SCHOOL *of* ENGINEERING and ELECTRONICS
MSc in Signal Processing and Communications

Digital Signal Processing Labs (PGEE11108)

Laboratory Session 3:

The Discrete Fourier Transform

Introduction

This session examines the computer implementation of the Fourier Transform – the Discrete Fourier Transform. The DFT lies at the heart of DSP. While other analytical transforms, such as the Fourier, Laplace and z-transforms provide valuable understanding, it is the ability to compute the DFT for discrete data that makes it so useful. The advent of fast versions of the DFT (the FFT) essentially launched modern DSP.

For images we will investigate the notion of spectrum and simple filtering in the frequency domain. As we will see, in all cases, we can interpret the effect of these filters in the frequency domain easily while their effect is not obvious in the time domain.

Theory

The discrete time analogue of the Fourier transform is the Discrete Time Fourier Transform (DTFT). This enables one to evaluate the frequency component of a discrete time signal at any frequency. It is defined as:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

Since the signal is discretely sampled, the DTFT is periodic in frequency, i.e. $X(e^{j(\omega+2\pi k)}) = X(e^{j\omega})$.

The Discrete Fourier Transform (DFT) can be thought of as a discrete sampling (in frequency) of the DTFT for *finite duration* signals, as it is defined as:

$$X_N(k) = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}$$

Where the signal duration is from 0 to N-1 and the frequencies correspond to $\omega = 2\pi k / N$.

In fact, the DFT is defined for values of k beyond 0 to N-1 in a periodic way just as the DTFT. Because the DFT is "sampled" in both time and frequency it is periodic in *both* time and frequency and implicitly treats signals as being periodic. *It is therefore also the discrete time analogue of the Fourier series*.

The DFT also an easily computed inverse (unlike the DTFT):

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X_N(k)e^{j2\pi kn/N}$$

We see that both the DFT and the IDFT are linear operations acting on N-dimensional vectors. They therefore can be written as matrices and have standard matrix properties.

We will explore the implementation of the DFT and its uses for approximating the DTFT; the role of phase and the relationship between DFT and circular convolution. We will also examine various useful mathematical properties of the DFT.

Experiments

DFT of simple elementary signals

1. In this experiment you will look at the DFT (in particular its magnitude) of some simple sine signals and examine the effect of *zero padding*. In each case the signal will be 100 samples long, but you will also examine the DFT with lengths of 100, 200 and 1000 (plus other lengths ***greater than 1000*** if you wish) and comment on what zero padding does to the spectrum.

   1.1. Create a **.m** file where you will enter the skeleton script to be used for the exercises to follow. Start by setting up the signal:

```
L = 100;                          % length of signal i.e. 100 samples
Fs = 1000;                        % sampling frequency in Hz
n = [0:L-1];
t = n/Fs;                         % time axis in seconds
T = 0.11;                         % sine period
x = sin(2*pi*t/T);                % the first 100 samples
```

   Add a fully labelled `plot` function in order to view the generated signal, x, and also calculate the signal's DFT using `fft(x)`. Plot the magnitude of the DFT against frequency (in Hz) up to the half-sampling frequency using a dB scale.

   Make sure that you add lines to your code to correctly label the axes (time in seconds and frequency in Hz) and add titles to all of the displays.

   1.2. Keeping L at 100, repeat the above with T = 0.011 and 0.005 seconds. Verify that the frequency domain representation is correct, i.e. that the dominant frequency component is correctly placed.

   1.3. Edit your code to further calculate the DFT of x padded with zeros to a length N

```
xp = [x,zeros(N,1)];
fp = fft(xp);
```

   Does it matter where the zeros go?

   Choose one of the sinusoid frequencies and **on the same plot** as the DFT in part 1.2, plot the zero-padded DFT with N = 200 and 1000.

   Recall, padding with zeros is a way of approximating the DTFT of the finite duration signal using the DFT. Can you explain the difference between the two functions?

