libraries and functions

```python
import numpy as np
import h5py
import matplotlib.pyplot as plt
def cost(y,x):
    return (np.linalg.norm(np.absolute(y)- np.absolute(x) ))/(np.linalg.norm(x))

def posterior(y, x, meth,alpha, gamma=1):
    if alpha < 0 or alpha > 1:
        return
    m,n = y.shape
    likelihood = np.sum(np.square(np.absolute(y-x)))
    up = np.absolute(x-np.roll(x, [1,0], [0,1]))
    down = np.absolute(x-np.roll(x, [m-1,0], [0,1]))
    left = np.absolute(x-np.roll(x, [0,1], [0,1]))
    right = np.absolute(x-np.roll(x, [0,n-1], [0,1]))
    if meth=="quadratic":
        prior = np.sum(np.square(up) + np.square(down) + np.square(left) + np.squa
    elif meth=="huber":
        prior_up = np.multiply(np.less_equal(up,gamma) , up**2/2) + np.multiply(np
        prior_down = np.multiply(np.less_equal(down,gamma) , down**2/2) + np.multi
        prior_left = np.multiply(np.less_equal(left,gamma) , left**2/2) + np.multi
        prior_right = np.multiply(np.less_equal(right,gamma) , right**2/2) + np.mu
        prior = np.sum(prior_up + prior_down + prior_left + prior_right)
    elif meth=="log":
        prior_up = gamma*up - gamma**2*np.log(1+up/gamma)
        prior_down = gamma*down - gamma**2*np.log(1+down/gamma)
        prior_left = gamma*left - gamma**2*np.log(1+left/gamma)
        prior_right =gamma*right - gamma**2*np.log(1+right/gamma)
        prior = np.sum(prior_up + prior_down + prior_left + prior_right)

    return (1-alpha)*likelihood + alpha*prior

#based on dynamic step size
def gradiant(y,x,meth, alpha, gamma=1):
    if alpha < 0 or alpha > 1:
        return
    m,n = y.shape
    likelihood = 2*(y-x)
    up = x-np.roll(x, [1,0], [0,1])
    down = x-np.roll(x, [m-1,0], [0,1])
    left = x-np.roll(x, [0,1], [0,1])
    right = x-np.roll(x, [0,n-1], [0,1])

    if meth=="quadratic":
        prior = 2*(up+down+left+right)
    elif meth=="huber":
        prior_up = np.multiply(np.less_equal(np.absolute(up),gamma) , up) + np.mul
        prior_down =  np.multiply(np.less_equal(np.absolute(down),gamma) , down) +
        prior_left =  np.multiply(np.less_equal(np.absolute(left),gamma) , left) +
        prior_right = np.multiply(np.less_equal(np.absolute(right),gamma) , right)
        prior = prior_up + prior_down + prior_left + prior_right
    elif meth=="log":
        prior_up = np.multiply(gamma*up , np.reciprocal(gamma + np.absolute(up)))
        prior_down = np.multiply(gamma*down , np.reciprocal(gamma + np.absolute(do
        prior_left = np.multiply(gamma*left , np.reciprocal(gamma + np.absolute(le
        prior_right =np.multiply (gamma*right , np.reciprocal(gamma + np.absolute(
        prior = prior_up + prior_down + prior_left + prior_right

    return (1-alpha)*likelihood + alpha*prior

def routine(imgNoisy, alpha, gamma, step, thresh, meth):
    old_model = np.copy(imgNoisy)
    old_posterior = posterior(imgNoisy, old_model,meth, alpha)
    posterior_val = []
    posterior_val.append(posterior)
    if meth=="huber" or meth=="log":
        for i in range(30):
            gradiant_img = gradiant(imgNoisy,old_model,meth, alpha,gamma)
            new_model = old_model - step*gradiant_img
            new_posterior = posterior(imgNoisy, new_model,meth, alpha,gamma)

            if new_posterior < old_posterior:
                step  = 1.1*step
                old_model = new_model
                old_posterior = new_posterior

            else:
                step = 0.5*step
            posterior_val.append(old_posterior)

    else:
        while step > thresh:
            gradiant_img = gradiant(imgNoisy,old_model,meth, alpha)
            new_model = old_model - step*gradiant_img
            new_posterior = posterior(imgNoisy, new_model,meth, alpha)

            if new_posterior < old_posterior:
                step  = 1.1*step
                old_model = new_model
                old_posterior = new_posterior

            else:
                step = 0.5*step
            posterior_val.append(old_posterior)

    return posterior_val, new_model
```

