Lecture 2

FOURIER TRANSFORMS

Sampling

What if we lived in the world of The Matrix?.

Daniel Dennet, in his book on Consciousness, imagined just that, but he went a bit further. He asked, imagine if we were in the Matrix, and a computer was pumping data into us, to make is think we were free, living in the real world. How much data, how fast would it have to get into us, to sustain the illusion?

It's a fun computation, and after you do it, you see the supercomputers of The Matrix are just bad dreams: no real computer could get that much data, that fast.

But you also get a feel for what the pioneers of motion pictures had to do. After all, they were trying to create a (two-dimensional) version of reality. And the only tool they had was the picture. More accurately, a sequence of pictures, strung out on a long piece of film.

Talk about data rates! How many pictures per second would you need, to make people believe they were seeing real life?

We ran a little experiement, a clip of Marlene Dietrich's famous chanteuse scene with Gary Cooper, and made our own little films. They're titled md2.mov, md4.mov, md8.mov. Or you can view them from the CD, in the folder ch2/sampling. The clip md2.mov has two pictures per second, etc

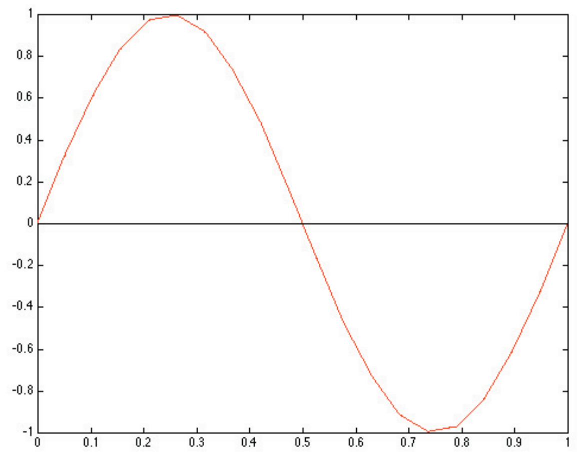
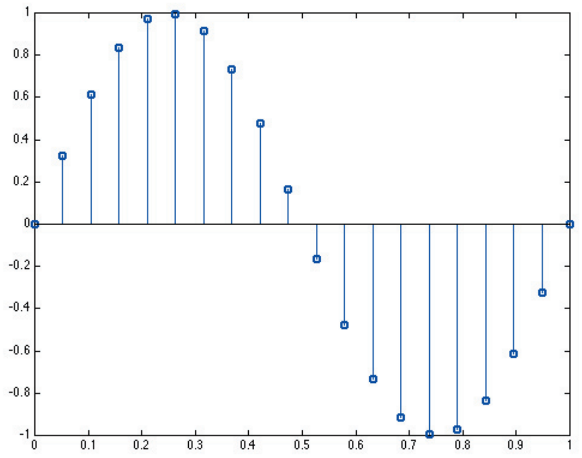
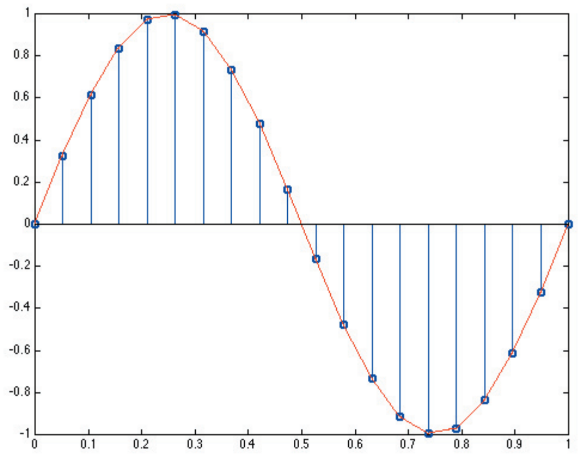
Play the clips. Of course md2 isn't going to fool anyone, but by md8, at 8 pictures per second, things seem pretty real.

