ELE 301, Fall 2010
Laboratory No. 2

1 Background

The objective of this lab is to illustrate the fact that complex signals can be synthesized from rather simple signals by the use of just a few elementary signal operations: envelope modulation, time shifts, and superposition (addition). We will use the application of music synthesis to explore these ideas in an interesting and fun context.

You will write a MATLAB program that inputs the score of a short piece of classical music (in more or less standard musical notation) and synthesizes a time sampled waveform corresponding to the specified music.

1.1 .wav files

In order to convert and save the matlab digital audio signal in .wav format use the following command

```
wavwrite(s,fs,N, 'e:\pathname\filename'); % save s in .wav file format
% with quantization to N bits.
```

This command quantizes values in `s` to `N` bits and then together with the sampling frequency `fs` converts the signal into .wav format and saves it to the file "filename" on drive "e:" and path "pathname". You should store files in your home directory or a subdirectory of your home directory. You can then listen to the signal by starting the audio player and opening this file.

1.2 Music Synthesis

Figure 1: Morning Mood

The first few bars of Morning Mood by E. Grieg are shown in Fig. 1. The position and note value of each note indicates the frequency, and duration of the waveform or signal associated with that note and the order of the notes indicates timing.

Suppose that a single note of frequency f, duration d and unit amplitude played at time 0 yields the waveform $w_{f,d}(t)$. Let the frequency, duration and amplitude of the j^{th} note in the above score be denoted by f_j, d_j and A_j respectively. Then we can construct the waveform described by the score as the sum:

$$m(t) = \sum_{j=1}^{M} A_j w_{f_j,d_j}(t - \tau_j) \tag{1}$$

were τ_j is the time at which the j^{th} note is played. The music is thus the sum, or superposition, of the all of the notes played at their appropriate times.

In principle, we can attempt to automate this process on a computer as follows: construct the waveforms $w_{f,d}(t)$ for each note in the score; then form the sum (1) using the appropriate delays and amplitudes as indicated in the score.

Of course we have to work with sampled versions of the waveforms. So the actual synthesis will be done using discrete-time signals intended to model sampled versions of the continuous-time waveforms.

1.3 Notes

Musical notation indicates the fundamental frequency of each note by its vertical location on the musical staff, and its duration by the note type: whole, half, quarter, eighth, etc. The musical notes are grouped into *octaves* with each octave containing 12 notes. The octave containing middle C covers the frequency range from 220Hz to 440Hz. Within each octave the notes are logarithmically spaced so that jumping one octave doubles the frequency. Thus the frequency of each note is $2^{1/12}$ times the frequency of the note below it. The frequencies of the notes in the middle C octave are shown in Figure 2.

A	220Hz
As	$2^{1/12}$*A ≈ 233Hz
B	$2^{1/12}$*As ≈ 247Hz
C	$2^{1/12}$*B ≈ 262Hz
Cs	$2^{1/12}$*C ≈ 277Hz
D	$2^{1/12}$*Cs ≈ 294Hz
Ds	$2^{1/12}$*D ≈ 311Hz
E	$2^{1/12}$*Ds ≈ 330Hz
F	$2^{1/12}$*E ≈ 349Hz
Fs	$2^{1/12}$*F ≈ 370Hz
G	$2^{1/12}$*Fs ≈ 392Hz
Gs	$2^{1/12}$*G ≈ 415Hz

Figure 2: Notes in the middle C octave.

1.4 Representing the score

We need to select a numerical representation for the note frequencies and durations. For the durations we will use 1 to represent a whole note, 2 to represent a half note, 4 for a quarter, and 8 for an eighth.

The frequency information can be represented in MATLAB be defining a variable for each note frequency: `s=2^(1/12); A=220; As=A*s; B=As*s; C=B*s; ...` etc. This has already been done for you and is available in the file `notes.m`. This M-file defines variables corresponding to three octaves. The variable names are: `A0, As0, B0, C0, Cs0, D0, Ds0, E0, F0, Fs0, G0, Gs0` for the octave below middle C; `A, As, B, C, Cs, D, Ds, E, F, Fs, G, Gs` for the octave containing middle C; and `A2, As2, B2, C2, Cs2, D2, Ds2, E2, F2, Fs2, G2, Gs2` for the octave above middle C.