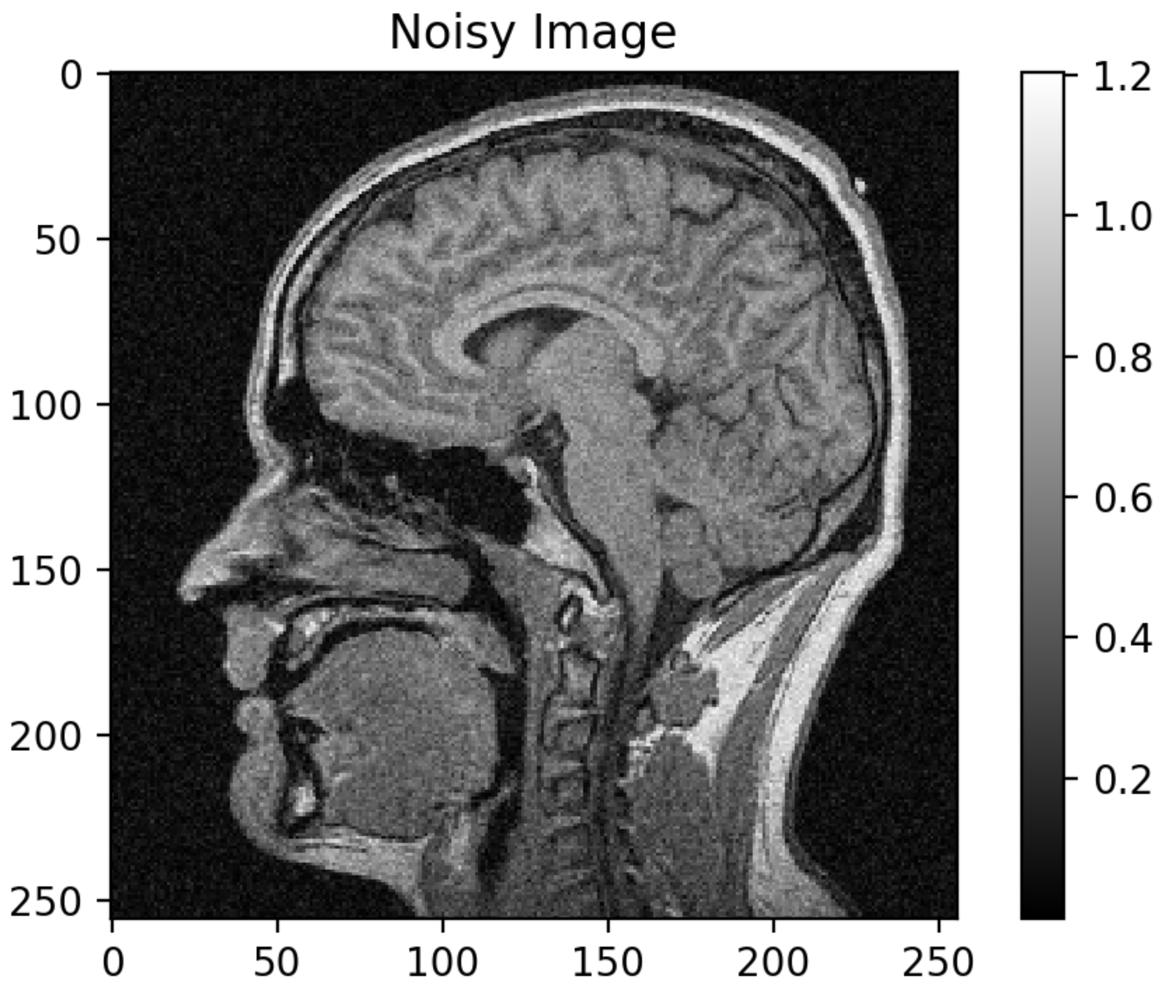
```
C:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:36:
FutureWarning: Conversion of the second argument of issubdtype from
`float` to `np.floating` is deprecated. In future, it will be treated
as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

Image Data

Loading [MathJax]/jax/input/LaTeX/config.js

