Grad and Denoise: HW 3

Problem #2: Programming Exercise 1

Image Derivatives a) to c)

The image derivatives were gathered by using a shifting in x and y in order to have a similar property of circulant matrices. This results in the same size output as the input. The figure below shows the image derivative in both directions and the total magnitude. From this the edges of the images can be clearly seen.
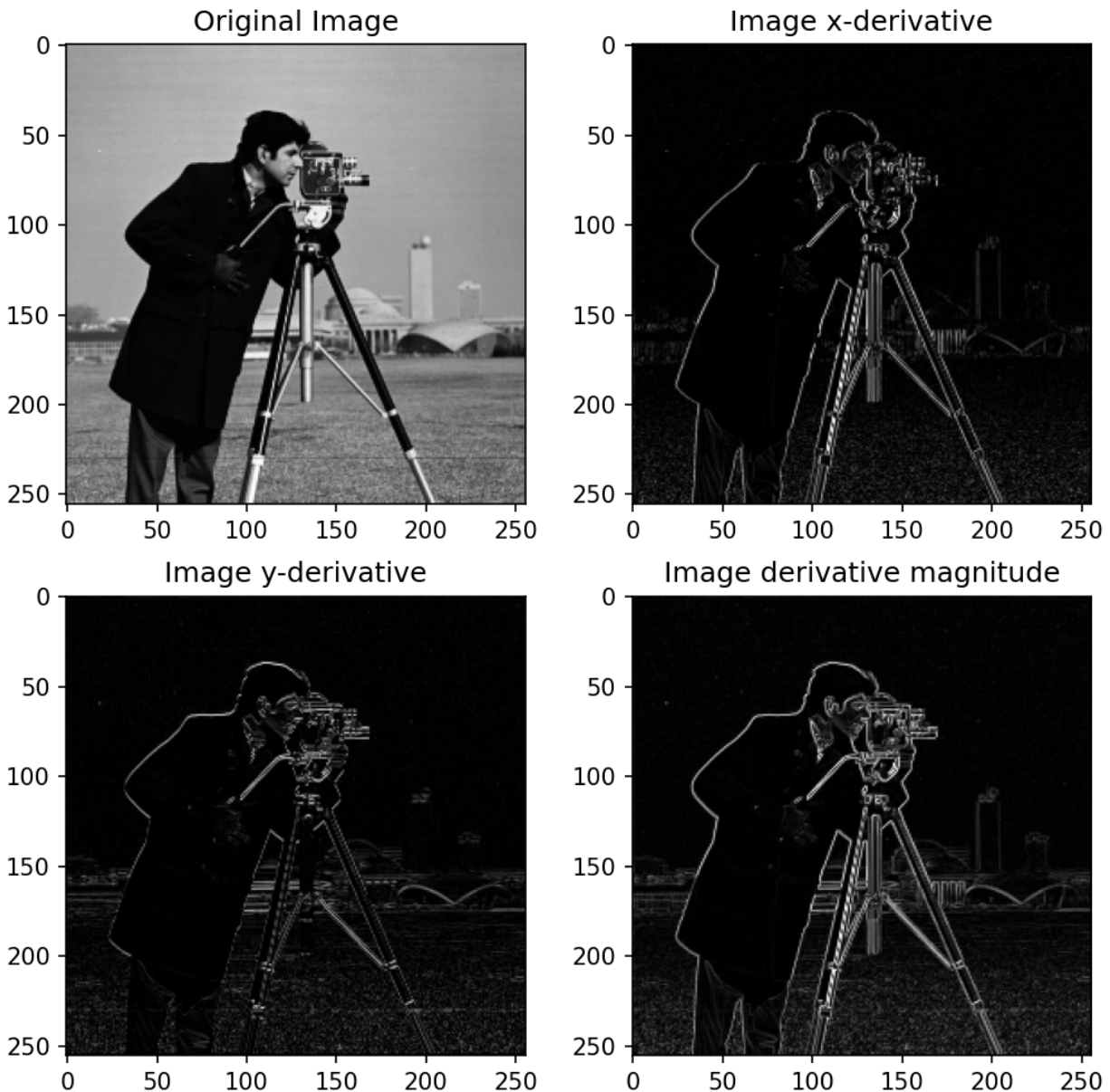

