The goal of this project is to learn more about short-time Fourier analysis (sliding-window DFT’s) by implementing a Matlab function and using it to analyze a “mystery” signal containing several variable-length tones. Note that this assignment is based on a project in “Computer-Based Exercises for Signal Processing Using Matlab 5” by McClellan et al.

1 Background

In class we have discussed the spectral analysis of signals containing one or more sinusoids. We examined the effects windowing and spectral sampling on the analysis. In particular we considered how each of the following parameters affected our ability to resolve closely-spaced sinusoids: window type, window length, DFT length.

In practical applications such as sonar, radar, and speech processing, we often encounter sinusoidal signals whose frequencies vary as a function of time. One method of analyzing these “non-stationary” signals consists of taking a series of windowed DFT’s of short sections of the signal. The result is called a short-time Fourier transform, which is one type of time-frequency distribution. Wavelet transforms and discrete cosine transforms are examples of other time-frequency distributions. In most applications, the magnitude of the short-time Fourier transform is the desired quantity (the phase may be computed, but often only the magnitude is plotted and analyzed). Plots of the STFT magnitude are typically referred to as spectrograms.

In this exercise, you will modify an existing Matlab function (provided on the course website) to create a flexible short-time Fourier spectral analysis function. You will use this function to analyze a signal that consists of a series of tones. One possible application of such a function is to use it to analyze touch-tone phone signals, where each digit is represented by a different tone (although this is not how a touch-tone phone works). For the purposes of this project, you may assume that the “phone” signal consists of a series of short-duration sinusoids of constant frequency. The duration of each sinusoidal pulse is not the same, but the length is typically between 15-20 milliseconds. Each of the 10 digits (0-9) is associated with a particular tone, as specified in Table 1 below. For example, the digit sequence 7-3-4-1-9 would consist of a 15-20 millisecond sinusoidal pulse at 3750 Hz, followed by one at 1750 Hz, followed by one at 2250 Hz, followed by one at 750 Hz, followed by one at 4750 Hz. The frequency of the pulses changes approximately every 15-20 milliseconds. This encoding scheme is similar to frequency shift keying (FSK), except that the digits are not constant length in time. The signal will have a constant amplitude. The sample rate is fixed at 10 kHz for all the examples used in this project.

Table 1: Digits and their associated sinusoidal frequencies
2 Exercises

This project consists of three exercises. In the first exercise you will implement a flexible short-time Fourier analysis program. In the second exercise you will examine “phone” signals generated by the function `tonegen` that is available on the course website. Finally, in the third exercise, you will analyze “phone” signals using your short-time Fourier program.

2.1 Short-Time Fourier Analysis

The function `stft_rect` given below (and on the course website) computes a series of DFT’s of rectangularly-windowed sections of the input signal vector `x`. The parameter `N` is the window length (in samples) and the parameter `nskip` is the number of samples the window is moved ahead for each consecutive section. For example, if `nskip`=1 then the first and second sections completely overlap except for one point. If `nskip` is equal to the window length, then consecutive sections do not overlap. If `nskip`>`N`, then the analysis skips some points entirely. The parameter `K` is the number points in each DFT. `K` must be greater than or equal to `N` to avoid loss of information.

```matlab
function Xk=stft_rect(x,N,nskip,K)
    % STFT_RECT Computes a short-time Fourier transform of the vector x
    % using a rectangular window. The user specifies the window
    % length, DFT length, and nskip parameter
    %
    % Usage: Xk=stft_rect(x,N,nskip,K)
    %
    % Variables:
    %    x = input vector
    %    N = rectangular window length
    %    nskip = # of samples window is moved for each consecutive section
    %    K = DFT length (choose power of 2 for faster calculation)
    %
    % Xk = matrix of DFT’s: #rows=K; #columns depends on length of input
    %
    % File : stft_rect.m
    % Author : Kathleen E. Wage
    % Date : November 20, 2001
    %---------------------------------------------------------------
    %%% Determine the number of sections
    xlen=length(x);
    nsections=length(1:nskip:xlen); % # of sections
    n=1:nskip:xlen; % vector of sample indexes: 1st pt of each section

    %%% Make x a column vector and pad with N zeros
    %%% Zero-padding done in case length x not equal to multiple of nsections
    x=[x(:); zeros(N,1)];
```
%% Loop through and compute FFT’s
Xk=zeros(K,nsections); % Initialize output with zeros
for ind=1:nsections
    nlo=1+(ind-1)*nskip; % first index of section
    nhi=nlo+N-1; % last index of section
    xsec=x(nlo:nhi); % section of x
    Xk(:,ind)=fft(xsec); % take FFT
end
return

For this exercise, create a more general short-time Fourier analysis function called \texttt{stft\_gen}. This function should

- allow the user to select the type of window that is used in the analysis
- return a vector of times and a vector of frequencies for plotting against. (Hint: To generate the vector of frequencies in Hz, the user will probably need to pass the sampling frequency as a parameter to the function.)

