Justine John A. Serdoncillo — HW 3

EE5761 - Image Processing & Applications — November 7, 2023

1) $\underline{\underline{A}} \in \mathbb{C}$, $M \times N$ matrix of rank $r \leq \min(M,N)$

a) Show that $\underline{\underline{A}}^* \underline{\underline{A}}$ is a Hermitian matrix & is positive semi-definite

$\underline{\underline{A}}^* = \overline{A_{ji}}$, $\underline{\underline{A}} = A_{ij}$

$$\underset{[N\times M]}{\underline{\underline{A}}^*} \; \underset{[M\times N]}{\underline{\underline{A}}} = \underset{[N\times N]}{\underline{\underline{B}}}$$

$\underline{\underline{B}}$ is hermitian if $\underline{\underline{B}} = \underline{\underline{B}}^*$ then prove $\underline{\underline{A}}^* \underline{\underline{A}} = (\underline{\underline{A}}^* \underline{\underline{A}})^*$

LHS: $\overline{\underline{\underline{A}}}^T \underline{\underline{A}}$

RHS: $\overline{\left(\overline{\underline{\underline{A}}}^T \underline{\underline{A}}\right)^T} = \overline{\left(\underline{\underline{A}}^T (\overline{\underline{\underline{A}}}^T)^T\right)} = \overline{\left(\underline{\underline{A}}^T \overline{\underline{\underline{A}}}\right)} = \overline{\underline{\underline{A}}}^T \underline{\underline{A}}$

LHS = RHS

$\boxed{\underline{\underline{B}} \text{ or } \underline{\underline{A}}^* \underline{\underline{A}} \text{ is a hermitian matrix}}$

b) $\underline{\underline{A}}^* \underline{\underline{A}}$ can be diagonalized by $\underline{\underline{U}}$ $\Rightarrow \underline{\underline{B}} = \underline{\underline{U}}\,\underline{\underline{D}}\,\underline{\underline{U}}^H$. How many positive eigenvalues does $\underline{\underline{A}}^* \underline{\underline{A}}$ have?

$\underline{\underline{A}}^* \underline{\underline{A}} = \underline{\underline{B}}$ $[N \times N]$ $\boxed{\text{can have } r \text{ positive eigenvalues}}$

bp2) ℓ = # eigenvalues $\sigma_1^2 \geq \sigma_2^2 \geq \dots \geq \sigma_\ell^2 > 0$ k^{th} column of $\underline{\underline{V}}$ unitary matrix is $\underline{V_k}$

find $\|\underline{V_k}\|_2^2$ and why?

$\|V_k\|_2^2 = \underline{V_k}^H \underline{V_k} = 1$ because $\underline{V_k}$ is a column of a unitary matrix w/ property $\underline{\underline{U}}\,\underline{\underline{U}}^H = \underline{\underline{I}}$ which mean that $\underline{V_k}^H \underline{V_k}$ will correspond to a diagonal element of $\underline{\underline{I}}$

$\boxed{\|\underline{V_k}\|_2^2 = 1}$

c) Consider $\underline{u}_k = \frac{1}{\sigma_k} \underline{\underline{A}} \underline{v}_k$ for $k \in 1, 2, \ldots l$, Show that $\|\underline{u}_k\|_2^2 = 1$

$$\|\underline{u}_k\|_2^2 = \underline{u}_k^H \underline{u}_k = \left[\frac{1}{\sigma_k} \underline{v}_k^T \underline{\underline{A}}^T\right]\left[\frac{1}{\sigma_k} \underline{\underline{A}} \underline{v}_k\right] = \frac{1}{\sigma_k^2} \underline{v}_k^H \overset{\underline{\underline{B}}}{\boxed{\underline{\underline{A}}^H \underline{\underline{A}}}} \underline{v}_k = \frac{1}{\sigma_k^2} \underline{v}_k^H \underline{\underline{B}} \underline{v}_k$$

due to diagonalizable by unitary matrices of $\underline{\underline{B}}$

$\underline{v}_k$ is column vector of $\underline{\underline{V}}$ & $\underline{\underline{D}}$ is diagonal matrix containg $\sigma_1^2 \ldots \sigma_k^2$

then $\underline{v}_k^H \underline{\underline{B}} \underline{v}_k = D_{kk}$ & $D_{kk} = \sigma_k^2$

$[1 \times N][N \times N][N \times 1] = [1 \times 1]$

then $\|\underline{u}_k\|_2^2 = \frac{1}{\sigma_k^2}(\sigma_k^2) = 1$ $\boxed{\|\underline{u}_k\|_2^2 = 1 \text{ ☺}}$

d) Show that $\underline{u}_k$ is an eigenvector of $\underline{\underline{A}}\underline{\underline{A}}^*$ with eigenvalue σ_k^2 ?

