Supplementary notes on digital signal processing (Part 1 Digital signals and DFT). (Kent and Read pp. 61-84 Hayward 66-76,).

1) Analog and digital signals.

Analog or **continuous** signals are what we find in the real world. They are **continuous** in both time and in amplitude. That is, a waveform of an analog signal is defined at all amplitude values and at all time values. Digital signals are *not* continuous, but **discrete**, both in time and amplitude. They are defined only at times that are integer multiples of the sampling interval Ts. { 0 Ts, 1 Ts, 2Ts, 3 Ts, etc. } Furthermore, the amplitude values of the continuous signal are mapped into a finite range and "rounded off" to integer values.

In a typical application, the effective range of amplitudes that a given set of A/D converters might handle may be limited to ±10 Volts.). An amplitude of -10 volts or smaller on input to the A/D would read 0 on the output. It is more common now for the digitized numbers to be at least 16 bits with digits coded as both positive and negative values in the range of about -32, 767 to +32,767. WaveSurfer gives numbers in that range. Standard digitized files, like Microsoft .wav files, often code back to real numbers in the range of -1.0 to +1.0.

2) Implications of the Nyquist theorem.

The Nyquist theorem says that if we are to adequately represent an analog signal, we must choose a sampling frequency F_S (= $1/T_S$) that is **at least twice** as high as the highest frequency component in our original signal. What happens if we don't do this? Then frequencies in our original signal that were higher than the Nyquist frequency , F_n= 1/2 F_S are "aliased": They look identical to signals with other frequencies *within* the Nyquist range (0 to F_S/2). How are they aliased? The diagram in Figure 1 discussed in in class shows how it works.

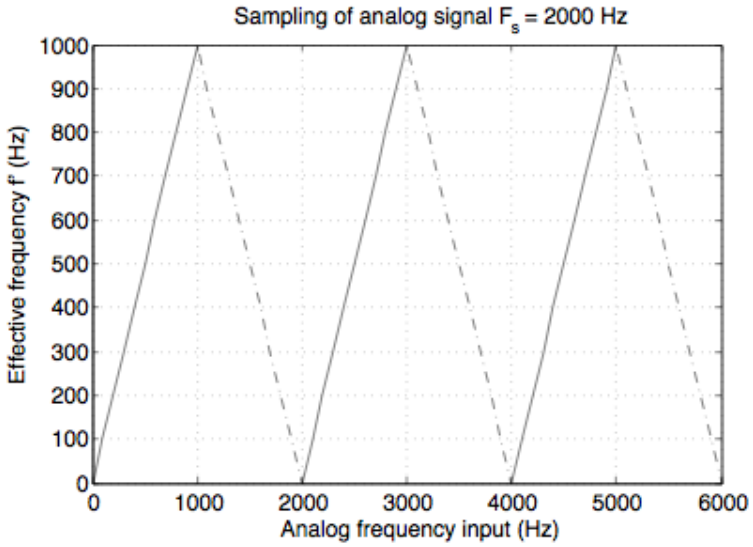


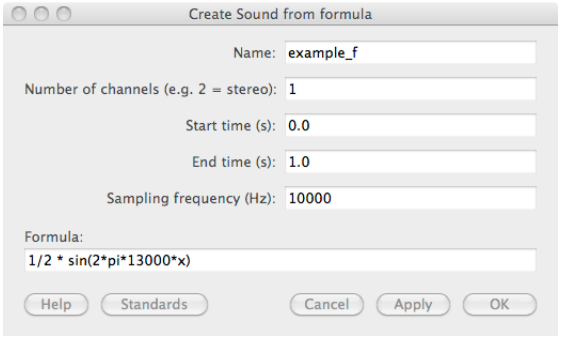
Figure 1. Illustration of aliasing. The effective frequency of an analog signal on the *x*-axis is given by the *y*-coordinate of the 'ribbon'.

Mathematically, it works like this. Call our original frequency f and call the resulting apparent frequency after digitization f*. Define k as the integer part of f/ F_n. So, if f is less than F_n, k is zero. If $f = F_S + 1$, $k = 2$; k effectively counts the number of bends in the ribbon counting from $f = 0$ on the x-axis. If k is **even** (including zero), we are operating on the solid lines of the ribbon plot, and $f^* = f - k F_n$. If k is odd, we are operating on the dashed lines of the ribbon plot and $f^* = (k+1) F_n - f$.
The key point is that below the Nyquist frequency, all is well; above it, things get squirrelly.

Examples.

