# EECE-5626 (IP&PR) : Midterm-2 Exam : 2021-FALL

## Friday November 26 for 160 Minutes between 12 noon -- 4 pm (EST)

## [50-Points]

---

**Instructions:**

1. You are required to read the NU Academic Integrity policy (given below) and sign below that the submitted work is your own work. This is a COE requirement.
2. You are required to complete this exam using Live Editor.
3. All your plots must be properly labeled and should have appropriate titles to get full credit.
4. Use of the equation editor to typeset mathematical material such as variables, equations, etc., is highly recommended. However, in the interest of time, a properly scanned image of a neatly hand-written fragment can be inserted as your work at the required space.
5. After completing this assignment, export this Live script to PDF. In the exam you will be submitting the PDF file. **DO NOT SUBMIT LIVE EDITOR (.MLX) FILE**. **IT WILL NOT BE GRADED**.
6. You should try to complete this exam in 150 minutes. Use additional ten minutes for submission activities. You will have only one chance for submission.

**Academic Integrity Policy**

A commitment to the principles of academic integrity is essential to the mission of Northeastern University. The promotion of independent and original scholarship ensures that students derive the most from their educational experience and their pursuit of knowledge. Academic dishonesty violates the most fundamental values of an intellectual community and undermines the achievements of the entire University.

As members of the academic community, students must become familiar with their rights and responsibilities. In each course, they are responsible for knowing the requirements and restrictions regarding research and writing, examinations of whatever kind, collaborative work, the use of study aids, the appropriateness of assistance, and other issues. Students are responsible for learning the conventions of documentation and acknowledgment of sources in their fields. Northeastern University expects students to complete all examinations, tests, papers, creative projects, and assignments of any kind according to the highest ethical standards, as set forth either explicitly or implicitly in this Code or by the direction of instructors. The full academic integrity policy is available at

http://www.northeastern.edu/osccr/academic-integrity-policy/

**Declaration**

By signing (i.e., entering your name in the given format below) and submitting this exam through the submission portal, I declare that I have read the Academic Integrity Policy and that the submitted work is my own work.

**Enter your name (Tyler B McKean)**:

**IF YOU DO NOT SIGN ABOVE, 10% POINTS WILL BE CUT FROM THE OVERALL SCORE.**

---

**Note:** All images required in this exam are available in a zip file named **`Midterm2-images.zip`**. Unzip this file and put its contents in your working directory.

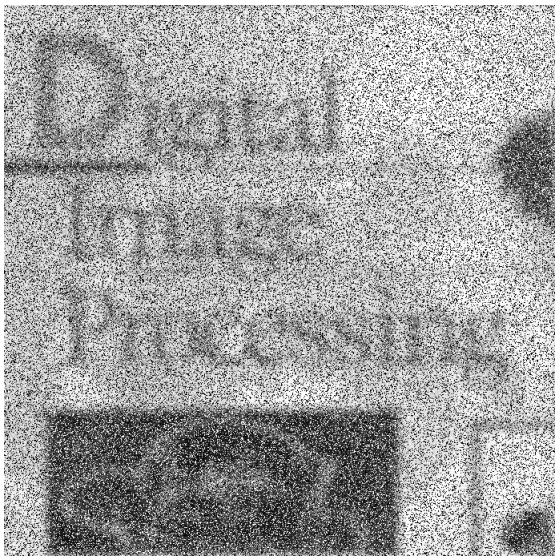
**Default Plot Parameters:** Execute the following code before any other code as well as everytime you restart this mlx file.

```
set(0,'defaultfigurepaperunits','points','defaultfigureunits','points');
set(0,'defaultaxesfontsize',10);
set(0,'defaultaxestitlefontsize',1.4,'defaultaxeslabelfontsize',1.2);
```

---

## Problem-1 [18-Points] Image Restoration

Consider the following image that is blurred and is contaminated by salt-and-pepper noise. It is available as **`DIP-distorted.tif`** image. We do not know how the blurring was done (i.e., which blurring kernel was used) or how much noise was added.



Since the additive noise is not a white (i.e., uncorrelated) Gaussian noise, the deblurring restoration methods discussed in class can not be used directly. Therefore, we want to use a combination of salt-and-pepper noise removal step followed by an appropriate deblurring restoration step to restore the above degraded image to its original quality ***as much as possible***.

