Starting with the front-end feature extraction.

First, the Mel-Filter should be built. From the definition of the Mel-frequency:

$$f_{mel} = mel(f) = 2595 \log\left(1 + \frac{f}{700}\right)$$

$$f = real(f_{mel}) = 700\left(e^{\frac{f_{mel}}{2595}} - 1\right)$$

The default start frequency is 0, and the highest frequency is the half of sampling frequency (from Nyquist sampling theorem). We also need to know the scale of FFT length per block in order to get resolution of the filtering. And the half-bandwidth for the triangle filter is the half of the FFT length.

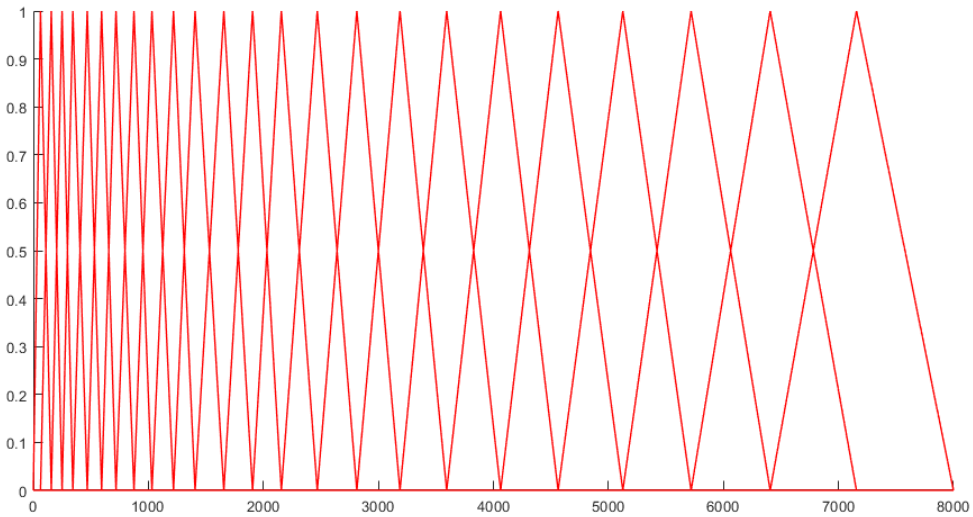
$$BW_{0.5} = \frac{nfft}{2} + 1; \quad R = \frac{f_s}{nfft}$$

Normally, there are 24 Mel-filter is built. So it will be 24 loops to create each.

After converting the frequency range into Mel-frequency range, the loop will pick the three frequency points (in normal frequency axis) as the three end points for creating each triangle filter, calculating the corresponding label by dividing the frequency by the resolution dt. Then linearly connect between two neighbor labels. The linear range will be the half-bandwidth.

```
f1=F(k-1);f2=F(k+1);f0=F(k);
n1=round(f1/R)+1;n2=round(f2/R)+1;n0=round(f0/R)+1;%Matlab index starts from 1
```

Then the frequency response of the Mel-Filter Bank will be:



With the Mel-filter bank, the process of MFCC feature extraction can be realized with following steps:

1. Initializing the Mel-filter bank for specific frequency start and ending range, and the FFT length.

2. Enframing the speech signal.

Intercepting the speech signal as one frame, the length of one frame is chosen by the user, and next frame starts from the middle of the last frame.

In the question, the sampling frequency is 16000Hz, and in order to get a 20ms frame, the length of each frame will be: $16000\text{Hz} \cdot 20\text{ms} = 320\ \text{units}$. So each frame contains 320 samples. And each two-frame neighbor frame will overlap 10ms, which means the increment of the frame is $16000\text{Hz} \cdot 10\text{ms} = 160\ \text{units}$.

Implemented with code:

```
for i=1:num_of_frame
frame(:,i)=a_2((i-1)*window_step+1:(i-1)*window_step+window_scale);
end
```

and the matrix frame stores the frames of the original speech sequence.

