Short-time analyses See also class slides on these topics.
Digital spectrograms, autocorrelation, LPC and related methods

Most speech software relies on *short-time analysis methods*. Digital spectrograms, LPC analysis (for formant estimation) and autocorrelation for pitch detection are important examples of such methods. These all involve the following 4 steps.
1) *Selecting* a segment called an *analysis frame*, of a larger signal.
2) *Windowing* the section chosen, usually with a Hamming, Hann or Gaussian window. This is sometimes followed by *padding* the frame with zeros in spectral analysis.
3) *Performing an analysis* on the windowed (and padded) frame and
4) *Advancing* the analysis frame to select a later chunk of the signal.

I usually work these problems in milliseconds, though it's always possible to use seconds. One quantity it is useful to calculate if you're working in is
$\boldsymbol{N_{spms} = F_s / 1000}$. This is the number of samples per milliseconds. It's good to keep this number to at least 2 decimal places. The analogous number, the number of samples per second could be called N_{sps} is just the sampling rate F_s. Note we will often use the short form F_s instead of the more descriptive F_{samp} for brevity in most of the rest of the course.

There are **two key properties** that apply to all kinds of short-term (short-time) analysis:
a) The **frame length** or **window length Dwin, which determines what time interval of the signal,** is selected each analysis frame. Frame length can be measured in milliseconds or in sample points. *One sample point corresponds* $\boldsymbol{T_s = 1/F_s}$ ***seconds*** or $\boldsymbol{T_{ms} = 1000/F_s}$ *ms.* Note the latter is the number of milliseconds per sample, so it is just $1/N_{spms}$ above. The conversion from length in points to duration in ms is $\boldsymbol{D_{win(ms)} = 1000/F_s \cdot M_{win}}$, where M_{win} is the same as M on the previous handout, the length of the window in points. (Dwin(s) = Mwin/Fs. Frame length has other names, including window length, frame duration, window duration.
b) The **frame advance interval or time step** determines how much is the analysis frame advanced at each step. The frame advance interval can also be measured either in milliseconds $\boldsymbol{D_{adv(ms)}}$ (or in seconds $\boldsymbol{D_{adv(s)}}$) or in sample points $\boldsymbol{N_{adv}}$. The frame advance is sometimes called **hop-size** because the analysis frame 'hops' through the signal by this amount. This can be measured in time units as D_{adv} (in seconds or ms) or in points as N_{adv}. If D_{adv} is measured in milliseconds, then the conversion is $\boldsymbol{N_{adv} = D_{adv(ms)} N_{spms}}$.

An equivalent, though less frequently encountered concept is **frame-rate**, measured in Hz, which is the number times a frame must advance to cover a whole second. You can convert back and forth from frame rate in Hz to a frame advance interval in ms by the same formula as converting a fundamental frequency to a period. If $D_{adv(ms)}$ is the frame advance interval in ms, then the frame rate F_{adv} is $[1000/ D_{adv(ms)}]$ Hz.

As in DFT analysis, different software systems express things like frame length and frame advance in different ways and using variations in the terminology. They all involve the same basic elements. Praat and WaveSurfer terminology will be discussed in class and summarized on a handout.

Digital spectrograms (H 49-53, 71-76; KR 77-85; J 33-45)
One application of short-time analysis is in the production of digital spectrograms. Proper choice of frame size, frame advance rate, can greatly affect the suitability of the analysis

produced for different purposes. Frame advance interval and frame duration should usually be chosen to match. For most processing, (e.g., with a Hamming window), frame advance interval should be *no more than* about 1/4 of the frame duration, So for a 25 ms window, the advance rate should be no more than about 6.25 ms. That means *adjacent frames overlap by about 3/4*. If this is not done, the spectrogram drawn can be quite grainy.

Similarly, filter spacing (frequency steps) and filter bandwidth discussed on a previous handout should be chosen to match. You typically want to have some overlap in the skirts of the filters. Filter frequency spacing (**ΔF**) should be no more than 1/4 of the analysis bandwidth for most applications. These values should be exceeded if you want the best looking spectrograms. Recall that the bandwidth is determined by the window type (Hamming, Gaussian, etc.) and the frame length.

