MUSC 208 Winter 2014
John Ellinger, Carleton College

Lab 6 Wavetables And Interpolation

Create a m208Lab6 folder on your computer. Open Octave and make the m208Lab6 folder your working directory.

```
octave-3.4.0:1> cd ~/Desktop/m208Lab6
octave-3.4.0:2> pwd
ans = /Users/je/Desktop/m208Lab6
```

By definition, a wavetable contains exactly one period of an arbitrary waveform. The number of samples in wavetables are generally a power of 2 for computing efficiency.

Generate A Single Cycle Sine Wave Wavetable

Create a new Octave function that generates one period of a sine wave for a given table length.

Open genSinTable.m In Your Text Editor

```
octave-3.4.0:168> edit genSinTable.m
```

Create The Gensintable Help Text and Write the Code

```
## [ret] = genSinTable( len )
##      returns one period of a sine wavetable
##      of length len samples

function [ ret ] = genSinTable ( len )
    ## you write the code
endfunction
```

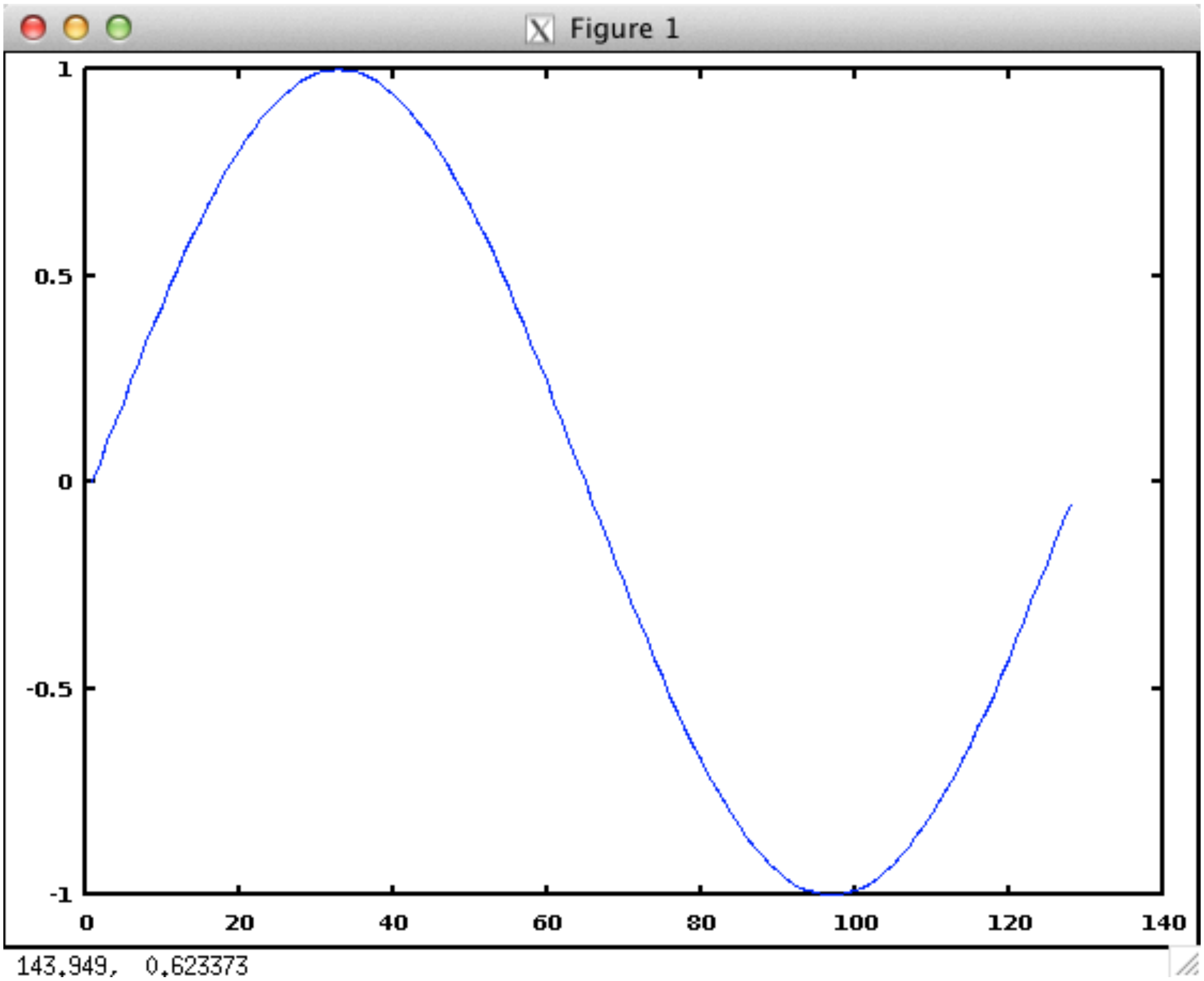
Create And Plot A Sine Wavetable

Generate a wave table of length 128 for a sine wave. Use the standard sine wave formula with an amplitude of 1.0 and a frequency of 1 Hz.

$$y(n) = A\sin\left(\frac{2\pi fn}{SR}\right)$$

Test Your Code

```
octave:50> wav = genSinTable( 128 );
octave:51> plot( wav );
```



Check The Help Text

```
octave:4> help genSinTable
'genSinTable' is a function from the file /Users/;

 [ret] = genSinTable( len )
     returns one period of a sine wavetable
     of length len samples
```

Octave Tips

Move to beginning of line: Type Ctrl-A

Move to end of line: Type Ctrl-E

Up arrow: recall previous commands

Down arrow: recall next commands

Different Plot Types

Try these different plot commands one line at a time in the Terminal.

```
octave:35>  wav16 = genSinTable( 16 );
octave:36>  plot( wav16 );
octave:37>  plot( wav16, "r" );
octave:38>  stem( wav16 );
octave:39>  stem( wav16, "." );
octave:40>  bar( wav16 );
octave:41>  barh( wav16 );
octave:42>  stairs( wav16 );
octave:43>  axis( [0 17 -1.1 1.1] ); # [xLo xHi yLo yHi]
octave:44>  grid; # turn grid on
octave:45>  grid; # turn grid off
octave:46>  hold; # don't erase existing plot
octave:47>  plot( wav16, "r" );
octave:48>  hold; # hold off
octave:49>  plot( wav16 );
```

History command

This command shows the last 15 commands used.

```
octave:80> history 15
 1094  plot( wav16 );
 1095  plot( wav16, "r" );
 1096  stem( wav16 );
 1097  stem( wav16, "." );
 1098  bar( wav16 );
 1099  barh( wav16 );
 1100  stairs( wav16 );
 1101  axis( [0 17 -1.1 1.1] ); # [xLo xHi yLo yHi]
 1102  grid; # turn grid on
 1103  grid; # turn grid off
 1104  hold; # don't erase existing plot
 1105  plot( wav16, "r" );
 1106  hold; # hold off
 1107  plot( wav16 );
 1108 history 15
```

Copy a Block of Text (Mac Terminal)

Hold down the Option key and you can copy a block of text. Unknown for windows. Test it and let me know.

```
octave:65>
octave:65>  wav16 = genSinTable( 16 );
octave:66>  plot( wav16 );
octave:67>  plot( wav16, "r" );
octave:68>  stem( wav16 );
octave:69>  stem( wav16, "." );
octave:70>  bar( wav16 );
octave:71>  barh( wav16 );
octave:72>  stairs( wav16 );
octave:73>  axis( [0 17 -1.1 1.1] ); # [xLo xHi yLo yHi]
octave:74>  grid; # turn grid on
octave:75>  grid; # turn grid off
octave:76>  hold; # don't erase existing plot
octave:77>  plot( wav16, "r" );
octave:78>  hold; # hold off
octave:79>  plot( wav16 );
octave:80>
```

Diary command

Records everything you do in an octave session. Type "help diary".

Turn The Diary On For Lab6

```
octave:85> # turn diary on with and save to a file named diary_m208Lab6.txt
octave:85> diary diary_m208Lab6.txt
```

You'll have to turn the diary off to see the text. You can open and close the diary at any time. For now just leave it running. We'll turn it off at the end of Lab 6.

Create Play A Wavetable

Repeat the sine table 250 times, play it, and save it to a wav file..

```
octave:33> wtab = genSinTable( 128 );
octave:34> wav = repmat( wtab, 1, 250 );
octave:35> help wavwrite
octave:36> wavwrite( wav', 44100, 16, "table128x250.wav" );
```

The single quote after wav' converts a row vector into a column vector wavwrite( ) expects a column vector, play samples( ) works withboth.

Wavetable Math

Question 1 - What frequency will you hear?

$$f = \frac{SR}{tableLength}$$

Question 2 - How long will the sound last?

