# EECE-5626 (IP&PR) : Homework-4

**Due on November 2, 2021 by 11:59 pm via submission portal**

**NAME**: Enter your Lastname, Firstname here

## Instructions

1. You are required to complete this assignment using Live Editor.
2. Enter your MATLAB script in the spaces provided. If it contains a plot, the plot will be displayed after the script.
3. All your plots must be properly labeled and should have appropriate titles to get full credit.
4. Use the equation editor to typeset mathematical material such as variables, equations, etc.
5. After completeing this assignment, export this Live script to PDF and submit the PDF file through the provided submission portal.
6. You will have two attempts to submit your assignment. However, make every effort to submit the correct and completed PDF file the first time. If you use the second attempt, then that submission will be graded.
7. Your submission of problem solutions must be in the given order, i.e., P1, P2, P3, etc. Do not submit in a random order.
8. Please submit your homework before the due date/time. A late submission after midnight of the due date will result in loss of points at a rate of 10% per hour until 8 am the following day, at which time the solutions will be published.

**Reading Assignment**: Chapters 4 from DIP4E and DSPUM3E.

## Table of Contents

Problem-1: DIP4E Problem 5.11 (Page 442)........2
(b) Negative........2
(d) is Negative 1........2
(e) Behavior in Constant Intensity........2
Problem-2: DIP4E Problem 5.32 (Page 444)........2
Problem-3: DIP4E MATLAB Project 5.3 (b) and (e) (Page 446)........5
(b) MATLAB function saltPepper4e........5
(e) Processing of testpattern512.tif Image with Salt Noise only........6
Problem-4: DIP4E MATLAB Project 5.9 (Page 449)........9
(a) MATLAB Function constrainedLsTF4e........9
(b) Restoration of boy-blurred.tif Image........10
(c) Explanation of results in (b)........11
Problem-5: DIPUM3E MATLAB Project 5.7, parts (c) and (d) (Page 318)........12
(c) Restoration using Parametric Wiener Filter........12
(d) Restoration using Noise-to-Signal Power Ratio........13
Problem-6: DIPUM3E Project 6.4 (Page 373)........14
(a) Create a Unit Square........14
(b) Rotate the Unit Square using Affine Shears........15
(c) Rotate apple-hill-2013.png using Affine Shears........16
Problem-7: DIP4E 7.3 (Page 588) and DIP4E 7.25 (a) & (c) (Page 591)........17
(a) DIP4E 7.3........17

(b) DIP4E 7.25 ..... 19
(a) HSI Components ..... 20
(c) Smoothing of Hue component ..... 21
Problem-8 DIP4E MATLAB Project 7.2 (Page 592) ..... 21
(a) MATLAB Function rgb2gray4e ..... 22
(c) Processing of Image sunflower.tif ..... 22
Problem-9 DIP4E MATLAB Project 7.6 (Page 592) ..... 24
(a) Equalization of strawberries-RGB-dark.tif Image ..... 24
(b) Explanation of Results in (a) ..... 25
Problem-10 DIPUM3E Project 7.7 (Page 455) ..... 26
(a) Smoothing of sunflower.tif Image ..... 26
(b) Sharpening of sunflower.tif Image ..... 26

## Problem-1: DIP4E Problem 5.11 (Page 442)

In answering the follwoing, refer to the contraharmonic filter in Eq (5-36):

### (b) Negative $Q$

Explain why the filter is effective in eliminatine salt noise when $Q$ is positive.

**Solution**: The reverse of (a) happens when the center point is large and its neighbors are small. The center pixel will now be the largest. However, the exponent is now negative, so the small numbers will dominate the result. The numerator can then be thought of as a constant raised to the power $Q + 1$, and the denominator as the same constant raised to the power $Q$. That constant is the value of the pixels in the neighborhood. So the ratio is just that value.

### (d) $Q$ is Negative 1.

Discuss the expected behavior of the filter when $Q = -1$.

**Solution**: When $Q = -1$, the value of the numerator at any location is equal to the number of pixels in the neighborhood $(m \times n)$. The terms of the sum in the denominator are 1 divided by individual pixel values in the neighborhood. For example, for a $3 \times 3$ neighborhood, the response of the filter when $Q = -1$ is: $9/(1/p_1 + 1/p_2 + \cdots + 1/p_9)$ where the $p$'s are the pixel values in the neighborhood. Thus, low pixel values will tend to produce low filter responses, and vice versa. If, for example, the filter is centered on a large spike surrounded by zeros, the response will be a low output, thus reducing the effect of the spike.

### (e) Behavior in Constant Intensity

Discuss the behavior of the filter (for positive and negative $Q$) in areas of constant intensity.

**Solution**: In a constant area, the filter returns the value of pixels in the area, independently of the value of $Q$.

## Problem-2: DIP4E Problem 5.32 (Page 444)

**Inverse filtering on additive Gaussian noise.**

**Additional Part**: Provide a MATLAB simulation of your explanation to this problem using **`book-cover.tif`** image. Towards this follow the following steps:

1. Read image book-cover.tif and blur it using the parameters of Example 5.8 and the blurring frequency response of Eq. (5.77). Display the blurred image.
2. Inverse filter the blurred image in the frequency domain and display the result. You may need to scale the image for proper display.
3. Generate Gaussian noise with $0$ mean and variance equal to $0.0001$ and inverse filter it. Display using proper scaling.
4. Add the two images in 2 and 3 and display the result via proper scaling.

**Solution**: The image in Fig. 5.29(g) is the sum of a blurred image plus noise. The image is blurred but the noise is not blurred, because it was added after blurring, according to the model in Fig. 5.1. Because we are dealing with a linear system, deblurring the composite image is the same as deblurring the blurred image and the noise image separately, and then adding the two results as given below

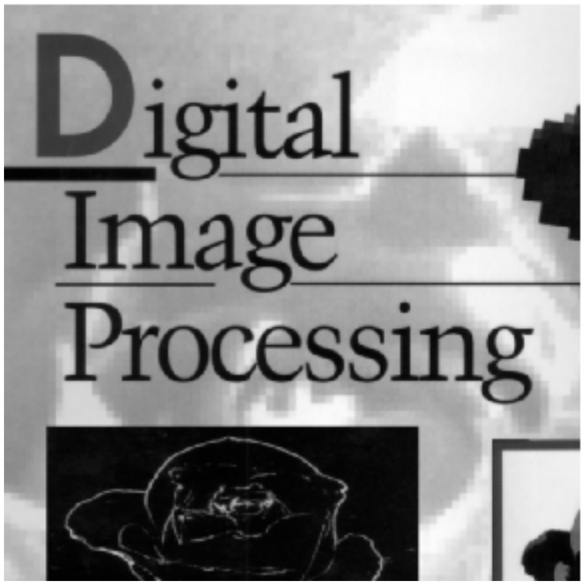
$$G(u,v) = H(u,v)F(u,v) + N(u,v) \qquad \Rightarrow \quad \frac{G(u,v)}{H(u,v)} = F(u,v) + \frac{N(u,v)}{H(u,v)}$$

Thus, the noise is deblurred in the frequency domain. Image of the noise is just gray dots on a dark background. When a deblurring filter is applied to an unblurred image, the result is an image that is blurred “in reverse.” This set of blurred dots is the streak pattern seen in Fig 5.29(h). The Wiener filter does not have this problem because it is set-up to deal with noise, while the inverse filter is not.

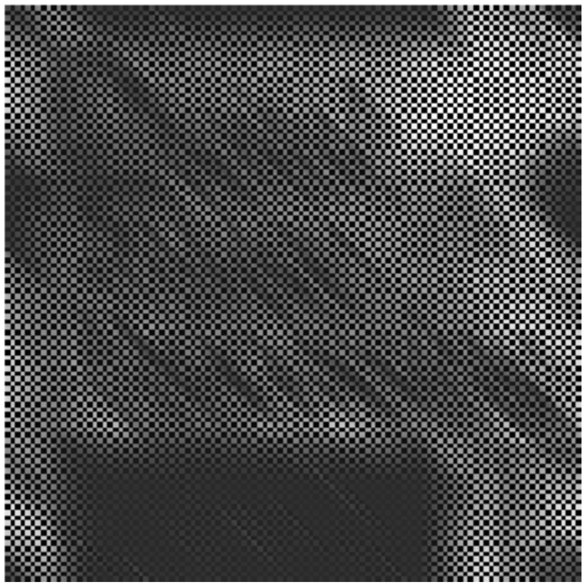
The following MATLAB simulation shows the above explanation graphically. The display of processed images appears to be distorted due to De Moivre effect at the shown image sizes in the editor. The exported images do not show this distortion.

```
clc; close all; clear;
f = imread('../artfiles/book-cover.tif'); [M,N] = size(f);
% Motion blurring of image
F = fftshift(fft2(f)); % 2-D DFT of f
a = 0.1; b = 0.1; T = 1; % Motion blur parameters in Example 5.8
Hmb = motionBlurTF4e(M,N,a,b,T); % Eq. (5.77)
Fmb = F.*Hmb; % Transform of motion blurred image
fmb = real(ifft2(Fmb)); % Motion blur image
fmbsc = intScaling4e(fmb,'full');
K = 20; fmbsc = ((1+K).^fmbsc-1)/K;
% imwrite(fmbsc,'book-cover-blurred.png');
figure('Units',"inches","Position",[0,0,8.25,4.25]);
subplot('Position',[0,0,4/8.25,4/4.25]); imshow(f);
title('Book-Cover Image','FontSize',14);
subplot('Position',[4.25/8.25,0,4/8.25,4/4.25]); imshow(fmbsc);
title('Motion-Blurred Image','FontSize',14);
```