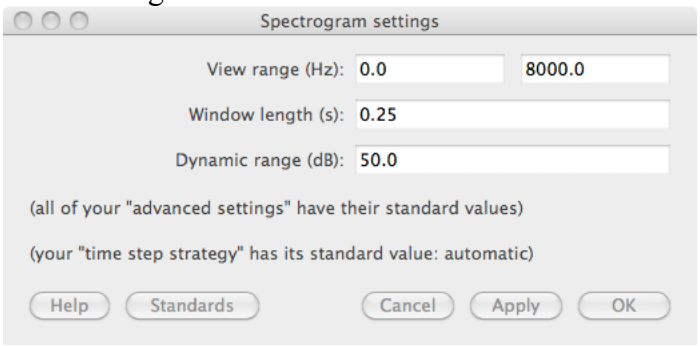
F_S= 10000 Hz. Calculate effective f* for the following list of analog f values. a) 1000; b) 4000, c) 5000; d) 6000; e) 9000; *f*) 13000; g) 19000 (all in Hz).
For the first three k=0 and $f^* = f$ so a) f*=1000; b) f*=4000; c) f*=5000.
For d), k= integer part of (6000/5000) = 1 which is odd, so $f^* = (k+1)\ x\ F_n - f$
= 2x5000-5000= 4000 Hz.
For e) k= integer part of (9000/5000) =1, which is odd, so $f^* = (k+1)\ x\ F_n - f$
= 2*5000-9000= 1000 Hz
For *f*) k= integer part of (13000/5000) =2, which is even, so $f^* = f - k F_n$
= 13000-2x5000=3000 Hz;
For g) k= integer part of (19000/5000) =3, which is odd so $f^* = (k+1)\ x\ F_n - f$
= 4 x 5000 - 19000 =1000 Hz;

Praat exercise:
Test the formula above by using a variant of the following ***Praat:New: Sound:Create sound from formula:***
Type the following into the formula
1/2 * sin(2*pi***13000***x)
of form window to check **example f)** above (13000 sinusoid sampled at 10000 Hz)

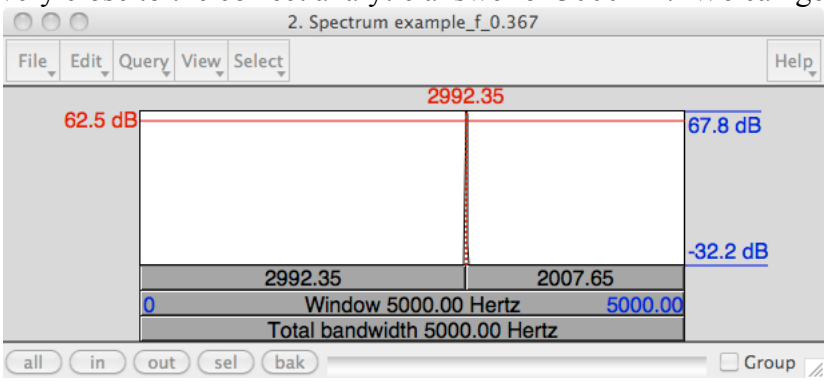


Make sure to get rid of the noise and to change both the frequency in the sin(...) and the sampling frequency.
Create the resulting sound and edit it.
To get a reasonably accurate frequency estimate, you need to

change the spectrogram settings to get a very narrow band analysis by making the window length 0.25 seconds.



Put the cursor in the middle of the resulting narrow stripe in the specrogram to measure it. Better yet, take a spectral section(Cntrl+L on PC or Cmnd+L on Mac). Here's mine. You can see we're very close to the correct analytic answer of 3000 Hz. We can get closer in Praat by changing the View Range to 2800 to 3200 for example.



A note on the motivation of this part of Ling 512. Some students part of the course is especially boring. It just is. It nonetheless may be *the most important part* for those if you who may go on to use acoustic phonetics. We need it to understand what nifty programs like Praat and WaveSurfer are doing. We also may need to know how to tweak their analysis parameters to get them to do what we want (or know that they just can't do what we need for a given problem). To do this we will consider a method for converting between things like 'window length (pts)', 'frame advance interval (pts)', DFT length (pts) all measured in sampled data points. By means of a set of formulae given in handouts, we can convert back and forth from these discrete counts of lengths in points to things of direct relevance to acoustic analysis, such as ΔF, the frequency spacing of filters (Hz); BW, the bandwidth of filters (Hz); $D_{Adv(ms)}$, the frame advance time step (ms) measured in 'real world' units. At the time of writing, every program I've ever used has made different choices about the types of control parameters it uses. For example Praat gives most of its parameters in terms of Hz for frequency or seconds for time and works out sample point lengths from those. WaveSurfer (in its default demo configuration at least), relies pretty heavily on the specifications of lengths in points and only implicitly defines other things. This course will show you how to compare at least some of the parameters in Praat to those in WaveSurfer and vice versa.

3) Amplitudes of DFT's as Filter bank outputs.

