Laboratory 6: Project Lab

1. Objectives

- Develop proficiency at solving a real-world DSP problem of your choice
- Develop Matlab independent programming and debugging skills
- Practice project presentation skills

2. Problem Choice

In the previous labs you have learned how to manipulate and analyze signals in both the time and frequency domains. This project laboratory will give you a chance to tackle a structured problem in DSP of your own choosing. You may either choose one of the projects below or develop your own. If you choose the latter option, you must first clear the projects with me.

Project Ideas

- Music Synthesis - sample a unique sound and figure out how to frequency-scale it, then play a melody using the frequency-scaled sounds. Make your own virtual fuzzbox or keyboard synthesizer.
- Speech Pitch Identification – change the frequency of a sampled section of speech to make a male's voice sound female or vice versa, without making the pauses between words sound unnaturally stretched or shortened. Hint: consider Matlab's resample function, with some programming to identify the pauses.
- Speech To Music – sing to make a short WAV file, and replace your voice with an instrument sample of the correct pitch. This is a combination of the top two projects. You can either do it for a single voice note, or tackle the harder problem of doing it for a short melody (for which you'll need to separately process several short-timed segments).
- Music Identification – take Lab 5 to the next step by creating a function that lets one sing a simple melody and have the program output a string showing what note is sung at every one-second interval.
- Instrument Identification – as above, but using an instrument, or perhaps a simple instrumental section of an MP3 recording.
- Heartbeat Recognition – create an M-file that takes the EKG recording used in the earlier lab and identifies the time location of each heart beat (ask about use of a *matched filter* to simplify this)
- Watermarking Of Audio Signals – develop a system that can embed copyright information (for this demonstration, say a 4bit digital sequence) into an arbitrary WAV file that does not noticeably alter the sound. For instance, the test of the algorithm might be to take a new recording, an arbitrary 4 bit sequence, watermark the recording with it, and then see if it can be read back. Ideally it would not be destroyed by MP3 compression.
- Acoustic Source Localization – develop a system that takes a stereo signal and determines the angle from which the sound is arriving. Hint: wavread returns a 2-column matrix for a stereo WAV, with the left channel in column 1 and the right channel in column 2. Think about what the difference of the phase angle of each DTFT's represents. This has applications in sonar imaging of submarines, radar imaging of planes/missiles, and robotics.

- Voice Removal – create an M-file that takes a short WAV recording of music that has vocals and instrument sounds, and removes just the vocals. Hint: consider using either using a bandpass filter designed using fdatool or else phase localization as described above.
- Hearing Impaired Simulator – An unfortunately common congenital deformation is frequency-selective hearing impairment. While the technology to diagnose the impairment at birth is impressive (the strength of the neural transmissions to the brain from the cochlea are measured by the traveling electromagnetic waves they emit while the ear is stimulated by tones of various frequencies), it would be helpful for the child's parents and speech therapist to understand exactly how the child hears sounds. Design a system that takes the output from an audiologist's chart (a matrix of frequencies in the first column and hearing loss at those frequencies in the second) and design a system that lets the user filter a WAV recording of his/her voice to simulate what the child hears. Essentially this involves creating an equiripple filter. A typical audiology report is shown in the Appendix.

3. Additional Ideas

There are three additional resources that may provide significant help, direction, and ideas to make your project more professional in appearance.

Debugging tools
The Read the Matlab help section on using the integrated debugger. In short, the debugger lets you place a "breakpoint" in your code. This looks like a red circle in your m-file editor and can be placed or removed by clicking next to a line of code in the thin gray strip between the line numbers on the left and the code on the right (Figure 1).

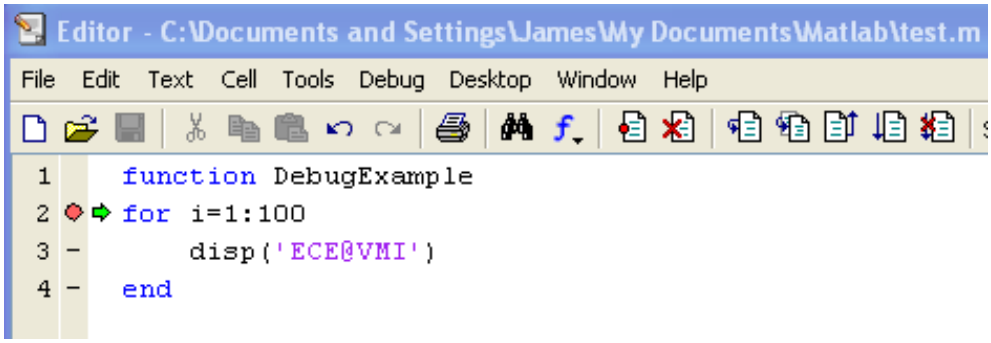

Figure 1: A section of the Matlab editor window, showing a breakpoint set at line 2. Click the breakpoint symbol to remove it or set it. The debug toolbar is shown to the right; only the final grouping of 5 functions are commonly used. They are, from left to right, single step over, single step into, step out, continue, and stop debugging.

