Non-Causal filtering Code (in matlab)
T2500234-v1

Brian Lantz

July 11, 2025

1 Introduction

I've created a new function in matlab to do non-causal bandpass filtering of time series data. The function is called `bandpass_viafft_fn_rolloff.m` and it lives in the {SeismicSVN}/seismic/Common/MatlabTools/ directory. There is also a simple test code named `bandpass_viafft_rolloff_test.m` which lives in the same directory. All the figures here were generated with that code.

In the frequency domain, the filter is a flat-topped band-pass filter between the frequencies $F_{lowpass}$ and $F_{highpass}$. It rolls off as $(F_{highpass}/freq)^n$ above the band-pass and rolls up at $(freq/F_{lowpass})^n$ below the band-pass. The phase is 0 at all frequencies. In the time domain, this preserves the shape of the signal in the pass-band, but the filter is non-causal.

The code works by taking the FFT of the time series, multiplying the FFT by a strictly real filter (i.e. non-causal) and taking the iFFT of the result. This returns a real timeseries of the same length as the original, but with certain frequencies attenuated. These are attenuated with no phase shift. This is particularly useful to look at microseismic data, but obviously not in real time. The code is attached at the end of this note. Use of the function is described in the help for the function.

2 Testing

I've done a few simple tests of the code. These test the code performance and give a sense of the code behavior. Basic parameters for these tests are

```
dT   = 0.01;
span = 100;
time = (0: dT: span-dT);

filt.lowpass  = 0.1;
filt.highpass = 0.3;
filt.rolloff  = 2;
signal_out = bandpass_viafft_fn_rolloff(signal_in, dT, filt.lowpass, filt.highpass,
    filt.rolloff,'q');
```

For these tests, the Nyguist frequency is 50 Hz and the frequency spacing is 1/100 Hz. The fft runs from 0 Hz to 99.99 Hz (if you are Matlab), or -49.99 Hz to 50 Hz (if you are a mathematician), or maybe that's -50 to +49.99 - clearly a bit of care is needed.

Figures 1 and 2 show the magnitude response of the filter. The phase isn't plotted because it is 0.

2.1 Nice Sine Waves

For sine waves which have an integer number of cycles in the 100 sec time window, the response is just what you would expect. The amplitude unchanged between 0.1 and 0.3 Hz, and is attenuated at other frequencies. There is no phase shift. Figure 3 through 5 show several examples of this. For all of these,

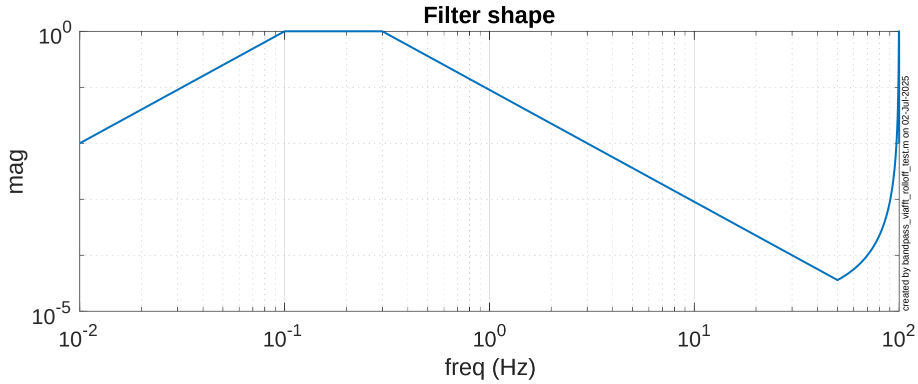

Figure 1: Magnitude of the bandpass filter used for the excess microseismic motion study. The pass-band is 0.1 to 0.3 Hz, and the filter rolls off as $1/freq^2$. The filter is symmetric about the Nyquist frequency.