A signal that is N points long, if subjected to a process called a Discrete Fourier Transform (DFT) can be viewed as producing a set of amplitude coefficients and phase coefficients of the following form.

$$A_i \sin(2\pi f_i/F_s\, t(n) + \Phi_i),$$

where A_i can be viewed as the amplitude of the output of the i^{th} filter; $t(n)$ is the time (in seconds) of the n-th sample for n = 1 to N (and Φ_i is an associated starting phase, which is not of much interest to us.) In our discussion, the index i ranges from 1 to L where L = ceiling(($N+1$)/2). (Ceiling is the 'round up to next integer' function. Ceiling(1) = 1 and Ceiling(2.001) = 3; Fi

Technical aside

A little more accurately, a DFT can be represented as follows.

We can describe a digital signal as a sequence of y(n) points n=1 to N as follows

$$y_n = \sum_{i=1}^{N} A_i sin(2\pi f_i/F_s t(n)/t(N) + \phi_i)$$

only here the frequency bin index ranges from 1 to N instead of 1 to L. The last roughly L of these coefficients from N-L+1 to N will have A_i be a kind of mirror image of the first L A_i amplitude values and the corresponding f_i phase values will be minus the value of their 1 to L counterparts. The ratio t(n)/t(N) is the relative time of point y(n) in the sequence. t(n)= 0 seconds and t(N) = (N-1)/Fs seconds. In filter bank interpretation of real signals, the part of the analysis above L points is irrelevant and we won't talk about it much if ever. Similarly, the ratio f_i/F_s represents the relative frequency of the i-th filter compared to the sampling rate. The f_i are evenly spaced starting f_1 = 0 Hz with steps of F_s / N as described below. A DFT can be implemented by using linear regression of the sequence π

Note (to self mainly) The Latex formula for this: y_n= \sum_{i=1}^N A_{i} sin(2 \pi f_{i}/F_s t(n)/t(N) +\phi_{i}). See the very nice web site **http://www.sciweavers.org/free-online-latex-equation-editor.** You can use these in you homework if you want if you paste in the graphic in Word or other document processor that allows you to paste graphics.

The DFT as we'll use it provides us with the equivalent of a "bank" of L filters. The center frequencies of each of those filters is denoted by f_i and does not depend on anything but N, *the length of the sequence on which the DFT is performed*. (In this case the "input sequence" is just the N points of the original signal; this can be modified. See section 4). The center frequency of the i-th filter is given by the following relation: $f_i = (i\text{-}1)\, F_s / N$. The spacing of center frequencies of adjacent filters is ΔF which is equal to F_s/N. **Example**. If F_s is10000 Hz and N is 100, then there will be 51 filters (100/2+1). The spacing of the filters, ΔF, is F_s/N= 10000/100 = 100 Hz. The first filter will have a center frequency of 0 Hz, the second 100, the third 200, the 50-th 4900, and the L-th (= 51-st here), 5000 Hz.

4) Bandwidths of DFT Filter bank analyses.

The analysis in the section 3 above uses as the input to the DFT a sequence of N points from the original signal. When this is done, the bandwidth of each filter is the proportional to as the frequency spacing BW = k_{win} (F_s/N), where k_{win} is a constant asociated with a window function. For the simplest case, a rectangular window is used, meaning that the signal is not modified., k_{win} is about 0.88, so = BW= 0.88 F_s/N. The low frequency cutoffs (f_{lower}) of each filter is 1/2 BW lower than the center frequency and the upper cutoffs (f_{upper}) are 1/2 bandwith. In the case discussed above with a rectangular

window, the BW is about 88 Hz, so the low frequency cutoff (3 dB point) is 44 Hz less than the center frequency (f_i for the i-th filter); and it is 44 Hz more than f_i for the upper cutoff.

