MUSIC 208 Homework 5-6

Five Questions. Source code for each question must to be turned in to the course Hand-in folder. Test your code before turning it in. Turn in the exact source code you've tested. No MS WORD documents. Put your name in comments at the top of each file. These are the five files I'm expecting.

1. gensin.m
2. gensin.ck
3. changeSpeed.m
4. frequencySweep.m
5. adsr.m

1. Write a Sine wave generator function in Octave called gensin.m

Create An Octave Function File That Generates A Sine Wave At A Given Amplitude, Frequency And Duration.

```
function [ ret ] = gensin (amp, freq, secs)
     ## your code goes here
endfunction
```

When completed you should be able to call it like this:

```
octave-3.4.0:22> wav = gensin( 0.2, 440, .5 );
octave-3.4.0:23> playsamples(wav);
```

Test your code

Test Durations

```
octave-3.4.0:64> wav = gensin( 1.0, 440, 1.0 );
octave-3.4.0:65> playsamples( wav );
octave-3.4.0:66> wav = gensin( 1.0, 440, 0.1 );
octave-3.4.0:67> playsamples( wav );
```

Test Frequencies

Can you hear the difference of 1 Hz between 440 and 441 Hz.

```
octave-3.4.0:72> w1 = gensin( 0.4, 440, 3.0 );
octave-3.4.0:73> w2 = gensin( 0.4, 441, 3.0 );
octave-3.4.0:74> playsamples( w1) ; playsamples( w2 );
```

You can when they're played together.

```
octave-3.4.0:75> playsamples( w1 + w2 );
```

You should clearly hear the beats produced between the two similar frequencies. Theory says the beating rate should be equal to the difference between the frequencies.

2. Write The gensin.ck Function In Chuck

Save the file gensin.ck. Submit it to the Course Hand-in folder.

You must use the Impulse object and chuck it to dac. No SinOsc allowed. Look up the documentation for Impulse, especially the .next method.

```
////////////// Begin
your_email_name_gensin.ck ///////////////

function void gensin( float amp, float freq, float secs )
{
    Impulse imp => dac;
    // Your code goes here
}
// Test it
gensin( 0.2, 440, 1.0 );
gensin( 0.8, 220, 1.5 );
gensin( 0.5, 500, 0.2 );

////////////// End
your_email_name_gensin.ck ///////////////
```

gensin.ck Help

Here's an example of how to use the Impulse function in ChucK

```
// The duration is 0.5 seconds = 22050 samples
// Every 100th sample has a value of 1.0
// and all the other samples have a value of 0

Impulse i => dac;

0 => int counter;
do
{
// the next several lines could be the Sin function
// instead of the % (mod) function
    if ( counter % 100 == 0 )
    {
        // set the next sample
        1.0 => i.next;
    }
    else
        0 => i.next;

        // increment counter
        counter++;
    1::samp => now;
} until (counter == 22050);

// you should hear 1/2 second of 44100/100 = 441 Hz tone
```

3. Write An Octave Function To Change The Speed Of Any Wave File

Save the file as changeSpeed.m Submit it to the Course Hand-in folder.

You should be able to reuse much of the interpWavtable.m file from Lab 6 with some changes.

```
################ YOUR NAME ################

## changeSpeed ( wav,rate )
##    wav contains the samples
##    rate is the speed of reading the samples
##    1.0 is normal speed, 2.0 is twice as fast
##    0.5 is twice as slow

function [ ret ] = changeSpeed ( wav, rate )
     SR = 44100;
     ## your code goes here
endfunction
```

Test It

A rate of 1.0 plays the sound at normal speed. Rates less than 1.0 slow it down, rates greater than 1.0 speed it up. Ensure that the entire file plays through once regardless of the rate. Test it with the Music208.wav file from Lab 3.

```
octave-3.4.0:147> [wav, FS, BITS] = wavread("music208.wav");
octave-3.4.0:149> w = changeSpeed( wav, 1.0 );
octave-3.4.0:150> playsamples( w );

octave-3.4.0:151> w = changeSpeed( wav, 1.267 );
octave-3.4.0:152> playsamples( w );

octave-3.4.0:153> w = changeSpeed( wav, 0.589 );
octave-3.4.0:154> playsamples( w );
```

