Spectrum by Similarity

Craig Stuart Sapp <craig@ccrma.stanford.edu>
Peabody Conservatory
14 November 2002

This *Mathematica* Notebook demonstrates the basic principle used to calculate a spectrum from a signal. A spectrum is a plot of sinusoid frequencies on the x-axis versus the amplitude of that sinusoid displayed on the y-axis. This notebook demonstrates how the amplitude of a sinusoid is calculated from any arbitrary signal. The basic principle is this: (1) multiply the signal by a sinusoid at the desired frequency along the x-axis, and (2) sum the discrete elements in this multiplied signal together to determine the amplitude of the sinusoid present in the original signal. This process is checking to see how *similar* the spectrum is to any given test sinusoid.

```
Needs["SCMTheory`"]
```

Part I: The Real World

First we define some variables to create a test signal. The Amp variable stores the list of amplitudes for each sinusoid component in the signal. Freq contains a list of the frequencies in the signal. Periods indicates how many fundamental periods (of 100 Hz in this case) are to be displayed in the plots and used in the calculations. Srate is the sampling rate of the signal.

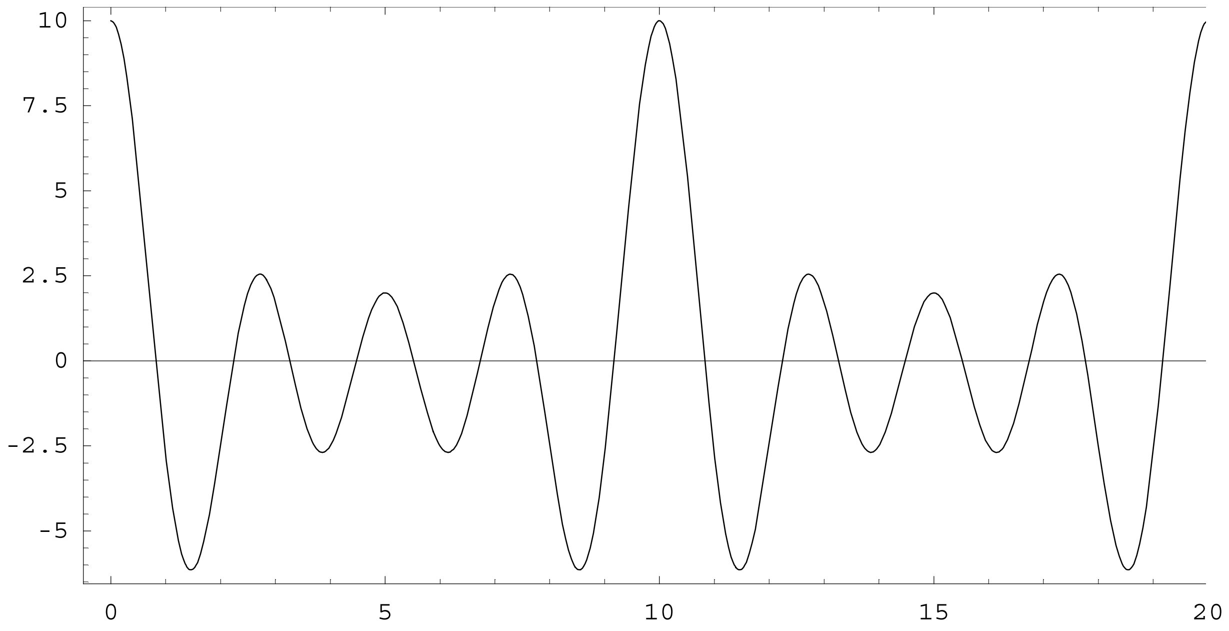
```
amp = {1,2,3, 4};
freq = {100, 200, 300, 400};
periods = 2;
srate = 1000;
```

The next cell defines a signal with the variable t representing time in seconds.

```
signal[t_] :=
 Apply[Plus, Table[amp[[i]] Cos[2 Pi freq[[i]] t], {i, 1, Length[freq]}]]
```

Here is a plot of signal[t] where the horizontal axis is in milliseconds:

```
continuous =
  Plot[signal[t/srate], {t, 0, 20}, Frame→True, AspectRatio→1/2];
```



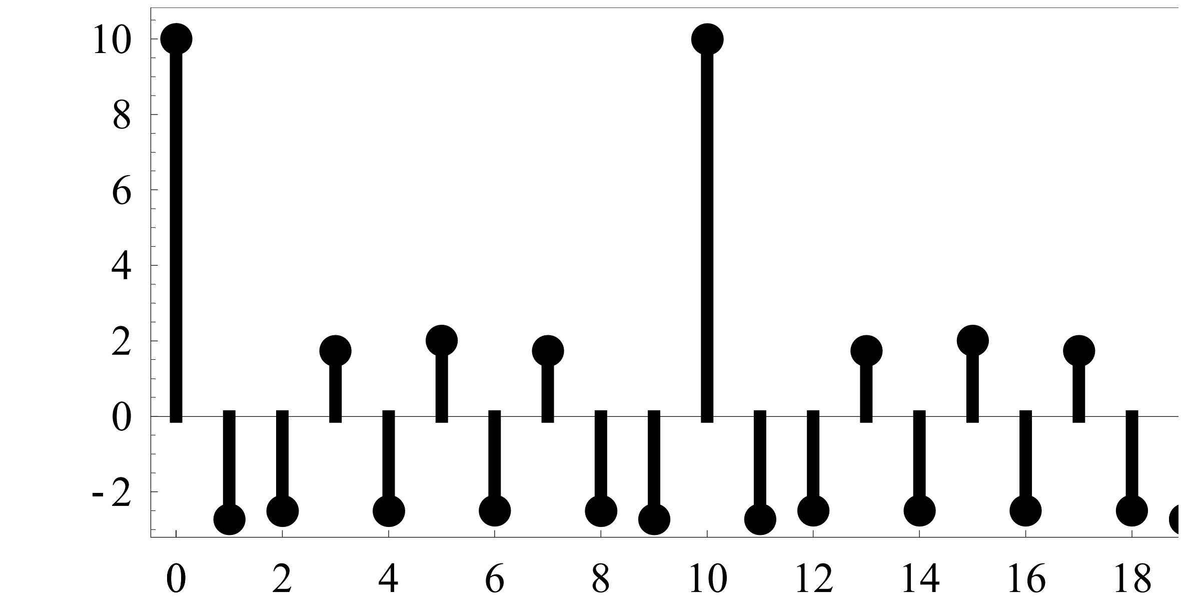
The previous picture is the continuous picture of the signal, but if a computer is processing this sound, it has to do it in discrete steps. The following command samples the signal at the sampling period (which is 1 millisecond in this case).

```
sampsignal= SampleFunction[signal[t], {t, 0, 19/srate, 1.0/srate}]
```

```
{10,-2.73607,-2.5,1.73607,-2.5,2.,-2.5,1.73607,-2.5,-2.73607,
 10.,-2.73607,-2.5,1.73607,-2.5,2.,-2.5,1.73607,-2.5,-2.73607}
```

