Computational and Variational Methods for Inverse Problems - Homework 4 Solution

Mohammad Afzal Shadab (ms82697)

mashadab@utexas.edu

Due Monday, November 24, 2021

---

1 Frequency-domain inverse wave propagation

The inverse problem is formulated as follows

$$\min_m \mathcal{J}(m) = \frac{1}{2}\sum_i^{N_f}\sum_j^{N_s}\sum_k^{N_d}\int_\Omega (u_{ij}(m) - u_{ij}^{obs})^2\delta(x-x_k)dx + \frac{\beta}{2}\int_\Omega |\nabla m|^2 dx \tag{1.1}$$

where N_f, N_s, and N_d are the numbers of frequencies, sources and receivers respectively. $u_{ij}^{obs}(\boldsymbol{x})$ denotes the given measurements (for frequency i and source j), u_{ij} is the acoustic wavefield and $\beta > 0$ is the regularization parameter.

Here $u_{ij}(\boldsymbol{x})$ depends on the medium parameter field $m(\boldsymbol{x}) = 1/c(\boldsymbol{x})^2$ through the solution of the Helmholtz problem

$$-\Delta u_{ij} - \omega_i^2 m u_{ij} = f_j \qquad \text{in } \Omega \tag{1.2}$$
$$u_{ij} = 0 \qquad \text{on } \partial\Omega \tag{1.3}$$

We use the fact that $m \in H^1(\Omega)$.

1.1 Infinite dimensional gradient

For single source and single frequency, equation (1.1) can be simplified as

$$\min_m \mathcal{J}(m) = \frac{1}{2}\sum_{k=1}^{N_d}\int_\Omega (u(m) - u^{obs})^2\delta(\boldsymbol{x}-\boldsymbol{x}_k)\,\mathrm{d}\boldsymbol{x} + \frac{\beta}{2}\int_\Omega |\nabla m|^2\,\mathrm{d}\boldsymbol{x} \tag{1.4}$$

subject to forward PDE problem

$$-\Delta u - \omega^2 m u = f \qquad \text{in } \Omega \tag{1.5}$$
$$u = 0 \qquad \text{on } \partial\Omega \tag{1.6}$$

Let's first derive the weak form of the PDE by multiplying with a test function $p \in H_0^1(\Omega)$

$$\int_\Omega (-\Delta u - \omega^2 m u - f)p\,\mathrm{d}\boldsymbol{x} = 0 \tag{1.7}$$

Now using integration by parts in the laplace term

$$\int_\Omega (\nabla u\cdot\nabla p - \omega^2 m u p - f p)\,\mathrm{d}\boldsymbol{x} - \int_{\partial\Omega} p\nabla u\cdot n\,\mathrm{d}s = 0 \tag{1.8}$$

Now, using the boundary condition $p = 0$ on $\partial\Omega$, we get the weak form

$$\text{Find } u \in H_0^1(\Omega): \int_\Omega (\nabla u\cdot\nabla p - \omega^2 m u p - f p)\,\mathrm{d}\boldsymbol{x} = 0 \quad \forall p \in H_0^1(\Omega) \tag{1.9}$$

Since we want to derive the gradient, we define a Lagrangian functional using inner problem's objective function and weak form of PDE, i.e. summate equations (2.8) and (1.9)

$$\mathcal{L}(u,p,m) = \frac{1}{2}\sum_{k=1}^{N_d}\int_\Omega (u(m)-u^{obs})^2\delta(x-x_k)\,\mathrm{d}\boldsymbol{x} + \frac{\beta}{2}\int_\Omega |\nabla m|^2\,\mathrm{d}\boldsymbol{x} + \int_\Omega (\nabla u\cdot\nabla p - \omega^2 mup - fp)\,\mathrm{d}\boldsymbol{x} \tag{1.10}$$

Now, we wish to derive the gradient of $\mathcal{J}$ with respect to $o \in \{u,p,m\}$. Using the calculus of variations the gradient is $o \to o + \epsilon\hat{o}$ and performing $\mathrm{d}/\mathrm{d}\epsilon$ with $\epsilon \to 0$, we get the weak form of the gradient as summarized below

