Multimedia
Module No: CM3106
Laboratory Worksheet Lab 1 (Week 2): MATLAB Basic Digital Signal Processing: Filters and the Fourier Transform

Prof. D. Marshall

Some of exercises are revised from CM0268. Some exercises (Fourier) are new. *You should make sure you are familiar with all the concepts below as these are essential for understanding the forthcoming lectures of Audio Synthesis (and later MPEG) and the imminent coursework.*

After working through this worksheet you should be familiar with:

- The creation, display and audio output of basic waveforms in MATLAB.
- The effect of basic lowpass, highpass and bandpass type filters on simple waveforms and audio.
- The creation and application of Infinite and Finite Impulse Response (IIR/FIR) Filters in MATLAB.
- The basic theory of frequency space transforms
- The basic operation of Fourier in MATLAB.
- How to display and visualise *frequency space* in MATLAB
- How to apply filters in *frequency space*

All Lab Materials available at:
http://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/PDF/tutorial.html

All Lab class support files available as a zip download

None of the work here is part of the assessed coursework for this module

MATLAB Basic Digital Signal Processing

1. **Basic Waveform creation and display**: Create respective sine, cosine, square and sawtooth wave forms each at the frequencies of 220 Hz, 440 Hz, 880 Hz. (You may choose appropriate sample frequencies and duration).

   - Display these waveforms in individual MATLAB figures.
   - Play each of these waveforms using MATLAB audio output.
   - Display each class of waveform (*i.e. all sines, cosines* etc.) in a single MATLAB figure — one figure for each class. Make sure each waveform is easily read in the figure.
   - Using `subplot` display all individual classes (as above) in a single MATLAB figure.
   - **Add** all the component sine wave forms together and display and output the sound of the resultant wave. Do the same for cosine, square etc.
     (**Note**: This is the essence of *Additive Synthesis* which we will study shortly.)
   - **Add** all the component sine and cosine wave forms together and display and output the sound of the resultant wave.
     (This is clearly also an example of *Additive Synthesis*.)

2. **Infinite/Finite Impulse Response Filters**: With reference to the lecture MATLAB demo *IIRdemo.m* and *subtract_synth.m* and also built in MATLAB *Signal Processing Toolbox* help and demos:

   - Create a 2nd order Butterworth IIR/FIR **lowpass** filter (see `help butter`) and apply it to the waveform as in subtract_synth.m. Display the result.
   - Create a 4th order Butterworth IIR/FIR **lowpass** filter (see `help butter`) and apply it to the waveform as in subtract_synth.m. Display the result. Compare it to the 2nd order filter above.
   - By loading in the inbuilt *handel* audio data (`load handel`) or importing any other wav file:

- Apply a 2nd order Butterworth IIR **lowpass** filter of varying frequencies form say 32 Hz to 16 KHz at intervals of 32, 125, 500, 1K, 2K, 4K and 16K Hz. Display and audition the audio results obtained by the filter.
- Create a 2nd order Butterworth IIR **highpass** filter and apply to the audio data over the same frequency intervals as above. Display and audition the audio results obtained by the filter.
- Create a 2nd order Butterworth IIR **bandstop** (see `help butter/buttord`) filter between respective frequency intervals as above.
- Create a 2nd order Butterworth IIR **bandpass** (see `help buttord`) filter between respective frequency intervals as above.

(**Note**: This is the essence of *Subtractive Synthesis* which we will study shortly.)

Fourier Transform

1. Compute the **forward Fourier transform** of $f(x) = \cos(2\pi f x)$, for some given frequency, f, (using Phasor Notation), change the phase (rotate in Fourier Space) of this result by $90\circ$ anticlockwise (*Hint: Think Phasors*) and compute its inverse Fourier transform. Display the resultant waveform. What should this waveform be?
(see fft_phase_eg.m example code.)

2. `fftshift()`: Look up MATLAB `help fftshift` and also `doc fftshift`. The use of this function is probably one the most confusing aspects of understanding Fourier theory and its implementation in MATLAB (infact this type of computation arises in many other FFT implementations also). Most computations of FFT represent the frequency from 0 – $N - 1$ samples with corresponding frequencies ordered accordingly. Therefore the 0 frequency is not really the *centre*. We frequently like to visualise the FFT as the **centre of the spectrum**. `fftshift()` rearranges the outputs of fft, fft2, and fftn by moving the zero-frequency component to the center of the array.

