JAMES COOK UNIVERSITY

COLLEGE OF SCIENCE, TECHNOLOGY AND ENGINEERING

EE4000

MATLAB COCHLEAR IMPLANT SIMULATION PROGRAM

DANE LENNON
BACHELOR OF ENGINEERING (COMPUTER SYSTEMS)

ASSIGNMENT

WEEK 13 [MAY 16]

Table of Contents

Table of Figures

1 Introduction

1.1 Background

Speech processing involves the analysis of speech waveforms and sampling and filtering the waveform for electrode stimulation. One common method is called the Spectral Maxima Sound Processor (SMP) which is employed by Chochlear implants. This processing strategy is to amplify the signal before making use of a bank of 16 band-pass filters who's purpose is to split the waveform into 16 separate signals. Each signal is then passed through a Low Pass Filter (LFP) for envelope detection purposes before the 6 largest amplitudes of each signal is detected and rest discarded. Lastly, each maximum amplitude is compressed and encoded in preparation stimulation of the electrodes in the ear.

1.2 Aim

This practical was conducted in order to simulate the SMSP strategy and asses it effectiveness in sampling speech waveforms. A hearing aid was also developed and tested for accuracy [1].

1.3 Apparatus

1) Matlab R2014b (9.4.0.150421) – 64-bit (maci64) September 15, 2014

2 Procedure and Results

2.1 Part A: MATLab simulation for the SMSP strategy

2.1.1 Procedure

The following is pseudocode representation of the MATLAB algorithm needed to implement a simulated SMSP process.

1. Import the waveform into MATLAB
2. Resample the waveform at 16kHz
3. Compare the original waveform with the resampled waveform
4. Use the MATLAB *fdatool()* to calculate the 16 vectors pertaining to the filter coefficients of each center frequency
5. Filter the waveform through the 16 band-pass filters creating 16 channels
6. Pass each channel through a LPF with a cut-off frequency of 200Hz
7. Sample each channel every 4milliseconds
8. Find the largest amplitude of the sampled channels
    a. Sort the channels largest to smallest
    b. Take the first index of each vector
9. Normalize the magnitudes of each amplitude to ensure values are between 0 and 1
10. Plot the normalized amplitudes in an electrodogram

The above steps describe the process that is shown in Appendix A: MATLAB Code of Simulation of SMSP Process. Below is detailed description of the code and the reasons for implementing the simulation in this way. Figure 1 shows and extract of the algorithm between lines 1 to 15. This is the setup and and initialization phase where all variables are cleared from memory and any open figures closed. The waveform is loaded into the MATLAB workspace in lines 9. The *audioread()* function returns a variable *y* containing the waveform and a variable *Fs* that is the sample rate of the matrix *y*. The initial sampling frequency (*Fs*) was set to 44.1kHz. The waveform was then resampled in line 13 using a new sampling frequency (*FsNew*) set to 16kHz. The wave was resampled to a smaller sampling frequency in order to reduce the number of calculations required to perform. This new sampling frequency was chosen to ensure anti-aliasing didn't occur. The largest frequency in the waveform was found to be approximately 5.4kHz and Equation 1 shows that the minimum sampling frequency required to avoid antialiasing is 10.8kHz. A value higher than the minimum is chosen as the roll-off of each sample is not ideal and antialiasing can still occur at the minimum.

$$F_s = 2BW = 2\times5.4 = 10.8$$

EQUATION 1 - SAMPLING FREQUENCY

```
1   %EE4000 Assignment Task 1 - Dane Lennon
2
3   clc; %clear variables
4   %clear all; %clear memory
5   close all; %close all figures
6
7   %% initialization
8   %load the audio file
9   %y = sampled data (a column vector of 21364 long is returned)
10  %Fs = sample rate (44100 is returned)
11  [y, Fs] = audioread('/Users/danelennon/Documents/MATLAB/EE4000 - DSP/mbet.wav');
12  FsNew = 16000;
13  y1 = resample(y,FsNew,Fs); %resample the data at 16000Hz
14  numChannels = 16; %set the number of channels
15  filterOrd = 1607; % set the filter order
16
```

FIGURE 1 – INITIALIZATION

Figure 2 shown below is the MATLAB code from lines 17 to 39 which first plot the original waveform, then the resampled waveform and final a third plot compares both waveforms on the same axis (see section 2.1.2 for the plots of this data).

```
17  %% Compare original waveform with resampled waveform
18  hold all;
19  figure(1);
20  subplot(3,1,1); %divide figure window into 3 axis
21
22  plot(y,'b') % plot the original waveform
23  title('Original waveform');
24  xlabel('time');
25  ylabel('amplitude');
26
27  subplot(3,1,2);
28  plot(y1, 'r'); %plot the resampled waveform;
29  title('Resampled waveform');
30  xlabel('time');
31  ylabel('amplitude');
32
33  subplot(3,1,3);
34  hold all;
35  plot(y,'b'); % plot the original waveform
36  plot(y1, 'r'); %plot the resampled waveform;
37  title('Waveforms compared');
38  xlabel('time');
39  ylabel('amplitude');
40
```

FIGURE 2 - PLOT AND COMPARE

The MATLAB *fdatool()* was then used to calculate the 16 sets of filter coefficients required for the filter bank. A band-pass filter response was chosen using an FIR filter type and a Kaiser window function. Table 1 shows the frequency specifications of each filter coefficient. A transition width of 50Hz was chosen for the purposes of this task. The upper and lower limits of the band-pass filter (F-Pass 1 and F-Pass-2) were found by dividing the bandwidth by two and adding or subtracting this value to the center frequency. The transition width was then added or subtracted to the upper or lower limit to find the stop frequencies.

