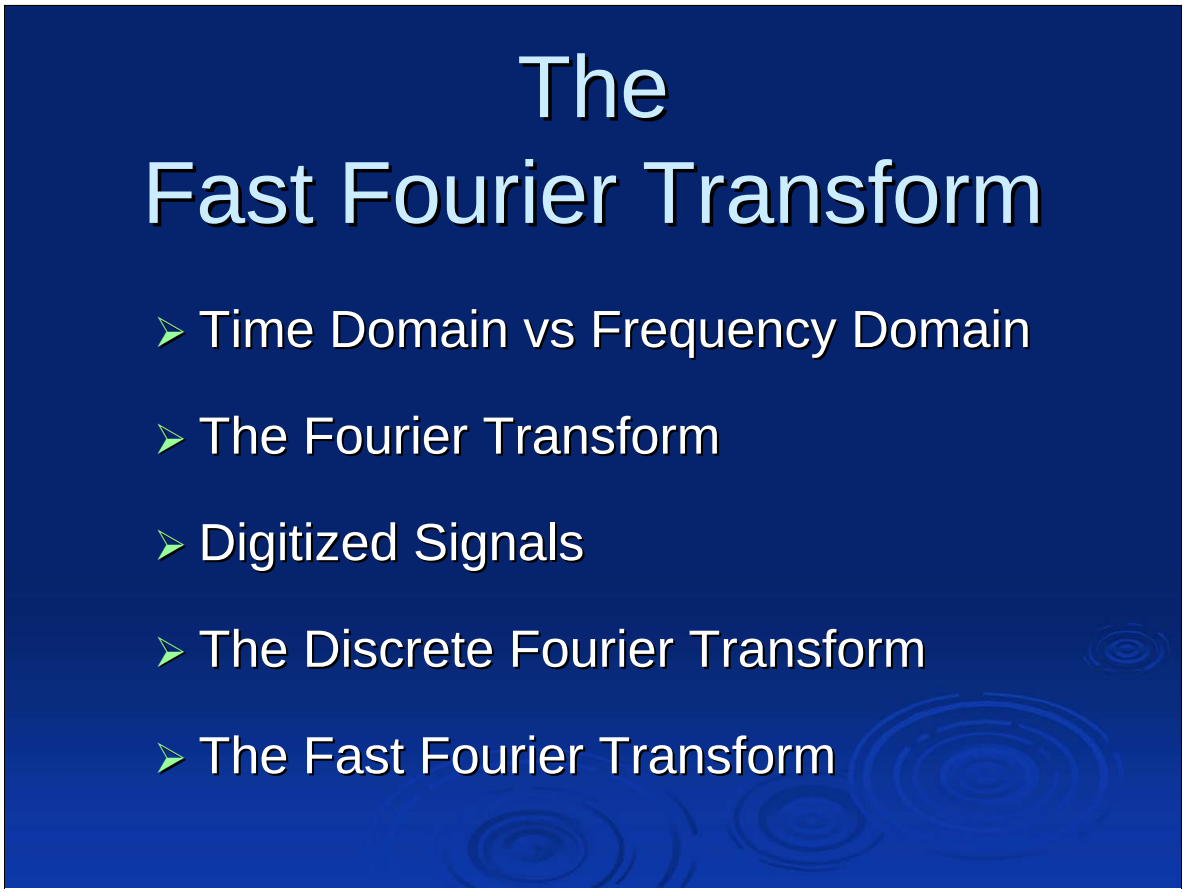
The Fast Fourier Transform

Basic FFT Stuff That's Good to Know

Dave Typinski, Radio Jove Meeting, July 2, 2014, NRAO Green Bank

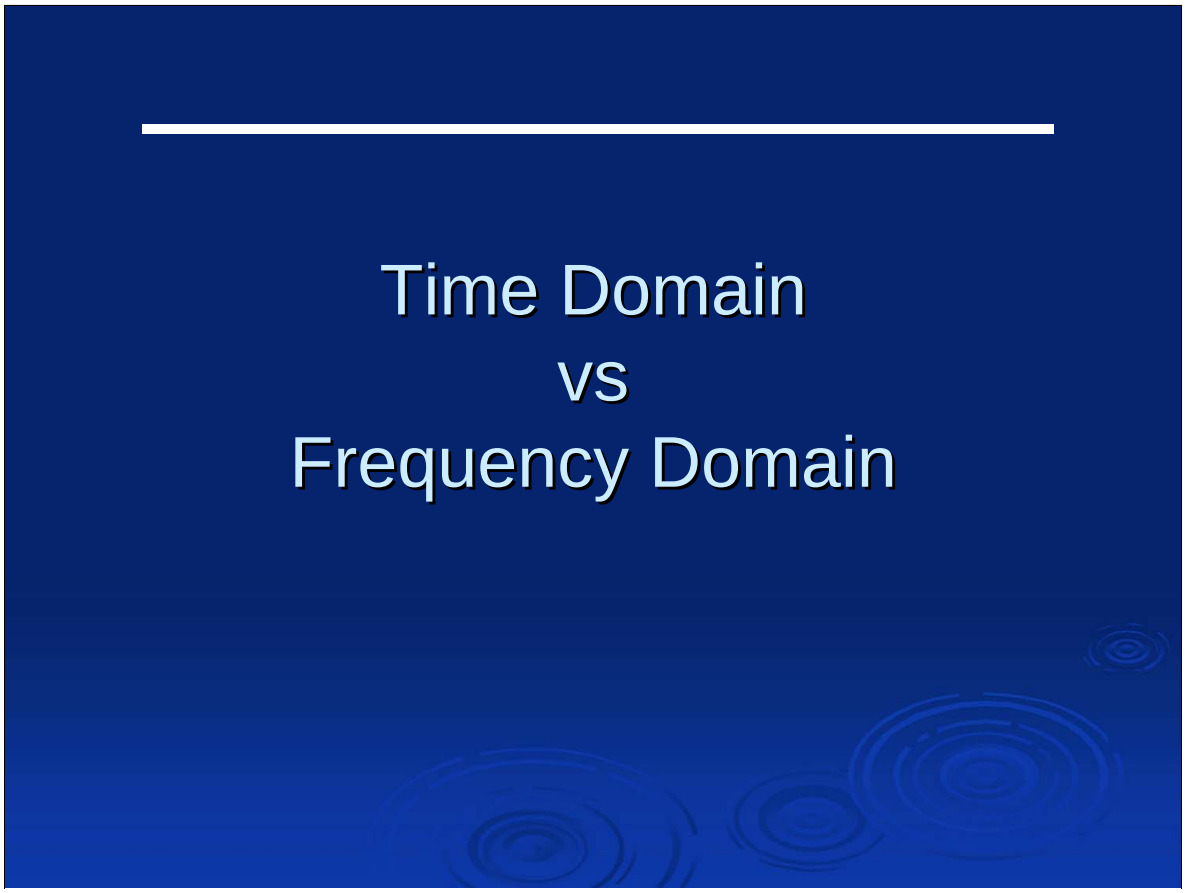
Ever wonder how an SDR-14 or Dongle produces the spectra that it does? Well, now you're going to get a very basic idea of how that's done. Everyone here has probably heard of the term "FFT" – but some may not know what it means.

In this talk, we'll cover most of the basic ideas that are good to know about FFT's. Mostly it's about the main factors that affect what one feeds into an FFT considering what one wants to get out of an FFT.

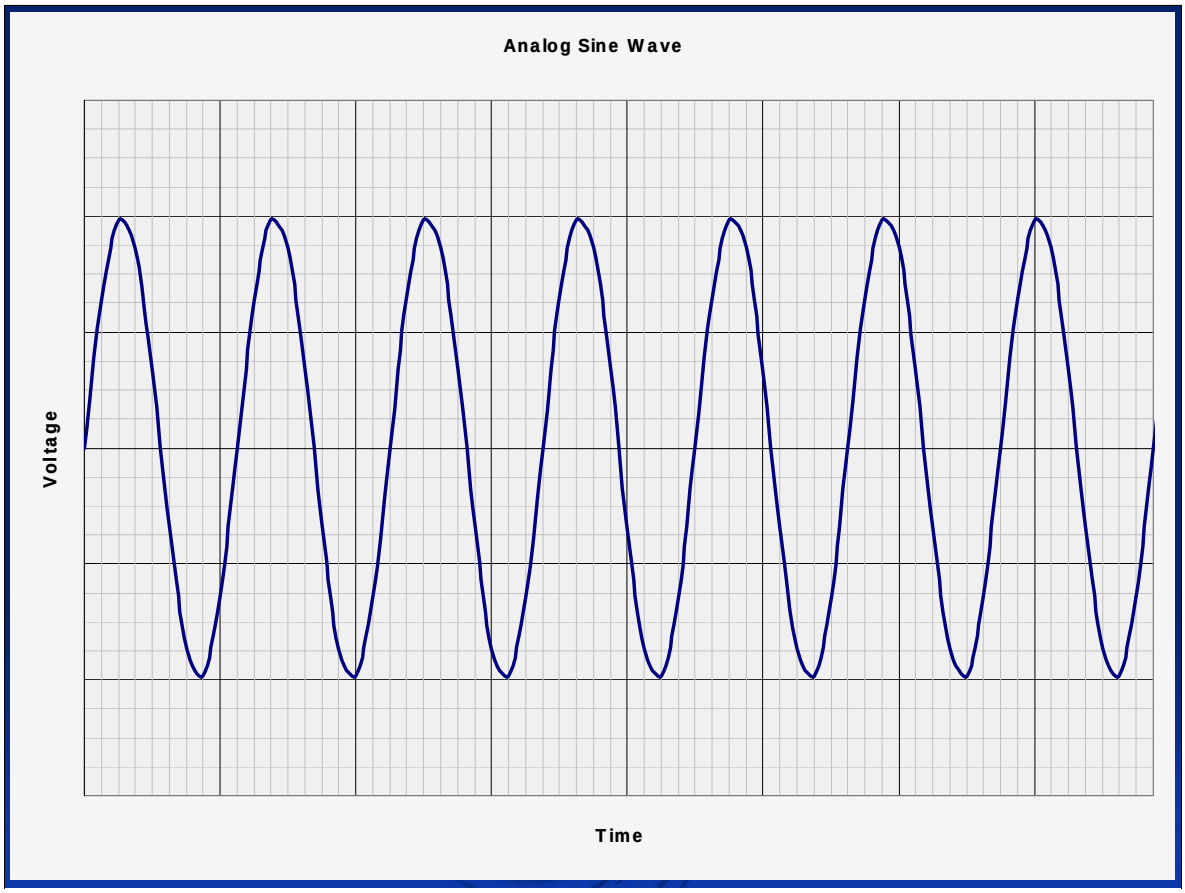


First, we'll review some basics – the difference between analog and digital signals, along with the analog and digital versions of the Fourier transform.

Then we'll discuss the fun and interesting FFT stuff.

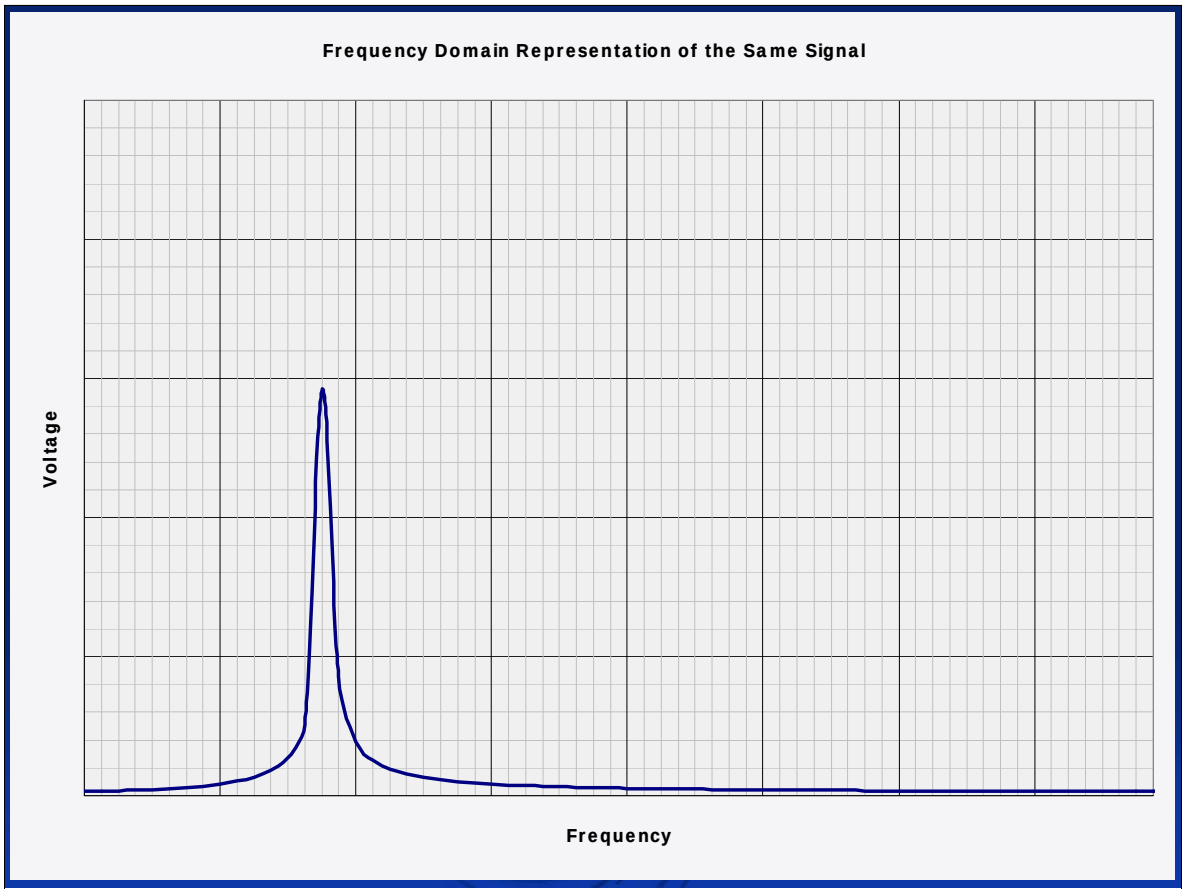


The first thing we really should understand is the difference between the frequency and time domain.



Here is a plot of a signal in the TIME DOMAIN

Useful for radio astronomy strip charts, EEG machines, thermostat controllers, and audio tape players



Here is the same signal in the FREQUENCY DOMAIN

Useful for spectrum analysis– e.g., a **radio spectrograph**

Ideally, this spike would be infinitely thin for a perfect and infinitely long single-frequency sine wave. The finite thickness shown here is a consequence of the imperfect, finite real world.

The question is: how do we get from there to here, from the time domain to the frequency domain?

which leads us to....

The Fourier Transform

The Fourier Transform

- A mathematical method of finding the frequency domain representation of a time domain function.
- The Fourier Transform Black Box

If you know the mathematical function of a signal – say, sin(x), then you can use the Fourier transform find the freq domain function.

How does it work? Open the box!

The Fourier Transform

- A mathematical method of finding the frequency domain representation of a time domain function.
- $$X(f) = \int_{-\infty}^{\infty} x(t)\, e^{-j2\pi f t}\, dt$$

where

$X(f)$ = frequency domain representation of signal

$x(t)$ = time domain representation of signal

f = frequency

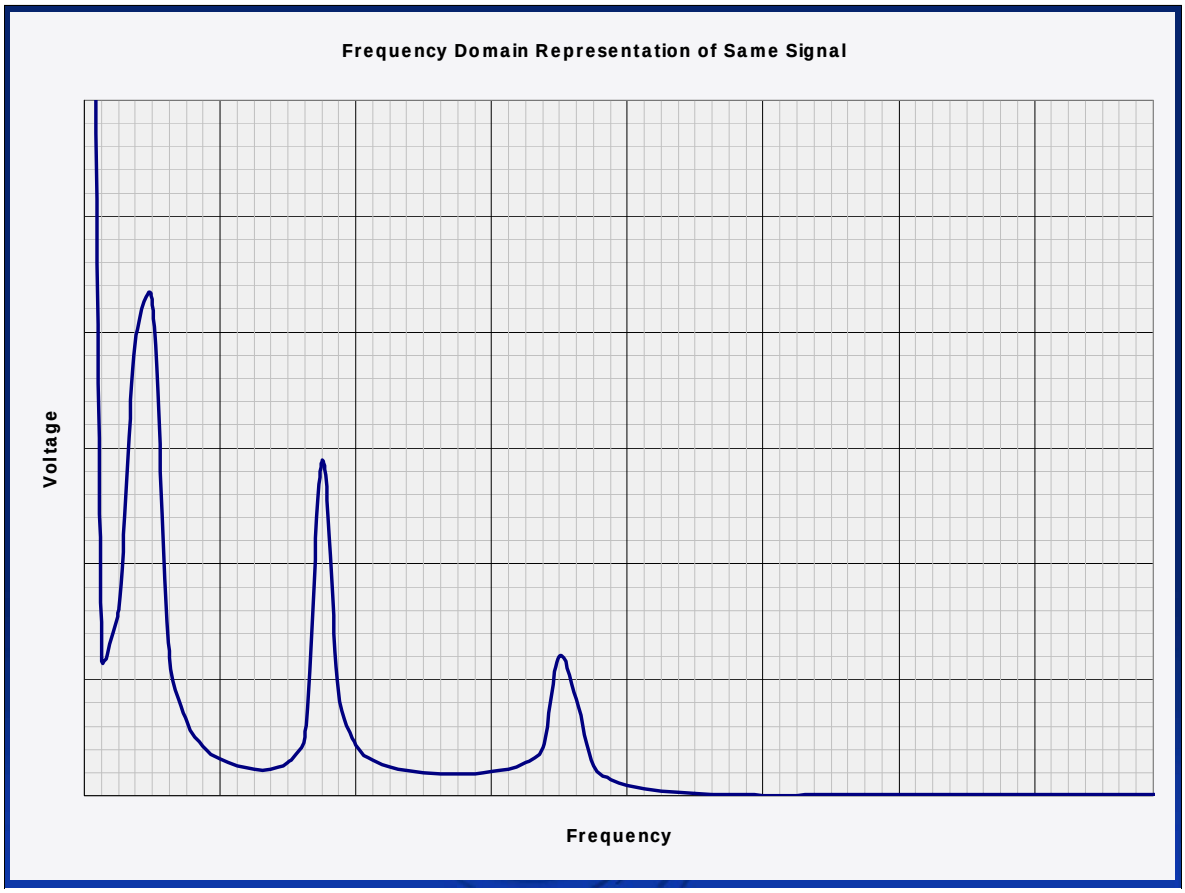
t = time

Quick, shut it!

The mathematical details are too mind numbing to go into in a half-hour talk. Nevertheless, here they are for the sake of thoroughness.

It is sufficient to say that this equation can transform a garden variety function like sin(x) into its frequency domain representation.

The Fourier transform works on any real world signal. This one is made up of three separate sine waves with different amplitudes, frequencies, and phases. For now, don't worry about how this signal is generated – we'll get to that later. We can safely treat it as a mathematical abstraction whose details aren't really important at this point.

