Lecture Notes for Fast Fourier Transform

CS227-Scientific Computing

November 16, 2011

Part I: What it Does

Sound generation and playback in MATLAB

Try the following commands:

```
>> t=linspace(0,1,40000);
>> y1=cos(2*pi*440*t);
>> sound(y1,40000)
```

Sound generation and playback in MATLAB

The command
`sound(y,n)`
treats the values in the vector `y` as measures of the intensity, or *amplitude* of sound sampled `n` times a second. The sample values have to lie strictly between -1 and 1. The 40 kHz sample rate used here is close to what is used for CDs (44100 samples per second).

Our synthetic sound is a sine wave that repeats 440 times every second. The *frequency* is thus 440 Hz, and this determines the pitch. (440 Hz is an 'A' above middle C).

Superimposing sinusoids

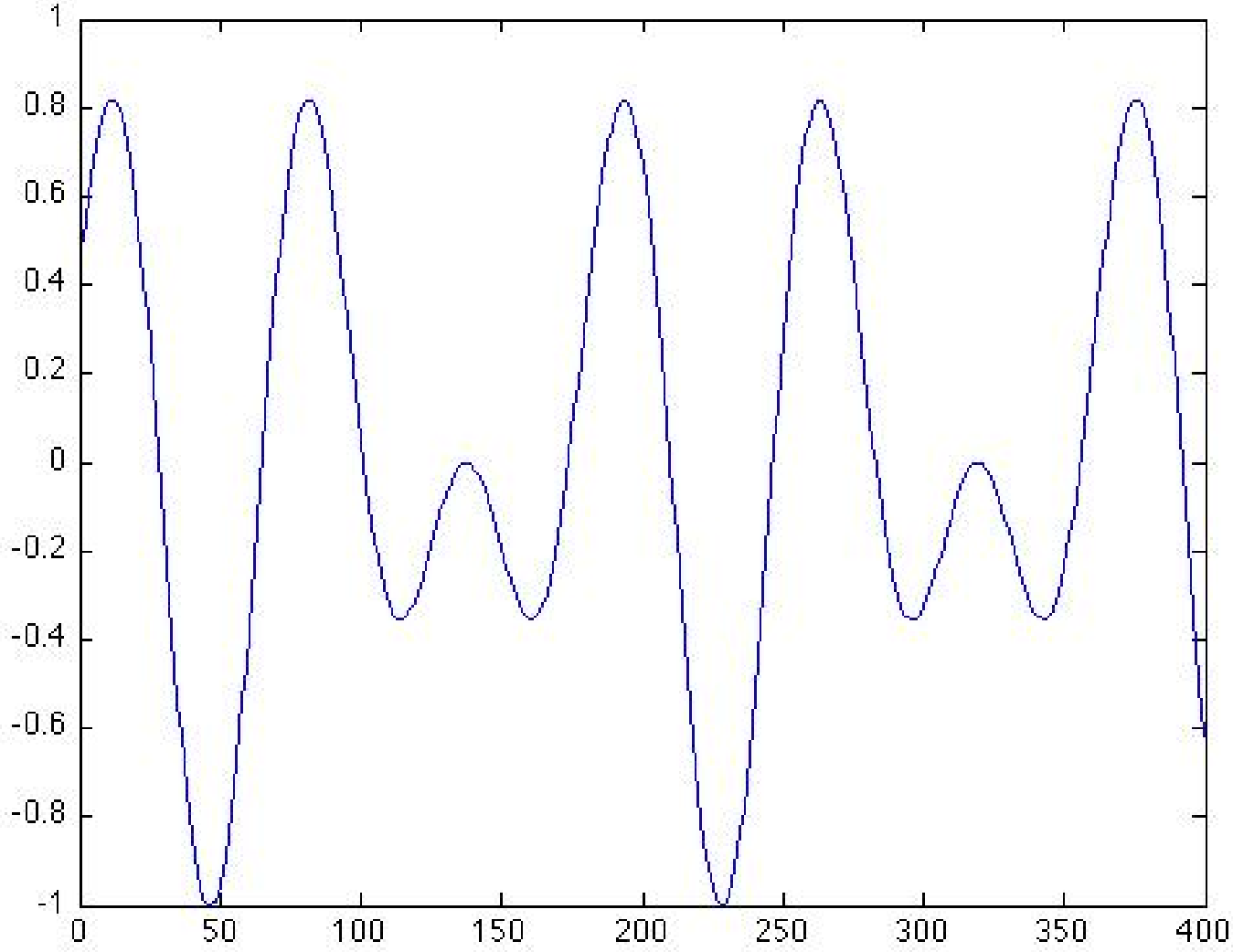
Now let's add another, higher-pitched, tone on top of this one, and listen to the result:

```
>> y2=sin(2*pi*660*t);
>> y=(y1+y2)/2;
>> sound(y,40000)
```

Note that we average the two sets of amplitudes so that the sample values don't go outside the range $(-1, 1)$.

