Technical reference to core functions for YANGstraight_source

Hideki Kawahara

12:06am, January 21, 2019
(Start date: January 13, 2019)

Contents

List of Figures

List of Tables

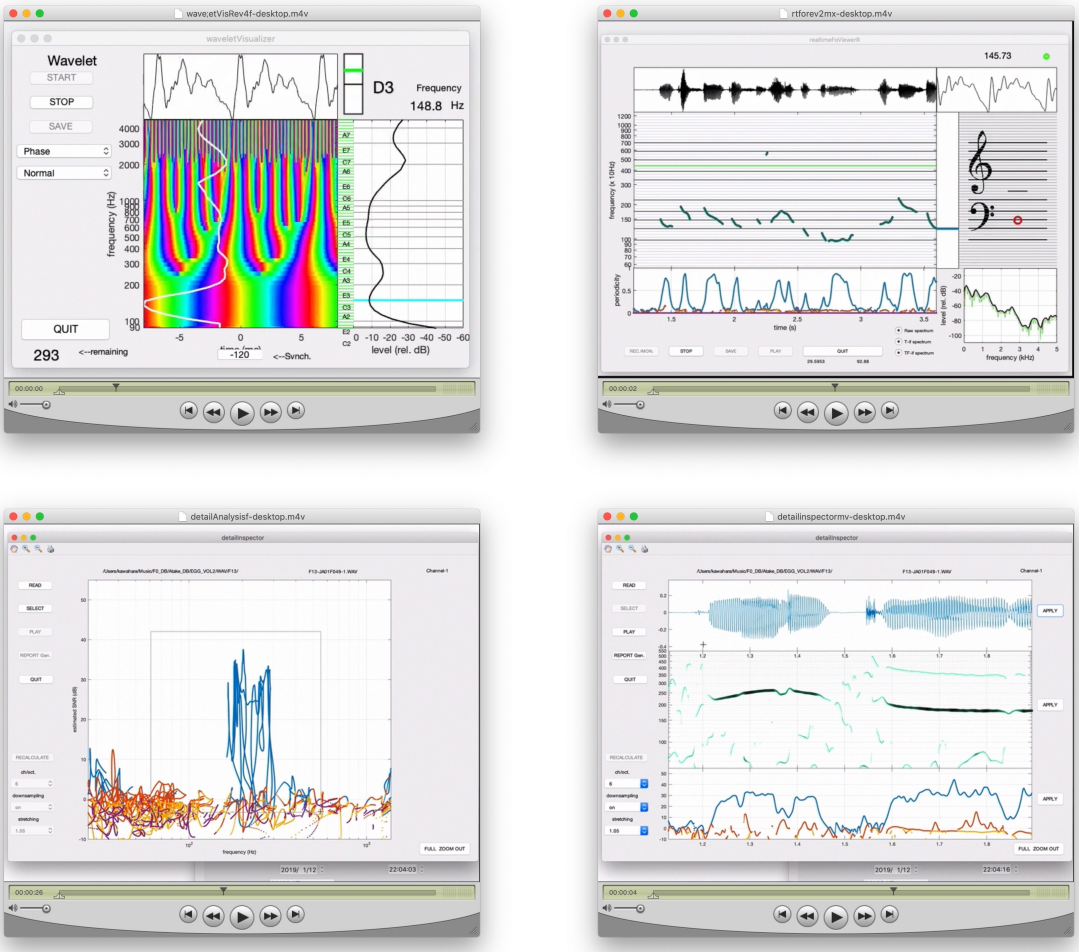

Figure 1: Interactive GUI tools based on these core functions. These images are snapshots of demonstration movies.

1 Introduction

This technical memo introduces details of core functions working behind interactive GUI tools. They are implemented using MATLAB and uses signal processing toolbox only. These core functions are functional with GNU Octave 4.4.0 on Mac OS in the default setting. These tools and core functions are in a GitHub repository below:
`https://github.com/HidekiKawahara/YANGstraight_source`

2 designAnalyticWavelet

Source attribute analysis using analytic signal wavelet

2.1 Syntax

```
output = designAnalyticWavelet(fs, fl, fh)
output = designAnalyticWavelet(fs, fl, fh, channels_oct)
output = designAnalyticWavelet(fs, fl, fh, channels_oct, mag)
output = designAnalyticWavelet(fs, fl, fh, channels_oct, mag, wintype)
```

2.2 Description

This function generates a set of filters which use analytic signals for their impulse response. The envelope of an analytic signal is one of following windowing functions; such as a six-term cosine series, Hann, Hamming, Blackman[1], Nuttall[2], Kaiser[3], and DPSS (Discrete Prolate Spheroidal Series) which is the discretized version of PSWF (Prolate Spheroidal Wave Function)[4]. The design parameters are the sampling frequency, the lowest frequency and the highest frequency of the complex exponent, filter frequency allocation density, and the ratio of the envelope and the carrier frequency, and the type of envelope functions.

$$w(t, f_c; c_{\text{mag}}, X) = w_e(t, f_c; c_{\text{mag}}, X) \exp(j2\pi f_c t), \tag{1}$$

where $j = \sqrt{-1}$ represents imaginary unit and $w_e(t, f_c; c_{\text{mag}}, X)$ represents an envelope function listed above. The symbol X represents the name of the envelope function, f_c represents the carrier frequency, and c_{mag} represents the stretching factor from the default setting. The default settings are as follows:

six-term cosine series The domain of definition is $[-3c_{\text{mag}}/f_c, 3c_{\text{mag}}/f_c]$.

$$w_e(t, f_c; c_{\text{mag}}, \text{six-term}) = \sum_{k=0}^{5} a_k \cos\left(\frac{\pi k t}{3 c_{\text{mag}} f_c}\right), \tag{2}$$

where

$$\begin{aligned}\{a_k\}_{k=0}^{5} = \{&0.2624710164, 0.4265335164, 0.2250165621,\\ &0.0726831633, 0.0125124215, 0.0007833203\}\end{aligned} \tag{3}$$

Hann windowing function The domain of definition is $[-c_{\text{mag}}/f_c, c_{\text{mag}}/f_c]$.

$$w_e(t, f_c; c_{\text{mag}}, \text{Hann}) = 0.5 + 0.5 \cos\left(\frac{\pi t}{c_{\text{mag}} f_c}\right), \tag{4}$$

Hamming windowing function The domain of definition is $[-c_{\mathrm{mag}}/f_c, c_{\mathrm{mag}}/f_c]$.

$$w_e(t, f_c; c_{\mathrm{mag}}, \mathrm{Hann}) = 0.53836 + 0.46164 \cos\left(\frac{\pi t}{c_{\mathrm{mag}} f_c}\right), \tag{5}$$

Blackman windowing function The domain of definition is $[-3c_{\mathrm{mag}}/2f_c, 3c_{\mathrm{mag}}/2f_c]$.

$$w_e(t, f_c; c_{\mathrm{mag}}, \mathrm{Blackman}) = 0.42 + 0.5 \cos\left(\frac{2\pi t}{3c_{\mathrm{mag}} f_c}\right) + 0.08 \cos\left(\frac{4\pi t}{3c_{\mathrm{mag}} f_c}\right), \tag{6}$$

Nuttall-12 windowing function The domain of definition is $[-2c_{\mathrm{mag}}/f_c, 2c_{\mathrm{mag}}/f_c]$.

$$w_e(t, f_c; c_{\mathrm{mag}}, \text{Nuttall-12}) = \sum_{k=0}^{3} a_k \cos\left(\frac{\pi k t}{2c_{\mathrm{mag}} f_c}\right), \tag{7}$$

where

$$\{a_k\}_{k=0}^{3} = \{0.355768, 0.487396, 0.144232, 0.012604\} \tag{8}$$

Note that this is the 12th item in Table II of the reference[2] and is not the version which is builtin MATLAB.

Kaiser windowing function The domain of definition is $[-2c_{\mathrm{mag}}/f_c, 2c_{\mathrm{mag}}/f_c]$. In this implementation, we used the built-in MATLAB function `kaiser` and adjusted its parameter to have the same side-lobe level with the six-term cosine series. Kaiser windowing function[3] is an approximation to the theoretically most concentrated function called prolate spheroidal wave function (PSWF)[4].

Discrete Prolate Spheroidal Series (DPSS) The domain of definition is $[-2c_{\mathrm{mag}}/f_c, 2c_{\mathrm{mag}}/f_c]$. In this implementation, we used the built-in MATLAB function `dpss` and adjusted its parameter to have the same side-lobe level with the six-term cosine series. DPSS is the discrete version of the theoretically most concentrated function called prolate spheroidal wave function (PSWF)[4].

2.3 Examples

2.3.1 Default setting

Default setting requires the sampling frequency and the lower- and the upper-frequency limits. For example:

```
fs = 44100;
fl = 55;
fh = 1200;
output = designAnalyticWavelet(fs, fl, fh);
```

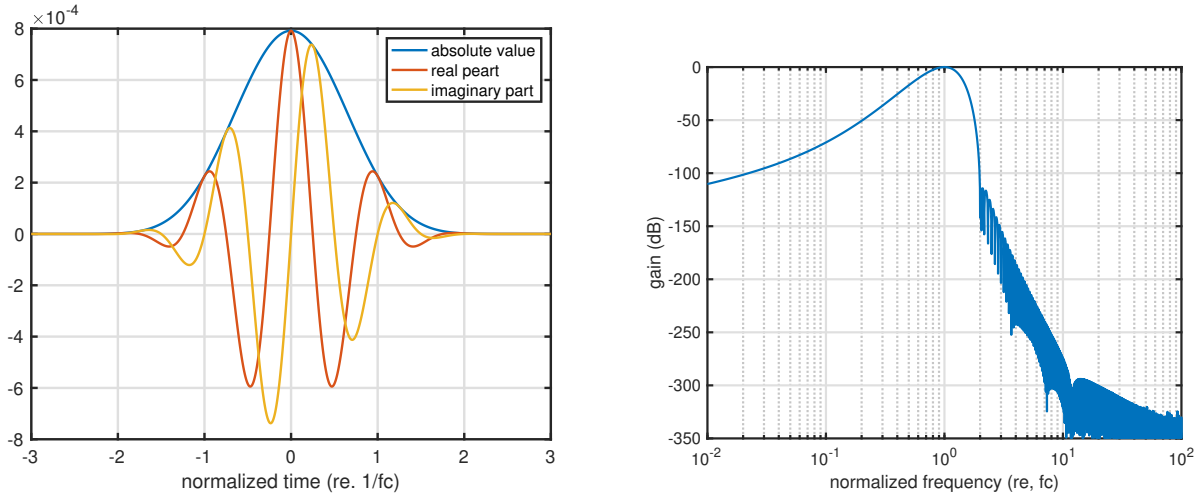

Figure 2: The left panel shows the shape of the impulse response normalized by the carrier period. The right panel shows the gain of the filter. The frequency axis uses the frequency normalized by its center frequency.

```
figure;
plot(output.wvlt(1).t_axis * fl, abs(output.wvlt(1).w), 'linewidth', 2);
hold all
plot(output.wvlt(1).t_axis * fl, real(output.wvlt(1).w), 'linewidth', 2);
plot(output.wvlt(1).t_axis * fl, imag(output.wvlt(1).w), 'linewidth', 2);
grid on;
set(gca, 'fontsize', 15, 'linewidth', 2);
xlabel('normalized time (re. 1/fc)');
legend('absolute value', 'real peart', 'imaginary part');
print -depsc defaultShapeExample.eps
```

This set of statements generates filter impulse responses from 55 Hz to 1,200 Hz with 12 filters in each octave density. The default envelope function is the six-term cosine series introduced in the reference[5]. The filters assume the sampling frequency of the input signal is 44,100 Hz.

Figure 2 shows the output of the default setting. An impulse response of one of the output is a complex numbered signal. The left panel of the plot shows absolute value, real part, and the imaginary part of one example. The absolute value is the six-term cosine series proposed by the author.

The following instructions draw the frequency response shown in the right panel of Fig. 2.

```
figure
fftl = 2^20;
fx = (0:fftl - 1) / fftl * fs;
```

```
semilogx(fx / fl, 20 * log10(abs(fft(output.wvlt(1).w, fftl))), 'linewidth', 2);
grid on;
axis([0.01 100 -350 0]);
grid on;
set(gca, 'fontsize', 15, 'linewidth', 2);
xlabel('normalized frequency (re, fc)');
ylabel('gain (dB)');
print -depsc defaultGainExample.eps
```

2.3.2 Frequency allocation

The default setting generates 12 filters in each octave. It is too dense to visualize. The following instructions yield s more visually relevant example.

```
fs = 44100;
fl = 55;
fh = 1200;
channels_oct = 3;
output = designAnalyticWavelet(fs, fl, fh, channels_oct);
fftl = 2^16;
fx = (0:fftl - 1) / fftl * fs;
figure;
for ii = 1:length(output.fc_list)
    semilogx(fx, 20 * log10(abs(fft(output.wvlt(ii).w, fftl))), 'linewidth', 2);
    hold all
end
grid on
axis([20 3000 -150 0]);
set(gca, 'fontsize', 15, 'linewidth', 2);
xlabel('frequency (Hz)');
ylabel('gain (dB)');
print -depsc filterAllocationExample.eps
```

Figure 3 shows filter allocation example for three channels per octave setting. Note that they are highly overlapped even with this sparse setting.

By changing the axis setting, the magnified view provides details showing how they overlap.

```
axis([20 3000 -3 0]);
print -depsc filterAllocationMagExample.eps
```

It is shown in the right panel of Fig. 3.