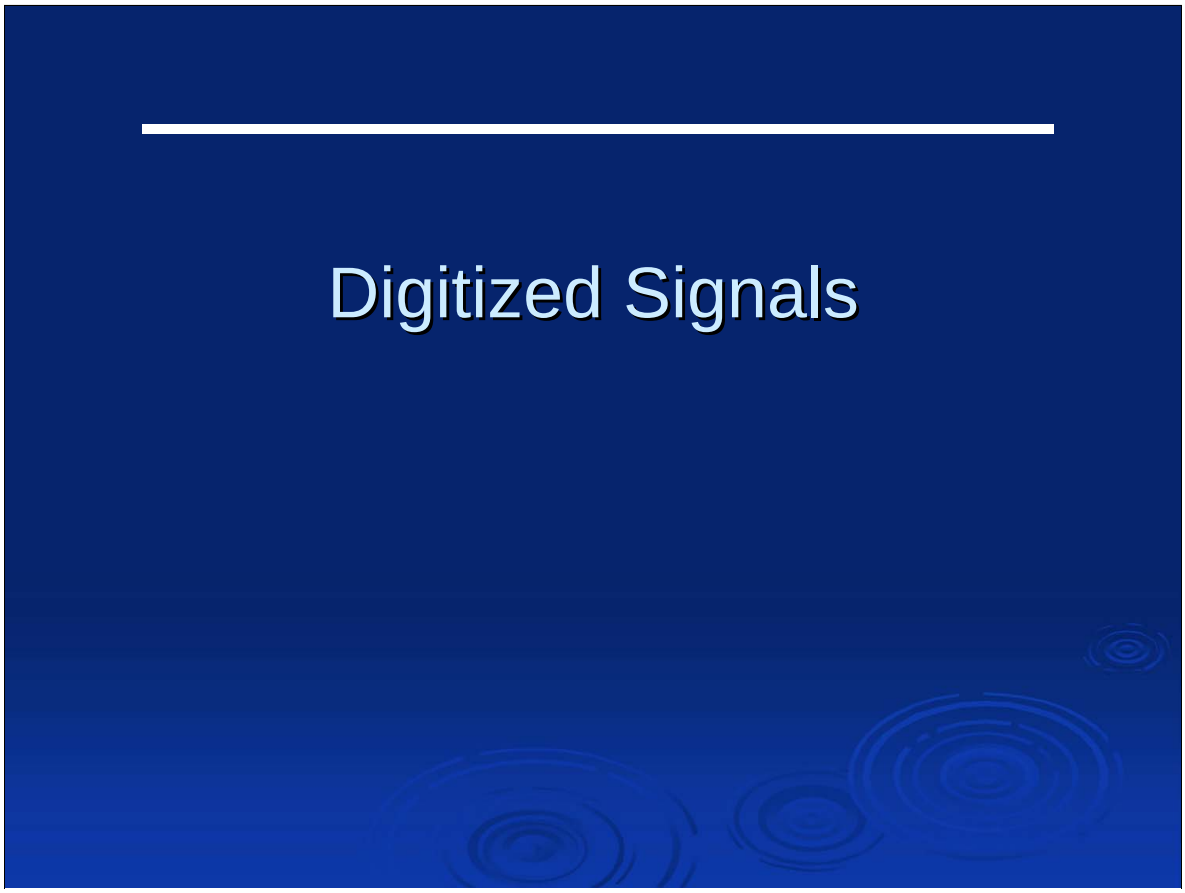


When we apply the Fourier transform to the equation for the three-sine-wave signal, we obtain a function that produces this plot.

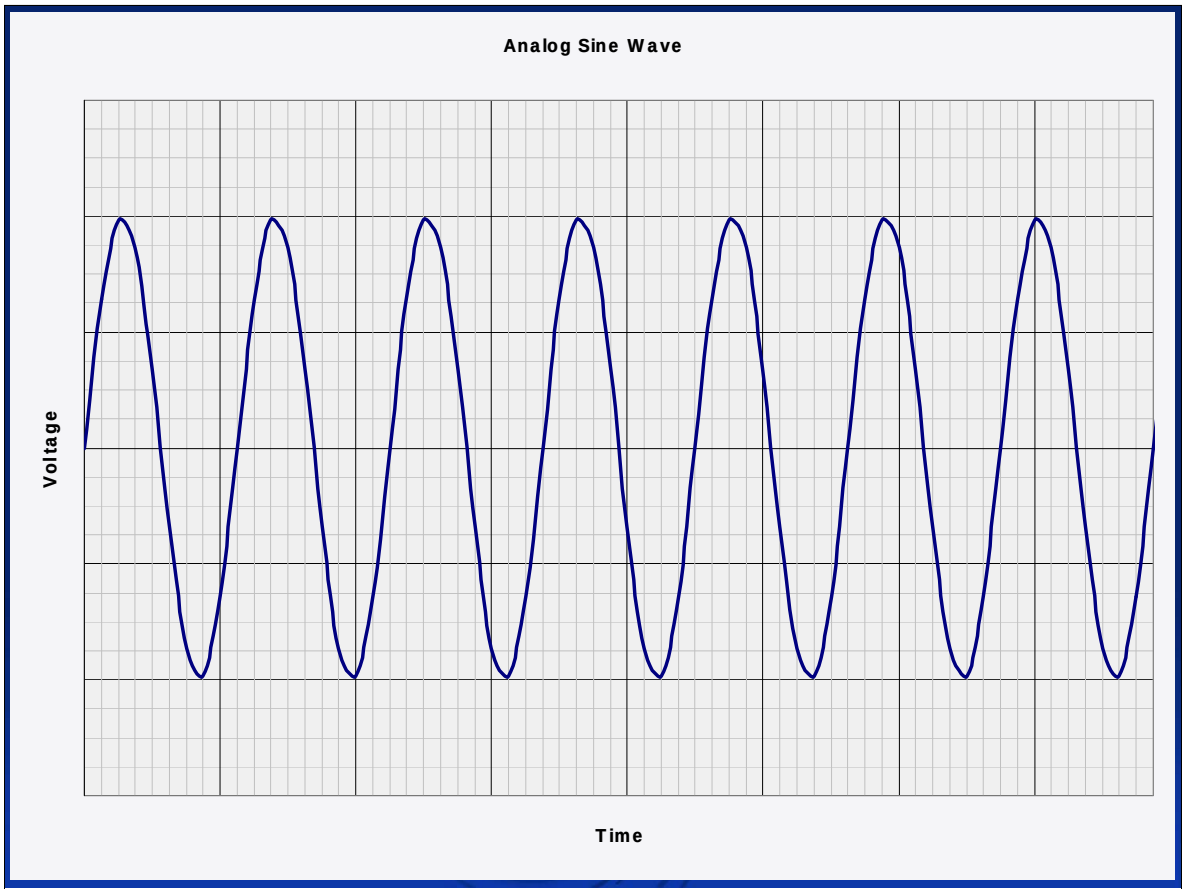
The three visible peaks represent the three distinct sine waves.

The tallest (off the top of the scale) peak at 0 Hz is the DC offset of the signal (more about that later).

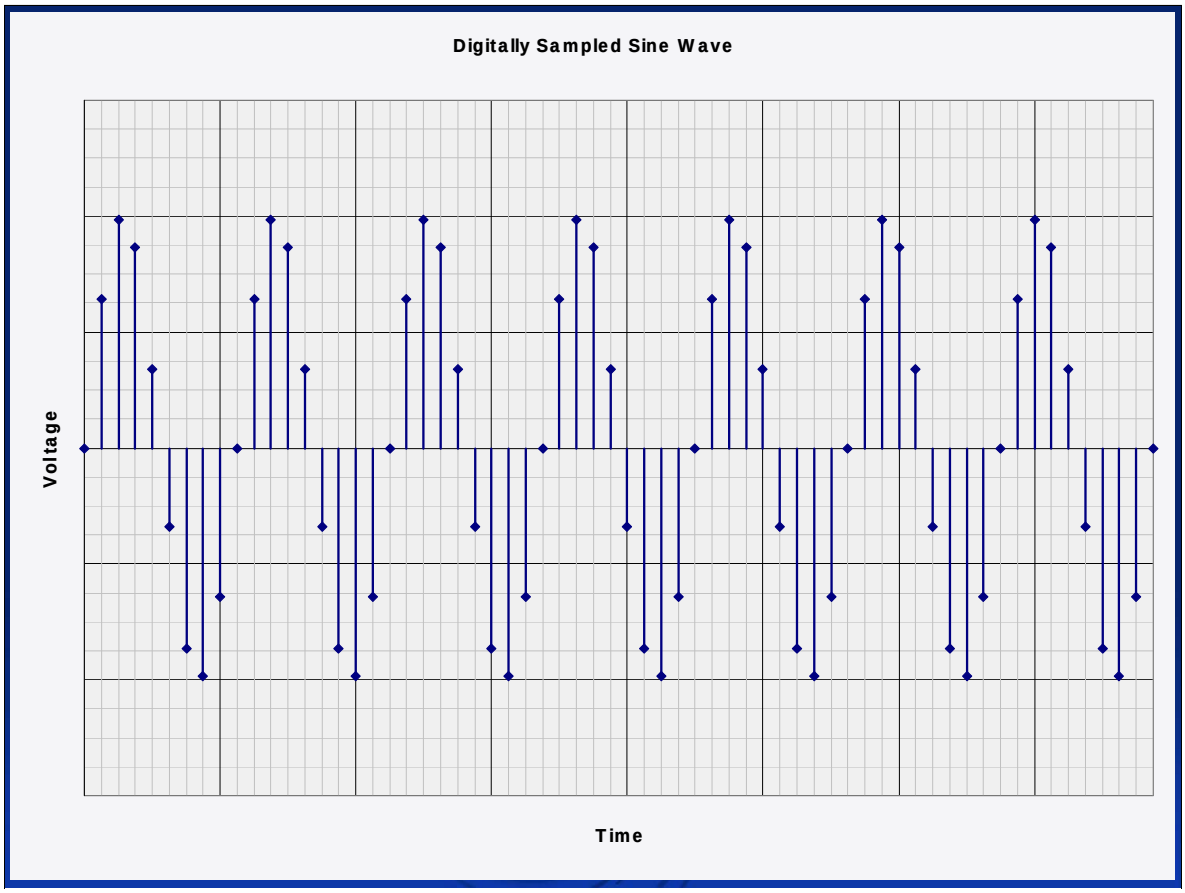
The amplitude of the peaks represents the amplitudes of the three sine wave components.



Before we can get into how to handle real-world signals – instead of mathematically perfect signals – we have to know a little about digitized signals.



Here's our friend, the analog sine wave.



And here's what it looks like to a computer.

The voltage is sampled periodically (as opposed to the continuous nature of the mathematical signals discussed previously).

When the data points are plotted, this is what it looks like.

This digital time domain data is the digital representation of a continuous analog signal.

The Discrete Fourier Transform

The Discrete Fourier Transform

- Abbreviated DFT
- A way to implement the Fourier Transform with discrete (i.e., digital) data.
- 

The analog Fourier transform is all fine and dandy if you have a perfect mathematical representation of a signal.

This never happens with real-world signals.

We need a way to handle imperfect signals, signals that can't be conveniently described by a few summed sine functions.

The way to do this is to sample the waveform digitally and do the Fourier transform discretely.

The Discrete Fourier Transform

$$X\left(kF\right)=\sum_{n=0}^{N-1}x\left(nT\right)e^{-j2\pi kFnT}$$

where

$X\left(kF\right)$ = frequency domain representation

$x\left(nT\right)$ = time domain representation

k = frequency channel number

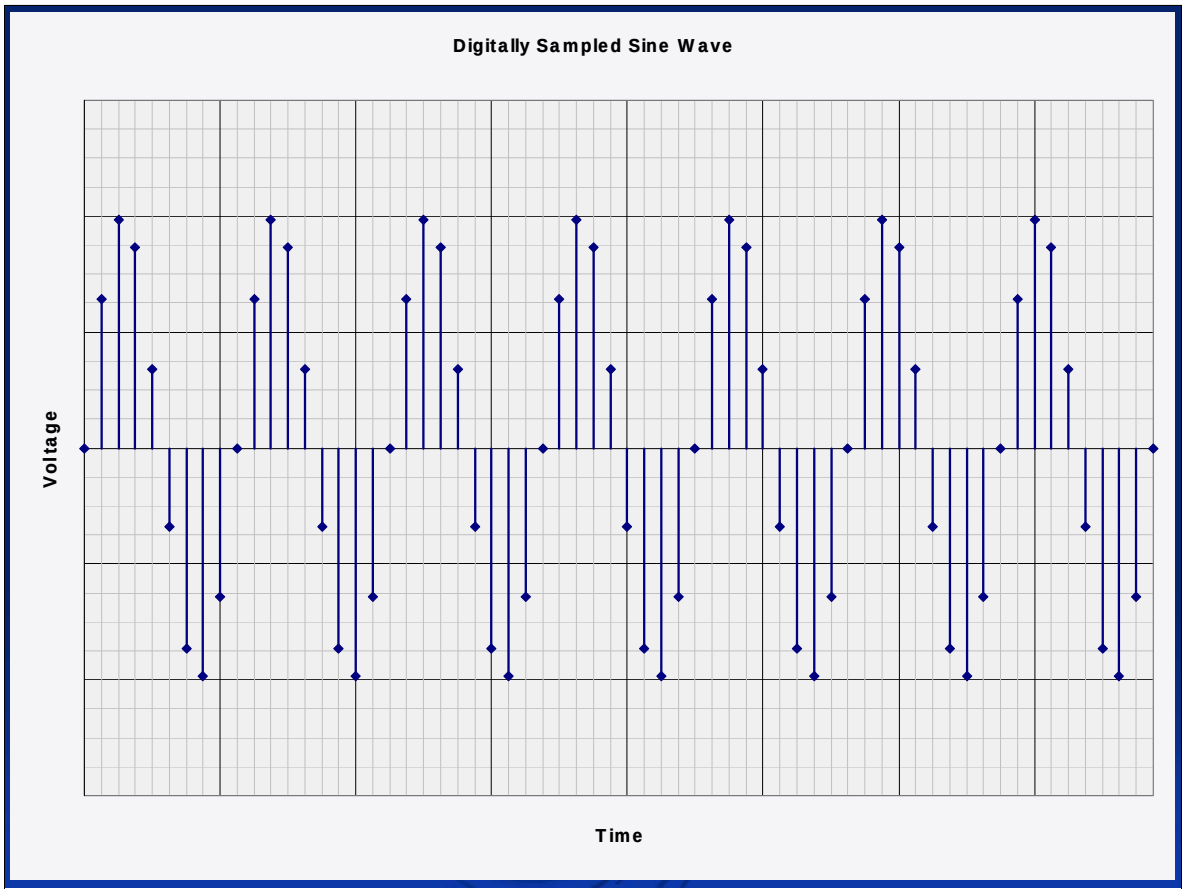
F = spacing between discrete frequencies

n = sample number

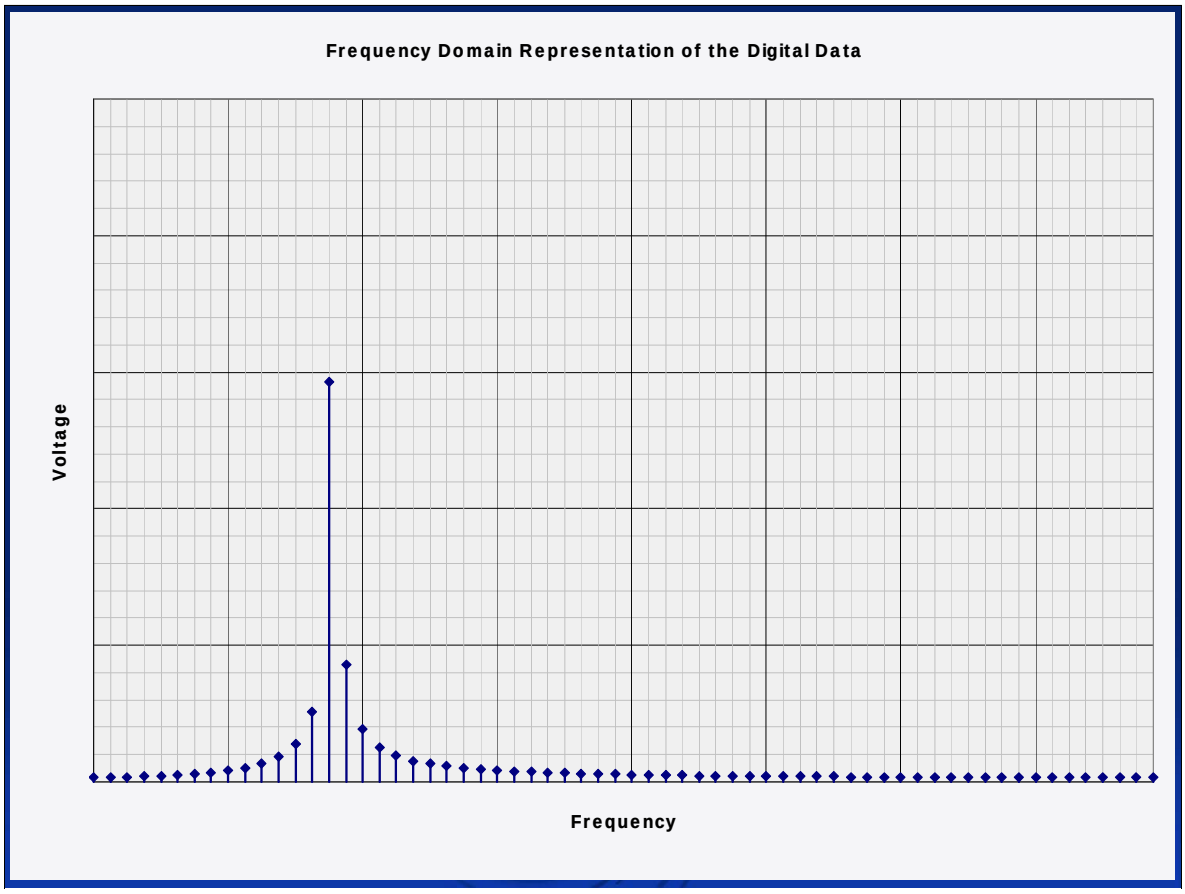
T = time between samples

N = number of samples in the FFT input

The gory details. Again, the mathematical details are beyond our purpose here. It is sufficient to say that If you have digital data, you can use the DFT to find the freq domain representation of your data. You don't perform square roots by hand, do you? Then don't worry about performing DFT's by hand, either.



Here's the digitized sine wave again.

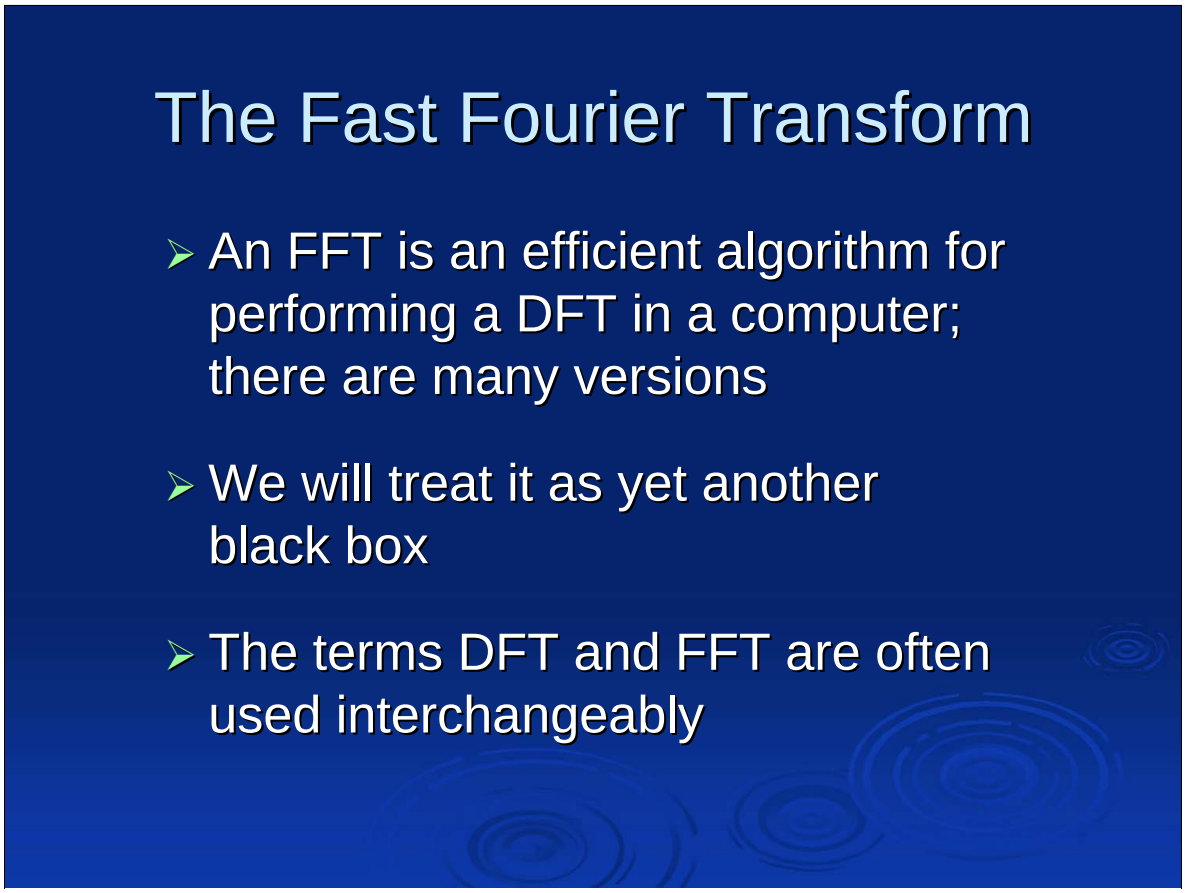


And here's what happens when you run that digitized sine wave through a DFT. You get the frequency domain representation of digital data. Digital data in, digital data out.

