Chapter 3

THE SPECTROGRAM

In this chapter, we'll examine an important tool for exploratory signal analysis: the spectrogram.

Summary

But first, a brief outline of the essentials from the previous chapters. At this point, you should be comfortable with:

- the terms: **time series**, **linear spectrum**, **spectral density**, and **window function**;
- construction (using the *fft* function) of the proper linear spectrum from a time series;
- construction of a linear spectrum that will produce a real time series (using the *ifft* function);
- consequences of the relationship, $\Delta f \Delta t = 1/N$, for sampled signals;
- construction and meaning of the spectral density;
- correction of the spectral density when a time-domain window function is used;
- **averaging** – both "mean-square" (or *rms*) and synchronized; and
- application and consequences of time-domain window functions.

Furthermore, you should be able to write computer code to implement any of the following operations using real data:

- given a time series, *x*, generate the linear spectrum, *X*, and the spectral densities, G_{XX} and S_{XX};
- check the spectral densities against the mean-square value in the time domain by "integrating" the spectral densities;
- generate averaged spectral densities, G_{XX} and S_{XX}, with and without time-domain windowing; and
- given a time series, *x*, and timing information, perform synchronous averaging in the time domain.

You should be starting to build a library of functions that perform the above operations and you should continue to build this library as the course progresses.

Introduction to the Spectrogram

The spectrogram is an important tool for time-frequency analysis of signals. The spectrogram is generated from the spectral densities of many short-time records. In effect, the spectrogram identifies characteristic rates or frequencies and their variations with time. One way of viewing the Fast Fourier Transform is that it compares the recorded signal to a large collection of pure tones and returns complex numbers whose magnitude indicates the degree of match to the pure tone and whose phase[1] indicates the relative timing of the tone. If the actual signal frequency is constant, then the same pure tone will always show the best match; if the actual signal changes frequency, then the best match will move from tone to tone tracking the actual frequency. The result is a map of the frequencies contained in the actual signal as they evolve with time. A color or a shade of gray shows the degree of match.

The following set of figures illustrates the construction of the spectrogram from the point of view of matching to pure tones. (We'll consider the practical construction details later.) Figure 3.1 contains four quadrants. The upper left quadrant shows a collection of pure tones (black "sinusoidal" curves that oscillate up and down smoothly) with different rates. Just below this collection of pure tones is a sample signal (blue) that starts at a steady high frequency, then transitions down to a lower frequency and remains at that lower frequency.

In the upper right quadrant, the first section of the sample signal is compared to the pure tones. Some of the pure tones oscillate faster (have higher frequencies) than the sample; other pure tones have lower frequencies; and one of the pure tones matches the sample signal's frequency (see the lower left quadrant).

[1] The phase is not used in the spectrogram; the spectrogram is built from the spectral density, which is proportional to the magnitude squared of the linear spectrum.

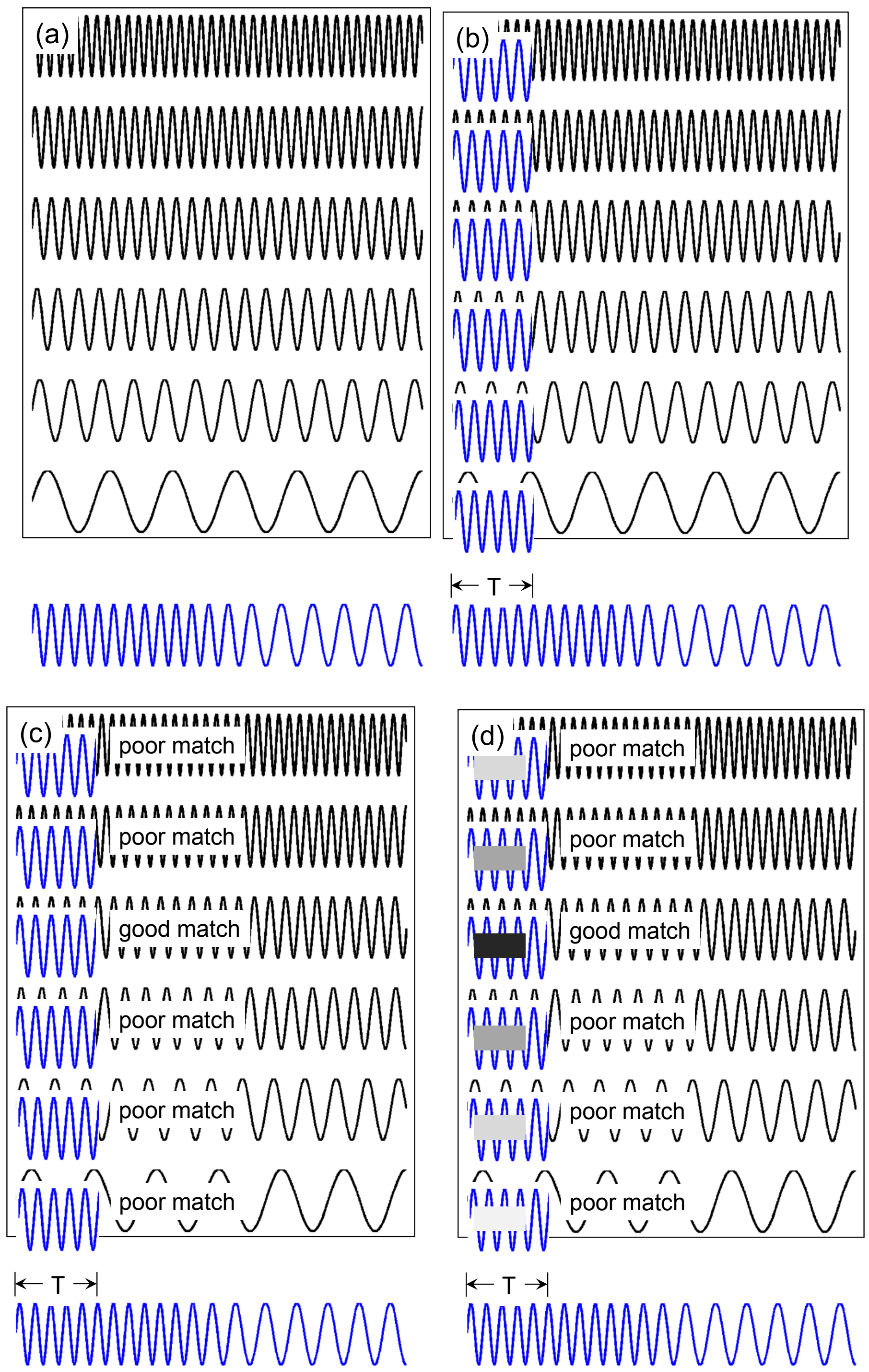


Figure 3.1. Constructing a map of frequency variation (Part 1): (a) Construct a set of pure tones (the six black curves); (b) extract a segment of the signal (blue curve) T seconds long and compare that segment to the pure tones; (c) a good match produces a high score, a poor match produces a low score; (d) convert the matching score into a shade of gray—darker equals better match.