### (a) [6-Points] Denoising

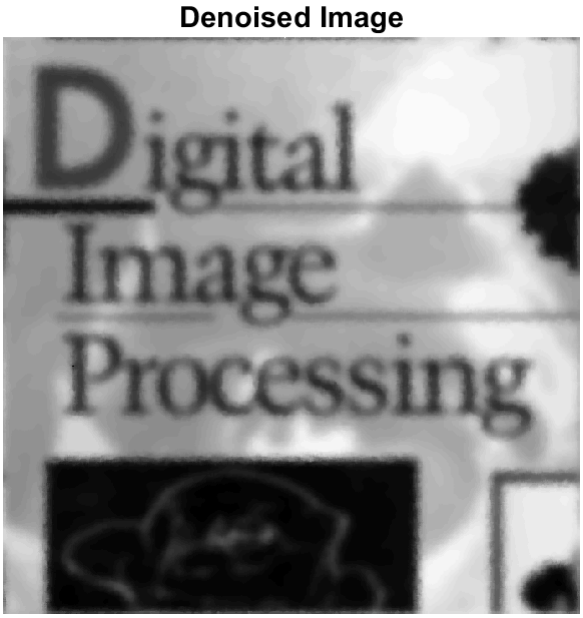
Read the **`DIP-distorted.tif`** image and store it as a variable **`fbn`** (for blurred, noisy) in MATLAB. Denoise this image so the result is a blurred image that can be restored in the next step. Keep in mind that the additive noise is of *salt-and-pepper* type. By trial and error, use the smallest possible kernel size that will remove the noise.

Use variable **`fblur`** to store your result. Display this blurred image.

**MATLAB script**:

```
clc; close all; clear;
```

```
% Enter your code below
fbn = imread('DIP-distorted.tif');
fblur = medfilt2(fbn,[7 7],'symmetric');
% Uncomment the following two lines for image display
figure('Position',[0,0,4,4.5]*72); subplot('Position',[0,0,1,1/4.5]);
imshow(fblur); title('Denoised Image'); pause(1)
```




---

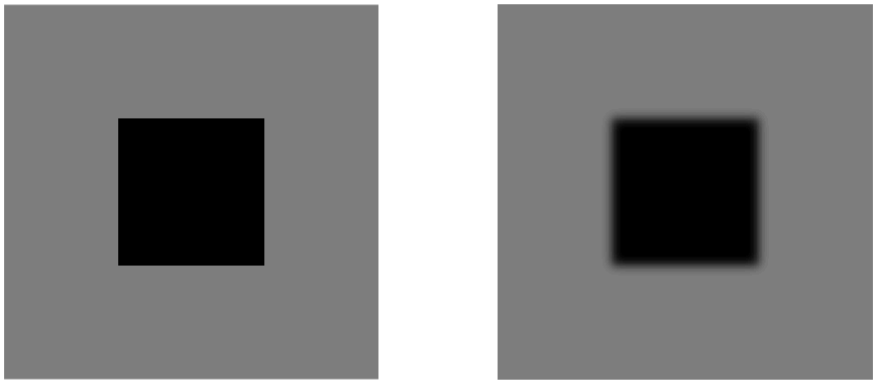
## (b) [6-Points] Determination of Blurring Kernel

By observing your denoised image above, estimate the blurring kernel that was used. First determine if the blurring is out-of-focus or a motion-blur and then estimate the size of the kernel. One approach to do this is to create an image (of the same as above) with a small black rectangle in a light gray background and blur that image using your estimated kernel to see if you get blurring similar to the big black rectangle in the **`fblur`** image. Adjust the size of the kernel until you are satisfied. Again you may have to do trial and error experimentation. However, you can use any approach you want to determine the blurring function.

Use variable **`psf`** for the blurring kernel.

**MATLAB script**:

```
% Enter your code below
psf = fspecial('gaussian',43,7);
image = intensityScaling(ones(512,512)./2);
image(157:356,157:356) = 0;
image_filt = imfilter(image,psf,'conv','same','symmetric');
figure, subplot(1,2,1), imshow(image), pause(1)
subplot(1,2,2), imshow(image_filt), pause(1)
```



Indicate the size and type (out-of-focus, Gaussian, motion-blur, etc.) of the blurring kernel that you estimated as your answer below.

