```
% Meta 2
function [medianAmpSpectrumMat, Q25AmpSpectrumMat, Q75AmpSpectrumMat,
meanAmpSpectrumMat, everyAmpSpectrumMat] = digitsAmpSpectrums(audioSignals,
windowType)
    % Each index of audioSignals contains the audio signals of each one of
the digits from that sample

    freqNum = 3000; % number of frequencies to plot
    medianAmpSpectrumMat = zeros(freqNum, 10); % matrix to store the median
amplitude spectrum of each digit
    Q25AmpSpectrumMat = zeros(freqNum, 10); % matrix to store the 25th
percentile amplitude spectrum of each digit
    Q75AmpSpectrumMat = zeros(freqNum, 10); % matrix to store the 75th
percentile amplitude spectrum of each digit
    meanAmpSpectrumMat = zeros(freqNum, 10); % matrix to store the mean
amplitude spectrum of each digit

    everyAmpSpectrumMat = cell(10, 1); % matrix to store the amplitude
spectrum matrix of each digit

    figure;
    annotation('textbox', [0 0.9 1 0.1], 'String', ['Median, Q25 and
Q75 Amplitude Spectrum of each digit, with window type ', windowType],
'EdgeColor', 'none', 'HorizontalAlignment', 'center', 'FontSize', 15);
    for digit = 1:10 % for each digit
        ampSpectrumMat = zeros(freqNum, 50);
        for i = 1:50 % for each sample

            if strcmp(windowType, 'hamming')
                windowedSignal = audioSignals{i}{digit} .*
hamming(length(audioSignals{i}{digit})); % apply hamming window
            elseif strcmp(windowType, 'hanning')
                windowedSignal = audioSignals{i}{digit} .*
hanning(length(audioSignals{i}{digit})); % apply hanning window
            elseif strcmp(windowType, 'blackman')
                windowedSignal = audioSignals{i}{digit} .*
blackman(length(audioSignals{i}{digit})); % apply blackman window
            else
                windowedSignal = audioSignals{i}{digit}; % do not apply any
window to the audio signal
            end

            %windowedSignal = windowedSignal - mean(windowedSignal); %
remove the mean of the signal
            curDigitFFT = fft(windowedSignal, 100000); % get the fourier
series coefficients of the current sample of the current digit
            curDigitFFT = curDigitFFT(1:floor(length(curDigitFFT)/2)); %
only take the first half of the data because the second half is symmetrical
```

```
            ampSpectrum = abs(curDigitFFT); % get the magnitude of the
complex fourier series coefficients
            ampSpectrum = ampSpectrum/length(curDigitFFT); % normalize by
the number of samples

            ampSpectrumMat(:, i) = ampSpectrum(1:freqNum); % store the
amplitude spectrum of the current sample of the current digit
        end

        everyAmpSpectrumMat{digit} = ampSpectrumMat;

        medianAmpSpectrumMat(:, digit) = median(ampSpectrumMat, 2); % get
the median amplitude spectrum of the digit
        Q25AmpSpectrumMat(:, digit) = quantile(ampSpectrumMat, 0.25, 2); %
get the 25th percentile
        Q75AmpSpectrumMat(:, digit) = quantile(ampSpectrumMat, 0.75, 2); %
get the 75th percentile
        meanAmpSpectrumMat(:, digit) = mean(ampSpectrumMat, 2); % get the
mean amplitude spectrum of the digit

        subplot(5, 2, digit);
        plot(medianAmpSpectrumMat(:, digit), 'Color', 'red');
        hold on;
        plot(Q25AmpSpectrumMat(:, digit), 'Color', 'green');
        plot(Q75AmpSpectrumMat(:, digit), 'Color', 'blue');
        hold off;
        title(digit - 1);
        xlabel('Frequency (Hz)');
        ylabel('Amplitude');
        legend('Median', 'Q25', 'Q75');
    end
end
```