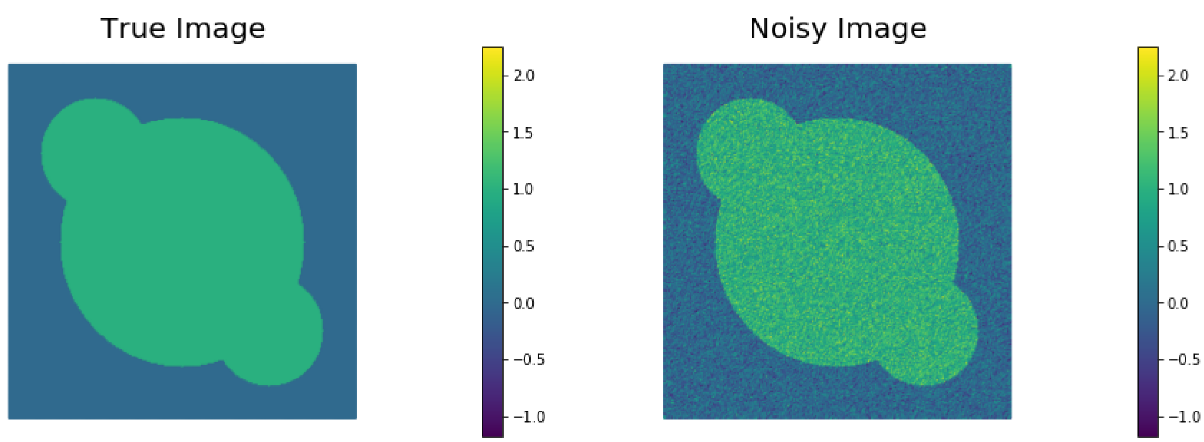
# Problem 3

In [1]:
```
from __future__ import print_function, division, absolute_import

import math

import matplotlib.pyplot as plt
%matplotlib inline

import numpy as np
import scipy.io as sio

import dolfin as dl

from hippylib import nb

import logging

logging.getLogger('FFC').setLevel(logging.WARNING)
logging.getLogger('UFL').setLevel(logging.WARNING)
dl.set_log_active(False)
```

In [2]:
```
class Image(dl.Expression):
    def __init__(self, Lx, Ly, data, **kwargs):
        self.data = data
        self.hx = Lx/float(data.shape[1]-1)
        self.hy = Ly/float(data.shape[0]-1)

    def eval(self, values, x):
        j = int(math.floor(x[0]/self.hx))
        i = int(math.floor(x[1]/self.hy))
        values[0] = self.data[i,j]
```

In [3]:
```
data = sio.loadmat('circles.mat')['im']

Lx = float(data.shape[1])/float(data.shape[0])
Ly = 1.

nx, ny = [256, 256]

mesh = dl.RectangleMesh(dl.Point(0,0),dl.Point(Lx,Ly),nx, ny)
Vm = dl.FunctionSpace(mesh, "Lagrange",1)
Vw = dl.VectorFunctionSpace(mesh, "DG",0)
Vwnorm = dl.FunctionSpace(mesh, "DG",0)

trueImage = Image(Lx,Ly,data,degree = 1)
m_true  = dl.interpolate(trueImage, Vm)

np.random.seed(seed=1)
noise_std_dev = .3
noise = noise_std_dev*np.random.randn(data.shape[0], data.shape[1])
noisyImage = Image(Lx,Ly,data+noise, degree = 1)
d = dl.interpolate(noisyImage, Vm)

# Get min/max of noisy image, so that we can show all plots in the same scale
vmin = np.min(d.vector().get_local())
vmax = np.max(d.vector().get_local())

plt.figure(figsize=(15,5))
nb.plot(m_true, subplot_loc=121, mytitle="True Image", vmin=vmin, vmax = vmax)
nb.plot(d, subplot_loc=122, mytitle="Noisy Image", vmin=vmin, vmax = vmax)
plt.show()
```



