ECE 280L Fall 2024

Laboratory 7:
Image Processing 2

Contents

1 Introduction

This lab expands on the topic of Digital Image Processing and specifically delves into the frequency domain. In this lab, you will further extend your programming abilities in MATLAB® and your knowledge of Fourier to explore digital image processing techniques based on the frequency content of an image.

This handout and supporting web pages will teach you about the fundamentals of two-dimensional Fourier representations of images and how to apply low, high, band-pass, and band-stop filters in the frequency domain. There is a Pundit page that accompanies this document and that has the example codes and images. See https://pundit.pratt.duke.edu/wiki/ECE_280/Imaging_Lab_2

2 Objectives

The objectives of this project are to:

- Become familiar with the two-dimensional Fourier representations of images, specifically the complex exponential representation.
- Understand and use MATLAB's built-in 2-D fast Fourier transform (`fft2`) and associated functions (specifically `fftshift`, `ifftshift`, and `ifft2`) to analyze the frequency content of an image and reconstruct an image from frequency components.
- Apply logical masks to an image's frequency content to filter the image.

3 Background

3.1 Hints

You can use MATLAB's `sprintf` and `eval` commands to dynamically generate a line of MATLAB code and then execute it. This will be very handy for exercises asking for multiple plots. For example, the solution for Exercise 6 could have the following loop in it:

```
% All the code to make Figures 1-3 and normalized v, u

ulim = [0.5 0.5 0.5 0.5 0.5 0.1 0.2 0.3 0.4 0.5];
vlim = [0.1 0.2 0.3 0.4 0.5 0.5 0.5 0.5 0.5 0.5];
for k=1:10
    filter = ...
    % Code for figures 3 through 5 here

    % Save Figure 5 with name based on what k is
    eval(sprintf('print -dpng IP2_EX6_Plot%0.0f.png', k))
end
```

And exercise 5's solution could have the following loop:

```
% All the code to generate Figures 1-7
% Code to save Figures 6 and 7
for k=6:7
    figure(k)
    eval(sprintf('print -dpng IP2_EX5_Plot%0.0f.png', k))
end
```

3.2 One-Dimensional Synthesis and Analysis

For a one-dimensional finite-duration discrete data set $x[m]$ containing M points, you can represent the information in the data set as a sum of either cosines and sines or of complex exponentials using discrete Fourier series. The specific formulas we will start with from class are as follows:

- Using trig functions:

$$x[m] = \sum_{k=0}^{M-1} A[k]\cos\left(\frac{2\pi}{M}km\right) + B[k]\sin\left(\frac{2\pi}{M}km\right)$$

  where

$$A[k] = \frac{1}{M}\sum_{m=0}^{M-1} x[m]\cos\left(\frac{2\pi}{M}km\right)$$
$$B[k] = \frac{1}{M}\sum_{m=0}^{M-1} x[m]\sin\left(\frac{2\pi}{M}km\right)$$

- Using complex exponentials:

$$x[m] = \sum_{k=0}^{M-1} \mathbb{X}[k]\exp\left(j\frac{2\pi}{M}km\right) where \mathbb{X}[k] \qquad = \frac{1}{M}\sum_{m=0}^{M-1} x[k]\exp\left(-j\frac{2\pi}{M}km\right)$$

As it happens, MATLAB moves the $1/M$ factor to the *synthesis* equation for $x[m]$ versus the analysis equation. Depending on the particular application, the $\frac{1}{M}$ might be in the synthesis equation, in the analysis equation, or there might even be a $\frac{1}{\sqrt{M}}$ in both! Going forward in this document we will end up using the following relationships to be consistent with MATLAB:

- Using trig functions:

$$x[m] = \frac{1}{M}\sum_{k=0}^{M-1} A[k]\cos\left(\frac{2\pi}{M}km\right) + B[k]\sin\left(\frac{2\pi}{M}km\right)$$

where

$$A[k] = \sum_{m=0}^{M-1} x[m] \cos\left(\frac{2\pi}{M}km\right)$$
$$B[k] = \sum_{m=0}^{M-1} x[m] \sin\left(\frac{2\pi}{M}km\right)$$

- Using complex exponentials:

$$x[m] = \frac{1}{M}\sum_{k=0}^{M-1} \mathbb{X}[k] \exp\left(j\frac{2\pi}{M}km\right)$$

where

$$\mathbb{X}[k] = \sum_{m=0}^{M-1} x[m] \exp\left(-j\frac{2\pi}{M}km\right)$$

EXAMPLE 1: Signal Analysis

Imagine that you have some $x[m]$ as follows:

$$x[m] = [1, 5, 2, 3, 1]$$

If we get the Fourier series representation for this, we are basically assuming that there is some signal $\hat{x}[m]$ that repeats $x[m]$ with a period of 5. We can thus get the "synthesis equation scaled" version of the discrete-time Fourier series using:

$$A[k] = \sum_{m=0}^{4} x[m] \cos\left(\frac{2\pi}{5}km\right)$$
$$B[k] = \sum_{m=0}^{4} x[m] \sin\left(\frac{2\pi}{5}km\right)$$

or

$$\mathbb{X}[k] = \sum_{m=0}^{4} x[m] \exp\left(-j\frac{2\pi}{5}km\right)$$

which would yield:

$A[k] = [12.0000, -1.1910, -2.3090, -2.3090, -1.1910]$
$B[k] = [0, 3.2164, 3.3022, -3.3022, -3.2164]$
$\mathbb{X}[k] = [12.0000 + 0.0000i, -1.1910 - 3.2164i, -2.3090 - 3.3022i, -2.3090 + 3.3022i, -1.1910 + 3.2164i]$

Note the relationships between $\mathbb{X}[k]$ and the combination of $A[k]$ and $B[k]$. These are slightly different from the relationships we get in Fourier series coefficients mainly because in the Fourier series there are different numbers of trig coefficients than there are exponential coefficients. Here, you can see that A and B seem to have redundant information and also each has the same number of elements as the original data set. This will become important later.

Pre-lab Deliverable (1/4): Which of the following accurately summarizes the relationship between A, B, and $\mathbb{X}$ in Example 1?