In the spectrogram, the degree of match is coded as a shade of gray (or a color) with darker shades indicating better match. If there were no background noise, then the degree of match would be zero (white); however, in any real measurement, the background noise will produce a small degree of match. In any event, true rates in the input signal will generate strong matches (dark colors) against a background of lighter grays[2].

The next figure (Figure 3.2) illustrates the process when the frequency of the sample signal changes. In this example, the sample signal has one constant higher frequency for the first third of the time, another constant lower rate for the last third of the time, and a smooth transition in rate in the middle third.

The upper left quadrant of Figure 3.2 shows the comparison between the middle section of the sample signal and the pure tones. During this time interval, the rate of the sample signal is changing so there will not be a perfect match to any of the pure tones; however, the match will be highest for the pure tone with a frequency equal to the average frequency of the sample signal and the match will taper off away from this frequency.

The upper right quadrant shows the match for the latter part of the sample signal. This part of the signal has a lower frequency so the match is better for a pure tone toward the bottom of the pure-tone collection.

In the lower left quadrant, the sample-signal segments are removed leaving the shades-of-gray blocks. In the lower right quadrant, the pure tones are removed leaving only the gray-shaded blocks. The red dashed line shows the actual change in frequency of the sample signal. Notice the relationship between the actual frequency change and the pattern of gray shading.

[2] If you preferred, you could invert the grayscale so that good matches are lighter and poor matches are darker.

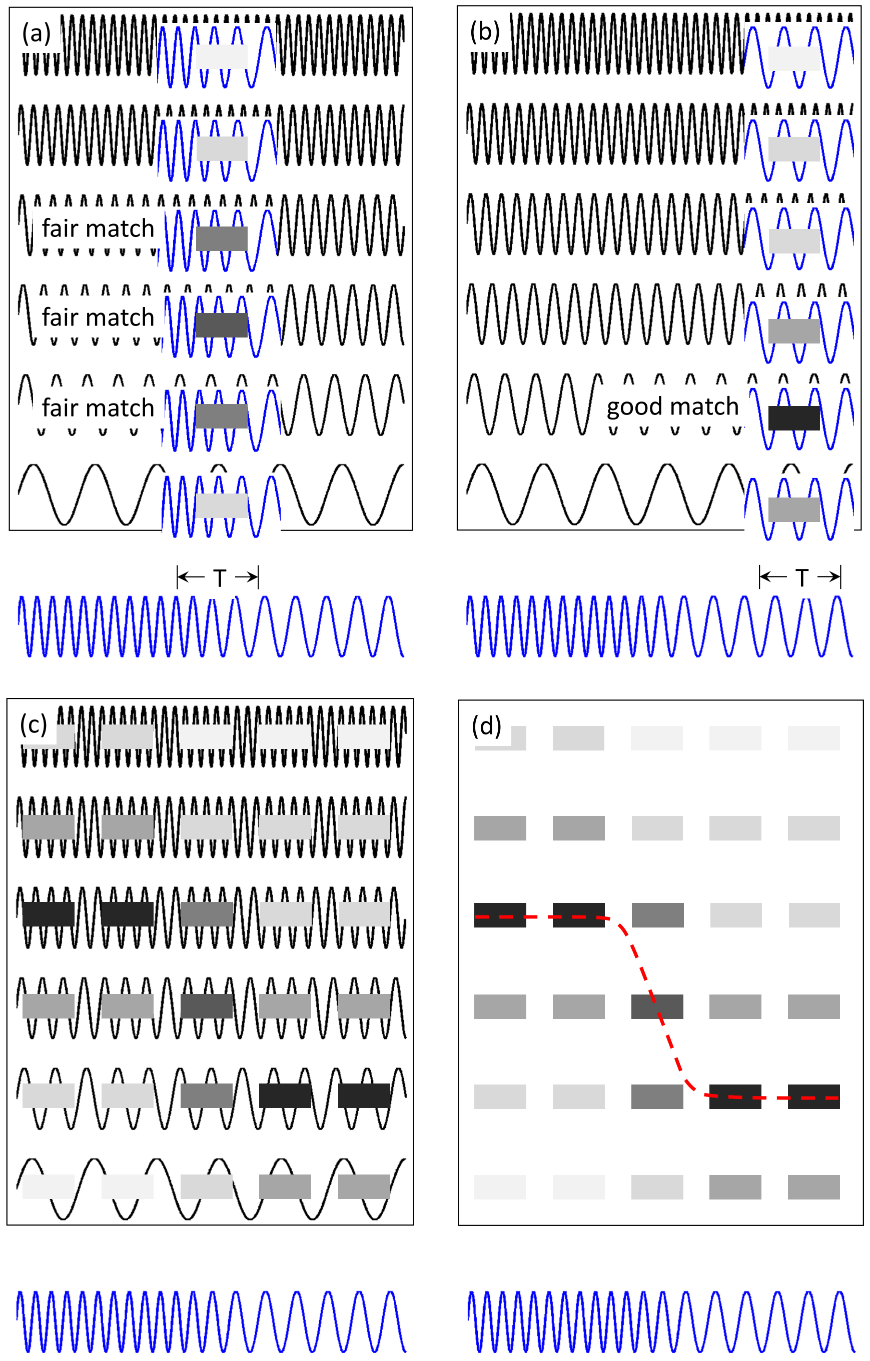


Figure 3.2. Constructing a map of frequency variation (Part 2): (a) Extract another segment of the signal *T* seconds long and get the matching scores; (b) extract another segment of the signal and get the matching scores; (c) translate all of the scores to shades of gray; (d) the gray-scale pixels form the frequency/time map.

An actual spectrogram is shown in Figure 3.3 for this sample signal. Here, many more pure tones are used. Notice that, when the signal frequency is changing quickly, the spectrogram does not resolve the frequency as clearly as when the signal has a constant frequency.