2.3.3 Stretching factor

This example shows the effect of the stretching factor. The following test script shows how it works.

```
fs = 44100;
fl = 55;
```

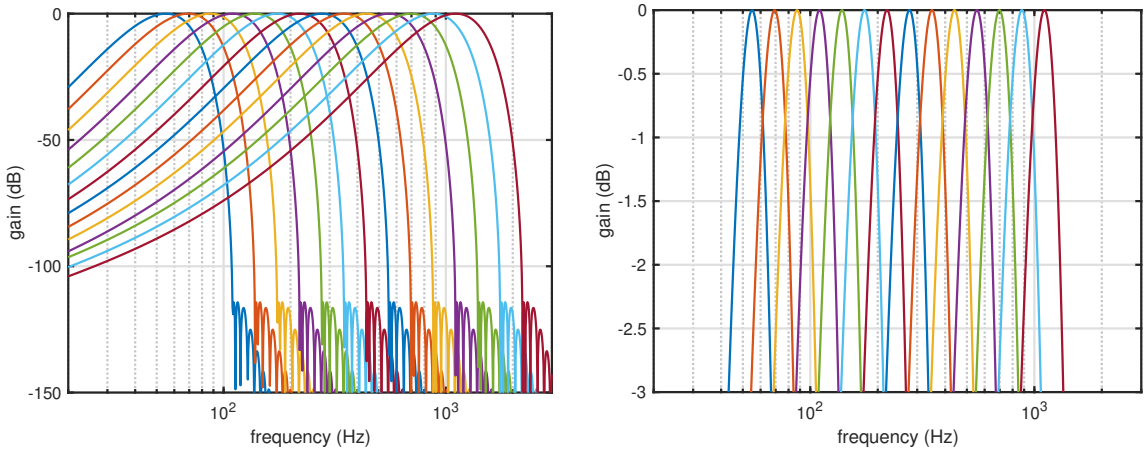

Figure 3: Filter allocation example. The density is 3 channels/oct.

```
fh = 1200;
channels_oct = 3;
mag_list = [1 1.05 1.25 1.5 2 3];
fftl = 2^17;
fx = (0:fftl - 1) / fftl * fs;
gain_figure = figure;
shape_figure = figure;
for ii = 1:length(mag_list)
    mag = mag_list(ii);
    output = designAnalyticWavelet(fs, fl, fh, channels_oct, mag);
    figure(shape_figure);
    plot(output.wvlt(1).t_axis * fl, abs(output.wvlt(1).w), 'linewidth', 2);
    grid on;
    hold all
    figure(gain_figure);
    semilogx(fx / fl, 20 * log10(abs(fft(output.wvlt(1).w, fftl))), 'linewidth', 2);
    hold all
    grid on;
end
figure(shape_figure);
set(gca, 'fontsize', 15, 'linewidth', 2);
xlabel('normalized time (re. 1/fc)');
legend(num2str(mag_list'));
print -depsc filteShapeStretchExample.eps
figure(gain_figure);
set(gca, 'fontsize', 15, 'linewidth', 2);
xlabel('normalized frequency (re. fc)');
ylabel('gain (dB)');
legend(num2str(mag_list'));
axis([0.01 100 -350 0])
```

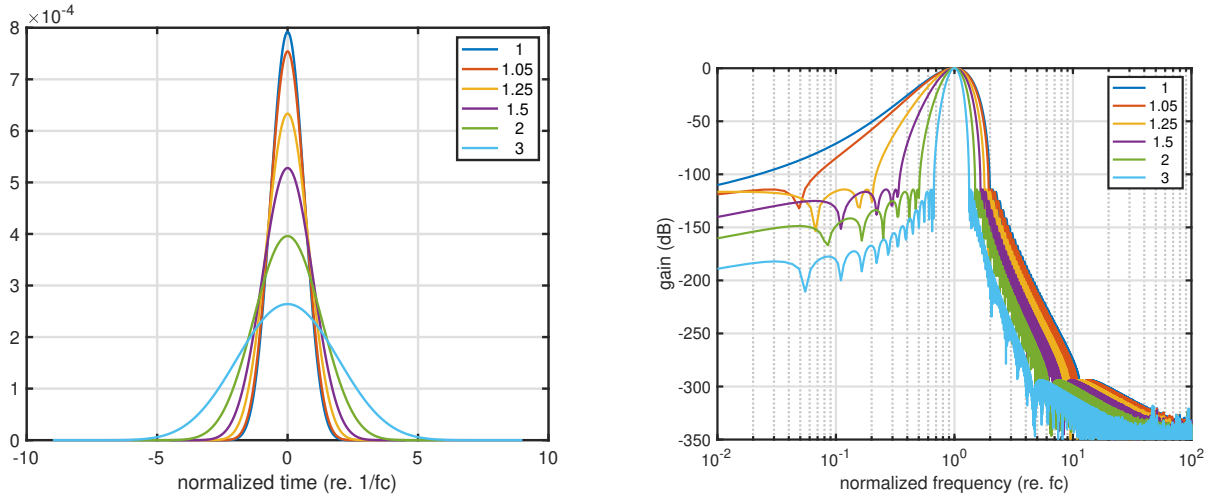

Figure 4: Filter shape and gain differences by changing the stretching factor.

```
print -depsc filteGainStretchExample.eps
```

Figure 4 shows the shape and gain for different stretching factor. The envelope is designed to have 0 dB gain at its peak. This condition makes the shape for smaller stretching factor higher.

2.3.4 Envelope functions

The following script generates filters using different envelope functions.

```
fs = 44100;
fl = 55;
fh = 1200;
channels_oct = 3;
mag = 1.0;
wintype_list = {'sixterm', 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
fftl = 2^17;
fx = (0:fftl - 1) / fftl * fs;
gain_figure = figure;
shape_figure = figure;
for ii = 1:length(wintype_list)
    output = designAnalyticWavelet(fs, fl, fh, channels_oct, mag, wintype_list{ii});
    figure(shape_figure)
    plot(output.wvlt(1).t_axis * fl, abs(output.wvlt(1).w), 'linewidth', 2);
    grid on;
    hold all
    drawnow
    figure(gain_figure)
```

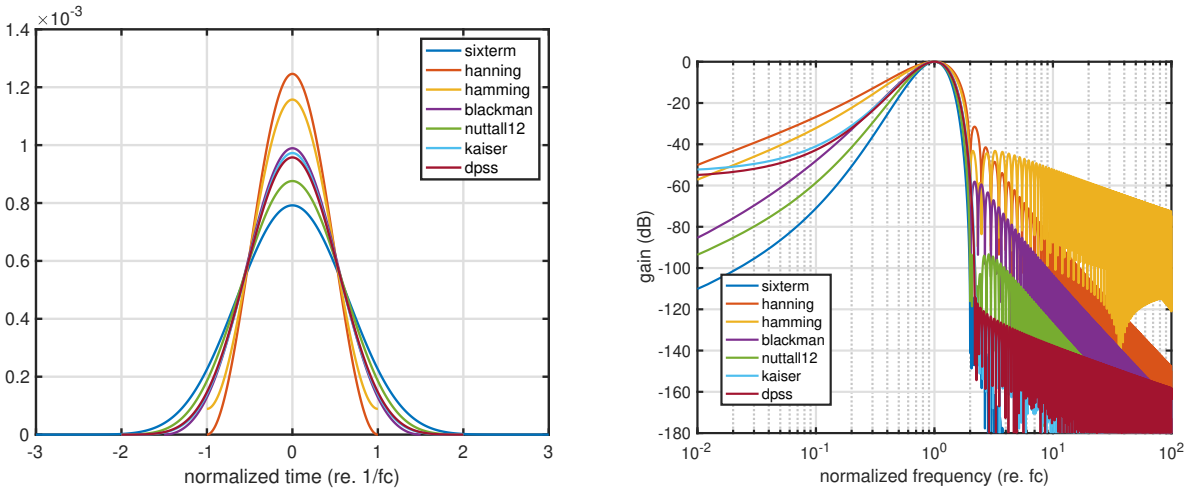

Figure 5: Filter shape and gain differences by changing the stretching factor.

```
    semilogx(fx / fl, 20 * log10(abs(fft(output.wvlt(1).w, fftl))), 'linewidth', 2);
    hold all
    grid on;
end
figure(shape_figure);
set(gca, 'fontsize', 15, 'linewidth', 2);
xlabel('normalized time (re. 1/fc)');
legend(wintype_list{:});
drawnow
print -depsc filteShapeEnvExample.eps

figure(gain_figure);
set(gca, 'fontsize', 15, 'linewidth', 2);
xlabel('normalized frequency (re. fc)');
ylabel('gain (dB)');
legend(wintype_list{:}, 'location', 'best');
axis([0.01 100 -180 0])
print -depsc filteGainEnvExample.eps
```

Figure 5 shows the shape and gain for different envelope functions. Note that Kaiser and DPSS design parameters are adjusted to have the same maximum side-lobe level with the six-term cosine series.

The following script calculates the normalized duration of each envelope function.

```
fs = 44100;
fl = 55;
fh = 1200;
```

Table 1: Duration of envelope functions

function	duration	length factor	reference & note
six-term	0.4444	3	[5]
Hann	0.2831	1	[1]
Hamming	0.3050	1	[1]
Blackman	0.3566	1.5	[1]
Nuttall-12	0.4020	2	[2] item-12 of Table
Kaiser	0.3619	2	[3] $\beta = 15.03$
DPSS	0.3676	2	[4] $NW = 4.72$

```
channels_oct = 3;
mag = 1.0;
wintype_list = {'sixterm', 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
duration_result = zeros(length(wintype_list), 1);
for ii = 1:length(wintype_list)
    output = designAnalyticWavelet(fs, fl, fh, channels_oct, mag, wintype_list{ii});
    duration_result(ii) = sqrt(sum(abs(output.wvlt(1).w) .^ 2 .* output.wvlt(1).t_axis .^ 2) ...
        / sum(abs(output.wvlt(1).w) .^ 2)) * fl;
end
```

Table 1 shows the duration of envelope functions.

2.3.5 Envelope functions with normalized duration

By using the duration results, the following script shows normalized-uration versions of envelope shapes and gain functions.

```
fs = 44100;
fl = 55;
fh = 1200;
channels_oct = 3;
mag_base = 1.0;
wintype_list = {'sixterm', 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
fftl = 2^17;
fx = (0:fftl - 1) / fftl * fs;
gain_figure = figure;
shape_figure = figure;
for ii = 1:length(wintype_list)
    mag = mag_base / duration_result(ii) * duration_result(1);
    output = designAnalyticWavelet(fs, fl, fh, channels_oct, mag, wintype_list{ii});
    figure(shape_figure)
    plot(output.wvlt(1).t_axis * fl, abs(output.wvlt(1).w), 'linewidth', 2);
    grid on;
    hold all
```

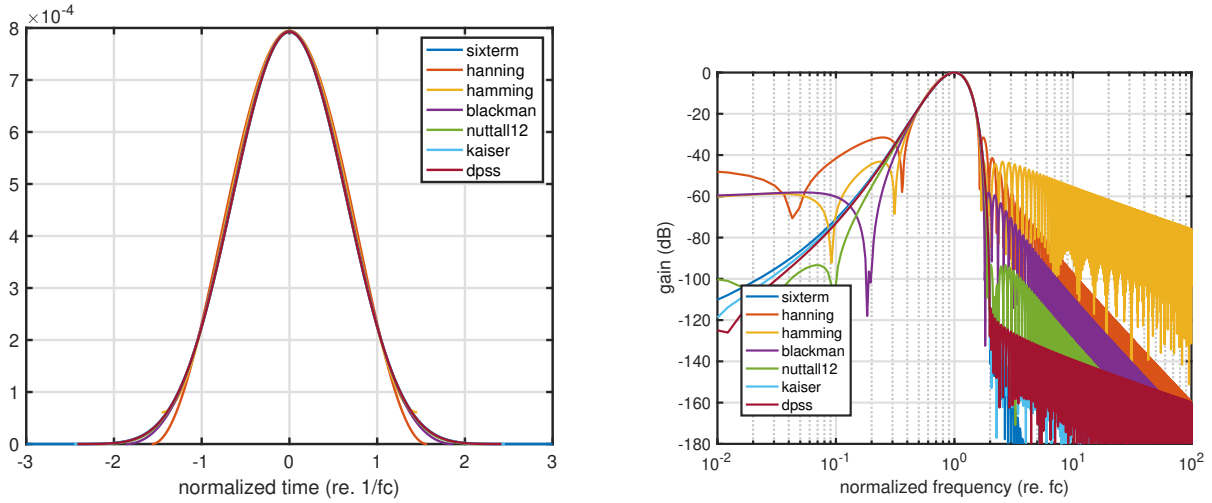

Figure 6: Filter shape and gain differences by changing the envelope function. This time, each envelope has the same duration to the six-term envelope.

```
    drawnow
    figure(gain_figure)
    semilogx(fx / fl, 20 * log10(abs(fft(output.wvlt(1).w, fftl))), 'linewidth', 2);
    hold all
    grid on;
end
figure(shape_figure);
set(gca, 'fontsize', 15, 'linewidth', 2);
xlabel('normalized time (re. 1/fc)');
legend(wintype_list{:});
drawnow
print -depsc filteShapeEnvNExample.eps

figure(gain_figure);
set(gca, 'fontsize', 15, 'linewidth', 2);
xlabel('normalized frequency (re. fc)');
ylabel('gain (dB)');
legend(wintype_list{:}, 'location', 'best');
axis([0.01 100 -180 0])
print -depsc filteGainEnvNExample.eps
```

Figure 6 shows the shape and gain for different envelope functions with duration normalization. Note that the shape around the time origin is close to each other. Similarly, the gain shape around the center frequency is also close to each other.

2.4 Input Arguments

fs The ampling frequency of the signal to be processed by the generated filters. Unit is Hz.

fl The lowest frequency of the generated filter carrier frequency f_c. The unit is Hz.

fh The upper frequency which limits filter generation. The unit is Hz.

channels_oct This parameter defines the density of filter allocation. The value of this parameter represents the number of filters in each octave. The default value is 12.

mag Temporal stretching factor c_{mag} of the envelope shape. The default value is 1.0.

wintype Name of envelope function. Predefined values are as follows; `sixterm, hann, hamming, blackman, nuttall, kaiser, dpss`. The default value is `sixterm`.

