# Signal Processing

Young Won Lim
2/19/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

# Based on

Signal Processing with Free Software : Practical Experiments
F. Auger

# IIR Filter Design (1)

**besselap** Return bessel analog filter prototype.
**besself** Generate a Bessel filter.
**bilinear** Transform a s-plane filter specification into a z-plane specification.
**buttap** Design lowpass analog Butterworth filter.
**butter** Generate a Butterworth filter.
**buttord** Compute the minimum filter order of a Butterworth filter with the desired response characteristics.
**cheb** Returns the value of the nth-order Chebyshev polynomial calculated at the point x.
**cheb1ap** Design lowpass analog Chebyshev type I filter.
**cheb1ord** Compute the minimum filter order of a Chebyshev type I filter with the desired response characteristics.
**cheb2ap** Design lowpass analog Chebyshev type II filter.
**cheb2ord** Compute the minimum filter order of a Chebyshev type II filter with the desired response characteristics.
**cheby1** Generate a Chebyshev type I filter with RP dB of passband ripple.
**cheby2** Generate a Chebyshev type II filter with RS dB of stopband attenuation.

https://octave.sourceforge.io/signal/overview.html

# IIR Filter Design (2)

**ellip** Generate an elliptic or Cauer filter with RP dB of passband ripple and RS dB of stopband attenuation.
**ellipap** Design lowpass analog elliptic filter.
**ellipord** Compute the minimum filter order of an elliptic filter with the desired response characteristics.
**iirlp2mb** IIR Low Pass Filter to Multiband Filter Transformation
**impinvar** Converts analog filter with coefficients B and A to digital, conserving impulse response.
**invimpinvar** Converts digital filter with coefficients B and A to analog, conserving impulse response.
**ncauer** usage: [Zz, Zp, Zg] = ncauer(Rp, Rs, n)
**pei_tseng_notch** Return coefficients for an IIR notch-filter with one or more filter frequencies and according (very narrow) bandwidths to be used with 'filter' or 'filtfilt'.
**sftrans** Transform band edges of a generic lowpass filter (cutoff at W=1) represented in splane zero-pole-gain form.

https://octave.sourceforge.io/signal/overview.html

# FIR Filter Design

**cl2bp** Constrained L2 bandpass FIR filter design.
**fir1** Produce an order N FIR filter with the given frequency cutoff W, returning the N+1 filter coefficients in B.
**fir2** Produce an order N FIR filter with arbitrary frequency response M over frequency bands F, returning the N+1 filter coefficients in B.
**firls** FIR filter design using least squares method.
**kaiserord** Return the parameters needed to produce a filter of the desired specification from a Kaiser window.
**qp_kaiser** Computes a finite impulse response (FIR) filter for use with a quasi-perfect reconstruction polyphase-network filter bank.
**remez** Parks-McClellan optimal FIR filter design.
**sgolay** Computes the filter coefficients for all Savitzsky-Golay smoothing filters of order p for length n (odd).

https://octave.sourceforge.io/signal/overview.html

# filter (1)

```
: y = filter (b, a, x)
: [y, sf] = filter (b, a, x, si)
: [y, sf] = filter (b, a, x, [], dim)
: [y, sf] = filter (b, a, x, si, dim)
```

https://octave.sourceforge.io/octave/function/filter.html

# filter (2)

Apply a 1-D digital filter to the data x.

filter returns the solution to the following
linear, time-invariant difference equation:

$$\sum_{k=0}^{N} a(k+1)\,y(n-k) \;=\; \sum_{k=0}^{M} b(k+1)\,x(n-k) \qquad for \;\; 1 \le n \le length(x)$$

where N=length(a)-1 and M=length(b)-1.
The result is calculated over the first non-singleton dimension of x
or over dim if supplied.

An equivalent form of the equation is:

$$y(n) \;=\; -\sum_{k=1}^{N} c(k+1)\,y(n-k) + \sum_{k=0}^{M} d(k+1)\,x(n-k) \qquad for\, 1 \le n \le length(x)$$

where c = a/a(1) and d = b/a(1).

https://octave.sourceforge.io/octave/function/filter.html

# filter (3)

If the fourth argument si is provided,
it is taken as the initial state of the system and
the final state is returned as sf.
The state vector is a column vector whose length is equal to
the length of the longest coefficient vector minus one.
If si is not supplied, the initial state vector is set to all zeros.