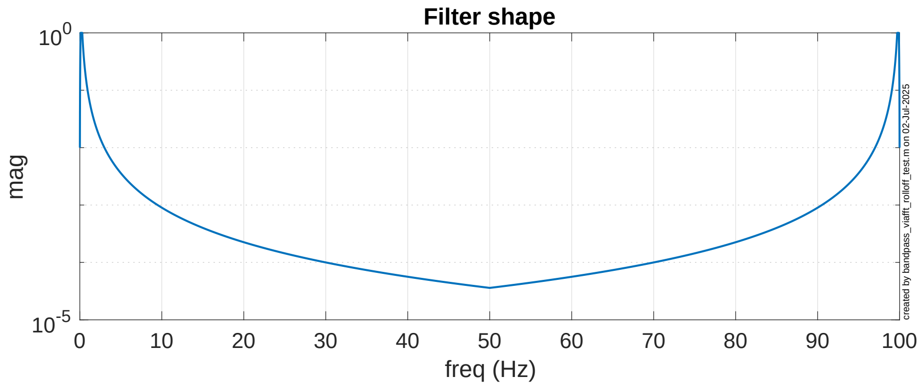

Figure 2: Band-pass filter plotted on a linear x scale to show the symmetry.

the blue is the input time series and the dashed red is the output time series after filtering. Because these sine waves have an integer number of cycles, they repeat exactly in the time domain, as calculated by the fft, and therefore there are no odd-looking transients at the beginning or the end of the time series.

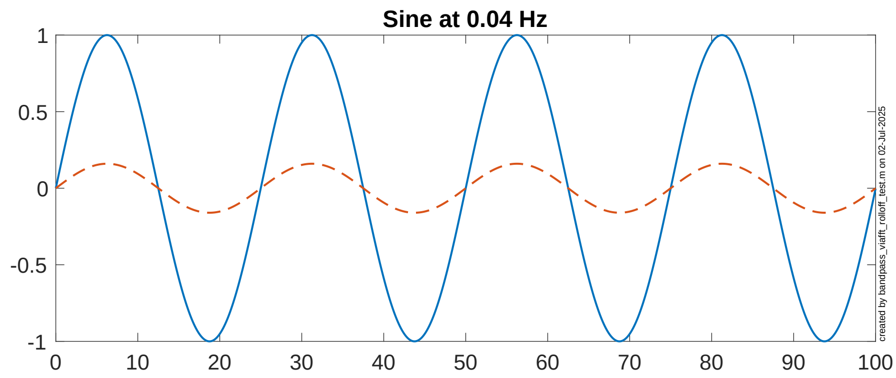

Figure 3: 40 mHz sine wave is attenuated by $(40/100)^2 = 0.16$. There is no phase shift of the output signal.

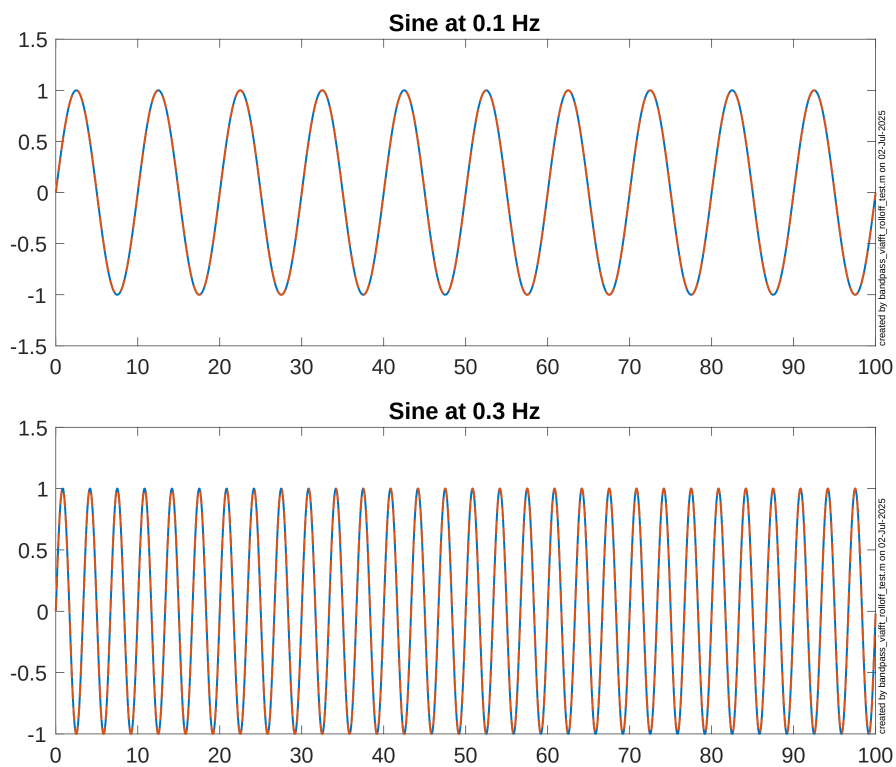

Figure 4: 100 mHz and 300 mHz sine waves are unchanged by the filter.