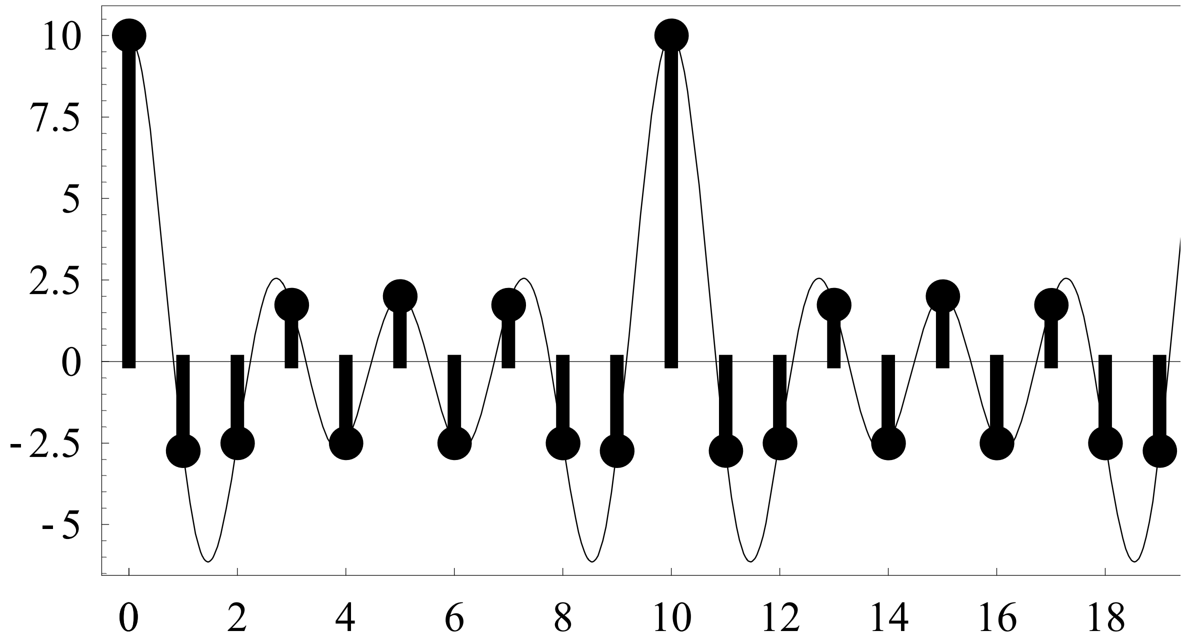
The following plot shows the discrete samples just calculated in the above cell:

```
discrete = SeqPlot[sampsignal, AspectRatio->1/2, Frame→True];
```



The following plot demonstrates that the sampled values really are derived from the continuous signal. Note that the samples line up with the original continuous function.

```
Show[ discrete, continuous];
```



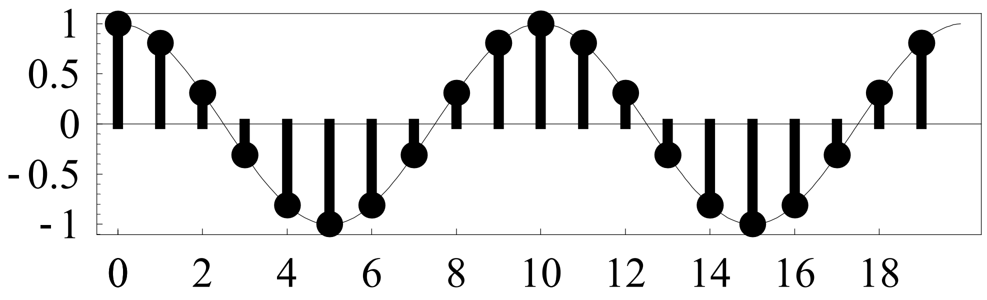
Now that samples from the test signal have been generated, let's make a table of the individual sinusoids which we know are present in the signal. The following cell makes a list of the samples for each sinusoid present in the signal.

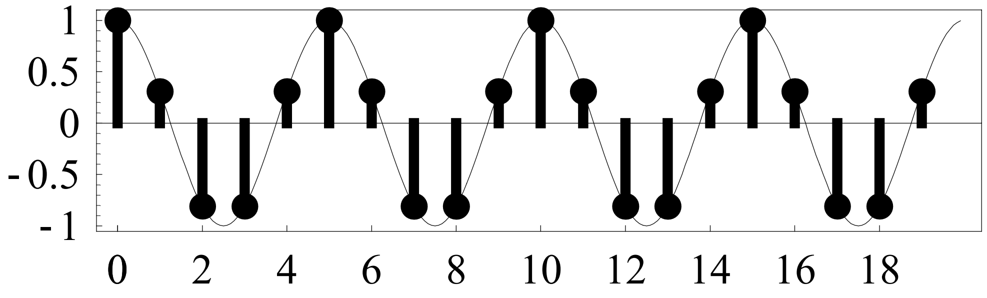
```
testsines =Table[ SampleFunction[Cos[2 Pi freq[[f]] t],
   {t, 0, 19/srate, 1.0/srate}], {f, 1, Length[freq]}]
```

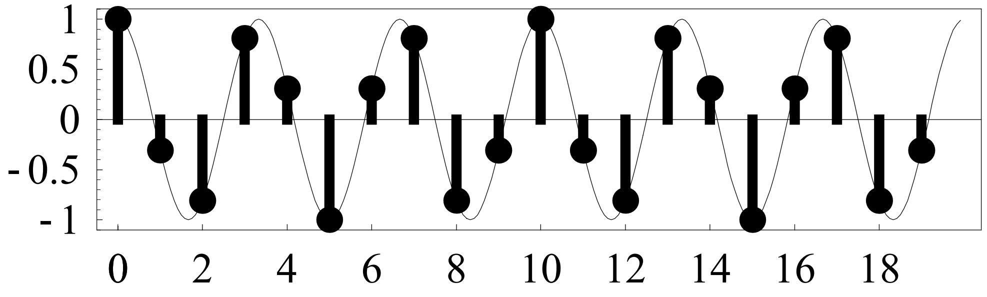
```
{{1,0.809017,0.309017,-0.309017,-0.809017,-1.,-0.809017,
  -0.309017,0.309017,0.809017,1.,0.809017,0.309017,-0.309017,
  -0.809017,-1.,-0.809017,-0.309017,0.309017,0.809017},
 {1,0.309017,-0.809017,-0.809017,0.309017,1.,0.309017,-0.809017,
  -0.809017,0.309017,1.,0.309017,-0.809017,-0.809017,
  0.309017,1.,0.309017,-0.809017,-0.809017,0.309017},
 {1,-0.309017,-0.809017,0.809017,0.309017,-1.,0.309017,0.809017,
  -0.809017,-0.309017,1.,-0.309017,-0.809017,0.809017,
  0.309017,-1.,0.309017,0.809017,-0.809017,-0.309017},
 {1,-0.809017,0.309017,0.309017,-0.809017,1.,-0.809017,
  0.309017,0.309017,-0.809017,1.,-0.809017,0.309017,0.309017,
  -0.809017,1.,-0.809017,0.309017,0.309017,-0.809017}}
```

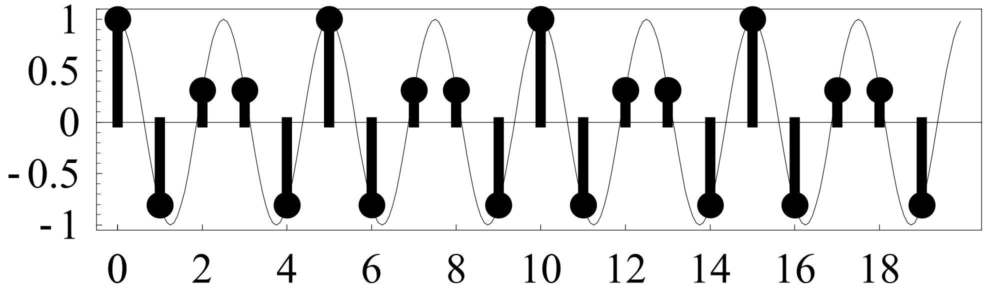
Here is a plot of the individual frequencies. Note that all of the amplitudes are one. Our job is to figure out what were the original amplitudes of each sinusoid in the original signal.

```
Map[SeqPlot[#, Frame→True, AspectRatio→1/4,
    Interpolate→True, InterpStyle→Thickness[0.001]]&, testsines];
```









In order to determine the amplitudes of the sinusoids in the original signal, you must do two steps (1) mulitply the signal by each test sinusoid, and (2) add up all of the elements in the resulting signal and then divied by the normalization factor. These sets of number will be the original amplitudes of the sinusoids that we started out adding to create the original signal.