I chose the Dietrich clip because there's a lot going on in the background behind Dietrich. People are fanning themselves, people are applauding. These are periodic motions. Try tracking some of the fans and some of the applause, in md2 through md8. Notice anything special?

In doing digital music, we have the same problems as the computers of the Matrix. Say you want to input one period of a sine wave, into a computer -- sin(2*pi*x) for x between zero and one. Well, it can't be done: there's an infinite number of points between zero and one. Take a computer with terabytes and terabytes of storage . . it'll never even get close to all that's stored in the numbers between zero and one.

(By the way, if you think the answer is a compression scheme, there's a nice theorem: most numbers between zero and one cannot be compressed).

The solution, the way to get music onto a computer, is the same solution the Hollywood producers used. Take snapshots of the function. This process is called sampling. And it isn't hard to see that if you sample at enough points, just like in the movies, things look real.

On the left, a graph of y(x) = sin(2*pi*x) for one period of the sine; on the right, the sin taken at only 20 points of the interval. Below, the two graphed together. Technically, what you have is y[j] = sin(2*pi*j/20) for j = 0, 1, 2, ... 19. Notice that the space between samples is (j+1)/20-j/20 = 1/20. The 20 represents the number of samples per second (assuming the x-axis here is time, and 0<x<1 represents one second!!). The number 20 is called the sampling frequency, and is usually denoted by an F or F with a subscript s. . It has units "samples per second". If F measures samples per second, 1/F measures the number of seconds which elapse between samples. Thus 1/F has the dimensions of a time, and it is usually written as T. It represents the time between samples. The "sampling times" themselves are jT = j/F. These are "equally-spaced" samples; we talk about non-equally spaced sampling at the end of the section.

In the function sin(2*pi*f*x), the variable f is called the frequency and is measured in "cycles per second" because, when x goes from zero to 1, 2*pi*f makes the sine go through f cycles. The number 2*pi*f is measured in "radians per second". When you do sampled functions, sin(2*pi*f*jT), f is now measured in "cycles per sample" and 2*pi*f*jT is radians per sample. Again, if T=1/20, then as j goes from 1 to 2, say, 2*pi*jT goes through 1/20 of a cycle of sine, and 2*pi*f*jT goes through f/20 cycles of a sine in one sample; hence cycles per sample.

Sampling is a form of compression. Instead of the pairs (x, sin(2*pi*x)) for an infinite number of x's, we have pairs (jT, sin(2*pi*jT)) for different j's. In fact, if we actually know the sampling frequency T, we can omit the jT altogether. This is the approach used in modern digital music; the sampling frequency is 44100 samples per second (an interesting number), or for professional equipment, 48,000 or 96,000 samples per second (we'll say more about these numbers later). This is the way music is encoded on a modern CD.

When the CD first came out, critics said, "you'll be able to hear the gaps between the music". It's a perfectly reasonable fear; something will be going on while we're not looking.

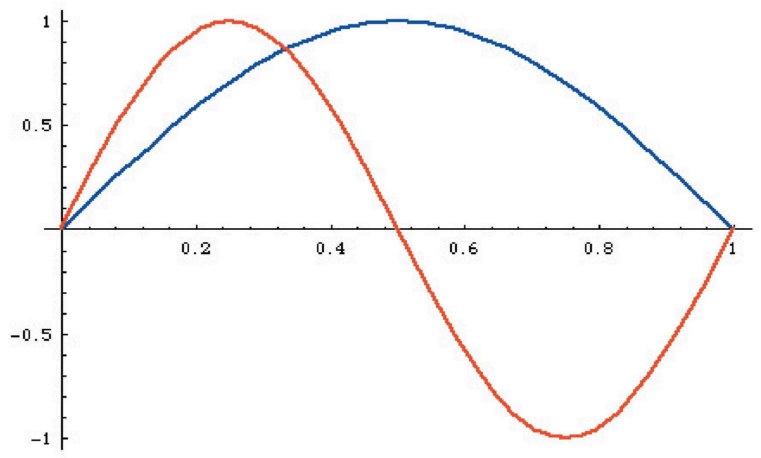
On the other hand, we sample all the time. When you're crossing a street, you look both ways. But you can't look both ways, so you look one way, then the other, then back again to see if anything has changed. You're sampling. While you're looking the other way, won't a car sneak up on you?