(i) Forward problem:

$$\boxed{\text{Find } u \in H_0^1(\Omega) : \delta_p\mathcal{L}(p;\hat{p}) = \int_\Omega (\nabla u\cdot\nabla\hat{p} - \omega^2 mu\hat{p} - f\hat{p})\,\mathrm{d}\boldsymbol{x} = 0 \quad \forall\hat{p}\in H_0^1(\Omega)} \tag{1.11}$$

For the strong form of the gradient, we perform integration by parts in the first term of (1.11) which finally yields the strong form of the gradient using arbitrariness of the test function $\hat{p}$

$$\boxed{\begin{aligned} -\Delta u - \omega^2 mu &= f \quad \text{in } \Omega \qquad (1.12)\\ u &= 0 \quad \text{on } \partial\Omega \qquad (1.13)\end{aligned}}$$

(ii) Adjoint problem:

$$\boxed{\text{Find } p \in H_0^1(\Omega) : \delta_u\mathcal{L}(u;\hat{u}) = \int_\Omega \sum_{k=1}^{N_d}\hat{u}(u-u^{obs})\delta(\boldsymbol{x}-\boldsymbol{x}_k)\,\mathrm{d}\boldsymbol{x} + \int_\Omega \nabla p\cdot\nabla\hat{u} - \omega^2 mp\hat{u}\,\mathrm{d}\boldsymbol{x} = 0 \quad \forall\hat{u}\in H_0^1(\Omega)} \tag{1.14}$$

Integrating the weak form by parts yields

$$\int_\Omega \left(\sum_{k=1}^{N_d}(u-u^{obs})\delta(\boldsymbol{x}-\boldsymbol{x}_k) - \Delta p - \omega^2 mp\right)\hat{u}\,\mathrm{d}\boldsymbol{x} = 0 \tag{1.15}$$

where the boundary term vanishes due to the boundary condition on $\hat{u} = 0$ on $\partial\Omega$. Using the arbitrariness of $\hat{u}$ we get the strong form

$$\boxed{\begin{aligned} -\Delta p - \omega^2 mp = -\sum_{k=1}^{N_d}(u-u^{obs})\delta(x-x_k) \quad &\text{in } \Omega \qquad (1.16)\\ p = 0 \quad &\text{on } \partial\Omega \qquad (1.17)\end{aligned}}$$

(iii) Gradient w.r.t. m:

$$\boxed{\delta_m\mathcal{L}(m;\hat{m}) = \beta\int_\Omega \nabla m\cdot\nabla\hat{m} - \omega^2 up\hat{m}\,\mathrm{d}\boldsymbol{x} \quad \forall\hat{m}\in H^1(\Omega)} \tag{1.18}$$

For the strong form of the gradient, we perform integration by parts in the first term of (1.18) and get

$$\delta_m\mathcal{L}(m;\hat{m}) = \int_\Omega \hat{m}(-\beta\Delta m - \omega^2 up)\,\mathrm{d}\boldsymbol{x} + \beta\int_{\partial\Omega}\hat{m}\nabla m\cdot n\,\mathrm{d}s \tag{1.19}$$

which finally yields the strong form of the gradient using the arbitrariness of the test function

$$\boxed{D_m\mathcal{J} = \begin{cases} -\beta\Delta m - \omega^2 up & \text{in } \Omega \\ \beta\nabla m\cdot n & \text{on } \partial\Omega\end{cases}} \tag{1.20}$$

1.2 Infinite dimensional Hessian action

For deriving the Hessian action in direction $\tilde{m}$, let's define a new Lagrangian to enforce the forward and adjoint equations

$$\mathcal{L}^H(u,p,m,\tilde{u},\tilde{p},\tilde{m}) = \delta_m\mathcal{L}(\tilde{m}) + \delta_p\mathcal{L}(\tilde{p}) + \delta_u\mathcal{L}(\tilde{u}) \tag{1.21}$$

