Telephony Documentation I

Technical University of Cluj-Napoca
Name: Radu Anca-Valentina
Group: e_2331_1

PART I: Dial Tone, Busy Tone, Ring Tone, Ringback Tone

Theoretical Description and Understanding of the Tones

1) Dial Tone

The dial tone is a continuous signal used for access signaling (informs the local exchange about the status of the telephone terminal) within the subscriber loop, having a frequency between 400-425Hz; in this case, the used frequency is of 420Hz.

2) Busy Tone

The busy tone is a discountinuous signal used for access signaling within the subscriber loop, having a frequency between 400-425Hz; in this case, the used frequency is of 400Hz. Its timing is: 0.5 ON (state), 0.5 OFF (state).

3) Ring Tone

The ring tone is, generally, a sinusoidal or an amplitude modulated sinusoidal signal used for alert signaling (notifies the telephone terminal about the reception of a call). Having a frequency between 20-40Hz and the amplitude between 80-150V, it may cause interference (noise pulses) in the neighboring pairs. Its timing depends on the local exchange and is usually: 2s ON, 2s OFF.

4) Ringback Tone

The ringback tone is an intermittent sinusoidal signal or a modulated sinusoidal signal, different from the busy tone. It is generated by the local exchange which the called subscriber is connected to. Its timing is, generally, identical to the one of the ring tone.

Implementation of the tones in Matlab

- In order to facilitate the starting and stopping of the tones, I used the player object. The 'Start' and 'Stop' buttons have callbacks to their corresponding functions later explained.

```
function Script1()
    persistent player
    % An object-oriented way to play sound in MATLAB, which you create from
    % a signal and sampling rate, and then control it
    % like a media player.
    % We use persistent player to store and reuse the audioplayer object
    % across callbacks (for stop/start control)
```

- <u>Implementation of the GUI</u>; this creates a bridge between the user and the code. Contains:
    - field for the sampling frequency input
    - field for displaying the signal time interval
    - dropdown for selecting the tone
    - Start button
    - Stop button.

```
    % Create the GUI window
    fig = figure('Name', 'Telephony Tones', ...
           'Position', [500 300 340 280], ...
           'MenuBar', 'none', ...
           'NumberTitle', 'off', ...
           'Resize', 'off');

    % Sampling frequency label
    uicontrol('Style', 'text', ...
              'Position', [50 220 200 20], ...
              'String', 'Sampling Frequency (Hz):', ...
              'FontSize', 10, ...
              'HorizontalAlignment', 'left');

    % Sampling frequency input
    fsInput = uicontrol('Style', 'edit', ...
                        'Position', [200 220 70 25], ...
                        'String', '8000', ...
                        'FontSize', 10);

    % Signal time interval label
    uicontrol('Style', 'text', ...
              'Position', [50 190 200 20], ...
              'String', 'Signal Time Interval:', ...
              'FontSize', 10, ...
              'HorizontalAlignment', 'left');
```

```matlab
% Signal time interval display
TsignalDisplay = uicontrol('Style', 'text', ...
                    'Position', [200 190 70 25], ...
                    'String', ' ', ...
                    'FontSize', 10, ...
                    'BackgroundColor', [1 1 1], ...
                    'HorizontalAlignment', 'left');

% Select tone label
uicontrol('Style', 'text', ...
          'Position', [50 160 200 20], ...
          'String', 'Select a Tone:', ...
          'FontSize', 10);

% Dropdown menu
toneMenu = uicontrol('Style', 'popupmenu', ...
          'Position', [50 130 220 25], ...
          'String', {'Choose Tone','Dial Tone', 'Busy Tone', ...
          'Ring Tone', 'Ringback Tone'}, ...
          'FontSize', 10);


% Start button
uicontrol('Style', 'pushbutton', ...
          'Position', [50 80 90 30], ...
          'String', 'Start', ...
          'FontSize', 10, ...
          'Callback', @startTone);

% Stop button
uicontrol('Style', 'pushbutton', ...
          'Position', [160 80 90 30], ...
          'String', 'Stop', ...
          'FontSize', 10, ...
          'Callback', @stopTone);
```

- <u>Implementation of the stop tone function (stopTone)</u> - it is aimed to stop the sound of each signal whenever after it starts, before it ends. This uses the player object I explained earlier.

```matlab
% Stop tone function
function stopTone(~, ~)
    try
        stop(player);
    end
end
```

- Implementation of the start tone function (startTone)

  Firstly, it calls the function stopTone in order to stop any signal that might be playing when it is called.

  Next, it uses an array to know which tone was selected.

  After that, it gets the sampling frequency from the user, otherwise having the default sampling frequency of 8000Hz, which is also chosen if the user's input is invalid.

```
    % Start tone callback function
    function startTone(~, ~)
        stopTone(); % stop any current tone

        val = toneMenu.Value; % the index (number) of the selected item
        tone = toneMenu.String; % the full list of options (a cell array of
strings)
        selectedTone = tone{val};

        % Get sampling frequency from input
        fsamp = str2double(fsInput.String);  % fsInput field returns a string,
str2double converts to double
        if isnan(fsamp) || fsamp <= 0 %if NaN or <=0, it is invalid
            fsamp = 8000; % default if invalid
        end
```

  Afterwards, a switch-case is implemented in order to generate the selected signal. For each case, the frequency, periods, steps, segments (where needed) and the signal are provided. Then in each case, the player object is created using the signal and the introduced sampling frequency and is sent to the audiocard using the play function.

  Moreover, the **TsignalDisplay** component (which displays the signal time interval) is used here to display the signal time interval, by computing the ratio between the length of the signal and the sampling frequency, displaying it with 3 decimals.

  For the Ring Tone, the signal is taken from a provided audio file, 'telephone-ring-02.wav'.

  Finally, in all cases, the signal in time and frequency domains plots are created and the signal is played.

```matlab
switch selectedTone
    case 'Dial Tone'
        % Values
        A = 1.2;
        f = 420;
        Tmax = 3;
        t = 1:1/fsamp:Tmax;
        s = A * sin(2 * pi * f * t)';  % create signal

        % Play tone
        player = audioplayer(s, fsamp);
        play(player);

        % Calculate and display signal duration
        TsignalDisplay.String = num2str(length(s)/fsamp, '%.3f');

        % Plot signal in time and frequency domains
        [S, freqv] = analiz_sp(s, fsamp);
        figure,
        subplot(2,1,1)
        plot(t, s)
        title('Dial Tone - Time Domain')
        xlabel('Time (s)')
        ylabel('Amplitude')

        subplot(2,1,2)
        plot(freqv, S)
        title('Dial Tone - Frequency Spectrum')
        xlabel('Frequency (Hz)')
        ylabel('Amplitude')
        grid on

    case 'Busy Tone'
        % Values
        A = 1;
        f = 400;
        t_ON = 0.5;
        t_OFF = 0.5;
        N_rep = 6;

        % ON and OFF segments
        time_ON = 0 : 1/fsamp : t_ON - 1/fsamp;
        time_OFF = 0 : 1/fsamp : t_OFF - 1/fsamp;
        signal_ON = A * sin(2 * pi * f * time_ON);
        signal_OFF = zeros(1, length(time_OFF));

        % Combine and repeat pattern
        signal = [signal_ON signal_OFF];
        signal_rep = repmat(signal, 1, N_rep)';

        % Play tone
        player = audioplayer(signal_rep, fsamp);
        play(player);

        % Display duration
        TsignalDisplay.String = num2str(length(signal_rep)/fsamp, '%.3f');
```

```matlab
            % Plot signal in time and frequency domains
            time_full = (0:length(signal_rep)-1) / fsamp;
            [S, F] = analiz_sp(signal_rep, fsamp);
            figure,
            subplot(2,1,1)
            plot(time_full, signal_rep)
            title('Busy Tone - Time Domain')
            xlabel('Time (s)')
            ylabel('Amplitude')

            subplot(2,1,2)
            plot(F, S)
            title('Busy Tone - Frequency Spectrum')
            xlabel('Frequency (Hz)')
            ylabel('Amplitude')

        case 'Ring Tone'
            % Load stereo ringtone
            [y, fsamp] = audioread('telephone-ring-02.wav');

            % ON/OFF durations
            t_ON = 1;
            t_OFF = 4;
            N_rep = 5;

            % ON and OFF segments
            n_ON = round(t_ON * fsamp);
            n_OFF = round(t_OFF * fsamp);
            signal_ON = y(720 : 720 + n_ON - 1, :);  % stereo audio
            signal_OFF = zeros(n_OFF, 2);           % stereo silence

            % Combine and repeat signal
            signal = [signal_ON; signal_OFF];
            signal_rep = repmat(signal, N_rep, 1);

            % Play tone
            player = audioplayer(signal_rep, fsamp);
            play(player);

            % Display duration
            TsignalDisplay.String = num2str(size(signal_rep,1)/fsamp, '%.3f');
            % Plot signal in time and frequency domains
            figure,
            subplot(2,1,1)
            plot(signal_rep)
            title('Ring Tone - Time Domain (Stereo)')
            xlabel('Sample')
            ylabel('Amplitude')
            legend('CH1','CH2')


            [S1,F1] = analiz_sp(signal_rep(:,1), fsamp); %stereo channel 1
            [S2,F2] = analiz_sp(signal_rep(:,2), fsamp); %stereo channel 2
            subplot(2,1,2)
            plot(F1, S1), hold on
            plot(F2, S2)
```

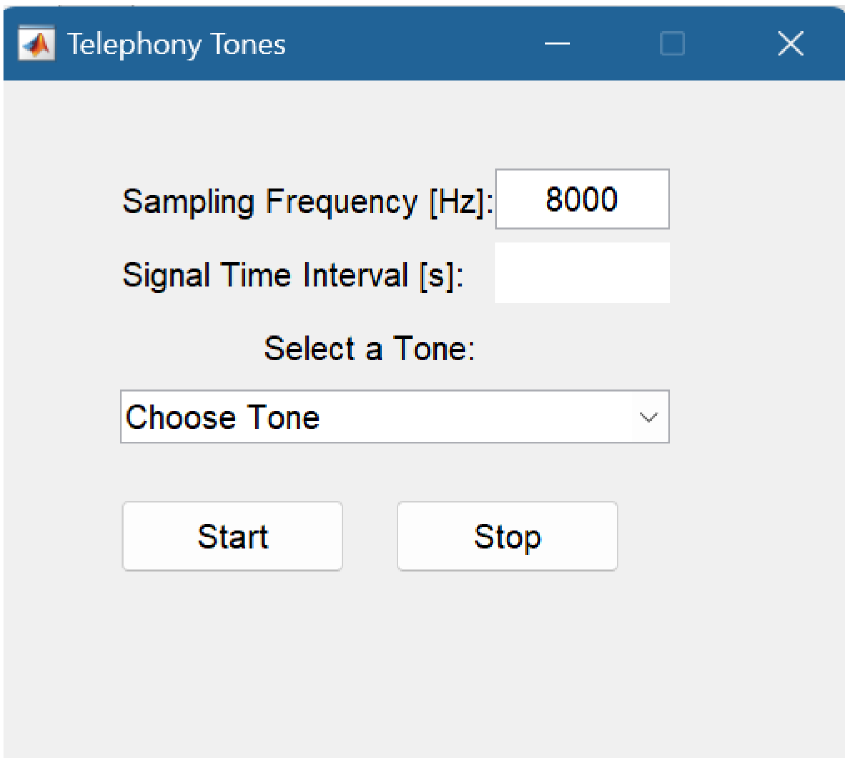
```
                title('Ring Tone - Frequency Spectrum')
                xlabel('Frequency (Hz)')
                ylabel('Amplitude')
                legend('CH1','CH2')

            case 'Ringback Tone'
                % Values
                A = 1;
                f = 400;
                t_ON = 1;
                t_OFF = 4;
                N_rep = 6;

                % ON and OFF segments
                time_ON = 0 : 1/fsamp : t_ON - 1/fsamp;
                time_OFF = 0 : 1/fsamp : t_OFF - 1/fsamp;
                signal_ON = A * sin(2 * pi * f * time_ON);
                signal_OFF = zeros(1, length(time_OFF));

                % Combine and repeat pattern
                signal = [signal_ON signal_OFF];
                signal_rep = repmat(signal, 1, N_rep)';

                % Play tone
                player = audioplayer(signal_rep, fsamp);
                play(player);

                % Display signal duration
                TsignalDisplay.String = num2str(length(signal_rep)/fsamp, '%.3f');

                % Plot signal in time and frequency domains
                time_full = (0:length(signal_rep)-1) / fsamp;
                [S, F] = analiz_sp(signal_rep, fsamp);
                figure,
                subplot(2,1,1)
                plot(time_full, signal_rep)
                title('Ringback Tone - Time Domain')
                xlabel('Time (s)')
                ylabel('Amplitude')

                subplot(2,1,2)
                plot(F, S)
                title('Ringback Tone - Frequency Spectrum')
                xlabel('Frequency (Hz)')
                ylabel('Amplitude')
        end
    end
end
```

The code used for the spectral analysis (for the analiz_sp function):

```
function [P1,f] = analiz_sp(s,fe)

Y = fft(s);
L=length(s);

P2=abs(Y/L);
P1=P2(1:floor(L/2)+1);
P1(2:end-1)=2*P1(2:end-1);
f=fe/L*(0:L/2);
s=P1;

end
```

Description and Interpretation of Results

The GUI containing the aforementioned components:



Telephony Tones

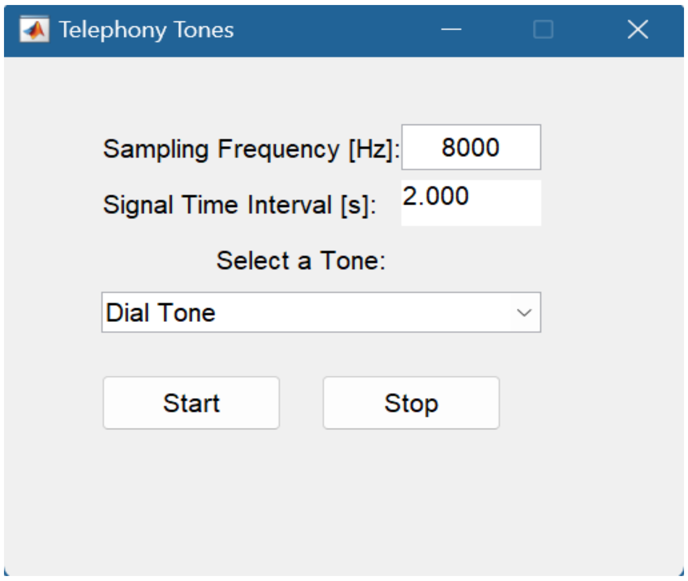
Sampling Frequency [Hz]: 8000

Signal Time Interval [s]:

Select a Tone:

- We can select the sampling frequency, or use the 8000Hz default one, in the 'Sampling Frequency [Hz]' field.
- After generating the signal (pressing the Start button), the duration of the signal, in seconds, will appear in the 'Signal Time Interval [s]' field.
- To select the Tone, we use the 'Choose Tone' dropdown menu.
- To generate and play the signal, use 'Start' button.
- To stop the sound of the tone, use 'Stop' button.

1) Dial Tone


Telephony Tones

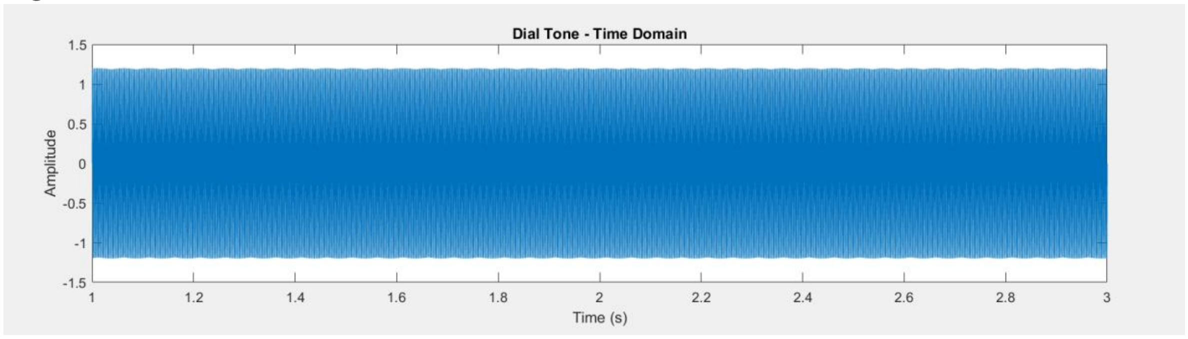
Sampling Frequency [Hz]: 8000

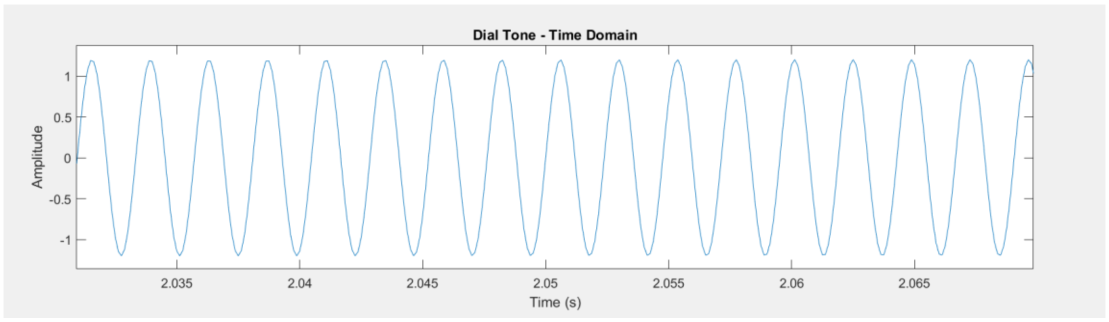
Signal Time Interval [s]: 2.000

Select a Tone:

Dial Tone

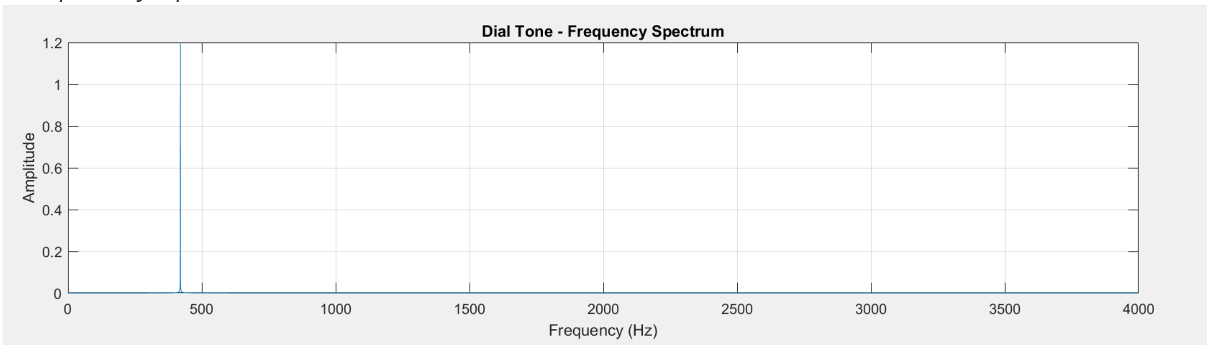
Signal in time domain:

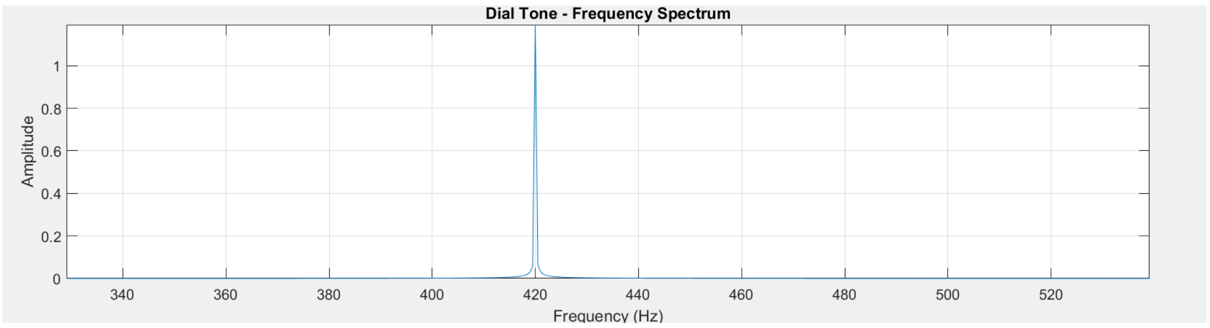




The signal time interval is computed correctly, lasting 2s. The dial tone in the time domain appears as a clean, continuous sinusoidal wave. It is noticeable that the signal accurately fits the description in the theoretical part.