Well, no: not if the cars are travelling 35mph. We pretty much can see the street is clear and it's going to stay clear for awhile. On the other hand, if cars travelled 500 miles an hour, we couldn't take my eyes away. We might indeed miss something, while I was turned the other way.

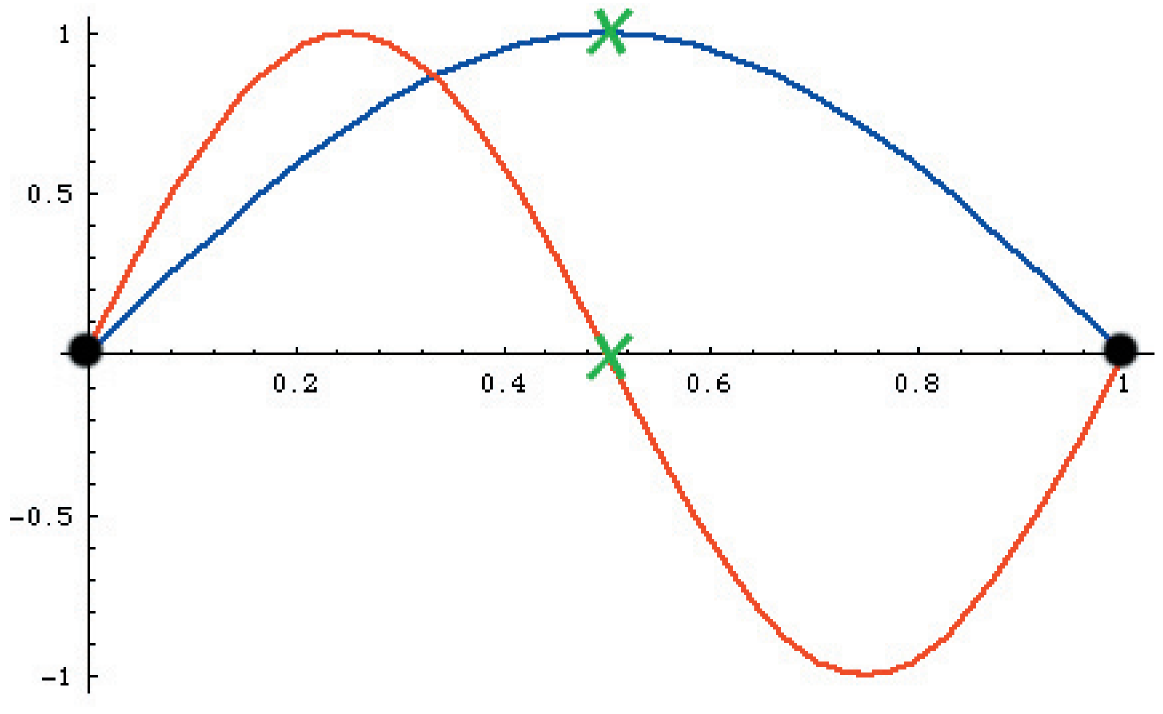
How fast you have to sample depends on how fast new events are coming in. The simplest way to model this for music is to look at sampling a sine wave, y = sin(2*pi*f*x). How fast must I sample it, in order to tell what the frequency is? (how small is T?).

My intuition is, the higher the frequency, the faster the sine is changing(the derivative is the speed of change, and y'=2*pi*f*cos(2*pi*f*x), which is proportional to the frequency f. So: to detect a frequency f, I figure I have to sample f times per second.

