ELEN 4810 Homework 5

Due Wednesday, November 18. You can submit your answers to the analytical questions via either
- Hardcopy submission at the beginning of class on Wednesday, November 18, or
- Electronic submission (in pdf form) on Courseworks. Please name it in the format of ‘yl3027_hw5_writeup.pdf’.

Thanks.

Analytical Questions

Please complete problems 10.24, 3.21, 3.23 in Oppenheim and Schafer (3rd Edition). Justify your answers!

Computational Questions

1 The Discrete Fourier Transform Matrix

Problem setting. For any discrete signal $x$ such that $\text{supp}(x) \subseteq [N]_0 \triangleq \{0, \ldots, N - 1\}$, we can represent $(x[n])_{n \in [N]_0}$ as a vector $x$ in $\mathbb{C}^N$. The DFT allows us to represent $x$ in the frequency domain by its coordinates $X[k]$ over the collection of $N$ basis vectors

$$e^{(k)} \triangleq \left( e^{j \frac{2\pi k n}{N}} \right)_{n \in [N]_0} \in \mathbb{C}^N \quad k = 0, \ldots, N - 1 ,$$

making computation over the frequency domain tractable for the class of signals with finite length. Let’s learn a little more about this process.

Before we go any further, please convince yourself that the set of vectors $\{e^{(k)}\}_{k \in [N]_0}$ are orthogonal, i.e.

$$\langle e^{(k)}, e^{(j)} \rangle \equiv \sum_{n=0}^{N-1} e^{j \frac{2\pi k n}{N}} \cdot e^{-j \frac{2\pi j n}{N}} = 0 \quad \forall i, j \in [N]_0, i \neq j$$

1 Recall $\text{supp}(x) \triangleq \{ n \in \mathbb{Z} : x[n] \neq 0 \}$. 
(hint: use the geometric sum). Thus the coordinates $X[k]$ of $x$ are given by the inner product

$$
X[k] = \langle x, e^{(k)} \rangle = \sum_{n=0}^{N-1} x[n] e^{-j \frac{2\pi k}{N} n},
$$

for $k = 0, \ldots, N - 1$. Representing $(X[k])_{k \in [N]}$ as a vector $X$, we have the linear system

$$
X = Fx \quad F \in \mathbb{R}^{N \times N}, \quad F_{k,n} = e^{-j \frac{2\pi k}{N} n}.
$$

The action of applying $F$ on a given vector $x \in \mathbb{C}^N$ can be done in MATLAB via the command $\text{fft}(x)$, where $x$ is simply the length $N$ array with entries from $x$.

For this problem we practice working with the DFT by creating a matrix $G$, with the following modifications from $F$:

1. Pick a random $N$ between 1-100, $x \in \mathbb{R}^N$ (or $\mathbb{C}^N$), and compare the $\ell_2$-norms of $x$ and $X$ by typing the commands

   ```matlab
   N = randi(100); x = randn(N,1); normx = norm(x); normX = norm(fft(x));
   ```

   Notice that $F$ does not preserve the $\ell_2$-norm: verify that $FF^* = NI$ and thus $\|Fx\|_2^2 = N \|x\|_2^2$.

   We would like $G$ to be a unitary matrix - a rotation of the coordinate axes in a vector space. Although $G$ may modify the coordinates from $x$, the $\ell_2$-norm is unaffected.

2. Recall from lecture that the DFT can be seen as a discrete sampling of the frequency values from the DTFT. We have

   $$
   X[k] = X(e^{j\frac{2\pi k}{N}}) \quad \forall k \in [N]_0.
   $$

   over an ordered set of $N$ frequencies in the interval $[0, 2\pi)$, however it is often preferable to instead sample frequencies over $[-\pi, \pi)$. To rearrange the frequency values, MATLAB circularly shifts $X$ via the function $\text{fftshift}()$. To see this in action try typing the commands

   ```matlab
   N=100; x=cos(2*pi*25/100*(0:N-1));
   stem((0:N-1)-ceil(N/2), abs(fftshift(fft(x))));
   ```

   We would also like $G$ to correspond to $X(e^{j\omega})$ over the range $[-\pi, \pi)$.

