Lab 9 Information

- The material in this section is informational. Please read through the section as it helps you work on the lab exercises in the next section. There may be code examples in this informational section. You are welcome to copy-and-paste them to MATLAB to run the code, but no submission is needed on any test run.

Fast Calculation of DFT in MATLAB

The MATLAB function `fft` implements the Fast Fourier Transform (FFT) algorithm to calculate the Discrete Fourier Transform (DFT) of a finite-length discrete-time signal. To do this lab, you do not need to understand how the FFT algorithm works. We will talk more about that in class. It suffices to know that the `fft` function calculates the DFT of a finite-length signal.

Let `x` be a signal vector of length N in MATLAB. It represents an $N-$length discrete-time signal. The following line of MATLAB code calculates the N-point DFT of `x` and put the N-point DFT coefficients in the vector `X`:

```
X = fft(x);
```

The first element in `X` is the DFT coefficient X_0, the second element is X_1, and so on. The last element gives X_{N-1}. You may also specify the size of the DFT to be calculated by the function `fft`. For example,

```
M = 256;
X = fft(x, M);
```

calculates the 256-point DFT of `x`, regardless of the length of the signal vector `x`. If the length of `x` is longer than 256, `fft` will truncate `x` to keep the first 256 elements and then calculate the 256-point DFT of the truncated vector. On the other hand, if the length of `x` is smaller than 256, `fft` will pad zeros to the end of `x` to make the resulting vector of length 256 and then calculate the 256-point DFT of zero-padded vector.

Typically, one would choose the size of the FFT to be a power of 2 larger than the length of the signal vector `x` in order to obtain the most efficient algorithm for the calculation of the DFT of `x`.

Fast Calculation of DTFT in MATLAB

Recall that the M-point DFT coefficients X_k's of a signal of length N are the frequency domain samples of the signal's DTFT $X(e^{j\hat{\omega}})$, as long as $M \geq N$. Specifically,

$$X_k = X(e^{j\hat{\omega}})\Big|_{\hat{\omega}=\frac{2\pi k}{M}} \tag{1}$$

for $k = 0, 1, \ldots, M-1$. Hence, we may use the MATLAB function `fft` to efficiently calculate the values of the DTFT of a signal at the normalized radian frequency sample points $\hat{\omega} = \frac{2\pi k}{M}$ for $k = 0, 1, \ldots, M-1$. Note that for $\frac{M}{2} \leq k \leq M-1$, the frequency sample point $\pi \leq \frac{2\pi k}{M} < 2\pi$, which is outside of our conventional/canonical range of $[-\pi, \pi)$ for expressing the normalized radian frequency. Nevertheless by subtracting 2π from the frequency sample point $\frac{2\pi k}{M}$, we have (why?)

$$X_k = X(e^{j\hat{\omega}})\Big|_{\hat{\omega}=\frac{2\pi k}{M}} = X(e^{j\hat{\omega}})\Big|_{\hat{\omega}=-\frac{2\pi(M-k)}{M}} \tag{2}$$

for $\frac{M}{2} \leq k \leq M-1$. Thus, the DFT coefficients X_k, for $\frac{M}{2} \leq k \leq M-1$ (i.e., the elements in the second half of the FFT vector `X` obtained above), are the frequency samples of the DTFT $X(e^{j\hat{\omega}})$ over the range of negative normalized radian frequency from $-\pi$ to 0. When we plot the magnitude (or the phase) of the DTFT, we usually plot from normalized radian frequency $-\pi$ to π along the horizontal axis. The MATLAB function `fftshift` wraps the second half of a vector to the first half of the vector. For example,

```
>> fftshift([1 2 3 4])

ans =

     3     4     1     2

>> fftshift([1 2 3 4 5])

ans =

     4     5     1     2     3
```

We may use this function to wrap the second half of the FFT vector `X` to the first half to move all the negative frequency samples of the DTFT to the beginning of the vector for plotting from normalized radian frequency $-\pi$ to π.

Putting all these together in an example, the following piece of code uses the function `dtft` that we developed in the previous labs to calculate the DTFT of the signal vector `x`, and uses the built-in MATLAB function `fft` to calculate the DFT of `x` for comparison:

```
N=65000; % signal length
M=2^16; % FFT size (M >= N)
n=0:N-1; % discrete time vector
x = (0.8.^n).*cos(0.5*pi*n); % signal

% Calculate DFT and measure calculation time
tic;
X_DFT = fft(x,M);
toc;

% Calculate DTFT and measure calculation time
w = -pi:2*pi/M:pi-2*pi/M; % freq vector
tic;
X_DTFT = dtft(x,w);
toc;

% Plot magnitude and phase of DTFT (blue solid line) and DFT (red x)
figure;
subplot(2, 1, 1)
plot(w/pi, fftshift(abs(X_DFT)), 'rx');
hold on;
plot(w/pi, abs(X_DTFT), 'b-');
hold off;
grid on;
```

```
title('Magnitude')
xlabel('Normalized Radian Frequency (\times \pi rad/sample)');
ylabel('Amplitude');
legend('DFT','DTFT');
subplot(2, 1, 2)
plot(w/pi, fftshift(angle(X_DFT)/pi), 'rx');
hold on;
plot(w/pi, angle(X_DTFT)/pi, 'b-');
hold off;
grid on;
title('Phase')
xlabel('Normalized Radian Frequency (\times \pi rad/sample)');
ylabel('Phase (\times \pi rad)');
legend('DFT','DTFT');
```

