

MUSC 208 Winter 2014  
John Ellinger, Carleton College

## Lab 2 Octave: Octave Function Files

### Setup

Open /Applications/Octave

### The Working Directory

Type `pwd` on Unix did on Windows (followed by Return) at the Octave prompt to see the full path of Octave's working directory.

```
octave-3.4.0:92> pwd  
ans = /Users/je/m208
```

The full path name is `/Users/je/m208`. On the mac the top level of the hard drive is `/`. There is a folder at the top level called `Users`. Inside that folder is my home folder `je`. Inside that folder is the `m208` folder. I created this ahead of time to hold all the octave, chuck, and Pd files I've created for this class.

On windows the top level folder is `C:\`

When you work in the Terminal as we'll do in this class you should avoid using spaces in file or folder names. There are two common naming systems used by programmers.

1. underscores: `this_is_a_long_file_name`
2. camel case: `thisIsALongFileName`

It's important when you work with Octave that you know where Octave's working directory is, because that's where it will create and look for files.

### Set the Working Directory

You need to be familiar with three commands to move around in Octave: `cd`, `ls`, and `pwd`.

`cd` = Change Directory (folder)  
`ls` = LiSt files  
`pwd` = Print Working Directory.

## Specifying Mac and Windows Pathnames in Octave

The Mac uses forward slashes to separate directories. The top level of the hard drive is the first /. This example indicates that a folder named Volumes is at the top level of the hard drive. Inside Volumes is a folder named MC16 with another folder inside MC16 named m208.

```
/Volumes/MC16/m208
```

Windows uses \ to separate directories. In the Octave Terminal on windows a single \ is interpreted as an "escape" character so you need to separate directories with a double backslash \\. Also the top level hard drive is usually C:\ which becomes C:\\. The example given above when translated to the Octave windows terminal becomes:

```
C:\\Volumes\\MC16\\m208
```

The following picture illustrates file system navigation on a Mac. Comments are in red. The first line changes Octave's working directory to the m208 folder on my USB thumb drive.

```
octave-3.4.0:1> cd /Volumes/MC16/m208
octave-3.4.0:2> ls
Lab2
octave-3.4.0:3> cd Lab2
octave-3.4.0:4> ls
ck      m      pd
octave-3.4.0:5> cd m
octave-3.4.0:6> ls
add3.m          allzeros.m          oneAnd99zeros.m      rampRiseFall150.m
allones.m       noise100.m          onesAndMostlyZeros.m sine100samplePeriod.m
allrandom.m     onOff50.m           ramp0to100.m
octave-3.4.0:7> cd ck
error: ck: No such file or directory
octave-3.4.0:7> cd ../ck
octave-3.4.0:8> ls
EqualLoudnessTest.ck
octave-3.4.0:9> cd ../pd
octave-3.4.0:10> ls
AliasingDemo.pd
octave-3.4.0:11> pwd
ans = /Volumes/MC16/m208/Lab2/pd
octave-3.4.0:12> cd ../m
octave-3.4.0:13> pwd
ans = /Volumes/MC16/m208/Lab2/m
octave-3.4.0:14> ls
add3.m          allzeros.m          oneAnd99zeros.m      rampRiseFall150.m
allones.m       noise100.m          onesAndMostlyZeros.m sine100samplePeriod.m
allrandom.m     onOff50.m           ramp0to100.m
octave-3.4.0:15> # thats the working directory for my Octave files
octave-3.4.0:15>
```

**cd means Change Directory**  
**ls means vfiles**  
 there's one directory called Lab 2  
 cd to the Lab2 folder (directory)  
 list files  
 there are three folders  
 cd to the m folder (octave file suffix is .m)  
 list all files in the m folder  
 oneAnd99zeros.m  
 onesAndMostlyZeros.m  
 ramp0to100.m  
 rampRiseFall150.m  
 sine100samplePeriod.m  
 cd to the ck folder  
 oops, no ck folder in the m folder  
 ../ means move to the parent folder of the current folder  
 list files in ck  
 one file  
 cd to the pd folder  
 list files  
 one file  
 pwd means Print Working Directory  
 octave will now look for and save files in the pd directory  
 we really want to be in the m folder for Octave  
 prove it  
 yes  
 list files in the m directory  
 oneAnd99zeros.m  
 onesAndMostlyZeros.m  
 ramp0to100.m  
 rampRiseFall150.m  
 sine100samplePeriod.m

## Functions with a single return value

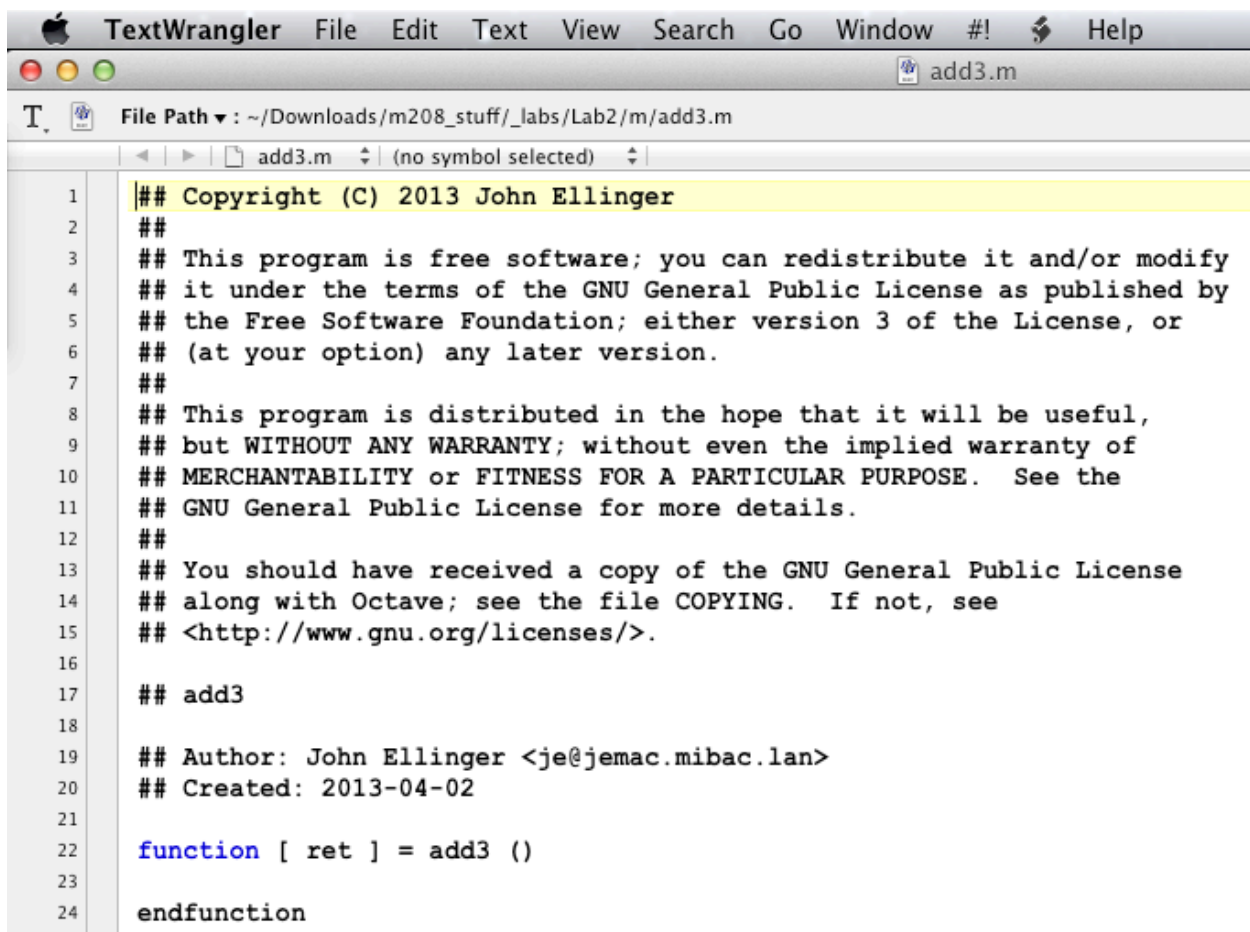
Octave functions are plain text files that contain code to do one specific thing. These functions can be used by in other Octave programs by simply calling the function name. That keeps the main program neater because you can encapsulate commonly used commands in separate files. Our first function will add any three numbers we tell it to.

### Create an Octave function file to add three numbers.

Enter this command after the > and type Return:

```
octave-3.4.0:3> edit add3.m
```

TextWrangler should open with this skeleton code already in place.

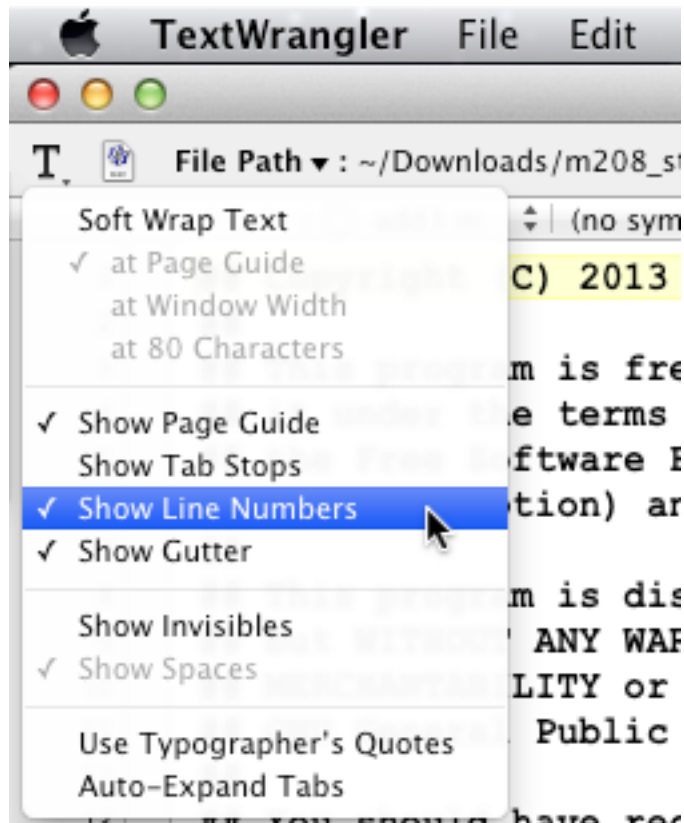


```

1  ## Copyright (C) 2013 John Ellinger
2  ##
3  ## This program is free software; you can redistribute it and/or modify
4  ## it under the terms of the GNU General Public License as published by
5  ## the Free Software Foundation; either version 3 of the License, or
6  ## (at your option) any later version.
7  ##
8  ## This program is distributed in the hope that it will be useful,
9  ## but WITHOUT ANY WARRANTY; without even the implied warranty of
10 ## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
11 ## GNU General Public License for more details.
12 ##
13 ## You should have received a copy of the GNU General Public License
14 ## along with Octave; see the file COPYING. If not, see
15 ## <http://www.gnu.org/licenses/>.
16
17 ## add3
18
19 ## Author: John Ellinger <je@jemac.mibac.lan>
20 ## Created: 2013-04-02
21
22 function [ ret ] = add3 ()
23
24 endfunction

```

The line numbers you see down the left side of the window can be turned on using the "T" popup menu just under the Red close button at the top left of the window.



Octave is very strict about matching the file name with the function name, they must be the same.

```

1  ## Copyright (C) 2013 John Ellinger
2  ##
3  ## This program is free software; you can redistribute it and/or modify
4  ## it under the terms of the GNU General Public License as published by
5  ## the Free Software Foundation; either version 3 of the License, or
6  ## (at your option) any later version.
7  ##
8  ## This program is distributed in the hope that it will be useful,
9  ## but WITHOUT ANY WARRANTY; without even the implied warranty of
10 ## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
11 ## GNU General Public License for more details.
12 ##
13 ## You should have received a copy of the GNU General Public License
14 ## along with Octave; see the file COPYING. If not, see
15 ## <http://www.gnu.org/licenses/>.
16
17 ## add3
18
19 ## Author: John Ellinger <je@jemac.mibac.lan>
20 ## Created: 2013-04-02
21
22 function [ ret ] = add3 ()
23
24 endfunction

```

1. The file name at the top of the window.
2. `## add3`. Comments you add below this line will appear when you type "help add3" at the Octave prompt.
3. `function [ ret ] = add3 ()`. You'll add your code between this line and the `endfunction` statement.

In Octave you would add three numbers like this:

```
octave-3.4.0:4> 133 + 6537 + 440
ans = 7110
```

or

```
octave-3.4.0:5> a = 133;
octave-3.4.0:6> b = 6537;
octave-3.4.0:7> c = 440;
octave-3.4.0:8> a+b+c
ans = 7110
```

We'll use the second approach to create the code.

## Create the code

Modify the code to read:

```
22 function ret = add3 (a, b, c)
23     [ ret ] = a + b + c;
24 endfunction
```

Note: ret, a, b, and c are made up variable names that will be used in the body of the function. We could also have written:

```
22 function theSumOfThreeNumbersIs = add3 (number1, number2, number3)
23     theSumOfThreeNumbersIs = number1 + number2 + number3;
24 endfunction
```

In general you want to use variable names that help make the code self documenting. In this simple example a, b, and c work fine and require less typing.

Now add the help message:

```
15 ## <http://www.gnu.org/licenses/>.
16
17 ## ret = add3 (a, b, c)
18 ## adds the three numbers a b c and returns the result
19
20 ## Author: John Ellinger <je@jemac.mibac.lan>
21 ## Created: 2013-04-02
22 function ret = add3 (a, b, c)
23     [ ret ] = a + b + c;
24 endfunction
```

Save the file.

## Testing

Return to the Octave prompt and ask for help:

```
octave-3.4.0:9> help add3
`add3' is a function from the file /Users/jc/Downloads/m208_stuff/_labs/Lab2/m/add3.m

add3

Additional help for built-in functions and operators is
available in the on-line version of the manual. Use the command
`doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at http://www.octave.org and via the help@octave.org
mailing list.
octave-3.4.0:10> |
```

Let's use the function.

```
octave-3.4.0:14> add3( 133, 6537, 440 );
octave-3.4.0:15> add3( 133, 6537, 440 )
ans = 7110
```

Remember lines with an ending semicolon do not display output.

Try this:

```
octave-3.4.0:16> x = add3( 15, 45, 67 );
octave-3.4.0:17> y = add3( 1.78, 22, -12 );
octave-3.4.0:18> z = add3( 1.0, .55, .019 );
octave-3.4.0:19> x
x = 127
octave-3.4.0:20> y
y = 11.780
octave-3.4.0:21> z
z = 1.5690
octave-3.4.0:22> x+y+z
ans = 140.35
octave-3.4.0:23> average = (x+y+z)/9
average = 15.594
```

## Error checking

As long as we call `add3` with three numbers everything works. It's an error if we have less than three. If we have more than three numbers the function works by using the first three.



```
octave-3.4.0:24> add3( 133, 6537 )
error: `c' undefined near line 23 column 16
error: called from:
error:   /Users/je/Downloads/m208_stuff/_labs/Lab2/m/add3.m at line 23, column 6
```

We'll introduce error checking next time.