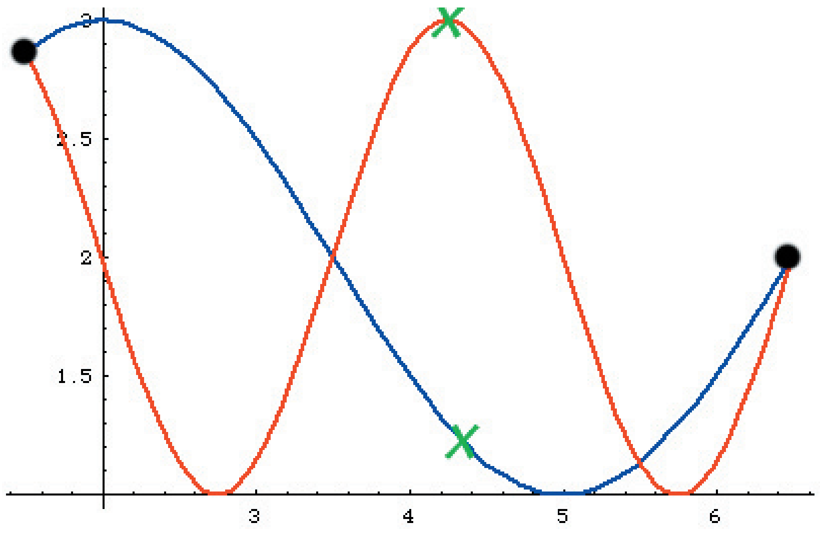
Which turns out to be wrong. Check out the picture below, in which y = sin(pi*x) is graphed in blue, and z = sin(2*pi*x) in red.



To be able to tell the difference, we need to sample . . well, once? No, I don't think so. If F=1, y[0/1] = z[0/1] so we'll need to sample another time to distinguish the two. Try T=1/2, F = 2:



There! If T = 1/2, j=0, 1 and we sample at 0/2, 1/2. Again y[0/2] = z[0/2], but y[1/2] ≠ z[1/2]. And, the picture below shows this isn't just for sines defined on 0<x<1.



So there's your sampling theorem: in order to capture the correct frequency of y=sin(2*pi*f*x), you have to choose a sampling rate of F greater than or equal to 2*f. 2*f is called the *Nyquist sampling frequency*, after the British mathematician Whittaker, who discovered it. Really, after the communications engineer Nyquist who rediscovered it. 1/T ≤ 2*f or T*f ≥ .5

If you fail to choose Nyquist, you'll miss out on some frequencies. But that turns out to be a deceptive way of speaking. Like when I'm crossing the street and I "miss seeing a car coming" is a deceptive way of saying "I'm dead."

We want to look at a couple of examples, to see what happens when we “miss” frequencies. The first goes back to the Dietrich movie clip. Compare md8 with md2 again, focusing on the gentleman who offers Dietrich a glass of champagne. First look at md8. The scene, right after Cooper salutes her and she tips her hat to him. The gentleman is fanning himself rather quickly. If you go to md2, however, the fanning almost disappears; it looks like a kind of fluttering, like the gentleman can’t keep his hand steady.

The fan, however has not disappeared. One way of thinking about it is that the sampling has distorted the frequency of the fan motion. A high frequency fan motion has been made to appear like a lower frequency fan. Just like with our sines, y = sin(pi*x) appearing the same as z = sin(2*pi*x), when we sampled at T=1 instead of T=1/2.

This distortion is called *aliasing*; a high frequency looks like a lower one. You can predict it exactly. Here’s the math:

Let’s say you’re sampling at a rate 2F; the highest frequency you can distinguish then is F. So what happens to a frequency F+f, when you sample at points j/2F? What does F+f alias into?

$$\begin{aligned}
&\sin\left(2\pi(F+f)\frac{j}{2F}\right) = \\
&\sin\left(2\pi(F)\frac{j}{2F}\right)\cos\left(2\pi(f)\frac{j}{2F}\right) + \cos\left(2\pi(F)\frac{j}{2F}\right)\sin\left(2\pi(f)\frac{j}{2F}\right) = \\
&\sin(j\pi)\cos\left(2\pi(f)\frac{j}{2F}\right) + \cos(j\pi(F))\sin\left(2\pi(f)\frac{j}{2F}\right) = \\
&0\cdot\cos\left(2\pi(f)\frac{j}{2F}\right) + (-1)^j\sin\left(2\pi(f)\frac{j}{2F}\right) = \\
&(-1)^j\sin\left(2\pi(f)\frac{j}{2F}\right)
\end{aligned}$$