3. Processing each frame with FFT, filtering, log computing and DCT.

```
current=frame(i,:)'%pick one frame
```

perform the FFT, and calculates its energy. Then filtering with the Mel-filter bank.

```
mag_fft=abs(fft(current)).^2;      temp=mel*mag_fft(1:n2);
```

Log computation: `temp=log10(temp);`

Finally, the first 13 value from the DCT transformation is the MFCC feature elements.

```
temp=dct(temp);         mfcc_vector(i,:)=temp(1:13)';
```

And the feature extraction in the front-end is done.

As there are 10 folders with each containing 10 wav file. So an efficient way to load those data is loading the root path.

```
path='\Data\Data\Speaker';
```

realized by a 2-depth loop, one for choosing the speaker, and the second for choosing which data of this speaker is used to extract the feature. In the outer loop:

```
current_speaker=strcat(path,num2str(i));
current_wav=strcat(current_speaker,'\*.wav');
files=dir(current_wav);%reading all the wav file name into the matrix
```

The inner loop:

```
[a,fs]=audioread(strcat(current_speaker,'\',files(j).name));
mfcc_vector=mfcc_c(a,fs,24,512); %calling the mfcc function we built above
```

The MFCC uses 24 Mel-filters and 512 length FFT block.

For one speaker, the feature extracted should be appended to a same matrix.

Performing feature extraction for 70% of voice data, which means wo only use the first 7 wav files to build the model. So, when the inner loop index is smaller than 7, all the feature vectors will be stored in the speakers' training matrix, that is:

```
x=[x;mfcc_vector]; %in the inner loop concatenates all generate vectors
…
speaker{i}.mfcc=x;%inner loop is done, store the concatenated as a attribute
```

Otherwise the generated feature vectors will be stored as a new cell separately, that is:

```
speaker{i}.test{j+3-10}=mfcc_vector;
```

Additionally, the GMM-UBM requires a general speech data set, so the extracted vectors should be also added to a UBM matrix, which supposed to contain all the speakers' training speech features. So the matrix *x* can be reused.

```
ubm_train=[ubm_train;x];
```

After installing the dataset, turn to the training of the GMM-UBM model. UBM model is a GMM model, which can be estimated from the EM algorithm. As the UBM training matrix is obtained, we can use the ML-EM algorithm to estimate the UBM Gaussian Model. The ML-EM is recalled from the last assignment, and little modified so that it can be used in the case of arbitrary number of Gaussian distributions a GMM contains.

Training the GMM-UBM model with the UBM train data, and the returned parameters of GMM will be stored as a structure.

```
[ubm.weight, ubm.mu, ubm.sigma] = EM(ubm_train', M);
```

Once the UBM training is done, we need to get each speaker's GMM model by doing self-adaption on UBM model with specific MFCC feature vectors: `speaker{i}.mfcc.`

The design for MAP algorithm is similar to the ML-EM, as the estimation step is the same, while the maximization step remains the same except MAP algorithm will perform the weight (between the parameters lately calculated and those from UBM) calculation and applying for the Gaussian parameters additionally. Also, for MAP part, iteration is not needed, which means we only need to perform one E step and another one M step.

The design for the additional part of MAP is: refer to [1], the expression for the data-dependent adaption coefficient α_i^ρ is:

$$\alpha_i^\rho = \frac{n_i}{n_i + r^\rho}$$

Where n_i is the N_k used in the lecture notes (the sum of the posterior probability for the latent variable for all data in dataset). And the r^ρ is a fixed relevance factor, in the algorithm, we use the same as [1] mentioned: $r^\rho = 16$.

MAP: After performing the E step and M step in ML-EM once, turn to update the parameters with the adaption coefficient with the $n_i(N_k)$ calculated from this time's M step.

