EE 341, Fall 2008

Handout for Lab 1: Elementary Music Synthesis

How to generate a note

It may seem trivial, but probably the best point to start this Lab is by writing some Matlab code that will generate and play a note of music. In this Lab, a musical note is a sinusoid of a certain frequency (which?), a certain duration (how many seconds, or how many samples?), and sampled at a certain sampling rate (which one?). To create the sinusoid in Matlab, you need to choose values for the three parameters. You can then use those parameters to generate a vector that you can use as input to Matlab's `sin` function (see `help sin` for details). Once you've created the sinusoid, you can play it using the `sound` function (see `help sound` for details).

How to concatenate notes and rests

Now that we know how to create a note, the next step is to concatenate a series of notes, separated by short rests. For this, you can use the square brackets ("`[`" and "`]`", see `help horzcat` and `help vertcat` for details), or if you prefer, the `cat` function (see `help cat` for details). The rest between notes is just some silence, that you can generate with the `zeros` function (see `help zeros` for details). The amount of rest doesn't matter much, experiment a little bit to see what sounds ok (hint, it doesn't have to last longer than a quarter of a second, and probably a lot less).

Hint: using a for-loop may save you time

When creating the whole song, you can of course write the entire concatenation of all the notes and rests as one long Matlab statement. But it may be easier and save you time if you used a `for`-loop for it. In that case, you would need to store the frequency and duration of each note in one (or two) array(s), and then do something like this:

```
rest = … % define the rest between notes here
song = []; % start with an empty song
for i = 1:N,
  thisnote = generatenote(frequency(i),duration(i),samplingrate)
  song = … % concatenate this note and a rest to the song
end;
```

Of course, you need to properly define all the variables and arrays, write the generatenote function and fill in some blanks here.

How to apply an envelope to a note

To improve the quality of the note, we apply an ADSR envelope to it (see Lab instructions for details). This consists of 2 steps: first we must construct the envelope, and then we must apply it to the note. The envelope is a vector that has the same length as a note, and that has values between 0 and 1, indicating the volume of each sample of the note. To generate the linear parts of the envelope, you may find the `linspace` function

helpful (see `help linspace` for details). It's probably easiest to generate each linear piece first, and then concatenate them to form the whole envelope. Question: How would you do the envelope with a decaying exponential? To apply the envelope to the note, you can use the "`.*`", or pointwise multiply operator (see the bit on Array operations under `help punct`).

How to create overlapping notes

The last improvement to the quality of the song is to use overlapping notes. Would you still use concatenation of notes to accomplish this? If so, how? If not, why not? How would you do it instead? (Hint: You can mix two signals by adding them together. So, if you create one signal that is one note followed by a bunch of zeros, and another signal that is a bunch of zeros followed by another note, what would the result be? How would you use this technique to construct the entire song?)

Hint: using "subfunctions" may save you time

To keep your solution for each assignment in a single file, and to avoid repetition of commands, you could use Matlab's support for subfunctions. A subfunction is a function inside a function, for example:

```
function main
apples = 3;
oranges = 4;
result = compare(apples,oranges);
disp(result);

function same = compare(fruit1,fruit2)
if (fruit1 == fruit2)
  same = 1;
else
  same = 0;
end;
```

When you save these statements in a single m-file, the `compare` function then acts as a subfunction to the `main` function. It will only be known to and can only be called from the statements in the main function, and nowhere else. Note: you can not have subfunctions in a Matlab script (that is, an m-file whose first statement is not "function …"). Therefore, just use a dummy function definition that has no input and output parameters at the beginning of your m-file, such as "function main" in the example above.

Turning in your work

Besides submitting your m-files using the online E-submit tool (follow the "Turn-in area" link on the class webpage), you must also submit a hard-copy write-up of your work. In this write-up, briefly discuss the problems you encountered, the solutions that you came up with, and any choices you made regarding length of rests, details of the envelope, etc. Also, answer the (guiding) questions asked in this hand-out.

Good luck!