TABLE 1 - FREQUENCY SPECIFICATIONS TASK 1

index	Center Freq	BW	F Stop 1	F pass 1	F pass 2	F Stop 2
coeff1	250	124	138	188	312	362
coeff2	375	126	262	312	438	488
coeff3	500	124	388	438	562	612
coeff4	625	126	512	562	688	738
coeff5	750	124	638	688	812	862
coeff6	937	250	762	812	1062	1112
coeff7	1187	250	1012	1062	1312	1362
coeff8	1437	250	1262	1312	1562	1612
coeff9	1687	250	1512	1562	1812	1862
coeff10	2000	376	1762	1812	2188	2238
coeff11	2375	374	2138	2188	2562	2612
coeff12	2812	500	2512	2562	3062	3112
coeff13	3312	500	3012	3062	3562	3612
coeff14	3875	626	3512	3562	4188	4238
coeff15	4563	750	4138	4188	4938	4988
coeff16	5375	874	4888	4938	5812	5862

Each coefficient was developed by the *fdatool()* and the data exported to the MATLAB workspace. These coefficients were then used in the *filter()* function to filter the resampled waveform *y1* effectively splitting the waveform into 16 separate waves.

```
41  %% Filter Outputs (Filtering through 16 channels)
42  %make a vector of zeros 16 rows deep and length(y1) long
43  y1 = y1'; %transverse the vector
44  out = zeros(16,length(y1));
45  out(1,:)=filter(coeff1,1,y1);
46  out(2,:)=filter(coeff2,1,y1);
47  out(3,:)=filter(coeff3,1,y1);
48  out(4,:)=filter(coeff4,1,y1);
49  out(5,:)=filter(coeff5,1,y1);
50  out(6,:)=filter(coeff6,1,y1);
51  out(7,:)=filter(coeff7,1,y1);
52  out(8,:)=filter(coeff8,1,y1);
53  out(9,:)=filter(coeff9,1,y1);
54  out(10,:)=filter(coeff10,1,y1);
55  out(11,:)=filter(coeff11,1,y1);
56  out(12,:)=filter(coeff12,1,y1);
57  out(13,:)=filter(coeff13,1,y1);
58  out(14,:)=filter(coeff14,1,y1);
59  out(15,:)=filter(coeff15,1,y1);
60  out(16,:)=filter(coeff16,1,y1);
61
```

FIGURE 3 - FILTER BANK SIMULATION

Each output of the filter was then passed through a LPF as shown below in Figure 4 in lines 69 to 71. The LPF was created using the *fir1()* function shown in lines 65. The *fdatool()* tool returned filters that were of order 1606 and the LPF required a cutoff frequency of 200Hz. The cutoff frequency is normalized by dividing by the Nyquist rate which is found using Equation 2 shown below.

$$Nyquisit\ rate = \frac{F_s}{2}$$

EQUATION 2 - NYQUISIT RATE

```
61      %% Signal rectification of each channel using a low pass filter
62
63      %using the fir1 function, creat low pass filter coefficients for each
64      %channel
65 -    [b,a] = fir1(1606,200/(0.5*FsNew));
66
67 -    out = abs(out); %get the absolute value of the filter outputs
68      %now pass each channel though the low pass filter
69 -    for i=1:numChannels
70 -        lowPass(i,:) = filter(b,a,out(i,:));
71 -    end
72
```

FIGURE 4 - SIGNAL RECTIFICATION USING LPF

Each output of the LPF was then resampled at a rate of 4 milliseconds by taking every 64th index as shown below in Figure 5 in lines 75 to 78. Remembering the sampling frequency was set to 16kHz, it follows that in order to gain the amplitudes at every 4ms the index spacing is found by using Equation 3 shown below. After every 64th index is selected, the data is sorted in descending order and saved in an array *X*. The sorting of the data is conducted to ensure only the 6 largest amplitudes of each channel is plotted in the electrodogram. Lastly, the data is normalized to the maximum value of the array (seen in line 85) to force the values to be between 0 and 1.

$$index = 0.004 \times 16000 = 64$$

EQUATION 3 - FINDING THE INDEX

```
72      %% sample at every 4msecs
73      %Get every 64th magnitude so that array is taken every 4msec
74 -    col = 1;
75 -    for i=1:64:length(lowPass)
76 -        sample64(:,col) = lowPass(:,i); %sample every 64th column ('i' steps by 64)
77 -        col = col + 1; %increment the column count
78 -    end
79
80      %sort sampled data into descending order to find the largest amplitudes
81      %X is the 16x122 matrix, Y is the index matrix showing how the index's were sorted
82 -    [X,Y] = sort(sample64,'descend');
83
84      %Normalize maximum amplitude to be a maximum of one
85 -    X = X./max(max(X));
86
```

FIGURE 5 - SAMPLE EVERY 4MS

The normalized data in variable *X* exists as a 122 x 16 matrix. By adding the matrix *X* with the index matrix *Y*, the data in each row of the matrix *X* is separated into is relative channel for plotting. This is achieved in line 91 shown in Figure 6 below. The first For Loop declared in line 99 is used to iterate across each row of the 16 channels. The second For Loop at line 100 is used to only plot the 6 largest magnitudes out of the 16 possible channels for that row number. The *time* variable is multiplied by 0.0004 at each loop iteration in order to plot the 6 largest magnitudes at 4ms intervals. It is in this way that the electrodogram is methodically plotted (see .

```
87      %% plot the data in the elctrodogram
88
89      %by adding the sorted data (X) with the index matrix (Y) the data is
90      %separated into 16 channels forming the electrodogram layout
91 -    data = X + Y;
92 -    figure(3);
93 -    title('Electrodogram');
94 -    xlabel('Time (sec)');   %label x axis
95 -    ylabel('Channel');    %label y axis
96
97 -    hold on
98
99 -    for i = 1:length(data)
100 -       for j = 1:6 %only want the 6 greatest magntidudes out of the 16 channels
101 -           time = i*0.0004;
102 -           plot(time,data(j,i),'bo')
103 -       end
104 -   end
105 -   hold off
106
```

FIGURE 6 - PLOTTING THE ELECTRODOGRAM

2.1.2 Results and Discussion

Figure 7 and Figure 8 show the original waveform and the resampled waveform. Figure 9 shows how the resampling has effected the waveform (shown in red) with respect to the original waveform (shown in blue). The resampling has decreased the amount of samples taken which has reduced the length of the waveform. Figure 9 shows how the resampling has changed the waveform (shown in red) when compared to the original waveform (shown in blue).