## Functions with multiple return values

**Create an Octave function file to return the sum, product, and mean of three numbers.**

Enter this command after the > and type Return:

```
octave-3.4.0:3> edit sumProdMean3.m
```

```
15  ## <http://www.gnu.org/licenses/>.
16
17  ## sumProdMean3
18  ## sumProdMean3 (a, b, c) returns the sum, product, and mean of three numbers
19
20  ## Author: John Ellinger <je@jemac.mibac.lan>
21  ## Created: 2013-04-02
22
23  function [ s, p, m ] = sumProdMean3 (a, b, c)
24      s = a+b+c;
25      p = a*b*c;
26      m = s/3;
27  endfunction
```

Test help:

```
octave-3.4.0:40> help sumProdMean3
`sumProdMean3' is a function from the file /Users/je/Downloads/m208_stuff/_labs.
an3.m

sumProdMean3
sumProdMean3 (a, b, c) returns the sum, product, and mean of three numbers
```

Test the function:



```

octave-3.4.0:41> [s,p,m] = sumProdMean3(10, 15, 30)
s = 55
p = 4500
m = 18.333
octave-3.4.0:42> [s,p,m] = sumProdMean3(10, 15, 30);
octave-3.4.0:43> s
s = 55
octave-3.4.0:44> p
p = 4500
octave-3.4.0:45> m
m = 18.333

```

## Lab 2 - Octave: Periodic Signals

### Question 1: What does a waveform sound like if every sample is 0?

Open /Applications/Octave

At the Octave prompt type this followed by return. From now on I'll assume you know that you need to type return to execute the command.

```
octave-3.4.0:125> edit allzeros.m
```

A TextWrangler window will open titled allzeros.m.

Fill in the function body.

One method to create 44100 zeros would be:

```

SR = 44100;
n = 1:SR;
nT = n .* 0;
ret = nT;

```

A more compact version could be written as:

```
ret = [ 1:44100 ] .* 0;
```

Another way to do it is:

```
ret = zeros(1, 44100);
```

To get help on the zeros function, enter:  
 help zeros

Here's an example that shows what happens.

```
octave-3.4.0:22> # square matrix
octave-3.4.0:22> zeros(4)
ans =

    0    0    0    0
    0    0    0    0
    0    0    0    0
    0    0    0    0

octave-3.4.0:23> # 1 row, 4 columns
octave-3.4.0:23> zeros(1,4)
ans =

    0    0    0    0

octave-3.4.0:24> # 4 rows, 1 column
octave-3.4.0:24> zeros(4,1)
ans =

    0
    0
    0
    0
```

Now execute these commands at the Octave prompt.

```
octave-3.4.0:10> wav = allzeros;
octave-3.4.0:11> wav( 1:20 )
ans =

    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0

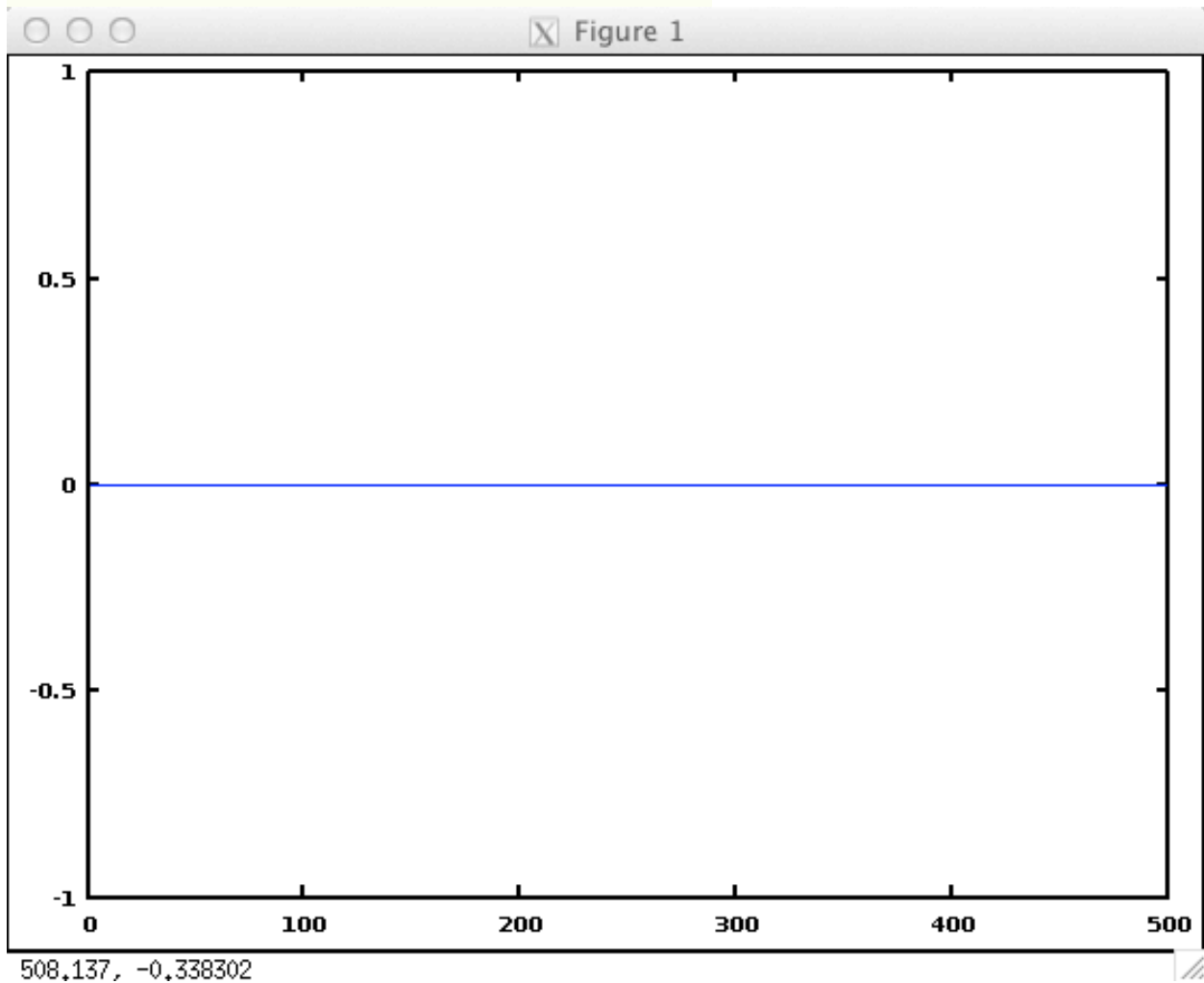
octave-3.4.0:12> playsamples(wav);
```

What did you hear? The answer should be nothing.

A steady stream of zeros will not make the speaker membrane move in and out, so no sound wave is produced.

Plot it.

```
octave:6> plot( wav( 1 : 500 ) );
```



## Question 2. What does a waveform sound like if every sample is 1?

At the Octave prompt execute this:  
edit allones.m

Change the code you used in Question 1 to use ones instead of zeros. You could use:  
ret = ones(1, 44100);

Now execute these commands:

```

octave-3.4.0:26> wav = allones;
octave-3.4.0:27> wav(1:20)
ans =

    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1

octave-3.4.0:28> playsamples(wav);

```

What did you hear? The answer should be two clicks.

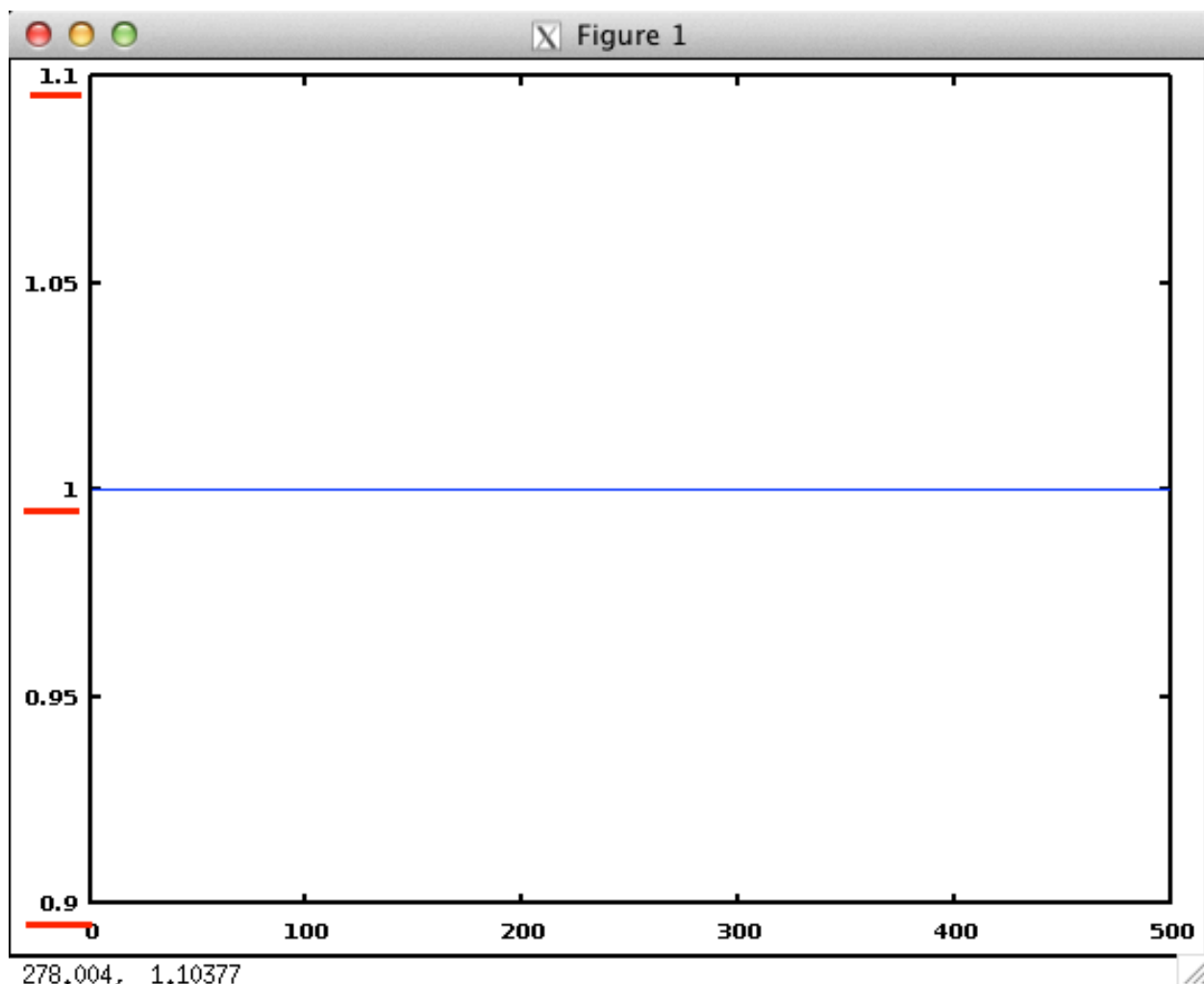
The first 1 pushed the speaker membrane all the way out starting a positive pressure wave of compressed air. The next 44,099 ones held the speaker membrane in its out position. When the wave ended, the speaker membrane returned inwards creating a negative pressure wave. The sudden discontinuities of the speaker membrane were heard as clicks.

Plot it.

```

octave:17> plot( wav( 1 : 500 ) );

```



Look at the Y axis limits. They range from 0.9 to 1.1 because of Octave autoscaling.

Let's change the Y axis limits. First look for help about axis using the lookfor command.

```
octave:18> lookfor axis
axis          Set axis limits for plots.
caxis         Set color axis limits for plots.
feather       Plot the '(U, V)' components of a vector field emanating from equidistant p
oints on the x-axis.
gca           Return a handle to the current axis object.
loglogerr     Produce two-dimensional plots on double logarithm axis with errorbars.
semilogx      Produce a two-dimensional plot using a logarithmic scale for the X axis.
semilogxerr   Produce two-dimensional plots using a logarithmic scale for the X axis and
errorbars at each data point.
semilogy      Produce a two-dimensional plot using a logarithmic scale for the Y axis.
semilogyerr   Produce two-dimensional plots using a logarithmic scale for the Y axis and
errorbars at each data point.
xlabel        Specify x-, y-, or z-axis labels for the current axis.
xlim          Get or set the limits of the x-axis of the current plot.
ylim          Get or set the limits of the y-axis of the current plot.
zlim          Get or set the limits of the z-axis of the current plot.
datetick      Add date formatted tick labels to an axis.
auplot        Plot the waveform data, displaying time on the X axis.
octave:19>
```

Then get specific help on the axis command.

```
octave:19> help axis
'axis' is a function from the file /usr/local/Cellar/octave/3.6.4/share/octav
e/3.6.4/m/plot/axis.m

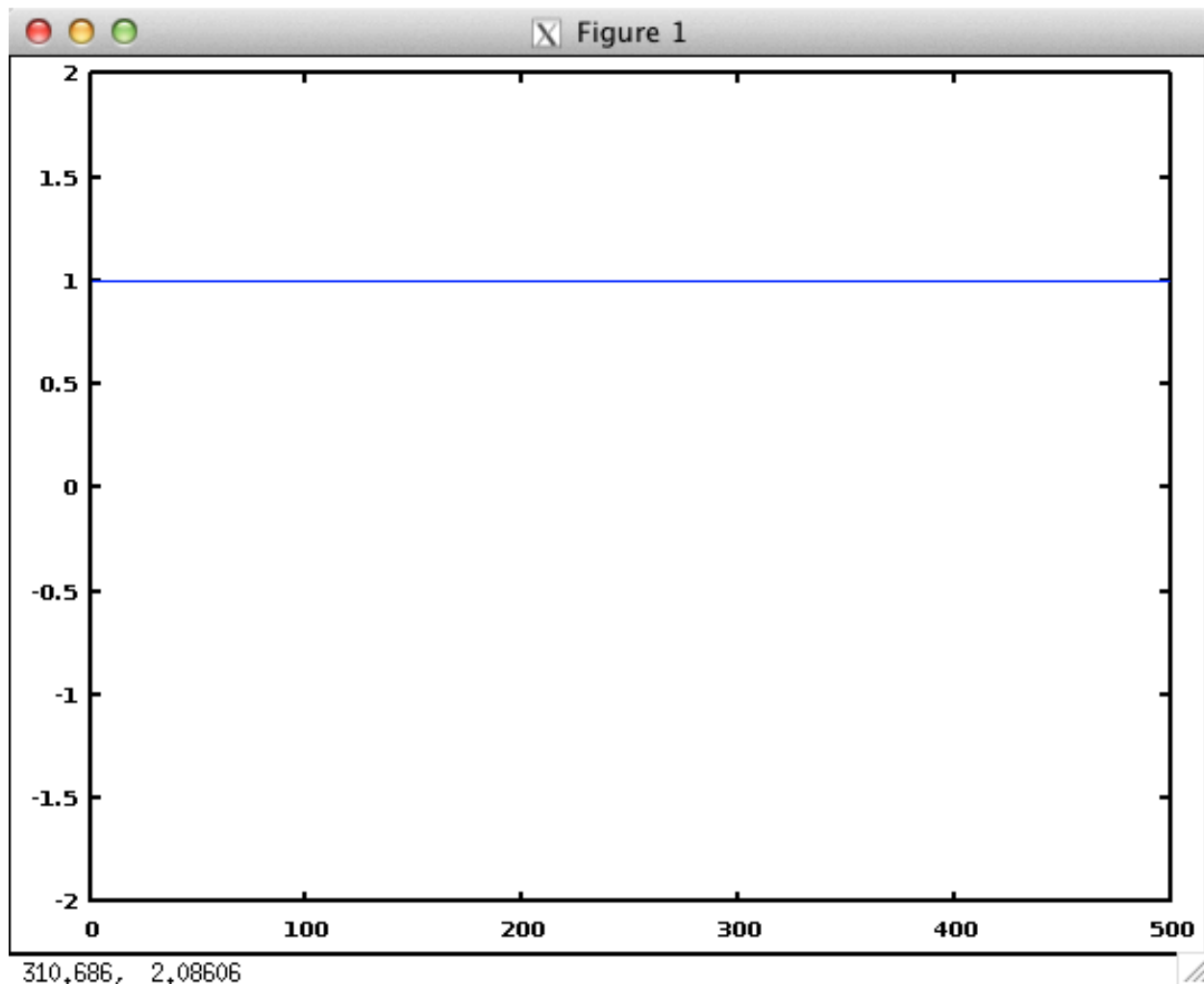
-- Function File: axis ()
-- Function File: axis ([X lo X hi])
-- Function File: axis ([X_lo X_hi Y_lo Y_hi])
-- Function File: axis ([X_lo X_hi Y_lo Y_hi Z_lo Z_hi])
-- Function File: axis (OPTION)
-- Function File: axis (... , OPTION)
-- Function File: axis (H, ...)
-- Function File: LIMITS = axis ()
    Set axis limits for plots.

    The argument LIMITS should be a 2-, 4-, or 6-element vector. The
```

We'll use the third one. Execute this.

```
octave:20> axis( [ 0 500 -2 2 ] );
```

The Y axis should immediately change.



