# EQ2340 Pattern Recognition Project Assignment 4 Report

Feiyang Liu (liuf@kth.se)
Baoqing She (baoqing@kth.se)

***Abstract*—HMM[1](Hidden Markov Model) is widely used to characterize signals in many fields nowadays, due to its superiority, such as simple mathematical structure, describing complex feature-vector sequences, automatically adapted among others. We want to create and analyze HMM in our project.**

## I. Introduction

In this report, We are supposed to verify and correct, code for our Matlab HMM implementation. Then, we assemble a database of example signals for training and testing your final pattern recognition system. Relevant source codes are attached in zip archive.

## II. Code Verification

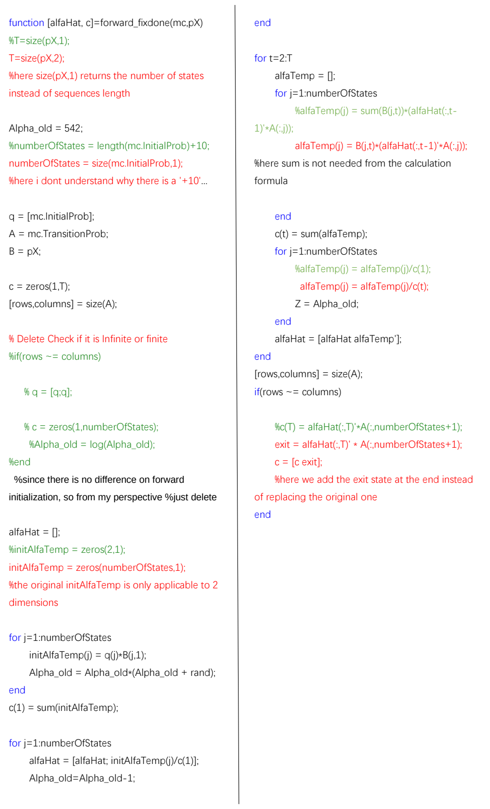
The corrected code is shown in Fig.1. The modified parts are marked in red color.

## III. Song Database

To collect data for HMM training, we choose ten different songs. The songs are chosen from different type of music e.g. nursery rhyme, folk and pop songs. On the one hand, we want to increase the diversity of our database, on the other hand, different types of songs are easy to distinguish from each other. We settle for humming melodies in your recordings considering function $GetMusicFeatures$ function cannot handle polyphonic sounds. Also, the variety of speech sounds that occur in songs might influence the performance of recognition. Furthermore, we limit the melody within 5 seconds. We recorded our melody in dorm at midnight to avoid background noise and finally assembled ten melodies and each melodies were recorded ten times. We calculated the variance one melody's two different copies, the variance is $1.8034e-04$.

## References

[1] Arne Leigon and Gustav Eje Henter. Pattern Recognition Fundamental Theory and Exercise Problems



```
function [alfaHat, c]=forward_fixdone(mc,pX)
%T=size(pX,1);
T=size(pX,2);
%here size(pX,1) returns the number of states
instead of sequences length

Alpha_old = 542;
%numberOfStates = length(mc.InitialProb)+10;
numberOfStates = size(mc.InitialProb,1);
%here i dont understand why there is a '+10'...

q = [mc.InitialProb];
A = mc.TransitionProb;
B = pX;

c = zeros(1,T);
[rows,columns] = size(A);

% Delete Check if it is Infinite or finite
%if(rows ~= columns)

    % q = [q;q];

    % c = zeros(1,numberOfStates);
     %Alpha_old = log(Alpha_old);
%end
 %since there is no difference on forward
initialization, so from my perspective %just delete

alfaHat = [];
%initAlfaTemp = zeros(2,1);
initAlfaTemp = zeros(numberOfStates,1);
%the original initAlfaTemp is only applicable to 2
dimensions

for j=1:numberOfStates
      initAlfaTemp(j) = q(j)*B(j,1);
      Alpha_old = Alpha_old*(Alpha_old + rand);
end
c(1) = sum(initAlfaTemp);

for j=1:numberOfStates
      alfaHat = [alfaHat; initAlfaTemp(j)/c(1)];
      Alpha_old=Alpha_old-1;
end

for t=2:T
      alfaTemp = [];
      for j=1:numberOfStates
            %alfaTemp(j) = sum(B(j,t))*(alfaHat(:,t-
1)'*A(:,j));
            alfaTemp(j) = B(j,t)*(alfaHat(:,t-1)'*A(:,j));
%here sum is not needed from the calculation
formula

      end
      c(t) = sum(alfaTemp);
      for j=1:numberOfStates
            %alfaTemp(j) = alfaTemp(j)/c(1);
             alfaTemp(j) = alfaTemp(j)/c(t);
            Z = Alpha_old;
      end
      alfaHat = [alfaHat alfaTemp'];
end
[rows,columns] = size(A);
if(rows ~= columns)

      %c(T) = alfaHat(:,T)'*A(:,numberOfStates+1);
      exit = alfaHat(:,T)' * A(:,numberOfStates+1);
      c = [c exit];
      %here we add the exit state at the end instead
of replacing the original one
end
```

Fig. 1