Adjusting bandwidth by windowing and filter spacing by padding. By modifying the input sequence, it is possible to "delink" analysis bandwidth from the spacing of the filters in such a way as to get narrower spacing between adjacent filters while maintaining a relatively wide bandwidth of analysis filters. One way to do this is to define a modified input sequence that consists of M points of the original signal followed by P zero values. The length of the DFT used is then $N = M + P$. For example, suppose we wanted to have more filters in the analysis of the signal in section 3. We could take the 100 points (M = 100), add 900 zeros to the end (P = 0) and do an analysis of the $N = M+P$ points. In this case that gives us 100 + 900 =1000 points. [1] The rules about number of filters and filter spacing from section 3 still apply. That is N=1000; filter spacing, $\mathbf{\Delta F}$, is F_s/N = 10000/1000 = 10 Hz. The number of filters $L = N/2 + 1 = 1000/2 + 1 = 501$. The first filter is centered at zero, the second at 10 Hz, the third at 30, the (L-1)th (number 500) at 4990 Hz and the L^{th} at 5000. The bandwidths are now determined by M, rather than N. (They were in section 3 as well, but the in that case M was the same as N, there were no extra padding zeros). The bandwidths, are given by $\mathbf{BW} = k_{win} \times F_s/M$. Here, M = 100, so the bandwidths are still about **88** Hz as in section 3. In the new example of section 4 (M=100, N=1000), the pass bands of adjacent filters overlap substantially. For example, filter 11, centered at 1000 Hz has f_l= 1000-88/2 = 956 Hz and f_u= 1000+88/2= 1044 Hz. (The lower cutoff f_l will be 1/2 BW below f_c and the upper cutoff, f_u ,will be 1/2 BW above f_c). Thus its pass band is 956 to 1044 Hz. The next filter (number 12) has f_c of 1010 Hz and has a pass band of 966-1054 Hz. There is thus a 78 Hz overlap in the pass bands of the two filters. This is equal to the difference between the upper cutoff of filter 11 lower cutoff of filter 12 and (1044-966 = 78). It is also simply the difference between the bandwidth and the filter spacing, i.e. BW- ΔF. In this case that's 88 Hz - 10 Hz = 78 Hz. This kind of overlap can actually be very useful if DFT's are to be used for spectral displays, as shown in the figures below.

5) Window shapes and bandwidth.

A process called windowing can be used to produce better behaved filters in DFT analysis. Suppose we are interested in M points of a signal, e.g. the 100 points of the previous example. We can define this signal as the set of points $x(i)$, i=1 to 100. We can window a signal of M points by multiplying each sample of the signal by a corresponding point in the window. That is we define a "pseudo signal" $w(i)$, i =1 to M and we produce a new signal $y(i) = x(i)\, w(i)$, i.e. the point by point product of the two sequences (the signal and the window). If $w(i)$ is a sequence of M ones, (i.e. $w(1)$=1, $w(2)$=1, $w(3)$=1....$w(M)$=1), then we call the window a "rectangular window" (that is, as a flat topped plateau that can be viewed as dropping off to zero when $i > M$ or $i < 1$). It is advantageous, however, to use a different kind of window for most spectral analysis. Roughly speaking, these could be called "tapered" windows. They typically start at or near zero at $w(1)$, gradually raise to a value near 1.0 near the middle of the window $w(M/2)$ and then drop back down to near zero at $w(M)$. The most common type of tapered

window used in speech processing is a Hamming window (named after an engineer at Bell Laboratories).

What does such tapering do? Well, we haven't told you everything about DFT filter banks. It turns out, if you use the procedure of section 4 and just add a bunch of zeros to a signal, you produce a series of filters that has a lot of what is called stop-band ripple. The stop band of a band-pass filter is everything above or below its pass band (below f_l and above f_u). Although the center frequencies of and bandwidths of the DFTs are as advertised in section 4, the pass-band ripple is pretty bad. This ripple causes a single sine wave to have some non-negligible effect on filters with center frequencies quite remote from itself.

Figure 2 illustrates what happens if we look at 100 points of a **1000** Hz sine wave through a DFT with a rectangular window (no change to the 100 points), versus one with a 100 point Hamming window (F_s here= 10 kHz). The pass band around 1000 Hz is wider in the second row of Figures 2 and 3 with a Hamming window than that of with the rectangular window of row 1. But for row 2, the stop band ripple is much lower in amplitude, more than 40 dB below that of the center frequency. Thus the bandwidth of DFT analysis depends on both the window length M being analyzed and the shape of the window applied. (The spacing of the filters depends only on N, the total length of the windowed signal and any padded zeros). For a Hamming window, k_{win} is about 1.3 so the formula for the bandwidth is approximately $1.3 \times F_s/M$. So for F_s=10000, M=100, the bandwidth would be $1.3 \times 10000/100 = 130$ Hz.

