LAB#06

A huge thanks to Ms. Mehri for her kind support thus far ☺

Shayan Vassef, UIN = 674579894, $N_{id} = 894$

1 Tone detection to decode telephone numbers

The loaded variables after loading the. mat file in MATLAB:

Name	Size	Bytes	Class
Fs	1x1	8	double
tpn0	2048x1	16384	double
tpn1	2048x1	16384	double
tpn2	2048x1	16384	double
tpn3	2048x1	16384	double
tpn4	2048x1	16384	double
u0	256x1	2048	double
u1	256x1	2048	double
u2	256x1	2048	double
u3	256x1	2048	double
u4	256x1	2048	double
u5	256x1	2048	double
u6	256x1	2048	double
u7	256x1	2048	double
u8	256x1	2048	double
u9	256x1	2048	double

Q6.1

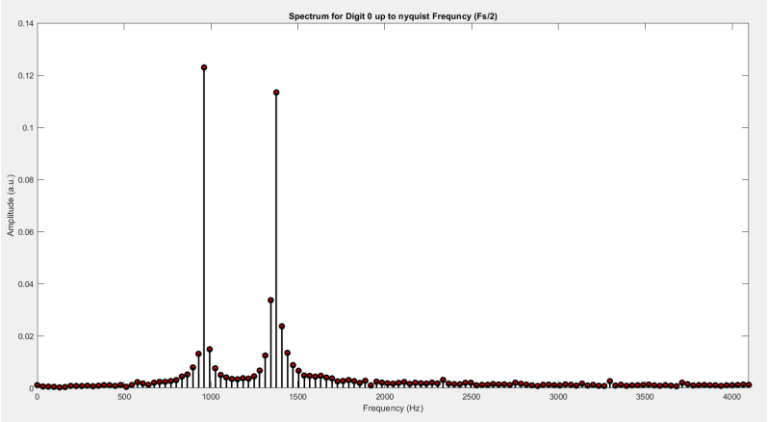

Fig 1. Fourier series for digit 0

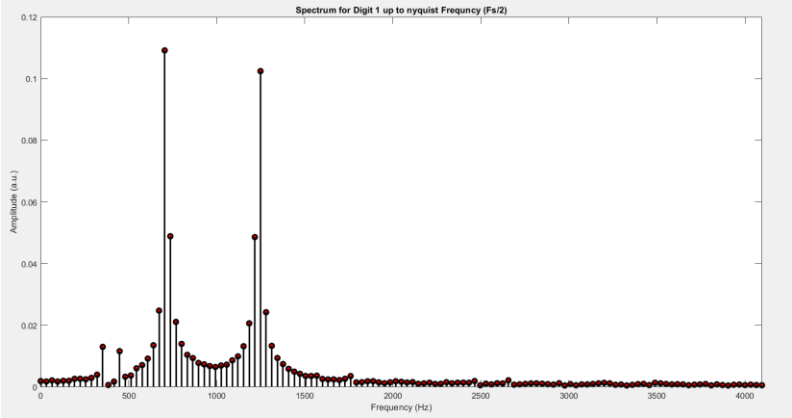

Fig 2. Fourier series for digit 1

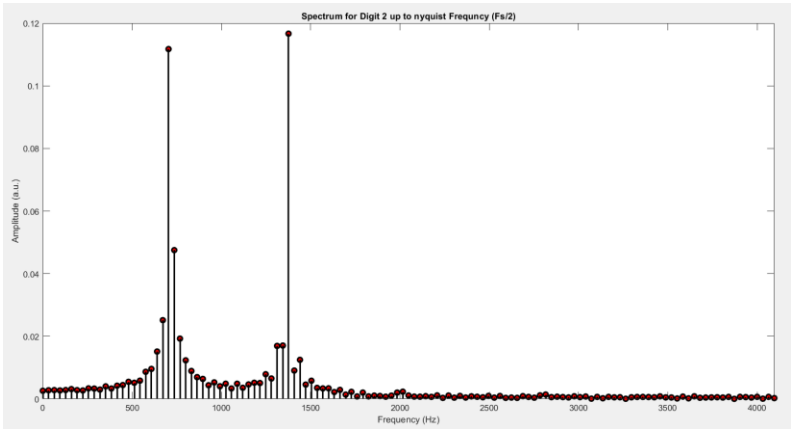

Fig 3. Fourier series for digit 2

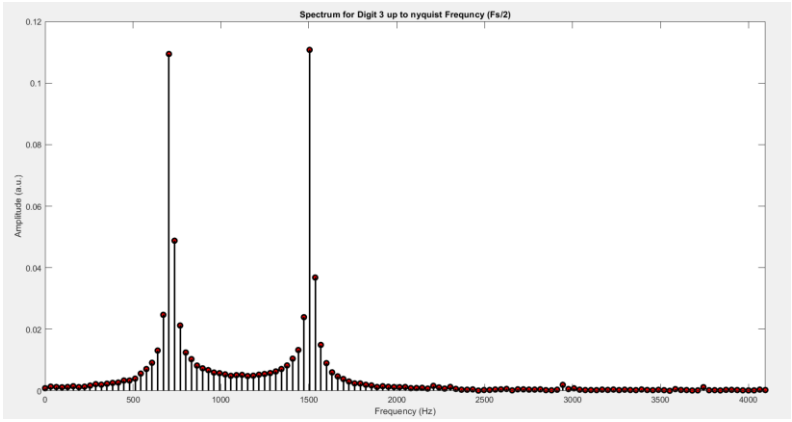

Fig 4. Fourier series for digit 3

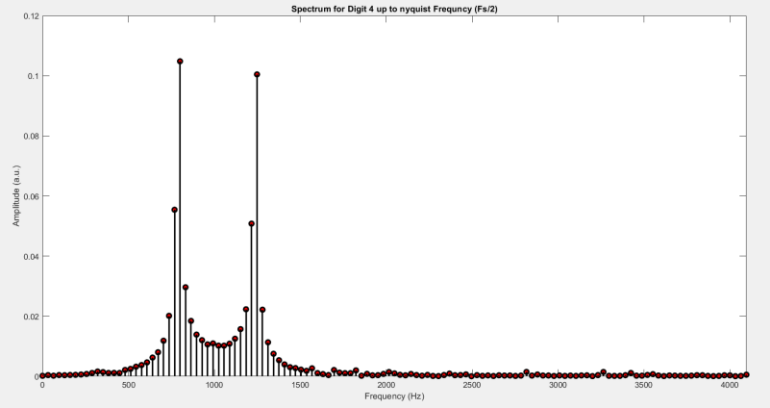

Fig 5. Fourier series for digit 4

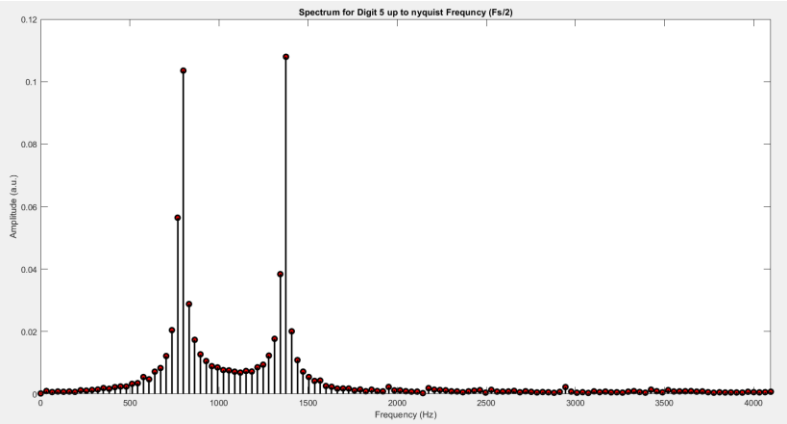

Fig 6. Fourier series for digit 5

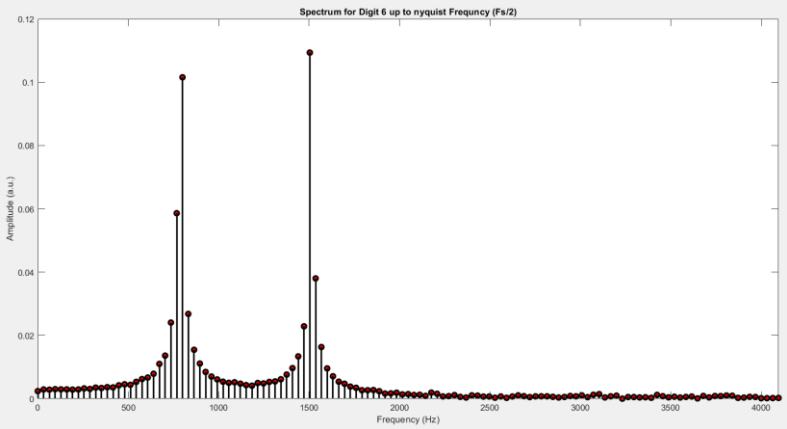

