Speech Recognition Project

ENAS 130

Gerardo Carranza

4/1/2014

The Speech Recognition Project describes the implementation of a system that can extract speech, identify certain characteristics, and match it to an existing speech segment from a database. The project covers the creation of the database, the speech sounds used for identification and analysis, how this speech was acquired, and what techniques were used to refine the system to match sounds more accurately.

Table of Contents

Introduction 2
- Statement of the Problem 2
- Organization 2

Speech Processing Steps 2
- Speech sounds used in program 2
- Data Acquisition 2
- Data segment selection 3
- Autocorrelation sequence as a speech feature vector 3
- Goodness Measure 4

Preliminary System 4
- Input Vocabulary 4
- Learning and test phoneme sets 4
- Feature Selection 5
- Database Structure 5

Automated System 5
- System description 5
- Input Speech Specification 6
- System Improvements 6
- User Interface 6

Results 7

Discussion 8
- Trained-speaker confusion matrix 8
- Untrained-speaker confusion matrix 8

Appendix - Program listings 9

1 Introduction

Statement of the Problem

The Speech Recognition Project seeks to implement a Matlab program that can extract a speech segment for analysis and recognize this speech segment from a test set of phonemes. A phoneme is a unique speech sound that can be described in terms of its formant frequencies and amplitudes. The significance of the project lies in that Matlab can use these numerical values to characterize speech.

Organization

The next section describes the speech sounds used in the program and answers the question why these speech sounds were considered for our database. It also covers how the speech sounds were recorded and distinguished from one another. Section 3 describes the preliminary system, which consisted of a learning set and a test set that examined the quality of the learning set. We also examine features that affect the accuracy of results, which enable us to create a functional database. Section 4 describes the automated system, which collects user input, compares it to the database, and chooses a best match. The system characterizes its own performance via a confusion matrix that shows when the spoken sound did not match the recognized sound. Section 4 also suggests possible improvements for the program. Section 5 presents the results of the automated system, one from a trained speaker, the second from an untrained one. The report ends with a discussion in Section 6. We examine errors in the program that occurred with both the trained and untrained speaker and how these errors can be minimized. If the results we find are similar, we know the program was user friendly for the untrained speaker.

2 Speech Processing Steps

Speech sounds used in program

The speech sounds my program attempts to recognize are phonemes. These sounds are appropriate because, as stated in Section 1, each phoneme has a unique formant frequency and amplitude. Their vowel sounds are more easily distinguishable than double consonants such as *ss* (as in hiss).

