Lecture 3

Spectrograms

It is tough to get timing info from a FFT: we saw that back in the Week 2 lecture, on Power and Phase. In fact, the FFT had a hard time telling whether things were going forward or backward. The information is completely lost when we look at power alone; it is kept in the phase, but is coded in ways that are hard to read.

The spectrogram is a clever trick to get time info. Say you have some data sampled at 1024Hz, and you have 3min of data -- 3072 data points. Chunk the data into pieces of size 128, and take a 128-point FFT. Each FFT will represent what has happened over an eighth-of-a-second. So you can track how the frequency changes over time!

Unfortunately, it ain't so easy: the Uncertainty Principle also has a word or two to say. A 128 point FFT of data sampled at 1024Hz can only see 8 frequency chunks. So you lose a good sense of what frequencies happen when.

That's uncertainty. The hope is, that by adjusting the size of the chunk, you can detect the frequencies you want. The technical term for "chunk" is "window", and you'll want to adjust window size.

There's a second issue: say your window is 128 points long. Win1 covers points 1 to 128; Win2 points 129 to 256, etc. But what if the event you want to see lasts from point 120 to 175? It's a long event, so it's no good trying to change window length. The problem is with *where* the windows are placed, not how long they are. If only the original window could be slid back eight points. But then it'd overlap eight points with the data in Win1....

So, while we're thinking of programming this, we also ought to implement overlap. Just in case.

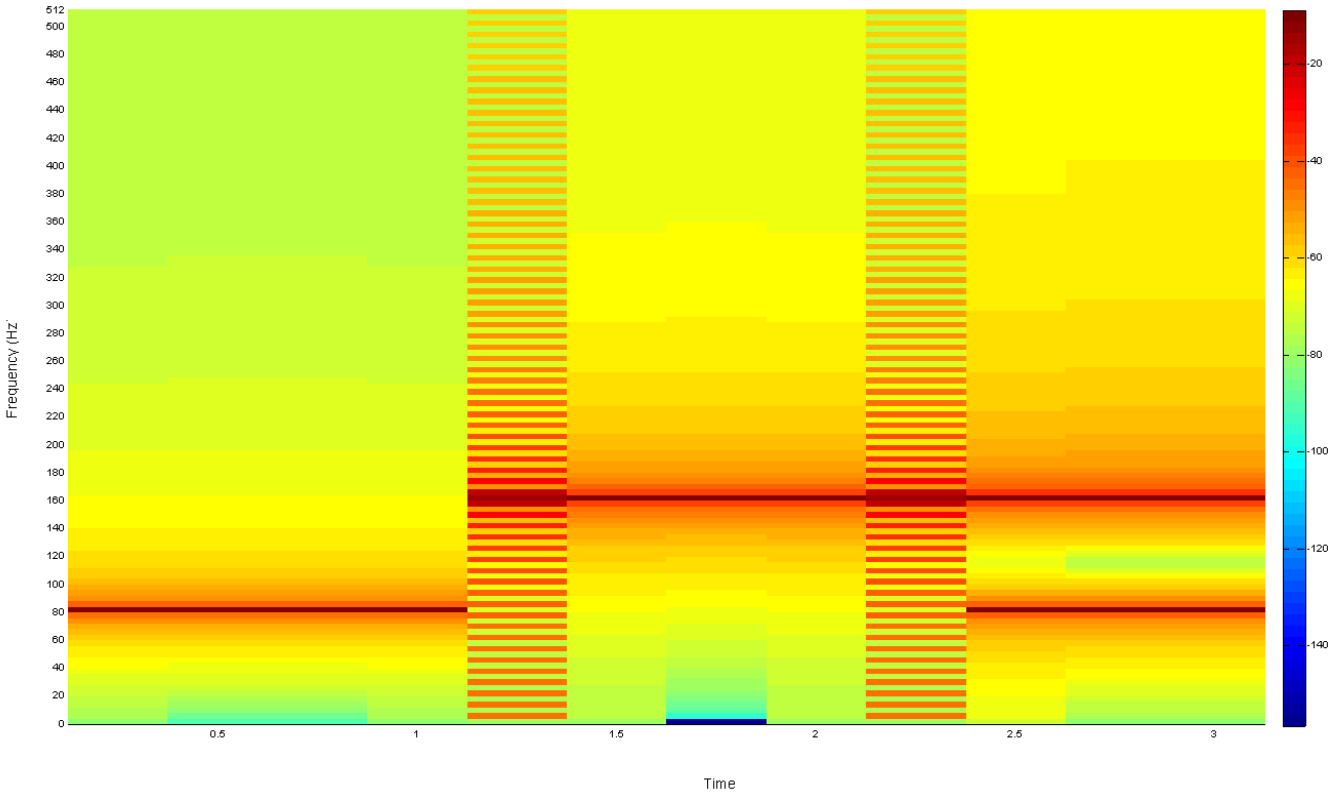
Finally, we're getting a bit too much data: we have frequency, and power -- but now we have time. That's a 3D graph!

Fortunately, there's a Matlab function that does it all; it's called spectrogram. Let's try it.

```
>> x=linspace(0, 1, 1024);

% I'm taking the interval from zero to one as my basic unit of time, and pretending that means
% "one second". I chunk it into 1024 pieces, so this gives me a sampling rate of 1024Hz.

>> s1=sin(2*pi*80*x);
>> s2=sin(2*pi*160*x);

% Two sine waves, at different frequencies. I want to see how uncertainty affects my
% ability to see the two frequencies.

>> s=[s1 zeros(1, 128) s2 zeros(1, 128) s1+s2];

% A signal starting with s1, pausing for 128/1024 sec, then s2, another pause, then
% the two frequencies together.
```

Now comes the big one:

>>spectrogram(s, ones(1, 256),0,256, 1024, ‘yaxis’)



Nice picture ... what’s it mean?

Let’s start with spectrogram(S, B, C, D, E, ‘yaxis’)

S is the signal.

B tells the window to use. ones(1, 256) gives bunch of zeros, followed by 256 ones, then back to zeros. A simple on/off window of length 256 points. You could taper at the ends, do anything you wanted, by changing ones(1, 256) to a different set of numbers.

C tells the overlap between windows; here it’s set to zero. This will make the graph look chunky: chunks of length 256, to be exact!

D tells me how long a FFT to use. 256 seems to make sense with a length 256 window...