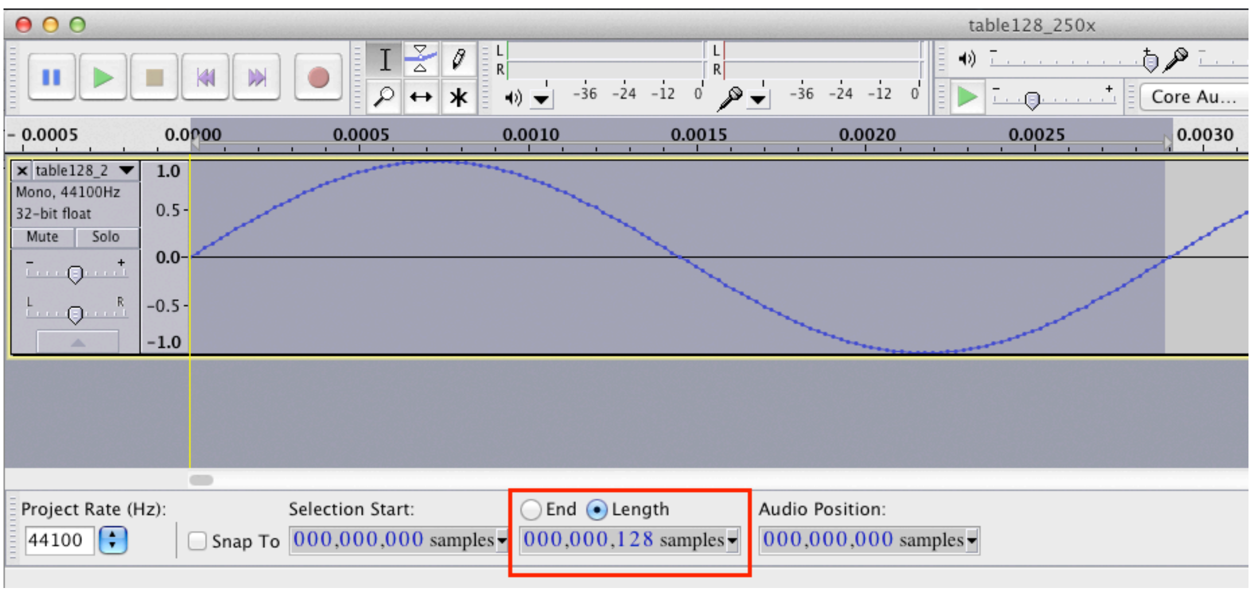
$$\text{seconds} = \frac{tableLength \times repeats}{SR}$$

Calculate Frequency and Duration In Octave

```
octave:37> frequency = 44100 / 128
frequency =  344.53
octave:38> seconds = 128 * 250 / 44100
seconds =  0.72562
```

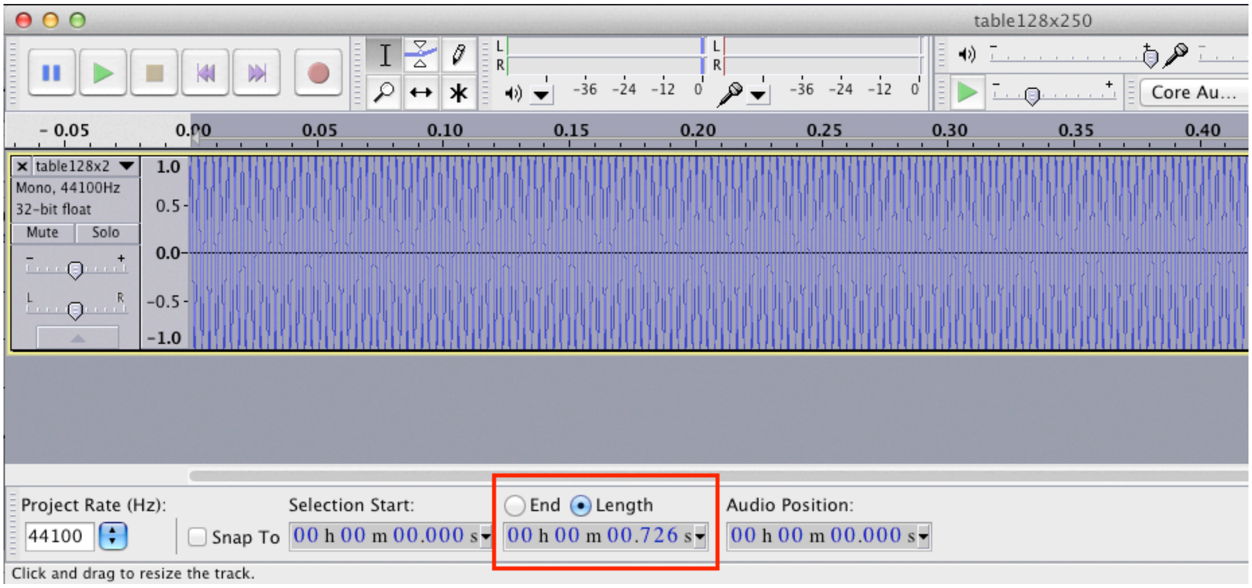
Find the Number of Samples in One Period in Audacity

Open the "table128_250x.wav" file in Audacity and zoom in until you can see one full period of the sine wave. Select one period of samples and change the Length popup to display samples. Audacity should report 128 samples.



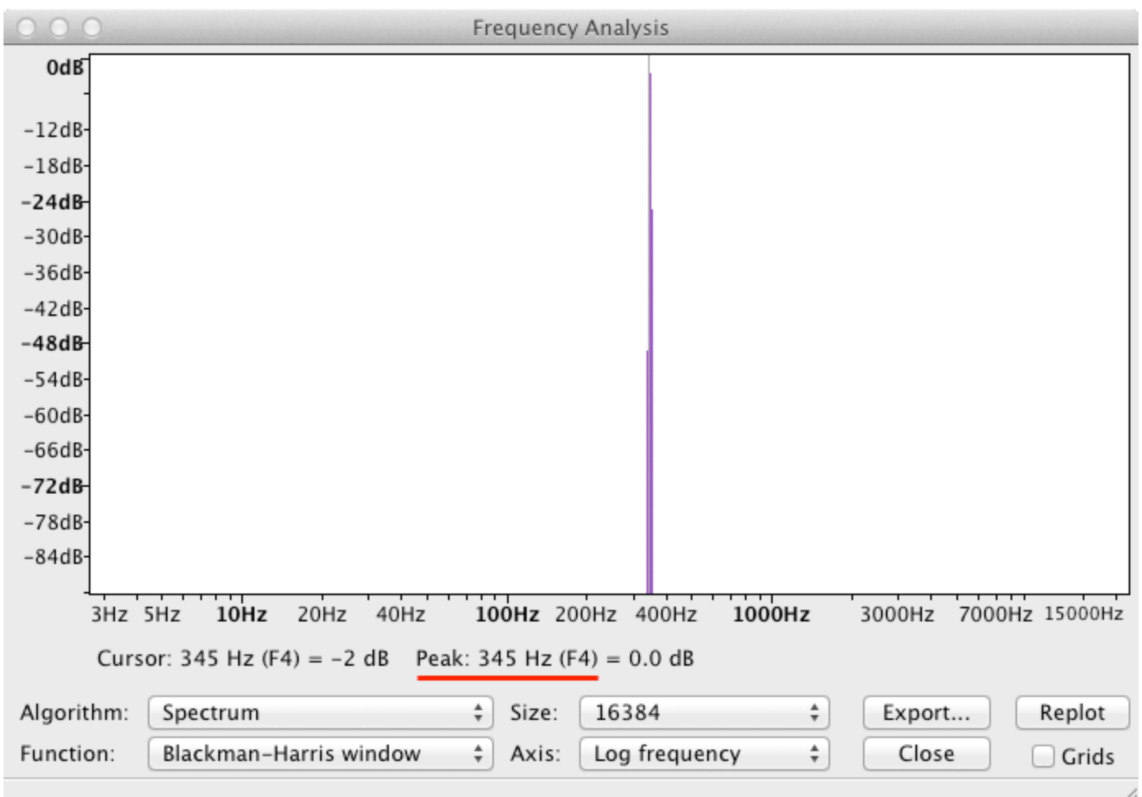
Calculate the Duration in In Audacity

Select the entire waveform and change the length popup menu to "hh:mm::ss+milliseconds". You should see a total duration of 726 milliseconds.



Calculate the Frequency In Audacity

Select the entire waveform and choose Plot Spectrum from the Analyze menu. Make these settings.



Wavetable Frequency

At a given sampling rate (44100), the length of the wavetable determines the frequency that will be heard when it is played.

$$f = \frac{SR}{tableLength}$$

Computer based wavetable lengths are almost always powers of two for computing efficiency. The frequencies produced by these power of two wavetable lengths are shown below. The sample rate is 44100.

Power of 2	Table Size	Frequency in Hz
2^1	2	22050
2^2	4	11025
2^3	8	5512.5
2^4	16	≈ 2756
2^5	32	≈ 1378
2^6	64	≈ 689
2^7	128	≈ 345
2^8	256	≈ 172
2^9	512	≈ 86
2^{10}	1024	≈ 43
2^{11}	2048	≈ 21.5
2^{12}	4096	≈ 10.8
2^{13}	8192	≈ 5.4
2^{14}	16384	≈ 2.7
2^{15}	32768	≈ 1.3

How Do You Make A Wavetable Play At Any Frequency

That question will be the subject for the rest of the lab 6.

Generate a Wavetable of Specified Length and Duration

```
octave:13> edit genExtSinTable.m
```

You Write the Code

```
## [ret] = genExtSinTable (len, secs)
##     return a waveform of duration secs seconds
##     using a wavetable of length len

function [ ret ] = genExtSinTable (len, secs)
    # you write the code
endfunction
```

generate one period table using wav = genSinTable(len)
calculate how many repetitions of len it will take to fill 44100 samples (one second)
use the floor function to round down to the nearest sample
use the repmat function to make as many copies as necessary to fill duration secs
return the wavetable

Play It

```
octave:41> wav = genExtSinTable( 64, 4 );
octave:42> playsamples( wav );
```

Verify The Duration

```
octave:43> length( wav ) / 44100
ans =  3.9996
```

Test Octave Help for genExtendedSinTable

```
octave:15> help genExtSinTable
'genExtSinTable' is a function from the file /Users/

 genExtSinTable (len, secs)
     return a waveform of duration secs seconds
     using a wavetable of length len
```

Write an Octave Function to Play Every Other Sample

```
octave-3.4.0:39> edit everyOtherSample.m
```

```
## everyOtherSample ( wav )
##      input: any waveform
##      output: a new wav containing every other sample
##      of the original

function [ ret ] = everyOtherSample ( wav )
    # you write the code
endfunction
```

Code Outline

```
function [ret] = everyOtherSample( wavIn )
    # declare an empty array to hold the output samples: wavOut = [ ];
    # declare a variable to hold the sample index for wavIn: inN;
    # declare a variable to hold the sample index for wavOut: outN;
    # write a loop to access every odd numbered sample of wavIn
    # beginLoop
        # if inN is odd,
        #     wavOut( outN ) = wavIn( inN );
        # endif
    # endLoop
    # ret = wavOut
endfunction
```

Test Help For everyOtherSample

```
octave:17> help everyOtherSample
'everyOtherSample' is a function from the file /Users/

 everyOtherSample ( wav )
     input: any waveform
     output: a new wav containing every other sample
     of the original
```

Test everyOtherSample

```
octave:47> wavIn = genExtSinTable( 128, 1.0 );
octave:48> wavOut = everyOtherSample( wavIn );
octave:49> playsamples( wavIn );
octave:50> playsamples( wavOut );
```

Question 1

How are the frequencies of wavIn and wavOut related? Prove it in Audacity.