Which leads us to...

The Fast Fourier Transform

Our purpose of being here today (well, for this presentation, anyway).

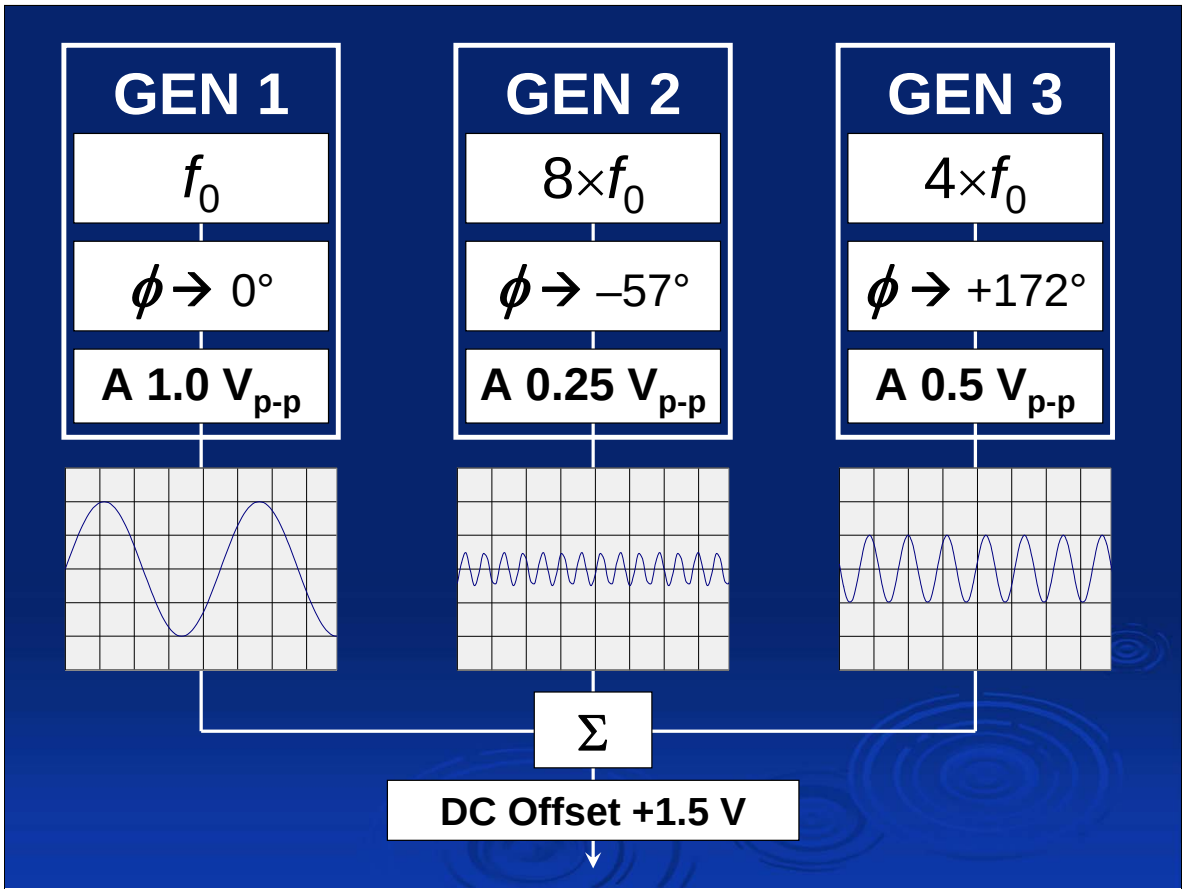


An **FFT** is an efficient algorithm that implements the DFT equation in a computer program that will execute quickly.

Note the “an FFT” – there are a whole bunch of them.

The code is simple to a computer, but complex by (normal) human standards; we will treat it as yet another black box.

The terms DFT and FFT are often used interchangeably, even though they are not quite the same thing. The **DFT** is the ***math***. The **FFT** is a ***computer program*** that makes a computer perform the necessary calculations.

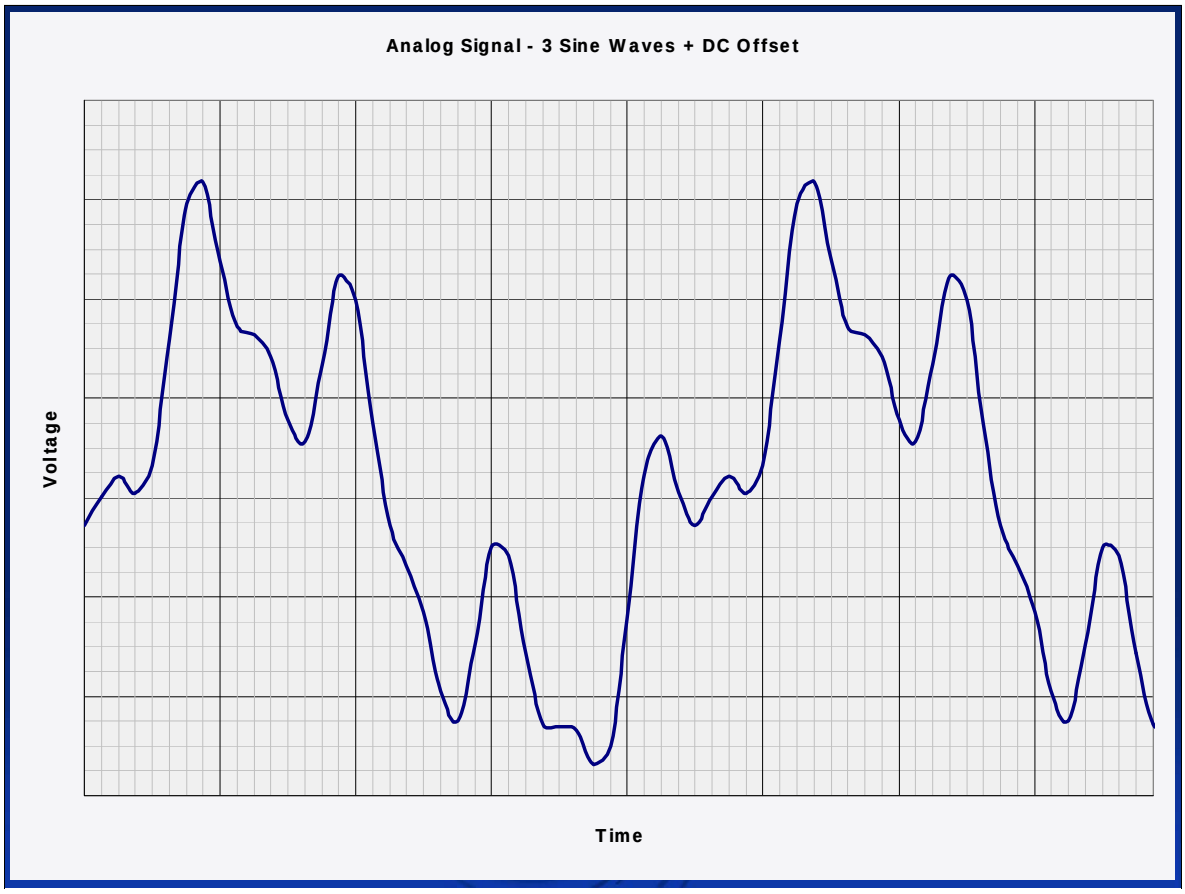


To see how the FFT behaves, let's come up with a test signal to feed it.

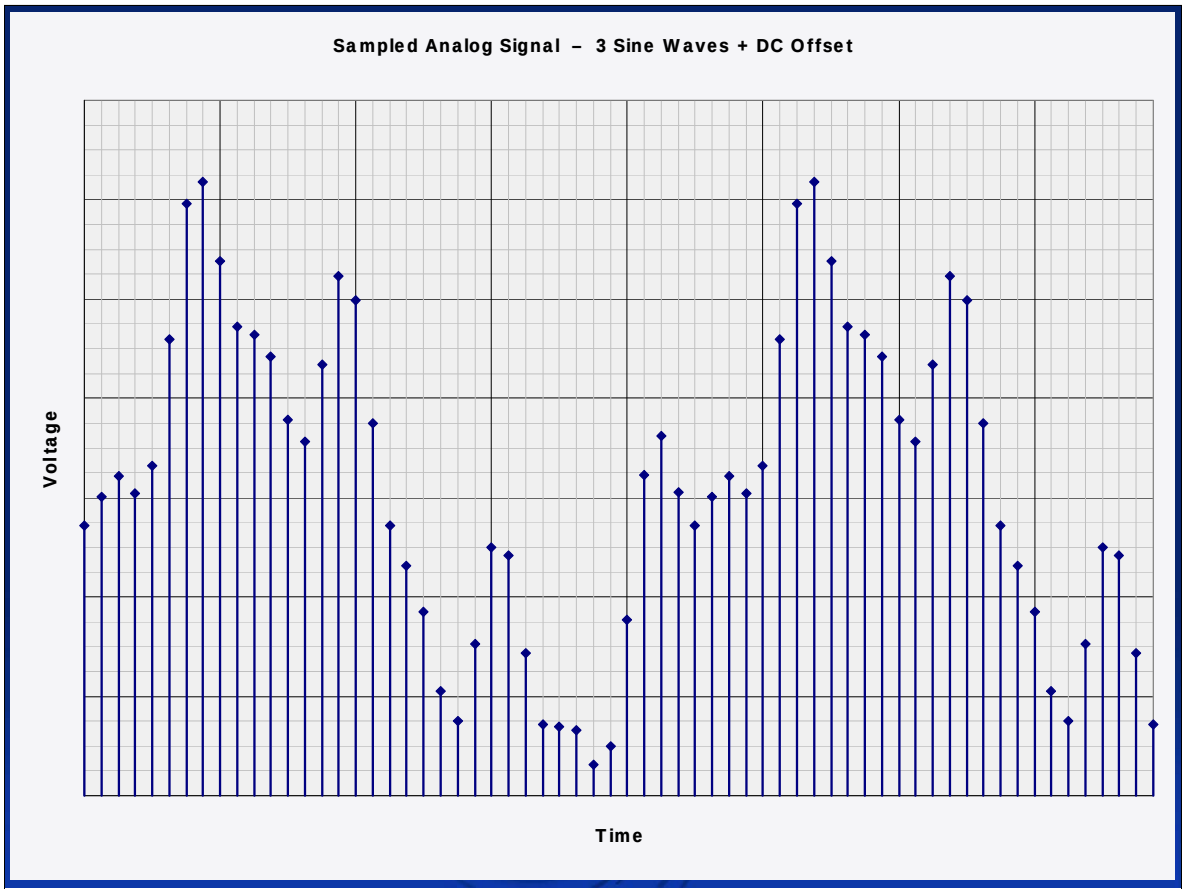
Imagine we have these three signal generators connected through a power combiner.

Each generator has an independent frequency, phase, and amplitude (or gain).

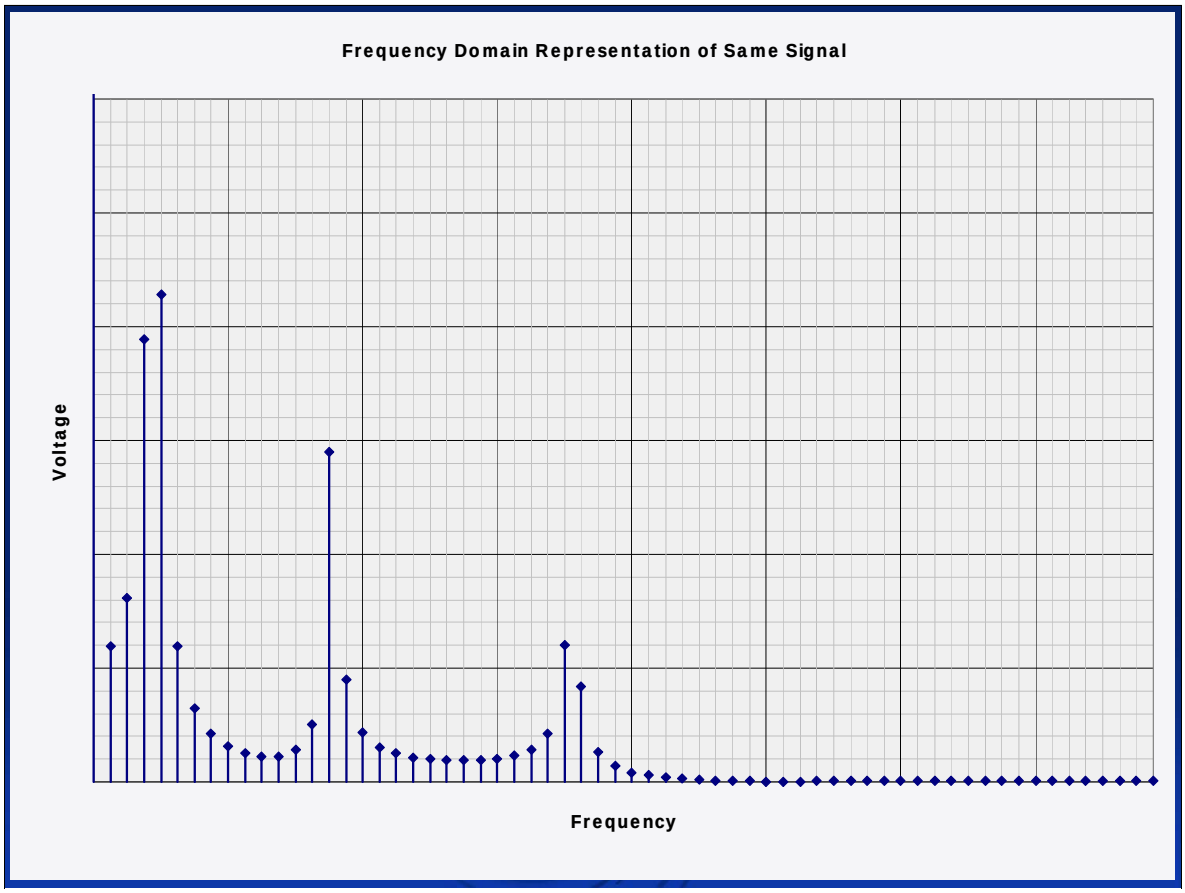
We also have at our disposal a bias tee to provide a DC offset to the final output of this rig.



Here's the output of the three summed sig gens, plus some DC offset. The DC offset isn't visible in this plot, but it's there – as the FFT output will show.



Here is the digital representation of the three sine waves.

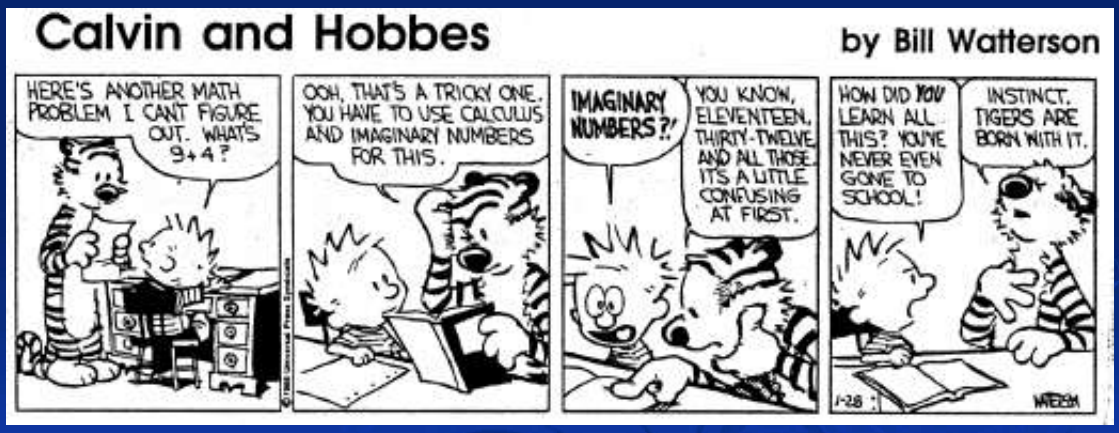


And here is the (digital) output of the FFT.

Important Factors for FFT Use

- Complex vs Real Data
- Nyquist and aliasing
- Sample rate, Block Size, and RBW
- Windowing and FFT leakage
- DC offset
- Amplitude Calibration

Complex vs Real Data



Yes, unfortunately, we do have to talk about complex numbers.

Complex vs Real Data

- The FFT is by nature an operation on complex numbers; i.e.:

$$a + jb \qquad A\left(\cos\phi + j\sin\phi\right) \qquad Ae^{j\phi}$$

- These contain amplitude *and phase* information
- **We deal here with *real-valued data only*, as from a single channel signal source – which contains amplitude information *only***

Complex number notation → rectangular form – polar form – exponential form

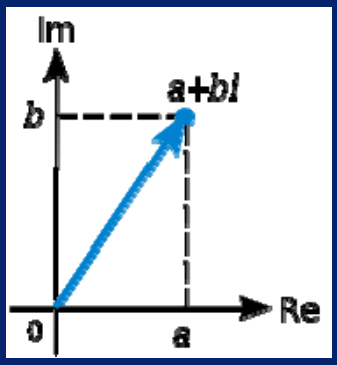
all the same thing

From a single channel digitizer, all you get is a real number; you have no phase information

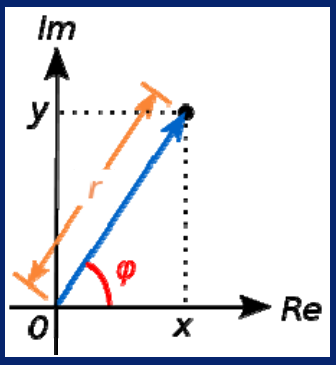
To get complex samples, you need a two channel digitizer with a 90° phase delay between the two channels. This delay must be 90° *at all frequencies of interest* – so for broadband signals, a simple delay line will not work.

In this talk, we work only with *real*-valued samples; i.e., amplitude information *only.*

Complex vs Real Data



$a + jb$



$A(\cos\phi + j\sin\phi)$

and via Euler to

$Ae^{j\phi}$

Our task here isn't to dive too deep into complex numbers, but here are a few graphics to show you what the rectangular and polar forms represent on the complex plane ---

--- and how you can get from a complex number to its magnitude by way of Pythagoras. We'll need that when converting the complex-valued output of the FFT into usable real-valued numbers.

Complex vs Real Data

- When the FFT is provided with real-valued input data, it still produces a complex-valued output, usually of the form $a + jb$
- These contain amplitude *and phase* information

In this talk, we work only with *real*-valued samples; i.e., amplitude information *only.*

An FFT provides complex-valued outputs.

If the input data is real-valued (as opposed to complex), the FFT simply treats the data as being complex with a zero-value imaginary part. This makes the phase information in the complex-valued output data of questionable value. There might be a use for it, but I have not discovered one.