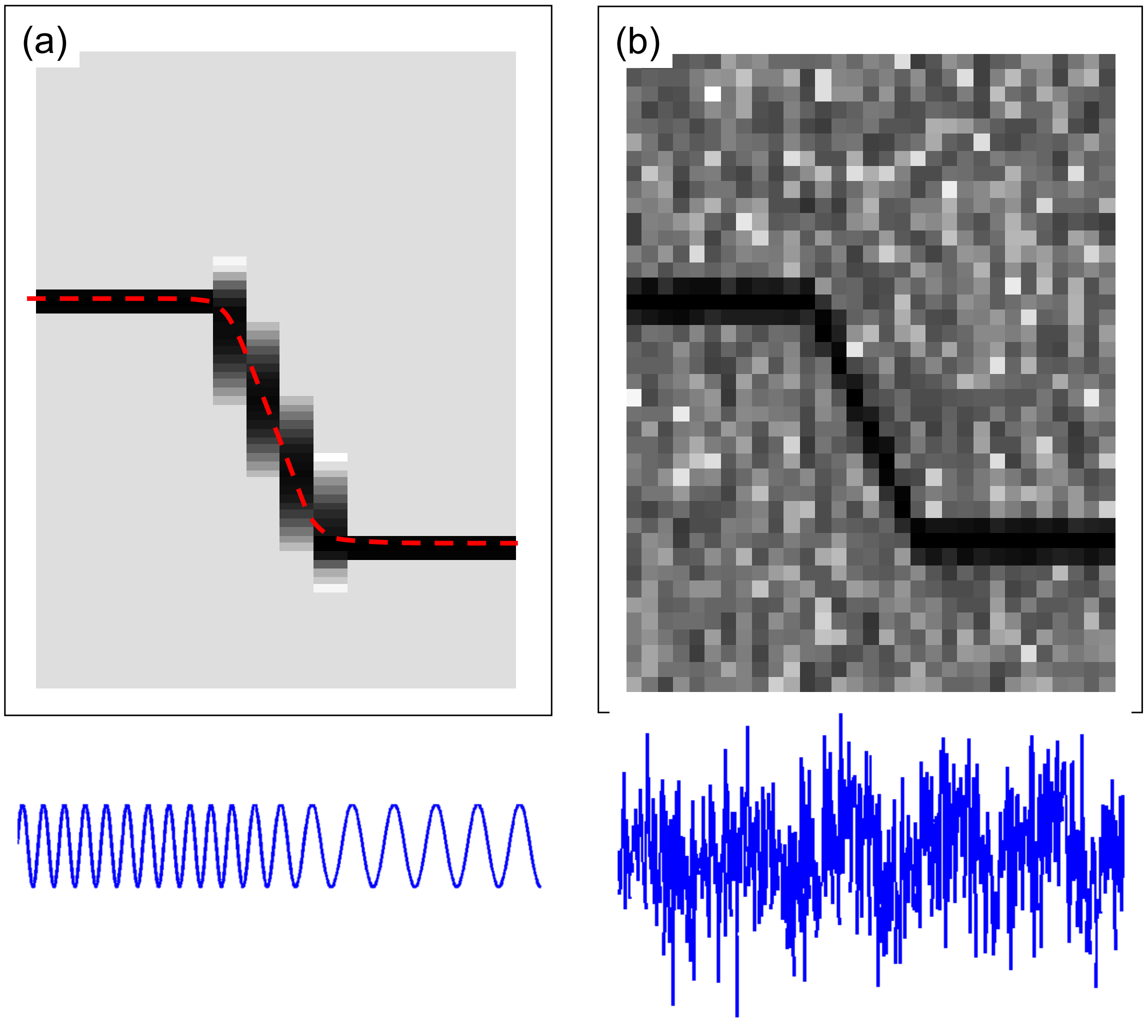


Figure 3.3. Constructing a map of frequency variation (Part 3): (a) Much smaller pixels are generated by using many more pure tones and shorter time segments of the signal; (b) real signals have noise in the background—the spectrogram shows details of the signal even in the presence of large-amplitude noise. Compare the bottom blue traces: with noise on the left, without noise on the right.

The spectrogram on the left was generated from the sample signal but without any background noise. The spectrogram on the right was created from the same sample signal but with noise added. The blue curve below the spectrogram shows the degree of noise added: in the right-hand case, the underlying signal is not distinguishable in the blue curve but the frequency progression can still be seen in the spectrogram. This illustrates the power of the spectrogram in isolating frequencies even in the presence of substantial background noise.

One of the fundamental advantages of spectral analysis is the fact that this analysis does not depend on some assumed behavior of the signal. With a good model for the behavior of whatever is generating the signal, more sophisticated analysis techniques can produce more accurate frequency estimates or may be able to extract signals buried deeply in noise; however, if those model assumptions are not true, the results can be misleading. The spectrogram shows the time-frequency behavior without prior assumption about that behavior[3] making the spectrogram an excellent exploratory tool.

Spectral analysis is, however, limited in its ability to resolve frequencies and times. If we would like to resolve frequencies to within one cycle per second (one hertz), we need to analyze blocks of data that are one second long. If it is acceptable to resolve frequencies more coarsely, to 10 Hz for example, then we can use shorter time blocks—one tenth of a second blocks for 10 Hz resolution. If we analyze with shorter time blocks, then we can see changes that happen faster but at the expense of determining the frequency accurately. If we analyze with longer time blocks to resolve the frequencies more closely, then rapid changes with time are smeared out in the display (see Figure 3.4). This relationship between resolution in frequency and resolution in time is fundamental to spectral analysis.

[3] This analysis does attempt to fit the actual signal with the large set of pure tones and that is, in a sense, an assumption about the behavior; however, this is not an assumption about the mechanics underlying the signal. Furthermore, this same "assumption" is applied uniformly regardless of the nature of the signal.