In terms of the Z Transform,
y is the result of passing the discrete-time signal x
through a system characterized
by the following rational system function:

$$H(z) = \frac{\sum_{k=0}^{M} d(k+1) z^{-k}}{1 + \sum_{k=1}^{N} c(k+1) z^{-k}}$$

https://octave.sourceforge.io/octave/function/filter.html

# filter2 (1)

```
: y = filter2 (b, x)
: y = filter2 (b, x, shape)
```

https://octave.sourceforge.io/octave/function/filte2r.html

# filter2 (2)

Apply the 2-D FIR filter b to x.

If the argument shape is specified,
return an array of the desired shape. Possible values are:

| | |
|---|---|
| "full" | pad x with zeros on all sides before filtering. |
| "same" | unpadded x (default) |
| "valid" | trim x after filtering so edge effects are no included. |

Note this is just a variation on convolution,
with the parameters reversed and b rotated 180 degrees.

https://octave.sourceforge.io/octave/function/filte2r.html

# freqz (1)

```
: [h, w] = freqz (b, a, n, "whole")
: [h, w] = freqz (b)
: [h, w] = freqz (b, a)
: [h, w] = freqz (b, a, n)
: h = freqz (b, a, w)
: [h, w] = freqz (..., Fs)
: freqz (...)
```

https://octave.sourceforge.io/octave/function/freqz.html

# freqz (2)

Return the complex frequency response h of the rational IIR filter
whose numerator and denominator coefficients are b and a, respectively.

The response is evaluated at n angular frequencies between 0 and 2*pi.

The output value w is a vector of the frequencies.

If a is omitted, the denominator is assumed to be 1
(this corresponds to a simple FIR filter).

If n is omitted, a value of 512 is assumed.
For fastest computation, n should factor into a small number of small primes.

If the fourth argument, "whole", is omitted
the response is evaluated at frequencies between 0 and pi.

https://octave.sourceforge.io/octave/function/freqz.html

# freqz (3)

**freqz** (b, a, w)

Evaluate the response at the specific frequencies in the vector w.
The values for w are measured in radians.

[...] = **freqz** (..., Fs)

Return frequencies in Hz instead of radians assuming a sampling rate Fs.
If you are evaluating the response at specific frequencies w,
those frequencies should be requested in Hz rather than radians.

**freqz** (...)

Plot the magnitude and phase response of h rather than returning them.

https://octave.sourceforge.io/octave/function/freqz.html

# freqz_plot

: **freqz_plot** (w, h)
: **freqz_plot** (w, h, freq_norm)

Plot the magnitude and phase response of h.

If the optional freq_norm argument is true,
the frequency vector w is in units of normalized radians.
If freq_norm is false, or not given, then w is measured in Hertz.

https://octave.sourceforge.io/octave/function/freqz_plot.html

# conv

: **conv** (a, b)
: **conv** (a, b, shape)

Convolve two vectors a and b.

The output convolution is a vector with length equal to
length (a) + length (b) - 1.
When a and b are the coefficient vectors of two polynomials,
the convolution represents the coefficient vector of the product polynomial.

The optional shape argument may be

shape = "full"

Return the full convolution. (default)

shape = "same"

Return the central part of the convolution with the same size as a.

https://octave.sourceforge.io/octave/function/conv.html

# conv2

: **conv2** (A, B)
: **conv2** (v1, v2, m)
: **conv2** (..., shape)

Return the 2-D convolution of A and B.

The size of the result is determined by the optional shape argument which takes the following values

shape = "full" Return the full convolution. (default)
shape = "same" Return the central part of the convolution with the same size as A. The central part of the convolution begins at the indices floor ([size(B)/2] + 1).
shape = "valid" Return only the parts which do not include zero-padded edges. The size of the result is max (size (A) - size (B) + 1, 0).

When the third argument is a matrix, return the convolution of the matrix m by the vector v1 in the column direction and by the vector v2 in the row direction.

https://octave.sourceforge.io/octave/function/conv2.html

# fftconv

: **fftconv** (x, y)
: **fftconv** (x, y, n)

Convolve two vectors using the FFT for computation.

c = fftconv (x, y) returns a vector of length equal to length (x) + length (y) - 1. If x and y are the coefficient vectors of two polynomials, the returned value is the coefficient vector of the product polynomial.

The computation uses the FFT by calling the function fftfilt. If the optional argument n is specified, an N-point FFT is used.

See also: deconv, conv, conv2.

https://octave.sourceforge.io/octave/function/fftconv.html

# deconv

: **deconv** (y, a)

Deconvolve two vectors.

[b, r] = deconv (y, a) solves for b and r such that y = conv (a, b) + r.

