BIOM9660 LABORATORY:
INTRODUCTION TO SOUND PROCESSING

Introduction:

Building an implantable bionic device is really only part of the work that must be done in order for patients to gain benefit from the device. The question of what to do with the implant once it is implanted is extraordinarily important. This is true for all implantable bionics, irrespective of their application. In this introductory laboratory, we will explore sound processing for an auditory neuroprostheses – the so-called 'bionic ear'.

For many years, multi channel cochlear implants for the hearing impaired have remained largely the same. Today they are more sophisticated, and have more features such as reverse telemetry and mechanisms to record the neural responses they elicit, but they remain with the same number of stimulation channels as they had two decades ago. Electrically, they behave almost unchanged in their stimulus waveforms.

In stark contrast to the electrical behaviour of cochlear implants, the nature of how this electrical behaviour is utilised has changed significantly over this same period. Indeed the same 20 year-old implanted device can now be instructed to give electrical stimulation that delivers information to the auditory system with truly remarkable results. Where once the ability to converse over the telephone using a cochlear implant was rare, it is now commonplace. The changes that have taken place are largely to do with sound processing. An introduction to this vast field of research is the subject of this laboratory – that is, how does one take sound and turn it into electrical stimulation?

Part A: Exploring Frequencies

Sampling and the Nyquist Rate:

The healthy, young, human ear can resolve frequencies within a range of roughly 20 to 20,000 cycles per second (Hertz or Hz). As we grow older, our ability to hear the extremes of this range diminish.

Students of mathematics will recall the so-called 'fourier transform' which allows for the transformation of time-domain signals - such as a '.WAV' sound recording which, if plotted, would typically be expressed in terms of amplitude and time - to the frequency-domain. A frequency-domain plot will show how much of a signal (e.g. the same '.WAV' file as before) comes from a particular frequency or frequencies.

For a pure frequency (e.g. a sinusoid at a particular, constant frequency), the frequency-domain plot will show only one frequency as the contributor to the signal. A sound such as the word, “Hello” will contain many frequencies simultaneously and is a combination of multiple pure frequencies occuring at once.

Procedure:

Open Octave (a free Matlab clone) and write a script to produce a pure, 5 Hz signal that lasts for two seconds. Recall that you can write a script in a text editor such as "notepad" and save it as (e.g.) **`my_script.m`**. To run the script within Octave, simply type:

```
octave > my_script
```

HINTS:

```
% signal frequency
frequency = 5;                                  % Hz
% duration of the signal
duration = 2;                                   % seconds
% Sampling frequency (the number of datapoints per second)
fs = 10;                                        % Hz
% Set up a time vector with the correct number of data points
time = [0:(duration*fs)-1];
% x is some signal, in this case it is a frequency Hz cosine.
x = cos(frequency*2*pi*time/fs);
buffer = sprintf('TIME DOMAIN PLOT OF PURE FREQUENCY SIGNAL');
plot(time/fs,x,'-x'),title(buffer), axis([0 2 -1 1]);
xlabel('time (s)');
ylabel('Amplitude');
```

Save this plot for safe keeping. You should do this with all plots you generate within this lab (once you've finalised their form).

HINT:

```
print('my_plot.jpg', '-djpg');
```

WARNING: You will not be warned when overwriting existing files! (except for right now).

Questions/Post Processing:

1. Does the plot look like a sinusoid? If not, what can be changed to make it look right?
2. Why is fs = 10?
3. Plot, save and upload the same sinusoid when fs = 5, 9, 11, and 50.

   As always, save all your plots for further reference/study.

Making Sound:

What you have observed in the exercise above highlights the Nyquist Rate – that is, the rate at which one must sample in order to unambiguously resolve all elements of the signal (cycles in this case). The Nyquist rate is twice the frequency of a given signal (or the highest frequency of the signal) . Note that while the sinusoidal shape is lost, each cycle was saved in the fs = 10 plot. In the fs = 5 plot, all cycles were lost. Sampling at the Nyquist rate doesn't guarantee a reconstruction of the signal, it only guarantees that you won't miss the the number of cycles. In fact, unless the sampling frequency is infinite, any attempts at reconstruction of an original, non-trivial waveform

will result only in an approximation of that signal.

As noted above, a normally-hearing human can hear frequencies between (roughly) 20 and 20,000 Hz. In order to be able to capture the 20 kHz frequencies, the sound must be sampled at (at least) 40 kHz. Ever wonder why compressed music files (e.g. 'MP3s') are often sampled at 44 kHz?

