Institute of Phonetic Sciences,
University of Amsterdam,
Proceedings 25 (2003), 163–170.

# THE INTERACTIVE DESIGN OF AN F0-RELATED SPECTRAL ANALYSER[*]

*Ton Wempe and Paul Boersma*

## Abstract

We present a spectral analysis method that results in formant measurements that are more independent of the fundamental frequency of the signal than several other methods, such as fixed-window Fourier analysis, linear prediction, and interpolated pitch-related Fourier analysis. In order that the reader will be able to replicate our results, we give Praat scripts for generating the test signal, as well as for all the analyses. We show that the Praat program is the appropriate platform for developing, prototyping, testing, debugging, and optimizing speech analysis methods.

## 1 Introduction

For measuring dynamic formant contours, spectrograms ought to have both a good time resolution, but without visible individual periods, and a good frequency resolution, but without visible harmonics of the fundamental frequency (F0). These goals cannot be achieved by the usual technique of using a fixed time window and computing its Fourier transform to obtain the frequency spectrum. If we do not want to see the harmonics, the bandwidth of the window has to be more than F0. This broad-band spectrogram will require a time window much shorter than the period, so that we will see pitch-synchronous changes in the spectrogram. If we do not want to see those, our time window will have to be at least three periods, which will lead to a poor time resolution (e.g. in formant transitions) and to the visibility of the separate harmonics. This paper investigates several possible ways to overcome some of these problems: the LPC spectrogram, interpolated pitch-related DFT, and pitch-synchronous truncated filtering. We use an artificial vowel-like test signal to compare the various methods. The truncated filtering method will turn out to be the least sensitive to F0.

## 2 The test signal

The test signal should be able to illustrate the dependence of measured formant values on F0: as a source signal, therefore, we take a 0.5 seconds long pulse train with an F0

[*] This paper is identical to the paper in the Proceedings of the 15th ICPhS, August 2003, Barcelona, except for the postscript.

linearly falling from 350 to 150 Hz. To compute a vowel-like sound, we convolved this source with the sum of three damped sine waves, whose resonance frequencies were 820, 1300, and 2300 Hertz, respectively. Each of the resonances had a bandwidth equal to 1/12 of the resonance frequency. Script 1 generates this [a]-like vowel. We choose this parallel formant generation because LPC formant analysis methods tend to perform very poorly on such signals. Figure 1 shows a couple of periods of this signal. The vertical dotted lines show the times of the source pulses. The second visible period is slightly longer than the first, as can be seen at the source pulses. The shapes of the two visible periods are slightly different, because the formants of every period spill over into the next.

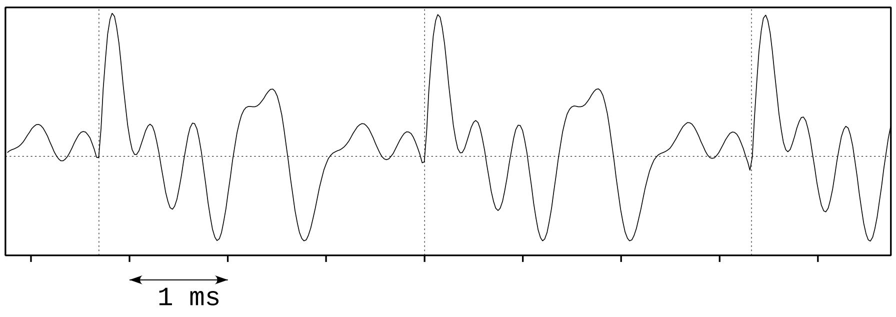

Figure 1: Part of the test signal (0.009 seconds).

## 3 Three mediocre methods

The most widely used method for frequency analysis is the broad-band Fourier spectrogram. We applied this to the test signal in 1-millisecond steps, using a Gaussian window with a half-duration of 5 milliseconds (giving a -3 dB bandwidth of 260 Hz). The result is Figure 2. We see that there are very few times at which the maxima are positioned in the vicinity of the underlying formants. Instead, the maxima tend to ride on harmonics of the F0, and therefore tend to fall as a function of time. At many time points, moreover, we see three or four maxima in the 500–1500 Hz region, rather than two.