Question 2

How are the durations of wavIn and wavOut related? Prove it in Octave.

Write an Octave Function to Play Every Sample Twice

```
octave-3.4.0:42> edit everySampleTwice.m
```

```
## [ret] = everySampleTwice (wav)
##     input: any wav
##     output: a new waveform containing every sample twice
##     n1 n2 n3 becomes n1 n1 n2 n2 n3 n3 ...

function [ ret ] = everySampleTwice ( wav )
    # you write the code
endfunction
```

Code Help

```
function [ret] = everySampleTwice( wavIn )
    # declare an empty array to hold the output samples: wavOut = [ ];
    # declare a variable to hold the sample index for wavIn: inN;
    # declare a variable to hold the sample index for wavOut: outN;
    # beginLoop
        # wavOut( outN ) = wavIn( inN );
        # Update outN and inN as needed
        # wavOut( outN ) = wavIn( inN );
        # Update outN and inN as needed
    # endLoop
    # ret = wavOut
endfunction
```

Test everySampleTwice Help

```
octave-3.4.0:41> help everySampleTwice
`everySampleTwice' is a function from the file /Users/je/De

 everySampleTwice (wav)
     input: any waveform sample values computed previously
     output: a new waveform containing every sample twice
     n1 n2 n3 becomes n1 n1 n2 n2 n3 n3 ...
```

Test everySampleTwice

```
octave:90> wavIn = genExtSinTable( 128, 1.0 );
octave:91> wavOut = everySampleTwice( wavIn );
octave:92> playsamples( wavIn );
octave:93> playsamples( wavOut );
```

Question 1

How are the frequencies of wavIn and wavOut related? Prove it in Audacity.

Question 2

How are the durations of wavIn and wavOut related? Prove it in Octave.

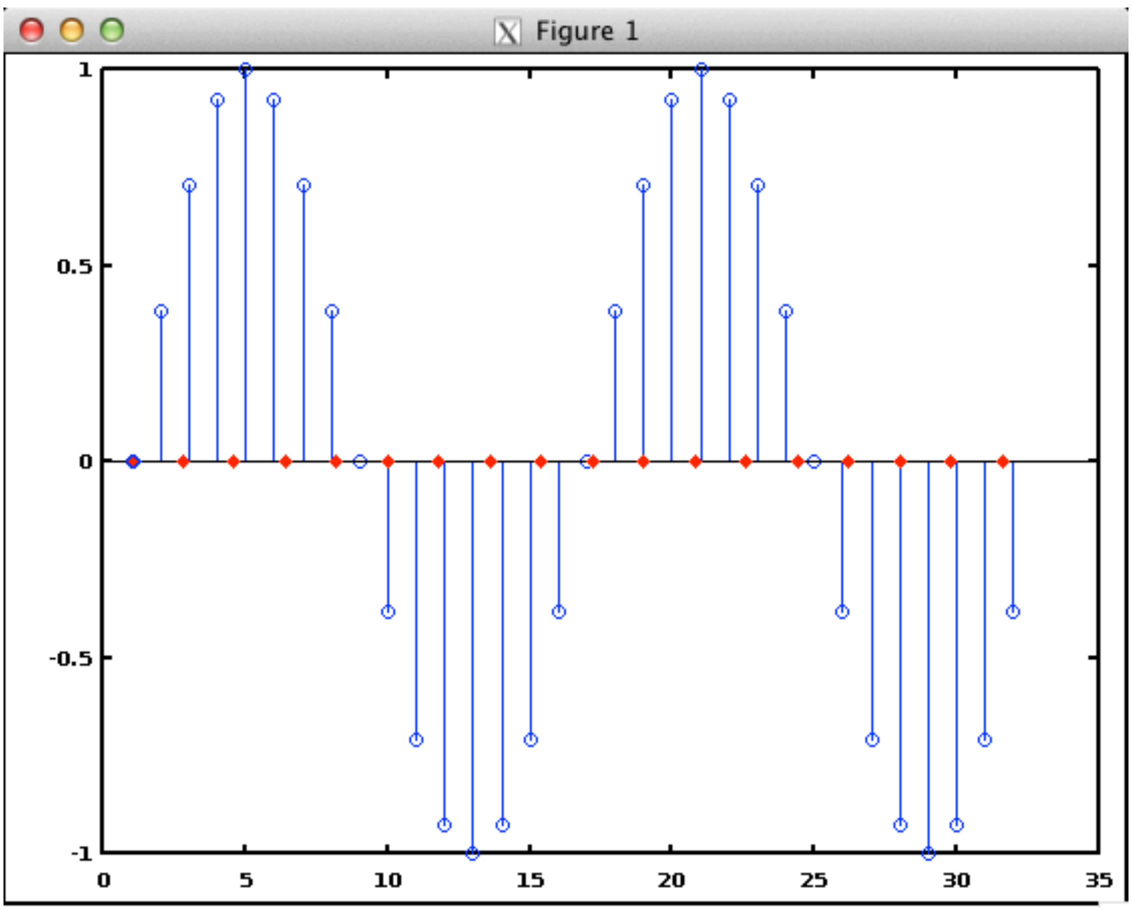
Wavetable Interpolation

Phase Increment

All examples up to this point have accessed points on the time axis (x axis) where actual samples exist. Samples are uniformly spaced at a distance of 1/SR called the phase_increment. A phase_increment of 1.0 is defined to be 1/SR. The everyOtherSample function used a phase_increment of 2.0. Non integer phase_increments will fall in between existing samples. Phase increments must be positive numbers greater than zero.

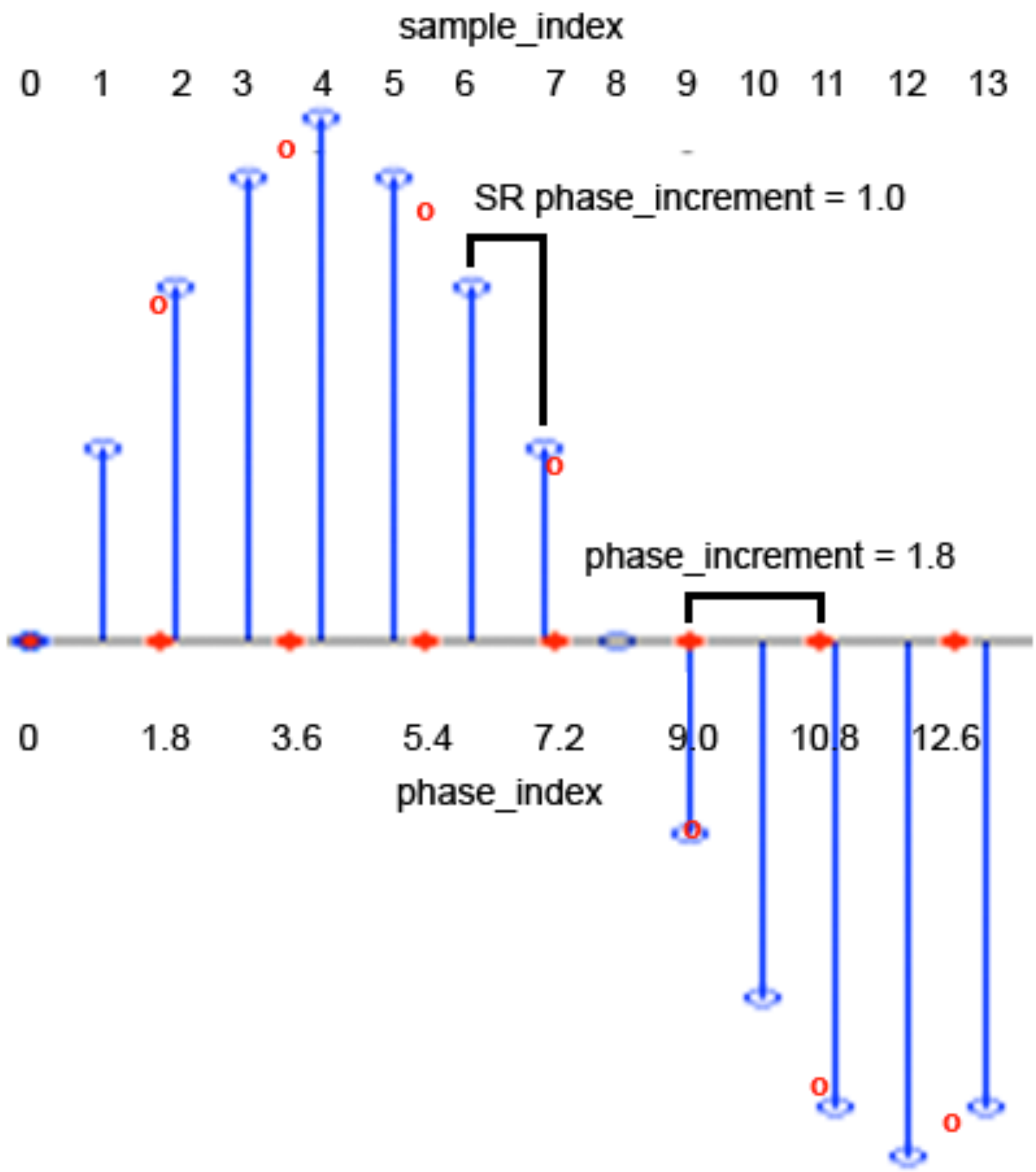
Phase Index

The blue circles in the picture below represent two periods of a sine wavetable of length 16. The blue lines are spaced uniformly using a phase increment of 1.0. The red dots are also uniformly spaced using a phase increment of 1.8.