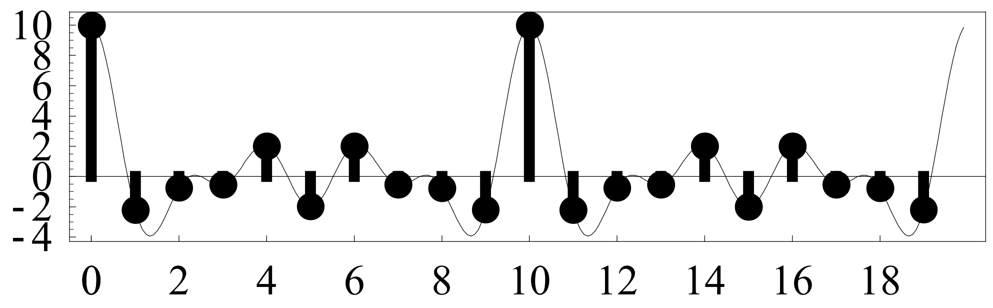
Below is the first step which involves multiplying the original signal by each of the test sinusoids:

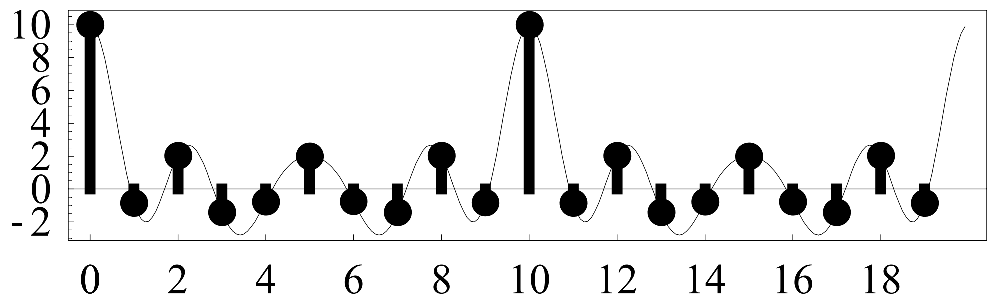
```
similar = Map[(sampsignal * #)&, testsines]
```

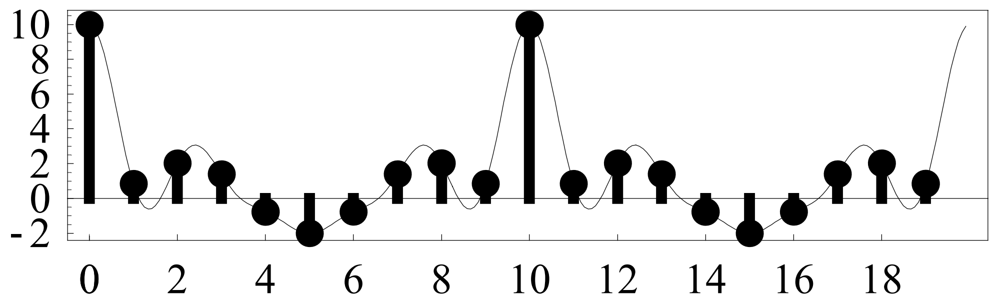
```
{{10,-2.21353,-0.772542,-0.536475,2.02254,-2.,
  2.02254,-0.536475,-0.772542,-2.21353,10.,-2.21353,-0.772542,
  -0.536475,2.02254,-2.,2.02254,-0.536475,-0.772542,-2.21353},
 {10,-0.845492,2.02254,-1.40451,-0.772542,2.,-0.772542,
  -1.40451,2.02254,-0.845492,10.,-0.845492,2.02254,-1.40451,
  -0.772542,2.,-0.772542,-1.40451,2.02254,-0.845492},
 {10,0.845492,2.02254,1.40451,-0.772542,-2.,-0.772542,
  1.40451,2.02254,0.845492,10.,0.845492,2.02254,1.40451,
  -0.772542,-2.,-0.772542,1.40451,2.02254,0.845492},
 {10,2.21353,-0.772542,0.536475,2.02254,2.,2.02254,0.536475,
  -0.772542,2.21353,10.,2.21353,-0.772542,0.536475,
  2.02254,2.,2.02254,0.536475,-0.772542,2.21353}}
```

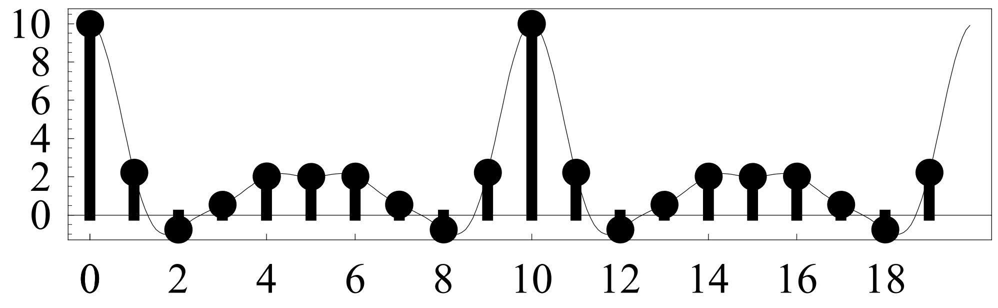
Here is a plot of the data generated above. These plots do not have any physical meaning, but are a step on the way to calculating the amplitudes of the sinewaves in the original signal:

```
Map[SeqPlot[#, Frame→True, AspectRatio→1/4,
    Interpolate→True, InterpStyle→Thickness[0.001]]&, similar];
```









Next, each of these multiplied signals need to be added, sample by sample:

```
Map[Apply[Plus, #]&, similar]
```

```
{10.,20.,30.,40.}
```

Finally, normalize the values:

```
% / 10
```

```
{1.,2.,3.,4.}
```

Now compare this result to the amplitudes of the sinusoids present in the original signal:

```
amp
```

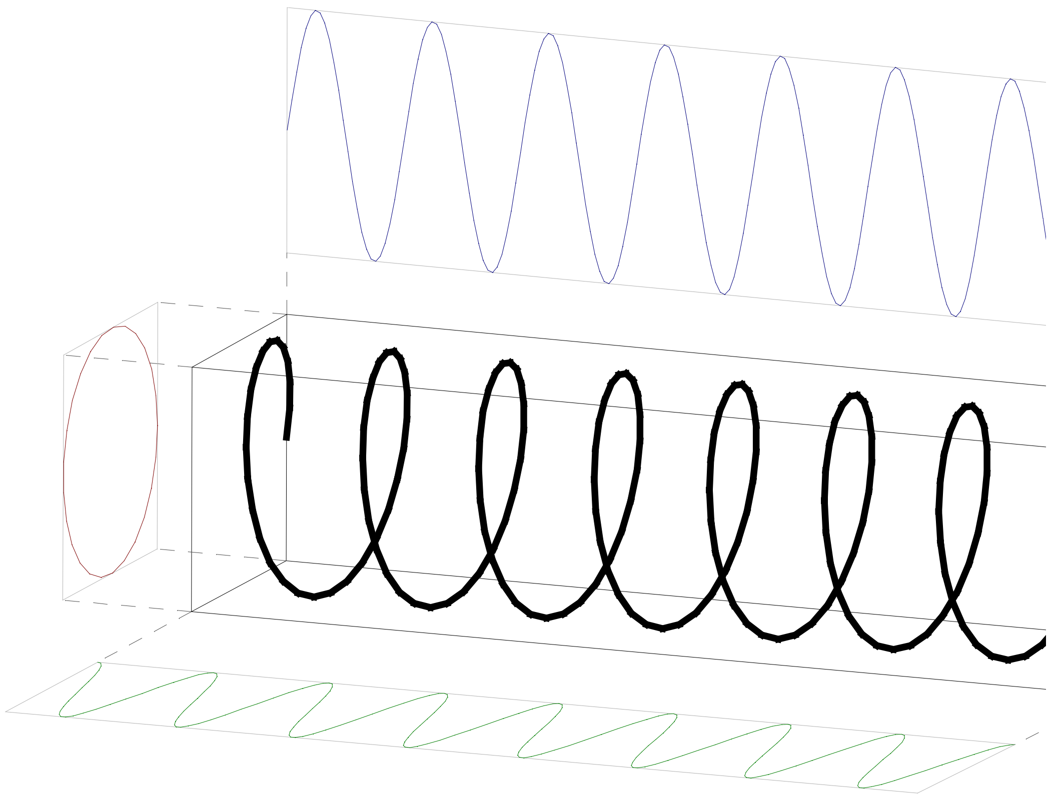
```
{1,2,3,4}
```

Part II: The Complex World

You saw in the previous section that the amplitudes of the original sinusoids can be extracted by multiplying the spectrum by the individual sinusoids that we want to find the amplitude of in the signal. In practice, we will need a more comprehensive sinusoid, called the complex sinusoid. This sinusoid can handle any phase shift of the real sinusoids found in the signal and will be demonstracted in the next section.