In [4]:
```python
class PDTVDenoising:
    def __init__(self, Vm, Vw, Vwnorm, d, alpha, beta):
        self.alpha   = dl.Constant(alpha)
        self.beta    = dl.Constant(beta)
        self.d       = d
        self.m_tilde  = dl.TestFunction(Vm)
        self.m_hat = dl.TrialFunction(Vm)

        self.Vm = Vm
        self.Vw = Vw
        self.Vwnorm = Vwnorm

    def cost_reg(self, m):
        return dl.sqrt( dl.inner(dl.grad(m), dl.grad(m)) + self.beta)*dl.dx

    def cost_misfit(self, m):
        return dl.Constant(.5)*dl.inner(m-self.d, m - self.d)*dl.dx

    def cost(self, m):
        return self.cost_misfit(m) + self.alpha*self.cost_reg(m)

    def grad_m(self, m):
        grad_ls = dl.inner(self.m_tilde, m - self.d)*dl.dx
        TVm = dl.sqrt( dl.inner(dl.grad(m), dl.grad(m)) + self.beta)
        grad_tv = dl.Constant(1.)/TVm*dl.inner(dl.grad(m), dl.grad(self.m_tilde))*dl.dx

        grad = grad_ls + self.alpha*grad_tv

        return grad

    def Hessian(self,m, w):
        H_ls = dl.inner(self.m_tilde, self.m_hat)*dl.dx

        TVm = dl.sqrt( dl.inner(dl.grad(m), dl.grad(m)) + self.beta)
        A = dl.Constant(1.)/TVm * (dl.Identity(2)
                                   - dl.Constant(.5)*dl.outer(w, dl.grad(m)/TVm )
                                   - dl.Constant(.5)*dl.outer(dl.grad(m)/TVm, w ) )

        H_tv = dl.inner(A*dl.grad(self.m_tilde), dl.grad(self.m_hat))*dl.dx

        H = H_ls + self.alpha*H_tv

        return H

    def compute_w_hat(self, m, w, m_hat):
        TVm = dl.sqrt( dl.inner(dl.grad(m), dl.grad(m)) + self.beta)
        A = dl.Constant(1.)/TVm * (dl.Identity(2)
                                   - dl.Constant(.5)*dl.outer(w, dl.grad(m)/TVm )
                                   - dl.Constant(.5)*dl.outer(dl.grad(m)/TVm, w ) )

        expression = A*dl.grad(m_hat) - w + dl.grad(m)/TVm

        return dl.project(expression, self.Vw)

    def wnorm(self, w):
        return dl.inner(w,w)
```

In [5]:
```
def PDNewton(pdProblem, m, w, parameters):

    termination_reasons = [
                           "Maximum number of Iteration reached",      #0
                           "Norm of the gradient less than tolerance", #1
                           "Maximum number of backtracking reached",   #2
                           "Norm of (g, m_hat) less than tolerance"       #3
                           ]

    rtol          = parameters["rel_tolerance"]
    atol          = parameters["abs_tolerance"]
    gdm_tol       = parameters["gdm_tolerance"]
    max_iter      = parameters["max_iter"]
    c_armijo      = parameters["c_armijo"]
    max_backtrack = parameters["max_backtracking_iter"]
    prt_level     = parameters["print_level"]
    cg_coarse_tol = parameters["cg_coarse_tolerance"]

    Jn = dl.assemble( pdProblem.cost(m)   )
    gn = dl.assemble( pdProblem.grad_m(m) )
    g0_norm = gn.norm("l2")
    gn_norm = g0_norm
    tol = max(g0_norm*rtol, atol)

    m_hat = dl.Function(pdProblem.Vm)
    w_hat = dl.Function(pdProblem.Vw)

    converged = False
    reason = 0
    total_cg_iter = 0

    if prt_level > 0:
        print( "{0:>3} {1:>15} {2:>15} {3:>15} {4:>15} {5:>15} {6:>7}".format(
                "It", "cost", "||g||", "(g,m_hat)", "alpha_m", "tol_cg", "cg_it") )

    for it in range(max_iter):

        # Compute m_hat
        Hn = dl.assemble( pdProblem.Hessian(m,w) )
        solver = dl.PETScKrylovSolver("cg", "petsc_amg")
        solver.set_operator(Hn)
        solver.parameters["nonzero_initial_guess"] = False
        cg_tol = min(cg_coarse_tol, math.sqrt( gn_norm/g0_norm) )
        solver.parameters["relative_tolerance"] = cg_tol
        lin_it = solver.solve(m_hat.vector(),-gn)
        total_cg_iter += lin_it

        # Compute w_hat
        w_hat = pdProblem.compute_w_hat(m, w, m_hat)

        ### Line search for m
        mhat_gn = m_hat.vector().inner(gn)

        if(-mhat_gn < gdm_tol):
            self.converged=True
            self.reason = 3
            break

        alpha_m = 1.
        bk_converged = False
        for j in range(max_backtrack):
            Jnext = dl.assemble( pdProblem.cost(m + dl.Constant(alpha_m)*m_hat) )
            if Jnext < Jn + alpha_m*c_armijo*mhat_gn:
                Jn = Jnext
                bk_converged = True
                break

            alpha_m = alpha_m/2.

        if not bk_converged:
            self.reason = 2
            break

        ### Line search for w
        alpha_w = 1
        bk_converged = False
```

```
    for j in range(max_backtrack):
        norm_w = dl.project(pdProblem.wnorm(w + dl.Constant(alpha_w)*w_hat), pdProblem.Vwnorm)
        if norm_w.vector().norm("linf") <= 1:
            bk_converged = True
            break
        alpha_w = alpha_w/2.


    ### Update
    m.vector().axpy(alpha_m, m_hat.vector())
    w.vector().axpy(alpha_w, w_hat.vector())

    gn = dl.assemble( pdProblem.grad_m(m) )
    gn_norm = gn.norm("l2")

    if prt_level > 0:
        print( "{0:3d} {1:15e} {2:15e} {3:15e} {4:15e} {5:15e} {6:7d}".format(
                it, Jn, gn_norm, mhat_gn, alpha_m, cg_tol, lin_it) )

    if gn_norm < tol:
        converged = True
        reason = 1
        break

final_grad_norm = gn_norm

if prt_level > -1:
    print( termination_reasons[reason] )
    if converged:
        print( "Inexact Newton CG converged in ", it, \
            "nonlinear iterations and ", total_cg_iter, "linear iterations." )
    else:
        print( "Inexact Newton CG did NOT converge after ", self.it, \
            "nonlinear iterations and ", total_cg_iter, "linear iterations.")
    print ("Final norm of the gradient", final_grad_norm)
    print ("Value of the cost functional", Jn)

return m, w
```

