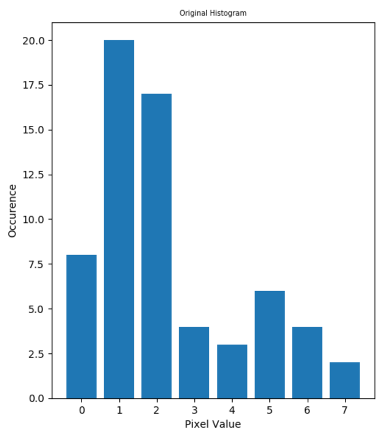
ECE 6258 Digital Image Processing
Problem Set 7

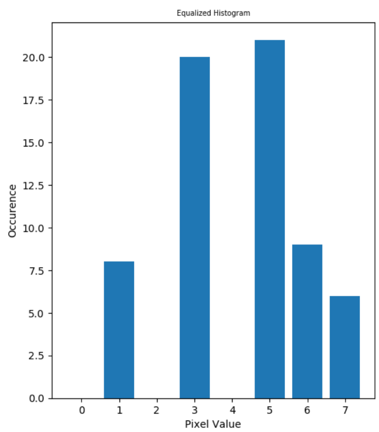
Harsh Bhate
Mail: bhate@gatech.edu

Problem 2.

r_k	n_k	$p_r(r_k) = \frac{n_k}{MN}$	$s_k = 7\sum_k p_r(r_k)$	Rounded s_k	$p_s(\tilde{s_k})$
0	8	0.125	0.875	1	0.125
1	20	0.3125	3.0625	3	0.3125
2	17	0.265625	4.921875	5	0.328125
3	4	0.0625	5.359375	5	0.328125
4	3	0.046875	5.6875	6	0.140625
5	6	0.09375	6.34375	6	0.140625
6	4	0.0625	6.78125	7	0.09375
7	2	0.03125	7	7	0.09375

Histograms



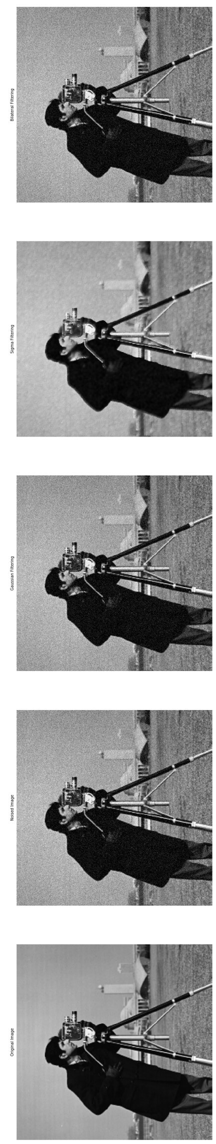


Problem 4.

MSE Values and output

```
The MSE for Gaussian Transform is 368.47.

The MSE for Sigma Transform is 355.20.

The MSE for Bilateral Transform is 368.47397.
```

Upon observing the MSE error and Visual Results, it is evident that the Sigma transform did the best job at denoising. The Sigma filter smoothened a lot of edges while significantly reducing the noise. The Gaussian Filter and Bilateral Filter ensured good edges but also contained lot of noise as evident in the north half of the image. The Gaussian filter does the best job at maintaining details as evident by the grass patches on the images.


Original Image
Noised Image
Gaussian Filtering
Sigma Filtering
Bilateral Filtering

