2. [Image Denoising]

Please download '*hw3_prob2.mat*' that includes **a noisy phantom image *f*** to be de-noised.

(a) Implement a denoising algorithm by **ADMM algorithm** with the following objective function:

$$\underset{x}{\text{minimize}} \;\; f(x) = g(x) + h_h(x) + h_v(x) = \frac{\mu}{2}\|x - f\|_2^2 + \|D_h x\|_1 + \|D_v x\|_1$$

$$\Rightarrow \underset{x, d_h, d_v, b_h, b_v}{\text{minimize}} \;\; \frac{\mu}{2}\|x - f\|_2^2 + |d_h| + |d_v| + \frac{\lambda}{2}\|d_h - D_h x - q_h\|_2^2 + \frac{\lambda}{2}\|d_v - D_v x - q_v\|_2^2$$

Please show the reconstructed image (μ=0.02, λ=0.0002, Stopping criterion: $\|x_k - x_{k-1}\|_2/\|x_k\|_2 < 10^{-4}$).
(20 pts)

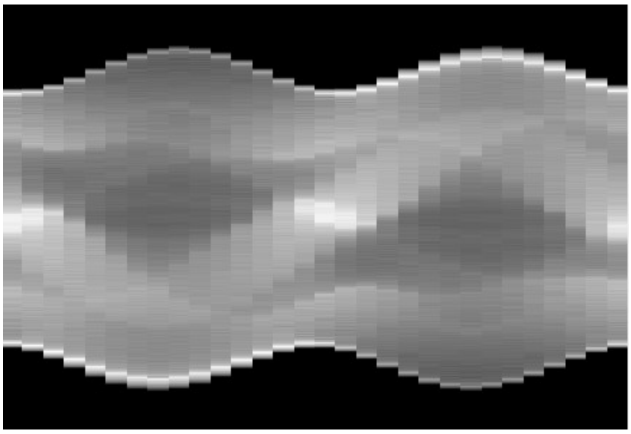
(b) From the result of part (a), the image de-noising can be improved by the reweighted L1-norm as follows:

$$\underset{x}{\text{minimize}} \;\; f(x) = g(x) + h_h(x) + h_v(x) = \frac{\mu}{2}\|x - f\|_2^2 + \|W_h(D_h x)\|_1 + \|W_v(D_v x)\|_1$$

You could conduct the image reconstruction by the reweighting process with appropriate hyper-parameter setting, in which you would have to define μ, λ and δ. Please show the reconstructed image with the hyper-parameters clearly that you found.
(20 pts)

3. [CT Image Reconstruction from Few Projections (Compressed Sensing)]

From the given sinogram that has very few projection angles (30 angles), as shown below, the CT image reconstruction needs to be performed by **Chambolle-Pock algorithm**.



The objective function is defined as follow:

$$\underset{x}{\text{minimize}} \;\; F(Kx) + G(x) = \frac{\mu}{2}\|Ax - b\|_2^2 + \|D_h x\|_1 + \|D_v x\|_1 \; (x \geq 0)$$

Please download '*hw3_prob3.mat*', and implement the Chambolle-Pock algorithm. Show the reconstructed image with the following parameters:

(μ=1, Stopping criterion: $\|x_k - x_{k-1}\|_2/\|x_k\|_2 < 5 \times 10^{-4}$)

(20 pts)

hw3-prob2-a

February 7, 2024

[1]:
```python
#
import numpy as np
import scipy.io
import matplotlib.pyplot as plt
```