changeSpeed.m Help

This code template is very similar to the interpWavetable.m function at the end of Lab 6. It's an outline of the solution I used.

```
function [ ret ] = changeSpeed ( wavIn, rate )
      ## initialize SR, numSamples, wavOut
      ## your code goes here

      ## find length of wavIn
      ## your code goes here

      ## number of samples to play is inversely related to rate
      ## solve this for numSamples: numSamples * rate = Length
      ## your code goes here

      ## declare empty array, used later in for loop
      ## your code goes here

      ## Roads phase_increment formula
      ## here the phase_increment is the rate
      ## your code goes here

      ## set initial value of prev_phase_index
      ## your code goes here

      ## set first samples equal to each other
      ## your code goes here

      for outN = 2:numSamples
            ## Roads phase_index formula
            ## your code goes here

            ## find the sample to the left
            ## your code goes here

            ## check to see if we need to wrap around use mod length
            ## your code goes here

            ## find the sample to the right
            ## your code goes here

            ## check to see if we need to wrap around
            ## your code goes here
```

```
            ## Interpolate formula
            ## your code goes here

            ## assign sample value to output
            ## your code goes here

            ## update previous_phase for next time through the loop
            ## your code goes here
      endfor

      ## return wavOut samples
      ## your code goes here

endfunction
```

4. Write An Octave Function To Create A Frequency Sweep

Save your file as frequencySweep.m. Submit it to the Course Hand-in folder.

You should be able to reuse much of the interpWavtable.m file from Lab 6 with some changes.

```
################ YOUR NAME ################

## frequencySweep (freqStart, freqEnd, secs )
##     return a smoothly changing waveform
##     from freqStart Hz to freqEnd Hz of duration secs seconds

function [ ret ] = frequencySweep (freqStart, freqEnd, secs )
     SR = 44100;

     ## create a sine wave at freqStart Hz lasting secs seconds
     ## gensin from Lab 6
     wvTbl = gensin( 1.0, freqStart, secs );

     # the rest of your code goes here

endfunction
```

Test it in Octave. You want to hear a sine wave tone whose frequency changes smoothly from freqStart to freqEnd over a duration of secs seconds.

Hint: The phase_increment does not remain constant. It changes slightly on each sample calculation.

This will produce a one octave sweep from 200 Hz to 400 Hz in 2 seconds

```
wav = frequencySweep( 200, 400, 2 );
```

Play it in Octave. You should be able to hear if you're on the right track.

```
playsamples( wav );
```

If it sounds ok, write it to a wave file and open it in Audacity. Use the Plot Spectrum menu command in Audacity to and check the beginning and ending frequencies.

```
wavwrite( wav', 44100, 16, "sweep_200_400.wav" );
```

Then test further.

```
wav = frequencySweep( 200, 500, 2.0  );
```

```
wav = frequencySweep( 500, 200, 2.0  );

wav = frequencySweep( mtof(60), mtof(61), 2.0  );
```

mtof() was part of homework 1-2 and needs to be in same folder as frequencySweep.m

frequencySweep.m Help

This code template is very similar to the interpWavetable.m function at the end of Lab 6. It's an outline of the solution I used

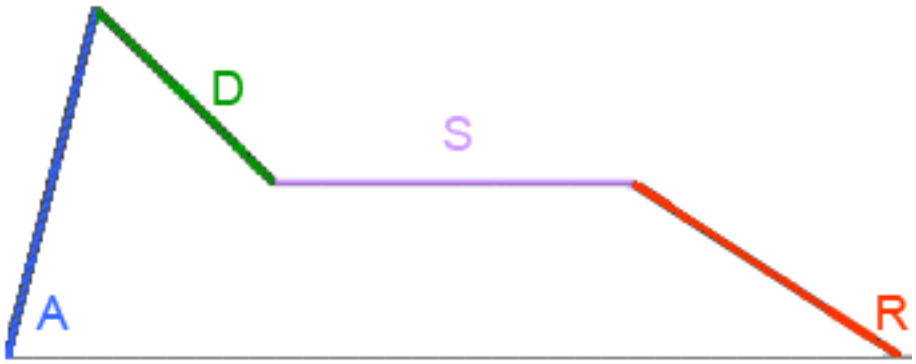
```
function [ ret ] = frequencySweep (freqStart, freqEnd, secs )
      ## initialize SR, numSamples, wavOut
      ## your ## your code goes here goes here

      ## create a wavetable using gensin at freqStart Hz lasting secs seconds
      ## your code goes here

      ## find numSamples in wavetable
      ## your code goes here

      ## set the phase_increment starting rate to 1.0, normal speed
      ## your code goes here

      ## calculate the phase_increment value at the final frequency
      ## it will be the ratio freqEnd / freqStart
      ## for example if freqEnd is one octave higher then
      ## phase_increment_end would be 2.0
      ## your code goes here

      ## Roads phase_increment formula
      ## calculate the phase_increment_delta
      ## equals phase_increment / tableLength
      ## your code goes here

      ## set initial value of prev_phase_index to 0
      ## your code goes here

      ## declare empty array, used later in for loop
      ## your code goes here

      ## set phase_increment start value
      ## your code goes here

      ## set first samples equal to each other
      ## your code goes here

      for outN = 2:tableLength
            ## add phase_increment_delta to phase_increment
            ## your code goes here
```

```
            ## Roads phase_index formula
            ## your code goes here

            ## find the sample to the left
            ## your code goes here

            ## find the sample to the right
            ## check for wrap around, rightSample > L, if so set to 1
            ## your code goes here

            ## Linear Interpolation formula
            ## your code goes here

            ## assign sample value to output
            ## your code goes here

            ## update prev_phase_index for next time through the loop
            ## your code goes here

      endfor

      ## return output samples
      ## your code goes here

endfunction
```