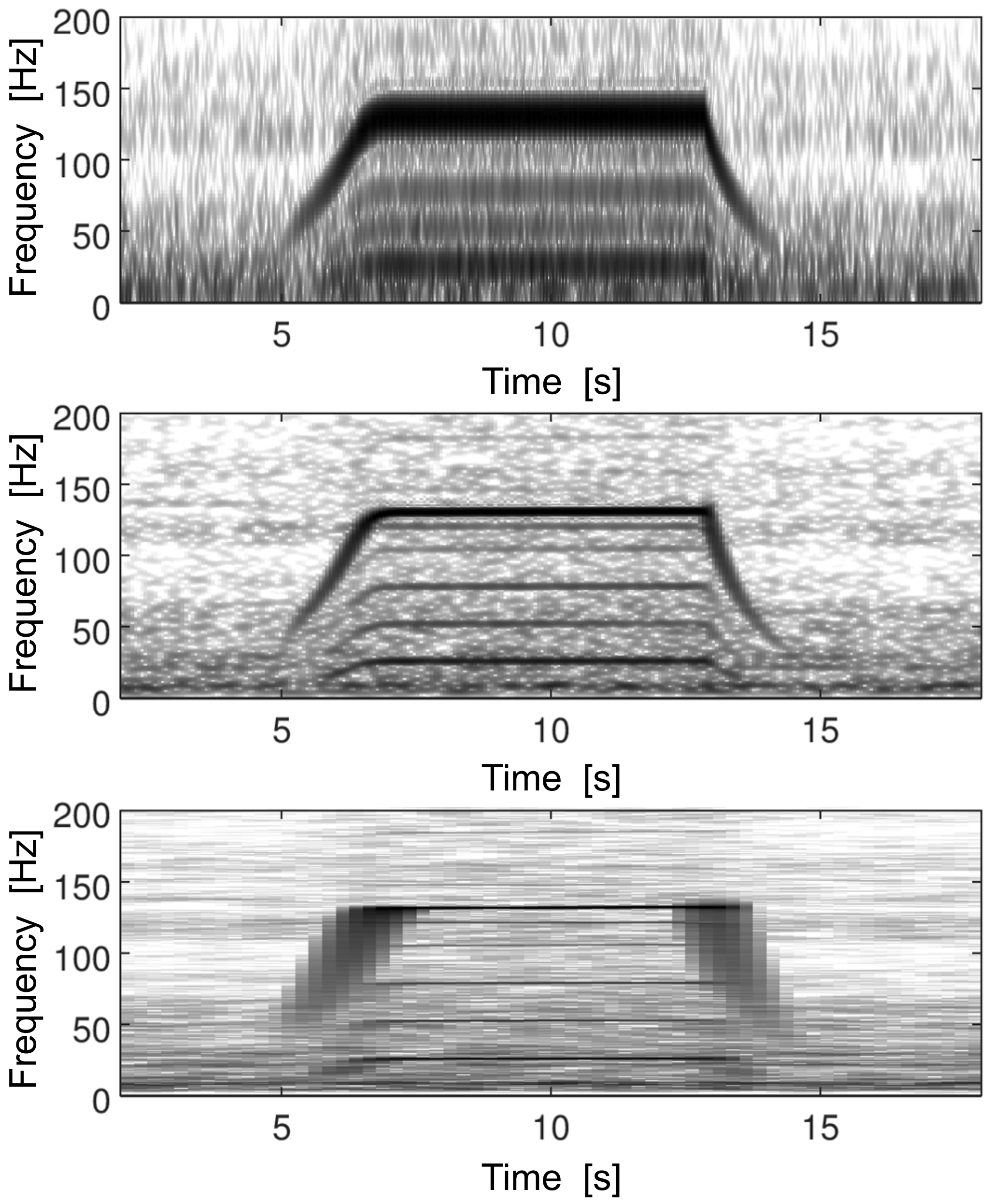


Figure 3.4. An example of the trade-off between resolution in time and resolution in frequency. A short time increment shows rapid changes in frequency more clearly; a long time increment gives better definition of steady frequencies. The example above is from a recording of a five-bladed fan through start-up, steady running, and shut-down. In the top spectrogram, the time increment is 0.1 seconds. The up-sweep and down-sweep are clear but the steady-frequency lines are broad and indistinct. In the bottom spectrogram, the time increment is 2 seconds. Here, the steady-frequency lines are sharp and the rate can be estimated accurately; however, the up-sweep and down-sweep portions are blurred. The middle spectrogram uses a time increment of 0.5 seconds and represents a good compromise in time and frequency resolution.

This tradeoff in resolution is particularly important for the lowest rates. For example, to resolve a line at 10 000 Hz, to within plus or minus 10 Hz, requires time blocks of 0.1 seconds (and so captures rather fast changes when they occur); the frequency is resolved to one tenth of one percent. With the same time blocks, a rate of 100 Hz would also be resolved to within plus or minus 10 Hz but that resolution is much poorer—ten percent—on a percentage basis. Consequently, resolution of the lower rate lines requires longer time blocks or acceptance of a higher frequency uncertainty.

Later in these notes, we will use the spectrogram for quantitative analysis; however, the spectrogram has great value in exploring signals—producing an overview of the nature of a recording. A little experimentation with the spectrogram is well worth the time. With some experience, you will be able to recognize some sounds from their pattern on a spectrogram[4].

Figure 3.5 shows several sounds on a spectrogram display. The first two ((a) and (b)) are whistles; a human whistle at a constant frequency (see (a)) is a surprisingly clean sinusoidal signal. Voiced sounds ((c) and (d)) are composed of many relatively short lines with each syllable producing its own set of frequencies. Each syllable generates a short, usually curved line at the base frequency of the vocal chords (between 100 and 200 Hz in adults) and then successive repetitions of this curved line stacked vertically, each repetition indicating a different multiple of the base frequency: a dramatically different pattern than whistling. Impact sounds (see (e)) are short in duration but wideband in frequency content; these events appear as vertical striations in the spectrogram. Random noise appears as a more or less uniform color or shade of gray with content over a wide range of frequency. This reflects the rather uniform but poor match of noise to many of the pure tones.

[4] But listen to the sound, too! Even if the recording is not of "sound" per se—it might be an accelerometer signal, for example—listening can provide insight that other forms of analysis do not. If interesting features are at very high frequencies, play the sound at a slower than normal rate; if interesting features are at very low frequencies, play the sound at a faster than normal rate. Human perception of sounds can be a powerful tool.