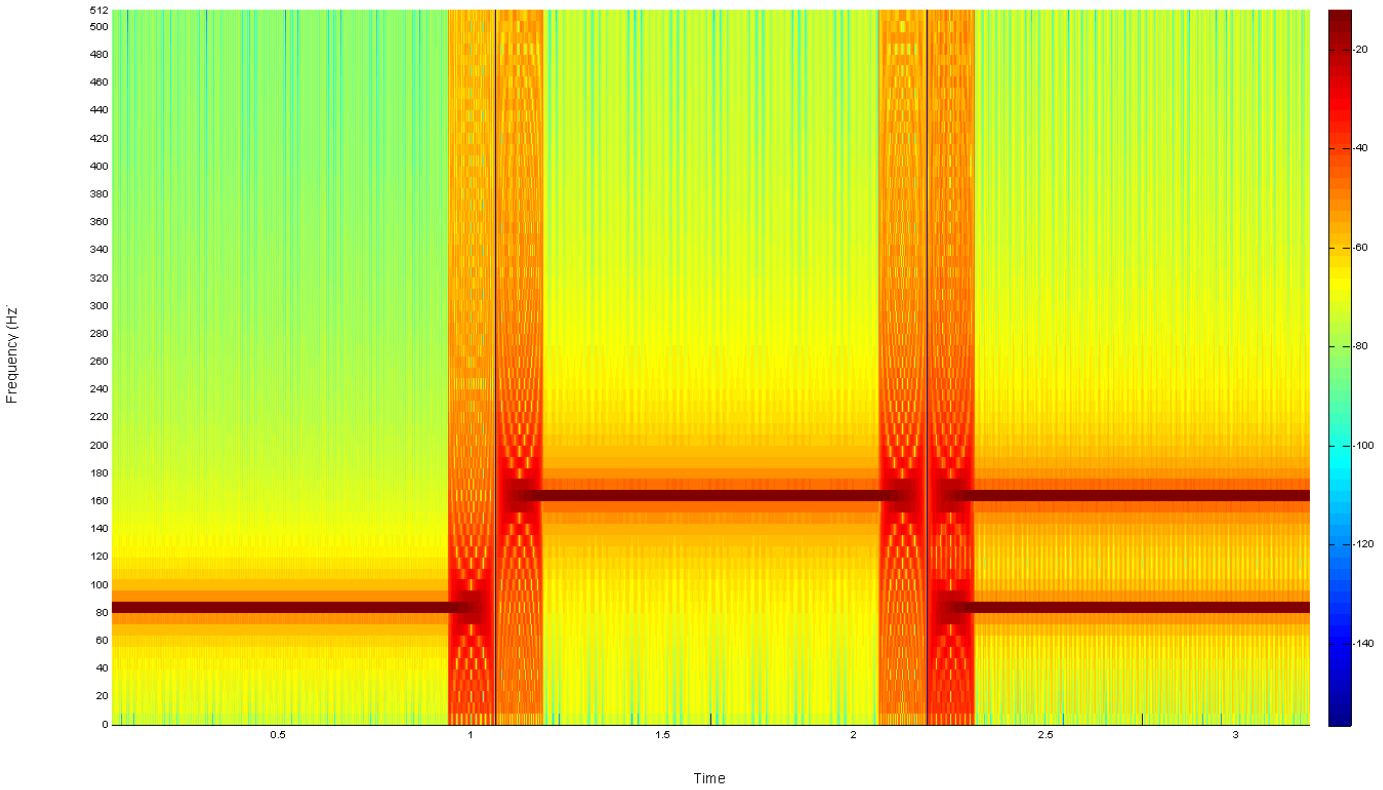
E tells me the sampling frequency. That allows Matlas to label the x-axis with the correct time, and the y-axis with the correct frequency.

And speaking of: ‘yaxis’ tells Matlab to put the frequency along the y-axis.

One second here: if x-axis is time, y-axis frequency, then z-axis is power. But -- spectrogram uses a trick: it uses false color to indicate the z-axis. (Actually the z-axis is still there, but for now think: color=height)

For comparison:

>>spectrogram(s, ones(1, 128),127,128, 1024, 'yaxis')



Here we have a shorter window, which makes things look less chunky, and we have an overlap of 127 points -- which produces a very smooth looking graph. With much amazing and interesting detail.

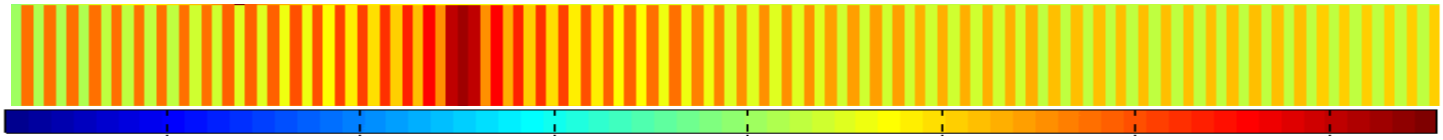
Very pretty, and one can spend hours, harmlessly playing with spectrograms. Let's look at the information we can get from it. Start by magnifying the picture until you can see the scales. The y, or frequency, axis runs from 0 to 512. That's because the sampling rate is 1024Hz, so the Nyquist limit is 512Hz. Similarly, the x or time axis is a bit over 3 seconds. That's because we took three one second sounds with two eighth-second pauses.

Now look at the picture itself. The color bar at the right tells us that the deeper reds represent the frequencies where the power spectrum is highest. In the first second of time, that's concentrated between 80 and 90 Hz. We in fact know that the s1 signal is a sine with 80Hz. We've lost 10Hz of resolution! That's about right, with a FFT of length 128: 1024/128 = 8Hz chunks. If you look at the spectrogram on the previous page, the power is much more clearly centered mear 80 -- in fact, the chunks are size 4Hz.

The next feature to note is the transition between s1 and s2. The spectrogram on the previous page has a wide bar, of length about .25 second, that represents the gap between s1 and s2. Let's check that: the spectrogram on that page has length 256 points. 256/1024 is indeed 1/4 of a second. This FFT cannot discriminate any smaller time interval, so the color of the bar, representing the power in that section, is constant across the whole .25sec.

For the picture on this page, that transitional interval is about 1/8 of a second long, which is exactly correct. However, there's something troubling in the computation: the interval between s1 and s2 is filled with 1/8 second of silence. There should be NO spectral power in that region. Yet both the spectrograms show some deep red. What's this about?