You may use the code in \texttt{stft\_rect} as a starting point for creating your function, or you may write a completely new function of your own. You may also use any code distributed in class as a reference to help you modify/create the \texttt{stft\_gen} function.

Generate some simple test signals of your own to verify that your function is working properly. For example, you might create a test signal by concatenating two vectors that contain different frequency sinusoids. Display the magnitude of the short-time Fourier transform output and verify that you can detect the presence of both sinusoids at the correct frequencies. Matrix data such as your STFT results may be displayed using Matlab commands such as \texttt{image}, \texttt{imagesc} (for scaled images), or \texttt{contour}. Note: You will probably want to use a log scale for your plots, e.g., \texttt{imagesc(20*log10(abs(Xk)))}. If you are using \texttt{image} or \texttt{imagesc} you may want to specify a different colormap than the default. Try \texttt{help colormap} and \texttt{help graph3d} for more information.

### 2.2 Tone Generator

The mystery signal you will analyze in Section 2.3 was generated by a Matlab function called \texttt{tonegen}. The purpose of this exercise is to familiarize you with the type of signals that \texttt{tonegen} produces.

The mystery signal is sampled at 10 kHz and is known to contain three components:

1. **Frequency-coded digits** The frequency-coded digits constitute the desired signal, \textit{i.e.}, the “phone signal”. This part of the signal is generated as described in Section 1. To avoid instantaneous jumps in frequency, \texttt{tonegen} lowpass filters the step frequency changes, which smoothes the frequency versus time characteristic. Note that the coded digits do not start at \(n = 0\), so one part of the problem in analyzing the mystery signal is to determine where the coded digit sequence starts.

2. **Time-varying interference signal** The second component added by \texttt{tonegen} is an interference that is much stronger than the frequency-coded signal. This interference is a sinusoid whose frequency is changing continuously with time. Its peak amplitude is about ten times stronger than
the coded signal. You are not required to eliminate the interference, but you should be able to
determine some of its parameters from your time-frequency plot.

3. Additive noise The third component added by tonegen is noise term. This component is zero-
mean Gaussian white noise with a variance that is equal to 3% of the amplitude of the interference
signal. Thus, there is some noise present, but the signal-to-noise ratio is favorable for processing
the coded signal. This noise component has been added to make the problem a bit more realistic,
but should not be a major factor in designing your processing scheme.

The function tonegen takes as input a five-element vector and produces an output signal that contains
the frequency-coded version of this input vector plus the interference and noise. As a precaution, there
is a second output, which is the actual five-element vector of digits used (in case the input vector did not
contain integers from 0 to 9). If you supply a second input argument, i.e., tonegen(digits, scale),
the value of scale is interpreted as a multiplier that will scale the additive interference and noise.
Therefore, you can create a signal free of noise and interference by using tonegen(digits, 0).
This will allow you to see what the coded signal looks like by itself. Note that with just one scalar input
argument (tonegen(0)) the function will generate a set of “random” digits based on the present clock
time.

If scale is not equal to 0, then tonegen always adds the interference signal in interfere.mat
to the coded digits it generates. (In other words, the interference is always the same even if the digits
change.) If you want to generate a new interference signal, you may use the function genint. You
may pass your own interference vector to tonegen as the third input.

Matlab m-files for tonegen and the associated function genint may be found on the course
website, along with a data file, interfere.mat, which contains the interference signal. For this exercise
experiment with tonegen by specifying different vectors of digits and different scale factors. Observe
how the time signals vary as a result. Include a few plots of the time signals produced by tonegen
along with a brief explanation in your writeup.

2.3 STFT Analysis of Signals Generated by tonegen

2.3.1 Four Test Cases

Use the function stft_gen you wrote in the first exercise to analyze the following signals. Your
writeup should include plots of the STFT results for each case, along with a description of the STFT
analysis parameters (window type, window length, DFT size, nskip) used in the analysis. Note
whether you can detect the presence of the 5 digits correctly from the STFT or not.

(a) [digits]=[9 8 1 2 3];
(b) [digits]=[4 3 4 7 6];
(c) [digits] = last 5 digits of your phone number
(d) The mystery signal contained in the file tonemyst.mat found on the course website. You can
load this signal into memory by typing load tonemyst.mat.
2.3.2 Selection of parameters

Write a short memo explaining the issues involved in designing a system to detect frequency-coded digits using the short-time Fourier transform. Discuss how the window, $n_{skip}$, and DFT size parameters should be chosen. In particular discuss the issue of computational complexity. Indicate how the parameters can be chosen to minimize computation. Explain your reasoning clearly and support your conclusions with plots or other calculations.