There's no real harm in doing even smaller time or frequency steps.1/4 of the frame advance and 1/4 of the bandwidth are usually adequate. But you'll see some improvement with smaller time advances and frequency steps. At some point, however, you'll just be wasting time (the computer's and more importantly yours). The Praat help page *Sound:To Spectrogram* suggests that there is no reason ever to advance time step by less than about 1/14 or more precisely $1/(8 \cdot \sqrt{\pi})$ or the frequency step by less than about 1/14 of the bandwidth.

Digital spectrogram: Example computation
Suppose I want to produce a digital spectrogram in a certain program. This program assumes a sophisticated user who will be able to specify frame advance and window length in sample points and also the number of points the want for the FFT[1]. (WaveSurfer lets you control spectrograms this way, sort of). I want to produce a 'standard' broadband spectrogram with a bandwidth of 300 Hz. I also want to produce as high a quality spectrogram as I can. Assume that the screen area to be devoted to the spectrogram is only 500 x 500 pixels (little dots on the screen that make up the picture) and I want every pixel to count. Assume I the target signal is 1000 ms in duration sampled at F_s = 10,000 Hz. What do I tell the program to do?

Solution
First it's a good idea to write down the *sampling rate Fs* =10,000 Hz, and immediately calculate the *sampling interval* T_{ms} = 1000/*Fs.* In this case that's 1000/10,00 0= 0.1000 ms (per sample). Its inverse, the number of samples per ms, N_{spms} =Fs/1000 is then 10.0 samples per ms. For accuracy, the sampling interval should be kept to four decimal places and N_{spms} should be kept to one decimal place

Frame advance: Although one can determine a reasonable frame advance from the frame length, in this case we want to have as detailed picture as we can. If time is no object, and I want every pixel to count, I want to have a frame advance that works out to one pixel wide. That can be determined by the length of the signal divided by the number of pixels devoted to the time axis, 500 points. (Note for a spectrogram the time axis will be the left to right, frequency up and down.) $\boldsymbol{D_{adv}}$ = 1000/500 = 2 ms frame rate advance. (There's no point doing more in this particular application, because the screen can't display more points on

[1] WaveSurfer has spectrogram controls that allow you to control these numbers in principle. However, the display on the screen uses a large value of N_{dft} than what you set in the controls.

the time axis.) I can convert that to sample points using $D_{adv} \cdot N_{spms} = D_{ms}/T_{ms}$ so that gives me $\boldsymbol{N_{adv} = 2 \cdot 10 = 20}$ sample points.

FFT size. We have 500 vertical pixels for the frequency axis and would like the center of a filter to be placed at each one. How do we work that out? Well, assume first we will display from 0 to Fs/ 2 Hz in our spectrogram (0-5000 Hz here). If we have 500 pixels to cover 5000 Hz, we want a filter about every 5000/500 = 10 Hz. This is the filter spacing Δ F. When we know Δ F, we can calculate the DFT length in points as $N_{dft} = F_s / \Delta F$ = 10,000/10 = 1000 points. (Most programs will actually calculate DFT via an FFT program that will make the number of points analyzed a power of 2. So the program might jack the value up to 1024 (the next power of 2) . This is usually OK. It will give you a little narrower spacing than needed, but it will err on the side of extra detail.

Picking a frame (window) length for a 300 Hz bandwidth. Recall in talking about FFTs we defined three lengths: *N* the total number of points in the DFT; *M*, the length of the signal analysis frame is windowed and P , the number of points added to pad the signal to get more filters resolution (i.e., to get the proper filter spacing, Δ F) We have a formula on a previous handout relating the duration of a Hamming window to it's bandwidth. It is : $bw_{Ham} \approx 1.3\ F_s/M$ Solving for M in the first, we get M=1.3 Fs/ $bw_{Ham..}$ We can answer the last question now M= 1.3 (10000/ 300) = about **43** sample points. This corresponds to $T_{ms} \cdot 43$ = 4.3 ms.