Get the adaption coefficient α and compute the new mean, the index *i* represents the i^{th} Gaussian Model. So the expression combines the new mean matrix with UBM mean matrix in a certain (α) proportion. Implemented by code:

```
alpha=zeros(M,1);
    for j=1:M
        alpha(j)=N_k(j)/(N_k(j)+16);
    end
```

And from updating law function:

$$\hat{\mu}_i = \alpha_i^m E_i(x) + (1 - \alpha_i^m)\mu_i$$

Where the $E_i(x)$ is the estimated new mean in the MAP M step, and μ_i is the UBM model's mean matrix.

```
new_mu=ones(dim,1)*alpha'.*mu+ones(dim,1)*(ones(M,1)-alpha)'.*ubm.mu;
```

Trying to implement all the parameters adaption function that [] has mentioned. With expressions of weights and variance matrix updating law:

$$\widehat{w}_i = \left[\frac{\alpha_i^w n_i}{T} + (1 - \alpha_i^w)w_i\right]\gamma$$

Implemented by code:

```
w=alpha.*N_k/N+(ones(M,1)-alpha).*(ubm.weight)';
w=w/sum(w); %ensure the sum of weight matrix is 1
```

For variance matrix:

$$\hat{\sigma}_i^2 = \alpha_i^v E_i(x^2) + (1 - \alpha_i^v)(\sigma_i^2 + \mu_i^2) - \hat{\mu}_i^2$$

This will require the M step's updated variance matrix, which is same as that part in ML-EM. And the updating law for the new variance matrix according to the MAP is:

```
    for j=1:M
        s(:,:,j)=alpha(j)*sigma_e(:,:,j)+...
            (1-alpha(j))*(sigma(:,:,j)+...
            diag(diag(mu_old(:,j)*mu_old(:,j)')))-...
            diag(diag(mu(:,j)*mu(:,j)'));
    end
```

Then the new mean new weight `w` and new variance `s` will be assigned to the new structure's attribute.

Then the adapted parameters are updated, and hence returning them to a new structure.

```
specific_mod.mu=new_muspecific_mod.weight=w;specific_mod.sigma=s;
```

To get 10 speakers GMM model, looping the MAP algorithm for 10 times to do the training:

```
for i=1:num_of_speaker
    speaker{i}.GMM=map(ubm,(speaker{i}.mfcc)');
end
```

Since we have built adapted GMM for every speaker, the verification can be realized by calculating the Loglikelihood. To verify a speech, calculating the Loglikelihood of the speech file in UBM-GMM model, and in the adapted speaker GMM models. So finally, 11 Loglikelihood value will be computed, and from [1], the ratio of Loglikelihood can be a basis to judge the test speech sequence belonging to which

speaker.

$$\Lambda(X) = \log p\left(X|\lambda_{hyp}\right) - \log p(X|\lambda_{ubm})$$

For the Loglikelihood function, the probability can be computed (for the fact that the expression of Gaussian Distribution is known) for each data point in all Gaussian Model and then accumulated, doing Log computation as the Log probability

```
p = p + weights(j) /(((2*pi)^(D/2))* sqrt(det(covariance)))…
 * exp(-0.5 *((m_D / covar)* m_D'));
…
logLikelihood = logLikelihood + log(p);
```

As there are 30 test speeches, 3 for each speaker, 3-depth loop need to built to do the verification: i for speaker choosing, j for test speech choosing, k for verifying speaker GMM choosing

```
test_data=speaker{i}.test{j};
```

And the current test data's Loglikelihood in UBM is:

```
log_ubm=LogLikelihood(ubm,test_data);
```

Then generate the Loglikelihood result of current test data in all speakers GMM model

```
log_gmm(k)=LogLikelihood(speaker{k}.GMM,test_data);
```

After the k-index loop finish, 11 Loglikelihood values are computed. Doing the subtraction, and find the maximum value in the 10 results, and the largest value's owner is the estimated speaker.

```
[~,estimated_speaker]=max(log_gmm-log_ubm*ones(10,1));
e=[e;estimated_speaker]; % store this test data's estimation result
…
confusion_matrix1(T,e); %generate the confusion matrix
```