Whereas if we sample a sine with frequency $F-f$ at the same j, you’ll get the $(-1)^j\sin\left(2\pi(f)\frac{j}{2F}\right)$, but with an extra factor of -1. The values aren’t the same, but the sound they make will be, and they’ll appear to be the same frequency.

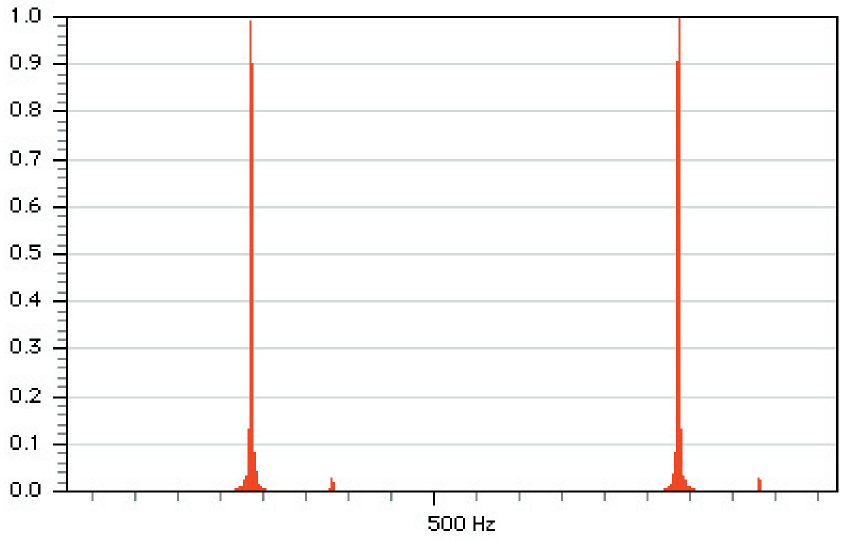
Which answers our question: a sine of frequency F+f aliases down to F-f.

For Dietrich, the aliasing left a gentleman without his fan. What’s the effect on some music? To do this example, we construct a sound in Matlab.

```
x=linspace(0,1, 1024);
s1=sin(2*pi*457*x)+sin(2*pi*557*x);
y=linspace(0,1, 2048);
»s2=sin(2*pi*457*y)+sin(2*pi*557*y);
```

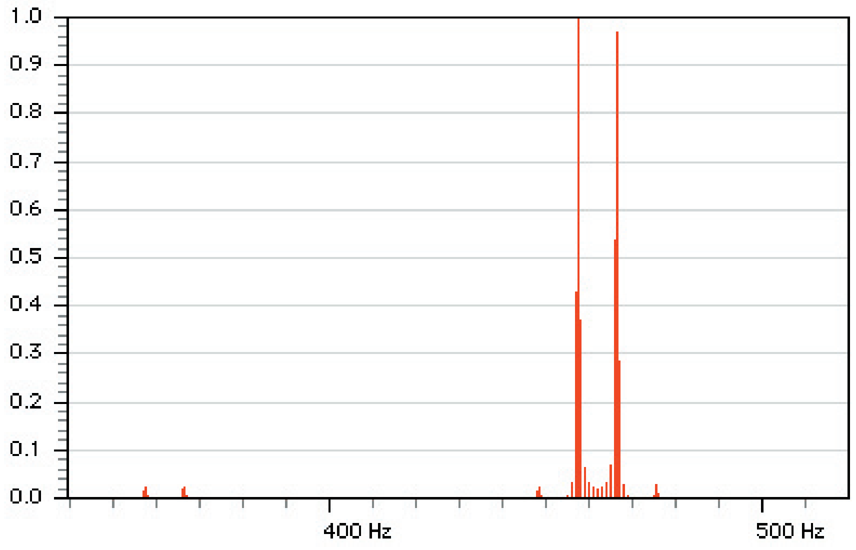
The sound s1 is a sum of sines, with frequencies 457hz, 557hz, sampled at 2F=1024. s2 is the same sound, except sampled at 2F=2048.