Figure 1: Image Derivative in x,y and magnitude

The whole image has a size of 256 by 256. The DCT of each 8 by 8 block is taken which means that there are a total of 32 by 32 sets of blocks. Although DCT is used, the form of the man can still be seen a bit which

is close to the edge data.
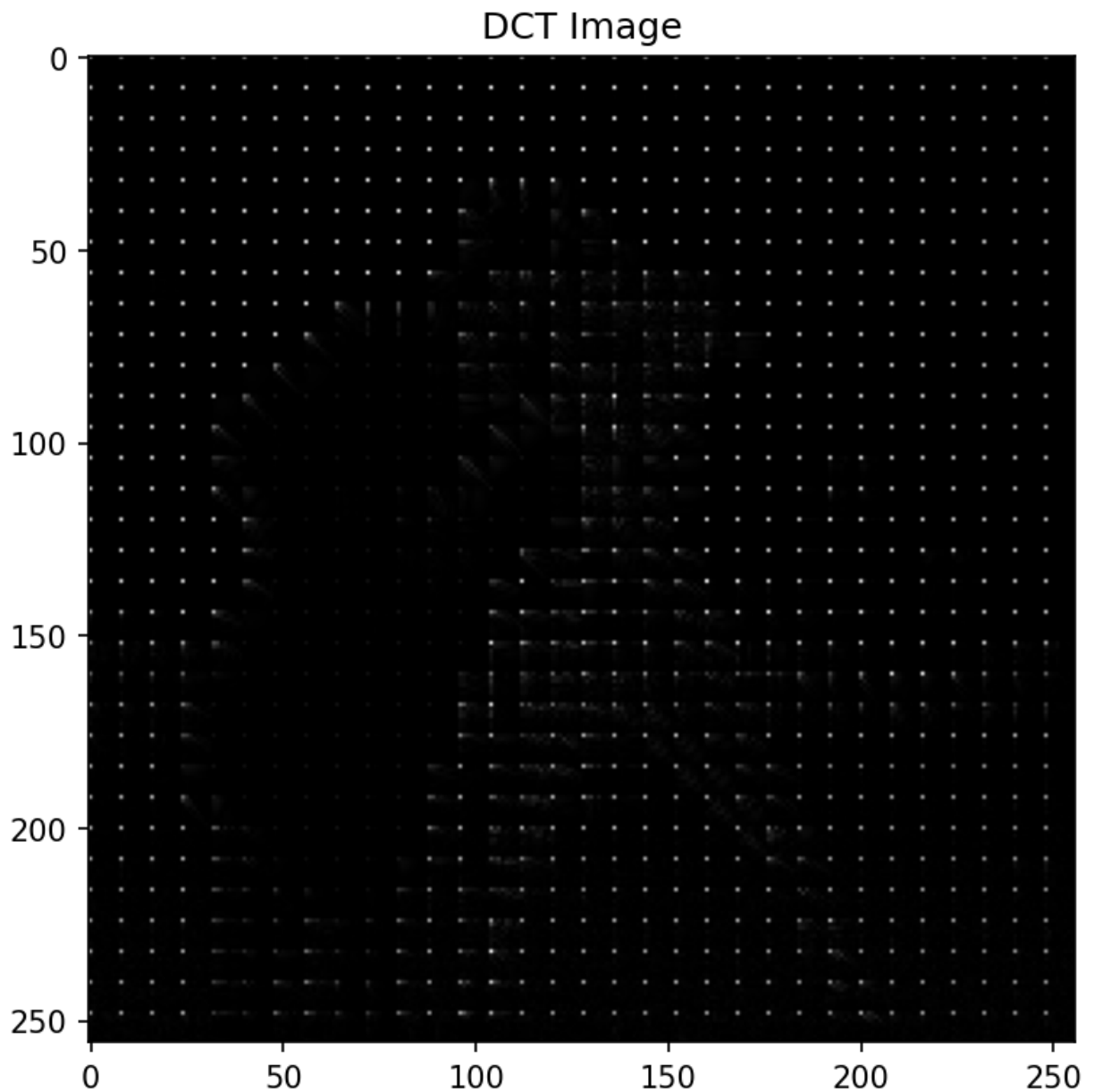
d) Discrete Cosine Transform (DCT)