Eigenvalue & Eigenvector has a property $\underline{\underline{B}} \underline{c} = \lambda \underline{c}$

plugging in $\underline{\underline{B}} = \underline{\underline{A}} \underline{\underline{A}}^*$, $\underline{c} = \underline{u}_k$ & $\lambda = \sigma_k^2$

$$\underline{\underline{A}} \underline{\underline{A}}^* \underline{u}_k = \sigma_k^2 \underline{u}_k$$

RHS: $\sigma_k^2 \left[\frac{1}{\sigma_k} \underline{\underline{A}} \underline{v}_k\right] = \sigma_k \underline{\underline{A}} \underline{v}_k$ LHC: $\underline{\underline{A}} \underline{\underline{A}}^* \frac{1}{\sigma_k} \underline{\underline{A}} \underline{v}_k$

$\frac{1}{\sigma_k} \underline{\underline{A}} \underline{\underline{A}}^* \underline{\underline{A}} \underline{v}_k = \sigma_k \underline{\underline{A}} \underline{v}_k$ True if $\underline{v}_k$ is eigenvector of A^*A w/ eigenvalue σ_k^2 which is definition of $\underline{\underline{V}}$ & $\underline{v}_k$

Same

$$\frac{1}{\sigma_k} \underline{\underline{A}} (\sigma_k^2 \underline{v}_k) = \sigma_k \underline{\underline{A}} \underline{v}_k$$

$$\boxed{\underline{u}_k \text{ is eigenvector of } \underline{\underline{A}}\underline{\underline{A}}^* \text{ w/ eigenvalue} = \sigma_k^2}$$

e) let $\underline{\underline{V}}_\ell$ $[N \times \ell]$ matrix whose k^{th} column is $\underline{v}_k$ above, & $\underline{\underline{U}}_\ell$ $[M \times \ell]$ where k^{th} column is u_k above. Let Σ_ℓ be the diagonal matrix w/ diagonal entry σ_k Show that $\underline{\underline{A}}\,\underline{\underline{V}}_\ell = \underline{\underline{U}}_\ell \underline{\underline{\Sigma}}_\ell$

$\underline{\underline{A}}\,\underline{\underline{V}}_\ell = \underline{\underline{U}}_\ell \underline{\underline{\Sigma}}_\ell$

$\times N][N \times L) = [M \times L][L \times L]$

$[M \times L] = [M \times L]$ ✓✓

Take k^{th} column of $\underline{\underline{V}}_\ell$ we have $\underline{v}_k$ & from previous sections we know that

$\underline{u}_k = \frac{1}{\sigma_k} \underline{\underline{A}}\,\underline{v}_k$

$\underline{\underline{A}}\,\underline{v}_k = \underline{u}_k \sigma_k$

Concatenating all columns we get that

$$\boxed{\underline{\underline{A}}\,\underline{\underline{V}}_\ell = \underline{\underline{U}}_\ell \underline{\underline{\Sigma}}_\ell}$$ ✓✓

---

f) Now Show that $\underline{\underline{\Sigma}}_\ell = \underline{\underline{U}}^* \underline{\underline{A}}\,\underline{\underline{V}}_\ell$

from above we know that $\underline{\underline{A}}\,\underline{\underline{V}}_\ell = \underline{\underline{U}}_\ell \underline{\underline{\Sigma}}_\ell$ then plug in becomes

$\underline{\Sigma}_\ell = \underline{\underline{U}}_\ell^* \underline{\underline{U}}\,\underline{\underline{\Sigma}}_\ell$ from part c the proof implied that $\underline{\underline{U}}^*_\ell \underline{\underline{U}} = \underline{\underline{I}}$

then

$\underline{\Sigma}_\ell = \underline{\underline{I}}\,\underline{\underline{\Sigma}}_\ell$

$$\boxed{\underline{\Sigma}_\ell = \underline{\Sigma}_\ell}$$

Justine John "JJ" A. Serdoncillo **EE 5561: Image Processing**
October 16, 2023

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.