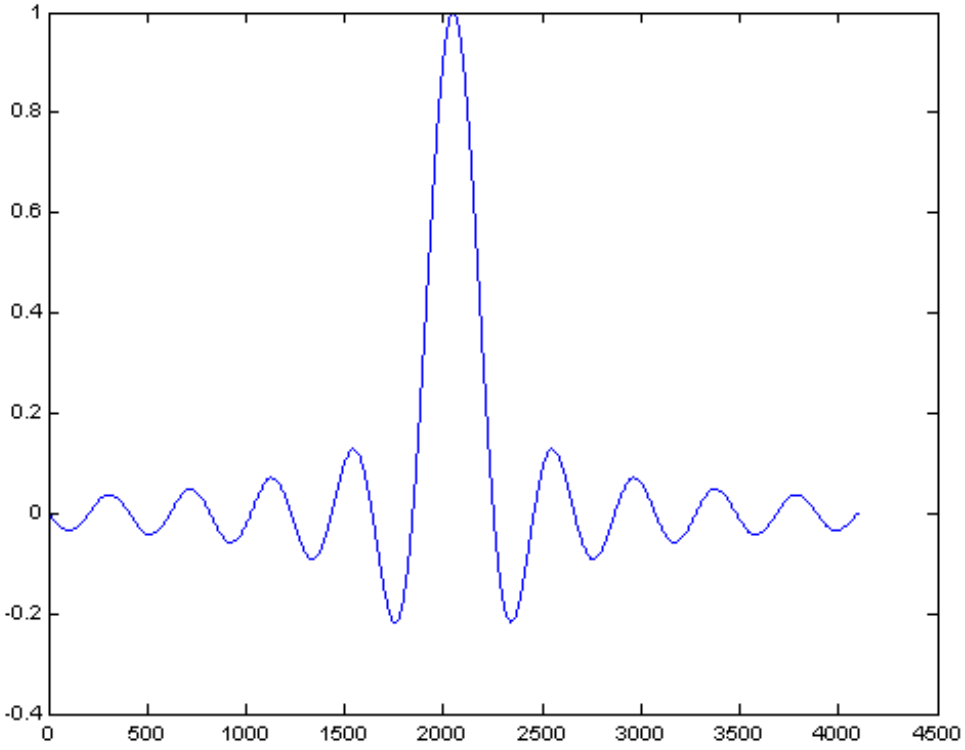
Let's look more closely at the gap: I've turned it on it's side.



The color is deepest red right at 160Hz, the frequency of s2. But there's an additional regular pattern of high-power bars, which are the reds, alternating with lower power greens. Moreover, the reds and greens both fade as we go further from the center frequency of 160Hz.

This pattern is the color of a Dirichlet kernel: from Week 2,

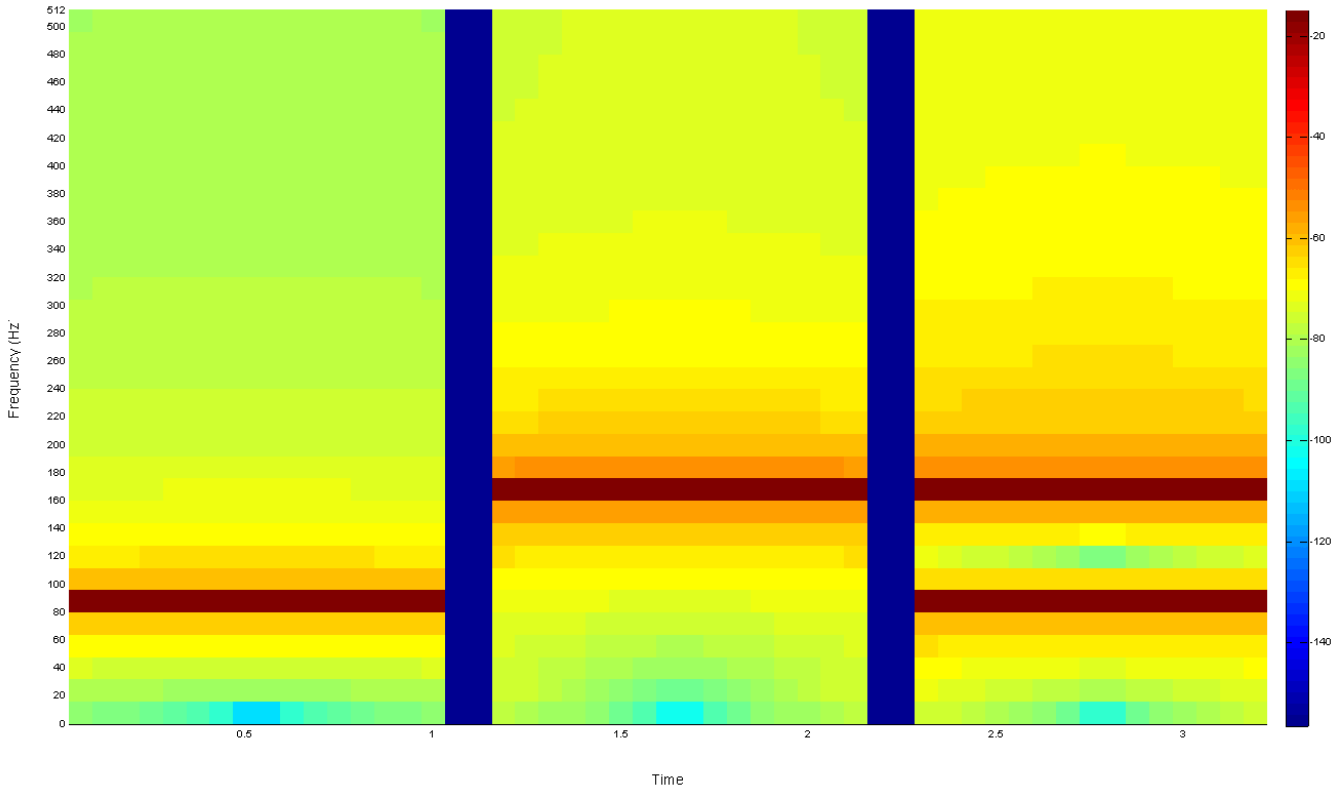
$$Dirichlet \;=\; \frac{\sin(\pi N\,[\,\omega/M - n/N])}{\sin(\pi\,[\,\omega/M - n/N])}$$



The bars are there because we chose a simple window -- instant on, instant off. The FFT of the window is a Dirichlet function. In short: the bars represent the FFT of the window, not the FFT of the empty space. This provides a powerful reason to take windows that aren't instant on/instant off.