Code

```
import dippykit as dip
import numpy as np
from skimage.restoration import denoise_bilateral as bilateral
from skimage.filters import gaussian

#Loading the image and assigning to f
f = dip.im_read("cameraman.tif")
f = dip.im_to_float(f)

#Assigning the noise to g
SIGMA = 20/255
g = dip.image_noise(f, mode='gaussian', mean=0, var=(SIGMA**2))


#Finding the Gaussian Noise
LEN, WIDTH = g.shape
SIZE = 6
sigma = 1/255
TEENY = 0.0000000000001
gaussianImage = np.zeros(g.shape)
# gaussianImage = gaussian(g,\
#                         sigma=1.0,\
#                         mode='wrap',\
#                         multichannel=False)
for n in range(LEN):
    for m in range(WIDTH):
        coeffSum = 0
        outputSum = 0
        for k in range(SIZE):
            for l in range(SIZE):
                xIndex = n - k
                yIndex = m - l
                if (xIndex < 0):
                    xIndex = LEN + xIndex
                if (yIndex < 0):
                    yIndex = WIDTH + yIndex
                coeff = (k**2+l**2)/(2*(sigma**2))
                h_kl = np.exp(-coeff)
                coeffSum = coeffSum + h_kl
                outputSum = outputSum + h_kl*g[xIndex,yIndex]
        gaussianImage[n,m] = (outputSum+TEENY)/(coeffSum+TEENY)
#Displaying the Images
dip.figure()
dip.subplot(1, 4, 1)
dip.imshow(f, 'gray')
dip.title('Original Image', fontsize='x-small')
dip.subplot(1, 4, 2)
dip.imshow(gaussianImage, 'gray')
dip.title('Gaussian Filtering', fontsize='x-small')
#Printing the MSE
mseGaussian = dip.metrics.MSE(f*255, gaussianImage*255)
print ("\n\tThe MSE for Gaussian Transform is %.2f.\n"%mseGaussian)

#Computing the Sigma Filter
SIZE = 6
```

```
p = 40/255
TEENY = 0.0000000000001
sigmaImage = np.zeros(g.shape)
for n in range(LEN):
    for m in range(WIDTH):
        coeffSum = 0
        outputSum = 0
        for k in range(SIZE):
            for l in range(SIZE):
                xIndex = n - k
                yIndex = m - l
                if (xIndex < 0):
                    xIndex = LEN + xIndex
                if (yIndex < 0):
                    yIndex = WIDTH + yIndex
                coeff = (g[xIndex,yIndex]**2 - g[n,m])**2
                coeff = coeff/(2*(p**2))
                h_kl = np.exp(-coeff)
                coeffSum = coeffSum + h_kl
                outputSum = outputSum + h_kl*g[xIndex,yIndex]
        sigmaImage[n,m] = (outputSum+TEENY)/(coeffSum+TEENY)
# Displaying the Images
dip.subplot(1, 4, 3)
dip.imshow(sigmaImage, 'gray')
dip.title('Sigma Filtering', fontsize='x-small')
#Printing the MSE
mseSigma = dip.metrics.MSE(f*255, sigmaImage*255)
print ("\n\tThe MSE for Sigma Transform is %.2f.\n"%mseSigma)



#Computing the Bilateral Filter
SIZE = 12
p = 50.0/255
sigma = 2/255
TEENY = 0.0000000000001
bilateralImage = np.zeros(g.shape)

# bilateralImage = bilateral(g,\
#                  win_size=SIZE, \
#                 sigma_spatial=1.0,\
#                 mode='symmetric',\
#                 multichannel=False)

for n in range(LEN):
    for m in range(WIDTH):
        coeffSum = 0
        outputSum = 0
        for k in range(SIZE):
            for l in range(SIZE):
                xIndex = n - k
                yIndex = m - l
                if (xIndex < 0):
                    xIndex = LEN + xIndex
                if (yIndex < 0):
                    yIndex = WIDTH + yIndex
                coeff_1 = (g[xIndex,yIndex]**2 - g[n,m])**2
```

```
                coeff_1 = coeff_1/(2*(p**2))
                coeff_2 = (k**2+l**2)/(2*(sigma**2))
                h_kl = np.exp(-coeff_1)*np.exp(-coeff_2)
                coeffSum = coeffSum + h_kl
                outputSum = outputSum + h_kl*g[xIndex,yIndex]
        bilateralImage[n,m] = (outputSum+TEENY)/(coeffSum+TEENY)
#Printing the MSE
mseBilateral = dip.metrics.MSE(f*255, bilateralImage*255)
print ("\n\tThe MSE for Bilateral Transform is %.5f.\n"%
    mseBilateral)
# Displaying the Images
dip.subplot(1, 4, 4)
dip.imshow(bilateralImage, 'gray')
dip.title('Bilateral Filtering', fontsize='x-small')
dip.show()
```

Problem 5.