Book-Cover Image

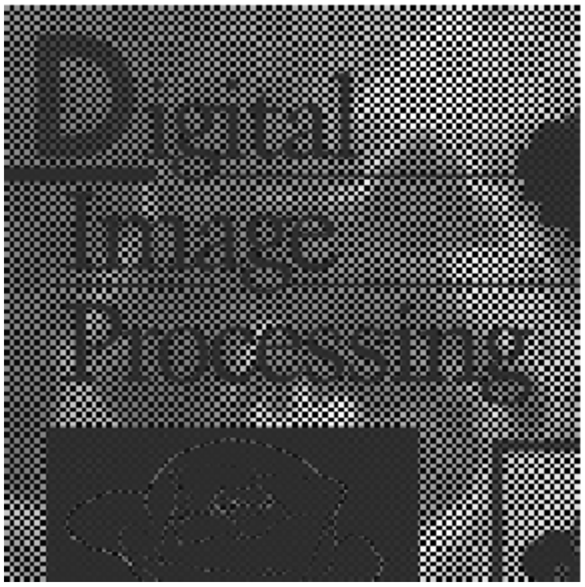


Motion-Blurred Image

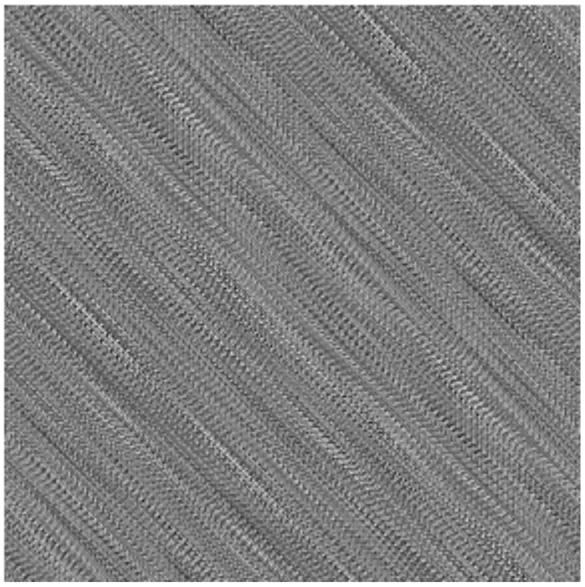


```
% Inverse Filtering of Motion-Blurred Image
Hmbinv = 1./Hmb; Fhat = Fmb.*Hmbinv;
fhat = real((ifft2(Fhat)));
fhat = intScaling4e(fhat,'full'); % Image normalization
fhatsc = ((1+K).^fhat-1)/K; % Exponential enhancement
fnoi = imnoise(zeros(size(fmb)),"gaussian",0,0.0001);
Fnoi = (fft2(fnoi));
fnoihat = real((ifft2(Fnoi.*Hmbinv)));
fnoihat = intScaling4e(fnoihat,'full');
% imwrite(fnoihat,'book-cover-Noise-invFilt.png');
figure('Units',"inches","Position",[0,0,8.25,4.25]);
subplot('Position',[0,0,4/8.25,4/4.25]); imshow(fhatsc);
title('Inverse Filtering of Motion-Blurred Image','FontSize',14);
subplot('Position',[4.25/8.25,0,4/8.25,4/4.25]); imshow(fnoihat);
title('Inverse Filtering of Gaussian Noise','FontSize',14);
```

Inverse Filtering of Motion-Blurred Image

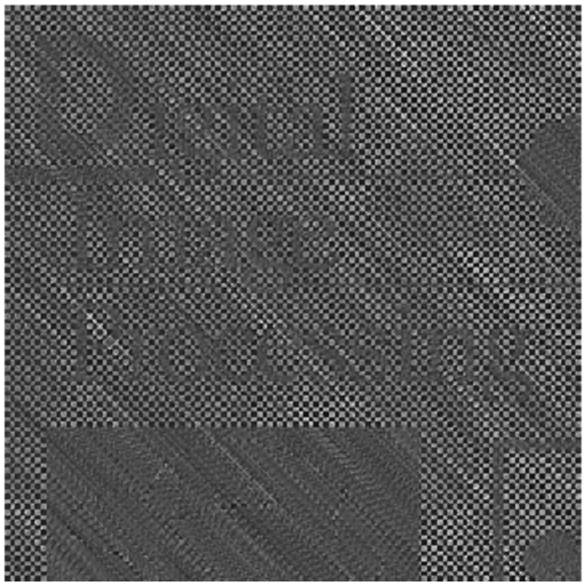


Inverse Filtering of Gaussian Noise



```
% Inverse Filtering of Noisy-Motion-Blurred Image
fmbnhat = fhat+fnoihat;
fmbnhatsc = intScaling4e(fmbnhat,'full');
K = 5; fmbnhatsc = ((1+K).^fmbnhatsc-1)/K;
% imwrite(fmbnhatsc,'book-cover-MBN-invFilt.png');
figure("Units","inches",'Position',[0,0,8,4.25]);
subplot("Position",[2/8,0,4/8,4/4.25]); imshow((fmbnhatsc),[]);
title('Inverse Filtering of Noisy-Motion-Blurred Image','FontSize',14);
```

Inverse Filtering of Noisy-Motion-Blurred Image



## Problem-3: DIP4E MATLAB Project 5.3 (b) and (e) (Page 446)

**Working with salt-and-pepper noise.**

### (b) MATLAB function `saltPepper4e`

**Solution**: Insert the code of your function below after the comments.

```
function g = saltPepper4e(f,ps,pp)
%SALTPEPPER4E corrupts an image with salt—and-pepper noise.
% g = SALTPEPPER4E(F,PP,Ps) corrupts image F with salt—and—pepper
% noise. Pepper values are 0 with probability PP and salt va1ues
% are 1 with probability PS. Image F can be floating point or can
% have integers values. The output, corrupted image G, is floating
% point with va'lues in the range [0,1]. This function is based on
% implementing Eq. (5-16) in DIP4E.
%

%-Pre1iminaries.
%--The sum of PS and PP shou1d not exceed 1.
if (ps + pp) > 1
    error('The sum pp + ps must not exceed 1')
end

%——Make sure that va1ues of f are in the range [0,1].
f = intScaling4e(f);

%——Image size.
```

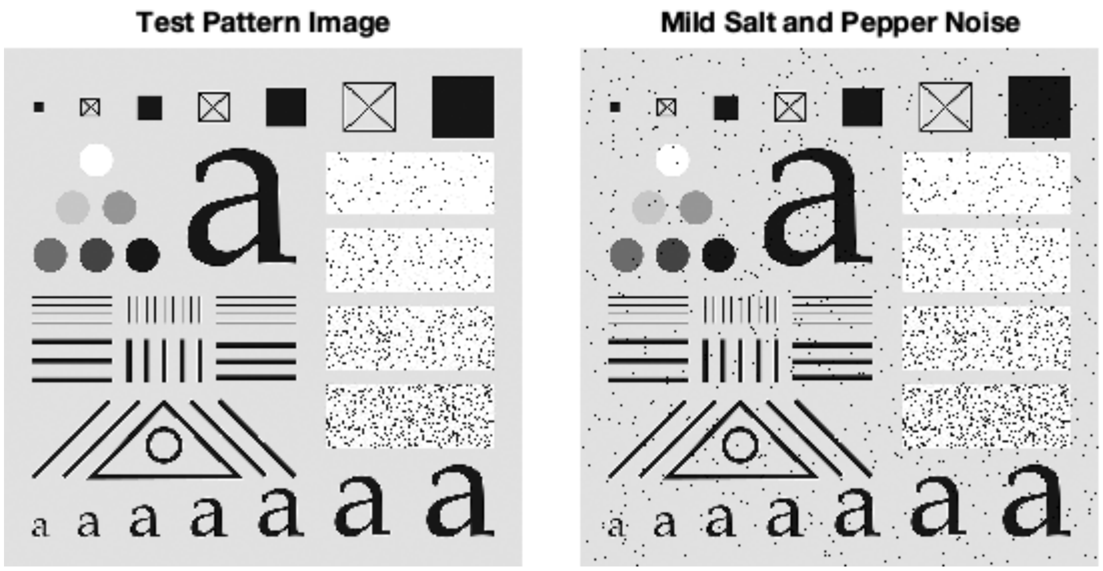
```
[M,N] = size(f);

%-Based on Eq. (5-16), the approach is to generate an M-by-N noise
% matrix R of uniform1y-distributed random numbers in the range (0,1).
% Then, PP*(M*N) of them wi11 have values <= pp. The coordinates
% of these points we call 0 (pepper noise). Simi1ar1y, ps*(M*N)
% points will have va1ues in the range > pp & <= (pp + ps). These we
% call 1 (salt noise). Assign a fixed va1ue to V (other than 0 and
% 1) to all other locations in R.

% We use R to corrupt f with salt–and–pepper noise as follows: Let
% v = 0.5. Then: Assign a 0 to all locations in f where corresponding
% locations in R are 0; assign a 1 to a11 1ocations in f where
% corresponding locations in R are 1; and "leave unchanged all
% locations in f where corresponding locations in R have value 0.5.

%–Generate R, using uniform random number generator rand.
R(1:M, 1:N) = 0.5; % Initialize R.
X = rand(M,N);
R(X <= pp) = 0;
u = pp + ps;
R(X > pp & X <= u) = 1;

%–Use R to corrupt f.
g=f; g(R==0) = 0; g(R==1) = 1;

end
```

## (e) Processing of `testpattern512.tif` Image with Salt Noise only

**Solution**: Because the noise density is half of the density in (c), the values of ps will have to be higher for the intermediate, heavy, and extra-heavy noise fields. Mild noise is supposed to be barely visible, so the same value, **`ps = 0.01`**, used in (c) will do.

### Mild Noise

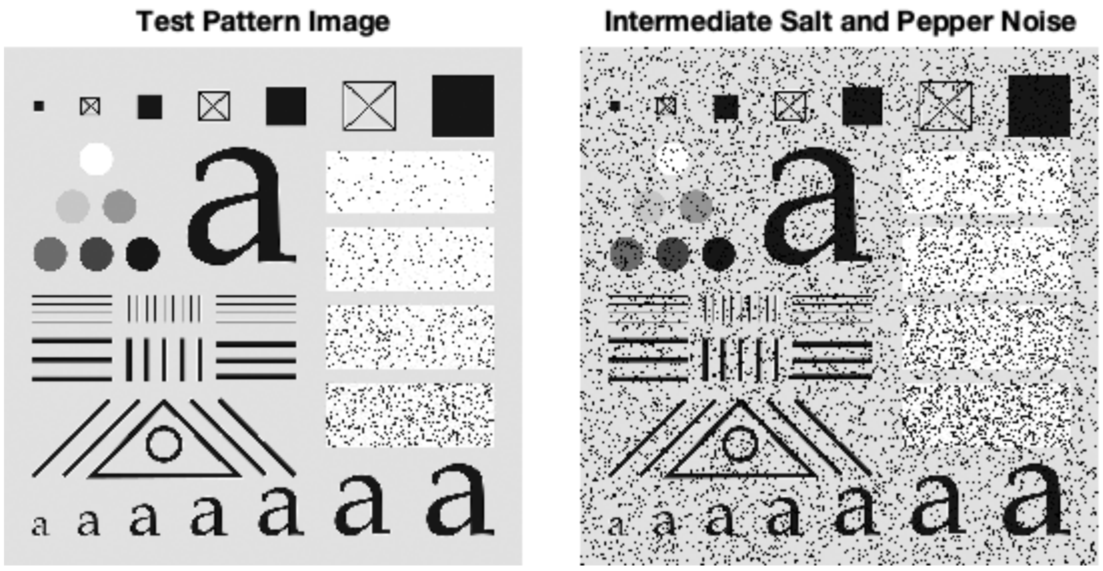
```
clc; close all; clear;
f = imread('../artfiles/testpattern1024.tif');
f = intScaling4e(f);
pp = 0; ps = 0.01;
gmild = saltPepper4e(f,pp,ps); gmild = intScaling4e(gmild);
figure("Units","inches","Position",[0,0,8,4]);
subplot('Position',[0,0,0.45,0.9]); imshow(f);
title('Test Pattern Image','FontSize',14);
subplot('Position',[0.5,0,0.45,0.9]); imshow(gmild);
title('Mild Salt and Pepper Noise','FontSize',14);
```



The noise is barely visible.