Figure 2: DCT on each 8x8 block

Problem #3: Programming Exercise 2

a) Proximal Gradient Descent

This is the result of the blurry image after 7500 iterations. It can be seen that the deblurred image looks fairly nice.
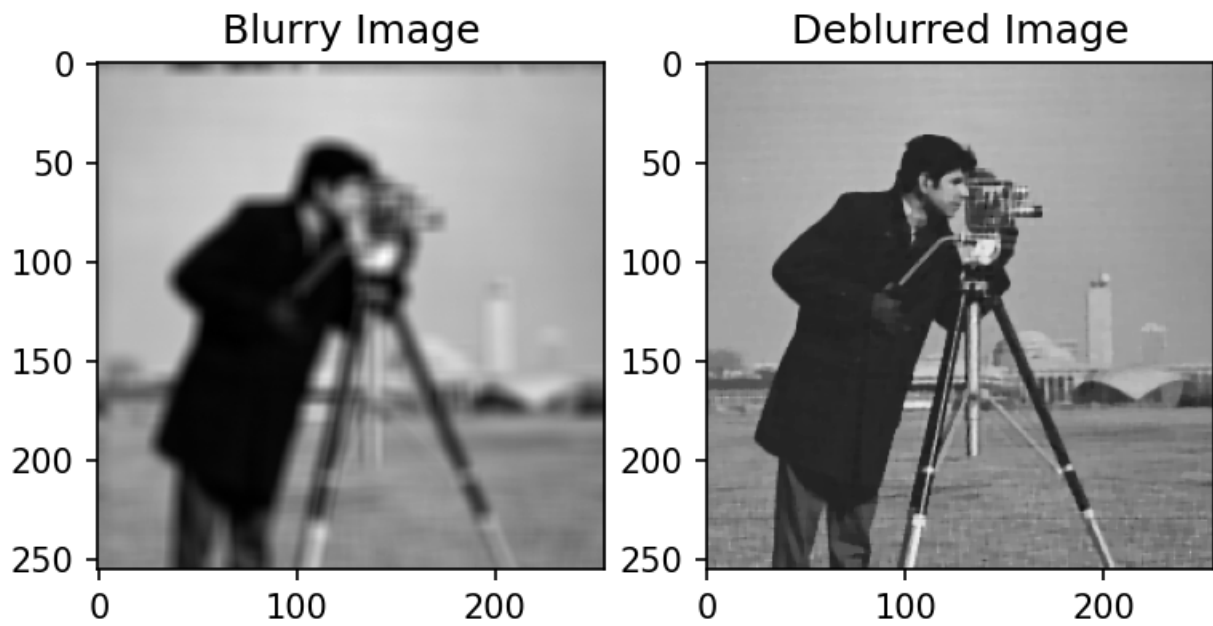
Figure 3: Denoising using PGD

