## Table of Contents



```
function db = make_table(song)
```

# ECE 203 - Lab 5 Part B


Tyler Roberts & Bob Wagner


# 2.2 Preprocessing

```
%Read in viva.mp3 file
[y,fs]=audioread(song);
y=mean(y,2);
%fs=44.1 kHz, this is a lot more data than needed
%Resample the signal at 8000Hz
old_smpl_rate=fs;
new_smpl_rate=8000;
y=resample(y,new_smpl_rate,old_smpl_rate);
```

```
Error using make_table (line 7)
Not enough input arguments.
```

# 2.3 Spectrogram

```
%Construct the spectrogram of the signal
%Define variables
window = 64e-3*8000;%ms %Integer indicating the length of the chunks that are bein
noverlap = 32e-3*8000;%ms %Number of samples that have an overlap bewtween adjacen
nfft=window; %Length of the fft being taken, equal to window in this case
fs=new_smpl_rate; %Sampling rate of the signal
%[S,F,T]=spectrogram(y,window,noverlap,nfft,fs);
[S,~,T]=spectrogram(y,window,noverlap,nfft,fs);
%S = spectrogram w/ only positive frequencies
%F = frequency vector for the vertical axis
%T = time vector for the horizontal axis

%Plot the magnitude of the spectrogram of the song
figure(1)
plot(T,abs(S));
xlabel('Time (ms)');
ylabel('Positive Frequencies of the Song');
```

```
%Plot the magnitude of the spectrogram of the song (logarithmic)
figure(2)
semilogy(T,abs(S));
xlabel('Time (ms)');
ylabel('Positive Frequencies of the Song');
```

# 2.4 Spectrogram Local Peaks

```
%Find the local peaks of the spectrogram

figure(3)
%input each gs x gs sub-matrix into gsMatrixEval, making sure to use this
%information to modify (boolean 1) the P matrix at specified x & y pos

gs=9;
P = zeros(size(S,1),size(S,2));

for k = 1:ceil(size(S, 1)/gs)
    for l = 1:ceil(size(S, 2)/gs)
        if(k == ceil(size(S, 1)/gs)) % x-dimension may be < gs
            if(l == ceil(size(S,2)/gs)) % both dimensions may be < gs
                %max_x = size(S,1)-(k*gs);
                %max_y = size(S,2)-(l*gs);
                [xpos_temp, ypos_temp] = gsMatrixEval( S(((k-1)*gs)+1:size(S,1),((
                if(~(xpos_temp == -1 || ypos_temp == -1))
                    P(xpos_temp, ypos_temp) = 1;
                end
            else % only x-dimension may be < gs, but *not* both
                [xpos_temp, ypos_temp] = gsMatrixEval( S(((k-1)*gs)+1:size(S,1),((
                if(~(xpos_temp == -1 || ypos_temp == -1))
                    P(xpos_temp, ypos_temp) = 1;
                end
            end
        elseif(l == ceil(size(S, 2)/gs)) % y-dimension may be < gs
            [xpos_temp, ypos_temp] = gsMatrixEval( S(((k-1)*gs)+1:((k)*gs),((l-1)*
            if(~(xpos_temp == -1 || ypos_temp == -1))
                P(xpos_temp, ypos_temp) = 1;
            end
        else % neither dimension could be < gs
            [xpos_temp, ypos_temp] = gsMatrixEval( S(((k-1)*gs)+1:((k)*gs),((l-1)*
            P(xpos_temp, ypos_temp) = 1;
        end
    end
end

imshow(uint8(255*P))
colormap(1-gray);
```

# 2.5 Thresholding

```
figure(4)
```

```
average = 9e5;
freqThreshold = 0;

while(average > 30) % while we have an average less than 30
    %freqThreshold = freqThreshold + .005;
    freqThreshold = freqThreshold + .02;
    columnsPerSecond = floor(size(S, 2) / T(1, size(T,2)));
    numFrequenciesMet = zeros(1, ceil(size(P,2)/columnsPerSecond));
    % iterate through P, size(P,1) x columnsPerSecond matricies at a time,
    % and throw out values that don't meet the threshold
    for r = 1: ceil(size(P,2)/columnsPerSecond)
        for k = 1:size(P, 1)
            if(r*columnsPerSecond > size(P,2))
                bound = size(P,2);
            else
                bound = r*columnsPerSecond;
            end
            for l = ((r-1)*columnsPerSecond)+1:bound
                if(P(k,l) == 1)
                    if(abs(S(k,l)) < freqThreshold) % if below threshold, del
                        P(k,l) = 0;
                    else % increase number of frequences that met the threshold
                        % for this size(P,1) x columnsPerSecond matrix
                        numFrequenciesMet(1,r) = numFrequenciesMet(1,r)+1;
                    end
                end
            end
        end
    end

    average = mean(numFrequenciesMet); % compute average/mean
end

fprintf('The average number of frequencies that have met the threshold per second

imshow(uint8(255*P))
colormap(1-gray);
```

# 2.6 Constructing the Table

```
% USE COMMAND 'find'

db = [];

found = find(P); %Nx1 matrix

for k=1:size(found,1)

    row = mod(found(k), size(P,1));
    if(row == 0) %% if edge case, set to edge (e.g. 21 mod 21 = 0 should have row
        row = size(P,1);
    end
    col = floor(found(k)/size(P,1))+1;
```

```
        db = [db; evalBox(S, P, T, row, col)];
    end
end
```

*Published with MATLAB® R2014a*