A complex sinusoid is a combination of a sinewave and a cosinewave which are at right angles to each other. Here is a plot of a complex sinusoid:

```
ComplexSinusoidPlot[0, 8, Amplitude->1];
```



The horizontal axis represent the real numbers, while the vertical axis represents the Imaginary numbers. The origin of the complex sinusoid is on the left, and you can see that the green projection of the sinusoid creates a cosine. The blue projection of the complex sinusoid (onto the imaginary axis) creates a sinewave. Thus the mathematical equation for the complex sinusoid is: csin(x) = cos(x) + i sin(x), where i is the square root of negative one, and csin is the complex sinusoid.

The complex sinusoid itself can be represented as an imaginary power of the number e (approximately equal to 2.71828): e^(i x). Here is a demonstration that there are two equal ways of calculating the value of a complex sinusoid (Euler's Identity):

```
xx = 23.8;
ans1 = Cos[x] + I Sin[xx]
ans2 = E^(I xx)
diff = ans1-ans2
```

```
0.235813-0.971798 ⅈ
```

```
0.235813-0.971798 ⅈ
```

```
0.+0. ⅈ
```

Since the difference between ans1 and ans2 is zero, the methods are equal. Try it for any other value of x to verify that the calculations are still equal.

Now let us use the complex sinusoids to measure the amplitudes of the original sinusoids used to make the signal. Here is the original signal:

```
sampsignal
```

```
{10,-2.73607,-2.5,1.73607,-2.5,2.,-2.5,1.73607,-2.5,-2.73607,
 10.,-2.73607,-2.5,1.73607,-2.5,2.,-2.5,1.73607,-2.5,-2.73607}
```

Analygous to part I, create a set of test sinusoids which will be used to examine the signal:

```
testsinescomplex =Table[ SampleFunction[E^(2 Pi I freq[[f]] t),
    {t, 0, 19/srate, 1.0/srate}], {f, 1, Length[freq]}]  // Chop
```

```
{{1,0.809017+0.587785 i,0.309017+0.951057 i,
  -0.309017+0.951057 i,-0.809017+0.587785 i,-1.,
  -0.809017-0.587785 i,-0.309017-0.951057 i,0.309017-0.951057 i,
  0.809017-0.587785 i,1.,0.809017+0.587785 i,0.309017+0.951057 i,
  -0.309017+0.951057 i,-0.809017+0.587785 i,-1.,-0.809017-0.587785 i,
  -0.309017-0.951057 i,0.309017-0.951057 i,0.809017-0.587785 i},
 {1,0.309017+0.951057 i,-0.809017+0.587785 i,-0.809017-0.587785 i,
  0.309017-0.951057 i,1.,0.309017+0.951057 i,
  -0.809017+0.587785 i,-0.809017-0.587785 i,0.309017-0.951057 i,
  1.,0.309017+0.951057 i,-0.809017+0.587785 i,
  -0.809017-0.587785 i,0.309017-0.951057 i,1.,0.309017+0.951057 i,
  -0.809017+0.587785 i,-0.809017-0.587785 i,0.309017-0.951057 i},
 {1,-0.309017+0.951057 i,-0.809017-0.587785 i,0.809017-0.587785 i,
  0.309017+0.951057 i,-1.,0.309017-0.951057 i,
  0.809017+0.587785 i,-0.809017+0.587785 i,-0.309017-0.951057 i,
  1.,-0.309017+0.951057 i,-0.809017-0.587785 i,
  0.809017-0.587785 i,0.309017+0.951057 i,-1.,0.309017-0.951057 i,
  0.809017+0.587785 i,-0.809017+0.587785 i,-0.309017-0.951057 i},
 {1,-0.809017+0.587785 i,0.309017-0.951057 i,0.309017+0.951057 i,
  -0.809017-0.587785 i,1.,-0.809017+0.587785 i,
  0.309017-0.951057 i,0.309017+0.951057 i,-0.809017-0.587785 i,1.,
  -0.809017+0.587785 i,0.309017-0.951057 i,0.309017+0.951057 i,
  -0.809017-0.587785 i,1.,-0.809017+0.587785 i,
  0.309017-0.951057 i,0.309017+0.951057 i,-0.809017-0.587785 i}}
```

```
complexsimilar = Map[(sampsignal * #)&, testsinescomplex]
```

```
{{10,-2.21353-1.60822 ⅈ,-0.772542-2.37764 ⅈ,
  -0.536475+1.6511 ⅈ,2.02254-1.46946 ⅈ,-2.,2.02254+1.46946 ⅈ,
  -0.536475-1.6511 ⅈ,-0.772542+2.37764 ⅈ,-2.21353+1.60822 ⅈ,
  10.,-2.21353-1.60822 ⅈ,-0.772542-2.37764 ⅈ,
  -0.536475+1.6511 ⅈ,2.02254-1.46946 ⅈ,-2.,2.02254+1.46946 ⅈ,
  -0.536475-1.6511 ⅈ,-0.772542+2.37764 ⅈ,-2.21353+1.60822 ⅈ},
 {10,-0.845492-2.60216 ⅈ,2.02254-1.46946 ⅈ,-1.40451-1.02044 ⅈ,
  -0.772542+2.37764 ⅈ,2.,-0.772542-2.37764 ⅈ,
  -1.40451+1.02044 ⅈ,2.02254+1.46946 ⅈ,-0.845492+2.60216 ⅈ,
  10.,-0.845492-2.60216 ⅈ,2.02254-1.46946 ⅈ,-1.40451-1.02044 ⅈ,
  -0.772542+2.37764 ⅈ,2.,-0.772542-2.37764 ⅈ,
  -1.40451+1.02044 ⅈ,2.02254+1.46946 ⅈ,-0.845492+2.60216 ⅈ},
 {10,0.845492-2.60216 ⅈ,2.02254+1.46946 ⅈ,1.40451-1.02044 ⅈ,
  -0.772542-2.37764 ⅈ,-2.,-0.772542+2.37764 ⅈ,
  1.40451+1.02044 ⅈ,2.02254-1.46946 ⅈ,0.845492+2.60216 ⅈ,
  10.,0.845492-2.60216 ⅈ,2.02254+1.46946 ⅈ,1.40451-1.02044 ⅈ,
  -0.772542-2.37764 ⅈ,-2.,-0.772542+2.37764 ⅈ,1.40451+1.02044 ⅈ,
  2.02254-1.46946 ⅈ,0.845492+2.60216 ⅈ},{10,2.21353-1.60822 ⅈ,
  -0.772542+2.37764 ⅈ,0.536475+1.6511 ⅈ,2.02254+1.46946 ⅈ,
  2.,2.02254-1.46946 ⅈ,0.536475-1.6511 ⅈ,-0.772542-2.37764 ⅈ,
  2.21353+1.60822 ⅈ,10.,2.21353-1.60822 ⅈ,-0.772542+2.37764 ⅈ,
  0.536475+1.6511 ⅈ,2.02254+1.46946 ⅈ,2.,2.02254-1.46946 ⅈ,
  0.536475-1.6511 ⅈ,-0.772542-2.37764 ⅈ,2.21353+1.60822 ⅈ}}
```

```
Map[Apply[Plus, #]&, complexsimilar]  // Chop
```

```
{10.,20.,30.,40.}
```

Finally, normalize:

```
% / 10
```

```
{1.,2.,3.,4.}
```

Now compare this to the amplitudes of the sinusoids present in the original signal:

```
amp
```

```
{1,2,3,4}
```

These amplitudes also exactly match the amplitudes found by using only real sinusoids.

Part III: Phase and why the complex world is better than the real world

The complex sinusoid is much more accurate at finding the original sinusoid amplitudes. This section will demonstrate this fact. A problem occurs when you try to find the amplitude of a sinusoid component in a signal which has an unknown phase. In part I, the phase was exactly 0 (or 90 degrees if you are thinking of sinewaves). But what happens if the phase of each sinusoid is random? Real sinusoids cannot extract the full amplitude of the original sinusoid in the signal, and this amplitude will more than likely be underreported.