Superimposing sinusoids

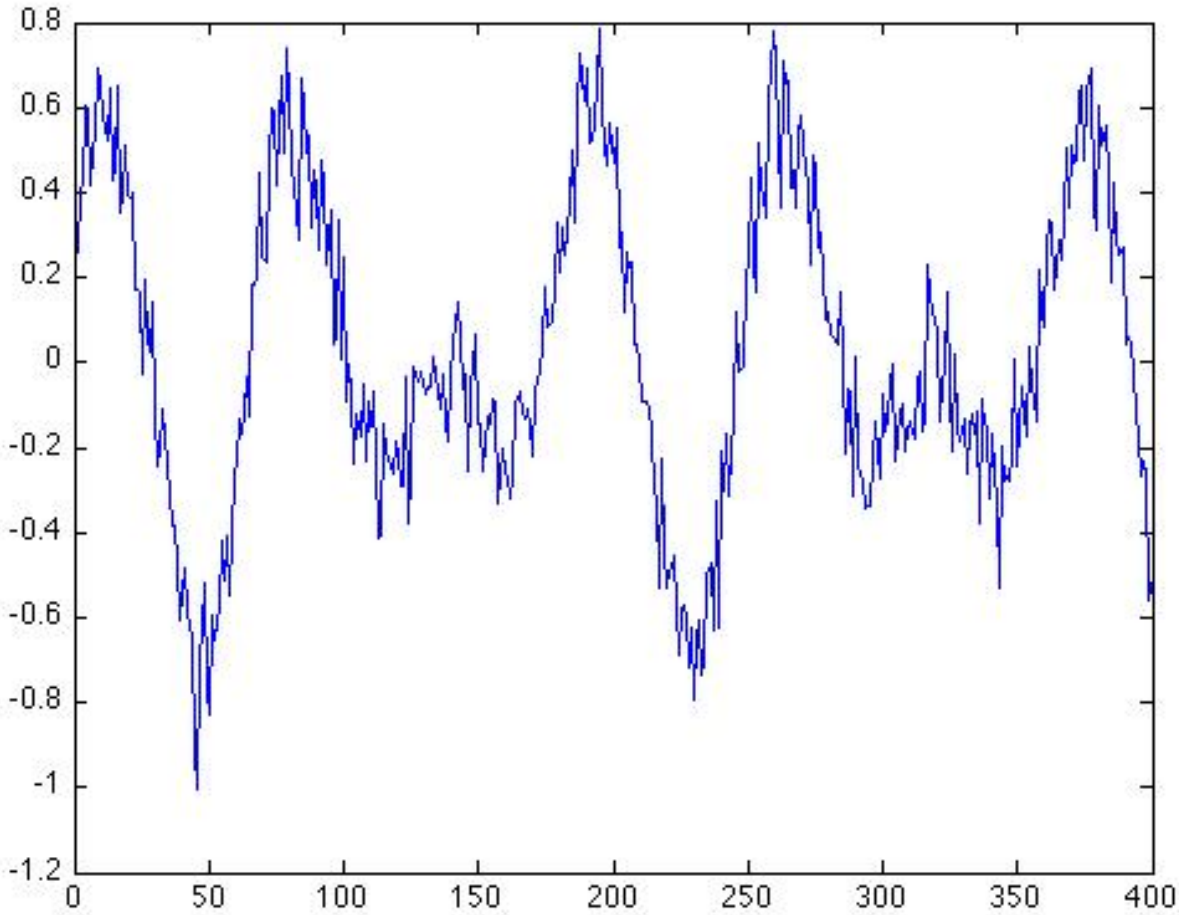
Here's what the first few hundred sample values look like:



Superimposing sinusoids

Just for fun, let's add a little noise:

```
>> z=0.7*y+0.1*randn(1,40000);
```



Extracting frequencies from the signal

The plots above display the signal as amplitude vs. time. But there is also a way to display the data as amplitude vs. frequency. For this we use the MATLAB function `fft`. This stands for *Fast Fourier Transform.*

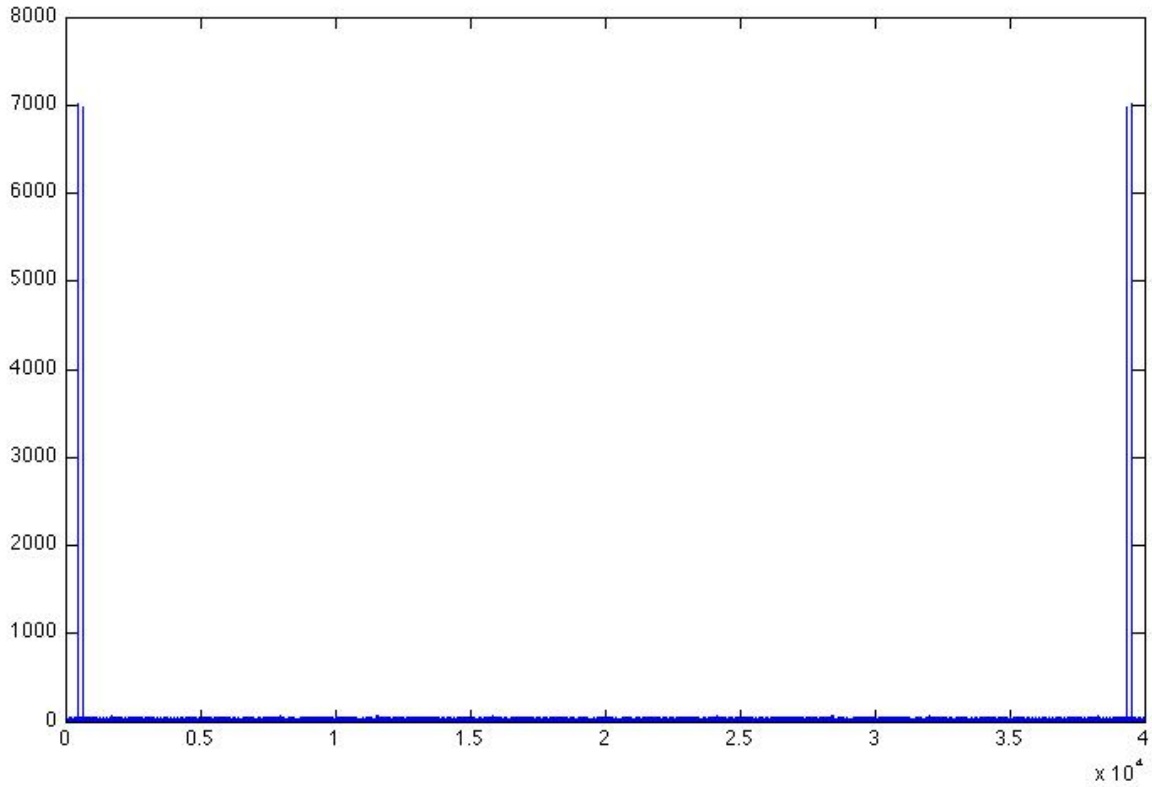
```
>> trans = fft(z);
>> trans(1:3)

ans =

 -19.0792   7.4440 + 1.3015i  14.2555 +13.1185i
```