Summary

a) Our one-pixel frame advance interval (time step) is D_{adv} = 2 ms and thus equivalently N_{adv} = 20 points).
b) Our one-pixel frequency step is ΔF = 10 Hz. Requires an N_{dft} = 1000 point DFT
c) Our frame (window) duration D_{win} is 4.3 ms (corresponding to M_{win} = 20 points) to get a bandwidth of 300 Hz with a Hamming window.

From this we see that our frequency step as a fraction of BW is ΔF / BW = 10/300 or only about 1/30 of our bandwidth. This is probably more detailed than necessary. Praat would cut if off at 1/14 or so and hence use a shorter FFT (smaller value of P padding points). Our frame advance D_{adv} is 2.0 ms. Compared to our window duration D_{win} 4.3 ms, our relative frame advance is D_{adv}/D_{win} = 2/0 / 4.3 or about 0.47 times our window length. In the abstract, this would be a little coarse. To get down to our recommended frame advance = 1/4 window duration, we'd need a shorter frame advance. But we only have 500 pixels to play with, so making the frame advance won't make the picture any better. If we had 1000 pixels to play with would get us there, we could use a relative frame advance to window duration ration of less than ¼ and satisfy the rules. Both Praat and WaveSurfer set pretty good default values for their spectrograms and you will not often have to resort to this much detail in making your own spectrograms. But every once in a while, you have a signal that doesn't look right. This example discusses just about everything there is to adjust in the computation of digital spectrograms. These are important if you want to save the information in spectrogram in a file for later processing (or inspection). Both Praat and WaveSurfer allow you to do this. There are other issues that don't have to do with the computing spectrogram itself, but only how it is displayed to the screen to make it look nice. These include the dynamic range – how many dB down from the maximum should we bother to show at all – and image brightness and contrast. If you have a nice color-mapped spectrogram on a good computer monitor, these don't matter too much. They can matter a great deal for getting nice gray-scale spectrograms and especially gray-scale printouts (the latter is the most difficult of all). WaveSurfer has the best graphic controls I've yet seen for this kind of application. I find Praat much harder to control.

In addition to spectrograms, there are several other important techniques in speech analysis use the short-time analysis framework.

Autocorrelation and related methods for pitch (F0) analysis. (H 76-77; KR 90-99; J 28-31)

There are a number of methods for determining the F0 of a signal. When we do this by hand, what we do is look for the shortest pattern that repeats itself almost exactly. Then we measure the duration of one cycle of the repeating pattern. (If the pitch isn't changing much, it's often more accurate to measure the duration of *n* such repeating cycles and divide by the duration of the whole group by *n* to measure an average period length).

Several automatic pitch measuring techniques also rely on measuring a kind of 'self-similarity' of an earlier part of the signal to a later copy of itself. The simplest of these conceptually is the average magnitude difference function (AMDF). It looks like this

{AMDF}Define y(i), i=1: N as the windowed section of a signal extracted as a single frame form a longer signal . Then the AMDF for the k-th lag is

$$\mathrm{AMDF}(k) = 1/N \sum_{n=1}^{N} \mathrm{abs}[\, y(n)-y(n+k)]$$

Note that if the period length is exactly *p* samples long, this and a signal is perfectly periodic then the AMDF will be exactly zero when *k* = *p*. When *k* is very different from *p*, the delayed or lagged part of the signal won't look at all like its predecessor, and the AMDF will be large.

Finding the lag *k* that shows a minimum in the AMDF function is sometimes used for pitch analysis (WaveSurfer has this as an option). However, a more commonly used method is based on autocorrelation (the correlation of a signal within itself). Autocorrelation also involves a comparison of a part of a windowed signal with a 'delayed' or 'lagged' copy of itself. The box below shows a standard method of calculating the autocorrelation of a signal.