By substituting the gradient expressions from (1.11), (1.14) and (1.18) above, we get

$$\begin{aligned}\mathcal{L}^H(u,p,m,\tilde{u},\tilde{p},\tilde{m}) = \beta\int_\Omega \nabla m\cdot\nabla\tilde{m} - \omega^2\tilde{m}up\,\mathrm{d}\boldsymbol{x} + \int_\Omega \nabla u\cdot\nabla\tilde{p} - \omega^2mu\tilde{p} - f\tilde{p}\,\mathrm{d}\boldsymbol{x}\\ + \int_\Omega\sum_{k=1}^{N_d}\tilde{u}(u-u^{obs})\delta(\boldsymbol{x}-\boldsymbol{x}_k) + \nabla u\cdot\nabla p - \omega^2m\tilde{u}p\,\mathrm{d}\boldsymbol{x}\end{aligned} \tag{1.22}$$

We wish to derive the action of the Hessian in direction $\tilde{m}$. Using the calculus of variations the gradient is $m\to m+\epsilon\hat{m}$ and performing $\mathrm{d}/\mathrm{d}\epsilon$ with $\epsilon\to 0$, we get the weak form of the Hessian action $\mathcal{H}(\tilde{m})\hat{m} = \delta_m\mathcal{L}^H$

$$\boxed{\mathcal{H}(\tilde{m})\hat{m} = \beta\int_\Omega \nabla\tilde{m}\cdot\nabla\hat{m} - \omega^2\hat{m}u\tilde{p} - \omega^2\hat{m}\tilde{u}p\,\mathrm{d}\boldsymbol{x} \quad \forall\hat{m}\in H^1(\Omega)} \tag{1.23}$$

For the strong form, again using the integration by parts in the first term, which gives

$$\mathcal{H}(\tilde{m})\hat{m} = \int_\Omega \hat{m}(-\beta\Delta\tilde{m} - \omega^2u\tilde{p} - \omega^2\tilde{u}p)\,\mathrm{d}\boldsymbol{x} + \int_\Omega \hat{m}\beta\nabla\tilde{m}\cdot n\,\mathrm{d}s \tag{1.24}$$

Finally, we get the strong form

$$\boxed{\mathcal{H}(\tilde{m}) = \begin{cases}-\beta\Delta\tilde{m} - \omega^2u\tilde{p} - \omega^2\tilde{u}p & \text{in } \Omega\\ \beta\nabla\tilde{m}\cdot n & \text{on } \partial\Omega\end{cases}} \tag{1.25}$$

1.3 Infinite dimensional gradient for general case

For a gradient expression for arbitrary number of sources and frequencies, we use the Lagrangian by implementing the original form of (1.1) and (1.2) in weak form,

$$\begin{aligned}\mathcal{L}(\{u_{ij}\},\{p_{ij}\},m) = \frac{1}{2}\sum_i^{N_f}\sum_j^{N_s}\sum_k^{N_d}\int_\Omega (u_{ij}(m)-u_{ij}^{obs})^2\delta(\boldsymbol{x}-\boldsymbol{x}_k)\,\mathrm{d}\boldsymbol{x} + \frac{\beta}{2}\int_\Omega|\nabla m|^2\,\mathrm{d}\boldsymbol{x}\\ + \sum_i^{N_f}\sum_j^{N_s}\int_\Omega \nabla u_{ij}\cdot\nabla p_{ij} - \omega_i^2mu_{ij}p_{ij} - f_jp_{ij}\,\mathrm{d}\boldsymbol{x}\end{aligned} \tag{1.26}$$

It is easy to observe that there are $N_f\times N_s$ constraints on PDE with each having the same form.

Now, again we use calculus of variations to find the first variations for each i and j in weak form:

(i) Forward problem: The weak form is

$$\boxed{\text{Find } u_{ij}\in H_0^1(\Omega):\ \delta_p\mathcal{L}(p_{ij};\hat{p}_{ij}) = \int_\Omega(\nabla u_{ij}\cdot\nabla\hat{p}_{ij} - \omega_i^2m\hat{p}_{ij} - f_j\hat{p}_{ij})\,\mathrm{d}\boldsymbol{x} = 0 \quad \forall\hat{p}_{ij}\in H_0^1(\Omega)} \tag{1.27}$$

Using similar approach as part 1.1 and 1.2, the strong form is

$$\boxed{\begin{aligned}-\Delta u_{ij} - \omega_i^2mu_{ij} &= f_j && \text{in } \Omega \quad (1.28)\\ u_{ij} &= 0 && \text{on } \partial\Omega \quad (1.29)\end{aligned}}$$