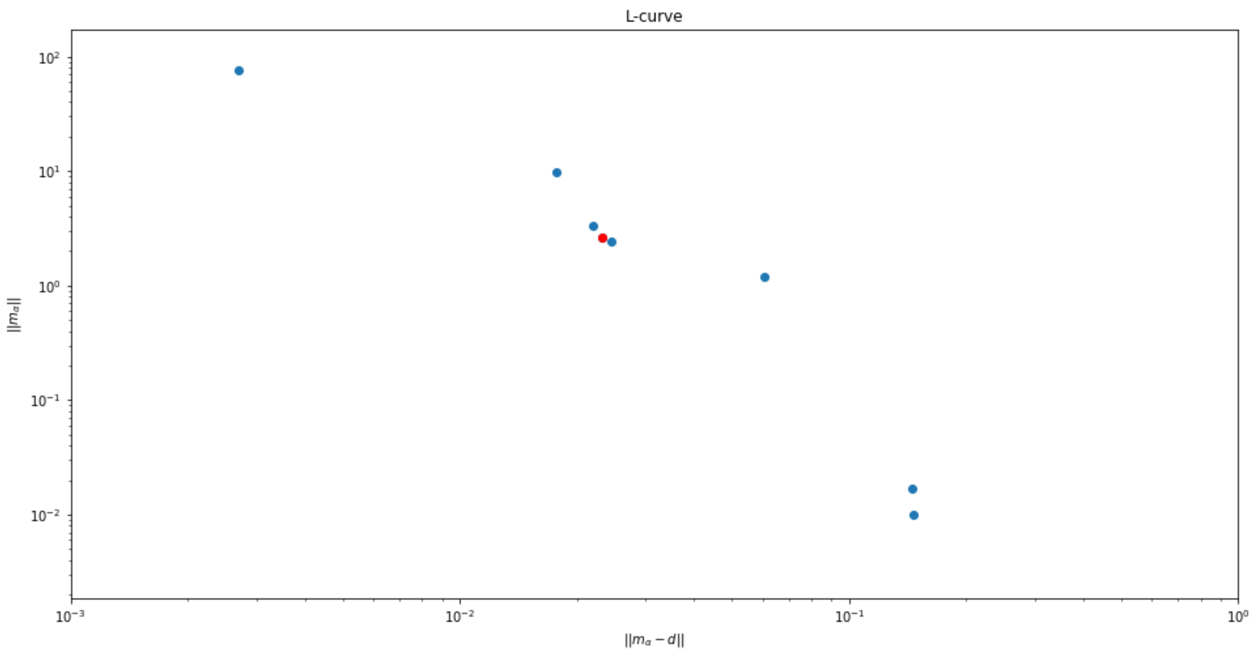
# Part A

In [6]:

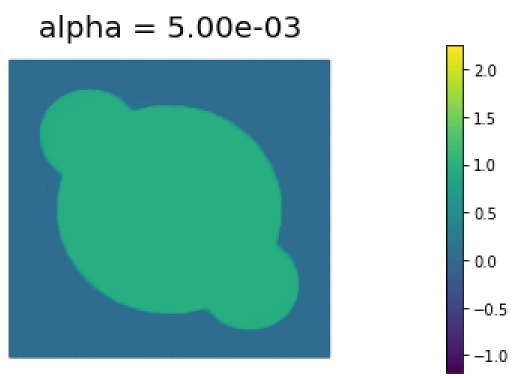
```python
alphas = (1e-4,0.5e-4,1e-3,0.5e-3,1e-2,0.5e-2,1e-1,0.5e-1,1)
misfit = np.zeros((len(alphas),))
reg = np.zeros((len(alphas),))
imgs = list()
for n,alpha in enumerate(alphas):
    print('running alpha = {}'.format(alpha))
    # run pd problem (from original code above)
    beta  = 1e-4
    pdProblem = PDTVDenoising(Vm, Vw, Vwnorm, d, alpha, beta)

    parameters = {}
    parameters["rel_tolerance"]         = 1e-6
    parameters["abs_tolerance"]         = 1e-9
    parameters["gdm_tolerance"]         = 1e-18
    parameters["max_iter"]              = 100
    parameters["c_armijo"]              = 1e-5
    parameters["max_backtracking_iter"] = 10
    parameters["print_level"]           = -1
    parameters["cg_coarse_tolerance"]   = 0.5

    m0 = dl.Function(Vm)
    w0 = dl.Function(Vw)

    m, w = PDNewton(pdProblem, m0, w0, parameters)

    # calculate the norm
    reg[n] = dl.assemble(pdProblem.cost_reg(m))

    # calculate misfit
    misfit[n] = dl.assemble(pdProblem.cost_misfit(m))

    # save images
    imgs.append(m)
```

```
running alpha = 0.0001
running alpha = 5e-05
running alpha = 0.001
running alpha = 0.0005
running alpha = 0.01
running alpha = 0.005
running alpha = 0.1
running alpha = 0.05
running alpha = 1
```

In [7]:
```
# plot L-curve
fig = plt.figure(figsize=(16,8))
plt.scatter(misfit,reg)
plt.scatter(misfit[-4],reg[-4],c='r')
ax = fig.axes[0]
ax.set_yscale('log')
ax.set_xscale('log')
plt.xlim([1e-3,1])
plt.title('L-curve')
plt.xlabel(r'$||m_{\alpha} - d||$');
plt.ylabel(r'$||m_{\alpha}||$')
plt.show()
```




In [8]:
```
# plot optimal alpha and image
plt.figure()
nb.plot(imgs[-4], vmin=vmin, vmax = vmax, mytitle="alpha = {0:1.2e}".format(alphas[-4]))
plt.show()
```