Complex vs Real Data

- To obtain the amplitude at each frequency, we simply find the vector magnitudes of the complex-valued FFT output data points:

$$A = |a + jb| = \sqrt{a^2 + b^2}$$

For our purposes, we simply convert the output to real numbers by finding the vector magnitude of the complex output values. This obtains the amplitude at each frequency.

Complex vs Real Data

- Excel → Tools → Data Analysis → Fourier Analysis
- Excel IMABS() function to find FFT amplitudes

F11 =IMABS(E11)

	A	B	C	D	E	F
1	FFT Talk Plots					
2	DT 2014					
3						
4						
5						
6						
7					FFT Block size = 128	
8	Sample	x	x		Fourier	Fourier
9	Number	(deg)	(radians)	Sin(4x)	Complex	Abs
10	1	0	0	0.00	0.642787609686513	0.64278761
11	2	10	0.174533	0.64	0.648662505105898-0.103806693382756i	0.656916186
12	3	20	0.349066	0.98	0.666643115146548-0.210630881283289i	0.699126892
13	4	30	0.523599	0.87	0.697852459452679-0.323802201291547i	0.769315228
14	5	40	0.698132	0.34	0.7443630080676-0.447344957970921i	0.86844332
15	6	50	0.872665	-0.34	0.809564403797935-0.586529994043204i	0.999705936
16	7	60	1.047198	-0.87	0.898848458382433-0.748770957896012i	1.169866017
17	8	70	1.22173	-0.98	1.02090055797228-0.945220666659729i	1.391287195
18	9	80	1.396263	-0.64	1.19026603765139-1.19387144073436i	1.685841706

All the plots in this talk were made using Excel 2003. Yes, Excel can do FFT analysis and can handle complex numbers.

The Fourier Analysis routine in the Data Analysis package creates complex-valued output data even when the input data is real-valued.

Excel contains a built-in function, IMABS(), to find the magnitude of a complex number – this is what we want so we can plot the amplitude of the FFT output at each frequency.

If your installation of Excel does not have these functions available, you can enable them by going to:

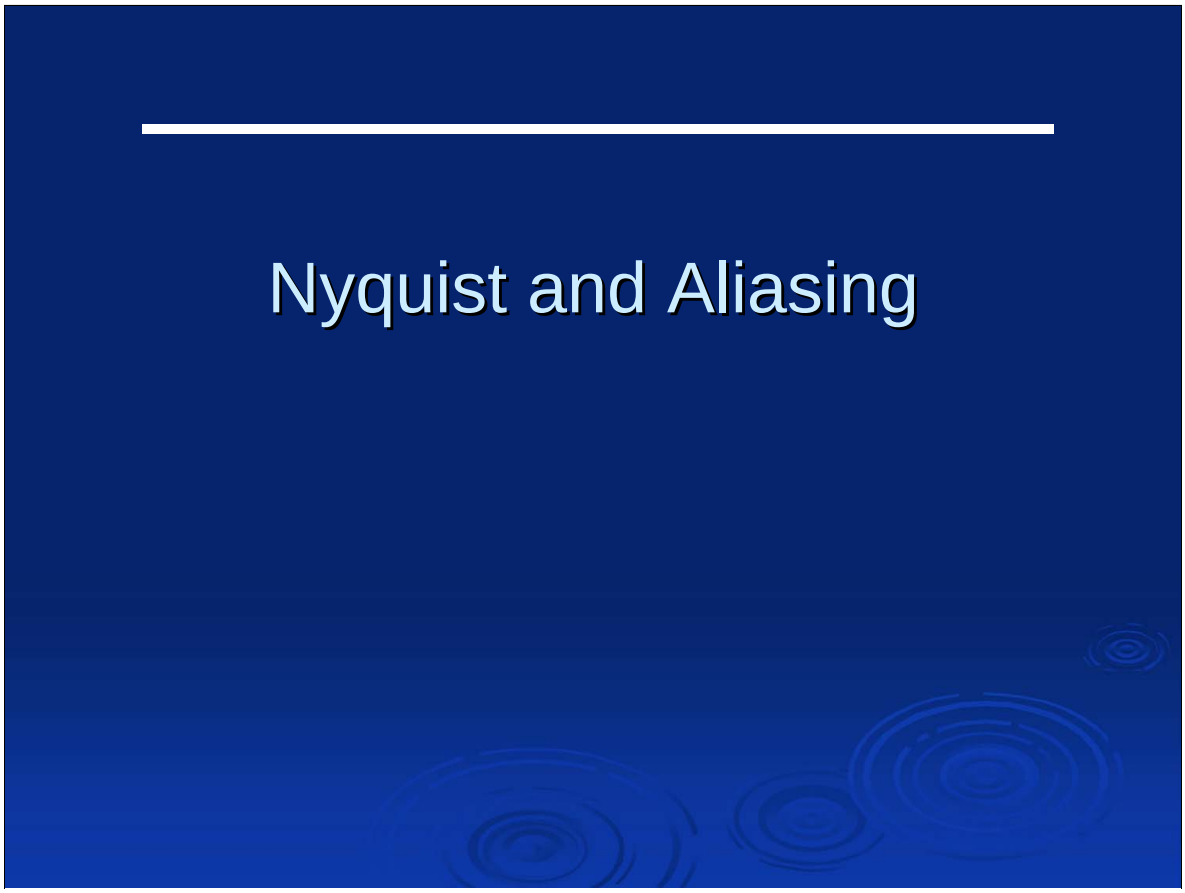
Excel menu bar → Tools → Add-Ins... → ensure the entry for Analysis ToolPak is checked.

This Excel file is available for anyone who wants to play with it:

http://www.typnet.net/AJ4CO/FFT/Plots.xls

This presentation is also available:

http://www.typnet.net/AJ4CO/FFT/FFT.pdf



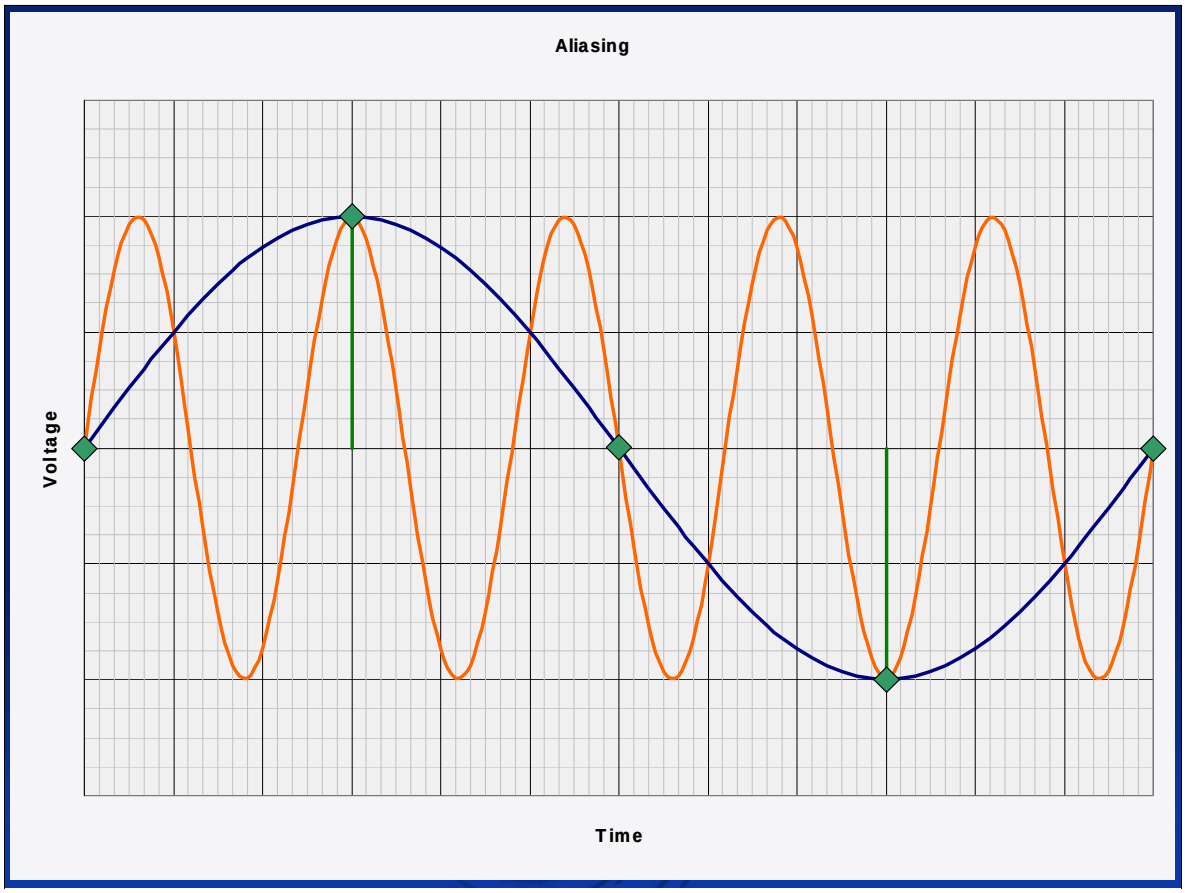
Now we **finally** get into the real meat of this presentation – the stuff that's **fun** and **good to know** about FFT's.

Nyquist says that the sample rate must be greater than twice the max freq **present** in the signal.

If f_max is the highest frequency component within some signal we want to digitally sample, Nyquist says that in order to fully capture all the information in the signal, the sample rate must be greater than twice f_max. That's for an infinitely long input to the FFT. For real world data sets, the sample rate should be greater than about 2.5 times the highest frequency component present in the data. Well, more or less. Point is you need a bit more than the theoretical minimum of 2 times.

NOTE: if the sample rate isn't high enough – or if there are components in the signal with frequencies greater than half the sample rate, you will get into aliasing, which is bad. We will discuss this next.

NOTE: Nyquist is not a low pass filter! If your signal has frequency components greater than half the sample rate, run it through a low pass filter before passing it to the FFT.



The green diamonds represent periodic voltage samples.

At this sample rate, the blue trace has 4 data points for one complete cycle – which by Nyquist is way more than enough to reconstruct the sine wave.

The orange trace only has less than 1 data point per cycle – which is way less than the 2 data points per cycle demanded by Nyquist.

Both of these signals would produce the data points shown here (the green dots).

The question is, when the FFT looks at the time series voltage samples, how can it know which of these two signals was really present, orange or blue or both?

It doesn't!

In general, the FFT output forces the assumption that the lowest frequency wave that fits the bill is the correct wave (even if it isn't).

This is what is meant by *aliasing*.

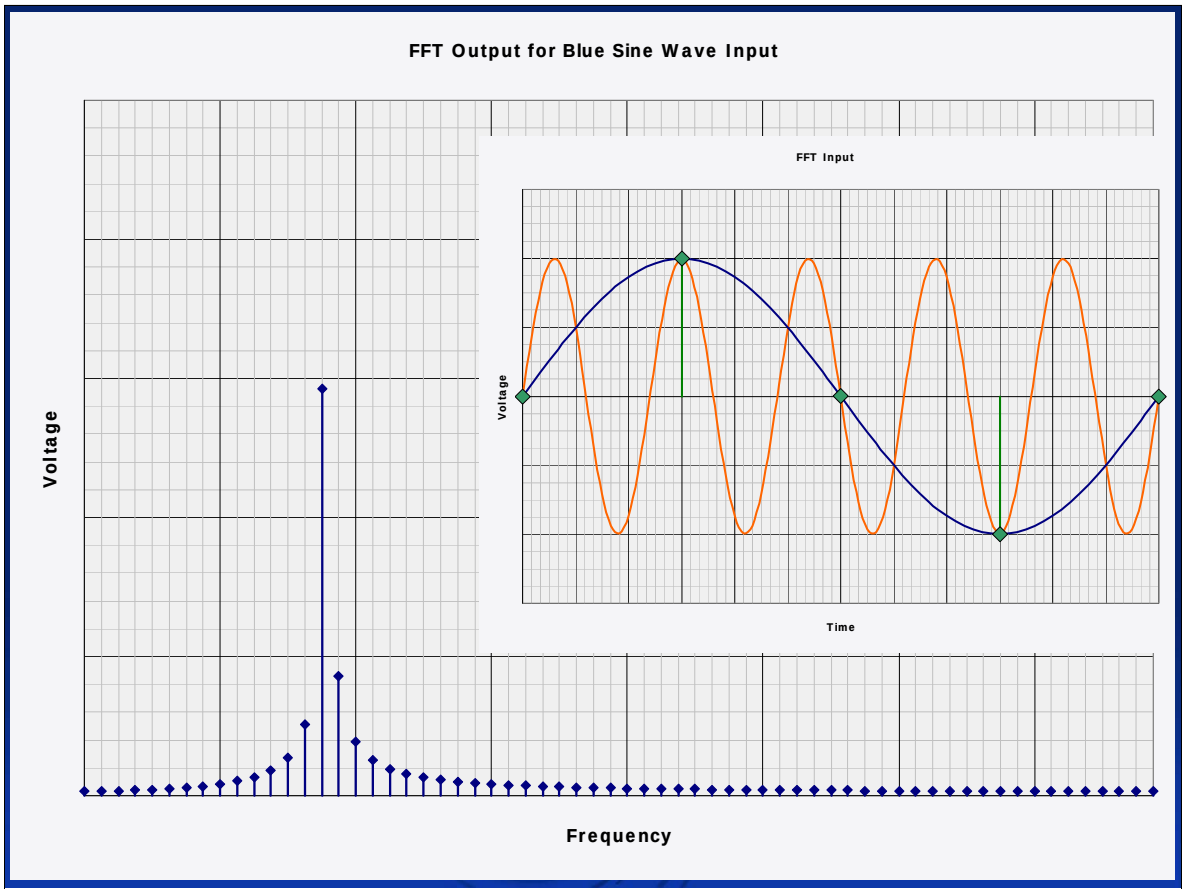
In this example, the orange trace is ALIASED to the lower frequency of the blue trace when the FFT spits out its frequency peaks.

One way to avoid aliases is to simply increase the sample rate so high that there will be no frequency components in the FFT input data that would produce aliases. This, however, is often impractical.

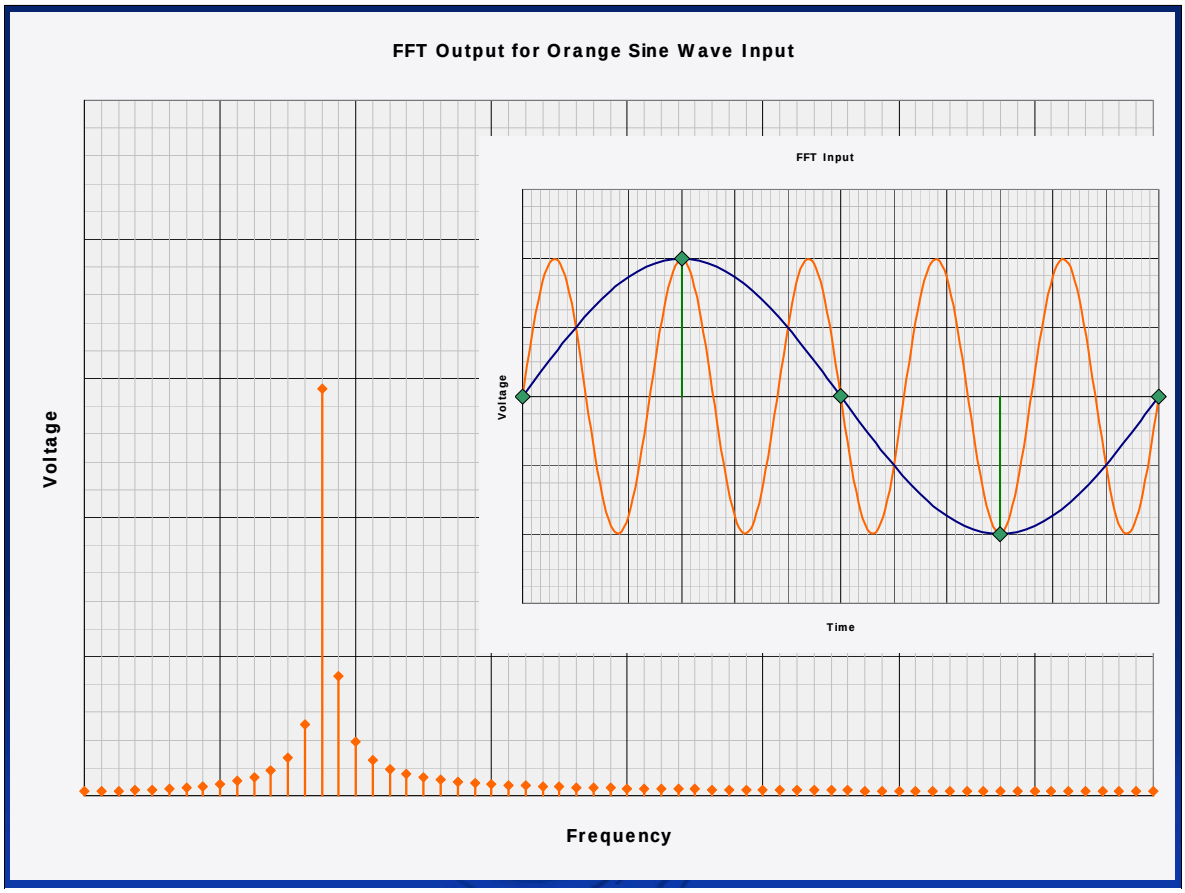
The other way to avoid aliasing is to run the RF signal through a low pass filter to attenuate frequencies that are too high, the ones that would produce aliases in the FFT output.

Most real-world applications use a combination of both methods.

Aliasing is the major reason for the 2.5 times f_max sample rate (as opposed to the 2.0 times f_max Nyquist rate). It's impossible to construct a real world low pass filter to block all the frequencies that are higher than some arbitrary cutoff. By sampling somewhat faster than might be theoretically needed, we avoid some of the aliasing that might occur from frequency components present in the real world signal just above half the sample rate.



Here's the FFT output for the blue sine wave.



Here's the FFT output for the orange sine wave. Note that it is the same as the FFT output for the blue sine wave. This is aliasing in action.