Phase increments less than 1.0 will produce a longer sound (more samples) at a lower frequency. Phase increments greater than 1.0 will produce a shorter sound (fewer samples) at a higher pitch.

In the picture below blue lines (phase_increment = 1.0) are spaced at integer sample_index locations on the x (time) axis. Existing sample values are shown as blue ellipses. Phase increment of 1.8 are shown as red todd on the x axis. Sample values for the red dot locations are shown as red circles and would have to be estimated.



Sample Estimation Methods

There are three estimation methods that vary is speed and quality: truncation, rounding, and interpolation.

Truncation

Truncation is the fastest method but introduces noise in the output. Truncation chooses the sample value to the left of the phase_index location. The Octave floor() function.

Rounding

Rounding is almost as fast but still introduces noise in the output. Rounding chooses the closest sample value to the left or right of the phase_index. The Octave round() function.

Interpolation

Linear interpolation is one of many interpolation methods. It is slower than truncation and rounding but results in the lowest noise, hence best quality. Linear interpolation estimates the sample value based on the phase_index's proportional distance between the samples to its left and right.

The phase_index is an integer multiple of the phase_increment that reports the current phase location on the x axis (time) as the you step through the wavetable. The phase_index must be capable of wrapping around to the beginning of the table when exceeds the length of the table.

The Phase Index Formula

$$phase\_index \ = \ \mathrm{mod}(\ previous\_phase \ + \ phase\_increment,\ tableLength \ );$$

Curtis Roads: Computer Music p. 92-93

The Octave mod function mod(x, y) is used to calculate the phase_index. The estimated sample output value can be found using this formula.

$$wavOut \ = \ wavIn[\ phase\_index \ ];$$

When the phase_index contain decimal places, the integer part indicates the sample value to the left of the phase_index and the fractional part indicates the fractional distance past the left sample. The unknown sample value is determined using truncation, rounding, or interpolation methods.

Truncation Example

We'll use a wavetable of length 128 that contains one period of a sine wave. A wavetable of length 128 will play back at ≈345 Hz (phase increment is 1.0). How do we find the phase_increment that will play the wavetable at 1000 Hz?

The Phase Increment Formula

The phase_increment value needed to play a desired frequency given a fixed wavetable size is given by this formula.

$$phase\_increment = \frac{tableLength * frequency}{SR}$$

Curtis Roads: Computer Music p. 92-93

```
octave:96> phase_increment = 128 * 1000 / 44100
phase_increment =  2.9025
```

Create an Octave function called truncateWavetable.m

```
octave:97> edit truncateWavetable.m
```

Enter this code in Octave. Comments are included to help you understand what's happening.

```
## [ret] = truncateWavetable  ( wavIn, amp, freq, secs )
##     computes output samples using truncation (floor function)
##     wavIn contains sample values from a wavetable or a wave file
##     amp is the wavOut amplitued
##     freq is the wavOut frequency
##     secs is the wavOut duration in seconds

function [ ret ] = truncateWavetable ( wavIn, amp, freq, secs )

    SR = 44100;
    numSamples = floor( SR * secs );

    ## initialize empty output array
    wavOut = [];

    ## find length of wavIn
    tableLength = length( wavIn );

    ## Roads phase_increment formula
    phase_increment = tableLength * freq / SR;

    ## set initial value of prev_phase_index
    prev_phase_index = 0;

    ## set first samples equal to each other
    wavOut( 1 ) = amp * wavIn( 1 );

    for outN = 2:numSamples
        ## Roads phase_index formula
        phase_index = mod( prev_phase_index + phase_increment, tableLength );

        ## the floor function strips off the decimal places = TRUNCATE
        inN = floor( phase_index ) + 1;

        ## check to see if we need to wrap around
        if inN > tableLength
            inN = mod( inN, tableLength );
        endif

        ## assign sample value to wavOut
        wavOut( outN ) = amp * wavIn( inN );

        ## update prev_phase_index for next time through the loop
        prev_phase_index = phase_index;
    endfor

    ## return wavOut samples
    ret = wavOut;
endfunction
```

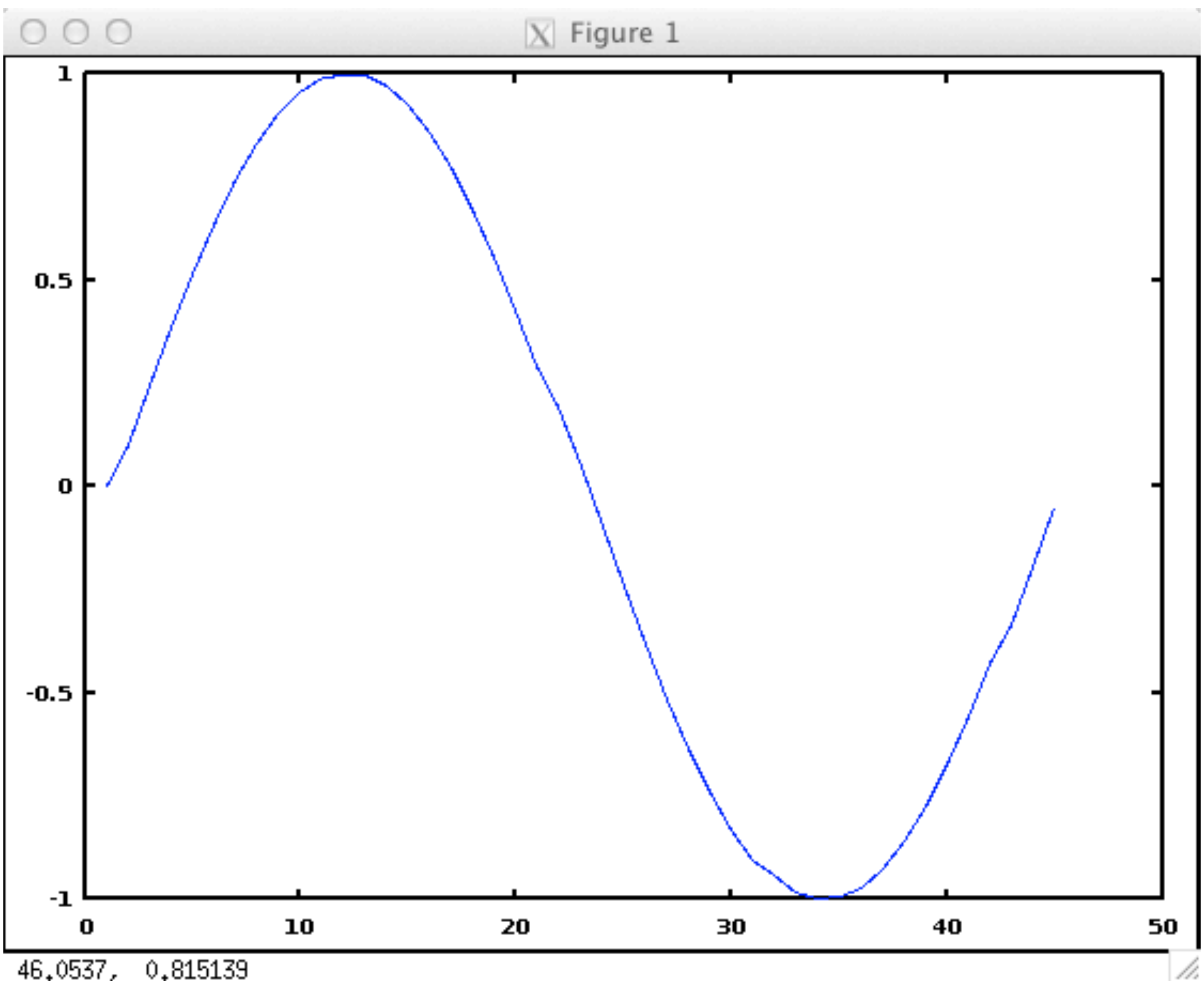
Test truncateWavetable

Given one period of a sine wave in a wavetable of size 128, create a new waveform with an amplitude of 1.0, a frequency of 1000 Hz, and a duration of one second. Play the output waveform and write it to a wav file.

```
octave:98> wavIn = genSinTable( 128 );
octave:99> wavOut = truncateWavetable( wavIn, 1.0, 1000, 1.0 );
octave:100> playsamples( wavOut );
octave:101> wavwrite( wavOut', 44100, 16, "truncate.wav" );
```