```
f = h5py.File('../data/assignmentImageDenoisingBrainNoisy.mat','r')
imageNoisy = f.get('imageNoisy')
imageNoisy = np.array(imageNoisy)
imageNoisy_real = np.zeros((256,256))
imageNoisy_imag = np.zeros((256,256))

for i in range(256):
    for j in range(256):
        a = imageNoisy[i,j]
        imageNoisy_real[i,j] = a[0]
        imageNoisy_imag[i,j] = a[1]

imageNoisy = np.vectorize(complex)(imageNoisy_real, imageNoisy_imag)
imageNoisy = imageNoisy.T

plt.figure()
plt.imshow(np.absolute(imageNoisy),cmap='gray')
plt.title('Noisy Image')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x15f151cdac8>
```



Noise Level Estimation

```
bg1 = imageNoisy[0:120, 0:38]
bg2 = imageNoisy[165:255, 0:30]
bg3 = imageNoisy[0:40, 235:255]
bg4 = imageNoisy[190:238, 220:255]

bg1 = np.reshape(bg1, (bg1.shape[0]*bg1.shape[1],1))
bg2 = np.reshape(bg2, (bg2.shape[0]*bg2.shape[1],1))
bg3 = np.reshape(bg3, (bg3.shape[0]*bg3.shape[1],1))
bg4 = np.reshape(bg4, (bg4.shape[0]*bg4.shape[1],1))
noise = np.vstack((bg1,bg2,bg3,bg4))
std = np.std(np.real(noise))
print(std)
threshold = 1e-7
step = 1
```

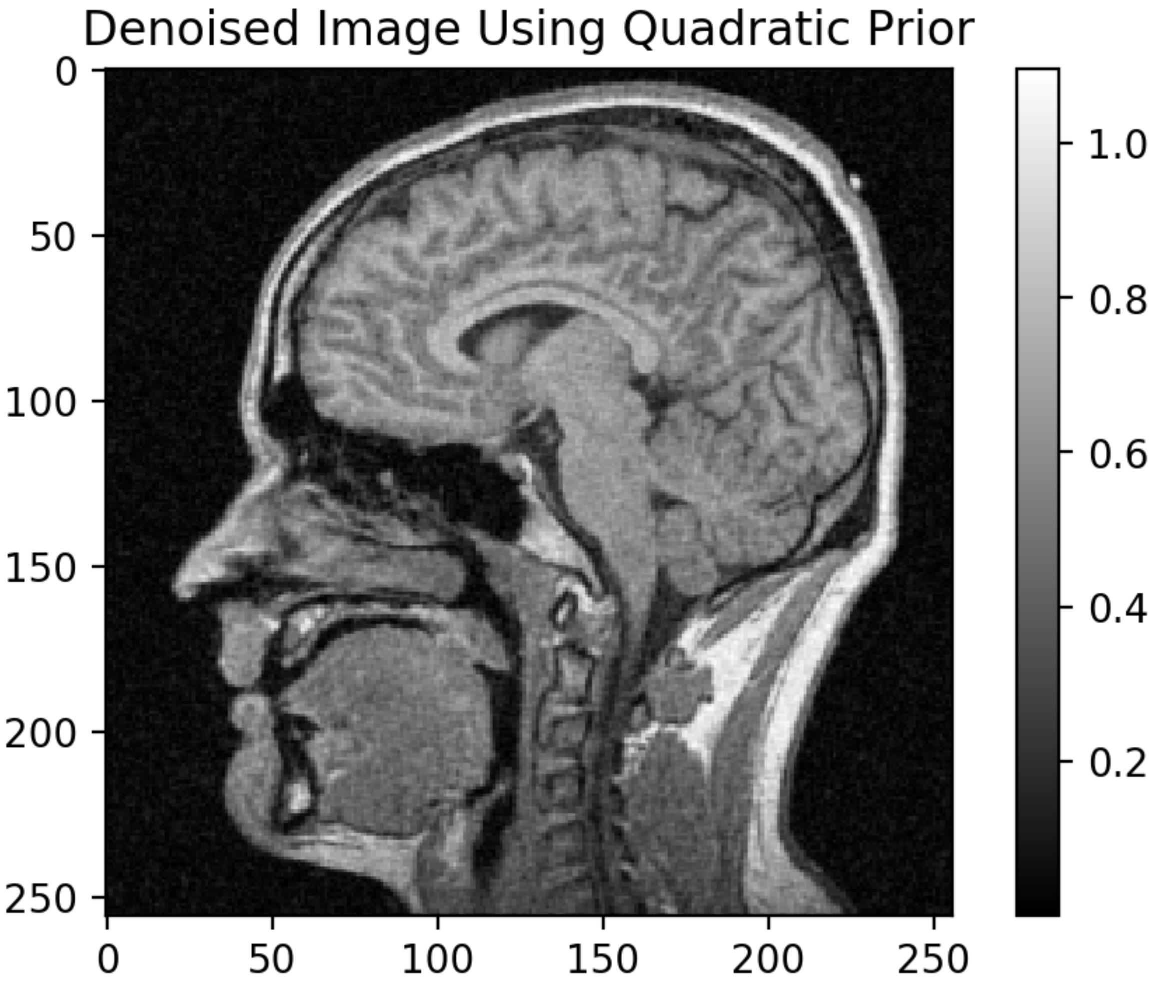
```
0.060921389237513904
```

Denoising Using Quadratic Prior