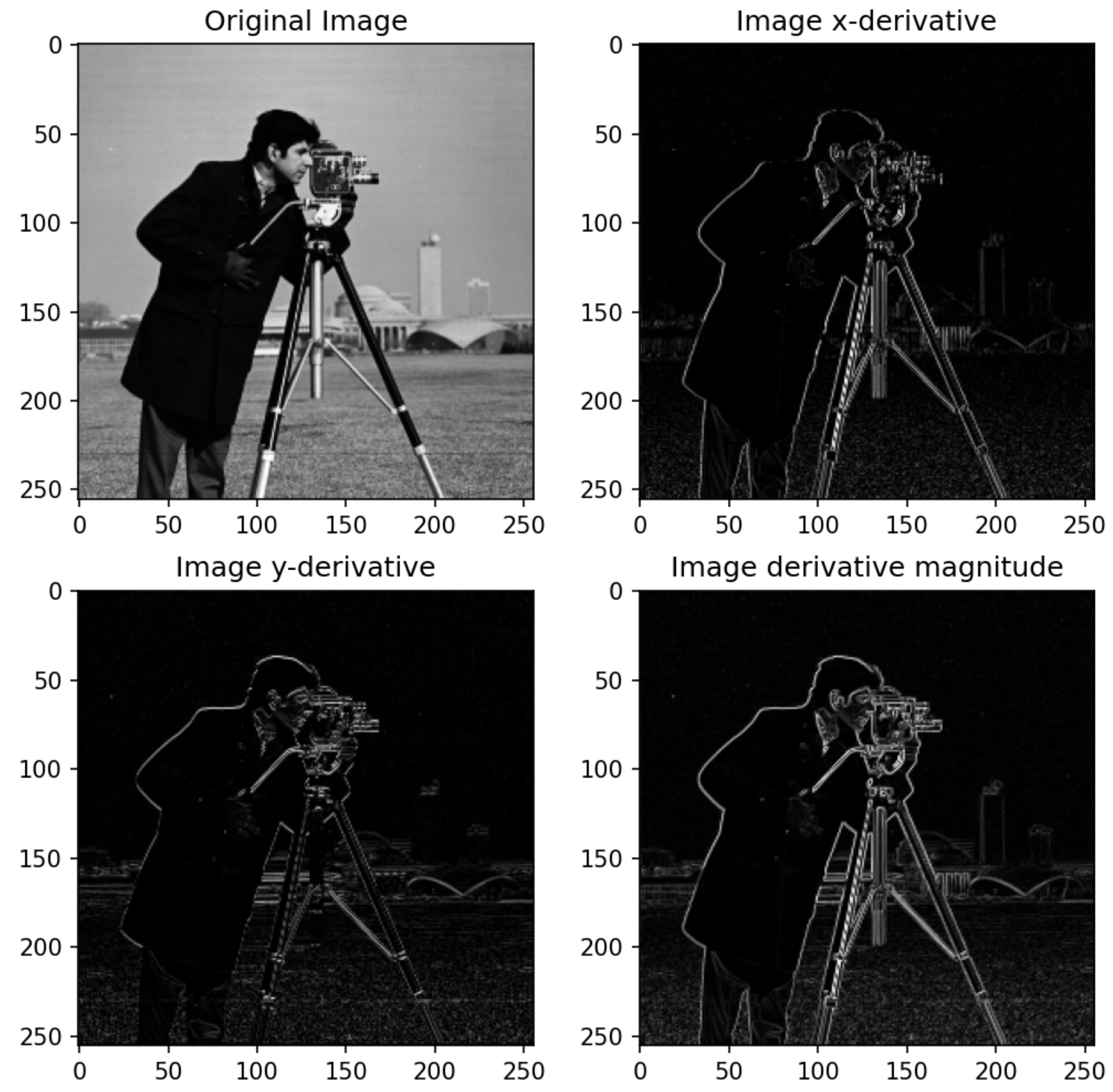

Figure 1: Image Derivative in x,y and magnitude

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)