Plot wavOut

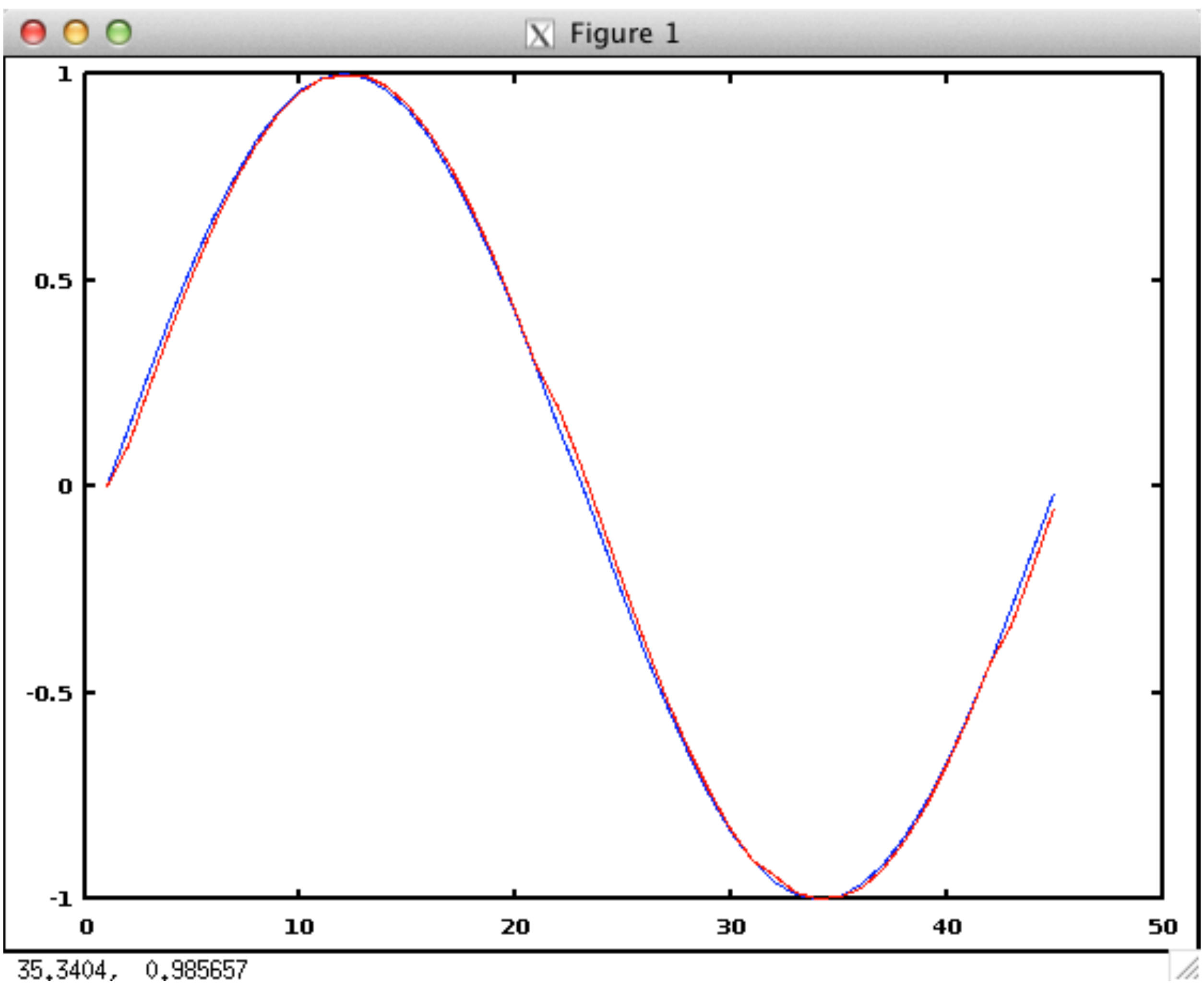
```
octave:102> plot( wavOut( 1:45 ) );
```



Compare a Pure Sine Wave with a Truncated Sine Wave

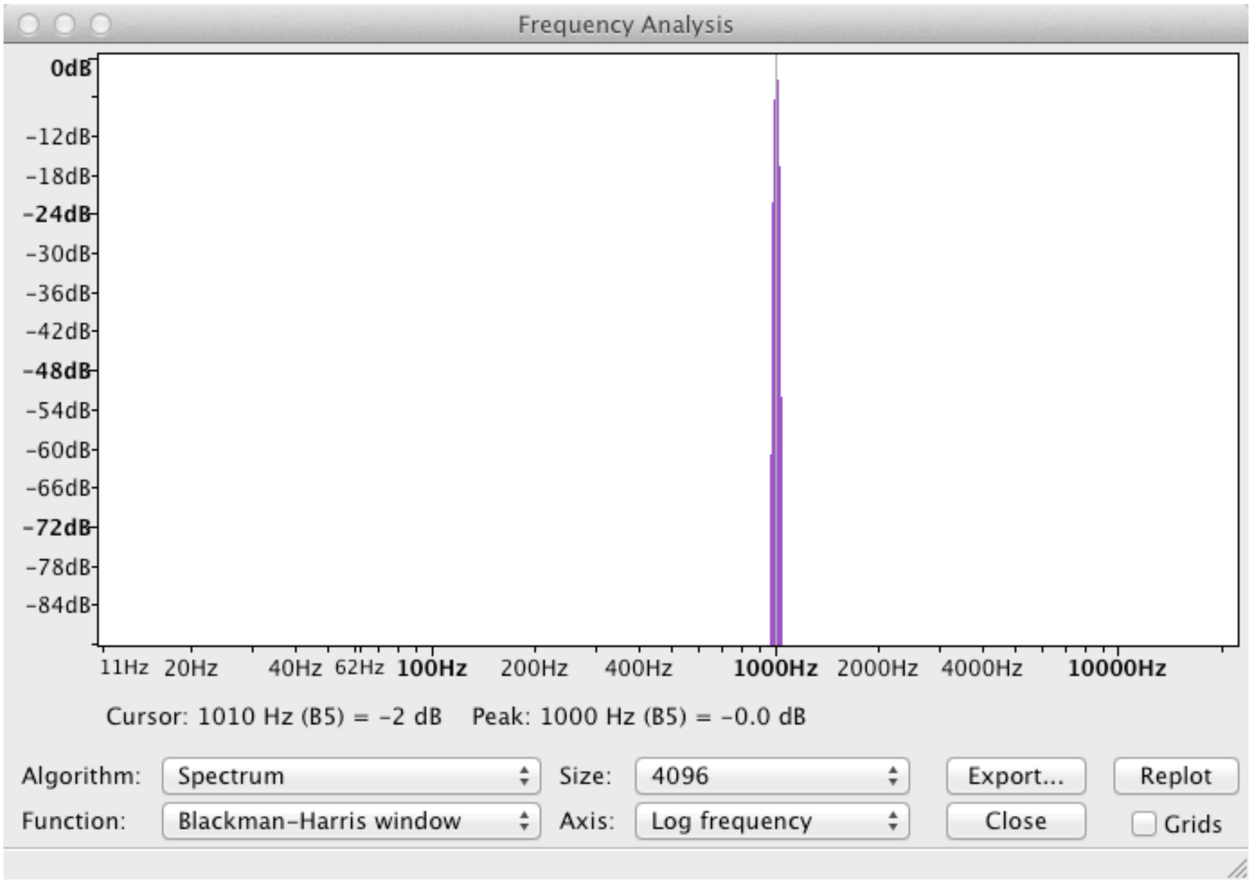
```
octave:194> # Pure Sine Wave at 1000 Hz
octave:194> SR = 44100;
octave:195> T = 1/SR;
octave:196> n = 0:SR-1;
octave:197> nT = n*T;
octave:198> sine1000 = sin( 2 * pi * 1000 * nT );
octave:199> wavwrite( sine1000', 44100, 16, "sine1000.wav" );
octave:200> plot( sine1000( 1:45 ) );
octave:201> hold
octave:202> plot( wavOut( 1:45 ), "r" );
```

The pure sine wave is blue and the truncated one is red. You can see minor deviations from the pure sine wave.These deviations introduce additional unrelated frequencies (noise) in the truncated sine wave.

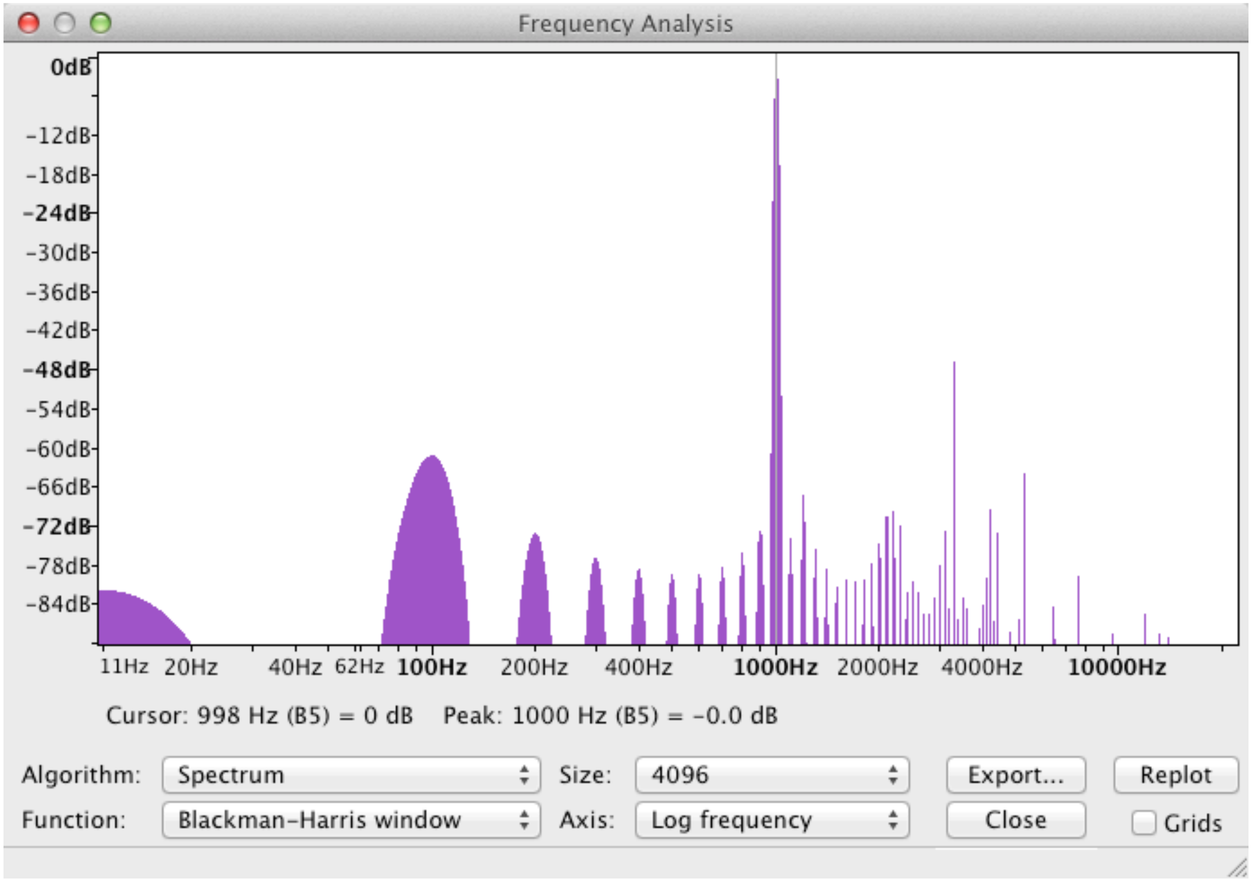


The Spectrum Of A Pure Sine Wave

This is the spectrum of the pure sine wave as shown in Audacity. There is a single frequency component at 1000 Hz.



