$$\min \quad F(Kx)+G(x)=\frac{\mu}{2}\|Ax-b\|_2^2+\|D_h x\|_0+\|D_v x\|_0$$

※ step1.

$$z_1^{n+1}=\mathrm{prox}_{\sigma^{-1}F}(\sigma^{-1}y_1^n+A\tilde{x}^n)=\frac{(\sigma^{-1}y_1^n+A\tilde{x}^n)+\mu b/\sigma}{1+(\mu/\sigma)}$$

$$z_2^{n+1}=\mathrm{prox}_{\sigma^{-1}F}(\sigma^{-1}y_2^n+D_h\tilde{x}^n)=\begin{cases}\sigma^{-1}y_2^n+D_h\tilde{x}^n & (|\sigma^{-1}y_2^n+D_h\tilde{x}^n|\geq\sigma^{-1})\\ 0 & (|\sigma^{-1}y_2^n+D_h\tilde{x}^n|<\sigma^{-1}).\end{cases}$$

$$z_3^{n+1}=\mathrm{prox}_{\sigma^{-1}F}(\sigma^{-1}y_3^n+D_v\tilde{x}^n)=\begin{cases}\sigma^{-1}y_3^n+D_v\tilde{x}^n & (|\sigma^{-1}y_3^n+D_v\tilde{x}^n|\geq\sigma^{-1})\\ 0 & (|\sigma^{-1}y_3^n+D_v\tilde{x}^n|<\sigma^{-1}).\end{cases}$$

※ step2.

$$y_1^{n+1}=y_1^n+\sigma(Ax^n-z_1^{n+1})$$

$$y_2^{n+1}=y_2^n+\sigma(D_hx^n-z_2^{n+1})$$

$$y_3^{n+1}=y_3^n+\sigma(D_vx^n-z_3^{n+1}).$$

※ step3.

$$x^{n+1}=\mathrm{prox}_{\tau G}(x^n-\tau\textstyle\sum_i K^T y_i^{n+1})$$

$$=\left[x^n-\tau(A^Ty_1^{n+1}+y_2^{n+1}+y_3^{n+1})\right]_+$$

$$\tilde{x}^{n+1}=x^{n+1}+\theta(x^{n+1}-x^n).$$

$\mu=1.$

$\tau\cdot\sigma\cdot\|A^2\|\leq 1$

※ normest → 행렬 A의 특이점이 가장 큰 값.

$$\tau=\frac{1}{(\text{normest})^2}$$

$$\sigma=\frac{1}{\tau\cdot(\text{normest})^2}.$$

$$\sigma^{-1}=\tau\cdot\text{normest}^2.$$

$$\theta=\frac{1}{\sqrt{(1+2\cdot\mu)}}$$

hw3-prob3-l0-norm

February 13, 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)
sigma_1 = (tau * norm_est**2)

theta = 1/np.sqrt(1 + 2 * mu)
```

[5]:
```python
y1 = np.zeros(b.shape)
y2 = np.zeros((256,256))
y3 = np.zeros((256,256))
x = np.zeros((256,256))

z1 = np.zeros(b.shape)
z2 = np.zeros((256,256))
z3 = np.zeros((256,256))

y2 = Dh(x)
y3 = Dv(x)
x = x.reshape(65536, 1)
y1 = A @ x
x = x.reshape(256,256)

for idx in range(10000):

    # step 1
    Dhx = Dh(x)
    Dvx = Dv(x)

    # step 2
```

```
    # HardThresholding
    z2 = sigma_1 * y2 + Dhx
    z3 = sigma_1 * y3 + Dvx
    z2[np.abs(sigma_1 * y2 + Dhx) < sigma_1] = 0
    z3[np.abs(sigma_1 * y3 + Dvx) < sigma_1] = 0


    x = x.reshape(65536, 1)
    z1 = ((sigma_1 * y1 + A @ x) + (mu * b/ sigma)) / (1 + mu / sigma)

    # step 3
    x_minus1 = x

    y1 = y1 + sigma * (A @ x - z1)
    y2 = y2 + sigma * (Dhx - z2)
    y3 = y3 + sigma * (Dvx - z3)


    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)

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

```
0 1.0
1 3.5274576204341277
2 1.3557814728326099
3 4.753474126239157
4 1.2435827896935654
5 3.817060833425024
6 0.9097140113056035
7 1.3939271807967724
8 0.5159256306848036
9 0.41495220115601467
10 0.18905458077982387
```

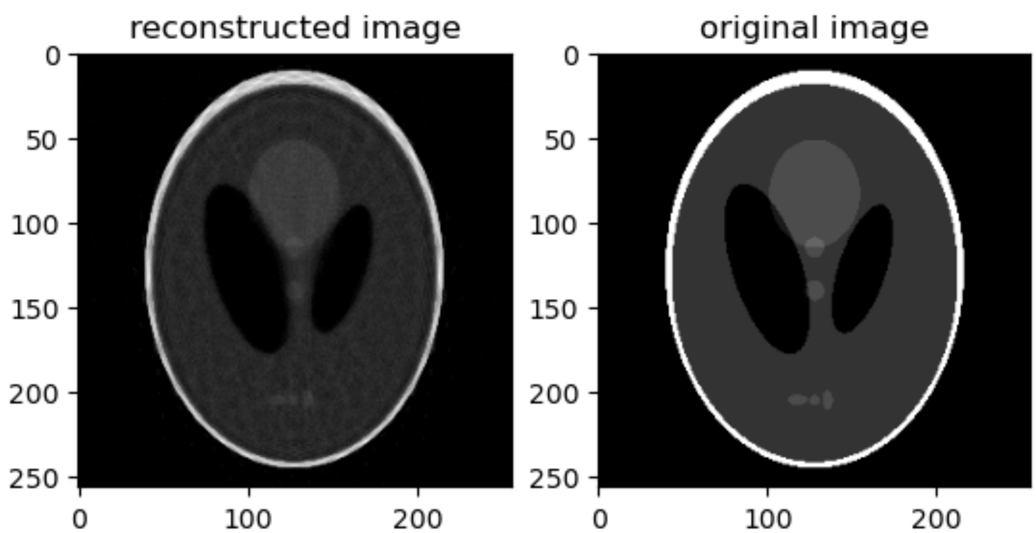
```
155 0.00021407338829309585
156 0.00021183643092429058
157 0.00020968856933822777
158 0.00020746446445694437
159 0.00020498247548997444
160 0.00020304495866980073
161 0.0002007890858422375
162 0.0001988301007671573
```

[6]:

```python
# 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()
```



[ ]:

```

```