1. $\mathbb{X} = A + Bi$
2. $\mathbb{X} = A - Bi$
3. $\mathbb{X} = -A + B$
4. $\mathbb{X} = Ai - B$

EXAMPLE 2: MATLAB fft

MATLAB has a built-in function to perform the *fast Fourier transform* of a discrete, finite-duration data set. The command is `fft` and the only argument it needs is the data set! For example:

```
x = [1, 5, 2, 3, 1]
X = fft(x)
```

yields the following (with some digits chopped to make it fit):

```
X = [12.000+0.000i, -1.191-3.216i, -2.309-3.302i,
         -2.309+3.302i, -1.191+3.216i]
```

This is exactly what we expected from the calculations above! But note that you are still not allowed to use `fft` or `ifft` in solving Exercises 1 and 2.

3.3 Two-Dimensional Synthesis and Analysis

Frequency analysis can be expanded to look in multiple dimensions. If we analyze an MxN image, we need to consider changes in levels (gray or each component of RGB) when we go from left to right as well as when we go from top to bottom. It turns out that the trigonometric analysis is fairly complicated, requiring *four* separate matrices to contain all the frequency information:

$$\begin{aligned} x[m,n] = \frac{1}{M}\frac{1}{N}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}\Bigg(& A[k,l]\cos\left(\frac{2\pi}{M}km\right)\cos\left(\frac{2\pi}{N}ln\right) + \\ & B[k,l]\cos\left(\frac{2\pi}{M}km\right)\sin\left(\frac{2\pi}{N}ln\right) + \\ & C[k,l]\sin\left(\frac{2\pi}{M}km\right)\cos\left(\frac{2\pi}{N}ln\right) + \\ & D[k,l]\sin\left(\frac{2\pi}{M}km\right)\sin\left(\frac{2\pi}{N}ln\right)\Bigg) \end{aligned}$$

which...yikes. And there are four different analysis equations - fortunately, we will not need them as we will be focusing on the complex exponential representation from this point forward.

As it happens, in the same way as the A and B values in the one-dimensional Fourier transform calculated above have redundant information, each of these A, B, C, and D values will as well. The exponential form will be more compact because (a) it stores two distinct pieces of information in a complex number and (b) it has no redundancy. The MATLAB versions of the exponential synthesis and analysis equations for a two-dimensional, discrete, finite-duration MxN data set $x[m, n]$ are:

$$x[m,n] = \frac{1}{M}\frac{1}{N}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}\left(X[k,l]\exp\left(j\frac{2\pi}{M}km\right)\exp\left(j\frac{2\pi}{N}ln\right)\right)$$

and

$$X[k,l] = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(x[m,n]\exp\left(-j\frac{2\pi}{M}km\right)\exp\left(-j\frac{2\pi}{N}ln\right)\right)$$

MATLAB has built-in commands to perform both of these calculations. Specifically, `fft2(x)` will do the two-dimensional fast Fourier transform for a discrete, finite-duration data set and `ifft2(X)` will do the two-dimensional inverse fast Fourier Transform for a discrete, finite-duration data set.

EXAMPLE 3: MATLAB fft2 and ifft2

Let's look at four different 2x2 matrices and their fast Fourier transforms:

$$x1 = \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} \qquad x2 = \begin{bmatrix}1 & 1\\0 & 0\end{bmatrix} \qquad x3 = \begin{bmatrix}1 & 0\\1 & 0\end{bmatrix} \qquad x4 = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$$

The fast Fourier transforms of each (using `fft2(x)`) are:

$$X1 = \begin{bmatrix}4 & 0\\0 & 0\end{bmatrix} \qquad X2 = \begin{bmatrix}2 & 0\\2 & 0\end{bmatrix} \qquad X3 = \begin{bmatrix}2 & 2\\0 & 0\end{bmatrix} \qquad X4 = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}$$

In each case, the $X(1, 1)$ in MATLAB (which is the $X[0, 0]$ value) will be the sum of the entries in the matrix. Note: for Fourier pairs where the $\frac{1}{N}$ or $\frac{1}{NM}$ term is in the *analysis* equation, the $X[0]$ or $X[0, 0]$ term is the *average* value of x - **however** MATLAB has moved the $\frac{1}{NM}$ to the *synthesis* equation so the $X[0, 0]$ term is *not* divided by NM and is thus the *sum* of the entries, not the *average*. Applying `ifft2()` to each of the X matrices will recover the original. Run the following code to see that in action. If you would like to see other forward and inverse transformations, you can certainly edit or add to this code.

```
x1 = [1, 1; 1, 1]
x2 = [1, 1; 0, 0]
x3 = [1, 0; 1, 0]
x4 = [1, 0; 0, 1]
```

```

X1=fft2(x1)
X2=fft2(x2)
X3=fft2(x3)
X4=fft2(x4)

x1a = ifft2(X1)
x2a = ifft2(X2)
x3a = ifft2(X3)
x4a = ifft2(X4)
```

3.4 Frequency Content Mapping

One important thing to consider when it comes to taking the fast Fourier transform of a matrix is how the frequencies are mapped. In a typical one-dimensional exponential Fourier series or transform, there is a 0-indexed Fourier term holding on to the average value and then complex conjugate terms at $\pm k$ holding on to the magnitude and phase of components of the signal oscillating at k times the fundamental frequency.

MATLAB, however, only allows for positive indices. As a result, the "center" or "zero frequency" component is going to be at the far left, or location (1), of a one-dimensional fft and at the top left, or location (1,1) of a two-dimensional fft. For all other entries, there will be complex conjugate pairs equidistant from the edges of the matrix to make sure the corresponding complex exponentials combine to eliminate the imaginary parts.

Note that if the matrix has an even number of rows or columns, there will be a row and/or column of Fourier coefficients just past the half-way point that will have some purely real values that do not match up with a complex conjugate.

EXAMPLE 4: One Dimensional Frequency Mapping

Consider the following program, which looks at the inverse fast Fourier transforms of a finite-duration duration signal with $M = 15$ components. The fast Fourier transform is built by either putting an M in the first entry or by putting $M/2$ in the two entries that represent the same frequency in two different directions:

```
M = 15;
for m=0:((M+1)/2-1)
    Xm = zeros(M,1);
    Xm(m+1) = M/2;
    if m==0
        Xm(m+1)= M;
    else
        Xm(M+1-m) = M/2;
    end
    x = ifft(Xm);
    figure(m+1); clf
    bar(x)
    fname = sprintf('IP2 E5 Plot%0.0f.png', m+1)
    print(fname, '-dpng')
end
```

If you look carefully at the resulting graphs of x, you will see that the frequency increases from 0 in Figure 1 to the highest frequency in Figure 8. You should also note that the magnitude of each of the components is equal to 1.