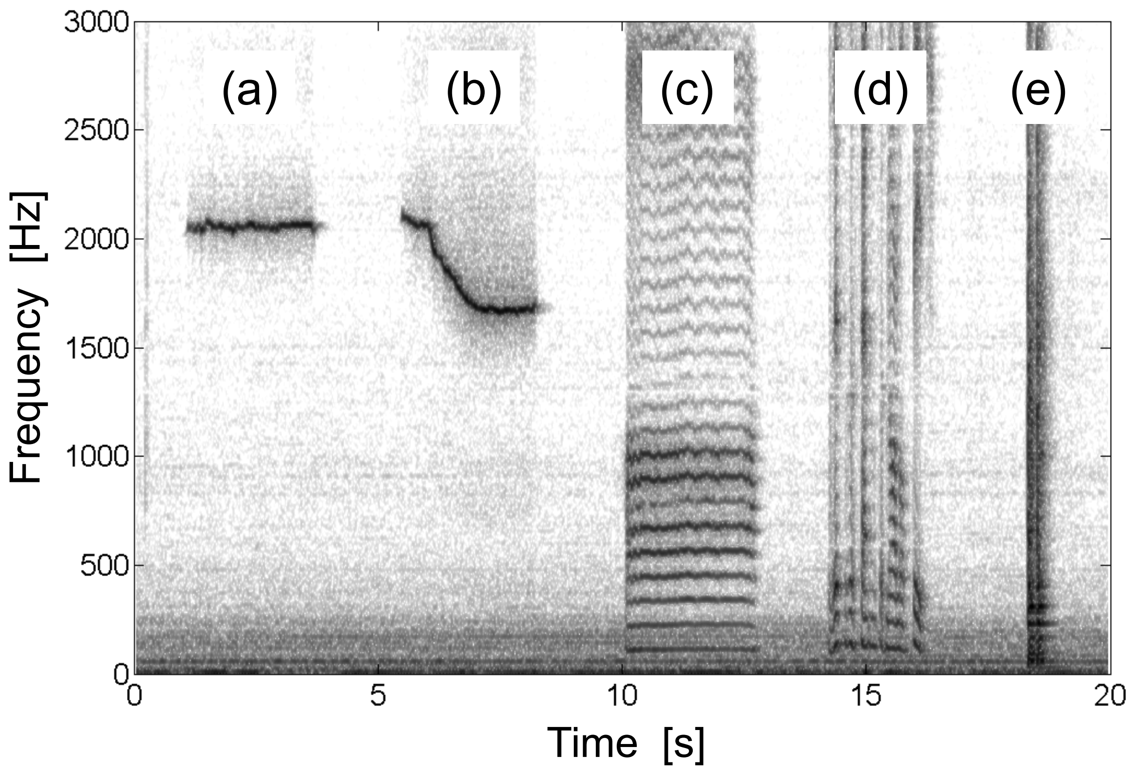


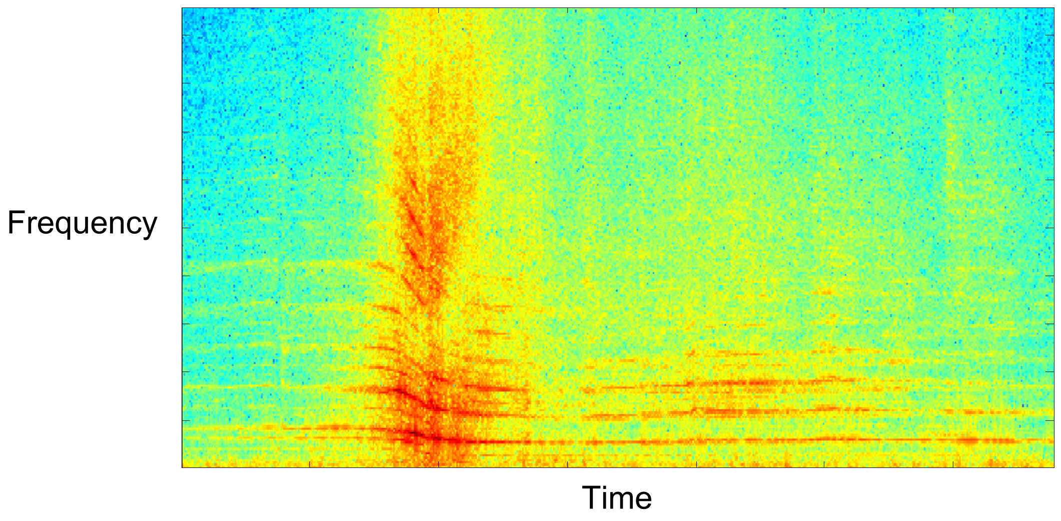
Figure 3.5. Various sounds as they appear on a spectrogram: (a) whistling a steady tone (just over 2000 vibrations per second); (b) whistling a sliding tone from high to low; (c) spoken "ahhhhhhh" showing basic vibration of vocal chords at about 110 vibrations per second and many multiples; (d) the spoken sentence, "This is an actual spoken sentence," and (e) an impact sound: dropping a piece of hard plastic on a table.

While there are many commercial spectrogram programs[5] (and spectrogram functions in analysis software packages), to further understand the use and optimization of spectrogram displays, we will consider, in the next section, details of the construction of a spectrogram. After this discussion, we will also examine some of the limitations of the ordinary spectrogram and develop methods for overcoming these limitations.

[5] As of this writing, Cornell University offers a free spectrogram program (Raven Lite) that is convenient for casual use (even in real time): http://www.birds.cornell.edu/brp/raven/RavenOverview.html or search for "Raven Lite".

Construction of the Spectrogram

A spectrogram shows the evolution over time of the frequency content of a single data channel. The spectrogram displays frequency on one axis (usually the vertical axis[6]) and time on the other axis (usually the horizontal axis). The spectral density value is shown by the color of the pixel. For example,



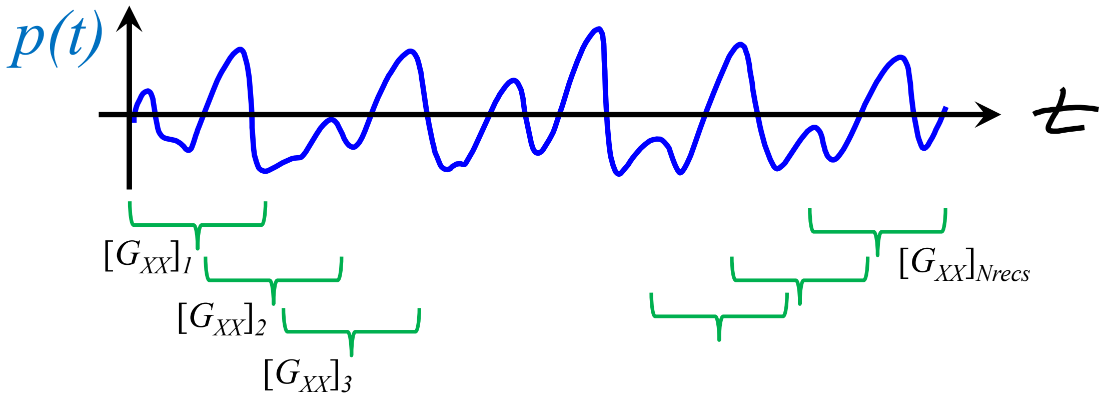
Here, there are a number of tones that shift downward in frequency starting when about one-quarter of the total time has elapsed. The tones are associated with the engine-exhaust pulses from a race car and the frequencies are Doppler-shifted as the race car passes the point of observation. A single averaged spectral density for this entire time period would miss the frequency shift and any tonal structure would be blurred.

The spectrogram is a collection of many single-sided spectral densities for records much shorter than the total file duration. A record length is selected and the spectral densities for all records are calculated. That information is stored in a two-dimensional matrix where the first dimension corresponds to frequency in the spectra and the second dimension corresponds to mid-point time of each record.

[6] Spectrogram displays in SONAR systems are often rotated 90 degrees from this convention: frequency runs from left to right and time increases downward.

The MatLab function, *imagesc*[7], displays the contents of this matrix with a color code that corresponds to the matrix element values. Any vertical cut through the spectrogram gives the spectral density at a specific time; any horizontal cut gives the time evolution of a single frequency.

Construction of the matrix for the spectrogram is straightforward. First, choose a record length. Since you will be generating many spectral densities for the spectrogram, computational efficiency can be important; consequently, you may want to choose the record length to be an integer power of two. That will yield the fastest execution time of the *fft* function. Then select a time-domain window function (the Hann window is good, general-purpose choice) and a percentage overlap. In averaged spectral density computation, an overlap of 50% is typical for the Hann window. For a spectrogram, the display appears smoother with a greater degree of overlap – often 75% to 90% overlap.



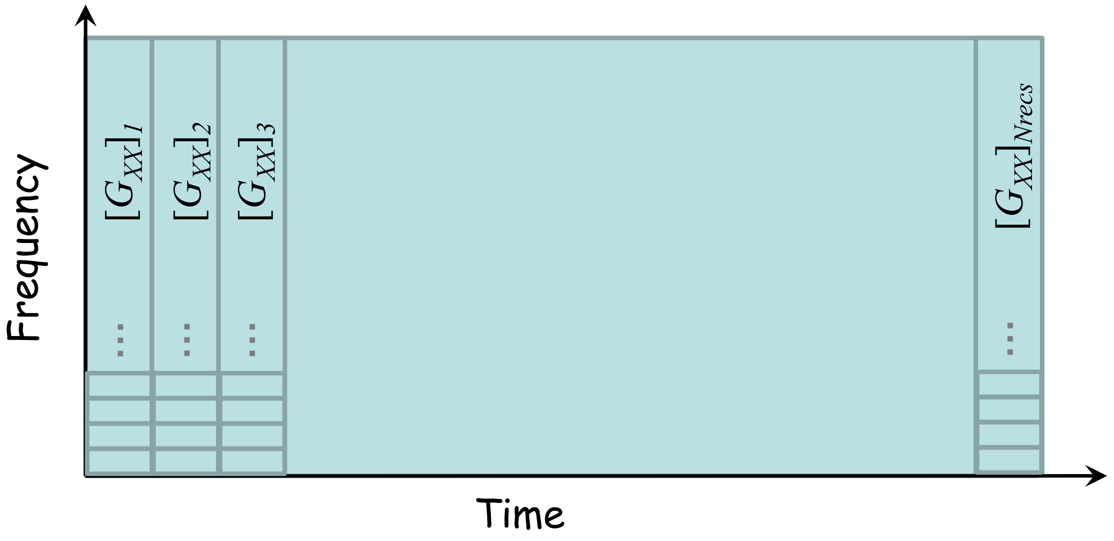
Overlap produces smoothing in the time direction. We can also smooth the display in the frequency direction by zero padding the time records prior to calculation of the linear spectra. If we double the length of each time record by appending zeros[8], then we get twice as many points in the frequency domain. The zeros add no new information so the extra frequency-domain points represent a smoothing interpolation in frequency. In spectrogram construction, overlap is

[7] Study the documentation for the *image* and *imagesc* functions for more detail. Other languages have comparable display functions

[8] If a time-domain window function is used, the window function should be applied *before* zero padding.

common while frequency smoothing is not as common. Of course, both overlap and zero padding lengthen the computation time (and memory storage requirements).