Sample Rate, Block Size, and RBW

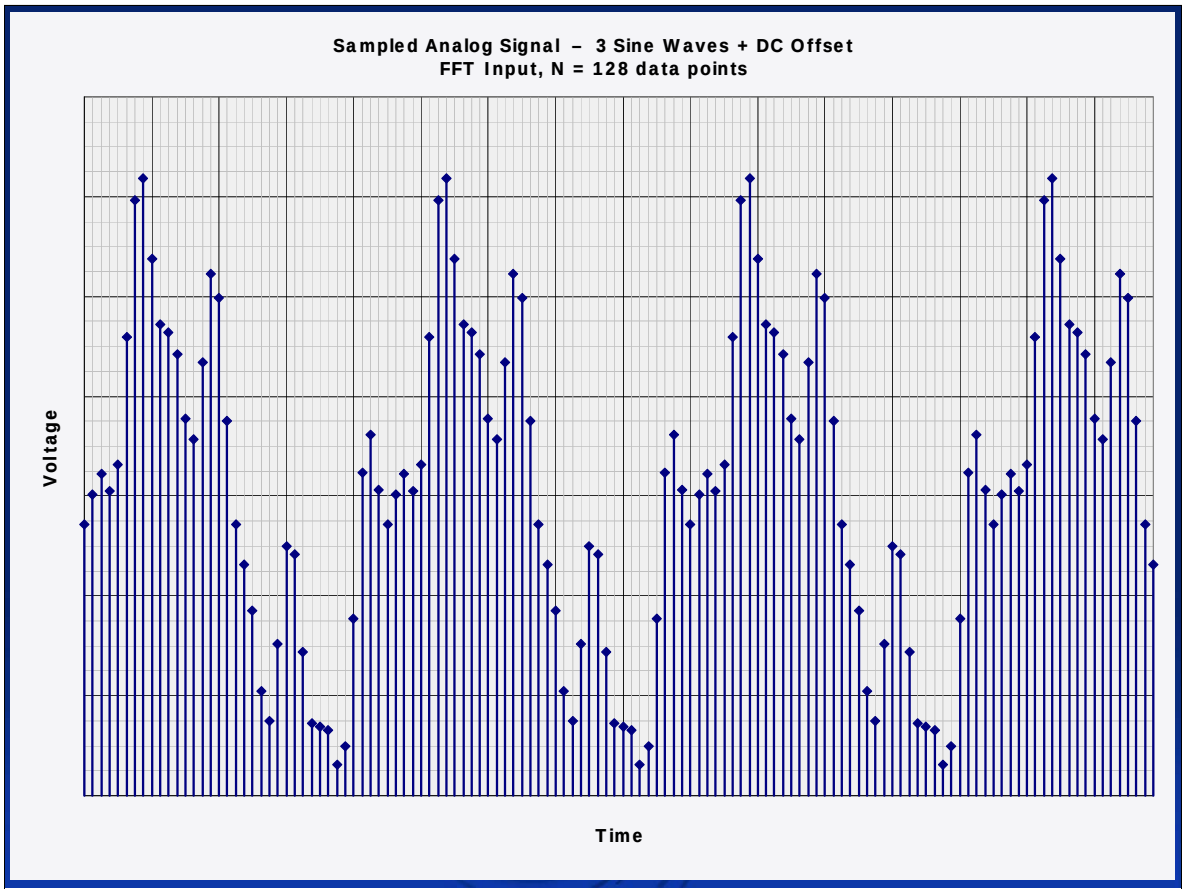
Sample Rate, Block Size, and RBW

- An FFT is record-based; it operates on *N* input samples at a time
- *N* is also known as the FFT block size
- *N* is usually a power of 2 (but not always)

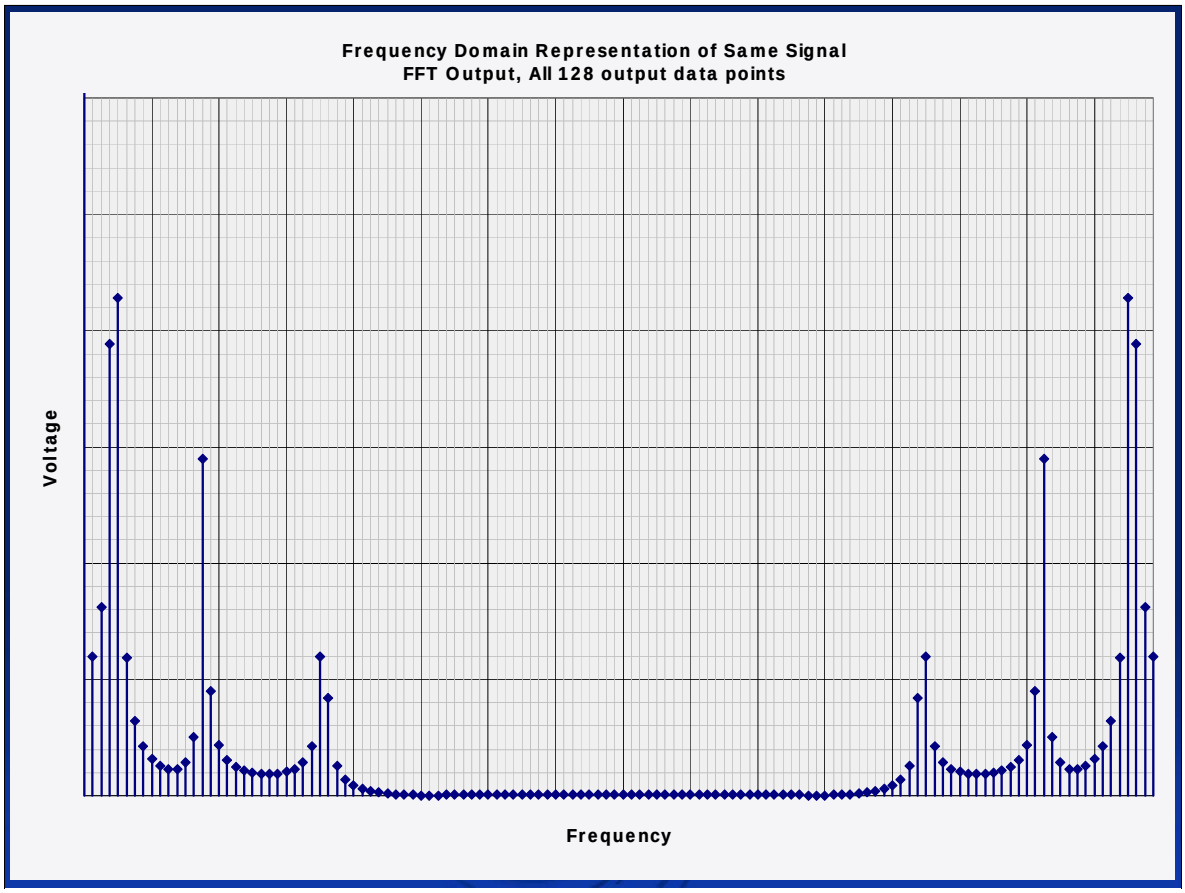
For example, Excel requires N to be an integer power of 2. Mathematica, on the other hand, will accept a block size of any length.

Sample Rate, Block Size, and RBW

- An FFT produces *N* output points in the frequency domain for *N* input samples in the time domain
- For real-valued input data, only the first (*N*/2)+1 output points are unique. The output spectrum is *folded* at the FFT output frequency bin representing half the sample rate.

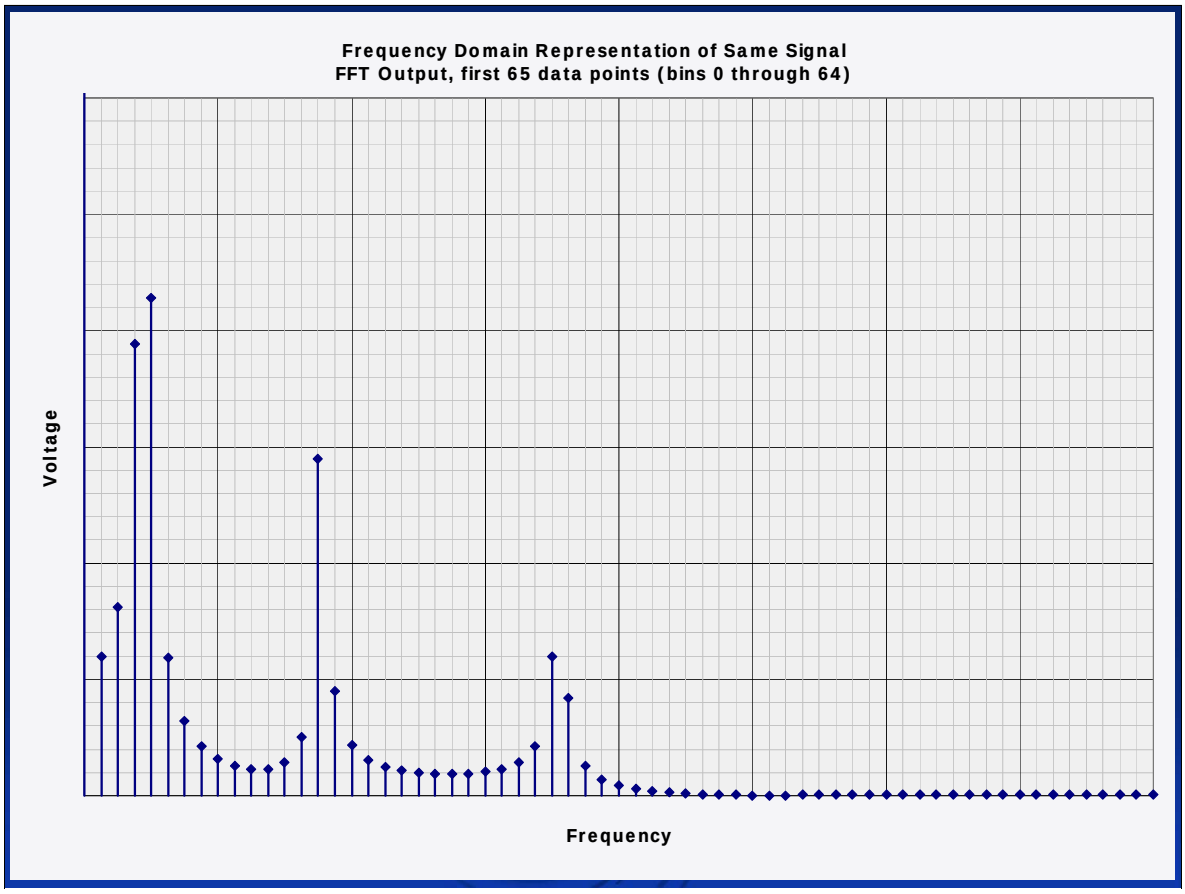


Here's our three-sine-wave signal again.



Here's the full FFT output.

Only the first (*N*/2)+1 output points are useful. (128/2)+1 = 65. The data is folded at bin # 64.



Here are the first (*N*/2)+1 = 65 output points, bins 0 through 64. This is the useful portion of the FFT output.

There's no reason one could not use the upper half of the FFT output instead – assuming one reversed their order to put the spectrum back in the right direction.

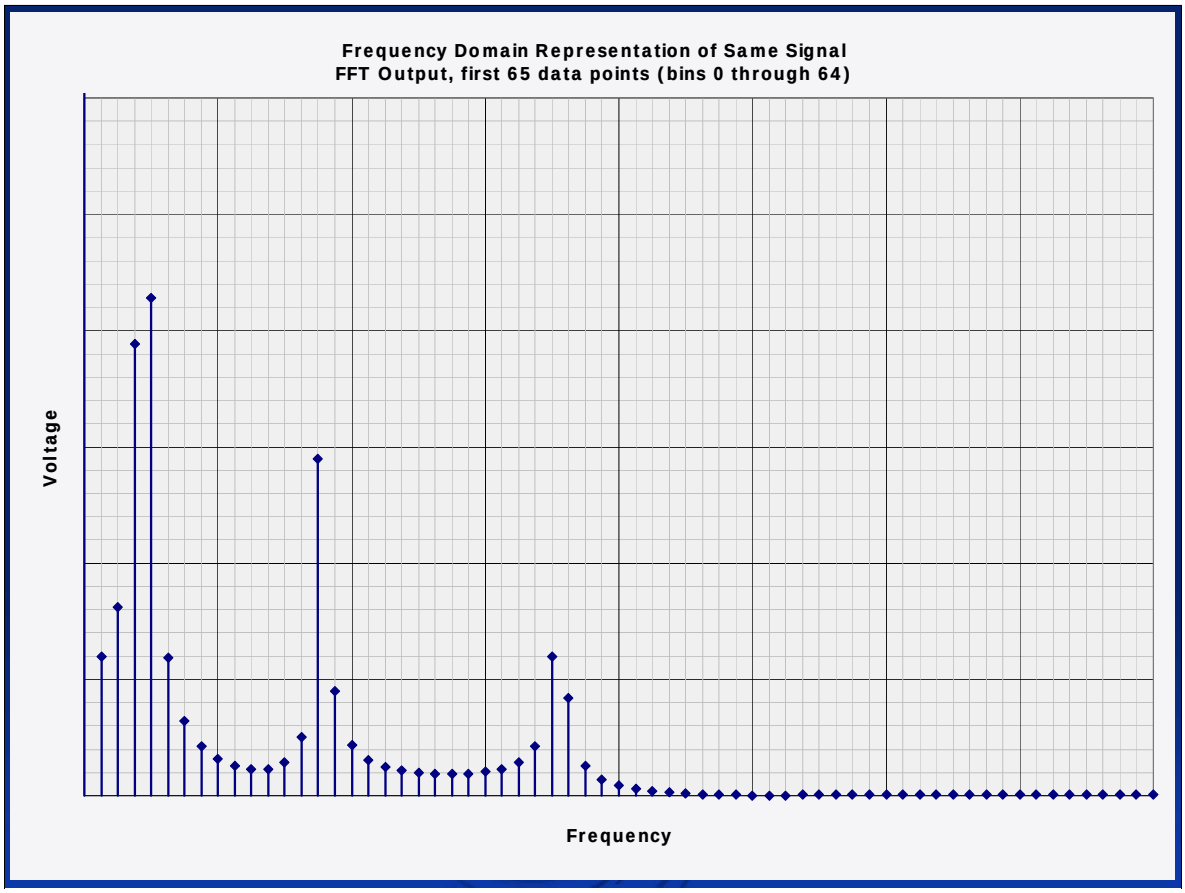
Common practice is to simply discard the upper half of the FFT output and use the lower half, since the lower half's elements are already in the right order, frequency-wise.

Sample Rate, Block Size, and RBW

- Resolution Bandwidth (RBW) is the spacing between the discrete frequencies in the *N* elements of the FFT output array

$$\text{FFT block duration} = \frac{N}{\text{Sample rate}}$$

$$\text{RBW} = \frac{1}{\text{FFT Block duration}} = \frac{\text{Sample rate}}{N}$$



Here again are the first (*N*/2)+1 output points, bins 0 through 64 of the FFT for the three-sine-wave signal.

The question is, how far apart in frequency are these data points?

We know that N = 128 input data points.

If we say that the sample rate is 10 MHz, then it follows that
RBW = (sample rate)/N = 10MHz/128 = 78.125 kHz

Element n of the N-element output array has a frequency of n(RBW).
NOTE: the first element of the output array is element zero, not element one!
So, the first FFT output bin is always 0(RBW) = 0 Hz. (This is where the DC offset lives.)

In this example, the element #64 frequency is 64(78.125 Hz) = 5 MHz.
Recall that 5 MHz is
a) the folding frequency of the FFT output array, and
b) the max frequency you can usefully represent with a 10 MHz sample rate (when you're using real-valued voltage samples).

Notice that the "peaks" appears a little asymmetrical – this happens when:
A) the exact frequency of the time series data (i.e., the sine waves) don't fall exactly in the center of one FFT output bin. In other words, the power in the frequency domain is shared among a couple neighboring bins. This can be mitigated somewhat by using a longer FFT block size – i.e., a smaller RBW – at the expense of time resolution (or shelling out for a faster digitizer if you want to keep the same time resolution). No free lunches!
and
B) the signal doesn't repeat with a period of exactly one FFT block. We'll talk about this in just a bit.

Sample Rate, Block Size, and RBW

- Real World Example (from the TWB receiver)
  - Sample rate = 10 MHz
  - FFT block size = *N* = 2,048
- Therefore:
  - FFT block duration = 2,048 / 10MHz = 204.8 μs
  - RBW = 1 / 204.8 μs = 10 MHz / 2,048 = 4.883 kHz
  - Folding frequency = 10 MHz / 2 = 5 MHz
  - Unique output bins = (N/2)+1 = 1,025 (bins 0 through 1,024)

$$\text{FFT block duration} = \frac{N}{\text{Sample rate}} \qquad \text{RBW} = \frac{1}{\text{FFT Block duration}} = \frac{\text{Sample rate}}{N}$$

Here's a real-world example; this is what we used in the tunable wideband receiver's digitizer and data analysis software.

Windowing and FFT Leakage

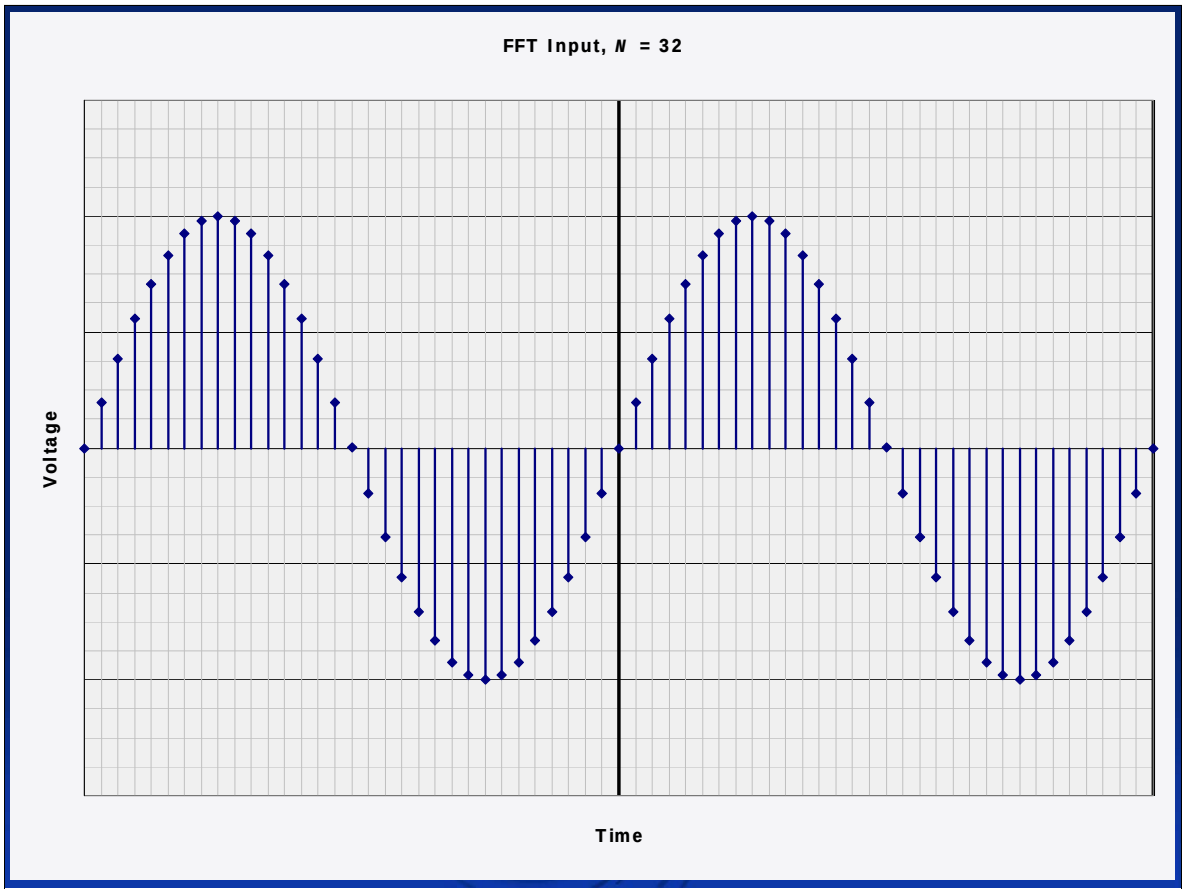
An attempt to deal with real world imperfections.