Data Acquisition

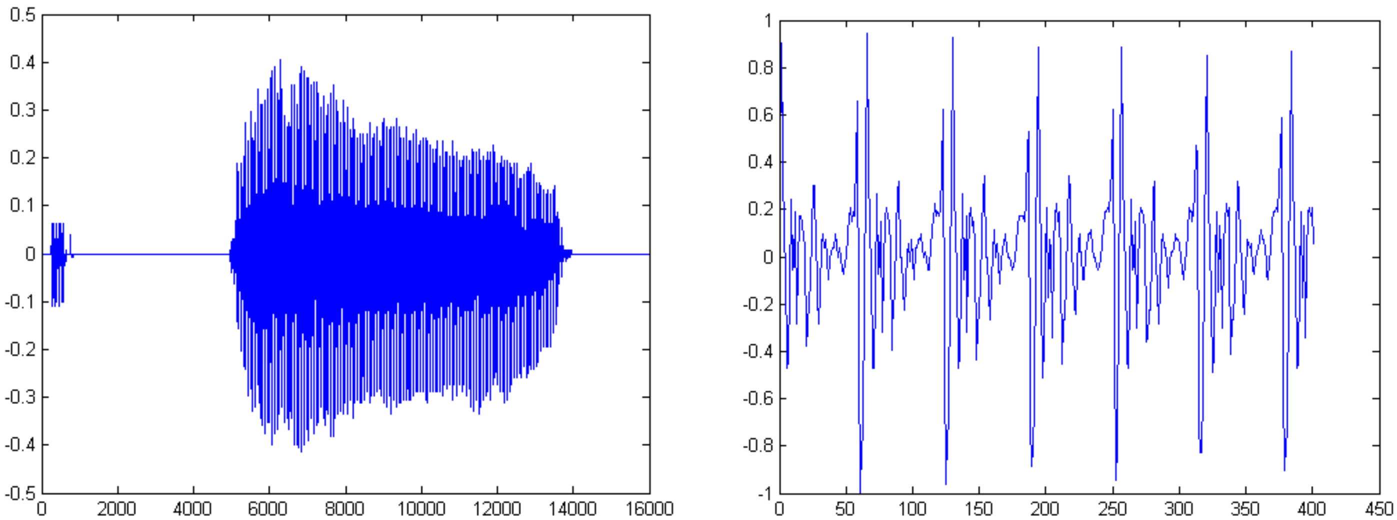
Sampling rate: 8000 Hz

Amplitude: 1(All samples are normalized to a max amplitude of 1)

Speech duration: 2 sec

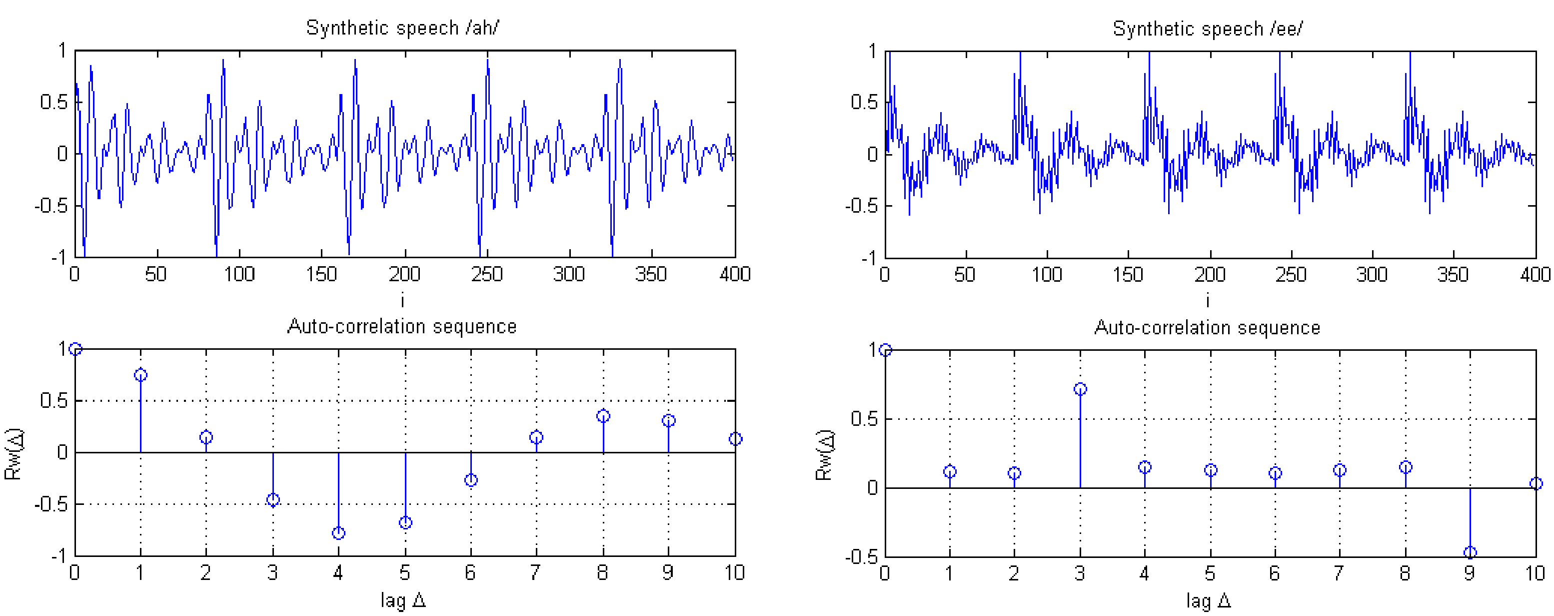
Conversion to floating-point variables for processing: ML_9_2.m recorded the speech segments for all the different phonemes and extracted these speech segments for analysis, using the getSeg function. The getSeg function further isolated the speech segment by extracting 401 elements of the speech.dat file about the midpoint.