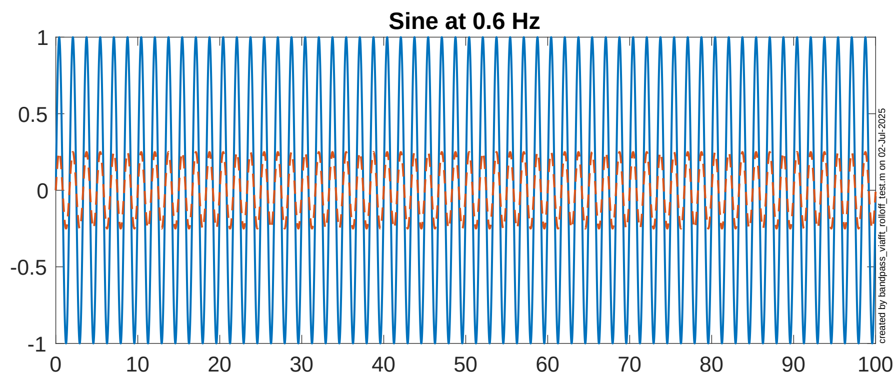

Figure 5: 600 mHz sine wave is attenuated by 0.25.

2.2 Examples of the Non-causal Effects

Because the filter is non-causal, you can often see some odd looking impacts. For example, for the impulse shown in figure 6, the output of the filter is clearly changing more that 10 seconds before the impulse arrives at the filter input.

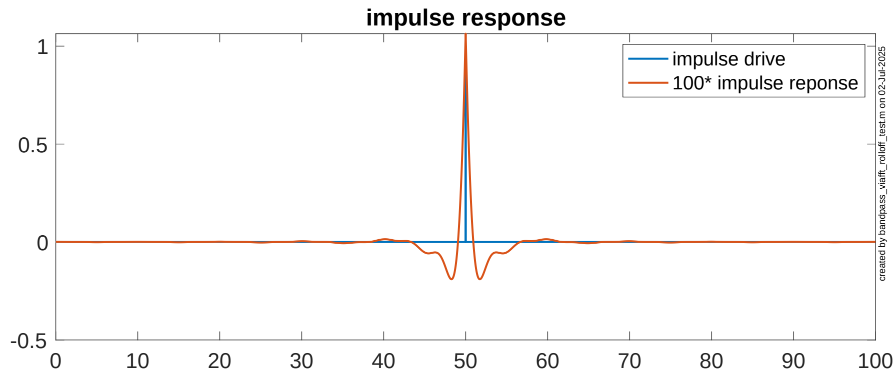

Figure 6: The impulse response of the filter clearly shows the non-causal behavior.

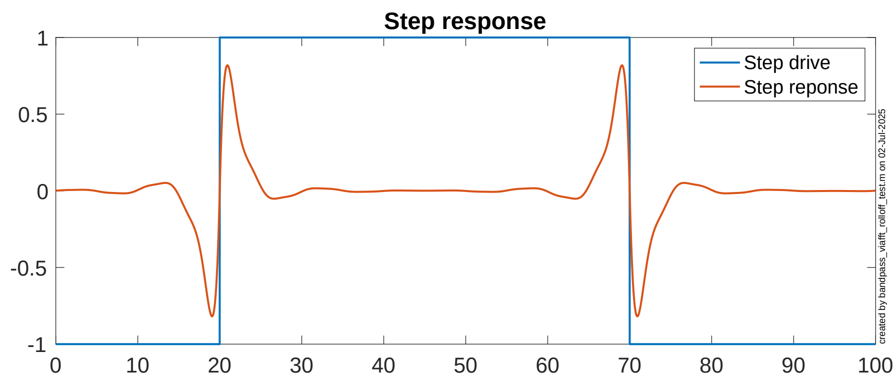

Figure 7: The step response of the filter also shows the non-causal behavior.