Windowing and FFT Leakage

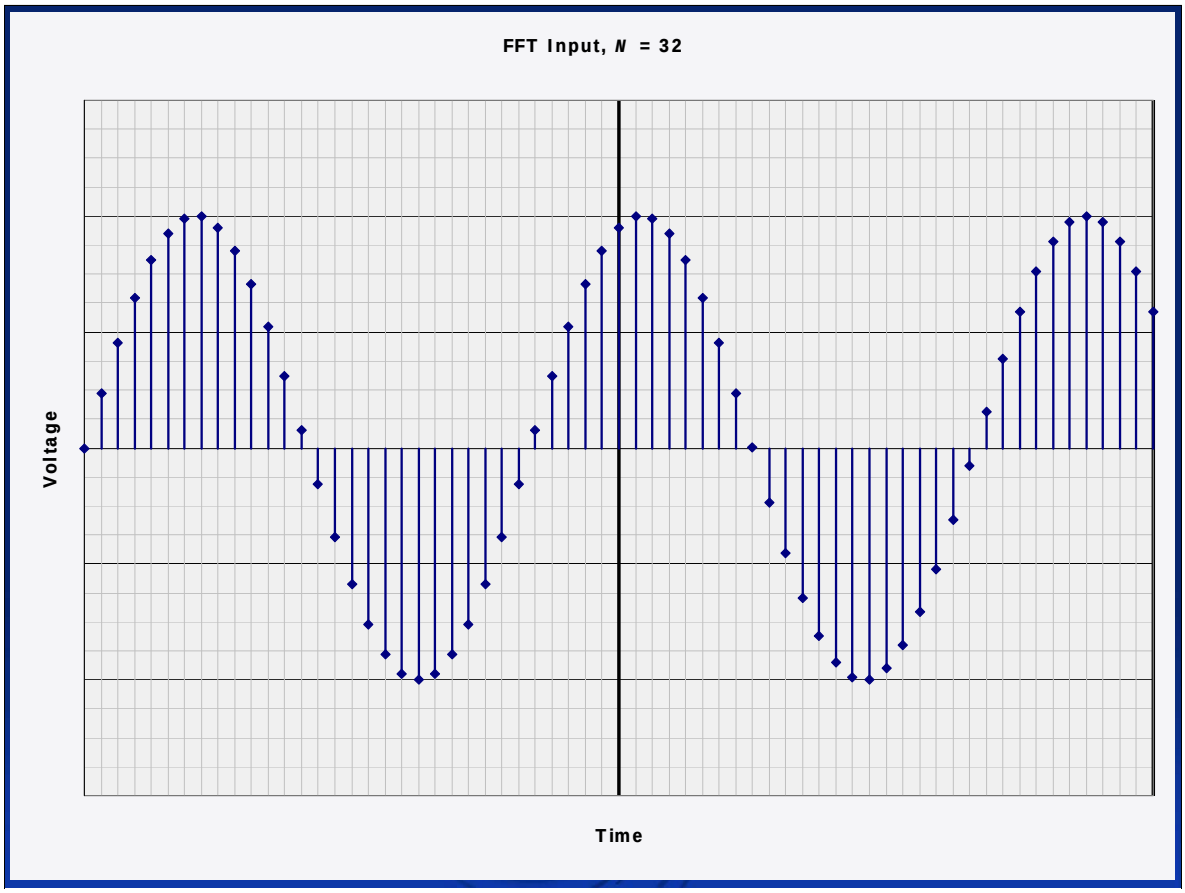
- DFT *assumes* an infinitely long periodic waveform
- Can lead to *leakage* if a *window* is not used
- More important for deterministic signals (e.g., an oscillator's sine wave output)
- Not so important for noise signals (e.g., cosmic radio emission)

The DFT – and by extension, the FFT – assumes that the FFT block of time series data (the input record) repeats *infinitely* in time. This often leads to visible effects in the FFT output.

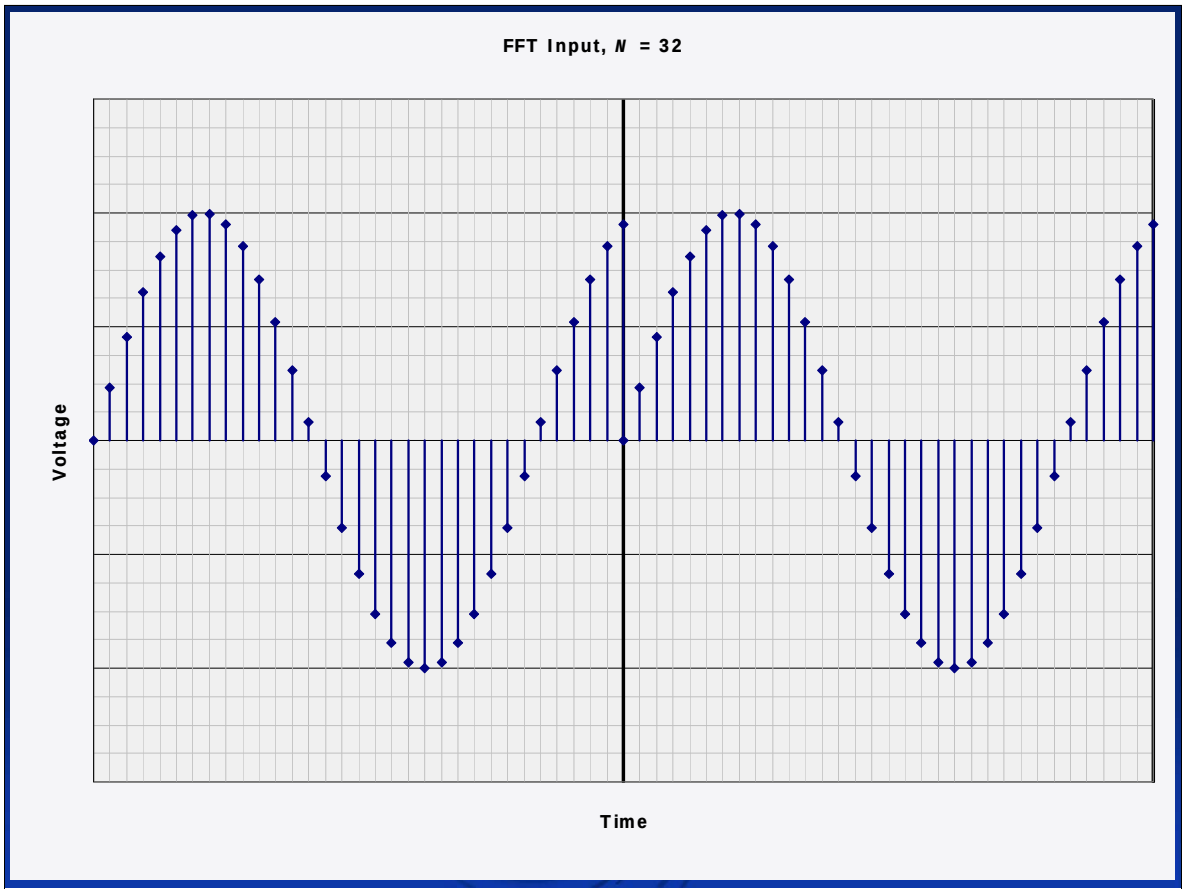
The best way to see this is by example.



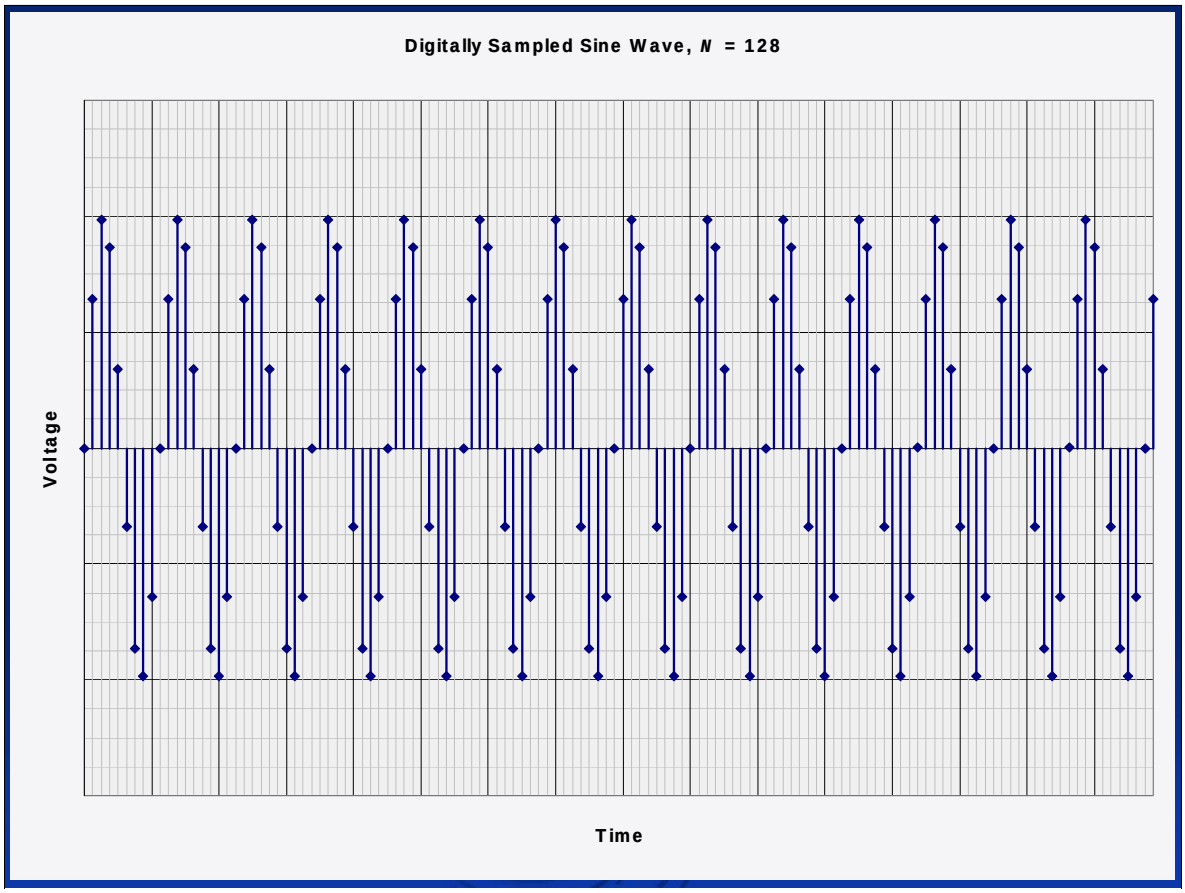
Here is a digitized signal the happens to fit perfectly in the FFT block. Two blocks are shown here (total of 64 data points). Note that the sine wave moves smoothly from one block to the next.



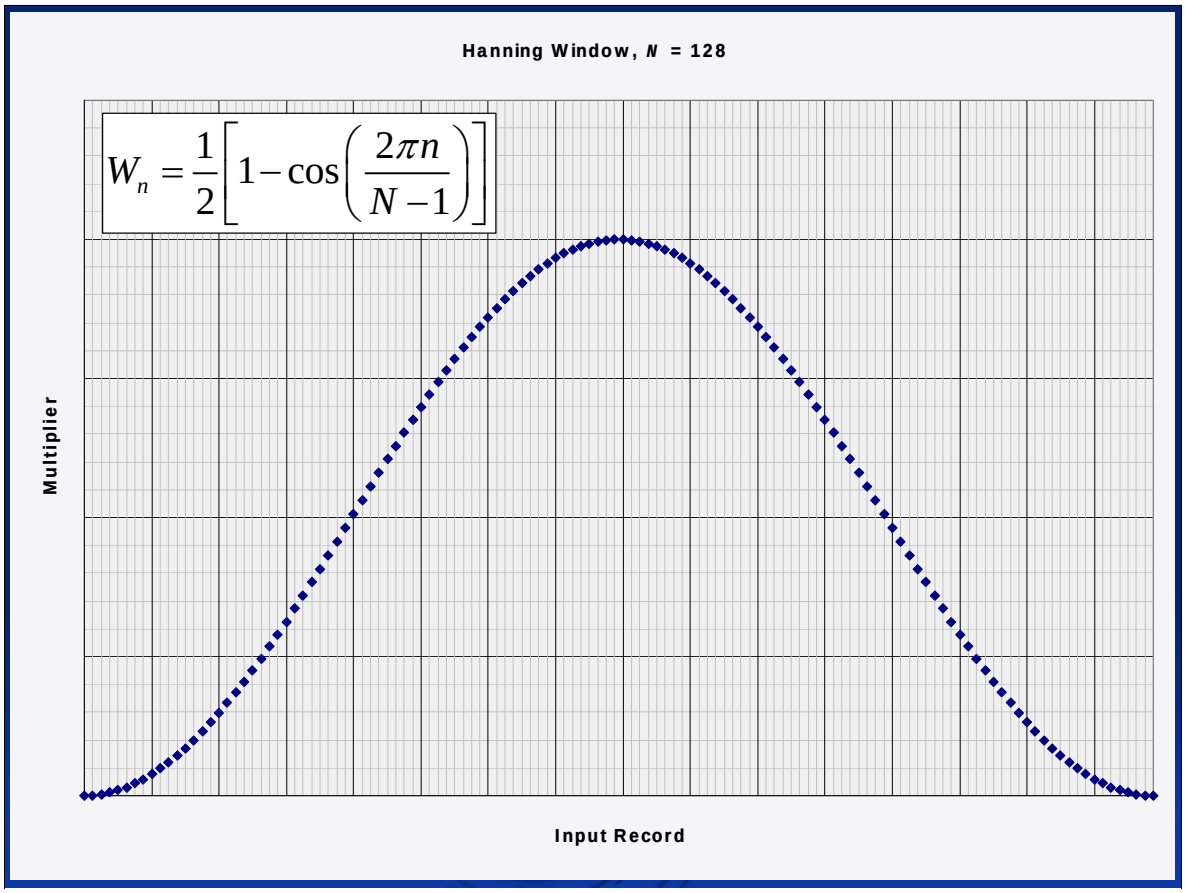
Here is a sine wave that does not fit nicely into one FFT block.



When the FFT is performed on the first input block, remember that the FFT *assumes* the signal in that block repeats ad infinitum. So, to the FFT, the signal actually looks like this, with the small discontinuity between the two blocks of data. This can introduce “leakage” into the FFT output. That is some of the spectral power “leaks” into multiple output bins.



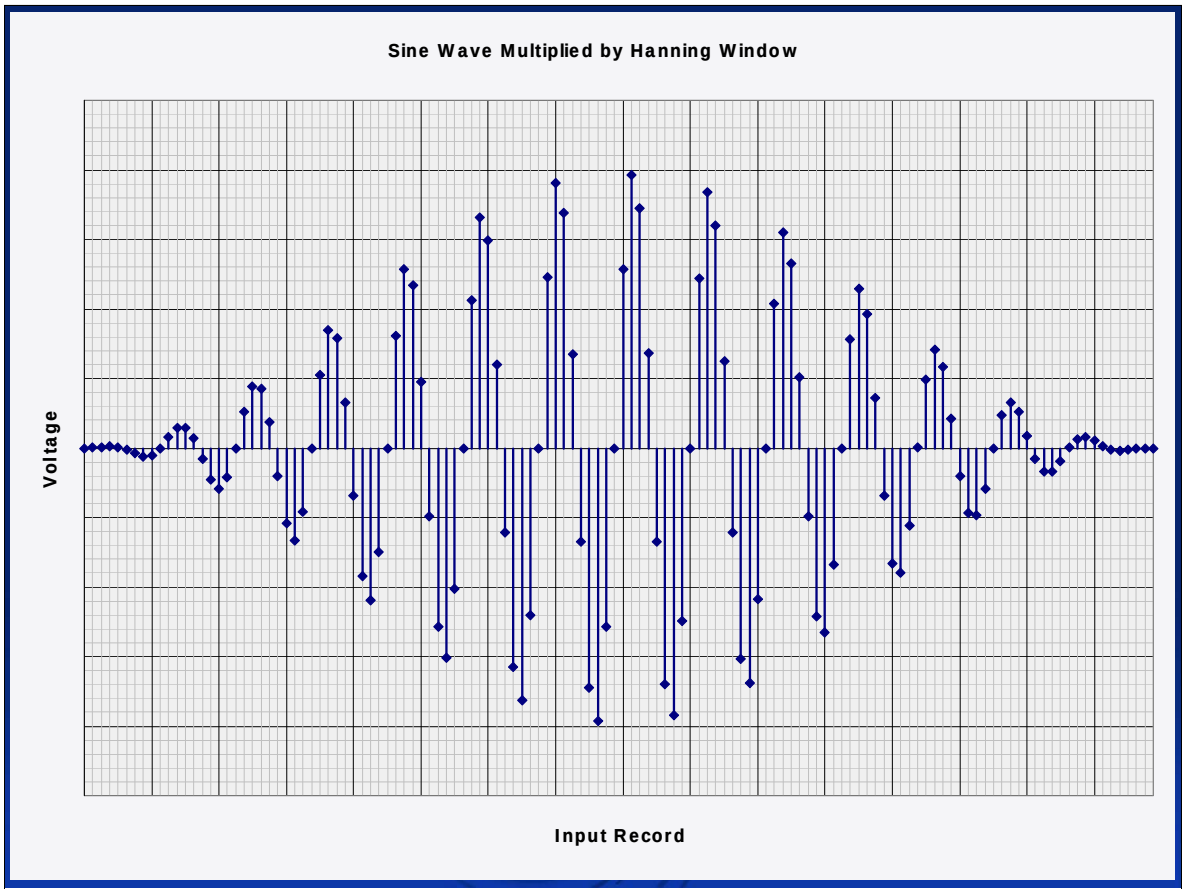
Here's a sine wave that doesn't quite fit an FFT block of 128 data points.



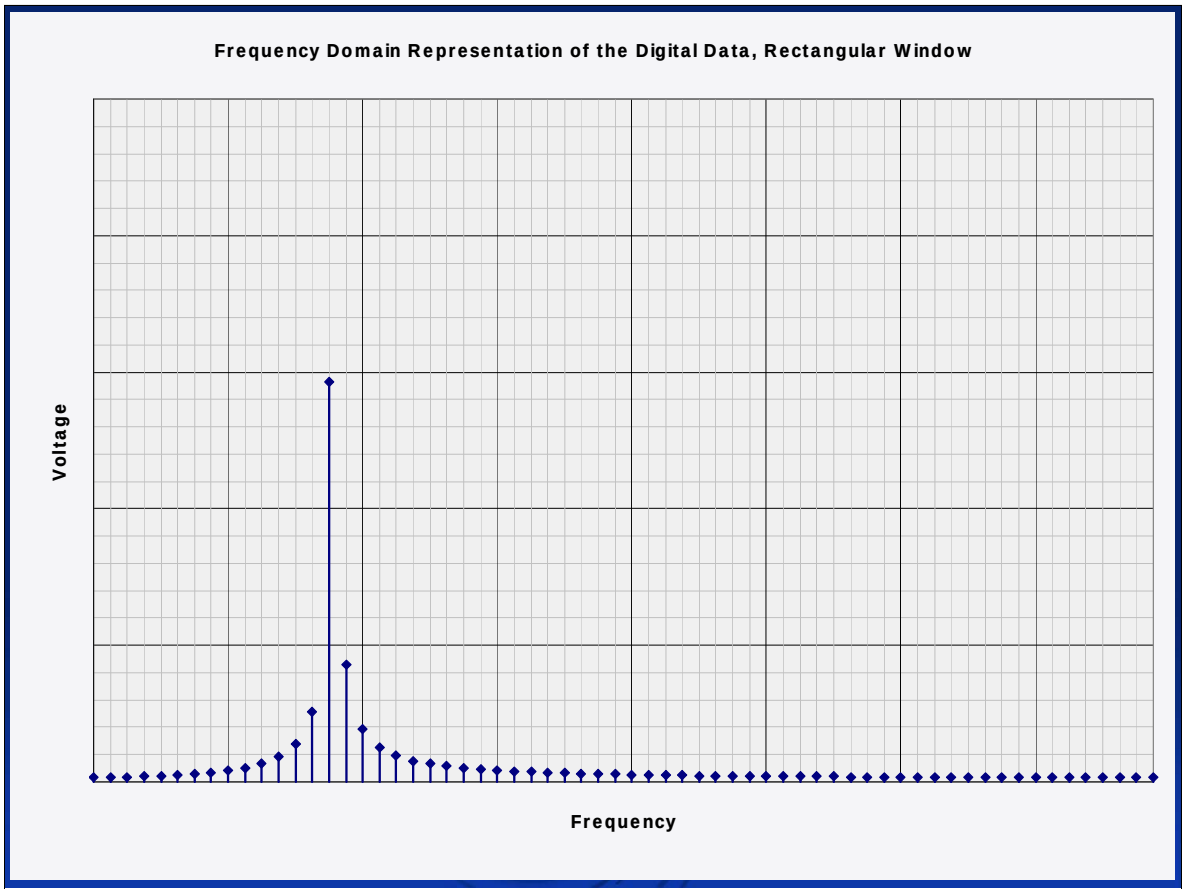
Remember the discontinuity between the two sine waves? Well, if we multiply each FFT block by a *window* to bring the ends of the block to zero, then the signal *can* repeat endlessly without discontinuities – because the end points are all forced to be zeroes.