2.5 Output Arguments

The output argument is a structure variable, which has the following fields.

input_parameters Copy of input (and default) values of design related parameters. The field is a structure with the following fields. `sampling_frequency, lower_frequency, higher_frequency, stretching_factor, channels_per_octave, wintype`

fc_list A row vector consisting of filter center frequencies. Unit is Hz.

wvlt A array of structure variable. Each variable represents a filter and consists of the following fields.

- **w** A complex numbered impulse response, an analytic signal.
- **bias** Number of samples preceding the center sample of the impulse response. The length of the impulse response is `2 * bias + 1`.
- **t_axis** The time axis for the impulse response. Unit is s (second).

2.6 Extended Capabilities

NA, yet.

2.7 See Also

`sourceAttributesAnalysis`, `waveletAttributesAnalyzer`

3 waveletAttributesAnalyzer

Source attribute analysis using analytic signal wavelet

3.1 Syntax

```
output = waveletAttributesAnalyzer(x, fs, wvltStr)
```

3.2 Description

This function analyzes the input signal using a wavelet filter bank made from an analytic signal given in the argument. It calculates the filtered outputs and their instantaneous frequencies, group delays, and amplitudes at the audio sampling rate resolution. This function uses the following simple implementation, instead of using the Flanagan's equation[6] and its variants.

$$\omega_i[n;k] = \angle \left[\frac{y[n+1;k]}{y[n;k]} f_s \right], \tag{9}$$

$$\tau_g[n;k] = -\frac{1}{\Delta\omega} \angle \left[\frac{y[n;k+1]}{y[n;k]} \right], \tag{10}$$

$$P[n;k] = |y[n+1;k] y[n;k]|, \tag{11}$$

where $\omega_i[n;k]$ represents the instantaneous angular frequency of the k-th filter output at discrete time n. The symbol $\tau_g[n;k]$ represents the group delay, and $P[n;k]$ represents (roughly speaking) squared signal. Note that they are not symmetric in time (for frequency and squared ones) and the filter channel (group delay). Note also that these implementations were not practical several decades ago. However, thanks to the advancement of the technology, current CPUs are efficient for calculating these functions.

3.3 Examples

3.3.1 Default analysis of 100 Hz pulse train

The following script produces a 100 Hz pulse train with 0.71 s in duration. Then, generates a wavelet filter bank using the default envelope function. The last line executes analysis using the generated wavelet.

```
fs = 44100;
fo = 100;
fl = 50;
fh = 1200;
duration = 0.071;
```

```
tt = (1:round(duration * fs))' / fs;
x = tt * 0;
nto = round(fs / fo);
x(1:nto:end) = 1;
wvltStr = designAnalyticWavelet(fs, fl, fh);
output = waveletAttributesAnalyzer(x, fs, wvltStr);
```

The following script is a preparation to make the vertical axis of the visualization of the results easy to understand.

```
total_channel = size(output.rawWavelet, 2);
ytickfreq = 100:100:1200;
ytickLoc = interp1(wvltStr.fc_list, 1:total_channel, ytickfreq, 'linear', 'extrap');
```

The following four scripts visualizes phase, instantaneous frequency, group delay, and power of the wavelet analysis results.

Phase

```
figure;imagesc(tt([1 end]),[1 total_channel],angle(output.rawWavelet'));
axis('xy');colormap(hsv)
set(gca, 'fontsize', 13, 'ytick', ytickLoc, 'yticklabel', num2str(ytickfreq'));
xlabel('time (s)')
ylabel('frequency (Hz)');
print -depsc sixt100HzPhaseMap.eps
```

Instantaneous frequency

```
figure; imagesc(log(max(fl, min(fh, output.inst_freq_map'))));
axis('xy'); colormap(jet)
set(gca, 'fontsize', 13, 'ytick', ytickLoc, 'yticklabel', num2str(ytickfreq'));
xlabel('time (s)')
ylabel('frequency (Hz)');
print -depsc sixt100HzInstFreqMap.eps
```

Group delay

```
figure; imagesc(max(-0.005, min(0.005,output.group_delay_map')));
axis('xy'); colormap(jet)
set(gca, 'fontsize', 13, 'ytick', ytickLoc, 'yticklabel', num2str(ytickfreq'));
xlabel('time (s)')
ylabel('frequency (Hz)');
print -depsc sixt100HzGroupDelayMap.eps
```

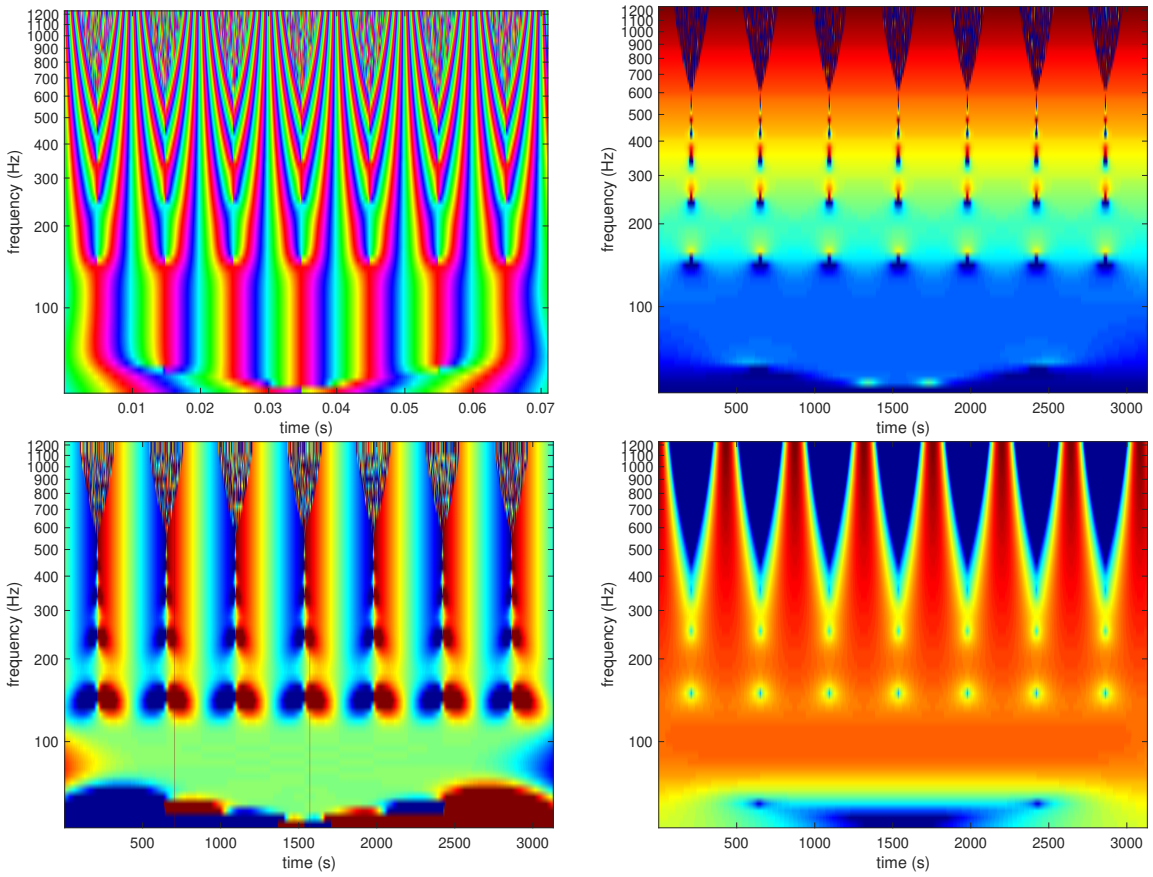

Figure 7: Visualization of the wavelet analysis results. (Top left) Phase, (Top right) Instantaneous frequency, (Bottom left) Group delay, and (Bottom right) Power in dB.

Power in dB

```
dB_map = 10 * log10(output.amp_squared_map');
figure; imagesc(max(max(dB_map(:))-80, dB_map));
axis('xy'); colormap(jet)
set(gca, 'fontsize', 13, 'ytick', ytickLoc, 'yticklabel', num2str(ytickfreq'));
xlabel('time (s)')
ylabel('frequency (Hz)');
print -depsc sixt100HzdBPwrMap.eps
```

Figure 7 shows the analysis results. Note that around the center, the instantaneous frequency, group delay, and dB power are temporally constant. This temporal constancy is a clue that the channel consisting of a dominant sinusoidal component.

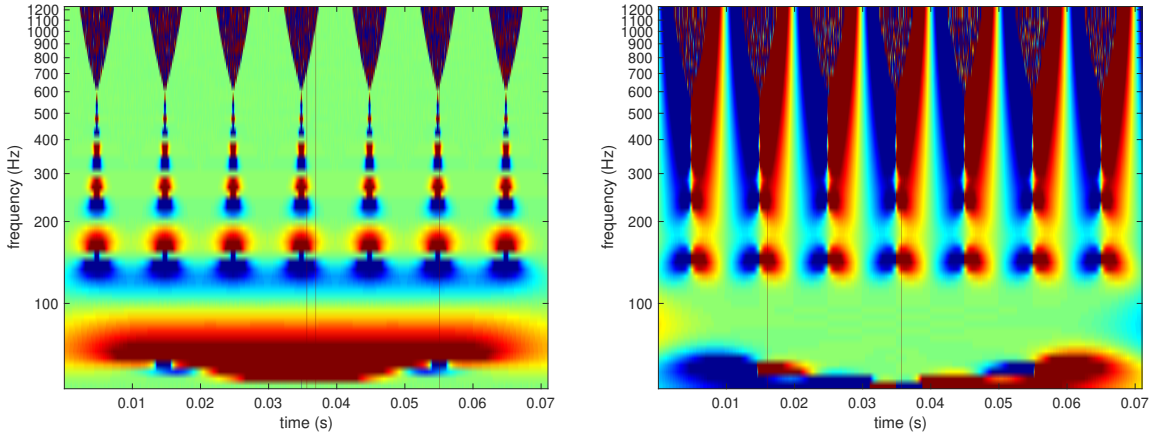

Figure 8: Visualization of the normalized instantaneous frequency map and the group delay map.

3.3.2 Visualization with normalization

Note that the real-time interactive tool `waveletVisualizer.m` uses these normalized versions of the instantaneous frequency and the group delay maps. For instantaneous frequency, what is interesting is the relation between the filter center frequency and its output's instantaneous frequency. For group delay, what is interesting is the relation between the period of the carrier of the filter and its group delay. The following visualizes these normalized deviations.

```
figure; imagesc(tt([1 end]),[1 total_channel], ...
    log(max(sqrt(0.5), min(sqrt(2), (output.inst_freq_map * diag(1 ./ wvltStr.fc_list))')))));
axis('xy'); colormap(jet)
set(gca, 'fontsize', 13, 'ytick', ytickLoc, 'yticklabel', num2str(ytickfreq'));
xlabel('time (s)')
ylabel('frequency (Hz)');
print -depsc sixt100HzInstFreqNrmMap.eps

%
figure; imagesc(tt([1 end]),[1 total_channel], ...
    max(-1, min(1, (output.group_delay_map * diag(wvltStr.fc_list))')));
axis('xy'); colormap(jet)
set(gca, 'fontsize', 13, 'ytick', ytickLoc, 'yticklabel', num2str(ytickfreq'));
xlabel('time (s)')
ylabel('frequency (Hz)');
print -depsc sixt100HzGroupDelayNrmMap.eps
```

Figure 8 shows normalized maps. The left image shows $\log(\omega_i(t, f_c)/f_c)$. The right image shows $\tau_g(t, f_c)f_c$. The values are truncated to fit inside $(\log(2^{-1/2}), \log(2^{1/2}))$ and $(-1, 1)$ for instantaneous frequency and group delay, respectively.

3.3.3 Fixed point in frequency and in time

The following script shows the cross-sectional behavior of instantaneous frequency and group delay. The instantaneous frequency has an interpretation as a power spectrum weighted average of neighboring frequencies. Similarly, the group delay has an interpretation as a (temporal) power weighted average of neighboring "time"[7]. This script uses the results of the previous subsection.

```
figure;
loglog(wvltStr.fc_list, max(0.1, output.inst_freq_map(1323:10:1764, :)));
hold on; loglog([fl fh], [fl fh], 'k');
grid on;
axis([fl fh fl fh]);
set(gca, 'fontsize', 13);
xlabel('frequency (Hz)');
ylabel('instantaneous frequency (Hz)');
title('sample 1323:10:1764');
print -depsc fixedPointInFrequency.eps

figure;plot(tt, tt+output.group_delay_map(:, 24:42));
hold on;plot(tt([1 end]), tt([1 end]), 'k')
grid on;
axis([tt(1) tt(end) tt(1) tt(end)]);
set(gca, 'fontsize', 13);
xlabel('time (s)')
ylabel('time + group delay (s)')
title('channel 24:42')
print -depsc fixedPointInTime.eps
```

Figure 9 shows cross-sectional plots of mappings from the center frequency of each filter and its output instantaneous frequency and from the time axis to the average time represented as the sum of time and the group delay. In the left panel, the region around 100 Hz has virtually constant instantaneous frequency and has an intersection the identity mapping at 100 Hz. It corresponds to the frequency of the fundamental component of the input signal

In the right panel, the regions around 0.1 to 0.7 s in 0.1 s steps have virtually constant average time crossing at the identity mapping at 0.1 to 0.7 s in 0.1 s steps. It corresponds to the time when pulses are located.

