Previous topic

gnuradio.fft

Next topic

gnuradio.trellis

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

gnuradio.filter

class gnuradio.filter.filterbank.**analysis_filterbank**(*mpoints*, *taps=None*)

Uniformly modulated polyphase DFT filter bank: analysis

See http://cnx.org/content/m10424/latest

class gnuradio.filter.filterbank.**synthesis_filterbank**(*mpoints*, *taps=None*)

Uniformly modulated polyphase DFT filter bank: synthesis

See http://cnx.org/content/m10424/latest

class gnuradio.filter.**firdes**

Finite Impulse Response (FIR) filter design functions.

gnuradio.filter.**pm_remez**(*int order*, *pmt_vector_double bands*, *pmt_vector_double ampl*, *pmt_vector_double error_weight*, *std::string const filter_type*, *int grid_density=16*) → pmt_vector_double

Parks-McClellan FIR filter design using Remez algorithm.

Calculates the optimal (in the Chebyshev/minimax sense) FIR filter inpulse response given a set of band edges, the desired response on those bands, and the weight given to the error in those bands.

Frequency is in the range [0, 1], with 1 being the Nyquist frequency (Fs/2)

class gnuradio.filter.**synthesis_filterbank**(*mpoints*, *taps=None*)

Uniformly modulated polyphase DFT filter bank: synthesis

See http://cnx.org/content/m10424/latest

class gnuradio.filter.**analysis_filterbank**(*mpoints*, *taps=None*)

Uniformly modulated polyphase DFT filter bank: analysis

See http://cnx.org/content/m10424/latest

class gnuradio.filter.**freq_xlating_fft_filter_ccc**(*decim*, *taps*, *center_freq*, *samp_rate*)

gnuradio.filter.optfir.**low_pass**(*gain*, *Fs*, *freq1*, *freq2*, *passband_ripple_db*, *stopband_atten_db*, *nextra_taps=2*)

Builds a low pass filter.

Parameters:
- **gain** – Filter gain in the passband (linear)
- **Fs** – Sampling rate (sps)
- **freq1** – End of pass band (in Hz)
- **freq2** – Start of stop band (in Hz)
- **passband_ripple_db** – Pass band ripple in dB (should be small, < 1)
- **stopband_atten_db** – Stop band attenuation in dB (should be large, >= 60)
- **nextra_taps** – Extra taps to use in the filter (default=2)

gnuradio.filter.optfir.**band_pass**(*gain*, *Fs*, *freq_sb1*, *freq_pb1*, *freq_pb2*, *freq_sb2*, *passband_ripple_db*, *stopband_atten_db*, *nextra_taps=2*)

Builds a band pass filter.

Parameters:

- **gain** – Filter gain in the passband (linear)
- **Fs** – Sampling rate (sps)
- **freq_sb1** – End of stop band (in Hz)
- **freq_pb1** – Start of pass band (in Hz)
- **freq_pb2** – End of pass band (in Hz)
- **freq_sb2** – Start of stop band (in Hz)
- **passband_ripple_db** – Pass band ripple in dB (should be small, < 1)
- **stopband_atten_db** – Stop band attenuation in dB (should be large, >= 60)
- **nextra_taps** – Extra taps to use in the filter (default=2)

`gnuradio.filter.optfir.`**`complex_band_pass`**(*gain*, *Fs*, *freq_sb1*, *freq_pb1*, *freq_pb2*, *freq_sb2*, *passband_ripple_db*, *stopband_atten_db*, *nextra_taps=2*)

Builds a band pass filter with complex taps by making an LPF and spinning it up to the right center frequency

Parameters:

- **gain** – Filter gain in the passband (linear)
- **Fs** – Sampling rate (sps)
- **freq_sb1** – End of stop band (in Hz)
- **freq_pb1** – Start of pass band (in Hz)
- **freq_pb2** – End of pass band (in Hz)
- **freq_sb2** – Start of stop band (in Hz)
- **passband_ripple_db** – Pass band ripple in dB (should be small, < 1)
- **stopband_atten_db** – Stop band attenuation in dB (should be large, >= 60)
- **nextra_taps** – Extra taps to use in the filter (default=2)

`gnuradio.filter.optfir.`**`band_reject`**(*gain*, *Fs*, *freq_pb1*, *freq_sb1*, *freq_sb2*, *freq_pb2*, *passband_ripple_db*, *stopband_atten_db*, *nextra_taps=2*)

Builds a band reject filter spinning it up to the right center frequency

Parameters:

- **gain** – Filter gain in the passband (linear)
- **Fs** – Sampling rate (sps)
- **freq_pb1** – End of pass band (in Hz)
- **freq_sb1** – Start of stop band (in Hz)
- **freq_sb2** – End of stop band (in Hz)
- **freq_pb2** – Start of pass band (in Hz)
- **passband_ripple_db** – Pass band ripple in dB (should be small, < 1)
- **stopband_atten_db** – Stop band attenuation in dB (should be large, >= 60)
- **nextra_taps** – Extra taps to use in the filter (default=2)

`gnuradio.filter.optfir.`**`stopband_atten_to_dev`**(*atten_db*)

Convert a stopband attenuation in dB to an absolute value

`gnuradio.filter.optfir.`**`passband_ripple_to_dev`**(*ripple_db*)

Convert passband ripple spec expressed in dB to an absolute value

`gnuradio.filter.optfir.`**`remezord`**(*fcuts*, *mags*, *devs*, *fsamp=2*)

FIR order estimator (lowpass, highpass, bandpass, mulitiband).