As an aside - if you were to replace the `((M+1)/2)` limit in the for loop with `M`, Figures 9 through 15 would look exactly like Figures 8 through 2, respectively, since the `Xm` matrix would look the same. These graphs are effectively showing the even, "cosine" part of the exponential decomposition. If you want to see the odd version, the Fourier terms would be (a) purely imaginary and (b) complex conjugate pairs:

EXAMPLE 4a: One Dimensional Frequency Mapping

```
clear
M = 15;
X = zeros(M,1);
for k=1:((M+1)/2)
    Xm = X
    Xm(k) = M/2/j
    if k==1
        Xm(k)= M
    elseif k~=1
        Xm(M+2-k) = -M/2/j
    end
    x = ifft(Xm)
    figure(k); clf
    bar(x)
end
```

NOTE: The 1/2 terms for the cosine and the $\pm 1/2j$ terms for sine should seem familiar! The addition of the M term here is needed to scale the the Fourier values up since MATLAB will scale them down by $1/M$.

Now that you have seen how the frequency information is made up of complex conjugate pairs in one dimension, we can explore exactly how that works in two dimensions.

EXAMPLE 5: MATLAB fft2 and ifft2 for Even Dimensions

Let's look at a 4x4 matrix of random integers and its fast Fourier transform:

$$x1 = \begin{bmatrix} 4 & 5 & 8 & 1 \\ 8 & 4 & 3 & 1 \\ 8 & 7 & 7 & 5 \\ 2 & 7 & 7 & 10 \end{bmatrix}$$

Using `fft2` on this matrix produces:

$$X1 = \begin{bmatrix} 84 & -6+2j & -4 & -6-2j \\ -3-5j & -5-3j & 5-1j & 11-11j \\ -2 & -8+2j & -18 & -8-2j \\ -3+5j & 11+11j & 5+1j & -5+3j \end{bmatrix}$$

Confirm that the `X1(1,1)` is the sum of the entries in `x1`. As you can see, there are entries in column 3 and in row 3 that are purely real and only show up once. Every other entry has a complex conjugate partner that you can find by starting at the `(1,1)` entry, taking steps right and down to locate one entry, then re-starting at the `X1(1,1)` entry and moving *up* and *left*. "But - there is neither up nor left from `X1(1,1)`!" This is where you have to note that in the same way we are assuming that `x1` repeats itself, we assume that `X1` repeats itself. Moving to the left from `X1(1,1)` means wrapping around to the far right after the first step; moving up means wrapping around to the bottom. So, if you take two steps right and one down to get to `X1(2,3)=`$5-1j$, you would find its partner by taking two steps to the left (meaning go from column 1 to column 4 and then column 3) and one step up (meaning go from row 1 to row 4) to get `X1(4,3)=`$5+1j$. The values that are guaranteed to be real have the same number of right/down steps from (1,1) as they do left/up steps.

Among other things, this representation means the *low* frequency components in a direction will be captured in the Fourier entries near the edges and the *high* frequency components will be captured near the center. As we will see later, this can make filtering more difficult than it needs to be. It would be easier if we could shift the fft such that the 0 frequency entry is at the center and the frequencies increase as you go out from there. Basically, we want to "unwrap" the fft information. Fortunately, MATLAB has a command for this: `fftshift`. This command will take the matrix and rearrange it as follows:

$$\text{X} = \begin{bmatrix} TL & TR \\ BL & BR \end{bmatrix}$$
$$\text{fftshift(X)} = \begin{bmatrix} BR & BL \\ TR & TL \end{bmatrix}$$

where TL, TR, BL, and BR represent the rectangular submatrices that result when you divide a matrix into four "equal" quadrants. If the matrix has an even number of rows or columns, the subdivision can be exact and the former (1,1) entry will move to just to the right and just below the "middle" of the matrix. If one or both of the dimensions are odd, the top left quadrant will be larger than the bottom right so that the (1,1) entry will move to the exact middle row and/or column. Here are some examples:

Dimensions	4x4	4x5	5x4	5x5
X	$\begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{bmatrix}$	$\begin{bmatrix} 1 & 5 & 9 & 13 & 17 \\ 2 & 6 & 10 & 14 & 18 \\ 3 & 7 & 11 & 15 & 19 \\ 4 & 8 & 12 & 16 & 20 \end{bmatrix}$	$\begin{bmatrix} 1 & 6 & 11 & 16 \\ 2 & 7 & 12 & 17 \\ 3 & 8 & 13 & 18 \\ 4 & 9 & 14 & 19 \\ 5 & 10 & 15 & 20 \end{bmatrix}$	$\begin{bmatrix} 1 & 6 & 11 & 16 & 21 \\ 2 & 7 & 12 & 17 & 22 \\ 3 & 8 & 13 & 18 & 23 \\ 4 & 9 & 14 & 19 & 24 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix}$
fftshift(X)	$\begin{bmatrix} 11 & 15 & 3 & 7 \\ 12 & 16 & 4 & 8 \\ 9 & 13 & 1 & 5 \\ 10 & 14 & 2 & 6 \end{bmatrix}$	$\begin{bmatrix} 15 & 19 & 3 & 7 & 11 \\ 16 & 20 & 4 & 8 & 12 \\ 13 & 17 & 1 & 5 & 9 \\ 14 & 18 & 2 & 6 & 10 \end{bmatrix}$	$\begin{bmatrix} 14 & 19 & 4 & 9 \\ 15 & 20 & 5 & 10 \\ 11 & 16 & 1 & 6 \\ 12 & 17 & 2 & 7 \\ 13 & 18 & 3 & 8 \end{bmatrix}$	$\begin{bmatrix} 19 & 24 & 4 & 9 & 14 \\ 20 & 25 & 5 & 10 & 15 \\ 16 & 21 & 1 & 6 & 11 \\ 17 & 22 & 2 & 7 & 12 \\ 18 & 23 & 3 & 8 & 13 \end{bmatrix}$

Because of the symmetry that results in the fft of matrices with odd numbers of rows and columns, we will focus on processing images with odd dimensions. It is certainly not impossible to process images with even dimensions - there are just (literally) edge cases we would need to consider that will not be worth the effort in this exercise. Given how `fftshift` works, it is important to note that for matrices with odd dimensions, applying `fftshift` a second time will *not* restore the original! For that reason, MATLAB has a built-in `ifftshift` command that performs nearly the same operation as `fftshift` except for odd dimensions the *bottom right* submatrix takes on the extra row and/or column.