{ACF}Define y(i), i=1: N as the windowed section of a signal extracted as a single frame form a longer signal . Then the raw autocorrelation coefficient for the k-th lag is

$$R(k) = \sum_{n=1}^{N-k} y(n)y(n+k)$$

The normalized autocorrelation (which we'll use) is then

$$r(k)= R(k)/R(0).$$

This is typically calculated for k = 0 to some maximum number, K, and the whole sequence r(0) , r(1),... r(k) is referred to as an autocorrelation function.

Note: The term lag refers to the *absolute* difference in sample numbers between y(n) and y(n + k). The pairs
[y(n), y(n+k)] and [y(n) and y(n-k)] are both said to have a lag of k samples.

The value of the autocorrelation is calculated over a number of lag values. For pitch analysis, when the lag value gets close to the period of the fundamental frequency, the autocorrelation value gets large. The maximum autocorrelation is always at lag 0, which compares a signal with a copy of itself. This is usually used to normalize the other values (each raw autocorrelation, so the value of the autocorrelation for any lag is always between +1 and -1. For pitch analysis, negative values are not important; it is large positive values that count). The basic rule for autocorrelation analysis for a pitch estimate is to pick the largest value over a range of lags and use the corresponding lag to estimate the period of the signal and from that, the fundamental frequency.

For a variety of reasons, neither AMDF nor autocorrelation works perfectly with speech (nor do any of a dozen other automatic pitch extraction methods). Pitch doubling and halving are common problems (octave errors). There are several rules of thumb that tend to lead to good analyses. First, a pitch range should be determined for a given voice. For example pitch range for an adult female speaker might be in the range of 175 to 250 Hz. To get a strong peak at a lag corresponding to the fundamental, the signal must contain at least two pitch periods. Since a Hamming window tapers off the ends of the signal, it is safest to aim for about 2.5 periods of the lowest frequency pitch to be analyzed. Then, to lessen pitch doubling and pitch halving errors, it is best to search only in the expected range of lags corresponding to the periods of the lowest and highest values expected. This is worked out below for the following. This then may need to be converted to lag counted in sample points.

Under the hood. There are numerous variations on the basic autocorrelation method. Praat and WaveSurfer both build in quite a bit of extra processing. For example Praat's default pitch analysis compensates for the effects of the windowing function on the autocorrelation. WaveSurfer uses a modified autocorrelation method where the upper limit of the summation in box ACF is N rather than N-k. In addition both pitch trackers use methods for interpolating both the peak value and the lag. The methods we described above force the estimated period to be a multiple of T_{samp}, the sampling interval. But speech is not so constrained. If some kind of interpolation is not done, this can result in noticeable inaccuracies in high F0 voices at lower sampling rates. Interpolations methods (Praat's is quite elaborate) can help this quite a bit.

Example Autocorrelation analysis . Suppose we have a sampling rate of 8000 Hz (Fn=4000) and are analyzing a voice with a 175 to 250 Hz range. The sampling interval T_{ms} is $1000/F_s = 1000/8000 = .125$ ms; that is, each sample point represents 0.125 of a ms. Correspondingly, the number of samples per ms is $N_{spms} = (F_s/1000) = 8$ samples per ms.

First, we can determine the period durations in ms of our search range.

(a) Our lowest expected F0 is 175 Hz corresponding a longest expected pitch period of $T0_{ms} = 1000/F0 = 1000/175 = 5.714$ ms.

(b) Our highest expected F0 is 250 Hz , corresponding to a highest expected pitch period of $T0_{ms} = 1000/F0 = 1000/250 = 4.00$ ms.

(c) Our window length should be about 2.5 times the longest expected period 2.5 x 5.714 = 14.285 ms.

We can convert the above lags by dividing by the number of milliseconds per sample (T_{ms} = 1000/F_S) = 14.285/ 0.100 or by multiplying by the number of samples per milliseconds (N_{spms}) = 8 · 14.285 = 114.28. This is the estimated frame length to find fundamental frequencies in the desired range. Since lags are in sample points, they must be converted to integers by rounding. How we round requires a little thought. (See additional notes at end of handout).

