In the name of God

Voice Recognition Systems

Mansour Saffar Mehrjardi   Ahmad Tavakoli

Voice Recognition systems (VR systems) are widely used in many new technologies. Robust speech recognition systems are used in military applications. For instance in F11 jets these systems recognize pilot's voice for authentication of their instruction. Car production companies are using this technology in their cars so the driver can steer the car only with speaking to the car! Hollywood has also shown the future of speech recognition systems in its movies. We see in the Oscar winner movie "Her" that these systems are used in an interactive super intelligent operating system in order to speak to the computer.

In this project we tried to develop an isolated speaker-dependent voice recognition system using MATLAB software. Isolated voice recognition systems should acquire their data with pause which means every time you want to record voice there must be a pause between every word pronounced. Speaker-dependent voice recognition systems only can recognize one particular's voice whereas voice-independent voice recognition systems can recognize everyone's voice independent of the speaker.

In this project we implemented an isolated speaker-dependent voice recognition system using MATLAB. In Speech signal processing algorithms first we extract information from the signal using several methods and then give these data to a data clustering system which recognizes specific patterns within these data and can categorize these data for further recognition. Among these recognition and data clustering algorithms are neural networks which we used in this project. Artificial Neural networks are inspired by the way biological nervous systems work for processing information. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

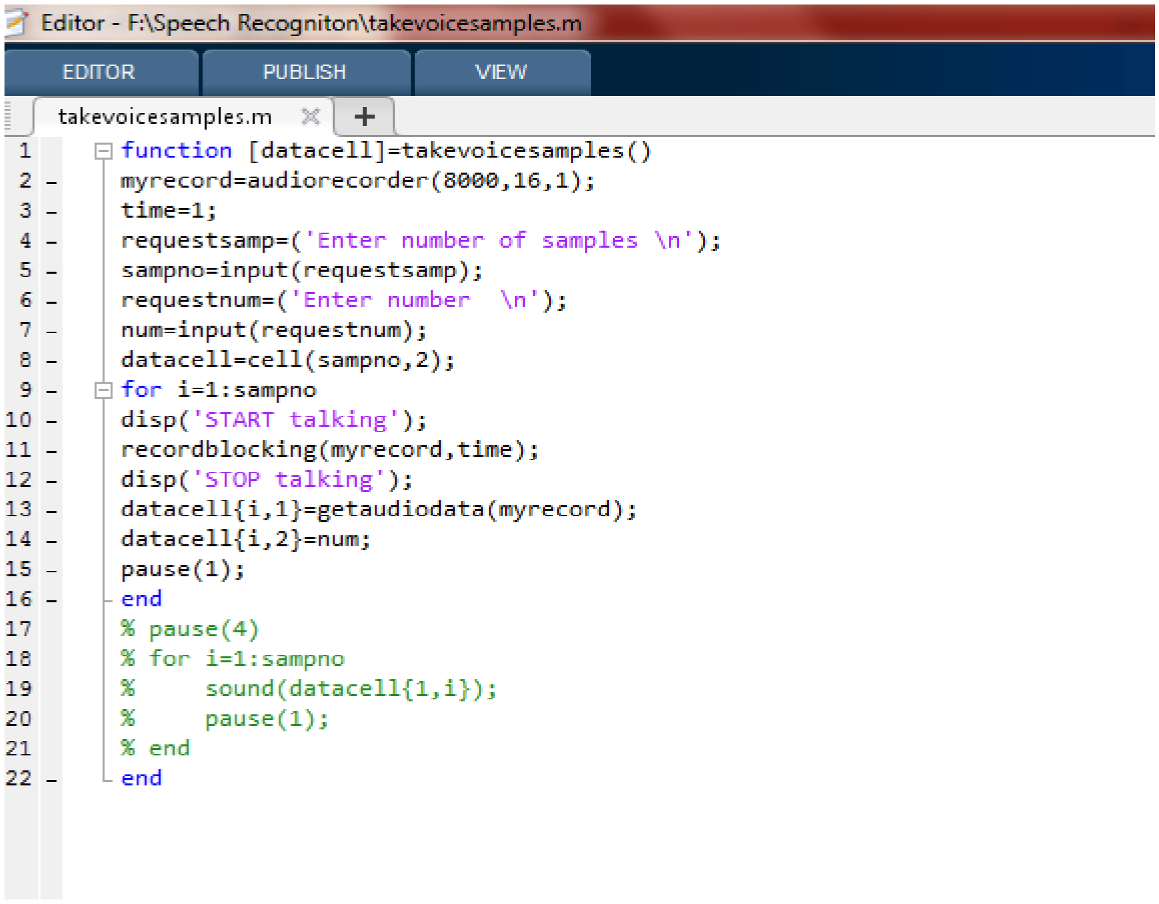
The process of this project is as below:

Data acquisition:

I recorded my voice 500 times saying numbers one to ten (each number 50 times). Each recording lasts 1 second and the sampling frequency is 8000Hz. Each recording is the filtered by a high pass IIR filter which is given by the function $H(z) = 1 - 0.98z$. This filter reduces low frequency noises in speech and also amplifies frequencies that is used in.

The magnitude response of this filter is shown in figure 1.1.

The MATLAB function which I used in this project is shown in figure 1.2.



```
function [datacell]=takevoicesamples()
myrecord=audiorecorder(8000,16,1);
time=1;
requestsamp=('Enter number of samples \n');
sampno=input(requestsamp);
requestnum=('Enter number  \n');
num=input(requestnum);
datacell=cell(sampno,2);
for i=1:sampno
disp('START talking');
recordblocking(myrecord,time);
disp('STOP talking');
datacell{i,1}=getaudiodata(myrecord);
datacell{i,2}=num;
pause(1);
end
% pause(4)
% for i=1:sampno
%     sound(datacell{1,i});
%     pause(1);
% end
end
```

Figure 1.2

The function takevoicesamples() stores 1second voice samples in a cell. Samples acquired by this function is the applied to 3 different functions which extract specific information from samples.

Data extraction:

MFCC (Mel frequency cepstrum coefficient) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

MFCCs are commonly derived using these steps:

I. Pre-emphasize the signal using the filter such as figure 1.1
II. Take short-time Fourier transform of the windowed signal. The window is normally hamming window.
III. Compute magnitude cepstrum and then apply the signal to a filter bank. This filter bank is triangular and is equally spaced in mel frequency domain.
IV. Energy of each filter bank part is calculated.
V. These energies are applied to a discrete cosine transform and the MFCCs are the amplitude of the resulting spectrum.

STFT (short-time Fourier transform)

Short time Fourier transform gives us signal's information both in time domain and also the frequency domain. Speech signal are non-stationary signals which means the amplitude of a certain frequency can vary with time, so we need to know which time does a frequency appears and when it vanishes. In STFT first we window the original signal, using hamming window which smooths signal fluctuations. Then we take FFT of the windowed signal. Then we go further in signal and according to hop size of SFTF which is actually the value of pace we take when going further in the signal and then windowing that part of signal.
If we choose the length of the window to be large then we can discern adjacent frequencies in the signal but processing of signal will take much more time. In contrast to what said before, if we choose the length of window to be short it reduces processing time but also reduces resolution of resulting STFT. Note that

hop size or simply the pace we choose, can also affect processing time and resolution of resulting STFT. To demonstrate what I said, figure 1.3 shows STFT of 5 seconds excerpt of a music sampled with frequency of 8KHz. Length of original signal is 40000. In figure 1.3 window length is 50 and hop size is also 50, so we get a high resolution STFT.