3.5 FFT for Grayscale Images

EXAMPLE 6: FFT of Coins

Now we can take a look at the frequency information for an image. In this case, we will use the built-in "coins.png" image that we looked at in the previous image processing lab. That image is 246x300, so we will remove the first row and first column before processing it. We can start by loading the image, looking at its fft, and looking at the shifted version of the fft:

```matlab
x = imread('coins.png');
x = x(2:end, 2:end);
figure(1); clf
image(x); axis equal; colormap gray; colorbar
X = fft2(x);
Xs = fftshift(X);
figure(2); clf
```

```
imagesc(abs(X)); axis equal; colormap gray; colorbar
figure(3); clf
imagesc(abs(Xs)); axis equal; colormap gray; colorbar
```

The results do not seem promising. The image of the fft does not seem to vary at all, and the image of the shifted fft just seems to have a pixel missing in the middle. If you zoom in on the top left corner of the *unshifted* graph, you will see that same bright point. What happened here is that the `X(1,1)` component is *huge* relative to the result of the components. For this image, `X(1,1)`≈7.6e+06. The next largest magnitude in `X` is an order of magnitude smaller! Given that, we may want to scale things a bit. Add the following to the code above:

```
figure(4); clf
imagesc(log10(abs(Xs))); axis equal; colormap gray; colorbar
```

Figure 4 gives you a much better idea how the frequency information is distributed within the image. The brightest spots are near the center of the imagine, indicating that there are some low-frequency components with larger magnitudes. We can look at two more images to see if they help us understand what is going on in the image. We are going to remove the `X(1,1)` component before shifting and graphing:

```
Xnodc = fft2(x); Xnodc(1,1) = 0;
Xnodcs = fftshift(Xnodc);
figure(5); clf
imagesc(abs(Xnodcs)); axis equal; colormap gray; colorbar
figure(6); clf
imagesc(log10(abs(Xnodcs))); axis equal; colormap gray; colorbar
```

Figure 5 now shows some points of light near the middle of the image but the vast majority of the image is dark. If you zoom in on the center of Figure 5, you will find a familiar pattern: the pixel some distance to the right and down from the center is the same color as the pixel that same distance to the left and up! That is a necessary condition for the complex conjugate symmetry we noted earlier.

Figure 6 once again gives more information about the frequency content. Note that the colorbar (representing the base-10 logarithms of the magnitudes) now stretches from about 0.5 to about 6 rather than the original colorbar which went from about 0.5 to nearly 7. The *next* strongest components are still quite strong, as evidenced in Figure 5. It is worth pointing out that MATLAB was smart enough to figure out that it could *not* graph $\log_{10(0)}$ at the center - that is not so much a black dot as it is a blank in the graph.

3.6 Low Pass Filtering

Now that we can obtain the frequency content of an image, we can filter it by selectively changing the relative magnitudes and phases for the different frequency components in the image. For this lab, we are *only* going to focus on changing the magnitudes and we are *only* going to apply binary filters.

We are going to look at two different filters shapes - round filters comprised of concentric circles or ovals and rectangular filters comprised of concentric squares or rectangles. We will normalize the frequency information such that a "frequency" of 1 will be the fastest frequency as determined by the *larger* dimension of the image. For example, the trimmed image with the coins has 245 rows and 299 columns. From a frequency perspective, that means we would take 122 steps down to get to the fastest vertical oscillations in the image but we could take 149 steps to the right to get to the fastest horizontal oscillations. For the filters we are creating with the coins image, a frequency of 1 will be mapped to the horizontal frequency in the far right column of the shifted fft matrix.

3.6.1 Round Filters

We are going to start with filters based on the distance from the center of the shifted frequency matrix (the "DC" component). These are the "round" filters mentioned above. For the purposes of this lab, they will end up being circular filters.

EXAMPLE 7: Round LPF Applied to Coins

The following code will allow you to visually investigate the filter, original frequency content, filtered frequency content, and filtered image based on a circular low-pass filter that eliminates all frequency components at radial distances greater than one half the maximum possible distance:

```matlab
x = imread('coins.png');
x = x(2:end, 2:end);
figure(1); clf
image(x); axis equal; colormap gray; colorbar

X = fft2(x);
Xs = fftshift(X);
figure(2); clf
imagesc(log10(abs(Xs))); axis equal; colormap gray; colorbar

[rows, cols] = size(x);
max_size = max(rows, cols);
rnorm = rows/max_size; cnorm = cols/max_size;
[v, u] = meshgrid(linspace(-cnorm, cnorm, cols),...
    linspace(-rnorm, rnorm, rows)) ;
filter = sqrt(u.^2+v.^2)<0.5;
figure(3); clf
imagesc(filter); axis equal; colormap gray; colorbar

Xsfiltered = Xs.*filter;
figure(4); clf
imagesc(log10(abs(Xsfiltered))); axis equal; colormap gray; colorbar

Xfiltered = ifftshift(Xsfiltered);
xfiltered = ifft2(Xfiltered);
```

```
26 figure(5); clf
27 imagesc(xfiltered, [0, 255]); axis equal; colormap gray; colorbar
```

You will want to make Figures 1 and 5 as large as possible on your screen to get the best idea of how the filtered image differs from the original.

Pre-lab Deliverable (2/4): How would you change line 16 in the code for Example 7 to change the filter to a high-pass filter?

1. **filter = sqrt**$(u^2 + v^2) > 0.5$;
2. **filter = sqrt**$(u^2 + v^2) < 1$;
3. **filter = sqrt**$(u^2 + v^2) = 1$;

Pre-lab Deliverable (3/4): What shape does the filter in Example 7 make in the frequency domain?