If the sinusoid we made above were to be fed into a speaker, we probably wouldn't be able to hear it because it is sub-sonic. We will now make a sound that we (hopefully) can hear.

Procedure:

1. Modify your script above to produce a signal of two seconds duration at a pure frequency of 4 kHz. Sample it at 44 kHz. Plot and save the first 0.005 s (HINT: see the 'axis' command).
2. Save the sound as a '.WAV' file.

   HINT:

   ```
   wavwrite(x',fs, "my_wave.wav"); % x' = transpose of x (rows vs cols)
   ```

3. Play the sound in a media player or equivalent – try not to annoy your neighbours.
4. Find the frequency spectrum by taking the absolute value of the fourier transform of the sound vector x.

   HINTS:
   The fourier transform produces real and imaginary components. As such, the *magnitude* of the signal (the square root of the sum of the squares of the real and imaginary parts of the fourier transform) is capable of being plotted on a simple 2D plot.

   ```
   fft_x = abs(fft(x, length(x)));
   ```

   Plotting '`fft_x`' (the wav file) as-is will produce a plot with units of "fft magnitude" and "sample number". Since "sample number" isn't particularly meaningful, this should be changed to something more sensible. What `fft_x` really contains is the contribution each frequency makes towards the overall sound. Therefore, scaling the horizontal axis to read "frequency" instead of "sample number" makes a great deal more sense. The sampling frequency, '`fs`' has units of samples per second. Each element of `fft_x` has units of cycles per sample. Multiplying these causes "samples" to cancel, leaving behind cycles per second or Hz.

   ```
   frequency_range = fs * [0 : length(x)-1]/length(x);
   ```

   Plotting `frequency range` -vs- `fft_x` is readily achieved by the plot command:

   ```
   plot(frequency_range,fft_x);
   ```

   Any plot worthy of presentation must have a title, and labelled axes:

   ```
   title('FREQUENCY DOMAIN PLOT OF PURE FREQUENCY SIGNAL');
   xlabel('Frequency (Hz)');
   ylabel('FFT Magnitude');
   ```

   Finally, the only part of the frequency range that is of interest is the Nyquist Range, so we

truncate the plot at half the sampling frequency, and set the maximum amplitude to the maximum of `fft_x`.

```
axis([0 fs/2 0 max(fft_x)]);
```

5. Plot and save the frequency spectrum.

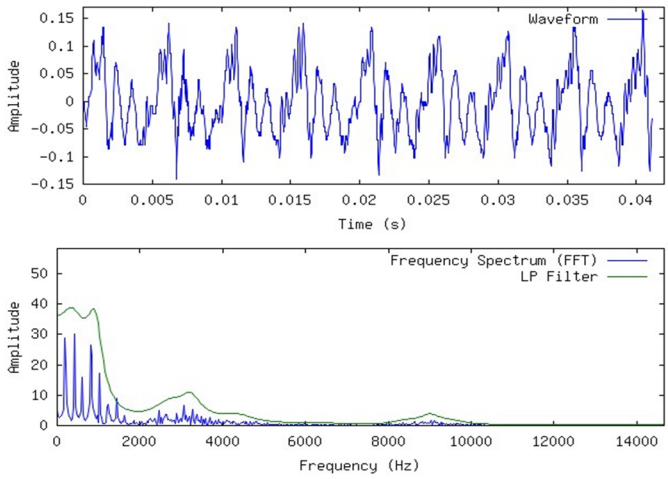
Questions/Post Processing:

1. What is the peak frequency of the sound?
2. Why are there no other frequencies present?

Part B: Formants

What is a formant?

The Fourier analysis illustrated in the sections above is problematic in capturing the relevant features of speech because it not only requires a relatively large sample, it also fails to find time-varying parameters that make up the subtleties of speech. Therefore, simply extracting the peak frequencies of a given signal provides insufficient information to represent the original sound unless it is a pure sound as we have seen above. Another way is needed.



If one were to construct a model of the human speech system, the logical approach would be to find a source of vibration to represent the glottis – an electronic buzzer for instance, and place this buzzer at one end of a tube to represent the vocal tract. The tube would ideally be of a soft and bendable material so that its geometry could be subtly changed to represent movements of the mouth, tongue, and lips. The buzzer would ideally have the capacity to change its intensity and frequency to replicate the glottis' capabilities.