(a') Our longest expected lag is the longest pitch period converted to lags = 5.174 ms x N_{spms} = 5.174 x 8 = 45.712 samples (lags). For some purposes, fractional lags are ok. But here we're determining a search range where the search bins are limited to integers. So we need to round. I will round up to 46 lags. (I would have rounded up even if the answer was 45.1 lags. Why?)

(b') Our shortest expected lag is the longest pitch period converted to lags = 4.0 ms x N_{spms} = 4.00 x 8 = 32 samples (lags). So we don't need to round here. But what if the answer had been 32.6 samples? I would have rounded down to 32. Why?

(c') The window length in samples should converted to lags is 14.285 ms x N_{spms} = 14.285 x 8 =

= 114.28 samples (lags). Here I round up again to 115 lags. (Why?)

So summarizing, we use an analysis window of ***M* = *N***$\boldsymbol{M = N_{win} = 115}$ samples

And we **search the autocorrelation function from lags 32 to 46**.

Suppose that we found the largest value of the autocorrelation in that lag range occurred at lag number 43. What is the period and the fundamental frequency?
The period is the sampling interval T_{ms} x the lag number (or equivalently, the lag number divided by N_{spms}) the number of samples per millisecond).

= .125 ms/sample · 43 sample lag = 5.375 ms, corresponding to 1000/5.375 = 186 Hz.

So our answer is an estimated **F0 of 186 Hz.**

Cepstral methods

The cepstrum can be viewed as involving the spectrum of a spectrum. (KR pp 78-81). To calculate the cepstrum of a signal one takes log of its amplitude spectrum as calculated by a DFT. (Similar results obtain if you use the dB spectrum.) Such a spectrum is often called a "log-magnitude spectrum." If a voiced speech signal was the input, the log magnitude spectrum shows a rapidly varying ripple, a "picket fence" type pattern where the pickets represent harmonics of the fundamental frequency. Often, this looks almost like the waveform of a sinusoid, except the x-axis is frequency instead of time and the amplitude of adjacent peaks varies from peak to peak. The spacing between the pickets represents the spacing of the harmonics and hence, indirectly the fundamental frequency. Signals with a "faster" ripple pattern (more ripples/kHz) have lower fundamental signals. In a cepstral analysis, roughly speaking, we "pretend" that this log-magnitude spectrum is a waveform (i.e. that the frequency axis represents time) and we do another DFT. When we plot the amplitudes of this second DFT, we typically find a large peak at a *lag value* corresponding to the period of the original fundamental frequency. The x-axis of the second spectrum (i.e. of the cepstrum) is back in a kind of peculiar time domain, sometimes called quefrency, but also known as the lag domain. It can be measured in time units of the original waveform and the period can be converted back to a fundamental frequency. This was once one of the standard ways of estimating F0. However, it is now regarded as inferior to autocorrelation based methods.

LINEAR PREDICTION (LPC)

As KR notes, p 72, linear prediction is a method that is based on predicting the n-th sample of speech, x(n), from a series of previous samples. The first use of linear prediction was in speech coding for transmission of telephone signals. Linear prediction is often referred to by the initials of this coding procedure, called **l**inear **p**redictive **c**oding. [Although there are several ways to calculate LPC coefficients, the most common one actually starts with an autocorrelation function. Johnson (pp 40-42) gives some hints of how formant information is contained in an autocorrelation-like function]

{LPC Extra (optional)}

If K is the order of the LPC, then the prediction formula is

$$x(n) = a_1\, x(n\text{-}1) + a_2\, x(n\text{-}2) + \ldots\ldots + a_{K\text{-}1}\, x(n\text{-}(K\text{-}1)\,) + a_K\, x(n\text{-}K) + e\,(n)$$

where a_i, i =1..K are linear prediction (LP) coefficients, $x(i)$ is the i-th sample of the signal; $e(n)$ is the prediction error or residual for the n-th sample. The term "lag" is also used here. $x(n\text{-}1)$ is the signal $x(n)$ lagged by one sample. LPC methods try to minimize the total RMS of the error, signal $e(n)$, as n ranges over some stretch of the signal.