```
post_quad, denoised_model_quad = routine(imageNoisy, 0.125, 1, step, threshold, "q
post_quad = post_quad[1:]
plt.figure()
plt.imshow(np.absolute(denoised_model_quad),cmap='gray')
plt.title('Denoised Image Using Quadratic Prior')

plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x15f156f3cc0>
```

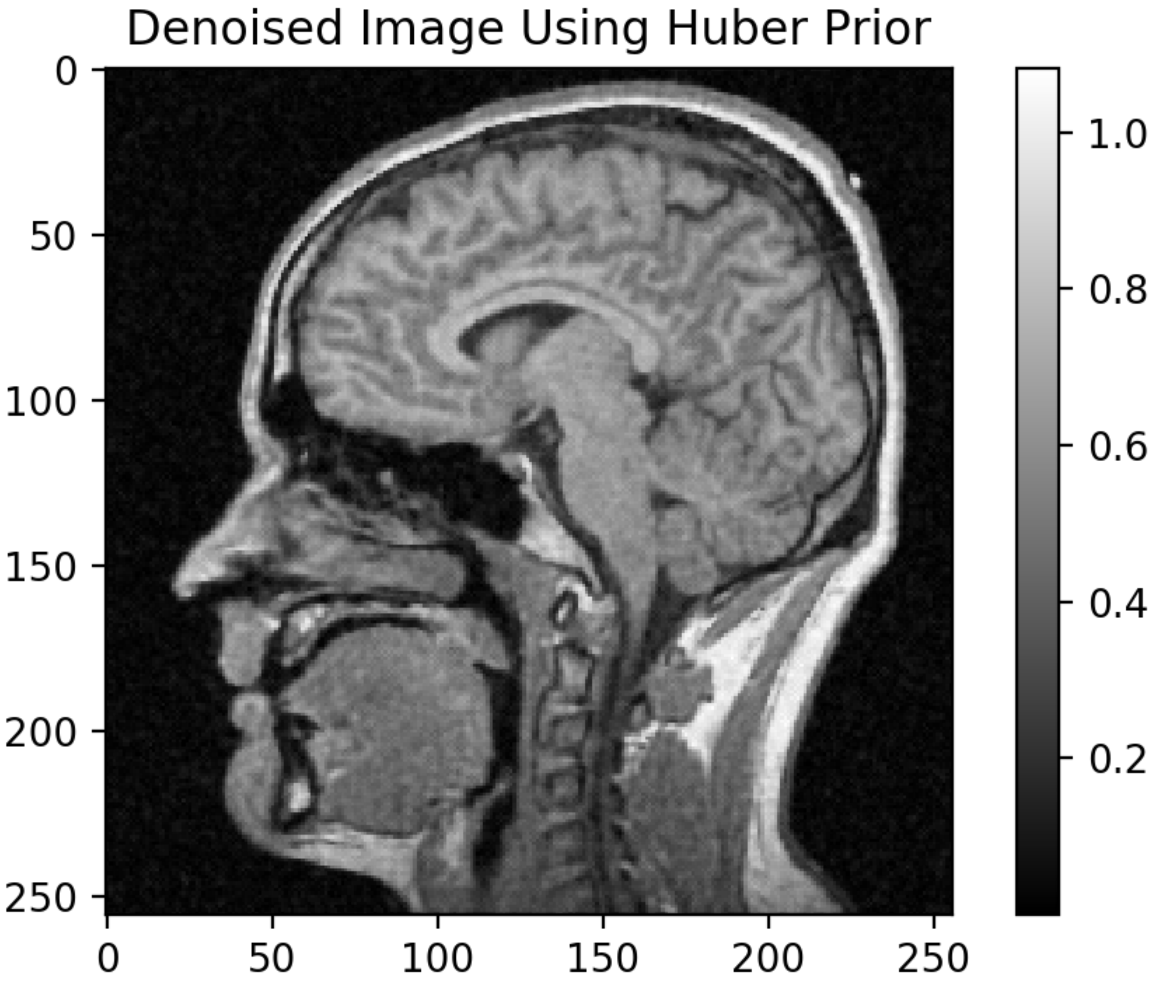
Loading [MathJax]/jax/input/LaTeX/config.js



Denoising Using Huber Prior