This is the spectrum of the truncated wavOut as shown in Audacity. The largest frequency component appears at 1000 Hz as expected. However, there is a lot of noise present.



Rounding Example

Create this Octave function.

```
octave:203> edit roundWavetable.m
```

Delete any and all text that appears in roundWavetable.m and then copy the contents of truncateWavetable.m and paste it into roundWavtable.m.

Make these minor changes highlighted in red.

```
## [ret] = roundWavetable ( wavIn, amp, freq, secs )
##     computes wavOut samples using rounding (round function)
##     wavIn contains sample values from a wavetable or a wave file
##     amp is the wavOut amplitued
##     freq is the wavOut frequency
##     secs is the wavOut duration in seconds

function [ ret ] = roundWavetable ( wavIn, amp, freq, secs )

    SR = 44100;
    numSamples = floor( SR * secs );

    ## declare empty array, used later in for loop
    wavOut = [];

    ## find length of wavIn
    tableLength = length( wavIn );

    ## Roads phase_increment formula
    phase_increment = tableLength * freq / SR;

    ## set initial value of prev_phase_index
    prev_phase_index = 0;

    ## set first samples equal to each other
    wavOut( 1 ) = amp * wavIn( 1 );

    for outN = 2:numSamples
        ## Roads phase_index formula
        phase_index = mod( prev_phase_index + phase_increment, tableLength );

        ## the round function rounds to the closest wavIn sample
        inN = round( phase_index ) + 1;

        ## check to see if we need to wrap around
        if inN > tableLength
            inN = mod( inN, tableLength );
        endif

        ## assign sample value to wavOut
        wavOut( outN ) = amp * wavIn( inN );

        ## update prev_phase_index for next time through the loop
        prev_phase_index = phase_index;
    endfor

    ## return wavOut samples
    ret = wavOut;

endfunction
```

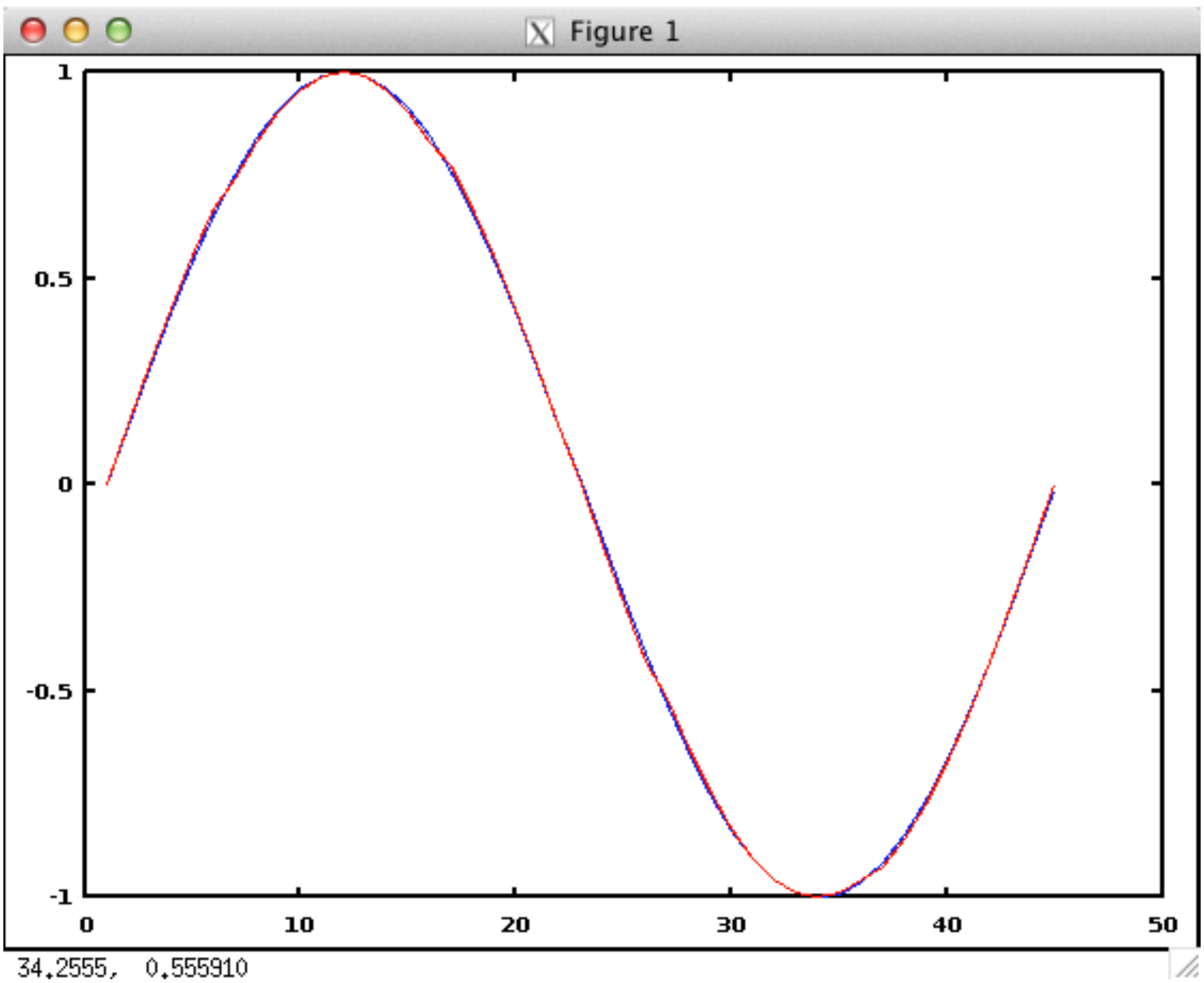
Test It

```
octave:226> wr = roundWavetable( wavIn, 1.0, 1000, 1.0 );
octave:227> playsamples( wr );
octave:228> wavwrite( wr', 44100, 16, "round.wav" );
```

Plot It

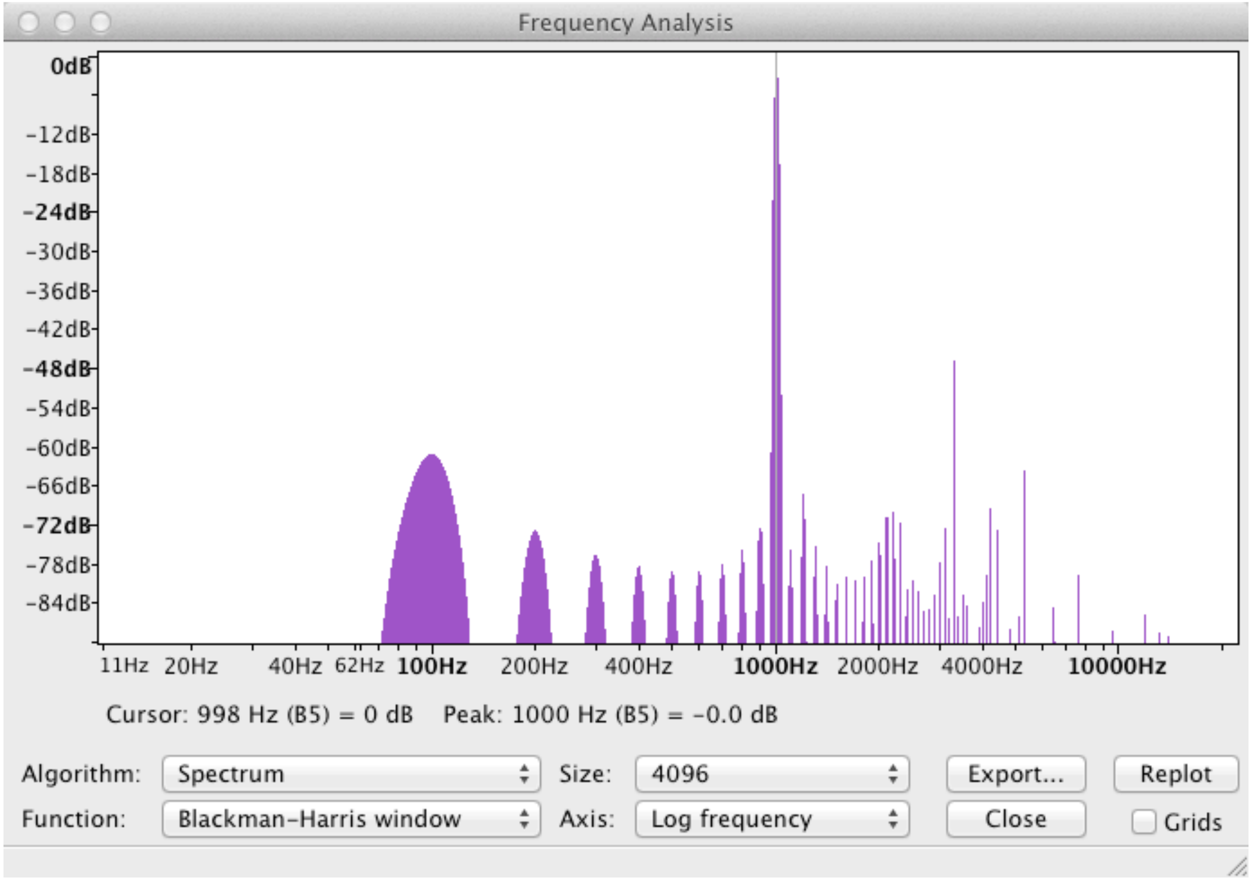
```
octave:229> clf; # clear figure (clear plots)
octave:230> plot( sine1000( 1:45 ) );
octave:231> hold
octave:232> plot( wr( 1:45 ), "r" );
```

The rounded sine wave (red) seems to match the pure sine wave (blue) more closely. Let's see if that reduced the noise.