For each record, apply the window function and compute the single-sided spectral density. Normally, these values are converted to decibels so that a greater range of spectral density can be shown by the limited color scale. This is not essential but it is almost always done[9]. Also, the zero-frequency point is usually dropped—the zero-frequency value can be large and doesn't usually add any information to the display. The resulting vector of spectral densities is then stored as one column in the image matrix. The time associated with that column is the mid-point time of the record. The next record is extracted, the spectral density is generated, and the resultant vector is stored as the next column in the image matrix.



Once the image matrix is full, it can be displayed using the MatLab *imagesc* function. (Note: the default display for *imagesc* is for the vertical axis to increase downward. In MatLab, add the line, *axis xy*, after the call to *imagesc*. This changes the display so that frequency increases upward.) The *imagesc* function

[9] Spectrograms can be constructed with functions other than spectral density. Later in these notes, the cross-correlation spectrogram will be introduced. In this case, each column is a normalized cross-correlation. These values would not be converted to decibels as the cross-correlation produces both positive and negative values.

scales the display automatically so that the smallest matrix value is mapped to the color (or gray value) at one end of the colormap and the largest matrix value is mapped to the color at the other end of the colormap. This may not be desirable and, shortly, we'll see how to control the color-range-to-matrix-value correspondence.

To use the *imagesc* function, you also need a time vector and a frequency vector. These can both be two-element vectors: the mid-point time for the first and last records in the time vector and the lowest and highest bin frequencies in the frequency vector. (See the section, Example of MatLab Coding, below.)

I find it useful to construct the time and frequency vectors to hold the entire series of times and frequencies in the display. In this case, the time values would be the mid-point times of each record and the frequency values would be the ordinary bin frequencies of the spectral density. If these vectors have more than two values, MatLab automatically selects the first and last values for constructing the image; however, having all of the intermediate values makes extracting data at a specific time and frequency simpler at the modest cost of a little extra memory.

Decibels and the spectrogram

The information in an acoustic or vibration recording can have an astonishing range in acoustic pressure or acceleration values. If plotted on a linear scale, the few high values would dominate the plot and much of the interesting information would be squashed down onto the time axis and lost from view. The standard approach for showing large ranges of values is to first take the logarithm of the values. The log-axis options in MatLab/Octave (*loglog*, *semilogy*, *semilogx*) perform this transformation automatically. For the spectrogram, we have to take the logarithm explicitly.

While simply taking the logarithm (base 10 or base e) would suffice for the spectrogram, most often the values are converted to decibels, which also includes a scaling. The decibel is defined as ten times the common logarithm (base 10 logarithm, *log10* in MatLab/Octave) of the ratio of two power- or energy-like quantities.

Spectral density is a power-like quantity: it is proportional to the square of the linear spectrum. The factor in the denominator of the ratio is an appropriate *reference* spectral density. For underwater acoustics, pressure references are based on one micropascal (1 μPa) so the reference spectral density would be (1 μPa)^2/Hz. For air acoustics, pressure references are based on twenty micropascals so the reference spectral density would be (20 μPa)^2/Hz, although, for infrasound (frequencies below 20 Hz), a reference pressure of one pascal may be used.

For acceleration, a reference acceleration of one "g" (9.8 m/s^2) is most common although a reference of 1 m/s^2 may also be used.

While the reference is often omitted, *good practice insists that the reference always be included to avoid confusion*.

If we are constructing a spectrogram based on an in-air acoustic measurement, we would compute the spectral densities, which have units of Pa^2/Hz. To convert to decibels[10] using the usual in-air reference, we would divide all spectral density values by (20 μPa)^2/Hz, take the common logarithm, and multiply by ten. Notice that division by the reference makes the argument of the logarithm dimensionless.

Once converted to decibels, the spectral-density matrix can be supplied to the *imagesc* function to produce the spectrogram.

[10] If all you cared about was the pattern of the spectrogram, you could simply take the logarithm of the spectral density (in effect, assuming a reference of 1 Pa^2/Hz). The *imagesc* function's autoscaling will produce the same color pattern as if you carried out the complete decibel conversion.

Exercise 3.1: The Spectrogram

Write a MatLab routine to read a wav file (see the appendix on file types in the previous chapter) and generate a spectrogram for that wav file. Make the fft size (N) a parameter that you can select (but choose N to be an integer power of two for speed). Use a Hann window and 75% overlap. Don't bother to smooth in frequency by zero padding the time records.

To test the spectrogram, construct a time series that is six seconds long with two seconds of zeros, two seconds of a sine wave with known frequency, and two seconds of zeros. This signal will allow you to check both the frequency scale and the time scale of your spectrogram. Choose the sample rate, the sine wave frequency, and the record length for a reasonable display. Use *audiowrite* in MatLab to store the time series as a wav file.

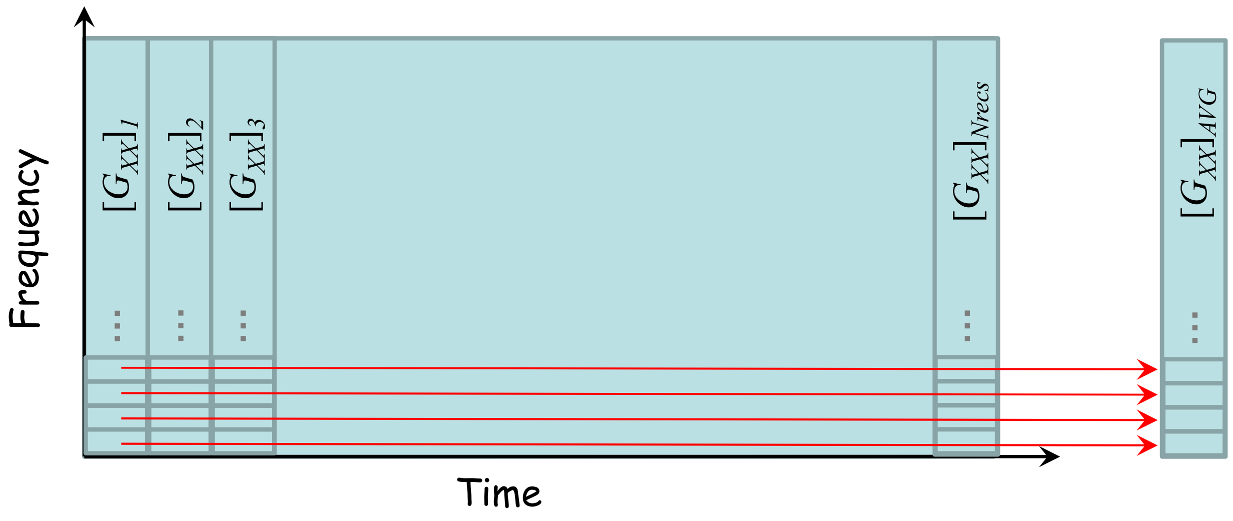
Once you are sure that your spectrogram is working properly, find a short wav file on the web or record a few seconds of some sound on your computer and save as a wav file. Generate the spectrogram of this file. Experiment with the fft size to see how that affects the display.