### Question 3. What does a waveform sound like when all samples are random?

At the Octave prompt execute this:  
`edit allrandom.m`

Change the code you used in Question 1 or 2 to use `rand`  
`ret = rand(1, 44100);`

Now execute these commands:

```

octave-3.4.0:127> wav = allrandom;
octave-3.4.0:128> length(wav)
ans = 44100
octave-3.4.0:129> wav(1:100)
ans =

Columns 1 through 8:
    0.4265862    0.0499383    0.0045888    0.8633293    0.6396447    0.0777699    0.9059608    0.1986397
Columns 9 through 16:
    0.2611531    0.6802464    0.5863600    0.4936277    0.1083623    0.0657752    0.5215081    0.6371048
Columns 17 through 24:
    0.4711225    0.7512032    0.9367746    0.1620678    0.3231622    0.1965898    0.1670319    0.4215921
Columns 25 through 32:
    0.6413130    0.1480023    0.7464444    0.0905297    0.9949723    0.7192969    0.2282198    0.8893306
Columns 33 through 40:
    0.0345513    0.0416174    0.0959472    0.6217750    0.4077891    0.1024974    0.8277974    0.3054618
Columns 41 through 48:
    0.8330227    0.8292638    0.6972918    0.3603887    0.5115804    0.5426709    0.2585607    0.2002402
Columns 49 through 56:
    0.5916115    0.8817492    0.4428844    0.0256947    0.8097156    0.8761132    0.7829345    0.6371281
Columns 57 through 64:
    0.1305806    0.7454377    0.8417498    0.1798228    0.4927523    0.2162750    0.9465306    0.8891874
Columns 65 through 72:
    0.6076831    0.9393852    0.6404988    0.4494569    0.8503897    0.4795021    0.6215608    0.6492570
Columns 73 through 80:
    0.5238225    0.9871107    0.8945542    0.9122400    0.4180767    0.4991818    0.5332238    0.1384914
Columns 81 through 88:
    0.1796142    0.9863986    0.7131054    0.4812540    0.5881990    0.5045967    0.3397863    0.6460222
Columns 89 through 96:
    0.7567690    0.2792097    0.7780263    0.2350570    0.8505868    0.7084707    0.3870974    0.7248190
Columns 97 through 100:
    0.1381305    0.9786735    0.0649410    0.8557518

octave-3.4.0:130> playsamples(wav);

```

What did you hear? The answer sounds like noise or the static between radio stations.

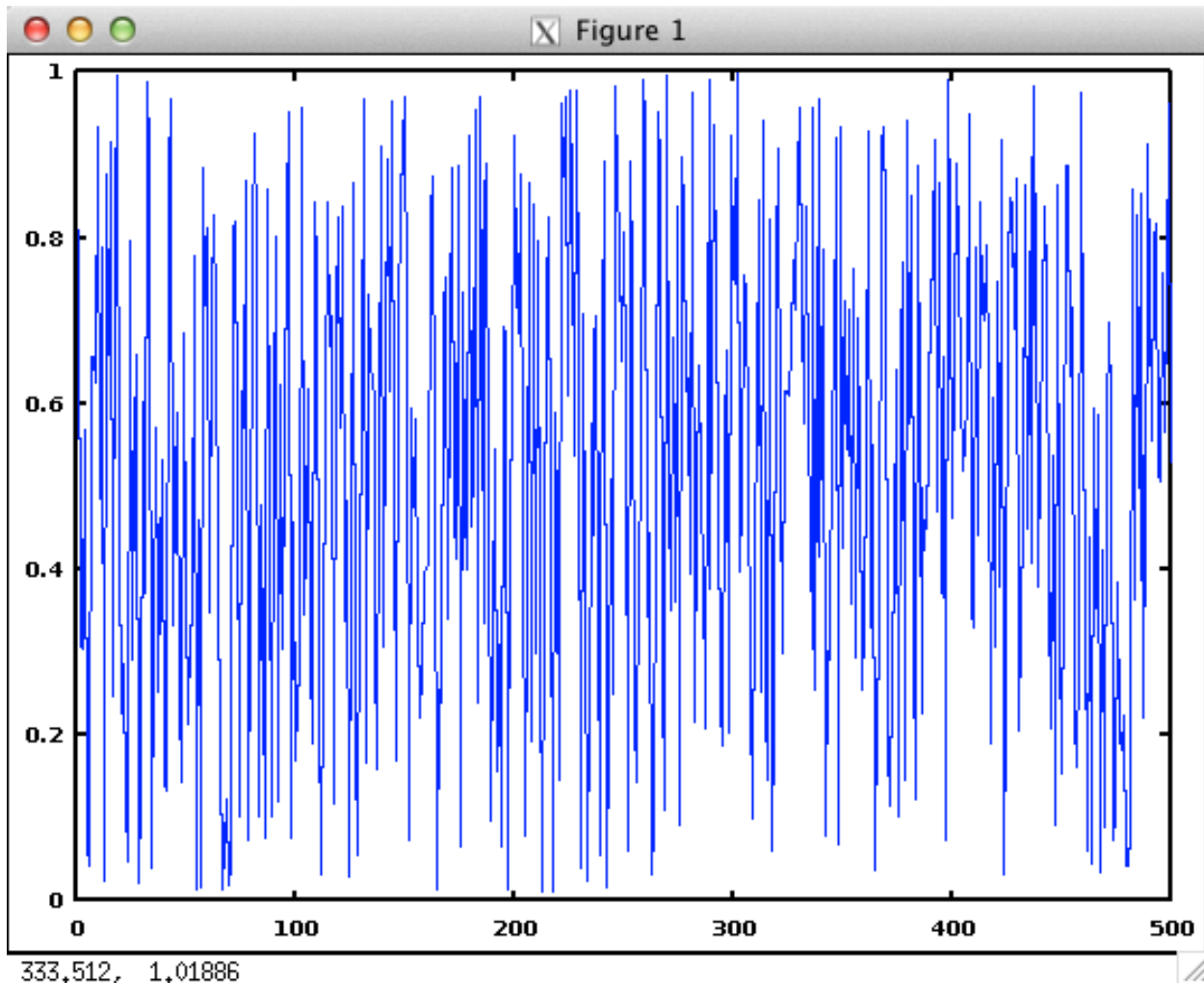
Plot it.

```

octave:22> plot( wav( 1 : 500 ) );

```





#### Question 4. What does a waveform sound like when every 100th sample is a 1 and all others are zero?

At the Octave prompt execute this:

```
edit oneAnd99zeros.m
```

##### Outline of steps

1. create a series of 100 zeros
2. set the first element to 1
3. repeat it 441 times to create one second of samples
4. return the 44100 samples at the end of the function

You can use the Octave command `repmat` to make multiple copies of a sequence (array, vector, one dimensional matrix)

**Read the octave help for `repmat`**

Here's some examples

```
octave-3.4.0:74> array = 1:4
array =
```

```
    1    2    3    4
```

```
octave-3.4.0:75> repmat(array, 1, 4 )
ans =
```

```
    1    2    3    4    1    2    3    4    1    2    3    4    1    2    3    4
```

```
octave-3.4.0:76> repmat(array, [1 4] )
ans =
```

```
    1    2    3    4    1    2    3    4    1    2    3    4    1    2    3    4
```

```
octave-3.4.0:77> repmat(array, 4, 1 )
ans =
```

```
    1    2    3    4
    1    2    3    4
    1    2    3    4
    1    2    3    4
```

```
octave-3.4.0:78> repmat(array, [4 1] )
ans =
```

```
    1    2    3    4
    1    2    3    4
    1    2    3    4
    1    2    3    4
```

When you've finished coding and saving the oneAnd99Zeros.m file, execute these commands:

```

octave-3.4.0:78> wav = oneAnd99zeros;
octave-3.4.0:79> length(wav)
ans = 44100
octave-3.4.0:80> wav(1:110)
ans =

Columns 1 through 24:
   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Columns 25 through 48:
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Columns 49 through 72:
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Columns 73 through 96:
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Columns 97 through 110:
   0   0   0   0   1   0   0   0   0   0   0   0   0   0

octave-3.4.0:81> playsamples( wav );

```

What did you hear? The answer is a sound with a pitch of 441 Hz.

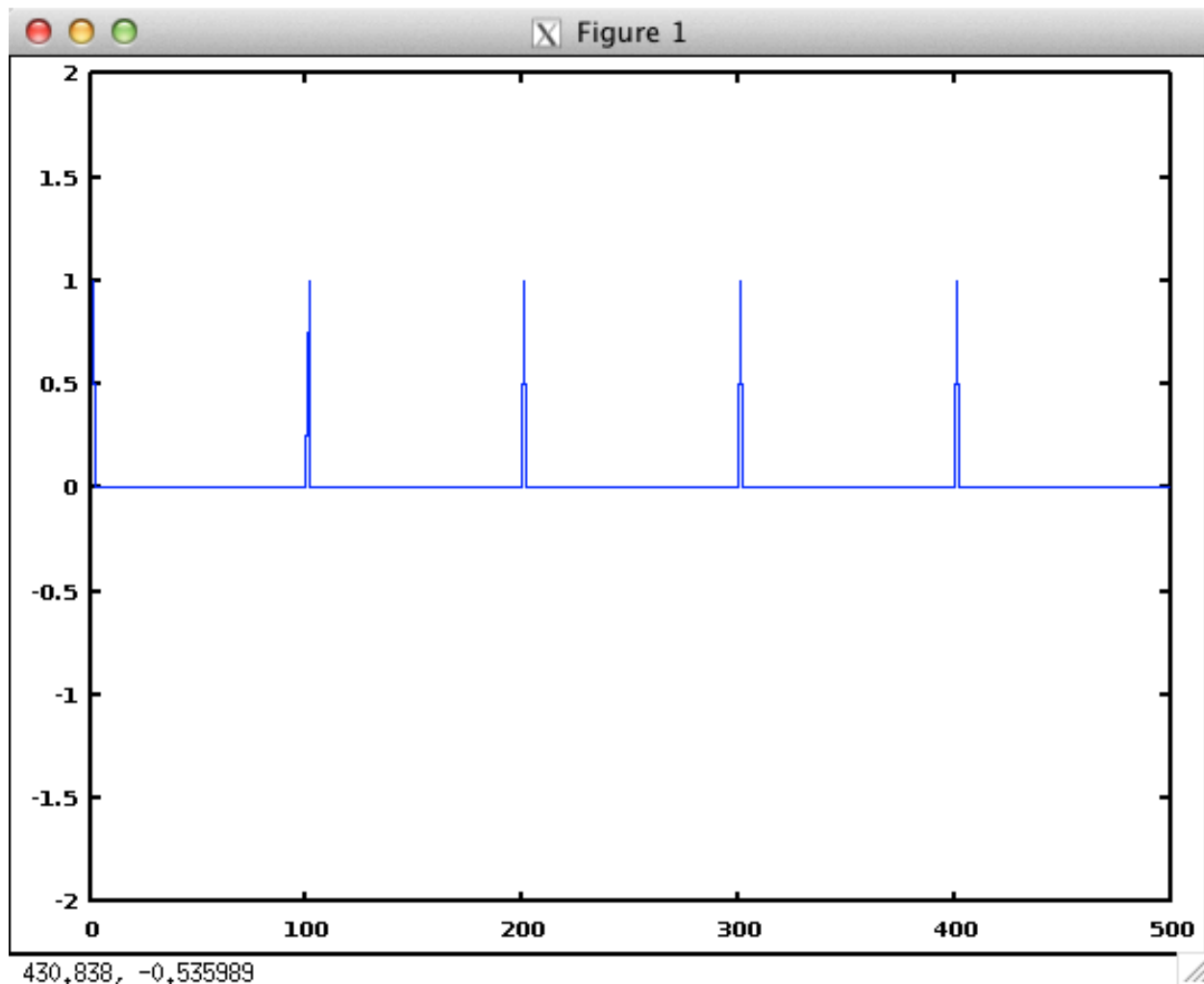
The speaker membrane moves out on every 1 starting the positive pressure wave followed immediately 99 zeros that create and a negative pressure wave. The positive pressure is periodic 441 times in one second creating the pitch.

Plot it.

```

octave:30> plot( wav( 1 : 500 ) );
octave:31> axis( [ 0 500 -2 2 ] );

```



This is a basic synthesizer waveform called a pulse wave.

**Question 5. What does a waveform sound like when the first 50 samples are 1 and the second 50 are 0?**

At the Octave prompt execute this:  
`edit onOff50.m`

**Outline**

1. Create 50 1's.
2. Create 50 0's.
3. join them together
4. repeat 441 times.

Hint:

```

octave-3.4.0:18> a = [ 1, 1, 1, 1 ]
a =

    1    1    1    1

octave-3.4.0:19> b = [ 0 0 0 0 ]
b =

    0    0    0    0

octave-3.4.0:20> [ a b ]
ans =

    1    1    1    1    0    0    0    0

```

When you've finished coding and saving the onOff50.m file, execute these commands:

```

octave-3.4.0:8> wav = onOff50;
octave-3.4.0:9> length(wav)
ans = 44100
octave-3.4.0:10> wav(1:110)
ans =

Columns 1 through 24:

    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1

Columns 25 through 48:

    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1

Columns 49 through 72:

    1    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0

Columns 73 through 96:

    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0

Columns 97 through 110:

    0    0    0    0    1    1    1    1    1    1    1    1    1    1

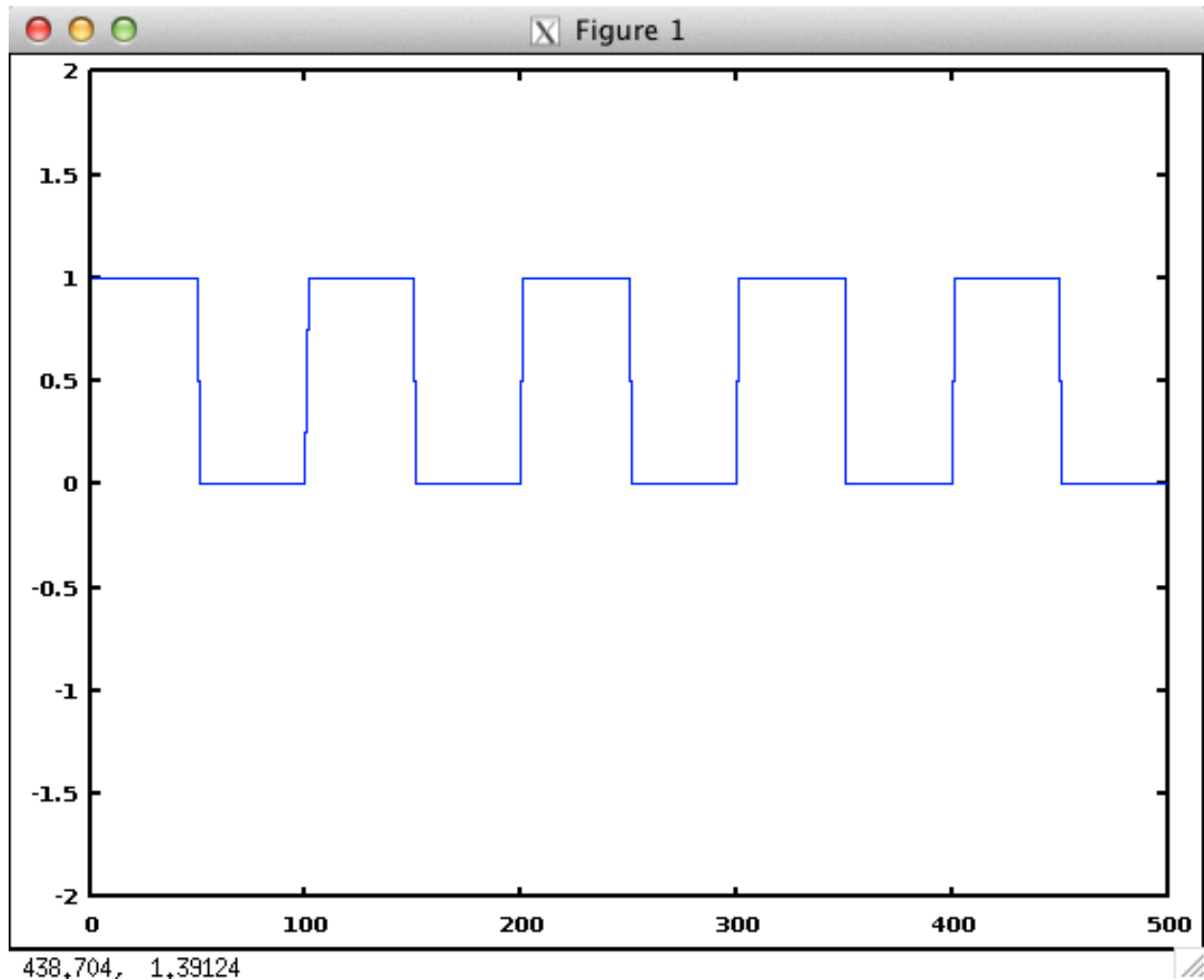
octave-3.4.0:11> playsamples(wav);

```

What did you hear? The answer is a sound with a pitch of 441 Hz.