Fig 7. Fourier series for digit 6

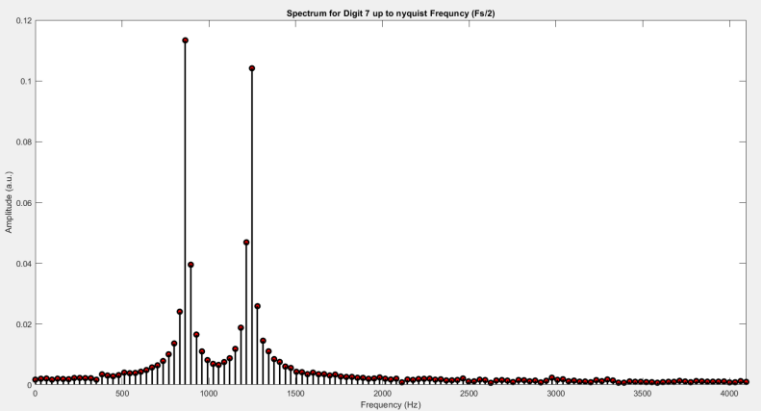

Fig 8. Fourier series for digit 7

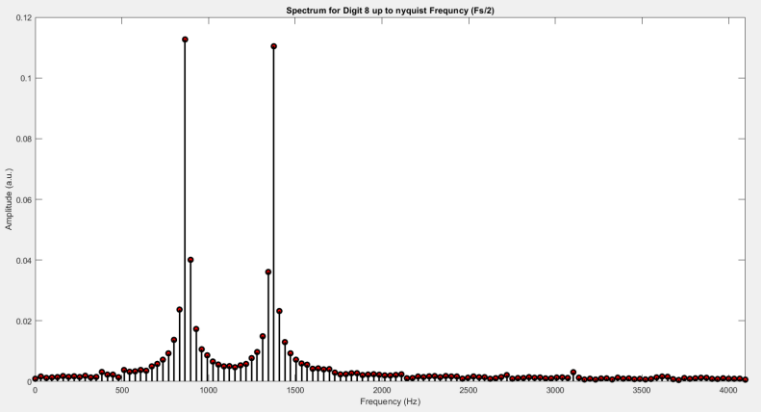

Fig 9. Fourier series for digit 8

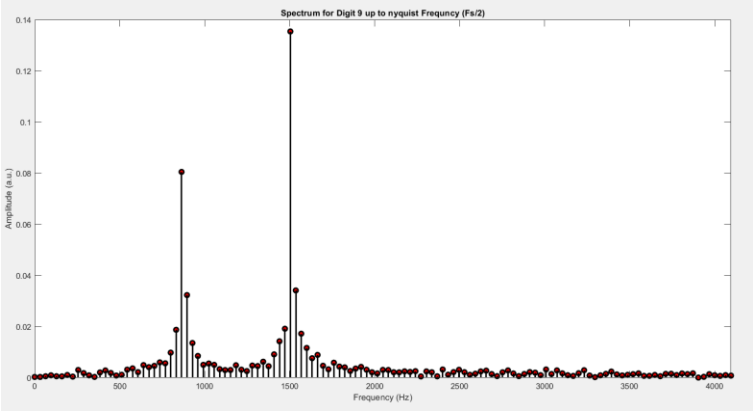

Fig 10. Fourier series for digit 9

digit	freq1	freq2	Energy
0	960	1376	15.97
1	704	1248	16.178
2	1376	704	16.185
3	1504	704	16.204
4	800	1248	15.993
5	1376	800	15.898
6	1504	800	16.037
7	864	1248	15.995
8	864	1376	15.908
9	1504	864	15.234

Q6.2

I proposed a general algorithm that covers all the parts (Q6.2, Q6.4, and Q6.5). In Q6.2, and Q6.4, specifically, there is no sign of noise, whereas Q6.5 adds more complexity by considering noise existence. The difference between Q6.2, and Q6.4 is that in Q6.4, we have no information regarding the length of each hidden digit, nor do we have any info regarding the pauses between each two consecutive digits. I designed an algorithm to be compatible with all of these conditions. The methodology behind my algorithm is to first estimate the start and end boundaries of each digit, and then find the corresponding frequencies which