**Tasks and remarks [3 pts].** For the sake of simplicity, we will assume $N$ is even. Please write the function $\text{sudftmtx}()$ - sticking to the following definition (names, inputs, outputs, ordering, etc.)

$$
[G, freqs] = \text{sudftmtx}(N),
$$

so that it takes as input an even integer $N$ and firstly produces a matrix $G \equiv G \in \mathbb{C}^{N \times N}$, with the following modifications from $F$:

1. $G$ is unitary: $GG^* = G^*G = I$;
2. Applying $G\ast x$ produces exactly the same result as $\text{fftshift}(\text{fft}(x))/\sqrt{N}$, i.e.

$$Gx = \frac{1}{\sqrt{N}} \left[ X \left( e^{-j(\pi + \frac{2\pi k}{N})} \right) \right]_{k=[N]_0} \in \mathbb{C}^N.$$  

Second, `sudftmtx()` produces the ordered set of frequencies $\text{freqs}$ centered at zero, taking the values

$$\text{freqs}(k + 1) = \frac{1}{\pi} \left( -\pi + \frac{2\pi k}{N} \right), \quad \forall k \in [N]_0,$n
and notice the frequencies have been normalized to live in the interval $[-1, 1)$.

Finally, a few tips that you may find helpful:

- MATLAB code runs significantly smoother if loops are avoided. On that note, observe that for a set of scalars $k_i, n_i, i = 1, \ldots, N$,

$$\begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_N \end{bmatrix} \begin{bmatrix} n_1 & n_2 & \cdots & n_N \end{bmatrix} = \begin{bmatrix} k_1 n_1 & k_1 n_2 & \cdots & k_1 n_N \\ k_2 n_1 & k_2 n_2 & \cdots & k_2 n_N \\ \vdots & \vdots & \ddots & \vdots \\ k_N n_1 & k_N n_2 & \cdots & k_N n_N \end{bmatrix}.$$  

- One way to rearrange the frequency values is to take advantage of periodicity and perform a circular shift: the function `circshift()` may be useful.

Overall this problem is pretty easy and should take up to no more than 20 lines of code.

Observations.

- How would we modify the construction $G$ to work with odd $N$?

2 Lowpass Transform Encoding

Problem setting. Next lets put the DFT to use by implementing a simple image encoder and decoder (in the spirit of JPEG). The idea here is that natural images tend to have very small high frequency components. Thus we can reduce information needed to describe each image by discarding these components, with the expectation that the image quality will not be severely affected.

Practically, we want a coding scheme that works for images of varying sizes without using a different DFT sizes for each image. We do this by dividing each image into small $N \times N$ patches, with $\log_2 N \in \mathbb{N}$. The basic encoder-decoder scheme is as follows:

1. Encoder.
   
   (a) Preprocessing: zeropad the image so that the length and width of the image so that it can be covered (not necessarily partitioned; see discussion below) by $N \times N$ patches.
   
   (b) For each patch, take the $N \times N$ 2D DFT, and throw away the high frequency components.
   
   (c) Return the remaining frequency values for each patch, as well as information about the frequency and patch locations.
2. **Decoder.**

(a) For each patch, take the frequency values retained and reproduce the lowpassed DFT for that patch. Pass this through the inverse $N \times N$ 2D DFT to produce an approximation of the patch image.

(b) ‘Stitch’ the set of patch images back together to approximate the input image.

A problem with partitioning the image into $N \times N$ patches - cutting evenly into squares - and simply discarding high frequency components is that pixel values on adjacent patch edges may not line up. One primitive way to improve such border effects is to overlap the patches so they cover the image, and average out the pixel values in the overlapping regions during reconstruction.

**Patching scheme.** In this problem we will overlap the patches via the scheme illustrated by the figure below:

![Patching scheme figure]

**Figure 1: Patching scheme.** A) An individual patch of size $N \times N$ consists of a border with overlap width $w \in \{0, \ldots, \left\lfloor \frac{N}{2} \right\rfloor \}$ and a subpatch of size $M \times M; M = N - 2w$. B) The input image before preprocessing is indicated by the rectangle in the bold red border, and is covered by patches in the order indicated by the numbers and the blue arrows. The dot-dashed border in red represents the zeropadding of the input image required for this scheme.