The set of prediction coefficients, $\{a_{1\ldots}a_K\}$, can be used to construct a kind of approximate "inverse filter" similar to that described on p 84 and 85 of KR. Although the assumptions of LPC analysis are close enough for many purposes (such as formant measurement), they are not good enough to get a valid estimate of pressure variation at the glottis required for research quality work on phonation. More elaborate methods are required for that purpose. The nature of the LPC inverse filter is to flatten (or "whiten ", making it more like white noise) the spectrum over the analysis range, as illustrated on the accompanying previous handout.

Properly done, LPC can be viewed as modeling (accounting for) slower varying trends in the spectrum due to formants and to glottal roll-off and to the radiation characteristic. The coefficients can be used to estimate a smoothed spectrum in which the rapid amplitude variation associated with glottal source (the harmonic pickets in the picket fence) is ignored and only more smoothly varying trends, including formants, are left. Given such a smoothly varying spectrum, it is much easier to estimate formant frequencies and bandwidths from the LPC-smoothed spectrum than from an FFT. Most modern formant estimation procedures use some type of LPC analysis.

How can you do an LPC analysis "properly"? LPC assumes an "all pole" model of the signal. There are three kinds of "poles" that LPC can find. The most important kind corresponds to normal formants. Each of these will require a frequency and a bandwidth parameter in the frequency domain. As a result, each typical formant (type 1 pole) requires inclusion of two additional LPC coefficients to model properly the spectrum. The other two types of poles are "degenerate formants" with frequencies fixed at either zero Hz (type 2), or at the upper cutoff frequency of the analysis (type 3). These effectively have only one free parameter each, corresponding to their bandwidths and consequently add only one additional coefficient to the pot for the LPC equation. Such degenerate formants are can actually quite useful in approximately modeling spectral shaping due to the glottal roll-off and radiation characteristics.

Rules of thumb for LPC analyses

1) Determine (or guess) the expected formant frequencies first four or five formants for the speaker you are analyzing. Use neutral tube expectations for a speaker of a given population

group. For example, a typical adult male voice has formant frequencies of (500, 1500, 2500, 3500) for the first four formants. Adult females will be typically about 1.15 times these values. Children's voices vary from 1.2 for perhaps 10 year olds to 2.0 for infants times the adult male values. We probably should use typical female values as the reference, as they are closer to population averages, but the adult male values just happen to be very easy to remember.

2) Limit the frequency analysis range to the lowest four or five expected formants. Try to put the cutoff about halfway between the q-th and [q + 1] -th formant, because having a formant too close to the folding frequency can cause some problems. So if you want to analyze 4 formants (q = 4), try to pick a cutoff between F4 and F5. Why limit to q = 4 or 5? Above this, speech spectra are much more complex than predicted by idealized "tube-systems" for a variety of reasons. For one thing, longitudinal standing waves no longer true at high frequencies, because the cross-section dimension of the vocal tract is a significant fraction of the wavelength of frequencies above about F4. One way to limit the frequency range this is to select a lower sampling rate when you first record the signals on the computer. However, it is often more useful to do it in software. (WaveSurfer and Praat both allow you to do this.) You may eventually want to measure more than formant frequencies of vowels in your signals. You can sample at a higher rate and then change the sampling rate in the computer; this can be done explicitly resampling the digital signal in the computer. It can also be done implicitly by some software that limits the frequency range that is analyzed without overtly changing the signal itself. Either method allows you to analyze only a part of the full Nyquist frequency range for the analysis of vowels.

3) Use preemphasis of about +6 dB per octave for your analysis. Preemphasis involves using a simple high-pass filter with shallow skirts to boost the higher frequency components of speech signals. It is usually used in spectrographic analysis to make the weaker energy in the high frequency regions more visible. Preemphasis has the advantage of reducing the effects of glottal roll-off and the radiation characteristic and makes LPC analysis better.