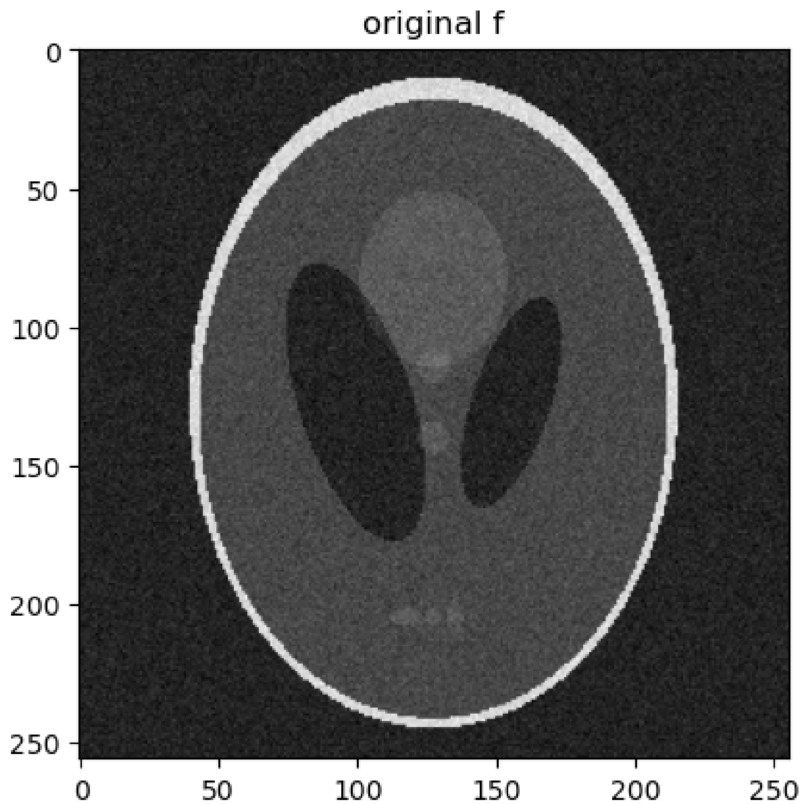
[2]:
```python
# MAT
mat_path = r"../HW3_package/hw3_prob2.mat"
mat_data = scipy.io.loadmat(mat_path)

mu = 0.02
lmbd = 0.0002

f = mat_data['f']
x_orig = mat_data['x_orig']
```

[3]:
```python
def Dh(u):
    rows, cols = u.shape
    d = np.zeros((rows, cols))
    d[:, 1:cols] = u[:, 1:cols] - u[:, 0:cols-1]
    d[:, 0] = u[:, 0] - u[:, cols-1]
    return d

def Dht(u):
    rows, cols = u.shape
    d = np.zeros((rows, cols))
    d[:, 0:cols-1] = u[:, 0:cols-1] - u[:, 1:cols]
    d[:, cols-1] = u[:, cols-1] - u[:, 0]
    return d

def Dv(u):
    rows, cols = u.shape
    d = np.zeros((rows, cols))
    d[1:rows, :] = u[1:rows, :] - u[0:rows-1, :]
    d[0, :] = u[0, :] - u[rows-1, :]
    return d

def Dvt(u):
```

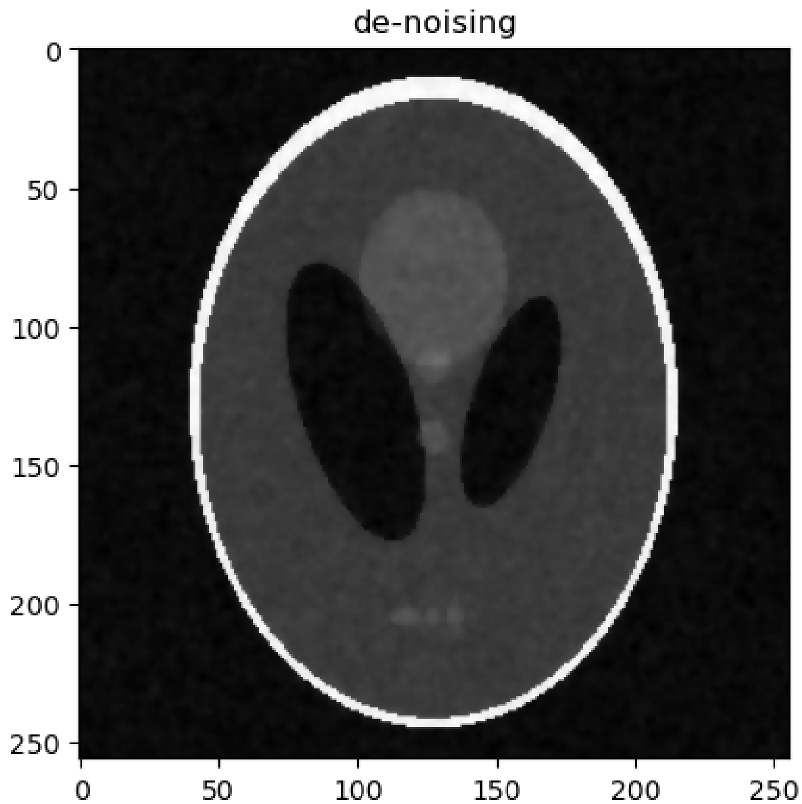
```
    rows, cols = u.shape
    d = np.zeros((rows, cols))
    d[0:rows-1, :] = u[0:rows-1, :] - u[1:rows, :]
    d[rows-1, :] = u[rows-1, :] - u[0, :]
    return d
```