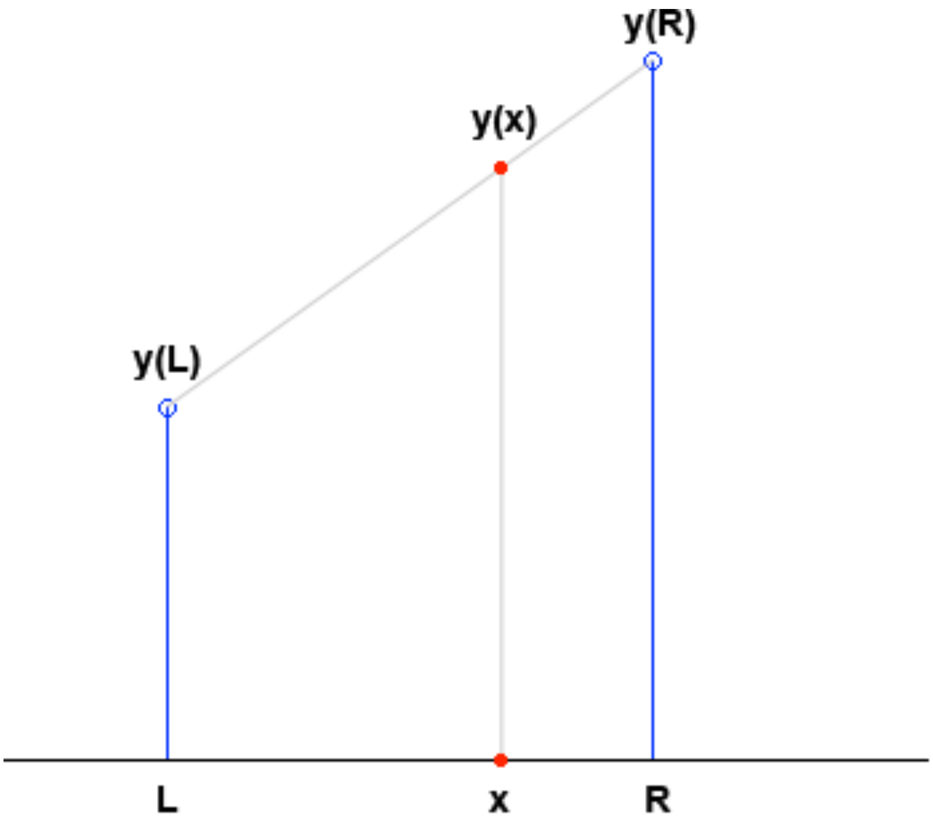
Plot the Spectrum in Audacity

There's still plenty of unwanted additional frequencies. However none in the 1-20 Hz band that were present in the truncate example



Linear Interpolation Formula

$$sampleValue = sampleLeft + fraction \bullet (sampleRight - sampleLeft)$$



$$\frac{x - L}{R - L} = fraction \Rightarrow x = L + fraction \bullet (R - L)$$

Here's a short example in octave.

```
octave:98> wt = genSinTable( 128 );
octave:105> phase_index = 56.368
octave:106> leftN = floor( phase_index )
octave:111> wt( leftN )
octave:107> rightN = leftN + 1
octave:112> wt( rightN )
octave:108> fraction = rem( phase_index, 1 )
octave:109> fraction = phase_index - leftN
octave:110> estVal = wt(leftN)+fraction*(wt(rightN)-wt(leftN))
```

You should see these results.

```
phase_index =  56.368
leftN =  56
rightN =  57
wt( leftN ) =  0.42756
wt( rightN ) =  0.38268
fraction =  0.36800
estVal =  0.41104
```

Linear Interpolation Example

```
octave-3.4.0:49> edit interpWavtable.m
```

Copy and paste the function body of truncateWavtable in the interpWavtable function body. Then make these changes.

```
## interpWavetable  ( wavIn, amp, freq, secs )
##     computes output samples using a linear interpolation formula
##     wavIn contains sample values from a wavetable or a wave file
##     amp is the wavOut amplitued
##     freq is the wavOut frequency
##     secs is the wavOut duration in seconds

function [ ret ] = interpWavetable ( wavIn, amp, freq, secs )

    SR = 44100;
    numSamples = floor( SR * secs );

    ## initialize empty output array
    wavOut = [];

    ## find length of wavIn
    tableLength = length( wavIn );

    ## Roads phase_increment formula
    phase_increment = tableLength * freq / SR;

    ## set initial value of prev_phase_index
    prev_phase_index = 0;

    ## set first samples equal to each other
    wavOut( 1 ) = amp * wavIn( 1 );

    for outN = 2:numSamples
        ## Roads phase_index formula
        phase_index = mod( prev_phase_index + phase_increment, tableLength );

        ## the floor function strips off the decimal places = TRUNCATE
        inN = floor( phase_index ) + 1;

        ## check to see if we need to wrap around
        if inN > tableLength
            inN = mod( inN, tableLength );
        endif

        ## find the samples to the left and right
        leftN = inN;
        rightN = inN + 1;

        ## check to see if we need to wrap around
        if rightN > tableLength
            rightN = mod( rightN, tableLength );
        endif

        ## Linear interpolation formula
        fraction = rem( phase_index, 1 );
        interpValue = wavIn(leftN) + fraction * (wavIn(rightN) - wavIn(leftN));

        ## assign sample value to wavOut
        wavOut( outN ) = amp * interpValue;

        ## update prev_phase_index for next time through the loop
        prev_phase_index = phase_index;
    endfor

    ## return wavOut samples
    ret = wavOut;
endfunction
```

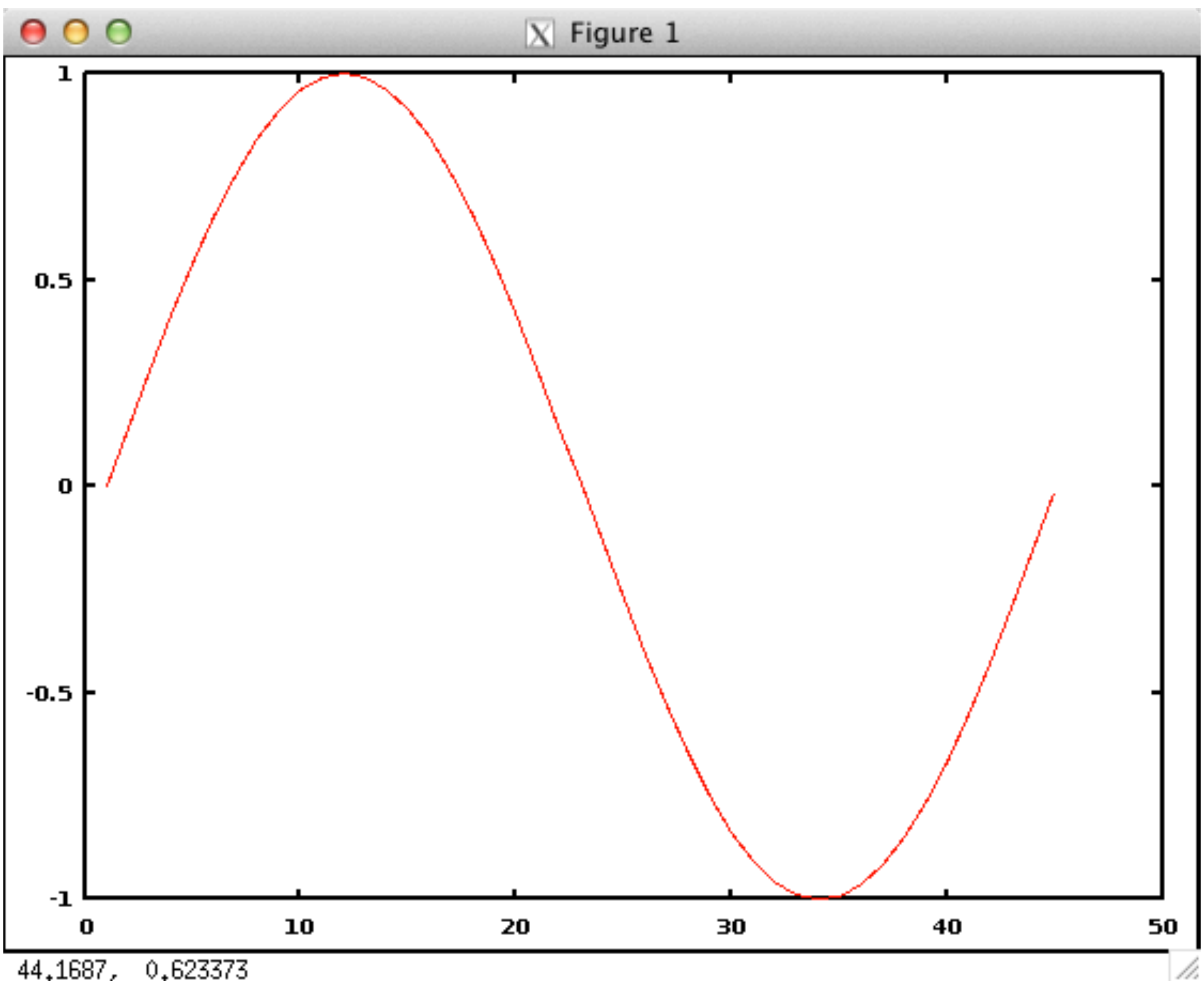
Test It

```
octave:241> wi = interpWavetable( wavIn, 1.0, 1000, 1.0 );
octave:242> playsamples( wi );
octave:243> wavwrite( wi', 44100, 16, "interp.wav" );
```

