The main objective of this project is to first learn how to conduct convolution in the frequency domain, and then compare the performance of doing convolution in the spatial vs. frequency domain.

1 Background

One of the benefits of frequency-domain analysis is that convolution can be performed in the frequency domain by multiplication, which is a much simpler calculation as compared to convolution. Suppose in the time domain, we need to calculate

\[ y[n] = x[n] * h[n]. \]

Also suppose \( Y(e^{j\omega}) \), \( X(e^{j\omega}) \), and \( H(e^{j\omega}) \) are the Fourier transforms of \( y[n] \), \( x[n] \), and \( h[n] \), respectively. Then in the frequency domain, we’ll have

\[ Y(e^{j\omega}) = X(e^{j\omega})H(e^{j\omega}) \]

Now if we perform the inverse Fourier transform (ift), we’ll get

\[ y[n] = ift(Y(e^{j\omega})) \]

When actually implementing the above procedure, there’re two caveats that you need to keep in mind. The first is that we know usually the length of \( h[n] \) is shorter than that of \( x[n] \). The length of the Fourier transform is the same as that of the corresponding time-domain signal. This would create a problem in multiplication, as that multiplication is element-wise multiplication, requiring \( X(e^{j\omega}) \) and \( H(e^{j\omega}) \) to have the same dimension. That is, \( x[n] \) and \( h[n] \) should have the same dimension. So what do we do? We pad \( h[n] \) with zeros to make it the same length as \( x[n] \)!

The second caveat is about the boundary effect. If we would like \( y[n] \) obtained using the frequency-domain method to look exactly like the one from time-domain convolution, then we have to take care of aliasing occurred at the boundary of the
signal (or images). To do so, we need to pad both \(x[n]\) and \(h[n]\) to the length of \(p + q - 1\), where \(p\) is the length of \(x[n]\) and \(q\) is the length of \(h[n]\).

In order to make the Fourier transform and the inverse Fourier transform as computationally efficient as possible, we usually pad \(x[n]\) and \(h[n]\) to the length of the smallest power of 2 that is larger than \(p + q - 1\). Since we are using images in this project, in the following, I use image as an example to show the detailed procedure.

Step 1: Padding. Suppose the size of the image is 256 \(\times\) 256, then we should pad it for 512 \(\times\) 512.

\[
[m, n] = \text{size}(x); \quad \text{get the dimension of image x}
\]

\[
xpad = \text{zeros}(512,512); \quad \text{allocate space for the padded x}
\]

\[
xpad(1:m,1:n) = x; \quad \text{copy x to the upper-left corner of xpad}
\]

% we need to do the same for h.
% I’ll leave it to you to figure out how

Step 2: Fourier transform.

\[
X = \text{fft2}(xpad); \quad \\text{fast Fourier Transform of images}
\]

% again, do the same for hpad

Step 3: Multiplication

\[
Y = X .* H \quad \text{element-wise multiplication}
\]

Step 4: Inverse transformation

\[
ypad = \text{ifft2}(Y);
\]

Step 5: Cut the upper-left corner which is where your convolution image resides. You might have noticed that the output you got from the frequency domain has a black border but only at the upper and left sides of the image, while the output from the time domain has the black border on all four sides. This is because the output from \texttt{conv2()} is of size \((m x + m h - 1, n x + n h - 1)\), and the output from \texttt{conv2f()} is of size \((m x, n x)\). So please note the changes in the following code listing. Then you should get exactly the same image from the time-domain as compared to the frequency domain, including the borders.

\[
\text{ypad} = \text{ypad}(1:m,1:n);
\]

\[
y = \text{ypad}(1:mx+mh-1, 1:nx+nh-1);
\]
2 Tasks

1. (30 pts) Write a function \( \text{function } [y] = \text{conv2f}(x, h) \) where \text{conv2f} represents the implementation of convolution between \( x \) and \( h \) in the frequency domain. This function should implement the procedure described in Sec. 1. Within the function, plot a \( 2 \times 2 \) figure of \( X \), \( H \), \( Y \), and \( y \). Use the cam-eraman image \( \text{http://web.eecs.utk.edu/~hqi/ece315/project/cameraman.pgm} \) and convolution kernel \( h \) (i.e., a 2-D impulse response)

\[
h = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}
\]

There are three items you need to take notes when plotting the Fourier transforms:

- First, since \( X \), \( H \), and \( Y \) will all be matrices of complex numbers, when you plot it, use \text{mag()} \text{abs()} to only plot the magnitude.