### Intermediate Noise

```
pp = 0; ps = 0.1;
ginter = saltPepper4e(f,pp,ps); ginter = intScaling4e(ginter);
figure("Units","inches","Position",[0,0,8,4]);
subplot('Position',[0,0,0.45,0.9]); imshow(f);
title('Test Pattern Image','FontSize',14);
subplot('Position',[0.5,0,0.45,0.9]); imshow(ginter);
title('Intermediate Salt and Pepper Noise','FontSize',14);
```

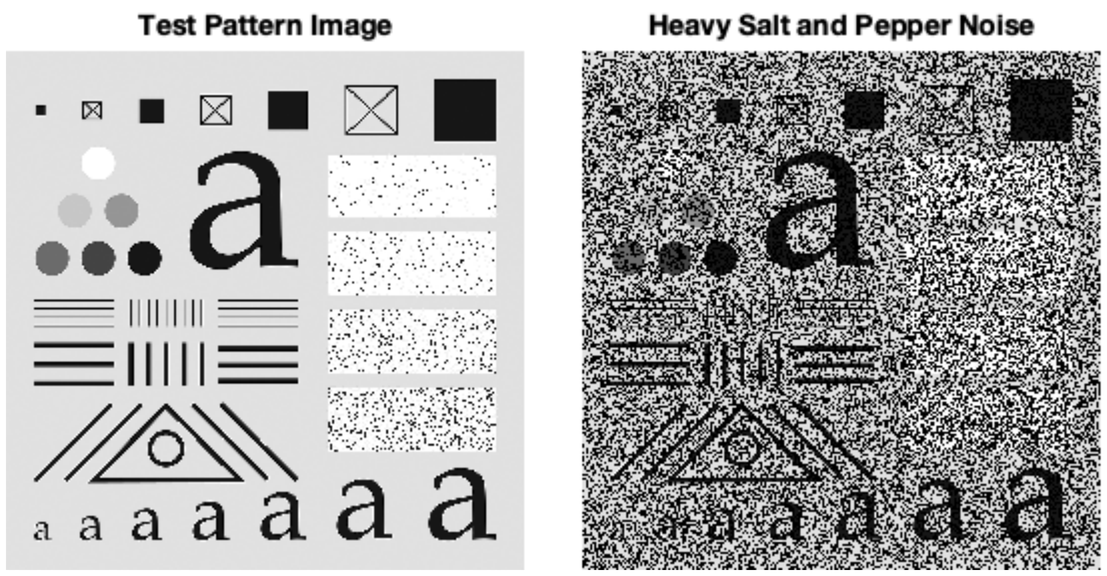


The noise is definitely visible so are all the features in the image.

### Heavy Noise

```
pp=0; ps = 0.4;
gheavy = saltPepper4e(f,pp,ps); gheavy = intScaling4e(gheavy);
```

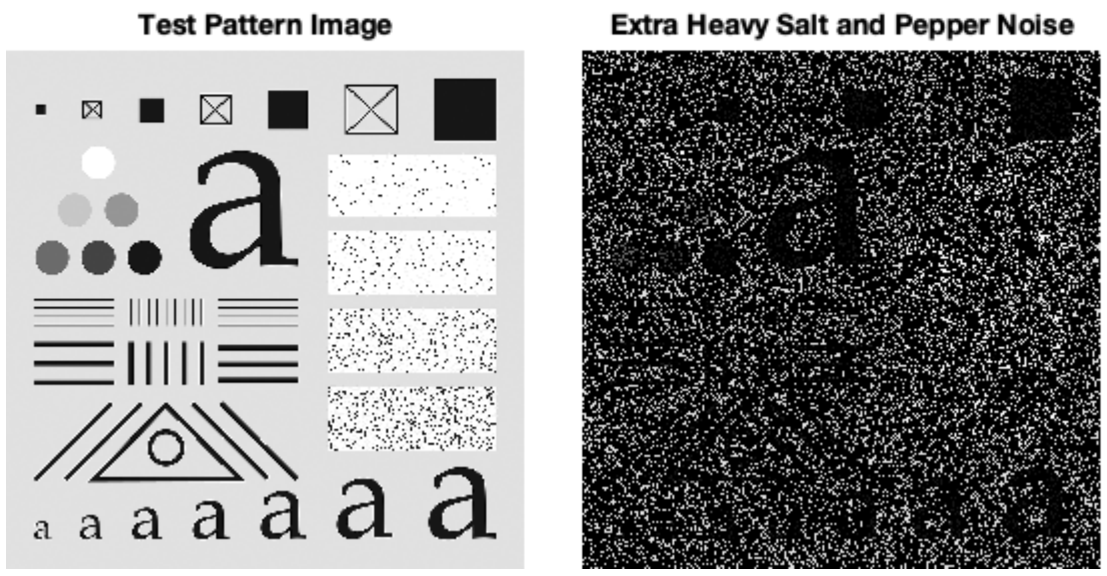
```
figure("Units","inches","Position",[0,0,8,4]);
subplot('Position',[0,0,0.45,0.9]); imshow(f);
title('Test Pattern Image','FontSize',14);
subplot('Position',[0.5,0,0.45,0.9]); imshow(gheavy);
title('Heavy Salt and Pepper Noise','FontSize',14);
```



Noise is objectionabie; some of the smaller and lighter image features are obscured by noise.

**Extra Heavy Noise**

```
pp=0; ps = 0.8;
geheavy = saltPepper4e(f,pp,ps); geheavy = intScaling4e(geheavy);
% K = 10; geheavy = log(1+(exp(K)-1)*geheavy)/K;
figure("Units","inches","Position",[0,0,8,4]);
subplot('Position',[0,0,0.45,0.9]); imshow(f);
title('Test Pattern Image','FontSize',14);
subplot('Position',[0.5,0,0.45,0.9]); imshow(geheavy);
title('Extra Heavy Salt and Pepper Noise','FontSize',14);
```



Noise is quite objectionable. Many of the image features areobscured by noise.

## Problem-4: DIP4E MATLAB Project 5.9 (Page 449)

**Constrained Least Squares (CLS) Filter.**

### (a) MATLAB Function `constrainedLsTF4e`

**Note***:* In the following, parameter **gam** is the same as $\lambda$ (lambda) that we used in lecture notes.

**Solution**: Insert the code of your function below after the comments.

```
function L = constrainedLsTF4e(H,gam)
%CONSTRAINEDLSTF4e constrained least squares filter transfer function.
%  L = CONSTRAINEDLSTF4E(H,GAM) implements the constrained least
%  squares filter transfer function defined in Eq. (5-89) of DIP4E.
%  H is the transfer function of the degradation, and GAM is the
%  gamma parameter in Eq. (5-89). This function requires that H be
%  of even dimensions. If H was obtained from a degraded image
%  whose dimensions are not even, delete a row, column, or both, to
%  make the size of the image be even in both directions. Then recompute H.
% Copyright 2017, R. C. Gonzalez & R. E. Woods

% Size of degradation function.
[M,N] = size(H);
% Center of the array.
cx = floor(M/2) + 1;
cy = floor(N/2) + 1;
% Check for even dimensions. isodd is a utility function.
if isodd(M) || isodd(N)
error('H must be of even dimensions.')
end
% compute transform of Laplacian operator.
% The Laplacian operator is Lap= [0 -1 0; -1 4 -1; 0 -1 0]. To make
% it into a function with even symmetry, we add a leading row and
% column of 0's, then embed it into the center of an M-by-N array of
% zeros. Because the leading row and column of 0's in Lap will
% overlap the 0s in the large array, we do not actually have to add
% these zeros to Lap physically. However, they are implicitly there,
% so the operator will be 4-by-4. This is the reason why H has to be
% of even dimensions, to preserve the even symmetry of the operator.
% see Example 4.10 in DIP4E for more details on this concept.

% Embed the operator.
p = zeros(M,N);
p(cx,cy) = 4; p(cx-1,cy)= -1; p(cx+1,cy)= -1;
p(cx,cy-1) = -1; p(cx,cy+1) = -1;
% obtain the centered transform, and then its absolute value squared.
P = dft2D4e(minusOne4e(p));
Pabs2 = abs(P).^2;
% obtain the rest of the terms for the constrained least squares
% filter transfer function.
Hstar = conj(H); Habs2 = abs(H).^2;
% Form the filter transfer function.
L = Hstar./(Habs2 + gam*Pabs2);
```

```
        end
```

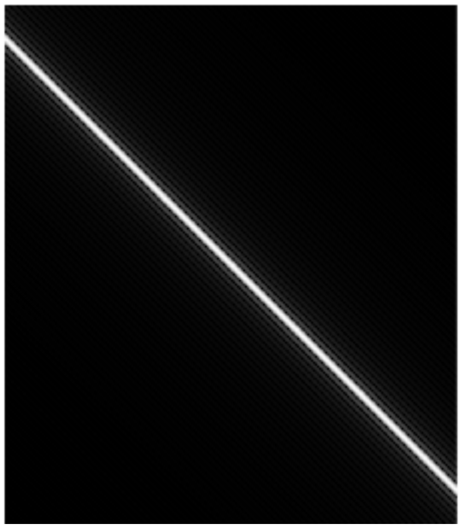
## (b) Restoration of `boy-blurred.tif` Image

Read the image **`boy-blurred.tif`** and restore it using the degradation function from Eq. (5-77).

**Solution**: MATLAB script.

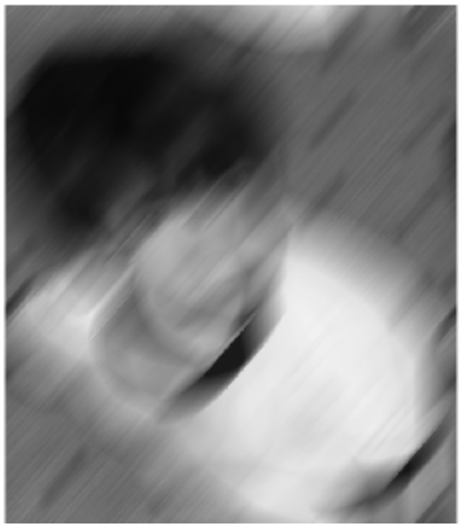
```
clc; close all; clear;
g = imread('../artfiles/boy-blurred.tif'); [M,N] = size(g);
g = intScaling4e(g); % will need it in floating point later.

% With reference to Fig. 5.26, we see by inspection that the
% parameters used to generate boy-blurred.tif are a=0.1, b=-0.1, and,
% as given in the problem statement, T = 1.
H = motionBlurTF4e(M,N,0.1,-0.1,1); % Motion-blur
% compute and display its spectrum.
SH = log(1+abs(H)); SH = intScaling4e(SH,'full');
figure("Units","inches","Position",[0,0,8,4]);
subplot('Position',[0.25,0,0.5,0.9]); imshow(SH);
title('Spectrum of the Motion-Blur TF','FontSize',14);
```