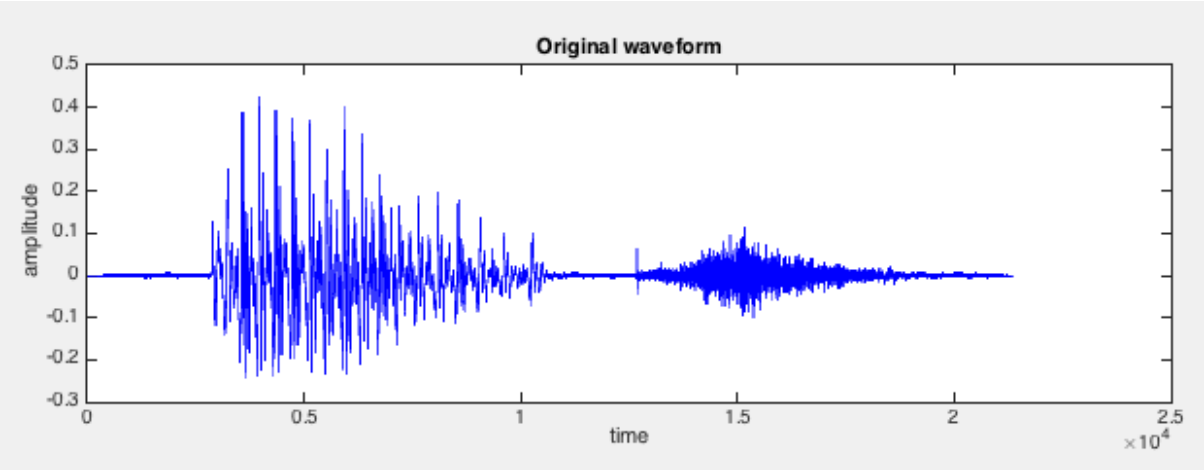

FIGURE 7 - ORIGINAL WAVEFORM

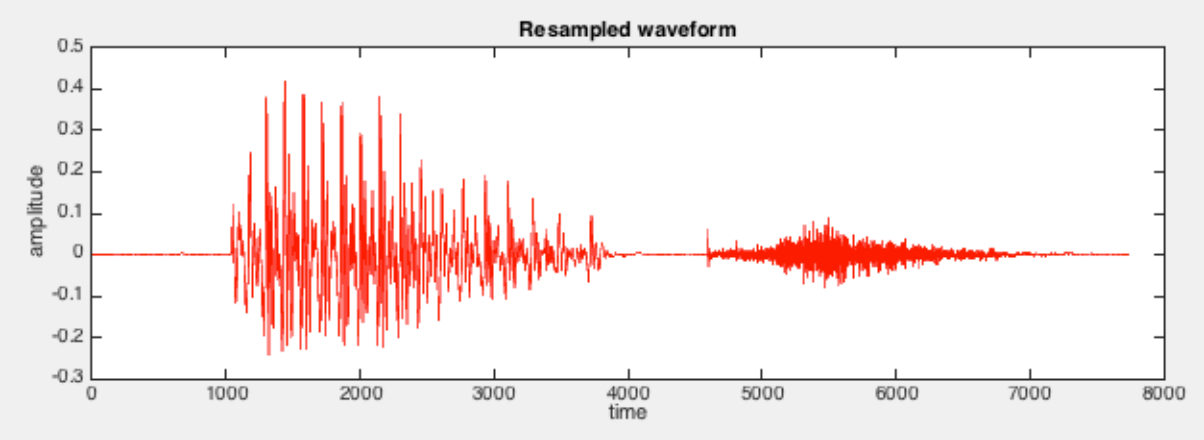

FIGURE 8 - RESAMPLED WAVEFORM

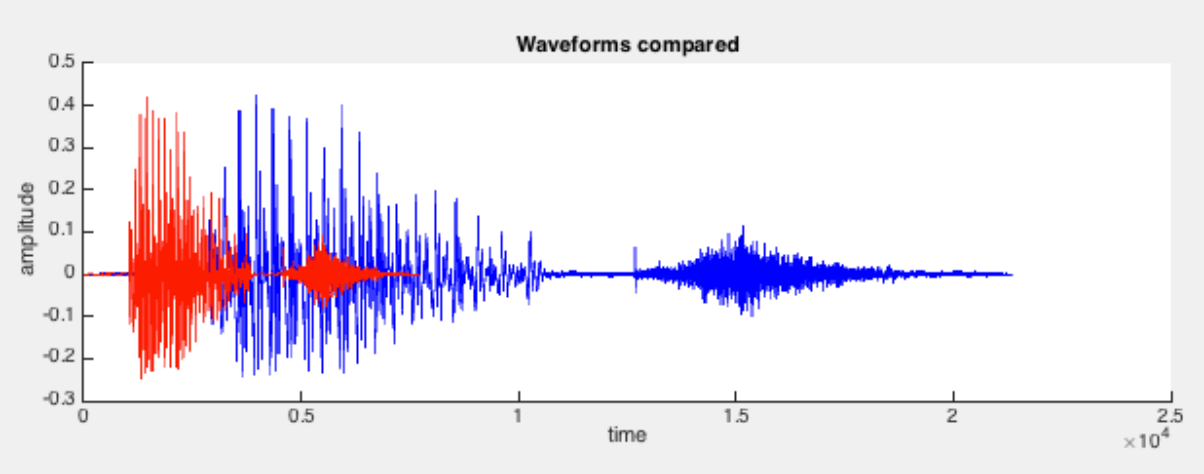

FIGURE 9 - WAVEFORMS COMPARED

The result of the method describe above in section 2.1.1 produced the electrodogram shown below in Figure 10. Initially, the lower and upper channels are activated the most often. At the 0.01ms time, the channel range between 8-12 is activated for a brief time before the channel range 0-4 dominates the activation alongside channels 10 and 12. In the time period of 0.04 to 0.05 the upper range of channels is most often activated.