Now, when 2F=2048, F=1024 and the frequencies 457hz, 557hz are less than F, hence sampling doesn't alias them. Play 2048.wav, in the .ch2/sampling folder. And, while we're at it, here's the spectrum.



We see the two different frequencies, about equally spaced either side of 512hz.

Now try s1. Here, 2F=1024, so F=512, which means that 557hz = F+f where f=45hz. When you sample, this ought to be aliased down to F-f = 467hz. Here's the spectrum for s1:

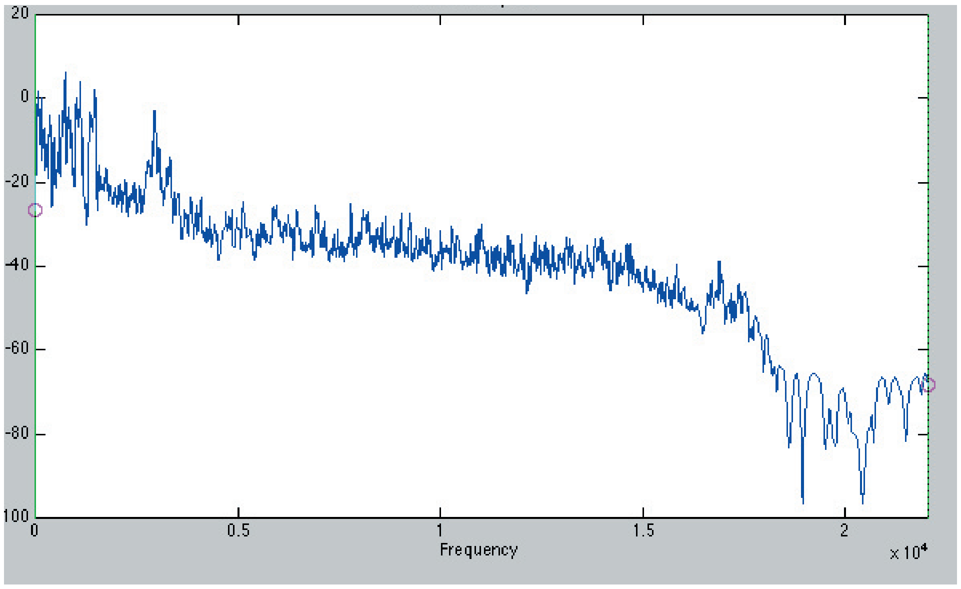


As predicted, but there's a little more to it than that: the the old frequency 447hz was well below the aliasing cut off, so it's unchanged. The 557hz sound used to be quite far away from 447, but now it's been aliased to 457hz, the two are within each other's *critical band*. Go ahead and play 1024.wav. There are now distinct beats. In a piece of music, this would be a disaster.

The moral here is that aliased frequencies don't disappear; they are changed into new frequencies that may never have been in the original sound. And when they get aliased, they may cause all sorts of problems, beats being only one.

Lab Problem: You can hear the effect with a series of clips constructed by starting with with the song, “Malaika”, by the Mahotella Queens, m32.wav or on the CD in /ch2/sampling. It is originally sampled at CD quality, 44100hz. we downsampled to a 32000hz mono signal, to make your work easier. Downsample to 16000hz, 4000hz, 2000hz and 1000hz. and listen to each. You'll especially enjoy the last.

Now look at what we hear. Find the spectrum of m32 and note there's little power past 2000hz, so we ought to be able to take a sampling rate of 4000hz, and keep the original sound. Play the sound downsampled to 4000hz and compare it with the orogina l. It sounds duller, as though it’s missing the high frequencies. And so it is; the spectrum picture was misleading. Here’s the original spectrum again, but this time plotted in decibels:



Ahem: oops! There are contributions to the spectrum all the way up to the maximum that 44100hz can give! And it seems the ear can hear some of this, too.