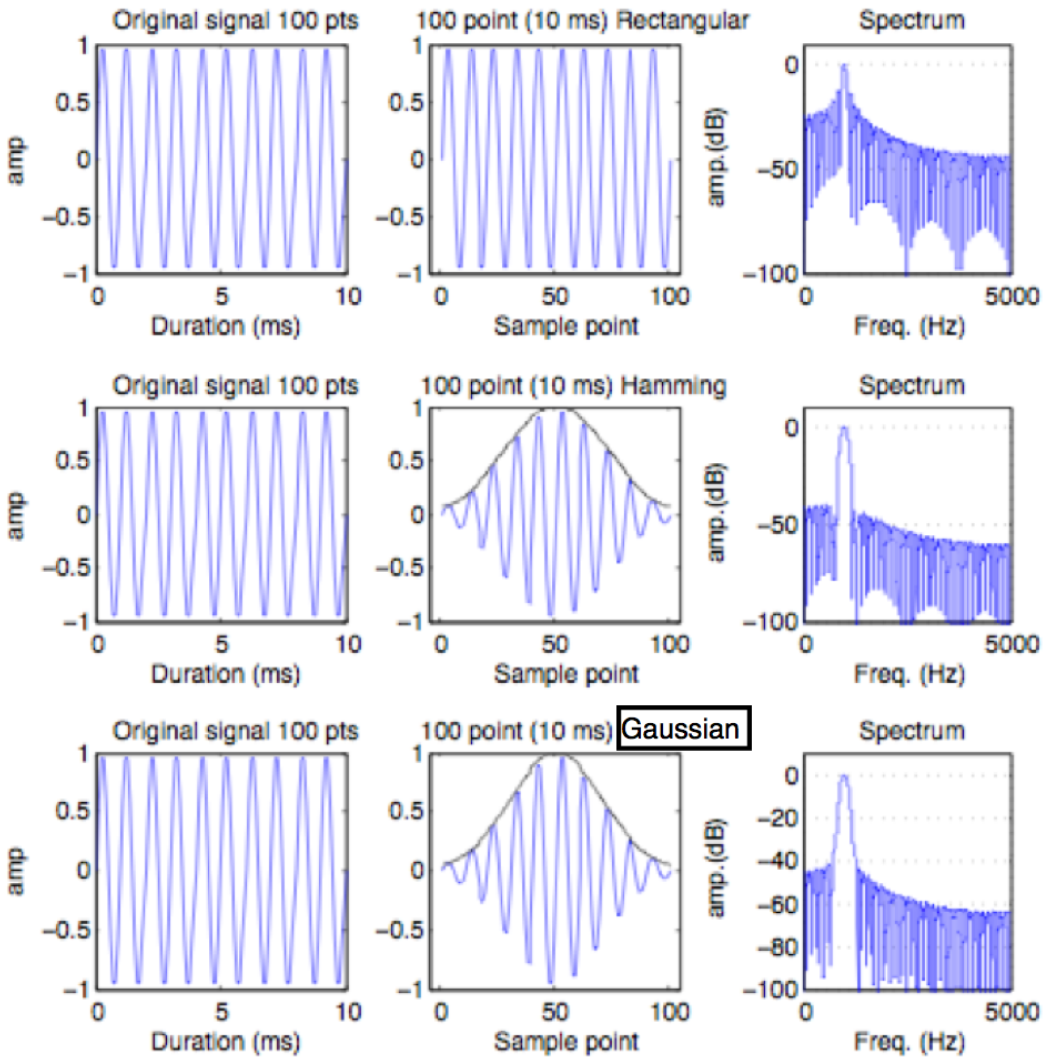


Figure 2: The *left hand panel* of each row shows the original signal, consisting of the first 100 points of a 1000 Hz sine wave. For each row, *the central panel* shows the right panel windowed by a 100 point window of the type indicated. The right panel shows the dB spectrum obtained via a DFT of the central panel of the same row. To get an interpolated spectrum, the 100 point sequence was 'padded' with extra zeros to make a total FFT sequence length of 8172 points. Window length, M = 100 points, F_s= 10000 Hz.

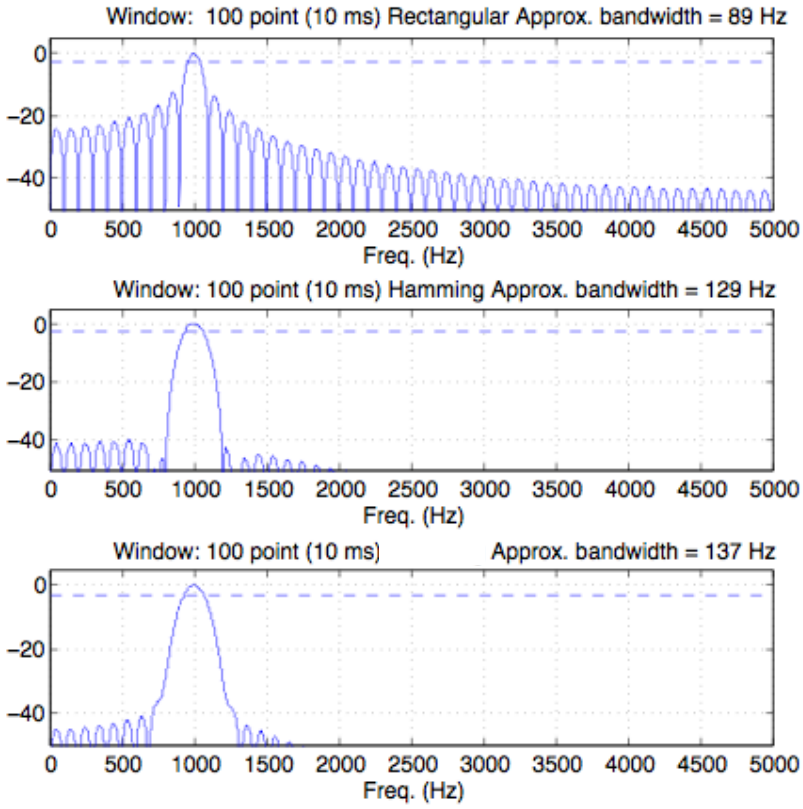


Figure 3. Rightmost panel of figure 2 shown in more detail.