`fft` returns a vector of *complex numbers* of the same size as its argument.

Extracting frequencies from the signal

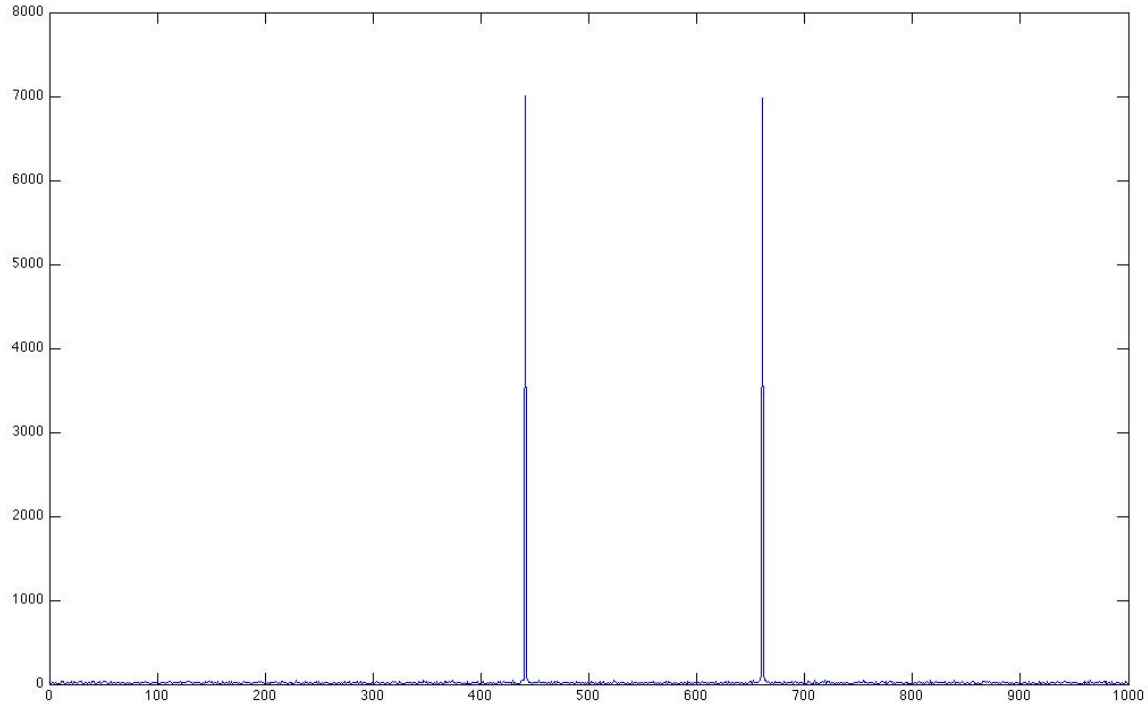
Let's plot the magnitudes of these complex numbers:

```
trans=fft(z);
plot(1:40000,abs(trans));
```



It's not very revealing, but let's zoom in for a closeup:

Extracting frequencies from the signal



The big spikes are at 440 Hz and 660 Hz, right where we put them! The added noise makes small contributions at a lot of different frequencies, so it appears like grass growing at the bottom of the picture.

Extracting frequencies from the signal

You'll note that in the original figure, there are two identical spikes at the far right end of the plot. The Fourier transform is symmetric about a frequency equal to half the sampling rate. You can view the information in the right-hand half as artifacts of the mathematical method used to produce the transform. (We'll see shortly why this is.) The real frequency information is in the left-hand half.

If you weighted the two sinusoidal components differently, this would be reflected in the differing height of the spikes.

Part II: How Does this Work (and what's with the complex numbers?)

Some basics about the complex numbers

We represent the point (a, b) in the plane as $a + bi$, a *complex number.*

If we declare $i \cdot i = -1$, then we can add and multiply any two points in the plane. Addition is just ordinary vector addition, but to multiply two such complex numbers, we *multiply* their *magnitudes* (*i.e.,* their lengths) and *add* the angles that they make with the positive x-axis.

Why is this? Because a point a distance of r from the origin making an angle of θ with the positive x-axis has coordinates $(r\cos\theta, r\sin\theta)$.