This particular window is called the *Hanning* window.

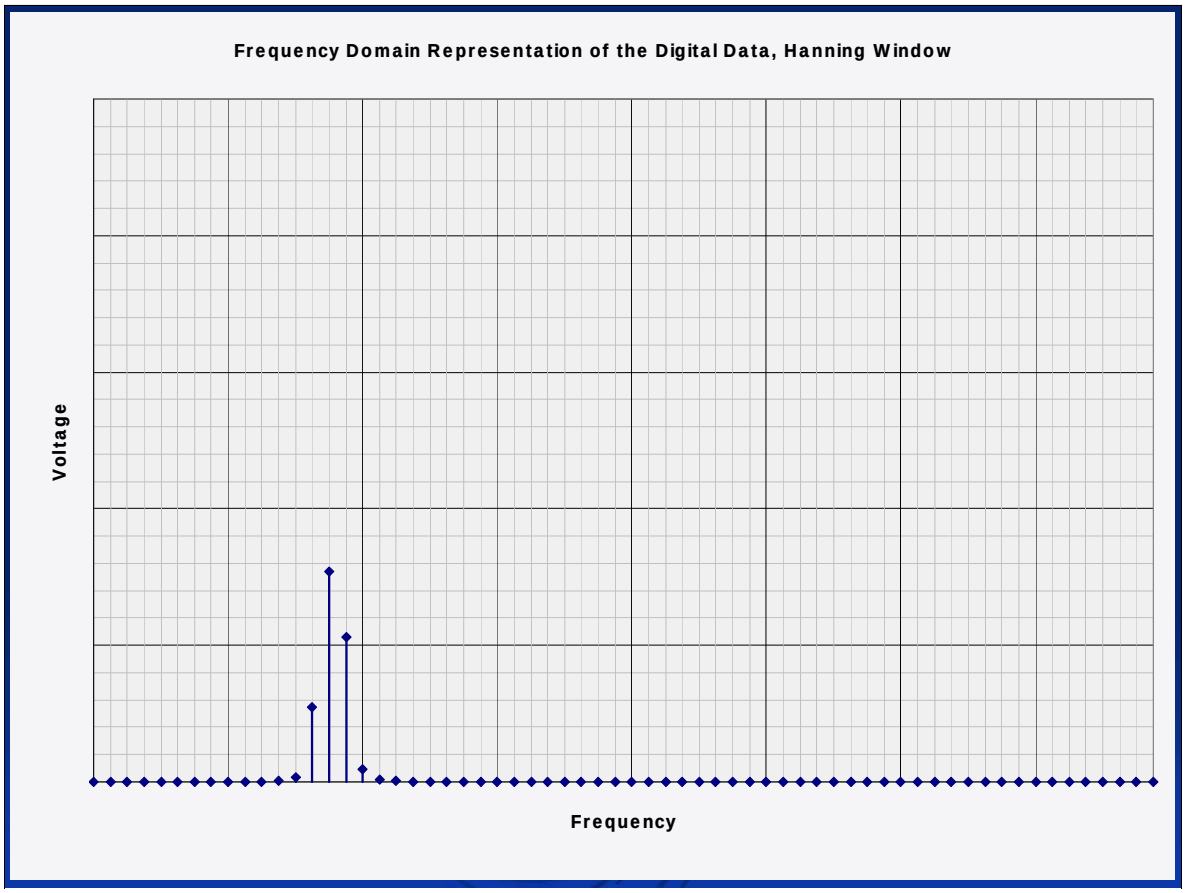
There are **many** other windows. The determination of which window to use for what application is much more a form of art than science. I think there are also a few tea leaves involved.



Here's the sine wave FFT block after the window has been applied (the sine wave's data points are multiplied by the window's data points, one by one).

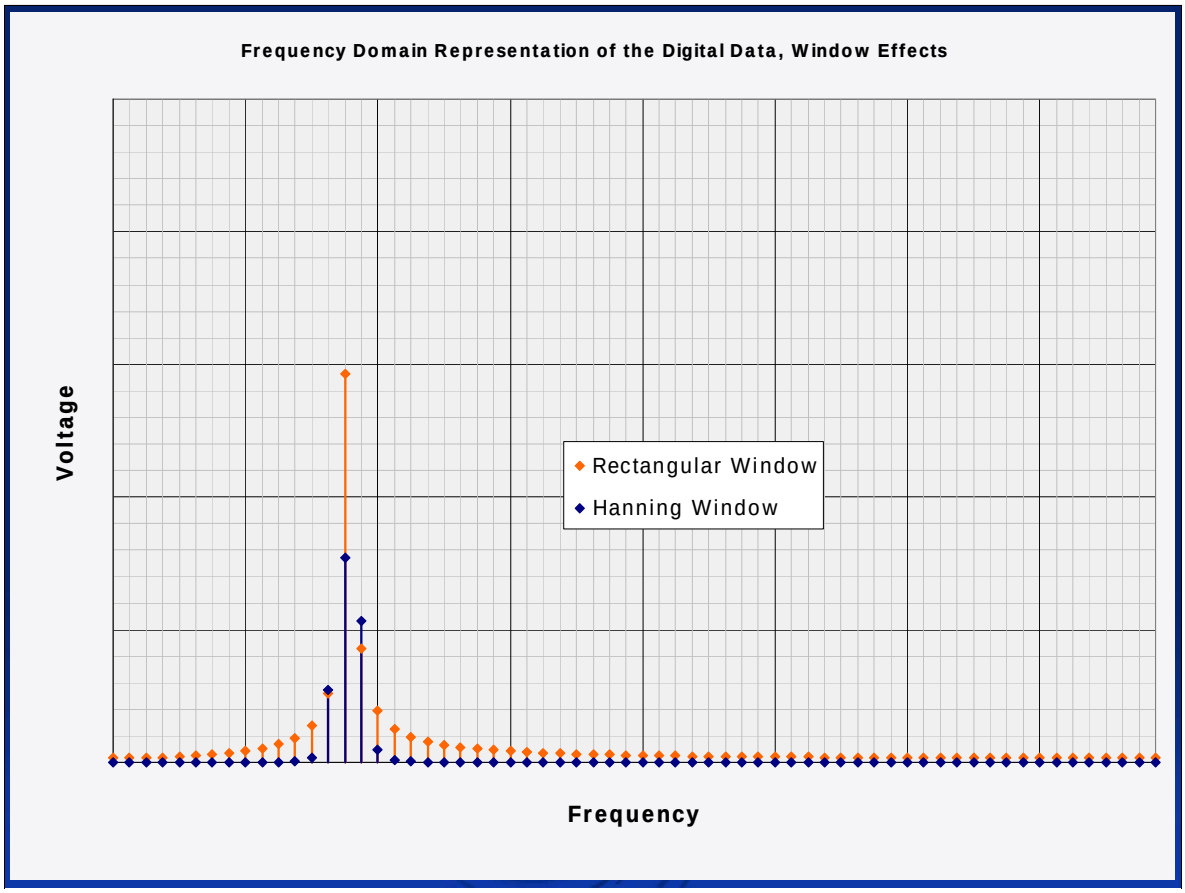


Here's the FFT output of the sine wave with no "windowing" performed – which is known as a *rectangular* window, because apparently it just had to have a name.



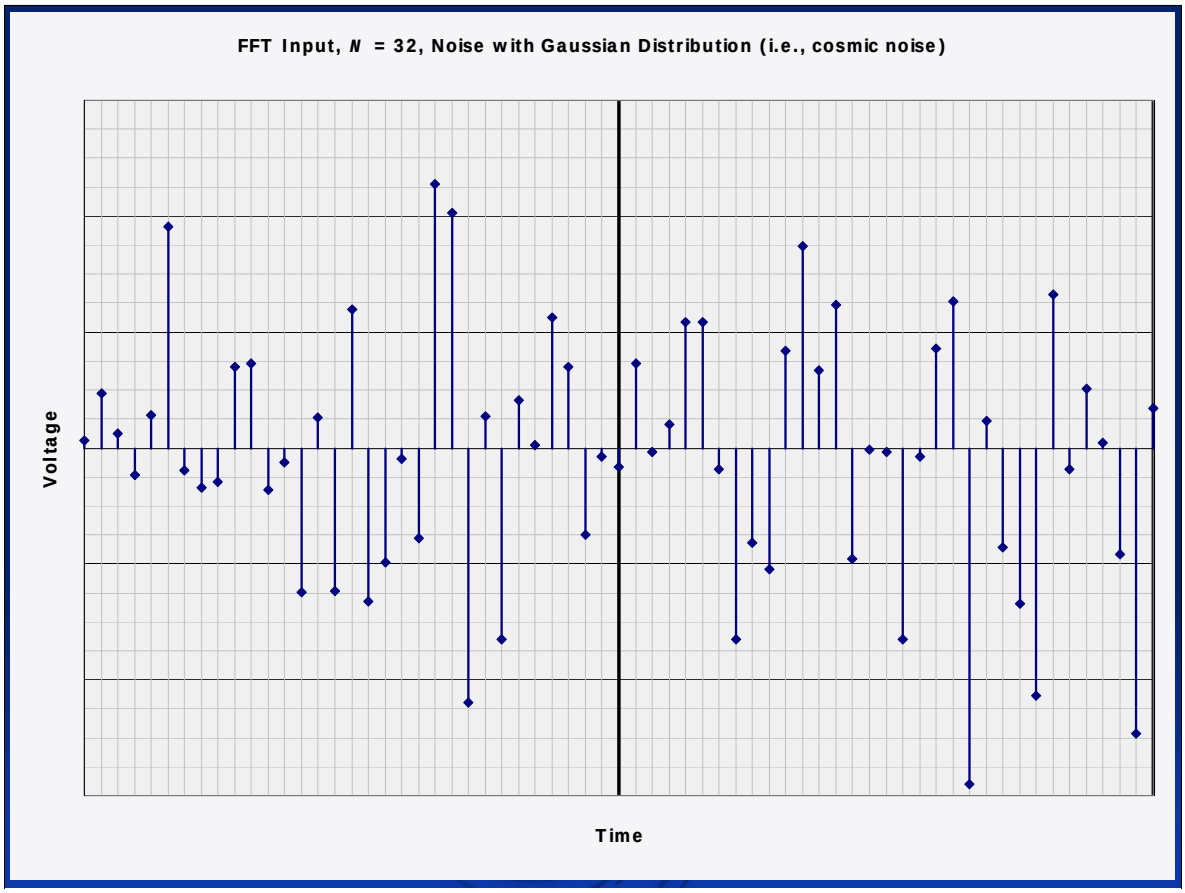
Here's the FFT output of the windowed sine wave. The overall signal level is lower due to the fact that the window attenuates some of the signal.

The important part, though, is that the window changes the shape of the peak and the level of the sidebands.



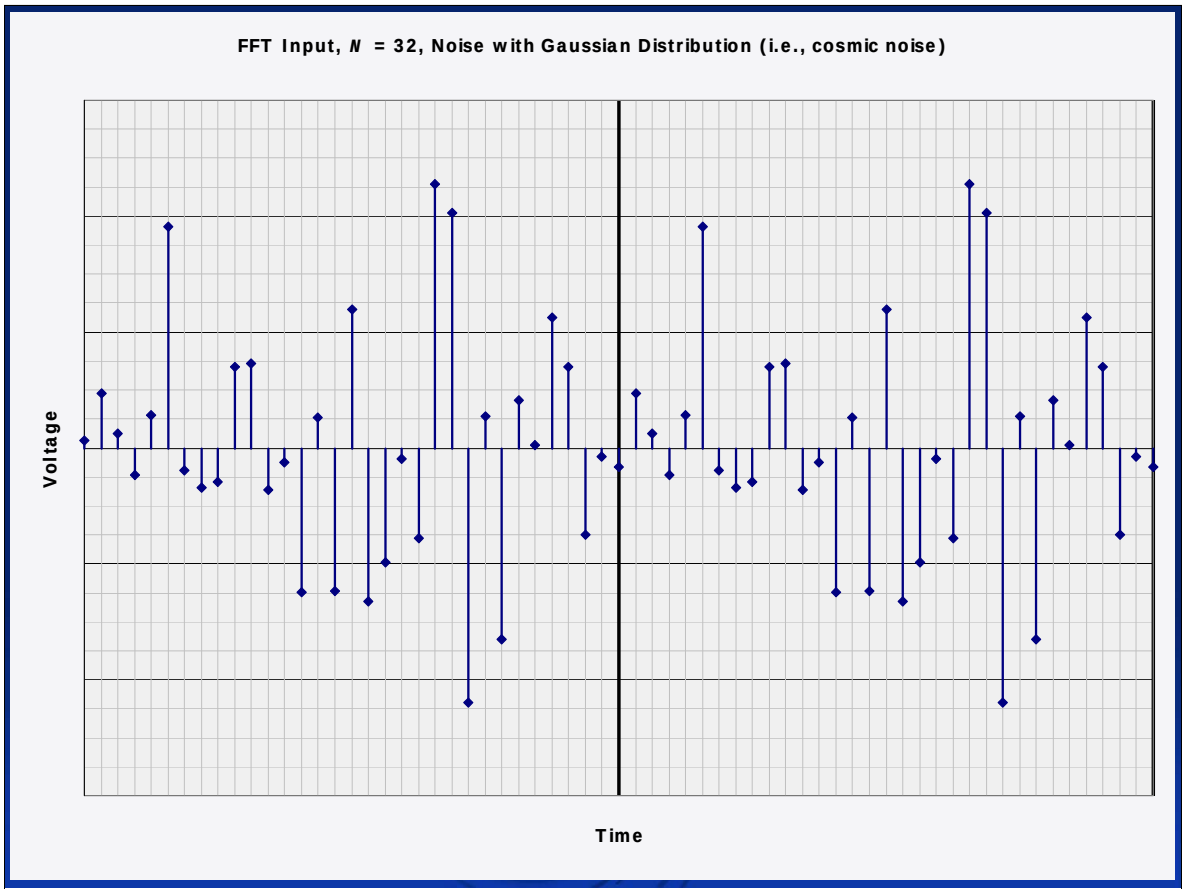
Here's a comparison of the rectangular window (Orange) with the Hanning window (blue), both applied to the original sine wave time series data.

In this particular case, the width of the peak stays about the same – but the sidebands have gone way down with the Hanning window – more so than can be accounted for by the window's attenuation alone.

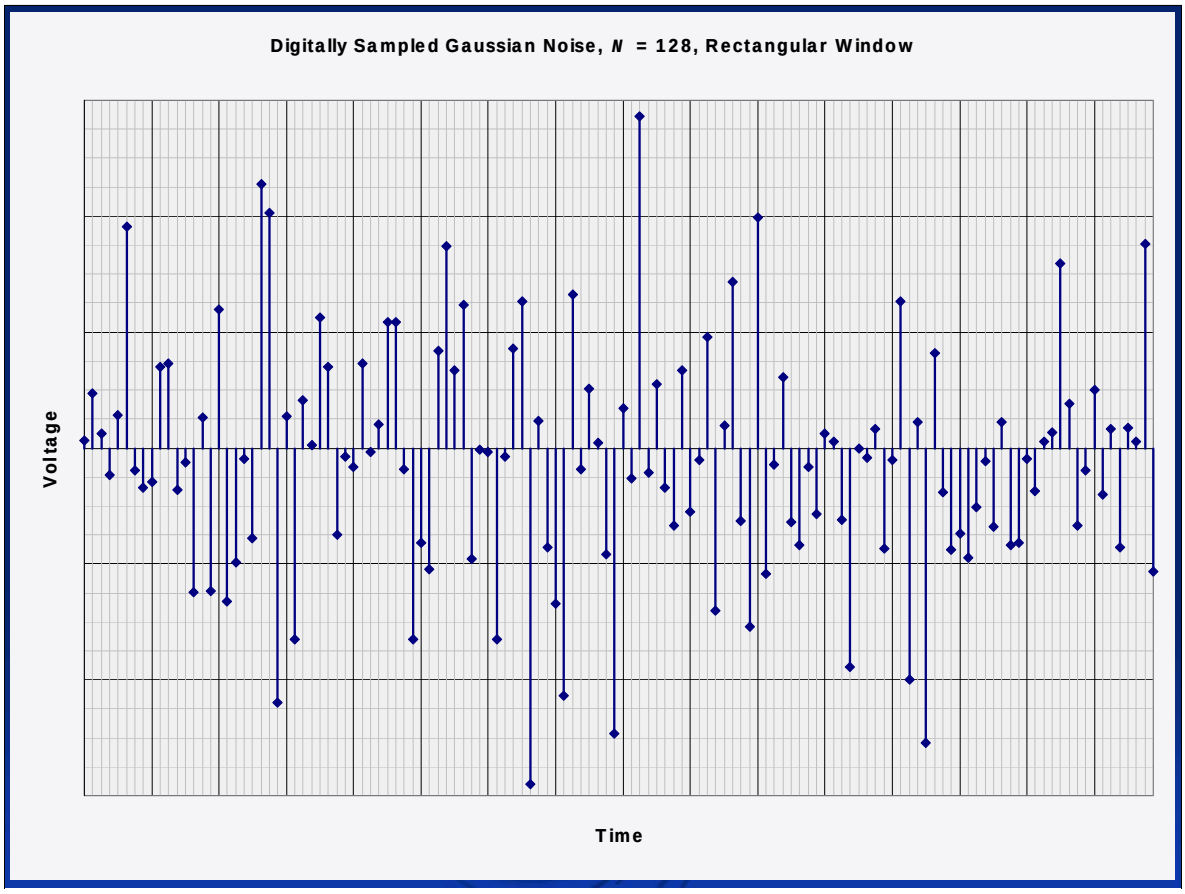


But what about noise? We don't receive nice smooth sine waves from cosmic radio sources. Our "signals" of interest are themselves noise.

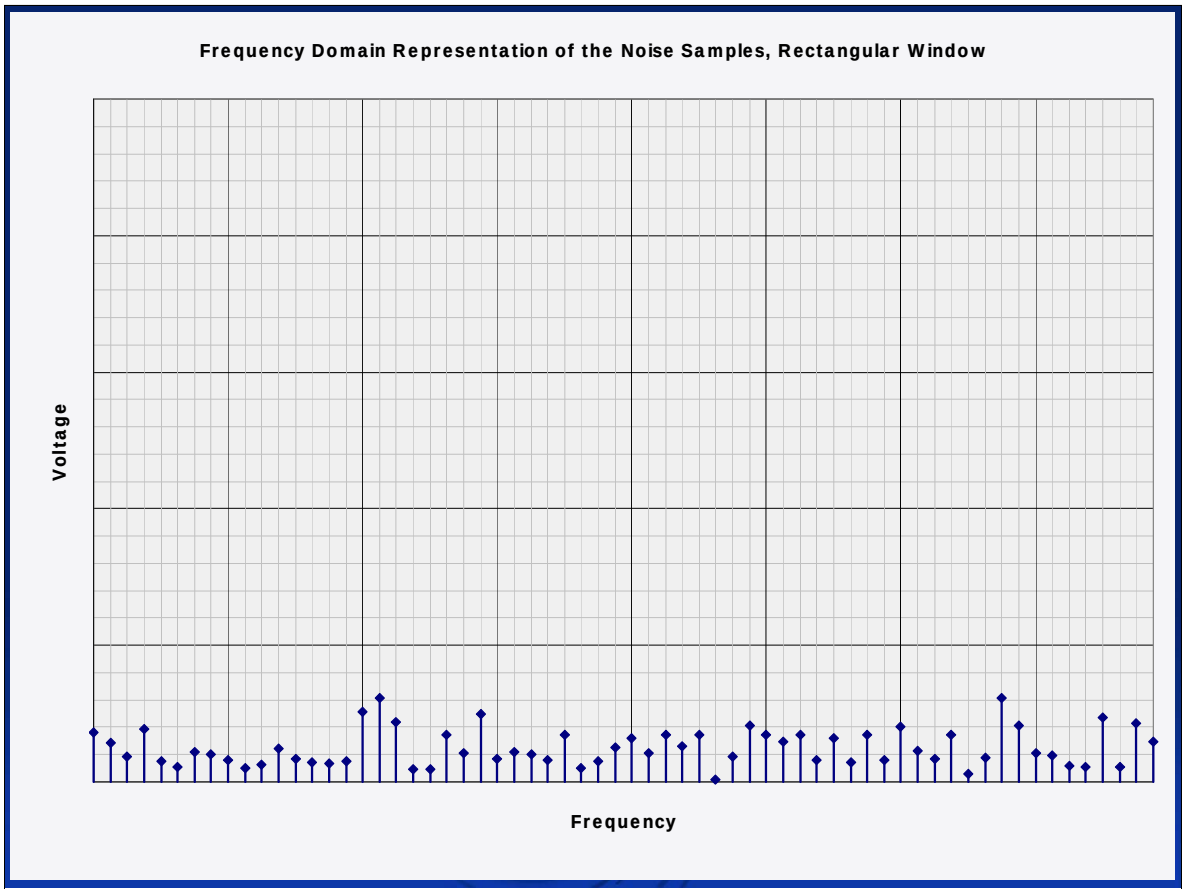
What happens when we repeat the first 32 samples ad infinitum? Will it create a discontinuity?