Plot it.

```
octave:27> plot( wav( 1 : 500 ) );
octave:28> axis( [ 0 500 -2 2 ] );
```



This is a basic synthesizer waveform called a square wave.

### Question 6. What does a waveform sound like if every 100 samples rise uniformly in amplitude from 0 to 1?

At the Octave prompt execute this:  
`edit ramp0to100.m;`

#### Outline

1. Define delta as 0.01;
1. Create the first 100 samples.  $n = 0:\text{delta}:1-\text{delta}$ . Because we're starting from zero, 100

samples end at 99.  
2. repeat 441 times.

Hint: If you create a vector (array, sequence, list) with 2 semicolons, the middle number is the increment between the beginning and end.

```
octave-3.4.0:44> ramp = 0:.1:1
ramp =

Columns 1 through 8:
    0.0000    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000
Columns 9 through 11:
    0.8000    0.9000    1.0000
```

Now execute these commands. format compact uses less whitespace in the display.

```
octave-3.4.0:162> wav = ramp0to100;
octave-3.4.0:163> length(wav)
ans = 44100
octave-3.4.0:164> format compact;
octave-3.4.0:165> wav(1:100)
ans =

Columns 1 through 9:
    0.0000    0.0100    0.0200    0.0300    0.0400    0.0500    0.0600    0.0700    0.0800
Columns 10 through 18:
    0.0900    0.1000    0.1100    0.1200    0.1300    0.1400    0.1500    0.1600    0.1700
Columns 19 through 27:
    0.1800    0.1900    0.2000    0.2100    0.2200    0.2300    0.2400    0.2500    0.2600
Columns 28 through 36:
    0.2700    0.2800    0.2900    0.3000    0.3100    0.3200    0.3300    0.3400    0.3500
Columns 37 through 45:
    0.3600    0.3700    0.3800    0.3900    0.4000    0.4100    0.4200    0.4300    0.4400
Columns 46 through 54:
    0.4500    0.4600    0.4700    0.4800    0.4900    0.5000    0.5100    0.5200    0.5300
Columns 55 through 63:
    0.5400    0.5500    0.5600    0.5700    0.5800    0.5900    0.6000    0.6100    0.6200
Columns 64 through 72:
    0.6300    0.6400    0.6500    0.6600    0.6700    0.6800    0.6900    0.7000    0.7100
Columns 73 through 81:
    0.7200    0.7300    0.7400    0.7500    0.7600    0.7700    0.7800    0.7900    0.8000
Columns 82 through 90:
    0.8100    0.8200    0.8300    0.8400    0.8500    0.8600    0.8700    0.8800    0.8900
Columns 91 through 99:
    0.9000    0.9100    0.9200    0.9300    0.9400    0.9500    0.9600    0.9700    0.9800
Column 100:
    0.9900

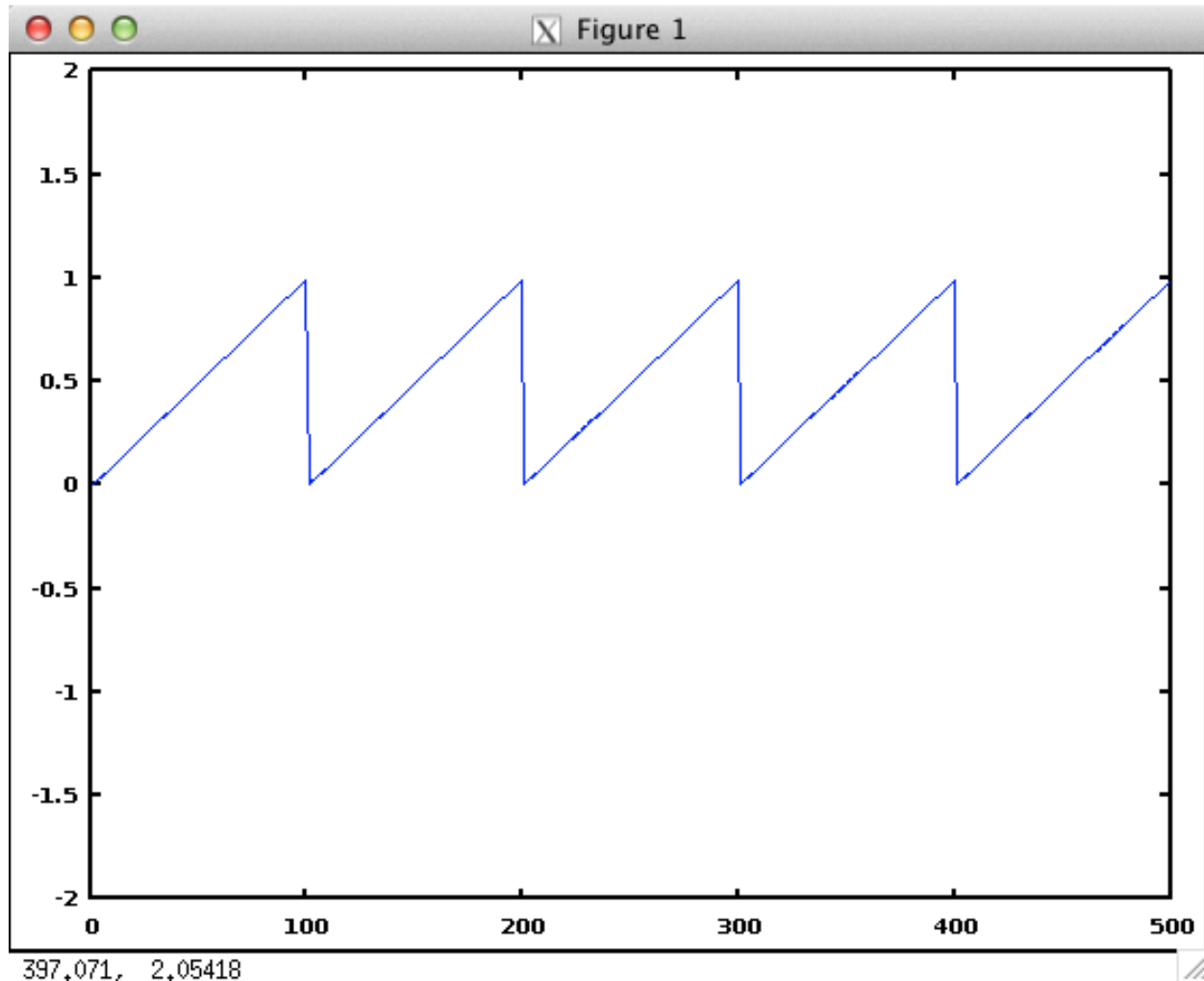
octave-3.4.0:166> playsamples(wav);
```

What did you hear? The answer is a sound with a pitch of 441 Hz with a different timbre.



Plot it.

```
octave:33> plot( wav( 1 : 500 ) );
octave:34> axis( [ 0 500 -2 2 ] );
```



This is a basic synthesizer waveform called a sawtooth wave.

**Question 7. What does a waveform sound like if every 50 samples rise uniformly in amplitude from 0 to 1 and the next 50 fall back to 0?**

At the Octave prompt execute this:  
`edit rampRiseFall50.m;`

**Outline**

1. Calculate the increment value to go from 0.0 to 1.0 in 50 samples. Call it delta.
2. Create the first fifty samples 0:delta:1
3. Create the remaining 50 samples. You'll need to go backwards using -delta. Don't repeat the 1 on the way down.

Now execute these commands:

```
octave-3.4.0:71> wav = rampRiseFall150;
octave-3.4.0:72> length(wav)
ans = 44100
octave-3.4.0:73> wav(1:100)
ans =

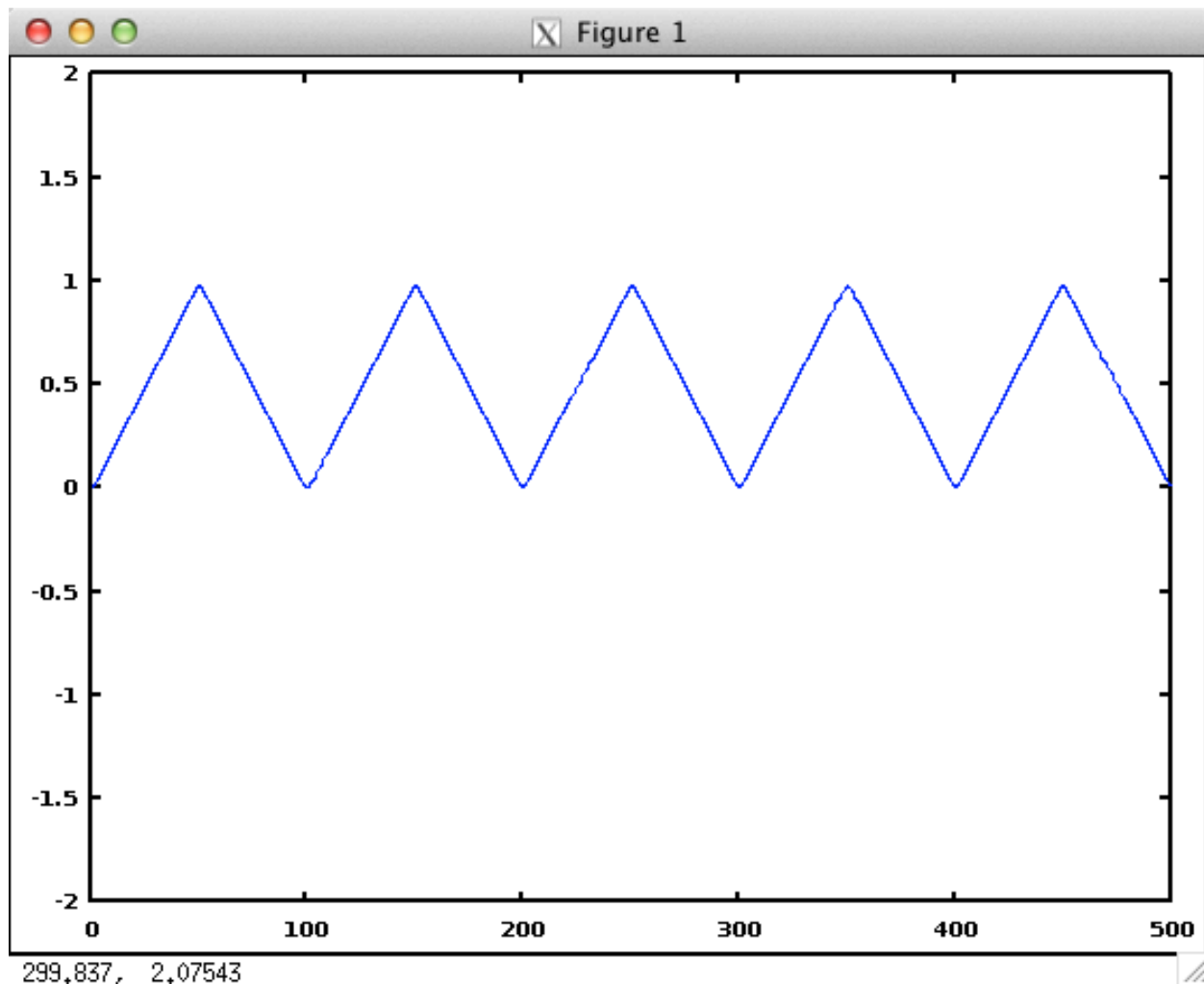
Columns 1 through 9:
    0.00000    0.02000    0.04000    0.06000    0.08000    0.10000    0.12000    0.14000    0.16000
Columns 10 through 18:
    0.18000    0.20000    0.22000    0.24000    0.26000    0.28000    0.30000    0.32000    0.34000
Columns 19 through 27:
    0.36000    0.38000    0.40000    0.42000    0.44000    0.46000    0.48000    0.50000    0.52000
Columns 28 through 36:
    0.54000    0.56000    0.58000    0.60000    0.62000    0.64000    0.66000    0.68000    0.70000
Columns 37 through 45:
    0.72000    0.74000    0.76000    0.78000    0.80000    0.82000    0.84000    0.86000    0.88000
Columns 46 through 54:
    0.90000    0.92000    0.94000    0.96000    0.98000    0.98000    0.96000    0.94000    0.92000
Columns 55 through 63:
    0.90000    0.88000    0.86000    0.84000    0.82000    0.80000    0.78000    0.76000    0.74000
Columns 64 through 72:
    0.72000    0.70000    0.68000    0.66000    0.64000    0.62000    0.60000    0.58000    0.56000
Columns 73 through 81:
    0.54000    0.52000    0.50000    0.48000    0.46000    0.44000    0.42000    0.40000    0.38000
Columns 82 through 90:
    0.36000    0.34000    0.32000    0.30000    0.28000    0.26000    0.24000    0.22000    0.20000
Columns 91 through 99:
    0.18000    0.16000    0.14000    0.12000    0.10000    0.08000    0.06000    0.04000    0.02000
Column 100:
    0.00000

octave-3.4.0:74> playsamples(wav)
```

What did you hear? The answer is a sound with a pitch of 441 Hz with a different timbre.

Plot it.

```
octave:36> plot( wav( 1 : 500 ) );
octave:37> axis( [ 0 500 -2 2 ] );
```



This is a basic synthesizer waveform called a triangle wave.

### Question 8. What does a sine wave with a period of 100 samples sound like?

At the Octave prompt execute this:  
`edit sine100samplePeriod.m;`

#### Outline

1. Define  $TWOPI = 2 * \pi$ ;
2. The period of one cycle of a sine wave is  $TWO\_PI$ ;
3. Define delta as the increment value necessary to divide  $TWO\_PI$  into 100 parts.
3. Create series `n = 0 : delta : TWO_PI - delta`; # 0-99 is 100 steps.
4. One period = `sin(n)`;
5. Repeat period 441 times.