Code

```python
import dippykit as dip
import numpy as np

#Resolution Guide
#4CIF Resolution = 704 x 480
#CIF Resolution = 352 x 240

X = dip.im_read("coatOfArms.png")
X = dip.im_to_float(X)
X = X*255

#Filter
SIZE = 25
filterSize = (SIZE, SIZE)
minor = 5**2
mild = 25**2
severe = 100**2

minorWindow = dip.window_2d (filterSize, window_type='gaussian', variance = minor )
mildWindow = dip.window_2d (filterSize, window_type='gaussian', variance = mild )
severeWindow = dip.window_2d (filterSize, window_type='gaussian', variance = severe )

#Output of the Filter
minorX = dip.convolve2d(X, minorWindow, mode = 'same', boundary = 'wrap')
mildX = dip.convolve2d(X, mildWindow, mode = 'same', boundary = 'wrap')
severeX = dip.convolve2d(X, severeWindow, mode = 'same', boundary = 'wrap')

#Laplacian Filtering
LaplaceKernel = np.array([[0,1,0],
                          [1,-4,1],
                          [0,1,0]],
                          dtype=np.float)

edgeMinorX = dip.convolve2d(X, LaplaceKernel, mode = 'same', boundary = 'wrap')
edgeMildX = dip.convolve2d(X, LaplaceKernel, mode = 'same', boundary = 'wrap')
edgeSevereX = dip.convolve2d(X, LaplaceKernel, mode = 'same', boundary = 'wrap')

#PSNR Calculations

PSNR_minor = dip.metrics.PSNR(dip.float_to_im(X/255), dip.float_to_im(edgeMinorX/255))
PSNR_mild = dip.metrics.PSNR(dip.float_to_im(X/255), dip.float_to_im(edgeMildX/255))
PSNR_severe = dip.metrics.PSNR(dip.float_to_im(X/255), dip.float_to_im(edgeSevereX/255))
```

```

#Displaying PSNR Output
print ("\n\tPSNR for Laplacian with minor blurring is %.2f \n" % PSNR_minor)
print ("\n\tPSNR for Laplacian with mild blurring is %.2f \n" % PSNR_mild)
print ("\n\tPSNR for Laplacian with severe blurring is %.2f \n" % PSNR_severe)

#Extended Laplacian Filtering
extLaplaceKernel = np.array([[1,1,1],
                             [1,-8,1],
                             [1,1,1]],
                             dtype=np.float)

extEdgeMinorX = dip.convolve2d(X, extLaplaceKernel, mode = 'same', boundary = 'wrap')
extEdgeMildX = dip.convolve2d(X, extLaplaceKernel, mode = 'same', boundary = 'wrap')
extEdgeSevereX = dip.convolve2d(X, extLaplaceKernel, mode = 'same', boundary = 'wrap')

#PSNR Calculations
PSNR_minor = dip.metrics.PSNR(dip.float_to_im(X/255), dip.float_to_im(extEdgeMinorX/255))
PSNR_mild = dip.metrics.PSNR(dip.float_to_im(X/255), dip.float_to_im(extEdgeMildX/255))
PSNR_severe = dip.metrics.PSNR(dip.float_to_im(X/255), dip.float_to_im(extEdgeSevereX/255))

#Displaying PSNR Output
print ("\n\tPSNR for extended Laplacian with minor blurring is %.2f \n" %PSNR_minor)
print ("\n\tPSNR for extended Laplacian with mild blurring is %.2f \n" %PSNR_mild)
print ("\n\tPSNR for extended Laplacian with severe blurring is %.2f \n" %PSNR_severe)

#Image Output
dip.figure()
dip.subplot(3, 4, 1)
dip.imshow(X, 'gray')
dip.title('Original Image', fontsize='x-small')
dip.subplot(3, 4, 2)
dip.imshow(minorX, 'gray')
dip.title('Minor Blurring', fontsize='x-small')
dip.subplot(3, 4, 3)
dip.imshow(mildX, 'gray')
dip.title('Mild Blurring', fontsize='x-small')
dip.subplot(3, 4, 4)
dip.imshow(severeX, 'gray')
dip.title('Severe Blurring', fontsize='x-small')
dip.subplot(3, 4, 5)
dip.imshow(edgeMinorX, 'gray')
dip.title('Laplacian on Minor Blurring', fontsize='x-small')
dip.subplot(3, 4, 6)
dip.imshow(edgeMildX, 'gray')
```

