
% Louis Goldstein
% 2 November 1992
% Modfied Jan 2003
% This demonstrates that a phasor is shifted in
% time by N samples when it is multiplied by z .^ N
% Also, the phase shift corresponds to w*n
% Where w is the frequency in radians per sample.

% Generate unshifted phasor:
srate = input ('Enter sampling rate in Hz: ')
f = input ('Enter sinusoid frequency in Hz: ')
N = input ('Enter power of z (shift operator): ')
dur = .1;
t = 0:1/srate:dur;
k = 0:dur*srate;              % k is sample number
w = 2*pi*f/srate;             % w is in radians per sample
unshifted = exp(j*w*k);

% Plot first ten samples in complex plane
% Begin by drawing a line from the origin to the first point.

first = [0 unshifted(1)];
plot (real(first), imag(first), 'r');
axis ('square');
axis ([-1,1,-1,1]);
hold on
xlabel ('real')
ylabel ('imag')
grid
for i = 1:10
  plot (real(unshifted (i)), imag(unshifted(i)), 'or')
            pause(1)
end
title ([' exp(j*w*k) for f=',num2str(f), ' Hz and SR=', num2str(srate),
';  w (Rad/sample) =', num2str(w)])
pause

% Now shift signal by muliplying by z ^ N
z = exp(j*w) ^ N;
shifted = unshifted * z;

% Plot first ten shifted samples in complex plane
first = [0 shifted(1)];
plot (first, 'g')
for i = 1:10
  plot (real(shifted (i)), imag(shifted(i)), 'xg')
            pause(1)
end
title ([' z ^', num2str(N), ' * exp(j*w*k) for f=',num2str(f), ' Hz and SR=',
num2str(srate),         ';  w (Rad/sample) =', num2str(w)])
pause

hold off

% Now plot unshifted and shifted signals as a function of time
y = real(unshifted);
axis ('normal')
subplot (211), plot (t(1:100), y(1:100), 'or-')
title (['real (exp(j*w), for f=',num2str(f),' Hz and SR=', num2str(srate)])
xlabel ('time (in secs)')

pause
```

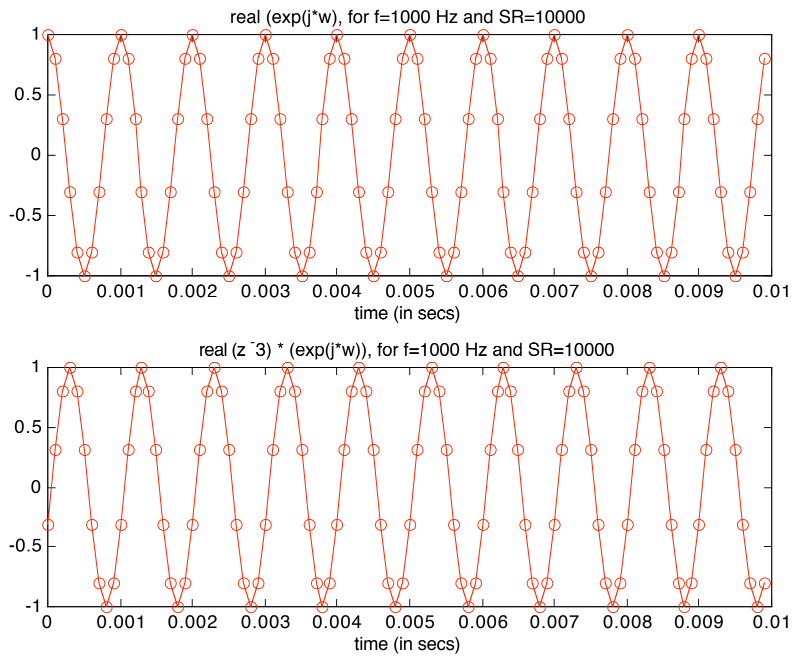
```
soundsc (y,srate)

y = real(shifted);
axis ('normal')
subplot (212), plot (t(1:100), y(1:100), 'or-')
title (['real (z ^', num2str(N), ') * (exp(j*w)), for f=',num2str(f),' Hz and
SR=', num2str(srate)])
xlabel ('time (in secs)')

pause
soundsc (y,srate)
```

Below are the results of running this script and requesting a sampling frequency of 10000 Hz, a sinusoid frequency of 1000 Hz, and -3 as a power of z. (34) shows the first 10 points of the phasor plotted in the complex plane, both with and without shifting (multiplication by z^N). The line in the complex plane represents the first point (k=0) plotted.

(34)



(35) shows the real part of the phasor, plotted as a function of time, both with and without shifting.
(35)


real (exp(j*w), for f=1000 Hz and SR=10000
time (in secs)
real (z ^-3) * (exp(j*w)), for f=1000 Hz and SR=10000
time (in secs)