Spectrum of the Motion-Blur TF

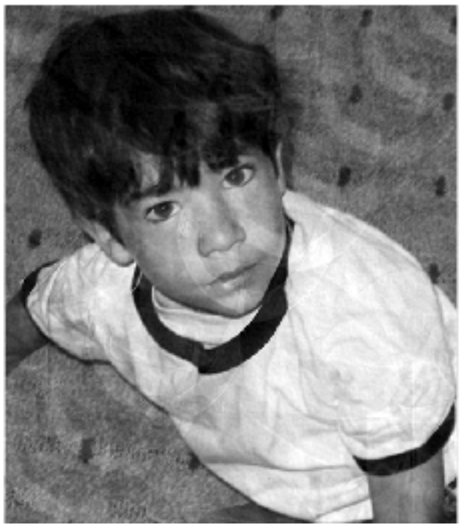


```
% Generate the filter transfer function using a value of gam midway
% in the range suggested in the project statement.
L = constrainedLsTF4e(H,10^(-3.5));

% Restore the blurred image using L
f_hat = dftFiltering4e(g,L,'none'); % Frequency domain filtering
figure("Units","inches","Position",[0,0,8,4]);
subplot('Position',[0,0,0.45,0.9]); imshow(g);
title('Boy Blurred Image','FontSize',14);
subplot('Position',[0.5,0,0.45,0.9]); imshow(f_hat);
title('Boy Respored Image','FontSize',14);
```

Boy Blurred Image



Boy Respored Image



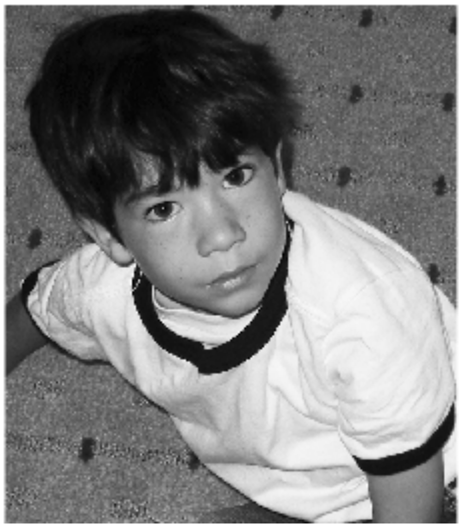
## (c) Explanation of results in (b)

Read the image **boy.tif** (this was the original before blurring) and explain why your restored image is not quite as good as the original.

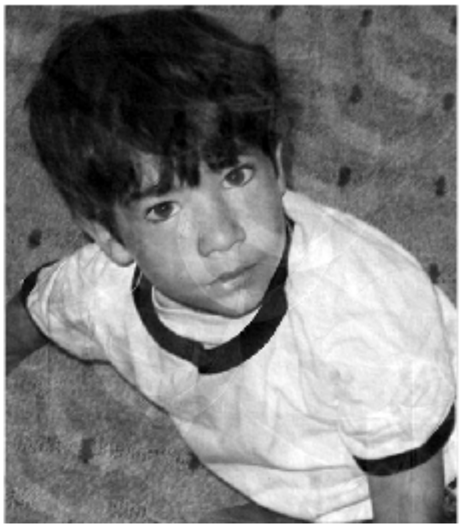
**Solution**: MATLAB script.

```
f = imread('../artfiles/boy.tif');
figure("Units","inches","Position",[0,0,8,4]);
subplot('Position',[0,0,0.45,0.9]); imshow(f);
title('Original Boy Image','FontSize',14);
subplot('Position',[0.5,0,0.45,0.9]); imshow(f_hat);
title('Boy Restored Image','FontSize',14);
```

Original Boy Image



Boy Restored Image



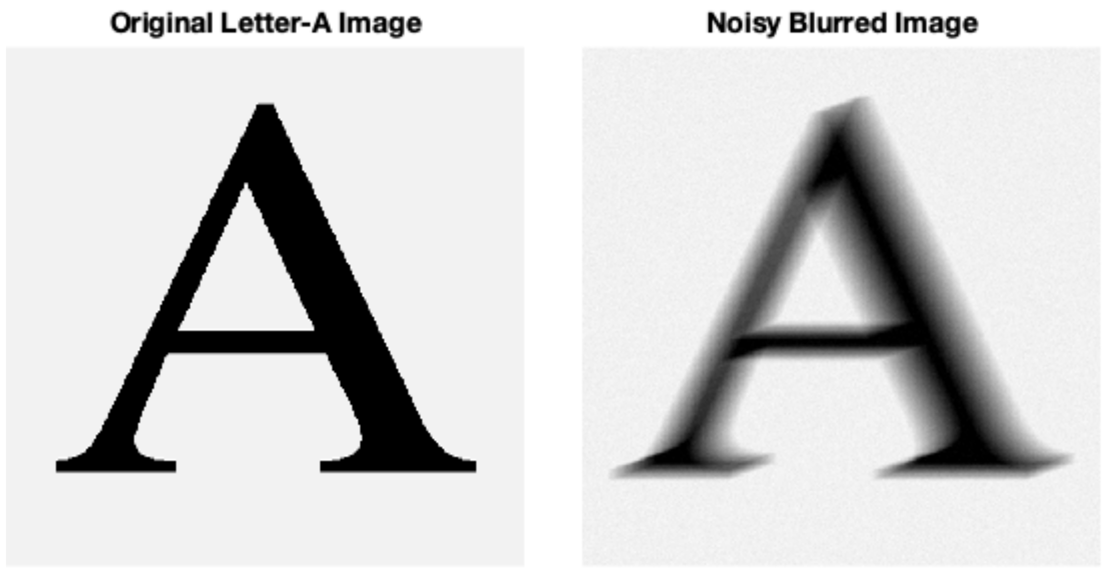
The reason the restored image is not quite as good as the original **boy.tif** image is that the exact inverse filter is unstable and the CLS inverse filter is an approximation. Furthermore, the blurred image in the frequency domain contained small complex-valued terms. These were deleted when the real part was taken and image was displayed. This is a numerical issue.

## Problem-5: DIPUM3E MATLAB Project 5.7, parts (c) and (d) (Page 318)

**Wiener Filtering**

Read the image **`letterA.tif`** and blur it using motion in the $+20$ degrees direction for a distance of $10\%$ of the image width. Then, add to the blurred image Gaussian noise of zero mean and variance of $0.0001$.

```
clc; close all; clear;
f = imread('../artfiles/letterA.tif'); [M,N] = size(f);
h = fspecial('motion',0.1*M,20); % Motion blur filter
gmb = imfilter(f,h,'conv','symmetric'); % Blurred image
gmbn = imnoise(gmb,'gaussian',0,0.0001); % Noisy blurred image
figure("Units","inches","Position",[0,0,8,4.5]);
subplot('Position',[0,0,0.45,0.9]); imshow(f);
title('Original Letter-A Image','FontSize',14);
subplot('Position',[0.5,0,0.45,0.9]); imshow(gmbn);
title('Noisy Blurred Image','FontSize',14);
```

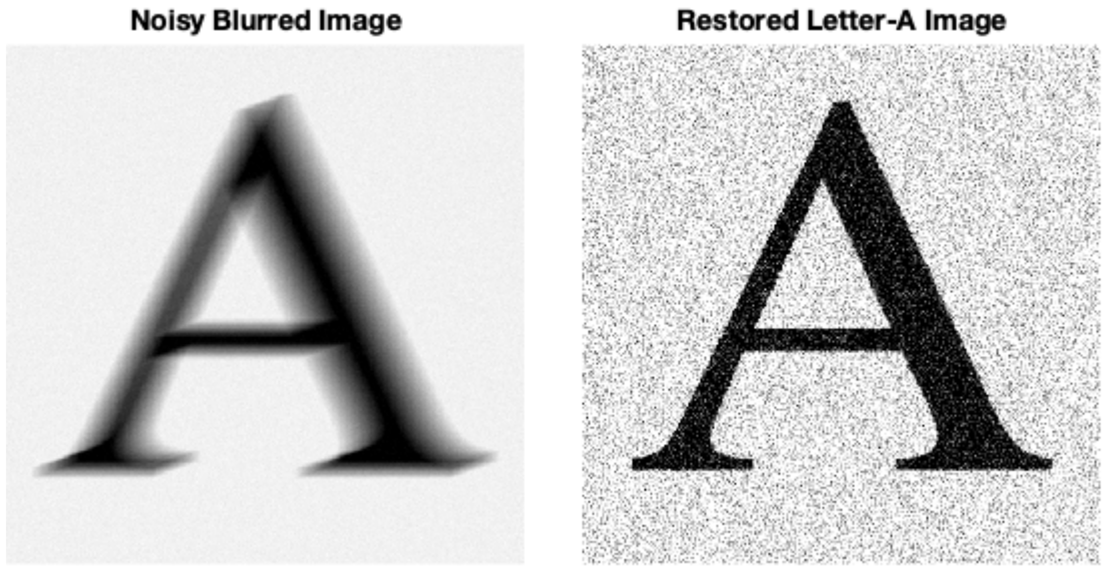


### (c) Restoration using Parametric Wiener Filter

**Solution**: The parametric Wiener filter basically assumes that NSPR is a  constant. The approach is to vary the constant and evaluate the visual results. The following value of NSPR achieved a good conpromise between achieving a restored letter-A and reducing the influence of noise and ghosting created by the motion component of the PSF.

```
NSPR = 0.0003; fhatc = deconvwnr(gmbn,h,NSPR);
figure("Units","inches","Position",[0,0,8,4.5]);
subplot('Position',[0,0,0.45,0.9]); imshow(gmbn);
```

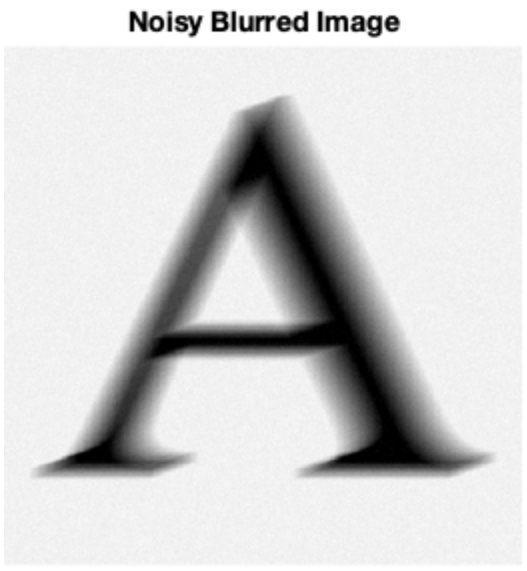
```
title('Noisy Blurred Image','FontSize',14);
subplot('Position',[0.5,0,0.45,0.9]); imshow(fhatc);
title('Restored Letter-A Image','FontSize',14);
```