Data segment selection



The segmentation process was meant to eliminate any silence from the beginning and ending of the original speech utterance (via a threshold), as well as the click transient. We then created a midpoint from these start and end values. From this we extracted +- 200 values from the midpoint, giving us 401 values. It was most convenient to acquire from the middle of the speech utterance so as not to have any variations from the expected phoneme.

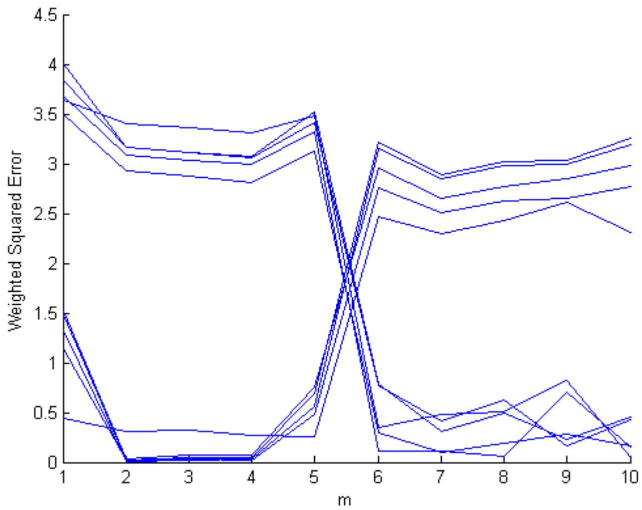
Autocorrelation sequence as a speech feature vector



The autocorrelation sequence is the cross-correlation of a signal with itself. The autocorrelation of time sequence used in the Speech Recognition Project describes the correlation between values of the speech segment array at different times. In other words, it indicates how values separated by an index value Δ, called the lag, are related.

Goodness Measure

The Goodness of fit measure is calculated using the weighted squared error between two phonemes. The two phonemes that exhibit the smallest weighted squared error are considered the most similar and therefore more likely to be the same phoneme. Mistakes can be due to a lack of clarity from the candidate phonemes used or the test phoneme spoken. As expected, the figure below shows that the first five candidates (*ah* phonemes) have a small weighted squared error with the first five test phonemes (also *ah* phonemes) but larger weighted squared error with the last five (*eh* phoneme).



3 Preliminary System

Input Vocabulary

The phonemes I used were the following: *ah, ee, er, oo, eh, ih.* I chose these because I believed them to be the most different from each other. Whereas *mm* and *nn* can be more easily mistaken for one another, the chosen set of phonemes are more clearly distinguishable.

Learning and test phoneme sets

The learning set was formed using the five odd numbered files that were recorded in ML_9_2. These were then combined to form a database (DB.dat file) in ML_9_3. The test set consisted of the five even numbered phoneme files from the recordings. We chose this set so as to vary the order.

Feature Selection

The number of R values used was 11 since the autocorrelation sequence has a total of 11 values (1x11 matrix). The feature set that produced the best performance was the one that used the highest R value, 11. This was the value that produced the least amount of errors and was optimal for feature selection.

numR	numErr
1	25
2	14
3	7
4	2
5	2
6	2
7	2
8	1
9	2
10	1
11	1

Database Structure

The final database was configured using DB.dat files. This configuration was the most accessible. There were difficulties in trying to load the 3D matrix (DB3D) onto the automated system; instead of loading a 3D matrix the program loaded a 5x66 matrix. The difficulties present in reconfiguring the 2D matrix into a 3D matrix once again proved to be too complicated of a task. This was solved combining the separate files into DB.dat files.