If you have a sine wave which is not an integral number of cycles, this is really a signal with a discontinuity. This means the time series will have odd effects at the beginning and the end, where the discontuity lives. This can be seen in figure 8.

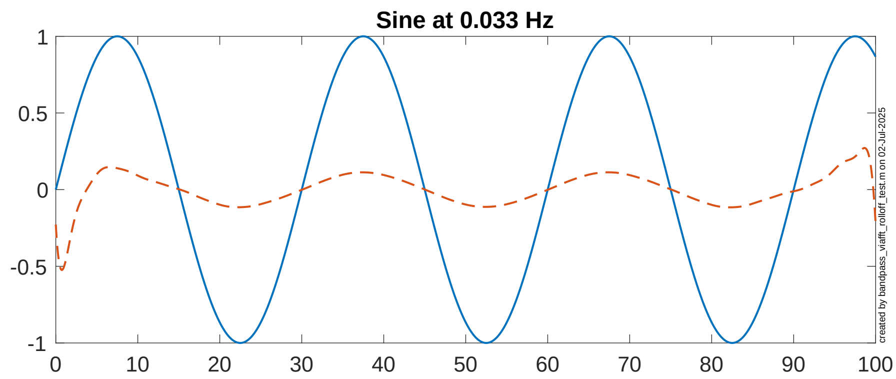

Figure 8: End effects are clear if the sine doesn't have an integral number of periods, i.e. it doesn't repeat exactly.

I've also plotted a few square waves so we can see what they look like.

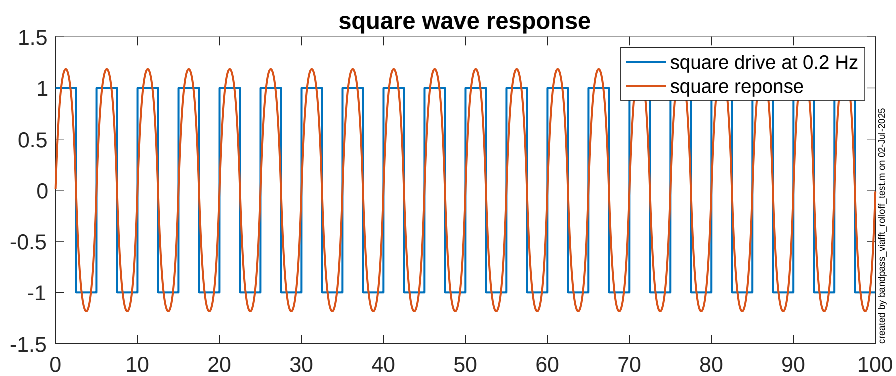

Figure 9: A square wave in the center of the pass-band looks nice.