will return the estimated digits based on the obtained table in Q6.1. Before jumping into the algorithm, I want to talk about how sequence values of any given signal affect the Fourier transform and if we can approximate the Fourier series of a signal by some portion of the original signal, where the peak frequencies[1] are preserved. I considered two scenarios, where the manipulated signal differs from the original signal. As for the first scenario, I padded the original signal with 1024 zeros, and as for the second scenario, I cut off the original signal by keeping the first 50 samples and padded the remaining signal length with zeros. You can find the plots of the Fourier series for these two cases along with the original case below:

[1] What I mean by peak frequencies are the major frequencies in a signal at which the biggest Fourier series happen (In this problem, each digit tends to have 2 peak frequencies).

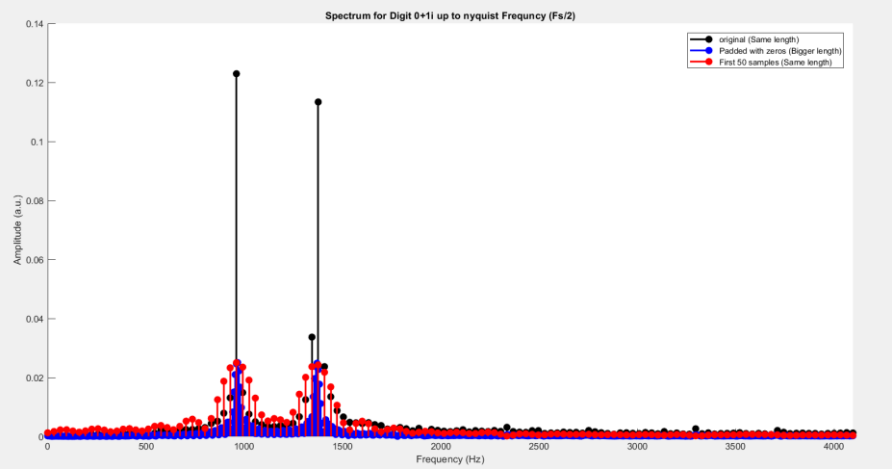

Fig 11. Extending using Padding vs cutting and padding (same length as original)

Based on the above figures, although all the information for the blue signal is preserved, padding and extending the signal length changes the frequency resolution[2], thereby changing the peak frequencies. In fact, the distribution of the Fourier series has not changed that much as the signal's energy is preserved; however, the peak frequencies of the manipulated signal and original signal are no longer matched. In the second scenario, where the length of the signal is preserved, we can see that although the distribution of the manipulated signal completely changed, the peak frequencies are still equal to the ones in the original signal. Via this comparison, we can conclude that keeping the first "n" samples of the signal and padding the rest with zeros will preserve the values of peak frequencies that are directly linked to the decoding of the hidden digits. The only question that remains is how to set the value of "n"? By intuition, we can say that n should be greater than the minimum consecutive samples in the original signal that contains the peak frequencies. After experimenting with some numbers, values bigger than 50 seem to work fine.