1. A square centered at (0,0)
2. A star centered at (u,v)
3. A circle centered at (u,v)
4. A circle centered at (0,0)

3.6.2 Rectangular Filters

In the section above, we looked at the radial distance from the center as a means of filtering out frequency content. We can also look at the vertical or horizontal distances separately to make rectangle-shaped filters rather than round ones.

EXAMPLE 8: Rectangular LPF Applied to Coins

The following code will allow you to visually investigate the filter, original frequency content, filtered frequency content, and filtered image based on a rectangular low-pass filter that eliminates all frequency components at vertical and horizontal distances greater than one half the maximum possible distance:

```
x = imread('coins.png');
x = x(2:end, 2:end);
figure(1); clf
image(x); axis equal; colormap gray; colorbar

X = fft2(x);
Xs = fftshift(X);
figure(2); clf
imagesc(log10(abs(Xs))); axis equal; colormap gray; colorbar

[rows, cols] = size(x);
max_size = max(rows, cols);
rnorm = rows/max_size; cnorm = cols/max_size;
```

```
14 [v, u] = meshgrid(linspace(-cnorm, cnorm, cols),...
15     linspace(-rnorm, rnorm, rows)) ;
16 filter = (abs(u)<0.5) & (abs(v)<0.5);
17 figure(3); clf
18 imagesc(filter); axis equal; colormap gray; colorbar
19
20 Xsfiltered = Xs.*filter;
21 figure(4); clf
22 imagesc(log10(abs(Xsfiltered))); axis equal; colormap gray; colorbar
23
24 Xfiltered = ifftshift(Xsfiltered);
25 xfiltered = ifft2(Xfiltered);
26 figure(5); clf
27 imagesc(xfiltered, [0, 255]); axis equal; colormap gray; colorbar
```

Note that the only change between this code and Example 7 is on line 16 where the filter is made. The square filter in this case has some extra frequency content relative to what the circular filter had. The round- and square- filtered images are certainly similar, and if you were to change the limits to 0.4, 0.3, 0.2, 0.1 as you did before, those images would be similar too. Given that, you will not need to explore that. However, one thing that we can more easily do with rectangular filters that is harder to do with round ones is set *different* cutoff frequencies in the horizontal and vertical directions.

3.7 HPF and BPF

You can also create frequency filters that *eliminate* the low frequency information and frequency filters that eliminate both the high *and* low frequency information. As above, these filters can be based on both round and rectangular shapes. Unlike images filtered with a low-pass filter, however, using a limit range from 0 to 255 no longer makes sense - HPF and BPF will certainly take out the DC component and the signal of interest will thus have both positive and negative parts. For that reason, it is important to use `imagesc` with colorbars and to recognize that the scales may change significantly depending on which frequencies are retained and which are eliminated.

EXAMPLE 9: Rectangular HPF Applied to Coins

The following code will allow you to visually investigate the filter, original frequency content, filtered frequency content, and filtered image based on a rectangular high-pass filter that eliminates all frequency components at vertical and horizontal distances less than one tenth the maximum possible distance:

```
x = imread('coins.png');
x = x(2:end, 2:end);
figure(1); clf
image(x); axis equal; colormap gray; colorbar

X = fft2(x);
```

```
 7 Xs = fftshift(X);
 8 figure(2); clf
 9 imagesc(log10(abs(Xs))); axis equal; colormap gray; colorbar
10
11 [rows, cols] = size(x);
12 max_size = max(rows, cols);
13 rnorm = rows/max_size; cnorm = cols/max_size;
14 [v, u] = meshgrid(linspace(-cnorm, cnorm, cols),...
15     linspace(-rnorm, rnorm, rows)) ;
16 filter = (abs(u)>0.1) | (abs(v)>0.1);
17 figure(3); clf
18 imagesc(filter); axis equal; colormap gray; colorbar
19
20 Xsfiltered = Xs.*filter;
21 figure(4); clf
22 imagesc(log10(abs(Xsfiltered))); axis equal; colormap gray; colorbar
23
24 Xfiltered = ifftshift(Xsfiltered);
25 xfiltered = ifft2(Xfiltered);
26 figure(5); clf
27 imagesc(xfiltered); axis equal; colormap gray; colorbar
```

Pay careful attention to the changes in this code versus the low-pass version. On line 16, both the relational operators *and* the logical operator have changed! On line 27, the color limits option has been removed. This image represents the information that was *lost* when we did the *low-pass* filter on the coins image earlier with the limits at 0.1 vertically and horizontally.

Pre-lab Deliverable (4/4): What change to line 16 of the code in Example 9 would change the filter to a circular bandpass filter?

1. `filter = (abs(u) > 0.2 & abs(u) < 0.3);`
2. `filter = (.2 < sqrt(`$u^2 + v^2$`) & sqrt(`$u^2 + v^2$`) < .4);`
3. `filter = (sqrt(`$u^2 + v^2$`) < 0.5);`

3.8 Color

Frequency filtering a color image is complicated by the fact that there are three different channels to work with. All of the techniques above for a grayscale image can be applied channel-by-channel to a color image. Note the following:

- As before, a reconstructed image only really makes sense for low-pass filters; the visualization for high-pass and band-pass filters will have different color limits. The visualizations can still be useful.
- MATLAB specifically wants unsigned 8-bit integers for each channel of a color image.

If you want to display a filtered image, you will need to re-cast the image matrix using `uint8()`.

- You can have different filter types and frequency limits for each channel!

There is an example you can look at on the Pundit page that shows how to filter a color image. The process is very similar to the rectangular low pass filtering of a gray scale image (in fact, that process is shown in the code); there are three other parallel tracks that operate on the red, green, and blue channels independently and then put them back together before displaying the image. There is no assignment associated with color images for this assignment.

4 Pre-Laboratory Assignment

For your prelab, provide answers on Gradescope to each of the (4) Pre-lab Deliverables based on the Background information provided above.