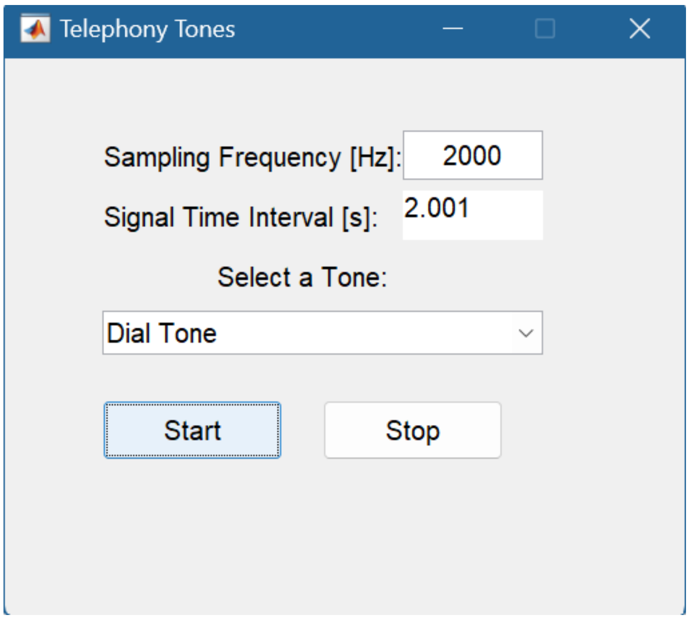
Frequency spectrum:





Here, the frequency spectrum of the dial tone shows a single, sharp peak centered at 420Hz, confirming that the signal consists of one sinusoidal frequency component with no harmonics or additional frequencies.

For a test frequency of 2000Hz **(in order to show functionality of the sampling frequency selection)**, these are the results:

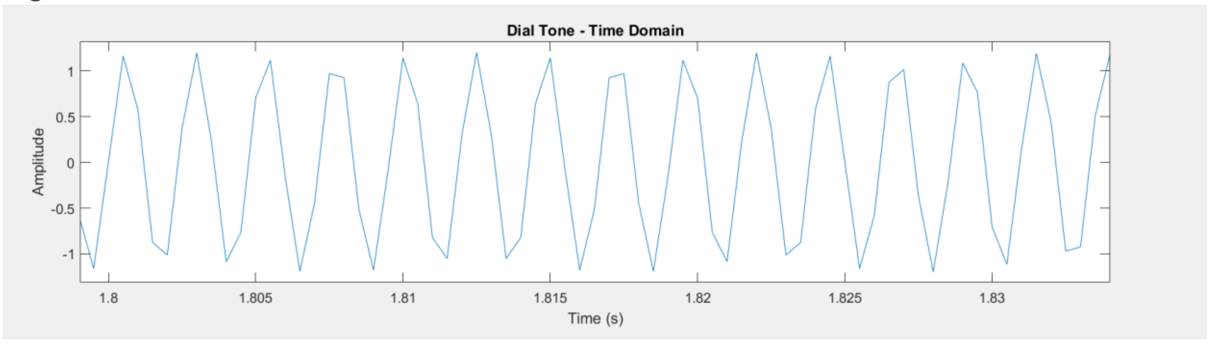

Telephony Tones

Sampling Frequency [Hz]: 2000

Signal Time Interval [s]: 2.001

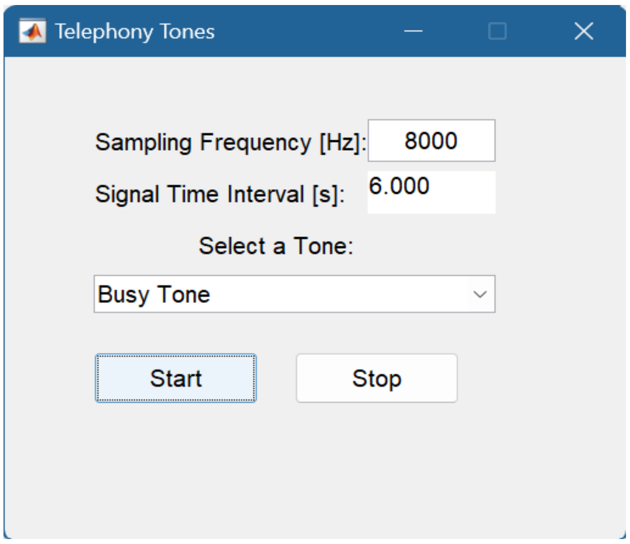
Select a Tone: Dial Tone

Signal in time domain:



The signal is visibly more distorted as the sampling frequency decreases.

2) Busy Tone


Telephony Tones

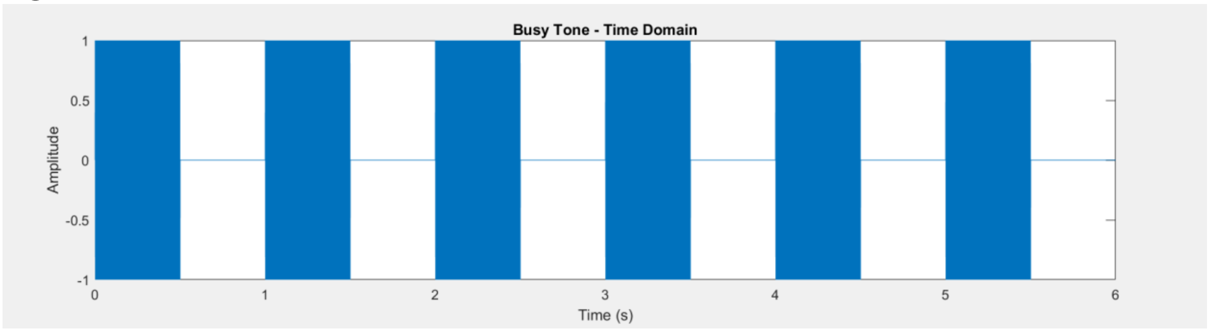
Sampling Frequency [Hz]: 8000

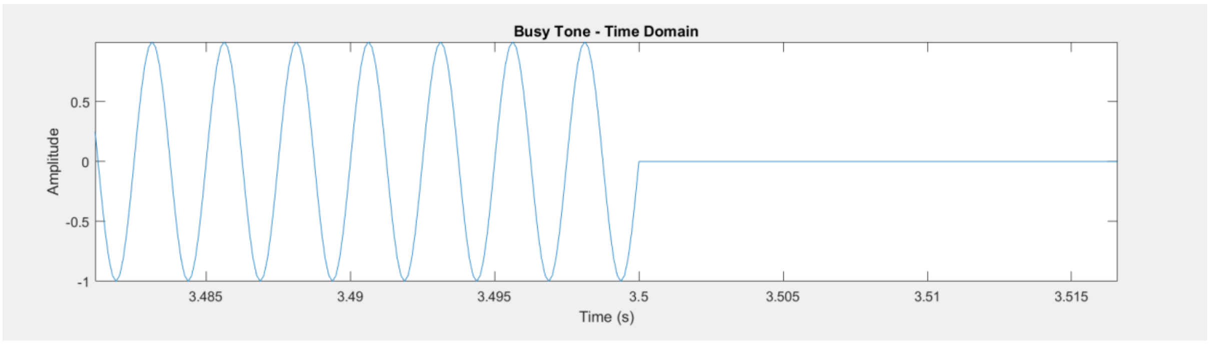
Signal Time Interval [s]: 6.000

Select a Tone:

Busy Tone

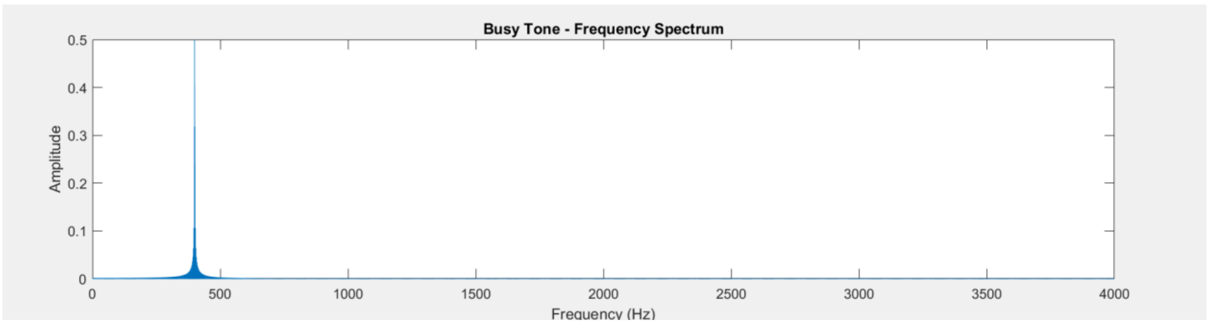
Signal in time domain

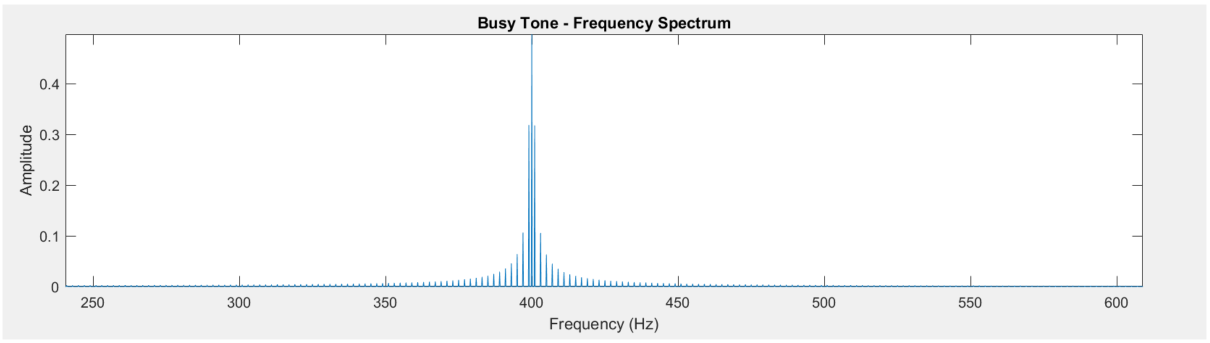




The signal time interval is computed correctly, lasting 6s. It is noticeable that the discontinuous signal accurately fits the description in the theoretical part. The 0.5 ON – 0.5 OFF are also clearly visible, due to the repetitive sequence of short bursts of sinusoidal tone followed by equal-length silent intervals, repeating every second.

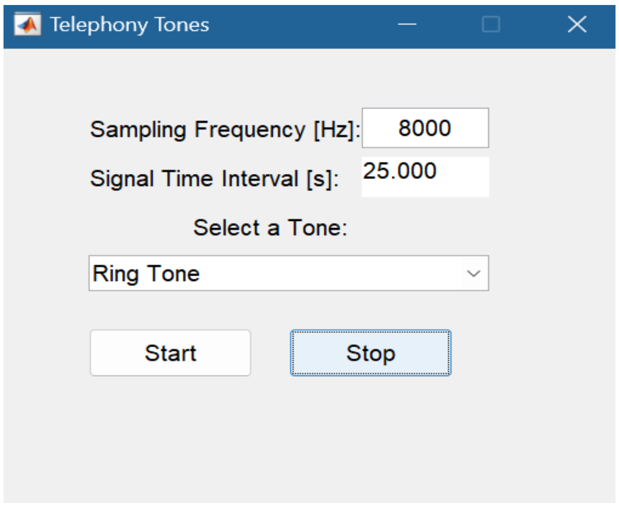
Frequency spectrum:





Here, it is visible that frequency spectrum shows a peak centered at 400 Hz, which corresponds to the frequency of the sinusoidal tone used in the signal. However, it is not a pure tone, as the spectrum also shows multiple smaller sidebands around the main peak, caused by the tone being interrupted by silence, this introducing discontinuities in the signal.

3) Ring Tone



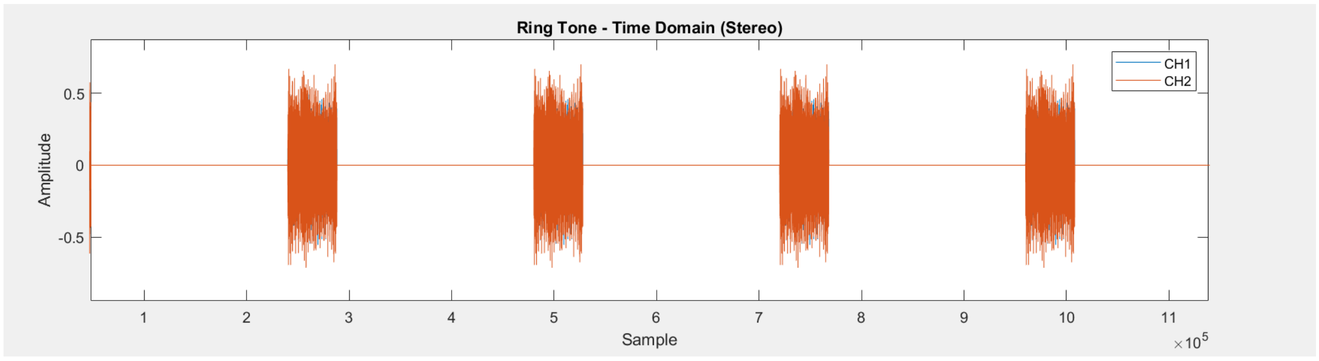
Sampling Frequency [Hz]: 8000

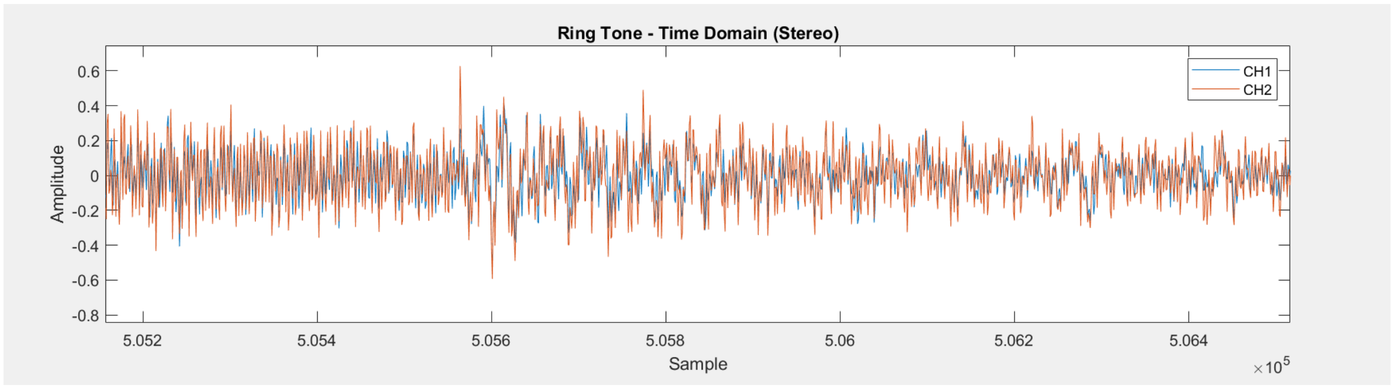
Signal Time Interval [s]: 25.000

Select a Tone:

Ring Tone

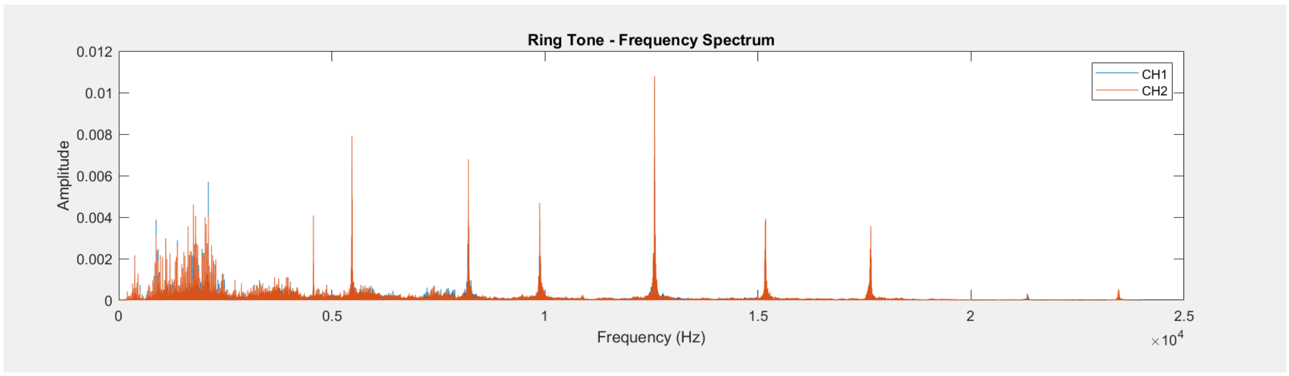
Signal in time domain:

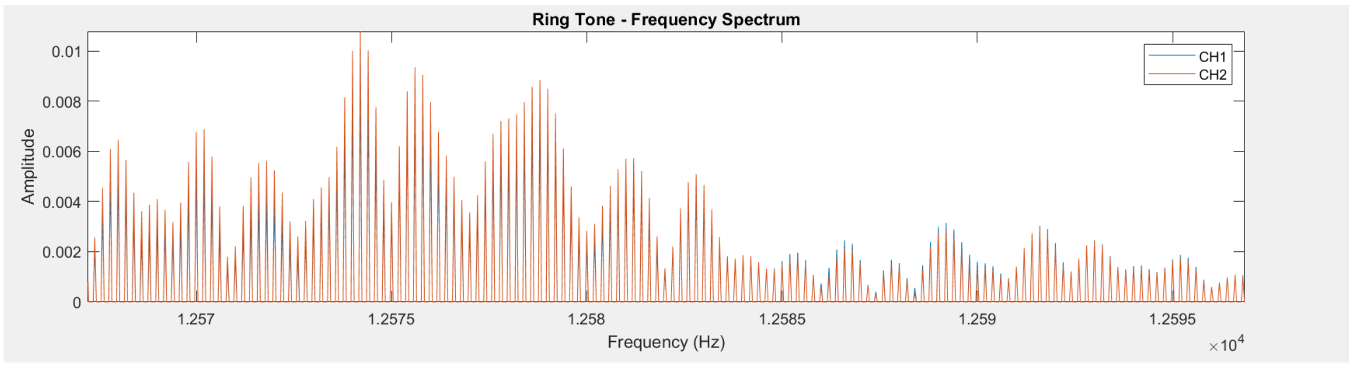




The signal time interval is computed correctly, lasting 25s. It is noticeable that the signal is stereo, having 2 components, one signal channel for each ear,.

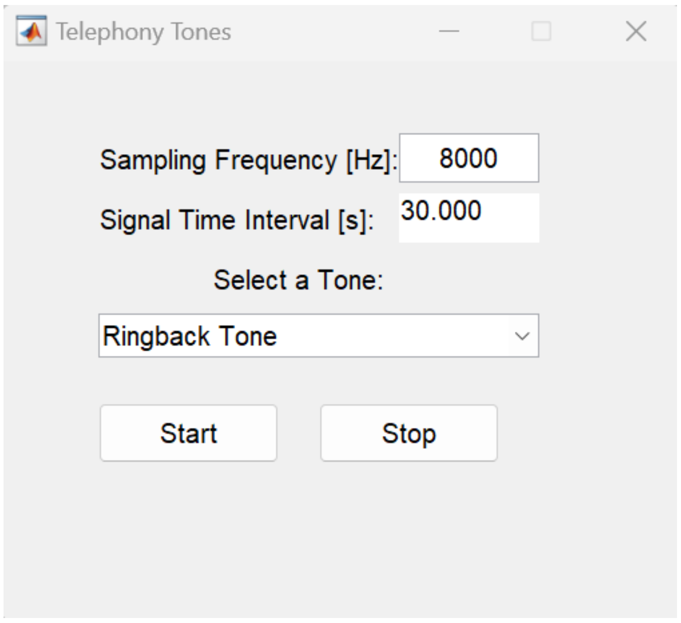
Frequency spectrum:





There are multiple spectral components visible in the spectrum, indicating a harmonic structure, probably due to each signal being composed by more sinewaves.