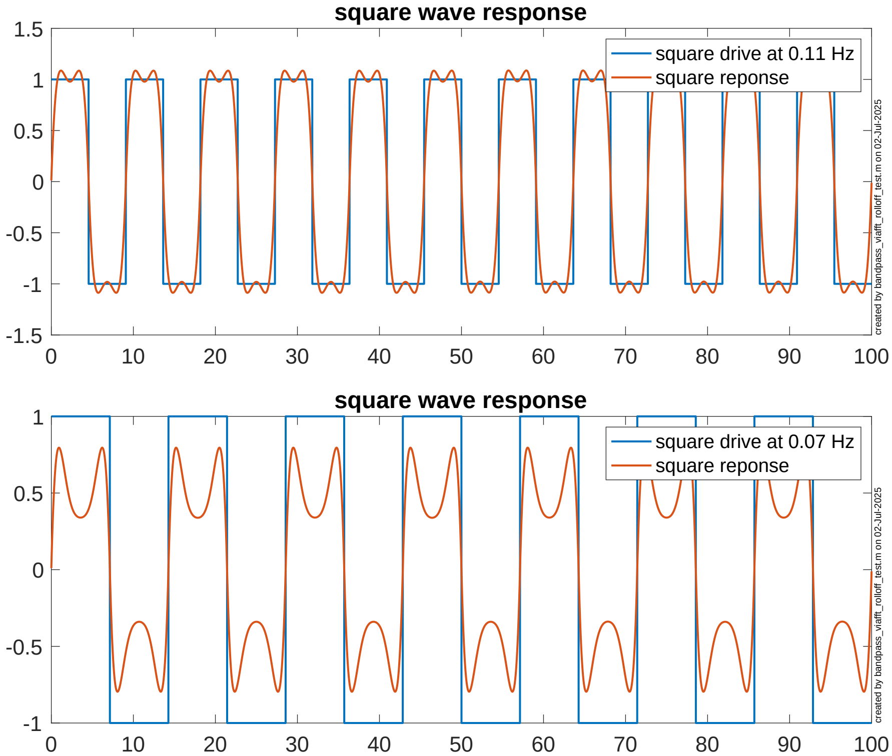

Figure 10: Square waves outside the pass-band look odd

3 Matlab Code

Here is a listing of the Matlab code. Hopefully it is well enough commented that it makes sense. The idea is to take an FFT of a time series, multiply the FFT by the filter, and take the inverse FFT of the result. "Multiplication in the frequency domain is the same as convolution in the time domain." The only complexity is that you must remember to filter the frequency components above the Nyquist frequency. (This is exactly equivalent to saying that you must filter the negative frequency components, it's just that matlab starts the frequency for the FFT at 0 instead of -nyquist). These are the conjugates, in the reverse order, as the low-frequency FFT terms. You also have to be sure you do the right thing with the element at the Nyquist frequency. If the time series has an even number of elements, then there is 1 DC component and 1 Nyquist frequency element, and everything else has a conjugate (alternatively, the DC and Nyquist are conjugates). However, if there are an odd number of elements, then there is still 1 DC element, but there are 2 conjugate Nyquist frequency elements. This means that there is a fence-post issue with the high-frequency filter which you need to count correctly. Summary - for even-number sets, you do not repeat the highest frequency filter element, but for odd-number sets you do repeat the highest frequency element.