[4]:
```
# 1
x = np.zeros(f.shape)

stopping_point = 1e-4
sigma = 200
lambda_v = 2
result = []

x = np.zeros(f.shape)

dh = np.zeros(f.shape)
dv = np.zeros(f.shape)
qh = np.zeros(f.shape)
qv = np.zeros(f.shape)


for iter in range(100000):
    x_minus1 = x

    # minimize x
    x = ((mu * f) / (mu + 4 * lmbd)) + ((lmbd / (mu + 4 * lmbd)) * (np.roll(x,␣
 ↪1, axis = 0) + np.roll(x, -1, axis = 0) + np.roll(x, 1, axis = 1) + np.
 ↪roll(x, -1, axis = 1)
                                                                    + Dht(dh␣
 ↪- qh) + Dvt(dv - qv)))

    # minimize dv, dh
    dh =  np.sign(Dh(x) + qh) * np.maximum(np.abs(Dh(x) + qh) - (1 / lmbd), 0)
    dv =  np.sign(Dv(x) + qv) * np.maximum(np.abs(Dv(x) + qv) - (1 / lmbd), 0)

    # minimize qv, qh
    qh = qh + (Dh(x) - dh)
    qv = qv + (Dv(x) - dv)


    # break check
    if  (np.linalg.norm((x - x_minus1), 2) / np.linalg.norm(x, 2)) <␣
 ↪stopping_point and iter > 1:
        break
```

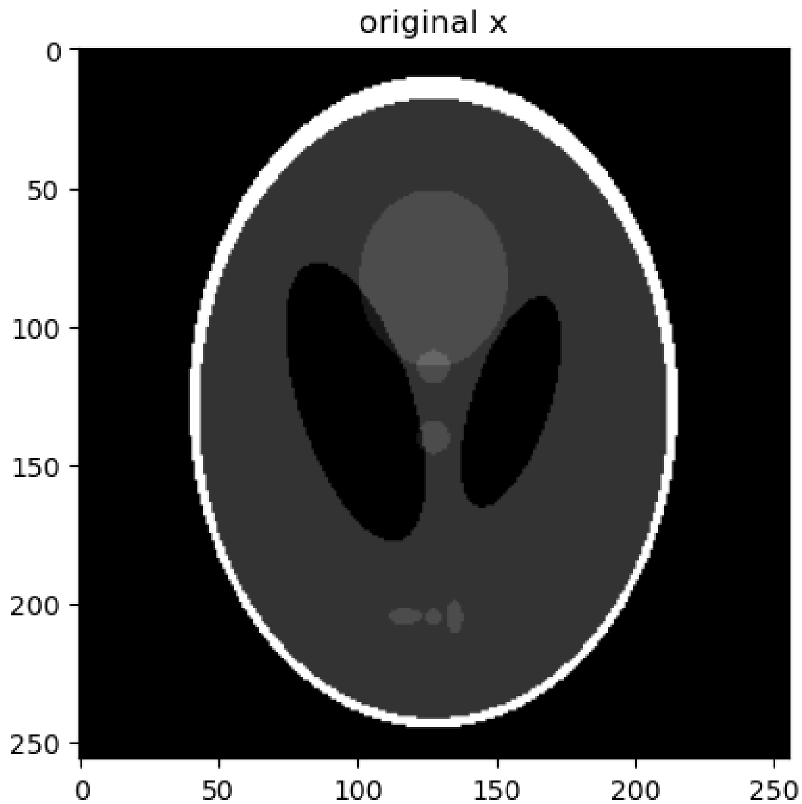
[5]:
```
# original f
plt.imshow(f, cmap = "gray")
plt.title( label = "original f")
plt.show()
```



[6]:
```
# original f
plt.imshow(x, cmap = "gray")
plt.title( label = "de-noising")
plt.show()
```



