



**0909.351.01**  
**DIGITAL SIGNAL PROCESSING – LAB 06**  
**DUE MARCH 27, 2009**  
**DTFT & DFT**

You may work in teams of up to two members, and then submit one report. If you do, make sure that every one in the group understands the solution of every question. However, if you are interested in DSP and would like to make sure that you get the most out of this course, I recommend that you work on your own. In order to further motivate you, here is an incentive: If you work alone, you will get a 15% bonus on the correct answers.

1. A 20-point sequence is given as follows:  $x[n]=[0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1\ 0]$ .
  - a. Determine the DFT,  $X[k]$  of this sequence using the `fft()` function. Plot the magnitude and phase responses using the stem command.
  - b. Now plot the magnitude and phase of the DTFT  $X(\omega)$ , using the code you wrote last week.
  - c. Verify that the DFT you compute in part (a) is indeed the sampled version of  $X(\omega)$  you computed in part (b). You can do this by plotting both on the same figure by using the `hold()` function
  - d. Is it possible to reconstruct the DTFT  $X(\omega)$  from  $X[k]$ ? If so, under what conditions. Write a piece of code to demonstrate this reconstruction property. Attempt to reconstruct the DTFT – using the formula given in the lecture slides – using several scenarios, some satisfying the condition stated above, and some that do not.
2. As it should be obvious to you by now, taking the Fourier transform does a lot more than just providing the spectral content of a signal. Several procedures that are fundamental to DSP, such as convolution, shifting, etc. can be computed in the frequency domain much more efficiently than in time domain.
  - a. Write a program that computes “linear convolution”, using the frequency domain approach. Evaluate your algorithm on a sample convolution, such as  $x=[3\ 6\ 9\ 12\ 9\ 6\ 3]$  and  $h=[1\ 0\ -2\ -2\ 0\ 1]$ ;
  - b. Write a program that computes “circular convolution”, using the frequency domain. Evaluate your algorithm on the same two sequences given above. Compute the circular convolution of these sequences manually to check your answer.
  - c. Write a program that computes “circular shift” using the frequency domain approach. Your program’s preamble should look like this:

```
function xs=circshift(x,m);
% Circular shift of m samples wrt to size N in sequence x - computed in
% frequency domain
% xs: circularly shifted sequence
% x: input sequence
% m: sample shift
```

- d. Under what condition circular and linear convolution are identical? Write a program that demonstrates this condition on the two sample sequences given above.
3. One of the practical issues in DSP is processing of really long or streaming data. If for example, you have a data that is 100,000 points long, the convolution operation (which really is filtering)

will take a very long time, regardless how efficient it is implemented. One way to make it more efficient is to partition the signal into smaller, say 1000, point blocks and then process them separately. It turns out however, you cannot just concatenate the individually processed blocks, as this introduces so-called boundary effects, where the block-processed signal and the convolution of the entire signal differ at the boundaries. The same issue applies if you would like to process streaming data, where the data comes in continuously. You can import the data at regular intervals, say 1000 points at a time, compute the convolution for each and then combine the pieces. But again, you cannot just concatenate these blocks, due to boundary effects. There is a solution, in fact two solutions, however, to address this issue. These solutions are known as the **overlap add** (already seen in class) and **overlap save** methods. Both methods are discussed extensively in the literature. Search the internet (also in Chapter 5 of Mitra), find out more about these techniques and implement one of them for a general case. The inputs to your function should be the original signal (length M), the filter (length N) and the block length, M1. The output is the filtered signal calculated through block convolution, as we did in class. Then, try it on a sample signal, again as we did in class. Specifically,

- Using **conv ()** function, compute the output sequence of the filter:  $y[n]=x[n]*h[n]$
- Use the overlap add or overlap save method of block convolution. Use blocks of M1 points each.
- Use FFT to implement the convolution
- Compare the three techniques in terms of computation time.

**BONUS:** For a real world challenge, try the following for M1=1000.

$$x[n] = \sum_{m=0}^{10} 0.9^m \cos 0.1\pi mn, \quad 0 \leq n \leq 1,000,000$$

$$h[n] = \cos 0.5\pi n, \quad 0 \leq n \leq 100.$$

- And for the grand finale: How fast **fft ()** really is compared to regular DFT? Create a sinusoid that has the three spectral components of 50, 100 and 250 Hz. Sample this signal at 1000 Hz. Determine the duration of the signal so that you end up with 1024 points. We will test various implementations of DFT on this three frequency test signal. Also read the help files on **tic** and **toc**, which you will use to determine how long each routine takes to compute.
  - Write a code that implements the analysis equation of the DFT. Go ahead and use a **for** loop and the **sum ()** command. This would be the least efficient implementation of the DFT. Run your code on the test signal and verify that the result is correct by comparing your result to that of the **fft ()** function.
  - If you compute the DFT at the same number of points as the length of the signal, then this can be done in one line, taking advantage of Matlab's vectorization, without using a for loop or the **sum ()** command. This implementation would be the most efficient implementation of the analysis equation. Verify your code on the test signal.
  - Finally, you can compute DFT using **fft ()**.
  - Compare the run times for three versions of the DFT computations. Discuss your observation.

## **BONUS QUESTION**

5. One of the most commonly used operations in DSP is decimation and interpolation. Decimation by  $M$  means creating a new signal by retaining every  $M^{\text{th}}$  sample of the original signal, whereas interpolation by  $M$  means inserting  $M-1$  zeros between every adjacent sample. These operations effectively change the sampling rate of the signal, and they have some interesting effects on the spectrum of the signal.

In this bonus question, I would like you to investigate the time and frequency effects of these two operations. In general, I am interested in the following:

- What happens to a signal's spectrum when it gets decimated or interpolated?
- Can the spectrum of the original signal be recovered in either case? If so, how?

To find out the answers, first read about this topic (Chapter 7 in your text, internet, etc.) and then attempt the following exercises. Write a report detailing your results and analysis.

- a. Generate a 10 kHz sinusoid sampled at 100kHz. Plot four cycles of this signal, along with its spectrum (magnitude and phase) in the  $[-\pi, \pi]$  interval. Use the **fft()** and **fftshift()** commands. Try once with the horizontal axis representing the discrete angular frequencies, and once with linear frequencies.
- b. Decimate the signal by a factor of 2. Plot the time domain signal, as well as the spectrum of the decimated signal in the  $[-\pi, \pi]$  interval. What does the spectrum look like? What frequency in Hz does  $\pi$  represent now?
- c. Upsample the original signal by a factor of 3 and repeat (b).
- d. Use the **fir1()** command to design an appropriate low pass filter to filter the upsampled signal in order to recover the original spectrum. Plot the magnitude spectrum of the designed filter. Filter the upsampled signal with this filter and plot the output in time and frequency domains. Comment on your results. Read the **fir1()** help files to learn how to use this command. We will later use it extensively.