4) Allow two coefficients per expected formant (8 coefficients if you follow steps 1 and 2 above) plus 2 or 3 "for the pot." The extra coefficients help compensate for glottal characteristics that are not eliminated by pre-emphasis and for violations of the analysis assumptions. There may be an extra weak formant and/or 'antiformants' or zeros because of nasalization or because of sub-glottal coupling (during the open cycle of a glottal period, the vocal tract is connected to the trachea). In addition, there may be 'degenerate' poles near Nyquist frequency (e.g. , if q = 4 an F5 may dip down close to the Nyquist frequency and LPC has to try to account for the effects of its low frequency skirt).

5) Examine several different analyses of several signals from the same speaker. Study the fit of the LPC smoothed spectrum to a raw DFT spectrum. See if you are consistently missing or blurring significant peaks. If so, add one or more coefficients. Also check to see if the cutoff frequency you picked in step 1 makes sense. Are you at least roughly splitting between F[q] and F[q+1]. If F[q+1] happens to be moving around and crossing your cutoff frequency, this can cause real problems for LPC as no consistent choice of analysis may work well for all frames.

6) Beware that you don't put too many coefficients in. If you do, the LPC model will try to model not only the smooth variation due to formant properties, but also the more jagged "picket-fence" variation due to harmonic structure. We'll examine such cases in class.

Technical notes

The relation between LPC coefficients and formant frequencies and bandwidths (for science majors, mainly)

The correspondence between pairs of LPC coefficients and frequency-bandwidth pair is not straightforward. The mathematics behind this involves factoring a polynomial and a few other twists. The relationship between formant frequency and bandwidths for a two formant system can be given as follows. The first formant has frequency F1 and bandwidth B1 and the second formant has F2 and B2. For F1 the following three quantities are defined: w_1= -exp(-2π T B1),. v_1= 2 exp(-π T B1) cos (2π T F1) ; u_1= 1-w_1-v_1. (T is T_s is the sampling rate in seconds = 1/ Fs) w_2,v_2, and u_2 are defined analogously for F2 and B2. The LPC coefficients for "flattening" the spectrum of such a system can be derived from the coefficients of the an expanded polynomial of the form (w_1+v_1 x + u_1 x^2) (w_2+v_2 x +u_2 x^2). That is

w_1w_2+ (w_1v_2+ w_2v_1) x, (w_1u_2+u_1+v_1v_2) x^2 +(v_1u_2+v_2u_1) x^3+ $u_1u_2x^4$.

The *i*-th LPC coefficient , a_i , is the multiplier of the *i*-th order (in *x*) term. So here a_1= w_1v_2+w_2v_1; a_2=w_1u2+u_1+v_1v_2, = $a_3$$v_1u_2$+$v_2u_1$, a_4= u_1u_2. Adding a third formant leads to an equation (w_1+v_1 x + u_1 x^2) (w_2+v_2 x +u_2 x^2) (w_3+v_3 x + u_3 x^2). This will add two more terms (in x^5 and x^6) to the expanded equation. Note that the relation between the a coeffieients and formant properties (frequencies and bandwidths) is indirect, so each of the formants will contribute to coefficients of each of the terms.

Notes rounding and number of digits to keep

There are a few things that must be made integers in DSP because they count discrete entities. These include

1) Number of sample points in a portion of a (windowed) digitized signal,
2) Number of lags in an autocorrelation
3) Number of coefficients (or the order) of an LPC analysis.

When an answer involves one of these things, you really should force it to an integer value. Everything else can be kept as a real or decimal number.

Varieties of rounding

Standard rounding to nearest integer (school rounding). Grade school rules say round up when the decimal part is greater or equal to .5 . Statistical and scientific variations on that rule say if the decimal part exactly .5 then round to the closest even number. Using school rounding will rarely lead to bad errors. So if you forget everything else, use it.

Floor and *ceiling* rounding functions. The floor function says round down to the integer part, throwing away any decimals. The ceiling function says round up to the next highest integer. Sometimes these are better things to do than standard rounding.