You may now ask what is the point of explaining all of this?

Well, that's what my approach relies on; my algorithm estimates the boundaries to the best possible (Two boundaries for each digit, a starting index and an ending index). We extract the hidden digits using start and end indexes and return the most accurate ones. As for the start index, each digit will be extracted using its corresponding start index and the following 256 samples afterwards. Likewise, as for the end index, each digit will be extracted using its corresponding end index and the previous 256 samples. Finally, we kind of evaluate these two approaches with respect to each other and return the most reliable one.

Now let's delve into the algorithm:

[2] Frequency resolution is the distance between each two consecutive samples in the frequency domain.

There exist 6 functions that were written explicitly for each part of the algorithm: EXP.m, EXP_INFO.m, Decide_BND.m, BND_Clean_Detection.m, BND_Contaminated_Detection.m, Decode_hidden_digits.m.

Besides that, there are two notebooks, Tone_detection1.m and Test.m that are setup using the aforementioned function.

Q6.3

$94\%5 = 4 \rightarrow Picking\ tpn4$

ans =

'Hidden digits in order are: (6,5,0,7)'

Q6.4

The algorithm is as same as what is used in Q6.2.

In the following, you can compare the accurate boundaries vs. estimated ones for two different signals (Manipulated versions of tpn#5)

'The accurate pair of boundries (start, end) for the clean signal is:

{(21,277),(533,789),(1045,1301),(1557,1813)}'

'The estimated pair of boundries (start, end) for the contaminated signal is:
{(20,279),(532,790),(1148,1301),(1556,1813)}'

'The accurate pair of boundries (start, end) for the clean signal is:

{(21,178),(478,565),(685,822),(892,999)}'

'The estimated pair of boundries (start, end) for the contaminated signal is:
{(19,178),(477,567),(684,822),(892,999)}'

Q6.5

Mistmatch_per_bits	SNR	digit4_encoded	digit3_encoded	digit2_encoded	digit1_encoded	mu	std
0	64.84078619	7	0	5	6	1E-04	0
0	24.4717122	7	0	5	6	1E-04	0.010416667
0	18.51911416	7	0	5	6	1E-04	0.020833333
0	14.90088208	7	0	5	6	1E-04	0.03125
1	12.33589941	7	0	0	6	1E-04	0.041666667
0	10.4510096	7	0	5	6	1E-04	0.052083333
1	8.953538911	0	0	5	6	1E-04	0.0625
0	7.705604587	7	0	5	6	1E-04	0.072916667
0	6.649581239	7	0	5	6	1E-04	0.083333333
0	5.428054256	7	0	5	6	1E-04	0.09375
1	4.665229722	0	0	5	6	1E-04	0.104166667
0	3.502337591	7	0	5	6	1E-04	0.114583333
0	3.018791299	7	0	5	6	1E-04	0.125
0	2.152934875	7	0	5	6	1E-04	0.135416667
2	1.899873443	7	7	0	6	1E-04	0.145833333
0	0.991585039	7	0	5	6	1E-04	0.15625
0	0.347805222	7	0	5	6	1E-04	0.166666667
2	-0.394265154	7	5	6	6	1E-04	0.177083333
2	-0.659794544	7	0	6	5	1E-04	0.1875
0	-1.140161846	7	0	5	6	1E-04	0.197916667
2	-1.533713909	7	-1	5	-1	1E-04	0.208333333
0	-1.818923254	7	0	5	6	1E-04	0.21875
1	-2.339892301	7	0	0	6	1E-04	0.229166667
1	-2.780742271	7	0	0	6	1E-04	0.239583333
2	-3.119358888	7	5	5	-1	1E-04	0.25