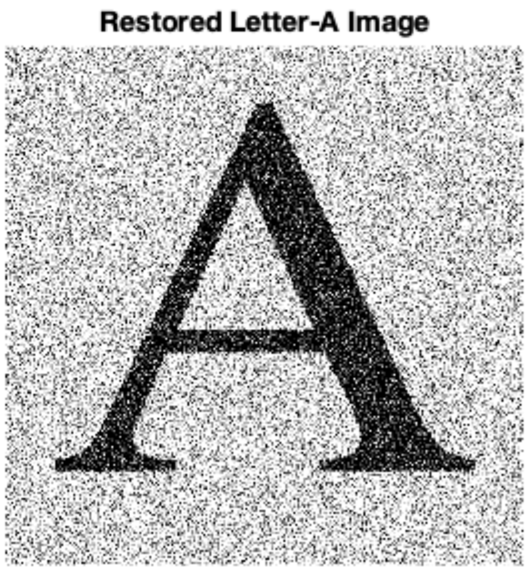


## (d) Restoration using Noise-to-Signal Power Ratio

**Solution**: For this approach, we need to obtain the noise only image as **`gmbn - gmb`**. We will need this image to compute Noise-to-Signal-Power-Ratio (NSPR).

```
% Compute an estimate of the NSPR.
gnoise = gmbn-gmb;
Sn = abs(fft2(gnoise)).^2; % Noise power spectrum.
nA = sum(Sn(:))/numel(gnoise); % Noise average power.
Sf = abs(fft2(f)).^2; % Image power spectrum.
fA = sum(Sf(:))/numel(f); % Image average power.
NSPR = nA/fA;
% Restore and display the resu1t.
fhatd = deconvwnr(gmbn,h,NSPR);
figure("Units","inches","Position",[0,0,8,4.5]);
subplot('Position',[0,0,0.45,0.9]); imshow(gmbn);
title('Noisy Blurred Image','FontSize',14);
subplot('Position',[0.5,0,0.45,0.9]); imshow(fhatd);
title('Restored Letter-A Image','FontSize',14);
```





As the above result shows, the parametric Wiener filter result based on a constant estimate of NSPR was slightly better than the computed NSPR. This is not all bad, particularly since we had to use the inacceble blurred image `gmb` to estimate `Sn` as well as the inacceble original image `f` to estimate `Sf`. If we had `f`, then restoration of the blurred, noisy version of `f` would be pointless.

## Problem-6: DIPUM3E Project 6.4 (Page 373)

**Rotation using Composite Affine Transforms.**

This project explores the compositions of affine transforms and implementing rotation as three successive shearing operations: ahorizontal shear, followed by a vertical shear, followed by another horizontal shear. This can sometimes be of practical interest because each sheering operation can be implemented by shifting and interpolating image pixels in only one direction. For a rotation angle $\theta$, ththe three affine shear matrices are

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad T_2 = \begin{bmatrix} 1 & \beta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad T_3 = \begin{bmatrix} 1 & 0 & 0 \\ \gamma & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where $\alpha = \gamma = -\tan(\theta/2)$ and $\beta = \sin(\theta)$.

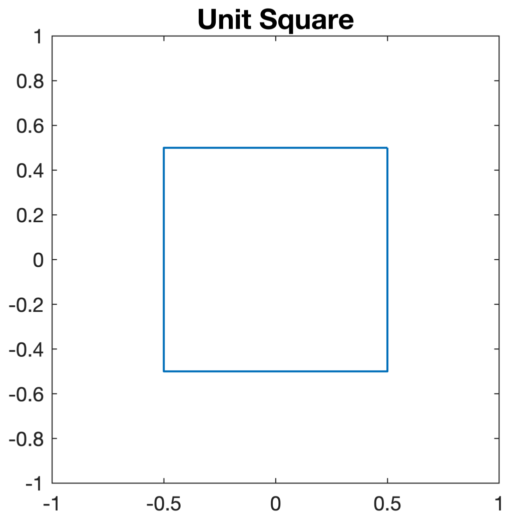
### (a) Create a Unit Square

Make a $4 \times 2$ matrix containing the four vertices $(0.5, 0.5)$, $(-0.5, 0.5)$, $(-0.5, -0.5)$, and $(0.5, -0.5)$ of a unit square centered at the origin. Plot the square containing these vertices.

**Solution**:

```
clc; close all; clear;
V = [0.5,0.5;-0.5,0.5;-0.5,-0.5;0.5,-0.5];
```

```
% before plotting, we duplicate the first vertex at the end so that the
% plotted vertices make a closed square.
V(end+1,:) = V(1,:);
figure("Units","inches","Position",[0,0,4,4.5]);
plot(V(:,1),V(:,2),'LineWidth',1);
axis equal; axis([-1,1,-1,1]);
title('Unit Square','FontSize',14);
```

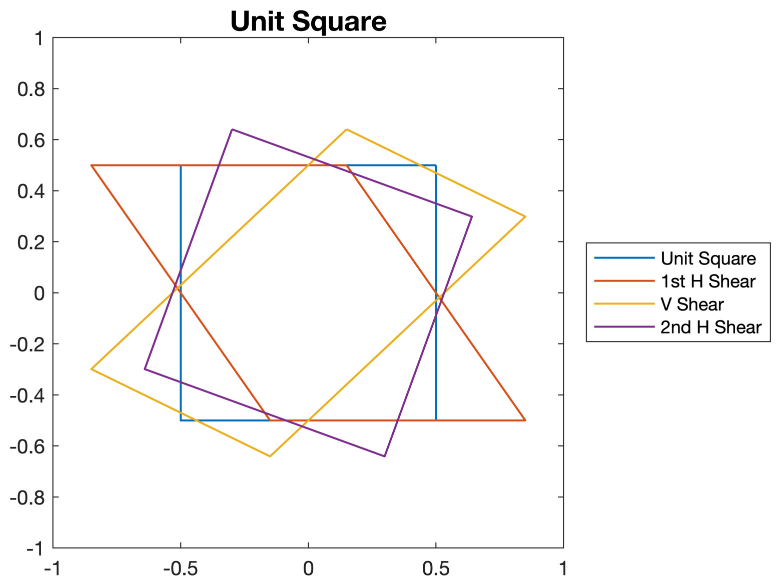


## (b) Rotate the Unit Square using Affine Shears

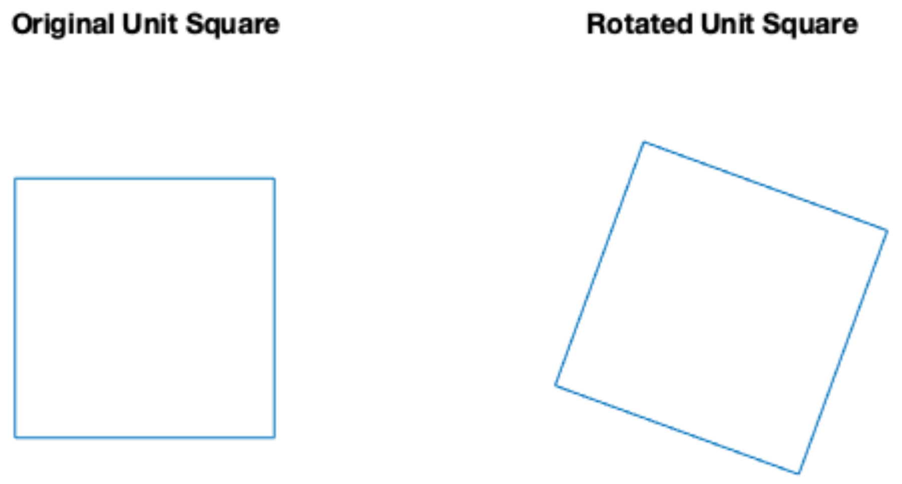
Make three **affine2d** objects **tform1**, **tform2**, and **tform3**, using the three affine transform matrices shown above with $\theta = 70^\circ$. In three steps, apply these affine transforms successively to the vertices of the square. At each step, superimpose the transformed shape on the plot from part (a).

**Solution**:

```
theta = 70;
T1 = [1,0,0; -tand(theta/2),1,0; 0,0,1];
T2 = [1,sind(theta),0; 0,1,0; 0,0,1];
T3 = [1,0,0; -tand(theta/2),1,0; 0,0,1];
tform1 = affine2d(T1); tform2 = affine2d(T2); tform3 = affine2d(T3);
V1 = transformPointsForward(tform1,V);
figure("Units","inches","Position",[0,0,6,4.5]);
plot(V(:,1),V(:,2),'LineWidth',1); % Original unit square
axis equal; axis([-1,1,-1,1]);
title('Unit Square','FontSize',14); hold on;
plot(V1(:,1),V1(:,2),'LineWidth',1); % First H shear
V2 = transformPointsForward(tform2,V1);
plot(V2(:,1),V2(:,2),'LineWidth',1); % Vertical shear
V3 = transformPointsForward(tform3,V2);
plot(V3(:,1),V3(:,2),'LineWidth',1); % Second H shear
legend('Unit Square','1st H Shear','V Shear','2nd H Shear',...
    'Location',"eastoutside");
```



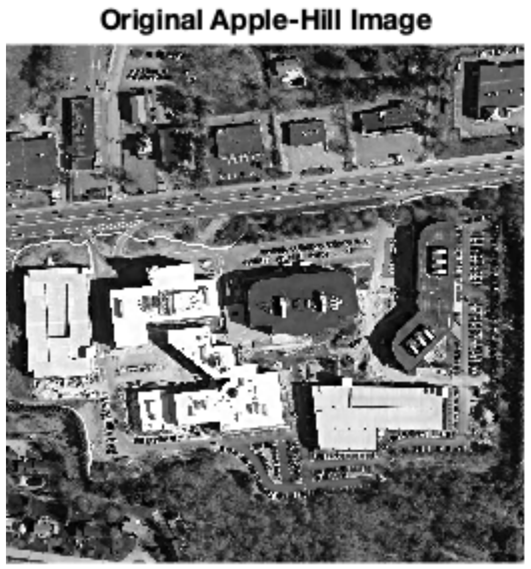
```
figure("Units","inches","Position",[0,0,8,4.5]);
subplot('Position',[0,0,0.45,0.9]);
plot(V(:,1),V(:,2),'LineWidth',1); % Original unit square
axis equal; axis([-1,1,-1,1]);
title('Original Unit Square','FontSize',14); axis off;
subplot('Position',[0.5,0,0.45,0.9]);
plot(V3(:,1),V3(:,2),'LineWidth',1); % Rotated unit square
axis equal; axis([-1,1,-1,1]);
title('Rotated Unit Square','FontSize',14); axis off;
```

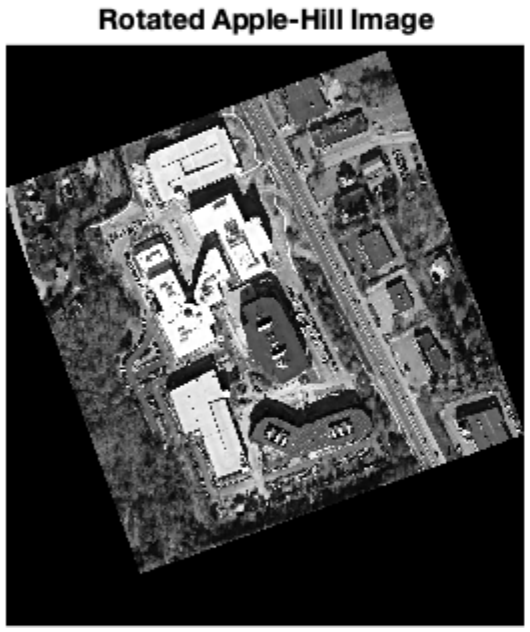


## (c) Rotate `apple-hill-2013.png` using Affine Shears

**Solution**:

```
f = imread('../artfiles/apple-hill-2013.png');
g1 = imwarp(f,tform1); g2 = imwarp(g1,tform2);
g3 = imwarp(g2,tform3);
figure("Units","inches","Position",[0,0,8,4.5]);
subplot('Position',[0,0,0.45,0.9]);
imshow(f); % Original image
title('Original Apple-Hill Image','FontSize',14);
subplot('Position',[0.5,0,0.45,0.9]);
imshow(g3(970:3100,1700:3600)); % Rotated Apple-Hill image
title('Rotated Apple-Hill Image','FontSize',14);
```