Another kind of window, the Gaussian window, shown in the third row of figures 2 and of figure 3 is one favored by Praat. It has a k_{win} of about 1.4. Its passband is wider, but it has very much reduced stop-band ripple.

Praat DEMO:
The program DFT_WindowingDemo2011.praat allows you to see what happens when you analyze one or two sinusoids with different combinations of window shapes and padding.

5. Illustration of DFT, windowing, and padding
Here are some very old Matlab generated pictures related to windowing. To analyze harmonics components of a vowel, we need at least two complete periods. In Figure 4 below, panel (A) shows $M = 200$ points, or 20 ms of a vowel [ae] spoken a female speaker ($F_s = 10{,}000$ Hz.) By inspecting the waveform, we can determine that the period is about 45 sample points or 4.5 ms. This corresponds to an F0 of about 220 Hz. Panel B shows a rectangular window (not very interesting, is it) of 200 points length. Panel C shows what happens when we multiply point for point the signal in A with the window in B. We get A back again. Panel D is the DFT of the windowed signal in C. It's pretty hard

to pick out the harmonics, the filters are spaced 50 Hz apart. The bandwidth for a rectangular window of length M is about $.9 \times M/F_s = 45$ Hz.

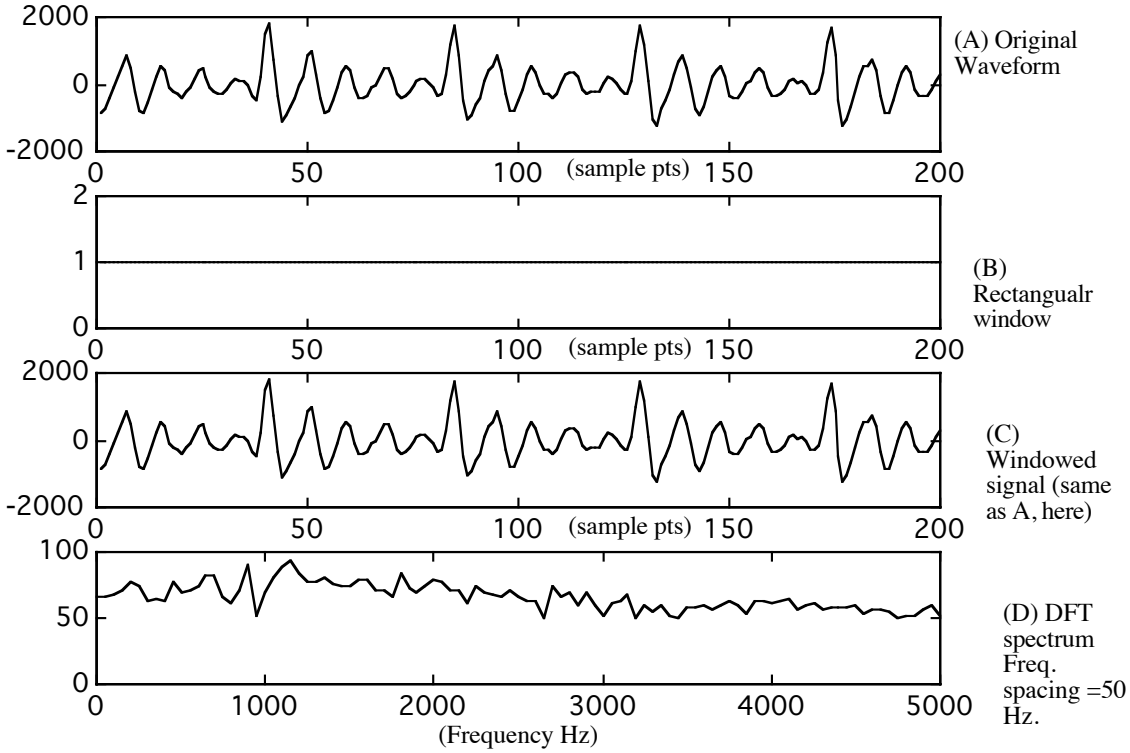


Figure 4. Analysis of 200 ms of a vowel.