Appendix

```
function [digits, SNR] = Decode_hidden_digits(unknown_signal, Fs, mu, std, reference_table)


if std == 0
   [EST_BND_end,EST_BND_start] = BND_Clean_Detection(unknown_signal);
else
   EST_BND_end = BND_Contaminated_Detection(unknown_signal, mu, std, 1);
   BND_END_flip = BND_Contaminated_Detection(flip(unknown_signal,1), mu, std, 1);
   EST_BND_start = length(unknown_signal) - flip(BND_END_flip,2) + 2; %  + 2 for the flip effect
end


Hidden_signal_end = zeros(4,256);
Hidden_digits_end = zeros(1,4);
% one time -> loop over the end indices
idx_end = 1:4;
if EST_BND_end(1) < 256
   Hidden_signal_end(1,:) = [unknown_signal(1:EST_BND_end(1));zeros(256 - EST_BND_end(1),1)]';
   idx_end = 2:4;
end
```

```
for i = idx_end
    Hidden_signal_end(i,:) = unknown_signal(EST_BND_end(i) - 256: EST_BND_end(i) - 1)';
end

for i = 1:4 % find Hidden digits using the estimated end indices
    digit_signal = Hidden_signal_end(i,:);
    pnts = length(digit_signal);
    spectrum = abs(fft(digit_signal)/pnts);
    hz = linspace(0,Fs/2,floor(pnts/2)+1);

    % Detect the frequencies of the two dominant peaks
    [~, sorted_indices] = sort(spectrum(1:length(hz)), 'descend');
    first_peak = sorted_indices(1);
    second_peak = sorted_indices(2);

    % Convert peak indices to frequencies
    freq1_end = hz(first_peak);
    freq2_end = hz(second_peak);

    Filtered_table = reference_table((reference_table.freq1 == freq1_end & reference_table.freq2 == freq2_end) | ...
                                (reference_table.freq1 == freq2_end & reference_table.freq2 == freq1_end),:);
    if size(Filtered_table,1) ~= 0
        Hidden_digits_end(i) = Filtered_table.digit;
    else

        Hidden_digits_end(i) = -1; % -1 is a symbol of unkown digit that was not found during the decoding
procedure!

    end
end

Hidden_signal_start = zeros(4,256);
Hidden_digits_start = zeros(1,4);
idx_start = 1:4;
% one time -> loop over the end indices
if length(unknown_signal) - EST_BND_start(4) + 1 < 256
    Hidden_signal_start(4,:) = [zeros(255 + EST_BND_start(4) -
length(unknown_signal),1);unknown_signal(EST_BND_start(4):end)]';
    idx_start = 1:3;
end

for j = idx_start
    Hidden_signal_start(j,:) = unknown_signal(EST_BND_start(j):EST_BND_start(j) + 255)';
end

for j = 1:4 % find Hidden digits using the estimated end indices
    digit_signal = Hidden_signal_start(j,:);
    pnts = length(digit_signal);
    spectrum = abs(fft(digit_signal)/pnts);
    hz = linspace(0,Fs/2,floor(pnts/2)+1);

    % Detect the frequencies of the two dominant peaks
    [~, sorted_indices] = sort(spectrum(1:length(hz)), 'descend');
    first_peak = sorted_indices(1);
    second_peak = sorted_indices(2);
```

```
    % Convert peak indices to frequencies
    freq1_start = hz(first_peak);
    freq2_start = hz(second_peak);

    Filtered_table = reference_table((reference_table.freq1 == freq1_start & reference_table.freq2 == freq2_start)|...
                                (reference_table.freq1 == freq2_start & reference_table.freq2 == freq1_start),:);
    if size(Filtered_table,1) ~= 0