However from the image below, it can be seen that a substantial result doesn't come until the 5000th iteration. The picture isn't as bad as the original image but the first few shots still look like a basic blurred image.
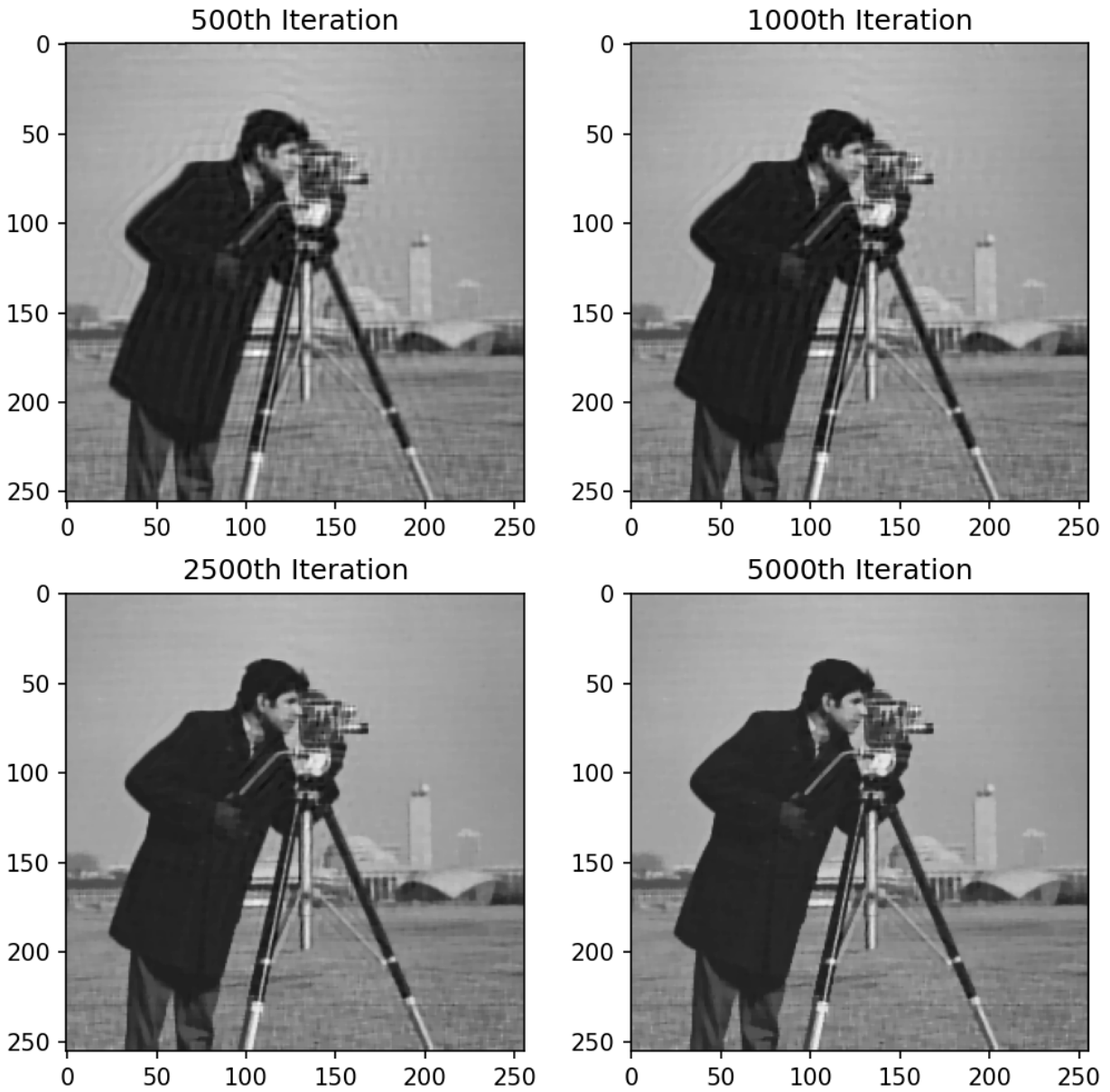
Figure 4: Intermediate Values of PGD Denoising

a) Proximal Gradient Descent

Using proximal gradient descent is lowkey interesting and there are a bit of extra steps to be done. However, I can say that the time needed is lower than PGD. At the same time the deblurred image looks pretty good aside from the small waves that can be seen.
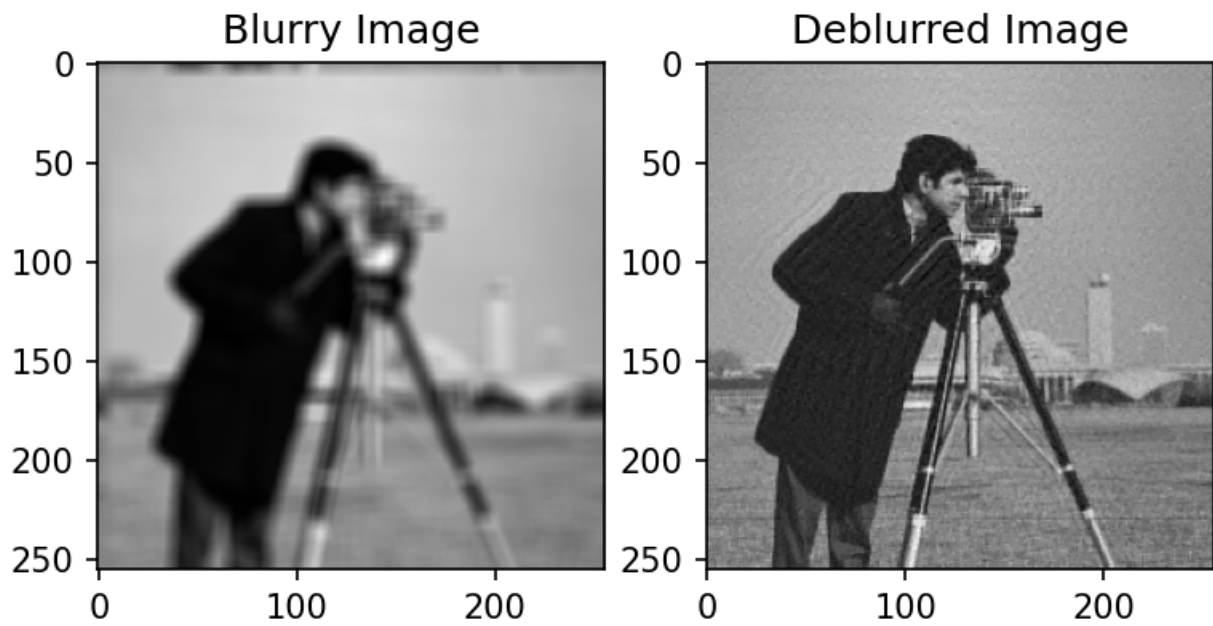
Figure 5: Denoising using ADMM