Lets test this theory by creating a signal which contains the same sineusoids as in the previous signal which also have the same amplitudes, but have arbitrary phases:

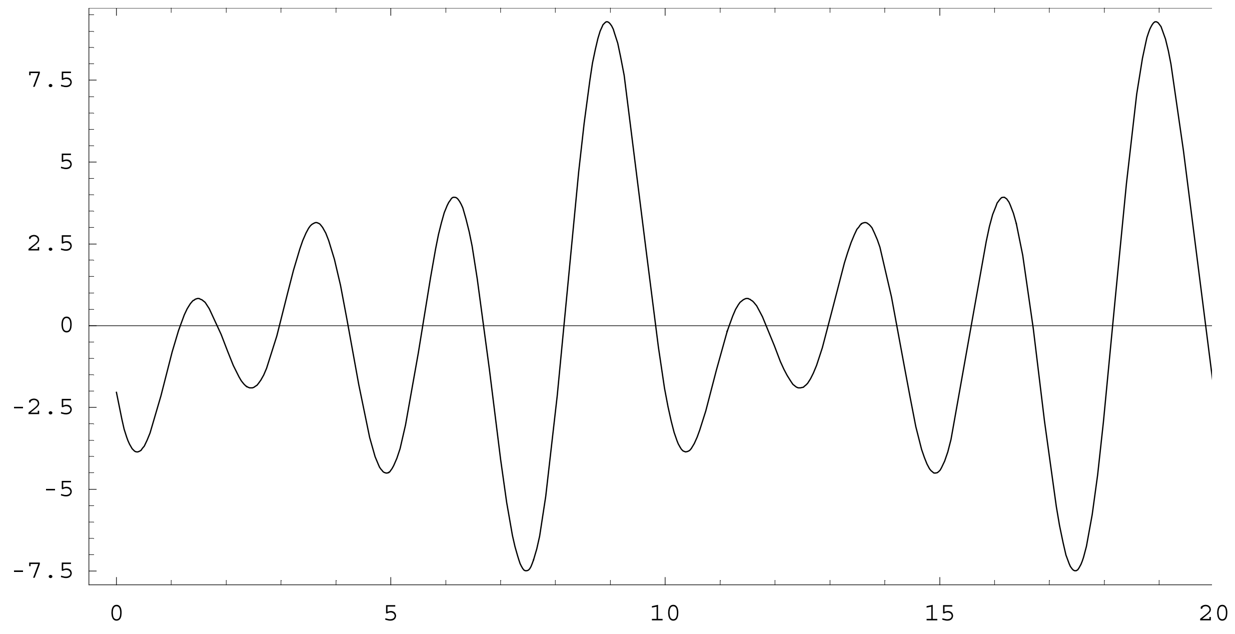
```
amp = {1,2,3, 4};
freq = {100, 200, 300, 400};
phase = {0.2, 1.2, 1.5, 3.0};
periods = 2;
srate = 1000;
```

The next cell defines a signal with the variable t representing time in seconds.

```
phasesignal[t_] := Apply[Plus, Table[
   amp[[i]] Cos[2 Pi freq[[i]] t+ phase[[i]] ], {i, 1, Length[freq]}]]
```

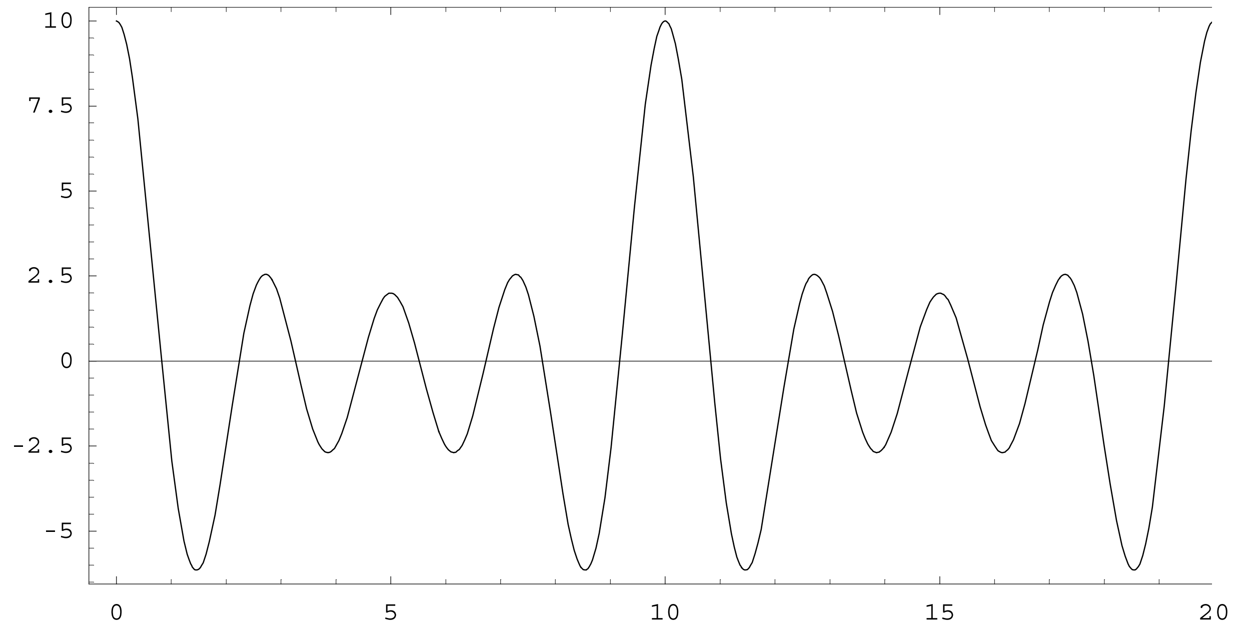
Here is a plot of signal[t] where the horizontal axis is in milliseconds:

```
phasecontinuous =
  Plot[phasesignal[t/srate], {t, 0, 20}, Frame→True, AspectRatio→1/2];
```



Notice that the above signal with different phases for each sinusoid looks different from the original signal:

```
continuous =
  Plot[signal[t/srate], {t, 0, 20}, Frame→True, AspectRatio→1/2];
```



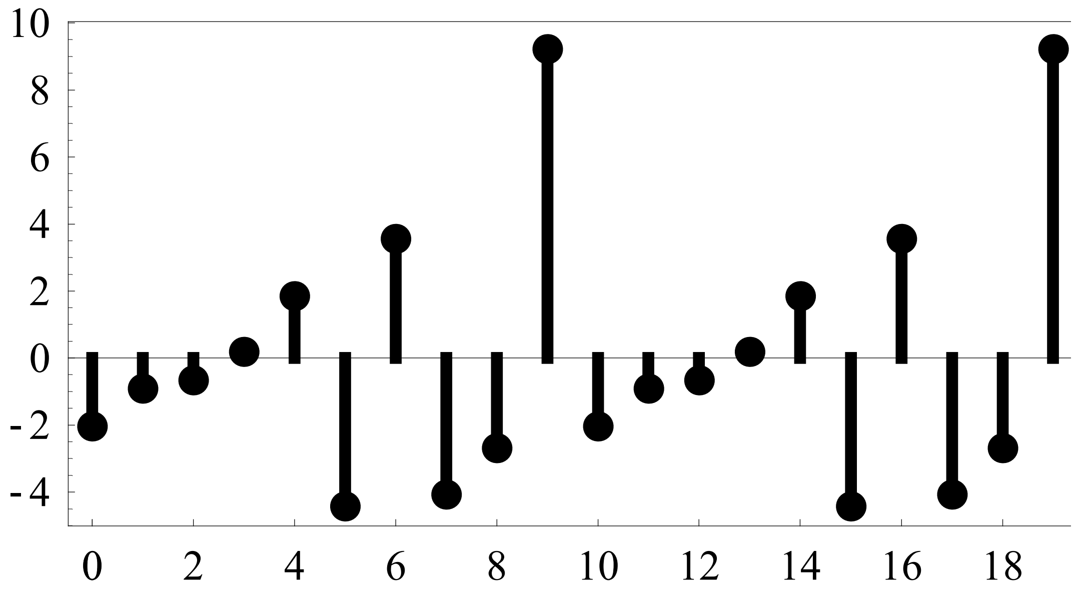
The previous picture is the continuous picture of the signal, but if a computer is processing this sound, it has to do it in discrete steps. The following command samples the signal at the sampling period (which is 1 millisecond in this case).

```
sampphasesignal=
 SampleFunction[phasesignal[t], {t, 0, 19/srate, 1.0/srate}]
```

```
{-2.04298,-0.912488,-0.667663,0.187638,1.84216,-4.42753,
 3.55848,-4.07,-2.69,9.22238,-2.04298,-0.912488,-0.667663,
 0.187638,1.84216,-4.42753,3.55848,-4.07,-2.69,9.22238}
```