4 Automated System

System description

Operation Steps

1. The program starts by displaying a numbered list of the phonemes stated above.
2. The user types the number of the phoneme to be spoken.
3. The program asks the user to speak the chosen phoneme.
4. The program acquires the microphone speech, extracts a speech segment, analyzes the data, and compares it to the previously formed database.

5. The phoneme displays the spoken phoneme (according to the number typed) alongside the phoneme the system recognized.
6. The program updates a confusion table.
7. The program asks the user to type the number of the next phoneme to be spoken or 0 for program termination.
8. Before terminating, the program displays the confusion table.

From program start to termination (HOWTO)

1. Start the program by running the file MySpeechProgram.m
2. The Command window will show the various phonemes in this program's database.
3. Choose the number corresponding to the phoneme and press Enter.
4. Record your speech segment when told by the program. If uttered too softly, repeat recording.
5. Program will then show spoken and recognized phoneme.
6. Input another phoneme value or zero to terminate.

Input Speech Specification

The words are used were the following: saw, bee, her, boo, met, sit.

Phoneme	Word
ah	saw
ee	bee
er	her
oo	boo
eh	met
ih	sit

System Improvements

I changed the getSeg function and improved the automated system so that if the phoneme was uttered too softly it would give the user the opportunity to record the speech segment until the recording was loud enough. Additional insights that led to this version included the fact that using all ten phoneme utterances would lead to more accurate results and that it was convenient to show users the word beside the phoneme to reduce errors.

User Interface

The request for user input is displayed by showing the user the phonemes used and the words beside them that describe the sound of the phoneme. The recognition results are displayed by showing the spoken (typed) phoneme with the phoneme that the system recognized. The confusion table, CM, is displayed with the phoneme candidate names right above each column.

Editor - C:\Users\Gerardo\Documents\MATLAB\MySpeechProgram.m

MySpeechProgram.m

```matlab
%Gerardo Carranza
%4/1/14

errCnt = 0; % initialize error count
phonemes = ['ah';'ee'; 'er'; 'oo'; 'eh'; 'ih'];
numPhn = max(size(phonemes));
CM = zeros(numPhn,numPhn); % intialize confusion matrix
recObj = audiorecorder(8000, 8, 1); % initialize microphone


disp('The system is trained for the following sounds')
x=(1:6)';
s=num2str(x);
words= [' as in saw- ';' as in met- ';' as in her- ';' as in boo- ';' as in met- ';' as in sit- '];
```

Command Window

```
>> MySpeechProgram
The system is trained for the following sounds
ah as in saw- 1
ee as in met- 2
er as in her- 3
oo as in boo- 4
eh as in met- 5
ih as in sit- 6
Type the  number of the sound you will speak - or 0 to stop- and hit Enter
```

Waiting for input

Command Window

```
The system is trained for the following sounds
ah as in saw- 1
ee as in met- 2
er as in her- 3
oo as in boo- 4
eh as in met- 5
ih as in sit- 6
Type the  number of the sound you will speak - or 0 to stop- and hit Enter 1
Start speaking now
End of recording
spoken => ah recognized => ah
Type the  number of the sound you will speak - or 0 to stop- and hit Enter
```

Waiting for input

Command Window

```
Start speaking now
End of recording
spoken => ah recognized => ah
Type the  number of the sound you will speak - or 0 to stop- and hit Enter 0
     ah    ee    er    oo    eh    ih
     1     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
>>
```

5 Results

Confusion matrices:

Trained-Speaker						Untrained Speaker					
ah	ee	er	oo	eh	ih	ah	ee	er	oo	eh	ih
3	0	0	0	0	0	3	0	0	0	0	0
0	3	0	0	0	0	0	3	0	0	0	0
0	0	1	0	3	0	0	0	2	0	2	0
0	0	0	4	0	0	0	0	0	4	0	0
0	0	0	0	3	1	0	0	0	0	3	1
1	3	0	0	0	0	3	0	0	0	1	0