**Removal of high frequency components.** To remove high frequency components in each patch, we retain the pixel values of the $N \times N$ 2D DFT only within a region centered around the zero frequency index, illustrated by the following figure,
Notice that we are setting a bound on the $\ell_2$-norm on the frequency values that are accepted. Specifically, suppose the frequencies of the DFT are indexed via $(u, v)_{u,v \in [N]}$, and $(u_0, v_0)$ is the zero-frequency index. Then we wish to keep only the frequency values satisfying

$$\frac{(u - u_0)^2 + (v - v_0)^2}{[N/2]^2} < f_c^2,$$

for some $f_c > 0$. The idea here is that if $f_c \approx 1$, then the white circle reaches the edge of the patch, and all horizontal and vertical frequency information will be retained - leaving out some diagonal frequency information. For $f_c > \sqrt{2}$ all frequency information is retained.

**Indexing.** The point of this problem is to familiarize you with the application of DFT in a practical setting. As a result we will deal with most of the tedious indexing for you in the skeleton code. However, please keep in mind a few indexing conventions when doing this problem:

- We will declare the location of each patch to be the index of its the top-left corner.
- Note the MATLAB dimension ordering convention: in a 2-D array, the number of rows is referenced as its size in the first dimension (i.e. `size(A,1)`) and the number of columns its size in the second dimension. In a 3-D array, the number of matrices stacked is its size in the third dimension (`size(A,3)`), etc.
- Also we will use the MATLAB linear indexing order: for a 2D-array $A$, calling $A(i)$ for $i \in \text{numel}(A)$ will grab the $i$-th element of $A$ by going down the row index for a fixed column, then moving to the next column, etc. - consistently with MATLAB dimension ordering.

  - For an illustration, see the blue arrows in Figure 1(b).
  - e.g. If $A = [1 \ 2; \ 3 \ 4]$, then $A(:,1) = [1 \ 3 \ 2 \ 4]$; and `reshape([1 \ 3 \ 2 \ 4], [2 \ 2])` gives $A$. 

**Figure 2: Removal of high frequency components.** The $N \times N$ 2D DFT patch is thus `windowed`; all pixel values in the white region are kept the same and all pixel values in the black region are set to zero.
Tasks and remarks [7 pts total]. Please write the functions `lowpass_encode()` and `lowpass_decode()` - sticking to the following definitions

\[
[ \text{lpary}, \text{plocs}, \text{flocs} ] = \text{lowpass}\_\text{encode}( \text{grayimary}, \log2N, \text{ovlp}, \text{cfq} ),
\]

\[
[ \text{dcdimary} ] = \text{lowpass}\_\text{decode}( \text{lpary}, \log2N, \text{ovlp}, \text{plocs}, \text{flocs}, \text{imsize} ).
\]

For this problem, we expect you to use the skeleton code. In addition to guiding you through the problem, they contain subfunctions that play a significant part in the encoder and decoder. Please read the following descriptions carefully.

Encoder [4 pts]. The encoder takes in a 2D array `grayimary` as the input grayscale image. For this problem we have provided reference images `cameraman.tif` and `telescopepenguin.jpg` as examples. To convert this to the required format for `grayimary`, use the function `imread()`.

The function also takes in parameters

- `log2N`, a positive integer which determines the patch size \( N = 2^{\log2N} \);
- the overlap width `ovlp`, which should be a small positive integer less than \( \left\lfloor \frac{N}{2} \right\rfloor \), and
- the cutoff frequency `cfq`, a positive real number as specified by \( f_c \) in Eqn. (7).

We will basically follow the outline for the encoder in the problem setting to produce the necessary output:

1. **Zeropadding and dividing the image into patches.** We will essentially do this for you. In `lowpass_encode.m`, we have included the subfunction

\[
[ \text{im expd}, \text{plocs} ] = \text{expdimg}( \text{grayimary}, N, \text{ovlp} ),
\]

which returns the zeropadded image `im expd` and the locations `plocs` of the \( N \times N \) patches from Figure 1(b). The array `im expd` is of a larger size than `grayimary`, and basically is zeropadded to contain the partition of the image by the \( M \times M \) subpatches, as well as the overlap. The array `plocs` is of size \( n_p \times 2 \), where \( n_p \) is the number of patches, and each row contains the row and column indices of a patch location, as per MATLAB dimension ordering.