[7]:
```
# original x
plt.imshow(x_orig, cmap = "gray")
plt.title( label = "original x")
plt.show()
```


original x
0
50
100
150
200
250
0
50
100
150
200
250

```
[ ]:
```

```
[ ]:
```

hw3-prob3

February 7, 2024

[1]:
```
#
import numpy as np
import scipy.io
import matplotlib.pyplot as plt
```

[2]:
```
# MAT
mat_path = r"../HW3_package/hw3_prob3.mat"
mat_data = scipy.io.loadmat(mat_path)

A = mat_data['A']
b = mat_data['b']
x_orig = mat_data['x_orig']
```

[3]:
```
def normest_numpy(A):
    """
    NumPy       A  2-        .
    """
    # SVD     . full_matrices=False        SVD    .
    U, s, V = scipy.sparse.linalg.svds(A)
    #
    return s[0]

def Dh(u):
    rows, cols = u.shape
    d = np.zeros((rows, cols))
    d[:, 1:cols] = u[:, 1:cols] - u[:, 0:cols-1]
    d[:, 0] = u[:, 0] - u[:, cols-1]
    return d

def Dht(u):
    rows, cols = u.shape
    d = np.zeros((rows, cols))
    d[:, 0:cols-1] = u[:, 0:cols-1] - u[:, 1:cols]
    d[:, cols-1] = u[:, cols-1] - u[:, 0]
    return d

def Dv(u):
```

```python
    rows, cols = u.shape
    d = np.zeros((rows, cols))
    d[1:rows, :] = u[1:rows, :] - u[0:rows-1, :]
    d[0, :] = u[0, :] - u[rows-1, :]
    return d

def Dvt(u):
    rows, cols = u.shape
    d = np.zeros((rows, cols))
    d[0:rows-1, :] = u[0:rows-1, :] - u[1:rows, :]
    d[rows-1, :] = u[rows-1, :] - u[0, :]
    return d
```

[4]:

```python
mu = 1
stopping_point = 2e-4

# # normest     tau  sigma
norm_est = normest_numpy(A)
tau = 1 / norm_est**2
sigma = 1 / (tau * norm_est**2)

theta = 0.01
```

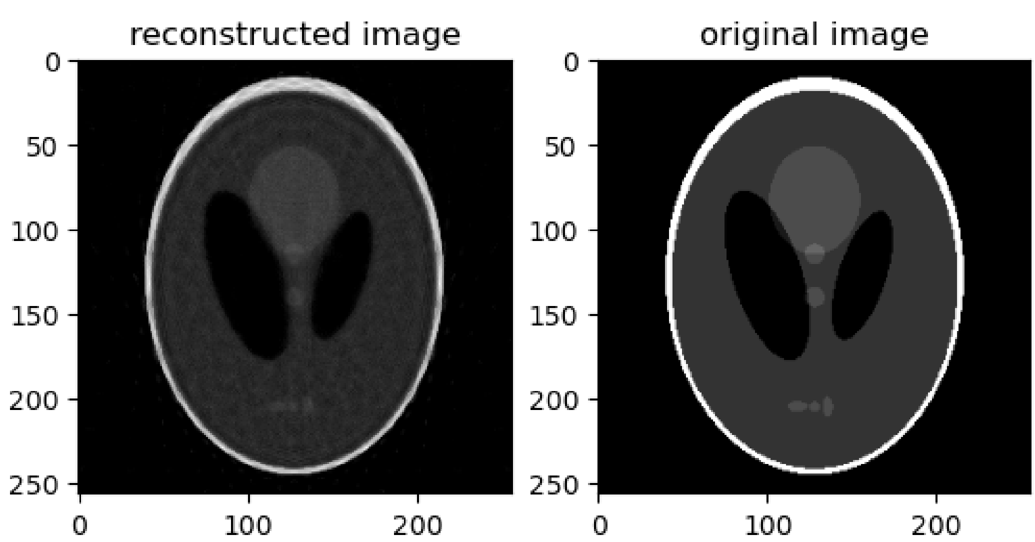
[5]:

```python
y1 = np.zeros(b.shape)
y2 = np.zeros((256,256))
y3 = np.zeros((256,256))
x = np.zeros((256,256))

for idx in range(10000):

    y2 = Dh(x)
    y3 = Dv(x)

    # update y
    y2 = np.sign(y2 + sigma * Dh(x)) * np.minimum(np.abs(y2 + sigma * Dh(x)), sigma)
    y3 = np.sign(y3 + sigma * Dv(x)) * np.minimum(np.abs(y3 + sigma * Dv(x)), sigma)

    x = x.reshape(65536, 1)
    y1 = ((y1 + sigma * A @ x) - (sigma * mu * b)) / (1 + sigma * mu)

    # update x
    x_minus1 = x

    y2 = Dht(y2)
    y3 = Dvt(y3)
```

```
    y2 = y2.reshape(65536, 1)
    y3 = y3.reshape(65536, 1)

    x = x - (tau * (A.T @ y1 + y2 + y3))
    x[x<0] = 0


    x = x + theta * (x - x_minus1)

    x = x.reshape(256,256)
    y2 = y2.reshape(256,256)
    y3 = y3.reshape(256,256)
    x_minus1 = x_minus1.reshape(256,256)

    if (np.linalg.norm((x - x_minus1), 2) / np.linalg.norm(x, 2)) <␣
↪stopping_point:
        break
```