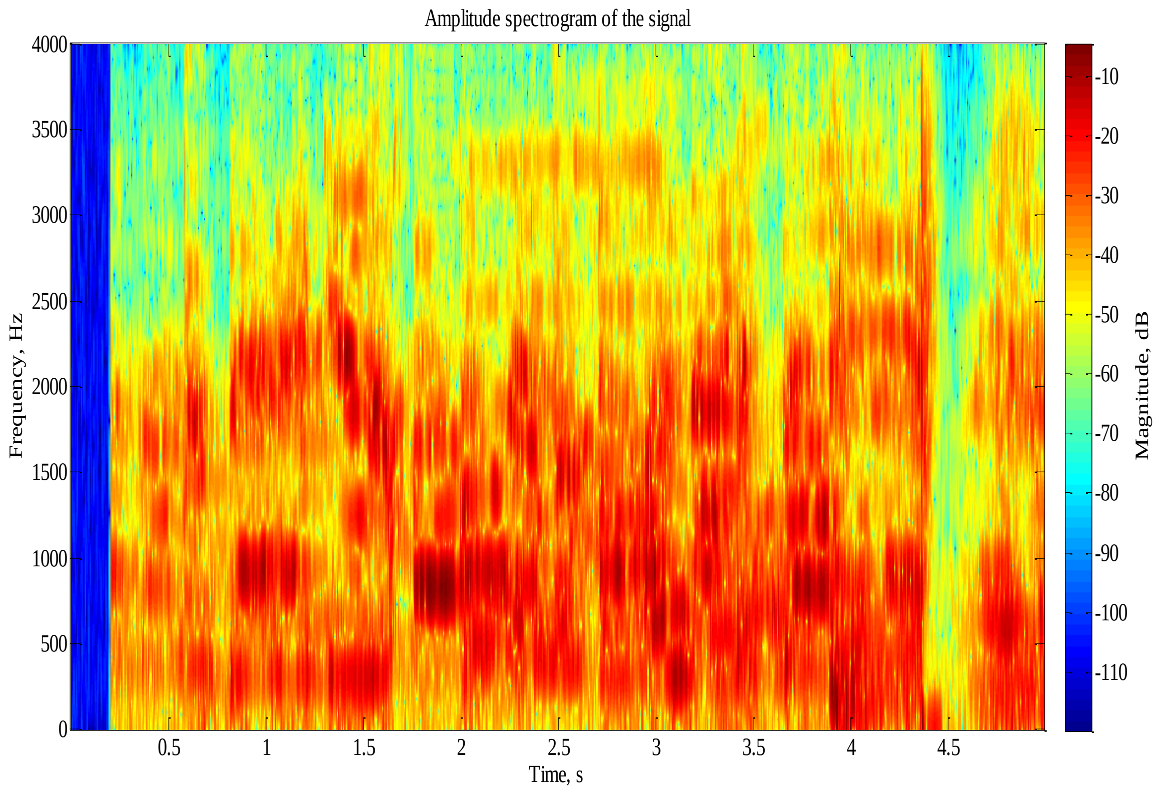


Figure 1.3

Note that horizontal axis is time (in seconds) and vertical axis is frequency (in Hertz). The color bar right to the signal shows the magnitude of signal in dB.

Figure 1.4 shows STFT of the previous signal in which window length is 200 and hop size is also 200. As you can see it has considerably lower resolution than Figure 1.3 but it takes less time to achieve this plot.