4) Ringback Tone


Telephony Tones

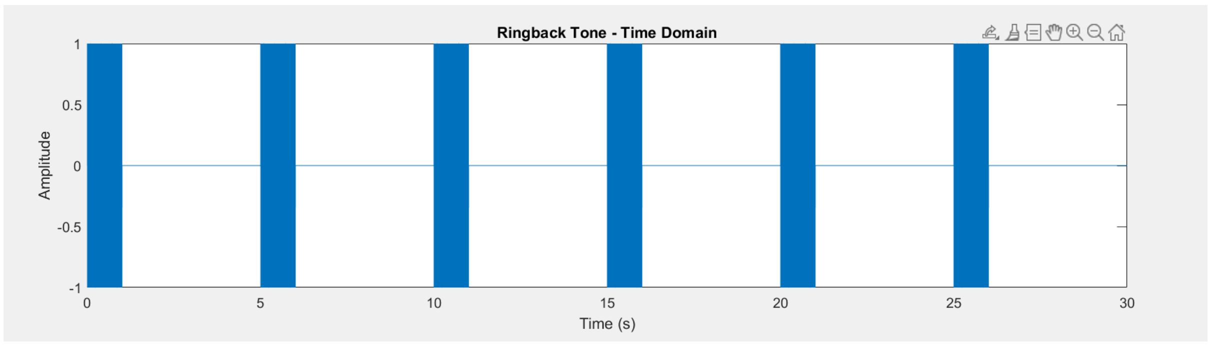
Sampling Frequency [Hz]: 8000

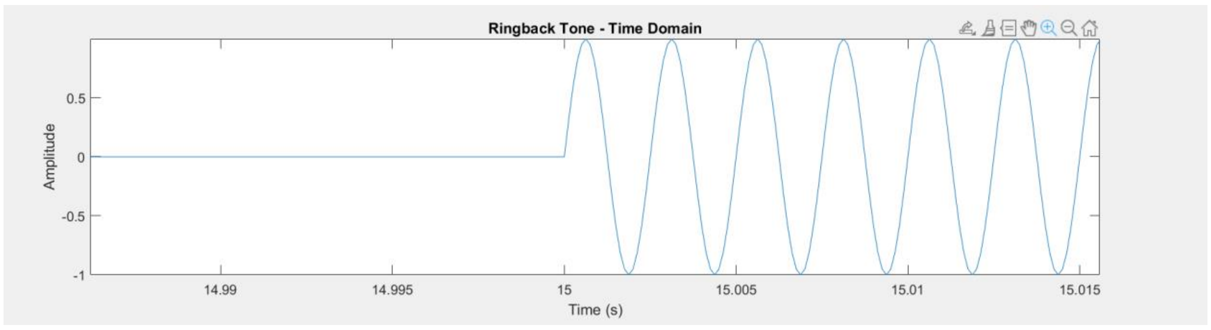
Signal Time Interval [s]: 30.000

Select a Tone:

Ringback Tone

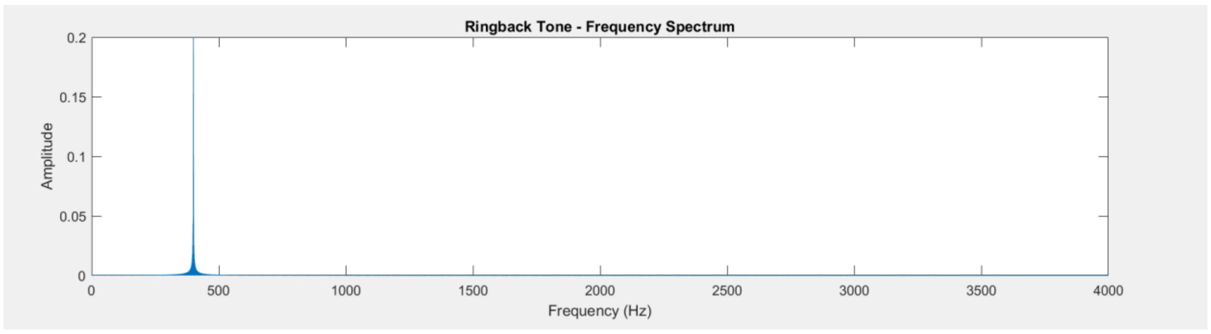
Signal in time domain:

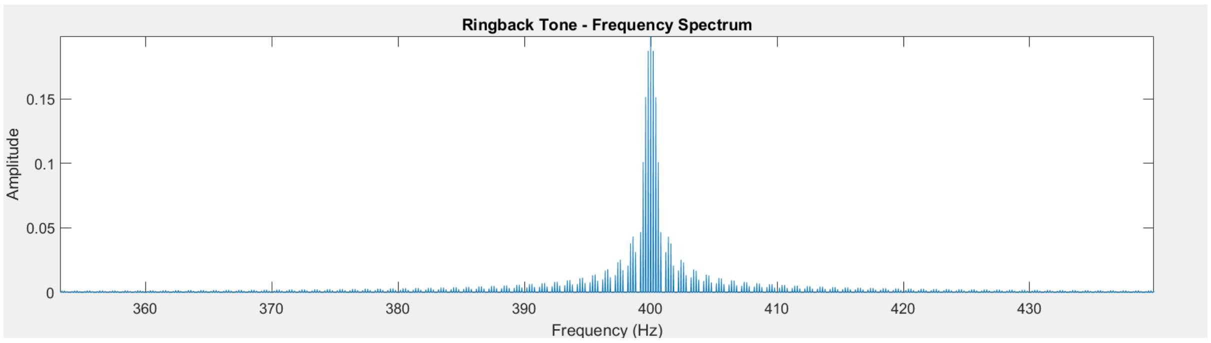




Here, we can notice the similarities of the Ringback Tone to the Busy Tone, but here the clear sinewave lasts 1s, due to the1s ON – 4s OFF timing, instead of the ½ ratio in the Busy Tone.

Frequency spectrum:





The peak frequency of 400Hz of the pure sinewave is visible in the spectrum. However, the spectrum has some distortion artifacts caused by the ring signal.

PART II: Address Signaling (Pulse and DTMF Signaling)

Theoretical Description and Understanding of the Signaling Techniques: Pulse Signaling, DTMF Signaling

1) Address Signaling

Address signaling refers to the process of encoding and transmitting the called subscriber's number to the local exchange in a specific format.

In analog access systems, each digit or symbol of the telephone number (or a terminal, equipment, or service identifier) can be encoded either as a sequence of pulses (Pulse Dialing method) or as a combination of tones (DTMF – Dual-Tone Multi-Frequency method).

2) Pulse Signaling

In pulse signaling, each digit of the dialed number is represented by a series of electrical pulses. These pulses are produced by momentarily interrupting the subscriber loop.

The number of pulses corresponds to the digit's value, with the exception of the digit 0, which is represented by 10 pulses.

3) DTMF Signaling

In Dual-Tone Multi-Frequency (DTMF) signaling, each digit or symbol is represented by a pair of tones (frequencies), one from a low and one from a high-frequency group, lasting about 100 ms.

Unlike pulse signaling, it also includes symbols like * and #. The frequencies, ranging from 700 Hz to 1700 Hz, are chosen to reduce distortion during voice transmission and they are situated in the lower portion of the voice frequency band, to reduce the risk of signal distortion during transmission.

Table 1.1: DTMF tones frequencies.

$F_{sup}[Hz]$ / $F_{inf}[Hz]$	1209	1339	1477	1633
697	1	2	3	A
770	4	5	6	B
852	7	8	9	C
941	$*$	0	#	D

Implementation of the address signaling methods in Matlab

- Implementation of the GUI: this creates, again, a bridge between the user and the code. Contains:
    - a dropdown menu for choosing the signaling method and its corresponding label "Choose Signaling Type"
    - a field for entering the phone number and its corresponding label "Enter Phone Number"
    - a checkbox for adding noise and its corresponding label "Add Noise"
    - a dropdown menu for choosing a filter (optional) "Select Filter"
    - a button for generating the signal "Generate Signal"

```
function Script2new
    % GUI figure
    fig = uifigure('Name', 'Telephony Signaling', 'Position', [450 300 400 260]);

    % Signaling type label
    uilabel(fig, ...
        'Position', [50 230 300 22], ...
        'Text', 'Choose Signaling Type:');

    % Signaling type dropdown
    signaling = uidropdown(fig, ...
        'Position', [50 200 300 22], ...
        'Items', {'Pulse Signaling', 'DTMF Signaling'}, ...
        'Value', 'DTMF Signaling');

    % Phone number field label
    uilabel(fig, ...
        'Position', [50 160 300 22], ...
        'Text', 'Enter Phone Number:');
```

```matlab
    % Phone number text field (input)
    phoneInput = uieditfield(fig, ...
        'text', ...
        'Position', [50 130 300 22]);

    % Noise checkbox
    noiseCheck = uicheckbox(fig, ...
        'Position', [50 100 300 22], ...
        'Text', 'Add Noise', ...
        'Value', false);  % Default state is no noise

        % Filter selection label
    uilabel(fig, ...
        'Position', [50 80 300 22], ...
        'Text', 'Select Filter:');

    % Filter selection dropdown
    filters = uidropdown(fig, ...
        'Position', [50 50 300 22], ...
        'Items', {'No Filter', 'Lowpass', 'Highpass', 'Bandpass', 'Chebyshev I', 
'Chebyshev II', 'Butterworth'}, ...
        'Value', 'No Filter');


    % Generate Signal button
    generateBtn = uibutton(fig, ...
        'Text', 'Generate Signal', ...
        'Position', [50 10 300 30], ...
        'ButtonPushedFcn', @(btn,event) generateSignal(signaling.Value, 
phoneInput.Value, noiseCheck.Value, filters.Value));
```

- Implementation of the function for signal generation (generateSignal)

Here, the 'if' loop checks if there was actually anything introduced in the "Enter Phone Number" field before pressing the button and gives out an error if it was empty.

The switch-case verifies the chosen address signaling method and displays a text in the Command Window letting us know:

- Which type of signal it is generating
- The introduced phone number

It also calls the later implemented functions for each signaling method.

```
% Callback function
function generateSignal(selection, phoneNumber, addNoise, selectedFilter)

    %Check if any number introduced
    if isempty(phoneNumber)
        uialert(fig, 'Please enter a phone number first.', 'Missing Input');
        return;
    end

    %Choose signaling method based on dropdown
    switch selection
        case 'Pulse Signaling'
            disp(['Generating Pulse Signal for number: ' phoneNumber]);
            generatePulseSignal(phoneNumber, addNoise, selectedFilter);
        case 'DTMF Signaling'
            disp(['Generating DTMF Signal for number: ' phoneNumber]);
            generateDTMFSignal(phoneNumber, addNoise, selectedFilter);
    end
end
```

- <u>Implementation of the function for the DTMF signal generation (generateDTMFSignal)</u>

The phoneInput.Value in the callback of the generateBtn button is passed to the function as phoneNumber.

Firstly, the signal parameters are defined:

- sampling frequency
- amplitudes (A1 scales the lower tone, A2 scales the higher tone)
- low&high frequency groups as arrays
- duration of the tone&pause of the signal (out of which we create the signal pause and the time vector for a tone)
- the DTMF frequency map for digits and the '*', '#' symbols (symbols only supported in DTMF)
- signal_all (the final output) which is a concatenation of all DTMF tones and pauses, defined as an empty array

```
% Generate the DTMF signal
function generateDTMFSignal(phoneNumber, addNoise, selectedFilter)

    %Signal parameters
    fe = 8e3;  % define the sampling frequency

    %Amplitudes for the 2 sinewaves
    A1 = 0.2;
    A2 = 0.3;

    fi = [697 779 852 941]; %low frequency group
    fs = [1209 1336 1477]; %high frequency group

    t_tone = 0.09; %tone time period
    t_pause = 0.01; %pause time period
    signal_pause = zeros(1, floor(t_pause * fe));
    t = 0 : 1/fe : t_tone - 1/fe;  % Time vector for a single tone

    %DTMF frequency map for digits and symbols
    dtmf_map = containers.Map( ...
        {'1','2','3', ...
         '4','5','6', ...
         '7','8','9', ...
         '0','*','#'}, ...
        {[1 1],[1 2],[1 3], ...
         [2 1],[2 2],[2 3], ...
         [3 1],[3 2],[3 3], ...
         [4 2],[4 1],[4 3]} ...
    );

    signal_all = []; %whole signal vector
```

Next, using a ‘for’ loop defined from 1 to the length of the phone number, we save each digit in the **digit** variable and check if it belongs to the DTMF map, resulting in an error if it does not. Only digits 0-9 and the symbols ‘*’, ‘#’ belong to the DTMF map.

We use the **idx** parameter to save the row-column pair that corresponds to each introduced digit using the DTMF frequency map, each row corresponding to a frequency in the low group and each column to one in the high group.

Consequently, we save the 2 frequencies for each digit in its corresponding variable (**f_inf** and **f_sup**) and we use them to generate the **signal.**

After creating the signal for a digit, we add a pause to it (**signal_pause**) and then we place it in the **signal_all** array created earlier.

The time vector for the whole signal, **t_all**, is created.

```
    %Generate DTMF signal for each digit
    for k = 1:length(phoneNumber)
        digit = phoneNumber(k); %save digit from phone number

        %Skip invalid characters not found in DTMF map
        if ~isKey(dtmf_map, digit)
            msgbox("DTMF signaling only accepts digits 0-9 and * #
symbols.",'Error','error')
            uialert(gcf, 'Pulse signaling only supports digits 0-9.', 'Invalid
Input');
            continue;
        end


        %Get corresponding low and high frequencies from the map
        idx = dtmf_map(digit); %row-column pair for each digit
        f_inf = fi(idx(1)); %low frequency of digit
        f_sup = fs(idx(2)); %high frequency of digit

        %Generate the tone by summing two sine waves
        signal = A1 * sin(2 * pi * f_inf * t) + A2 * sin(2 * pi * f_sup * t);

        %Add pause after the tone
        signal = [signal signal_pause];

        %Concatenate to the full signal
        signal_all = [signal_all signal];
    end



    % Time vector for entire signal
    t_all = 0 : 1/fe : (length(signal_all) - 1) / fe;  % Time vector for entire
signal
```

Subsequently, the ‘if’ condition “addNoise” for the checkbox gives us the option to add noise to the DTMF signal. If it is selected, the signal power and the SNR will also be computed and displayed in the Command Window.

Consequently:

- the Gaussian noise is generated using the wgn() function
- signal power calculated, converted in dbW anddisplayed in Command Window
- SNR calculated, converted in db and displayed in Command Window
- noise added to initial signal vector (**signal_all**)

```
    % Add noise if checkbox selected
    if addNoise
        % Signal Power
        power_signal_measured = rms(signal_all)^2; %compute signal power
        disp(['Signal Power: ', num2str(10*log10(power_signal_measured)), '
dbW']); %convert to dbW and display

        % Add Noise
        power_noise = 0;
        noise = wgn(1, length(signal_all), power_noise);  % gaussian noise with 0
dBW power

        % Display SNR
        disp(['SNR: ', num2str(10*log10(power_signal_measured) - power_noise), '
dB']); %convert to db and display

        % Concatenate signal with noise
        signal_all = signal_all + noise;
    end
```

Afterwards, a switch-case for each type of filter is implemented by using already implemented code for each. The filters are:

- Lowpass
- Highpass
- Bandpass
- Chebyshev I
- Chebyshev II
- Butterworth

```
% Apply selected filter
switch selectedFilter
    case 'Lowpass'
        fc = 1e3 / (fe/2);
        [signal_all, obj_lp] = lowpass(signal_all, fc, 'Steepness', 0.9);
        fvtool(obj_lp);
    case 'Highpass'
        fc = 1e3 / (fe/2);
        [signal_all, obj_hp] = highpass(signal_all, fc, 'Steepness', 0.8);
        fvtool(obj_hp);
    case 'Bandpass'
        fc = [530/(fe/2), 1530/(fe/2)];
        [signal_all, obj_bp] = bandpass(signal_all, fc, 'Steepness', 0.99);
        fvtool(obj_bp);
    case 'Chebyshev I'
        fc = 1e3 / (fe/2);
        [b, a] = cheby1(7, 1, fc, 'high');
        signal_all = filter(b, a, signal_all);

    case 'Chebyshev II'
        fc = [530/(fe/2), 1530/(fe/2)];
        [b, a] = cheby2(7, 1, fc, 'bandpass');
        signal_all = filter(b, a, signal_all);
    case 'Butterworth'
        fc = 1e3 / (fe/2);
        [b, a] = butter(7, fc, 'low');
        signal_all = filter(b, a, signal_all);
end
```

Finally, the representations of the signal in time domain and in frequency domain are plotted.

The sound of the whole signal is played.

```
    % Plot time-domain signal
    figure;
    plot(t_all, signal_all);  % use t_all for the full signal time vector
    title('Time Domain Signal');

    % Frequency domain representation
    [SignalTotal, F_total] = analiz_sp(signal_all, fe);
    figure;
    plot(F_total, SignalTotal);
    title('Signal Spectrum');

    sound(signal_all, fe);
end
```

- <u>Implementation of the function for pulse signal generation (generatePulseSignal)</u>

Firstly, the signal parameters are defined:

- sampling frequency
- pause time period
- tone time period
- inter digit pause time period

Then, the pause and tone segments are created, creating a pulse from the "mute" and "break" signals (signal_pulse), using the inter digit pause to create the end of the digit pulse.

```
% Pulse signal generation using input number
function generatePulseSignal(phoneNumber, addNoise, selectedFilter)

    %Parameters
    fsamp = 2e3;

    %Pulse timing setup-how long pulses last
    TM = 0.033; % mute (low) time for each pulse (33 ms)
    TB = 0.066; % break (high) time between pulses (66 ms)
    TInterD = 0.4;

    signal_M = zeros(1, floor(TM * fsamp)); % generate one mute segment (Break =
line interrupted = 0)
```

```
    signal_B = ones(1, floor(TB * fsamp)); % generate one break segment (Make = 
line active = 1)
    signal_InterD = zeros(1, floor(TInterD * fsamp)); % creates a long silence 
(vector of zeros) between two digits, that lasts TInterD seconds

    signal_pulse = [signal_B signal_M]; %create pulse
    signal_end = [signal_B signal_InterD]; % adding one last small active part 
(signal_B) to close the last pulse,
    % then a long pause (signal_InterD) to separate this digit from the next 
digit.
    signal = signal_InterD; % initialize the full signal before starting to build 
the actual pulses for the digits


    % Signal amplitude parameters
    A1 = 0.2;   % amplitude for the first component
    A2 = 0.3;   % amplitude for the second component
```

An ‘if’ condition is implemented in order to only allow digits 0-9 to be used to generate the pulse signal. An error is displayed when introducing anything else.

```
    %Input validation (only digits allowed)
    if ~all(isstrprop(phoneNumber, 'digit'))
        msgbox("Pulse signaling only accepts digits 0-9.",'Error','error')
        uialert(gcf, 'Pulse signaling only supports digits 0–9.', 'Invalid 
Input');
        return;
    end
```

Next, the characters are converted to digits and then to pulse counts, storing them in a pulse count vector (**digit_all**).

```
    % Convert characters to Digits to Pulse counts
    digit_all = zeros(1, length(phoneNumber)); % preallocate a vector to store 
pulse counts for each digit
    for i = 1:length(phoneNumber) % loop through each character in the phone 
number
        ch = phoneNumber(i); % get the current character
        if ch >= '0' && ch <= '9' % check if it is a valid digit (0-9)
            digit = str2double(ch); % convert the character to a numeric value
            if digit == 0
                digit = 10; % Replace 0 with 10 pulses
            end
            digit_all(i) = digit; %store the converted digit (pulse count) in the 
vector
```

```
        else
            warning(['Invalid digit in number: ' ch]); % show a warning if non-
digit character
        end
    end
```

The signal is built by looping through each digit's pulse count, getting the number of pulses for each digit and repeating the basic pulse (**signal_pulse**) for digit-1 times, due to the digit already starting with a pulse because of the first make (**signal_B**) in this line `signal_pulse = [signal_B signal_M]; %create pulse.`

A digit is finally created by adding a small make (**signal_B**) and a long inter digit pause to mark the end.

```
    % Build pulse signal from digits
    for i = 1:length(digit_all) % loop through each digit's pulse count
        digit = digit_all(i); % get the number of pulses for the current digit
        signal = [signal repmat(signal_pulse, 1, digit - 1)]; %repeat the basic
pulse (signal_pulse) (digit-1) times
        % because it already started with an initial Make (signal_B), so for
digit=1, no repetitions are needed, and so on
        signal = [signal signal_end]; % after sending pulses for the digit, add a
short active (signal_B) + long pause (signal_InterD)
    end
```

It is also possible to add noise to the pulse signal by using the same function implemented in the DTMF signal generator, but using the pulse parameters that were set.

```
    % Add noise if the checkbox is selected
    if addNoise
        % Signal Power
        power_signal_measured = rms(signal)^2; %compute signal power
        disp(['Signal Power: ', num2str(10*log10(power_signal_measured)), '
dbW']); %convert to dbW and display

        % Add Noise
        power_noise = 0;
        noise = wgn(1, length(signal), power_noise);  % gaussian noise with 0 dBW
power

        % Display SNR
        disp(['SNR: ', num2str(10*log10(power_signal_measured) - power_noise), '
dB']); %convert into db and display

        % Signal with noise
        signal = signal + noise;
    end
```

The signal time-domain and frequency-domain representations are also plotted.