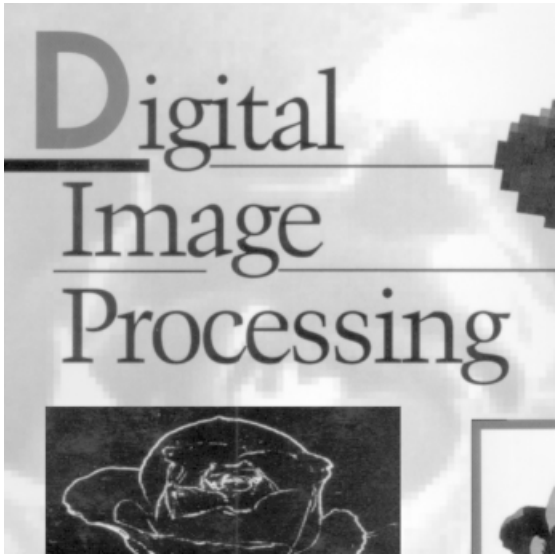
**Answer**: The type of noise appears to be a Gaussian noise and the filter kernel size I felt represented it was a $43 \times 43$ Gaussian kernel with $\sigma = 7$

---

## (c) [6-Points] Deblurring

Using the **`psf`** kernel and select any deblurring method (along with its appropriate parameter values) to obtain a restored image that appears close to the original image (shown below) in your opinion. Remember that the restoration will not be exact. Again, you will need to experiment with the parameters of the deblurring function.

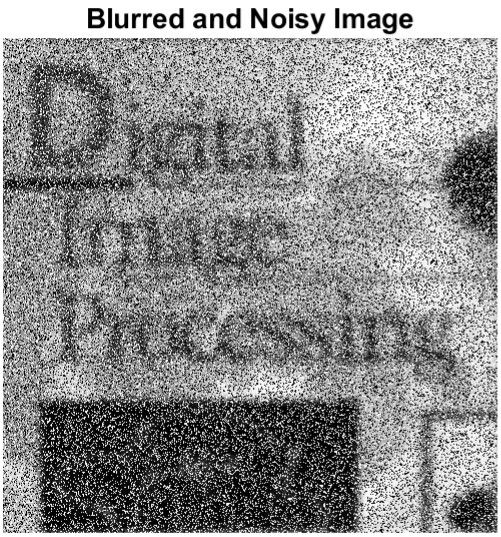
Use variable **`frest`** for this image and display it along with the **`fbn`** image. Comment on your final result.