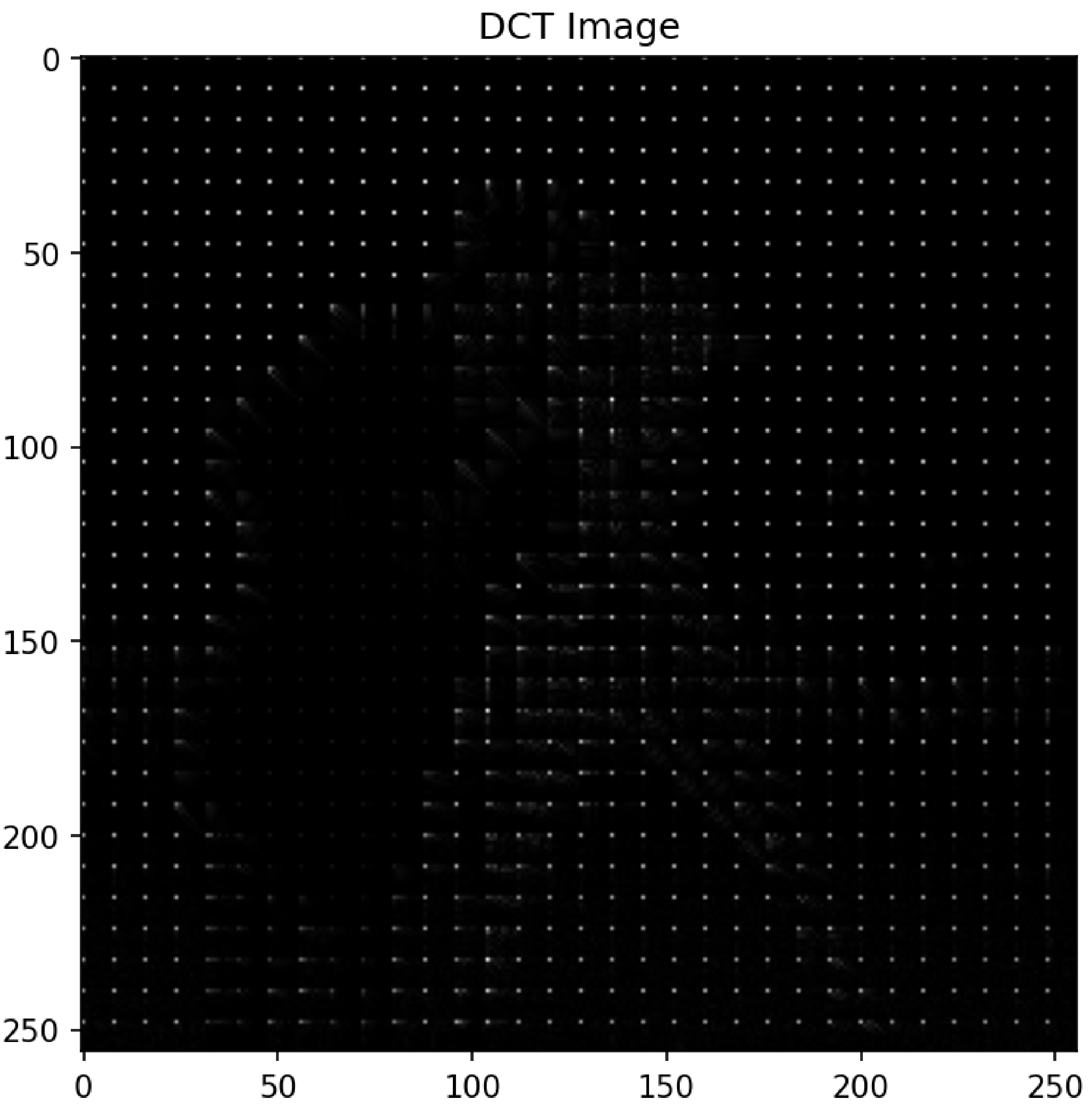

Figure 2: DCT on each 8x8 block

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.

