ECE 3780 Laboratory 5

The Continuous-Time Fourier Transform Spectral Analysis of Speech Signals Purpose

This lab investigates how to use, interpret and compute the continuous-time Fourier transform (FT) for analysis and synthesis of continuous-time aperiodic signals. In particular the spectral analysis of speech signals is to be conducted. This includes the use of the FFT (fast Fourier transform algorithm) and some properties of the Fourier spectra.

I The continuous-Time Fourier Transform

A Theory

The continuous-time Fourier transform:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(w)e^{jwt}dw \tag{1}$$

$$X(w) = \int_{-\infty}^{\infty} x(t)e^{-jwt}dt \tag{2}$$

extends the continuous-time Fourier series to allow frequency-domain analysis of aperiodic (as well as periodic) continuous-time signals. Equation (2) transforms a signal from the time domain to the frequency domain represented by the complex valued function (Fourier Transform) $X(\mathrm{w})$. On the other hand formula (1) describes how to reconstruct a signal $x(t)$ from its Fourier spectra $X(\mathrm{w})$.

B Computation of Fourier Spectra

In practice the signal cannot be recorded for all values of t. If the signal $x(t)$ is nonzero only over the finite time interval $[0,T]$, then the Fourier transform can be approximated by

$$T_s\sum_{n=0}^{N-1} x(nT_s)\exp(-jwnT_s) \tag{3}$$

where T_s is the sampling rate and$sT = NT_s$.

Since we are interested in digital computation of (3), we restrict w to the discrete set of values: $\left\{0, \frac{2\pi}{T}, 2\frac{2\pi}{T}, \ldots, (n-1)\frac{2\pi}{T}\right\}$. Thus setting $w = k\frac{2\pi}{T} = \frac{2\pi}{NT_s}$ in (3), where k takes on the discrete values $\{0,1,\ldots, N\text{-}1\}$ we obtain

$$X[k] = T_s\sum_{n=0}^{N-1} x[n]\exp\left(-jk\frac{2\pi}{N}n\right) \tag{4}$$

Here $x[n] = x(nT_s)$ and $X[k]$ is the approximation of $X(w)$ at the frequency $k2\pi/T$.

It should be observed that $X[k]$ is periodic with period N. By replacing k by $k+N$ in (4), we see that this is indeed the casse.

The original time-domain sequence $\{x[n] = x(nT_s), n = 0, 1, \ldots, N-1\}$ is obtained from the sequence of frequency-domain $\{X[k], k = 0, 1, \ldots, N-1\}$ by the inverse relationship

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\exp\left(jk\frac{2\pi}{N}n\right) \tag{5}$$

Formulas (4) and (5) can be easily implemented in Matlab. In fact, if N samples $\{x[n] = x(nT_s), n = 0, 1, \ldots, N-1\}$ are stored in vector **x** then the Matlab function

$$\mathbf{F} = fft(\mathbf{x}) \tag{6}$$

calculates the sum in (4).

For reasons of computational efficiency, *fft* returns the positive frequency samples before the negative frequency samples. Hence

$$Xp = T_s F\left(1:\frac{N:}{2}+1\right) \tag{7}$$

Extracts only the positive frequency components of **F** in (6) and multiplies them by sampling period to estimate X(w). Furtheremore

$$W = (2\pi/T_s)(0:N/2)/N$$

Creates the contimuos frequency axis, which starts at zero and ends at the Nyquist frequency $w_s/2$, where $w_s = 2\pi/T_s$ is the sampling frfequency in rad/sec.
Finally the command *plot*(**w**,abs(***Xp***)) and *plot*(**w**, angle(***Xp***)) give the magnitude and phase spectra of the signal, respectively.
The function *ifft* is the inverse operation of *fft*. i.e. it implements formula (5).

Part I Zero *padding*

In Matlab, the fft command computes the DFT using the formula given above when the length of the signal is not a power of two, and it uses a faster algorithm otherwise. The numerical results are the same.
FFT(X,N) is the N-point FFT, padded with zeros if X has less than N points and truncated if it has more.

Problem 1:

Create a length 32, 243 Hz sinusoidal signal in Matlab using a sampling frequency of 1000 Hz using a time index of n = 0,1, …, 31. Plot the spectrum by using stem(abs(fft(x))) where x is the signal of interest.

Question: From the position of the peak of this plot, what is the estimated frequency?

As you notice, the estimated frequency is not the real frequency. In order to be more precise, zero padding can be used so that with more samples in the frequency spectrum one can be more accurate in this estimation.

Question: Use fft(x,256), plot the spectrum and finding the maximum, estimate the frequency.

Note that the sidelobes of the last plot show better the sinc function structure in the frequency domain of the rectangular window that is used when only a finite amount of samples are available.

Part II *Time domain windowing*

Sidelobes occurs when a finite amount of samples are available. The signal is multiplied by a rectangular window in the time domain and its Fourier transform is then the convolution in frequency of the spectrum of the signal convolved with the spectrum of the rectangular window. Sidelobes can bury low intensity components as it is shown next.

Problem 2

Create a multicomponent length 64 signal that consists of 3 complex exponentials of the form x = A*exp(i*2*pi*fo*n) + B*exp(i*2*pi*f1*n) + C*exp(i*2*pi*f2*n) using Matlab notation.

The amplitudes and the normalized frequencies are A=1, B=0.1, C=1e-5, fo=0.25, f1=0.28, and f2=–0.2513.

Plot a normalized spectrum in decibels using
xf = abs(fft(x)).;
xf = xf/max(xf);
xf = 20*log10(xf);

Question: Can you see the small amplitude component?

In order to alleviate this problem, use a Hanning window. In Matlab, you can create this window by using w = Hanning(64). Multiply the signal with this window. Note that you need to multiply each element, that is, x(1)*w(1), x(2)*w(2) and so on, it is not a vector multiplication. Plot the normalized spectrum of x*w in decibels too.

Questions: Can you see the small component? Is there a serious consequence from doing this?

Note: In order to help you answer the last question, plot the length 64 Hanning window and its spectrum a as well as the spectrum of the length 64 rectangular window.

Part III *Filtering*

You can define a time domain filter $h(n)$ and implement filtering by using convolution. In the frequency domain, this operation corresponds to a simple multiplication. Using the FFT, this approach is usually faster and easier to implement.

Problem 3:

Create sinusoidal signal x1 with a frequency of 500 Hz using a sampling frequency of 2500 Hz. Create a second sinusoidal signal x2 using now a frequency of 1000 Hz with the same sampling frequency. Listen to both signals using the sound command in Matlab. Use 1000 samples.
Form a third signal x3 by adding the two signals and listen to it.
Plot the spectrum of x3 and define a band pass binary filter in the frequency domain H(k) that will allow you to isolate the tone at 500 Hz.
H(k) = 1 for all frequency indexes k where the desired signal is, and it is zero otherwise.

Question: Is your filter going to introduce distortion? That is, are you modifying the output phase in a nonlinear way? Justify your answer.

Listen to the filtered output. You will need to reconstruct the signal using the inverse command in Matlab ifft.

Compare the original x1 samples with the filtered x3 samples. Note: The ifft command will return a complex signal. Use only the real part of it, the imaginary part is close to zero and these values are introduced due to truncations during the computation of the inverse fft.