```
post_huber, denoised_model_huber = routine(imageNoisy, 0.4, 0.3, step, threshold,
post_huber=post_huber[1:]
plt.figure()
plt.imshow(np.absolute(denoised_model_huber),cmap='gray')
plt.title('Denoised Image Using Huber Prior')
plt.colorbar()
```

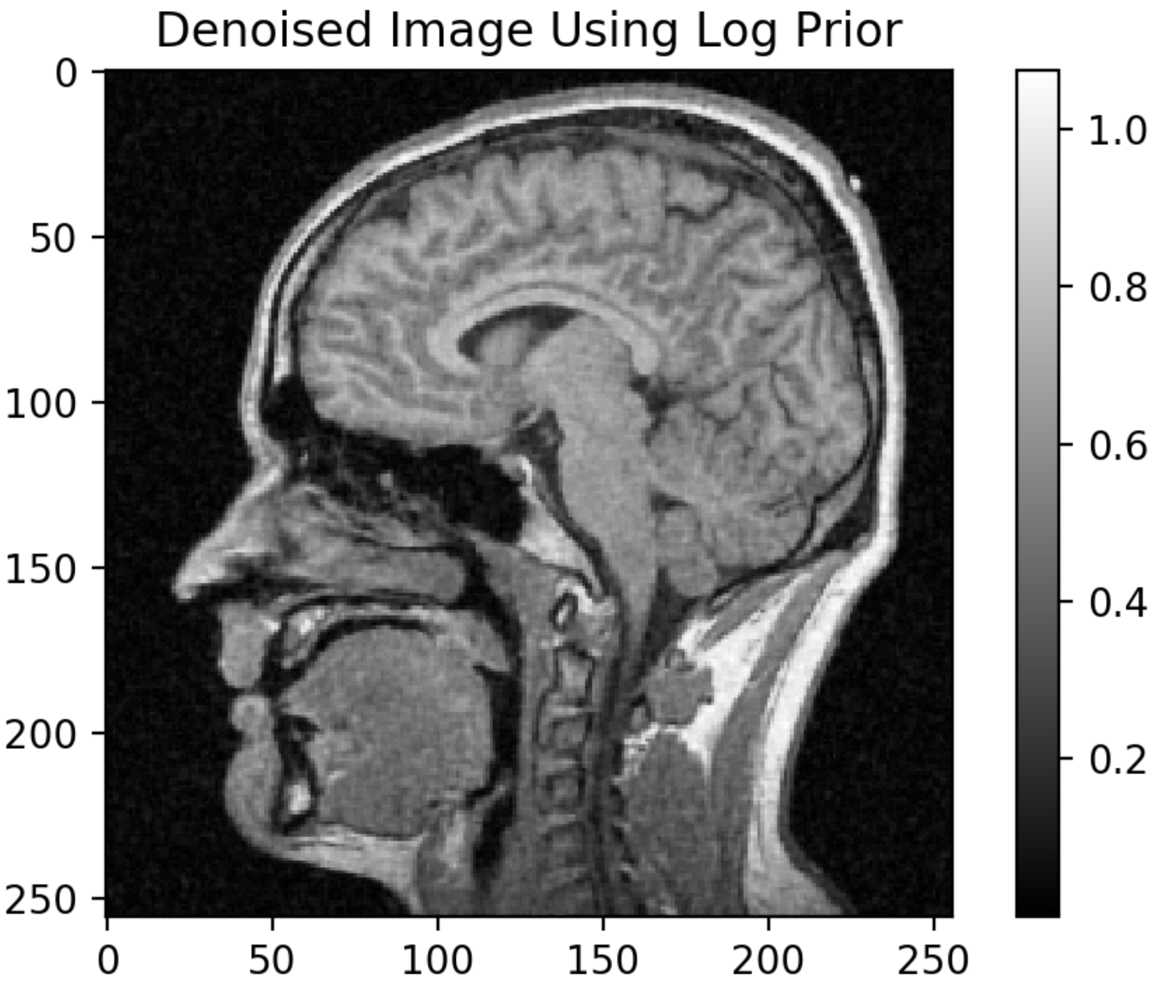
```
<matplotlib.colorbar.Colorbar at 0x15f159f46a0>
```



Denoising Using Log Prior