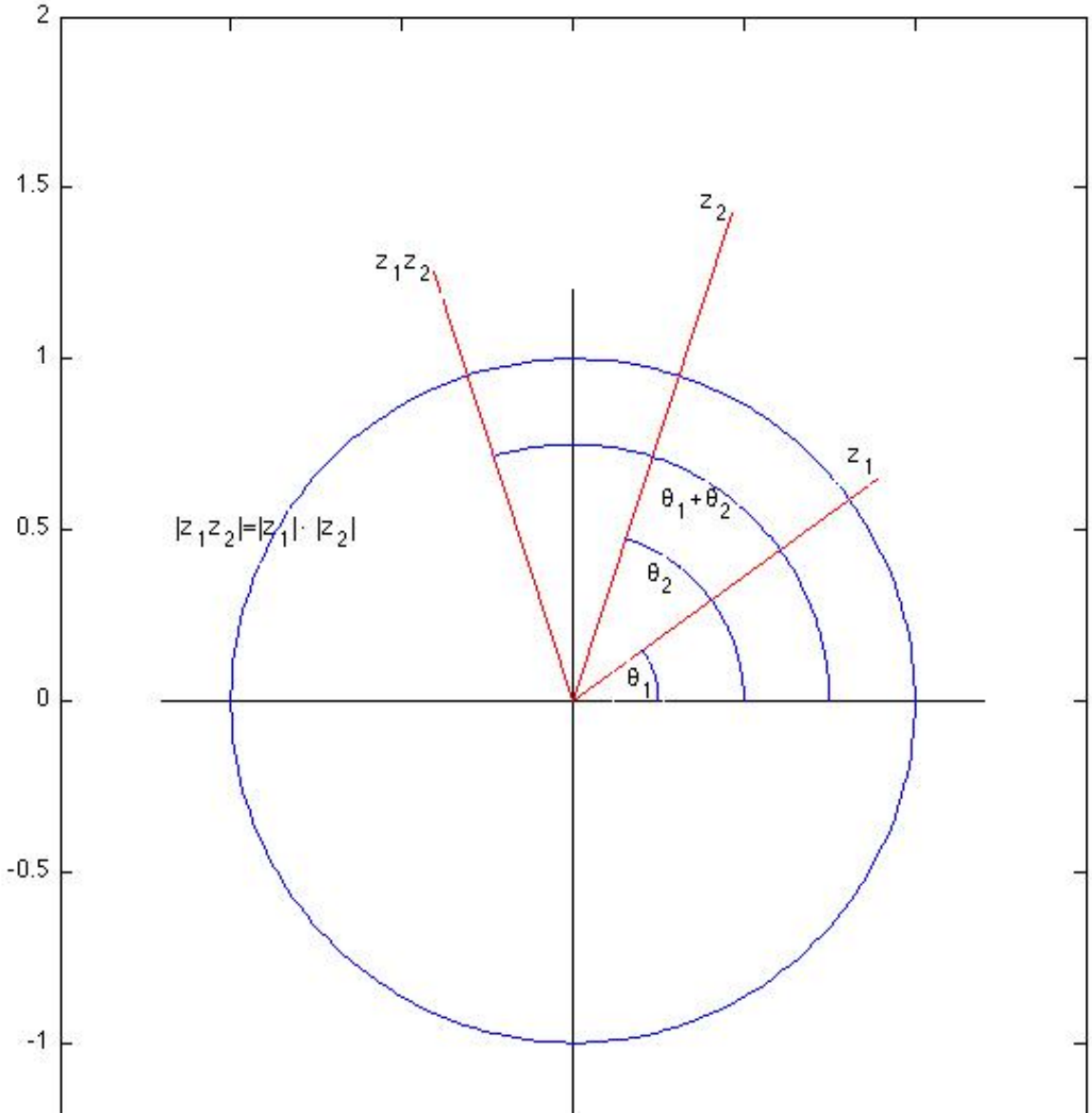
So to get the multiplication to work out right with $i^2 = -1$ we would have to have:

$$\begin{aligned}(r_1 \cos\theta_1 + i \cdot r_1 \sin\theta_1) \cdot (r_2 \cos\theta_2 + i \cdot r_2 \sin\theta_2) &= \\ r_1 r_2((\cos\theta_1 \cos\theta_2 - \sin\theta_1 \sin\theta_2) + i(\cos\theta_1 \sin\theta_2 + \sin\theta_1 \cos\theta_2)) &= \\ r_1 r_2(\cos(\theta_1 + \theta_2) + i(\sin(\theta_1 + \theta_2)).\end{aligned}$$

Amazingly, it all works out. Multiplication is associative and distributive, every nonzero element has a multiplicative inverse, *etc.*

The multiplicative inverse of $z = x + iy$ is $\frac{x-iy}{x^2+y^2} = \frac{\bar{z}}{|z|^2}$.
$\bar{z} = x - iy$ is called the *complex conjugate* of z.

Complex Multiplication



Complex Exponentiation

But there is more amazement to come. Imagine a particle moving counterclockwise around the unit circle with a speed of 1 unit per second. It starts at the point $(1, 0) = 1 + 0 \cdot i = 1$, so at time t its position is $z(t) = \cos t + i \cdot \sin t$. This is our basic picture for the simplest kind of periodic motion–for sinusoids. This has a period of 2π seconds and a fequency of $\frac{1}{2\pi}$ hertz.

The particle's velocity vector at time t points in a direction perpendicular to $z(t)$ and has a magnitude of 1. It's just $z(t)$ rotated 90 degrees counterclockwise; that is, the velocity is $i \cdot z(t)$.