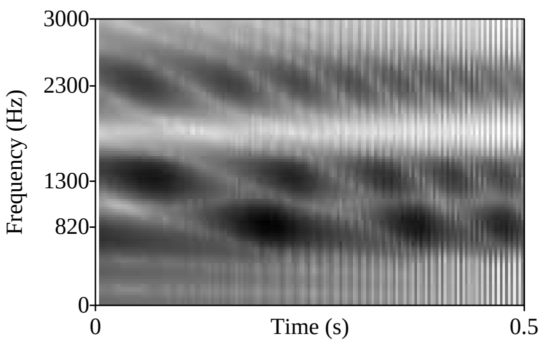

Figure 2: Broad-band Fourier spectrogram.

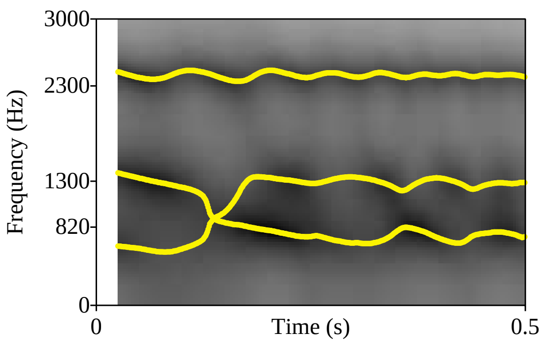

Figure 3: Long window LPC spectrogram.

Another widely used method, especially for finding the positions of vowel formants, is LPC (linear predictive coding) analysis. We did this in Praat by first down-sampling the test signal to 12 kHz, then choosing “To LPC” with 10 poles and a Gaussian window with a half-duration of 25 ms, then choosing “To Spectrogram”. The result is shown in Figure 3, together with the formant contours that Praat derived from the LPC analysis. The formant curves in Figure 3 are seen almost to follow the maxima in the Fourier spectrogram of Figure 2. This effect is less severe for cascaded-formant test signals, but even in that case the resulting formant measurements strongly depend on F0, on the number of poles, on the maximum frequency (6 kHz in this case), and on the degree of pre-emphasis (0 in this case) [1].

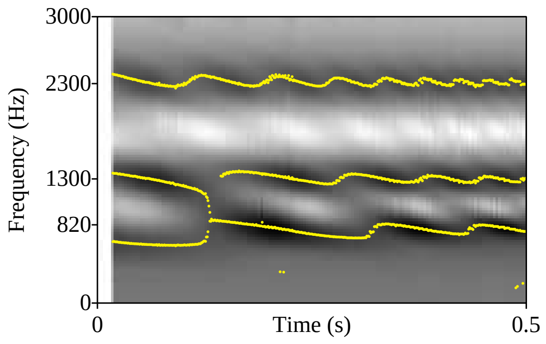

Figure 4:
Interpolated pitch-related Fourier spectrogram.

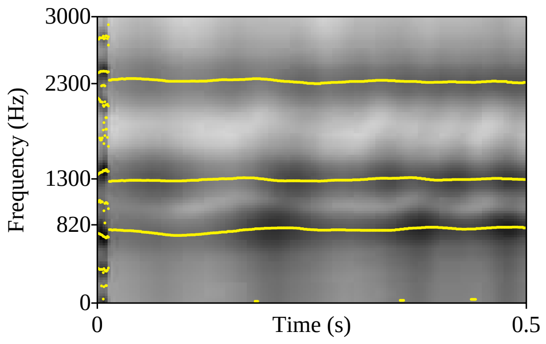

Figure 5:
Truncated-filter spectrogram.

The third method we tried is based on extracting parts from the waveform with a duration of approximately one pitch period, cut at zero crossings. A discrete Fourier transform was computed (giving the energy in each harmonic), then interpolated with third-degree polynomials in the log-power domain. The result is shown in Figure 4. The maxima show almost as much F0-dependent variation as the methods of Figures 2 and 3.