# Part B

In [9]:

```
betas = ((10,1,1e-1,1e-2,1e-3,1e-4))
img2s = list()
for n,beta in enumerate(betas):
    print('running beta = {}'.format(beta))
    # run pd problem (from original code above)
    alpha  = 1e-3
    pdProblem = PDTVDenoising(Vm, Vw, Vwnorm, d, alpha, beta)

    parameters = {}
    parameters["rel_tolerance"]         = 1e-6
    parameters["abs_tolerance"]         = 1e-9
    parameters["gdm_tolerance"]         = 1e-18
    parameters["max_iter"]              = 100
    parameters["c_armijo"]              = 1e-5
    parameters["max_backtracking_iter"] = 10
    parameters["print_level"]           = 1
    parameters["cg_coarse_tolerance"]   = 0.5

    m0 = dl.Function(Vm)
    w0 = dl.Function(Vw)

    m, w = PDNewton(pdProblem, m0, w0, parameters)

    # convert to proper numpy array
    mv = m.compute_vertex_values(m.function_space().mesh())

    # save images
    img2s.append(mv)
```

```
running beta = 10
 It          cost          ||g||       (g,m_hat)        alpha_m          tol_cg   cg_it
  0    3.215866e-02    4.688506e-04   -4.512554e-01    1.000000e+00    5.000000e-01       1
  1    2.809259e-02    2.045145e-04   -6.799610e-03    1.000000e+00    4.126404e-01       1
  2    2.757790e-02    1.066850e-04   -7.973416e-04    1.000000e+00    2.725314e-01       1
  3    2.750190e-02    4.459642e-05   -1.087279e-04    1.000000e+00    1.968368e-01       1
  4    2.748923e-02    1.378772e-05   -1.884316e-05    1.000000e+00    1.272637e-01       1
  5    2.748776e-02    3.335028e-06   -2.341911e-06    1.000000e+00    7.076211e-02       1
  6    2.748766e-02    5.985899e-07   -1.672768e-07    1.000000e+00    3.480200e-02       1
  7    2.748766e-02    6.196984e-08   -5.967569e-09    1.000000e+00    1.474413e-02       2
  8    2.748766e-02    1.821491e-10   -9.615729e-11    1.000000e+00    4.744000e-03       2
Norm of the gradient less than tolerance
Inexact Newton CG converged in  8 nonlinear iterations and  11 linear iterations.
Final norm of the gradient 1.8214911959722557e-10
Value of the cost functional 0.027487655126984766
running beta = 1
 It          cost          ||g||       (g,m_hat)        alpha_m          tol_cg   cg_it
  0    3.628838e-02    6.995184e-04   -4.317911e-01    1.000000e+00    5.000000e-01       1
  1    2.705554e-02    3.463109e-04   -1.600986e-02    1.000000e+00    5.000000e-01       1
  2    2.602971e-02    2.210317e-04   -1.680592e-03    1.000000e+00    3.546399e-01       1
  3    2.588171e-02    1.126698e-04   -2.090098e-04    1.000000e+00    2.833230e-01       1
  4    2.585210e-02    4.216966e-05   -4.137667e-05    1.000000e+00    2.022825e-01       1
  5    2.584748e-02    1.471085e-05   -6.582899e-06    1.000000e+00    1.237527e-01       1
  6    2.584669e-02    5.542744e-06   -1.159120e-06    1.000000e+00    7.309260e-02       2
  7    2.584655e-02    1.560673e-06   -2.100023e-07    1.000000e+00    4.486596e-02       2
  8    2.584654e-02    2.859651e-07   -1.959513e-08    1.000000e+00    2.380732e-02       2
  9    2.584654e-02    3.470454e-08   -8.581276e-10    1.000000e+00    1.019087e-02       3
 10    2.584654e-02    2.787888e-09   -1.703223e-11    1.000000e+00    3.550158e-03       3
 11    2.584654e-02    1.642246e-12   -1.437009e-13    1.000000e+00    1.006218e-03       4
Norm of the gradient less than tolerance
Inexact Newton CG converged in  11 nonlinear iterations and  22 linear iterations.
Final norm of the gradient 1.6422463224981146e-12
Value of the cost functional 0.025846538768234412
running beta = 0.1
 It          cost          ||g||       (g,m_hat)        alpha_m          tol_cg   cg_it
  0    4.717350e-02    9.781427e-04   -3.989779e-01    1.000000e+00    5.000000e-01       1
  1    2.667765e-02    5.259445e-04   -3.811451e-02    1.000000e+00    5.000000e-01       1
  2    2.569011e-02    3.737548e-04   -1.864372e-03    1.000000e+00    4.370433e-01       1
  3    2.546822e-02    2.303229e-04   -2.985478e-04    1.000000e+00    3.684240e-01       1