(n, fo, ao, w) = remezord (f, a, dev) (n, fo, ao, w) = remezord (f, a, dev, fs)

(n, fo, ao, w) = remezord (f, a, dev) finds the approximate order, normalized frequency band edges, frequency band amplitudes, and weights that meet input specifications f, a, and dev, to use with the remez command.

- f is a sequence of frequency band edges (between 0 and Fs/2, where Fs is the sampling frequency), and a is a sequence specifying the desired amplitude on the bands defined by f. The length of f is twice the length of a, minus 2. The desired function is piecewise constant.

- dev is a sequence the same size as a that specifies the maximum allowable deviation or ripples between the frequency response and the desired amplitude of the output filter, for each band.

Use remez with the resulting order n, frequency sequence fo, amplitude response sequence ao, and weights w to design the filter b which approximately meets the specifications given by remezord input parameters f, a, and dev:

b = remez (n, fo, ao, w)

(n, fo, ao, w) = remezord (f, a, dev, Fs) specifies a sampling frequency Fs.

Fs defaults to 2 Hz, implying a Nyquist frequency of 1 Hz. You can therefore specify band edges scaled to a particular applications sampling frequency.

In some cases remezord underestimates the order n. If the filter does not meet the specifications, try a higher order such as n+1 or n+2.

`gnuradio.filter.optfir.`**`lporder`**(*freq1*, *freq2*, *delta_p*, *delta_s*)

FIR lowpass filter length estimator. freq1 and freq2 are normalized to the sampling frequency. delta_p is the passband deviation (ripple), delta_s is the stopband deviation (ripple).

Note, this works for high pass filters too (freq1 > freq2), but doesn't work well if the transition is near f == 0 or f == fs/2

From Herrmann et al (1973), Practical design rules for optimum finite impulse response filters. Bell System Technical J., 52, 769-99

`gnuradio.filter.optfir.`**`bporder`**(*freq1*, *freq2*, *delta_p*, *delta_s*)

FIR bandpass filter length estimator. freq1 and freq2 are normalized to the sampling frequency. delta_p is the passband deviation (ripple), delta_s is the stopband deviation (ripple).

From Mintzer and Liu (1979)

class `gnuradio.filter.pfb.`**`channelizer_ccf`**(*numchans*, *taps=None*, *oversample_rate=1*, *atten=100*)

Make a Polyphase Filter channelizer (complex in, complex out, floating-point taps)

This simplifies the interface by allowing a single input stream to connect to this block. It will then output a stream for each channel.

class `gnuradio.filter.pfb.`**`interpolator_ccf`**(*interp*, *taps=None*, *atten=100*)

Make a Polyphase Filter interpolator (complex in, complex out, floating-point taps)

The block takes a single complex stream in and outputs a single complex stream out. As such, it requires no extra glue to handle the input/output streams. This block is provided to be consistent with the interface to the other PFB block.

class `gnuradio.filter.pfb.`**`decimator_ccf`**(*decim*, *taps=None*, *channel=0*, *atten=100*, *use_fft_rotators=True*, *use_fft_filters=True*)

Make a Polyphase Filter decimator (complex in, complex out, floating-point taps)

This simplifies the interface by allowing a single input stream to connect to this block. It will then output a stream that is the decimated output stream.

class `gnuradio.filter.pfb.`**`arb_resampler_ccf`**(*rate*, *taps=None*, *flt_size=32*, *atten=100*)

Convenience wrapper for the polyphase filterbank arbitrary resampler.

The block takes a single complex stream in and outputs a single complex stream out. As such, it requires no extra glue to handle the input/output streams. This block is provided to be consistent with the interface to the other PFB block.

class `gnuradio.filter.pfb.`**`arb_resampler_fff`**(*rate*, *taps=None*, *flt_size=32*, *atten=100*)

Convenience wrapper for the polyphase filterbank arbitrary resampler.

The block takes a single float stream in and outputs a single float stream out. As such, it requires no extra glue to handle the input/output streams. This block is provided to be consistent with the interface to the other PFB block.

class `gnuradio.filter.pfb.`**`arb_resampler_ccc`**(*rate*, *taps=None*, *flt_size=32*, *atten=100*)

Convenience wrapper for the polyphase filterbank arbitrary resampler.

The block takes a single complex stream in and outputs a single complex stream out. As such, it requires no extra glue to handle the input/output streams. This block is provided to be consistent with the interface to the other PFB block.

class `gnuradio.filter.pfb.`**`channelizer_hier_ccf`**(*n_chans*, *n_filterbanks=1*, *taps=None*, *outchans=None*, *atten=100*, *bw=1.0*, *tb=0.2*, *ripple=0.1*)

Make a Polyphase Filter channelizer (complex in, complex out, floating-point taps)

class `gnuradio.filter.`**`rational_resampler_fff`**(*interpolation*, *decimation*, *taps=None*, *fractional_bw=None*)

class `gnuradio.filter.`**`rational_resampler_ccf`**(*interpolation*, *decimation*, *taps=None*, *fractional_bw=None*)

class `gnuradio.filter.`**`rational_resampler_ccc`**(*interpolation*, *decimation*, *taps=None*, *fractional_bw=None*)