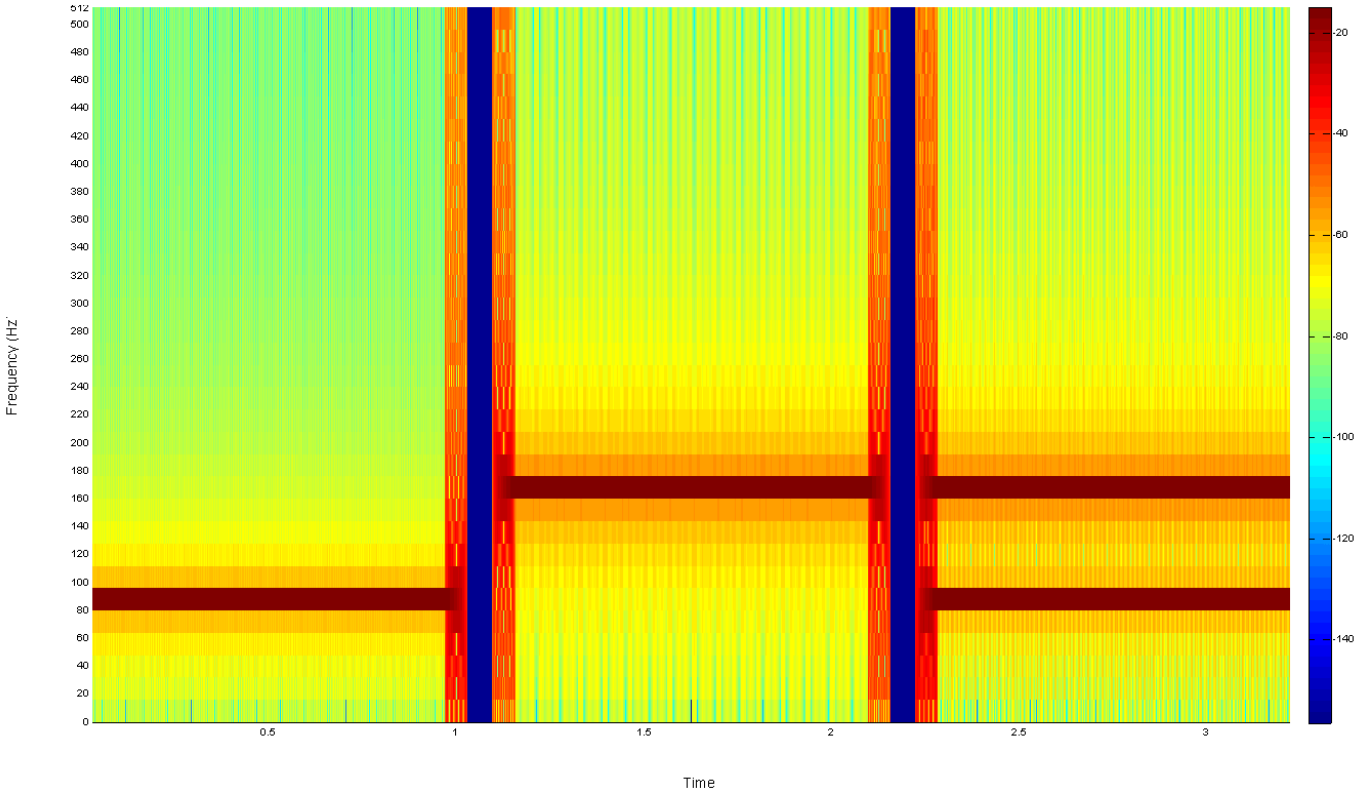
There's one more lesson to learn from the spectrogram:

>>spectrogram(s, ones(1, 64),0,64, 1024, ‘yaxis’)



The gap between s1 and s2 is represented by a thick blue bar. The color chart shows that blue represents the lowest possible power. This spectrogram gets it right! How does that happen? First, because the window size is 64, which means that the window fits neatly into the gap: in this window, there truly is no power in the signal, and the FFT reports this accurately. Second: there’s no overlapping-- we set the overlap to zero -- so the window doesn’t slop over into portions where there’s s1 or s2. Compare with

>>spectrogram(s, ones(1, 64),63,64, 1024, ‘yaxis’)

