libraries and functions

```
import numpy as np
import h5py
import matplotlib.pyplot as plt
import numpy as np
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
```

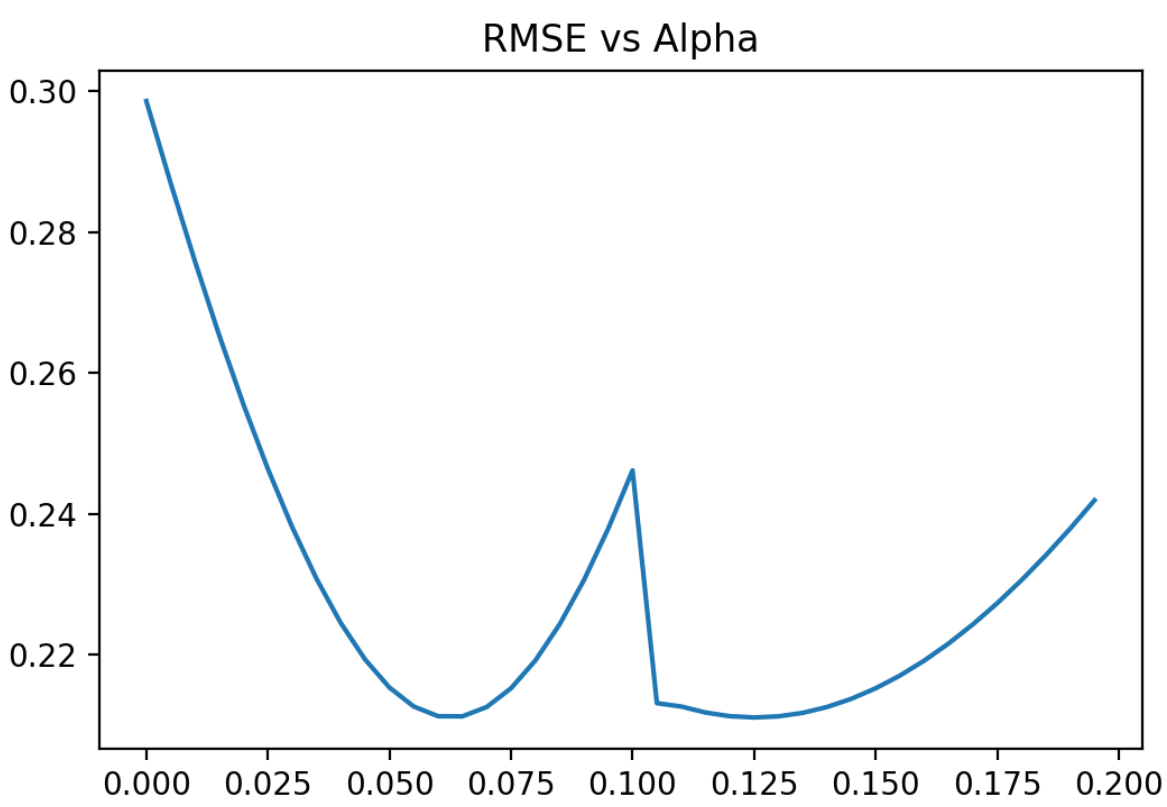
Image Data

```
f = h5py.File('../data/assignmentImageDenoisingPhantom.mat','r')
imageNoiseless = f.get('imageNoiseless')
imageNoiseless = np.array(imageNoiseless)
imageNoiseless = imageNoiseless.T
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
```

optimization for huber prior

```
threshold = 1e-7
alpha_opt_list = []
alpha_opt_list.append(0)
alpha = alpha_opt_list[0]
step = 1

cost_quad=[]
while alpha <= 0.2:
    post, denoised_model_quad = routine(imageNoisy, alpha, 1, step, threshold, "qu
    cost_quad.append(cost(denoised_model_quad, imageNoiseless))
    alpha+=0.005
    alpha_opt_list.append(alpha)

alpha_opt_list = alpha_opt_list[:-1]
plt.figure()
plt.plot(alpha_opt_list, cost_quad)
plt.title('RMSE vs Alpha')
```

```
Text(0.5,1,'RMSE vs Alpha')
```



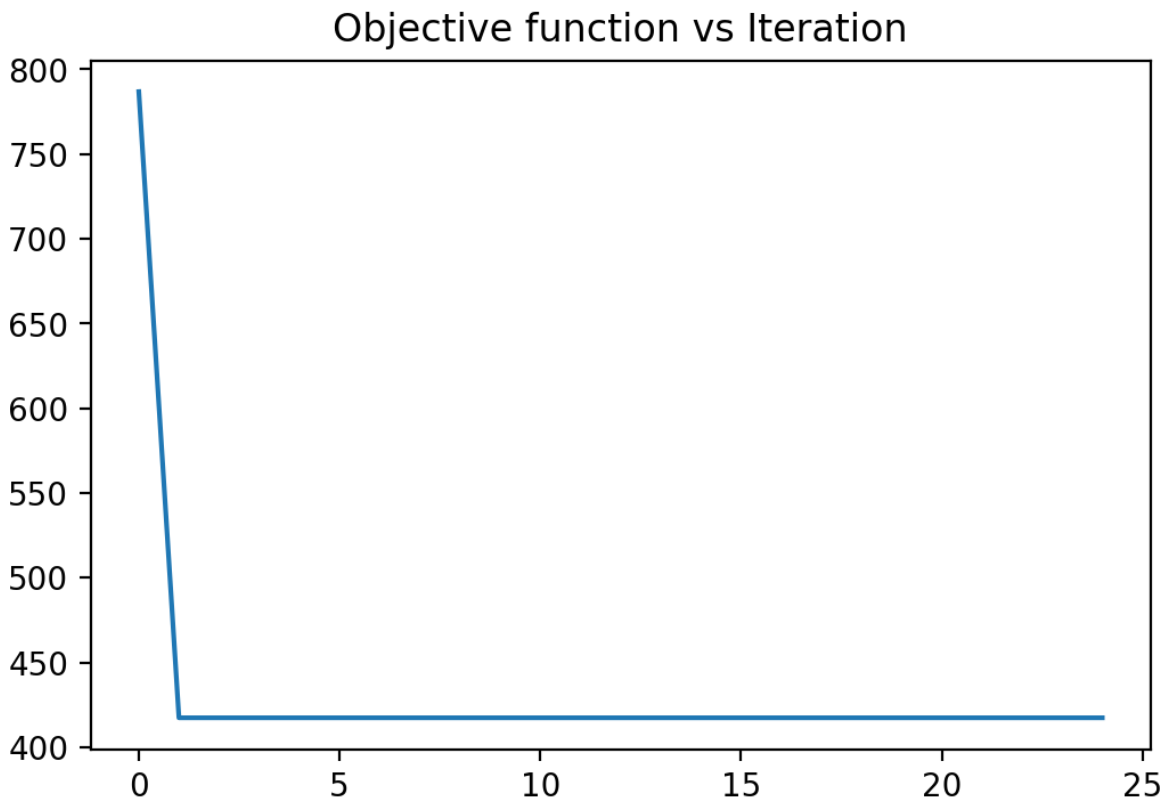