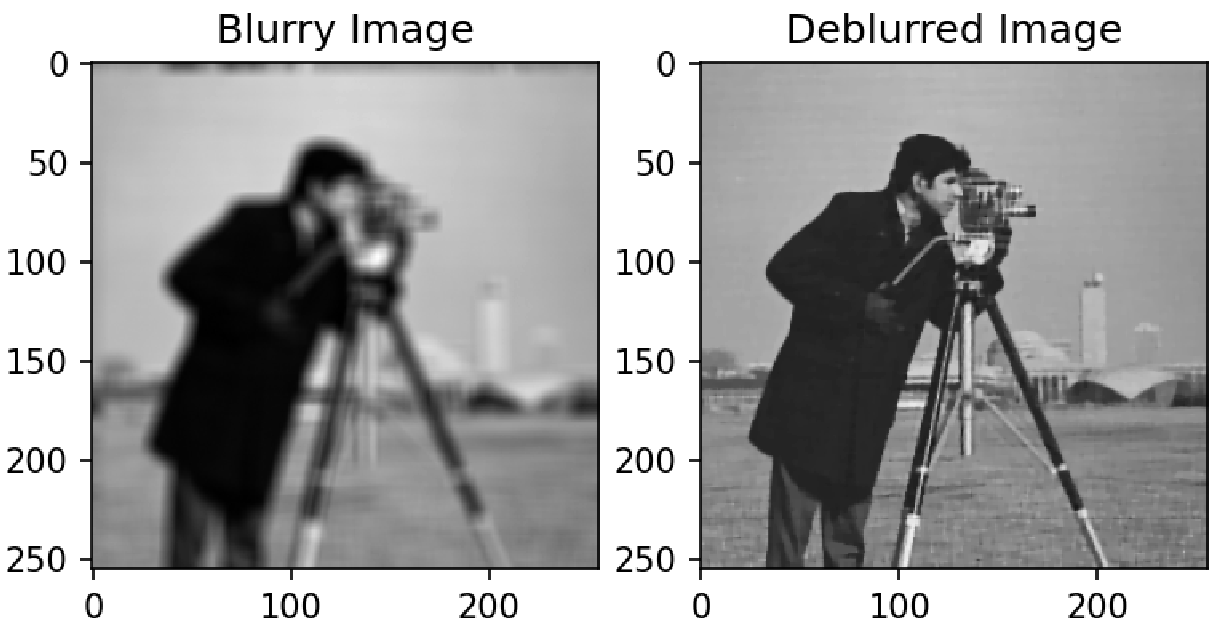

Figure 3: Denoising using PGD

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.