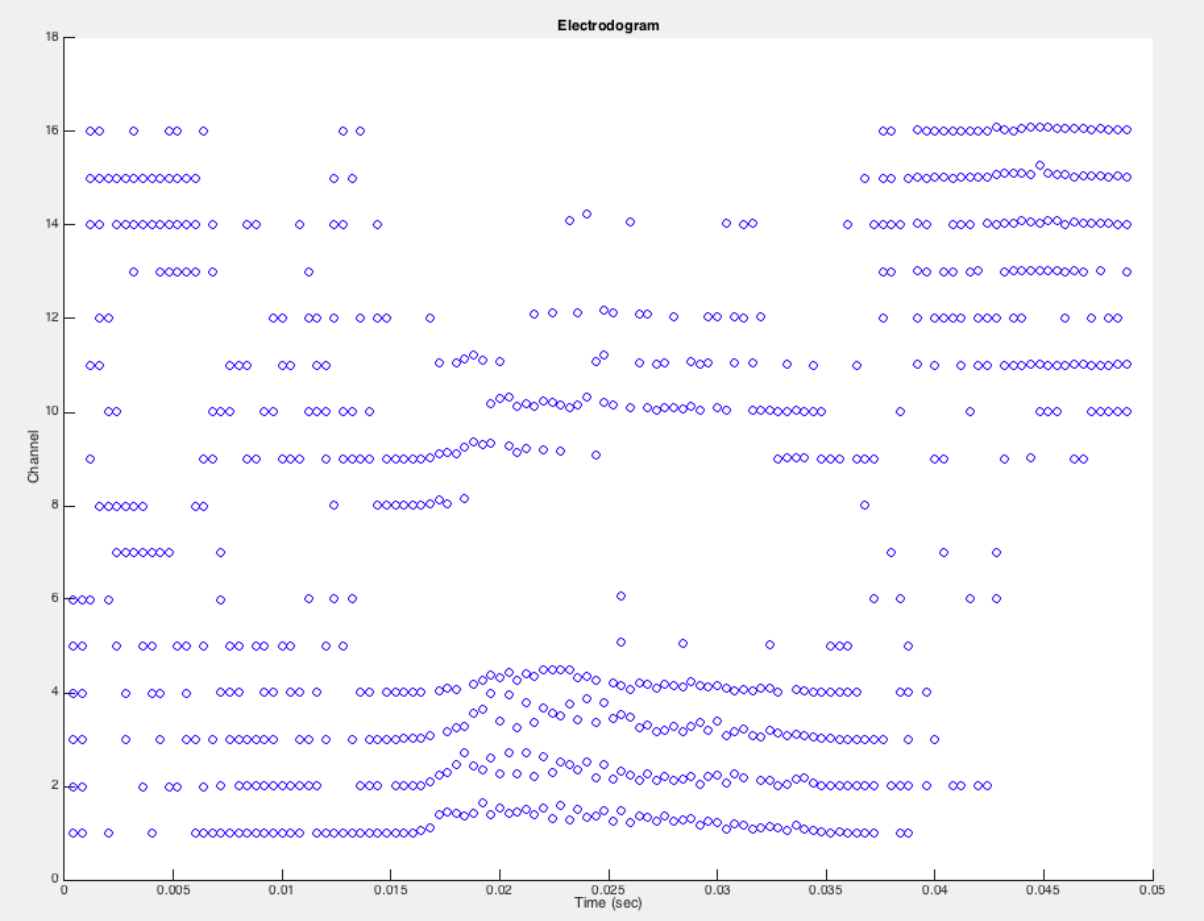

FIGURE 10 - ELECTRODOGRAM

2.2 Part B: Designing a hearing aid

2.2.1 Procedure

Step 1. The *hearingloss.mat* file is first loaded into the MATLAB workspace as per line 9 in Figure 11. Note the *X* variable contains the amplitudes of the signal for normal hearing level while the *Y* variable contains the amplitudes simulating hearing loss of the normal signal as shown below in Table 2.

```
7    %% initialization
8    %load the hearingloss.mat file into the workspace
9 -  load('hearingloss.mat')
10 - load('coeffTask2.mat')
11
12   % load the filter gain factors into an array 15 long
13 - gainsDB = [9; 14; 37; 53; 57; 41; 34; 38; 40; 43; 45; 46; 48; 49; 50; 51];
14
15   %% calc gains
16 - gains = zeros(1,16);
17 - for i=1:16
18 -     gains(i) = 10^(gainsDB(i)/20);
19 - end
20
```

FIGURE 11 - INITIALIZATION AND CALCULATION OF GAINS

TABLE 2 - SIGNALS

Signal	Vector Size
X – Normal Hearing Level	22529
Y – Simulating Hearing Loss	22591

Step 2. The coefficient matrix was developed using the MATLAB *fdatool()* in a similar manner as outline in Task 1 of this assignment. Table 3 shows the filter specifications used for each filter design in the *fdatool()*. A transition width of 50 Hz was selected and a bandwidth for each center frequency chosen considering the need to ensure pass-band overlap for each filter. The filter coefficient matrix was the exported and loaded into the MATLAB workspace as shown in line 10 in Figure 11.