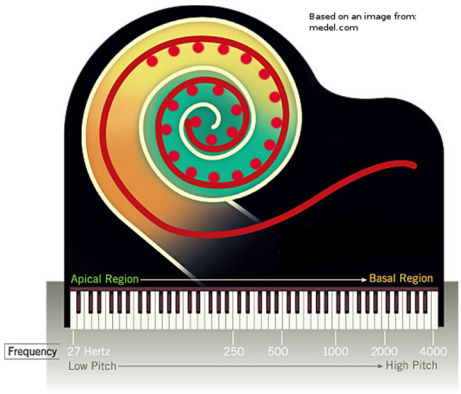
As the buzzer buzzes and the sound is emitted from the tube, there are resonant frequencies which characterise the sound. These resonant frequencies are referred to as ***formants***. The way we'll use to discover the formants is by way of a filtering technique called linear prediction which, conveniently, is contained within the `lpc()` function available in both Matlab and Octave. Suffice for the

purposes of this laboratory, we shall use this function as a tool to find the formants and not delve further into the issue. Should you wish to explore speech analysis in more detail, a more thorough study of linear prediction methods would be a good place to start.

The vowel sound “a” (as in “say”) is shown in the figure on the previous page. In the upper panel, the time-domain signal is shown. In the lower panel, the frequency-domain signal is shown (the jagged signal with many peaks and troughs) along with a smooth, linear prediction coefficient filter plot with some distinctive peaks at 350.4, 951.3, 2626.8, 3253.7, 4423.7, etc. and another at about 9 kHz. These peaks (some of which are rather subtle) are known as the formant frequencies and are referred to as f1, f2, etc.

The essence of the sound can be extracted from these formants and it is these formants that form the basis of speech processing strategies that have allowed cochlear implants to become so successful.

Recall that cochlear implants are 'tonotopically mapped' – that is, the position of each electrode within the cochlea corresponds to a particular frequency range. In the frequency spectrum on the previous page, the horizontal axis could be divided into several ranges, each range (with some overlap between them) would represent the location and therefore the frequency range of a single electrode within the cochlea.



While modern speech processing techniques are much more sophisticated than what we will explore here, the basics are:

1. formants correspond to electrodes[1]. In the actual stimulation strategies employed in the early days of cochlear implants, the stimulation was presented at a rate of F0;
2. by determining the formants of a given sound, the formants may be 'mapped' to the stimulation;
3. the charge to be delivered by the stimulus is determined by the amplitude of the original signal.

Procedure:

1. On the BIOM9660 Moodle, there are several sounds available for you to use in “.wav” format (see the “Technology” section of the Main Page). Choose one (or more) of these for use in exploring formants in the next steps of this laboratory.
2. On the same Moodle page there are some .m files. These are the files necessary to avoid the need for specialised toolboxes to do the sound processing. These files are distributed under the GNU license, and the authors are credited within the code. The files are contained within a zip file – unzip the file in the working directory you're using for Octave.
3. Build a new “.m” file to plot the sound in the time-domain. Much of the code you need is contained in the other .m file, but to read the '.wav' file sounds, use the following.

1. As the astute reader will have already noticed, it is possible (indeed probable) that one or more of the formants falls within the same range of frequencies elicited by a particular electrode. This is one of the reasons why the sounds from cochlear implants are not perfect, and why the appreciation of music among cochlear implant recipients is an important topic of research.

HINTS:

```
% clear memory in case matrices are already defined - very important!!!
clear;
% the filename to process
buffer=sprintf("filename.wav");
% find out sampling rate, etc.
[x, fs, bits]=wavread(buffer);
% x = waveform vector; fs = sampling frequency; bits = bits per sample
% Make sure that the file is in the correct format before proceeding
[n, nChan]=size(x);
if nChan > 1
      error('The type of the wave file must be mono (not stereo)');
end
```

Questions/Post Processing:

Save the plot for future reference/study.

PART C: Epochs

The plot generated in the last section shows the time-domain plot for the *entire* sound. As sound is a dynamic flow of changing frequencies over time, we cannot represent an entire sound all at once with the formants and expect to glean much (or any) useful information from it. As such, we need to consider small time-steps or *epochs* of a given sound, find the formants of each epoch, and deliver the important features of the sound within each epoch to a speaker or a cochlear implant in a step-wise fashion.