6 Discussion

Trained-speaker confusion matrix

The results showed a few errors when the trained speaker (me) ran the program. When doing the *er* sound, the program mistook it for the eh sound 3 times because I did not have my mouth close enough to the microphone. When saying *eh*, the program mistook it for ih. When saying *ih* the program mistook it for *ah* and *ee*, which demonstrated a flaw in the candidate *ih* arrays.

Untrained-speaker confusion matrix

The results showed similar results to that of the trained speaker. Once again, the *ih* test proved to be the most unsuccessful, with *ah* and *oo* being the most successful. *Eh* and *er* were mixed up occasionally as well.

The program still needs to be refined. The *ih* array should be recorded again (which I did) and we should also consider increasing number of phonemes that make up the database. The quality of my computer's microphone is probably not as good as a professional one, so that should also be considered. The quality of this program relies heavily on the quality of sounds in the database, as well as that from the user input.

7 Appendix - Program listings

MySpeechProgram.m

```
errCnt = 0; % initialize error count

phonemes = ['ah';'ee'; 'er'; 'oo'; 'eh'; 'ih'];

numPhn = max(size(phonemes));

CM = zeros(numPhn,numPhn); % intialize confusion matrix

recObj = audiorecorder(8000, 8, 1); % initialize microphone



disp('The system is trained for the following sounds') %Initial display

x=(1:6)';

s=num2str(x);

words= [' as in saw- ';' as in bee- ';' as in her- ';' as in boo- ';' as in met- ';' as in sit- ']; %Words

disp([phonemes, words, s])


n=input('Type the  number of the sound you will speak - or 0 to stop- and hit Enter '); %Input phoneme number


while(n~=0)

   error=1;

   while error == 1 %while sound is too soft continue recordings

      disp('Start speaking now') %prompt speaker

      recordblocking(recObj, 2); % record for 2 sec

      disp('End of recording'); % indicate end


      signal = getaudiodata(recObj)'; % write data in real-valued array


      [s err] = getSeg(signal); % form 401-sample segment
```

```
    error=err; %Once error equals zero, it will extit the while loop
    pause(2)
    Rtest = autocor(s,10);

end

Emin = 1000; %Emin set to large # to not affect later comparison
candMin = phonemes(2,:);

%Program compares Rtest to the database as in Assignment 12.
for iCand = 1:numPhn

    cand = phonemes(iCand,:);
    candDBfile = [ cand 'DB.dat']; %Extract DB file to be tested
    DBc = load(candDBfile); % Load DB file

    for iRow = 1:10
        candR = DBc(iRow, :); %Extracts candidate from DB file
        dif = Rtest - candR; %Compares candidate to test file
        E = dif*dif'; %compute weighted square error

        if E <= Emin
            %Chooses candidate from DB file most similar to test file
            Emin = E; %Sets error to minimum error, Emin
            candMin=cand; %Assigns corresponding candidate name
            candNum= iCand; %Assigns corresponding candidate phoneme #
        end
    end
```

```
    end

    testName=phonemes(n,:); %spoken phoneme name
    %Displays phoneme name(test file) along with the candidate
    %that was the best match
    disp([ 'spoken => ' num2str(testName) ' recognized => ' num2str(candMin)])

    CM(n,candNum) = CM(n,candNum) +1; %Confusion matrix update
    %Display for next input
    n=input('Type the  number of the sound you will speak - or 0 to stop- and hit Enter ');
end

%Final display
disp('     ah    ee    er    oo    eh    ih ')
disp(CM)
```