[6]:

```
# original f
x = x.reshape((256, 256))
plt.subplot(121)
plt.imshow(x.T, cmap = "gray")
plt.title( label = "reconstructed image")


x = x.reshape((256, 256))
plt.subplot(122)
plt.imshow(x_orig, cmap = "gray")
plt.title( label = "original image")

plt.show()
```



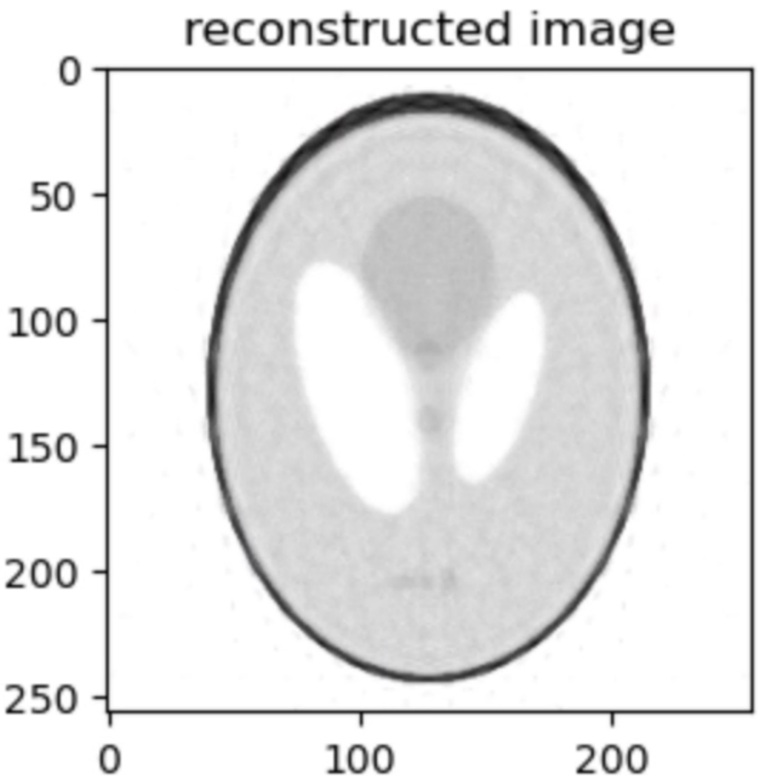
질문

Chambolle-Pock 알고리즘을 이용하여 문제를 풀 때, y_1^{n+1}을 업데이트 하는 과정에서 문제가 생겼습니다.

$$y_1^{n+1} = (y^n + \sigma * Ax^n)/1 + \sigma\mu$$

인 수식을 이용하여 아래와 같이 코드로 구현한 뒤, 결과를 확인하니 아래와 같은 결과가 나왔습니다.

y1 = ((y1 + sigma * A @ x) + (sigma * mu * b)) / (1 + sigma * mu)



올바른 결과를 찾기 위해 여러가지 시도를 해보다가 아래와 같이 계산을 해주었습니다.

$$y_1^{n+1} = (y^n - \sigma * Ax^n)/1 + \sigma\mu$$

코드

y1 = ((y1 + sigma * A @ x) - (sigma * mu * b)) / (1 + sigma * mu)

y1을 업데이트 시켜줄 때, 아래와 같은 결과가 나와서 이 부분에 대해 명확히 이유를 알고자 합니다.