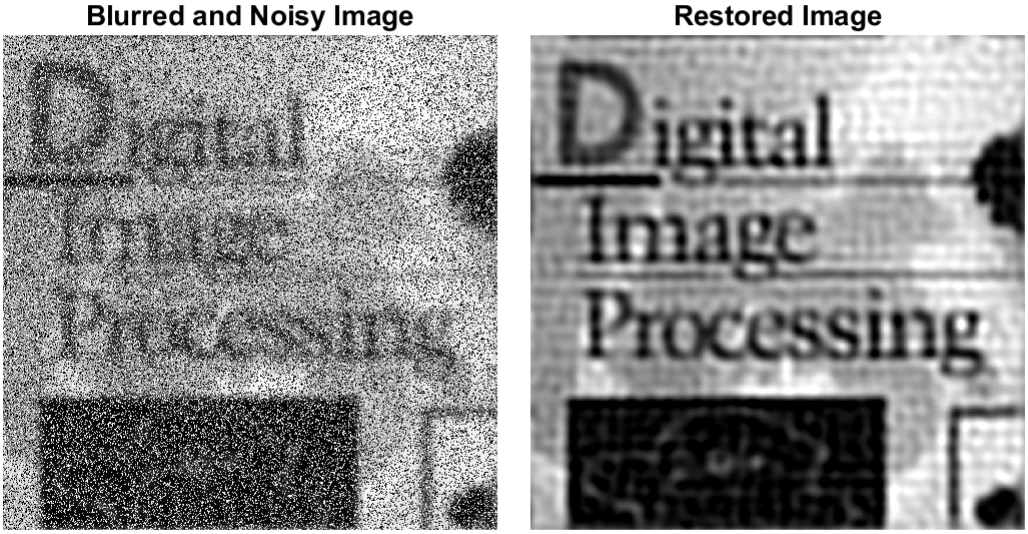

**MATLAB script**:

```
% Enter your code below
fblur = edgetaper(fblur,psf);
noise = fbn - fblur;
Sn = abs(fft2(noise)).^2;
nA = sum(Sn(:))/numel(noise);
Sf = abs(fft2(fblur)).^2;
fA = sum(Sf(:))/numel(fblur);
R = nA/fA;
NCORR = fftshift(ifft2(Sn));
ICORR = fftshift(ifft2(Sf));
frest = deconvwnr(fblur, psf,NCORR,ICORR);

% Uncomment the following four lines for image display
subplot('Position',[0,0,3.75/8.5,3.75/4]);
imshow(fbn); title('Blurred and Noisy Image'); pause(1)
```




```
subplot('Position',[4/8.5,0,3.75/8.5,3.75/4]);
imshow(frest); title('Restored Image'); pause(1)
```



**Comment**: Here I was not able to eliminate the ripple in the final image.

---

## Problem-2 [12-Points] Color Image Processing

This problem uses the color image **`Peppers-color.tif`**, shown below.

Let it be denoted by **`fRGB`** variable which is in the RGB color space. Let **`fYIQ`** variable denote the color image in the YIQ color space.

### (a) [5-Points] Enhancement Approach-1

Process each component of **`fRGB`** through the following enhancement filters. Display these images side by side.

1. An edge crispening filter using the $5 \times 5$ Laplacian with $-24$ in the center. Let the resulting image be denoted by **`LRGB`**.
2. An unsharp contrast enhancement filter with $\alpha = 0.5$. Let the resulting image be denoted by **`URGB`**.

**MATLAB script**:

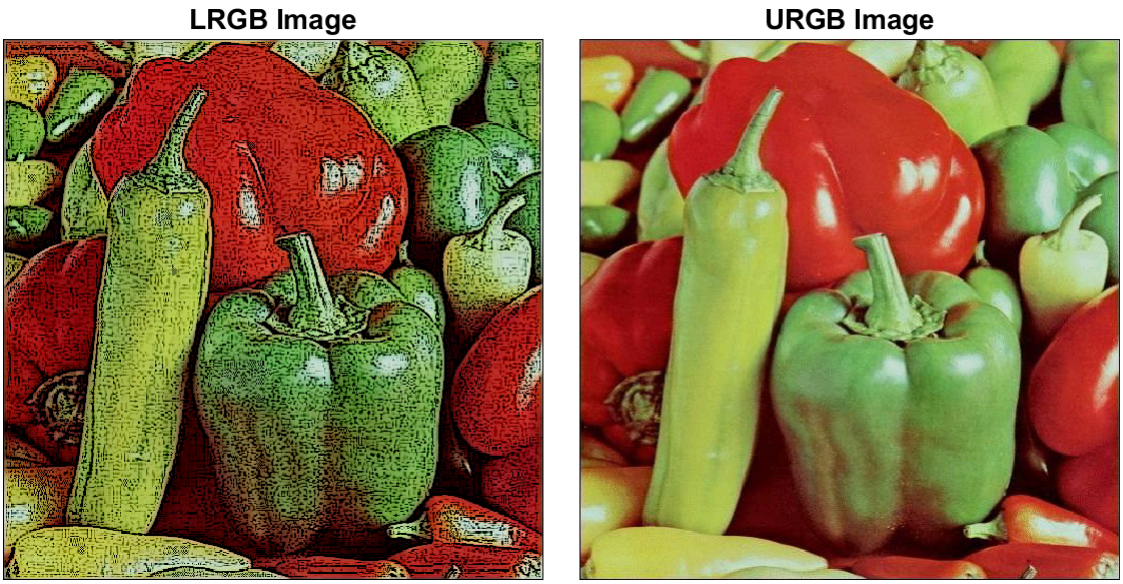
```
clc; close all; clear;
% Enter your code below.
lap = [1,1,1,1,1;1,1,1,1,1;1,1,-24,1,1;1,1,1,1,1;1,1,1,1,1]
```

```
lap = 5×5
     1     1     1     1     1
     1     1     1     1     1
     1     1   -24     1     1
     1     1     1     1     1
     1     1     1     1     1
```

```
fRGB = imread('Peppers_color.tif');
fR = fRGB(:,:,1);
fG = fRGB(:,:,1);
fB = fRGB(:,:,1);
```

```
fRfilt = imfilter(fR,lap,'same');
fGfilt = imfilter(fG,lap,'same');
fBfilt = imfilter(fB,lap,'same');
ffilt = cat(3, fRfilt, fGfilt, fBfilt);
LRGB = imsubtract(fRGB,ffilt);
fHSI = rgb2hsi(fRGB);
fI = fHSI(:,:,3);
Ieq = histeq(fI);
fHSIeq = fHSI;
fHSI(:,:,3) = Ieq;
URGB = hsi2rgb4e(fHSIeq);

% Display processed images (uncomment the following lines for display)
figure('Position',[0,0,8.5,4]*72);
subplot('Position',[0,0,3.75/8.5,3.75/4]);
imshow(LRGB); title('LRGB Image');
subplot('Position',[4/8.5,0,3.75/8.5,3.75/4]);
imshow(URGB); title('URGB Image');
```