getSeg function

```
function [speechSeg, err] = getSeg(micSpeech)

tau=0.1; % Initialize threshold value
err=0; %Initialize error condition to zero
speechSeg = zeros(1,401);
%% Volume Verification
if max(abs(micSpeech))>.3



micSpeech=micSpeech/max(abs(micSpeech)); % Normalize micSpeech
```

```
%% First and last index value, istrt and iend
for i = 1:length(micSpeech)
    if micSpeech(i)>tau
        istrt = i;                      % remember first i value that exceeds tau
        break                           % exit loop
    end
end

for i = 1:length(micSpeech)
    if micSpeech(length(micSpeech)-i+1)>tau
        iend = length(micSpeech)-i+1; % remember last i value that exceeds tau
        break % exit loop
    end
end



%% Extraction of elements

imid= round((istrt+iend)/2); %Calculate midpoint value

speechSeg = micSpeech(imid-200:imid+200);% Extract 401 elements about midpoint

else
    err=1;
    disp('The microphone speech was too softly uttered, please try again.');
end
```

autocor function

```
function [ R ] = autocor( w, del_max )

%The autocorrelation sequence indicates how values separated

%by and index value delta,called the lag, are related


R = zeros(1,del_max+1); % Initialize array of zeroes


for i = 1:del_max+1

    R(i) = w(1:length(w)-i+1)*w(i:length(w))';  %Inner product

end


R=R/max(R); % Normalize array R to max value of 1


end
```

ML_12_2.m

```
phonemes = ['ah';'ee'; 'er'; 'oo'; 'eh'; 'ih'];

for numR=1:11


    errCnt=0; % Initializes error count


    numPhn=max(size(phonemes));  % number of phonemes


for iPhn=1:numPhn


    phnName = phonemes(iPhn,:); %phoneme name
```

```
for iTst=2:2:10

    testNameNum = iTst; % test name number

    % testFile to be compared to each of the candidates, e.g. ah2.dat
    testFile = [ phnName num2str(testNameNum) '.dat'];

    test = testFile(1:2); %Phoneme name of test file
    speech = load(testFile);

    % auto-correlation sequence on even numbered test file
    testR = autocor(speech, 10);


    Emin = 1000; %Emin set to large # to not affect later comparison
    candMin = phonemes(2,:);

    for iCand = 1:numPhn

        cand = phonemes(iCand,:);
        candDBfile = [ cand 'DB.dat']; %Extract DB file to be tested
        DBc = load(candDBfile); % Load DB file

        for iRow = 1:5
            candR = DBc(iRow, :); %Extracts candidate from DB file
            dif = testR - candR; %Compares candidate to test file
            E = dif(1:numR)*dif(1:numR)'; %compute weighted square error
```

```
        if E <= Emin
            %Chooses candidate from DB file most similar to test file
            Emin = E; %Sets error to minimum error, Emin
            candMin=cand; %Assigns corresponding candidate name
        end

    end

  end
```

ML_12_1.m

```
phonemes = ['ah';'ee'; 'er'; 'oo'; 'eh'; 'ih'];
errCnt=0; % Initializes error count
numPhn=max(size(phonemes));  % number of phonemes

for iPhn=1:numPhn

    phnName = phonemes(iPhn,:) %phoneme name
    for iTst=2:2:10

        testNameNum = iTst; % test name number

        % testFile to be compared to each of the candidates, e.g. ah2.dat
        testFile = [ phnName num2str(testNameNum) '.dat'];

        test = testFile(1:2); %Phoneme name of test file
        speech = load(testFile);
```

```
% auto-correlation sequence on even numbered test file
testR = autocor(speech, 10);



Emin = 1000; %Emin set to large # to not affect later comparison
candMin = phonemes(2,:);

for iCand = 1:numPhn

   cand = phonemes(iCand,:);
   candDBfile = [ cand 'DB.dat']; %Extract DB file to be tested
   DBc = load(candDBfile); % Load DB file

   for iRow = 1:5
      candR = DBc(iRow, :); %Extracts candidate from DB file
      dif = testR - candR; %Compares candidate to test file
      E = dif*dif'; %compute weighted square error

      if E <= Emin
         %Chooses candidate from DB file most similar to test file
         Emin = E; %Sets error to minimum error, Emin
         candMin=cand; %Assigns corresponding candidate name
      end

   end

end
```

```
        %Displays phoneme name(test file) along with the candidate
        %(from all 5 rows of the 6 DB files)that was the best match
        disp([ test ' ' candMin])

        if strcmp(test, candMin) == 0
            %Increments error if phoneme names don't match
            errCnt = errCnt + 1 ;
            disp (' ') %Display space, easier to see where error occured
        end


    end
end
errCnt %Final error count
```