```
post_log, denoised_model_log = routine(imageNoisy, 0.4, 0.6, step, threshold, "log
post_log=post_log[1:]
plt.figure()
plt.imshow(np.absolute(denoised_model_log),cmap='gray')
plt.title('Denoised Image Using Log Prior')
plt.colorbar()
```

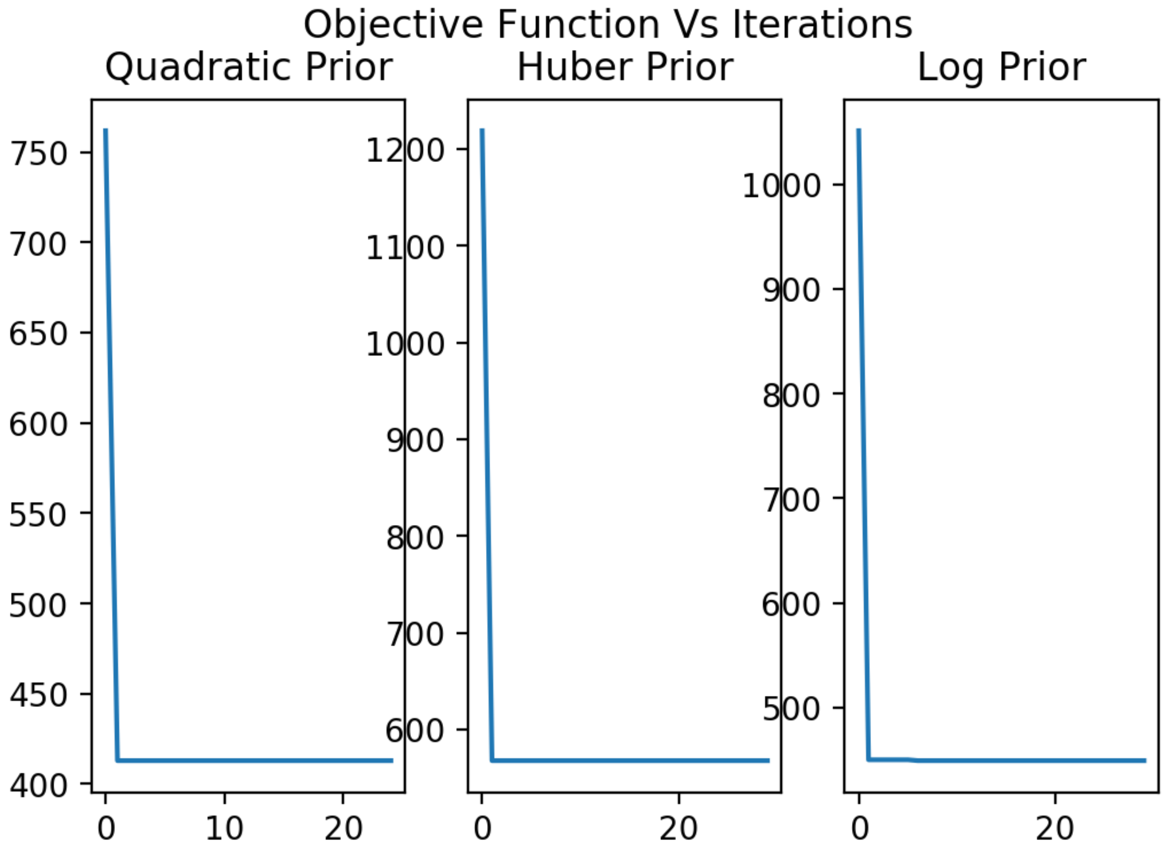
```
<matplotlib.colorbar.Colorbar at 0x15f15779828>
```



Posterior Value vs Iteration

```
f, ((ax1, ax2, ax3)) = plt.subplots(1, 3, sharex='col')
plt.suptitle('Objective Function Vs Iterations')

x_axis = np.arange(25)
ax1.plot(x_axis, post_quad)
ax1.set_title('Quadratic Prior')

x_axis = np.arange(30)
ax2.plot(x_axis, post_huber)
ax2.set_title('Huber Prior')