## Problem-7: DIP4E 7.3 (Page 588) and DIP4E 7.25 (a) & (c) (Page 591)

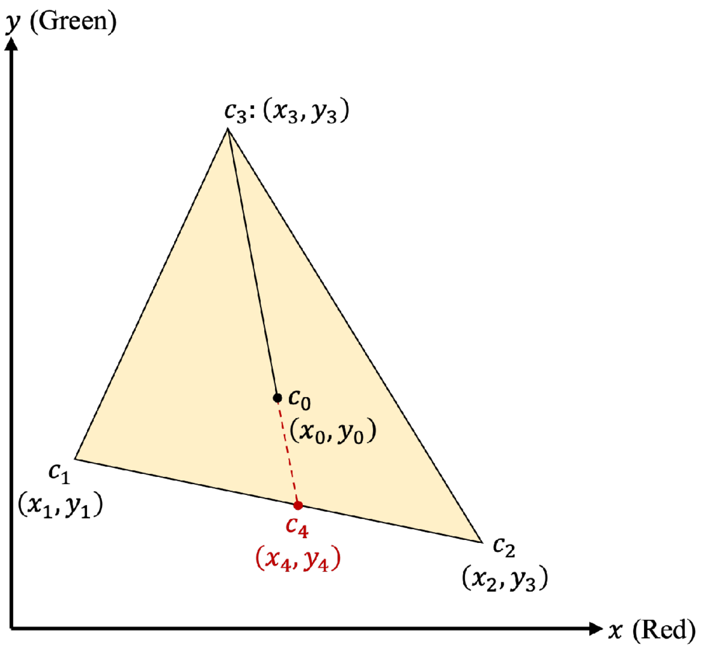
**The following two parts, (a) and (b), are not related.**

### (a) DIP4E 7.3

Study Problem 7.2 to solve this problem.

Consider any three valid colors $c_1$, $c_2$, and $c_3$ with coordinates $(x_1, y_1)$, $(x_2, y_2)$, and $(x_3, y_3)$ in the chromaticity diagram of Fig.7.5. Derive the necessary general expression(s) for computing the relative percentages of $c_1$, $c_2$, and $c_3$ composing a color that is known to lie within the triangle whose vertices are at the coordinates of $c_1$, $c_2$, and $c_3$.

**Solution**: Consider the following figure, in which $c_1$, $c_2$, and $c_3$ are the given vertices of the color triangle and let $c_0$ be an arbitrary color point contained within the triangle or on its boundary. The key to solving this problem is to realize that any color on the border of the triangle is made up of proportions from the two vertices defining the line segment that contains the point. The contribution to a point on the line by the color vertex opposite this line is $0\%$.



The line segment connecting points $c_3$ and $c_0$ is shown extended (dashed segment) until it intersects the line segment connecting $c_1$ and $c_2$. The point of intersection is denoted by $c_4$. Because we have the values of $c_1$ and $c_2$, if we knew $c_4$, we could compute the percentages of $c_1$ and $c_2$ contained in $c_4$ by using the method described in Problem 7.2. Let the ratio of the content of $c_1$ and $c_2$ in $c_4$ be denoted by $R_{12}$. If we now add color $c_3$ to $c_4$, we know from Problem 7.2 that the point will start to move toward $c_3$ along the line shown. For any position of a point along this line we could determine the percentage of $c_3$ and $c_4$ using the method described in Problem 7.2. What is important to keep in mind that the ratio $R_{12}$ will remain the same for any point along the segment connecting $c_3$ and $c_4$. The color of the points along this line is different for each position, but the ratio of $c_1$ to $c_2$ will remain constant.

So, if we can obtain $c_4$, we can then determine the ratio $R_{12}$ and the percentage of $c_3$ in color $c_0$. The point $c_4$ is not difficult to obtain. It is obtained by intersecting line from $c_1$ to $c_2$ with the extended line from $c_3$ to $c_0$. Let the coordinates of $c_0$ be $(x_0, y_0)$ which are known and let the coordinates of $c_4$ be $(x_4, y_4)$ which are unknown. Equation of line joining $c_1$ and $c_2$ is

$$\frac{y - y_1}{x - x_1} = \frac{y_1 - y_2}{x_1 - x_2} \quad \Rightarrow \quad y = \underbrace{\left(\frac{y_1 - y_2}{x_1 - x_2}\right)}_{a_{12}} x + \underbrace{y_1 - \left(\frac{y_1 - y_2}{x_1 - x_2}\right) x_1}_{b_{12}} \triangleq a_{12}x + b_{12}.$$

Similarly, equation of line joining $c_3$ and $c_0$ is

$$\frac{y-y_3}{x-x_3} = \frac{y_3-y_0}{x_3-x_0} \quad \Rightarrow \quad y = \underbrace{\left(\frac{y_3-y_0}{x_3-x_0}\right)}_{a_{30}} x + \underbrace{y_3 - \left(\frac{y_3-y_0}{x_3-x_0}\right) x_3}_{b_{30}} \triangleq a_{30}x + b_{30}.$$

Let the intersection of these two lines be at $c_4 : (x_4, y_4)$, that is,

$$y_4 = a_{12}x_4 + b_{12} \text{ and } y_4 = a_{30}x_4 + b_{30}.$$

Solving for $x_4$ and $y_4$, we obtain

$$x_4 = \left(\frac{b_{30}-b_{12}}{a_{12}-a_{30}}\right) \triangleq \alpha, \quad a_{12} - z_{30} \neq 0 \text{ and } y_4 = a_{12}\alpha + b_{12} \triangleq \beta.$$

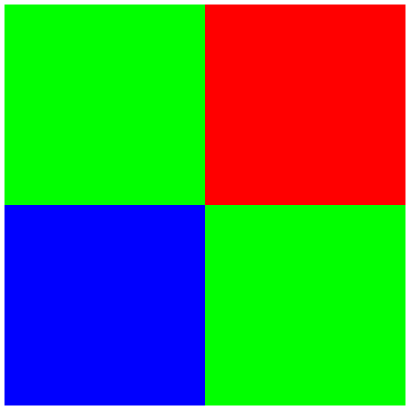
Thus, the point $c_4$ has coordinates $(\alpha, \beta)$ as given above.

At this juncture we have the percentage of $c_3$ and the ratio between $c_1$ to $c_2$. Let the percentages of these three colors composing $c_0$ be denoted by $p_1$, $p_2$, and $p_3$, respectively. We know that $p_1 + p_2 = 100 - p_3$, and that $p_1/p_2 = R_{12}$, so we can solve for $p_1$ and $p_2$.

Finally, note that this problem could have been solved the same way by intersecting one of the other two sides of the triangle. Going to another side would be necessary, for example, if the line we used in the preceding discussion had an infinite slope. A simple test to determine if the color of $c_0$ is equal to any of the vertices should be the first step in the procedure; in this case no additional calculations would be required.

### (b) DIP4E 7.25

Consider the following $500 \times 500$ RGB image, in which the squres are fully saturatedred, green, and blue, and each of the color is at maximum intensity. An HSI image is genrated from this image.



Answer the following questions.

```
clc; close all; clear;
fR = zeros(500); fG = fR; fB = fR;
fR(1:250,251:500) = 1;
fG(1:250,1:250) = 1; fG(251:500,251:500) = 1;
fB(251:500,1:250) = 1;
fRGB = fR; fRGB(:,:,2) = fG; fRGB(:,:,3) = fB;
```

```
    imwrite(fRGB,'P7-25.png');
```

### (a) HSI Components

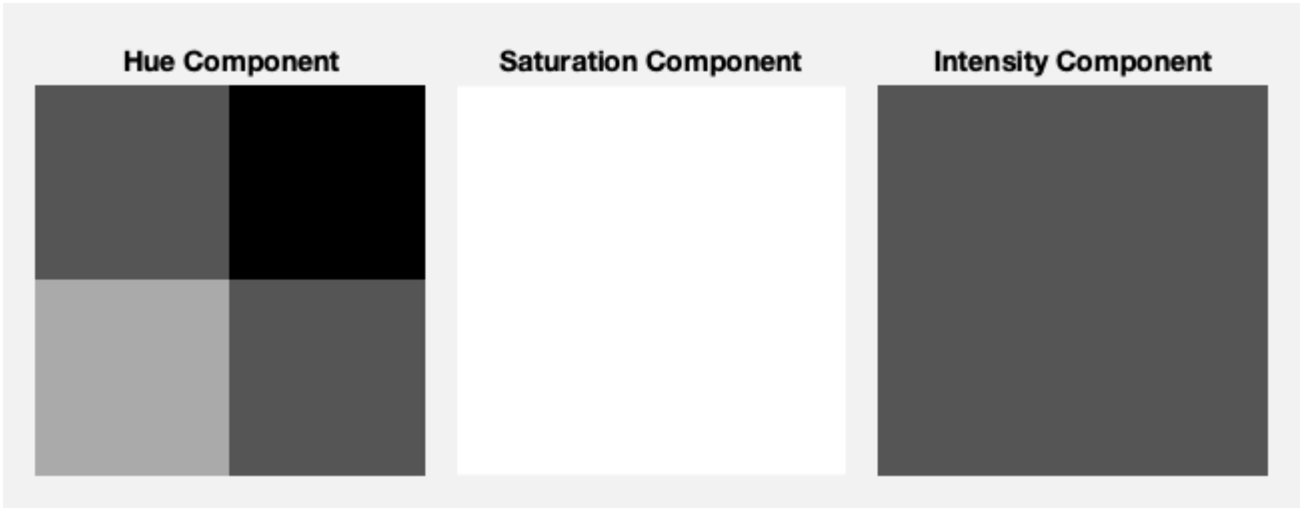
Describe the appearance of each HSI component image.

**Solution**: Recall from Fig. 7.12 that the value of hue is an angle. Because the range of values of the hue component image $H$ is normalized to $[0, 1]$, we see from that figure, for example, that as we go around the circle in the counterclockwise direction a hue value of $0$ corresponds to red, a value of $1/3$ to green, and a value of $2/3$ to blue. Thus, the important point to be made in the hue component image is that the gray value in the image corresponding to the red square should be black, the value corresponding to the green square should be lower-mid gray, and the value of the square corresponding to blue should be a lighter shade of gray than the square corresponding to green. As you can see from the left image below, this indeed is the case for the squares. For the shades of red, green, and blue in the figure below, the exact values are $H = 0$, $H = 0.33$, and $H = 0.67$, respectively.