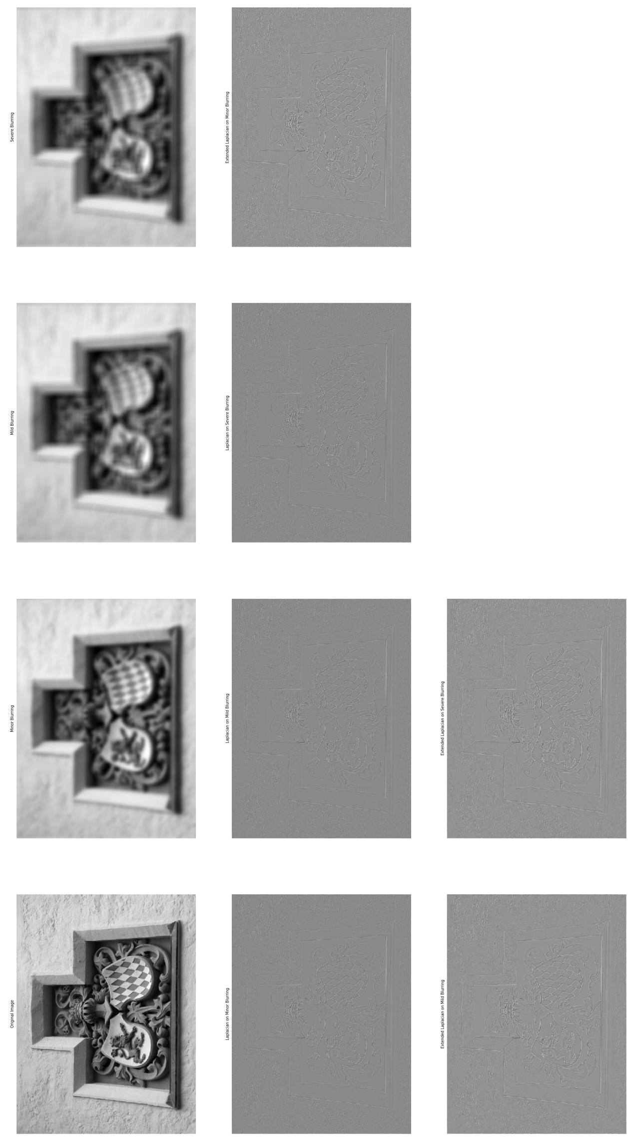
```
dip.title('Laplacian on Mild Blurring', fontsize='x-small')
dip.subplot(3, 4, 7)
dip.imshow(edgeSevereX, 'gray')
dip.title('Laplacian on Severe Blurring', fontsize='x-small')
dip.subplot(3, 4, 8)
dip.imshow(extEdgeMinorX, 'gray')
dip.title('Extended Laplacian on Minor Blurring', fontsize='x-small')
dip.subplot(3, 4, 9)
dip.imshow(extEdgeMildX, 'gray')
dip.title('Extended Laplacian on Mild Blurring', fontsize='x-small')
dip.subplot(3, 4, 10)
dip.imshow(extEdgeSevereX, 'gray')
dip.title('Extended Laplacian on Severe Blurring', fontsize='x-small')
dip.show()
```

Outputs

```
PSNR for Laplacian with minor blurring is 27.73

PSNR for Laplacian with mild blurring is 27.73

PSNR for Laplacian with severe blurring is 27.73


PSNR for extended Laplacian with minor blurring is 27.78

PSNR for extended Laplacian with mild blurring is 27.78

PSNR for extended Laplacian with severe blurring is 27.78
```

Observations

It can be observed that as blurring increases, the Laplacian filter loses track of fine edge details while maintaining structural edge details. Often, weakly blurred images leave a lot of lose edge details. At the same time, a strongly blurred image loses a lot of finer details.
It was also observed that an extended Laplacian filter provides a much finer edge details as compared to regular Laplacian filter.


Severe Blurring
Extended Laplacian on Minor Blurring
Mild Blurring
Laplacian on Severe Blurring
Minor Blurring
Laplacian on Mild Blurring
Extended Laplacian on Severe Blurring
Original Image
Laplacian on Minor Blurring
Extended Laplacian on Mild Blurring

Problem 7.