Be careful: do not use these variable names in your program for other purposes. In particular `Fs` has been used for the note F sharp, so you cannot use it for the sampling frequency.

We can represent the first few bars of Morning Mood in MATLAB by 3 matrices and a scalar as follows:

```
%% MUSIC: MORNING MOOD
%% ——————————————————————————
% nf=Notes to play
% nd=Duration of each note: 8,4,2,1= 1/8,1/4,1/2,1
% na=Relative amplitude of each note
% TD=Time duration of one whole note in secs
nf=[G E D C D E G E D C D E G E G A2 E A2 G E D C ];
nd=[4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4/5];
na=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ];
TD=1.5;
%% ——————————————————————————
```

So `nf(j)` is the frequency of the j^{th} note, `TD/nd(j)` is its duration in seconds and `na(j)` is its relative amplitude.

1.5 Generating a note: the instrument

A primitive model for a note of frequency f and duration d is a sine wave of that frequency and duration followed by a short period of silence. This can be written as the continuous time signal

$$w_{f,d}(t) = \sin(2\pi f t)(u(t) - u(t-d)), \quad t \in [0, D]$$

where D is the total duration and d is the note duration.

The above model does not produce anything resembling realistic instrument sound. Real instruments have at least two additional important characteristics. First, when a particular note is played not only is the fundamental frequency is generated but also higher harmonics of the fundamental. The amplitudes of the harmonics are usually significantly less than that of the fundamental.

In addition to harmonics, the instrument also produces a characteristic note *envelope*. Figure 3 shows the waveform of actual piano notes. The signal envelope has a significant rise time and a slower exponential-like decay. In the primitive model, the note envelope is just a rectangular pulse $u(t) - u(t-d)$. Producing a more realistic sound requires replace this term by an envelope signal $e(t)$.

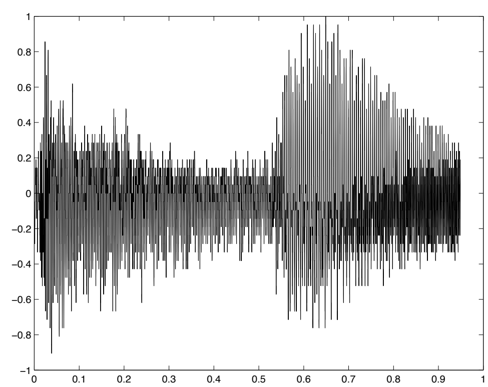

Figure 3: Piano notes.

ELE 301
Laboratory No. 2

2 Lab Procedure

Several MATLAB .m files will be provided: `notes.m`, `odetojoy.m`, `mornmood.m`, and `furelise.m`. The file `notes.m` contains the note definitions. Look in the file to see what variables have been defined. The example program included at the end indicates how this file could be used. The files `odetojoy.m` (short), `mornmood.m` (medium), and `furelise.m` (long) are files defining musical scores. When you are working with scores start with short ones first to test your program.

WARNING: Some programs you will write, although not very long, are complex and can be memory hungry. Write the program in simple steps. Test each step before proceding to the next, and use variables efficiently.

2.1 Task 1: Make a single note

Write three MATLAB programs that will generate a single note waveform of specified frequency and duration. The first program is the master program, `mkmusic.m`, that will define variables, and for the moment plot and play the result. The other two programs, `myinst.m` and `mknote.m`, are called by the master program. The program `myinst.m` specifies the instrument characteristics and the program `mknote.m` generates the actual note waveform.