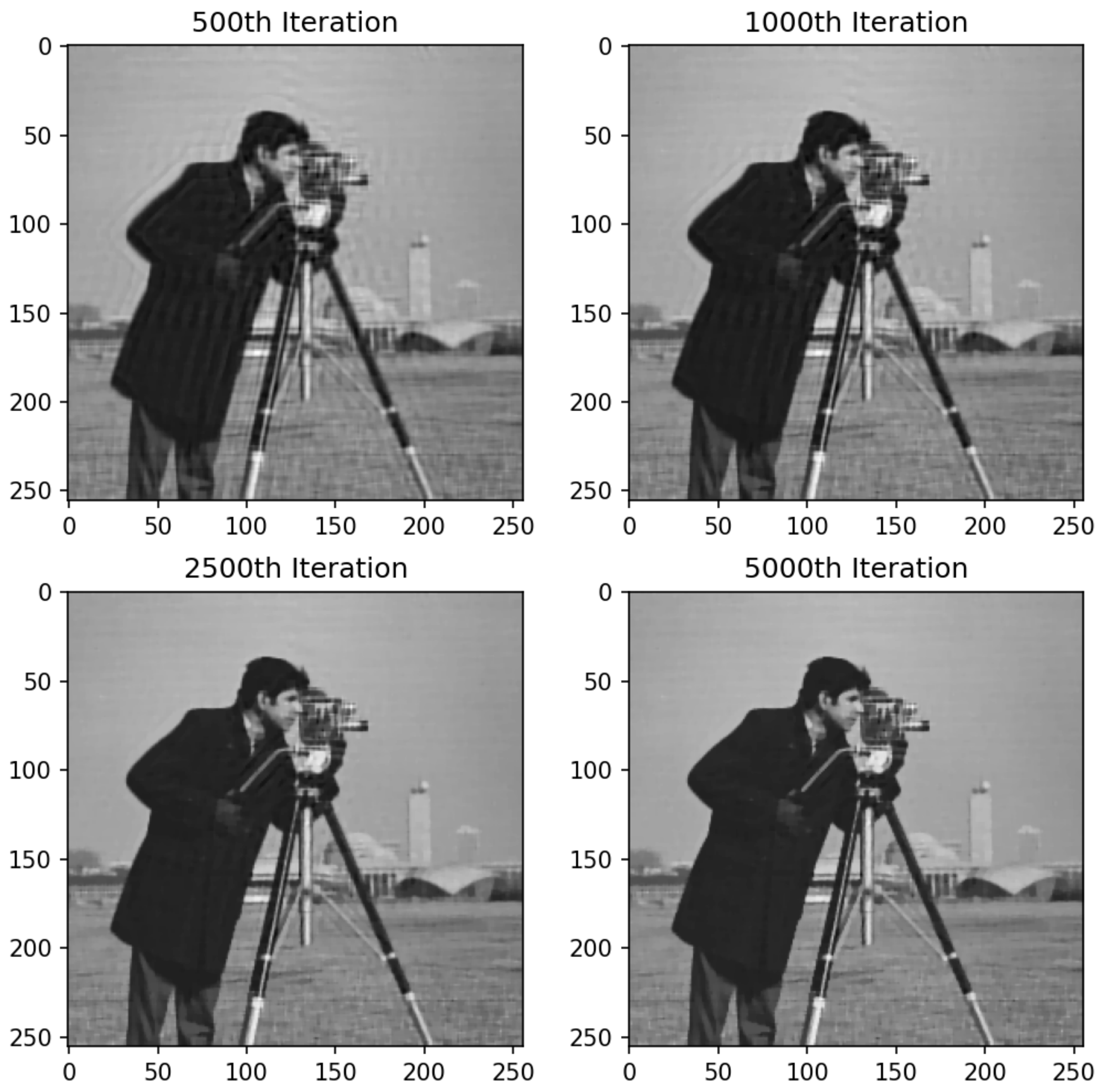

Figure 4: Intermediate Values of PGD Denoising

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.

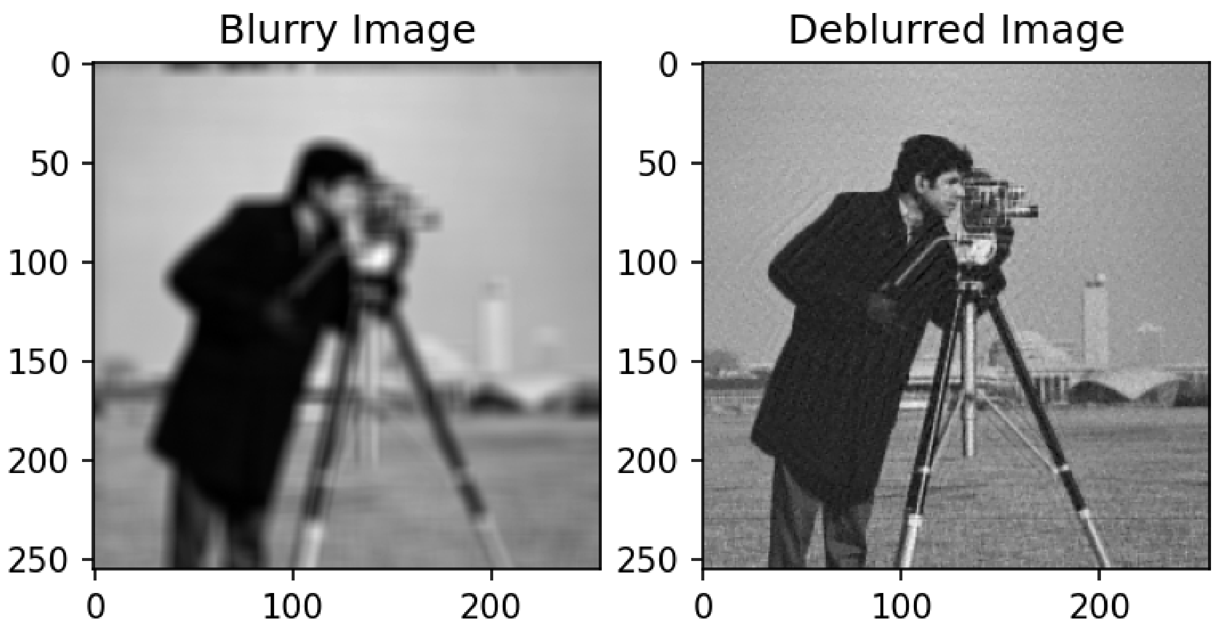

Figure 5: Denoising using ADMM

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.