QUADRATIC PRIOR

```
alpha_opt = 0.125
post, denoised_model_quad = routine(imageNoisy, 0.125, 1, step, threshold, "quadra
post = post[1:]
post_alpha1, denoised_model_quad_alpha1 = routine(imageNoisy, 1.2*alpha_opt, 1, st
post_alpha2, denoised_model_quad_alpha2 = routine(imageNoisy, 0.8*alpha_opt, 1, st

cost_noisy = cost(imageNoisy, imageNoiseless)
cost_quad_denoised = cost(denoised_model_quad, imageNoiseless)
cost_quad_alpha1 = cost(denoised_model_quad_alpha1, imageNoiseless)
cost_quad_alpha2 = cost(denoised_model_quad_alpha2, imageNoiseless)

print('RMSE for noisy image : %s' %(cost_noisy))
print('RMSE for denoised image using alpha=%s and gamma=%s for quad prior : %s' %(
print('RMSE for denoised image using alpha=%s and gamma=%s for quad prior : %s' %(
print('RMSE for denoised image using alpha=%s and gamma=%s for quad prior : %s' %(

#plot of posterior vs iteration

plt.figure()
x_axis = np.arange(25)
plt.plot(x_axis, post)
plt.title('Objective function vs Iteration')
```

```
RMSE for noisy image : 0.29857915712437444