Figure 5 shows how we can use padding *to decrease the spacing between filters* in the DFT. We take the same signal as in Figure 1 in panel (A) M=200 points. We multiply it by a rectangular window for 200 points again. But now we add P=824 extra zeros (padding with zeros) to the end of the signal and to the end of the window. Then multiply point by point. This gives us panel C, which is like signal in A, but the time axis is squished and extra points are added. We now get a better picture of the harmonics, because the spacing of the filters, ΔF, in the FFT is $F_s/N = 10000/1024$ or a little less than 10 Hz. The effective bandwidth of the filters, however is determined by the length of the non-zero part of the signal (M) in C and the window type. Here, with a rectangular window, BW= $.88\ F_s/M = .88 * 10000/200 = 44$ Hz. The harmonics are a little more prominent.

Finally, **Figure 6** shows how we can use "padding" to decrease ΔF, the spacing between filters in the DFT, while we can use Hamming windowing to produce a cleaner estimate of the harmonic structure. We take the same signal as in Figure 1 in panel (A) M=200 points. We multiply it by a Hamming window for 200 points . We now add 924 extra zeros (padding with zeros) to the end of the signal and to the end of the window. Then multiply point by point. This gives us panel C, which is like figure 5, but with a Hamming window applied to the signal before padding.

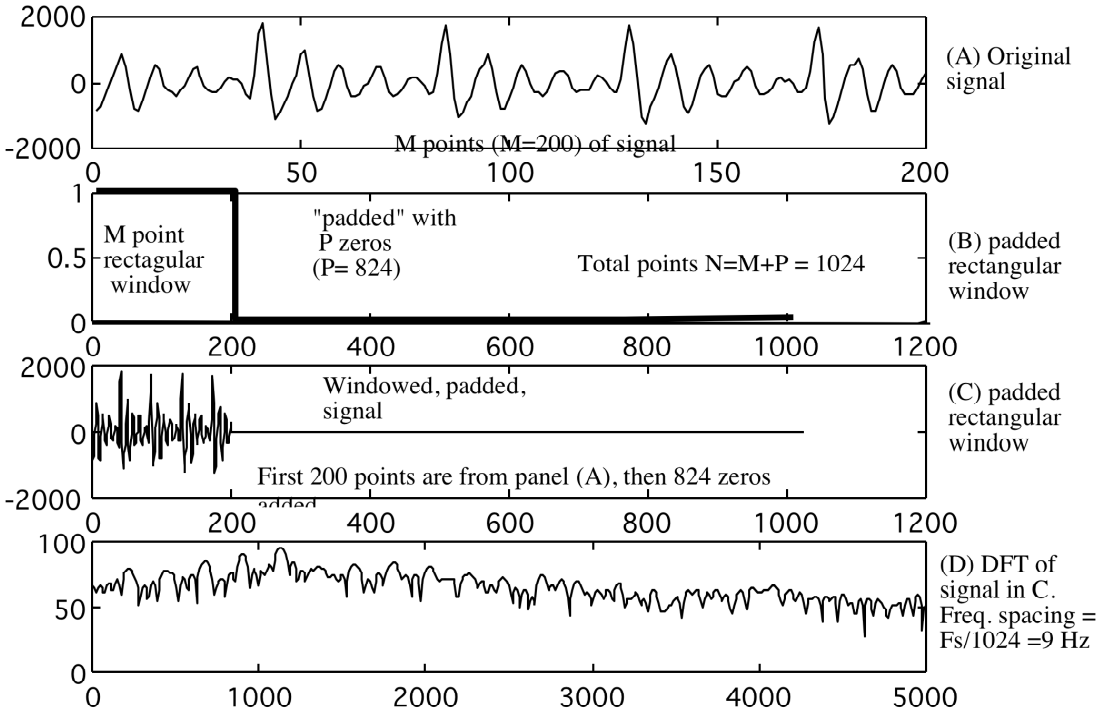


Figure 5. Analysis of same vowel signal as in figure 4 with padding to interpolate the spectrum.

We now get a better picture of the harmonics, because the spacing of the filters in the FFT is F_s/N = 10000/1024 or a little less than 10 Hz. The effective bandwidth of the filters is determined by the length of the non-zero part of the signal in C and by the window type. Since it is a Hamming window, BW= $1.3 \times F_s/M$ = **1.3 ×** 10000/200 = 65 Hz. The harmonics are very prominent, though the peaks are now wider. As a tradeoff, we get less garbage between the harmonics, which are "stop band artifacts" in rectangular window.