5. Write An Octave Function to Create an ADSR Envelope

Save the file as adsr.m. Submit it to the Course Hand-in folder.

One of the most widely used envelopes in digital audio is the **ADSR envelope** which stands for Attack Decay Sustain Release.



The attack begins at time zero and ends at decay start at which point it has reached the maximum attack level.

The Decay begins when the attack ends and continues until Sustain begins, decaying from the attack level to the sustain level over that period of time.

The Sustain level remains constant until the release begins.

The Release ends at time 1 at which point it has fallen to zero.

You can create an ADSR line segment function if you know these five values: decay start, sustain start, release start, attack level, sustain level. All parameters are assumed to be in the range of 0-100. You do not need to implement range checking, just make sure you get valid results with valid parameters.

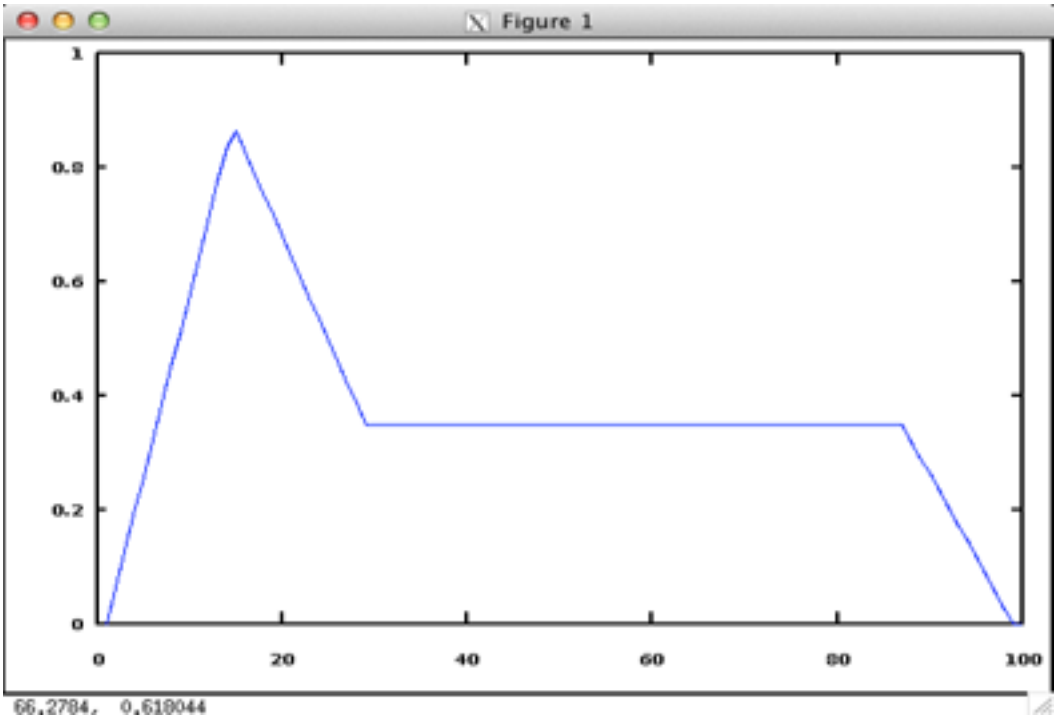
The Octave ADSR Function

```
################ YOUR NAME ################

## adsr (dx,sx,rx,ay,sy)
##      dx = decay start 0-100
##      sx = sustain start 0-100
##      rx = release start 0-100
##      ay = attack level 0-100
##      sy = sustain level 0-100
```

```
function [ ret ] = adsr (dx,sx,rx,ay,sy)
     y = 1:100;

     ## atack
     for ix = 1:ax
          # calculate attack segment
     endfor
     # insure y(1) = 0

     ## decay
     for ix = dx:sx
          # calculate decay segment
     endfor

     ## sustain
     for ix = sx:rx
          # calculate sustain segment
     endfor

     ## release
     for ix = rx:100
          # calculate release segment
     endfor

     # insure y(100) = 0

     # normalize all y values to range 0 - 1
     y =  y * 0.01.

     ret = y;
endfunction
```