The following normalized plots are more relevant for frequency component and event detection. The following script generated the plots.

```
figure;loglog(wvltStr.fc_list, max(0.1, output.inst_freq_map(1323:10:1764, :) ...
    * diag(1 ./ wvltStr.fc_list)));
grid on;
axis([fl fh 1 / 2 2]);
set(gca, 'fontsize', 13);
xlabel('frequency (Hz)');
ylabel('frequency deviation (ratio)');
title('sample 1323:10:1764');
```

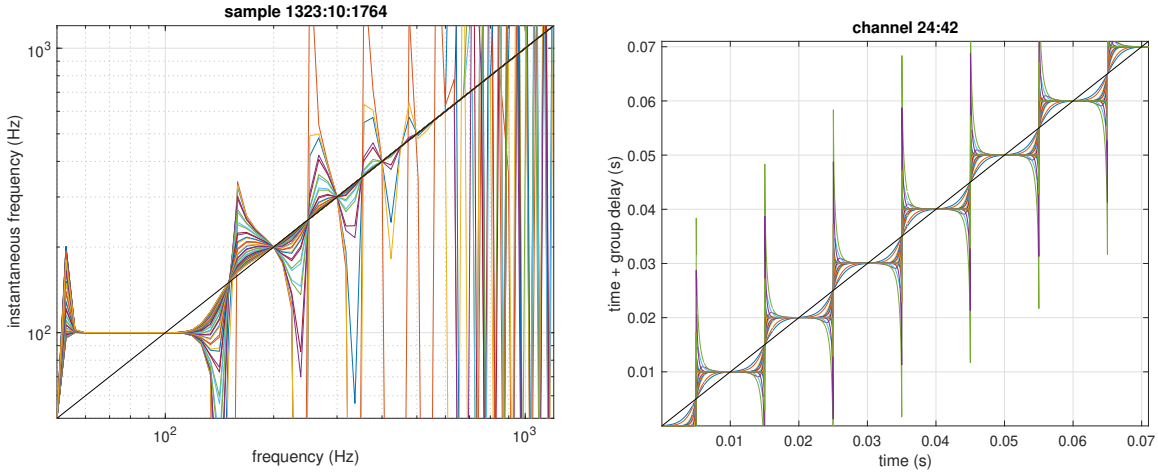

Figure 9: Fixed point analysis of the center frequency to the output instantaneous frequency mapping and the response center time to the average time mapping.

```
print -depsc fixedPointInFrequencyN.eps

figure;plot(tt, output.group_delay_map(:, 24:42) * diag(wvltStr.fc_list(24:42)));
grid on;
axis([tt(1) tt(end) -2 2]);
set(gca, 'fontsize', 13);
xlabel('time (s)')
ylabel('normalized group delay (re. 1/fc)')
title('channel 24:42')
print -depsc fixedPointInTimeN.eps
```

Figure 10 shows the corresponding normalized plots of Fig. 9. The following procedure yields fundamental component candidates and event candidates. Assume $u(\lambda)$ represents a normalized value defined on variable λ. Let U represents the set of candidates.

$$U = \left\{ \lambda \left| u(\lambda) = 0 \wedge \frac{du(\lambda)}{d\lambda} < 0 \right. \right\} \tag{12}$$

3.3.4 Effects of envelope shape

The following script shows differences caused by envelope shape difference. These results show that the six-term cosine series provides the best accuracy. The general behavior is consistent with the reference[8].

Instantaneous frequency

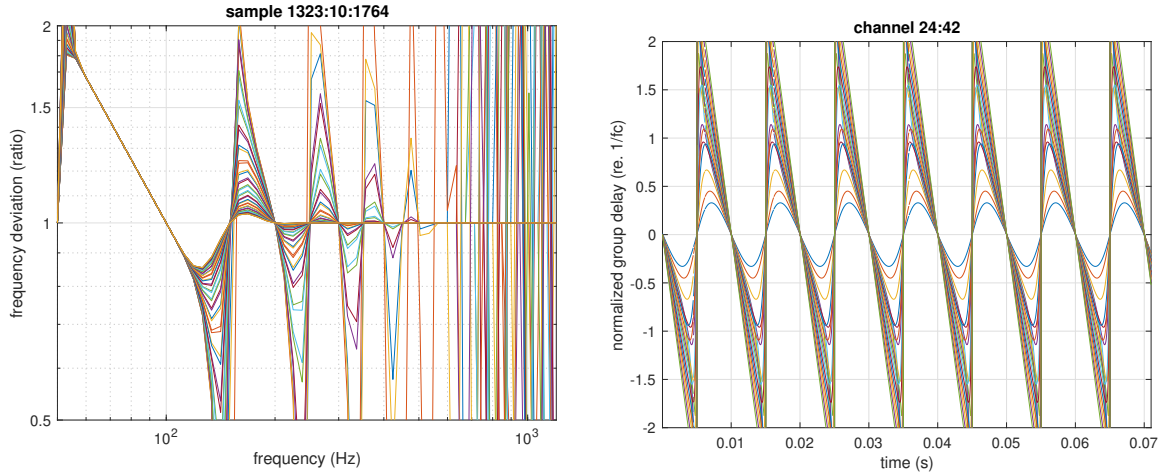

Figure 10: Fixed point analysis with normalized representations. Original mappings are the center frequency to the output instantaneous frequency mapping and the response center time to the average time mapping.

```
fs = 44100;
fo = 100;
duration = 0.071;
tt = (1:round(duration * fs))' / fs;
x = tt * 0;
nto = round(fs / fo);
x(1:nto:end) = 1;

fl = 50;
fh = 1200;
channels_oct = 12;
mag_base = 1.25;
wintype_list = {'sixterm', 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
mag_duration_list = [0.4444 0.2831 0.3050 0.3566 0.4020 0.3619 0.3676];
figure('position', [680   218   574   760]);
for ii = 1:length(wintype_list)
    mag = mag_base / mag_duration_list(ii) * mag_duration_list(ii);
    wvltStr = designAnalyticWavelet(fs, fl, fh, channels_oct, mag, wintype_list{ii});
    output = waveletAttributesAnalyzer(x, fs, wvltStr);
    subplot(7,1,ii);
    plot(tt(tt > 0.025 & tt < 0.045), output.inst_freq_map(tt > 0.025 & tt < 0.045, 12:14), ...
        'linewidth', 2);
    set(gca, 'fontsize', 13, 'linewidth', 2);
    tmpdata = output.inst_freq_map(tt > 0.025 & tt < 0.045, 12:14);
    maxy = max(abs(tmpdata(:) - 100));
    text(0.0253, 0.6 * maxy + 100, wintype_list{ii}, 'fontsize', 15);
    axis([0.025 0.045 (maxy * [-1 1] + 100)]);
```

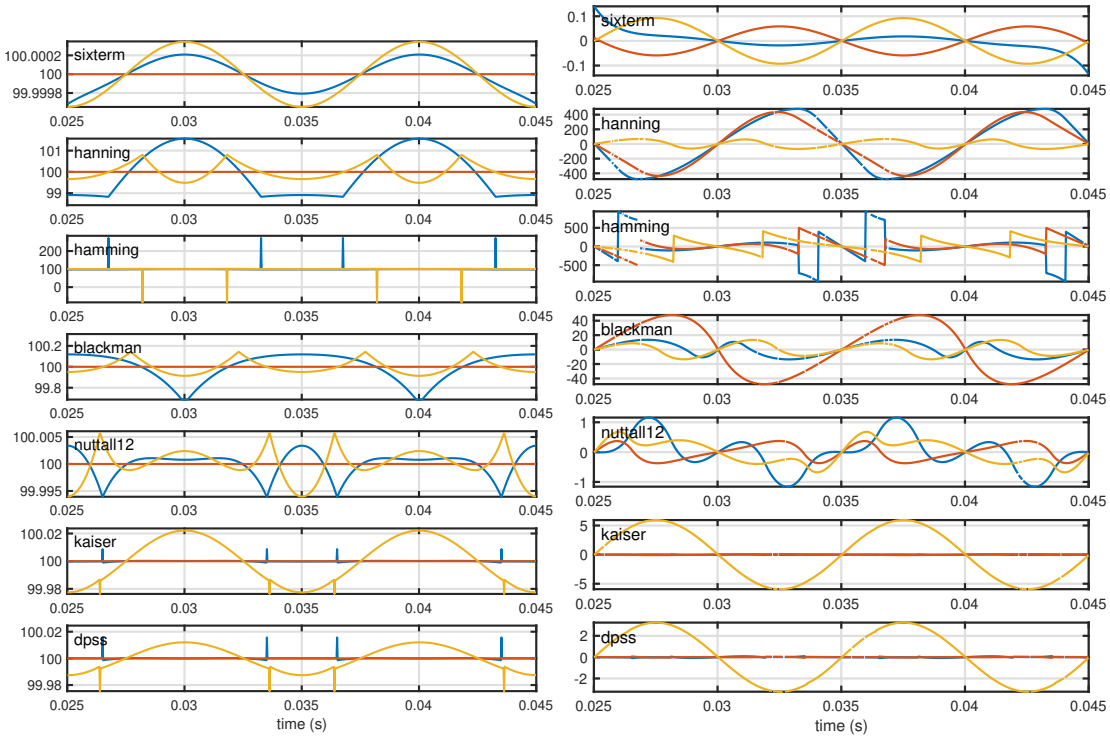

Figure 11: Effect of envelope shape on the instantaneous frequency and the group delay. The vertical axis of the instantaneous frequency plot represents values in Hz. The vertical axis of the group delay plot represents values in micro second.

```
    grid on;
    drawnow
end
xlabel('time (s)')
print -depsc instFreqWin100Hz.eps
```

Group delay

```
fs = 44100;
fo = 100;
duration = 0.071;
tt = (1:round(duration * fs))' / fs;
x = tt * 0;
nto = round(fs / fo);
x(1:nto:end) = 1;
```

```
fl = 50;
fh = 1200;
channels_oct = 12;
mag_base = 1.25;
wintype_list = {'sixterm', 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
mag_duration_list = [0.4444 0.2831 0.3050 0.3566 0.4020 0.3619 0.3676];
figure('position', [680   218   574   760]);
for ii = 1:length(wintype_list)
    mag = mag_base / mag_duration_list(ii) * mag_duration_list(ii);
    wvltStr = designAnalyticWavelet(fs, fl, fh, channels_oct, mag, wintype_list{ii});
    output = waveletAttributesAnalyzer(x, fs, wvltStr);
    subplot(7,1,ii);
    plot(tt(tt > 0.025 & tt < 0.045), 1000000 * output.group_delay_map(tt > 0.025 & tt < 0.045, 11:13), ...
        'linewidth', 2);
    set(gca, 'fontsize', 13, 'linewidth', 2);
    tmpdata = 1000000 * output.group_delay_map(tt > 0.025 & tt < 0.045, 11:13);
    maxy = max(abs(tmpdata(:)));
    text(0.0253, 0.6 * maxy, wintype_list{ii}, 'fontsize', 15);
    axis([0.025 0.045 maxy * [-1 1]]);
    grid on;
    drawnow
end
xlabel('time (s)')
print -depsc groupDelayWin100Hz.eps
```

3.4 Input Arguments

This function has the following input arguments.

x Input signal column vector. This function assumes the number of columns is 1.

fs A scalar variable representing the sampling frequency of the input signal. The unit is (Hz). This value should be the same as the value used to design the analysis wavelet using `designAnalyticWavelet`.

wvltStr A structure variable consisting of analysis wavelet information. This variable is the output of the function `designAnalyticWavelet`.

3.5 Output Arguments

The following shows the example of the analysis results calculated using the scripts introduced in the previous sections. It shows the fields of the output structure variable `output`.

```
          rawWavelet: [3131 × 56 double]
     amp_squared_map: [3131 × 56 double]
       inst_freq_map: [3131 × 56 double]
    group_delay_map: [3131 × 56 double]
```

```
  elapsedTimeForFiltering: 0.0344
elapsedTimeForPostProcess: 0.0082
              elapsedTime: 0.0429
```

The name of each field is self-explanatory. Brief descriptions follow.

rawWavelet Complex valued matrix. Each column consists of each filter output. The sampling rate is the same as the input.

amp_squared_map Real-valued matrix. Each column consists of the absolute value of the product of neighboring samples of each filter output. The sampling rate is the same as the input.

inst_freq_map Real-valued matrix. Each column consists of the instantaneous frequency of each neighboring filter output. The sampling rate is the same as the input.

group_delay_map Real-valued matrix. Each column consists of the group delay of each filter output. The sampling rate is the same as the input.

elapsedTimeForFiltering The elapsed time for filtering using the wavelet filter bank.

elapsedTimeForPostProcess The elapsed time for post-processing.

elapsedTime The total elapsed time of this function.

3.6 Extended Capabilities

NA

3.7 See Also

`sourceAttributesAnalysis, designAnalyticWavelet`

4 sourceAttributesAnalysis

Source attribute analysis using analytic signal wavelet

4.1 Syntax

```
outStr = sourceAttributesAnalysis(x, fs)
outStr = sourceAttributesAnalysis(x, fs, range)
outStr = sourceAttributesAnalysis(x, fs, range, outStr)
outStr = sourceAttributesAnalysis(x, fs, range, low_frequency, high_freuency)
outStr = sourceAttributesAnalysis(x, fs, range, low_frequency, high_freuency, ...
            channels_in_octave);
outStr = sourceAttributesAnalysis(x, fs, range, low_frequency, high_freuency, ...
            channels_in_octave, dsOn)
outStr = sourceAttributesAnalysis(x, fs, range, low_frequency, high_freuency, ...
            channels_in_octave, dsOn, stretching_factor)
outStr = sourceAttributesAnalysis(x, fs, range, low_frequency, high_freuency, ...
            channels_in_octave, dsOn, stretching_factor, wintype)
outStr = sourceAttributesAnalysis(x, fs, range, low_frequency, high_freuency, ...
            channels_in_octave, dsOn, stretching_factor, wintype, ...
            integration_time)
outStr = sourceAttributesAnalysis(x, fs, range, low_frequency, high_freuency, ...
            channels_in_octave, dsOn, stretching_factor, wintype, ...
            integration_time, sampling_multiplier)
```

4.2 Description

This function analyzes the input signal for extracting the fundamental frequency and excitation events. However, the event extraction function is the topic for the next release. In addition to the instantaneous frequency and group delay, it yields fo candidates based on the fixed point analysis, which is briefly outlined in 3.3.3

4.3 Examples

4.3.1 Default use

The following script introduces the most straightforward example use. The sample signal is an excerpt from the CMU ARCTIC database[9]. The sentence is “For the twentieth time that evening the two men shook hands.” The file has two channels. The first channel is a recorded speech signal, and the second channel is an EGG (ElectorGlotGraph) signal.