The sounds available on the BIOM9660 Moodle are typically two seconds in duration. If we (somewhat arbitrarily) choose 20 ms to be our epoch, then we will have 100 epochs within our two-second sound. The objective of this section of the laboratory is to divide the sound into 100 epochs, determine the formant frequencies for each epoch, and reconstruct the sound using only formant frequencies. Note that for a two second sound sampled at 44 kHz, each epoch will have 880 samples.

1. Building upon the .m file you started above, break up the sound into 100 epochs by adding the following to your .m file:

```
% How many epochs?
N=100 ;
% How many datapoints in each epoch?
points_per_epoch = length(x)/N ;
% Ditch any decimals if length(x)/N is not an integer!
points_per_epoch = floor(points_per_epoch) ;
for i=1:N
      % get an epoch
      epoch(i,:) = x(((i-1)*points_per_epoch)+1:i*points_per_epoch);
endfor
```

PART D: Formants of the Epochs

Now that we have 100 epochs, we need to find the formants of each epoch so that these can be used in the formation of the electrical stimulation to be delivered (preferably in real time) to the implant and be recognised as part of the longer sound.

The details of the following code are beyond the scope of this laboratory, but suffice to say that it will fit a curve to the spectral peaks of the sound contained within each epoch, and find the formants. To do this, add the following to your .m file.

```
% Find the formants
%
% The following is a 'rule of thumb' of formant estimation
number_of_coefficients = 2+fs/1000;
for i=1:N
      % determine a polynomial finding the spectral peaks
      % of the epoch using lpc
      spectral_peaks = lpc(epoch(i,:)',number_of_coefficients);
      r=roots(spectral_peaks);% find roots of this polynomial
      r=r(imag(r)>0.01);       % only look for + roots up to fs/2
      % convert the complex roots to Hz and sort from low to high.
      formants=sort(atan2(imag(r),real(r))*fs/(2*pi));
      % print first five formants (the rest are still stored in ffreq)
      for j=1:5
            ffreq(i,j) = formants(j); % save for later
            fprintf('Epoch(%d), Formant(%d) =%.1f Hz\n',i,j,formants(j));
      endfor
endfor
```

After debugging, one may wish to avoid the formants from being printed. It is safe to comment out the fprintf line above by placing a % symbol at the start of the line.

PART E: Loudness of the Epochs

At this point, the formants for each epoch are stored in the matrix: `ffreq`. It is now possible to generate sounds based upon these pure formant frequencies but first we need to know how loud each epoch should be. There are numerous ways to do this, some more sophisticated than others, but suffice for the purposes of this laboratory to determine the loudness based on the loudest sound within a given epoch of the original sound. Modify the .m file to include the following. The appropriate location to place this within the file is left to the student to determine.

```
amplitude(i) = max(epoch(i,:));
```

PART F: Building the sounds

The formants we have found above represent pure frequencies. Using just one of the formants to build a sound will of course present itself as an epoch of a pure frequency followed by another epoch of another pure frequency – and so-on until the end of the sound. Building this sound is rather simple. Recalling that N is the number of epochs in the sound...

```
for i=1:N
      % time of each sample
      time=(0:length(epoch(i,:))-1)/fs;        % sampling times
      % Construct a signal using f1 only to represent this epoch
      sound1(i,:) = amplitude(i) * sin(2 * pi * ffreq(i,1) * time);
endfor
```

At this point we have N, pure frequency sounds. Adding these together in series will approximate the original sound using only the first formant frequency. This can be done as follows:

```
% Add all the sounds together in series to build a new sound, 2s long,
% based only on the pure formant frequencies
%
wave1 = 0; % an empty matrix that we'll add to below
% Piece together all the epochs into one sound
for j=1:N
      wave1 = [wave1 sound1(j,:)];
endfor
% Save the sound to disk for subsequent playing
wavwrite(wave1',fs, "f1.wav"); % x' = transpose of x (rows vs cols)
```

Play the sound in a media player. Depending upon the nature of the original sound (e.g. whether or not it contained vowel or consonant sounds, etc.) the approximated sound using only the first formant may sound terrible, and bear little resemblance to the original. However, if we were to add another formant to the sound – that is, another pure frequency to each epoch, the sound may improve. Adding a second formant to the sound is rather simple:

```
sound2(i,:) = amplitude(i) * (sin(2*pi*ffreq(i,1)*time) +
sin(2*pi*ffreq(i,2)*time)));
```

This produces a sound containing the first two formants. Modify the code in this section to build five sounds (sound1, sound2, ...sound5) and save these sounds as f1.wav, f1f2.wav, ...f1f2f3f4f5.wav.