It is given that the image is fully saturated, so the saturation component image $S$ will be constant with value $1$. This is shown in the center image below.

Similarly, all the squares are at their maximum value so, from Eq. (7-19), the intensity image $I$ also will be constant, with value $1/3$ [the maximum value of any (normalized) pixel in the RGB image is $1$, and it is given that none of the squares overlap]. This is shown in the right image below.

```
clc; close all; clear;
fRGB = imread('../artfiles/DIP4E-P7-25.png');
fHSI = rgb2hsi(fRGB);
fH = fHSI(:,:,1); fS = fHSI(:,:,2); fI = fHSI(:,:,3);
figure("Units","inches","Position",[0,0,9,3.5],...
    'Color',[0.95,0.95,0.95]);
subplot('Position',[0.025,0,0.3,0.9]); imshow(fH);
title('Hue Component','FontSize',14);
subplot('Position',[0.35,0,0.3,0.9]); imshow(fS);
title('Saturation Component','FontSize',14);
subplot('Position',[0.675,0,0.3,0.9]); imshow(fI);
title('Intensity Component','FontSize',14);
```

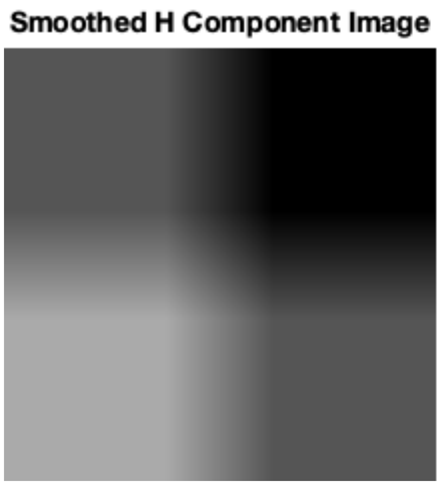


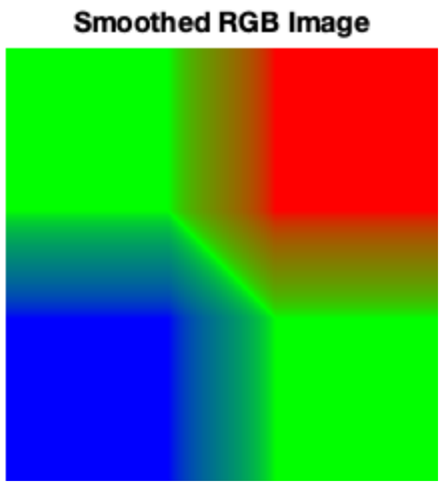
**(c) Smoothing of Hue component**

The hue component of the HSI image is smoothed using an averaging kernel of size $125 \times 125$. Describe the appearance of the result.

**Solution**: The image on the left below shows the result of blurring the hue component image. When the averaging mask is fully contained in a square, there is no blurring because the value of each square is constant. When the mask contains portions of two or more squares the value produced at the center of the mask will be between the values of the two squares, and will depend the relative proportions of the squares occupied by the mask. To see exactly what the values are, consider a point in the center of red mask in the image on the left below and a point in the center of the green mask on the top left. We know from (a) above that the value of the red point is $0$ and the value of the green point is $0.33$. Thus, the values in the blurred band between red and green vary from $0$ to $0.33$ because averaging is a linear operation. Image on the right below shows the result of generating an RGB image with the blurred hue component images and the original saturation and intensity images. The values along the line just discussed are transitions from green to red. From Fig. 7.12 we see that those transitions encompass the spectrum from green to red that includes colors such as yellow [all those colors are present in the image on the right, although they are somewhat difficult to see]. The reason for the diagonal green line in this image is that the average values along that region are nearly midway between red and blue, which we know from Fig. 7.12 is green.

```
h = fspecial('average',125);
fHsmooth = imfilter(fH,h,'conv','replicate');
fHSIsmooth = fHSI; fHSIsmooth(:,:,1) = fHsmooth;
fRGBsmooth = hsi2rgb(fHSIsmooth);
figure("Units","inches","Position",[0,0,6.5,3.5]);
subplot('Position',[0,0,3/6.5,3/3.5]); imshow(fHsmooth);
title('Smoothed H Component Image','FontSize',14);
subplot('Position',[3.5/6.5,0,3/6.5,3/3.5]); imshow(fRGBsmooth);
title('Smoothed RGB Image','FontSize',14);
```





## Problem-8 DIP4E MATLAB Project 7.2 (Page 592)

**Conversion of RGB to Gray Image**

## (a) MATLAB Function `rgb2gray4e`

**Solution**: Insert the code of your function below after the comments.

```
function g = rgb2gray4e(f,method)
%RGB2GRAY4E Converts RGB image to grayscale.
% G = RGB2GRAY4E(F,METHOD) converts 24-bit RGB color image,
% F, to an 8-bit grayscale image, G. If METHOD = 'average',
% the average method is used. If METHOD = 'ntsc', the NTSC method
% is used. This is the defauit.

%—First scale to the [0,1] range. Function intScaling4e can handle
% RGB images.
f = intScaling4e(f);

%-Set defau1t.
if nargin == 1
    method = 'ntsc';
end

%—Convert f to grayscale.
g = zeros(size(f));
switch method
    case 'average'
        g = (f(:,:,1) + f(:,:,2) + f(:,:,3))/3;
    case 'ntsc'
        g = 0.2989*f(:,:,1) + 0.5870*f(:,:,2) + 0.1140*f(:,:,3);
end

%—Convert g to uint8.
g = im2uint8(g);

end

% Insert your code below
```

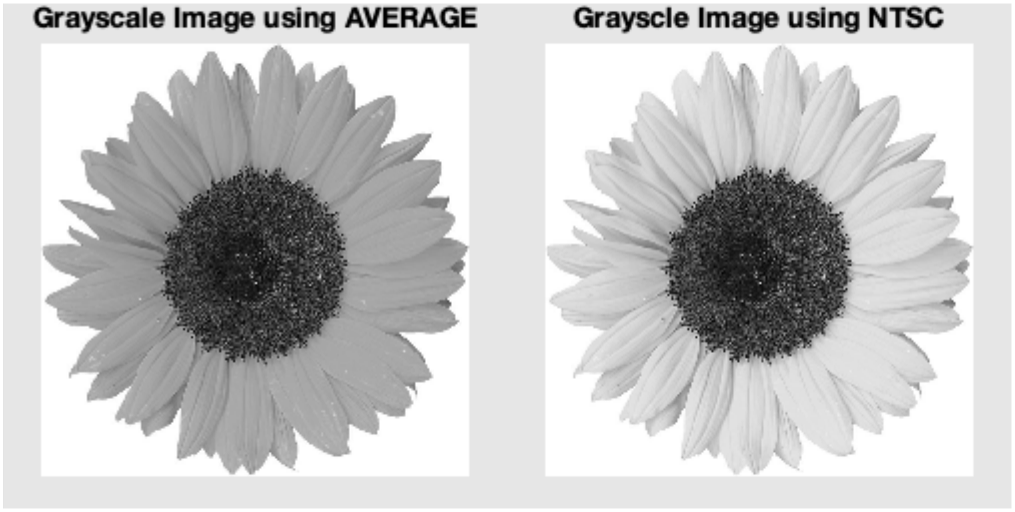
## (c) Processing of Image `sunflower.tif`

Use the function **rgbcube4e** to generate an image of the color image **sunflower.tif**. Convert the image to gray using both the '**average**' and '**ntsc**' methods. Compare their constrasts visually, and by showing the image difference. Explain the difference between the two.

**Solution**: MATLAB script.

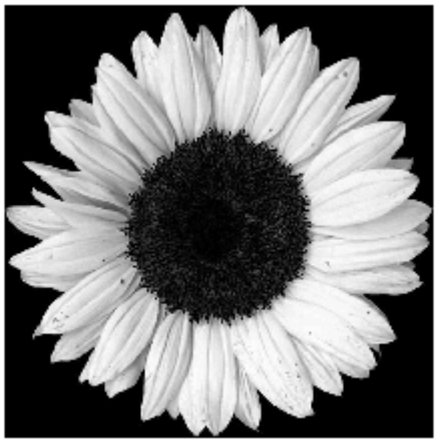
```
clc; close all; clear;
f = imread('../artfiles/sunflower.tif');
figure('Units',"inches","Position",[0,0,3.5,3.5],'Color',0.9*[1,1,1]);
subplot('Position',[0.25,0.225,3,3]/3.5);
imshow(f); title('Sunflower Image','FontSize',14);
```

```
% Convert image to gray using both methods and display the results
ga = rgb2gray4e(f,'average');
gb = rgb2gray4e(f,'ntsc');
figure('Units',"inches","Position",[0,0,7,3.5],'Color',0.9*[1,1,1]);
subplot('Position',[0.25/7,0.225/3.5,3/7,3/3.5]);
imshow(ga); title('Grayscale Image using AVERAGE','FontSize',14);
subplot('Position',[3.75/7,0.225/3.5,3/7,3/3.5]);
imshow(gb); title('Grayscle Image using NTSC','FontSize',14);
```



```
% Compute the absolute difference image
gd = abs(intScaling4e(ga) - intScaling4e(gb));
figure('Units',"inches","Position",[0,0,3.5,3.5]);
subplot('Position',[0.25,0.225,3,3]/3.5);
imshow(intScaling4e(gd,'full'));
title('Difference Image','FontSize',14);
```

Difference Image



**Comment**: From the above image, it is clearly evident that the difference image is quite different (leaving aside the scaling). The lighter the value the biggest the difference, especially in the petals area. This result is also expected by observing the two grayscale images.

---

## Problem-9 DIP4E MATLAB Project 7.6 (Page 592)

**Histogram Equalization**

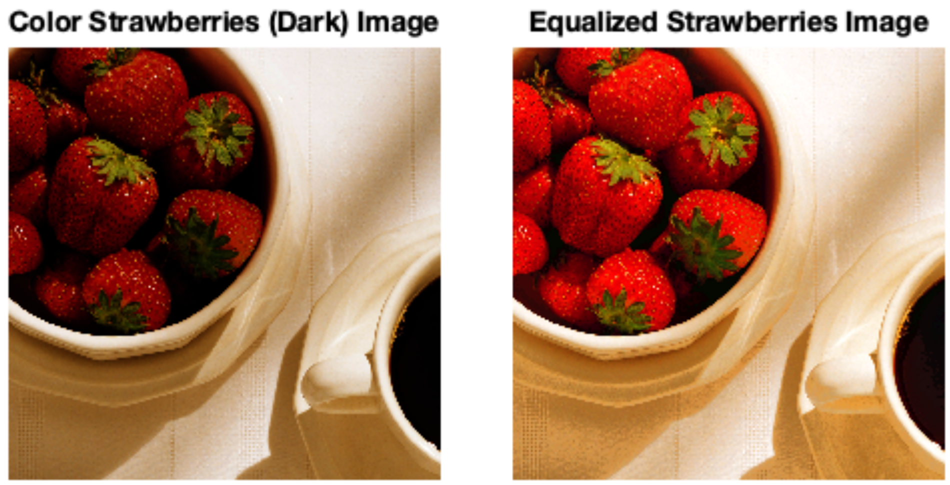
### (a) Equalization of strawberries-RGB-dark.tif Image

Read the image **`strawberries-RGB-dark.tif`** and histogram equalize it. Show the original and equalized images.