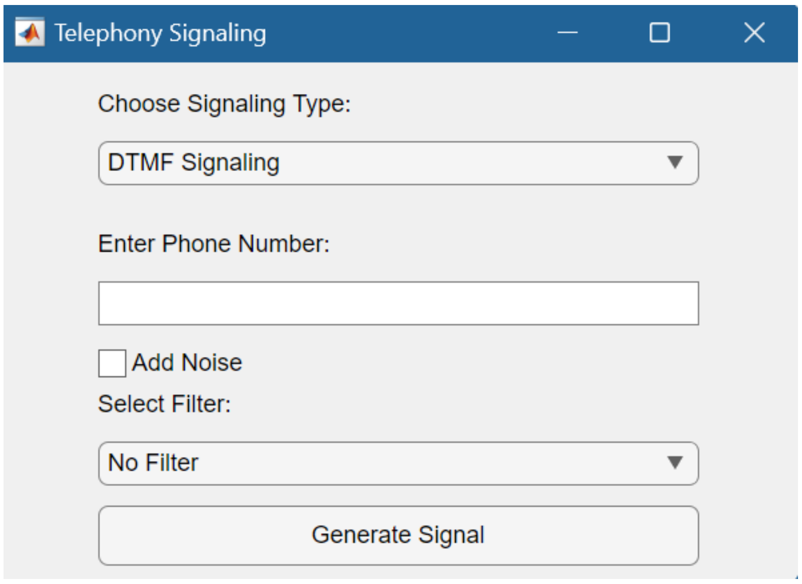
```
    % Plot time-domain signal
    time = (0 : length(signal) - 1) / fsamp;
    figure;
    plot(time, signal);
    title('Pulse Signal (Time Domain)');


    % Frequency domain representation
    [Signal, F] = analiz_sp(signal, fsamp);
    figure;
    plot(F, Signal);
    title('Pulse Signal Spectrum');
end
end
```

Finally, for the spectrum analysis, the same analiz_sp function was used.

```
function [P1,f] = analiz_sp(s,fe)

Y = fft(s);
L=length(s);

P2=abs(Y/L);
P1=P2(1:floor(L/2)+1);
P1(2:end-1)=2*P1(2:end-1);
f=fe/L*(0:L/2);
s=P1;

end
```

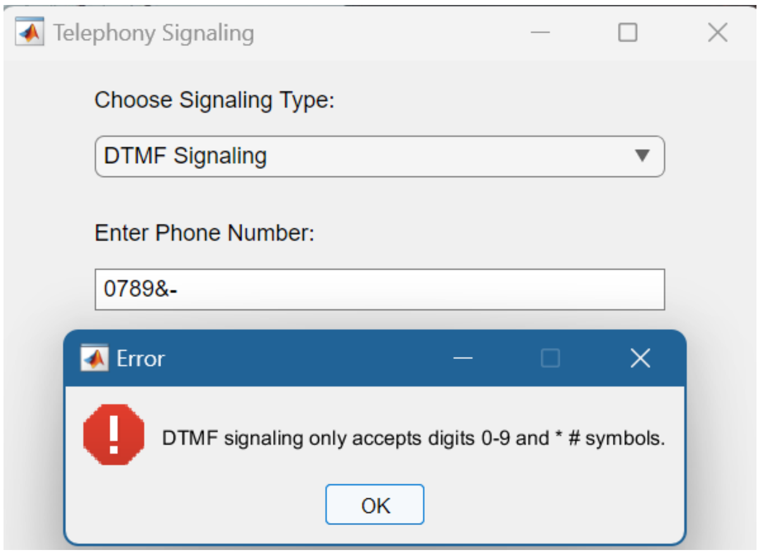
Description and Interpretation of Results

The GUI containing the aforementioned components:



- Select the addres signaling type: DTMF Signaling or Pulse Signaling
- Enter the phone number (for DTMF, also * and # symbols)
- Optionally add noise
- Optionally add filter
- Generate Signal

DTMF Signaling Character Verification



Telephony Signaling

Choose Signaling Type:

DTMF Signaling

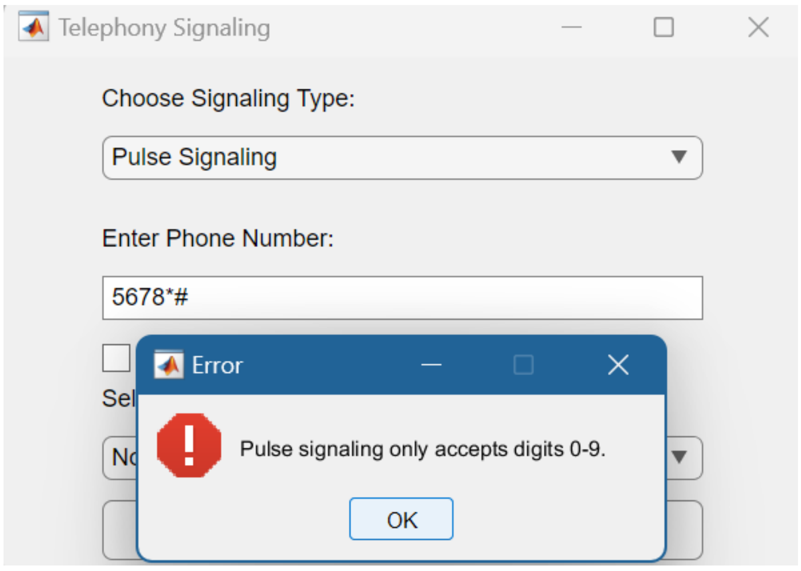
Enter Phone Number:

0789&-

Error

DTMF signaling only accepts digits 0-9 and * # symbols.

Pulse Signaling Character Verification



Telephony Signaling

Choose Signaling Type:

Pulse Signaling

Enter Phone Number:

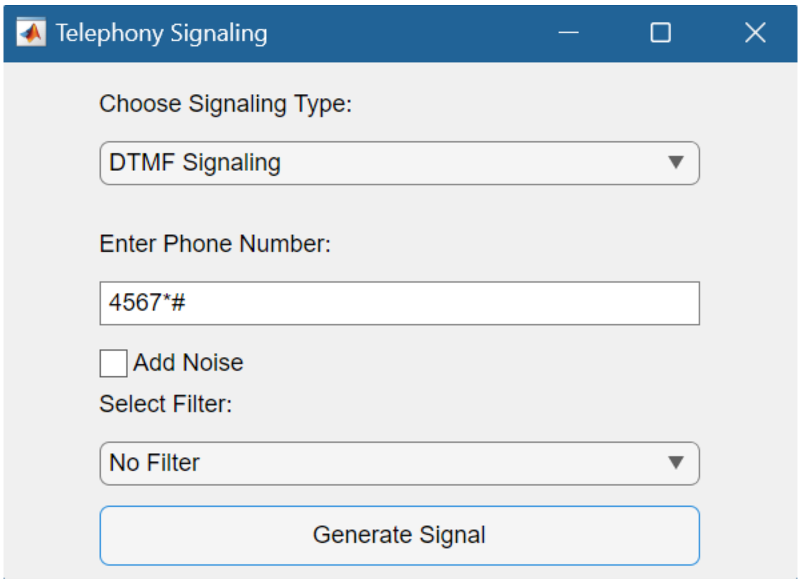
5678*#

Error

Pulse signaling only accepts digits 0-9.

1) DTMF Signaling

Introduced number:


Telephony Signaling

Choose Signaling Type:

DTMF Signaling

Enter Phone Number:

4567*#

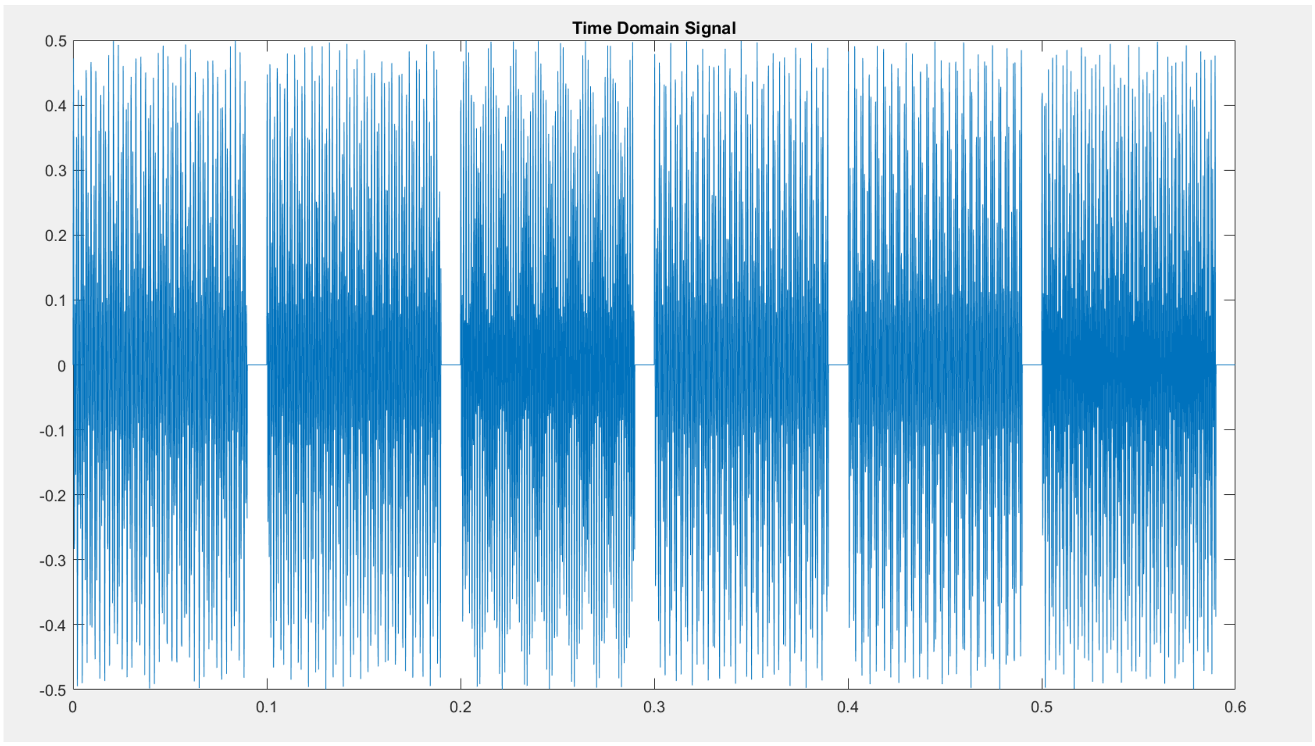
☐ Add Noise

Select Filter:

No Filter

Generate Signal

Signal in time domain:



There are 6 blocks of visible, one for each digit introduced. They last for the same amount of time, t_tone=0.09s and have equal pauses between them, t_pause=0.01s.

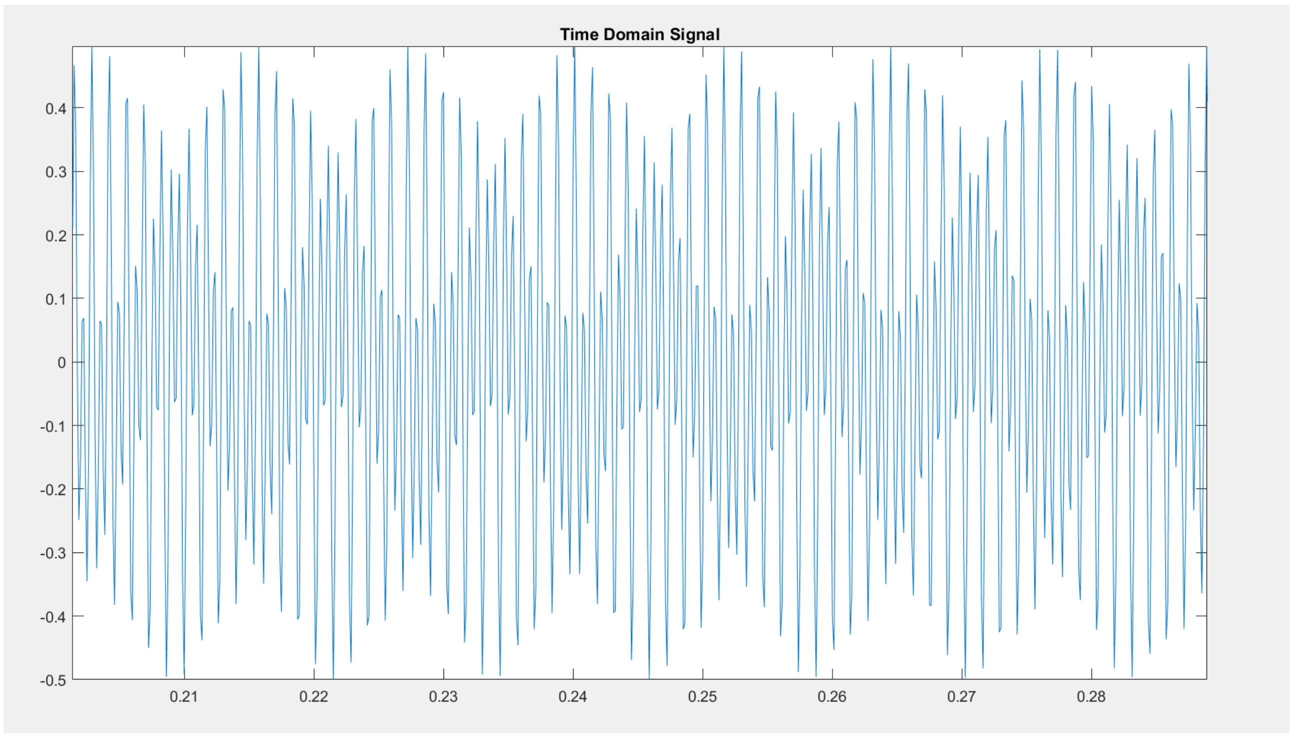
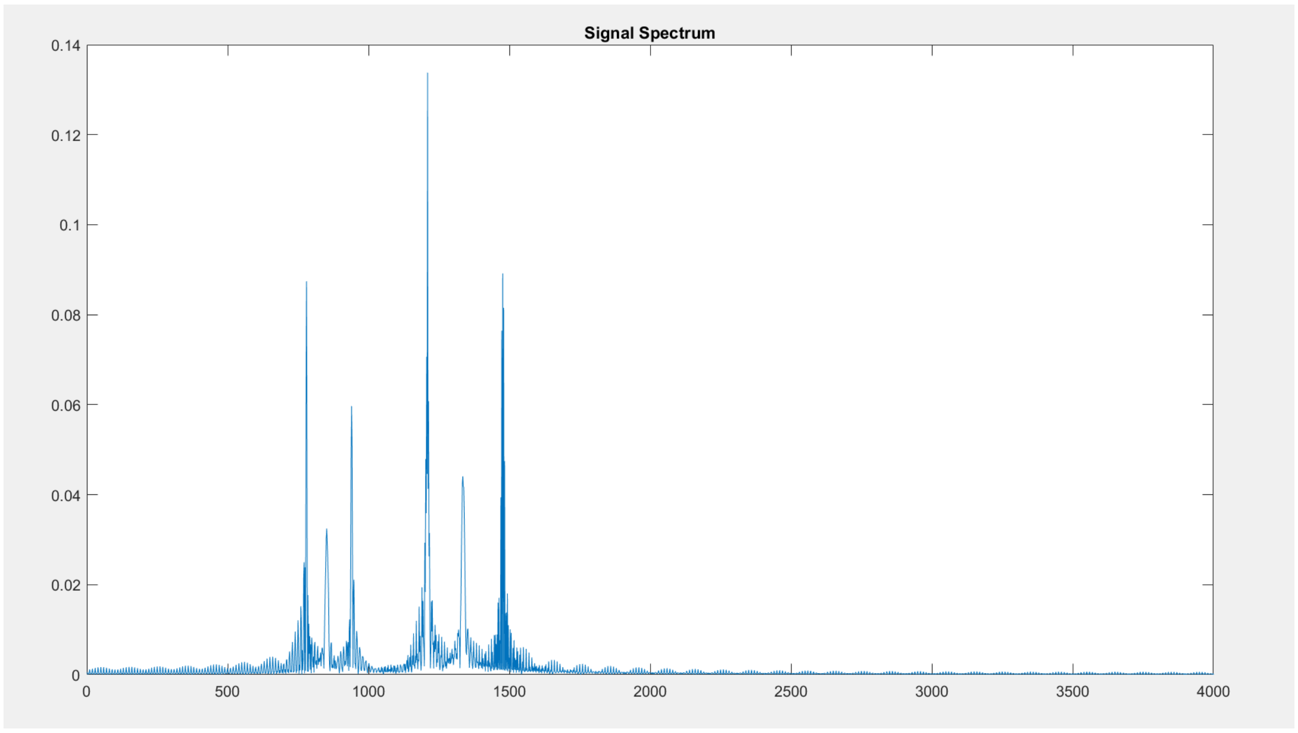
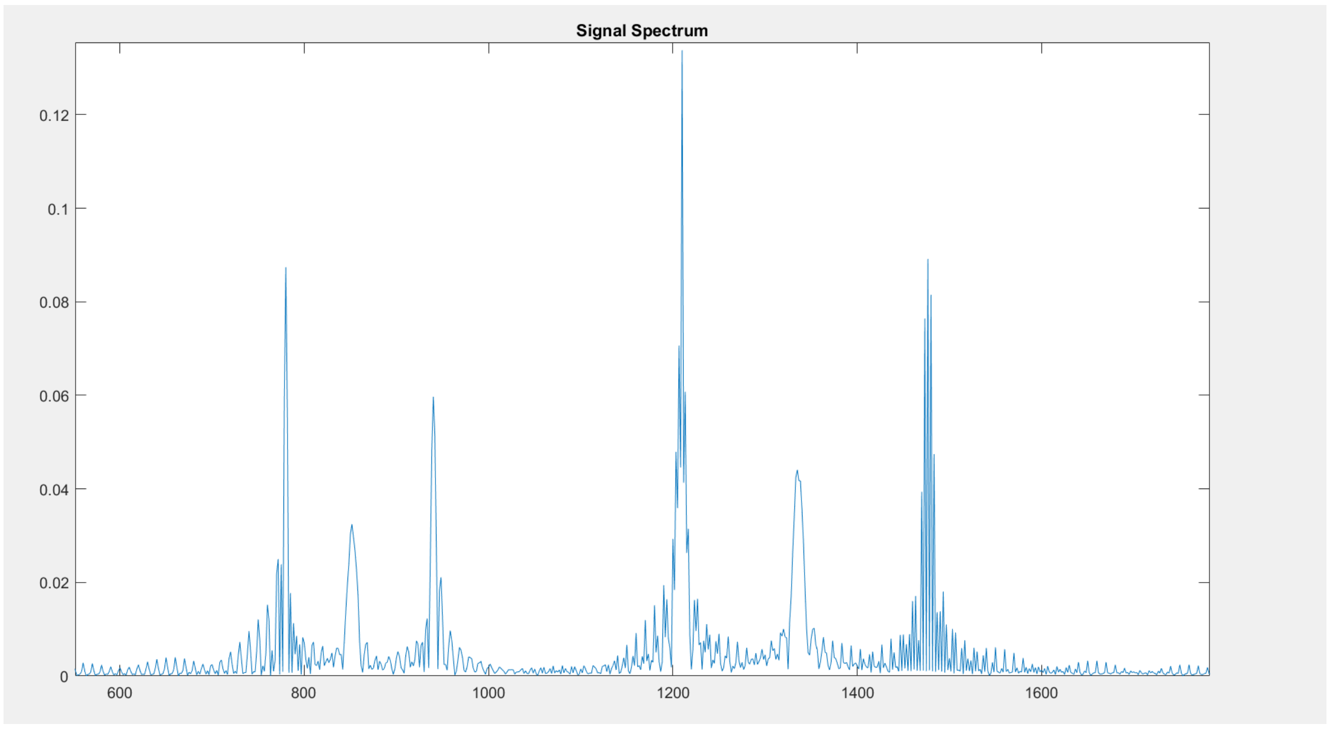


Figure 1: Close-up of digit 6

Each tone block is the sum of two sine waves, one for the low frequency and one for the high frequency.

Frequency spectrum:





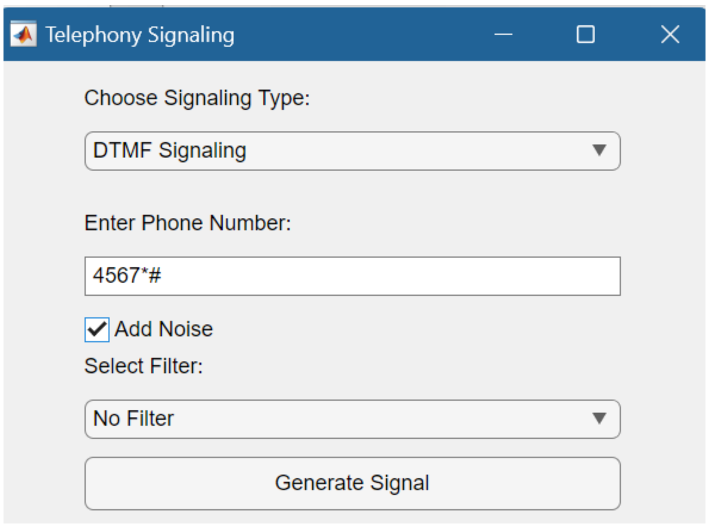
The spectrum shows distinct peaks at:

- **3 low frequencies** (779, 852, 941 Hz)
- **3 high frequencies** (1209, 1336, 1477 Hz)

These correspond exactly to the expected DTMF frequencies for the digits introduced.