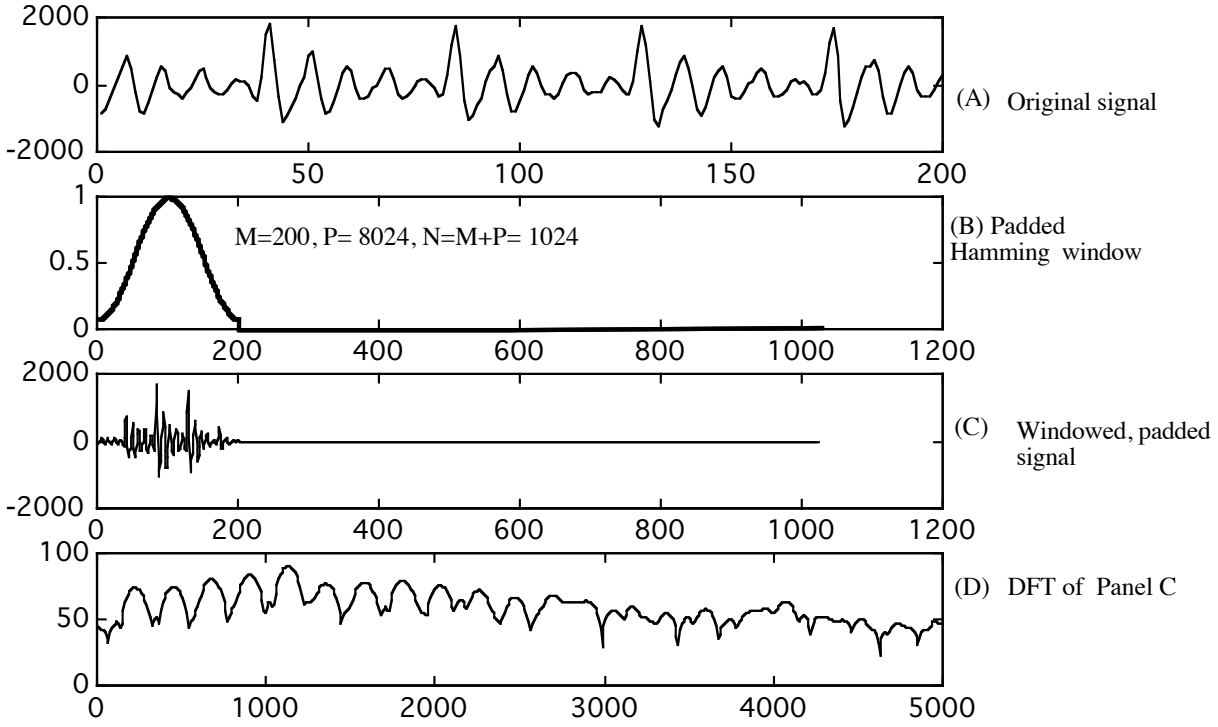


Figure 6. Analysis of same signal as Figure 3, but with a Hamming window and padding. Harmonics are much easier to locate, though their peaks are wider.

Some notes on spectral sections in WaveSurfer and Praat.

WaveSurfer

In the demonstration layout, WaveSurfer's (up to version 1.8 at least) *spectral section* menus allow you to change window type and window length, *M*. The FFT length *N* is always the same as the window length *M*. That is, WaveSurfer's section plots do NOT allow you to specify zero padding. This means that at short window lengths, you get a very grainy spectrum. You can use the 'export' button to output the results to a file.

Praat

Praat allows more flexibility with spectral sections (slices) obtained when viewing a sampled signal via the Praat signal editor. However, user control of the window length is is rather indirect and difficult to figure out from the documentation. You must select the chunk of speech you want in an Edit pane (where waveform and spectrogram are displayed), then from the File dropdown of the Edit pane, choose *Extract windowed selection*. To A pop-up box allows you to choose a window shape and to manipulate the padding indirectly (as a fraction of the duration of the signal selected). When you've made your choices, the windowed section of the waveform will be on the list in the *Praat objects* pane with the default title "Sound slice". You can then select the new sound slice and choose *Analyze: Spectrum-:To Spectrum*. A minimal pop-up box will be appear. If you leave the *Fast* box, checked you will get an FFT padded to next power of two. If you uncheck the box a DFT of exactly the window length will be performed. The resulting spectral section ('slice') will be in a new object called "Spectrum slice" It is safest to

examine the spectral slice using Praat's inspect button at the side of the *Praat objects* pane.
If you want to pad the windowed signal with exactly M points, you can do it, but it's tricky. You can create a sound from formula of exactly the length you want but give it zero amplitude. (the formula "x*0" will do, choose the End Time of the signal to be M/Fs samples long. You can then
1) edit the new null signal
2) copy it into the clipboard,
3) then edit the windowed signal you produced
4) and paste the null signal in the clipboard to the end.

If you study the Praat script DFT_WindowingDemo2011.praat and see how one way it can be done in a script. There are undoubtedly other better ways.