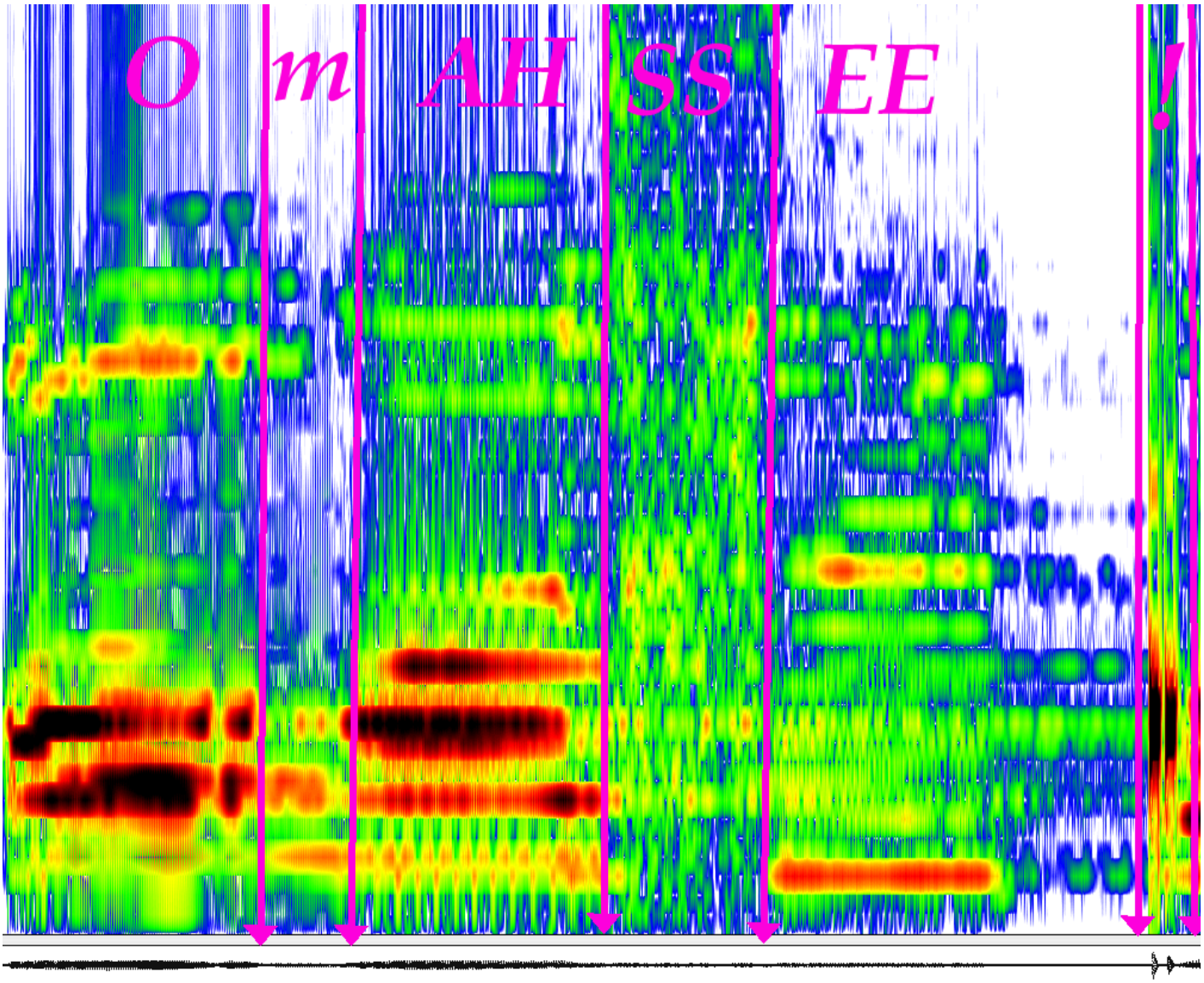


The window size is still 64, so it still sits inside the gap. But the overlap of windows is about 98%, and so the empty space is contaminated.

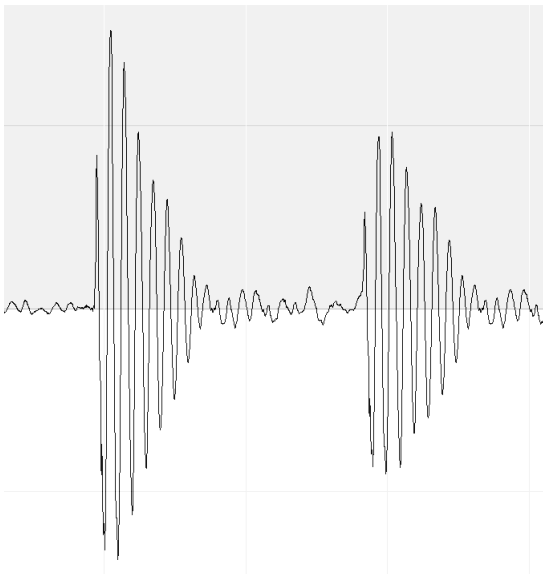
Lab Project: Let's start with hayat.wav, Hayat Sana Tesekkur Ederim, from the album Deliveren by the Turkish singer sezen Aksu. It's a nice song to do a spectrogram on; it starts with voice, then a simple and finally a more complex orchestral accompaniment. You get to see where the spectra of voice & instruments lie. It is sampled at industry standard 44100Hz, which may be too much data for your computer. In that case, downsample by a factor of eight or sixteen!

Next, Isilongo.wav, from the Mahotella Queen's song Isilongo Sesoka. In Isilongo, the singer uses one of the clicks in an African click language. Many linguists believe that the click languages are the 'oldest' human languages (what can *that* mean?). Your task is to use a spectrogram to locate the click. Then fiddle with the window length until you can locate the beginning of the click as accurately as possible. Now plot the sound itself, and magnify it using Matlab graphing tools, to locate the beginning of the click. How do the two estimates of the click onset compare?

For the next application, we're going to look closely at the first four phonemes of the song Isilongo.



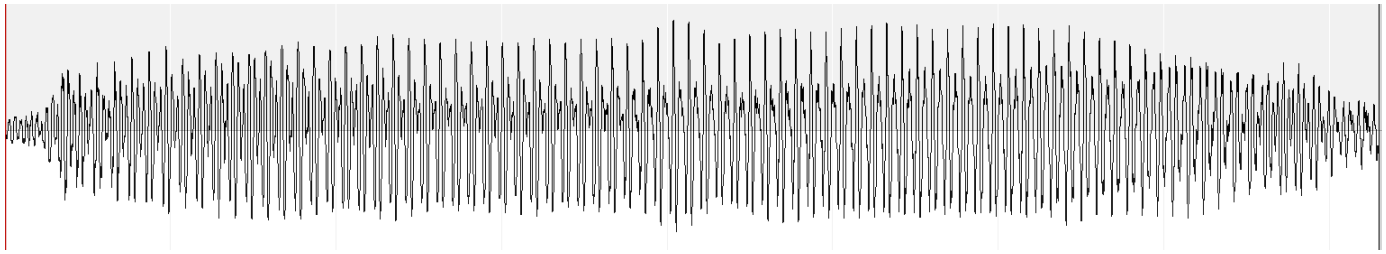
Of course the click ! looks very different from the other sounds. Here's what the sound -- not the spectrum-- looks like:



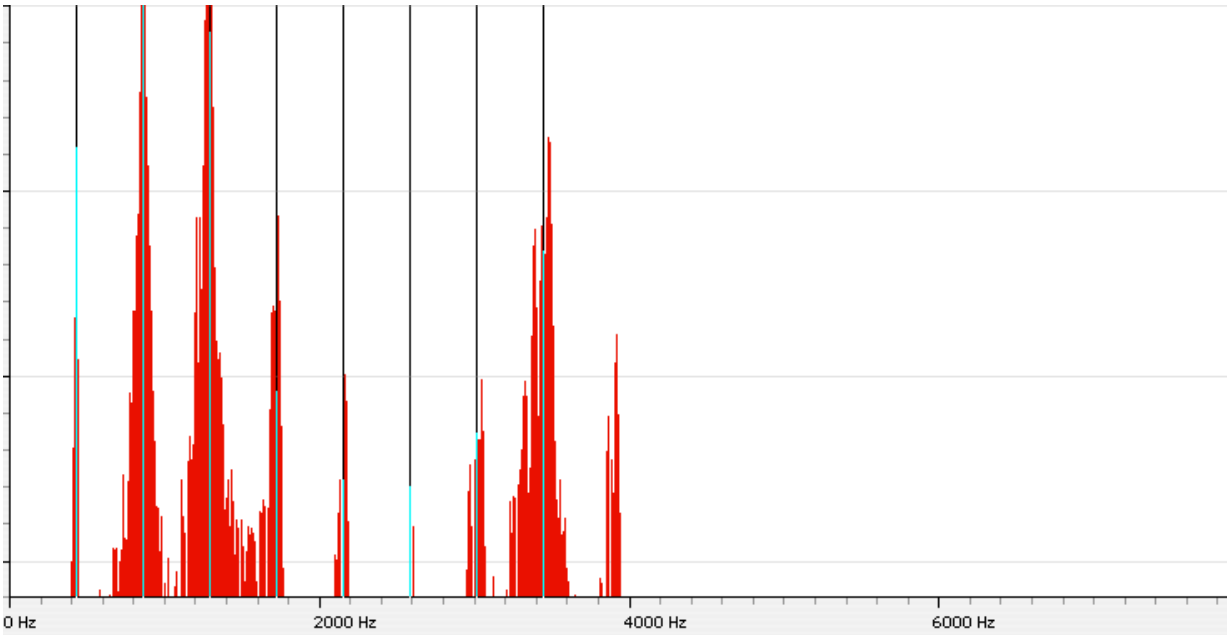
It's reminiscent of the artificial clicks we constructed in the Uncertainty supplement of Chapter Three. Those were characterized by a fundamental frequency -- which you can see as a dark band in the spectrogram -- followed by high frequency harmonics, but falling off slowly in power.

