Lab 8 – Discrete Fourier transform (DFT) & FFT

Objectives: In this lab we will use DFT to process signals

8.1 DFT for frequency analysis of signals (theory + script)

This task gives a simple example of continuous-time signal processing using DFT. Connections are made between CTFT, DTFT, and DFT. For the theory part, write/sketch in your notebook and upload the scan.

Consider the continuous-time signal $p(t) = \cos(\omega_1 t) + \cos(\omega_2 t)$. For simplicity assume that $\omega_1 < \omega_2$. This part is based on Section 7.4 of the Proakis textbook.

(a) What is the Fourier transform $P(\omega)$ of $p(t)$? Sketch it.

(b) Let $p(t)$ be sampled at frequency $\omega_s = \frac{2\pi}{T_s}$ which is greater than Nyquist rate of $p(t)$ giving $p[n] = p(nT_s)$. What is the Fourier transform $P(e^{j\omega})$ of $p[n]$? Sketch it in the interval $[-3\pi, 3\pi]$.

(c) Consider a windowing of the signal $p[n]$ given by $x[n] = p[n] \times w[n]$ where $w[n]$ is rectangular window function which is 1 for $0 \le n \le L - 1$ and 0 otherwise. What will be the Fourier transform $X(e^{j\omega})$ of $x[n]$? (Hint: use multiplication property of DTFT). Provide a rough sketch of $X(e^{j\omega})$. What is the effect of windowing on the spectrum? Note that any practical signal processing will involve a window function $w[n]$ of some kind to limit computations.

(d) We can consider $x[n]$ as an L-length signal. Let $\omega_1 = \frac{\omega_s}{8}$ and $\omega_2 = \frac{3\omega_s}{8}$. Note that explicit value of ω_s is not needed and should not be assumed here. Write a script to generate $x[n]$ for **L = 20**. Compute its N-point DFT $X[k]$ using the `fft` command and plot magnitude of DFT for **N = 1000**. Is your answer consistent with the sketch obtained in part (c)? Repeat for **L = 50, L = 100**. What do you notice as **L** changes (focus on the shape instead of peak value)? Plot in a single figure for various **L.**

(e) Let $\omega_1 = \frac{\omega_s}{8}$ and $\omega_2 = \frac{\omega_s}{8} + \frac{\omega_s}{40}$. Repeat (d) for **L = 20, 50, 100**. Plot a single figure for various **L.** What do you notice? From this, what can you conclude about the length of the signal (L) and the frequency resolution?

(f) Repeat (d) and (e) when $w[n]$ is an L-length Hanning window (you can use the matlab command `hann` to get this window).

(g) From the plot of DFT, how would you estimate ω_1 and ω_2 given ω_s? Use this method to find ω_1 and ω_2 in part (d) when **L = 100, N = 1000** and $\omega_s = 8000\ rad/s$. Repeat the estimation when **L = N = 100**. How accurate are your estimates? How does **N** affect your answers?

(h) Repeat (g) to estimate ω_1 and ω_2 in part (e).

8.2 Convolutions using DFT (function + script)

As mentioned in the class, we can use the DFT to compute time-domain convolutions. Though we must be careful to distinguish between linear convolution (which is usually what we want) and circular convolution (which appears naturally in DFT properties).

(a) Write a function `[c1,c2] = dftconv(x1,x2)` that takes two input vectors x1 and x2 of same length and returns their circular convolution (c1) and linear convolution (c2) computed using DFT. You must not directly perform convolutions using summation formula but instead compute it using DFT and inverse DFT of the appropriate signals (theory for this is summarized in first page of Lecture 21 class notes). You can use the matlab commands `fft` and `ifft` to compute the DFT and inverse DFT of required signals.

(b) In a script, generate two vectors x1 and x2 randomly of length N = 10 and pass them as inputs to the above function. Use the Gaussian random number generator to create the inputs, for example, `x1 = randn(1,N)`. The outputs of your function should match those of the commands `cconv` and `conv`, respectively (read up MATLAB documentation of these commands). You can compare the outputs by plotting results of `dftconv` and `cconv` and `conv` all in the same figure with 2x2 subfigures.

(c) What is the theoretical computational complexity of evaluating the convolution as implemented by the function `dftconv` using FFT based methods?

(d) How does it compare with the computational complexity of directly evaluating the convolution in time domain?

8.3 Direct computation of the DFT (function + script)

In this task we will perform direct DFT computation.

(a) Write a function `directdft` that computes the N-point DFT of an input vector x and returns an output vector X. Your function should implement the DFT equations directly (do not use `fft` command). You should take advantage of MATLAB's vector and matrix-based syntax instead of using nested for loops. The output of your algorithm should agree with MATLAB's `fft` output on the same input.

(b) In a script test your function for various N-length input signals as follows

- x = [ones(L,1), zeros(N-L,1)], for fixed L = 5 repeat for N = 5, 10, 100
- x = sin(w0*n), x = cos(w0*n), x = e^(j*w0*n), for w0 = 3*pi/10 and N = 20
- x = sin(w0*(n-1)), x = cos(w0*(n-2)), x = e^(j*w0*(n-3)), w0 = 3*pi/10, N = 20
- x = [1, zeros(N-1,1)], N = 10
- x = (0.9)^(n), N = 20

In the same figure plot the input signal x and the magnitude and phase of the output X. Make sure that you can explain all the observed plots.

8.4 tic toc tic toc (script)

In this experiment we will compare the time taken (empirically) to compute the DFT using the above `directdft` function and inbuilt `fft` command. This we will do by using the `tic` and `toc` MATLAB routines (read up documentation as for how to use these to measure time). As the N is increased from 1000 to 20000 in steps of 1000 generate a random N-length signal and compute its DFT using both the methods and record the time taken for each method using the `tic` and `toc` routines. Plot the time taken as function of N for each method in the same figure. Adjust your range of N as required if the script is too slow or too fast. Note that these routines provide only an approximate estimate of computational time and may not be precise in measuring very short durations.

8.5 The radix-2 FFT (function, optional)

Write a function `radix2fft` which takes as input an N-length vector x and returns an N-length vector X. To compute X, implement the decimation-in-time radix-2 FFT algorithm for an input vector whose length is a power of 2 i.e. assume that $N = 2^m$. Note that this function will have a recursive structure. Specifically, `radix2fft` is called within itself until it reaches the stopping criteria of N = 2. What is the DFT when N = 2? Verify that the output of your function matches with that of `directdft` and `fft`.