Complex Exponentiation

So, to stretch a point, z is a solution to the differential equation (with initial condition):

$$z' = i \cdot z, z(0) = 1.$$

Now if things work out the way they do for real-valued differential equations, this means

$$\cos t + i \cdot \sin t = z(t) = e^{it}.$$

Substituting $t = \pi$ yields the astonishing equation

$$e^{i\pi} = -1.$$

The Fourier Transform matrix

Let us choose an integer $n > 0$, and let $\omega = e^{\frac{2\pi i}{n}}$. Note that $\omega^{-1} = \bar{\omega}$.

The numbers $1, \omega, \omega^2, \ldots, \omega^{n-1}$ are the n distinct n^{th} roots of 1. Geometrically, they are the vertices of a regular n-sided polygon inscribed inside the unit circle, with one vertex at the point $(1, 0)$.

Physically, you can think of the numbers $1, \omega, \omega^2, \ldots$ as the positions of a point at times $0, \frac{1}{n}, \frac{2}{n}$, *etc.* This represents sinusoidal motion with a period of 1 and a frequency of 1 cycle per time unit.

Similarly, the sequence $1, \omega^2, \omega^4, \ldots$ represents sinusoidal motion with twice the frequency (2 cycles per time unit); $1, \omega^3, \omega^6, \ldots$ motion with frequency of 3 cycles per time unit, *etc.*

The Fourier Transform matrix

The $n \times n$ Fourier Tranform matrix has these sequences as rows (and also as columns—it's symmetric).

Here it is in the case $n = 5$, so $\omega = e^{\frac{2\pi i}{5}}$:

$$F_5 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega^8 \\ 1 & \omega^3 & \omega^6 & \omega^9 & \omega^{12} \\ 1 & \omega^4 & \omega^8 & \omega^{12} & \omega^{16} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega & \omega^3 \\ 1 & \omega^3 & \omega & \omega^4 & \omega^2 \\ 1 & \omega^4 & \omega^3 & \omega^2 & \omega \end{bmatrix}.$$

The *adjoint* A^H of a complex matrix A is its conjugate transpose. In the case of the Fourier Transform matrix, this is practically the same matrix as F_n: It just has its rows shuffled.

$$F_5^H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \overline{\omega} & \overline{\omega^2} & \overline{\omega^3} & \overline{\omega^4} \\ 1 & \overline{\omega^2} & \overline{\omega^4} & \overline{\omega} & \overline{\omega^3} \\ 1 & \overline{\omega^3} & \overline{\omega} & \overline{\omega^4} & \overline{\omega^2} \\ 1 & \overline{\omega} & \overline{\omega^3} & \overline{\omega^2} & \overline{\omega} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^4 & \omega^3 & \omega^2 & \omega \\ 1 & \omega^3 & \omega & \omega^4 & \omega^2 \\ 1 & \omega^2 & \omega^4 & \omega & \omega^3 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 \end{bmatrix}.$$

The Fourier Transform matrix

The magic is that F_n^H is (almost) the same as F_n^{-1}. (There's an additional constant factor.)

To see why, let's take the dot product of column j of F_n with column k of F_n^H, We get

$$1 + \omega^{(j-k)} + \omega^{2(j-k)} + \cdots + \omega^{(n-1)(j-k)}.$$

We can use a little trick to show **this sum is equal to 0.** It's just the identity

$$x^n - 1 = (x-1)(1 + x + \cdots + x^{n-1}).$$

If we substitute $x = \omega^{(j-k)}$, the left-hand side is 1-1= 0. However, if $j \neq k$, then $x \neq 1$, so the second factor on the right-hand side is 0.

On the other hand, if $j = k$ then every term in the dot product is 1, so the sum is equal to n. Thus we have

$$F_n \cdot F_n^H = F_n^H \cdot F = nI.$$

The Fourier Transform

So let's say we have a column vector $\mathbf{y}$ with n components.

Then

$$\mathbf{y} = \frac{1}{n} F_n^H \mathbf{z}$$

where $\mathbf{z} = F_n \mathbf{y}$. The vector $\mathbf{z}$ is called the *Discrete Fourier Transform* of $\mathbf{y}$.

If $\mathbf{z} = [c_0 \cdots c_{n-1}]^T$, then

$$\mathbf{y} = \frac{1}{n}(c_0 \mathbf{u}_0 + \cdots + c_{n-1}\mathbf{u}_{n-1}),$$

where $\mathbf{u}_j$ is column j of F^H. In other words, we an write *any* vector $\mathbf{y}$ as a weighted sum of sinusoids of different frequencies. If we think of $\mathbf{y}$ as giving the amplitude of a signal at different moments of **time**, then the weights (or rather, their absolute values, since they are complex numbers) give the strength of the signal at different **frequencies**.

The Fourier Transform

Let's give a few examples of this. Suppose first $n = 100$, so that $\omega = e^{\frac{2\pi i}{100}}$.

Consider the vector

$$\mathbf{y} = (1, \cos\frac{2\pi}{10}, \cos\frac{2\cdot 2\pi}{10}, \ldots, \cos\frac{99\cdot 2\pi i}{10}).$$

We can write

$$\mathbf{y} = \frac{1}{2}((1, \omega^{10}, \omega^{20}, \ldots, \omega^{99\cdot 10}) + (1, \omega^{-10}, \omega^{-20}, \ldots, \omega^{-99\cdot 10}))$$

The Fourier Transform

From what we observed earlier,

$$F_{100}\mathbf{y}$$

has the value n in the 11^{th} and the 91^{st} components. (This accounts for the symmetry we observed in the transform.)