Note that `expdimg()` does not actually return you the pixel patches themselves, but you should be able to grab the patches easily with `plocs` and `im expd`.

2. **Discarding high frequencies.** For each \( N \times N \) patch, consider the application of

\[
\text{fftshift(fft2(patch))},
\]

to produce an array where the zero-frequency index \((u_0, v_0)\) is somewhere near the center. Your code should solve the following questions:

(a) How to compute the zero frequency index \((u_0, v_0)\)?
(b) Given `cfqs`, which indices \((u, v)\) are frequency values retained from equation (7)? Return, i. a logical \( N \times N \) array `fkeep`, with value 1 to indicate a frequency location kept, and 0 otherwise.
ii. the $n_f \times 2$ array $\text{flocs}$ of frequency locations, where $n_f$ is the number of frequencies kept, in the same format as $\text{plocs}$. Each row should be a row-column index pair, and the going down the rows should give retained frequency locations in MATLAB linear indexing order - down first column, next column, etc., in the $N \times N$ DFT grid.

3. Finally, by using $\text{plocs}$ to get the patches, apply $\text{fftfshift(fft2())}$ to each patch. Then use $\text{fkeep}$ or $\text{flocs}$ to produce the $n_f \times n_p$ array $\text{lpary}$, where each of the $n_p$ columns contains the frequency values corresponding to $\text{flocs}$ for a single patch.

Each task should be solvable in a few lines of code - see skeleton file.

**Decoder [3 pts]**. The decoder takes all the problem parameters and information provided by the encoder, as well as the original image size $\text{imsize}$, to 'stitch' together an approximation to the original image $\text{dcdimary}$.

1. **IDFT**. Construct the stack of patches. Approximate each patch by taking the corresponding column from $\text{lpary}$, putting each frequency value down into the corresponding location from $\text{floc}$ into a $N \times N$ array - say, $\text{DFTary}$ - then apply the command

   $$\text{ifft2(fftshift(DFTary))}.$$ 

*Hint:* Avoid looping when allocating frequency locations by converting the set of subscript indices $\text{flocs} = (u_j, v_j)_{j \in [n_f]}^2$ into linear indices, say $\text{flins} = [i_j]_{j \in [n_f]}^j$. Then you can simply assign, for example, $\text{lpary}(\text{flins}) = \text{lpary}(1:l)$ for the $l$-th patch.

2. **Stitching**. Again we essentially do this part for you. The skeleton file $\text{lowpass_decode.m}$ contains a subfunction

   $$\text{[ dcdimary ] = stitch( patches, plocs, ovlp, imsize ),}$$

where $\text{patches}$ is an $N \times N \times n_p$ array containing a stack of patches. The function returns a reconstruction of the original image, with size $\text{imsize}$. This is done in the manner described in the discussion on patching in the problem setting.

Again, each task should be solvable in a few lines of code - see skeleton file.

**Observations.**

- How many values are returned in total from $\text{lpary}$, $\text{plocs}$ and $\text{flocs}$ (in terms of $n_p$ and $n_f$)? How would you compute the compression ratio provided by the encoder?

- Try running the encoder and decoder on the images with various values for $\log2N$, $\text{ovlp}$ and $\text{cfq}$. For any fixed $\log2N$, what is the best achievable compression ratio while keeping image quality acceptable? Use $\text{imagesc()}$ along with $\text{colormap(gray)}$ to display images.

- This scheme is actually pretty poor compared to industrial standards. Considering the encoder returns $\text{flocs}$, what are some immediate ways to improve the encoding scheme?

- Notice that setting $\text{ovlp} > 0$ improves border effects, but only marginally. What other artifacts can you notice, and what approaches can you suggest to mitigate these effects?

\[2\text{Recall that } [n_f] \triangleq \{1, \ldots, n_f\}, \text{ as opposed to } [n_f]_0.\]
3 Submission

Submission Instructions. Please upload and submit the completed functions `sudftmtx.m`, `lowpass_encode.m` and `lowpass_decode.m` directly (no `.zip` file) to Courseworks before the beginning of class on Wednesday, Nov. 18. Please do not submit observations, or image files.