When you run the m-file, execution will halt where you set the breakpoint. You can now examine all of the variables by either hovering the cursor over a variable name in the editor or by going to the main Matlab command window. There you will notice that the prompt changed from its usual `>>` to a `K>>` prompt to let you know you are in debug mode. If your function created a variable called `index`, for instance, typing `index` at the `K>>` prompt will make Matlab respond with something like

```
>> ans = 3
```

if the variable `index` was 3 at the moment that the breakpoint was reached. You can then step through the program execution line-by-line by clicking the single-step-over icon (see Figure 1). Single step over keeps execution in the current m-file; if you are debugging from m-file foo.m and foo calls the 1000-line m-file bar, single-step-over runs the entire function bar with the single click. Single-step-in does the opposite; it opens up bar.m and continues debugging operations at the first line of bar. Step out is less-commonly used; if you step out of foo and into the first line of bar, and see that the error you are tracking is not in bar, step out brings you back into foo. Continue makes the m-file

execute forward automatically until it either hits another breakpoint (you can have several) or completes executing the program. Lastly, the stop debugging button halts debugging and clears the temporary variables created out of the workspace. Read Matlab's help for more information.

GUI development using GUIDE

Matlab can create very impressive GUI's using GUIDE. You will need to learn how to use callback functions to do this; you will learn quickly if you have had prior experience with object-oriented languages. Read the Matlab help on GUIDE for details. The idea behind GUI programs and callbacks is that the GUI has many places to put code. One place automatically runs its code when the GUI is started. Every user interface element also has a place to store code that executes whenever it is touched. In the example in Figure 2, every slider and radio button when touched will call one master function. The master function queries each slider and radio button to determine how they are set, does some computation, and then plots the results to the graph windows.

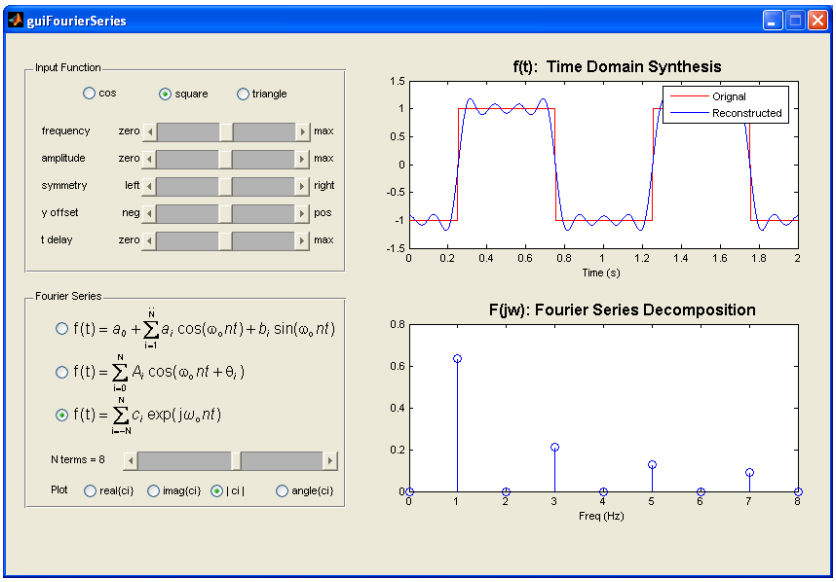

Figure 2: A GUI (Graphic User Interface) made in Matlab using GUIDE. Although not easy to make, GUI programs provide the most professional appearance. This one is fairly complex, although they can be as simple as just an input box with an output text string.

FDATool

The most impressive and easy-to-use tool I have discovered for designing digital filters is the FDATool (Filter Design and Analysis Tool) in Matlab's Signal Processing Toolbox. To learn how to use it, take a guided tour by typing "demo" at the Matlab prompt and then choose Toolboxes, Signal Processing, Filter Design and Analysis, and then Run This Demo. Don't choose Toolbox, Filter Design; that is not part of the signal processing toolbox and instead covers a variety of advanced and rather specialized techniques typically covered in graduate school level courses in signal processing.

FDATool is an interactive, GUI-driven applet that makes short work of designing digital filters in Matlab. I suggest you use FIR equiripple types and experiment with the required sampling rates. For a given set of filter requirements, as you reduce your sampling frequency you will also reduce the required order of the filter…to a point. After that point the filter algorithms will fail to converge, and you will know that you have designed an optimal filter in the sense that it is the lowest order possible that meets your criteria. The demonstration described above gives an excellent overview to this tool.

4. Report

Because most of these DSP projects are inherently multimedia in nature, a written report is not the most natural medium to communicate your work. Instead, one lab period will be devoted to project demonstrations. You can check out a laptop from the department with Matlab installed to present, or you can use a personal laptop. I encourage you to have a prepackaged demonstration in PowerPoint rather than relying exclusively on Matlab. Your grade is based upon the following scale:

		D		C		B	A
		Very Weak	Weak	Low Ave	High Ave	Strong	Exceptional
33.3%	Difficulty						
33.3%	Works?						
33.3%	Presentation						

Definitions:
Difficulty: How challenging is the problem proposed to be solved? For instance, is it a strict filter application, or does it require more extensive programming? Does it use a GUI interface?
Works?: Does it work? If so, how well? If not, how close?
Presentation: Professionalism, graphics quality, enthusiasm, lack of distracting mannerisms, timeliness

5. Appendix

Audiology chart example. Each entry for hearing gain may range from 0dB (normal) to -120dB in 10dB intervals. The child shown below suffers from moderate to severe high frequency hearing loss, and as a result will hear some low vowel sounds well, but not higher-frequency consonant sounds such as the "sh" sound you studied in Lab 5.

frequency	gain (dB)
20Hz	0
50Hz	-10
100Hz	-10
200Hz	-20
500Hz	-20
1kHz	-30
2kHz	-40
5kHz	-50
10kHz	-60
20kHz	-60