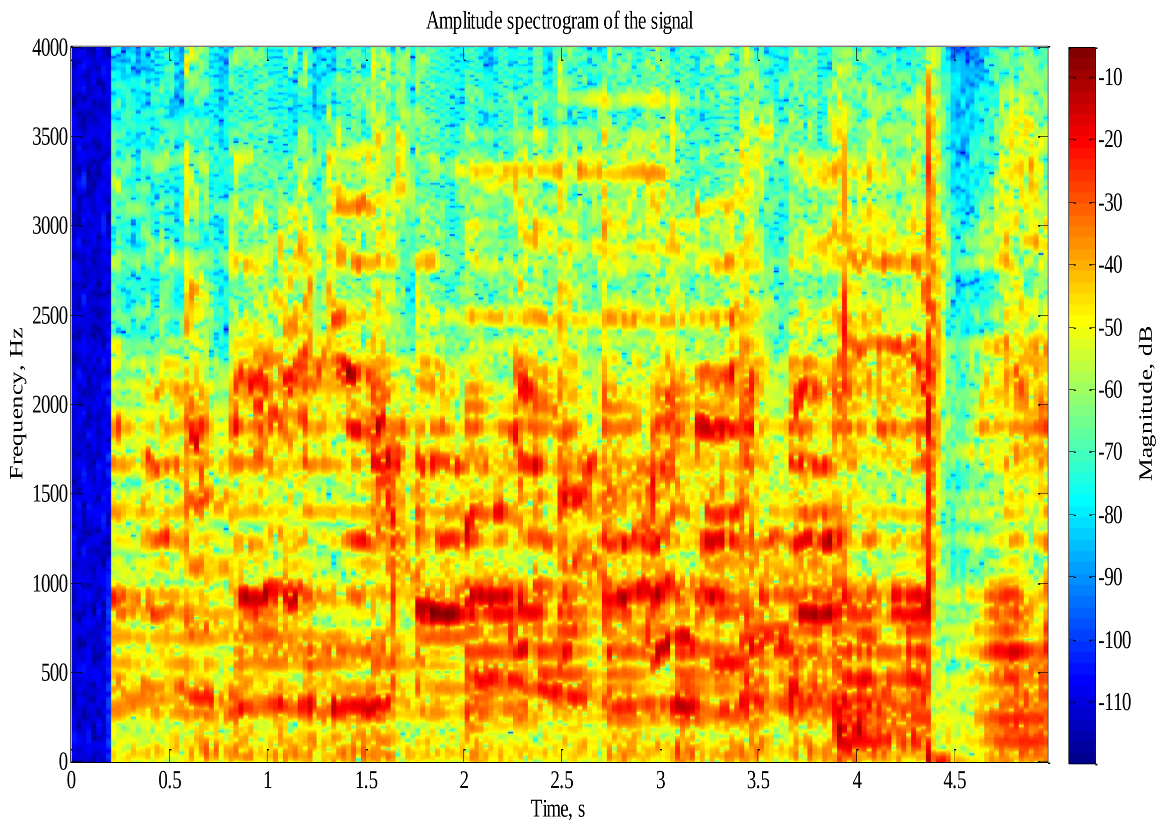


Figure 1.4

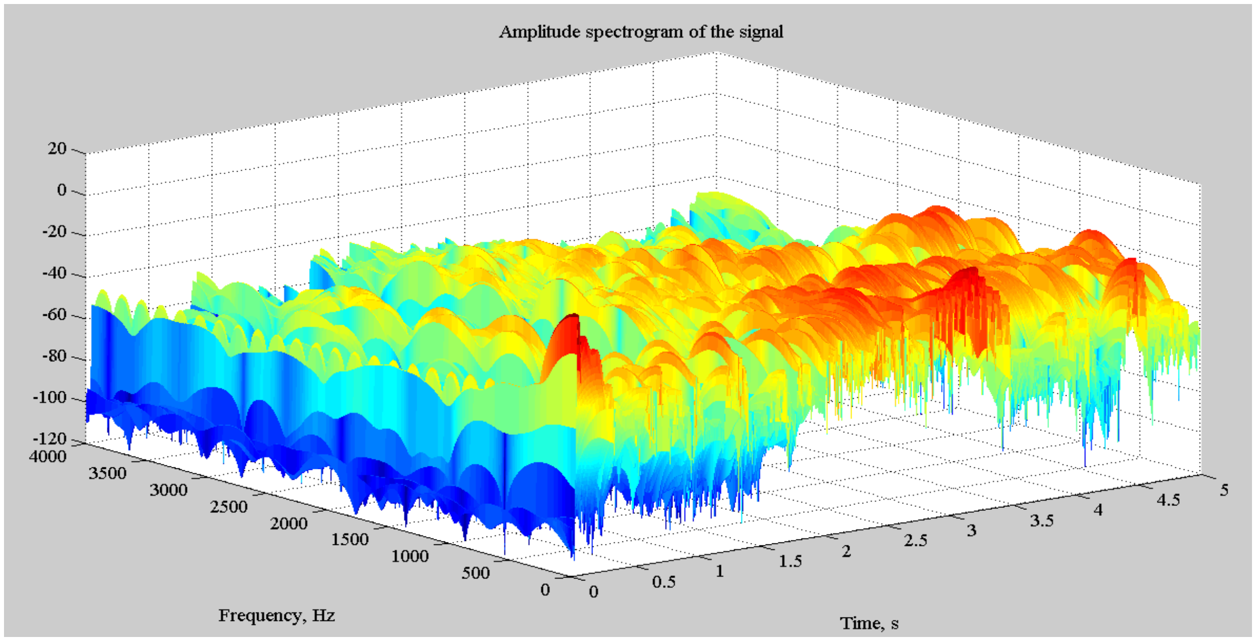
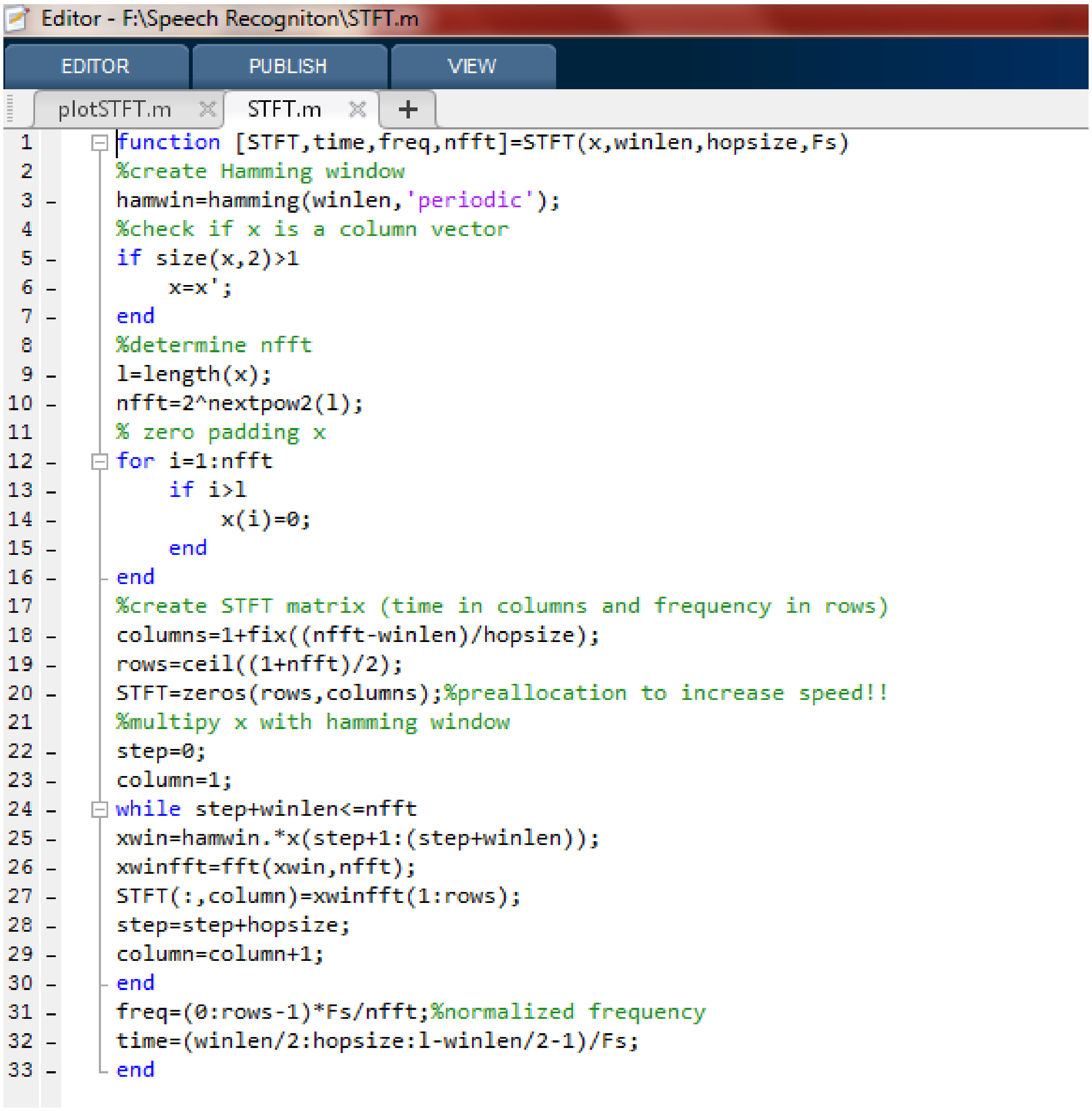


Figure 1.5

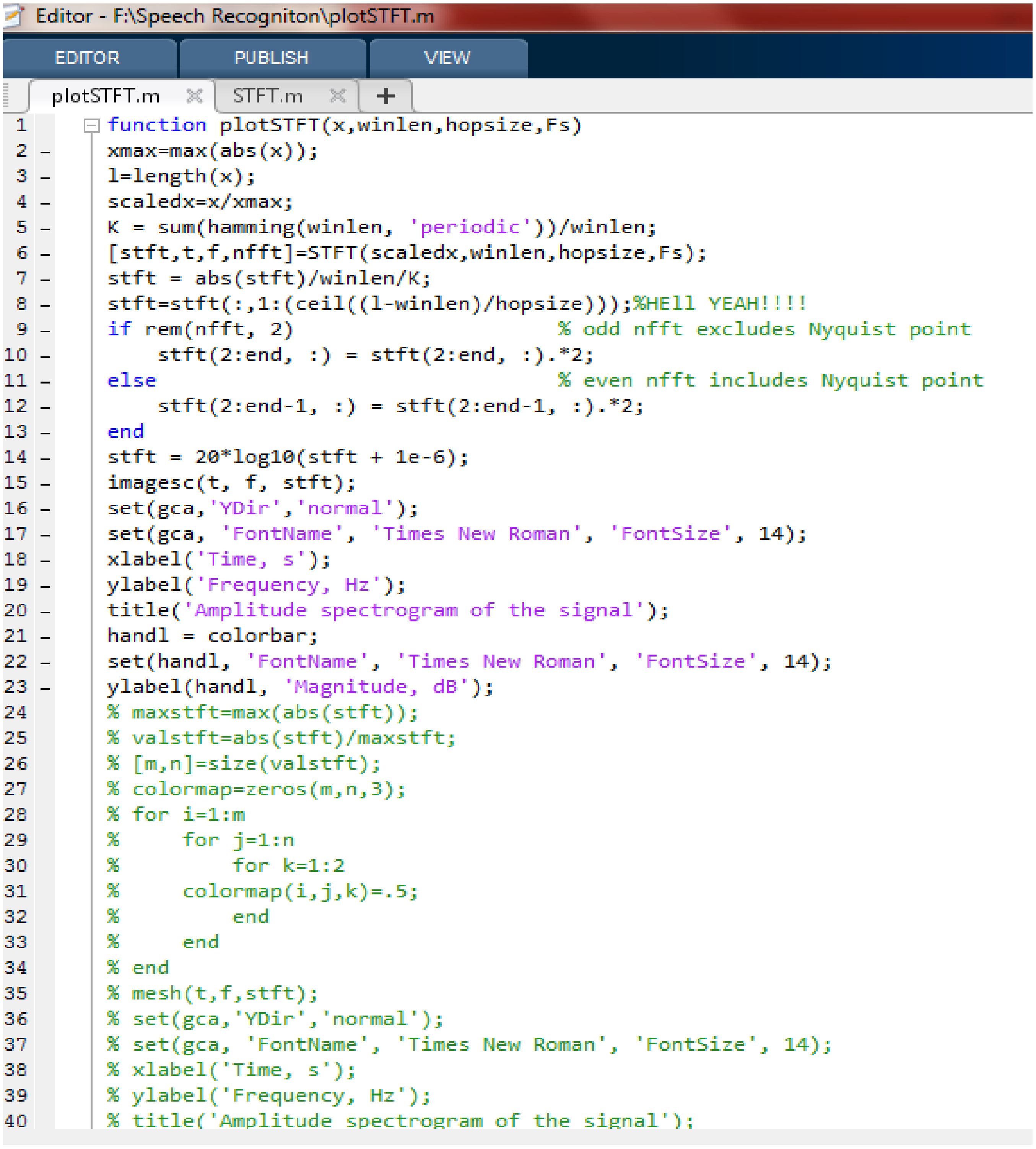
STFT can also be shown using mesh in MATLAB which gives 3-D plot of STFT as shown in Figure 1.5.