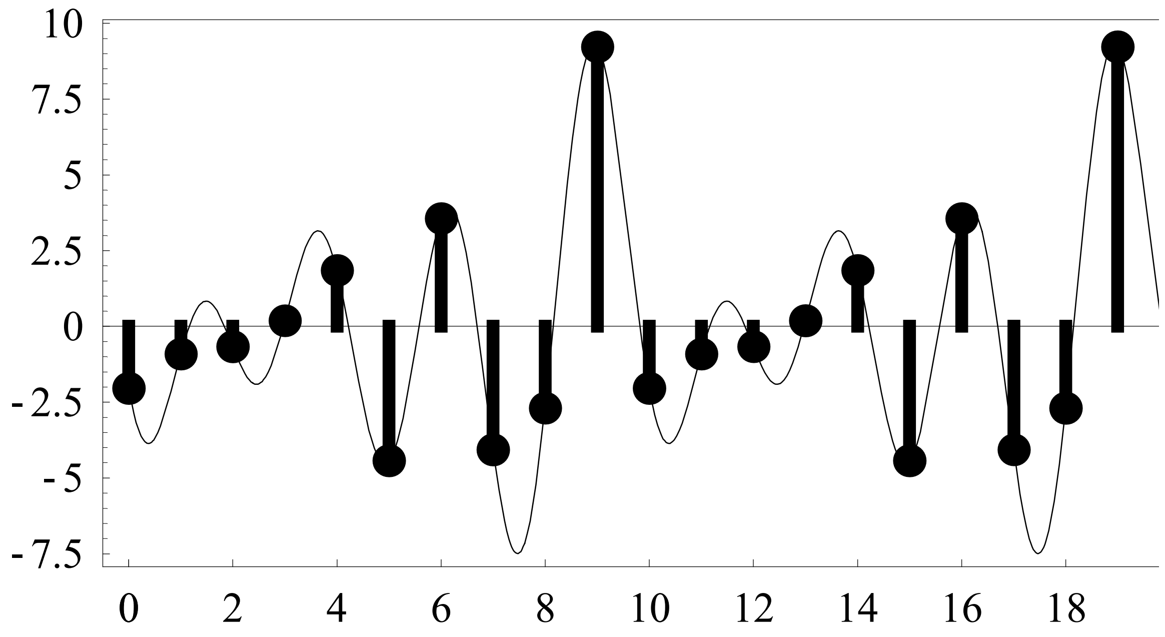
The following plot shows the discrete samples just calculated in the above cell:

```
phasediscrete = SeqPlot[sampphasesignal, AspectRatio->1/2, Frame→True];
```



The following plot demonstrates that the sampled values really are derived from the continuous signal. Note that the samples line up with the original continuous function.

```
Show[phasediscrete, phasecontinuous];
```



```
similar = Map[(sampphasesignal * #)&, testsines]
```

```
{{-2.04298,-0.738218,-0.206319,-0.0579835,-1.49034,4.42753,
  -2.87887,1.2577,-0.831255,7.46106,-2.04298,-0.738218,-0.206319,
  -0.0579835,-1.49034,4.42753,-2.87887,1.2577,-0.831255,7.46106},
 {-2.04298,-0.281974,0.540151,-0.151803,0.569258,-4.42753,
  1.09963,3.2927,2.17625,2.84987,-2.04298,-0.281974,0.540151,
  -0.151803,0.569258,-4.42753,1.09963,3.2927,2.17625,2.84987},
 {-2.04298,0.281974,0.540151,0.151803,0.569258,4.42753,1.09963,
  -3.2927,2.17625,-2.84987,-2.04298,0.281974,0.540151,
  0.151803,0.569258,4.42753,1.09963,-3.2927,2.17625,-2.84987},
 {-2.04298,0.738218,-0.206319,0.0579835,-1.49034,-4.42753,
  -2.87887,-1.2577,-0.831255,-7.46106,-2.04298,0.738218,-0.206319,
  0.0579835,-1.49034,-4.42753,-2.87887,-1.2577,-0.831255,-7.46106}}
```

```
Map[Apply[Plus, #]&, similar]
```

```
{9.80067,7.24716,2.12212,-39.5997}
```

Finally, normalize (divide) by the number of samples per fundamental frequency (10 samples from 100 Hz):

```
% / 10
```

```
{0.980067,0.724716,0.212212,-3.95997}
```

Now compare this to the amplitudes of the sinusoids present in the original signal:

```
amp
```

```
{1,2,3,4}
```

Notice that the amplitudes this time are not equal to the amplitudes of the sinusoids. Depending on the phase of the sinusoids, the measured amplitudes will range from the full amplitude to the negative full amplitude.

Measuring the correct original amplitude

The only way to extract the original amplitudes from the signal is to use complex sinusoids which resist the effects of a change in phase. This section demonstrates that you can measure the correct original amplitudes of the sinusoids from the original signal.

```
complexphasesimilar = Map[(sampphasesignal * #)&, testsinescomplex];
```

```
finalsim = Map[Apply[Plus, #]&, complexphasesimilar]  // Chop
```

```
{9.80067-1.98669 ⅈ,7.24716-18.6408 ⅈ,2.12212-29.9248 ⅈ,-39.5997-5.6448 ⅈ}
```

The amplitude is taken as the magnitude of the complex number, which is also the absolute value of the complex number:

```
finalsim // Abs
```

```
{10.,20.,30.,40.}
```

Finally, normalize (divide) by the number of samples per fundamental frequency (10 samples from 100 Hz):

```
% / 10
```

```
{1.,2.,3.,4.}
```

Now compare this to the amplitudes of the sinusoids present in the original signal:

```
amp
```

```
{1,2,3,4}
```

These amplitudes also exactly match the amplitudes of the arbitrary-phased signal, while using the real test sinusoids did not give the correct amplitudes.

That is not all -- you can also determine the phase of the orignal sinusoids in the signal:

```
- Phase[finalsim]
```

```
{0.2,1.2,1.5,3.}
```

Here is the original phase, which matches exactly:

```
phase
```

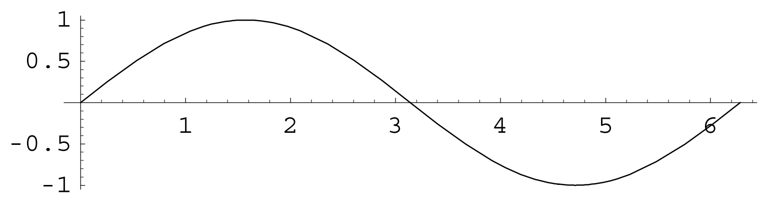
```
{0.2,1.2,1.5,3.}
```

Part IV: Symmetry

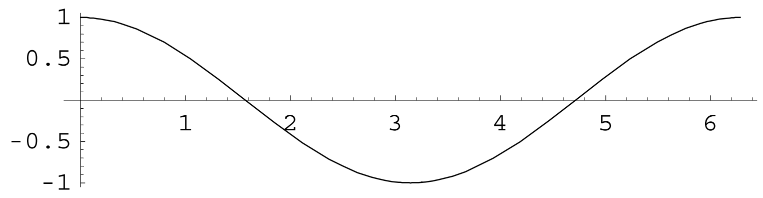
Even and Odd functions

The problem with using only a real sinsoid is that you have to choose a sinusoid which will not correctly identify the amplitude of another sinusoid with the same frequency. This section demonstrates this fact.

```
Plot[Sin[x], {x, 0, 2 Pi}, AspectRatio→1/4];
```

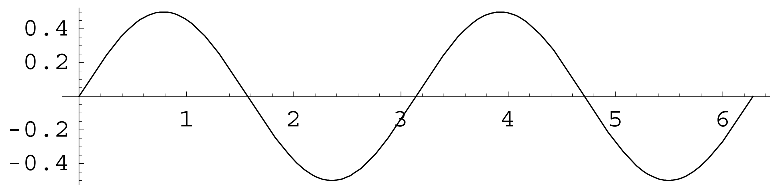


```
Plot[Cos[x], {x, 0, 2 Pi}, AspectRatio→1/4];
```