This is especially more amazing, when one notes that this is just done within 500 iterations which is much less than the 7500 iterations for PGD. At the same time, the first few shots aren't as good but the jump from 250 to 500 iterations is a big one which shows the speed of convergence of the ADMM method.
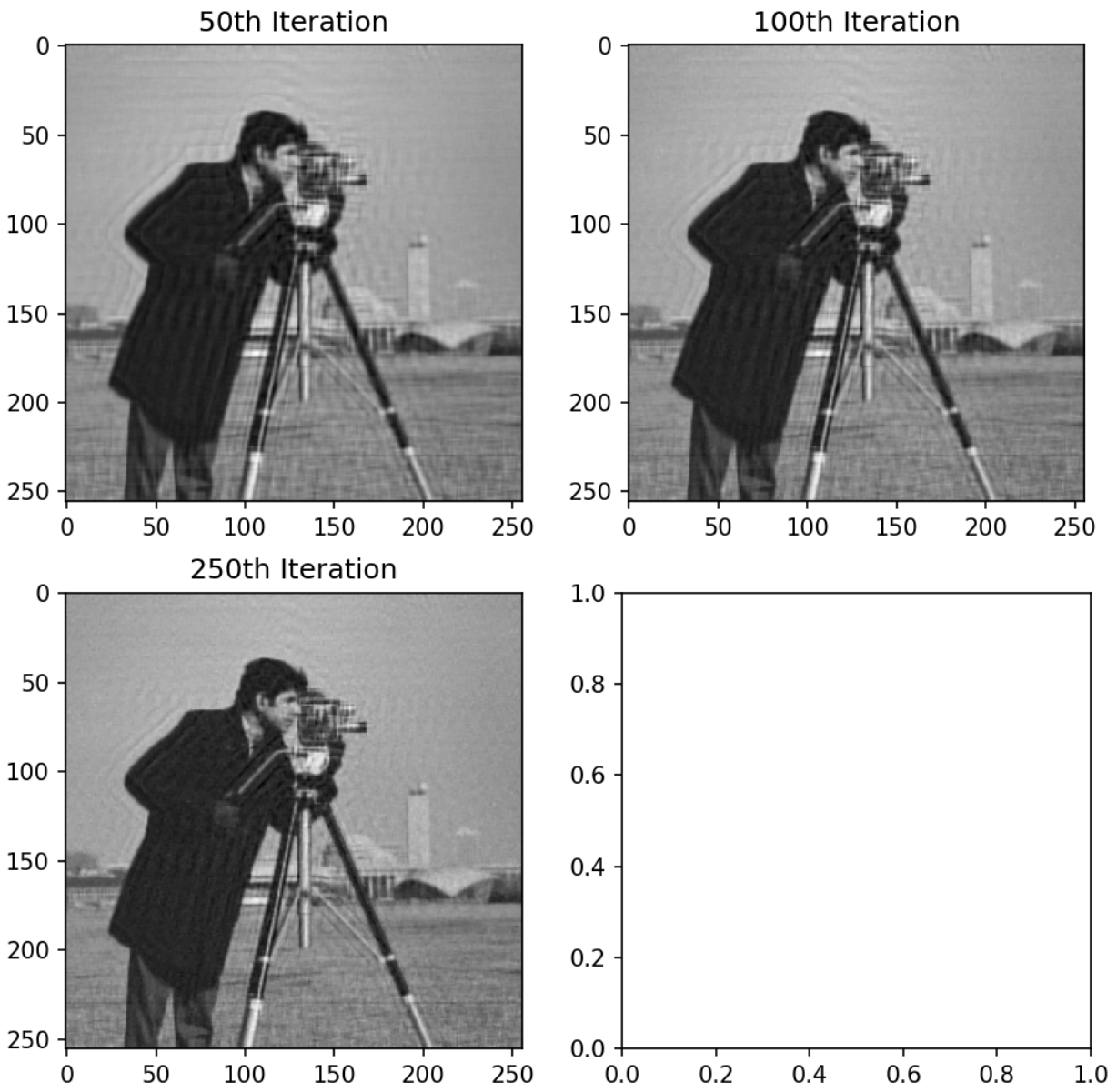
Figure 6: Intermediate Values of ADMM Denoising