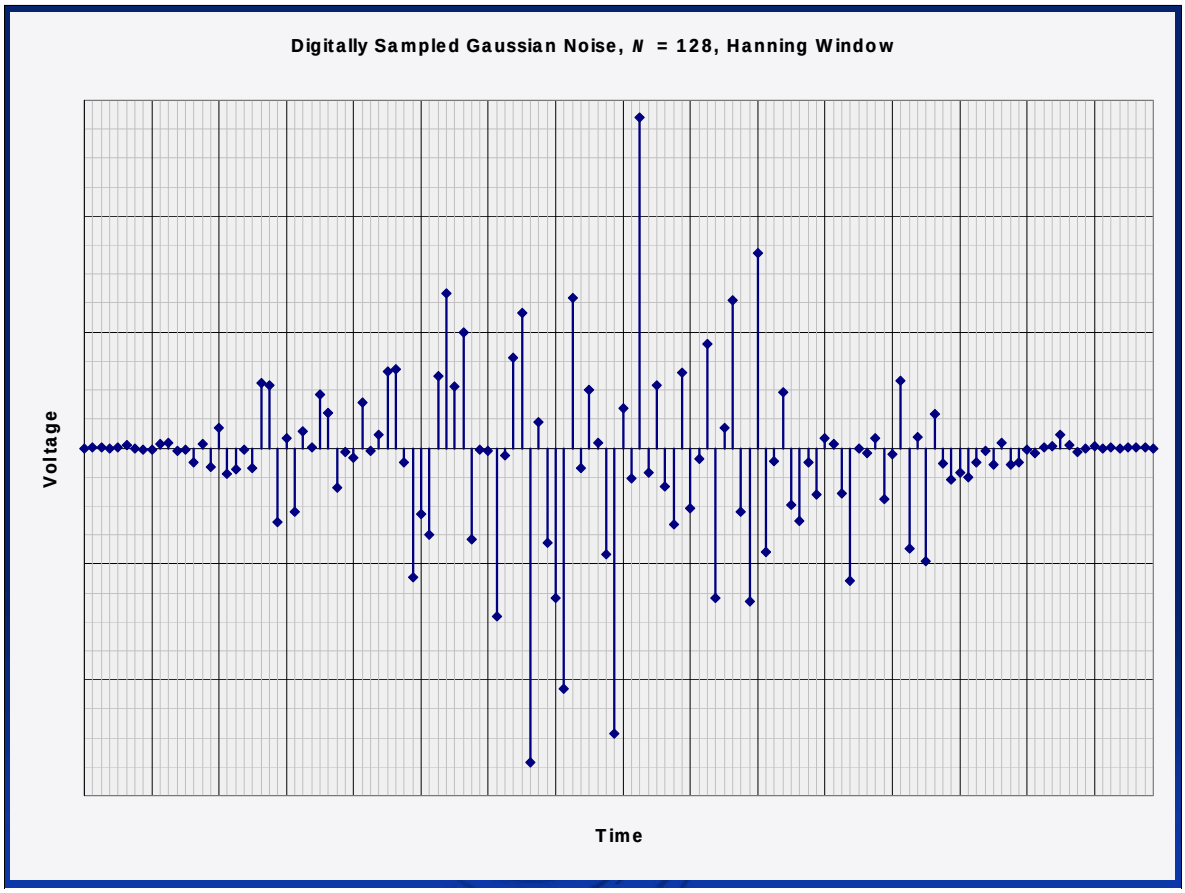
Where is the discontinuity? We can't tell, because the signal itself is noise, which is by definition random.



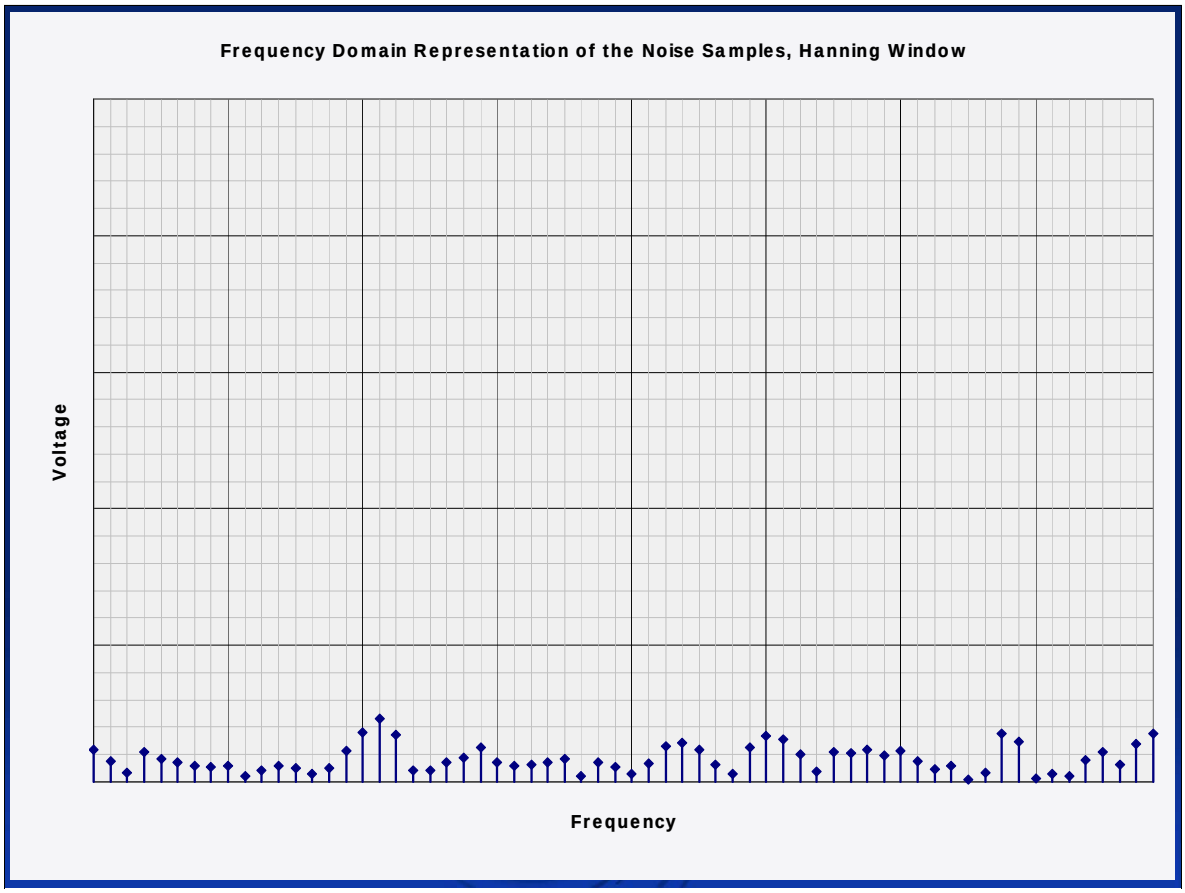
Cosmic radio noise has a Gaussian distribution. Here's a 128 point FFT block of Gaussian noise.



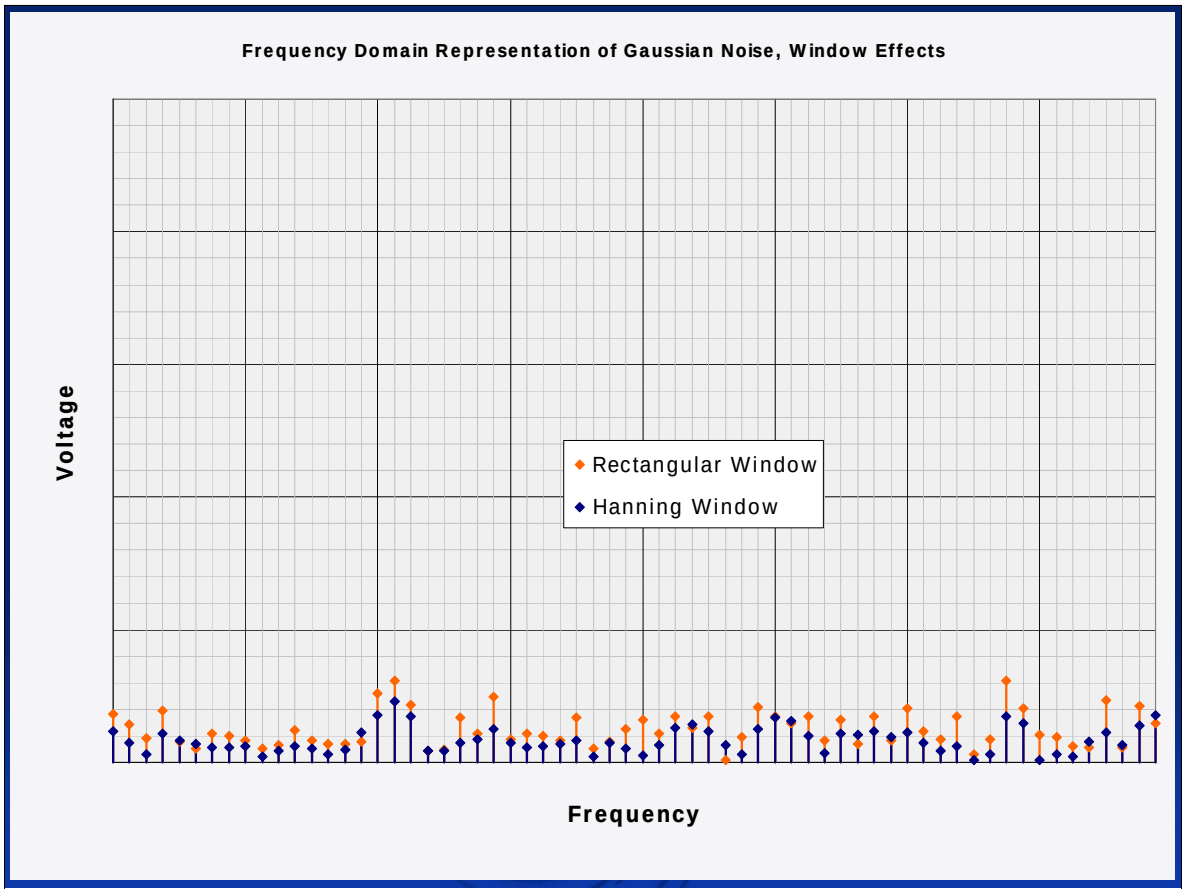
Here's the FFT output.



Here's the same noise after application of the Hanning window.



And here's the FFT output.

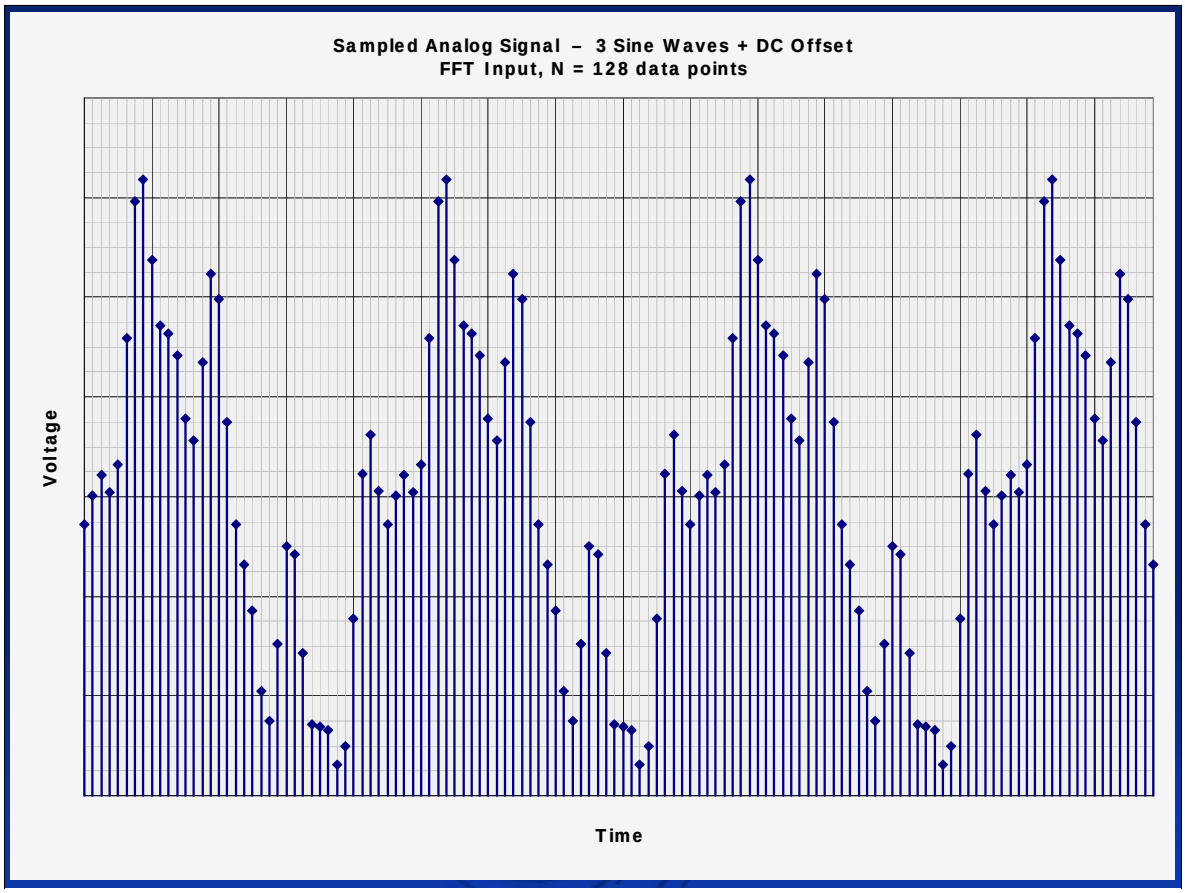


Here are both FFT outputs for the noise input. The windowed version is slightly lower in amplitude as one would expect. Other than that, there is no meaningful difference – because noise is, well, random.

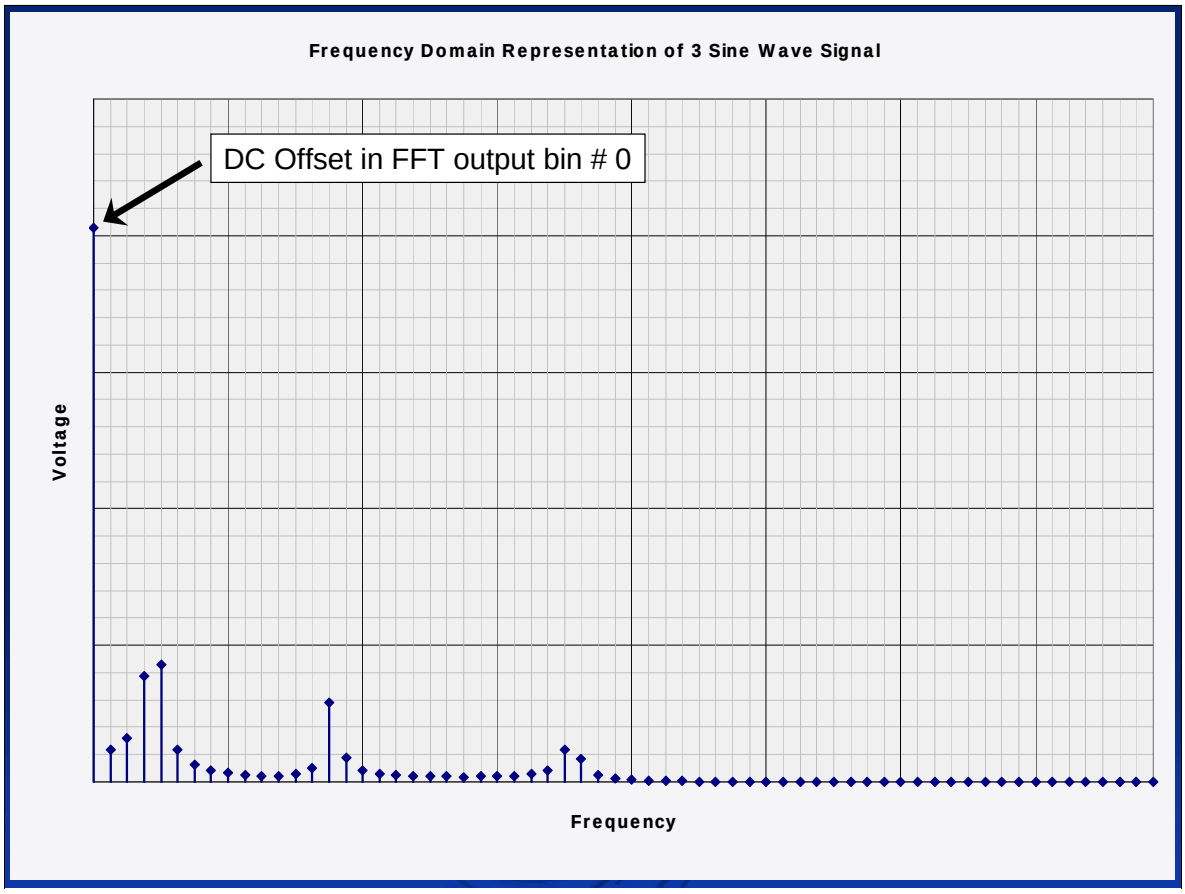
DC Offset

DC Offset

- Any DC offset in the input data is provided by the FFT in the first element of the output array – the 0 Hz frequency bin
- Ideally, DC offset does not change the amplitude of any other FFT output array element



Here’s the same old three-sine-wave signal... again...



ANd here's where the DC offset shows up. We couldn't see it on the other plots because it was off the chart, scale high. This plot has bee rescaled to allow the DC offset to show up as an actual data point.

Amplitude Calibration

Finally, the last section!

Amplitude Calibration

- Just like any other instrument, the amplitude of the FFT must be calibrated against a known standard
- Common to sqaure the voltage samples and divide by 50 ohms to convert voltage to power
- Other conversion factors can be applied to calibrate in dBm relative to a standard

References for Further Reading

- Witte, R., *Spectrum and Network Measurements*, Prentice-Hall (1991).
- Lyons, R., *Understanding Digital Signal Processing*, Addison Wesley Longman (1997).
- Frerking, M., *Digital Signal Processing in Communication Systems*, Van Nostrand (1994).
- *Who is Fourier?*, Transnational College of LEX (1995).

These books are outstanding; very highly recommended.

Witte, R., *Spectrum and Network Measurements*, Prentice-Hall (1991).

Lyons, R., *Understanding Digital Signal Processing*, Addison Wesley Longman (1997).

Frerking, M., *Digital Signal Processing in Communication Systems*, Van Nostrand (1994).

Who is Fourier?, Transnational College of LEX (1995).