Reading file and execute analysis

```
dataBaseDir = '/Users/kawahara/Music/CMU_arctic/';
talker_dir = {'cmu_us_bdl_arctic', 'cmu_us_jmk_arctic', 'cmu_us_slt_arctic'};
talkerId = 3;
fileNames = dir([dataBaseDir talker_dir{talkerId} '/orig/*.wav']);
fileId = 3;
[x, fs] = audioread([fileNames(fileId).folder '/' fileNames(fileId).name]);
outStr = sourceAttributesAnalysis(x(:, 1), fs); % speech signal is in ch-1
```

Visualization of the fundamental component candidates

```
tsig = (1:length(x)) / fs;
tx = outStr.time_axis_wavelet;
fl = outStr.wvltStrDs.fc_list(1);
fh = outStr.wvltStrDs.fc_list(end);
figure;
semilogy(outStr.time_axis_wavelet, outStr.fixed_points_freq(:, 1:4), '.', ...
    'linewidth', 2);
hold all;
semilogy(tsig, x(:, 1)/max(abs(x(:, 1))) * 20 + 80, 'k');
grid on;
axis([tx([1 end]) fl fh]);
set(gca, 'fontsize', 14, 'linewidth', 2);
xlabel('time (s)');
ylabel('frequency (Hz)');
title([talker_dir{talkerId} '  ' fileNames(fileId).name], 'interpreter', 'none');
legend(num2str((1:4)'));
```

Visualization of the estimated SNR of the candidates

```
figure;
plot(outStr.time_axis_wavelet, outStr.fixed_points_measure(:, 1:4), '.', ...
    'linewidth', 2);
hold all
plot(tsig, x(:, 1)/max(abs(x(:, 1))) * 4 - 10, 'k');
grid on;
axis([tx([1 end]) -15 50]);
set(gca, 'fontsize', 14, 'linewidth', 2);
xlabel('time (s)');
ylabel('estimated SNR (dB)');
title([talker_dir{talkerId} '  ' fileNames(fileId).name], 'interpreter', 'none');
legend(num2str((1:4)'));
```

Figure 12 shows the analysis results. The first four salient candidates are selected based on the estimated SNR. Fixed-points of the center frequency to the instantaneous frequency mapping yielded the candidates. In this default analysis condition, the input signal was downsampled based on the highest center frequency and the original audio sampling frequency. The filter density of this default condition is 24 channels/octave. The sampling frequency of this file is 32,000 Hz. The total elapsed time was 1.05 s. The input

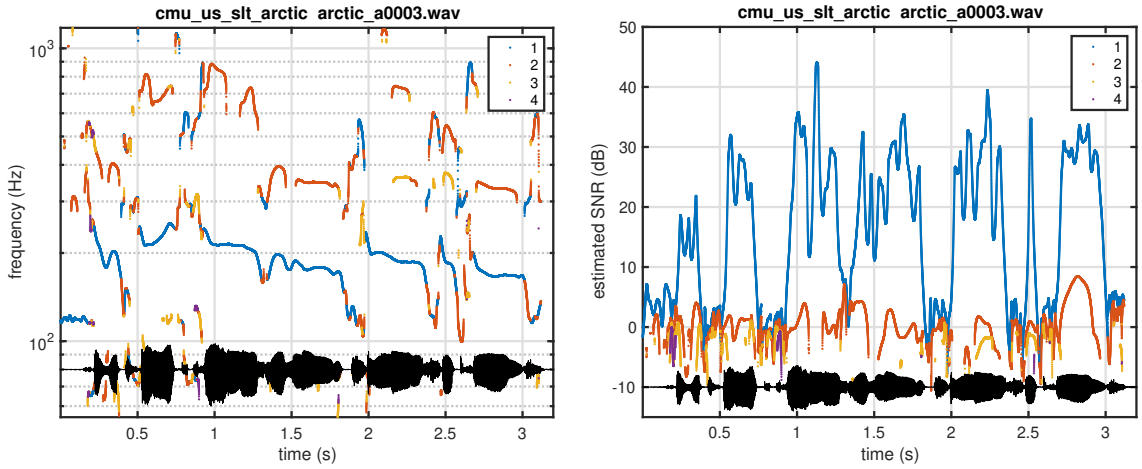

Figure 12: Analysis example of an excerpt from CMU ARCTIC. The left panel shows the fo frequency candidates. The right panel shows the estimated SNR of each candidate. The first four salient candidates are selected.

signal length is 3.2 s. The system used for this analysis is MacBookPro 13" Core i7, 2.9 GHz and 16 GB memory. The MATLAB version is R2018b.

4.3.2 Detailed analysis at the audio sampling rate

The function allows users to customize the analysis condition. This section provides an example.

The following script displays the scatter plot of the candidate frequency and the candidate SNR at the same time. This plot provides information to determine the detailed analysis conditions.

```
outStr = sourceAttributesAnalysis(x(:, 1), fs); % speech signal is in ch-
fl = outStr.wvltStrDs.fc_list(1);
fh = outStr.wvltStrDs.fc_list(end);
figure;
semilogx(outStr.fixed_points_freq(:, 1:4), outStr.fixed_points_measure(:, 1:4), '.');
grid on;
set(gca, 'fontsize', 14, 'linewidth', 2);
axis([fl fh -15 50]);
xlabel('frequency (Hz)');
ylabel('estimated SNR (dB)');
title([talker_dir{talkerId} '  ' fileNames(fileId).name], 'interpreter', 'none');
print -depsc foCandScatterExample.eps
```

Then the following script shows how to set all the customizable parameters.

```
fl = 100;
fh = 320;
channels_in_octave = 6;
dsOn = 0;
mag_duration_list = [0.4444 0.2831 0.3050 0.3566 0.4020 0.3619 0.3676];
stretching_factor = 1.01 / mag_duration_list(7) * mag_duration_list(1);
wintype = 'dpss';
integration_time = 8;
outStr = sourceAttributesAnalysis(x(:, 1), fs, [1 length(x)], fl, fh, ...
    channels_in_octave, dsOn, stretching_factor, wintype, ...
    integration_time);
```

Note that `mag_duration_list` provides conversion of the stretching factor between the six-term series and DPSS.

The following script shows how to visualize the results.

```
tsig = (1:length(x)) / fs;
tx = outStr.time_axis_wavelet;
fl = outStr.wvltStrDs.fc_list(1);
fh = outStr.wvltStrDs.fc_list(end);
figure;
subplot(211)
semilogy(outStr.time_axis_wavelet, outStr.fixed_points_freq(:, 1:4), '.', ...
    'linewidth', 2);
hold all;
semilogy(tsig, x(:, 1)/max(abs(x(:, 1))) * 20 + 80, 'k');
grid on;
axis([tx([1 end]) fl fh]);
set(gca, 'fontsize', 14, 'linewidth', 2);
xlabel('time (s)');
ylabel('frequency (Hz)');
title([talker_dir{talkerId} '  ' fileNames(fileId).name], 'interpreter', 'none');
legend(num2str((1:4)'), 'location', 'eastoutside');
subplot(212)
plot(outStr.time_axis_wavelet, outStr.fixed_points_measure(:, 1:4), '.', ...
    'linewidth', 2);
hold all
plot(tsig, x(:, 1)/max(abs(x(:, 1))) * 4 - 10, 'k');
grid on;
axis([tx([1 end]) -15 50]);
set(gca, 'fontsize', 14, 'linewidth', 2);
xlabel('time (s)');
ylabel('estimated SNR (dB)');
title([talker_dir{talkerId} '  ' fileNames(fileId).name], 'interpreter', 'none');
legend(num2str((1:4)'), 'location', 'eastoutside');
print -depsc foFreqSNRASRCandExample.eps
```

Figure 13 shows how to find relevant and detailed analysis conditions. It also shows the detailed analysis results. Note that the detailed analysis yielded the fo candidates at the audio sampling rate. The elapsed time for this detailed analysis was 0.85 s. This elapsed time is still faster than real-time.

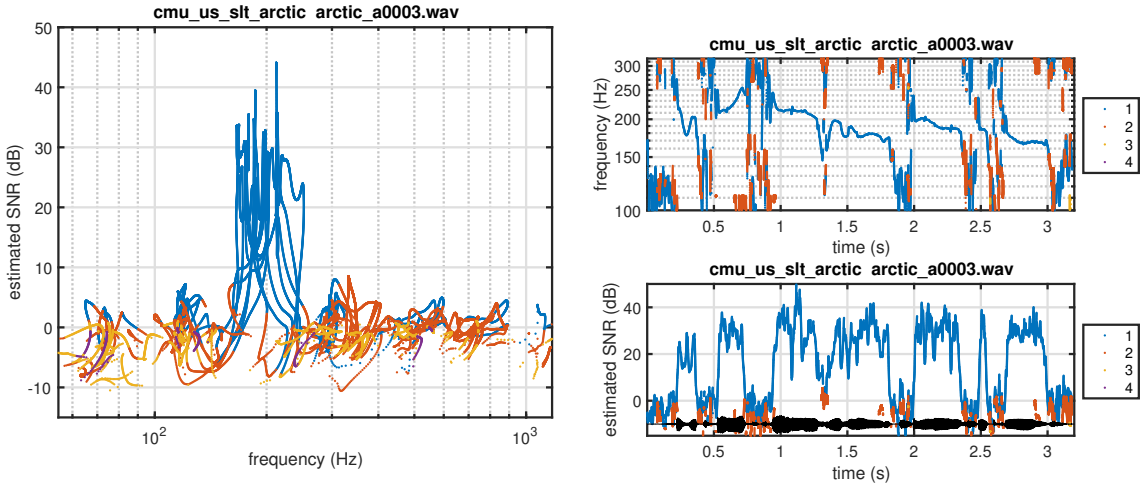

Figure 13: Analysis example of an excerpt from CMU ARCTIC. The left panel shows the scatter plot of the instantaneous frequency and the estimated SNR. The right panel shows the analysis results at the audio sampling rate resolution. The scatter plot suggested the relevant fo search range.

4.3.3 Speeding up for repetitive calculation

The following form is for efficient repetitive use of the designed wavelet.

```
outStr = sourceAttributesAnalysis(x, fs, range, outStr)
```

The input argument `outStr` is the output of other calling from. The calling form shown above skips designing process of wavelet and uses the previously designed one. The following script shows how to compare with and without initialization condition on elapsed time. The following script is for testing "DPSS" envelope.

```
fs = 44100;
fl = 55;
fh = 1200;
duration = 1;
ch_in_oct = 6;
dsOn = 1;
mag = 1.05;
winType = 'dpss';
tt = (1:duration * fs)' / fs;
x = randn(length(tt), 1);
initialoutput = sourceAttributesAnalysis(x, fs, [1 length(x)], fl, fh, ...
    ch_in_oct, dsOn, mag, winType);
duration_list = 0.1 * 2 .^ (0:1/2:4);
elapsed_time_mean = zeros(length(duration_list), 2);
```

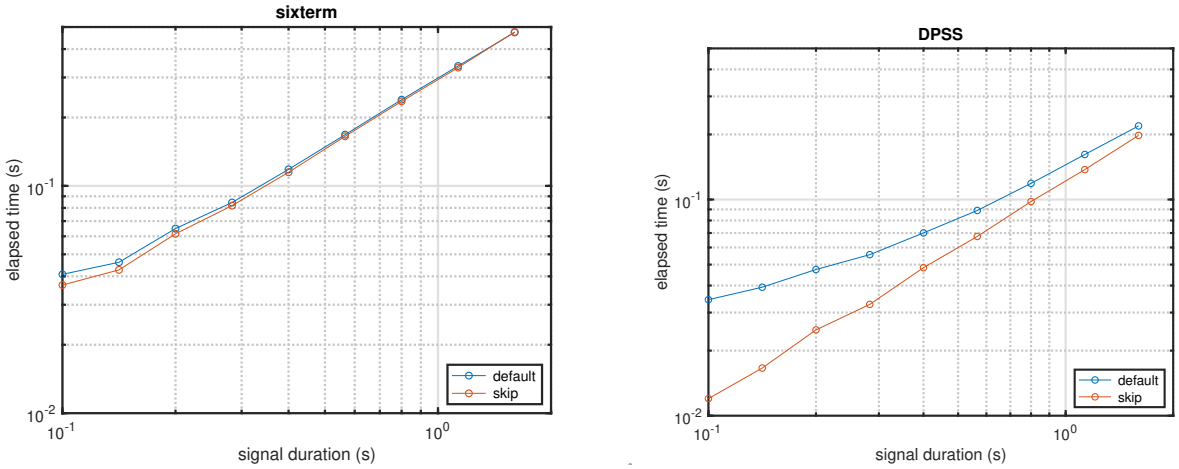

Figure 14: Elapsed time difference between with and without initialization. The left plot shows the default setting with the six-term envelope. The right plot shows the results using "DPSS" function.

```
elapsed_time_SD = zeros(length(duration_list), 2);
iteration = 20;
for ii = 1:length(duration_list)
    tmp_default = zeros(iteration, 1);
    tmp_skip = zeros(iteration, 1);
    for jj = 1:iteration
    x = randn(round(duration_list(ii) * fs), 1);
    out_default = sourceAttributesAnalysis(x, fs, [1 length(x)], fl, fh, ...
        ch_in_oct, dsOn, mag, winType);
    out_skip = sourceAttributesAnalysis(x, fs, [1 length(x)], initialoutput);
    tmp_default(jj) = out_default.elapsedTime;
    tmp_skip(jj) = out_skip.elapsedTime;
    end
    elapsed_time_mean(ii, :) = [mean(tmp_default), mean(tmp_skip)];
    elapsed_time_SD(ii, :) = [std(tmp_default), std(tmp_skip)];
end
set(gca, 'fontsize', 14, 'linewidth', 2);
title('DPSS');
xlabel('signal duration (s)');
ylabel('elapsed time (s)');
legend(' default', ' skip', 'location', 'southeast')
axis([0.1 2 0.01 0.5]);
print -depsc dpssEtimeSkip.eps
```

Figure 14 compares the elapsed time with and without initialization when repeating the same filtering. The results are the average of 20 iterations. These indicates that the cosine series initialization is negligible. However, the initialization time is not negligible

for “DPSS.”