If we apply the transform instead to the vector

$$\mathbf{y}' = (1, \cos\frac{2\pi}{11}, \cos\frac{2\cdot 2\pi}{11}, \ldots, \cos\frac{99\cdot 2\pi i}{11}),$$

then $F_{100}\mathbf{y}'$ no longer exhibits the massive cancellation we saw in computing $F_{100}\mathbf{y}$. But it is close, which is what we observed in our original example.

Part III: The Discrete Fourier Transform in MATLAB

The Fourier Transform

The Discrete Fourier Transform is a terrific tool for signal processing (along with many, many other applications). However the catch is that to compute $F_n\mathbf{y}$ in the obvious way, we have to perform n^2 complex multiplications. If we are transforming a vector with 40,000 components (1 second of sampled audio on a CD), we need 1.6 billion such multiplications.

In 1965. James Cooley and John Tukey published an algorithm for computing this that used only $n \cdot \log_2 n$ multiplications. With $n = 40,000$, this is about 600 thousand multiplications, which is almost *three thousand times* faster than the obvious method.

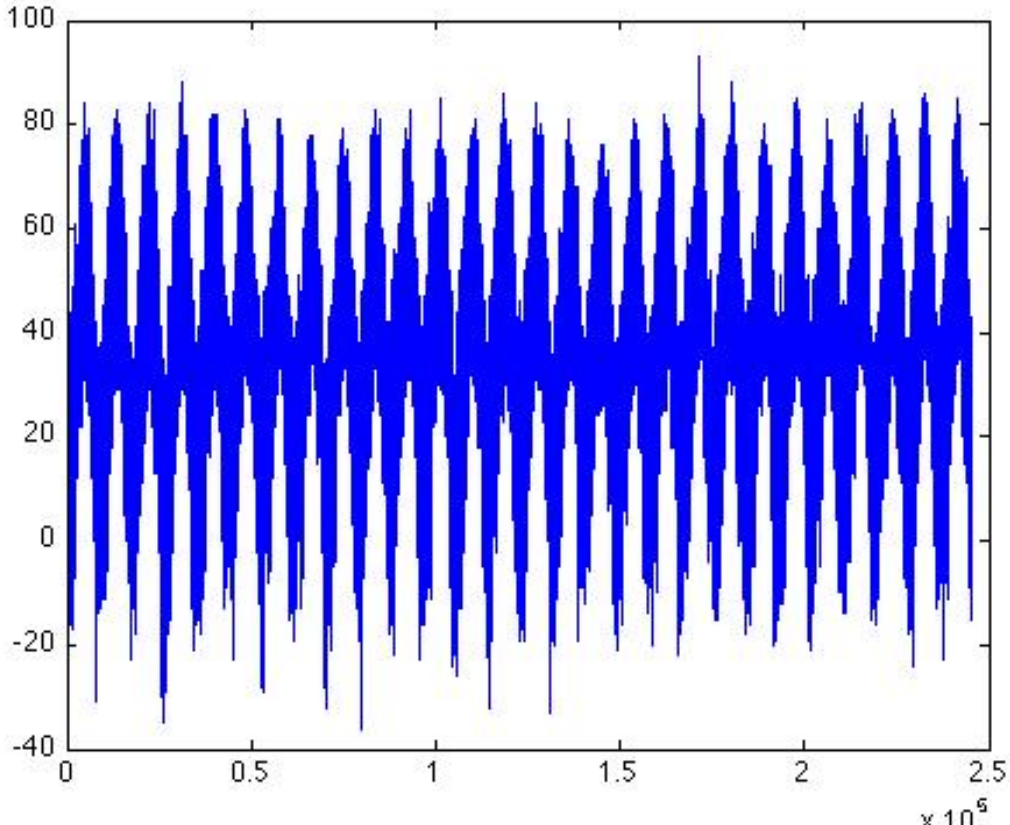
The so-called Fast Fourier Transform is not a different transform from the DFT, it's just a different way of computing it.

MATLAB `fft` and `ifft`

In MATLAB you just type
`z = fft(y)`
to get a complex vector `z` that is the DFT of `y`. The inverse transform, which, as we have seen, is almost the same thing, is gotten by
`y = ifft(z)`.