**Solution**: As discussed in class and in Example 7.11 of DIP4E, a proper approach for histogram-equalizing an RGB image is to conver it to the HSI color system and then work with the intensity component,

MATLAb script.

```
clc; close all; clear;
fRGB = imread('../artfiles/strawberries-RGB-dark.tif');
fHSI = rgb2hsi4e(fRGB); % Convert to HSI system
fI = fHSI(:,:,3); % Extract the I component
Ieq = histeq(fI); % Histogram equalize the I component
fHSIeq = fHSI; % Create equalized image in HSI system
fHSIeq(:,:,3) = Ieq; % Use the Ieq in place of the original I component
fRGBeq = hsi2rgb4e(fHSIeq); % Equalized RGB system image
figure('Units',"inches","Position",[0,0,7,3.5]);
subplot('Position',[0.25/7,0.225/3.5,3/7,3/3.5]); imshow(fRGB);
title('Color Strawberries (Dark) Image','FontSize',14);
subplot('Position',[3.75/7,0.225/3.5,3/7,3/3.5]); imshow(fRGBeq);
title('Equalized Strawberries Image','FontSize',14);
```



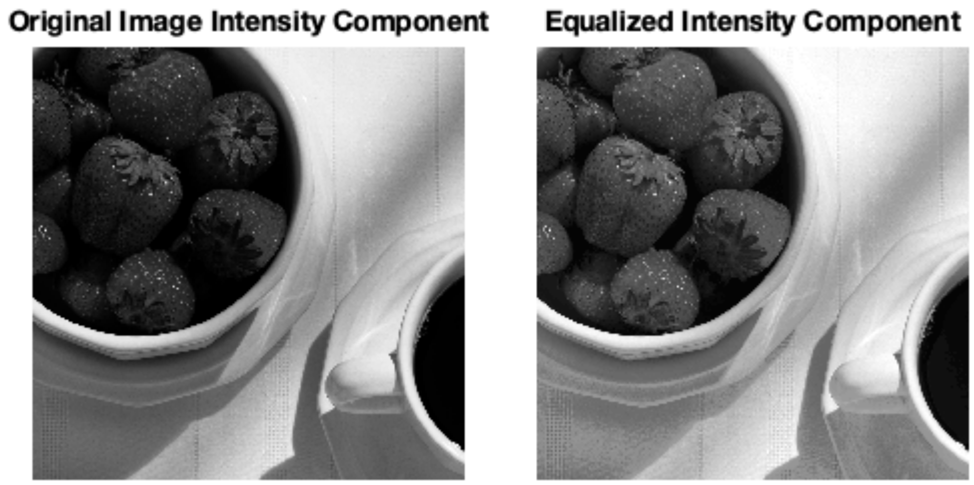
**Comment**: From the above comparison of color images, it is obvious that the histogram equalization on the intensity component did an excellent job of bringing out the color contrast.

## (b) Explanation of Results in (a)

Explain why the results you obtained look as they do. Include various images from (a) in your explanation,

**Solution**: To explain what happened in (a), let us visually compare the intensity component, **`fI`**, in the original color image and the intensity-equalized component, **`Ieq`**.

```
figure('Units',"inches","Position",[0,0,7,3.5]);
subplot('Position',[0.25/7,0.225/3.5,3/7,3/3.5]); imshow(fI);
title('Original Image Intensity Component','FontSize',14);
subplot('Position',[3.75/7,0.225/3.5,3/7,3/3.5]); imshow(Ieq);
title('Equalized Intensity Component','FontSize',14);
```


Original Image Intensity Component
Equalized Intensity Component

As you can see above from the left image, the contrast between strawberries in the original intensity component is very low. The right image shows that the histogram equaliztion did an excellent joc of spread intensity levels in strawberries, but did not significantly alter the levels in the reaming parts of the intensity image. The hue and saturation components were not altered but the intensity component was defintely altered. This improved the color contrast of the "dark" strawberries image.

## Problem-10 DIPUM3E Project 7.7 (Page 455)

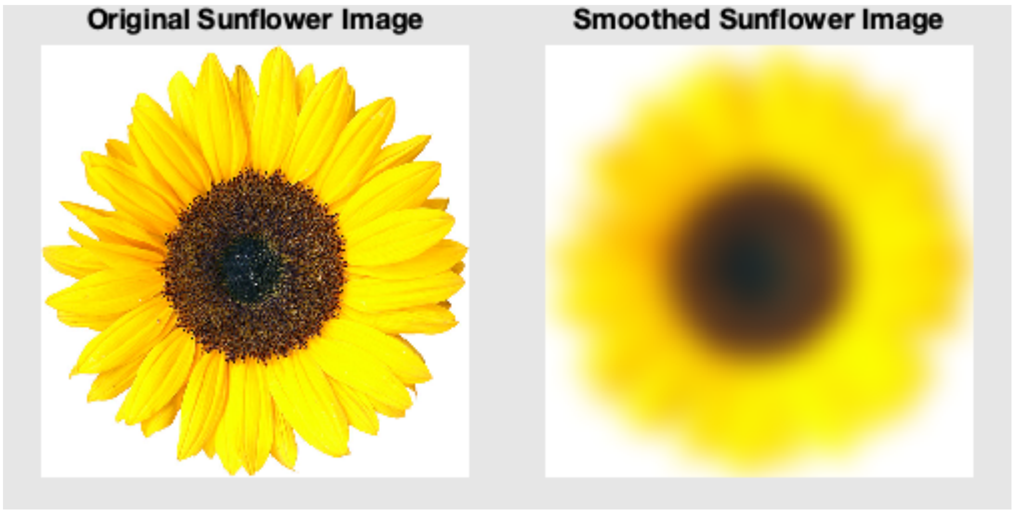
**Color Image Filtering**

### (a) Smoothing of `sunflower.tif` Image

Read the image **`sunflower.tif`** and smooth it using a Gaussin kernel large enough that the resulting image contains a dark blob in the center surrounded by a yellow region with very little detail. The colors of the white, yellow, and dark regions should be preserved.

**Solution**: MATLAB script.

```
clc; close all; clear;
f = imread('../artfiles/sunflower.tif');
fR = f(:,:,1); fG = f(:,:,2); fB = f(:,:,3);
sigma = 25; % This gives the kernel size of 2*ceil(2*25)+1 = 101
fRfilt = imgaussfilt(fR,sigma);
fGfilt = imgaussfilt(fG,sigma);
fBfilt = imgaussfilt(fB,sigma);
ffilt = cat(3,fRfilt,fGfilt,fBfilt);
figure('Units',"inches","Position",[0,0,7,3.5],'Color',0.9*[1,1,1]);
subplot('Position',[0.25/7,0.225/3.5,3/7,3/3.5]); imshow(f);
title('Original Sunflower Image','FontSize',14);
subplot('Position',[3.75/7,0.225/3.5,3/7,3/3.5]); imshow(ffilt);
title('Smoothed Sunflower Image','FontSize',14);
```



In the smoothed image on the right, all objectives of the problem were met.

### (b) Sharpening of `sunflower.tif` Image

Read the image sunflower.tif and sharpen it using the two laplacian kernels in Eqs. (3-29) and (3-30). Explain the reason for the differences in your results.

**Solution**: In the Lapacian kernels, if we add $1$ to the middle value, then the filtering effect is to add the original image back to the Laplacian image, that is,

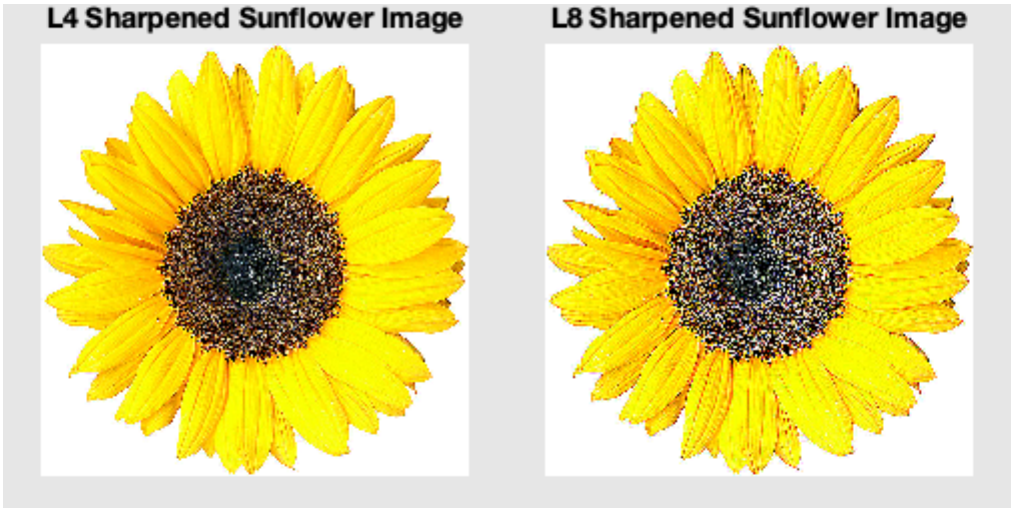
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} + \delta(m,n) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \triangleq h_{\mathrm{L4}}(m,n) \quad \Rightarrow \quad g(m,n) = f(m,n) + f(m,n) * h_{\mathrm{L4}}(m,n)$$

and

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} + \delta(m,n) = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \triangleq h_{\mathrm{L8}}(m,n) \quad \Rightarrow \quad g(m,n) = f(m,n) + f(m,n) * h_{\mathrm{L8}}(m,n)$$

MATLAB script:

```
hL4 = [0,-1,0;-1,5,-1;0,-1,0];
hL8 = [-1,-1,-1;-1,9,-1;-1,-1,-1];
ffilt4 = imfilter(f,hL4,'symmetric');
ffilt8 = imfilter(f,hL8,'symmetric');
figure('Units',"inches","Position",[0,0,7,3.5],'Color',0.9*[1,1,1]);
subplot('Position',[0.25/7,0.225/3.5,3/7,3/3.5]); imshow(ffilt4);
title('L4 Sharpened Sunflower Image','FontSize',14);
subplot('Position',[3.75/7,0.225/3.5,3/7,3/3.5]); imshow(ffilt8);
title('L8 Sharpened Sunflower Image','FontSize',14);
```



**Comments**: Observation of the above two images shows that the right image processed using the $L8$ Laplacian is slightly more sharper. The reason for this result is that the $L8$ kernel does derivatives in the diagonal directions in addition to the horizontal and vertical directions, which kernel $L4$ does not. The circular area in the center of the original image has a considerable amount of "tight' strucrure, whch responds to $L8$ kernel much more aggresively than they would to the $L4$ kernel. Note carefully that all edges of petals in the right image are sharper also.