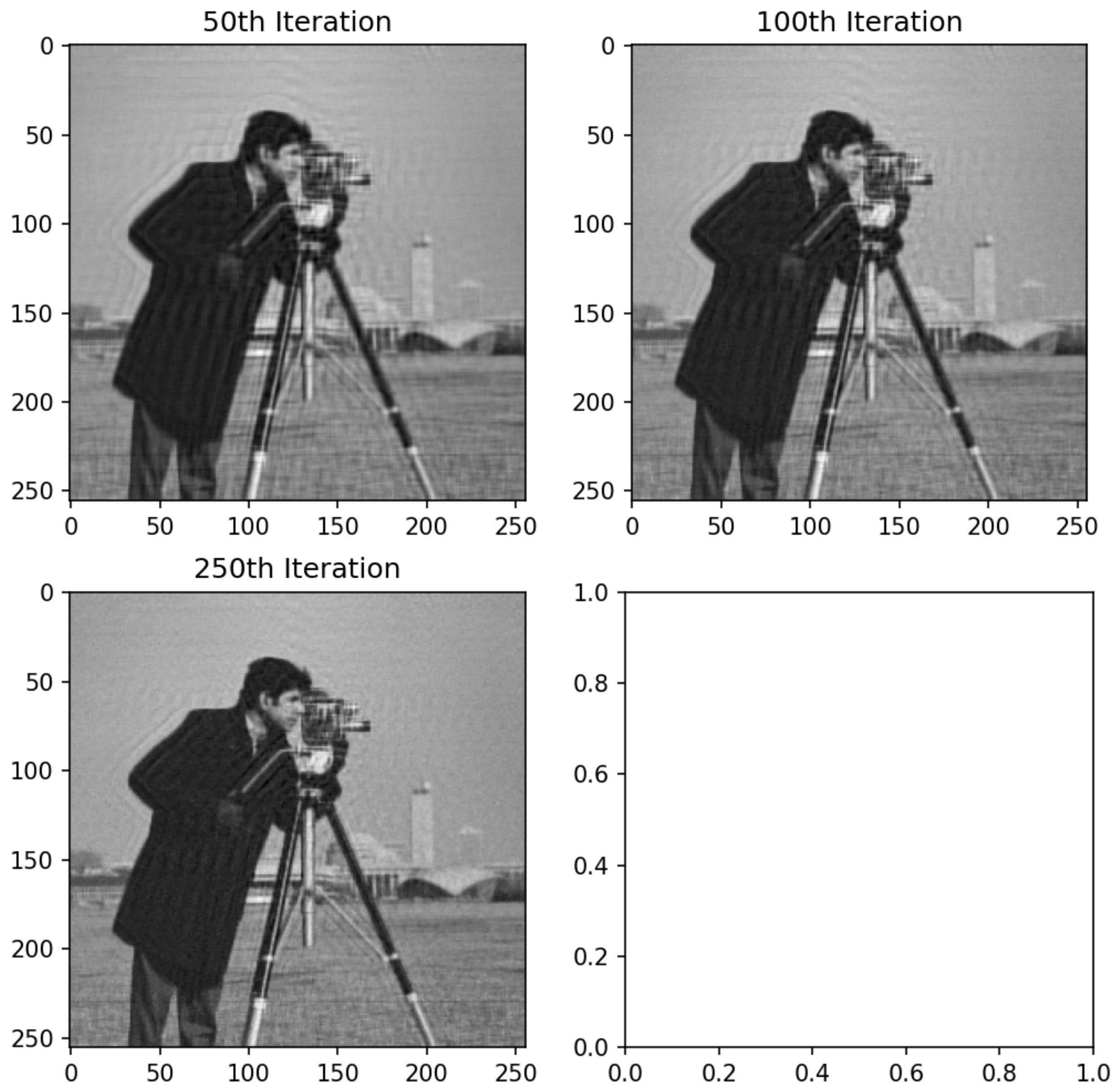

Figure 6: Intermediate Values of ADMM Denoising

Appendix

Problem 2

```python

'''
Justine Serdoncillo
EE 5561 - Image Processing
Problem Set 3
November 7, 2023
'''

# %% Problem Statement
"""
    2) [9 pts] Programming Exercise 1: In this exercise, you will familiarize yourself with
    function handles. Write function handles to calculate the following:
    a) The image derivative in x direction (the built-in function numpy.diff(Python)/diff(MATLAB)
    may be useful),
    b) the image derivative in y direction,
    c) the magnitude of the gradient vector in (x, y) directions,
    d) discrete cosine transform (you may use the scipy.fftpack.dct(Python)/dct2(MATLAB)
    function for this) of each 8    8 distinct block in the image.
    For a, b, and c make the derivative operator circulant. Thus, the output should be the same
    size as the image. Verify all 4 function handles on the cameraman image.
"""
# %%

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as img
import scipy
import imageio as iio
from scipy.fftpack import dct, idct
from scipy.signal import convolve2d

# %% Function Handles
def gradX(image):
    shift = np.roll(image, 1, axis=1)
    x = image - shift
    return x

def gradY(image):
    shift = np.roll(image, 1, axis=0)
    y = image - shift
    return y

def gradMag(image):
    x = gradX(image)
    y = gradY(image)
    mag = np.sqrt(x**2 + y**2)
    return mag

def dct2c(img):
    return dct(dct(img, axis=0, norm = 'ortho'), axis=1, norm = 'ortho')

def DCT(image, size):
    [Nx,Ny] = image.shape
    dct = np.zeros_like(image)
    for x in range(0,Nx,size):
        for y in range(0,Ny,size):
            dct[x:x+8,y:y+8] = dct2c(image[x:x+8,y:y+8])
    return dct


# %% Importing Images
man_img  = np.complex64(iio.imread('cameraman.tif'))

# %% Problem 2
def prob2():