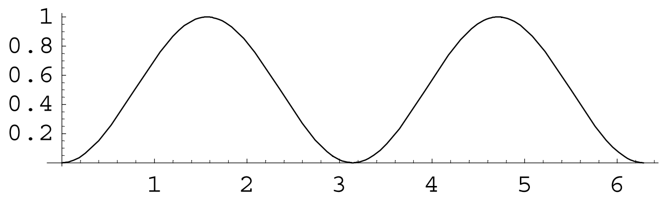
If you multiply the sine and the cosine functions together, you will get a sinewave with twice the frequency of the original:

```
Plot[Cos[x] Sin[x], {x, 0, 2 Pi}, AspectRatio→1/4];
```



This implies that there is no Cosine quality in a Sine and no Sine in a Cosine. However, if you multiply a sine by a sine, all of the function will be positive:

```
Plot[Sin[x] Sin[x], {x, 0, 2 Pi}, AspectRatio→1/4];
```



```
Integrate[Sin[x]  Sin[x], {x, 0, 2 Pi}] / Pi
```

```
1
```

```
Integrate[Cos[x]  Sin[x], {x, 0, 2 Pi}] / Pi
```

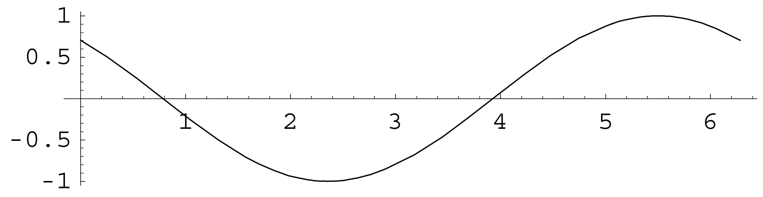
```
0
```

Thus, a sinewave cannot "see" a cosinewave and a cosinewave cannot "see" a sinewave. This is related to the symmetry of each function. A cosine was has even symmetry because it looks the same on each side of the line x=0 if you place a mirror on that line. A sine function has odd symmetry because it can be rotated 180 from the point (x,y) = (0,0) and still look the same. All functions can be broken down into two pieces: one which has even symmetry, and one which has odd symmetry.

An arbitrary phased sinusoid which is neither a sinewave or a cosine wave can be broken down into two pieces, one which is symmetric and one which is antisymmetric. The symmetric piece corresponds to a cosine and the antisymmetric piece corresponds to a sine. The amplitudes of the cosine and sine add up to the amplitude of the original arbitrary-phased sinusoid.

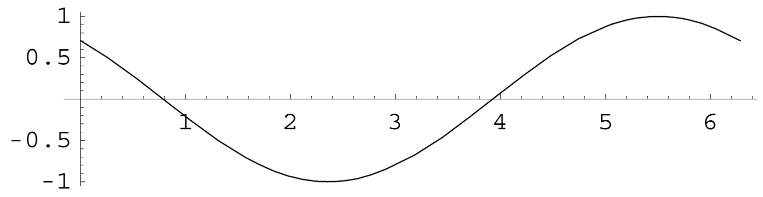
For example, try a phase of Pi/4 which is half-way between a cosine with 0 phase and a sine with 0 phase (or a cosein with Pi/2 phase):

```
Plot[Cos[x + Pi/4], {x, 0, 2 Pi}, AspectRatio→1/4];
```



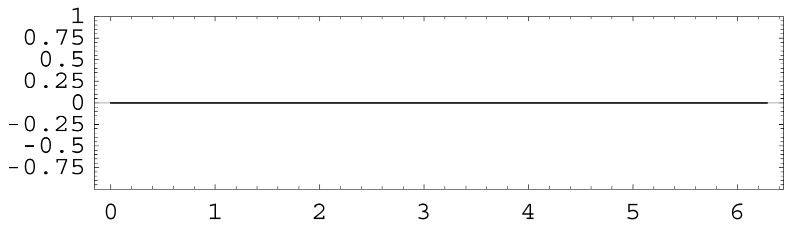
This sinusoid can be broken down into two sinusoids: one sinewave and one cosine which can be added together to create the original phased sinusoid:

```
Plot[ (Cos[x] - Sin[x])/Sqrt[2], {x, 0, 2 Pi}, AspectRatio→1/4];
```



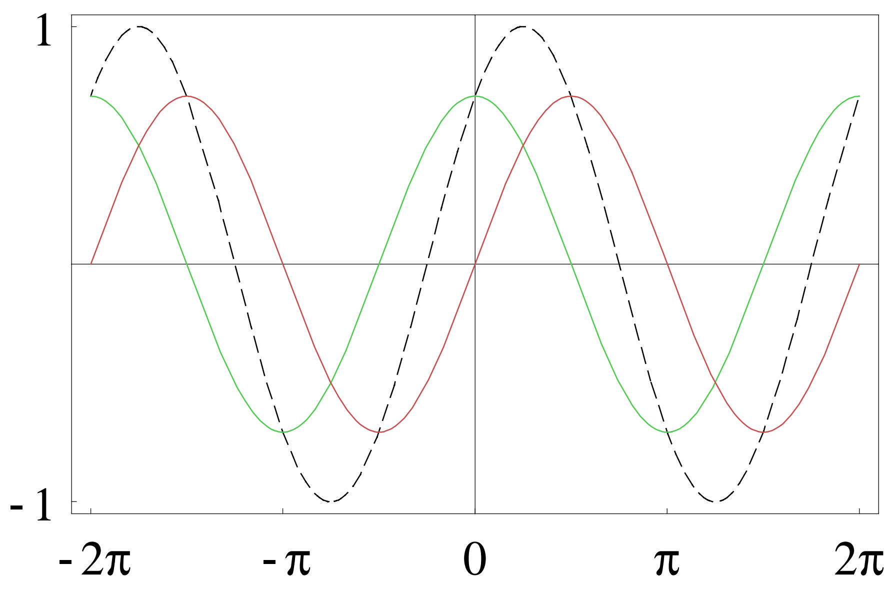
Check to see that the functions are equal by plotting their difference:

```
Plot[( Cos[x] - Sin[x])/Sqrt[2] - Cos[x + Pi/4],
  {x, 0, 2 Pi}, AspectRatio→1/4, PlotRange→{-1,1}, Frame→True];
```

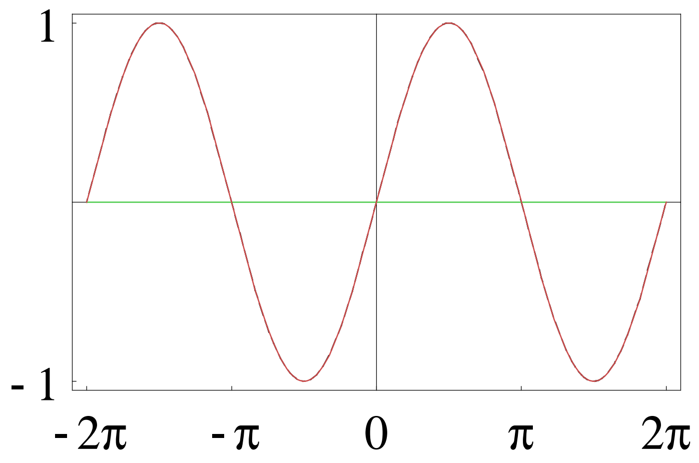


Here is a function which will extract the sine and cosine of any sinusoid:

```
EOSPlot[Pi/4];
```



```
Table[EOSPlot[x 2 Pi], {x, 0,1, 1/50}];
```



With one sine and one cosine, you can extract the amplitude of a sinusoid with any phase in a signal. Remember that the complex sinusoid is a cosine on the real axis and a sine on the imaginary axis, so you now know why the complex sinusoid can extract the amplitude of a sinusoid which has any phase.