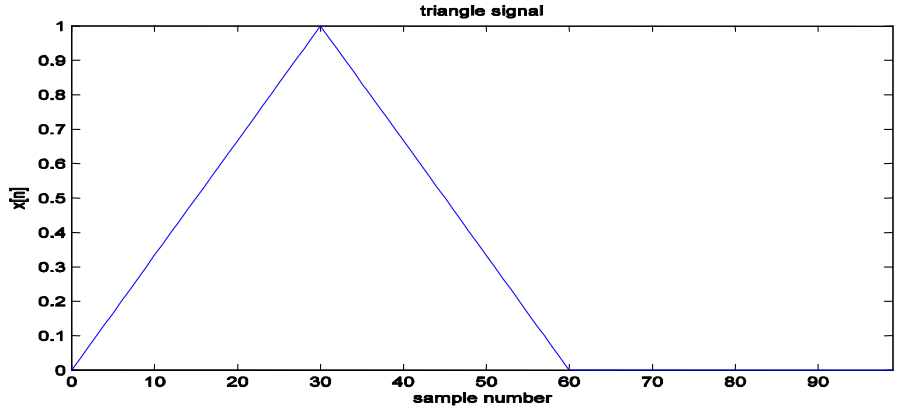
2. Next we will examine the phase of the signal.

   2.1. Create a **.m** file that generates an impulse signal:

$$x[n] = \begin{cases} 1 \text{ when } n = 0 \\ 0 \text{ otherwise} \end{cases}$$

   Use a signal length of 100 (remember in MATLAB the first point in a vector is `x[1]`). Calculate the DFT of the impulse signal and plot its magnitude (using a linear scale) and phase in separate subplots. Repeat the process with the same signal delayed by 1, 10, 50 and 80 samples. Can you explain what you see? You might wish to try using the `unwrap` command (use the help command to understand this).

   2.2. Create a **.m** file that generates a triangle signal (note the signal is of length 100) that looks like:



Calculate the signal's DFT and plot its magnitude (using a dB scale) and phase against frequency up to half the sampling frequency.

2.3. Take the product of the impulse DFT from 2.1, for the different delays, and the DFT of the triangle signal. Apply the inverse DFT (using the `ifft` command) and plot the resulting signals (beware: MATLAB may have generated some imaginary components due to finite precision arithmetic – take only the real component). What do you see?

2.4. **Circular convolution** – the last exercise should have shown an elementary example of circular convolution. Now let's look at a slightly more complex example. Take the product of the triangle signal DFT with itself and then inverse transform. Try to sketch what you expect to see before you plot the result.

Implementation tricks for the DFT

3. **The Fast Fourier Transform**

The DFT can be considered as an orthogonal matrix, **F**, with complex elements. As such, it has all the standard properties of a matrix: determinant, eigenvalues/vectors, etc. The DFT also has special structure that often allows it to be factorized into simpler (matrix) operations. For example in 1965 Cooley and Tukey proposed a fast implementation of the DFT called the Fast Fourier Transform (FFT) based upon the following decimate in time (DIT) factorization:

$$X_N(k) = \sum_{n_1=0}^{1} \sum_{n_2=0}^{(N/2)-1} x[n_1 + 2n_2] e^{-j2\pi k(n_1+2n_2)/N}$$

$$= \underbrace{\sum_{n_2=0}^{(N/2)-1} x[2n_2] e^{-j2\pi k n_2/(N/2)}}_{DFT_{N/2}(x_{even})} + e^{-j2\pi k/N} \underbrace{\sum_{n_2=0}^{(N/2)-1} x[2n_2+1] e^{-j2\pi k(n_2)/(N/2)}}_{DFT_{N/2}(x_{odd})}$$

Remember that the N/2 DFTs are defined for frequencies, k, outside of 0 to N/2-1.

Make sure that you understand the structure of the DIT formula above.