USEFUL TOOL: Modify your routine so that you can record a sound on your computer (see the appendix on Recording in the previous chapter) and generate a spectrogram of that sound.

Relationship of the spectrogram to mean-square averaging

There is an interesting relationship between the spectrogram and ordinary (mean-square) signal averaging. The spectral-density matrix for the spectrogram contains all the information necessary to construct the mean-square averaged spectral density. The figure below shows the matrix arrangement for the spectrogram with several red arrows overlaid. If this matrix is summed along the arrows and the results divided by the number of records, the averaged spectral density is produced.

Warning: to maintain the equivalence between integration of the spectral density and mean-square value in the time domain, this summation must be done using with the physical values (in Pa^2/Hz) of the spectral density, NOT WITH THE DECIBEL VALUES. *Summing decibel values is equivalent to* ***multiplication*** *of physical values*.



Given a spectral-density matrix, G_{XX_mat}, in suitable form for a spectrogram (but still in Pa^2/Hz, not decibels), the averaged spectral density over the time period of the spectrogram is (in MatLab/Octave) simply,

```
Gxx_avg = mean(Gxx_mat, 2);
```

The "2" indicates averaging over the second matrix dimension (the row dimension in MatLab). AFTER computing the averaged spectral density, you may convert to decibels for generation of the spectrogram:

```
p_ref = 20e-6;   %   in-air reference pressure of 20 micropascals
G_ref = p_ref^2;   %   frequency reference is 1 Hz
Gxx_mat = 10*log10(Gxx_mat./G_ref);
```

Many modern computer languages like MatLab, Octave, Python, Julia, and so on perform vector and matrix operations with particular efficiency. This has led to the recommendation to use vector and matrix operations whenever possible. If you want a challenge, you can construct a mean-square averaging routine with this philosophy. The *fft* function will operate on a matrix in a form similar to the spectrogram matrix. If you arrange the individual time records as columns[11], applying the *fft* will process the entire matrix returning linear-spectrum values

[11] Be careful to make the appropriate dimension modifications if you are using Python as matrix order (row/column) is reversed.

column-by-column. Then, you can calculate the corresponding spectral densities in place (and discard the upper "negative-frequency" values) and calculate the averaged spectral density as shown above.

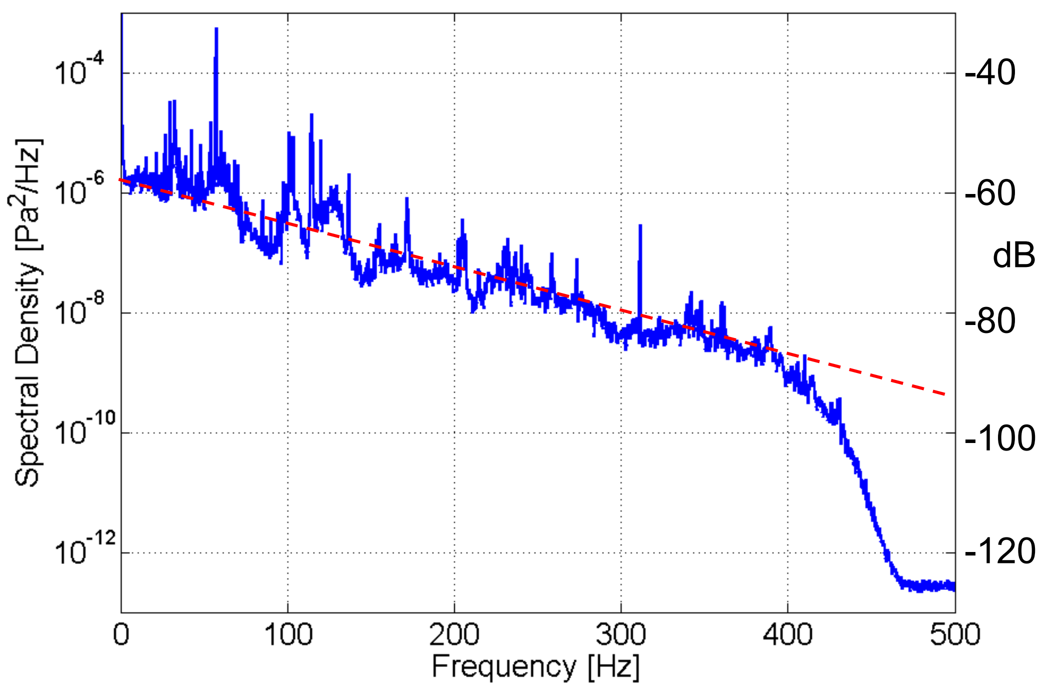
However, always give clarity and accuracy higher priority in your programming than cleverness and efficiency: there's no prize for calculating incorrect results more quickly than your co-workers or competitors. Write clear, well-documented code and check (line-by-line if necessary) your work thoroughly. Only after you have a solid piece of code should you worry about efficiency. Make sure the efficient version produces the same results as the correct version!

Enhancements

The tradeoff between resolution in time and resolution in frequency discussed above is a primary consideration in generating a useful spectrogram. Beyond that tradeoff, there are several possibilities for enhancing the display.

Pre-conditioning

Studying the averaged spectral density of a recording can suggest better ways of conversion to a spectrogram. The spectral density of outdoor ambient noise at a test location is shown below. The total range required to display the spectral density is about ten orders of magnitude (from 10^{-13} to 10^{-3} $\mathrm{Pa}^2/\mathrm{Hz}$)! A linear mapping from spectral density to color would lose almost all of the detail so the usual procedure of conversion to decibels (dB) is a wise first step. While this conversion would often be done relative to the normal in-air reference spectral density (of $(20\times10^{-6}\ \mathrm{Pa})^2/\mathrm{Hz}$), for simplicity, consider the conversion here as 10 times the common logarithm of the spectral density. (In other words, use a reference value of 1 $\mathrm{Pa}^2/\mathrm{Hz}$ for this example.)



Limiting the frequency range

This recording was made at a sampling rate of 1000 samples per second so the highest usable frequency is 500 Hz; however, the anti-aliasing filter starts rolling off shortly above 400 Hz so the information above 400 Hz has little value. Simply ignoring the spectral density above 400 Hz reduces the required range from 100 dB to 60 dB (roughly).

Limiting the decibel range

There may be large peaks or deep drops in the spectral density that are unimportant for your analysis. The *imagesc* function takes whatever the maximum and minimum values are to determine the color mapping. You can control the range in several ways (see the section below, Example of MatLab Coding). This is another effective way of concentrating attention on features you care about.