ax3.plot(x_axis, post_log)
ax3.set_title('Log Prior')
```

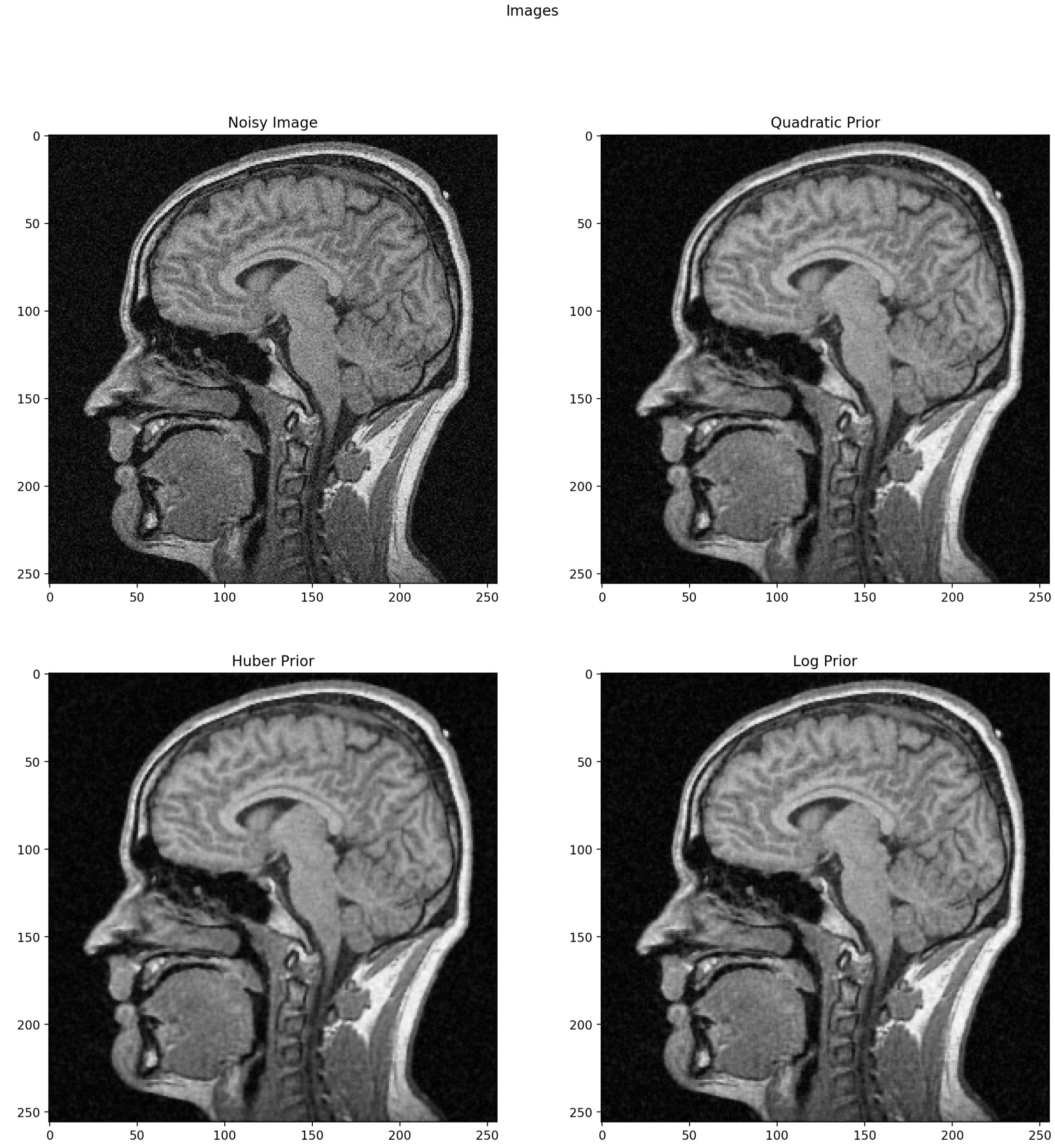
```
Text(0.5,1,'Log Prior')
```



Subplot

```
f, (((ax1, ax2), (ax3, ax4))) = plt.subplots(2, 2, figsize=(15,15))
plt.suptitle('Images')


ax2.imshow(np.absolute(denoised_model_quad),cmap='gray')
ax2.set_title('Quadratic Prior')

ax3.imshow(np.absolute(denoised_model_huber),cmap='gray')
ax3.set_title('Huber Prior')


ax4.imshow(np.absolute(denoised_model_log),cmap='gray')
ax4.set_title('Log Prior')

ax1.imshow(np.absolute(imageNoisy),cmap='gray')
ax1.set_title('Noisy Image')
```

```
Text(0.5,1,'Noisy Image')
```



Published from q2.py using Pweave 0.30.3 on 10-02-2019.

Loading [MathJax]/jax/input/LaTeX/config.js