RMSE for denoised image using alpha=0.125 and gamma=1 for quad prior :
0.2111178046295691
RMSE for denoised image using alpha=0.15 and gamma=1 for quad prior :
0.2152557004561629
```

```
RMSE for denoised image using alpha=0.1 and gamma=1 for quad prior :
0.2461763203595474
```

```
Text(0.5,1,'Objective function vs Iteration')
```



optimization of parameters for log prior

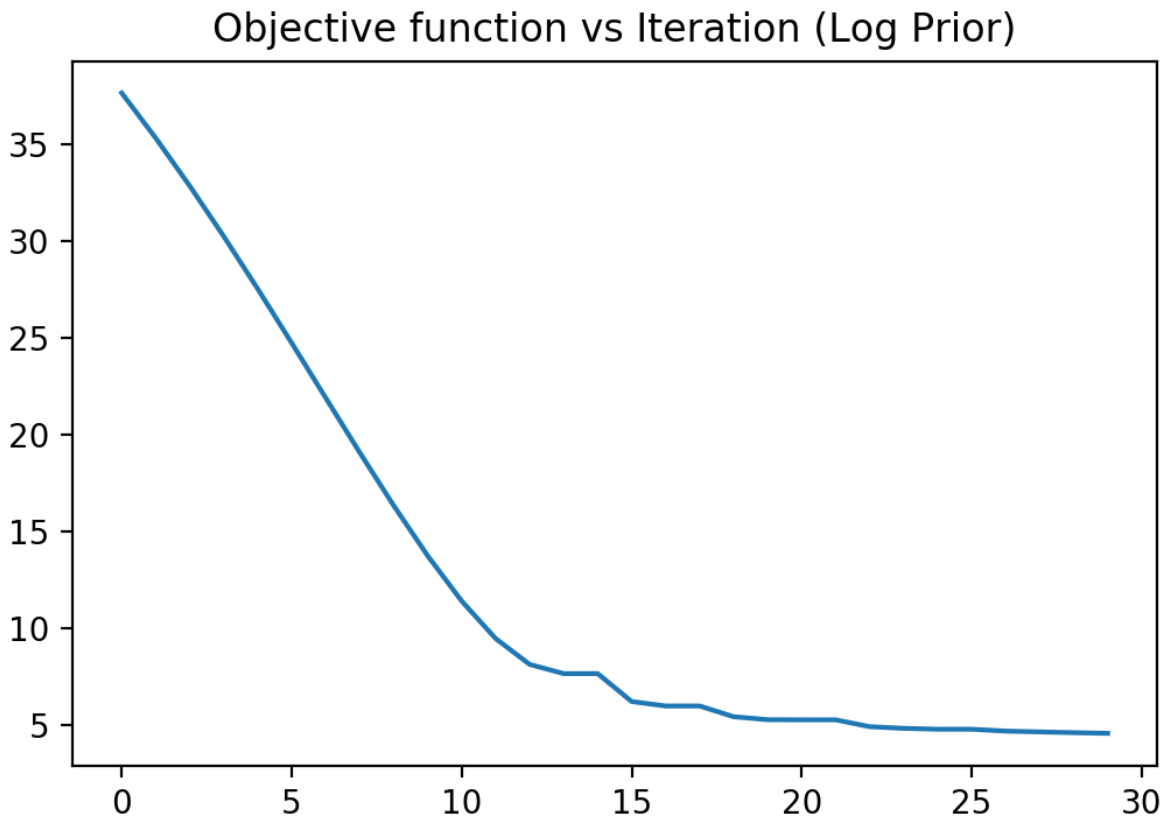
```
#threshold = 1e-6
#alpha_opt_list = []
#alpha_opt_list.append(0.995)
#alpha = alpha_opt_list[0]
#step = 1
#gamma_opt = []
#gamma_opt.append(0.001)
#
#cost_quad=[]
#while alpha <= 1:
#    gamma = 0.001
#    gamma_list = []
#    gamma_list.append(gamma)
#    cost_quad_list = []
#    while gamma < 0.002:
#        post, denoised_model_quad = routine(imageNoisy, alpha, gamma, step, thres
#        cost_quad_list.append(cost(denoised_model_quad, imageNoiseless))
#        gamma+=0.0001
#        gamma_list.append(gamma)
#    alpha+=0.0005
#    alpha_opt_list.append(alpha)
#    cost_quad.append(cost_quad_list)
#    gamma_opt.append(gamma_list)
```