Appendix

Problem 2

```
1
2  '''
3  Justine Serdoncillo
4  EE 5561 - Image Processing
5  Problem Set 3
6  November 7, 2023
7  '''
8
9  # %% Problem Statement
10 """
11     2) [9 pts] Programming Exercise 1: In this exercise, you will familiarize yourself with
12     function handles. Write function handles to calculate the following:
13     a) The image derivative in x direction (the built-in function numpy.diff(Python)/diff(
        MATLAB)
14     may be useful),
15     b) the image derivative in y direction,
16     c) the magnitude of the gradient vector in (x, y) directions,
17     d) discrete cosine transform (you may use the scipy.fftpack.dct(Python)/dct2(MATLAB)
18     function for this) of each 8    8 distinct block in the image.
19     For a, b, and c make the derivative operator circulant. Thus, the output should be the
        same
20     size as the image. Verify all 4 function handles on the cameraman image.
21 """
22 # %%
23
24 import numpy as np
25 import matplotlib.pyplot as plt
26 import matplotlib.image as img
27 import scipy
28 import imageio as iio
29 from scipy.fftpack import dct, idct
30 from scipy.signal import convolve2d
31
32 # %% Function Handles
33 def gradX(image):
34     shift = np.roll(image, 1, axis=1)
35     x = image - shift
36     return x
37
38 def gradY(image):
39     shift = np.roll(image, 1, axis=0)
40     y = image - shift
41     return y
42
43 def gradMag(image):
44     x = gradX(image)
45     y = gradY(image)
46     mag = np.sqrt(x**2 + y**2)
47     return mag
48
49 def dct2c(img):
50     return dct(dct(img, axis=0, norm = 'ortho'), axis=1, norm = 'ortho')
51
52 def DCT(image, size):
53     [Nx,Ny] = image.shape
54     dct = np.zeros_like(image)
55     for x in range(0,Nx,size):
56         for y in range(0,Ny,size):
57             dct[x:x+8,y:y+8] = dct2c(image[x:x+8,y:y+8])
58     return dct
59
60
61 # %% Importing Images
62 man_img  = np.complex64(iio.imread('cameraman.tif'))
63
64 # %% Problem 2
65 def prob2():
```

```python
66      print(f"Cameraman has size {man_img.shape}")
67      x = gradX(man_img)
68      y = gradY(man_img)
69      M = gradMag(man_img)
70      D = DCT(man_img, 8)
71      print(f"Derivatives has size {x.shape}")
72
73      # Plot the figures
74      fig, ax = plt.subplots(2,2, figsize=(8,8), dpi=150)
75      ax[0,0].imshow(np.abs(man_img), cmap="gray")
76      ax[0,0].set_title("Original Image")
77      ax[0,1].imshow(np.abs(x), cmap="gray")
78      ax[0,1].set_title("Image x-derivative")
79      ax[1,0].imshow(np.abs(y), cmap="gray")
80      ax[1,0].set_title("Image y-derivative")
81      ax[1,1].imshow(np.abs(M), cmap="gray")
82      ax[1,1].set_title("Image derivative magnitude")
83
84      fig, ax = plt.subplots(figsize=(6,6), dpi=150)
85      ax.imshow(np.abs(D), cmap="gray")
86      ax.set_title("DCT Image")
87
88  # %% Main Function
89  if __name__ == "__main__":
90      prob2()
```