We'll be focusing, however, on the vowels O, AA. EE, which have very different spectra than the consonants. Therein lies a story, about speech production in humans.

In the vowels O and AH, you see very dark bands, regularly spaced. Isolating the sound, it looks like:



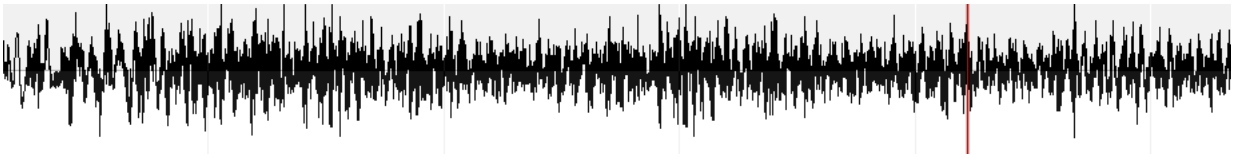
And the spectrum confirms what the graph suggests:



The spectrum consists of a fundamental and some fairly noisy harmonics. The reason for this has to do with the way the vocal tract produces vowels. Air is forced from the lungs, through the vocal folds, which vibrate and are

re-inforced through the vocal tract. It isn't very different from our examples from the Exponentials supplement to Chapter Two, where we looked at spectra of a chime and a didgeridoo. Vowels are produced by the same kind of open tube, forced at one end.

Contrast the SS sound:



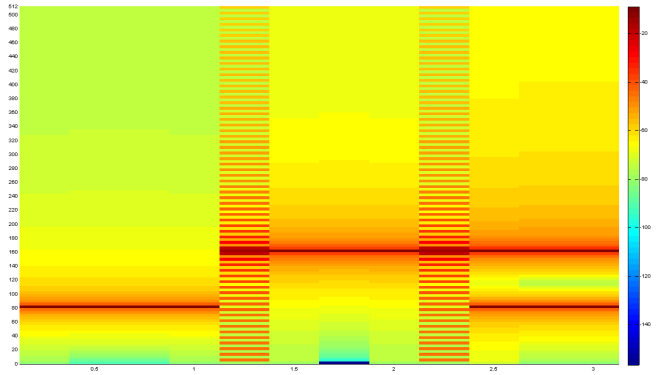
The last thing it looks like is a fundamental and one or two harmonics; it looks more like random noise. And, if the spectrogram is any evidence, it has much more high frequency content than a vowel does.

How are consonants like SS produced?

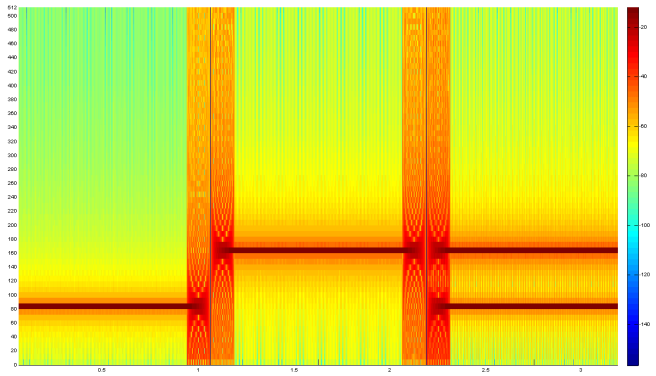
Finally, a point from the baby boomer set. As humans age, their perception of high frequencies deteriorates. This makes it difficult to detect consonants, and because of this, speech can sound like a collection of vowels all blurred together. Any system that can boost consonant volume, while leaving vowel volumes alone, might be used as a hearing aid. Here's a paper on the basics of using wavelets for detecting and altering the different components of human speech.

Finally, we'll look a little more closely at the spectrograms. Compare

>>spectrogram(s, ones(1, 256),0,256, 1024, 'yaxis')



>>spectrogram(s, ones(1, 256),255,256, 1024, 'yaxis')