        Hidden_digits_start(j) = Filtered_table.digit;
    else
        Hidden_digits_start(j) = -1; % -1 is a symbol of unkown digit that was not found during the decoding
procedure!
    end

end

if length(Hidden_digits_start(Hidden_digits_start == -1)) < length(Hidden_digits_end(Hidden_digits_end == -1))
    digits = Hidden_digits_start;
else
    digits = Hidden_digits_end;
end

SNR = snr(unknown_signal, mu + std * randn(length(unknown_signal),1));

end
```

```
function [EST_BND] = BND_Contaminated_Detection(unknown_signal, mu, std, WE)

Input = [];
filename = 'Experiment_result.xlsx';

for i = 1:20
    Noise = mu + std * randn(length(unknown_signal),1);
    New = unknown_signal + Noise;
    Input = [Input;transpose(diff(New.^2))];
end


INFO = EXP_INFO(Input, std);
T = cell2table(INFO,...
'VariableNames',{'Exp_Number' 'Lower','Upper','Threshold','N_Boundries', 'Bounds'});
filteredTable = T(T.N_Boundries == 4, :);
sortedTable = sortrows(filteredTable, {'Lower','Upper','Threshold'});
if WE
    writetable(sortedTable,filename,'Sheet','4Bounds')
end

NEW_TB = sortedTable(:,{'Lower','Upper','Threshold','N_Boundries'});
[~,indexToUniqueRows,indexBackFromUnique] = unique(NEW_TB);
% Get unique values and their counts while preserving order
[uniqueValues, ~, idx] = unique(indexBackFromUnique, 'stable');
counts = accumarray(idx, 1);
% Create a table with unique values and their counts
uniqueTable = table(uniqueValues, counts, 'VariableNames', {'Value', 'Count'});
sortedUniqueTable = sortrows(uniqueTable, 'Count', 'descend');
% idx2 -> The true index of unique entries in sortedUniqueTable
```

```
[~, idx2, ~]= unique(sortedUniqueTable{:,{'Count'}},'stable');
if length(idx2) > 1
    MAT = zeros(length(idx2) - 1,4);
    for i = 1:length(idx2) - 1
        index_range_0 = idx2(i):idx2(i+1)-1;
        index_range_1 = sortedUniqueTable(index_range_0,:).Value;
        idx_range = indexToUniqueRows(index_range_1);
        BND = cell2mat(sortedTable(idx_range,:).Bounds);
        %MAT = [MAT;BND]
        [mostFrequentRow, ~] = Decide_BND(BND);
        MAT(i,:) = mostFrequentRow;
    end
else
    MAT = cell2mat(sortedTable(1,:).Bounds);
end
MAT
% It's time to estimate the best boundries!
EST_BND = zeros(1,4);
k_values = 1:size(MAT,1);
for m = 1:4
    data = MAT(:,m);
    % Define a range of potential values for k
    % Initialize variables to store clustering results and quality measures
    silhouette_values = zeros(numel(k_values), length(data));

    % Perform k-means clustering for different values of k
    for i = 1:numel(k_values)
        k = k_values(i);
        [cluster_indices, ~] = kmeans(data, k, 'Distance','sqeuclidean');

        % Calculate silhouette score
        silhouette_values(i,:) = silhouette(data, cluster_indices)';
    end

    % Find the best k based on the SSE or silhouette score
    if all(all(isnan(silhouette_values)))
        best_k_silhouette = 1;
    else
        best_k_silhouette = k_values(find(silhouette_values == max(silhouette_values), 1));
    end

    %fprintf('Best k based on Silhouette Score: %d\n', best_k_silhouette);