The following script checks the effects of initialization for very short segment. This applies for real-time applications.

```
fs = 44100;
fl = 55;
fh = 1200;
duration = 0.1;
ch_in_oct = 6;
dsOn = 1;
mag_base = 1.05;
wintype_list = {'sixterm', 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
mag_duration_list = [0.4444 0.2831 0.3050 0.3566 0.4020 0.3619 0.3676];
iteration = 100;
elapsed_time_median = zeros(length(wintype_list), 2);
for ii = 1:length(wintype_list)
    mag = mag_base / mag_duration_list(ii) * mag_duration_list(1);
    x = randn(round(duration * fs), 1);
    tmp_default = zeros(iteration, 1);
    tmp_skip = zeros(iteration, 1);
    initialoutput = sourceAttributesAnalysis(x, fs, [1 length(x)], fl, fh, ...
        ch_in_oct, dsOn, mag, wintype_list{ii});
    for jj = 1:iteration + 1
        x = randn(round(duration * fs), 1);
        out_default = sourceAttributesAnalysis(x, fs, [1 length(x)], fl, fh, ...
            ch_in_oct, dsOn, mag, wintype_list{ii});
        out_skip = sourceAttributesAnalysis(x, fs, [1 length(x)], initialoutput);
        tmp_default(max(1, jj - 1)) = out_default.elapsedTime;
        tmp_skip(max(1, jj - 1)) = out_skip.elapsedTime;
    end
    elapsed_time_median(ii, :) = [median(tmp_default), median(tmp_skip)];
end
figure;barh(elapsed_time_median);grid on;
set(gca, 'fontsize', 14, 'linewidth', 2, 'yticklabel', wintype_list);
xlabel('elapsed time (s)');
print -depsc winAndSpeedInit.eps
```

The following script checks the effect of the segment duration and the type of envelope functions.

```
fs = 44100;
fl = 55;
fh = 1200;
duration = 0.1;
ch_in_oct = 6;
dsOn = 1;
mag_base = 1.05;
wintype_list = {'sixterm', 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
mag_duration_list = [0.4444 0.2831 0.3050 0.3566 0.4020 0.3619 0.3676];
duration_list = 0.1 * 2 .^ (0:1/2:4);
iteration = 10;
elapsed_time_median = zeros(length(duration_list), 2);
```

```
figure;
for ii = 1:length(wintype_list)
    mag = mag_base / mag_duration_list(ii) * mag_duration_list(1);
    x = randn(round(duration * fs), 1);
    for kk = 1:length(duration_list)
        tmp_default = zeros(iteration, 1);
        tmp_skip = zeros(iteration, 1);
        initialoutput = sourceAttributesAnalysis(x, fs, [1 length(x)], fl, fh, ...
            ch_in_oct, dsOn, mag, wintype_list{ii});
        for jj = 1:iteration
            x = randn(round(duration_list(kk) * fs), 1);
            out_default = sourceAttributesAnalysis(x, fs, [1 length(x)], fl, fh, ...
                ch_in_oct, dsOn, mag, wintype_list{ii});
            out_skip = sourceAttributesAnalysis(x, fs, [1 length(x)], initialoutput);
            tmp_default(jj) = out_default.elapsedTime;
            tmp_skip(jj) = out_skip.elapsedTime;
        end
        elapsed_time_median(kk, :) = [median(tmp_default), median(tmp_skip)];
    end
    loglog(duration_list, elapsed_time_median(:, 2), 'o-', 'linewidth', 2);
    grid on;hold on;
    axis([0.1 2 0.01 0.3]);
    drawnow
end
set(gca, 'fontsize', 14, 'linewidth', 2);
xlabel('signal duration (s)');
ylabel('elapsed time (s)');
legend(wintype_list, 'location', 'southeast');
print -depsc winDurationAndSpeedInit.eps
```

Figure 15 shows the effect of initialization and envelope function on the elapsed time. These plots are outputs of the scripts introduced above. For my surprise, virtually no significant effect exists. The left panel shows that the initialization effect is significant for "DPSS."

4.3.4 Calibration of SNR and effects of downsampling

This section test the accuracy of the estimated SNR. The following script calculates the estimated SNR of the test signals with known SNR using different envelope functions.

$$x[n] = x_p[n] + r[n]\sigma(x_p[n])10^{-\frac{c_{\mathrm{SNR}}}{20}} \tag{13}$$

where c_{SNR} represents the given SNR (Signal to Noise Ratio) in dB. The value $\sigma(x_p[n])$ represents the standard deviation of the periodic pulse train $x_p[n]$. The signal $r[n]$ is Gaussian white noise. Note that other document[10] is this release provides the comprehensive calibration results.

The first script tests all combinations of downsampling and the fundamental frequency.

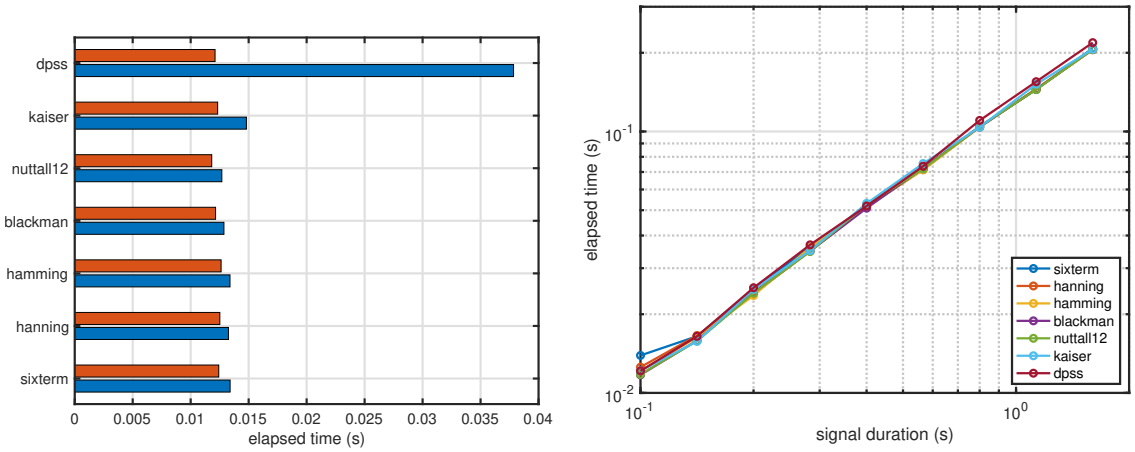

Figure 15: Effects of initialization and envelope function. The left panel shows initialization effects. The right panel shows effects of envelope function.

```
fs = 44100;
fo = 240;
fl = fo * 2 ^ (-1 / 2);
fh = fo * 2 ^ (1 / 2);
duration = 1;
ch_in_oct = 12;
dsOn = 0;
mag_base = 1.05;
wintype_list = {'sixterm', 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
mag_duration_list = [0.4444 0.2831 0.3050 0.3566 0.4020 0.3619 0.3676];
integration_time = 15;
iteration = 1;
snr_list = 0:10:80;
snr_median = zeros(length(snr_list), 1);
tt = (1:round(duration * fs))' / fs;
xp = tt * 0;
nto = round(fs / fo);
xp(1:nto:end) = 1;
figure;
for ii = 1:length(wintype_list)
    mag = mag_base / mag_duration_list(ii) * mag_duration_list(1);
    x = randn(round(duration * fs), 1);
    for kk = 1:length(snr_list)
        snr = snr_list(kk);
        tmp_default = zeros(iteration, 1);
        tmp_skip = zeros(iteration, 1);
        initialoutput = sourceAttributesAnalysis(x, fs, [1 length(x)], fl, fh, ...
            ch_in_oct, dsOn, mag, wintype_list{ii}, integration_time);
        for jj = 1:iteration
```

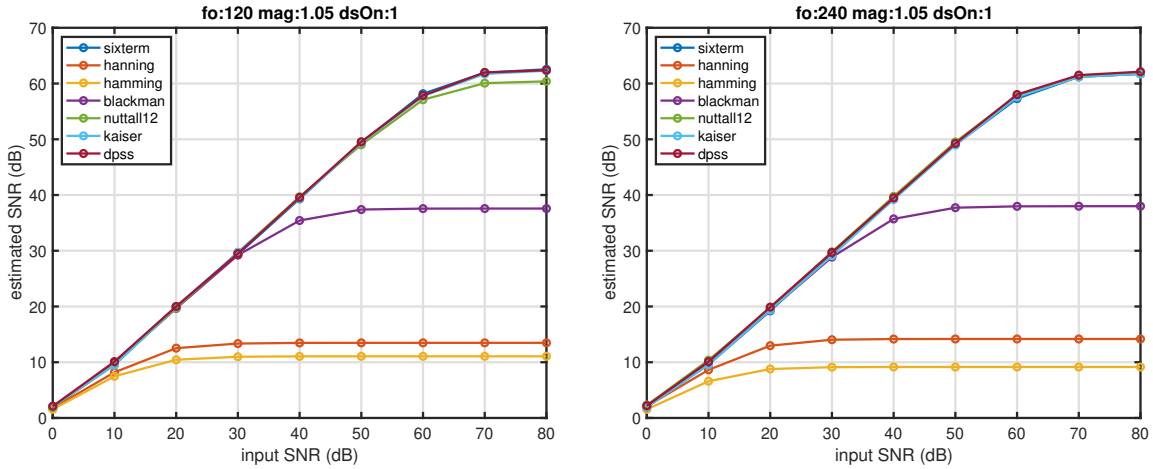

Figure 16: Calibration of estimated SNR with automatic downsampling.

```
            x = xp + randn(round(duration * fs), 1) * std(xp) * 10 ^ (-snr / 20);
            out_skip = sourceAttributesAnalysis(x, fs, [1 length(x)], initialoutput);
            tx = out_skip.time_axis_wavelet;
            tmp_skip(jj) = median(out_skip.fixed_points_measure(tx > 0.2 & tx < duration - 0.2, 1));
        end
        snr_median(kk) = median(tmp_skip);
    end
    plot(snr_list, snr_median, 'o-', 'linewidth', 2);
    grid on;hold on;
    drawnow
end
set(gca, 'fontsize', 14, 'linewidth', 2);
xlabel('input SNR (dB)');
ylabel('estimated SNR (dB)');
legend(wintype_list, 'location', 'northwest');
title(['fo:' num2str(fo) ' mag:' num2str(mag_base) ' dsOn:' num2str(dsOn)]);
outFname = ['winSNRcalibration' num2str(dsOn) 'fo' num2str(fo) 'Hz.eps'];
print('-depsc',  outFname);
```

Figure 16 shows the default use, which is with automatic downsampling. The left panel shows the results using 120 Hz pulse train and the right panel shows the results using 240 Hz pulse train. These indicates that there are no significant difference between accuracy between three envelope functions; the six-term series, Kaiser, and DPSS.

Figure 17 shows the results without downsampling. The results using Hamming show strongly biased estimates. The result using the six-term series is virtually linear up to 80 dB SNR. The results using Kaiser and DPSS deviates from 60 dB SNR from the linear identity mapping. This indicates that the six-term series is the best for detailed analysis. However, issue is that is it necessary this linearity in analyzing actual speech data.

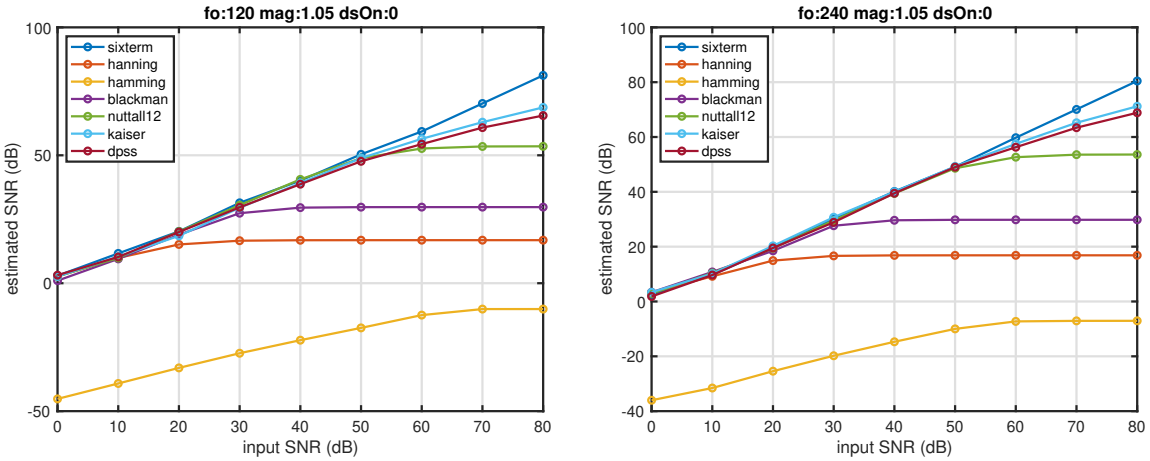

Figure 17: Calibration of estimated SNR without automatic downsampling.