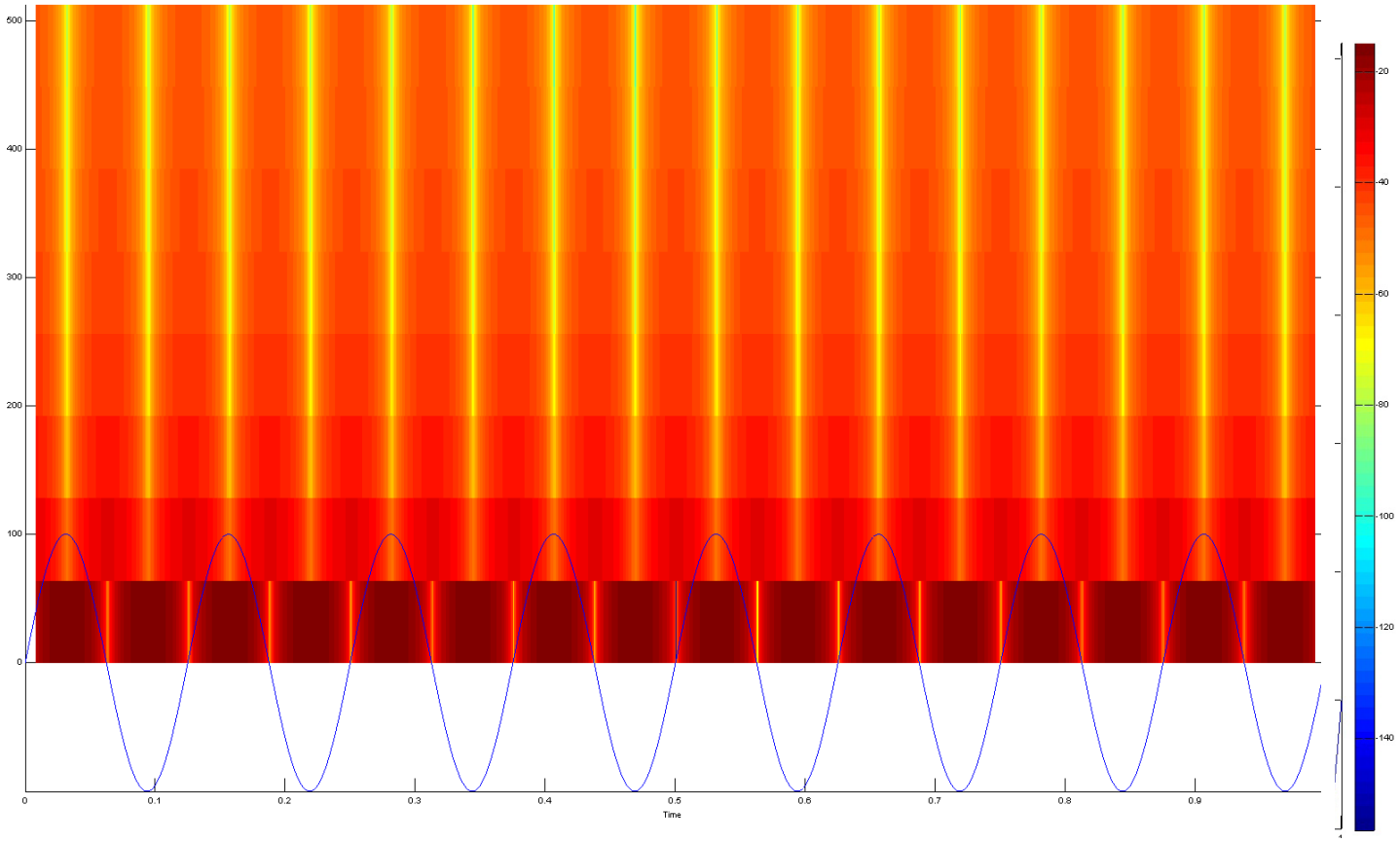


The spectrogram with the massive overlap seems to have all kinds of very fine detail -- even in the first third of the graph. What is that?

We'll take a simple model:

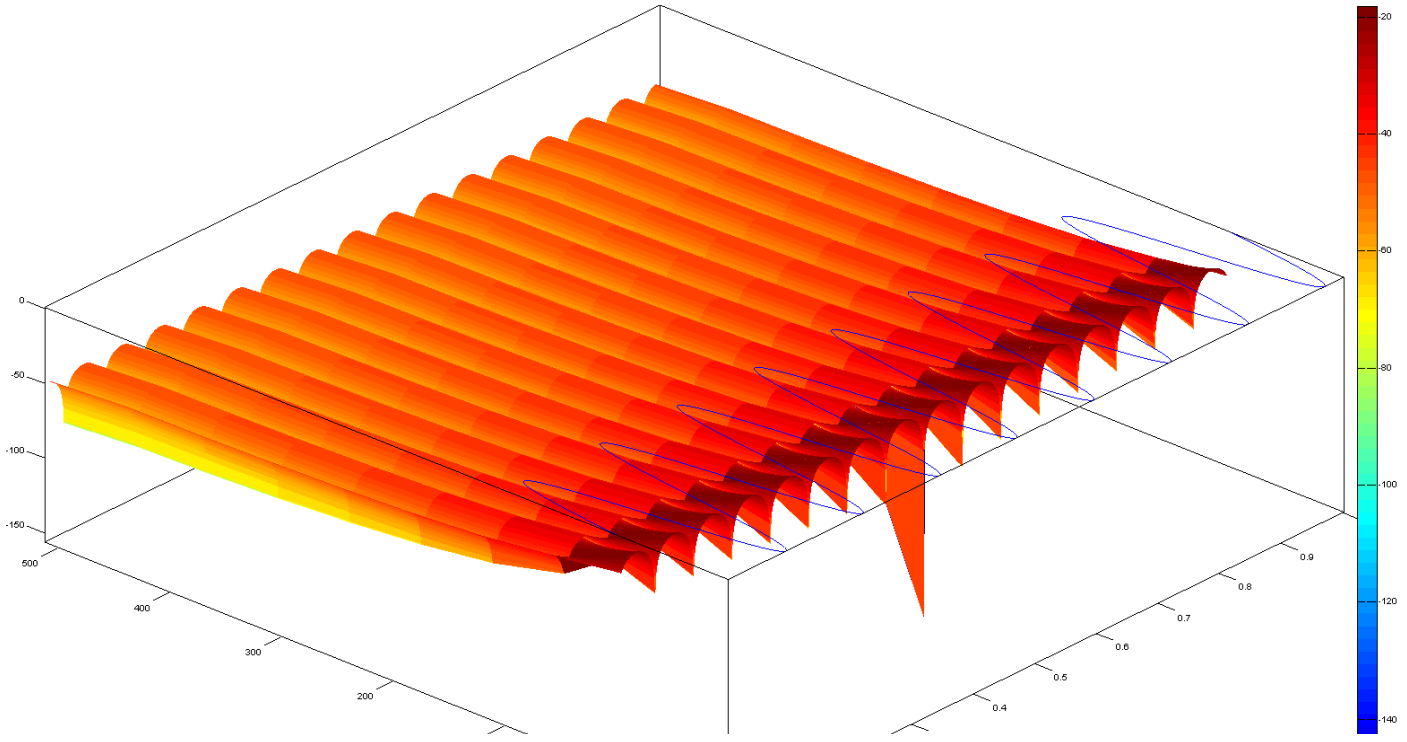
```
>> x=linspace(0, 1, 1024);
>> s =sin(2*pi*8*x);
>> plot(x, 100*s)
>> hold
>> spectrogram(s, ones(1, 16),15,16, 1024, 'yaxis')
```

Here's two views of the same spectrogram:



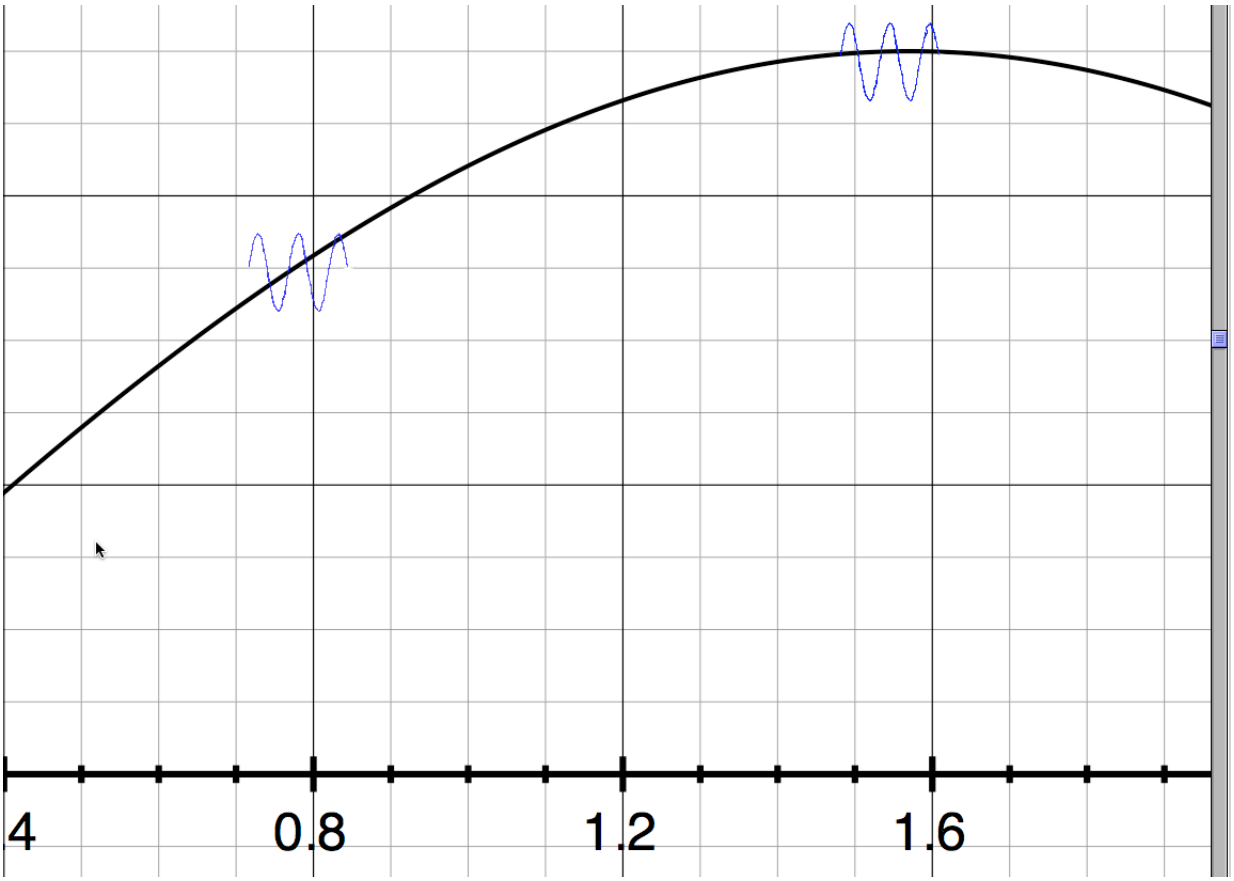
This is the “fine detail”, in close-up, and the blue curve is my sine curve. Looking at it, it’s tempting to think, ‘wow! the spectrogram is picking out the individual peaks of the sine curve. Groovy! Unfortunately, red indicates power; yellow indicates significantly less power.

This would be easier in 3D. And Matlab has this amazing tool, the ‘rotate’ tool, that allows just that. Here’s the same graph, but rotated so the high and low are clearly visible:



This is a good time to ask, 'what is going on here?' Start by following one frequency in the FFT. We've forced it into a window of length 16, so its real and imaginary components look like tiny sine or cosine waves.

In contrast, the signal s itself has frequency 8Hz, so it takes 128 points to go through a cycle. In contrast to the FFT, it is a long slow wave. Moreover, as the window overlap is a full 16 points, the tiny wave is moved, point by point, against the large wave. The FFT computation looks something like the (exaggerated) view below:



Here the little blue wave represents the exponential used in computing the power in the FFT. So: which of the two blue waves above give the greatest power? The way to compute the FFT is to multiply the exponential by the function, and integrate.

The first little wave is in a region of the sine curve that looks approximately like $f(x) \approx x$ the second, $f(x) \approx 1 - x^2$ (Taylor series computation would show that in fact, $f(x) \approx 1 - \frac{1}{2}x^2$). Then the contributions to the energy at each of the two locations would be

$$\int_{-\pi}^{\pi} x \cdot \sin(5x)\ dx; \qquad \int_{-\pi}^{\pi} (1 - \frac{1}{2}x^2) \cdot \sin(5x)\ dx = -\frac{1}{2}\int_{-\pi}^{\pi} x^2 \cdot \sin(5x)\ dx$$

We could integrate by hand, or have Matlab do the integral; instead we've brought in another program, Mathematica, to do the integration.

In[2]:=
Integrate[x*Sin[5x], {x, -Pi, Pi}]

Out[2]=
2 Pi

5

In contrast,

```
In[3]:=
Integrate[x^2*Sin[5x], {x, -Pi, Pi}]

Out[3]=
            2                 2
2 - 25 Pi    -2 + 25 Pi
---------- + ----------      = 0
  125           125
```

Even if you're a bit off from being exactly centered at zero, the contribution is small:

```
In[5]:=
Integrate[(x-.01)^2*Sin[5x], {x, -Pi, Pi}]

Out[5]=
                            2
0.0799 - 0.02 Pi - Pi
---------------------- +
         5

                              2
  -0.0799 - 0.02 Pi + Pi
  ----------------------
            5
```

Whoops! What is that as a number?

```
In[6]:=
NIntegrate[(x-.01)^2*Sin[5x], {x, -Pi, Pi}]

Out[6]=
-0.0251327
```

The little blue wave sees a much larger contribution from the straight-line portion of the curve; very roughly, the little blue wave picks out the slope of the curve. And that slope is near zero near where the large function peaks. So that's what the fine detail in the spectrogram represents, very roughly.

By the way -- must we say 'little blue wave'? It sounds like the wave in the big blue house.

There is a fancier name: "wavelet". Which is where we go next.

Finally -- thought! -- we might want to use the spectrogram to try to understand the signals we already have. Try doing spectrograms of the signals

```
z1=y.^2.*sin(2*pi*8*y + cos(2*pi*3*y));
z2=sin(2*pi*8*y^3);
```

In z2, can you detect the instantaneous frequency?