optimum values for log prior

```
alpha_opt = 1
gamma_opt = 0.0014
post, denoised_model_log = routine(imageNoisy, alpha_opt, gamma_opt, step, thresho
post = post[1:]
post_alpha2, denoised_model_log_alpha2 = routine(imageNoisy, 0.8*alpha_opt, gamma_
post_gamma1, denoised_model_log_gamma1 = routine(imageNoisy, alpha_opt, 1.2*gamma_
post_gamma2, denoised_model_log_gamma2 = routine(imageNoisy, alpha_opt, 0.8*gamma_

cost_noisy = cost(imageNoisy, imageNoiseless)
cost_log_denoised = cost(denoised_model_log, imageNoiseless)
cost_log_alpha2 = cost(denoised_model_log_alpha2, imageNoiseless)
cost_log_gamma1 = cost(denoised_model_log_gamma1, imageNoiseless)
cost_log_gamma2 = cost(denoised_model_log_gamma2, imageNoiseless)

print('RMSE for noisy image : %s' %(cost_noisy))
print('RMSE for denoised image using alpha=%s and gamma=%s for log prior : %s' %(a
print('RMSE for denoised image using alpha=%s and gamma=%s for log prior : %s' %(0
print('RMSE for denoised image using alpha=%s and gamma=%s for log prior : %s' %(a
print('RMSE for denoised image using alpha=%s and gamma=%s for log prior : %s' %(a

x_axis = np.arange(30)
plt.figure()
plt.plot(x_axis, post)
plt.title('Objective function vs Iteration (Log Prior)')
```

```
RMSE for noisy image : 0.29857915712437444
RMSE for denoised image using alpha=1 and gamma=0.0014 for log prior :
0.063155318931628
RMSE for denoised image using alpha=0.8 and gamma=0.0014 for log prior
: 0.2516109070098188
RMSE for denoised image using alpha=1 and gamma=0.0016799999999999999
for log prior : 0.0633786184686772
RMSE for denoised image using alpha=1 and gamma=0.0011200000000000001
for log prior : 0.06346713317421625
```

```
Text(0.5,1,'Objective function vs Iteration (Log Prior)')
```



optimization for huber prior

```
#threshold = 1e-6
#alpha_opt_list = []
#alpha_opt_list.append(0.995)
#alpha = alpha_opt_list[0]
#step = 1
#gamma_opt = []
#
#cost_quad=[]
#while alpha <= 1:
#    gamma = 0.003
#    gamma_list = []
#    gamma_list.append(gamma)
#    cost_quad_list = []
#    while gamma < 0.004:
#        post, denoised_model_quad = routine(imageNoisy, alpha, gamma, step, thres
#        cost_quad_list.append(cost(denoised_model_quad, imageNoiseless))
#        gamma+=0.0001
#        gamma_list.append(gamma)
#    alpha+=0.0005
#    alpha_opt_list.append(alpha)
#    cost_quad.append(cost_quad_list)
#    gamma_opt.append(gamma_list)
```

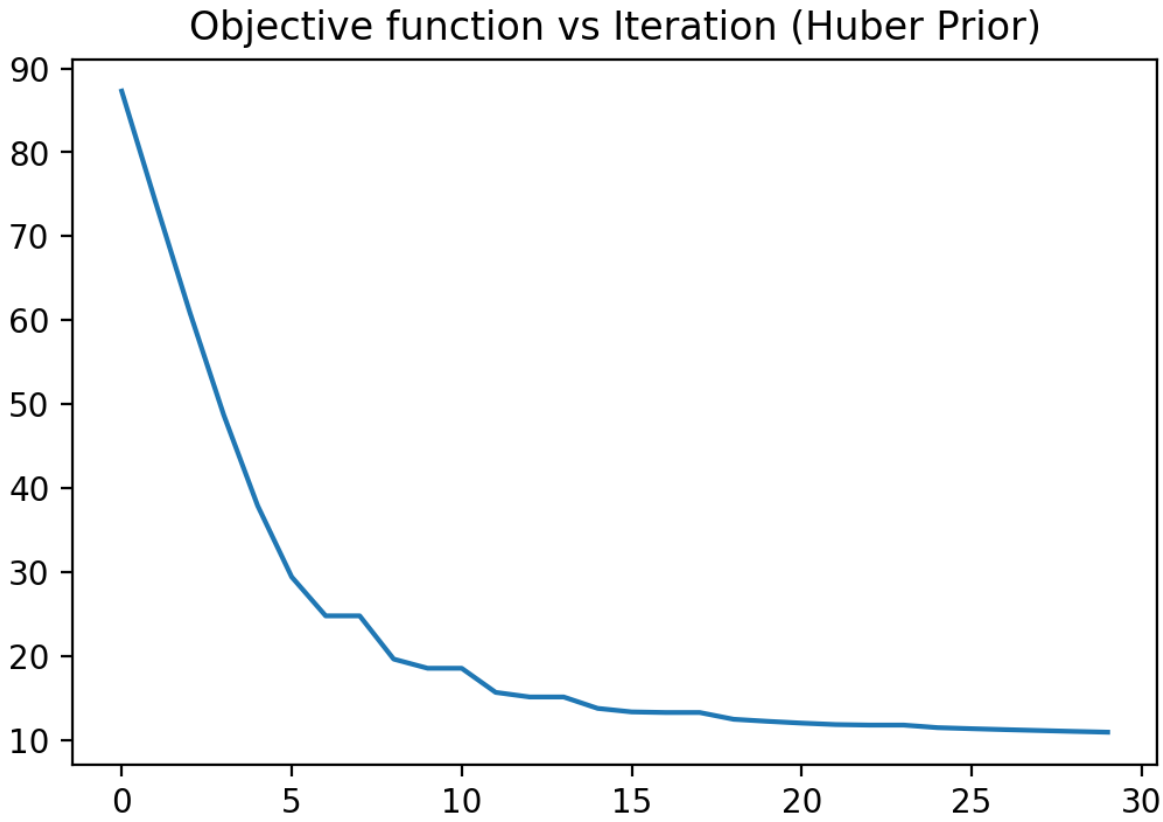
optimum values for huber prior

```
alpha_opt = 1
```

```
post, denoised_model_huber = routine(imageNoisy, alpha_opt, gamma_opt, step, thres
post = post[1:]
post_alpha2, denoised_model_huber_alpha2 = routine(imageNoisy, 0.8*alpha_opt, gamm
post_gamma1, denoised_model_huber_gamma1 = routine(imageNoisy, alpha_opt, 1.2*gamm
post_gamma2, denoised_model_huber_gamma2 = routine(imageNoisy, alpha_opt, 0.8*gamm

cost_noisy = cost(imageNoisy, imageNoiseless)
cost_huber_denoised = cost(denoised_model_huber, imageNoiseless)
cost_huber_alpha2 = cost(denoised_model_huber_alpha2, imageNoiseless)