There's a moral in here. For example, say you were trying to compress sound. You look at the high frequency spectrum and say, “Look, the frequencies after about 4000hz fall off by 25db. We can cut those out, then go with a sampling rate of 8000hz. Let’s see, instead of 44100 samples per second, we’d have 8000. That’s like a compression down to 18% of the original! Wow.” Except it doesn’t sound so wow. Compression, alas, is not that simple.

Now try sampling at 4000hz and 2000hz. Note the spectrum of 4000hz has a nice peak at about 1100hz, and another at about 1500hz. Now compute the spectrum for the 2000hz downsampled music, where the highest frequency it can pick up is 1000hz. This means the frequencies at 1500hz and 1100hz that were in the previous have been aliased down to 900hz and 500hz. Check it out: sure enough, the spectrum shows strong peaks at around 900hz and 500hz. Compare with the original at 4000hz; see that there were no peaks at 900hz and 500hz, so these are now a result of aliasing.

Of course, the real test is the listening test: the music at 2000hz hasn’t just got an extra frequency or two; it sounds *weird* -- certainlly by comparison 8000hz, and even by comparison with 4000hz. The weirdness is due to something we haven’t quite encountered before:

the frequencies that get aliased down weren't just random noise; they were music. When they alias down they form form another kind of music. Strange music, because F+f becomes F-f so all the frequencies are reversed.

But music nonetheless. It's as though a second song were being played ontop of the first song.

The technical phrase is that the noise produced by aliasing is *correlated* with the original music. The ear picks up on that much more readily than it would just random sounds. There's a kind of hierarchy of badness here; some noise is worse than others.

The final moral here is that we can't afford aliasing, if we expect to do serious work with signals. Any signal has to be sampled at twice the highest frequency available, or the result will be corrupted. Since the ear can hear sounds up to about 20000hz, you need to sample at about 40000hz. The CD samples at 44100hz. Co-incidence?

The 44100hz standard for CD's came about because , after you sampling music, it had to be stored somewhere. The commercial system in the eighties that had the best storage capacity was professional videotape used, for example, to record TV shows. With a little futzing, one could convert the equipment to handle 44100 samples per second of sound. So the standard for CD players was born.

But there's another issue. Any recording session involves microphones, and those microphones pick up noise in the air, which often has frequencies much higher than 20000hz. If you don't eliminate the noise somehow, it will alias back down into all sorts of places you don't really want it. For this reason, professional recording begins with filtering: you pass the music though a filter, which eliminates all frequencies above 20000hz. At this point, the sound is sampled, and there's no aliasing.

In practice, it gets very complicated. First of all, the filter has to be applied before the sound is sampled. That means the signal isn't digital yet, so the filter has to be analog. These are hard to implement, and in fact they can't simply pass all frequencies below 20000hz, unchanged, and cut off all frequencies above 20000hz, completely. There needs to be a little leeway. That's why 44100 is better than 40000; it gives leeway between 22050 and 20000.

The next problem is with the sampling itself. We described it very mathematical -- f(jT) and it looks very clean. But in fact it has to be done, again, by analog circuits. Resistors and capacitors and diodes and transistors. And none of them are precise and mathematical. A capacitor takes time to discharge, and time to recharge.

One could try to optimize sampling by sampling *adaptively*. If a musical piece had only human voice for a while, then the instruments came in, the part with voice doesn't have high frequencies, so it could be sampled at a low rate, after which the high sampling rate for the instruments is used. You could compress music this way; probably it wouldn't even be very hard to code. See the article "adaptive.pdf" in the c2/sampling folder of the CD for a modern application.

And, while we're at it . . . we've talked about the need to be careful when you sample music. But there's another field where no-one cares how people sound: the telephone! If you accept the idea that the telephone will be used to transmit voice, what's the proper frequency to sample?