    print(f"Cameraman has size {man_img.shape}")
    x = gradX(man_img)
    y = gradY(man_img)
    M = gradMag(man_img)
    D = DCT(man_img, 8)
    print(f"Derivatives has size {x.shape}")

    # Plot the figures
    fig, ax = plt.subplots(2,2, figsize=(8,8), dpi=150)
    ax[0,0].imshow(np.abs(man_img), cmap="gray")
    ax[0,0].set_title("Original Image")
    ax[0,1].imshow(np.abs(x), cmap="gray")
    ax[0,1].set_title("Image x-derivative")
    ax[1,0].imshow(np.abs(y), cmap="gray")
    ax[1,0].set_title("Image y-derivative")
    ax[1,1].imshow(np.abs(M), cmap="gray")
    ax[1,1].set_title("Image derivative magnitude")

    fig, ax = plt.subplots(figsize=(6,6), dpi=150)
    ax.imshow(np.abs(D), cmap="gray")
    ax.set_title("DCT Image")

# %% Main Function
if __name__ == "__main__":
    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
ax[1,1].imshow(saves[3], cmap="gray")
ax[1,1].set_title("5000th Iteration")

# %% Here, implement ADMM
# JJ's code starts here
z = np.zeros_like(y)
u = np.zeros_like(y)
aTV = 1e-2
rho = 0.1
no_iter=20
save = [50, 100, 250]
saves = []
x = np.copy(z)
for ind in range(500):
    b = HT(y) + rho*(z-u)

    # cg_solve equivalent
    x0 = np.zeros_like(b)
    xcg = np.copy(x0)
    r = b - A(xcg)
    p = np.copy(r)
    rsold = np.sum(r * np.conj(r))
    for i in range(no_iter):
        Ap = A(p)
        alpha = rsold/np.sum(p * Ap)
        xcg = xcg + alpha * p
        r = r - alpha * Ap
        rsnew = np.sum(r * np.conj(r))
        p = r + rsnew / rsold * p
        rsold = np.copy(rsnew)
    x = xcg.copy()

    z = TV_denoise(x + u, weight=aTV)
    u += x - z
    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("50th Iteration")
ax[0,1].imshow(saves[1], cmap="gray")
ax[0,1].set_title("100th Iteration")
ax[1,0].imshow(saves[2], cmap="gray")
ax[1,0].set_title("250th Iteration")
```