The questions are listed below for your convenience:

1. Which of the following accurately summarizes the relationship between A, B, and $\mathbb{X}$ in Example 1?
   (a) $\mathbb{X} = A + Bi$
   (b) $\mathbb{X} = A - Bi$
   (c) $\mathbb{X} = -A + B$
   (d) $\mathbb{X} = Ai - B$
2. How would you change line 16 in the code for Example 7 to change the filter to a high-pass filter?
   (a) **filter = sqrt**$(u^2 + v^2) > 0.5$;
   (b) **filter = sqrt**$(u^2 + v^2) < 1$;
   (c) **filter = sqrt**$(u^2 + v^2) = 1$;
3. What shape does the filter in Example 7 make in the frequency domain?
   (a) A square centered at (0,0)
   (b) A star centered at (u,v)
   (c) A circle centered at (u,v)
   (d) A circle centered at (0,0)
4. What change to line 16 of the code in Example 9 would change the filter to a circular bandpass filter?
   (a) `filter = (abs(u) > 0.2 & abs(u) < 0.3);`
   (b) `filter = (.2 < sqrt(`$u^2 + v^2$`) & sqrt(`$u^2 + v^2$`) < .4);`
   (c) `filter = (sqrt(`$u^2 + v^2$`) < 0.5);`

5 Instructions

5.1 Exercise 1: Calculating Fourier Transforms for 1D Finite-Duration Discrete Data Sets

1. Create a new MATLAB script and title it `IP2_EX1and2.m`.
2. Write a function `analysis` which accomplishes the following:
   (a) The function should take an array x of discrete data as an argument.
   (b) It should return the array of A, B, and $\mathbb{X}$ values for that x.
   (c) Given an input of length M, the calculation of A, B, and $\mathbb{X}$ should be based on the equations:

$$A[k] = \sum_{m=0}^{M-1} x[m]\cos\left(\frac{2\pi}{5}km\right)$$
$$B[k] = \sum_{m=0}^{M-1} x[m]\sin\left(\frac{2\pi}{5}km\right)$$
$$\mathbb{X}[k] = \sum_{m=0}^{M-1} x[m]\exp\left(-j\frac{2\pi}{5}km\right)$$

   (d) For the calculation of $\mathbb{X}$, your function must use a summation of complex exponentials and not use any built-in Fourier functions
3. Test your program with the input used in Example 1 ($x = [1, 5, 2, 3, 1]$) and verify that your returned A, B, and $\mathbb{X}$ match the output from the example.

   Checkpoint (1/7): Show your TA your program and its output when the input is $x = [1, 5, 2, 3, 1]$.

5.2 Exercise 2: Synthesizing 1D Data Sets from Fourier Transforms

1. In the same MATLAB script, write a function `trig_synth` which will take real-valued arrays A and B of length M and return the corresponding x, based on the equation:

$$x[m] = \frac{1}{M}\sum_{k=0}^{M-1} A[k]\cos\left(\frac{2\pi}{M}km\right) + B[k]\sin\left(\frac{2\pi}{M}km\right)$$

2. Verify that your function works by using the A and B output arrays from Exercise 1 as input and checking that you get the correct x back as output.
3. Write a second function, `exp_synth`, which will take a complex-valued array $\mathbb{X}$ of length M and return the corresponding x according to the equation:

$$x[m] = \frac{1}{M}\sum_{k=0}^{M-1}\mathbb{X}[k]\exp\left(j\frac{2\pi}{M}km\right)$$

- For this function, you must use a summation of complex exponentials and *not use* any built-in inverse Fourier function.

4. Verify that your second function works correctly by using the $\mathbb{X}$ output from Exercise 1 as function input and checking that you get the correct x back as output.

   Checkpoint (2/7): Show your two verified functions to your TA.

   Deliverable (1/17): Include your MATLAB file with the three functions (`analysis`, `trig_synth`, and `exp_synth`) in your lab report.

5.3 Exercise 3: Exploring Round LPFs Applied to Coins

1. Create a MATLAB script and title it `IP2_EX3.m`. Copy in the code from Example 7.

   Discussion (1/1): Why, to get `Xsfiltered`, do we multiply `Xs` and `filter`?

2. Run the code as written and save the final filtered image *(not the filtered FFT image)* as `IP2_EX3_Plot1.png`.

   Checkpoint (3/7): Show your TA the output of the code. What are some similarities and differences between the original image and the filtered image?

3. Decrease the radius from 0.5 to 0.4 and run the code again, saving the final filtered image this time as `IP2_EX3_Plot2.png`.

4. Repeat this three more times, setting the radius to 0.3, 0.2, and 0.1 and saving the resultant final filtered image for each run as `IP2_EX3_Plot3.png`, `IP2_EX3_Plot4.png`, and `IP2_EX3_Plot5.png`, respectively.

   Deliverable (2/17): In your lab report, discuss how changing the radius of the filter affected the resulting filtered image.

   Deliverable (3/17): Include the following files in your report:

   - `IP2_EX3.m`
   - `IP2_EX3_Plot1.png`
   - `IP2_EX3_Plot2.png`
   - `IP2_EX3_Plot3.png`
   - `IP2_EX3_Plot4.png`
   - `IP2_EX3_Plot5.png`

5.4 Exercise 4: Exploring Round LPFs Applied to Noise

1. Create a new MATLAB script and title it `IP2_EX4.m`. Copy in the code from Example 7, or the code you just ran in Exercise 3, with the filter radius set to 0.5.

2. Change the original matrix `x` so that instead of the coins image it is a 399×399 matrix of random integers between 0 and 255.

3. Run the program with a filter radius of 0.5 and save the resulting final filtered image as `IP2_EX4_Plot1.png`.

   Checkpoint (4/7): Show your TA the output of the code. What are some similarities and differences between the original image and the filtered image?

4. Run the code four more times, setting the filter radius to 0.4, 0.3, 0.2, and 0.1 and saving the resulting final filtered images, in order, as:

   (a) `IP2_EX4_Plot2.png`
   (b) `IP2_EX4_Plot3.png`
   (c) `IP2_EX4_Plot4.png`
   (d) `IP2_EX4_Plot5.png`

   Deliverable (4/17): In your lab report, discuss how changing the radius of the filter affected the resulting filtered image.