The edge case of the matrix are the elements a and i. The operations on these pixels will appear as follows:

$$a = \frac{1}{4} \cdot i + \frac{1}{2} \cdot a + \frac{1}{4} \cdot b$$

$$i = \frac{1}{4} \cdot j + \frac{1}{2} \cdot i + \frac{1}{4} \cdot a$$

For all other cases, three successive elements will be added. Thus, each row of a colum will have circular symmetry at the edge cases. Thus, the filter H is:

$$\mathbf{H} = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} \end{bmatrix}$$

The output of the filter for a 3×3 image f vectorized as 9×1 is given by:

$$\hat{f} = H \cdot f$$

where $\hat{f} \in R^{9\times1}$, $H \in R^{9\times9}$ and $f \in R^{9\times1}$.

Problem 9.

f. True and Estimated Variance Values

Image Number	1	2	3	4	5
σ^2	0.018	0.027	0.032	0.036	0.059
$\widehat{\sigma_f^2}$	0.00125	0.01031	0.015029	0.01898	0.04368
$\widehat{\sigma_t^2}$	0.0104	0.0191	0.02411	0.02862	0.0437

Code

```python

import numpy as np
import dippykit as dip

def noise_estimation(im: np.ndarray):
    """
    Given an image, this function determines the variance in its pixel
    values and displays a histogram of the image along with a fitted normal
    distribution.
    """
    # Calculate histogram
    im_hist, bin_edges = np.histogram(im, 30)
    bin_centers = bin_edges[:-1] + (np.diff(bin_edges) / 2)
    # Normalize histogram to make a PDF
    im_hist = im_hist.astype(float)
    im_hist /= np.sum(im_hist)
    # Calculate the mean and variance values
    mean = np.sum(im_hist * bin_centers)
    var = np.sum(im_hist * (bin_centers ** 2)) - (mean ** 2)
    # Calculate the values on the normal distribution PDF
    norm_vals = np.exp(-((bin_centers - mean) ** 2) / (2 * var)) \
            / np.sqrt(2 * np.pi * var)
    # Normalize the norm_vals
    norm_vals /= np.sum(norm_vals)
    # Rescale variance
    var /= (255 ** 2)
    print('Variance: {}'.format(var))
    dip.figure()
    dip.bar(bin_centers, im_hist)
    dip.plot(bin_centers, norm_vals, 'r')
    dip.legend(['Fitted Gaussian PDF', 'Histogram of image'])
    dip.xlabel('Pixel value')
    dip.ylabel('Occurrence')
    dip.show()

def main():
    ######## PART (a): EDIT HERE ########
    img = img = dip.im_read("WiseonRocks_noise_1.png")
    # img = dip.im_to_float(img)
    # img = img*255
    print ("The shape of IMG is %s \n" %str(img.shape))
    ######## PART (b): EDIT HERE ########
```

```
    noise_estimation(img)

    ######## PART (c)-(e): EDIT HERE ########
    SIZE = 100
    print ("Image 1")
    flatRegion = img[0:SIZE, 100:(100+SIZE)]
    notFlatRegion = img[250:(250+SIZE), 150:(150+SIZE)]
    noise_estimation(flatRegion)
    noise_estimation(notFlatRegion)

    print ("Image 2")
    img = img = dip.im_read("WiseonRocks_noise_2.png")
    flatRegion = img[0:SIZE, 100:(100+SIZE)]
    noise_estimation(img)
    notFlatRegion = img[250:(250+SIZE), 150:(150+SIZE)]
    noise_estimation(flatRegion)
    noise_estimation(notFlatRegion)

    print ("Image 3")
    img = img = dip.im_read("WiseonRocks_noise_3.png")
    flatRegion = img[0:SIZE, 100:(100+SIZE)]
    noise_estimation(img)
    notFlatRegion = img[250:(250+SIZE), 150:(150+SIZE)]
    noise_estimation(flatRegion)
    noise_estimation(notFlatRegion)

    print ("Image 4")
    img = img = dip.im_read("WiseonRocks_noise_4.png")
    flatRegion = img[0:SIZE, 100:(100+SIZE)]
    noise_estimation(img)
    notFlatRegion = img[250:(250+SIZE), 150:(150+SIZE)]
    noise_estimation(flatRegion)
    noise_estimation(notFlatRegion)

    print ("Image 5")
    img = img = dip.im_read("WiseonRocks_noise_5.png")
    flatRegion = img[0:SIZE, 100:(100+SIZE)]
    noise_estimation(img)
    notFlatRegion = img[250:(250+SIZE), 150:(150+SIZE)]
    noise_estimation(flatRegion)
    noise_estimation(notFlatRegion)

if __name__ == '__main__':
    main()
```