ML_11_2.m

```
DB3D= load('DB3D');
%reshape(DB3D, [ 5 11 6]);

phonemes = ['ah';'ee'; 'er'; 'oo'; 'eh'; 'ih'];

for i=1:max(size(phonemes))
    testPhonName= phonemes(i,:)
    for j=2:2:10
        test= load([testPhonName int2str(j) '.dat']);
        testR= autocor(test,10);

        DB3size= size(DB3D)
```

```
        Emin=1000;

        for k=1:DB3size(3)
            for m=1:DB3size(1)
                candD= autocor(DB3D(m,:,k))
                dif= testR -candD;

                E = dif(1:numR)*dif(1:numR)'; %compute weighted square error

                if E <= Emin
                    %Chooses candidate from DB file most similar to test file
                    Emin = E; %Sets error to minimum error, Emin
                    DB3num=k;          %Assigns corresponding candidate name
                end
            end
        end

        DB3name=phonemes(k,:)
        disp(['test: ' testPhonName 'matched to candidate: ' DB3name])
        if strcmp(testPhonName, DB3name) == 0
            %Increments error if phoneme names don't match
            errCnt = errCnt + 1;
            %disp (' ') %Display space, easier to see where error occured
        end

    end
end
```

errCnt

ML_11_1.m

```
phonemes = ['ah';'ee'; 'er'; 'oo'; 'eh'; 'ih'];


DB3D = zeros(5,11,max(size(phonemes)));
filename='';
for i=1:max(size(phonemes))


    filename = [phonemes(i,:) 'DB.dat'];
    pDB = load(filename);
    DB3D(:,:,i)= pDB;


end
save('DB3D.dat','DB3D','-ascii')
```

ML_10_2.m

```
EmC=load('EmC.dat'); %loads EmC.dat file
%My one test phoneme file, used to compare to all the other candidates, was
%ah2.dat
for i=1:8
   hold on % allows to superimpose multiple lines in a current plot
   plot(1:10,EmC(i,:)) %%plots Em2 values for each trial(i # of ones in c)
   xlabel('m'); %Represents the mth candidate in the database
   ylabel('Weighted Squared Error'); % Represents goodness of fit measure
   pause(1)
```

```
end
 hold off
```

ML_10_1.m

```
Em2=load('Em2.dat'); %loads Em2.dat file

for i=1:10
    hold on % allows to superimpose multiple lines in a current plot
    plot(1:10,Em2(i,:)); %plots Em2 values for each row/trial
    xlabel('m'); %Represents the mth candidate in the database
    ylabel('Weighted Squared Error'); % Represents goodness of fit measure
    pause(1)
end

hold off
```

ML_9_3v.m

```
    t=0:length(phoneme)-1;

    plot(t,phoneme,'b'); %Sets color of array to blue, plots array
    axis([0 400 -1 1]); % Sets axes so all graphs are consistent
    grid on;
    hold on % holds current plot so autocorrelation plot
            % can be superimposed if necessary
```

```
    if mod(i,2)== 1
        Rw = autocor(phoneme,10); % Autocorrelation sequence

        tr= (0:10)*length(phoneme)/10;

        stem(tr,Rw,'r'); %Graphs superimposed plot in red, easier to see
        grid on;


    end

    hold off % return to default mode where previous plots are erased

    pause(1)

end
```