Problem 3

```python
from scipy.io import loadmat
import matplotlib.pyplot as plt
import numpy as np
from cg_solve import cg_solve
try:
    from skimage.restoration import denoise_tv_chambolle as TV_denoise
except ImportError:
    from skimage.filters import denoise_tv_chambolle as TV_denoise

# example usage of the functions:
# ----cg_solve------------------------------------------
#     x_recon = A^-1.b
#     no_iter: numver of iterations
#     x_recon = cg_solve(b,A,no_iter)
# ----TV_denoise----------------------------------------
#     weight: Weight of the TV penalty term
#     x_denoised = TV_denoise(x_noisy, weight)


# %% Load the image
I = plt.imread('cameraman.tif').astype(np.float64) # original image
y = loadmat('Assignment3_blurry_image.mat')['y']   # blurred image

# Blur Kernel
ksize = 9
kernel = np.ones((ksize,ksize)) / ksize**2

[h, w] = I.shape
kernelimage = np.zeros((h,w))
kernelimage[0:ksize, 0:ksize] = np.copy(kernel)
fftkernel = np.fft.fft2(kernelimage)

sigm = np.sqrt(0.1)
alpha = np.sqrt(sigm**2/ np.max(np.abs(fftkernel)))

def H(x):
    return np.real(np.fft.ifft2(np.fft.fft2(x) * fftkernel))

def HT(x):
    return np.real(np.fft.ifft2(np.fft.fft2(x) * np.conj(fftkernel)))

def A(x):
    return HT(H(x)) + rho*x

# %% Here, implement proximal gradient
step_size = alpha**2/sigm**2
# JJ's code starts here
x = np.copy(y)
aTV = 1e-2
save = [500, 1000, 2500, 5000]
saves = []
for ind in range(7500):
    x_n = x + step_size * HT(y - H(x))
    x = TV_denoise(x_n, weight=aTV)
    if ind in save:
        saves.append(x)

fig, ax = plt.subplots(1,2, figsize=(6,6), dpi=150)
ax[0].imshow(y, cmap="gray")
ax[0].set_title("Blurry Image")
ax[1].imshow(x, cmap="gray")
ax[1].set_title("Deblurred Image")

fig, ax = plt.subplots(2,2, figsize=(8,8), dpi=150)
ax[0,0].imshow(saves[0], cmap="gray")
ax[0,0].set_title("500th Iteration")
ax[0,1].imshow(saves[1], cmap="gray")
ax[0,1].set_title("1000th Iteration")
ax[1,0].imshow(saves[2], cmap="gray")
ax[1,0].set_title("2500th Iteration")
```

```python
71  ax[1,1].imshow(saves[3], cmap="gray")
72  ax[1,1].set_title("5000th Iteration")
73
74  # %% Here, implement ADMM
75  # JJ's code starts here
76  z = np.zeros_like(y)
77  u = np.zeros_like(y)
78  aTV = 1e-2
79  rho = 0.1
80  no_iter=20
81  save = [50, 100, 250]
82  saves = []
83  x = np.copy(z)
84  for ind in range(500):
85      b = HT(y) + rho*(z-u)
86
87      # cg_solve equivalent
88      x0 = np.zeros_like(b)
89      xcg = np.copy(x0)
90      r = b - A(xcg)
91      p = np.copy(r)
92      rsold = np.sum(r * np.conj(r))
93      for i in range(no_iter):
94          Ap = A(p)
95          alpha = rsold/np.sum(p * Ap)
96          xcg = xcg + alpha * p
97          r = r - alpha * Ap
98          rsnew = np.sum(r * np.conj(r))
99          p = r + rsnew / rsold * p
100         rsold = np.copy(rsnew)
101     x = xcg.copy()
102
103     z = TV_denoise(x + u, weight=aTV)
104     u += x - z
105     if ind in save:
106         saves.append(x)
107
108 fig, ax = plt.subplots(1,2, figsize=(6,6), dpi=150)
109 ax[0].imshow(y, cmap="gray")
110 ax[0].set_title("Blurry Image")
111 ax[1].imshow(x, cmap="gray")
112 ax[1].set_title("Deblurred Image")
113
114 fig, ax = plt.subplots(2,2, figsize=(8,8), dpi=150)
115 ax[0,0].imshow(saves[0], cmap="gray")
116 ax[0,0].set_title("50th Iteration")
117 ax[0,1].imshow(saves[1], cmap="gray")
118 ax[0,1].set_title("100th Iteration")
119 ax[1,0].imshow(saves[2], cmap="gray")
120 ax[1,0].set_title("250th Iteration")
```