Visually compare the above images and comment on the effectiveness of each technique.

**Comparison and Comments**:

Here the edges of the peppers image have been darkered and the enhanced image now has a more illuminated look to the colors

---

## (b) [5-Points] Enhancement Approach-2

Process the Y-component component of **`fYIQ`** through the following enhancement filters and then convert the resulting image back to RGB color space. Display these images side by side.

1. An edge crispening filter using the $5 \times 5$ Laplacian with $-24$ in the center. Let the resulting RGB-converted image be denoted by **`LYIQ`**.

2. An unsharp contrast enhancement filter with $\alpha = 0.5$. Let the resulting RGB-converted image be denoted by **`UYIQ`**.

**MATLAB script**:

```
% Enter your code below.
fYIQ = rgb2ntsc(fRGB);

% Display processed images (uncomment the following lines for display)
% figure('Position',[0,0,8.5,4]*72);
% subplot('Position',[0,0,3.75/8.5,3.75/4]);
% imshow(LYIQ); title('LYIQ Image');
% subplot('Position',[4/8.5,0,3.75/8.5,3.75/4]);
% imshow(UYIQ); title('UYIQ Image');
```

Visually compare the above images and comment on the effectiveness of each technique.

**Comparison and Comments**:

---

### (c) [2-Points] Comparison of Color Spaces

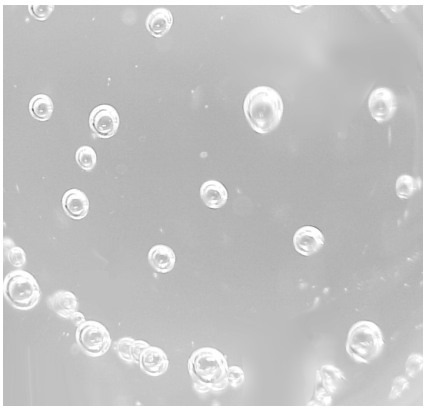
Now visually compare processed images in each color space and explain which color space is more suitable for edge-crispening the **`Peppers-color`** image.

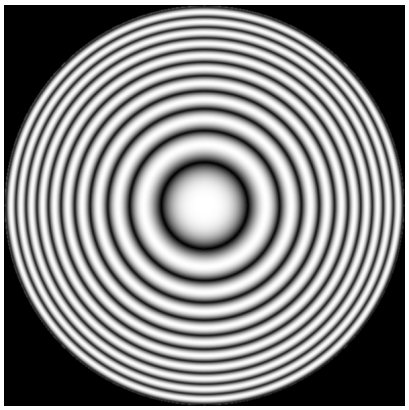
**Answer**:

---

## Problem-3 [10-Points] Image Coding

This problem uses the following three images, **`bubbles.tif`**, vase**`.tif`**, and **`zoneplate.tif`**.





### (a) [3-Points] Coding Redundancy

Using the first-order entropy of each image, determine which of the above three images contains the most coding redundancy? The entropy of an image is determined using its normalized histogram. Visually explain your answer.