MATLAB code for SFTF is shown in Figure 1.6 and Figure 1.7 shows MATLAB code for plotting STFT of a signal.



```
function [STFT,time,freq,nfft]=STFT(x,winlen,hopsize,Fs)
%create Hamming window
hamwin=hamming(winlen,'periodic');
%check if x is a column vector
if size(x,2)>1
    x=x';
end
%determine nfft
l=length(x);
nfft=2^nextpow2(l);
% zero padding x
for i=1:nfft
    if i>l
        x(i)=0;
    end
end
%create STFT matrix (time in columns and frequency in rows)
columns=1+fix((nfft-winlen)/hopsize);
rows=ceil((1+nfft)/2);
STFT=zeros(rows,columns);%preallocation to increase speed!!
%multipy x with hamming window
step=0;
column=1;
while step+winlen<=nfft
xwin=hamwin.*x(step+1:(step+winlen));
xwinfft=fft(xwin,nfft);
STFT(:,column)=xwinfft(1:rows);
step=step+hopsize;
column=column+1;
end
freq=(0:rows-1)*Fs/nfft;%normalized frequency
time=(winlen/2:hopsize:l-winlen/2-1)/Fs;
end
```

Figure 1.6



```
function plotSTFT(x,winlen,hopsize,Fs)
xmax=max(abs(x));
l=length(x);
scaledx=x/xmax;
K = sum(hamming(winlen, 'periodic'))/winlen;
[stft,t,f,nfft]=STFT(scaledx,winlen,hopsize,Fs);
stft = abs(stft)/winlen/K;
stft=stft(:,1:(ceil((l-winlen)/hopsize)));%HEll YEAH!!!!
if rem(nfft, 2)                         % odd nfft excludes Nyquist point
    stft(2:end, :) = stft(2:end, :).*2;
else                                    % even nfft includes Nyquist point
    stft(2:end-1, :) = stft(2:end-1, :).*2;
end
stft = 20*log10(stft + 1e-6);
imagesc(t, f, stft);
set(gca,'YDir','normal');
set(gca, 'FontName', 'Times New Roman', 'FontSize', 14);
xlabel('Time, s');
ylabel('Frequency, Hz');
title('Amplitude spectrogram of the signal');
handl = colorbar;
set(handl, 'FontName', 'Times New Roman', 'FontSize', 14);
ylabel(handl, 'Magnitude, dB');
% maxstft=max(abs(stft));
% valstft=abs(stft)/maxstft;
% [m,n]=size(valstft);
% colormap=zeros(m,n,3);
% for i=1:m
%     for j=1:n
%         for k=1:2
%     colormap(i,j,k)=.5;
%         end
%     end
% end
% mesh(t,f,stft);
% set(gca,'YDir','normal');
% set(gca, 'FontName', 'Times New Roman', 'FontSize', 14);
% xlabel('Time, s');
% ylabel('Frequency, Hz');
% title('Amplitude spectrogram of the signal');
```

Figure 1.7

Filterbanks:

In signal processing, a filter bank is an array of band-pass filters that separates the input signal into multiple components, each one carrying a single frequency sub-band of the original signal. Mel frequency filter bank is a triangular filter banks which are equally spaced in Mel frequency domain. The relation between Mel frequency and conventional frequency measured in Hz is as Equ1.1. The filterbank used in this project is shown in Figure 1.8.

Equ 1.1 : $Mel(f) = 2595 \log_{10}(1 + \frac{f}{700})$

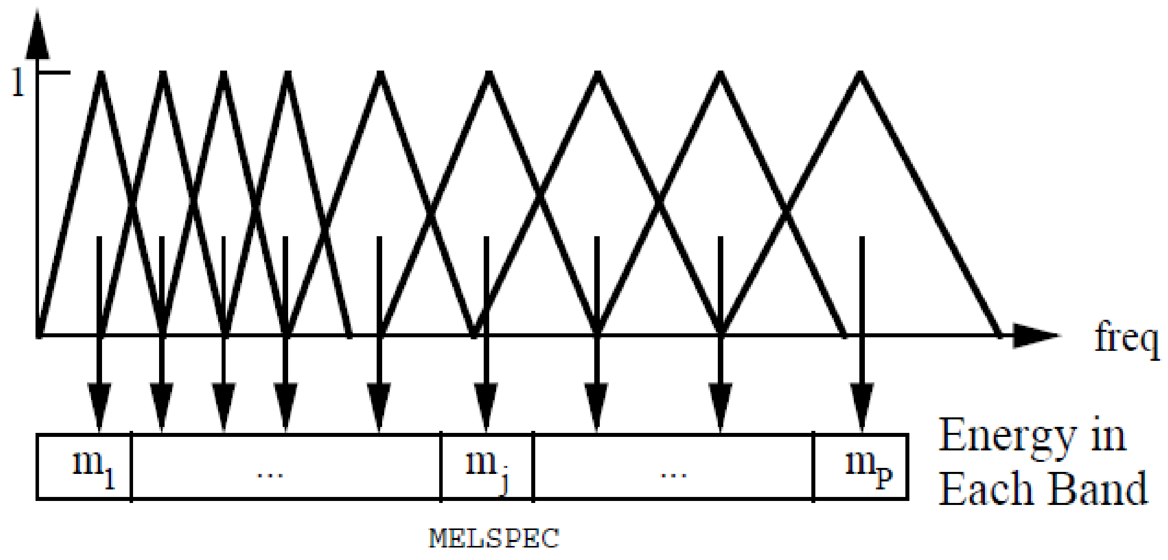
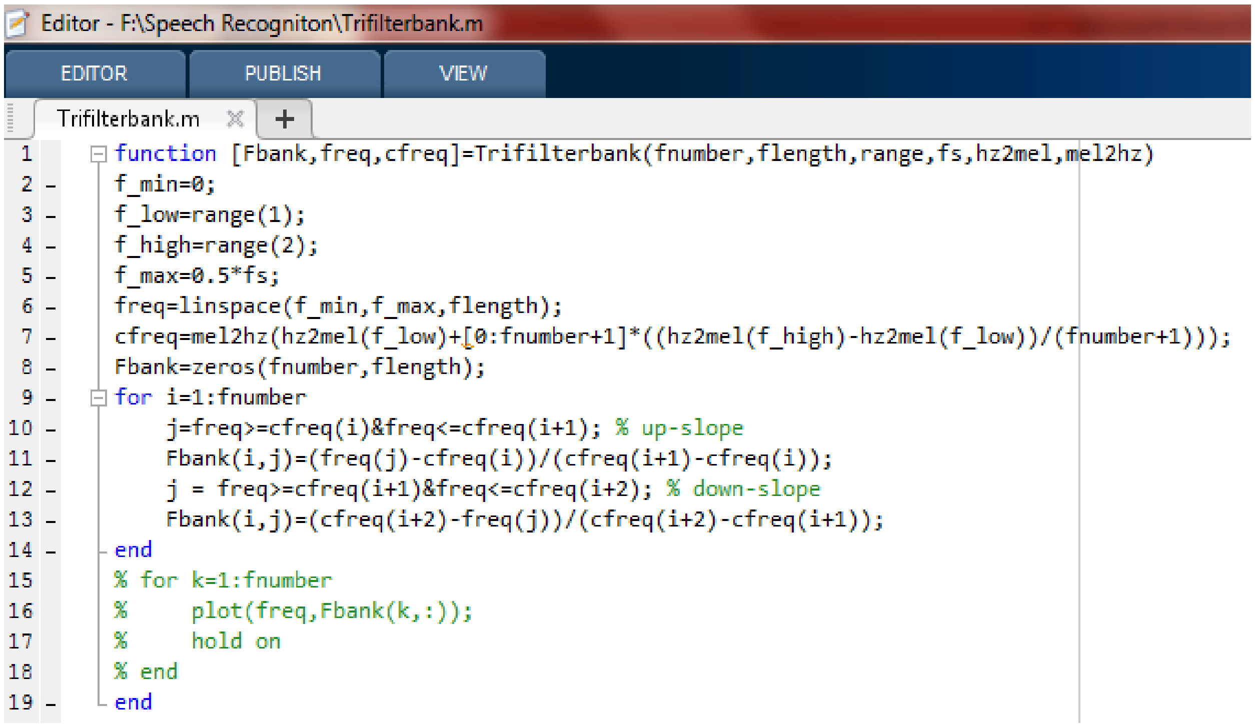


