CPSC 535
Assignment 2a: The Discrete Fourier Transform

The goal of this assignment is to explore the properties of the discrete Fourier transform (DFT), a staple in image and signal processing.

In class we looked at how a rectangular function is transformed into a series of complex coefficients. These coefficients represent the frequency content of the rectangular pulse. When the coefficients are expanded and added, they reconstruct the original function. For this assignment you will reproduce that analysis using the DFT, in two dimensions, while animating the expansion and reconstruction process.

Section 3.2 of Sonka, Hlavac and Boyle covers the Fourier transform.

1 Animate the Expansion of a DFT

Working in Octave, create an image of size 32 by 32, in which every pixel has a grey value of 64. Within this image, place a square region of pixels with a grey value of 192, size 16 by 16, in the center. View this image, call it f, using **gsplot**. Compute the two-dimensional discrete Fourier transform of this image, F, using the Octave function **fft2**.

Octave addresses the elements of F as shown in Figure 1(a), that is, indices start at 1 and go up to the number of rows or columns. Unfortunately, these indices do not refer to the spatial frequencies, u and v directly. These are given in Figure 1(b) and are necessary because the DFT is a periodic function. For the purposes of animation in this assignment, we let $w = |u| + |v|$, and add expanded coefficients in order of increasing w.

Write Octave code to do the animation as follows.

1. Initialize the reconstructed image to all zeros, $\hat{f} = \mathbf{0}$.
2. For $k = 0 \ldots w_{max}$ do
   - Create an image containing only the coefficients from F for which $w = k$, F_k, using
   $$F_k(i,j) = \begin{cases} F(i,j) & w(i,j) = k \\ 0 & \text{otherwise} \end{cases}$$
   - Expand the coefficients in F_k, using the inverse two-dimensional FFT, **ifft2**, to get f_k.
   - Update the reconstructed image using
   $$\hat{f} = \hat{f} + f_k$$
   - Plot $\hat{f}$.

Hints:

1. The easiest way to compute F_k is to create the array w that contains values of w for each row and column, then use the Octave statement:

```
Fk = F .* (w == k);
```

2. To help create the array w, it is useful to know that you can make an array of identical rows by multiplying a single row by a column vector containing all ones, e.g.,

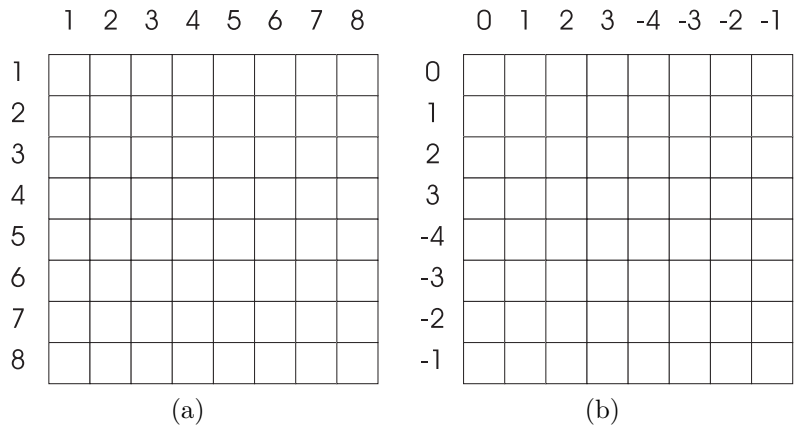

Figure 1: Addressing the output of DFTs in Octave: (a) the row and column indices used by Octave, and (b) the corresponding values of u and v.

```
u = ones(n,1) * [0:n/2-1 -n/2:-1];
```

3. The Octave **pause** function may be useful for slowing down the speed of the animation, if necessary.

4. **ifft2** returns complex numbers. If you are doing things correctly then the F_ks have the correct symmetry properties to make the f_ks real. Use this when converting the complex variables to real for plotting.

When this animation is working, try changing the original function f. Make it wider, narrower, rotate it, make it asymmetrical, and try different shapes. Also, take a look at the Fourier basis functions themselves. This should help you to answer the questions that follow.

2 Images

While the animation in the previous section can help you understand the properties of the DFT, it is important to relate this to what you see in an image. You will be provided with a disk containing a *bmp* image file with your picture, should you desire it, that you can use as a sample image. You may opt to use a different image.

Although Matlab offers utilities for reading a variety of image file formats, Octave is somewhat limited. I prefer to use **pnm_read** and **pnm_write**. These are provided for you. To convert your *bmp* file to a *pgm* file of smaller dimension I suggest the following:

```
bmptoppm image.bmp | ppmtopgm | pnmscale 0.25 > image.pgm
```

Repeat the animation done for the previous section, but rather than plot the image data, save it to a *pgm* image file. You can then view the animated sequence using *xv*.

Hints:

1. Assign names to the file in the output image sequence using the following

```
name = sprintf( "base.%04d.pgm", k );
```

   where k is the frame number in the sequence. This will give the file names *base.0000.pgm*, *base.0001.pgm*, and so on.

2. View the animation with **xv** using

```
xv -wait 0 -wloop -expand 4 base.*.pgm
```

The naming convention outlined above will guarantee that the images are displayed in the right order.

3 Questions

1. What does the transform of an image of all zeros and a single pixel with a value of one, i.e., a discrete Dirac delta function, look like? What is the significance of the delta function with respect to convolution?

2. If you rotate an image by θ degrees, what happens to the DFT of the image? Hint: try the following in Octave.

   - `x1 = zeros(32);`
   - `x1(14:18, 8:24) = 255;`
   - `pnm_write( "pgm-raw", "x1.pgm", x1 );`
   - from the unix command prompt run *xv x1.pgm*. You should see a black image with a white rectangle. Use the *clear rotate* algorithm from xv's Algorithm menu. Rotate 30 degrees. Save the rotated image as **x2.pgm**.
   - `x2 = pnm_read( "x2.pgm" );`
   - `X1 = fft2(x1);`
   - `X2 = fft2(x2);`
   - Now look at the two DFTs for x1 and x2.

3. Compare the DFTs of two one-dimensional functions, each with a rectangular pulse, but in different positions. What can you say about the magnitude (use **abs** in Octave) of the DFTs? About the phase (use **arg** in Octave)? What can you say in general about the effect of translation in an image on the DFT?

Hand In

1. Cover sheet with your name, course number, and assignment number only.

2. Name and student ID number inside the cover sheet.

3. Answers to the questions.

4. E-mail your **.m** file of Octave code that animates the DFT expansion for both f and your sample image to your TA.

Marking

Assignment grades will be based on

1. answers to the questions, and

2. the quality of your Octave code.

Your code should be correct, readable, well-documented, and modular. Octave and Matlab are not very readable languages, but you should still be able to produce good-quality code that is maintainable.