**MATLAB script**: The entropy is computed using the Book toolbox function **`ntrop`**.

```
clc; close all; clear;
% Enter your code below
f1 = imread('bubbles.tif'); f2 = imread('vase.tif'); f3 = imread('zoneplate.tif');
e1 = ntrop(f1); e2 = ntrop(f2); e3 = ntrop(f3);
redundant_image = 'zoneplate.tif'
```

```
redundant_image =
'zoneplate.tif'
```

```
if (e1 < e3) && (e1 < e2)
    redundant_image = 'bubble.tif';
    entropy = e1;
elseif (e2 < e1) && (e2 < e3)
    redundant_image = 'vase.tif'
    entropy = e2;
end
disp([redundant_image, ' has more coding redundancy.']);
```

```
bubble.tif has more coding redundancy.
```

**Visual Explanation**:

The Bubbles image contains mostly grey intensity values that depicts the underwater image. These values contain the most redundancy since their insn't much variation in the gray areas of the image.

---

## (b) [2-Points] Compression Ratios

For the image from (a) above that has the most coding redundancy, what is the maximum achievable compression ratio?

**Solution**:

```
Max_compression = 8/entropy
```

```
Max_compression = 1.3613
```

Max compression that can be performed on the bubbles image was about 1.36

---

## (c) [3-Points] Huffman Encoding

Use the image from (a) above that has the most coding redundancy and Huffman encode it. What is the actual compression ratio?

**MATLAB script**:

```
% Enter your code below
```

```
%fjpg = im2jpeg4e(f1,2,8);
%fjpg.huffman
huffman = mat2huff(f1);
CR = imratio(f1,huffman)
```

```
CR = 1.3548
```

Here the compression ratio after Huffman encoding the image is about 1.35, so almost exactly what was calculated in the previous section

---

### (d) [2-Points] Huffman Decoding

Decode the Huffman encoded result and verify that the Huffman encoding/decoding process is lossless by computing the root-mean-squared-error (rmse) between encoded and decoded images.

**Note**: Do not display encoded and decoded images.

**MATLAB script**:

```
% Enter your code below
huffdecode = huff2mat(huffman);
rmse = compare4e(f1,huffdecode)
```

```
rmse = 0
```

Clearly, the Huffman encoding/decoding process is lossless.

---

## Problem-4 [10-Points] Morphological Edge Detection

This problem uses the grayscale image **`monastery.png`**.

## (a) [1-Point] Image Distortion

Read this image and call it **`f`**. Distort it by an additive salt-and-pepper noise affecting $2\%$ of pixels. Call it image **`g`**.

```
clc; close all; clear
```

```
% Enter your code below
f = imread('monastery.png');
g = imnoise(f,'salt & pepper',0.02);
% Display Original and Distorted images (uncomment the following line for display)
imshowpair(f,g,'montage');
```

## (b) [4-Points] Morphological Filtering and Edge Detection

Process **g** for noise removal and edge detection using morphological techniques. For noise removal use a radius $3$ disk structuring element and for edge detection use a radius$5$ disk structuring element. Call these filtered images **`fhat1`** and **`fedge1`**, respectively.

```
% Enter your code below
B1 = strel('disk',3);
B2 = strel('disk',5);

fhat1 = imclose(imopen(g,B1),B1);
fedge1 = imdilate(fhat1,B2)-imerode(fhat1,B2);
% Display two images (uncomment the following line for display)
imshowpair(fhat1,imcomplement(fedge1),'montage');
```

## (c) [3-Points] Median Filtering and Sobel Edge Detection

Now process **g** using a size $3 \times 3$ median filter for noise removal to obtain **fhat2**. Process **fhat2** using Sobel masks for horizontal and vertical edge detections. The edge detected image **fedge2** should be the sum of the absolute horizontal edge image and absolute vertical edge image.

```
% Enter your code below
fhat2 = medfilt2(g);
gv = edge(fhat2,'sobel','nothinning','vertical');
gh = edge(fhat2,'sobel','nothinning','horizontal');
fedge2 = gv + gh;
% Display two images (uncomment the following line for display)
imshowpair(fhat2,imcomplement(fedge2),'montage');
```

## (d) [2-Points] Comparison and Discussion

Compare and comment on your results in (b) and (c).

**Comparison**:

Comparing the results, the sobel filter performed a much cleaner edge detection along with the median filter removing more of the S&P noise. The performance of the dilation-erosion cause the image to be slightly blurred on the edges. The sobel filter was able to detect the vertical and horzontal lines much cleaner and more distinct.