Figure 1.8

MATLAB code to create Mel frequency filter bank is shown in Figure 1.9. Note that **hz2mel** and **mel2hz** are function handles that are defined according to Equ1.1.



```matlab
function [Fbank,freq,cfreq]=Trifilterbank(fnumber,flength,range,fs,hz2mel,mel2hz)
f_min=0;
f_low=range(1);
f_high=range(2);
f_max=0.5*fs;
freq=linspace(f_min,f_max,flength);
cfreq=mel2hz(hz2mel(f_low)+[0:fnumber+1]*((hz2mel(f_high)-hz2mel(f_low))/(fnumber+1)));
Fbank=zeros(fnumber,flength);
for i=1:fnumber
    j=freq>=cfreq(i)&freq<=cfreq(i+1); % up-slope
    Fbank(i,j)=(freq(j)-cfreq(i))/(cfreq(i+1)-cfreq(i));
    j = freq>=cfreq(i+1)&freq<=cfreq(i+2); % down-slope
    Fbank(i,j)=(cfreq(i+2)-freq(j))/(cfreq(i+2)-cfreq(i+1));
end
% for k=1:fnumber
%     plot(freq,Fbank(k,:));
%     hold on
% end
end
```

Figure 1.9

Discrete cosine transform (DCT):

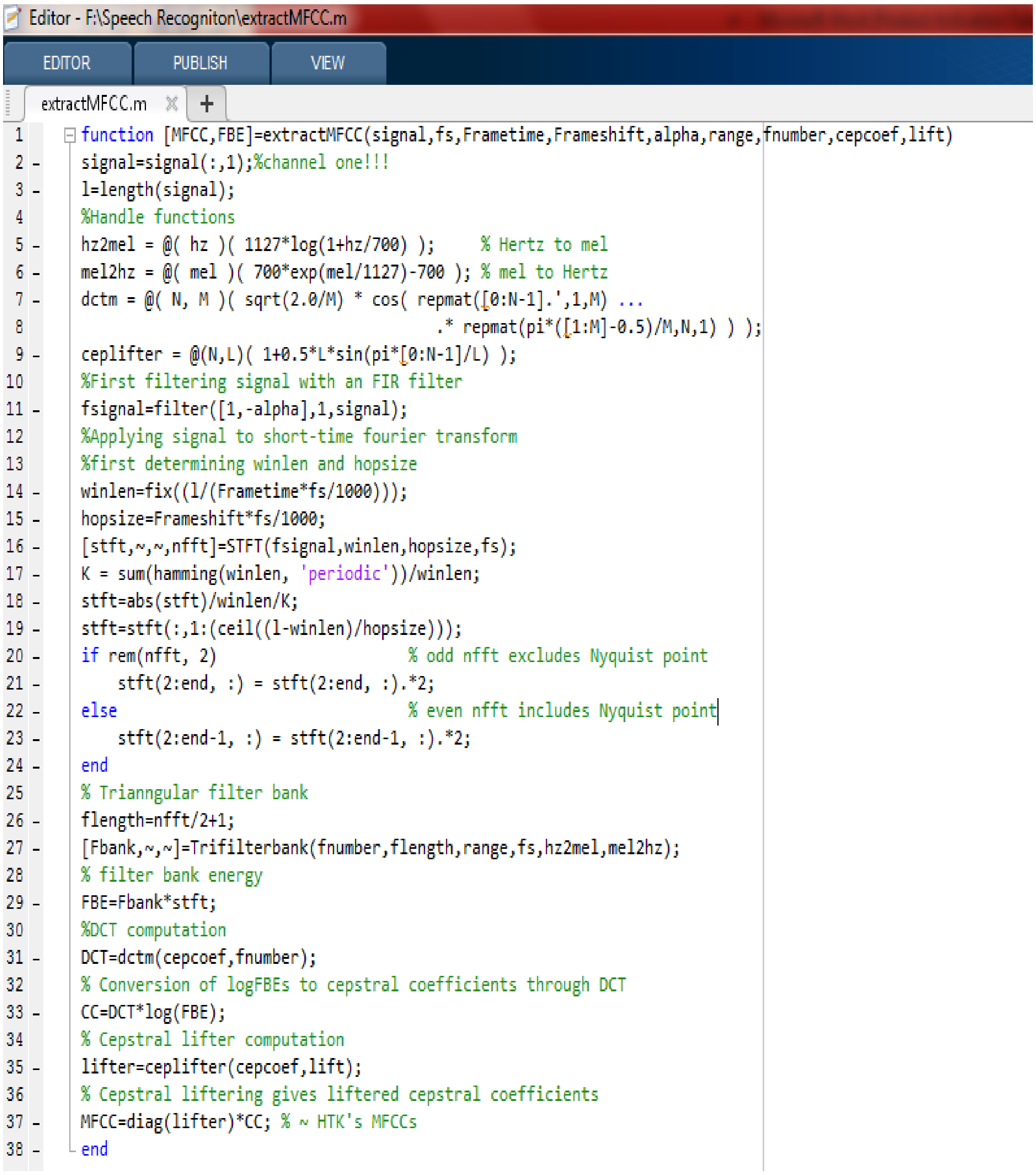
Discrete cosine transform expresses a signal in terms of cosine function. Its application in signal processing is for signal compaction. In this project we use DCT2 formula shown in Equ1.2.

Equ 1.2: $C(i) = \sqrt{\frac{2}{N}}\ \sum_{j=1}^{N} \cos(\frac{\pi i}{N}(j-0.5))$

In Equ1.2 N is the number of filter banks used filter bank design.

MFCC extraction:

MATLAB code to extract MFCC from voice is shown in Figure1.10.