Questions/Post Processing:

1. Play the sounds starting with the f1 sound, and working up to the f1f2f3f4f5 sound.
2. What do you observe as you increase the number of formants?
3. How many formants are needed before the sound is decipherable?
4. The laboratory has a setup to produce your own two-second sound bytes. Try your own sounds with different vowels, sharp consonants (e.g. “bye-bye”) and make observations on these sounds and the processed sounds that come about as a result of the above.
5. Contribute sounds to the BIOM9660 course. If you speak other languages, please contribute two-second sound bytes in these languages. Note that tonal languages such as Chinese Mandarin are particularly problematic for cochlear implant sound processing. Can you comment on why this is the case?

PART G: Electrical Stimulation

If an electrode array comprising four electrodes from the BIOM9660 implant were to be implanted into a deaf person's cochlea, it has the capacity to restore useful auditory sensations.

From what has been discovered above, we have most of the necessary information to send electrical stimulation to the implant.

Recalling the mapping of electrode position to frequency (or a range of frequencies) in the cochlea, we can expect that this mapping varies from patient to patient based upon the depth of insertion of the electrode, and several other factors. Also, the sensation of some frequencies will be elicited from more than one electrode (perhaps those associated with positions within the cochlea that lie between two electrodes. For a particular patient, let us assume that some experimentation has determined that:

- stimulation of electrode 1 elicits sounds in the rage of 50 to 750 Hz
- stimulation of electrode 2 elicits sounds in the range of 500 to 1700 Hz
- stimulation of electrode 3 elicits sounds in the range of 1500 to 4000 Hz
- stimulation of electrode 4 elicits sounds in the range of 3500 to 11,000 Hz

Consider the sounds generated above, each containing epochs of 1, 2, 3, or 4 frequencies that represent the formants of the sound. These formant frequencies can be used to determine which one, two, three or four electrodes should be used in the delivery of electrical stimulation. While there are several approaches that can be taken, one is to fix a pulse width, and vary only the stimulus amplitude in order to convey the sound.

Questions/Post Processing:

1. From what you now know about formant frequencies and sound processing, write an Octave script that takes a two second .wav file, breaks it up into 100 epochs, finds the formant frequencies of each epoch, and determines which electrodes to use to stimulate the patient whose frequency data is described above. The output of the file should be a list of stimulation of the form:

   PULSEWIDTH, INTER-PHASE DELAY, INTER-STIMULUS DELAY
   ELECTRODE, AMPLITUDE
   ELECTRODE, AMPLITUDE
   ...
   ELECTRODE, AMPLITUDE

2. From the list of stimulation parameters, produce an “electrodogram” (time on the vertical axis, electrode on the horizontal axis, amplitude colour encoded). This will require some programming in Octave or Matlab, but the extensive hints shown here should help.

HINTS: Produce a vector of zeros called (e.g.) stimulus:

```
stimulus = zeros(N, Etrodes);
```

where `N` is the number of epochs, `Etrodes` is the number of electrodes.

Using `if` or `case (or switch)` statements, determine which electrode(s) will deliver the sound to the cochlear implant electrodes during each epoch. For instance, if the formant frequency f1 is 572 Hz, then the first electrode will be stimulated. Remember that each electrode covers a *range* of frequencies and more than one formant could require stimulus from the same electrode. If f1 = 572 Hz in epoch 10, following change would need to be made to the `stimulus` vector:

```
stimulus(10, 1) = amplitude(10);
```

If f2 in epoch 10 was 720 Hz, then the same electrode would be stimulated. If f3 in epoch 10 was 1511 Hz, then,

```
stimulus(10, 2) = amplitude(10);
```

And so on up to the fourth formant. Some epochs will have one, some will have two, others three or four electrodes that will need to be stimulated in order to generate the sound.

The Octave/Matlab command to produce the electrodogram plot is `imagesc`. For the above example,

```
imagesc([1,N], [1:Etrodes], (stimulus'));
```

Should do the trick. Remember to label your axes, save your plot in your notes.

```
colorbar;
```

Adds an elegant touch to the plot.

References:

[1] Atal, B. S. and Hanauer, S. L. (1971) "Speech Analysis and Synthesis by Linear Prediction of the Speech Wave", *J. Acoust. Soc. Am.*, 50, 637-655.