DTMF+Noise

Checked add noise:


Telephony Signaling

Choose Signaling Type:

DTMF Signaling

Enter Phone Number:

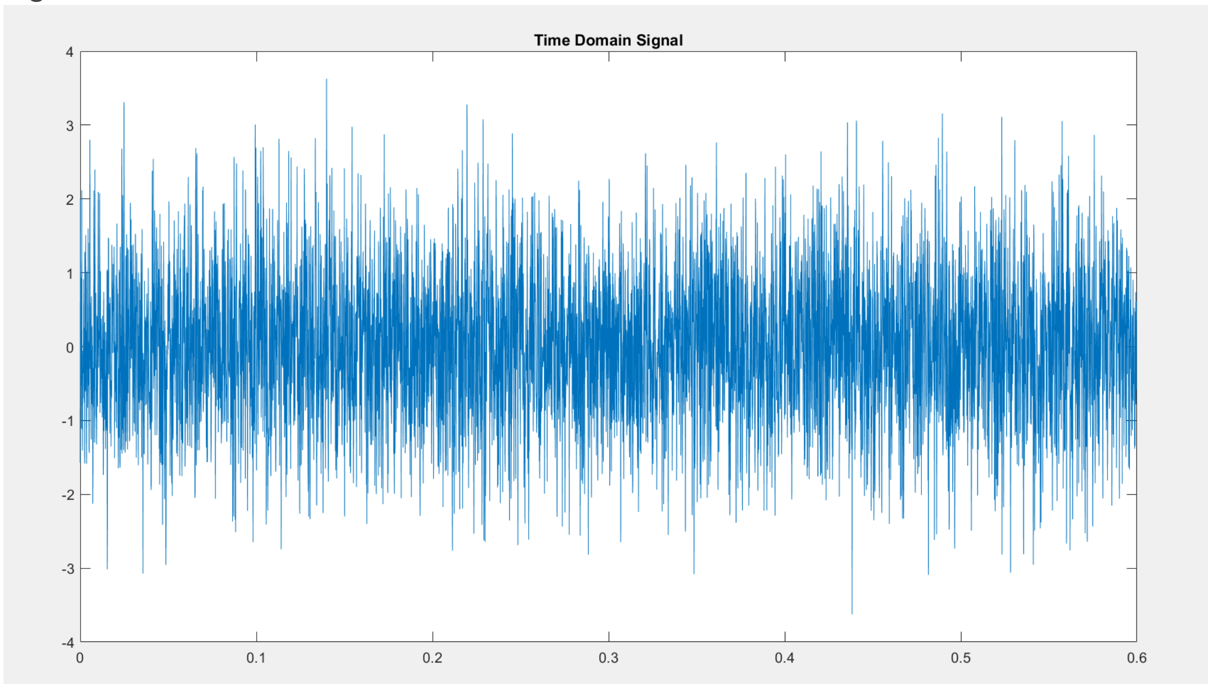
4567*#

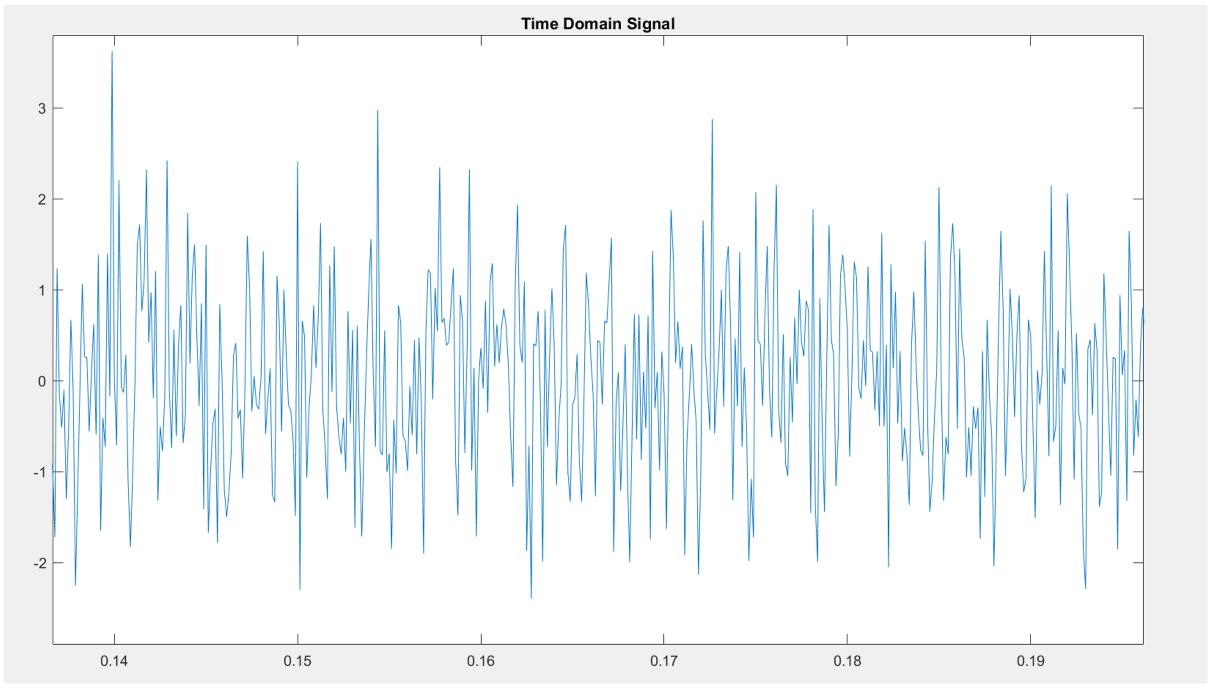
[✓] Add Noise

Select Filter:

No Filter

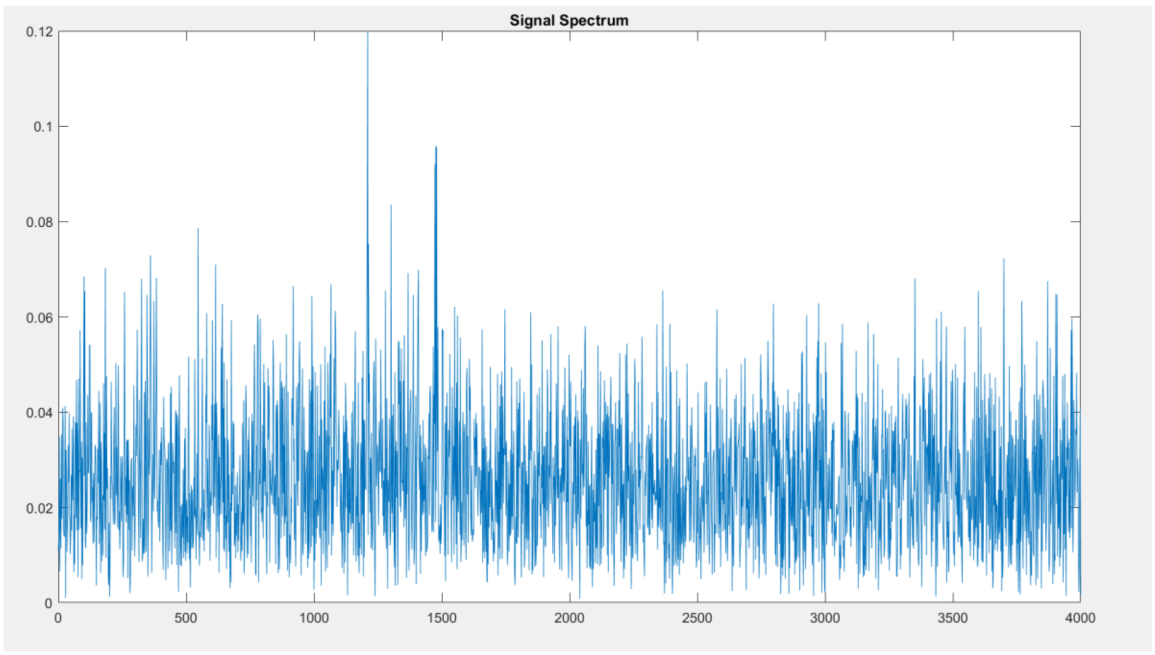
Signal in time domain:

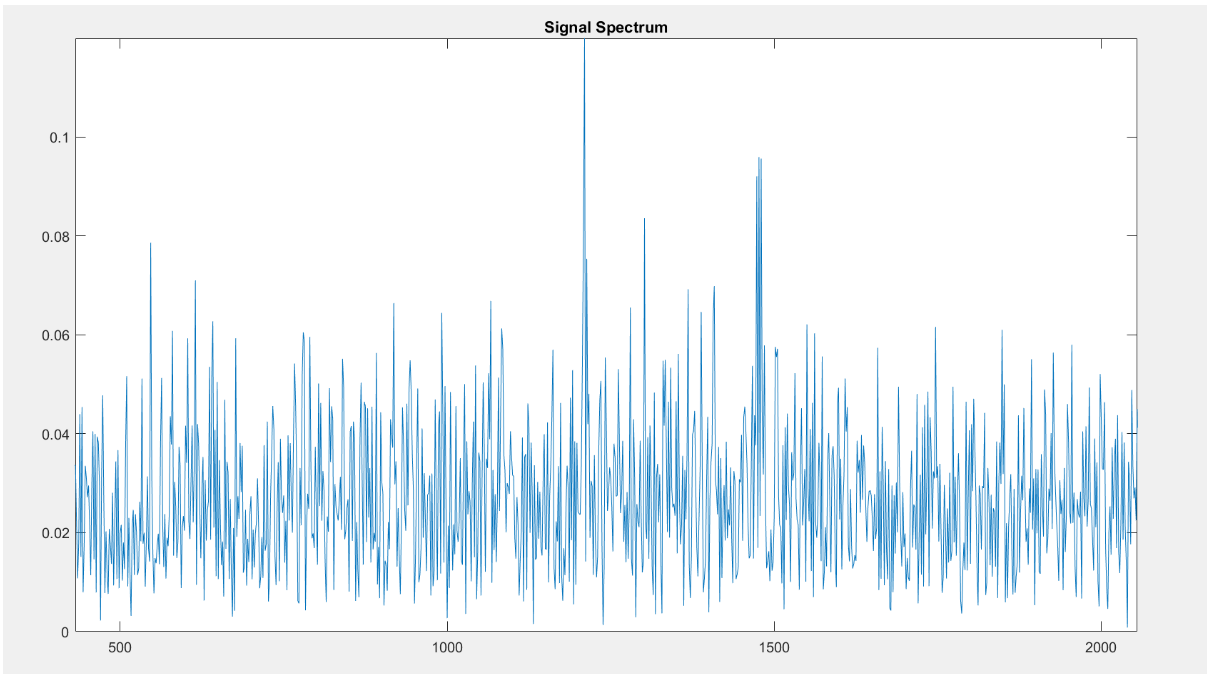




The time-domain signal now appears noisy due to the added Gaussian noise, which masks the clean DTMF tones.
Unlike the initial waveform (without noise) where each digit and pause was clearly visible, the noise overwhelms the signal, making the tone segments hard to distinguish.

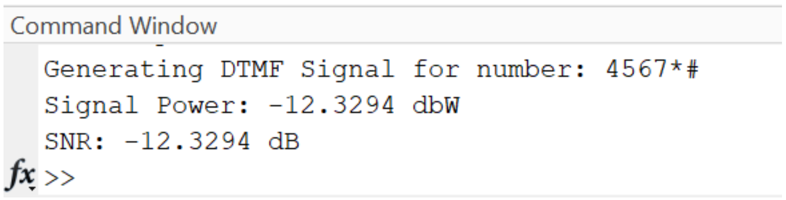
Frequency Spectrum:





In the spectrum with added Gaussian noise, the DTMF peaks are still partially visible but much harder to distinguish than in the clean signal. However, some DTMF frequency components still stand out a bit above the rest, particularly around 1200–1500 Hz, indicating that the tones are still present.

Signal Power and SNR results:

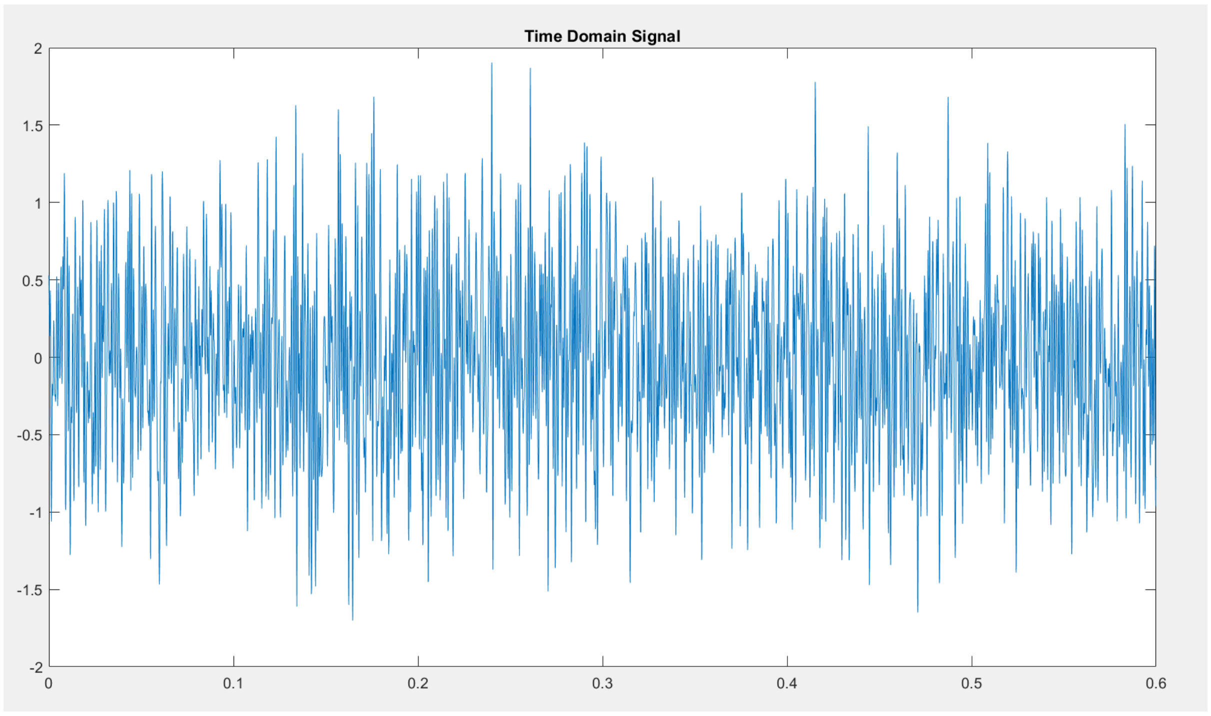


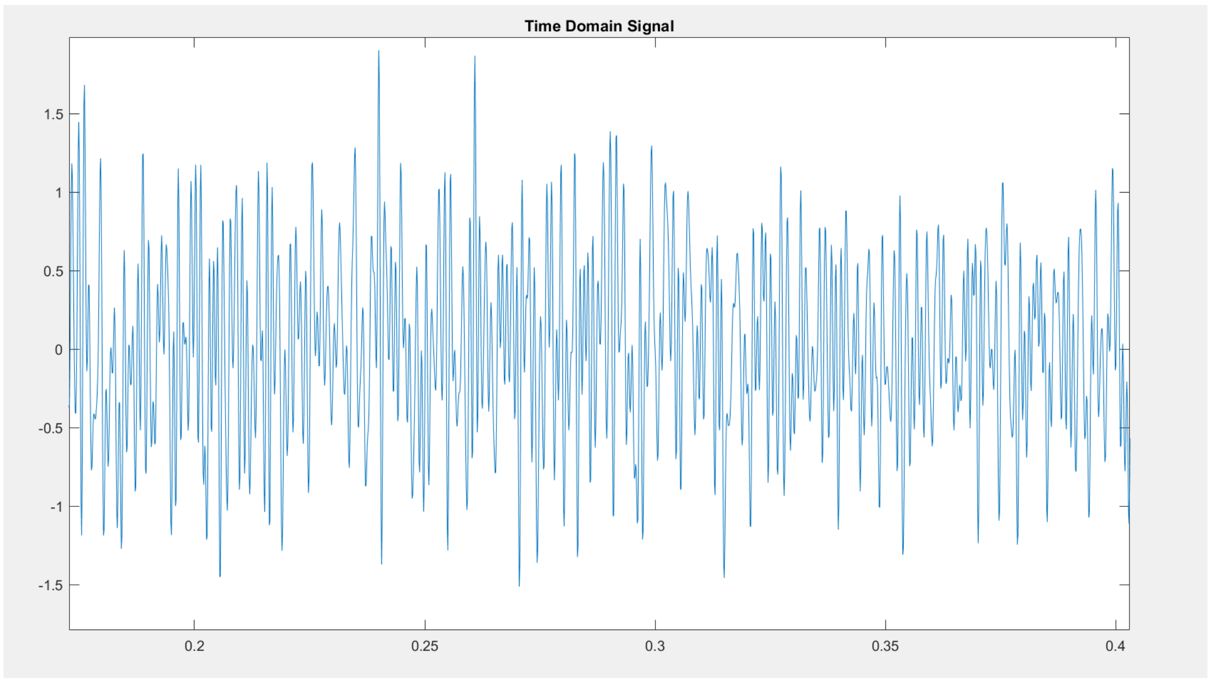
```
Generating DTMF Signal for number: 4567*#
Signal Power: -12.3294 dbW
SNR: -12.3294 dB
>>
```

Filters

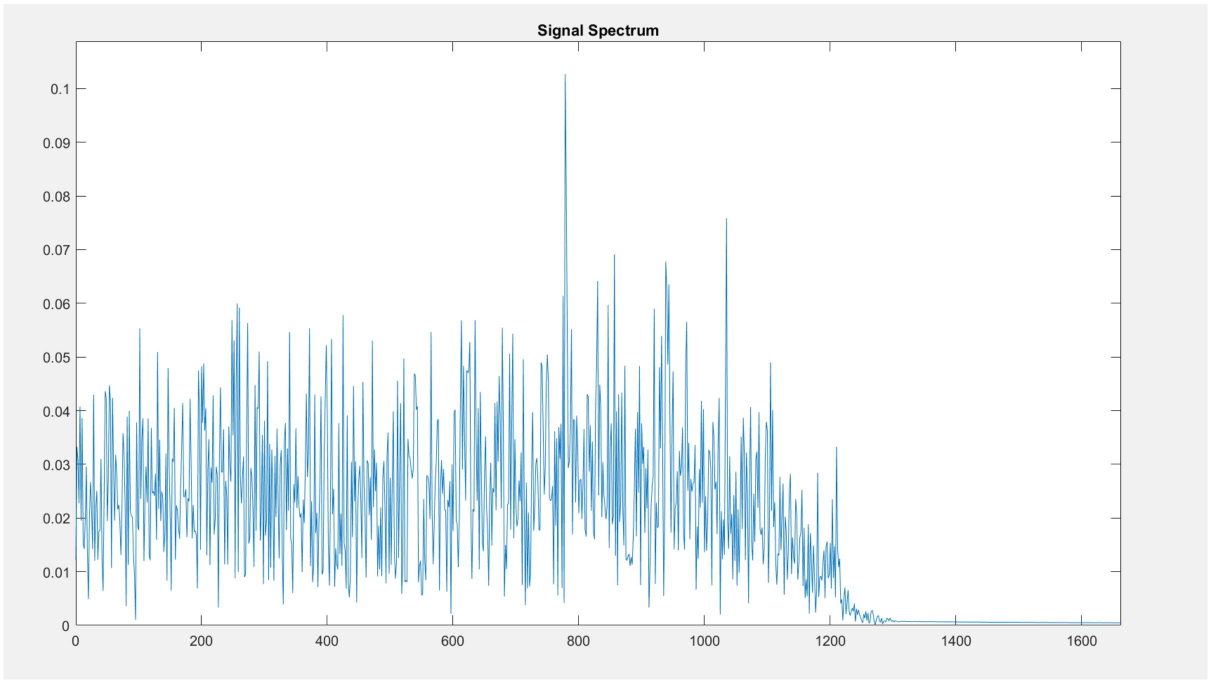
- **LOWPASS**

Signal in time domain:

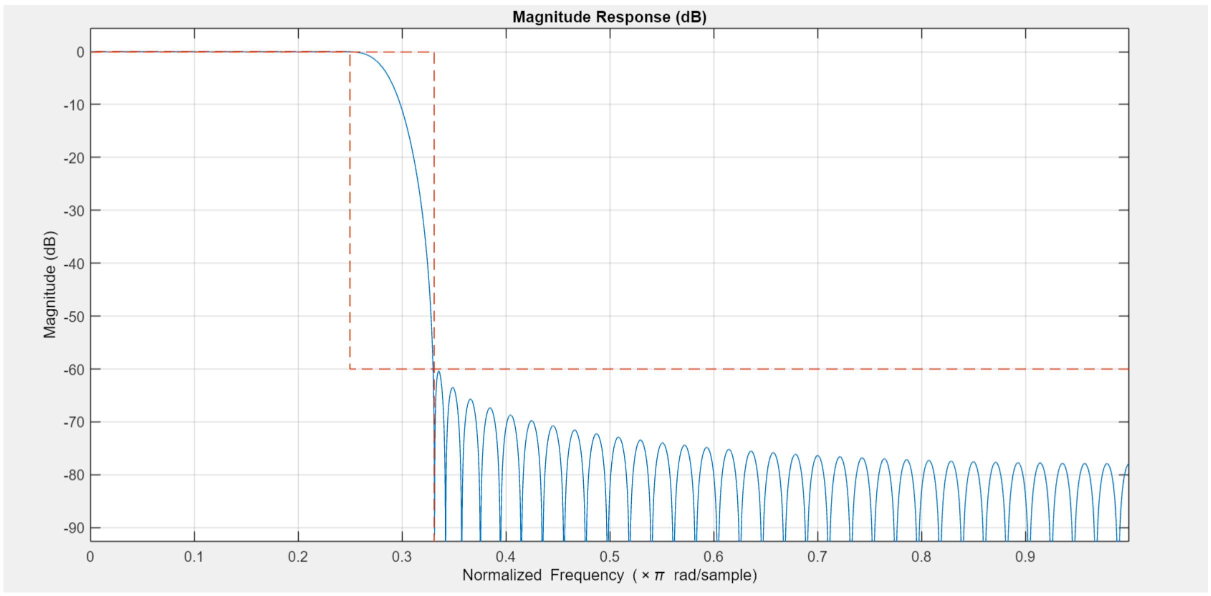




Frequency Spectrum:

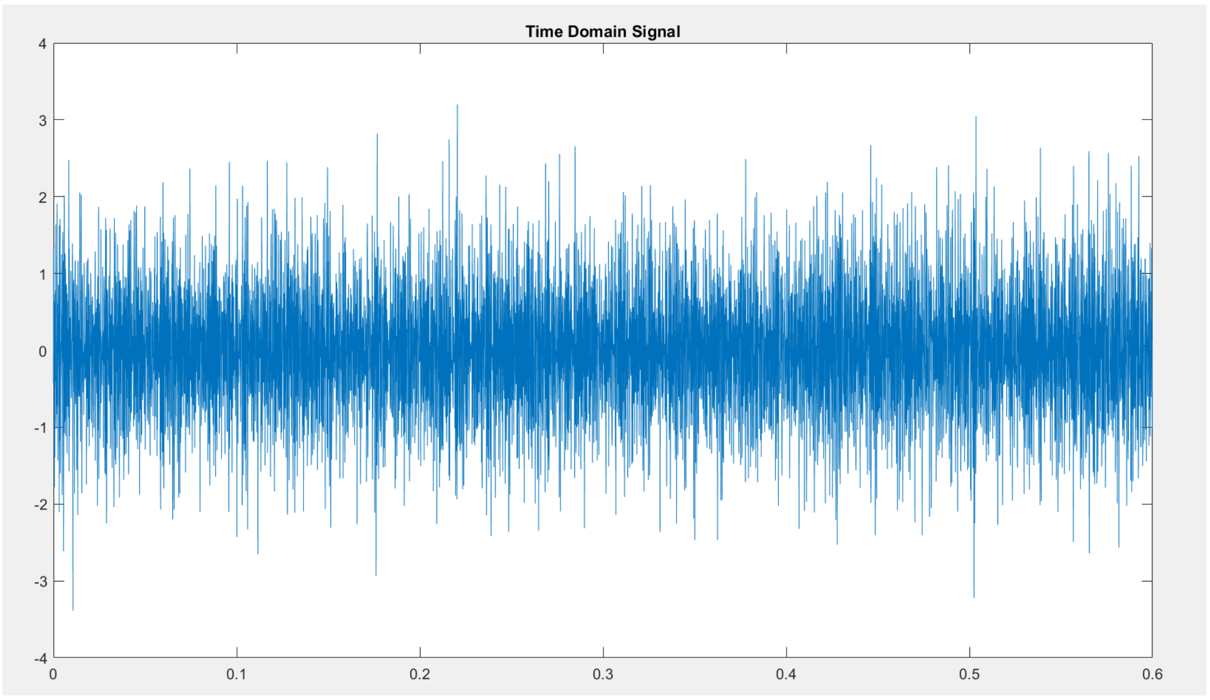


Magnitude response:



❖ HIGHPASS

Signal in time domain:



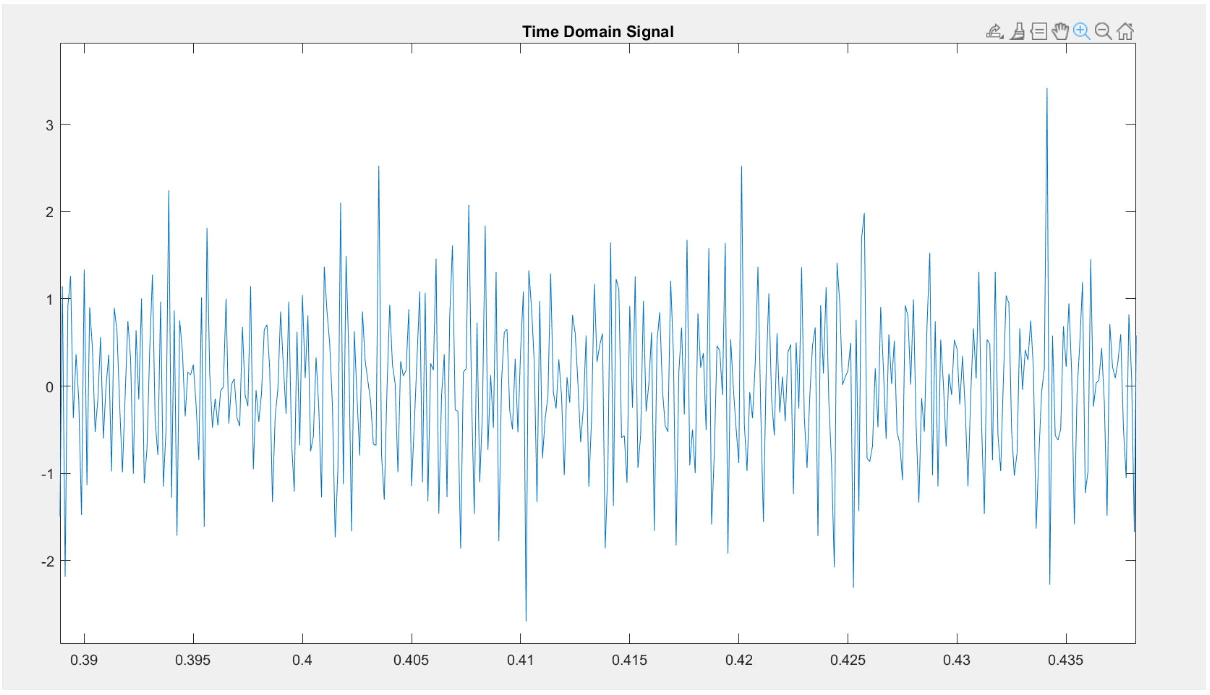
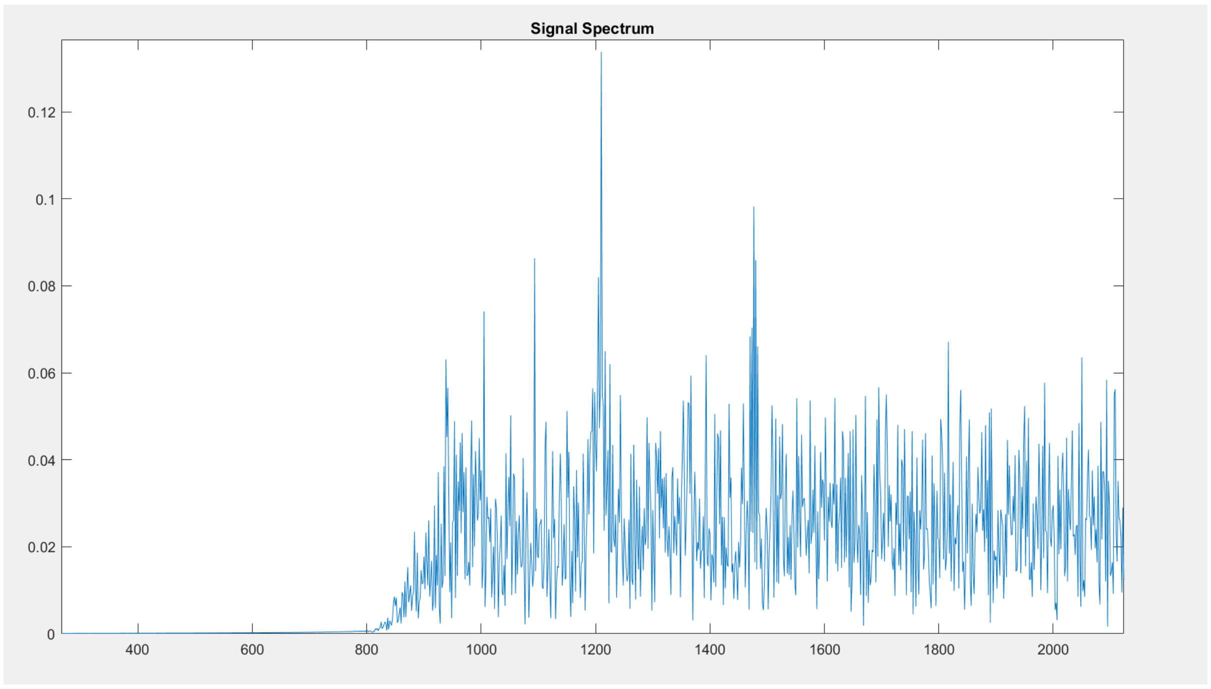
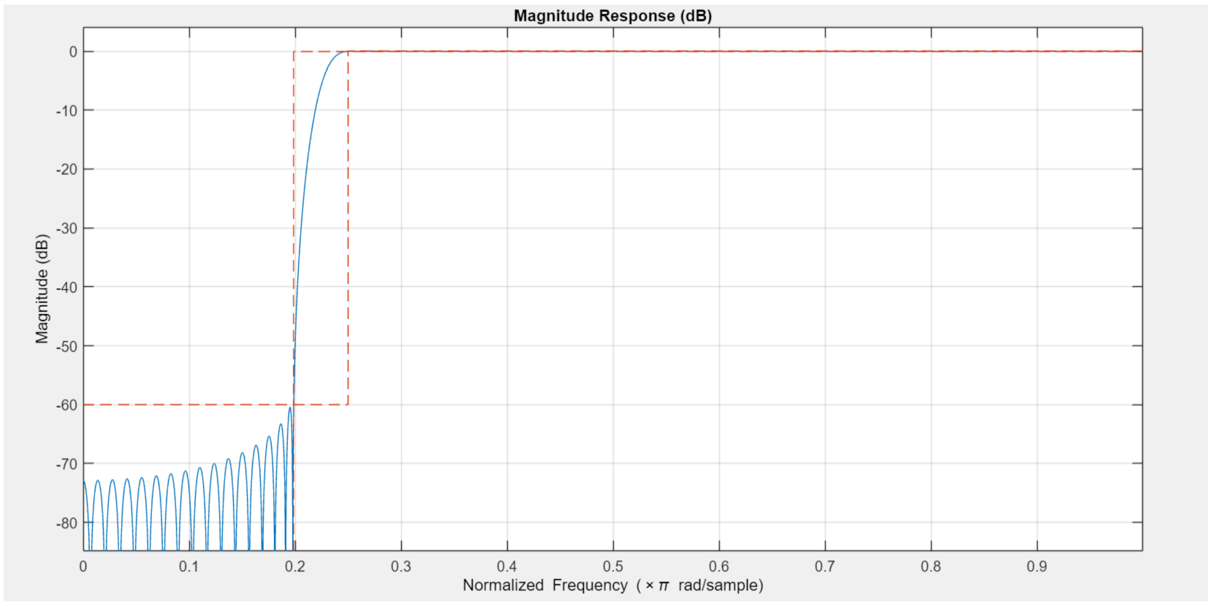


Figure 2: Close-up of digit 7

Frequency Spectrum:

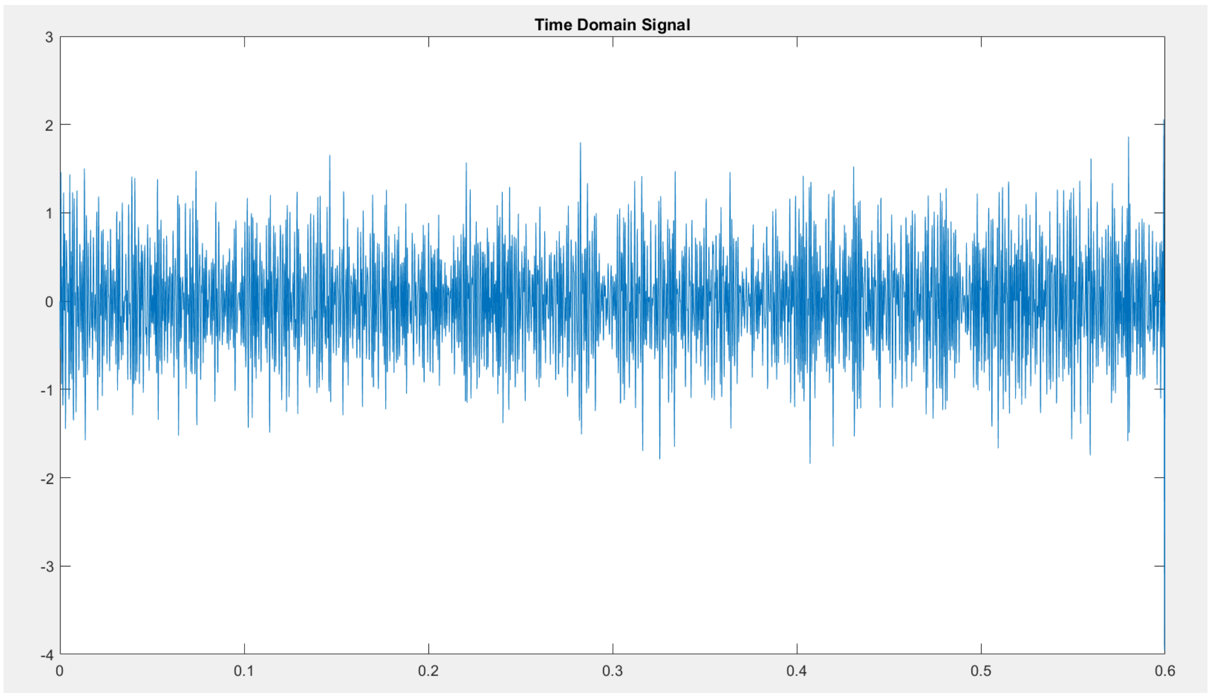


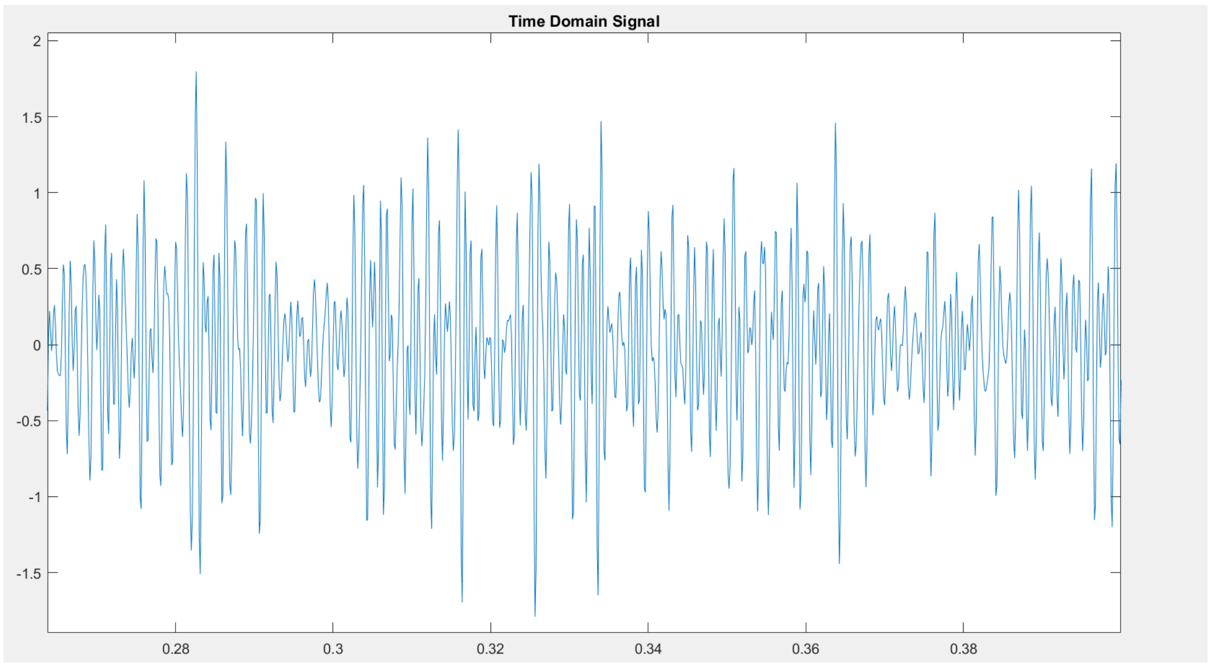
Magnitude response:



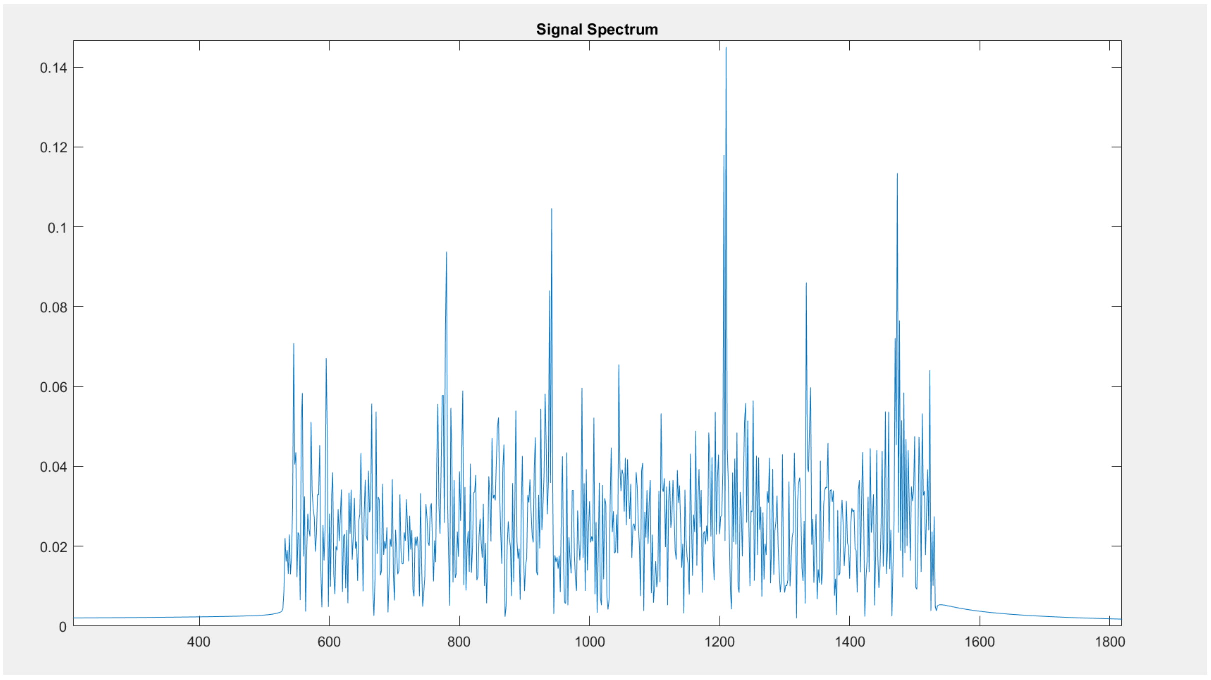
❖ BANDPASS

Signal in time domain:

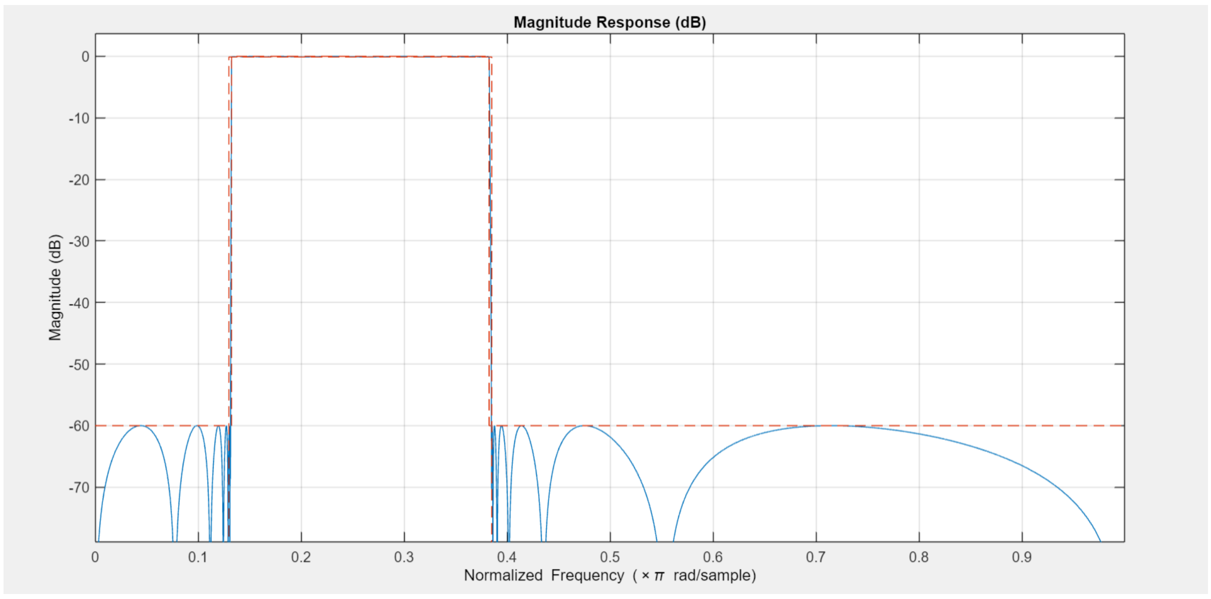




Frequency Spectrum:

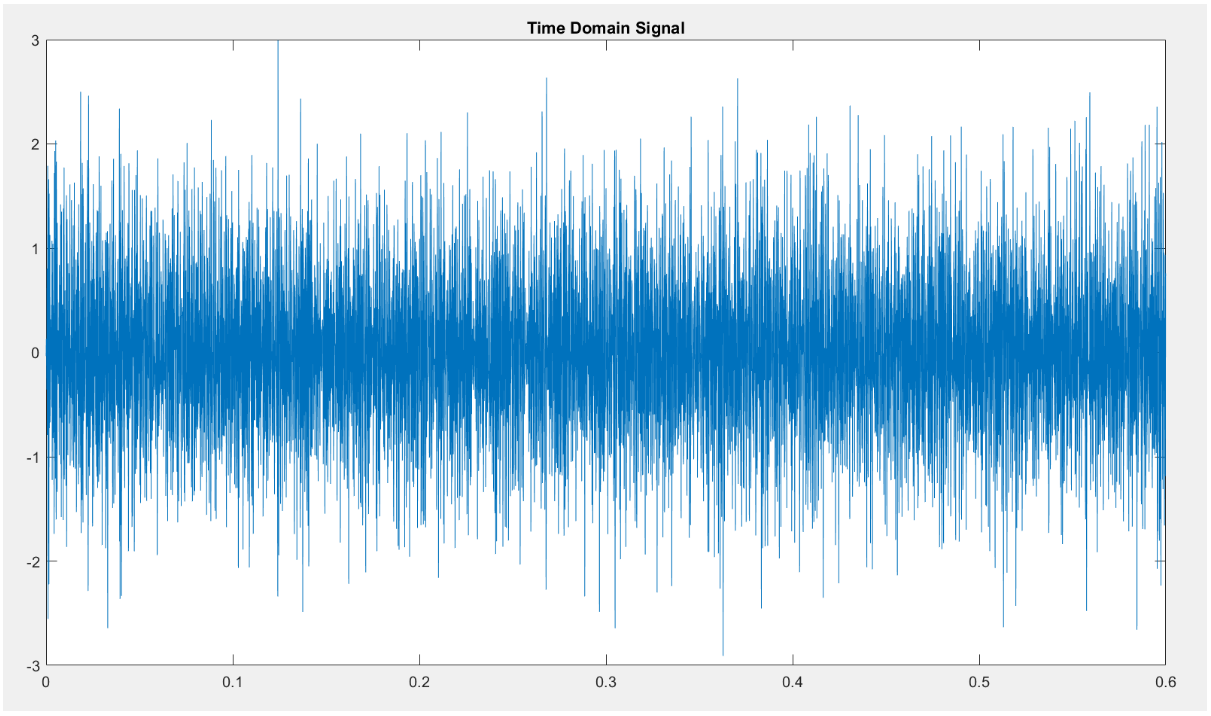


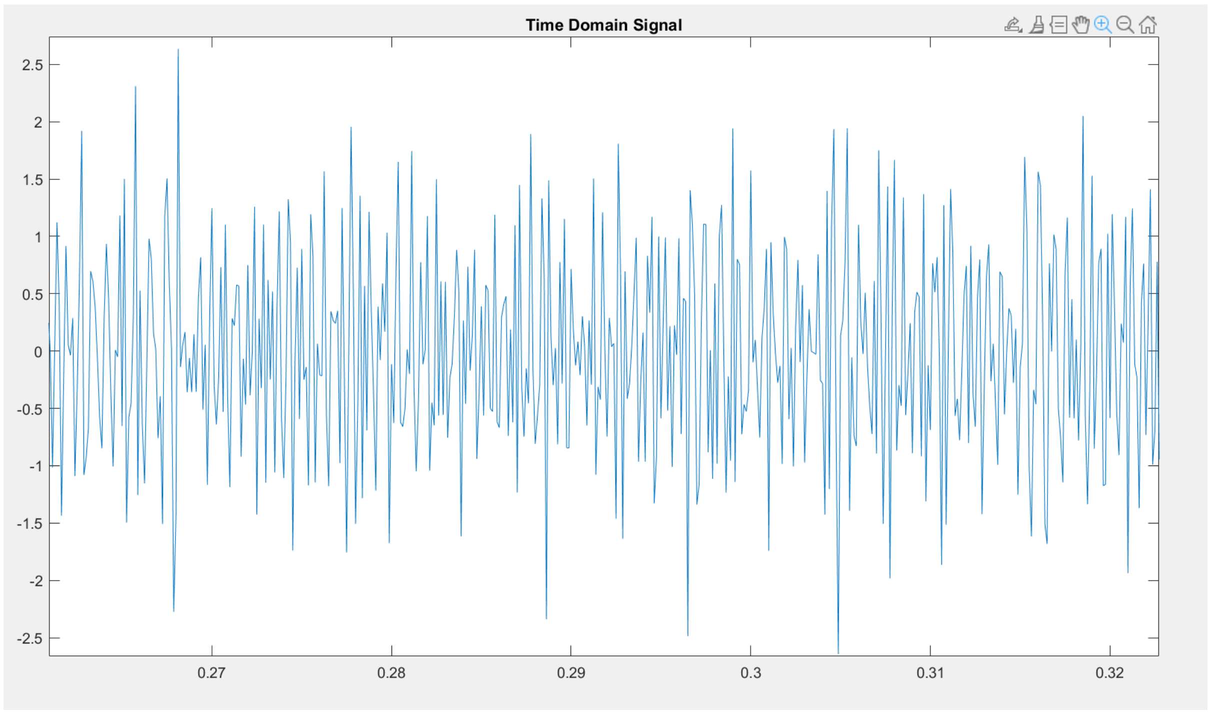
Magnitude response:



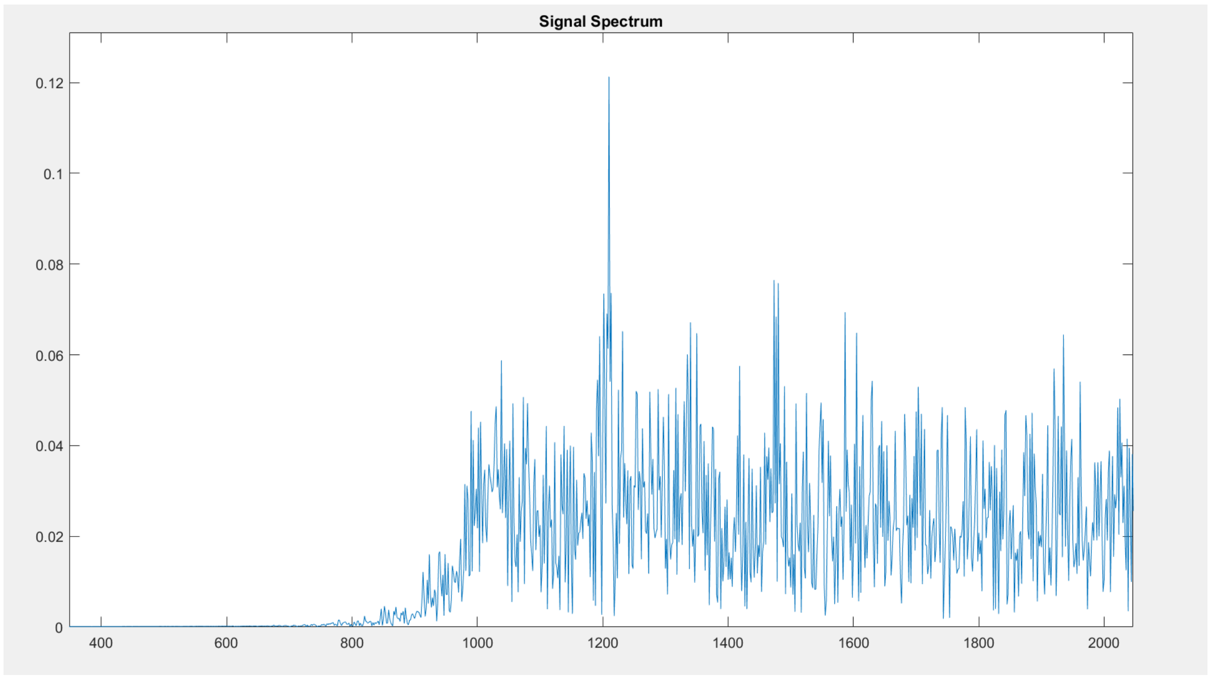
❖ CHEBYSHEV I (HIGHPASS)

Signal in time domain:



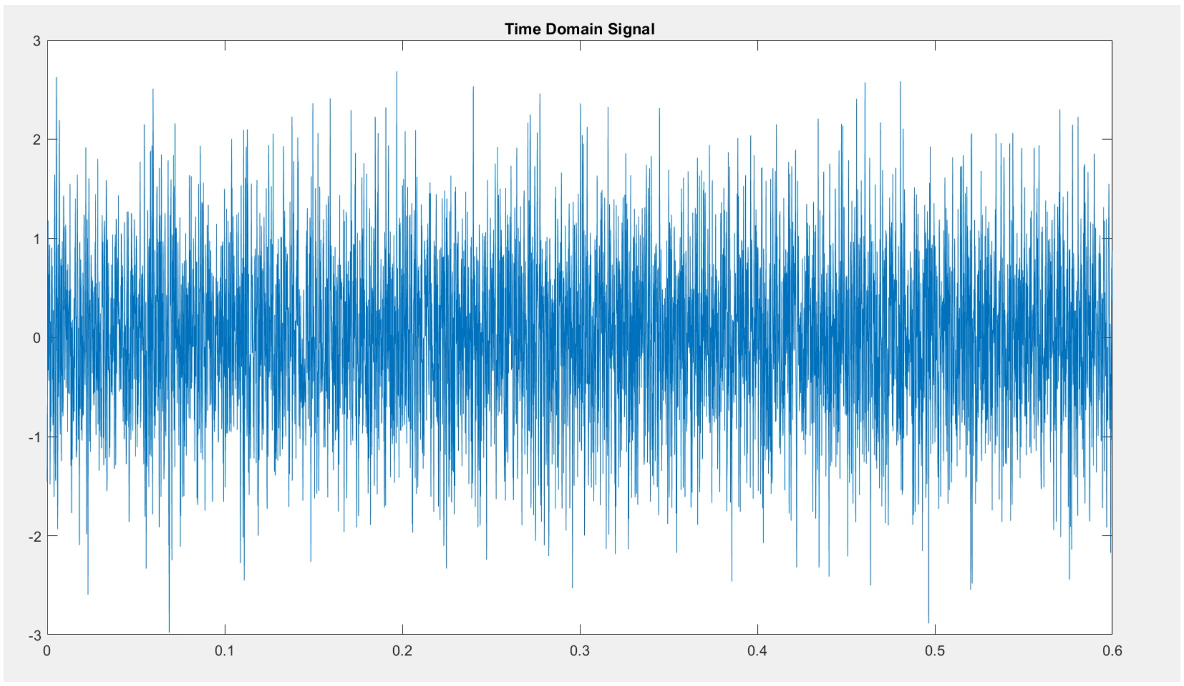


Frequency Spectrum:



❖ CHEBYSHEV II (LOWPASS)

Signal in time domain:



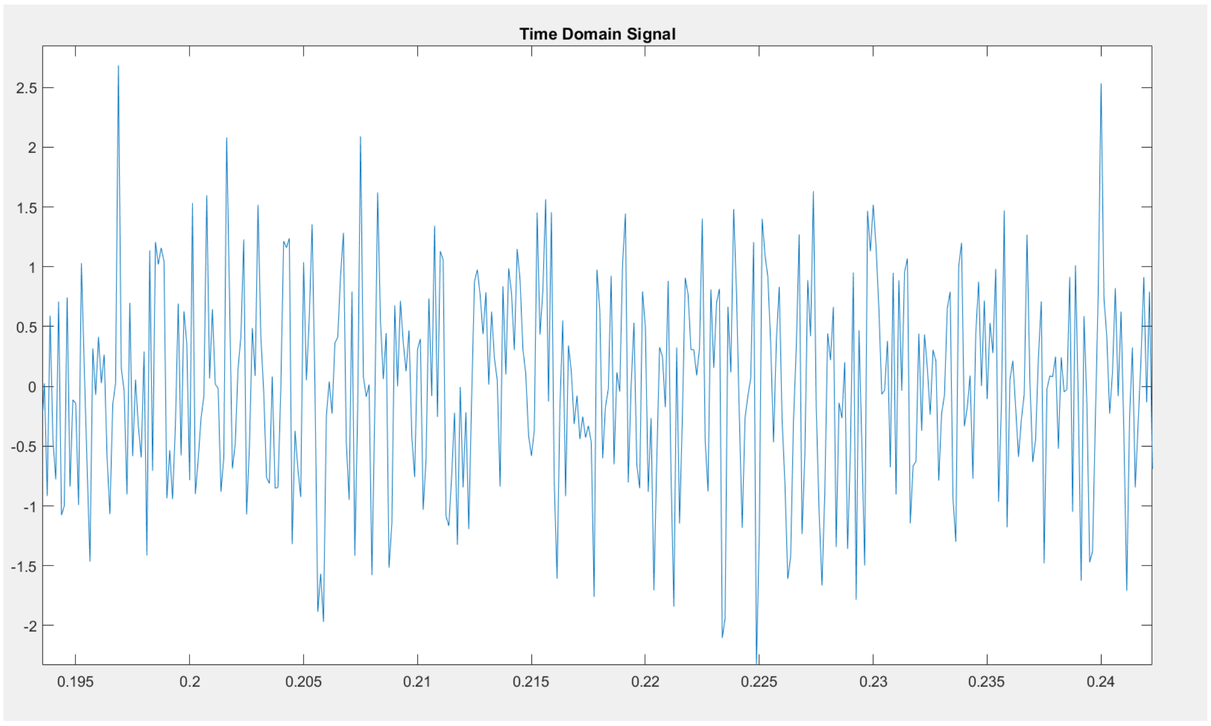
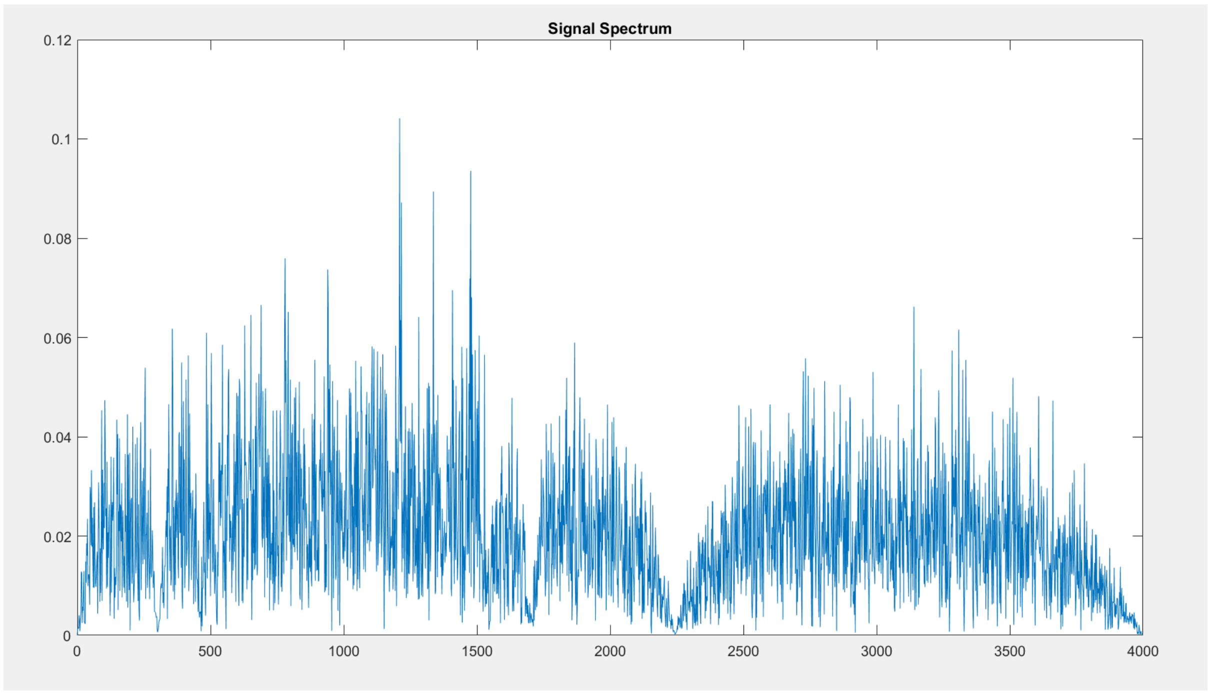
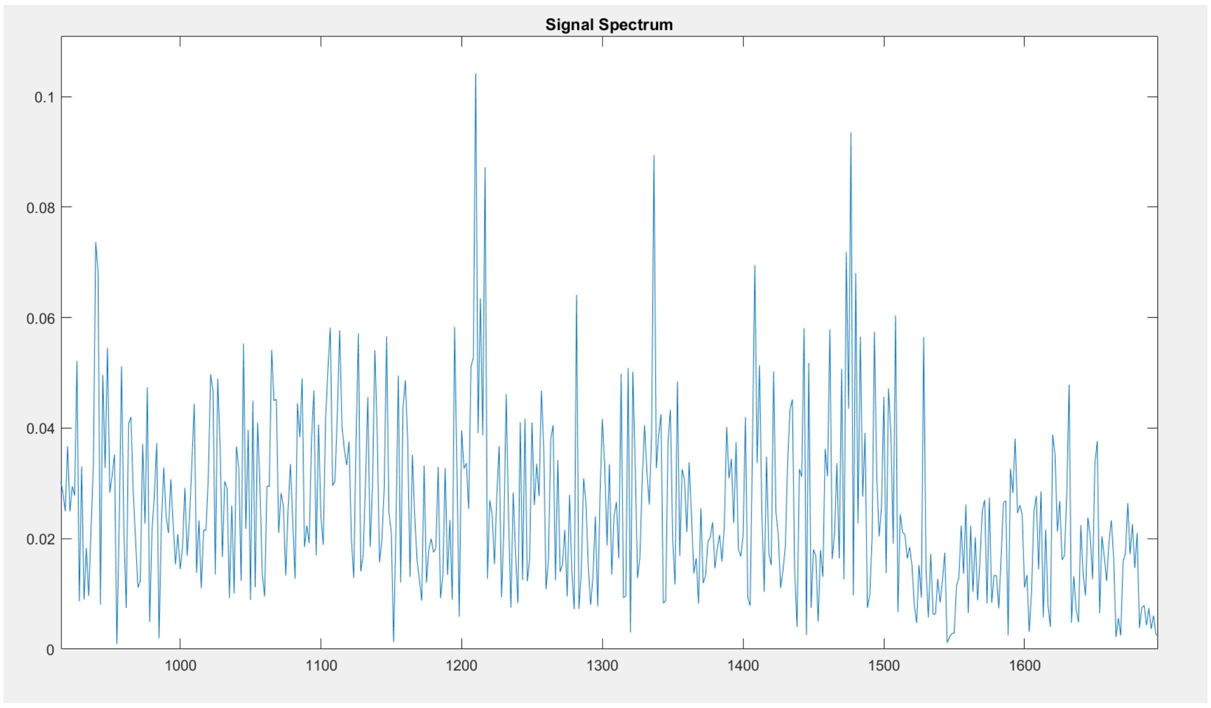


Figure 3: Close-up of digit 6

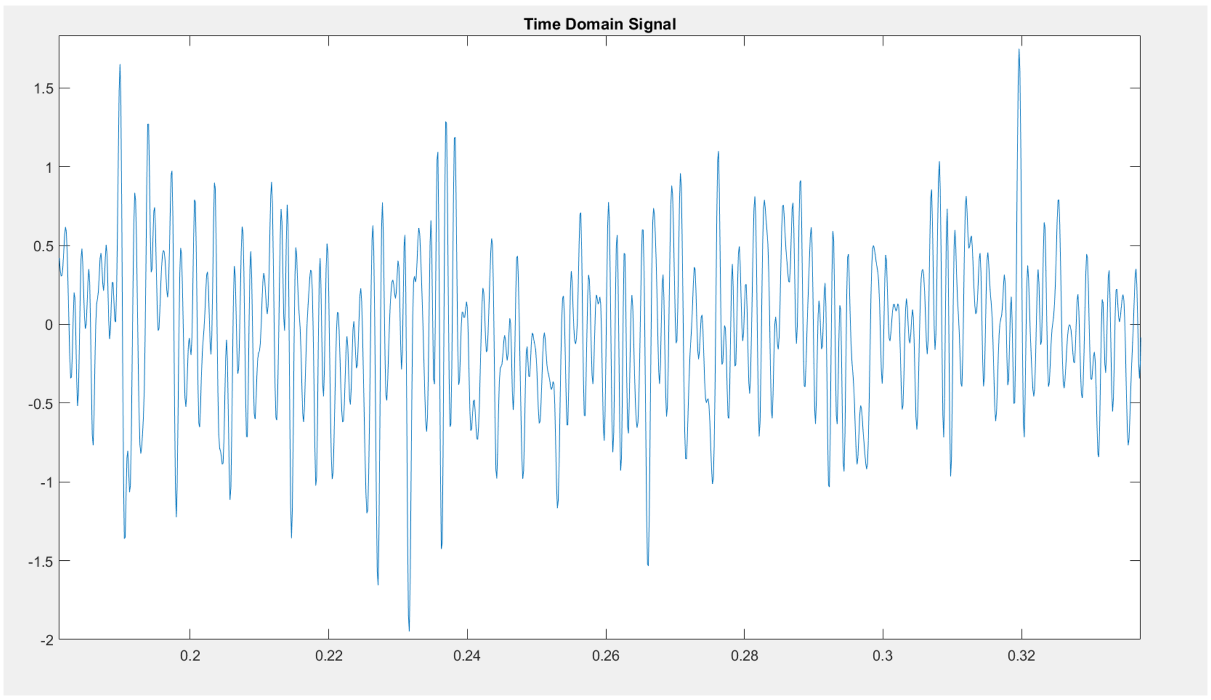
Frequency Spectrum:





BUTTERWORTH (LOWPASS)

Signal in time domain:



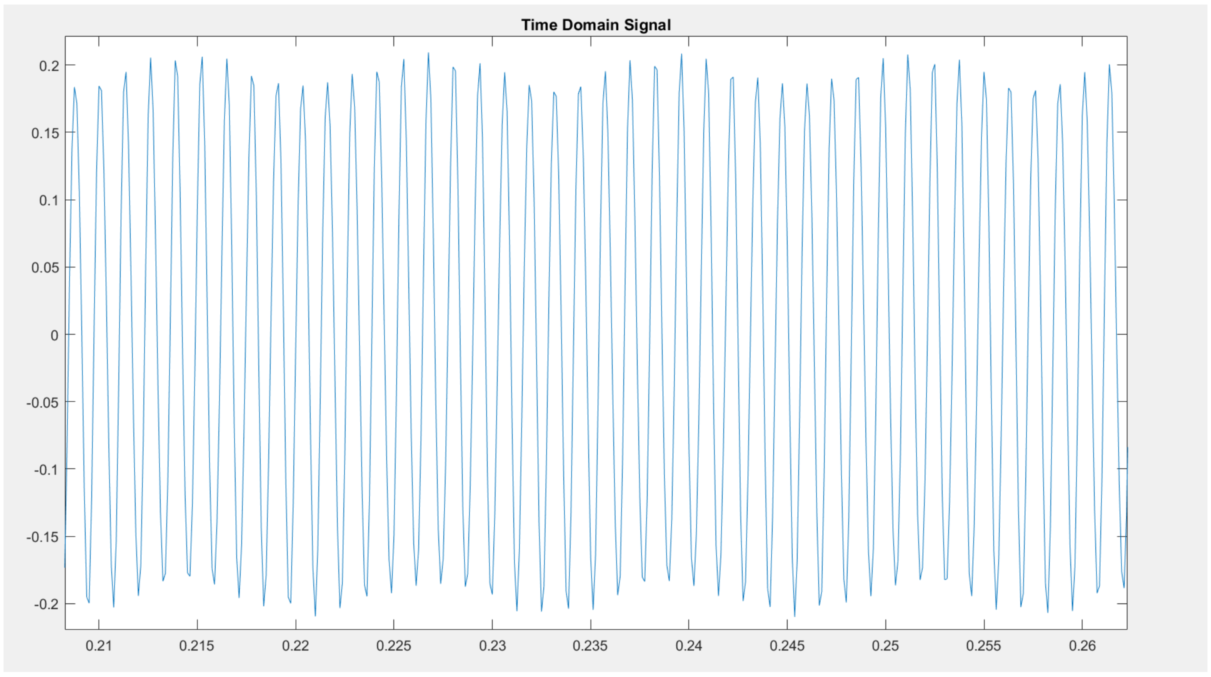
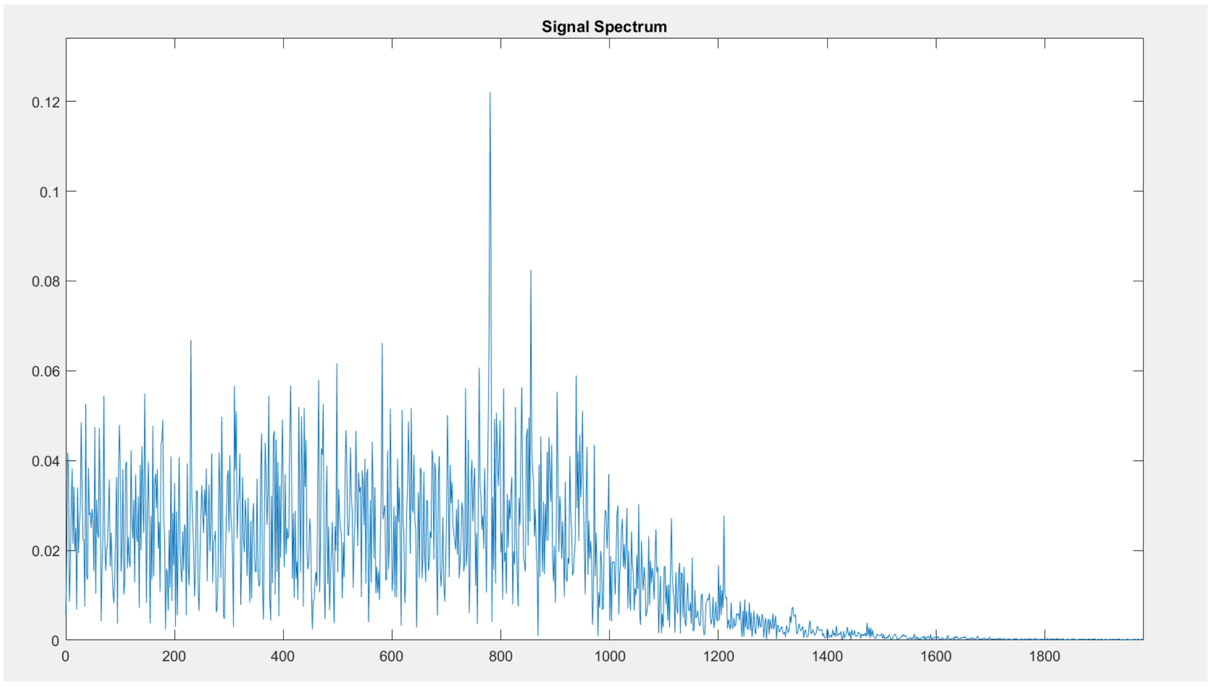


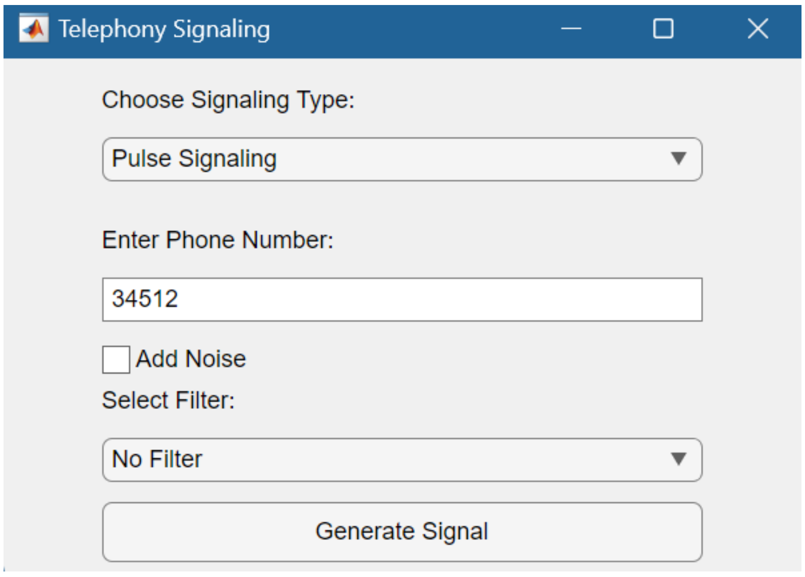
Figure 4: Close-up of digit 6

Frequency Spectrum:



2) Pulse Signaling

Introduced number:



Telephony Signaling

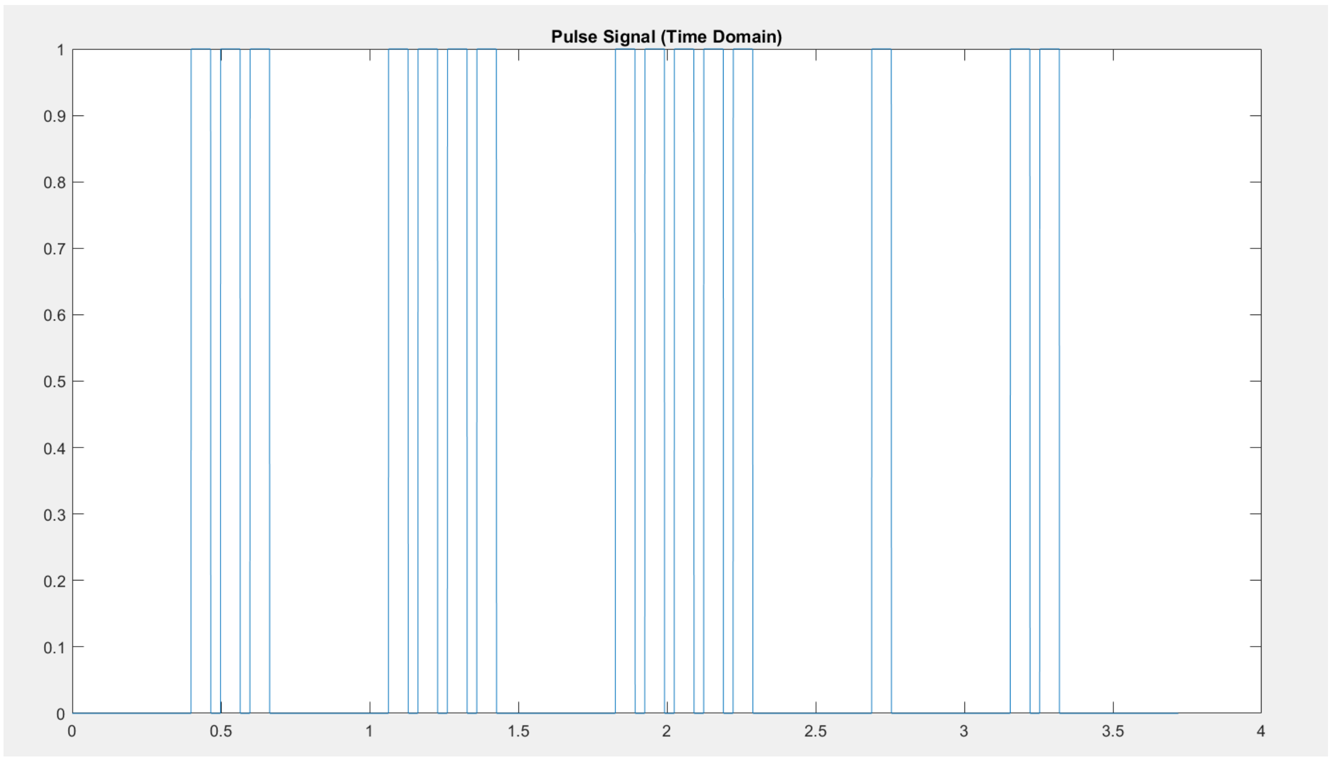
Choose Signaling Type: Pulse Signaling

Enter Phone Number: 34512

Add Noise

Select Filter: No Filter

Signal in time domain:

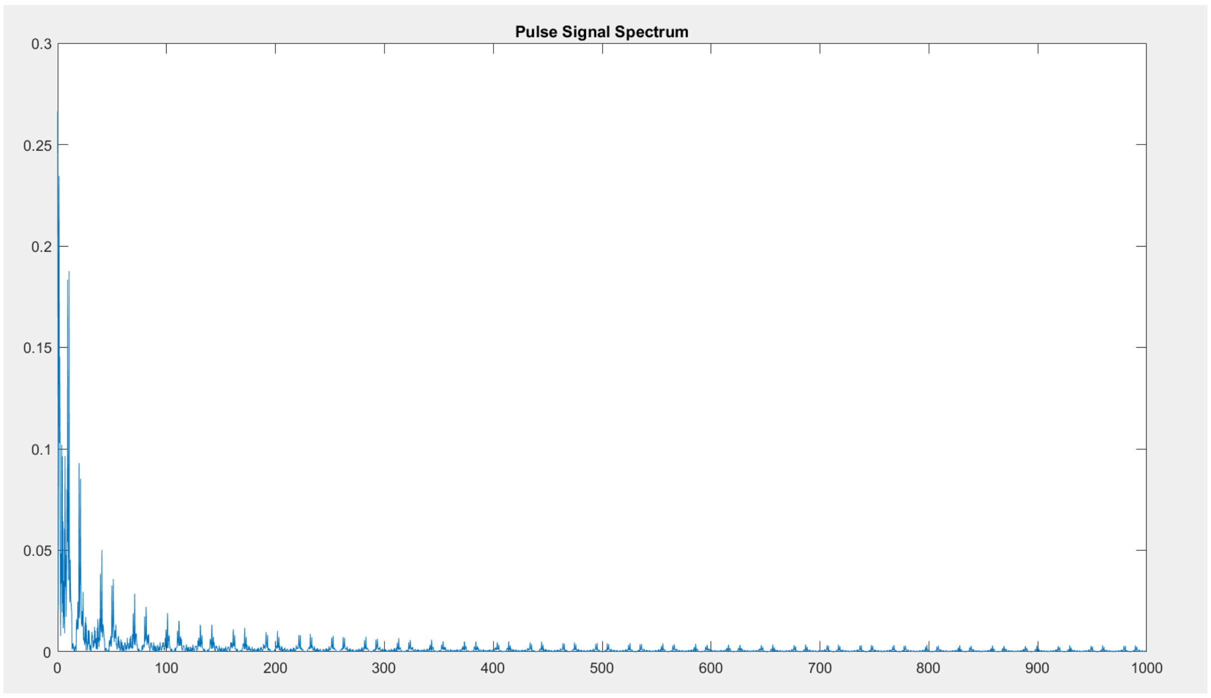


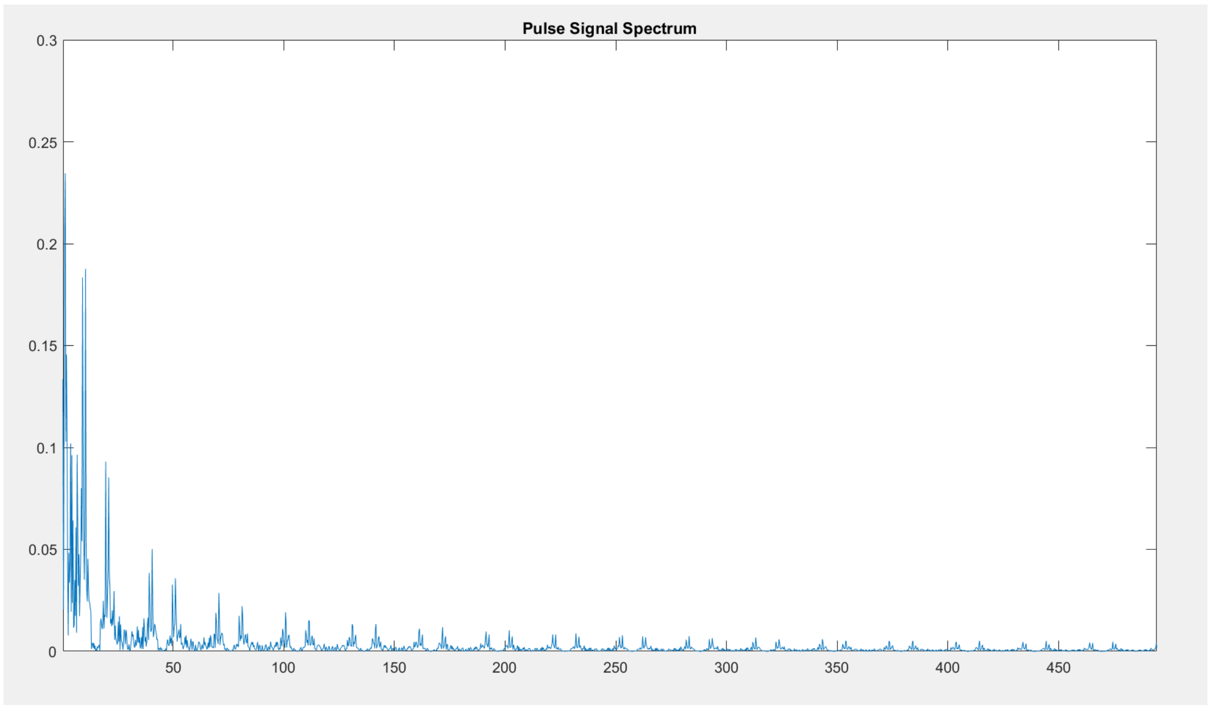
Each group of narrow vertical bars corresponds to the pulses for a digit, with the number of pulses matching the digit's value (e.g., 3 pulses for '3', 4 for '4').

Between each digit, there is a longer pause, separating them.

The sharp transitions between 1 and 0 simulate the "make" and "break" of a telephone line loop used in rotary dialing.

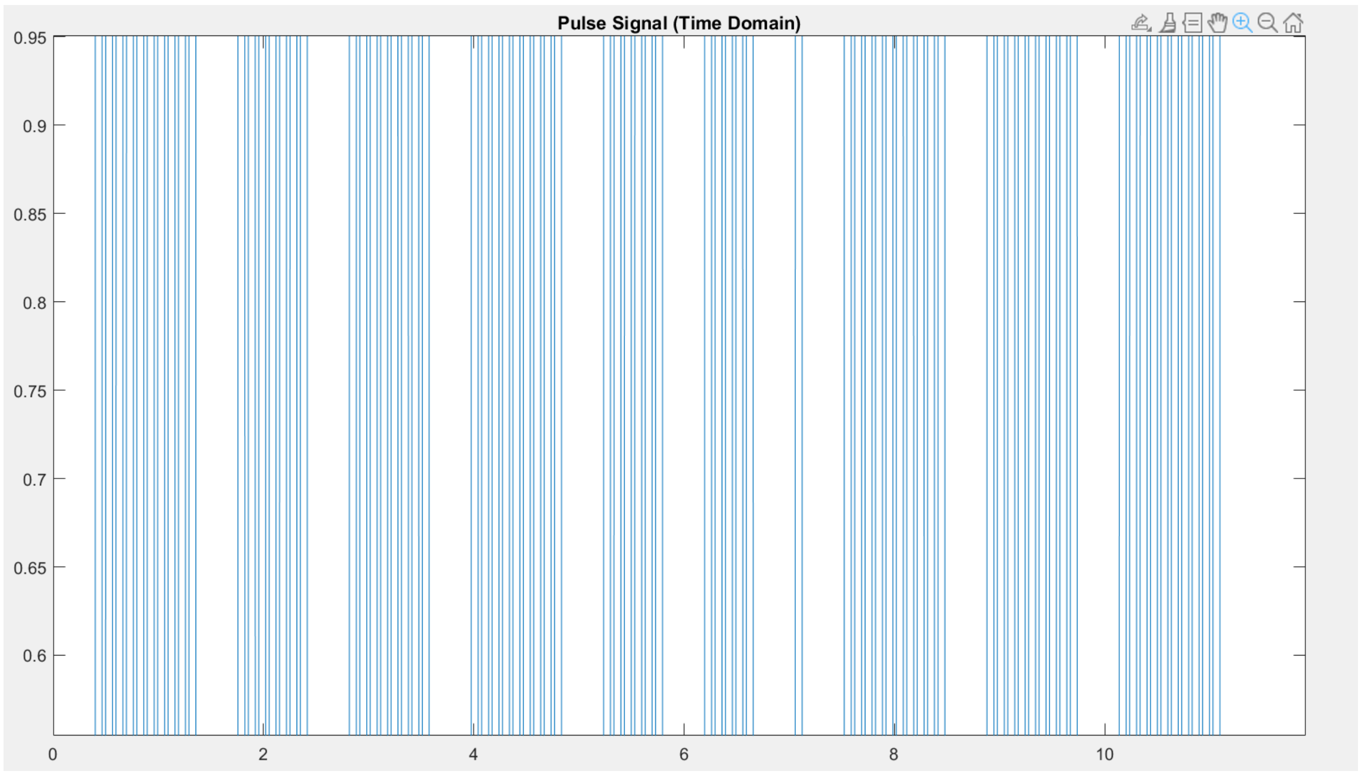
Frequency spectrum:

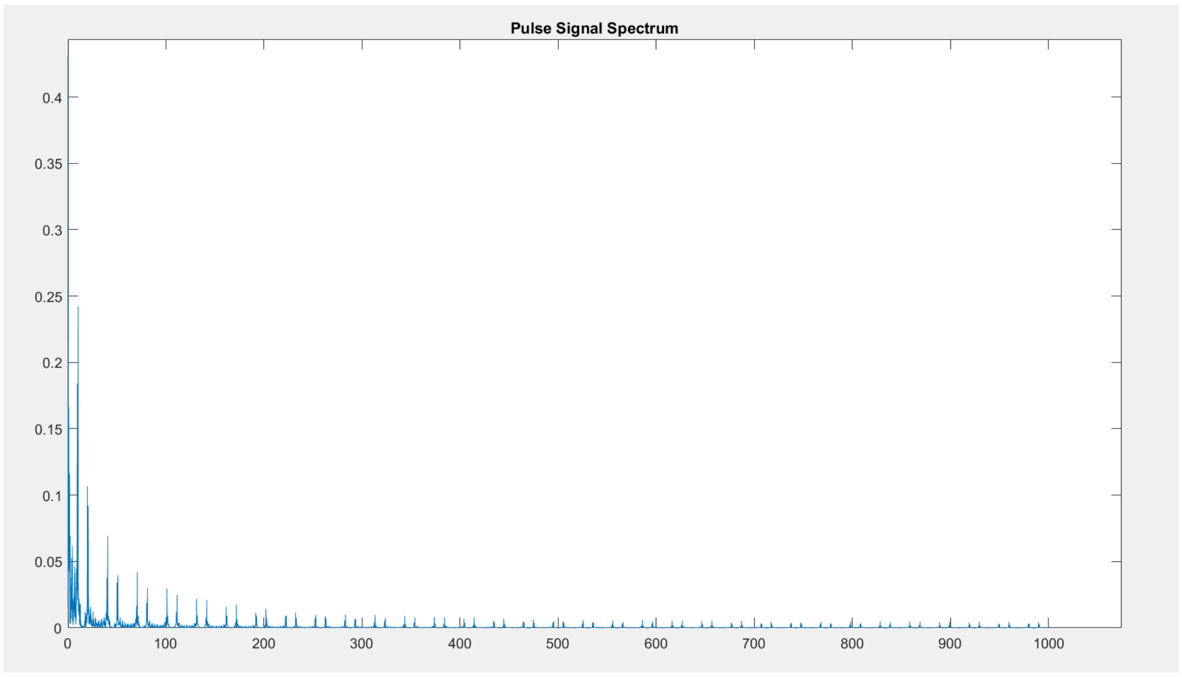




This spectrum shows that most of the energy in the pulse signal is at low frequencies, with a big peak near 0 Hz. That's because pulse dialing uses sharp on/off changes (1s and 0s).

The same can be said for any numbers introduced. Here is one more example for the phone number .


Pulse Signal (Time Domain)


Pulse Signal Spectrum