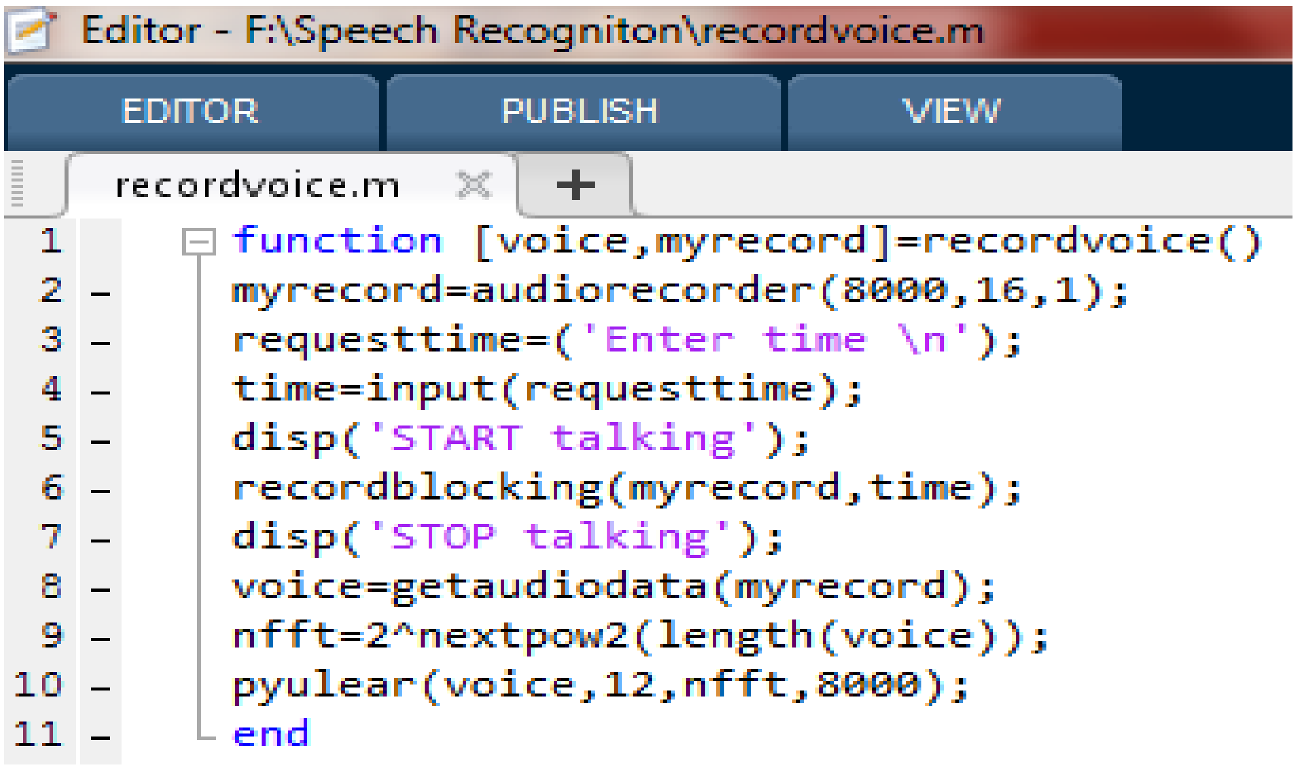
```
function [MFCC,FBE]=extractMFCC(signal,fs,Frametime,Frameshift,alpha,range,fnumber,cepcoef,lift)
signal=signal(:,1);%channel one!!!
l=length(signal);
%Handle functions
hz2mel = @( hz )( 1127*log(1+hz/700) );     % Hertz to mel
mel2hz = @( mel )( 700*exp(mel/1127)-700 ); % mel to Hertz
dctm = @( N, M )( sqrt(2.0/M) * cos( repmat([0:N-1].',1,M) ...
                                   .* repmat(pi*([1:M]-0.5)/M,N,1) ) );
ceplifter = @(N,L)( 1+0.5*L*sin(pi*[0:N-1]/L) );
%First filtering signal with an FIR filter
fsignal=filter([1,-alpha],1,signal);
%Applying signal to short-time fourier transform
%first determining winlen and hopsize
winlen=fix((l/(Frametime*fs/1000)));
hopsize=Frameshift*fs/1000;
[stft,~,~,nfft]=STFT(fsignal,winlen,hopsize,fs);
K = sum(hamming(winlen, 'periodic'))/winlen;
stft=abs(stft)/winlen/K;
stft=stft(:,1:(ceil((l-winlen)/hopsize)));
if rem(nfft, 2)                    % odd nfft excludes Nyquist point
    stft(2:end, :) = stft(2:end, :).*2;
else                               % even nfft includes Nyquist point
    stft(2:end-1, :) = stft(2:end-1, :).*2;
end
% Trianngular filter bank
flength=nfft/2+1;
[Fbank,~,~]=Trifilterbank(fnumber,flength,range,fs,hz2mel,mel2hz);
% filter bank energy
FBE=Fbank*stft;
%DCT computation
DCT=dctm(cepcoef,fnumber);
% Conversion of logFBEs to cepstral coefficients through DCT
CC=DCT*log(FBE);
% Cepstral lifter computation
lifter=ceplifter(cepcoef,lift);
% Cepstral liftering gives liftered cepstral coefficients
MFCC=diag(lifter)*CC; % ~ HTK's MFCCs
end
```

Figure 1.10

PSD(Power spectra density):

Another way to extract information from speech signal is using PSD or power spectrum density. PSD shows the average power of signal in a frequency band. A useful and convenient way to acquire PSD of signal is using Yule-Walker method to estimate PSD of speech signal. MATLAB has the function **pyulear** which can give us PSD estimation. In this project, in addition to MFCC feature extraction, we also give PSD of each sample voice to ANN for clustering and recognition.

Note that order of **pyulear** in MATLAB is 12 for applications of speech processing. MATLAB code to acquire Yule-Walker PSD estimation of a signal is shown in Figure 1.11.



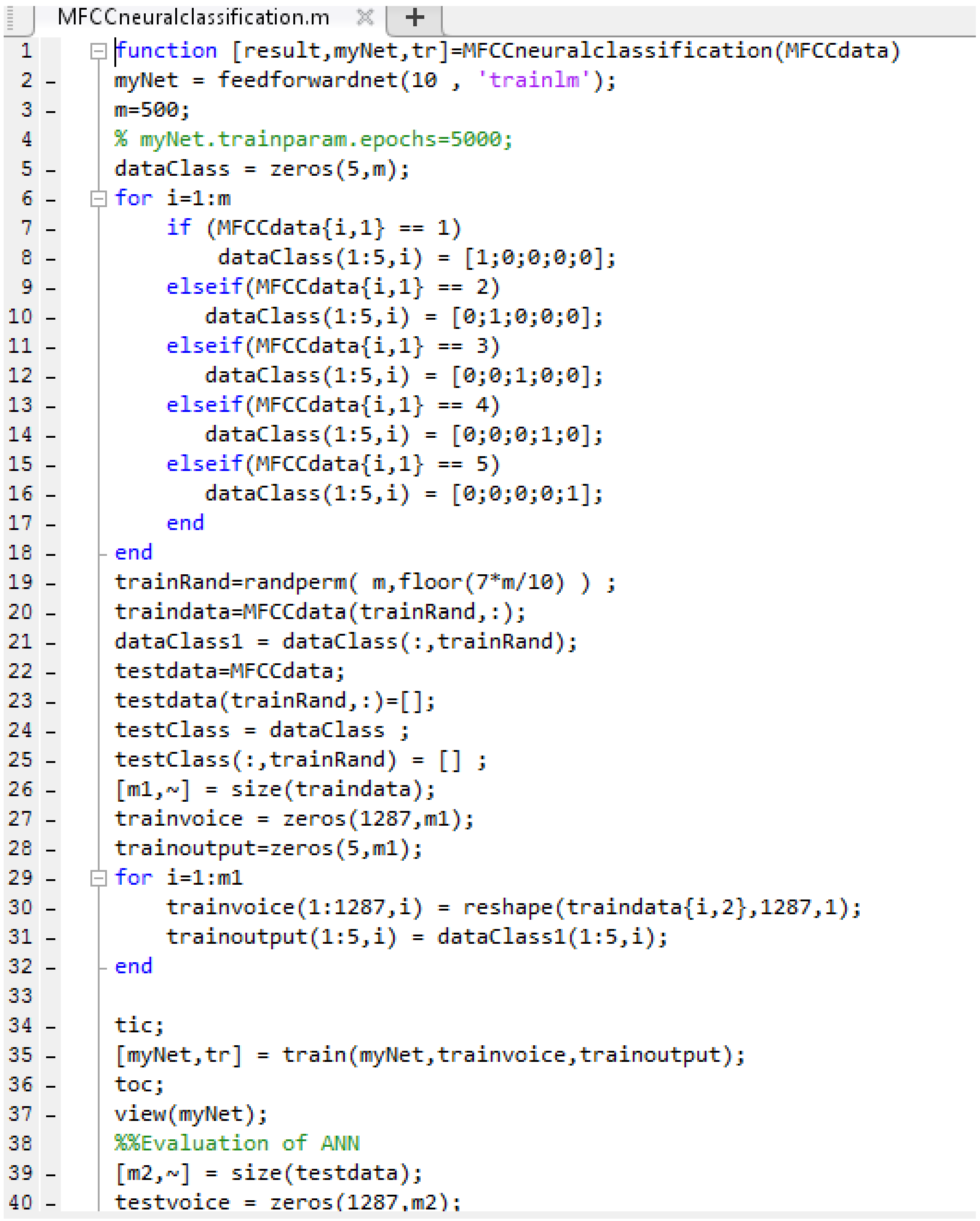
```
function [voice,myrecord]=recordvoice()
myrecord=audiorecorder(8000,16,1);
requesttime=('Enter time \n');
time=input(requesttime);
disp('START talking');
recordblocking(myrecord,time);
disp('STOP talking');
voice=getaudiodata(myrecord);
nfft=2^nextpow2(length(voice));
pyulear(voice,12,nfft,8000);
end
```

Figure 1.11

Voice Recognition:

After extracting information from speech signal using MFCC and Yule-Walker PSD estimation, we give these data to neural network. We assess both methods and compare them.