    [cluster_indices, centroids] = kmeans(data, best_k_silhouette);
    CLL = num2cell(zeros(best_k_silhouette,4));
    for i = 1:best_k_silhouette
        index_per_cluster = find(cluster_indices == i);
        CLL(i,1) = {i};
        CLL(i,2) = {length(index_per_cluster)};
        CLL(i,3) = {data(index_per_cluster)};
        CLL(i,4) = {round(centroids(i))};
    end
    TT = cell2table(CLL,'VariableNames',{'Clust_Num' 'L_Data' 'Data' 'Centroid'});
    TT_sorted = sortrows(TT,{'L_Data','Centroid'},'descend');
%     if size(TT_sorted(TT_sorted.L_Data == TT_sorted.L_Data(1),:),1) > 1
%         display(TT_sorted)
%     end
```

```
    EST_BND(m) = TT_sorted.Centroid(1);
end
end
```

```
 function [end_indices,start_indices] = BND_Clean_Detection(unknown_signal)
start_indices = zeros(1, 4);
end_indices = zeros(1,4);

% Initialize a variable to track the current "Vi"
current_Vi = 0;
current_Vj = 0;

if unknown_signal(1) ~= 0
    start_indices(1) = 1;
    current_Vi = 1;
end


for i = 2:length(unknown_signal)
    if unknown_signal(i) ~= 0 && unknown_signal(i - 1) == 0
        current_Vi = current_Vi + 1;
        start_indices(current_Vi) = i;
    elseif unknown_signal(i) == 0 && unknown_signal(i - 1) ~= 0
        current_Vj = current_Vj + 1;
        end_indices(current_Vj) = i;
    end
end
end
```

```
function [mostFrequentRow, mostFrequentRowIndex] = Decide_BND(MAT)
[m, n] = size(MAT); % m: number of rows, n: number of columns

mostFrequentRow = zeros(1, n); % Initialize the row with zeros
maxFrequency = zeros(1, n); % Initialize the max frequency counts

for col = 1:n
    [uniqueValues, ~, frequency] = unique(MAT(:, col));
    counts = accumarray(frequency, 1);

    [maxCount, maxIndex] = max(counts);

    mostFrequentRow(col) = uniqueValues(maxIndex);
    maxFrequency(col) = maxCount;
end

% Display the most frequent row and its index
mostFrequentRowIndex = find(ismember(MAT, mostFrequentRow, 'rows'));
end
```

```
function [INFO] = EXP_INFO(Input, std)
NUM_EXP = size(Input,1);
INFO = zeros(NUM_EXP * 1000,6);
%Columns= {'EXP NUM' 'Lower-band','Upper-band','Threshold', 'Num Boundries', 'Bounds'};
```

```
INFO(:,1) = transpose(repelem(linspace(1,NUM_EXP,NUM_EXP),1000));
INFO(:,2) = transpose(repmat(repelem(linspace(0, 1e-5, 10), 100),1,NUM_EXP));
INFO(:,3) = transpose(repmat(repelem(linspace(std * 2.5 * 1e-3 + 1.5 * 1e-5, std * 2.5 * 1e-2 + 1.5 * 1e-4, 10),
10),1,NUM_EXP * 10));
INFO(:,4) = transpose(repmat(linspace(64,100,10),1,NUM_EXP * 100));

INFO = num2cell(INFO);
for i = 1:NUM_EXP * 1000
   O = EXP(cell2mat(INFO(i,2)),cell2mat(INFO(i,3)),cell2mat(INFO(i,4)), Input(cell2mat(INFO(i,1)),:));
   INFO(i,6) = {O};
   INFO(i,5) = {length(O)};
end
end
```

```
function [Boundry] = EXP(Lower, Higher, eta, input)
index = [];
for i = 1:length(input)
   if (Lower <= input(i)) && (input(i) <= Higher)
      index = [index, i];
   end
end

DN = diff(index);
cls = find(DN>eta);
Boundry = [];


if index(1) > eta
   Boundry = index(1);
end

for i = cls
   Boundry = [Boundry, index(i+1)];
   %sprintf('Selected values in IN are %d',IN(i+1))
   %sprintf('Selected values in DN are %d',DN(i))
end
end
```