## 4 The truncated-filter method

After seeing the mediocre results of Figures 2, 3, and 4, we conclude that a reliable measurement of the resonance frequencies must be synchronized with the pitch period. Ideally, we would like to excise the periods precisely at the dotted lines in Figure 1. This is because if the signal is seen as composed of three sine waves, these sine waves are not smooth at the transitions between the periods (i.e. at the times of the pulses in the source signal). If we cut out the second visible period of Figure 1 in this way, we get the signal of Figure 6.

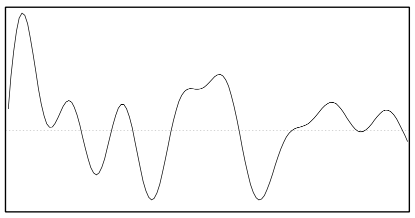

Figure 6:
A single period (3.324 ms) of the test signal.

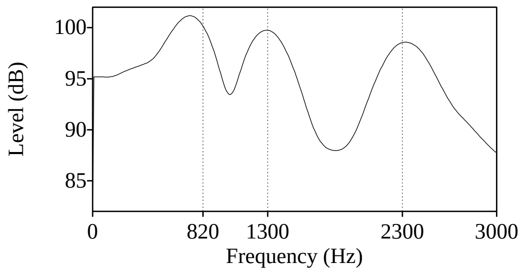

Figure 7:
Truncated-filter spectrum.

On this single period, we run a so-called truncated-filter spectrographic analysis. This method is originally based on a hardware implementation of a segment spectrograph that used a truncated filtering, with damped sine waves as impulse responses [2]. A later version was implemented as a Praat script that used a symmetric second-order filter in the frequency domain [3]. Iterative Praat sessions revealed that the ripple in the low part of the spectrum could be removed completely by using two filters in the frequency domain, one with a damped sine and one with a damped cosine impulse response. A different kind of improvement, namely one that speeded up the computation by a factor of 20, was a time-domain convolution with a damped sine, implemented as a two-sample recursive filter. The ripple that was re-introduced in this way was subsequently removed by including a recursive damped-cosine filter; such a filter can be implemented as the digital time derivative of the damped-sine filter. This final method is presented in script 2. Figure 7 shows the result for the period of Figure 6. The damping was set to zero.

In order to create a spectrogram, it is necessary to find a way to detect all the periods. For the spectrogram in Figure 5, we use a simple heuristic that can be automated in a Praat script: we first look for positive large peaks in the waveform that are approximately on one-period distances from each other, then guess that each underlying pulse lies 0.2 ms before such a peak. Script 3 implements the spectrogram of Figure 5 (in regions where no F0 can be determined, as in the very beginning of the test signal, a fixed window of 10 ms is used). When comparing Figure 5 with Figures 2, 3 and 4, we see that the F0 dependence for the truncated-filter method is much less than for the other methods.

## 5 Real speech

Truncated-filter spectra work well for parallel as well as cascaded formants. LPC spectra work well for cascaded formants only. So how about real speech? Our testing shows that vowel formants are measured accurately if a pre-emphasis filter (a filter with a slope of 6 dB/octave above 50 Hz) is applied prior to the analysis. Such a pre-emphasis filter has traditionally been used for broad-band spectrograms and LPC formant analysis of speech, and turns out to be useful for the truncated-filter analysis as well. Script 3 therefore contains a button for the application of pre-emphasis.

## 6 Interactive design with Praat

The analysis method presented in this paper could not have been developed easily without the use of a dedicated script language. There exist several interpreted script languages that could be useful for developing speech analysis methods, including general mathematical script languages. The script language available in the Praat program [4] has the following advantages:

1. There are handy built-in commands for phoneticians. Script 2 contains a single-line **Edit** command for viewing intermediate results. Likewise, script 3 contains a **Draw** command for drawing each separate spectrum to the Picture window.
2. Debugging commands (like **Edit** and **Draw** above, or the more general **pause**, **exit**, **echo**, and **printline** commands) can be easily commented out by using the semicolon (**;**) as a comment marker. In the style of scripts 2 and 3 this is distinguished from the number sign (**#**) that signals explanatory comments (other commenting-out tricks include the use of **if 0** and boolean flags).
3. In general-purpose mathematical script languages, spectra and sounds are seen as series of samples containing values in arbitrary units, whereas in Praat these objects are known by their physical quantities expressed in Hertz, seconds, or Pascal. This saves lots of conversion code and reduces programming errors.
4. The inputs and outputs of the analysis are Praat's Sound and Spectrogram objects. The script writer need not write any code for handling these data types.