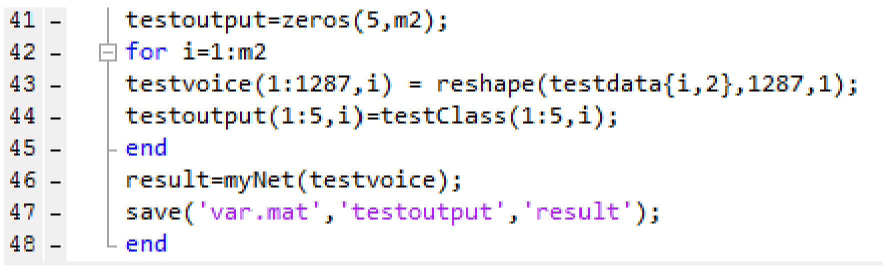
MATLAB code for ANN voice recognition based on MFCC is shown in Figure1.12.



MFCCneuralclassification.m

```
function [result,myNet,tr]=MFCCneuralclassification(MFCCdata)
myNet = feedforwardnet(10 , 'trainlm');
m=500;
% myNet.trainparam.epochs=5000;
dataClass = zeros(5,m);
for i=1:m
    if (MFCCdata{i,1} == 1)
        dataClass(1:5,i) = [1;0;0;0;0];
    elseif(MFCCdata{i,1} == 2)
       dataClass(1:5,i) = [0;1;0;0;0];
    elseif(MFCCdata{i,1} == 3)
       dataClass(1:5,i) = [0;0;1;0;0];
    elseif(MFCCdata{i,1} == 4)
       dataClass(1:5,i) = [0;0;0;1;0];
    elseif(MFCCdata{i,1} == 5)
       dataClass(1:5,i) = [0;0;0;0;1];
    end
end
trainRand=randperm( m,floor(7*m/10) ) ;
traindata=MFCCdata(trainRand,:);
dataClass1 = dataClass(:,trainRand);
testdata=MFCCdata;
testdata(trainRand,:)=[];
testClass = dataClass ;
testClass(:,trainRand) = [] ;
[m1,~] = size(traindata);
trainvoice = zeros(1287,m1);
trainoutput=zeros(5,m1);
for i=1:m1
    trainvoice(1:1287,i) = reshape(traindata{i,2},1287,1);
    trainoutput(1:5,i) = dataClass1(1:5,i);
end

tic;
[myNet,tr] = train(myNet,trainvoice,trainoutput);
toc;
view(myNet);
%%Evaluation of ANN
[m2,~] = size(testdata);
testvoice = zeros(1287,m2);
```

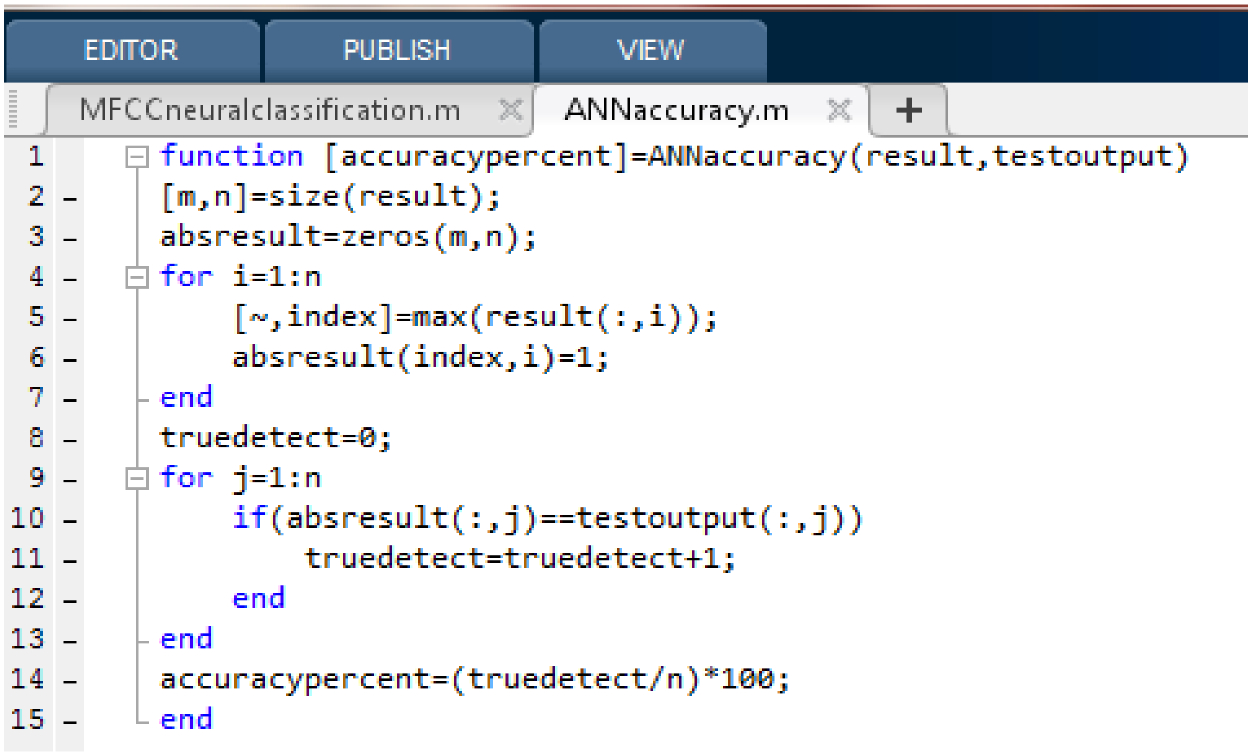


```
testoutput=zeros(5,m2);
for i=1:m2
testvoice(1:1287,i) = reshape(testdata{i,2},1287,1);
testoutput(1:5,i)=testClass(1:5,i);
end
result=myNet(testvoice);
save('var.mat','testoutput','result');
end
```

Figure 1.12

In this ANN, number of epochs is considered 5000 epochs but since the training method is trainlm, which means in every epoch all neuron coefficients are updated, training normally takes 10 to 14 epochs. Number of layers is 1 and there are 10 neurons in the code above.