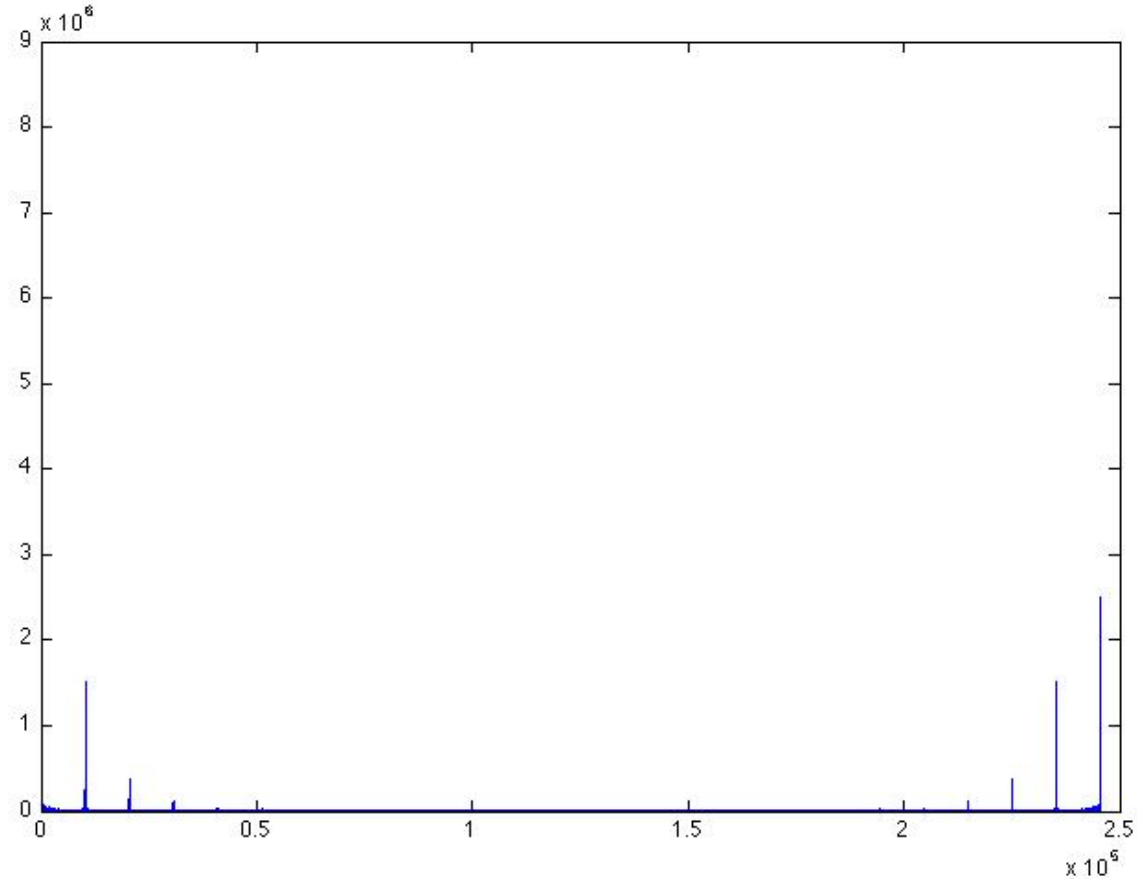
Example: Extracting periodic information from climate data

The Fourier transform was used to analyze 28 years of hourly temperature data, in degrees Fahrenheit, from a weather station in Colorado. You can see the seasonal variation, with low temperatures in the winter (the data start January 1, 1976) and high temperatures in the summer.



The climate data in the frequency domain.

As with our original examples, we apply `fft` to the vector of data values and plot the absolute values of the resulting transform. Here are several different views, at differing resolutions.



The climate data in the frequency domain.

The full transform exhibits the kind of symmetry we always see when we transform real-valued data.

The general rule is this: if we number the components of the transform $\mathbf{z}_0, \ldots, \mathbf{z}_{n-1}$, then for $j > 0$,

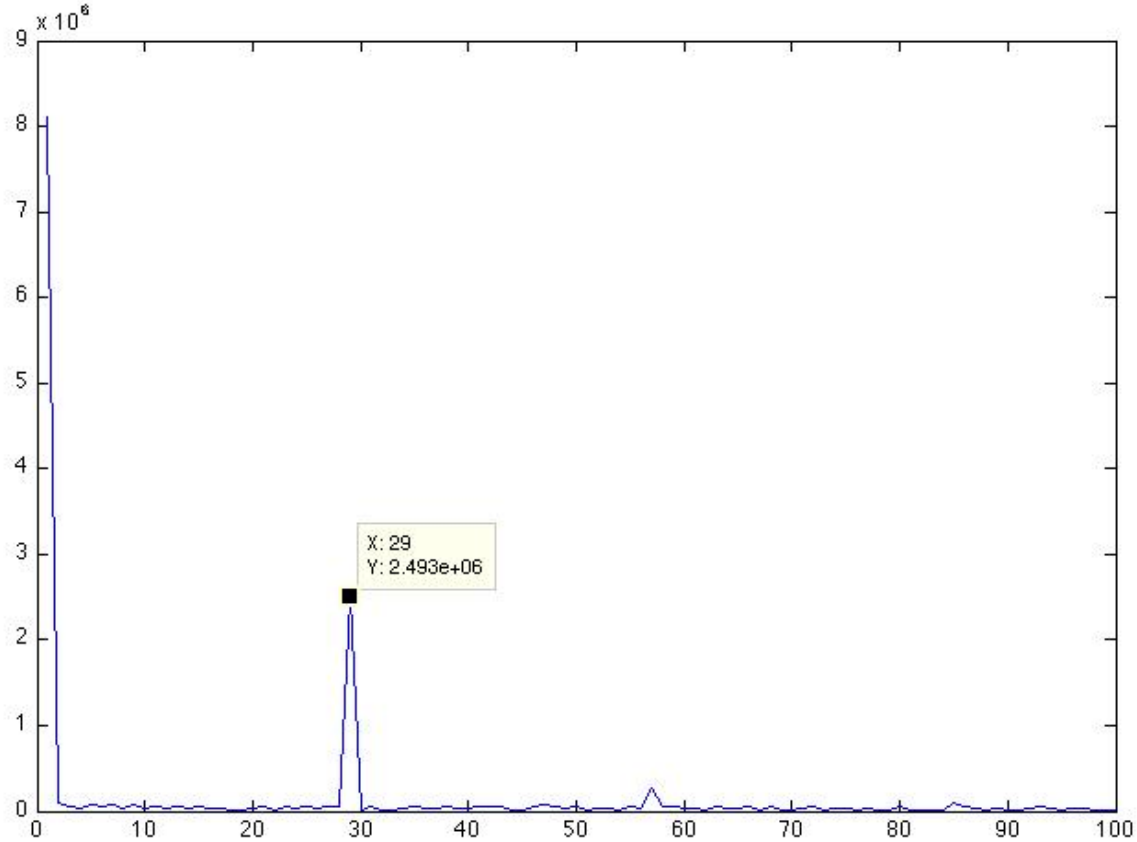
$$\mathbf{z}_j = \bar{\mathbf{z}}_{n-j}.$$