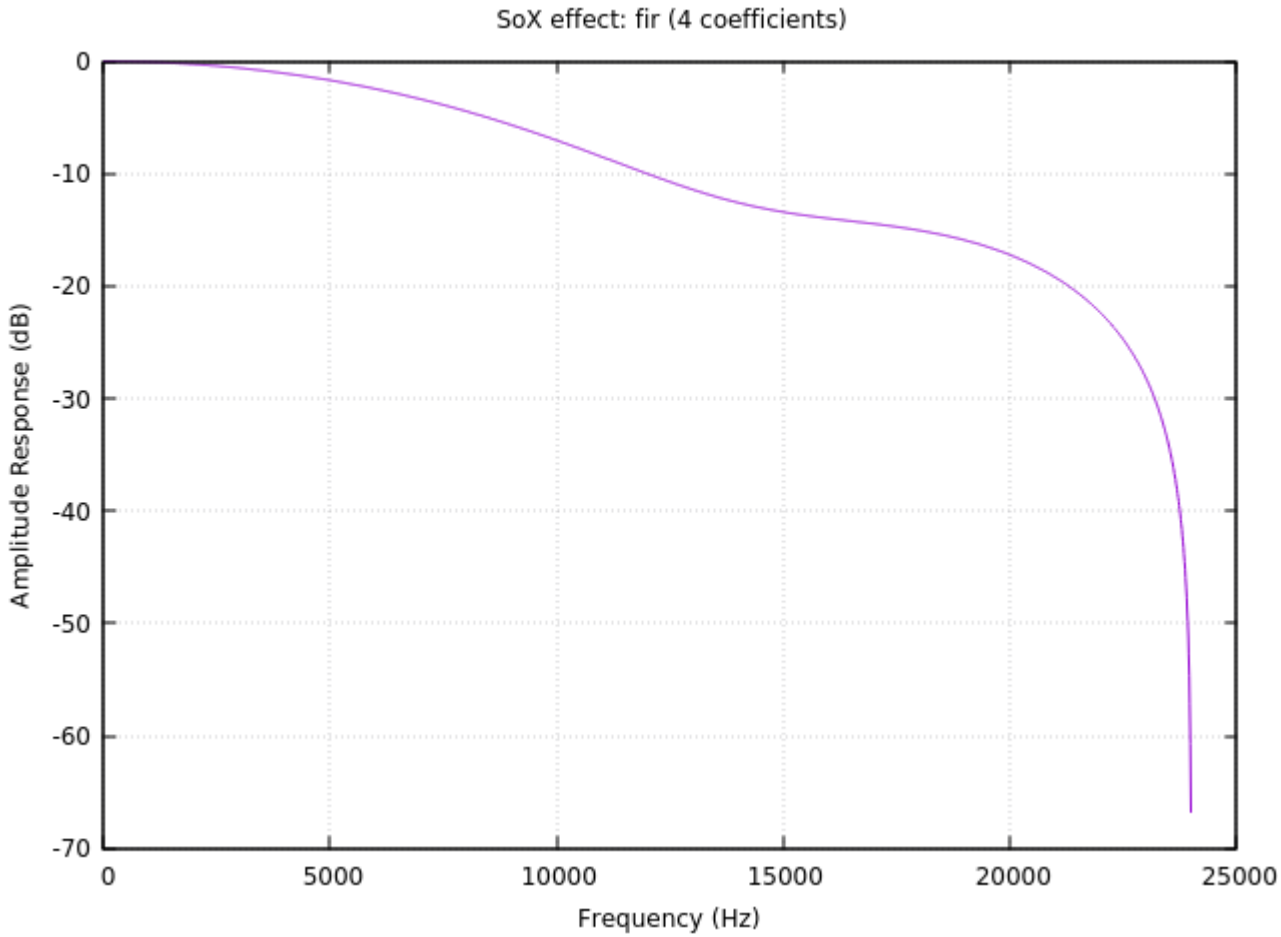
If y and a are polynomial coefficient vectors, b will contain the coefficients of the polynomial quotient and r will be a remainder polynomial of lowest order.

https://octave.sourceforge.io/octave/function/deconv.html

# --plot gnuplot | octave

```
sox --plot gnuplot s6s.wav -n fir 0.1 0.2 0.4 0.3          >fir1.plt
sox --plot gnuplot s6s.wav -n fir coeff.txt                >fir2.plt
sox --plot gnuplot s6s.wav -n biquad .6 .2 .4 1 -1.5 .6 >fir3.plt
sox --plot gnuplot s6s.wav -n fir 0.2 0.2 0.2 0.2 0.2      >fir4.plt
```

# --plot gnuplot | octave



## References

[1] F. Auger, Signal Processing with Free Software : Practical Experiments