

Physics 22 - The Physics of Musical Sound

Second MATLAB Workshop Notes

Carl Grossman

During this session you will learn how to write programs in MATLAB that will ease the task of making complex calculations. For one, it will save you a lot of typing because the same set of instructions can be read over and over again from a file. In addition, there are many commands that you can't use in a line-by-line mode. For example, a loop that makes many similar calculations dozens of times is easily done in a program but is very tedious to do one at a time. We will need the loop commands in the first lab to construct complex sounds. Let's start with an example of setting up a melody and adding more stuff to it. It's not too difficult to do this by hand, without a program, but adding new melodic elements is a cinch when everything is in a program. We'll start with the usual line-by-line entry, like the first workshop, to create a major scale (just intonation). By the way, the notes here are a quarter of a second long (2,500 samples with a sample rate of 10,000 samp/sec) which, in musical terms, are eight notes in 4/4 time and a meter of 120, sixteenth notes in 4/4 with a meter of 60, et cetera.

```
>> x = 1:2500
>> x = x/10000;
>> do = sin(2*pi*440*x);
>> re = sin(2*pi*9/8*440*x);
>> mi = sin(2*pi*5/4*440*x);
>> fa = sin(2*pi*4/3*440*x);
>> so = sin(2*pi*3/2*440*x);
>> la = sin(2*pi*5/3*440*x);
>> ti = sin(2*pi*15/8*440*x);
>> do2 = sin(2*pi*2*440*x);
>> z = [do re mi fa so la ti do2];
>> sound(z, 10000)
```

All of this could have been read from a file and played one line at a time (I'm tired of the computer expression "executed," it's too gory, so I'll use the more musical term play). In the FILE menu select New followed by M-File (hold the mouse down because it is a double menulist). This will create a new window that is just like a simple word processor; you can type, cut 'n paste and erase just like you were writing a paper. You can even cut 'n paste what you have already entered from the workspace into your new file window. Shortly after entering some of your program, save it as a ".m" file (a.k.a. dot-em file). In other words, give your file a name with the ending .m and MATLAB will recognize it as a program. You have to be careful not to use the name of an existing .m file or you will get errors or weird results. For example, I called this program Bach.m for reasons you will see shortly. The classroom computers have user areas where you can save stuff temporarily, but you will want to save it on a disk. If you don't have a disk, I'll put your programs on the Classes Server. After pasting in your earlier commands, edit out the prompt characters () and any other junk so that the .m file is pure commands. You can add comments if the line starts with a % sign. The window should have something like this;

```
x = 1:2500;
x = x/10000;
% Comment - Define the A major scale with the arrays do - ti and do2.
```

```

do = sin(2*pi*440*x);
re = sin(2*pi*9/8*440*x);
mi = sin(2*pi*5/4*440*x);
fa = sin(2*pi*4/3*440*x);
so = sin(2*pi*3/2*440*x);
la = sin(2*pi*5/3*440*x);
ti = sin(2*pi*15/8*440*x);
do2 = sin(2*pi*2*440*x);
% Make an ascending scale in the variable z and play it
z = [do re mi fa so la ti do2];
sound(z,10000)

```

You can run the program two ways. Simply choose Save and Execute from the DEBUG menu and the program will run (or produce errors if there are any bugs in it which, at first, is more likely the case). Another way to run a program is to first save it and then call it with a command from the MATLAB Command Window. Just type the file name without the .m on the end. (Technical detail: MATLAB looks for .m files in specific folders which are defined by choosing the Set Path menu-item in the File menu. I will set the machines up to find the files in the user folder.) In my case, I just type Bach and the program is played. Run your program and make any changes if there are errors. Now edit your program to play a phrase from a Bach Two Part Invention instead of the scale:

```

z = [do re mi fa re mi do so so do2 do2 ti ti do2 do2];
sound(z,10000)

```

Since this is a two part invention, let's add the second part. The first measure is a rest, so I define a rest array of zeros (there are other ways to do this, but this works). Edit your program to include the following:

```

st = 0*do;
z2 = [st st st st st st st st do re mi fa re mi do];
z = (z + z2)/2;
sound(z,10000)

```

Now you can use Save and Execute or type the command:

```
>> Bach
```

(or whatever you call your program)

So far, we have only used the program mode to store up command sequences in larger lists, but the complexity is similar to what you would do in a line-by-line, command mode of operation. The next task is to implement real programming structures in MATLAB. Most computer programming languages have the same basic elements, one of which is called the FOR LOOP. FOR LOOPS will do a series of calculations over and over for a desired number of iterations and keep track of how many times the loop was played. The following program uses a loop to calculate the overtone series up to 20 harmonics then forms an arpeggio (sequence of intervals of a third or larger) and plays it.

```

time = 1:2000;
time = time/10000;
f= 110;
wave = sin(2*pi*f*time);

```

```

for n=2:20
freq = f*n;
wave = [wave sin(2*pi*freq*time)];
end
sound(wave,10000)

```

You can also select the frequency from an array. For example, lets define a new array called f.

```

>> f = 110:110:20*110;

```

The following program reads the frequencies from the array f. Notice that this is set up so that you can define any set of frequencies you want and the program will play them in order.

```

time = 1:2000;
time = time/10000;

wave = sin(2*pi*f(1)*time);
for n=2:length(f)
wave = [wave sin(2*pi*f(n)*time)];
end
sound(wave,10000)

```

Instead of playing the notes in order, let's add them together inside the loop and make a new waveform. Be sure to save this with a different name, otherwise you might choose the Save and Execute command and overwrite your arpeggio program.

```

time = 1:2000;
time = time/10000;

wave = sin(2*pi*f(1)*time);

for n=2:length(f)
wave = wave + sin(2*pi*f(n)*time);
end

wave = wave/max(wave);
sound(wave,10000)

```

The last thing I want you to play with is the amplitude function, both in the form of varying overtone combinations and in the form of the envelope function. Here is an example of different overtone amplitudes. The pause command waits for you to hit the return key.

```

time = 1:20000*2;
time = time/20000;
f= 110;
wave = sin(2*pi*f*time);

for n=3:2:100
amp = 1/n;
freq = f*n;

```

```

wave = wave + amp*sin(2*pi*freq*time);
end
wave = wave/2/max(wave);
plot(wave(2:600))
pause

```

```

sound(wave,20000);

```

The last amplitude exercise is the mess with the envelope function

```

t = 1:4000;
t = t/10000;
tatt = .2;

x = sin(2*pi*440*t).*(1-exp(-t/tatt)).*(1-exp((t-.5)/tatt));
z = x;
x = sin(2*pi*440*9/8*t).*(1-exp(-t/tatt)).*(1-exp((t-.4)/tatt));
z = [z x];
x = sin(2*pi*440*5/4*t).*(1-exp(-t/tatt)).*(1-exp((t-.4)/tatt));
z = [z x];
x = sin(2*pi*440*4/3*t).*(1-exp(-t/tatt)).*(1-exp((t-.4)/tatt));
z = [z x];
x = sin(2*pi*440*3/2*t).*(1-exp(-t/tatt)).*(1-exp((t-.4)/tatt));
z = [z x];
x = sin(2*pi*440*5/3*t).*(1-exp(-t/tatt)).*(1-exp((t-.4)/tatt));
z = [z x];
x = sin(2*pi*440*15/8*t).*(1-exp(-t/tatt)).*(1-exp((t-.4)/tatt));
z = [z x];
x = sin(2*pi*440*2*t).*(1-exp(-t/tatt)).*(1-exp((t-.4)/tatt));
z = [z x];
sound(z,10000)

```