ML_9_3.m

```
phonemes = ['ah';'ee'; 'er'; 'oo'; 'eh'; 'ih'];
[numR numC] = size(phonemes);
for nphone = 1:numR
    phoneme = phonemes(nphone,:)
    DB = zeros(5,11);
    DBrow = 1;
    for i=1:10
        filename = [phoneme num2str(i) '.dat'];
```

```
        humphone = load(filename);
        Rw = autocor(humphone,10);
        DB(DBrow,:) = Rw;
        DBrow=DBrow+1;
    end
    ...
        filenameDB = [phoneme 'DB.dat'];
    save(filenameDB, 'DB', '-ascii') % save data file on disk
end
```

ML_9_2V.m

```
phonemes = ['ah']; % Initialize phonemes array (size of 1 for this assignment)


i=1;
phoneStr = phonemes(i,:) %Initialize phoneStr to name of phoneme



for i=1:10
    filename = [phoneStr int2str(i) '.dat'] %ah#.dat file name
    phoneme = load(filename); %load ah#.dat file

    t=0:length(phoneme)-1; %Set interval from 0 to 400

    subplot(5,2,i),plot(t,phoneme); %Plot 401 point array
    title(filename); %Title each ah#.dat file

end
```

ML_9_2.m

```
% define ADC specs: Fs=8000Hz, 8bits , one channel
recObj = audiorecorder(8000, 8, 1);


phonemes = ['ah';'ee'; 'er'; 'oo'; 'eh'; 'ih'];
%
i=3;
phoneStr = phonemes(i,:)
answer = 1;
%
for f_num =1:10
   % get speech from microphone = micSpeech
   disp('Start speaking now')        % prompt speaker
   recordblocking(recObj,2);         % record for 2 sec
   disp('End of recording');         % indicate end
   micSpeech = getaudiodata(recObj)';


   % Create data files from microphone speech
   [speechSeg, err] = getSeg(micSpeech)


   if err == 0
      filename = [phoneStr int2str(f_num) '.dat']
      save(filename, 'speechSeg', '-ascii')
      answer = input('Another file (1/0) ')
   end
```

ML_9_1.m

```
mySpeech= load('my_Speech.dat'); %microphone data file


[ speechSeg, err ] = getSeg(mySpeech) %Call getSeg function

plot(mySpeech)

%This displays Rw and err
```

ML_8_2.m

```
%% W Array

Fs=8000; Ts=1000/Fs; r=.975;%Set sampling freq, sample period (ms), decay rate


F1=270; F2=2290; F3=3010; % Set frequencies F1,F2,F3


mpf = 100; % male pitch frequency in Hz

mpitchPeriod = round(Fs/mpf); % male pitch period in terms of sample values

% round to form integer


d=[1 zeros(1,mpitchPeriod-1)]; % array d of one followed by zeroes

d = [d d d d d]; %Concatenate d several times to make 400 point array


u=AR(d,F1,r); v=AR(u,F2,r);w=AR(v,F3,r); %Apply d to AR functions


w=w/max(abs(w)); % Normalize w to max absolute value of 1


t=(0:length(d)-1); %Define 400 point array


subplot(2,1,1),plot(t,w); %Plot w as a function of t

xlabel('i');

title('Synthetic speech /ee/'); %Set title for plot
```

```
%% Autocorrelation Sequence
del_max=10;

Rw = autocor(w,del_max) %Call and display autocorrelation sequence

subplot(2,1,2),stem(0:del_max,Rw); % Stem plot of Rw
title('Auto-correlation sequence');
xlabel('lag \Delta'); ylabel('Rw(\Delta)'); grid on;
```

ML_8_1.m

```
w = ones(1:10); % Initialize array of ones
del_max = 9; %Initialize del_max to 9

R = autocor(w, del_max) %Call autocor function, display array R
stem(0:del_max,R); %Plot stem plot
```