1. Computing Fourier Transform (by hand)

(a) Compute the DFT of the following 1-D signal sequences using the equation,

\[ F(u) = \sum_{x=0}^{M-1} f(x)e^{-j2\pi xu/M} \]

(i) \([2, 3, 4, 5]\) for \(F(1)\)
(ii) \([-9, 8, -7, 6, -5]\) for \(F(2)\)

(b) Given a 4-bit grayscale image of size 3x3 pixels. If Discrete Fourier Transform (DFT) is applied to the image,

\[
\begin{array}{ccc}
15 & 5 & 5 \\
13 & 3 & 12 \\
9 & 0 & 11 \\
\end{array}
\]

what is the value of

(i) \(F(0,0)\)
(ii) \(F(1,1)\)

2. Understanding DFT through Convolution (1D Example)

(a) Suppose \(a\) and \(b\) are two vectors of the same length \(N\). If so, we define their convolution to be the vector

\[ z = a \ast b \]

\(a\) and \(b\) can represent two different polynomials that are to be multiplied. For example, suppose we have

\[ a = [1 \ 2 \ 3 \ 4] \text{ and } b = [5 \ 6 \ 7 \ 8] \]

then we have

\[ a(u) = 1 + 2u + 3u^2 + 4u^3 \]
\[ b(u) = 5 + 6u + 7u^2 + 8u^3 \]

Multiplying both polynomials result in

\[ a(u) \ast b(u) = 5 + 16u + 34u^2 + 60u^3 + 61u^4 + 52u^5 + 32u^6 \]
In Matlab,

\[
\begin{align*}
\text{>> } & a = [1 \hspace{0.5em} 2 \hspace{0.5em} 3 \hspace{0.5em} 4] \\
\text{>> } & b = [5 \hspace{0.5em} 6 \hspace{0.5em} 7 \hspace{0.5em} 8] \\
\text{>> } & \text{conv}(a,b)
\end{align*}
\]

For **circular convolution**, the b vector is imagined to be periodic, and indexed backward from 0, thus resulting in multiplying

\[
 a_0b_0, \ a_1b_{-1}, \ a_2b_{-2}, \ a_3b_{-3}, \ldots \text{ and iterates accordingly.}
\]

Thus, the product \(a(u)b(u)(1+u^n)\) can be computed, and the coefficients \(u^n\) to \(u^{2N-1}\) are extracted.

Continuing where we left,

\[
\begin{align*}
\text{>> } & \text{c = conv([a a], b)} \\
\text{>> } & \text{c = c(length(a)+1:2*length(a))}
\end{align*}
\]
gives you the extracted coefficients from circular convolution from element \(N\) to \(2N-1\). As you would have noticed, the remaining coefficients are from normal convolution.

(b) Next, perform DFT on both vectors \(a\) and \(b\), and multiply them together,

\[
\begin{align*}
\text{>> } & p = \text{fft}(a').*\text{fft}(b');
\end{align*}
\]

In Matlab, we calculate the forward and inverse transforms of Fourier Transform with \texttt{fft} and \texttt{ifft}, respectively. Here \texttt{fft} stands for **Fast Fourier Transform (FFT)**, which is a fast and efficient method of performing the DFT. For clarity, we perform \texttt{fft} on a column vector (instead of a row vector) so that we can visualise the values better.

Inverse the answer we got earlier,

\[
\begin{align*}
\text{>> } & \text{ifft}(p)'
\end{align*}
\]

Ah! Some of the values seem to have imaginary numbers, but \(0.000i\) actually indicates no imaginary value. This way will give you only the real part,

\[
\begin{align*}
\text{>> } & \text{real(ifft(p)')}
\end{align*}
\]
which are the same values as performing circular convolution.

What can we conclude here?

3. **Fourier Transform of Images (2D)**

(a) In Matlab, the relevant functions for 2-D DFT are

- \texttt{fft2}, which takes the DFT of a 2-D matrix
- \texttt{ifft2}, which takes the inverse DFT of a 2-D matrix
- \texttt{fftshift}, which shifts a transform to place the DC coefficient in the center
>> x = ones(8)
>> fft2(x)
This produces the DFT of a 8x8 matrix full of ones. Note that the DC coefficient is indeed the sum of all the matrix values. Sometimes, it is also divided by the number of pixels, MxN to produce the average gray level value of the image f (in the range of 0 – 1).

Try

>> g = [zeros(8,4) ones(8,4)]
>> gf = real(fft2(g))
>> gf2 = fftshift(real(fft2(g)))

In the last line above, we perform DFT on the matrix g first, then obtain the real values, and finally, shift the values to the center. All 3 operations in just one line!

(b) Now, let's try with some simple images. Generate a simple image consisting of a single edge:

>> a = [zeros(256,128) ones(256,128)];
>> imshow(a);
>> af = fftshift(fft2(a));
>> imshow(af);

In this case, showing the DFT image results in “Warning: Displaying real part of complex input”. In Matlab, warnings are not crucial, they do not prevent the task from being performed, but the warned problem should be rectified if possible. In this case, showing only the real values of af using real will solve the warning.

The displayed spectrum image is totally black except for the middle DC component (little white dot) and it's not possible to see anything else...

To view its spectrum clearly, we can perform amplitude rescaling using logarithm transformation, and then scale the matrix for display as an image:

>> af1 = log(1+abs(af));
>> imshow(mat2gray(af1));

With the above steps, can you write a small function to help you view transforms, if you need to do that frequently in the future?

(c) Try and display the (center-shifted) DFT spectrums of the following images, which are generated with the given lines:

(i) Image with a box

>> b=zeros(256,256);
>> b(78:178,78:178)=1;
(ii) Image with a circle

\[
\begin{align*}
&\text{>> } [x, y] = \text{meshgrid}(-128:127, -128:127); \\
&\text{>> } z = \text{sqrt}(x.^2 + y.^2); \\
&\text{>> } c = (z < 15);
\end{align*}
\]

(d) Convert the Fourier-transformed shifted images obtained in part (c) back to their original images using \textit{ifft2} and \textit{ifftshift}.

(e) Display the DFT spectrum of the following images:
   (i) cameraman.tif
   (ii) rice.tif
   (iii) pout.tif

Compare their spectrums. Are you able to make some conclusions just based on the spectrum alone?

**Homework**

1. Write a program to accomplish a 2-D DFT operation (normally performed with \textit{fft2}) using only 1-D DFT operations (using \textit{fft}). You are actually making use of the separability property of the 2-D DFT.

2. After obtaining a spectrum image transformed by DFT, think of some ways you can manipulate this spectrum image to filter out low-frequency values (high-pass filter) or filter out high-frequency values (low-pass filter).