Execute this code:

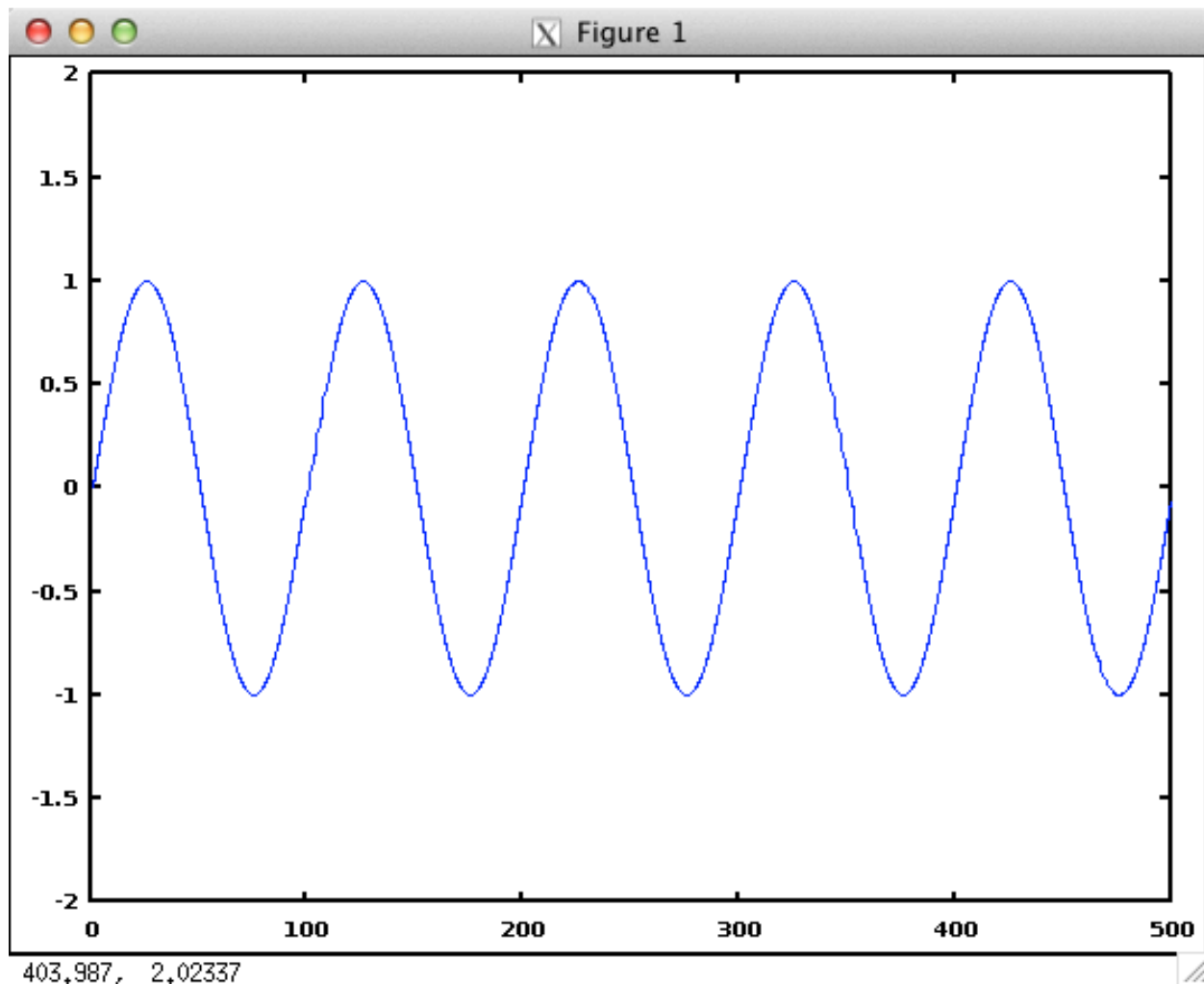
```
octave-3.4.0:152> wav = sine100samplePeriod;
octave-3.4.0:153> length(wav)
ans = 44100
octave-3.4.0:154> wav(1:100)
ans =

Columns 1 through 9:
    0.00000    0.06279    0.12533    0.18738    0.24869    0.30902    0.36812    0.42578    0.48175
Columns 10 through 18:
    0.53583    0.58779    0.63742    0.68455    0.72897    0.77051    0.80902    0.84433    0.87631
Columns 19 through 27:
    0.90483    0.92978    0.95106    0.96858    0.98229    0.99211    0.99803    1.00000    0.99803
Columns 28 through 36:
    0.99211    0.98229    0.96858    0.95106    0.92978    0.90483    0.87631    0.84433    0.80902
Columns 37 through 45:
    0.77051    0.72897    0.68455    0.63742    0.58779    0.53583    0.48175    0.42578    0.36812
Columns 46 through 54:
    0.30902    0.24869    0.18738    0.12533    0.06279   -0.00000   -0.06279   -0.12533   -0.18738
Columns 55 through 63:
   -0.24869   -0.30902   -0.36812   -0.42578   -0.48175   -0.53583   -0.58779   -0.63742   -0.68455
Columns 64 through 72:
   -0.72897   -0.77051   -0.80902   -0.84433   -0.87631   -0.90483   -0.92978   -0.95106   -0.96858
Columns 73 through 81:
   -0.98229   -0.99211   -0.99803   -1.00000   -0.99803   -0.99211   -0.98229   -0.96858   -0.95106
Columns 82 through 90:
   -0.92978   -0.90483   -0.87631   -0.84433   -0.80902   -0.77051   -0.72897   -0.68455   -0.63742
Columns 91 through 99:
   -0.58779   -0.53583   -0.48175   -0.42578   -0.36812   -0.30902   -0.24869   -0.18738   -0.12533
Column 100:
   -0.06279
```

What did you hear? The answer is a sound with a pitch of 441 Hz with a different timbre.

Plot it.

```
octave:39> plot( wav( 1 : 500 ) );
octave:40> axis( [ 0 500 -2 2 ] );
```



This is a basic synthesizer waveform called a sine wave.

### Question 9. What do 100 random samples sound like when they are repeated over and over?

At the Octave prompt execute this:

```
edit noise100.m;
```

#### Outline

You can do it in one line:

```
function [ ret ] = noise100 ()
    ret = repmat( rand( 1, 100 ), 1, 441 );
endfunction
```

Execute this code:

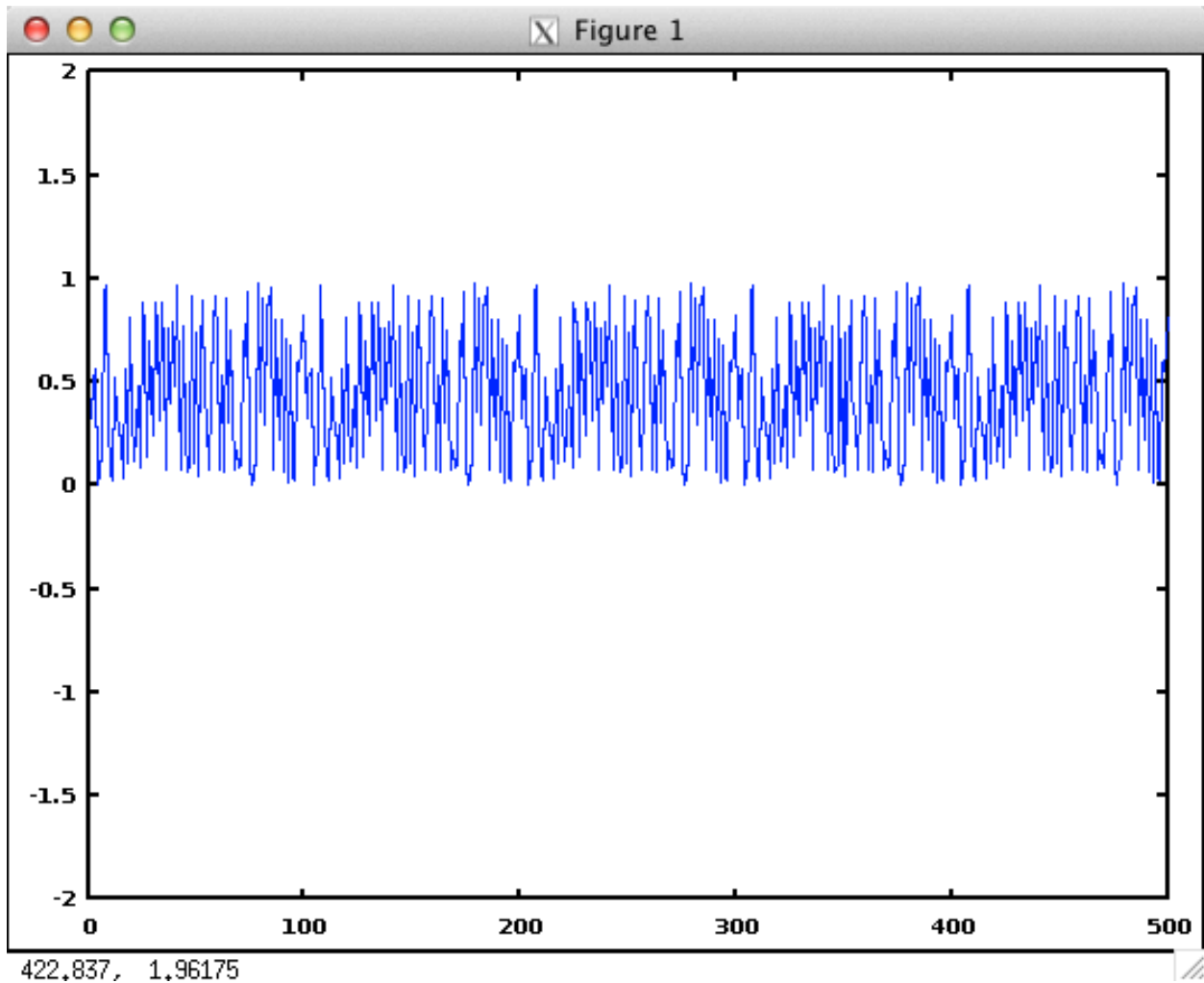
```
octave-3.4.0:7> wav = noisel100;
octave-3.4.0:8> length(wav)
ans = 44100
octave-3.4.0:9> wav(1:110)
ans =
```

Columns 1 through 8:							
0.9935502	0.2737149	0.1919758	0.9615369	0.1395210	0.3646722	0.1755401	0.5941906
Columns 9 through 16:							
0.1825054	0.9125813	0.6093672	0.8596646	0.1041617	0.3837216	0.7522683	0.0912236
Columns 17 through 24:							
0.4678029	0.8957740	0.6006018	0.5554513	0.3171908	0.3341472	0.6267440	0.5704044
Columns 25 through 32:							
0.6792057	0.3451074	0.9394340	0.5892555	0.3878672	0.3618361	0.6830262	0.4502316
Columns 33 through 40:							
0.9772417	0.7216628	0.6583418	0.4919254	0.0571763	0.1980651	0.4380504	0.8046154
Columns 41 through 48:							
0.9805499	0.7382933	0.2957941	0.9778336	0.4017014	0.3270928	0.0146002	0.4921344
Columns 49 through 56:							
0.6065277	0.3163401	0.4250290	0.3126163	0.3909213	0.4777542	0.9653179	0.1881613
Columns 57 through 64:							
0.8137903	0.9290822	0.3574428	0.5309356	0.8607347	0.2854213	0.1438435	0.0327482
Columns 65 through 72:							
0.7362324	0.6927067	0.3853467	0.3942685	0.5222611	0.7016789	0.1710537	0.0829819
Columns 73 through 80:							
0.5644920	0.9868701	0.4892096	0.5509644	0.8962624	0.7675932	0.1560192	0.4914195
Columns 81 through 88:							
0.4660655	0.7642065	0.9006957	0.2263458	0.6977150	0.6162237	0.4367169	0.1340299
Columns 89 through 96:							
0.0887375	0.0856385	0.2158713	0.2736820	0.7106022	0.0076461	0.2352724	0.3951634
Columns 97 through 104:							
0.8814811	0.9446910	0.4382075	0.4090498	0.9935502	0.2737149	0.1919758	0.9615369
Columns 105 through 110:							
0.1395210	0.3646722	0.1755401	0.5941906	0.1825054	0.9125813		

What did you hear? The answer is a sound with a pitch of 441 Hz with a different timbre.

Plot it.

```
octave:41> wav = noisel100;
octave:42> plot( wav( 1 : 500 ) );
```



## Plot The Six Basic Synthesis Waveforms in One Window

The Six basic waveforms you've created them

```
sine = sine100samplePeriod;
saw = ramp0to100;
square = onOff50;
triangle = rampRiseFall50;
pulse = oneAnd99zeros;
noise = allrandom;
```

**Create a new function file called sixBasicWaveforms.m.**

Write code to create and plot 500 samples each waveform in one window using the [ 0 500 -1.2 1.2 ] for the axis of each subplot.