When to use floor, ceiling and standard rounding in DSP

What's the best way to round depends on the ultimate purpose of your rounded integer. A little thought should make clear the right way to proceed.

Examples.

I want to do a broad-band analysis with a bandwidth of 300 Hz using a Hamming window at a sampling rate of 4.1 kHz. The desired bandwidth can be worked out from bw_{Hamm} = 1.3 x F_s/M. So M = 1.3*F_s/ bw_{Hamm} = 1.3 x 4100/300 = 17.77,

With school rounding, we'd make that 18 points. But that makes our actual bandwidth

1.3 x F_s/18 = 1.3 x 4100/18 = 296 Hz, which is a little 'narrower' than the 300 Hz of our desired specification. If we used the floor function and rounded 17.77 to 17, our achieved bandwidth would be 1.3*4100/17 = 313.52 Hz. This is a little 'wider' than our desired 300 Hz. But since we're doing a broad-band analysis in the first place, it's likely we'd rather our bandwidth be a little wider rather than a little narrower. (We probably want to smear harmonics to make the formant pattern stand out more).

On the other hand, if we wanted to do a narrow band analysis of say 45 Hz, so we could see harmonic structure more clearly, we would probably want to *round up*. 1.3*4100/45 = 118.4. Applying the ceiling function we would use 119. This gives a bandwidth (with a Hamming window still) of 1.3*4100/119 -= 44.79 Hz, a little narrower than the nominal 45 Hz target.

Similar logic applies to autocorrelation lag ranges. If we want to look for F0s in the range of say 80 to 400 Hz and we really mean we want to include 80 and 400 Hz as possible answers, we want to make sure our shortest lag is NO LONGER than 1000/400 ms and our longest lag is NOT SHORTER than 1000/80 ms. This means we should round the short lag down and the long lag up. The method of rounding of the window length in an autocorrelation analysis is not so important. But I'd probably also round that up.

Rounding in expected number of formants below a cutoff frequency.

If we use the rule of thumb for the number of expected formants below a given frequency, $N_{formant} = F_u$ / (2 x $F1_{Neut}$),
standard school-rounding is actually well-motivated here.
Why? Because in each 2 x $F1_{Neut}$ band, a neutral formant will be exactly in the middle.
Consider a female voice with $F1_{Neut}$ = 610 Hz. So F2 = 1830, F3= 3040.
Suppose we have a cutoff of 3200. Then $N_{formant}$ = 3200/(2*610) = 2.6 We'd better round that up to 3 or we'll miss F3 at 3040. If our cutoff is exactly at value corresponding to the *k*-th neutral formant F_k, then our answer for $N_{formant}$ will be exactly k-1+.5. (Try it setting the Fu to the values of F1, F2 and F3 above). One of the many reasons why we can't be more precise about the number of LPC coefficients is because it's difficult to know exactly whether a cutoff frequency will be above or below any fixed number of formants. As the decimal part of $N_{formant}$ gets close to .5, whether or not the next formant is actually in or out of the range becomes even dicier. Of course if we've looked at some spectrograms of the signal in question, we have a better idea where exactly F4 (or F_k for some other *k*) actually is for this voice, so we may be able make a better guess. In the absence of such knowledge, usual school rounding is probably the best place to start.

Decimals to keep: Summary

For durations of speech events like vowels, stop gaps etc, 1 ms precision is usually enough.
For frequencies, 1 Hz is more than enough for formant frequencies.
For measuring F0, and for all frequencies in intermediate calculations in digital signal processing, it's advisable to keep one decimal place in Hz. For dB, one decimal will definitely do. For DSP sampling, keep the number of samples per ms, N_{spms}, to at least two decimal place. Although I've used *N* as the first letter, this is really not an integer. The sampling rate 44100 Hz has an N_{spms} of 44.10 samples per millisecond. Rounding to 44 is not quite accurate enough for practical purposes. Keep the sampling period in ms, T_{ms}, to 4 decimal places. The sampling rate in seconds T_s needs to be kept to 7.

See example problems. (AcLpcPractice)