  4    2.541536e-02    1.381327e-04   -6.368838e-05    1.000000e+00    2.892165e-01       1
  5    2.539932e-02    8.032711e-05   -1.930024e-05    1.000000e+00    2.239767e-01       1
  6    2.539243e-02    4.556255e-05   -8.580952e-06    1.000000e+00    1.707991e-01       2
  7    2.538951e-02    2.512497e-05   -3.731890e-06    1.000000e+00    1.286348e-01       2
  8    2.538821e-02    1.122150e-05   -1.820580e-06    1.000000e+00    9.552285e-02       2
  9    2.538793e-02    4.567259e-06   -4.022044e-07    1.000000e+00    6.383811e-02       3
 10    2.538788e-02    1.278860e-06   -7.876652e-08    1.000000e+00    4.072699e-02       3
 11    2.538788e-02    2.292202e-07   -6.750081e-09    1.000000e+00    2.155093e-02       4
 12    2.538788e-02    2.890231e-08   -2.756476e-10    1.000000e+00    9.123910e-03       5
 13    2.538788e-02    2.451012e-09   -5.870234e-12    1.000000e+00    3.239820e-03       5
Norm of the gradient less than tolerance
Inexact Newton CG converged in  13 nonlinear iterations and  32 linear iterations.
Final norm of the gradient 2.451012104347244e-09
Value of the cost functional 0.025387880814962387
running beta = 0.01
 It          cost          ||g||       (g,m_hat)        alpha_m          tol_cg   cg_it
  0    6.973606e-02    1.312771e-03   -3.484198e-01    1.000000e+00    5.000000e-01       1
  1    2.806234e-02    6.330617e-04   -7.739738e-02    1.000000e+00    5.000000e-01       1
  2    2.574248e-02    4.801157e-04   -4.167117e-03    1.000000e+00    4.794878e-01       1
  3    2.540054e-02    3.292274e-04   -4.803380e-04    1.000000e+00    4.175682e-01       1
  4    2.531315e-02    2.126925e-04   -1.051142e-04    1.000000e+00    3.457821e-01       1
  5    2.528462e-02    1.294037e-04   -3.363246e-05    1.000000e+00    2.779269e-01       1
  6    2.527144e-02    8.277619e-05   -1.609133e-05    1.000000e+00    2.167843e-01       2
  7    2.526596e-02    5.237808e-05   -6.766335e-06    1.000000e+00    1.733833e-01       2
  8    2.526351e-02    3.010859e-05   -3.128623e-06    1.000000e+00    1.379207e-01       3
  9    2.526253e-02    1.705102e-05   -1.240443e-06    1.000000e+00    1.045682e-01       3
 10    2.526211e-02    7.723877e-06   -5.885363e-07    1.000000e+00    7.869184e-02       4
 11    2.526202e-02    3.196279e-06   -1.238683e-07    1.000000e+00    5.296295e-02       5
 12    2.526201e-02    9.984804e-07   -2.588686e-08    1.000000e+00    3.407037e-02       5
 13    2.526200e-02    2.575648e-07   -3.258643e-09    1.000000e+00    1.904251e-02       6
 14    2.526200e-02    4.774290e-08   -2.523342e-10    1.000000e+00    9.671587e-03       7
 15    2.526200e-02    5.902901e-09   -1.158380e-11    1.000000e+00    4.163983e-03       8
 16    2.526200e-02    4.785962e-10   -3.146083e-13    1.000000e+00    1.464156e-03       9
Norm of the gradient less than tolerance
Inexact Newton CG converged in  16 nonlinear iterations and  60 linear iterations.
Final norm of the gradient 4.785961950203523e-10
Value of the cost functional 0.02526200355791334
```

```
running beta = 0.001
 It          cost           ||g||       (g,m_hat)        alpha_m         tol_cg   cg_it
  0    1.025148e-01    1.655646e-03   -2.910938e-01    1.000000e+00    5.000000e-01       1
  1    3.153986e-02    7.026480e-04   -1.300344e-01    1.000000e+00    5.000000e-01       1
  2    2.597919e-02    5.794385e-04   -9.958353e-03    1.000000e+00    5.000000e-01       1
  3    2.543360e-02    4.244306e-04   -8.699766e-04    1.000000e+00    4.587311e-01       1
  4    2.530929e-02    3.026032e-04   -1.492125e-04    1.000000e+00    3.926069e-01       1
  5    2.526823e-02    2.069650e-04   -4.715712e-05    1.000000e+00    3.315059e-01       1
  6    2.525119e-02    1.403526e-04   -1.920532e-05    1.000000e+00    2.741593e-01       1
  7    2.523944e-02    1.145887e-04   -1.394584e-05    1.000000e+00    2.257692e-01       2
  8    2.523304e-02    7.542294e-05   -7.926084e-06    1.000000e+00    2.039978e-01       3