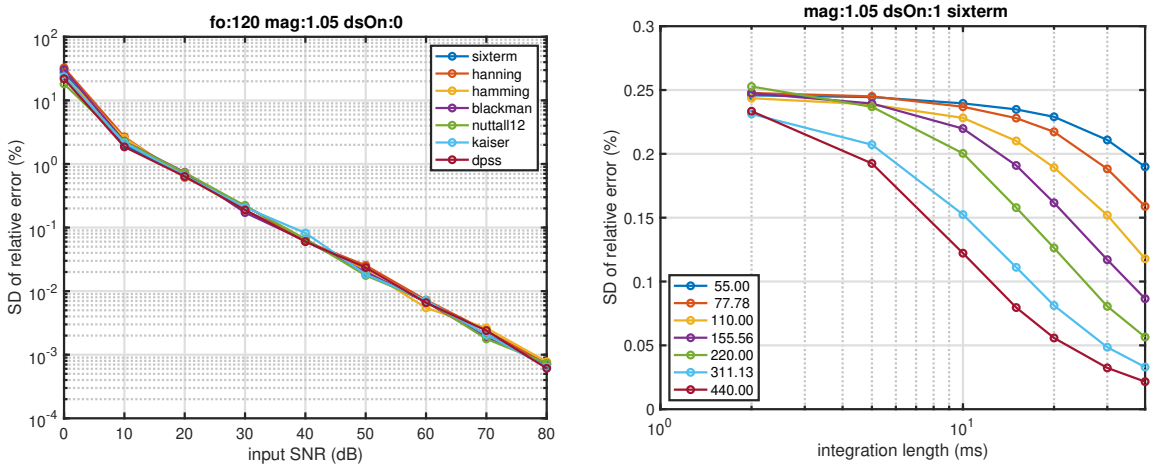

Figure 18: The left panel shows the relative fo estimation error (standard deveiation) on SNR. The right panel shows the relative fo estimation error on the integration time for each input fo. The SNR in the right panel is 30 dB.

4.3.5 SNR and fo estimation error and integration time

Figure 18 shows the relative fo error on SNR, envelope function, and the integration time. The following scripts output the results.

SNR and envelope function on fo error

```
fs = 44100;
fo = 120;
fl = fo * 2 ^ (-1 / 2);
fh = fo * 2 ^ (1 / 2);
duration = 1;
ch_in_oct = 12;
dsOn = 0;
mag_base = 1.05;
wintype_list = {'sixterm', 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
mag_duration_list = [0.4444 0.2831 0.3050 0.3566 0.4020 0.3619 0.3676];
integration_time = 15;
iteration = 1;
snr_list = 0:10:80;
fo_err_SD = zeros(length(snr_list), 1);
tt = (1:round(duration * fs))' / fs;
xp = tt * 0;
nto = round(fs / fo);
xp(1:nto:end) = 1;
figure;
for ii = 1:length(wintype_list)
    mag = mag_base / mag_duration_list(ii) * mag_duration_list(1);
    x = randn(round(duration * fs), 1);
    for kk = 1:length(snr_list)
        snr = snr_list(kk);
        tmp_default = zeros(iteration, 1);
        tmp_skip = zeros(iteration, 1);
        initialoutput = sourceAttributesAnalysis(x, fs, [1 length(x)], fl, fh, ...
            ch_in_oct, dsOn, mag, wintype_list{ii}, integration_time);
        for jj = 1:iteration
            x = xp + randn(round(duration * fs), 1) * std(xp) * 10 ^ (-snr / 20);
            out_skip = sourceAttributesAnalysis(x, fs, [1 length(x)], initialoutput);
            tx = out_skip.time_axis_wavelet;
            tmp_skip(jj) = ...
                 std(out_skip.fixed_points_freq(tx > 0.2 & tx < duration - 0.2, 1)) / fo * 100;
        end
        fo_err_SD(kk) = mean(tmp_skip);
    end
    semilogy(snr_list, fo_err_SD, 'o-', 'linewidth', 2);
    grid on;hold on;
    drawnow
end
set(gca, 'fontsize', 14, 'linewidth', 2);
xlabel('input SNR (dB)');
ylabel('SD of relative error (%)');
legend(wintype_list, 'location', 'northeast');
title(['fo:' num2str(fo) ' mag:' num2str(mag_base) ' dsOn:' num2str(dsOn)]);
outFname = ['winSNRFoErrTest' num2str(dsOn) 'fo' num2str(fo) 'Hz.eps'];
print('-depsc',  outFname);
```

The results indicates that the fo error is virtually independent on the envelope function

shape.

Fo error on the integration time Because the fo error is independent on the envelope function, the following tests uses the six-term series only.

```
fs = 44100;
fo = 240;
fl = fo * 2 ^ (-1 / 2);
fh = fo * 2 ^ (1 / 2);
fo_list = 55 * 2 .^ (0:1/2:3);
duration = 100;
ch_in_oct = 12;
dsOn = 1;
mag_base = 1.05;
wintype_list = {'sixterm'};%, 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
mag_duration_list = [0.4444 0.2831 0.3050 0.3566 0.4020 0.3619 0.3676];
integration_time_list = [2 5 10 15 20 30 40];
iteration = 1;
snr = 30;
fo_err_SD = zeros(length(integration_time_list), 1);
fo_raw_SD = zeros(length(fo_list), 1);
tt = (1:round(duration * fs))' / fs;
figure;
for jk = 1:length(fo_list)
    fo = fo_list(jk);
    fl = fo * 2 ^ (-1 / 2);
    fh = fo * 2 ^ (1 / 2);
    xp = tt * 0;
    nto = round(fs / fo);
    xp(1:nto:end) = 1;
    for ii = 1:length(wintype_list)
        mag = mag_base / mag_duration_list(ii) * mag_duration_list(1);
        x = randn(round(duration * fs), 1);
        x = xp + randn(round(duration * fs), 1) * std(xp) * 10 ^ (-snr / 20);
        for kk = 1:length(integration_time_list)
            tmp_default = zeros(iteration, 1);
            tmp_skip = zeros(iteration, 1);
            initialoutput = sourceAttributesAnalysis(x, fs, [1 length(x)], fl, fh, ...
                ch_in_oct, dsOn, mag, wintype_list{ii}, integration_time_list(kk));
            for jj = 1:iteration
                out_skip = sourceAttributesAnalysis(x, fs, [1 length(x)], initialoutput);
                tx = out_skip.time_axis_wavelet;
                tmp_skip(jj) = ...
                    std(out_skip.fixed_points_freq(tx > 0.8 & tx < duration - 0.8, 1)) / fo * 100;
                foSeq = out_skip.fixed_points_freq(tx > 0.8 & tx < duration - 0.8, 1);
                foSeq = foSeq - mean(foSeq);
            end
            fo_err_SD(kk) = mean(tmp_skip);
        end
        semilogx(integration_time_list, fo_err_SD, 'o-', 'linewidth', 2);
```

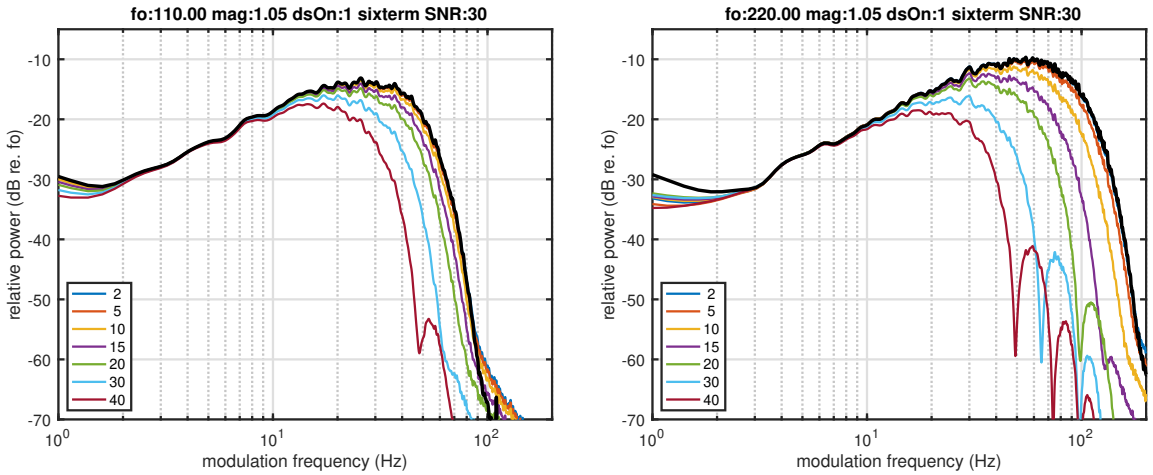

Figure 19: Power spectra of the estimated fo trajectories. The legend represents the integration time of the weighted averaging.

```
        grid on;hold on;
        drawnow
    end
    fo_raw_SD(jk) = std(out_skip.inst_freq_map(tx > 0.8 & tx < duration - 0.8, 7)) / fo * 100;
end
set(gca, 'fontsize', 14, 'linewidth', 2);
xlabel('integration length (ms)');
ylabel('SD of relative error (%)');
legend(num2str(fo_list', '%6.2f'), 'location', 'southwest');
title([' mag:' num2str(mag_base) ' dsOn:' num2str(dsOn) ' ' wintype_list{1}]);
outFname = ['winIntFoErrTest' num2str(dsOn) 'z.eps'];
print('-depsc',  outFname);
```

4.3.6 Spectrum of the estimated fo trajectory

Figure 19 shows the power spectra of the estimated fo trajectory normalized by the fo value. The black line in the plot shows the power spectra of the raw estimate of the fo. Because the filtering itself also has smoothing effects on the estimated raw fo trajectory, smoothing post processing does not affect significantly up to 5 ms integration time.

```
fs = 44100;
fo_list = 55 * 2 .^ (0:1/2:3);
duration = 100;
ch_in_oct = 12;
dsOn = 1;
mag_base = 1.05;
wintype_list = {'sixterm'};%, 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
```

```
mag_duration_list = [0.4444 0.2831 0.3050 0.3566 0.4020 0.3619 0.3676];
integration_time_list = [2 5 10 15 20 30 40];
iteration = 1;
snr_list = 0:10:80;
snr = 30;
fo_err_SD = zeros(length(integration_time_list), 1);
tt = (1:round(duration * fs))' / fs;
fftl = 32768;
figure;
for jk = 5%1:length(fo_list)
    fo = fo_list(jk);
    fl = fo * 2 ^ (-1 / 2);
    fh = fo * 2 ^ (1 / 2);
    xp = tt * 0;
    nto = round(fs / fo);
    xp(1:nto:end) = 1;
    for ii = 1:length(wintype_list)
        mag = mag_base / mag_duration_list(ii) * mag_duration_list(1);
        x = randn(round(duration * fs), 1);
        x = xp + randn(round(duration * fs), 1) * std(xp) * 10 ^ (-snr / 20);
        for kk = 1:length(integration_time_list)
            tmp_raw = zeros(iteration, 1);
            tmp_skip = zeros(iteration, 1);
            initialoutput = sourceAttributesAnalysis(x, fs, [1 length(x)], fl, fh, ...
                ch_in_oct, dsOn, mag, wintype_list{ii}, integration_time_list(kk));
            out_skip = sourceAttributesAnalysis(x, fs, [1 length(x)], initialoutput);
            tx = out_skip.time_axis_wavelet;
            tmp_skip(jj) = ...
                std(out_skip.fixed_points_freq(tx > 0.8 & tx < duration - 0.8, 1)) / fo * 100;
            foSeq = out_skip.fixed_points_freq(tx > 0.8 & tx < duration - 0.8, 1);
            foSeq = (foSeq - mean(foSeq)) / fo;
            fds = fs / out_skip.downSamplinbgRate;
            sgramF = stftSpectrogramStructure(foSeq,fds,2000,100,'nuttallwin');
            fx = sgramF.frequencyAxis;
            semilogx(fx, 10*log10(mean(sgramF.rawSpectrogram,2)), 'linewidth', 2);
            grid on;hold on;
            drawnow
        end
    end
end
fo_raw_seq = out_skip.inst_freq_map(tx > 0.8 & tx < duration - 0.8, 7) / fo;
fo_raw_seq = fo_raw_seq - mean(fo_raw_seq);
sgramF = stftSpectrogramStructure(fo_raw_seq,fds,2000,100,'nuttallwin');
fx = sgramF.frequencyAxis;
semilogx(fx, 10*log10(mean(sgramF.rawSpectrogram,2)), 'k', 'linewidth', 3);
set(gca, 'fontsize', 14, 'linewidth', 2);
xlabel('modulation frequency (Hz)');
ylabel('relative power (dB re. fo)');
legend(num2str(integration_time_list'), 'location', 'southwest');
title([' fo:' num2str(fo, '%5.2f') ' mag:' num2str(mag_base) ' dsOn:' ...
```