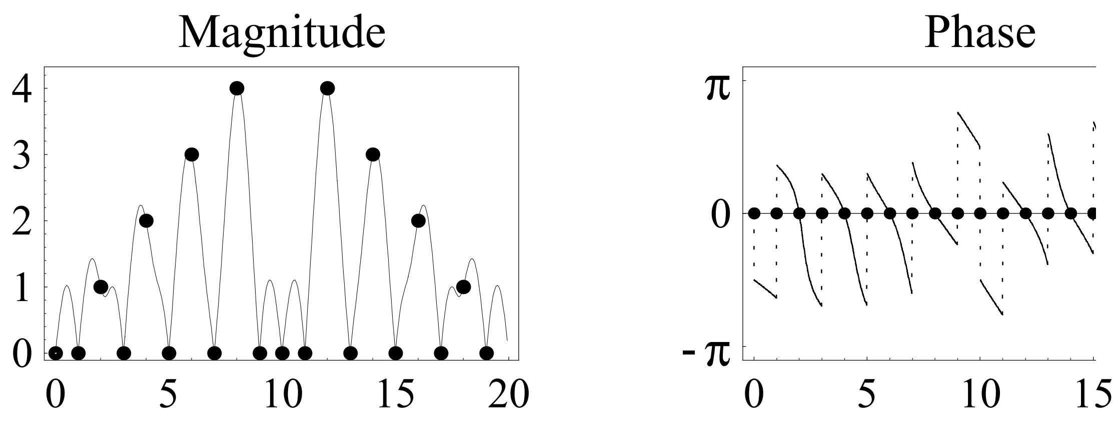
V Spectrum plots

The spectrum is usually given as a plot of amplitude v frequency of sinusoids, but there is also the phase plot which is the other part of the spectrum.

```
sampsignal
```

```
{10,-2.73607,-2.5,1.73607,-2.5,2.,-2.5,1.73607,-2.5,-2.73607,
 10.,-2.73607,-2.5,1.73607,-2.5,2.,-2.5,1.73607,-2.5,-2.73607}
```

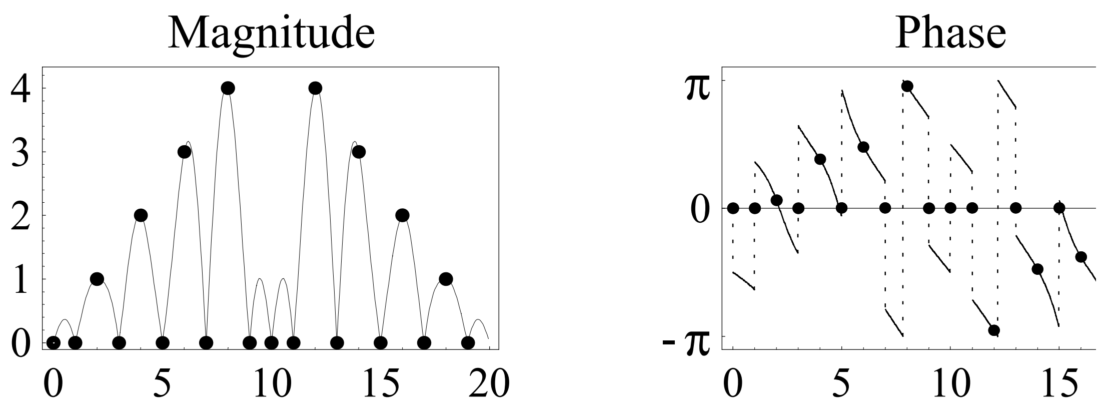
```
MagnitudePhasePlot[NumericDFT[sampsignal/10],LogScale→False];
```



```
sampphasesignal
```

```
{-2.04298,-0.912488,-0.667663,0.187638,1.84216,-4.42753,
 3.55848,-4.07,-2.69,9.22238,-2.04298,-0.912488,-0.667663,
 0.187638,1.84216,-4.42753,3.55848,-4.07,-2.69,9.22238}
```

```
MagnitudePhasePlot[NumericDFT[sampphasesignal/10],LogScale→False];
```

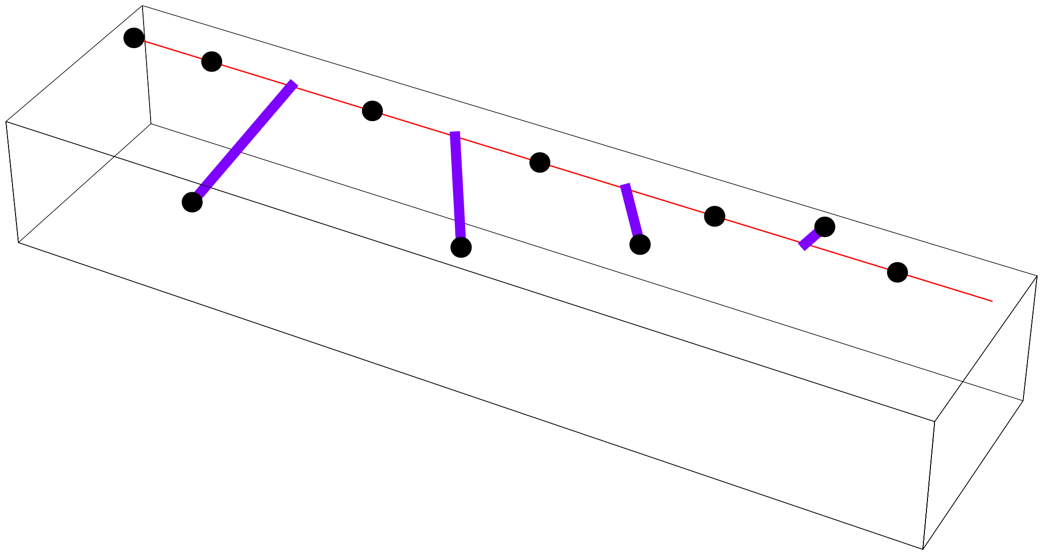


Here are functions to display the sinusoid frequencies, amplitudes and phases all in one spectral plot:

```
Spectrum3D[signal_] := Module[
  {spectrum, spectrumpoints, zeropts, bars, length},
  len= Length[signal];
  spectrum = NumericDFT[sampphasesignal/len];
  spectrum = Drop[spectrum, len/2];
  spectrumpoints =
   Transpose[{Table[x, {x, 0, len/2-1}], Re[spectrum], Im[spectrum]}] ;
  zeropts = Table[{x, 0, 0}, {x, 0, len/2-1}];
  bars = Map[Line,Transpose[{spectrumpoints, zeropts}]];
  Show[Graphics3D[
    {RGBColor[0,0,0],PointSize[0.02],  Map[Point,spectrumpoints],
     RGBColor[1,0,0],Line[{{0,0,0}, {10,0,0}}], RGBColor[0.5,0,1],
     Thickness[0.01],  bars}], PlotRange→All]

 ]
```

```
Spectrum3D[sampphasesignal];
```



It is interesting to view the continuous version of the spectrum in 3D:

```
Spectrum3DInterp[signal_, viewpoint_] := Module[
  {spectrum, spectrumpoints, zeropts,
   bars, length, interppoints, ispectrum, fixplot},

  len= Length[signal];
  spectrum = NumericDFT[sampphasesignal/Sqrt[len]];
  spectrum = Drop[spectrum, len/2];
  spectrumpoints =
   Transpose[{Table[x, {x, 0, len/2-1}], Re[spectrum], Im[spectrum]}] ;
  zeropts = Table[{x, 0, 0}, {x, 0, len/2-1}];
  bars = Map[Line,Transpose[{spectrumpoints, zeropts}]];

  ispectrum = NumericDFT[ZeroPad[signal,  len * 8]/(Sqrt[len] )];
  ilen = Length[ispectrum];
  ispectrum = Drop[ispectrum, ilen/2];
  interppoints = Transpose[
    {Table[N[x/9], {x, 0, ilen/2-1}], Re[ispectrum], Im[ispectrum]}] ;

  Show[Graphics3D[{RGBColor[0,0,0],PointSize[0.02],
     Map[Point,spectrumpoints], Thickness[0.015], RGBColor[1,0,0],
     Line[{{0,0,0}, {10,0,0}}], RGBColor[0.5,0,1], Thickness[0.01],
     bars, Thickness[0.008],  RGBColor[1, 0.5, 0], Line[interppoints]}],
   PlotRange→{{-10,10},{-10,10},{-8,3}},AspectRatio→1/1,
   Boxed→False, ViewPoint→viewpoint]
]
```

```
animation = Table[Spectrum3DInterp[sampphasesignal,
    {10 Cos[x],10 Sin[x],0.5}], {x, 0, 2 Pi-2 Pi/50, 2 Pi/50}];
```