Do this in the following sequence of steps:

1. Write the program `mkmusic.m`. This program should
   - clear memory and figures.
   - define the note frequency variables. This is easy: you can use the M-file `notes.m` just like a regular matlab built-in command.
   - call the program `myinst.m` (which you will write later) to define the instrument variables and characteristcs.
   - define the variables:

     `TD` - the duration, in seconds, of one whole note.
     `nf` - the frequency of the note to be played
     `nd` - the duration (1, 2, 4, 8) of the note to be played
     `na` - the amplitude of the note to be played
     `tt` - the time vector for the note

     For now just put in one note, so `nf`, `nd` and `na` will each have one entry.
   - call the program `myinst.m` (which you will write later) to define the instrument variables and characteristcs.
   - call a program `mknote.m` (which you will write later) to generate the note specified by frequency, duration and amplitude.
   - Plot the note waveform vs time (for the single note)
   - Play the note using the MATLAB `sound` command.
2. Write the program `myinst.m` to specify the instrument characteristics. This will include the harmonic amplitudes for a note (relative to a unity amplitude note), and the note envelope. Use the variables:

   `ha` - for the harmonic amplitudes
   `env` - for the note envelope

   At first, assume that the instrument is primitive: it has no harmonics and its envelope is just the pulse $u(t) - u(t - d)$, where `d` is the note duration.

3. Write the program `mknote.m` that generates a note of frequency `nf`, duration `nd` and amplitude `na` (with the instrument characteristics in `ha` and `env`, which for now don't matter) and returns it in the vector `n`. This will be very similar to the program `harmon.m` that you wrote in Lab 1. You can make this program a function or a script.

4. Test your programs by generating notes at middle C of durations 1, 1/2, 1/4 and 1/8, and unity amplitude. The sound output should be shorter for a 1/8 note (`nd=8`) than for a 1/4 note etc.. Demonstrate your program to the TA before proceeding further.

5. Once you have the above programs working you can experiment with instrument sounds by varying the harmonic content and envelope shapes, i.e., by refining the constants in your `myinst.m` program. Generally: more harmonics with larger amplitudes give an organ-like sound, moderate harmonic content and an exponential-like envelope will give a crude piano-like sound. Note that you could make the envelope depend on the note duration. How?

   When you are happy with your instrument sound, specifiy your harmonic amplitudes and your selected envelope function(s) here:

   Demonstrate your program to the TA. Then print out your programs and attach them to this handout.

2.2 Task 2: Music

Now that you can make one note the next task is put the notes together to play an entire score. While this is conceptually simple, obtaining and efficient implementation requires thinking things through carefully.

The first step is to add a line to the `mkmusic` program to load the score variables. The command `odetojoy`, for example, loads the score variables for the first few bars of Ode to Joy. This will take the place of your definitions for the single note played in the previous part of the lab.

Now one way to play the score is to add a loop that simply calls `mknote` for each note in the score. Try this and play the result.

You should detect two dawbacks to this approach: it sounds roughly like a one finger piano player because there is too much delay between notes; and we are still using the low quality MATLAB `sound` command.

To avoid using the `sound` command we want to generate the waveform for the entire score first, save it as a .wav file, and then play it later with the windows audio player. This should also solve the problem of excessive delay between notes.

We want to generate the waveform for the whole score using a discrete time version of equation (1). In doing this we will allow the note waveforms to overlap so that we model playing the next note while the previous one is still sounding. In principal this is easy: generate each note one at a time in the correct order and then add it as a subvector in the correct place to a big vector that represents the waveform for the entire score. The correct place corresponds roughly to the value of τ_j in equation (1). The only tricky parts are:

1. Computing how long the big vector should be. This allows you to define it (and hence reserve the required memory space) before the construction begins.
2. Keeping track of where the next note should be added into the big vector. (Use a pointer?)
3. Avoiding running out of memory!

You need to think about the above items before coming to the lab.

1. Modify your program `mkmusic.m` to generate the waveform for the whole score and then store your waveform as a .wav file to your directory. Use the windows audio player to play it. (Note you can't have the file opened by two programs at the same time. So if your MATLAB program is to write to the file, then you must first close it in any other program.)
2. Test your program on a short score first (e.g. `odetojoy.m`). Debug and then try `mornmood.m` and `furelise.m`. Play your version of Für Elise for the TA.
3. Add reverberation by shifting the signal by various delays, scaling them with decreasing weights for longer delays and adding them to the original signal. Try to find values that improve sound quality. Record them below:

4. Want to do more? You should be able to easily modify your music program to synthesize a score with several voices. The files `odeto2.m` contains the notes for a second voice for Ode to Joy. See if you can synthesize the music and play it.