Plot It

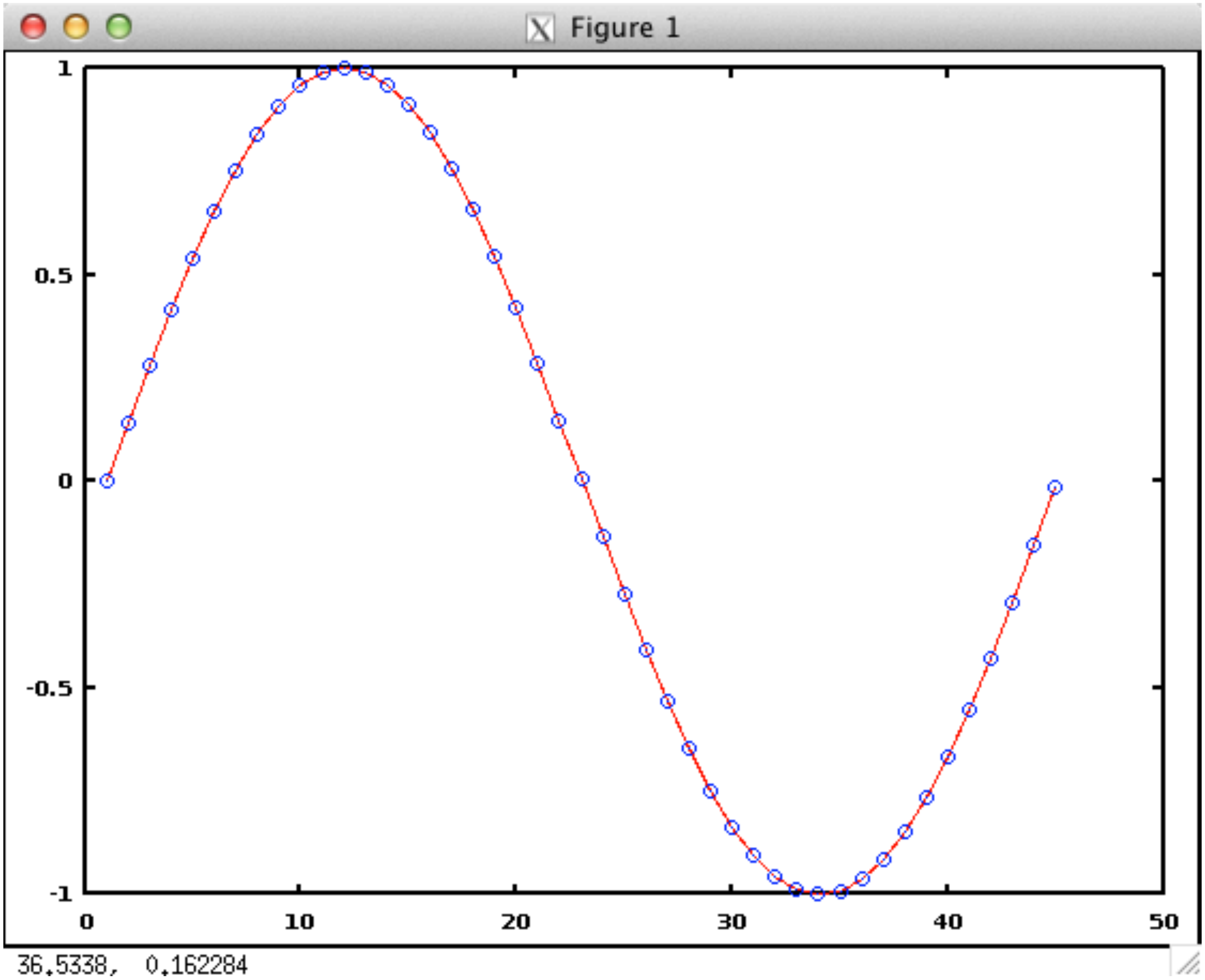
```
octave:262> clf; # clear figure (clear plots)
octave:263> plot( sine1000( 1:45 ) );
octave:264> hold
octave:265> plot( wi( 1:45 ), "r" );
```

It matches the pure sine wave so closely the blue line has disappeared.



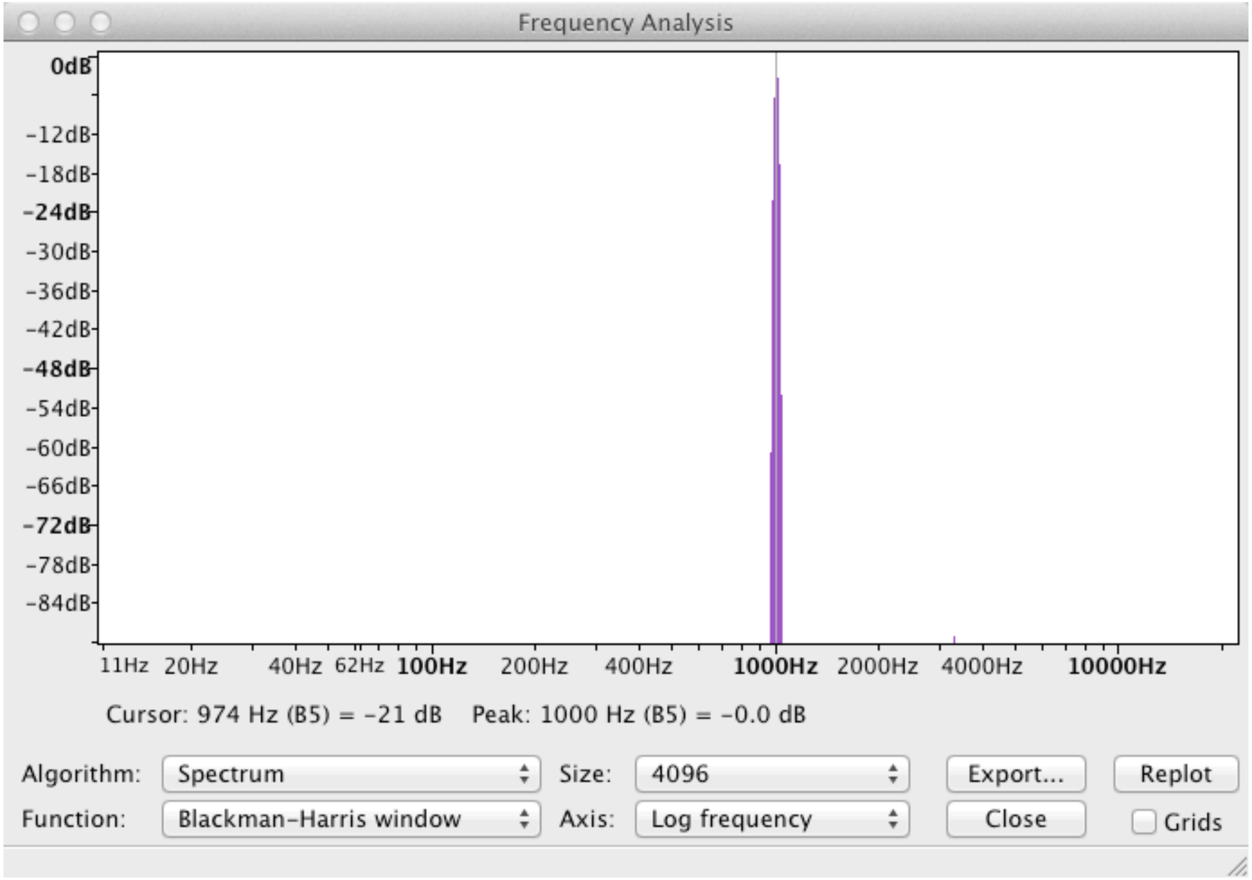
Type this.

```
octave:271> plot( sine1000( 1:45 ), "o" );
```



Plot the Spectrum in Audacity

The largest amplitude frequency appears at the 1000 Hz mark. The noise is gone. That means a small 128 sample table can produce a virtually pure sine wave using linear interpolation. That's the basic method of optimizing waveform storage requirements on many commercial hardware synthesizers and virtual software synthesizers.



Frequency Analysis
0dB
-12dB
-18dB
-24dB
-30dB
-36dB
-42dB
-48dB
-54dB
-60dB
-66dB
-72dB
-78dB
-84dB
11Hz 20Hz 40Hz 62Hz 100Hz 200Hz 400Hz 1000Hz 2000Hz 4000Hz 10000Hz
Cursor: 974 Hz (B5) = -21 dB Peak: 1000 Hz (B5) = -0.0 dB
Algorithm: Spectrum
Size: 4096
Export...
Replot
Function: Blackman-Harris window
Axis: Log frequency
Close
Grids