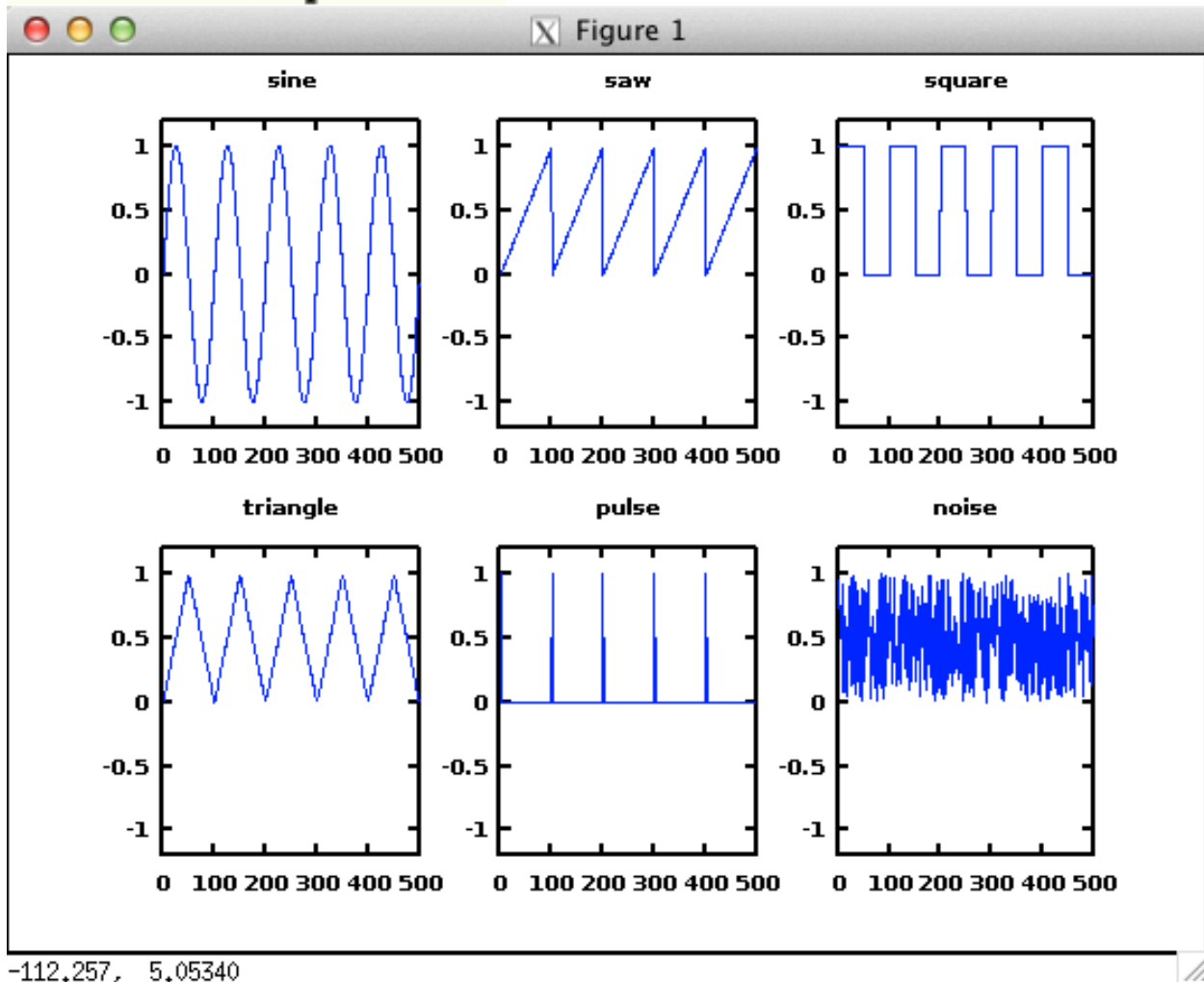


Hint:

```
octave:57> help subplot
```

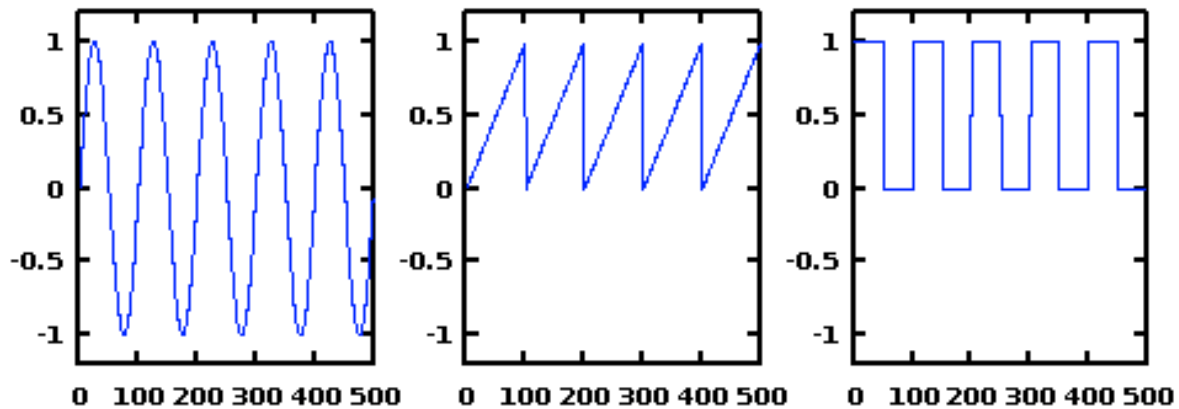
and

```
octave:62> help title
```



## Bipolar and Unipolar Waveforms

If you look at the Y axis limits of the six waveforms you'll notice that the sine wave is the only one that has both positive and negative values and is symmetrical around zero. The sine wave is a bipolar waveform. The remaining five are unipolar waveforms that have only positive values.



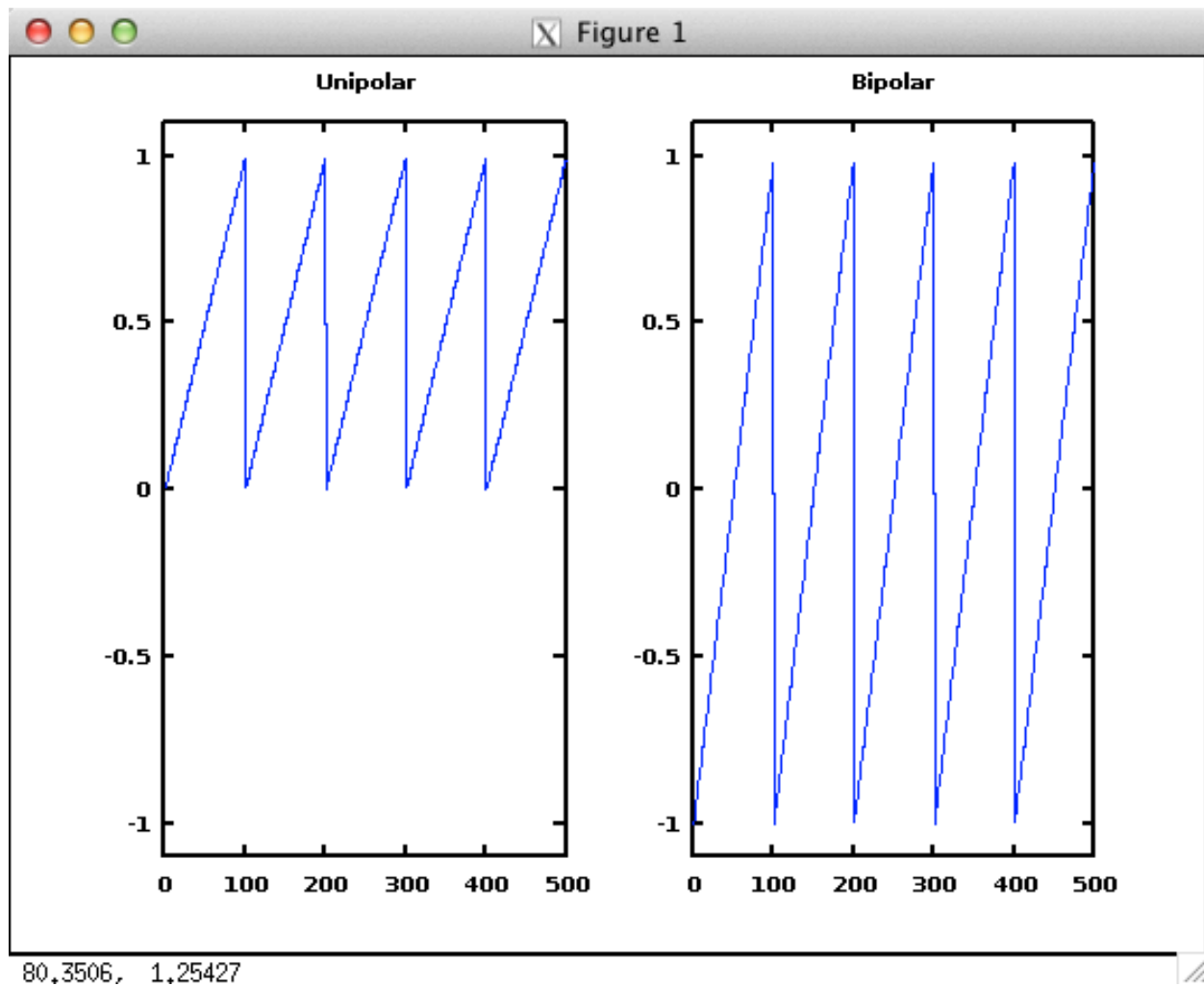
All audio waveforms on a synthesizer are bipolar, usually from  $-1$  to  $+1$ .

## Convert a Unipolar Waveform to a Bipolar Waveform

Here's the plan to convert the sawtooth wave. The others will be similar.

If you multiply every sample of the sawtooth wave by 2 and then subtract 1 its amplitude will range from  $-1$  to  $+1$ ;

See if you can do it.



## Listen to all six waveforms

Create a new function file `play6.m`

```

## play6

## Author: John Ellinger <je@jemac.mibac.lan>
## Created: 2014-01-07

function [ ret ] = play6 () # function return variable is called ret

    sine = sine100samplePeriod;
    saw = ramp0to100;
    square = onOff50;
    triangle = rampRiseFall50;
    pulse = oneAnd99zeros;
    noise = allrandom;

    # convert the five unipolar waveforms to bipolar
    saw = saw * 2 - 1;
    square = square * 2 - 1;
    triangle = triangle * 2 - 1;
    pulse = pulse * 2 - 1;
    noise = noise * 2 - 1;

    # lower the volume
    sine = sin * 0.5;
    saw *= 0.5; # shorthand for same thing
    square *= 0.5;
    triangle *= 0.5;
    pulse *= 0.5;
    noise *= 0.5;

    # create a half second of silence - rest
    rest = zeros( 1, 22050 );

    # create a melody
    ret = [ sine rest saw rest square rest triangle rest pulse rest noise ];

    # play it
    playsamples( ret );

endfunction

```

Execute it.

```
octave:160> wav = play6();
```

## Save this as a .wav file

```
| octave:161> wavwrite( wav', 44100, 16, "sixSynthWaves.wav" );
```

The file will be saved in the Octave "working directory". Type pwd at the Octave prompt

to see the path.

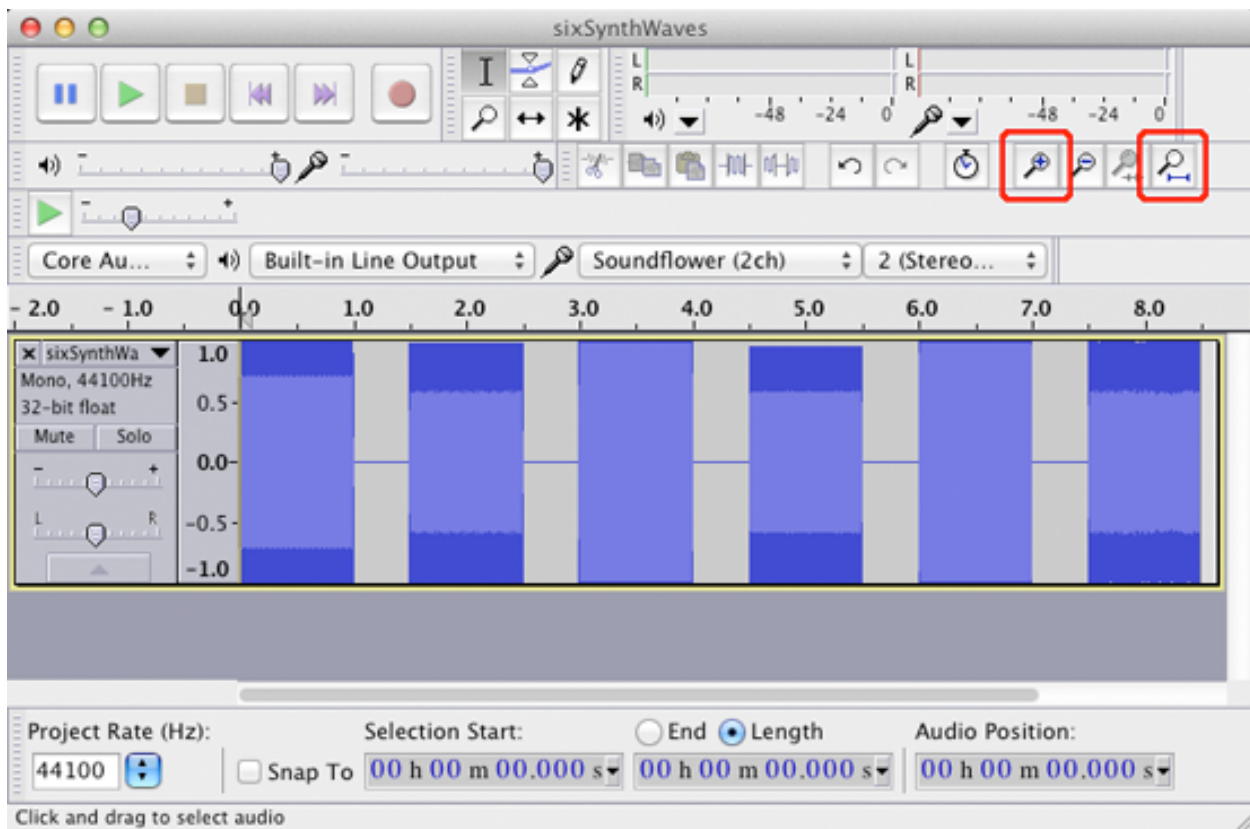
```
octave-3.4.0:92> pwd
ans = /Users/je/m208
```

Type ls to make sure it's there.

```
octave:162> ls
add3.m          noise.m         play6.m         sixWaveforms.m
allones.m       noise100.m      ramp0to100.m    x.m
allrandom.m     onOff50.m       rampRiseFall150.m
allzeros.m      oneAnd99zeros.m sine100samplePeriod.m
gensin.m        onesAndMostlyZeros.m sixSynthWaves.wav
```

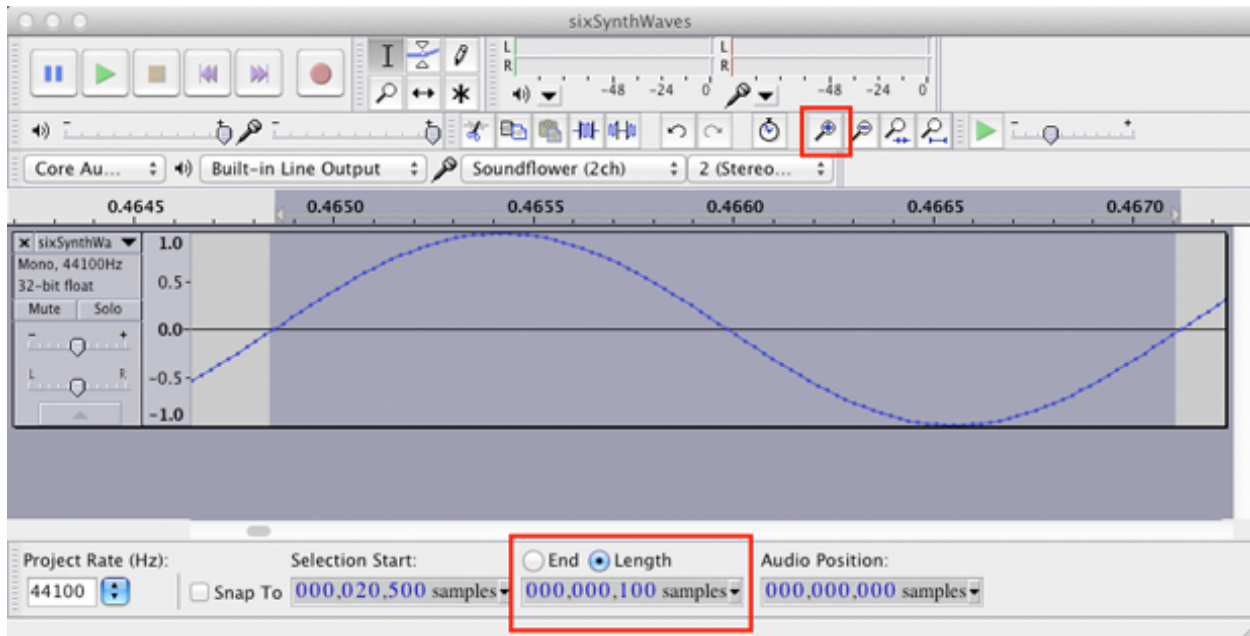
## Open sixSynthWaves.wav in Audacity

**NEEDS new pict.**



Click once in each block to set the cursor position. Then use the tool on the left outlined in red to zoom in to view the waveform shape. Return to the full waveform view by clicking the highlighted tool on the right.

You can estimate the frequency by counting samples. In this picture the sine wave wave was zoomed to the sample level and one period of the wave was selected. At the bottom of the window the Length button was selected and the popup menu underneath was set to display samples. Because we know the sampling rate is 44100 samples per second and one period is 100 samples the frequency is  $44100/100 = 441$  Hz.