Plot It

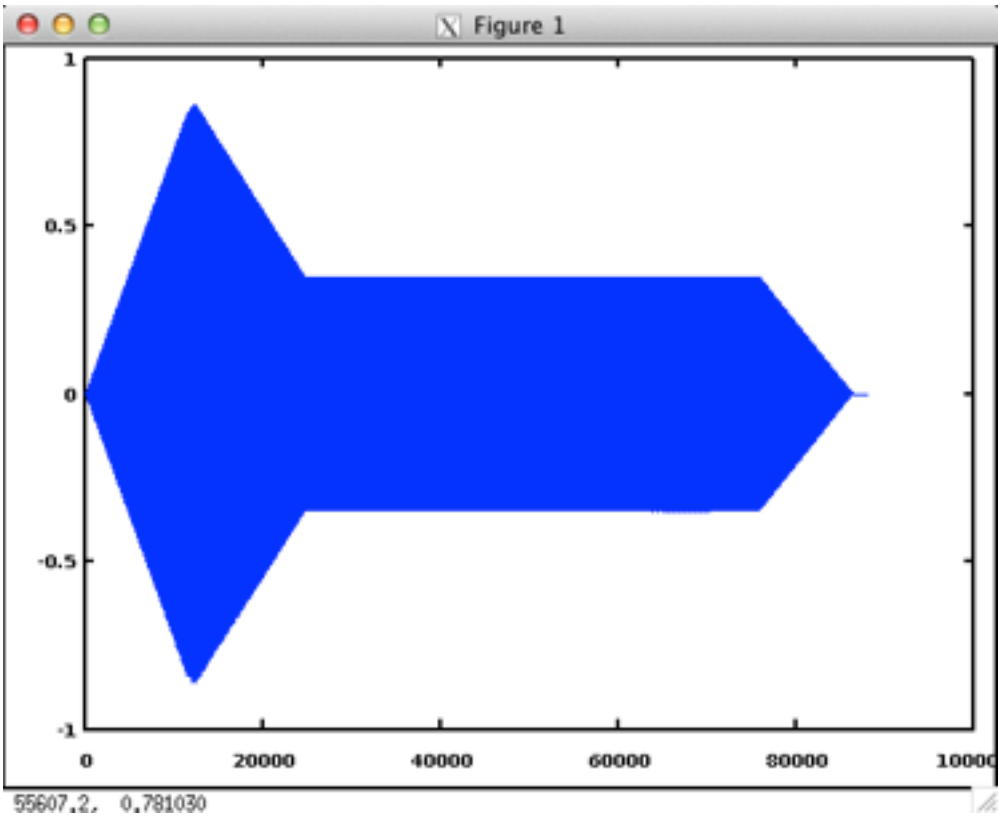
```
octave-3.4.0:388> env = adsr( 15, 30, 88, 90, 35 );
octave-3.4.0:389> plot(env)
```



Test It

```
octave-3.4.0:9> secs = 2.0;
octave-3.4.0:10> swav = gensin( 1.0, 440, secs );
octave-3.4.0:11> env = adsr( 15, 30, 88, 90, 35 );
octave-3.4.0:12> ienv = interpWavtable( env, 1.0, 1/secs, secs );
octave-3.4.0:13> wav = swav .* ienv;
octave-3.4.0:14> plot( wav );
octave-3.4.0:15> playsamples( wav );
```

This is a plot of the waveform after the envelope has been applied.



ADSR Help

No help. You should have learned line (slope) formula in algebra or geometry.