   Deliverable (5/17): Include the following files in your report:

   - `IP2_EX4.m`
   - `IP2_EX4_Plot1.png`
   - `IP2_EX4_Plot2.png`
   - `IP2_EX4_Plot3.png`
   - `IP2_EX4_Plot4.png`
   - `IP2_EX4_Plot5.png`

5.5 Exercise 5: Further Exploring Round LPFs Applied to Noise

1. Create a new MATLAB script title `IP2_EX5.m` and copy in the code from Exercise 4.

2. Change the filter so that it is a LPF with a maximum filter radius of 0.006.

   - This will only capture the DC component and the fundamental frequency in each direction.

3. Run the program and zoom in on the center of Figure 3. You should see a "+" that indicates that there are only five frequency components passed through.

   Checkpoint (5/7): Show your TA the reconstructed image from the program with filter radius set to 0.006.

4. Create a sixth figure and use `imagesc` to visualize the reconstructed image `xfiltered` in this figure, but *remove* the `[0, 255]` scaling from this `imagesc` command. Save this image as `IP2_EX5_Plot6.png`.

5. Add a seventh figure as follows, saving the displayed image as `IP2_EX5_Plot7.png`:

```
figure(7); clf
surfc(xfiltered); colormap gray; shading interp; colorbar
```

6. Run your code a few times and observe how the last figures change.

   Deliverable (6/17): In your lab report, discuss what the image and surface look like. What is the relationship between the two?

   Deliverable (7/17): Include the following files in your report:

   - `IP2_EX5.m`
   - `IP2_EX5_Plot6.png`
   - `IP2_EX5_Plot7.png`

5.6 Exercise 6: Exploring Rectangular LPFs Applied to Coins

1. Create a MATLAB script and title it `IP2_EX6.m`. Copy in the code from Example 8.

2. Run the code with the following combinations of limits on $|u|$ and $|v|$. Title the final filtered image from each run with the given limits and your netID, and save each image as `IP2_EX6_PlotN.png`, with N ranging from 1 to 9.

   (a) `abs(u) < 0.5`, `abs(v) < 0.5`
   (b) `abs(u) < 0.5`, `abs(v) < 0.4`
   (c) `abs(u) < 0.5`, `abs(v) < 0.3`
   (d) `abs(u) < 0.5`, `abs(v) < 0.2`
   (e) `abs(u) < 0.5`, `abs(v) < 0.1`
   (f) `abs(u) < 0.4`, `abs(v) < 0.5`
   (g) `abs(u) < 0.3`, `abs(v) < 0.5`
   (h) `abs(u) < 0.2`, `abs(v) < 0.5`
   (i) `abs(u) < 0.1`, `abs(v) < 0.5`

   Deliverable (8/17): In your lab report, discuss how changing the limits on $|u|$ and $|v|$ changed the resulting filtered image.

   Deliverable (9/17): Include the following files in your report:

   - `IP2_EX6.m`
   - `IP2_EX6_Plot1.png`
   - `IP2_EX6_Plot2.png`
   - `IP2_EX6_Plot3.png`
   - `IP2_EX6_Plot4.png`
   - `IP2_EX6_Plot5.png`
   - `IP2_EX6_Plot6.png`
   - `IP2_EX6_Plot7.png`
   - `IP2_EX6_Plot8.png`
   - `IP2_EX6_Plot9.png`

5.7 Exercise 7: Exploring Rectangular LPFs Applied to Noise

1. Create a new MATLAB script titled `IP2_EX7.m` and copy in the code from Example 8 again, or the code you just used for Exercise 6.

2. Change the original matrix `x` so instead of the coins image it is a 399×399 matrix of random integers between 0 and 255.

3. Run the code with the following limits on $|u|$ and $|v|$. Title the final filtered image from each run with the given limits and your netID, and save each image as `IP2_EX7_PlotN.png`, with N ranging from 1 to 4.

(a) `abs(u) < 0.5`, `abs(v) < 0.5`
(b) `abs(u) < 0.1`, `abs(v) < 0.5`
(c) `abs(u) < 0.5`, `abs(v) < 0.1`
(d) `abs(u) < 0.1`, `abs(v) < 0.1`

Deliverable (10/17): In your lab report, discuss how changing the limits on $|u|$ and $|v|$ changed the resulting filtered image.

Deliverable (11/17): Include the following files in your report:

- `IP2_EX7.m`
- `IP2_EX7_Plot1.png`
- `IP2_EX7_Plot2.png`
- `IP2_EX7_Plot3.png`
- `IP2_EX7_Plot4.png`

5.8 Exercise 8: Further Exploring Rectangular LPFs Applied to Noise

1. Create a new MATLAB script titled `IP2_EX8.m` and copy in the code from Exercise 5.
2. Change the filter to be a rectangular filter with limits of 0.006 in each direction.
   - Zoom in on the center of Figure 3 and you should see a square that indicates that there are nine frequency components passed through.
3. Run your code several times.

   Checkpoint (6/7): Show your TA the code's output. How do the last figures change from run to run?
4. From any one run, save Figure 6 as `IP2_EX8_Plot6.png` and save Figure 7 as `IP2_EX8_Plot7.png`.

   Deliverable (12/17): In your lab report, answer the following questions:
   - What do the image and surface look like?
   - How do they compare to Exercise 5's figures?

   Deliverable (13/17): Include the following files in your report:
   - `IP2_EX8.m`
   - `IP2_EX8_Plot6.png`
   - `IP2_EX8_Plot7.png`

   NOTE: if you want to look at the narrow filters with the coins, you need to change the limits to reflect the largest dimension of the image. Each normalized frequency is some integer multiple of $2/\max(N, M)$. To get the 3x3 square filter for the coins, set your limits to be $< 3/298$. If you want a 5x5 box, use $5/298$.

5.9 Exercise 9: Exploring Rectangular HPFs Applied to Coins

1. Create a MATLAB script titled `IP2_EX9.m` and copy in the code from Example 9.

2. Run the code with the following combinations of limits on $|u|$ and $|v|$. Title the final filtered image from each run with the given limits and your netID, and save each image as `IP2_EX9_PlotN.png`, with N ranging from 1 to 9.