The general formula for finding the frequency at any sample rate is:

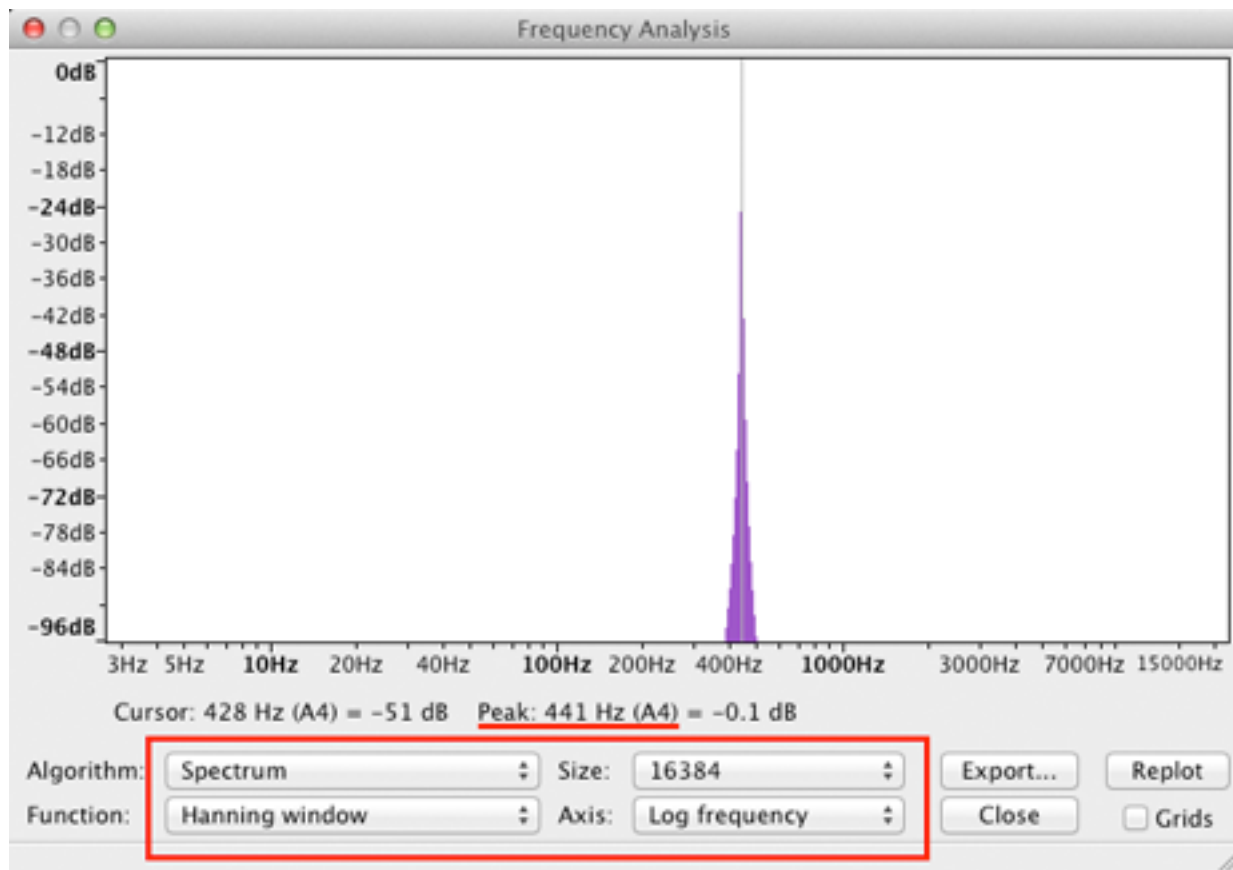
$$f = \frac{\text{SampleRate}}{\text{samplesInOnePeriod}}$$

## Examine the frequency spectrum of each block

Select each block and choose Plot Spectrum from the Analyze menu.

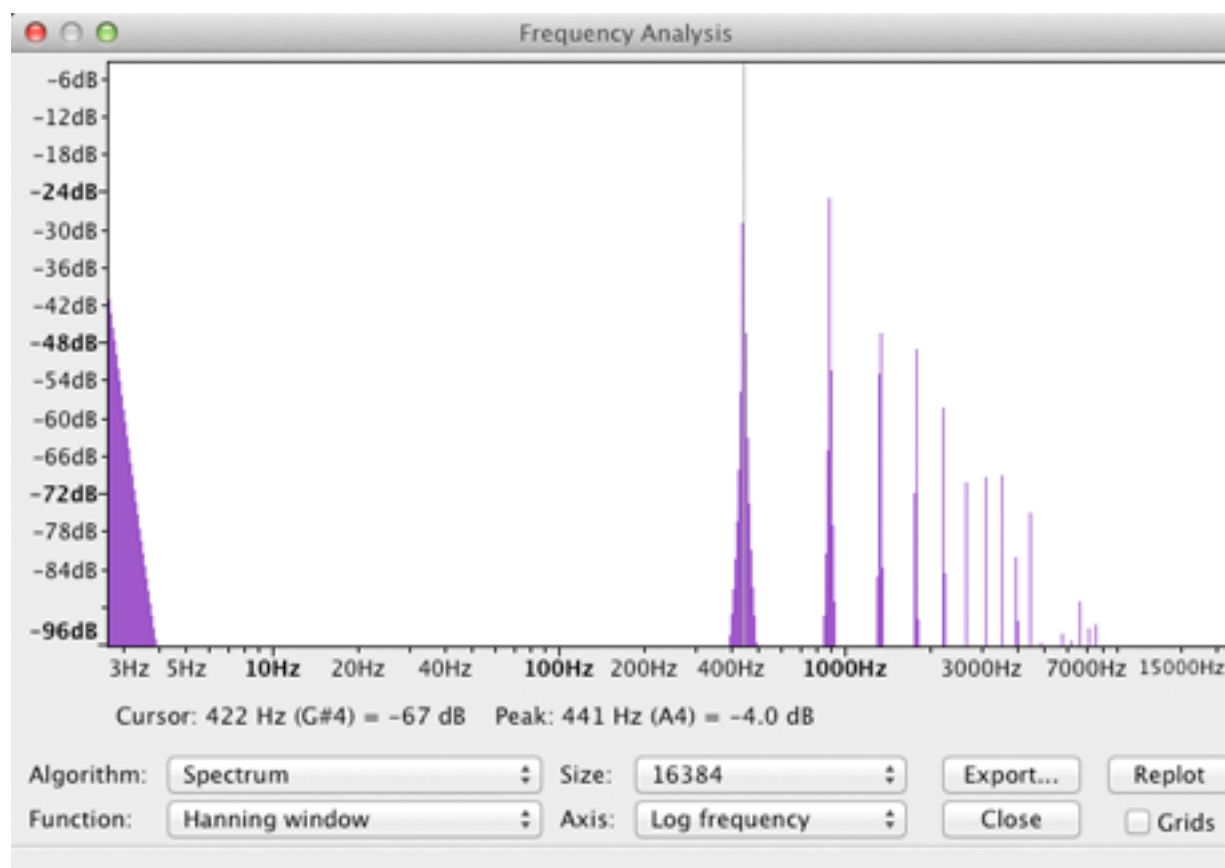
### Sine wave

A pure sine wave is the only sound that consists of one and only frequency. All other wave forms have multiple frequency components.