Run for i speakers and j test speeches per speaker, the simulation result (estimated label store in e), the confusion matrix, is:

	Speaker1	Speaker2	Speaker3	Speaker4	Speaker5	Speaker6	Speaker7	Speaker8	Speaker9	Speaker10
Speaker1	2.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
Speaker2	1.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	0.00	0.00
Speaker3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00	0.00	0.00
Speaker4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	0.00	1.00
Speaker5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00	0.00	0.00
Speaker6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00	0.00	0.00
Speaker7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00	0.00	0.00
Speaker8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00	0.00	0.00
Speaker9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00	0.00	0.00
Speaker10	1.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	0.00	0.00

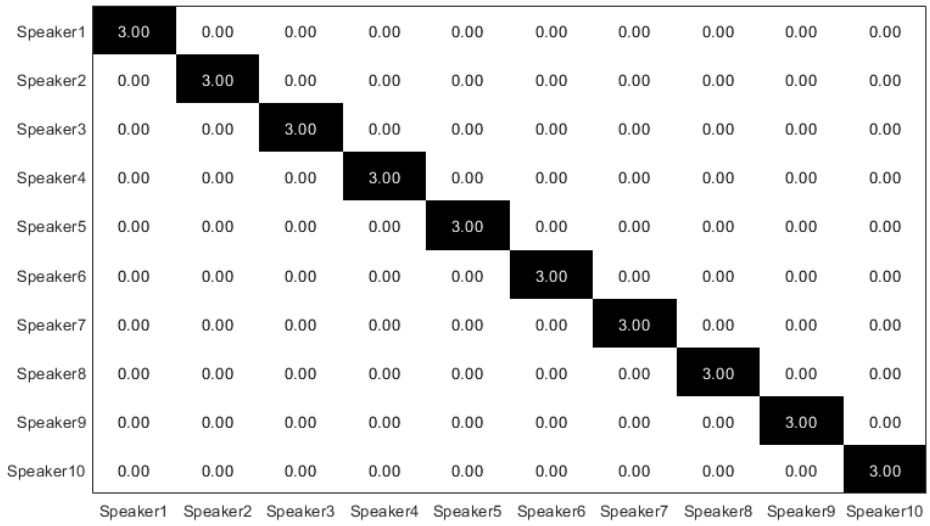
This confusion matrix shows a disappointing result that most test utterances are classified into Speaker8. This may be attributed to the too specific model for each

speaker, i.e. updating too much details. And probably the Speaker 8's statistical property takes the dominant place in the GMM, which may be caused by a relatively longer training utterance length to the other speakers.

Refer to [2], the performance of speech identification is better when only altering the mean matrix. So change the code to keep the weight and sigma the same as UBM:

```
specific_mod.weight=ubm.weight; specific_mod.sigma=ubm.sigma;
```

And the reconstruct the GMM for every speaker, then verify with the same test speech data sets. The returned result is:



It shows a relatively good performance as each test data is correctly recognized. It is noticed that the test and training data does not involve any noise which means the test data is perfect. For general cases, noise is very common for speech verification, and it will largely affect the performance of the system and in this program, no anti-noise algorithm is applied so that it may not work well in a general case. And the method that only adapting the mean matrix should be picked instead of adapting all parameters.

And in this case, the de-silence algorithm is implemented in an extreme simple way, that is removing the part of utterance that the magnitude is very close to 0, so that its performance is not ideal. To go further, the de-silence should be realized by the theory of voice endpoint detection, which is related to some property of voice such as the instantaneous energy of uttering, the minimum lasting time

Reference:

[1] Douglas A. Reynolds, Thomas F. Quatieri, and Robert B. Dunn, "Speaker Verification Using Adapted Gaussian Mixture Models", Digital Signal Processing 10, 19–41 (2000)

[2] ZHOU Guo-xin，GAO Yong, "Research on Speaker Identification Based on GMM-UBM Model", Radio engineering, 2014,44 (12): 14－17