Here is the code

```
function filtered_timeseries = bandpass_viafft_fn_rolloff(input_timeseries, sample_time
    , lowfreq_request, highfreq_request, slope, quiet)
%bandpass_viafft_fn_rolloff  non-causal bandpass, attenuates reject band as f^n or f^-n
%  filtered_timeseries = bandpass_viafft(input_timeseries, sample_time, lowfreq,
    highfreq, N)
%
% input_timeseries is the original timeseries.
% sample_time   time between samples in original timeseries, e.g. 1/2048
```

```
% lowfreq       in Hz, lowest freq to keep.
% highfreq      in Hz, highest freq to keep
% N            roll-off of attenuation band, eg 2 to rolloff as f^2
%
% the data will be bandpass filtered by an ideal flattop bandpass window.
% we take an fft, attenuate the other freq's, and take the inverse fft.
% there is no windowing or detrending, so the first and last few points
% will likely be weird end effects if data set is
% not periodic (which is usually true)
%         f-low    f-high
%  roll up  +-------+
%   as     /         \   attenuate as
%   f^N   /           \    f^-N
%        /             \
% if you pick frequencies which are not identical to point is the freq
% band of the fft, the code will pick the closest freq, and tell you what
% it picked.
%
% This does not downsample, so the original and filtered time series have
% the same number of points
%
% for example:
%  bandpass_viafft_fn_rolloff = bandpass_viafft(input_timeseries, Ts, 0.1, 0.5, 2);
%   will make a bandpass between 0.1 and 0.5, rolling off like f^2 outside the passband
%
%  filtered_timeseries = bandpass_viafft(input_timeseries, sample_time, lowfreq, highfreq, N, quiet)
%     optional input 'quiet'
% if you specify 'q' as the last input, it will suppress the commentary.
% useful for big loops.
% BTL, June 2025, adapted from bandpass_viafft

if nargin >5
    if strncmpi('q',quiet,1)
       show_msg = false;
    else
        warning('the quiet input should be either ''q'' or omitted')
        show_msg = false;
    end
else
    show_msg = true;
end


n_points_orig = length(input_timeseries);

data_size     = size(input_timeseries);
if data_size(1) == 1  % it is a row vector
    input_timeseries = input_timeseries';
    input_was_row = true;
else
    input_was_row = false;
end

duration      = sample_time * n_points_orig;  % number of sec of  data
dF            = 1/duration;
nyquist_freq  = 1/2 * (1/sample_time); % nyquist freq of original data


inputdata_fft  = fft(input_timeseries);
fft_points = length(inputdata_fft);
```

```
freq = dF * (0: fft_points-1)';

%  do the calc for the low freq end of the bandpass filter

if lowfreq_request < 0
    lowfreq_request = 0;
    disp('The low frequency edge must not be less than 0, resetting to 0')
end

lowfreq_exclude_index = round(lowfreq_request/dF) + 1 - 1;
% last freq index to exclude. zero freq is point 1

lowfreq       = dF * round(lowfreq_request/dF);  % first freq to keep

if lowfreq ~= lowfreq_request
    if show_msg
        disp(['Resetting low freq edge of filter to ',num2str(lowfreq),...
            ' Hz (was ',num2str(lowfreq_request),' Hz)']);
    end
end


%  do the calc for the high freq end of the bandpass filter
if highfreq_request >= nyquist_freq
    highfreq_exclude_band = [];
    if show_msg
        disp(['The high frequency edge is at or above the nyquist freq (',...
            num2str(nyquist_freq),' Hz)'])
        disp('so there will be NO HIGH FREQ FILTERING')
    end
else
    highfreq_index1 = round(highfreq_request/dF) + 1 + 1 ; % where to start exclusion band
    % is this even or odd? -
    %    do we repeat the last point (odd)
    %    or not (last freq is unique, even number of points)
    if (fft_points/2) == round(fft_points/2)
        highfreq_index2 = fft_points/2 + 1;  % where to end exclusion band
        evennumber = true; % the highest freq is NOT repeated
    else
        highfreq_index2 = floor(fft_points/2) + 1;  % where to end exclusion band
        evennumber = false; % the highest freq IS repeated.
    end
    highfreq_exclude_band = (highfreq_index1:highfreq_index2);
    highfreq       = dF * round(highfreq_request/dF);

    if highfreq ~= highfreq_request
        if show_msg
           disp(['Resetting high freq edge of filter to ',num2str(highfreq),...
               ' Hz (was ',num2str(highfreq_request),' Hz)']);
        end
    end
end
%

fft_filter = ones(size(inputdata_fft));


% filtered_fft   = inputdata_fft;  % start with all the data
% now set the data outside the filter band to 0,
```

```
% This is two bands of frequency, but
% this is a 2 sided fft, with 0 Hz as the first element, so there are
% 4 bands to set to 0. bands 2 and 3 link up at the nyquist freq,
% so there are really only 3 distinct bands.


% simple example of short vector
% point#   1   2   3    4   5       6         7    8    9   10
% freq:    0  dF  2dF  3dF 4dF 5dF=nyquist  4dF  3dF  2dF  dF
% exclude  B1 B1   .    .   B3        B3       B3  .      .   B2

% first low freq end  (band 1)
if lowfreq_exclude_index > 0
    fft_filter(1:lowfreq_exclude_index) = (freq(1:lowfreq_exclude_index)./lowfreq).^
        slope;
end


fft_filter(highfreq_exclude_band) = (highfreq ./ freq(highfreq_exclude_band)).^slope;

if evennumber
    % don't repeat the last index
    fft_filter(highfreq_index2+1: fft_points) = fft_filter([highfreq_index2 - 1: -1:
        2]);
else
    % do repeat the high freq point
    fft_filter(highfreq_index2+1: fft_points) = fft_filter([highfreq_index2:     -1:
        2]);
end

if show_msg
    figure; loglog(freq, fft_filter)
end

filtered_data = fft_filter .* inputdata_fft;
filtered_timeseries = ifft(filtered_data);

if input_was_row == true
    filtered_timeseries = filtered_timeseries';
end

end
```