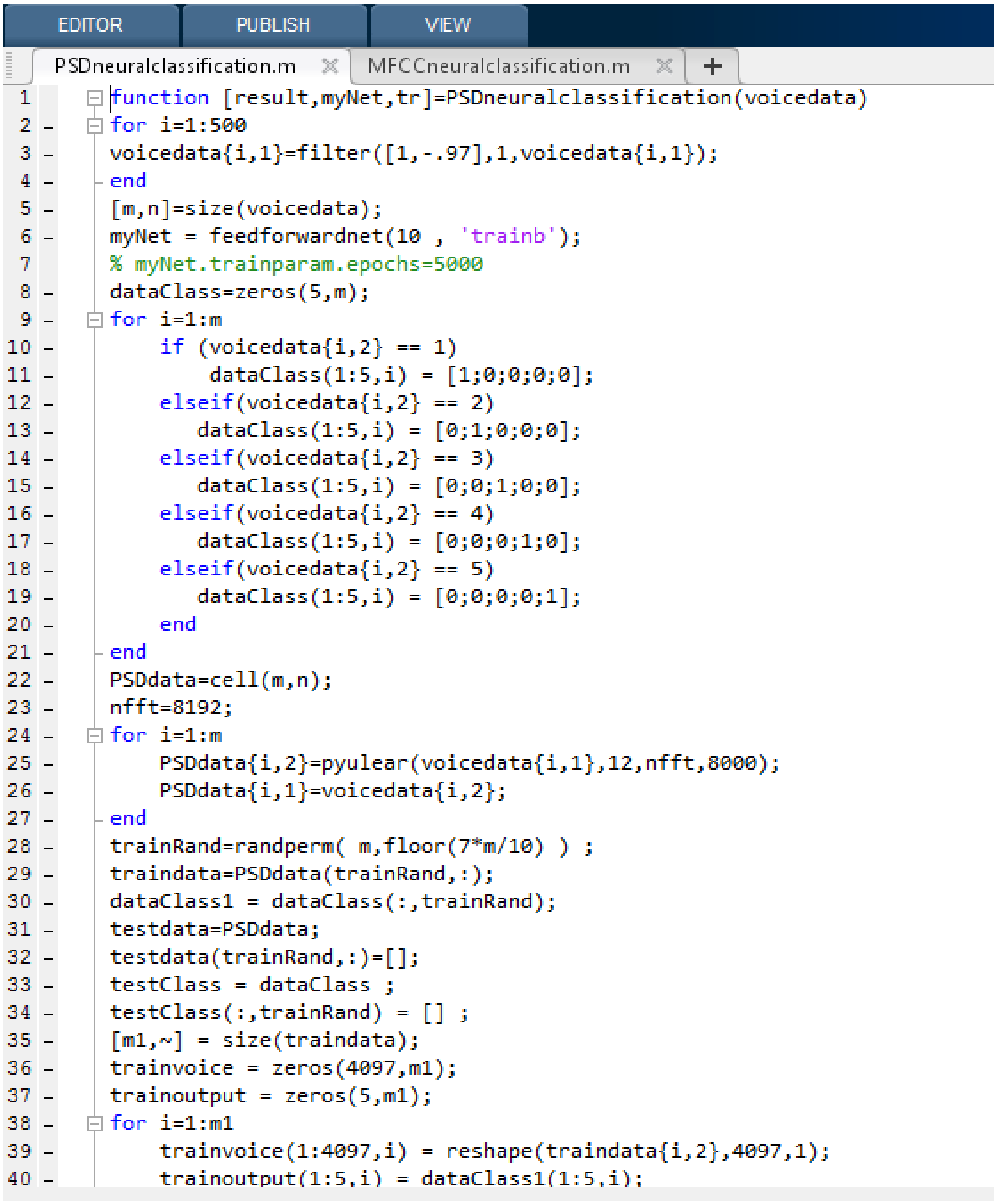
For determining the accuracy of ANN we need to compare result of ANN with the test data that we used for testing ANN. MATLAB code for accuracy assessment of ANN is shown in Figure1.13.



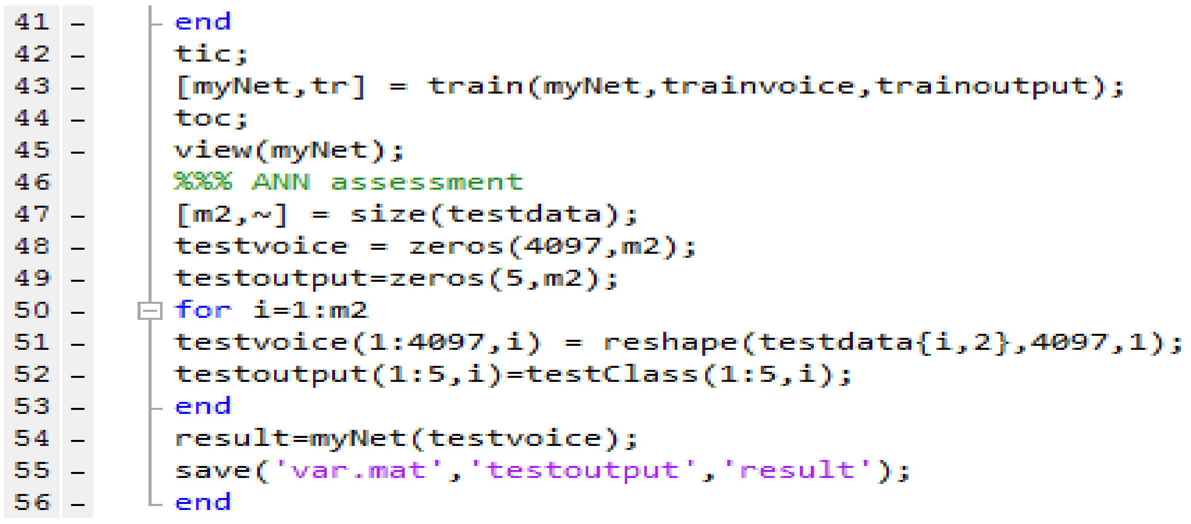
```
function [accuracypercent]=ANNaccuracy(result,testoutput)
[m,n]=size(result);
absresult=zeros(m,n);
for i=1:n
    [~,index]=max(result(:,i));
    absresult(index,i)=1;
end
truedetect=0;
for j=1:n
    if(absresult(:,j)==testoutput(:,j))
        truedetect=truedetect+1;
    end
end
accuracypercent=(truedetect/n)*100;
end
```

Figure 1.13

We also train the ANN using PSD estimation data. MALAB code for this ANN is shown in Figure1.14.



```
function [result,myNet,tr]=PSDneuralclassification(voicedata)
for i=1:500
voicedata{i,1}=filter([1,-.97],1,voicedata{i,1});
end
[m,n]=size(voicedata);
myNet = feedforwardnet(10 , 'trainb');
% myNet.trainparam.epochs=5000
dataClass=zeros(5,m);
for i=1:m
    if (voicedata{i,2} == 1)
        dataClass(1:5,i) = [1;0;0;0;0];
    elseif(voicedata{i,2} == 2)
       dataClass(1:5,i) = [0;1;0;0;0];
    elseif(voicedata{i,2} == 3)
       dataClass(1:5,i) = [0;0;1;0;0];
    elseif(voicedata{i,2} == 4)
       dataClass(1:5,i) = [0;0;0;1;0];
    elseif(voicedata{i,2} == 5)
       dataClass(1:5,i) = [0;0;0;0;1];
    end
end
PSDdata=cell(m,n);
nfft=8192;
for i=1:m
    PSDdata{i,2}=pyulear(voicedata{i,1},12,nfft,8000);
    PSDdata{i,1}=voicedata{i,2};
end
trainRand=randperm( m,floor(7*m/10) ) ;
traindata=PSDdata(trainRand,:);
dataClass1 = dataClass(:,trainRand);
testdata=PSDdata;
testdata(trainRand,:)=[];
testClass = dataClass ;
testClass(:,trainRand) = [] ;
[m1,~] = size(traindata);
trainvoice = zeros(4097,m1);
trainoutput = zeros(5,m1);
for i=1:m1
    trainvoice(1:4097,i) = reshape(traindata{i,2},4097,1);
    trainoutput(1:5,i) = dataClass1(1:5,i);
```



```
    end
    tic;
    [myNet,tr] = train(myNet,trainvoice,trainoutput);
    toc;
    view(myNet);
    %%% ANN assessment
    [m2,~] = size(testdata);
    testvoice = zeros(4097,m2);
    testoutput=zeros(5,m2);
    for i=1:m2
    testvoice(1:4097,i) = reshape(testdata{i,2},4097,1);
    testoutput(1:5,i)=testClass(1:5,i);
    end
    result=myNet(testvoice);
    save('var.mat','testoutput','result');
    end
```

Figure 1.20

Deduction:

We tested both ANNs with voice data and the result showed that MFCC based trained ANNs work considerably more precise than PSD based trained ANNs. When using MFCC based training for ANN, we used both trainb and trainlm method for training ANN. The trainb method trains the ANN more rapidly than trainlm method does but in some cases it has less accuracy than trainlm method. In MFCC based training both trainb and trainlm method lead to an astonishing 97% accuracy when they're tested, so we can surely say that MFCC based voice recognition is significantly more robust and precise than PSD based Voice recognition.

References:

I. Mathworks.com

II. HTK book by Steve Young et al. Cambridge University Engineering Department

III. htk.eng.cam.ac.uk

IV. Discrete time signal processing, Alan V Oppenheim. 3rd edition.