TABLE 3 - FILTER SPECIFICATIONS FOR TASK 2

Filter	Center Freq	BW	F Stop 1	F pass 1	F pass 2	F Stop 2
1	250	200	100	150	350	400
2	500	200	350	400	600	650
3	750	200	600	650	850	900
4	1000	200	850	900	1100	1150
5	1250	200	1100	1150	1350	1400
6	1500	250	1325	1375	1625	1675
7	1750	250	1575	1625	1875	1925
8	2000	250	1825	1875	2125	2175
9	2250	250	2075	2125	2375	2425
10	2500	500	2200	2250	2750	2800
11	2750	500	2450	2500	3000	3050
12	3000	500	2700	2750	3250	3300
13	3250	500	2950	3000	3500	3550
14	3500	500	3200	3250	3750	3800
15	3750	250	3575	3625	3875	3925

Step 3. The decibel gains given in the assignment outline were loaded into an array. The decibel gains were defined using Equation 4 shown below.

$$G_{dB} = 20\log_{10}(G)$$

EQUATION 4 - DB GAINS

Therefore, using Equation 4 and solving for G, the equation is rearranged as shown below in Equation 5.

$$G = 10^{\frac{G_{dB}}{20}}$$

EQUATION 5 – GAINS

Equation 5 was used in conjunction with a For Loop in lines 17 to 19 to calculate the gains necessary to amplify the hearing loss signal.

Step 4. Signal X and Signal Y were then plotted and compared to gain an understanding of the differences in amplitude. Lines 22 to 43 plot these 3 graphs as shown in Figure 12.

```
21      %% Compare the two waveforms
22 -    hold all;
23 -    figure(1);
24 -    subplot(3,1,1); %divide figure window into 3 axis
25
26 -    plot(X,'b') % plot the original waveform
27 -    title('Original waveform');
28 -    xlabel('time');
29 -    ylabel('amplitude');
30
31 -    subplot(3,1,2);
32 -    plot(Y, 'r'); %plot the hearing loss waveform;
33 -    title('Hearing Loss waveform');
34 -    xlabel('time');
35 -    ylabel('amplitude');
36
37 -    subplot(3,1,3);
38 -    hold all;
39 -    plot(X,'b'); % plot the original waveform
40 -    plot(Y, 'r'); %plot the resampled waveform;
41 -    title('Waveforms compared');
42 -    xlabel('time');
43 -    ylabel('amplitude');
44
```

FIGURE 12 - PLOT SIGNAL X AND Y

Step 5. Using the filter coefficients and the gains already calculated, the hearing loss waveform was passed through a filter bank of 15 band-pass filters in order split the waveform into 15 different waves whilst consecutively applying the gains to each of the filters. This was achieved using lines 47 to 62 shown in Figure 13.

```
45      %% Filter Outputs (Filtering through 15 channels)
46 -    Y = Y'; %transverse the vector
47 -    filterOut = zeros(15,length(Y)); %make a vector of zeros 15 rows deep and length(Y) long
48 -    filterOut(1,:)=filter(coeff1,1,Y*gains(1));
49 -    filterOut(2,:)=filter(coeff2,1,Y*gains(2));
50 -    filterOut(3,:)=filter(coeff3,1,Y*gains(3));
51 -    filterOut(4,:)=filter(coeff4,1,Y*gains(4));
52 -    filterOut(5,:)=filter(coeff5,1,Y*gains(5));
53 -    filterOut(6,:)=filter(coeff6,1,Y*gains(6));
54 -    filterOut(7,:)=filter(coeff7,1,Y*gains(7));
55 -    filterOut(8,:)=filter(coeff8,1,Y*gains(8));
56 -    filterOut(9,:)=filter(coeff9,1,Y*gains(9));
57 -    filterOut(10,:)=filter(coeff10,1,Y*gains(10));
58 -    filterOut(11,:)=filter(coeff11,1,Y*gains(11));
59 -    filterOut(12,:)=filter(coeff12,1,Y*gains(12));
60 -    filterOut(13,:)=filter(coeff13,1,Y*gains(13));
61 -    filterOut(14,:)=filter(coeff14,1,Y*gains(14));
62 -    filterOut(15,:)=filter(coeff15,1,Y*gains(15));
63
```

FIGURE 13 - TASK TWO FILTER BANK

Step 6. The outputs of each of the filters was then recombined in order to gain a single waveform as shown in Figure 14 lines 65 to 68.

```
64      %% combine all the signals
65 -    signal = zeros(1,length(Y));
66 -    for i=1:14
67 -        signal = signal+filterOut(i,:);
68 -    end
69
70 -    signal = signal(500:end)
71 -    figure(2)
72 -    hold all
73 -    plot(signal,'r')
74 -    plot(X,'g')
75 -    sound(signal)
```

FIGURE 14 - COMBINING THE FILTER OUTPUTS

Step 7. Finally, the amplified signal was plotted in lines 70 and the sound tested for accuracy and validity. The first 500 bits of the signal were removed due to the phase shift that the filtering had induced. Removing these bits aligned the combined filter outputs with the original waveform.

2.2.2 Results and Discussion

Figure 15 and Figure 16 in section 2.2.2 show how signal Y's amplitude has been significantly reduced due to the simulation of the hearing loss. Figure 17 compares the signals and it can be seen that signal Y has been significantly reduced.