Pre-whitening the spectrum

Notice, in the figure above, that the interesting portion of the spectral density has an overall slope: lower-frequency values are, in general, higher than the values at higher frequency. This slope adds nothing to the spectrogram visualization but forces an unnecessarily larger range of values to display. If this overall slope persists over the time period for the spectrogram, the slope can be subtracted from the values in the spectrogram matrix. (Use the decibel values in the matrix to find and to subtract this overall background shape.) This operation of flattening the background is called pre-whitening[12].

While conversion to decibels does permit display of a wide range of values, subtle features can be exposed by expanding the range of color spanned by the interesting features.

Display/print resolution

Another subtle aspect to generation of spectrograms is the limitation in plotting or printing a figure. For example, if a spectrogram is produced using a 2^{14} fft size, then there would be more than 8000 frequency points. From a computational standpoint, this is fine; however, if your display (or printer) only has 1000 pixels in the vertical direction[13], the high frequency resolution of the computed spectrogram will be lost. Tonal components that are stable enough in frequency to stay within a few bins in the spectrogram *may not display at all* when the image is translated to 1000 pixels.

To solve this problem, display a limited frequency range (even if you have to make multiple plots). If you need high resolution in frequency, you probably can restrict your view to a fraction of the entire frequency range calculated. Make sure your frequency (or time) resolution is compatible with your screen display, your image generation, or your printer.

[12] Historically, pre-whitening was a popular technique in signal processing when hardware systems had small useable amplitude ranges; however, pre-whitening is still be a valuable, if not forgotten, technique.
[13] The monitor in front of me as I write this is set to its maximum resolution: 1280 by 1024. Even if I use the entire vertical extent of the monitor to display a spectrogram, I still only have 1024 pixels available.

Colormaps

A subtle problem can arise when converting the numeric values of the spectrogram matrix to colors. Some mappings to color have perceptual jumps—apparent large transitions for small changes in numeric value. The popular colormap, *jet*, has this issue. These perceptual jumps can cause "features" to appear in the spectrogram that are not associated with behavior of the data. Furthermore, a color plot may be reproduced as a black-and-white figure and the conversion may accentuate these jumps or introduce other problems. (*Jet* is particularly bad in conversion to grayscale.) Many color-blind people will have difficulty interpreting color mappings that contain both red and green.

These issues have been addressed recently by the development of better colormaps. The default colormap in MatLab was recently changed from *jet* to *parula*. *Parula* was designed to have a perceptually smooth transition over the entire range of color. *Parula* is proprietary to MatLab so other colormaps have been generated that are open source. The *matplotlib* used frequently with Python has a number of these modern colormaps available (e.g., *Magma*, *Inferno*, *Plasma*, *Viridis*) and several open-source colormaps have been adapted for use in MatLab.

Of course, grayscale is a fine option[14] although not as attractive as a color rendition.

Decimation

If all of the interesting spectral information is toward the low-frequency end of the spectrum, then you may have to use a large record length to resolve that information in the spectrogram. Consider, instead, decimating[15] the data first. For

[14] If you're basing a discovery on some subtle feature of a spectrogram using color, repeat the calculation using grayscale to be sure that feature is not an artifact.

[15] The process of decimation consists of low-pass filtering the original data and then retaining every Nth point. If the sampling rate is being reduced, the low-pass filter must suppress the higher-frequency spectral content to prevent aliasing. The MatLab function (in the Signal Processing Toolbox), *decimate*, performs both the filtering and down sampling.

example, if the portion of the spectrum that you want to study is below 1000 Hz and the data sampling rate was 48 000 samples per second, you can decimate by a factor of 16 to reduce the sampling rate to 3000 samples per second, which puts the new $f_s/2$ safely above 1000 Hz. Now a record length of 1024 points gives the frequency resolution that would have required 16 384 points per record with the original time series. Furthermore, the run time will be considerably shorter with 1024-point records.

Example of MatLab Coding

To use *imagesc* for constructing a spectrogram, you must provide three variables: a time vector, a frequency vector, and a value matrix. Suppose that these are called, *times*, *freqs*, and *dBvalues*, respectively. You will, in effect, be graphing the function, *dBvalues*(*times*, *freqs*). If you have been working with spectra, you shouldn't have any trouble with the frequency vector. At minimum, the vector, *freqs*, must have two values: the lowest and highest frequency for the spectral-density values in the spectrogram matrix. MatLab distributes the values uniformly between these two end points. (You may want to make the vector, *freqs*, hold all of the frequency values that correspond to each point in any of the spectra. MatLab will select the first and last value and ignore the rest.)

You must also generate the time vector, *times*. This vector must have at least the first and last time values that correspond to the spectrogram matrix. The time-vector values should be the mid-point times for the first and last records. (As for the frequency vector, you can construct the time vector to hold the time values for all of the records used; MatLab will take the first and last values and ignore the rest.) This will take some thought the first time you set up a spectrogram – check your time vector to be sure that it makes sense.

Once you have the three variables, constructing the spectrogram is straightforward. Since it is a graphics command, you should treat it as a figure:

```
figure(1)        %  Call the spectrogram figure number 1
imagesc(times, freqs, dBvalues);
axis xy          %  This command forces the vertical axis to increase upward
colormap(jet)  %  This command selects the color mapping for dBvalues
hcb = colorbar;   %  This command displays a color bar for relating color to value
ylabel(hcb, 'dB', 'fontsize', 16)   %  Optional label for colorbar
set(gca, 'fontsize', 18)  %  set font size for numbers on axes
xlabel('Time [s]' , 'fontsize', 18)
ylabel('Frequency [Hz]' , 'fontsize', 18)
```

Keep in mind that MatLab autoscales the color plot to accommodate the entire range of values in *dBvalues*. If you have a few very large values or a few very small values, you may lose definition of the features that you want to see. You can force the mapping by adding a fourth argument (a two-element vector) to the call to imagesc:

```
dB_min = -80;  dB_max = -20;
imagesc(times, freqs, dBvalues, [dB_min, dB_max]);
```

Notice how the two-element vector can be entered as a single argument. In this form, MatLab ties the bottom of the color scale to dB_min and any level below dB_min will be mapped to the color of the bottom of the scale and MatLab ties the top of the color scale to dB_max. The color scale will be mapped linearly to values between dB_min and dB_max. To make this work, you must specify both the minimum and the maximum values.

Alternately, you can force the autoscaling to show what you want by "clipping" the data. For example, if you want to set the lowest level for display to -100, the following line,

```
dBvalues(dBvalues < -100) = -100;
```

will set all values less than -100 to -100. (Notice the nice, compact method of specifying the values to change.) This technique allows you to set a lower (or upper) limit without restricting the upper (or lower) limit.

The version of spectrogram described above can appear blocky. Instead of using *imagesc*, you can use *surface* to generate the spectrogram image. *Surface* requires *x*, *y*, and *z* (time, frequency, and dB, that is) coordinates *for every point* so it requires about three times the memory as *imagesc*; however, the *shading interp* command can be used to smooth (interpolate) the display. As discussed earlier, overlap in time and zero-padding may provide adequate smoothing without resorting to *surface*.