This is possible due the invariant *shift property* of the Fourier Transform (*http://en.wikipedia.org/wiki/Discrete_Fourier_transform*)
Note: for filtering and other similar processing it is does not matter which method of FFT ordering you use so long as you can compute the frequency '*coordinates*' correctly for each method.
(See lowpass.m for example use of `fftshift()`)

(a) Create a white square on a black background image (Hint: use `ones()` and `zeros()` computer the FFT of this image. Display the magnitude spectrum of the image with and without `fftshift()` applied. Note the differences in display.

(b) Load in an Image into MATLAB, compute its FFT. Display the magnitude spectrum of the image with and without `fftshift()` applied. Note the differences in display.

3. Create a sine wave of frequency 440Hz and of 1/4 second duration at sample rate 11.025KHz sample rate and store it in a MATLAB array, perform the Fourier transform on the array. Plot the a graph of the Fourier *magnitude spectrum*. Label the x-axis with suitable frequencies. Check you have the peak of the plot labelled correctly. Plot a *spectrogram* of the array. (Hint: to force efficient *fast Fourier transform* computation round the array size to a power of two before you call the MATLAB `fft()` function — always try to do this)
(See fourierspectraeg.m for example)

4. Create a **square** wave of frequency 440Hz and of 1/4 second duration at sample rate 11.025KHz sample rate and store it in a MATLAB array, perform the Fourier transform on the array. Plot the a graph of the Fourier *magnitude spectrum*. Label the x-axis with suitable frequencies. Check you have the peak of the plot labelled correctly. Plot a *spectrogram* of the array.

   - Now perform a *lowpass* filter in this waveform. Set a cut-off frequency of around 100-200Hz, experiment with different cut-off frequency values. (See lowpass.m for an example)
     (**Note**: This is the essence of *Subtractive Synthesis* which we will study shortly.)

5. Load some audio data, compute the FFT of the audio data apply an ideal *lowpass* filter and butterworth filter with a cut-off frequency of 240Hz to the data. By plotting the waveforms and listening to the audio output of both files compare the two outputs.

6. Load some audio data, compute the FFT of the audio data apply an ideal *highpass* filter and butterworth filter with a cut-off frequency of 240Hz to the data. By plotting the waveforms and listening to the audio output of both files compare the two outputs.

7. Load some audio data, compute the FFT of the audio data partition the data into respective filter bands of 0-200Hz,200-400Hz,400-800Hz,800-1200Hz and 1200Hz and above. Audition each band separately.
   (**Note**: This implements a *bandpass* filter — This technique will be useful later for MP3 audio compression and related techniques but with different frequency bandwidths.)

8. Using the `Daphex.m` and `im2sound.m` code as an example, use any photograph of your choice and embed it in a spectrogram. Experiment with `im2sound`'s parameters to note the different spectrogram and also output audio. Note: please email any suitable good examples to me — include the original image and the parameters used for `im2sound`, or any modified code used to create this result.

Further Practice

MATLAB Basic Digital Signal Processing

1. Using MATLAB's Symbolic Toolbox `fourier()` function — see `help/doc fourier` — evaluate the *Fourier transform* of the following functions:

   (a) $f(x) = xe^{-}|x|$

   (b) $f(x) = e^{-x^2}$

   (c) $f(x) = e^{ax^2}$, where a is a constant.

   (d) $f(x) = x^n$, for some integer, n.

2. Using MATLAB's Symbolic Toolbox `ifourier()` function — see `help/doc ifourier` evaluate the *inverse Fourier transform* of the following functions:

   (a) $F(u) = e^{\frac{-u^2}{2}}$,

   (b) $F(u) = e^{-|u|}$

3. Create a **sawtooth** wave of frequency 440Hz and of 1/4 second duration at sample rate 11.025KHz sample rate and store it in a MATLAB array, perform the Fourier transform on the array. Plot the a graph of the Fourier *magnitude spectrum*. Label the x-axis with suitable frequencies. Check you have the peak of the plot labelled correctly. Plot a *spectrogram* of the array.

4. Load an audio sample of data into a MATLAB array. Perform the Fourier transform on the array. Plot the a graph of the Fourier *magnitude spectrum*. Label the x-axis with suitable frequencies. Check you have the peak of the plot labelled correctly. Plot a *spectrogram* of the array.