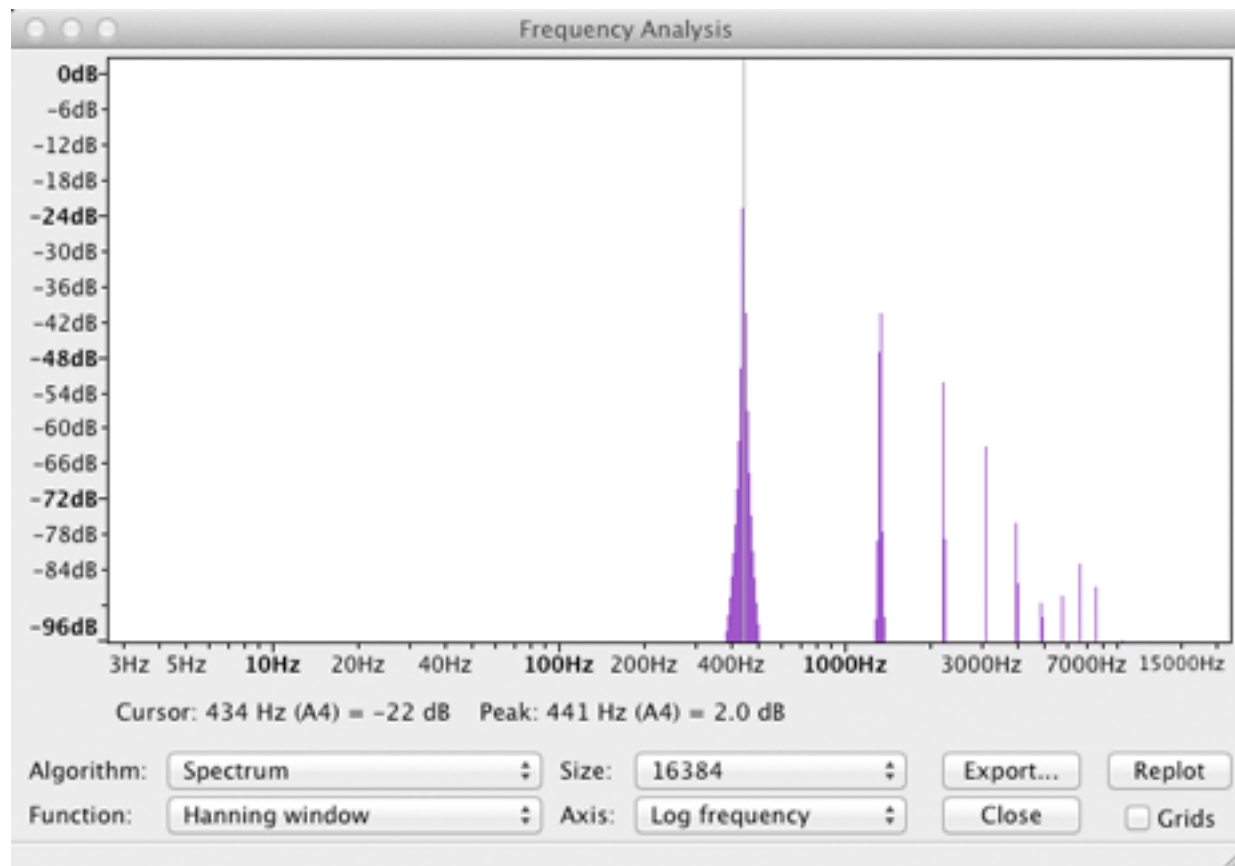


**Sawtooth wave**

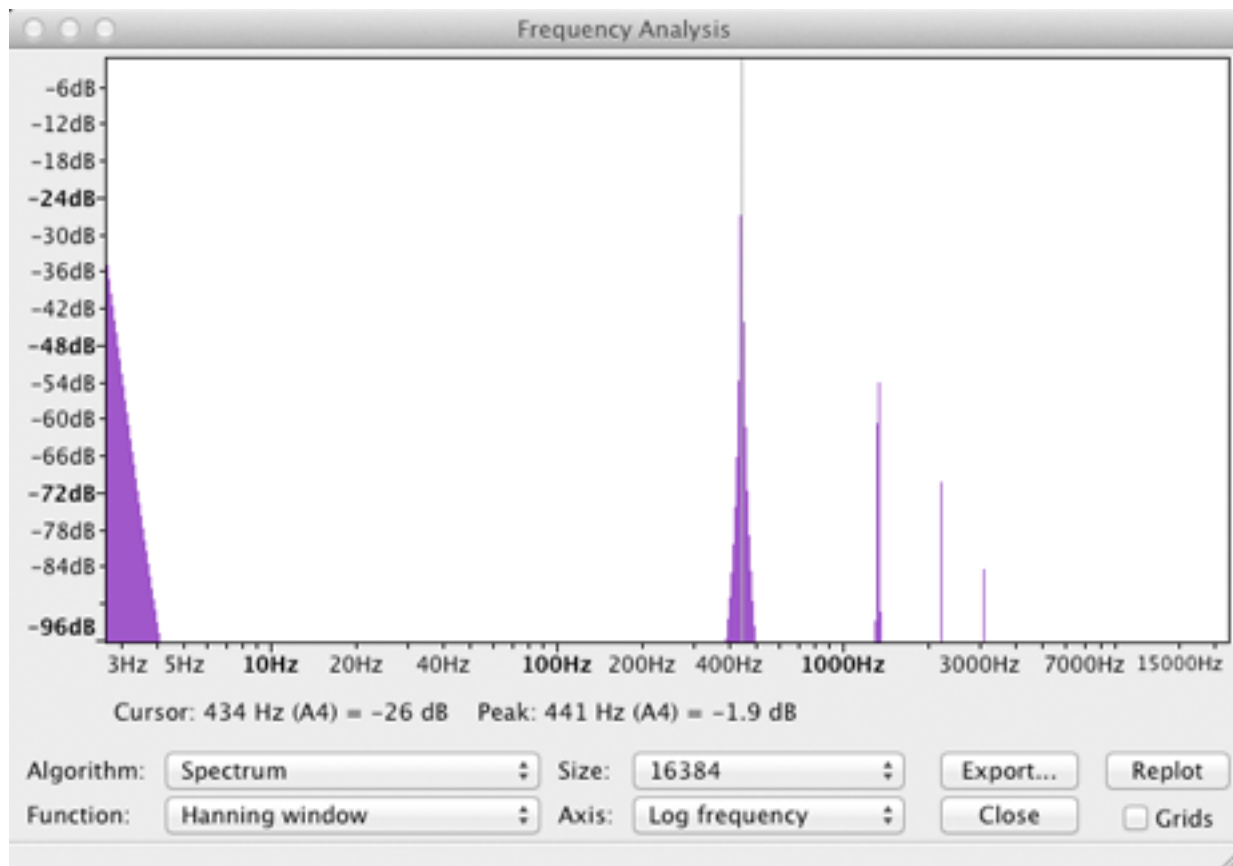




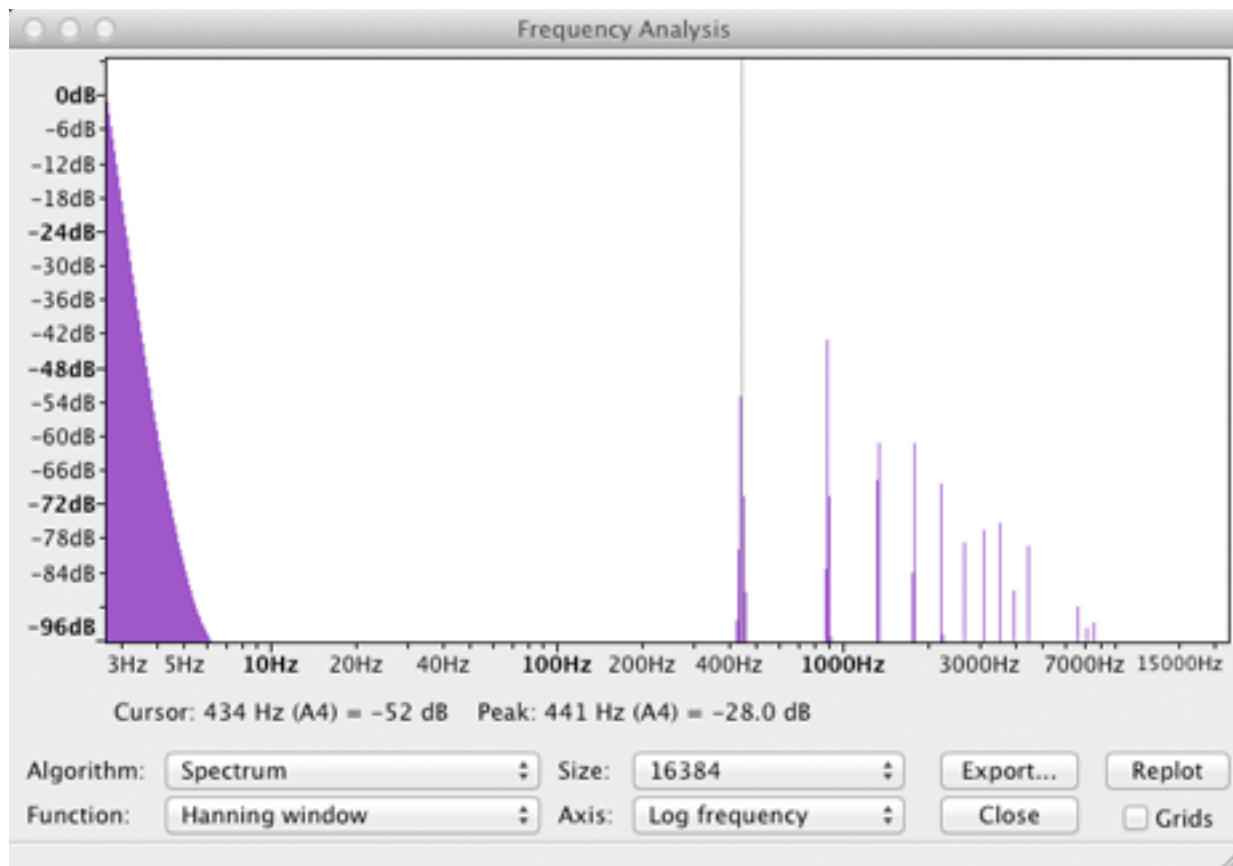
**Square wave**



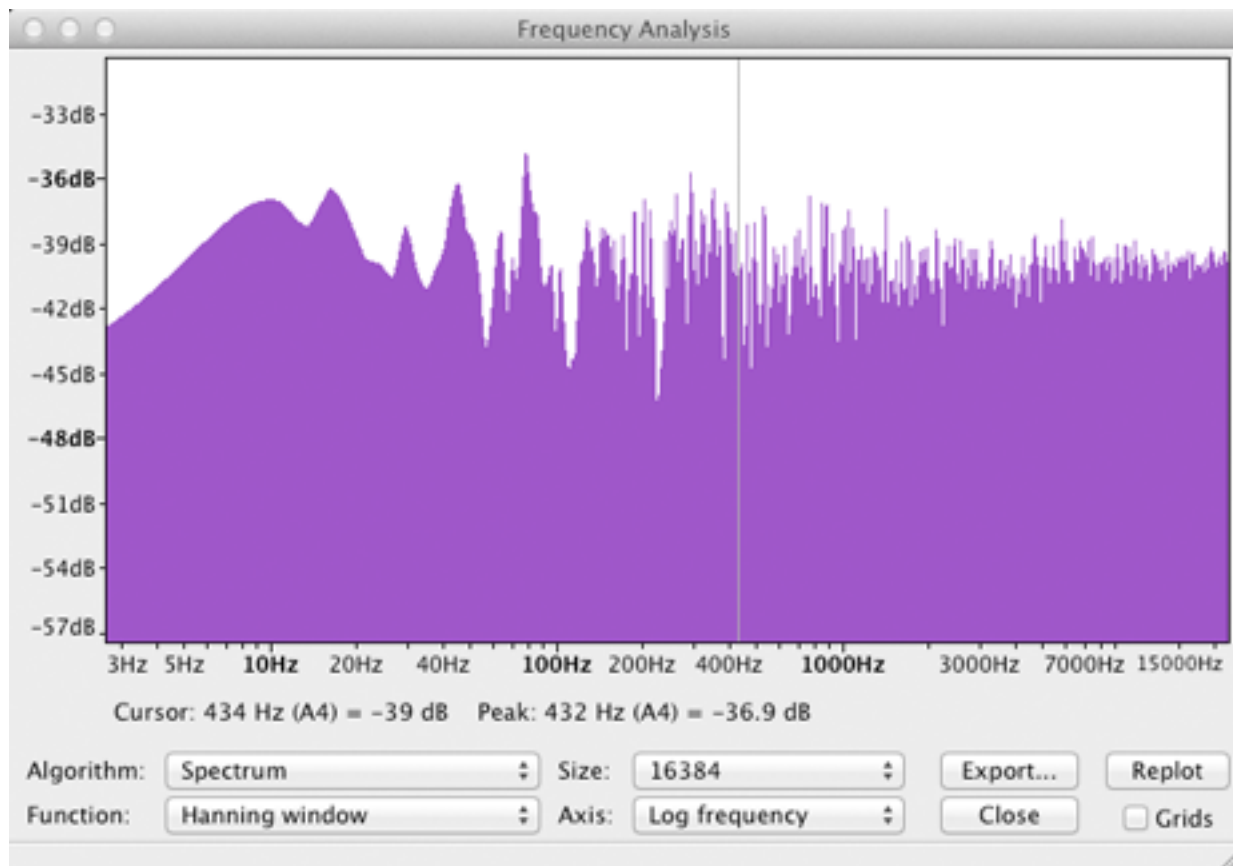
**Triangle wave**



**Pulse wave**



Noise



It's the number, spacing, and amplitude of the frequency components that give each of these 441 Hz sounds their different tone color or timbre.

## Lab 2 - Chuck: Alias Demo

### The Nyquist Frequency

A cornerstone theorem of digital sampling is known as the Nyquist-Shannon theorem. It basically states that in order to accurately sample any signal whose highest frequency is  $F_{\max}$ , the sampling rate must be at least twice that,  $2 * F_{\max}$ .

The range of human hearing is often stated as being within 20 Hz to 20000 Hz. It would require a sampling rate of at least 40000 Hz to capture a 20000 Hz signal. The standard audio CD rate of 44100 Hz leaves a bit of headroom. The highest frequency that can accurately be captured at the audio CD rate is 22050 Hz.

The Nyquist frequency or Nyquist limit is the highest frequency that can be accurately sampled and is equal to the Sample Rate divided by two,  $SR/2$ .

Aliasing happens when there are frequencies higher than half the sampling rate present in the signal. In some cases aliasing can produce audible artifacts that were not present in the original signal according to this formula:

$$aliasedFrequency = frequencyThat'sTooHigh - sampleRate$$

For example if a frequency of 30000 Hz was present in the original signal that was being sampled at the audio CD rate, it would be aliased to a signal at -14100 in the samples. Don't let negative frequencies bother you they sound exactly like positive frequencies, they just start with a different phase.

Here's a ChuckK demo that illustrates aliasing. Execute this code.

```

1 // Define a sine oscillator
2 SinOsc s => dac;
3
4 // first frequency to play is 34100 Hz, well above the audible range
5 34100 => int playFreq;
6
7 // ending frequency is 54100 Hz, well above the audible range
8 54100 => int endFreq;
9
10 // add 1000 Hz to playFreq each time through the loop
11 1000 => int deltaFreq;
12
13 while (playFreq < endFreq)
14 {
15
16     playFreq + deltaFreq => playFreq;
17     playFreq => s.freq;
18     // print output to Console Monitor window \t is a TAB character
19     <<< "Actual/Aliased frequency: ", playFreq, "\t", playFreq - 44100 >>>;
20     1::second => now; // play for one second
21 }
22

```

## Lab 2 - Chuck: Equal Loudness Contours

A tone at a given decibel level with a low frequency may not be perceived as being at the same loudness as a tone at the same decibel level but with a high frequency. Let's test it in Chuck.

Open /Applications/miniAudicle

Type this code.

```

arguments
1 SinOsc s => dac;
2 200 => s.freq;
3 Std.dbtorms( 70 ) => s.gain;
4 1::second => now;
5
6 0 => s.gain;
7 200::ms => now;
8
9 1000 => s.freq;
10 Std.dbtorms( 70 ) => s.gain;
11 1::second => now;

```

Line 1: Create a sine wave oscillator, s, and chuck it to dac (Digital Audio Converter = speaker).

Line 2: chuck 200 (Hz) to the oscillator frequency.

Line 3: Convert 70 dB into amplitude to set the gain (volume) of the oscillator.

Std is a ChuckK library that contains many Standard functions used in audio processing. Two of these are Std.rmstodb that converts an amplitude in the range  $\pm 1$  to its decibal value and the inverse Std.dbtorms that converts a decibal value in the range 100 to 0 (softest) to an amplitude value. Because the oscillator gain is expecting an amplitude value from 0 to 1 we need to convert the dB value using Std.dbtorms.

You can look up functions in ChuckK's standard library on this page: <http://chuck.cs.princeton.edu/doc/program/stdlib.html>

[function]: float **rmstodb** ( float **value** );

- converts linear amplitude to decibels (dB)

[function]: float **dbtorms** ( float **value** );

- converts decibels (dB) to linear amplitude



Line 4: Process audio for one second. You'll hear the sound.

Line 6-7: Silence for 200 milliseconds

Line 8: chuck 1000 (Hz) to the oscillator frequency.

Line 9: Convert 70 dB into amplitude to set the gain (volume) of the oscillator.

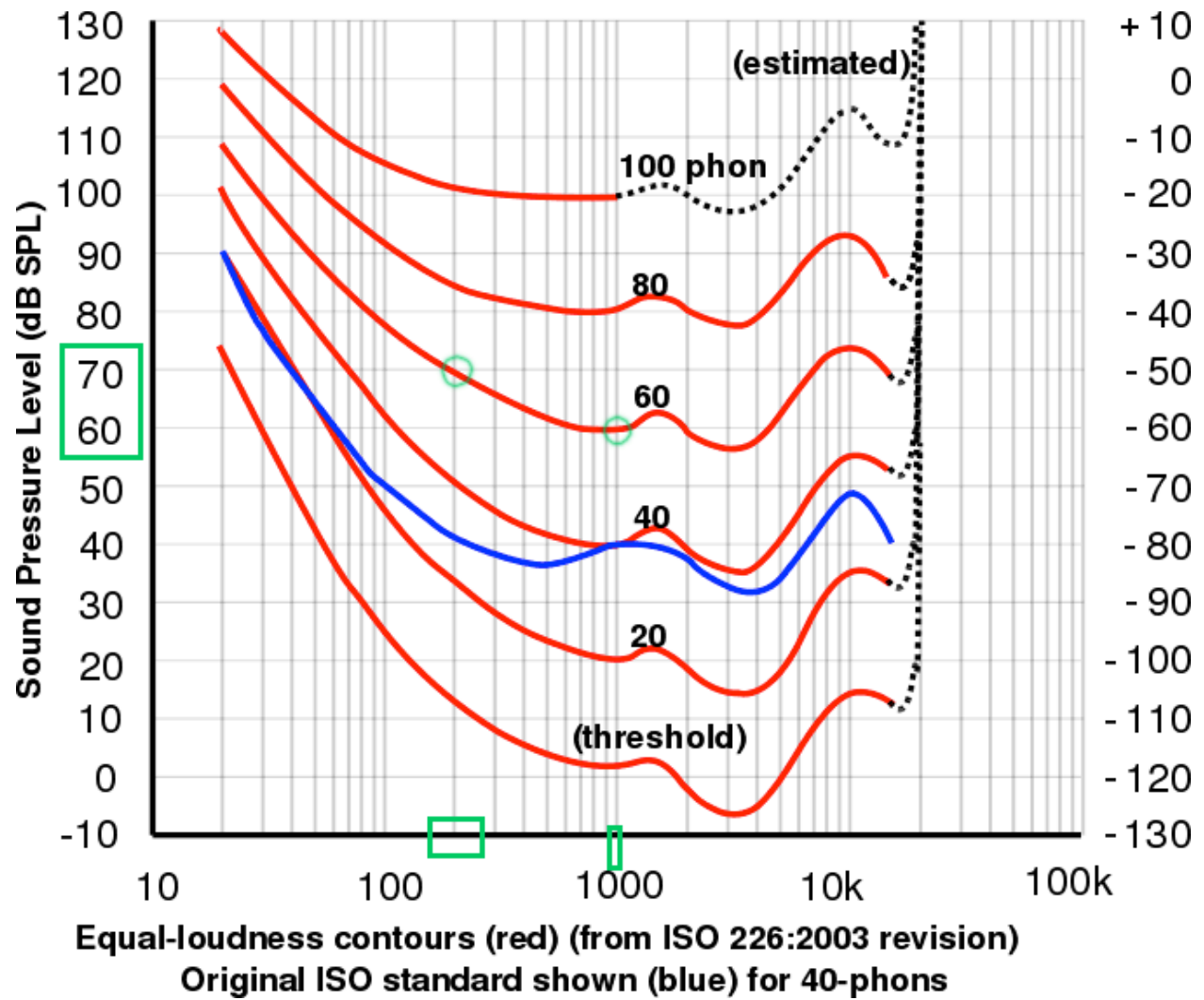
Line 10: Process audio for one second. You'll hear the sound.

I heard the 1000 Hz as distinctly louder than the 200 Hz tone.

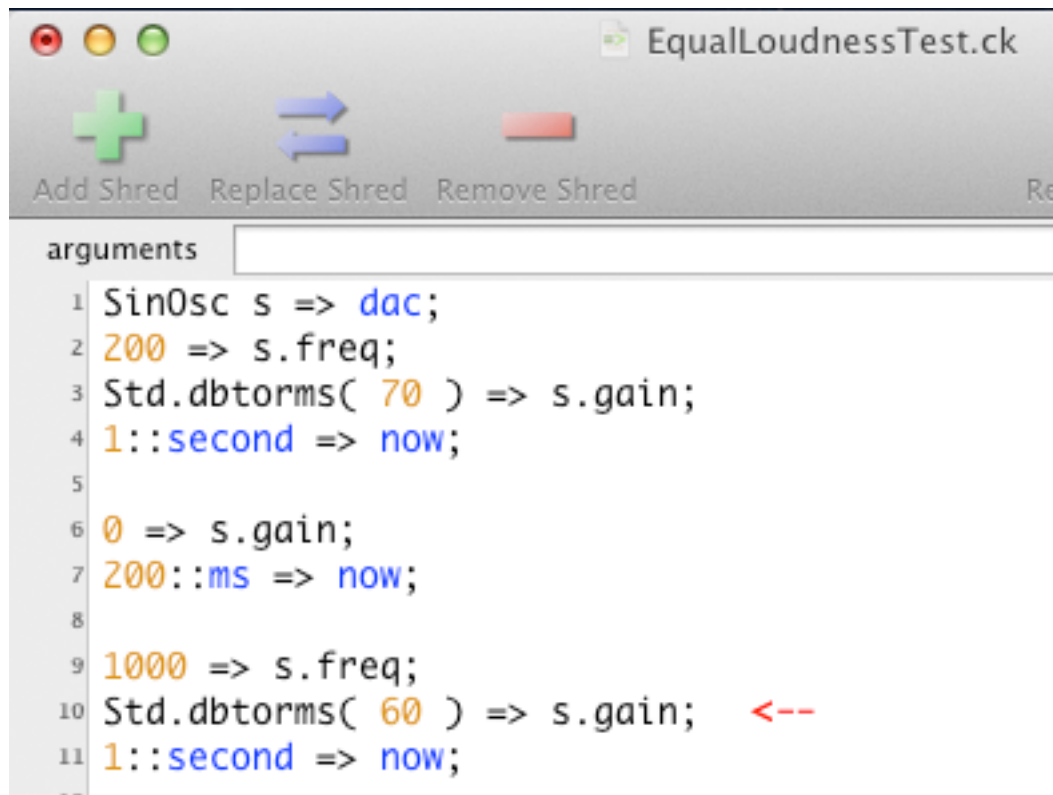
## **The Equal Loudness Contour Chart**

The Equal Loudness Contour Chart shows lines that represent the decibel levels where frequencies are perceived to have the same volume.

The 60 phon line shows that a 200 Hz sine tone at 70 dB should sound as loud as a 1000 Hz sine tone at 60 dB.



Try it.



```
1 SinOsc s => dac;
2 200 => s.freq;
3 Std.dbtorms( 70 ) => s.gain;
4 1::second => now;
5
6 0 => s.gain;
7 200::ms => now;
8
9 1000 => s.freq;
10 Std.dbtorms( 60 ) => s.gain; <--
11 1::second => now;
```

Done.