This means

$$|\mathbf{z}_j| = |\mathbf{z}_{n-j}|,$$

which accounts for the symmetry when we plot it.

The climate data in the frequency domain.



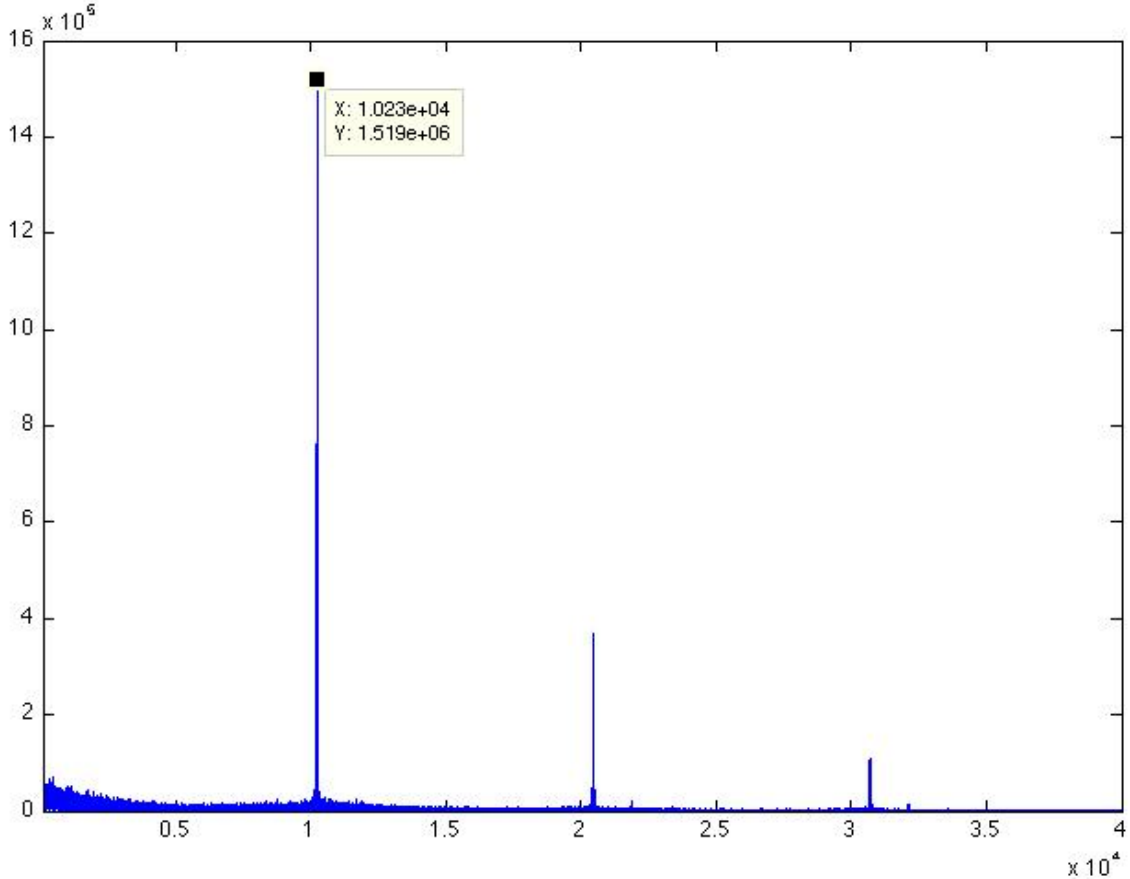
The climate data in the frequency domain.

The closeup of the left-hand end of the plot shows the large DC component at $x = 1$. This gives no information about temperature data.

The next large spike occurs at $x = 29$. Since we have to adjust by 1 (because MATLAB indexes vectors starting at 0), this corresponds to a frequency of 28. But 28 what? It is 28 cycles per the complete sampling interval—in this case, 28 cycles per 28 years, or 1 cycle per year. This is the annual temperature fluctuation.

What about the smaller spikes to the right? The next two of these occur at $x = 57$ and $x = 85$, corresponding to frequencies of 56 and 84 cycles in the interval, or 2 cycles per year and 3 cycles per year. Such 'overtones' at integer multiples of the fundamental frequency are common in this transformed data.

The climate data in the frequency domain.



The climate data in the frequency domain.

When we zoom out we see spikes at $x \approx 10230$ and small multiples thereof. What is this?

The entire sampling interval contains 245448 samples. $245448 = 10227 \times 24$, so this represents exactly 10227 days.

Thus these spikes correspond to a frequency of one cycle per day.

The Fourier transform shows very clearly the daily temperature fluctuation superimposed on the yearly cycle.