cost_huber_gamma1 = cost(denoised_model_huber_gamma1, imageNoiseless)
cost_huber_gamma2 = cost(denoised_model_huber_gamma2, imageNoiseless)

print('RMSE for noisy image : %s' %(cost_noisy))
print('RMSE for denoised image using alpha=%s and gamma=%s for huber prior : %s' %
print('RMSE for denoised image using alpha=%s and gamma=%s for huber prior : %s' %
print('RMSE for denoised image using alpha=%s and gamma=%s for huber prior : %s' %
print('RMSE for denoised image using alpha=%s and gamma=%s for huber prior : %s' %

x_axis = np.arange(30)
plt.figure()
plt.plot(x_axis, post)
plt.title('Objective function vs Iteration (Huber Prior)')
```

```
RMSE for noisy image : 0.29857915712437444
RMSE for denoised image using alpha=1 and gamma=0.0034 for huber prior
: 0.060565432909687245
RMSE for denoised image using alpha=0.8 and gamma=0.0034 for huber
prior : 0.19254268794947618
RMSE for denoised image using alpha=1 and gamma=0.004079999999999999
for huber prior : 0.060674236401269274
RMSE for denoised image using alpha=1 and gamma=0.00272 for huber
prior : 0.061073599565947136
```

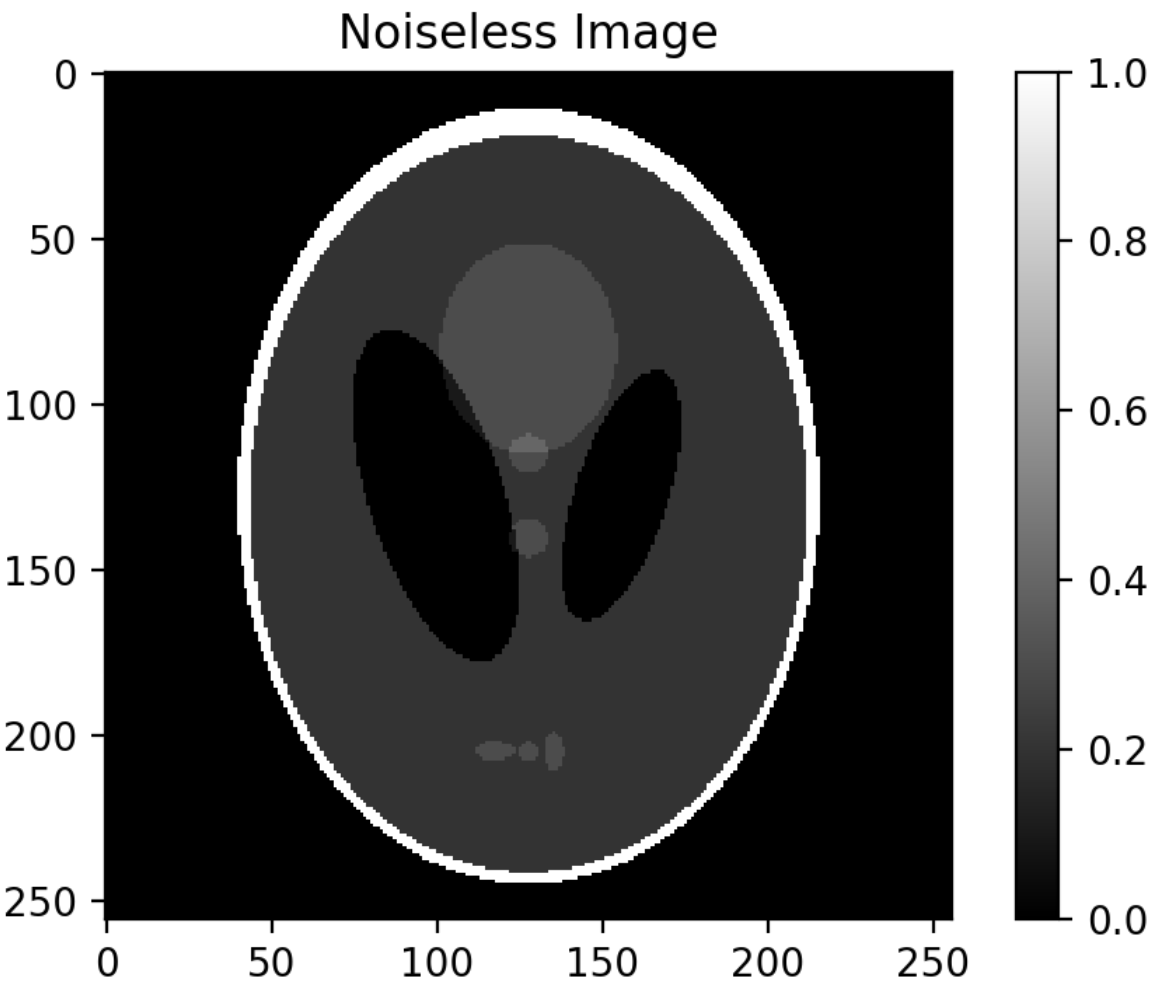
```
Text(0.5,1,'Objective function vs Iteration (Huber Prior)')
```

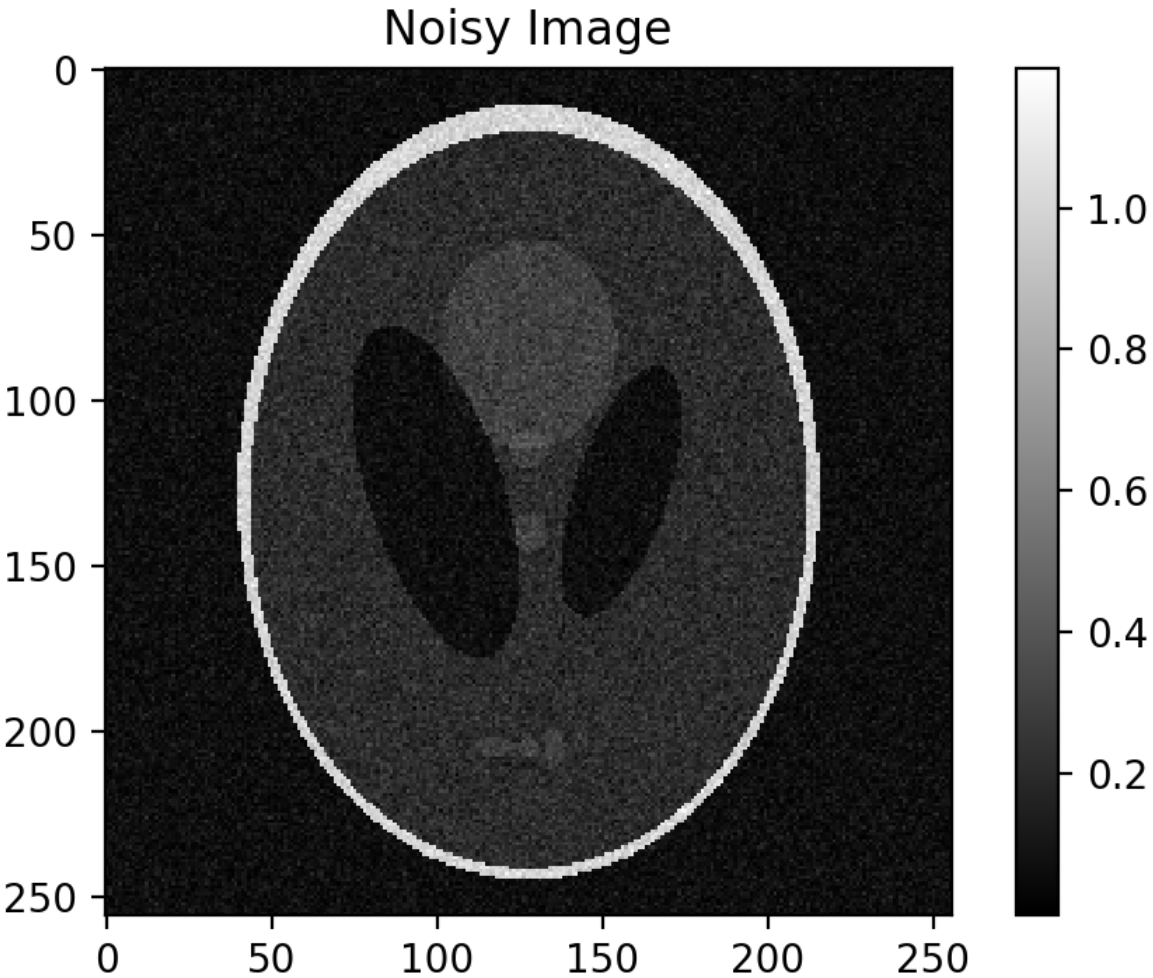


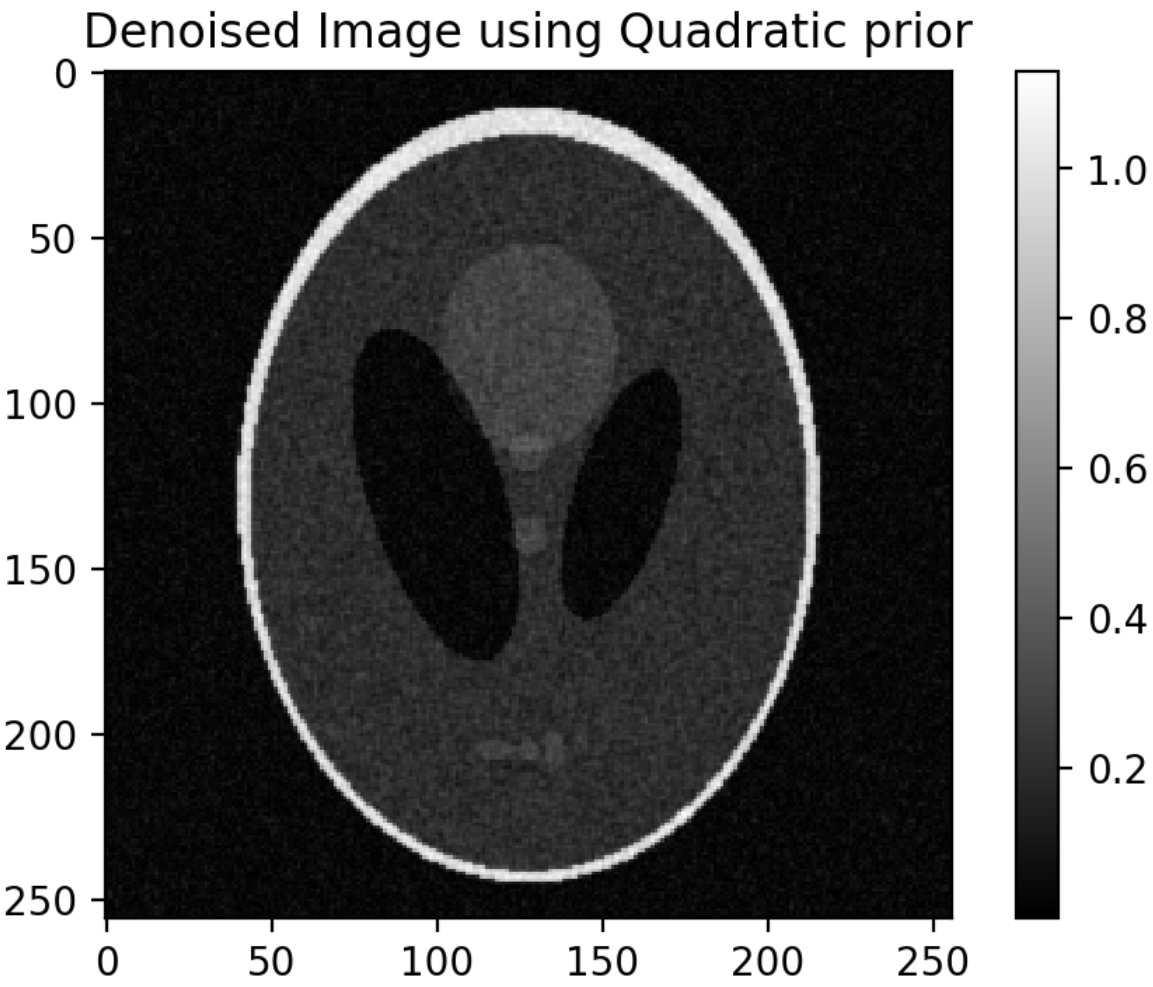
Plots

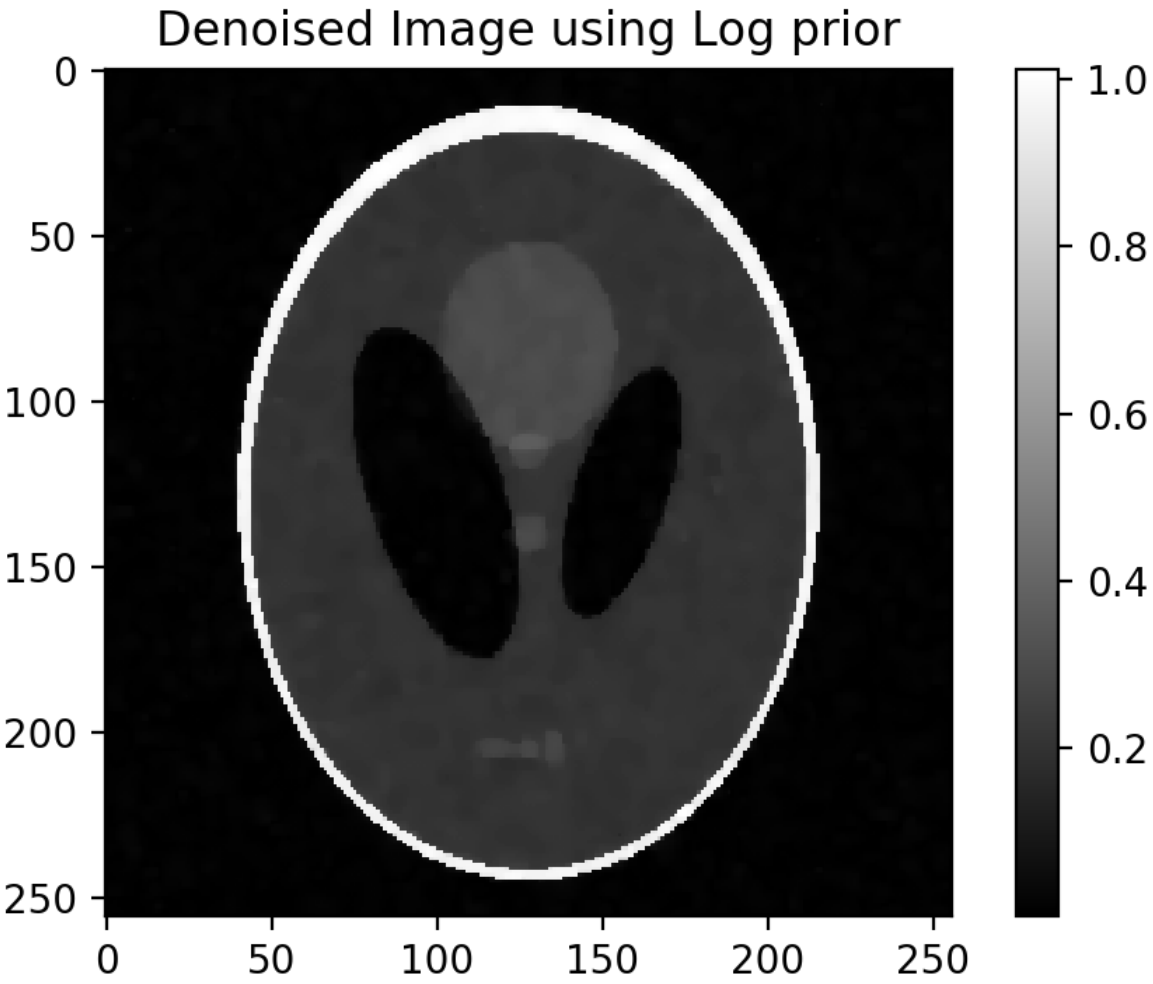
```
plt.figure()
plt.imshow(imageNoiseless,cmap='gray')
plt.title('Noiseless Image')
plt.colorbar()

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

plt.figure()
plt.imshow(np.absolute(denoised_model_quad),cmap='gray')
plt.title('Denoised Image using Quadratic prior')
plt.colorbar()

plt.figure()
plt.imshow(np.absolute(denoised_model_log),cmap='gray')
plt.title('Denoised Image using Log prior')
plt.colorbar()

plt.figure()
plt.imshow(np.absolute(denoised_model_huber),cmap='gray')
plt.title('Denoised Image using Huber prior')
plt.colorbar()
```

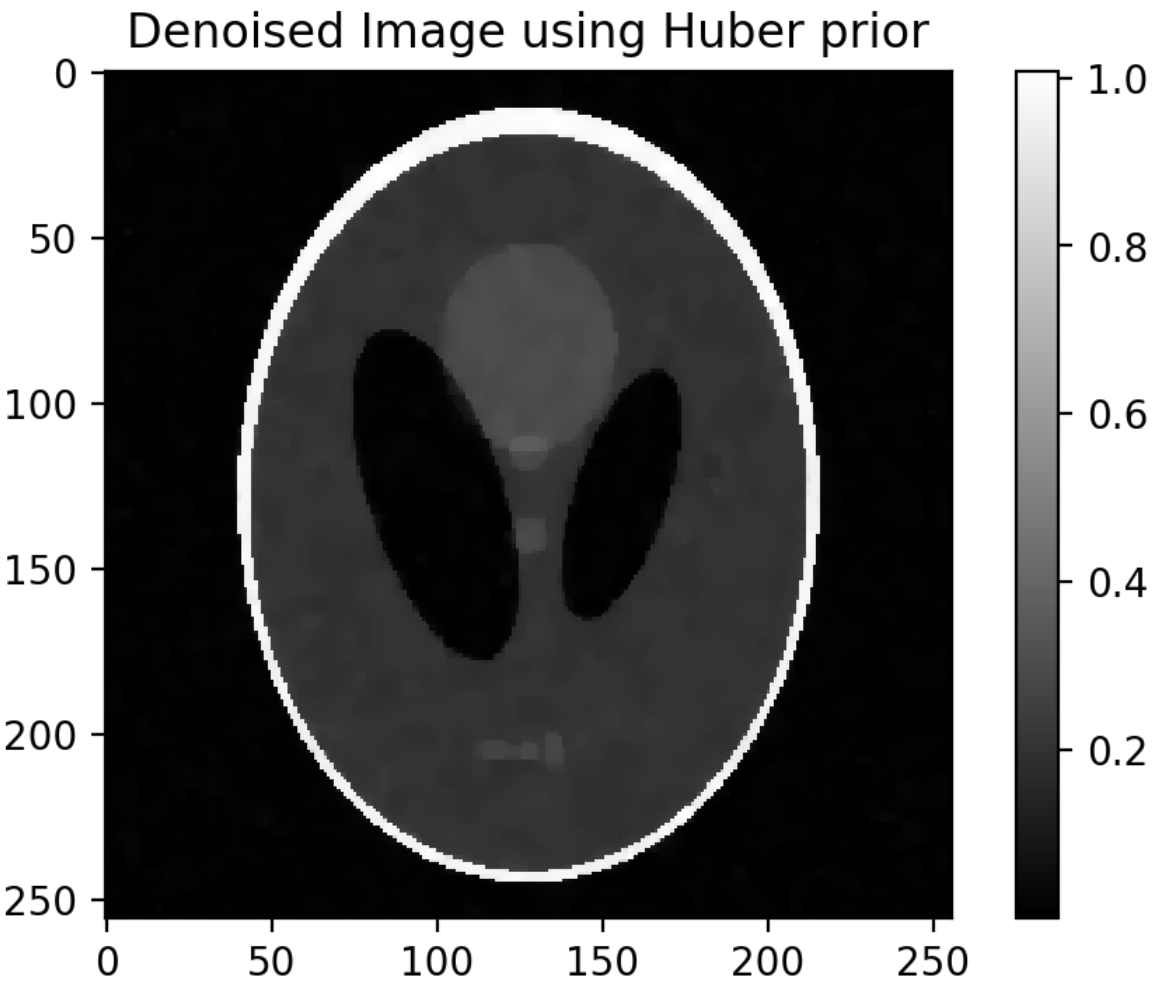
```
<matplotlib.colorbar.Colorbar at 0x18b0e9a1fd0>
```











Published from q1.py using Pweave 0.30.3 on 10-02-2019.