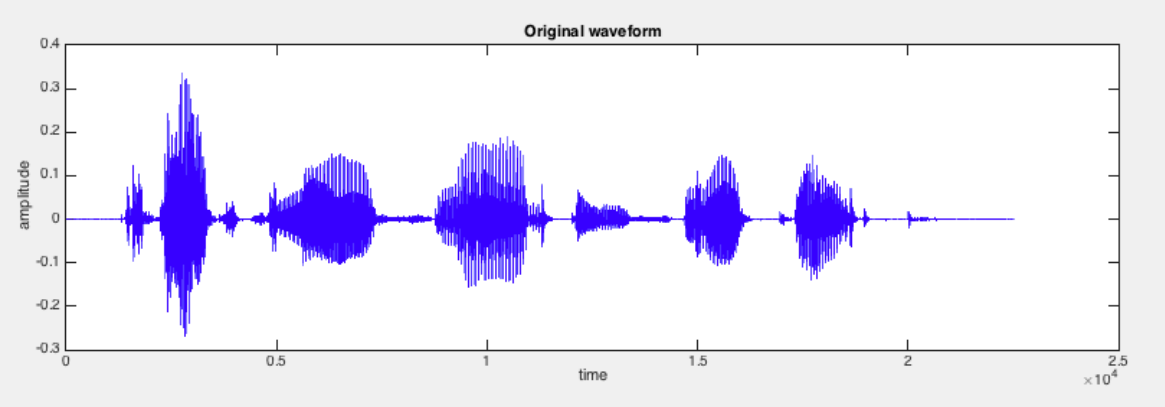

FIGURE 15 - SIGNAL X

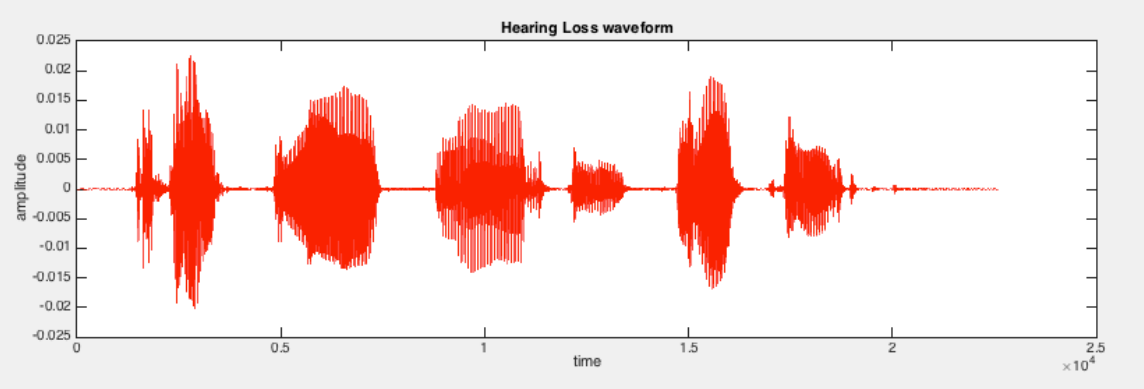

FIGURE 16 - SIGNAL Y

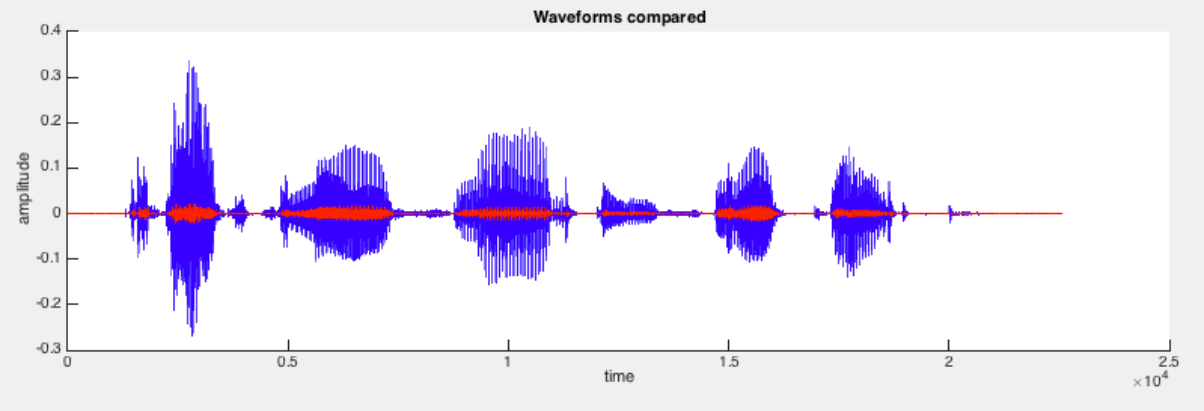

FIGURE 17 - SIGNAL X AND Y COMPARED

After the hearing loss signal was passed through the bank of 15 filters and the calculated gains applied, the signal was plotted and the result is shown below in Figure 18. Figure 19 shows that the simulated hearing aid (shown in red) has been able to very closely represent the normal hearing level waveform (shown in green) with only minor differences in amplitudes and phases observed.