## References

[1] Gautam K. Vallabha & Betty Tuller, (2002): "Systematic errors in the formant analysis of steady-state vowels." *Speech Communication* **38**, 141–160.

[2] Ton Wempe, (1979): "An experimental segment spectrograph based on some notes on frequency analysis of speech segments." *Proceedings of the Institute of Phonetic Sciences of the University of Amsterdam* **5**, 44–102.

[3] Ton Wempe, (2001): "F0-related formant measurements." *Proceedings of the Institute of Phonetic Sciences of the University of Amsterdam* **24**, 167–187.

[4] Paul Boersma & David Weenink, (1992–2003): Praat, a system for doing phonetics by computer. http://www.praat.org.

## Script 1: GeneratePitchSweepVowel.praat

```
form Generate vowel with pitch sweep
   choice Method 2
      button Cascade
      button Parallel
   positive Duration_(sec) 0.5
   positive Initial_F0_(Hz) 250
   positive Final_F0_(Hz) 150
   positive F1_(Hz) 820
   real Amplitude_F1 1.0
   positive F2_(Hz) 1300
   real Amplitude_F2 1.0
   positive F3_(Hz) 2300
   real Amplitude_F3 1.0
   real Formant/Bandwidth 12
```

```
endform

quality = 'Formant/Bandwidth'

# Create voice source signal.
Create PitchTier...  sweep  0.0  duration
Add point...  0  initial_F0
Add point...  duration  final_F0
To PointProcess
To Sound (pulse train)...  44100  1.0  0.05  2000

if  method$ = "Cascade"
    # Filter the sound with the first three formants.
    for  i  to  3
        Filter with one formant (in-line)...  f'i'  f'i'/quality
    endfor
    # Add some extra formants to get a flatter spectrum.
    for  i  from  4  to  9
        Filter with one formant (in-line)...  i*1200-600  200
    endfor
else
    # Construct the impulse response of the filter.
    Create Sound...  filter  0  duration  44100  0
    for  i  to  3
        Formula...  self + amplitude_F'i' * sin (2 * pi * f'i' * x) * exp (- pi * f'i'  /  quality * x)
    endfor
    # Convolve the sound with the impulse response.
    plus Sound sweep
    Convolve
endif

# Make it pleasant for our ears.
Scale...  0.99
```

## Script 2:  truncatedFilterSpectrum.praat

```
form  Truncated-filter spectrum
    real  Filter_width_/_F0  0
    positive  Maximum_frequency_(Hz)  5000
    positive  Frequency_step_(Hz)  10
endform
Copy...  sound
tmin = Get starting time
tmax = Get finishing time
bandwidth = 'Filter_width_/_F0'  /  (tmax - tmin)
fsamp = Get sample rate
Copy...  filtered
nfreq = maximum_frequency  div  frequency_step + 1
Create Matrix...  spectrum  0  maximum_frequency nfreq  frequency_step  0  1  2  2  1  1  0
for  ifreq  to  nfreq
    frequency = (ifreq - 1) * frequency_step
    p = -2 * exp (- pi * bandwidth  /  fsamp) * cos (2 * pi * frequency  /  fsamp)
    q = exp (- 2 * pi * bandwidth  /  fsamp)
    select  Sound  filtered

    # Filter with a damped sine.
    Formula...  Sound_sound [col] - p * self [col - 1] - q * self [col - 2]
    Multiply...  2 * pi * frequency  /  fsamp
    energy_sin = Get energy...  tmin  tmax

    # Debugging trick for seeing the sine-filtered sound.
    ;Edit
    ;pause Frequency 'frequency' Hertz
    # Filter with a damped cosine.
    Formula... Sound_sound [col] - Sound_sound [col - 1] - p * self [col - 1] - q * self [col - 2]
    energy_cos = Get energy... tmin tmax
    # Store the result.
    select Matrix spectrum
    Set value... 1 ifreq sqrt (energy_sin + energy_cos)
endfor
To Spectrum
# The simplest debugging trick:
# exit before the intermediate objects are thrown away
;exit
# Clean up.
select Sound sound
plus Sound filtered
plus Matrix spectrum
Remove
select Spectrum spectrum
```