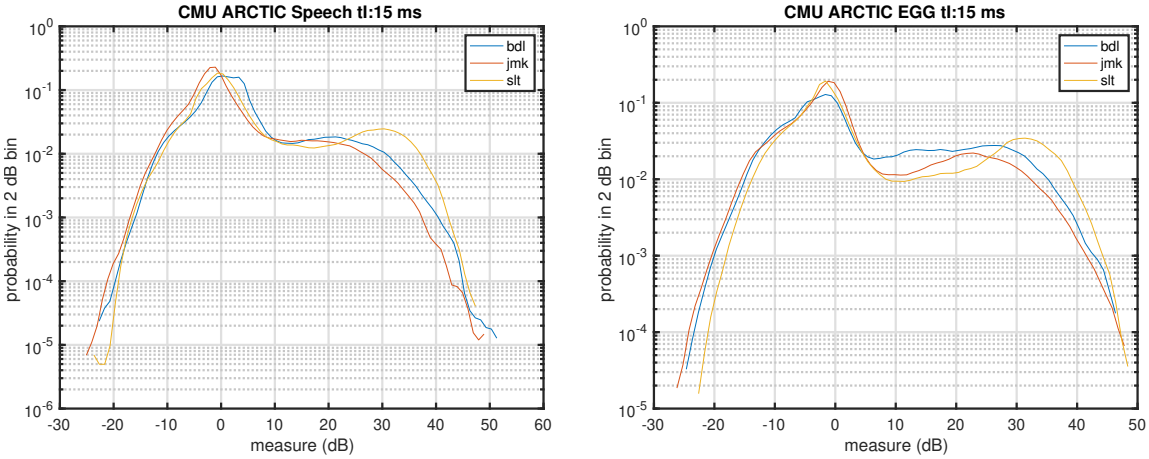

Figure 20: Estimated SNR distribution of CMU ARCTIC database.

```
   num2str(dsOn) ' ' wintype_list{1} ' SNR:' num2str(snr)]);
axis([1 200 -70 -5]);
outFname = ['winIntFoTrjSpec' num2str(dsOn) 'fo' num2str(fo, '%04.0f') 'Hz.eps'];
print('-depsc',  outFname);
```

4.3.7 SNR distribution in CMU ARCTIC speech database

Figure 20 shows distribution of the real speech and corresponding EGG signals. This suggests that Kaiser, DPSS, and the six-term series envelope functions are practically indistinguishable as far as the distribution.

```
baseDir = '/Users/kawahara/Music/CMU_arctic/';
talkerDir = {'cmu_us_bdl_arctic','cmu_us_jmk_arctic','cmu_us_slt_arctic'};

wintype_list = {'sixterm', 'hanning', 'hamming', 'blackman', 'nuttall12','kaiser','dpss'};
wtypeId = 1;
mag_effective = 1.05;
channels_oct = 12;
dsOn = 1;
fl = 55;
fh = 1200;
integration_time = 15;

for sourceID = 1:2
    switch sourceID
        case 1
            sourceType = 'Speech';
        case 2
            sourceType = 'EGG';
```

```
    end
    figure;
    for tlkrID = 1:length(talkerDir)
        origDir = [baseDir talkerDir{tlkrID} '/orig'];
        dirContents = dir([origDir '/arc*.wav']);
        selected_measure_agg = [];
        for fileId = 1:50%length(dirContents)
            pathname = [origDir '/' dirContents(fileId).name];
            disp(pathname);
            [x, fs] = audioread(pathname);
            output = sourceAttributesAnalysis(x(:, sourceID), fs, [1 length(x)], fl, fh, ...
                channels_oct, dsOn, mag_effective, wintype_list{wtypeId}, integration_time);
            fc_list = output.wvltStrDs.fc_list;
            tmp_measure = output.fixed_points_measure(:, 1:4);
            tmp_measure = tmp_measure(:);
            selected_measure = tmp_measure(~isnan(tmp_measure) & tmp_measure ~= -Inf & tmp_measure ~= Inf);
            selected_measure_agg = [selected_measure_agg; selected_measure(:)];
        end
        sorted_measure = sort(selected_measure_agg);
        sorted_measure = sorted_measure(diff(sorted_measure([1 1:end])) ~= 0);
        prob = (1:length(sorted_measure)) / length(sorted_measure);
        %semilogy(20 * log10(sorted_measure), prob, 'linewidth', 2);
        minM = sorted_measure(1);
        maxM = sorted_measure(end);
        half_bin = 1;
        centerM = minM + half_bin:maxM - half_bin;
        probU = interp1(sorted_measure, prob, centerM + half_bin, 'linear', 'extrap');
        probL = interp1(sorted_measure, prob, centerM - half_bin, 'linear', 'extrap');
        semilogy(centerM, probU-probL);grid on;
        grid on;
        hold all
        drawnow
    end
    set(gca, 'fontsize', 14, 'linewidth', 2)
    xlabel('measure (dB)');
    ylabel('probability in 2 dB bin');
    legend('bdl', 'jmk', 'slt')
    title(['CMU ARCTIC ' sourceType ' tI:' num2str(integration_time) ' ms'], 'interpreter', 'none');
    foutName = ['CMUarctic' sourceType num2str(integration_time) 'msTmp.eps'];
    print('-depsc',  foutName);
end
```

4.4 Input Arguments

The most flexible version of calling this function is as follows:

```
outStr = sourceAttributesAnalysis(x, fs, range, low_frequency, high_freuency, ...
     channels_in_octave, dsOn, stretching_factor, wintype, ...
     integration_time, sampling_multiplier)
```

These arguments are described below:

x A single column vector consisting of the input signal.

fs Sampling frequency. The unit is Hz.

range Two element vector. The first element determines the starting sample index and the second element determines the finishing sample index.

low_frequency The lowest frequency of the center frequency of the filters. The unit is Hz.

high_freuency The upper limit frequency of the center frequency of the filters. The unit is Hz.

channels_in_octave The density of the filter allocation in each octave.

dsOn Switch for determining the use of downsampling. 1: enable downsampling. 0: disable dowsampling.

stretching_factor Temporal stretching factor of the envelope function.

wintype A string variable defining the envelope function of the analytic signal. `'sixterm'`, `'hanning'`, `'hamming'`, `'blackman'`, `'nuttall12'`,`'kaiser'`,`'dpss'` are the registered names.

integration_time The length of the measure integration time represented as the length of Blackman windowing function. The unit is ms.

sampling_multiplier The `high_freuency` value times this number determines the target sampling frequency of downsampling.

The minimum requisite parameters are `x` and `fs`.

4.5 Output Arguments

The following shows an example output, contents of fields of the output structure variable.

```
            estPeriod: [25597 × 107 double]
            wvltStrDs: [1 × 1 struct]
           rawWavelet: [25597 × 107 double]
       inst_freq_map: [25597 × 107 double]
       if_smooth_map: [24958 × 107 double]
  downSamplinbgRate: 4
              fftlds: 4096
    half_aaf_length: 13
```

```
                w_aaf: [27 × 1 double]
   time_axis_wavelet: [1 × 25597 double]
    signal_time_axis: [1 × 102400 double]
          gd_dev_map: [24958 × 107 double]
          if_dev_map: [24958 × 107 double]
         n_effective: 24958
     mix_rev_measure: [24958 × 107 double]
         mix_measure: [24958 × 107 double]
  fixed_points_freq: [25597 × 107 double]
fixed_points_measure: [25597 × 107 double]
    fixed_points_amp: [25597 × 107 double]
         elapsedTime: 1.0530
```

Each field has the following meaning.

estPeriod This is a non-linearly converted SNR estimate. The value $R(r)$ of the estimated SNR (r) has the following form.

$$R(r) = \frac{1}{1 + \exp\left(-\dfrac{r-15}{6}\right)}, \tag{14}$$

where the hard corded numbers -15 and 6 are results of several trials.

wvltStrDs This is a copy of the input argument. It consists of the designed filter bank for the wavelet analysis.

rawWavelet This is a matrix of filter outputs. Each column vector is the output of each filter. The sampling rate is the same as the input signal.

inst_freq_map This is a matrix of the resulted attribute, instantaneous frequency. Each column vector is the sequence of the instantaneous frequency of each output. Note, this is not symmetric in terms of discrete sampling points. The sampling rate depends on downsampling condition.

if_smooth_map This is a matrix of the smoothed instantaneous frequency. The smoothing is a power weighted average of the raw instantaneous frequency. A Blackman windowing function is this smoothing function.

downSamplinbgRate This number represents the rate used for downsampling.

fftlds This field is useless now. This will be removed.

half_aaf_length This represents the half length of the smoothing function for power weighted average calculation n_h. The length of the function is $2n_h + 1$ samples.

w_aaf This column vector consists of the smoothing function/

time_axis_wavelet This row vector represents the time axis of the raw wavelet analysis results.

signal_time_axis This row vector represednts the time axis of the input signal.

gd_dev_map This matrix variable consists of an intermediate measure based on the group delay deviation. Each column represents the measure for each filter output. This measure is one component for calculating the estimated SNR.

if_dev_map This matrix variable consists of an intermediate measure based on the temporal deviation of the change of the instantaneous frequency. Each column represents the measure for each filter output. This measure is one component for calculating the estimated SNR.

n_effective This scalar variable represents the number of the non-zero elements of the calculated measures.

mix_rev_measure This matrix consists of the reciprocal of each mixed measure, which was made from the group delay-based measure and the instantaneous frequency change-based measure.

mix_measure This matrix consists of the calculated measure, which is the estimated SNR. It is calculated from the group delay-based measure and the instantaneous frequency change-based measure.

fixed_points_freq This matrix consists of the ordered candidates. The ordering is bases on the estimated SNR. The value of each candidate is the smoothed instantaneous frequency of the corresponding output. The number of the fixed points depends of the analysis results and is not a fixed number. The other element of this matrix are zero.

fixed_points_measure This matrix consists of the ordered candidates. The ordering is bases on the estimated SNR. The value of each candidate is the estimated SNR of the corresponding output. The number of the fixed points depends of the analysis results and is not a fixed number. The other element of this matrix are zero.

fixed_points_amp This matrix consists of the ordered candidates. The ordering is bases on the estimated SNR. The value of each candidate is the absolute value of the corresponding output. The number of the fixed points depends of the analysis results and is not a fixed number. The other element of this matrix are zero.

elapsedTime This scalar variable represents the elapsed time in second.

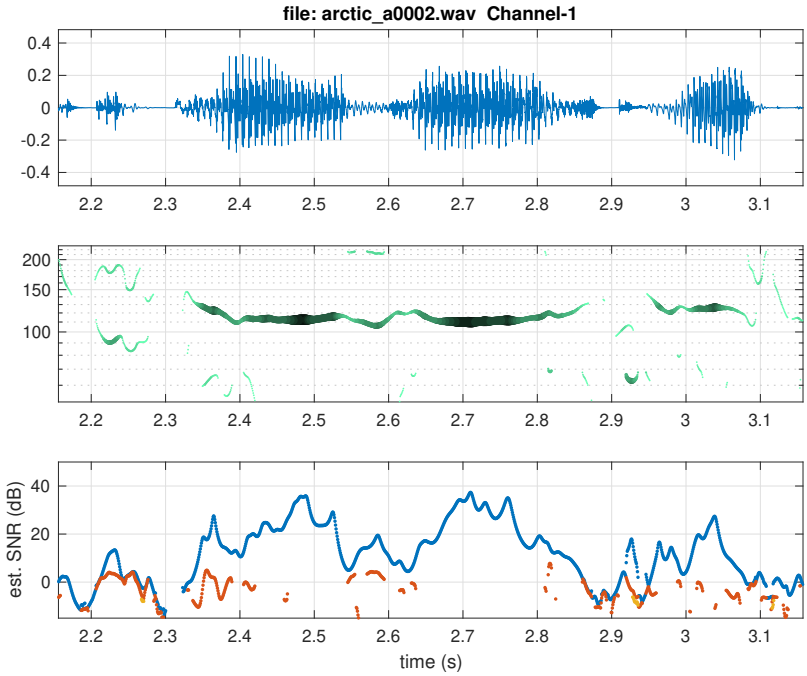

Figure 21: Example output of the interactive GUI tool.

4.6 Extended Capabilities

The interactive GUI tool, `detailInspector.m` provide a speedy way for conducting the detailed analysis. The tool outputs analysis conditions with the report image as a separate text file. You can use it to set detailed parameters for calling this function.

Figure 21 shows the generated report image (EPSF format). The following text is the contents of the associated text file, which consists of the analysis conditions.

```
Report created: 16-Jan-2019 02:07:05
Pathname: /Users/kawahara/Music/CMU_arctic/cmu_us_slt_arctic/orig/
FileName: arctic_a0003.wav
Sampling frequency:     32000.00
Analysis range:    0.05184    3.14819
Donsampled sampling_frequency:      2133.33
lower_frequency:        98.88
higher_frequency:       335.47
stretching_factor:  1.05
channels_per_octave:    12
```

```
wintype: sixterm
```

4.7 See Also

References

[1] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[2] A. H. Nuttall, "Some windows with very good sidelobe behavior," *IEEE Trans. Audio Speech and Signal Processing*, vol. 29, no. 1, pp. 84–91, 1981.

[3] J. Kaiser and R. W. Schafer, "On the use of the I_0-sinh window for spectrum analysis," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, no. 1, pp. 105–107, 1980.

[4] D. Slepian and H. O. Pollak, "Prolate spheroidal wave functions, Fourier analysis and uncertainty-I," *Bell System Technical Journal*, vol. 40, no. 1, pp. 43–63, 1961.

[5] H. Kawahara, K.-I. Sakakibara, M. Morise, H. Banno, T. Toda, and T. Irino, "A new cosine series antialiasing function and its application to aliasing-free glottal source models for speech and singing synthesis," in *Proc. Interspeech 2017*, Stocholm, August 2017, pp. 1358–1362.

[6] J. L. Flanagan and R. M. Golden, "Phase Vocoder," *Bell System Technical Journal*, vol. 45, no. 9, pp. 1493–1509, Nov 1966.

[7] L. Cohen, *Time-frequency analysis: Theory and Applications.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[8] H. Kawahara, "Pitfalls in digital signal processing," *The Journal of the Acoustical Society of Japan*, vol. 73, no. 9, pp. 592–599, 2017, [in Japanese].

[9] J. Kominek and A. Black, "The CMU Arctic databases for speech synthesis," *Proc. ISCA Workshop on Speech Synthesis*, pp. 223–224, 2004.

[10] H. Kawahara, K.-I. Sakakibara, M. Morise, and Y. Ishimoto, "Faster than real-time and audio sampling rate extraction of fo candidates using analytic signals with PSWF and cosine series envelope functions," (Last access: 20/January/2019). [Online]. Available: https://github.com/HidekiKawahara/YANGstraight_source