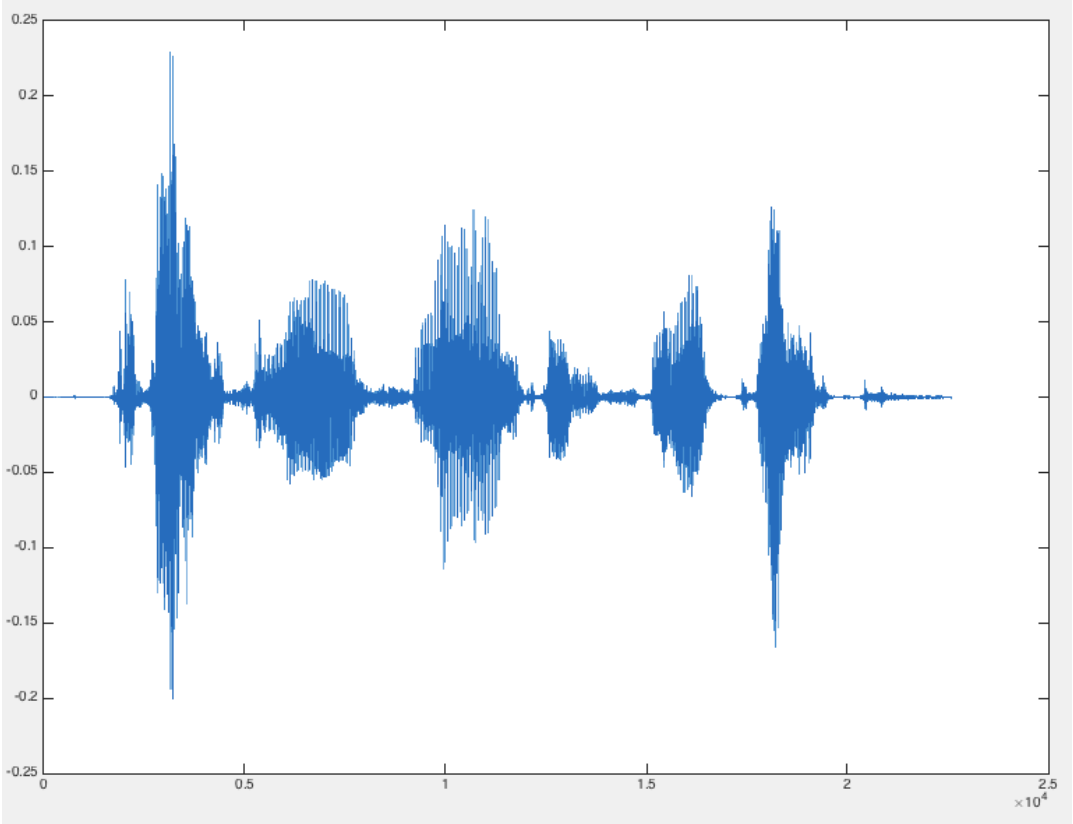

FIGURE 18 - AMPLIFIED SIGNAL

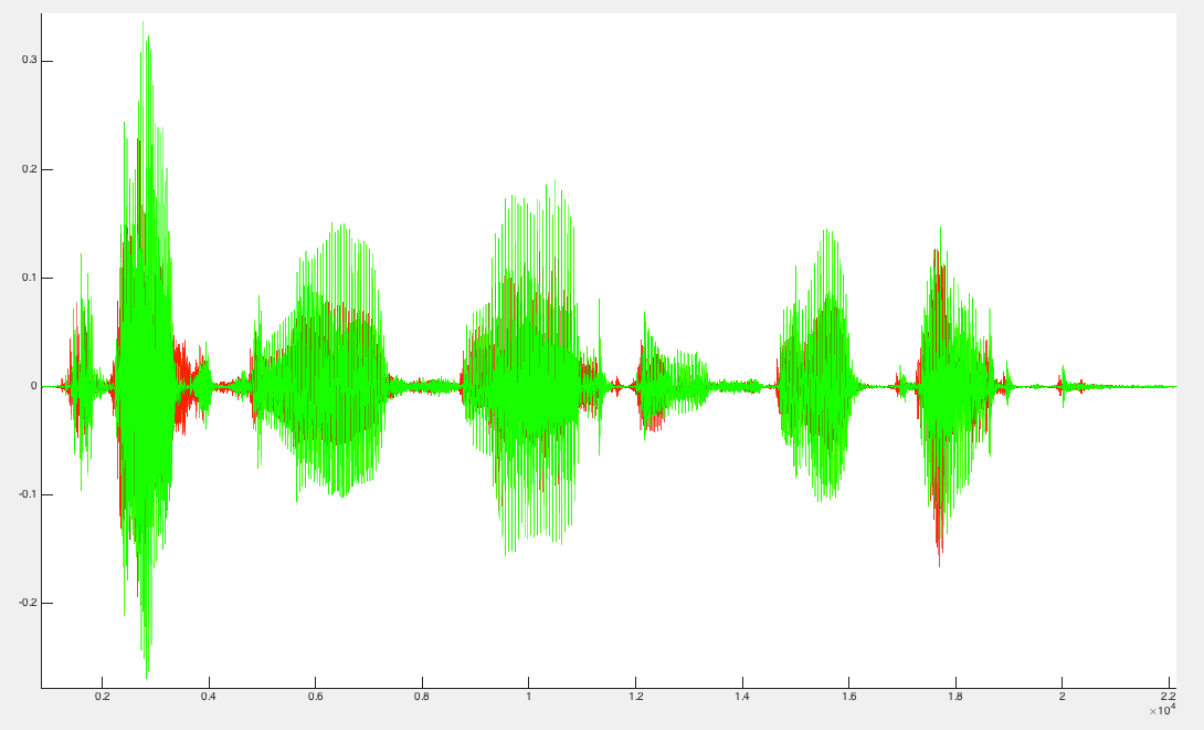

FIGURE 19 - AMPLIFIED SIGNAL COMPARED WITH NORMAL HEARING LEVEL

3 Conclusion

Task 1 of this assignment was to produce a simulated cochlear implant using the SMSP strategy. The algorithm was implemented using MATLAB and shown in Appendix A: MATLAB Code of Simulation of SMSP Process. The results of the algorithm proved effective and the electrodogram is shown in Figure 10. The original waveform was successfully split into 16 channels using band-pass filters whose coefficients were gained using the MATLAB *fdatool()*. The Chochlear implant was successfully simulated and met the needs of the requirements by selecting the 6 largest magnitudes as per the SMSP strategy. The aim of Task 2 was to develop a hearing aid and test its effectiveness by amplifying a signal simulating hearing loss. The simulated hearing aid included a bank of 15 band-pass filters and the gains for each filter applied before the outputs were recombined. The simulated hearing aid successfully met the aims of the task by amplifying the waveform ensuring the final product was legible and clear of noise.

4 References

[1] O. Kenny, “MATLAB Cochlear Implant Simulation Program .”