  9    2.522980e-02    5.099231e-05   -3.954495e-06    1.000000e+00    1.655031e-01       3
 10    2.522823e-02    3.119404e-05   -1.971330e-06    1.000000e+00    1.360839e-01       4
 11    2.522739e-02    1.811908e-05   -1.148228e-06    1.000000e+00    1.064364e-01       6
 12    2.522716e-02    8.134229e-06   -3.183610e-07    1.000000e+00    8.111900e-02       8
 13    2.522711e-02    3.558860e-06   -7.077675e-08    1.000000e+00    5.435165e-02       5
 14    2.522710e-02    1.306157e-06   -2.132864e-08    1.000000e+00    3.595092e-02       9
 15    2.522709e-02    4.151282e-07   -3.783174e-09    1.000000e+00    2.177972e-02       9
 16    2.522709e-02    1.145802e-07   -4.723640e-10    1.000000e+00    1.227851e-02      12
 17    2.522709e-02    2.718568e-08   -4.196412e-11    1.000000e+00    6.450737e-03      13
 18    2.522709e-02    3.126393e-09   -1.962066e-12    1.000000e+00    3.142133e-03      14
 19    2.522709e-02    1.635914e-10   -2.460703e-14    1.000000e+00    1.065556e-03      16
Norm of the gradient less than tolerance
Inexact Newton CG converged in  19 nonlinear iterations and  111 linear iterations.
Final norm of the gradient 1.63591382534743e-10
Value of the cost functional 0.025227094597085473
running beta = 0.0001
 It          cost           ||g||       (g,m_hat)        alpha_m         tol_cg   cg_it
  0    1.282146e-01    1.878333e-03   -2.515352e-01    1.000000e+00    5.000000e-01       1
  1    3.709647e-02    8.246034e-04   -1.607026e-01    1.000000e+00    5.000000e-01       1
  2    2.629233e-02    6.699870e-04   -1.990331e-02    1.000000e+00    5.000000e-01       1
  3    2.551796e-02    5.107932e-04   -1.287235e-03    1.000000e+00    4.932734e-01       1
  4    2.532907e-02    3.760710e-04   -2.397637e-04    1.000000e+00    4.307022e-01       1
  5    2.527585e-02    2.809804e-04   -6.090541e-05    1.000000e+00    3.695638e-01       1
  6    2.525275e-02    2.083764e-04   -2.591737e-05    1.000000e+00    3.194424e-01       1
  7    2.523666e-02    2.093514e-04   -1.888671e-05    1.000000e+00    2.750925e-01       2
  8    2.522714e-02    1.436948e-04   -1.160051e-05    1.000000e+00    2.757354e-01       3
  9    2.522229e-02    9.157339e-05   -5.950084e-06    1.000000e+00    2.284415e-01       4
 10    2.521969e-02    6.660670e-05   -3.183372e-06    1.000000e+00    1.823640e-01       5
 11    2.521833e-02    4.225752e-05   -1.683384e-06    1.000000e+00    1.555297e-01       6
 12    2.521755e-02    2.741449e-05   -1.064802e-06    1.000000e+00    1.238815e-01      12
 13    2.521730e-02    1.674160e-05   -3.109607e-07    1.000000e+00    9.978026e-02       5
 14    2.521718e-02    9.733531e-06   -1.573353e-07    1.000000e+00    7.797457e-02      11
 15    2.521713e-02    5.256972e-06   -5.468271e-08    1.000000e+00    5.945518e-02       9
 16    2.521712e-02    2.385232e-06   -2.298196e-08    1.000000e+00    4.369406e-02      13
 17    2.521711e-02    8.928081e-07   -6.018202e-09    1.000000e+00    2.943200e-02      11
 18    2.521711e-02    3.037183e-07   -1.200749e-09    1.000000e+00    1.800668e-02      17
 19    2.521711e-02    1.221779e-07   -1.762825e-10    1.000000e+00    1.050243e-02      18
 20    2.521711e-02    3.588898e-08   -2.043880e-11    1.000000e+00    6.661176e-03      18
 21    2.521711e-02    4.606343e-09   -1.406799e-12    1.000000e+00    3.610232e-03      20
 22    2.521711e-02    4.499865e-10   -3.599924e-14    1.000000e+00    1.293399e-03      26
Norm of the gradient less than tolerance
Inexact Newton CG converged in  22 nonlinear iterations and  187 linear iterations.
Final norm of the gradient 4.499865291416075e-10
Value of the cost functional 0.025217111584725067
```

With decreasing $\beta$, the number of Newton iterations and the cumulative number of conjugate gradient iterations increase. From problem 2, we saw that smaller values of $\beta$ causes ill-conditioning, so the Newton method requires more iterations for convergence (with the benefit of preserving edges better).