Using the factorization above write a MATLAB script to perform a 100 length DFT using two 50 length DFTs. Test it on the triangle signal. It may help to write this implementation down mathematically as the product of matrices acting on the vector `x[n]`.

4. **Two DFTs for the price of one**

In many practical applications we wish to apply the DFT to *real valued* signals. In this case the output sequence has the following symmetry:

$$X_N(k) = X_N^*(N-k)$$

where the star denotes complex conjugation.

The DFT output for real inputs is thus *half* redundant, and one obtains the complete information by only looking at roughly half of the outputs. In this case, the "DC" element $X_N(0)$ is purely real, and for even N the "Nyquist" element $X_N(N/2)$ is also real, so there are exactly N non-redundant real numbers in the first half + Nyquist element of the complex output X.
This redundancy can also be exploited as follows. Given two real N-point sequences `x[n]` and `y[n]` we can define a complex sequence:

```
v[n] = x[n]+j y[n]
```

In your lab books write down expressions for: $\left(V_N(k)+V_N^*(N-k)\right)/2$ and $\left(V_N(k)-V_N^*(N-k)\right)/2$ in terms of the DFTs of `x[n]` and `y[n]`. Hence create a MATLAB function, `dualFFT(x,y)`, that calculates the DFT of two real N-point sequences `x[n]` and `y[n]` using a single N-point DFT (take care on the indexing in MATLAB!). Test it on known pairs (e.g. a triangle signal and an impulse).

5. **A faster DFT for a real sequence**
Exercises 3 & 4 suggest that we might be able to perform a single *real* N-point DFT using only one N/2 point DFT. This is possible by noting the following:

   a) Your solution to part 3 can calculate a real N-point DFT using two real N/2 DFTs
   b) and your solution to part 4 allows you to calculate two real N/2 DFTs using one complex N/2-point DFT.

Hence create a MATLAB function, `realFFT(x)` that can calculate a real N-point DFT using a single `fft` command applied to an N/2-point sequence. Test the function on a known sequence.

Applications of the DFT to images

Please download the following images from the resources folder to your Matlab home directory:

- Image of Lena
- Landsat image
- sonar image
- sar image

We will also use the standard circuit and flowers images (circuit.tif, peppers.png). You are of course encouraged to try these programs out on images of your choice. Please also download the following functions and script:

- phase_only.m
- move_image.m
- SimpleFiltering.m

1. **Phase manipulation**:

As you have already seen in class, the phase of the spectrum is tightly related to the structure of the image. This will be illustrated in the following.

- Please load an image (for example `circuit.tif`).
- Display the image in a figure;
- Now use the `phase_only.m` function (`help phase_only`).
- Comments?

Modify the `phase_only.m` function program to write a program which replaces the phase of the spectrum by a random phase. Hints: use the `size` function to get the size of the image and the `rand` function to generate the random phase between 0 and 2π. Make sure that the generated image has the correct phase symmetry to ensure that the image is real valued.

Do the same with the amplitude of the spectrum (random between 0 and 1). What do you see?

Try it with various images

2. Displace an image using the FFT:

As we have seen a linear phase shift in the Fourier domain corresponds to a translation in the space domain. This could be used to translate an image in the space domain using simple multiplication in the Fourier domain. Try to write a program that does that.

Now download the program called move_image.m and see how you can use it.

How does it work? What do you see?

3. Simple Fourier Filtering:

Again as seen in class, the high frequencies of the spectrum corresponds to the edges while the low frequencies corresponds to the smooth 'filling'. An easy way to demonstrate that is to take the Fourier transform of an image, cut the high (resp low) frequencies and Fourier transform back the result. Download the `SimpleFiltering`.m program and make sure that you understand what it is doing. Try it out on various images. What are the issues involved with this type of filtering?

You will have seen in class the convolution theorem. Can we use it to perform filtering efficiently? This is the next Lab session…