The elements in the frequency vector `w` are chosen to coincide with the frequency sample points of the DFT given in (1). Run the code in MATLAB and you should observe the DFT finely samples the DTFT of the signal `x` in frequency. In addition, the functions `tic` and `toc` measure the time elapsed between their application. We may use them to roughly measure the run time of `dtft` and `fft` as in the code above. Running the code on my laptop, it takes 3.114 ms to complete the DFT calculation using `fft` and 47.41 s to complete the DTFT calculation using `dtft`. Thus, `fft` provides a speed-up of about 15000 times for the signal vector of length 65000!!

Lab 9 Exercises

- Unless stated otherwise, you must submit your solutions to all the lab exercises in this section.
- Your laboratory solutions should be submitted on Canvas as a single PDF. The simplest way is to put your codes and answers for all the lab exercises in a single MATLAB Publisher script and use `%%` to separate the codes and answers for different exercises into different sections as described in the information section of Lab 1.

Exercise 9.1: *(Effects of DFT size)*

This exercise investigates the effects of using DFTs of different sizes to calculate DTFT of a finite-length discrete-time signal. Consider the following discrete-time signal of length 100:

$$x[n] = [0.5 + \cos((\pi/30)n) + \cos((\pi/5)n) + \cos(\pi n + 2\pi/3)]\,[u[n] - u[n-100]]\,. \tag{3}$$

(a) Generate the signal $x[n]$ in MATLAB and store it in the vector `x`. Use your `dtft` function to calculate the DTFT of $x[n]$. Plot the magnitude and phase of the DTFT in solid blue lines.
Hint: As before, you'll need to generate a frequency vector `w` to calculate the DTFT. The elements of your frequency vector do not need to coincide with the frequency sample points of any DFT as in the information section above. However, you may want to generate a frequency vector with a fine step size (e.g., $\pi/1000$) to obtain smooth magnitude and phase plots for the DTFT of $x[n]$.

(b) Use the MATLAB function `fft` to calculate the 128-point DFT of $x[n]$. Plot magnitude and phase of the DFT in red crosses. Overlay your plots with the corresponding DTFT plots in the same figure. Compare the DFT coefficients with the DTFT values. Are the DFT coefficients frequency samples of the DTFT? If so, what is the sample frequency of the kth DFT coefficient?
Hint: To overlay the (magnitude and phase) plots of the DFT and DTFT properly in the same figure, you will also need a frequency vector to specify the horizontal axis of the magnitude and phase plots. However, this frequency vector may not be the same one that you use to generate the DTFT in (a). Read the piece of code in the information section carefully to figure out what frequency vector you need here.

(c) Repeat (b) but instead calculate the 512-point DFT of $x[n]$. Compare the result obtained using this DFT with that using the 128-point DFT in (b). Can you extrapolate what will happen if one uses an even larger-size DFT?

(d) Repeat (b) but instead calculate the 64-point DFT of $x[n]$. Compare the result obtained using this DFT with that using the 128-point DFT in (b). Explain any differences.

Exercise 9.2: *(Frequency-domain analysis using FFT)*

This exercise shows you how to perform frequency domain analysis using FFT and inverse FFT.

(a) Use `filterDesigner` to design an FIR Equiripple highpass filter with the following specifications:

- **Frequency Specifications: Units: Normalized**,
- **wstop** = 0.5,
- **wpass** = 0.6,
- **Astop** = 80 dB, and
- **Apass** = 0.5 dB.

Export the impulse response of the filter to the vector `h`. Apply the filter to the signal in (3) using the MATLAB function `conv`. Save the output to the vector `yconv` and plot it using `stem`. Explain plot of `yconv` based on the characteristics of the highpass filter.

(b) Perform the following frequency-domain analysis steps to obtain the filter output:

(1) Calculate the 256-point DFTs of `x` and `h` using the MATLAB function `fft`.

(2) Element-wise multiply the two DFTs to obtain the output DFT `Y256`.

(3) Calculate the 256-point inverse DFT of `Y256` to obtain the time-domain output signal `y256` using the MATLAB function `ifft`.

Use `stem` to plot the output vector `y256`. Compare the plots of `y256` and `yconv`. Does the frequency-domain analysis above give the same output vector `yconv` obtained using time-domain analysis?

(c) Repeat (b) using 128-point DFT and inverse DFT. Save the frequency-domain analysis output to the vector `y128`. Is `y128` the same as `yconv`? Explain why or why not.