- Second, from the magnitude plot, you somehow could not tell the pattern. It feels like the important features are everywhere in the plot. This is because the default origin of the Fourier transform is at the upper-left corner. In order to move the origin to be at the center, you should apply \text{fftshift()} outside \text{fft()} and before applying \text{ifft()} you should shift it back. That is, every place where you have \text{fft()}, use \text{fftshift(fft())}, and every place where you have \text{ifft()}, use \text{ifft(fftshift())} instead. Figure \[1\] gives you a comparison of \( X \), \( H \), \( Y \) with and without applying \text{fftshift()}. To learn more on what \text{fftshift} does, try the \text{help}.

- Third, the DC component is usually a large number. If you don’t rescale it properly, you would probably either see an all-black image or an all-white image, or a bright spot at the center of the image, which overshadows the rest of the image. What do you do about it? This is a bonus point (+10). Well, I was going to leave it to you as bonus, but on second thought, this is probably too much. The bonus part is moved to Task 4. The trick is to first apply \text{log10} on the magnitude, and then rescale the pixel value to 0 and 255. For example, \text{imshow(uint8(rescale(log10(1+abs(H)), 0, 255)))}.

2. (40 pts) Write a Matlab code \text{testConv2f1.m} using both the spatial domain method (use the MATLAB function \text{conv2(())}) and the frequency do-
Figure 1: Demonstration of using \texttt{fftshift} (top row) vs. not using \texttt{fftshift} (bottom row). From left to right: the magnitude plots of $X$, $H$, and $Y$, respectively. Note that the rays in $X$ correspond to the straight lines in the original image $x$, and they are perpendicular to each other. For example, the vertical line in $X$ corresponds to the horizontal edges in the image $x$. The magnitude plot of $H$ is the Fourier transform of the Gaussian lowpass filter which is probably smoother than what you have (a simple average lowpass filter). You should realize this is a low-pass filter, where high-frequency contents close to the boundary are blocked and only low-frequency contents towards the middle are kept in $Y$. 
main method (use the conv2f function you just wrote) to calculate the convolution between the cameraman image and the blur kernel in Task 1. Compare the results both visually and by reporting the execution time. Use the MATLAB function tic and tac. Plot a $2 \times 1$ figure to compare visually the result from the time-domain convolution and the frequency-domain convolution. Also report the execution time as caption of the figure. Indicate the kernel size in the caption too. Remember to comment out all the code in conv2f.m related to plotting, so that the comparison related to compute time would be fair.

3. Write a Matlab code, testConv2f2.m. Generate convolution kernels similar as Task 2 but of different sizes and compare the performance. Try at least 4 different sizes, e.g., $7 \times 7, 25 \times 25, 49 \times 49$, etc. Plot two figures. Use Matlab function figure(). The first figure is $4 \times 2$ with visual comparison of the convolution results from the spatial and frequency domains using the four kernels of different sizes. In the caption, specify the kernel size. In the second figure, plot a bar chart (or whichever ways that serve the purpose) comparing the different run times from the two methods when different kernel sizes are used.

4. Write a Matlab code, testAcorr.m that generate the autocorrelation image of cameraman.pgm in both spatial and frequency domains and compare the execution time. Autocorrelation is like a convolution with itself where the kernel is the image itself. Generate a $2 \times 1$ plot with results from time-domain and frequency domain. Again, in the caption, provide the run-times. Do some research online to and comment on where autocorrelation might be applied and why. Provide reference. This is the bonus +10 points.

3 What to Submit?

- a five-page report:
  - Page 1: The $2 \times 2$ plot specified in Task 1
  - Page 2: The $2 \times 1$ plot specified in Task 2
  - Pages 3 and 4: The two figures specified in Task 3.
  - Page 5: The $2 \times 1$ figure specified in Task 4 and the answer to the bonus point.

- the source code: a tar file that includes the four .m files.