Appendix A: MATLAB Code of Simulation of SMSP Process

```
%EE4000 Assignment Task 1 - Dane Lennon

clc; %clear variables
close all; %close all figures

%% initialization
%load the audio file
%y = sampled data (a column vector of 21364 long is returned)
%Fs = sample rate (44100 is returned)
[y,    Fs]    =    audioread('/Users/danelennon/Documents/MATLAB/EE4000    -
DSP/mbet.wav');
FsNew = 16000;
y1 = resample(y,FsNew,Fs); %resample the data at 16000Hz
numChannels = 16; %set the number of channels
filterOrd = 1607; % set the filter order

%% Compare original waveform with resampled waveform
hold all;
figure(1);
subplot(3,1,1); %divide figure window into 3 axis

plot(y,'b') % plot the original waveform
title('Original waveform');
xlabel('time');
ylabel('amplitude');

subplot(3,1,2);
plot(y1, 'r'); %plot the resampled waveform;
title('Resampled waveform');
xlabel('time');
ylabel('amplitude');

subplot(3,1,3);
hold all;
plot(y,'b'); % plot the original waveform
plot(y1, 'r'); %plot the resampled waveform;
title('Waveforms compared');
xlabel('time');
ylabel('amplitude');

%% Filter Outputs (Filtering through 16 channels)
y1 = y1'; %transverse the vector
out  =  zeros(16,length(y1));  %make  a  vector  of  zeros  16  rows  deep  and
length(y1) long
out(1,:)=filter(coeff1,1,y1);
out(2,:)=filter(coeff2,1,y1);
out(3,:)=filter(coeff3,1,y1);
out(4,:)=filter(coeff4,1,y1);
out(5,:)=filter(coeff5,1,y1);
out(6,:)=filter(coeff6,1,y1);
out(7,:)=filter(coeff7,1,y1);
out(8,:)=filter(coeff8,1,y1);
out(9,:)=filter(coeff9,1,y1);
out(10,:)=filter(coeff10,1,y1);
out(11,:)=filter(coeff11,1,y1);
out(12,:)=filter(coeff12,1,y1);
```

```
out(13,:)=filter(coeff13,1,y1);
out(14,:)=filter(coeff14,1,y1);
out(15,:)=filter(coeff15,1,y1);
out(16,:)=filter(coeff16,1,y1);

%% Signal rectification of each channel using a low pass filter

%using the fir1 function, creat low pass filter coefficients for each
%channel
[b,a] = fir1(1606,200/(0.5*FsNew));

out = abs(out); %get the absolute value of the filter outputs
%now pass each channel though the low pass filter
for i=1:numChannels
    lowPass(i,:) = filter(b,a,out(i,:));
end

%% sample at every 4msecs
%Get every 64th magnitude so that array is taken every 4msec
col = 1;
for i=1:64:length(lowPass)
    sample64(:,col) = lowPass(:,i); %sample every 64th column ('i' steps
by 64)
    col = col + 1; %increment the column count
end

%sort sampled data into descending order to find the largest amplitudes
%X is the 16x122 matrix, Y is the index matrix showing how the index's were
sorted
[X,Y] = sort(sample64,'descend');

%Normalize maximum amplitude to be a maximum of one
X = X./max(max(X));

%% plot the data in the elctrodogram

%by adding the sorted data (X) with the index matrix (Y) the data is
%separated into 16 channels forming the electrodogram layout
data = X + Y;
figure(3);
title('Electrodogram');
xlabel('Time (sec)');   %label x axis
ylabel('Channel');    %label y axis

hold on

for i = 1:length(data)
    for j = 1:6 %only want the 6 greatest magntidudes out of the 16 channels
        time = i*0.0004;
        plot(time,data(j,i),'bo')
    end
end
hold off
```

Appendix B: MATLAB Code for the Simulation of a hearing aid

```
%EE4000 Assignment Task 2 - Dane Lennon

clc; %clear variables
%clear all; %clear memory
close all; %close all figures

%% initialization
%load the hearingloss.mat file into the workspace
load('hearingloss.mat')
load('coeffTask2.mat')

% load the filter gain factors into an array 15 long
gainsDB = [9; 14; 37; 53; 57; 41; 34; 38; 40; 43; 45; 46; 48; 49; 50; 51];

%% calc gains
gains = zeros(1,16);
for i=1:16
    gains(i) = 10^(gainsDB(i)/20);
end

%% Compare the two waveforms
hold all;
figure(1);
subplot(3,1,1); %divide figure window into 3 axis

plot(X,'b') % plot the original waveform
title('Original waveform');
xlabel('time');
ylabel('amplitude');

subplot(3,1,2);
plot(Y, 'r'); %plot the hearing loss waveform;
title('Hearing Loss waveform');
xlabel('time');
ylabel('amplitude');

subplot(3,1,3);
hold all;
plot(X,'b'); % plot the original waveform
plot(Y, 'r'); %plot the resampled waveform;
title('Waveforms compared');
xlabel('time');
ylabel('amplitude');

%% Filter Outputs (Filtering through 15 channels)
Y = Y'; %transverse the vector
filterOut = zeros(15,length(Y)); %make a vector of zeros 15 rows deep and
length(Y) long
filterOut(1,:)=filter(coeff1,1,Y*gains(1));
filterOut(2,:)=filter(coeff2,1,Y*gains(2));
filterOut(3,:)=filter(coeff3,1,Y*gains(3));
filterOut(4,:)=filter(coeff4,1,Y*gains(4));
filterOut(5,:)=filter(coeff5,1,Y*gains(5));
filterOut(6,:)=filter(coeff6,1,Y*gains(6));
filterOut(7,:)=filter(coeff7,1,Y*gains(7));
filterOut(8,:)=filter(coeff8,1,Y*gains(8));
```

```
filterOut(9,:)=filter(coeff9,1,Y*gains(9));
filterOut(10,:)=filter(coeff10,1,Y*gains(10));
filterOut(11,:)=filter(coeff11,1,Y*gains(11));
filterOut(12,:)=filter(coeff12,1,Y*gains(12));
filterOut(13,:)=filter(coeff13,1,Y*gains(13));
filterOut(14,:)=filter(coeff14,1,Y*gains(14));
filterOut(15,:)=filter(coeff15,1,Y*gains(15));

%% combine all the signals
signal = zeros(1,length(Y));
for i=1:14
    signal = signal+filterOut(i,:);
end

signal = signal(500:end)
figure(2)
hold all
plot(signal,'r')
plot(X,'g')
sound(signal)
```