## Script 3: truncatedFilterSpectrogram.praat

```
form Truncated-filter spectrogram
    positive Time_step_(s) 0.005
    positive Minimum_pitch_(Hz) 75
    positive Maximum_pitch_(Hz) 600
    positive Maximum_frequency_(Hz) 5000
    positive Frequency_step_(Hz) 10
    boolean Pre-emphasize 0
endform
Copy... sound
if 'Pre-emphasize'
    Pre-emphasize (in-line)... 50
endif
tmin = Get starting time
tmax = Get finishing time
fsamp = Get sample rate
To PointProcess (periodic, peaks)... minimum_pitch maximum_pitch yes no
Rename... pulses
numberOfPulses = Get number of points
numberOfFrames = (tmax - tmin) / time_step - 1
nfreq = maximum_frequency div frequency_step + 1
Create Matrix... result tmin tmax numberOfFrames time_step tmin+time_step
... 0 maximum_frequency nfreq frequency_step 0 0
for iframe to numberOfFrames
    # Select one period.
    tmid = tmin + iframe * time_step
    select PointProcess pulses
    highIndex = Get high index... tmid
    if highIndex = 1 or highIndex > numberOfPulses
        pitch = undefined
    else
        begin = Get time from index... highIndex-1
        end = Get time from index... highIndex
        pitch = 1 / (end - begin)
        if pitch < minimum_pitch or pitch > maximum_pitch
        pitch = undefined
        else
        begin -= 0.0002
        end -= 0.0002
        endif
    endif
    if  pitch = undefined
        begin = tmid - 0.005
        end = tmid + 0.005
    endif
    select  Sound  sound
    Extract part...  begin  end  Rectangular  1  yes
    Rename...  period

    # Compute spectrum.
    execute  truncatedFilterSpectrum.praat 0  'maximum_frequency'  10

    # Show progress.
    ;Erase all
    ;Draw...  0  0  0  0  yes
    ;Text top...  yes  Frame 'iframe', pitch 'pitch' Hz

    # Store results.
    select  Matrix  result
    for  ifreq  to  nfreq
        Set value...  ifreq  iframe Spectrum_spectrum [1, ifreq] ^ 2
    endfor

    # Clean up.
    select  Sound  period
    plus  Spectrum  spectrum
    Remove

endfor
select  Matrix  result
To Spectrogram
```

## Postscript

The formula for the in-line filter with the cosine impulse response:

```
    Formula... Sound_sound [col] - Sound_sound [col - 1] - p * self [col - 1] - q * self [col - 2]
```

(script 2, under “# Filter with a damped cosine”) is, strictly speaking, not correct. Our Institute's mathematician, Jan van Dijk, provided us with the mathematically correct formula which results into the following code as a replacement:

```
if frequency = 0
        b = 0
        c = 0
        bb = 0
else
        epsilon = 0.5 * bandwidth  /  frequency
        b = 2 * exp (- epsilon * omega * tsamp) * cos (omega * tsamp)
        c = exp (- 2 * epsilon * omega * tsamp)
        bb = exp (- epsilon * omega * tsamp) * (cos (omega * tsamp) + epsilon * sin (omega * tsamp))
endif
Formula... 0.5 * Sound_sound [col] - Sound_sound [col-1] * (bb - 0.5 * b) - Sound_sound [col-2] * 0.5 *
... c + (b * self [col-1] - c * self [col-2])
```

In practice, however, the difference is negligible.