   (a) `abs(u) > 0.1`, `abs(v) > 0.1`
   (b) `abs(u) > 0.1`, `abs(v) > 0.3`
   (c) `abs(u) > 0.1`, `abs(v) > 0.5`
   (d) `abs(u) > 0.1`, `abs(v) > 0.7`
   (e) `abs(u) > 0.1`, `abs(v) > 0.9`
   (f) `abs(u) > 0.3`, `abs(v) > 0.1`
   (g) `abs(u) > 0.5`, `abs(v) > 0.1`
   (h) `abs(u) > 0.7`, `abs(v) > 0.1`
   (i) `abs(u) > 0.9`, `abs(v) > 0.1`

   Deliverable (14/17): In your lab report, discuss how changing the limits on $|u|$ and $|v|$ changed the resulting filtered image.

   Deliverable (15/17): Include the following files in your report:

   - `IP2_EX9.m`
   - `IP2_EX9_Plot1.png`
   - `IP2_EX9_Plot2.png`
   - `IP2_EX9_Plot3.png`
   - `IP2_EX9_Plot4.png`
   - `IP2_EX9_Plot5.png`
   - `IP2_EX9_Plot6.png`
   - `IP2_EX9_Plot7.png`
   - `IP2_EX9_Plot8.png`
   - `IP2_EX9_Plot9.png`

5.10 Exercise 10: Exploring Rectangular BPFs Applied to Coins

1. Create a new MATLAB script titled `IP2_EX10.m` and copy in the code from either Example 9 or Exercise 9.

   Checkpoint (7/7): Discuss with your TA how you would modify the filter to make it into a bandpass filter.

2. Based on your discussion with your TA, modify the filter to make it a rectangular band-pass filter that will keep $0.2 < |u| < 0.4$ and $0.3 < |v| < 0.5$.

   - *Hint:* your filter should resemble a sideways digit 0 on a seven-segment display.

3. Run the code and save the filter image, filtered FFT image, and final filtered image as `IP2_EX10_Plot3.png`, `IP2_EX10_Plot4.png`, and `IP2_EX10_Plot5.png`, respectively.

   Deliverable (16/17): In your lab report, discuss how changing the filter to a band-pass filter changed the resulting filtered image.

   Deliverable (17/17): Include the following files in your report:

   - `IP2_EX10.m`
   - `IP2_EX10_Plot3.png`
   - `IP2_EX10_Plot4.png`
   - `IP2_EX10_Plot5.png`

6 Lab Report

Like Lab 5, for your lab report, you should complete the 17 deliverables listed in the body of this manual. There is a LaTeXskeleton available on Canvas as well which you can use to format images and code.

Overall, the deliverables cover the following components:

6.1 Discussion (40 points)

In your lab report, respond to the following prompts:

1. Discuss how changing the radius of the filter in Exercise 3 affected the resulting filtered image. (5 points)
2. Discuss how changing the radius of the filter in Exercise 4 affected the resulting filtered image. (5 points)
3. Describe what the image and the surface in Exercise 5 look like. What is the relationship between the two? (5 points)
4. Discuss how changing the limits on $|u|$ and $|v|$ in Exercise 6 changed the resulting filtered image. (5 points)
5. Discuss how changing the limits on $|u|$ and $|v|$ in Exercise 7 changed the resulting filtered image. (5 points)
6. Describe what the image and the surface in Exercise 8 look like. How do they compare to Exercise 5's figures? (5 points)
7. Discuss how changing the limits on $|u|$ and $|v|$ in Exercise 9 changed the resulting filtered image. (5 points)
8. Discuss how changing the image filter to a band-pass filter in Exercise 10 changed the resulting filtered image. (5 points)

6.2 Graphics (42 points)

Include the following images in your lab report:

1. Exercise 3 (5 points)
   - IP2_EX3_Plot1.png
   - IP2_EX3_Plot2.png
   - IP2_EX3_Plot3.png
   - IP2_EX3_Plot4.png
   - IP2_EX3_Plot5.png
2. Exercise 4 (5 points)
   - IP2_EX4_Plot1.png
   - IP2_EX4_Plot2.png
   - IP2_EX4_Plot3.png
   - IP2_EX4_Plot4.png
   - IP2_EX4_Plot5.png
3. Exercise 5 (2 points)
   - IP2_EX4_Plot6.png
   - IP2_EX4_Plot7.png

4. Exercise 6 (9 points)
    - IP2_EX4_Plot1.png
    - IP2_EX4_Plot2.png
    - IP2_EX4_Plot3.png
    - IP2_EX4_Plot4.png
    - IP2_EX4_Plot5.png
    - IP2_EX4_Plot6.png
    - IP2_EX4_Plot7.png
    - IP2_EX4_Plot8.png
    - IP2_EX4_Plot9.png
5. Exercise 7 (4 points)
    - IP2_EX4_Plot1.png
    - IP2_EX4_Plot2.png
    - IP2_EX4_Plot3.png
    - IP2_EX4_Plot4.png
6. Exercise 8 (2 points)
    - IP2_EX4_Plot6.png
    - IP2_EX4_Plot7.png
7. Exercise 9 (9 points)
    - IP2_EX4_Plot1.png
    - IP2_EX4_Plot2.png
    - IP2_EX4_Plot3.png
    - IP2_EX4_Plot4.png
    - IP2_EX4_Plot5.png
    - IP2_EX4_Plot6.png
    - IP2_EX4_Plot7.png
    - IP2_EX4_Plot8.png
    - IP2_EX4_Plot9.png
8. Exercise 10 (6 points)
    - IP2_EX4_Plot3.png
    - IP2_EX4_Plot4.png
    - IP2_EX4_Plot5.png

6.3 Code (18 points)

Include the code from the following files in your report:

- IP2_EX1and2.m (10 points)
- IP2_EX3.m (1 point)
- IP2_EX4.m (1 point)
- IP2_EX5.m (1 point)
- IP2_EX6.m (1 point)
- IP2_EX7.m (1 point)
- IP2_EX8.m (1 point)
- IP2_EX9.m (1 point)
- IP2_EX10.m (1 point)

7 Revision History

- v2: October 2024 - Added a prelab; made multiple adjustments based on recommendations of Jenny Green (Pratt '25), Eduardo Bortolomiol (Pratt '26), and Adam Davidson.
- v1.1: Fall 2021
  - Minor corrections.
- v1.0: Spring 2021
  - First published version!