(ii) Adjoint problem: The weak form is

$$\boxed{\begin{aligned}\text{Find } p_{ij} \in H_0^1(\Omega) : \delta_u \mathcal{L}(u_{ij}; \hat{u}_{ij}) = \int_\Omega \sum_{k=1}^{N_d} \hat{u}_{ij}(u_{ij} - u_{ij}^{obs})\delta(\boldsymbol{x} - \boldsymbol{x}_k)\, \mathrm{d}\boldsymbol{x} + \\ \int_\Omega \nabla p_{ij} \cdot \nabla \hat{u}_{ij} - \omega_i^2 m p_{ij} \hat{u}_{ij}\, \mathrm{d}\boldsymbol{x} = 0 \quad \forall \hat{u}_{ij} \in H_0^1(\Omega)\end{aligned}} \tag{1.30}$$

The corresponding strong form is

$$\boxed{\begin{aligned} -\Delta p_{ij} - \omega_i^2 m p_{ij} &= -\sum_{k=1}^{N_d}(u_{ij} - u_{ij}^{obs})\delta(\boldsymbol{x} - \boldsymbol{x}_k) \qquad \text{in } \Omega \quad (1.31)\\ p &= 0 \qquad \text{on } \partial\Omega \quad (1.32)\end{aligned}}$$

(iii) Gradient w.r.t. m: The weak form is

$$\boxed{\delta_m \mathcal{L}(m; \hat{m}) = \beta \int_\Omega \nabla m \cdot \nabla \hat{m}\, \mathrm{d}\boldsymbol{x} - \int_\Omega \sum_i \sum_j \omega_i^2 u_{ij} p_{ij} \hat{m}\, \mathrm{d}\boldsymbol{x} \quad \forall \hat{m} \in H^1(\Omega)} \tag{1.33}$$

The corresponding strong form is

$$\boxed{D_m \mathcal{J} = \begin{cases} -\beta \Delta m - \sum_{i=1}^{N_f} \sum_{j=1}^{N_s} \omega_i^2 u_{ij} p_{ij} & \text{in } \Omega \\ \beta \nabla m \cdot n & \text{on } \partial\Omega \end{cases}} \tag{1.34}$$

Number of computations: For a single gradient computation, $N_f \times N_s$ state (forward) equations and $N_f \times N_s$ adjoint equations will be solved.

2 Inverse advection-diffusion-reaction problem

Here we consider an inverse problem for advection-diffusion-reaction equation, on the domain $\Omega = [0,1] \times [0,1]$:

$$\min_{m \in H^1(\Omega)} \Phi(m) := \frac{1}{2} \int_\Omega (u(m) - u^{obs})^2\, \mathrm{d}\boldsymbol{x} + \frac{\beta}{2} \int_\Omega |\nabla m|^2\, \mathrm{d}\boldsymbol{x} \tag{2.1}$$

subject to the PDE constraint

$$-\nabla \cdot (k \nabla u) + \boldsymbol{v} \cdot \nabla u + 100 \exp(m) u^3 = f \quad \text{in } \Omega \tag{2.2}$$

$$u = 0 \quad \text{on } \Omega \tag{2.3}$$

We take $f = \max\{0.5, \exp(-25(x-0.7)^2 - 25(y-0.7)^2)\}$, $k = 1$ and $\boldsymbol{v} = (1,0)^T$. The "true" reaction coefficient field m is defined as

$$m(x,y) = \begin{cases} 4 & \text{if } (x-0.5)^2 + (y-0.5)^2 < 0.2^2 \\ 8 & \text{otherwise} \end{cases} \tag{2.4}$$

Data is generated by adding noise of standard deivation of 0.01 in true values of the parameters.

Let's first define the Lagrangian

$$\mathcal{L}(u,p,m) = \frac{1}{2} \int_\Omega (u(m) - u^{obs})^2\, \mathrm{d}\boldsymbol{x} + \frac{\beta}{2} \int_\Omega |\nabla m|^2\, \mathrm{d}\boldsymbol{x} + \int_\Omega k \nabla u \cdot \nabla p + p \boldsymbol{v} \cdot \nabla u + 100 \exp(m) u^3 p - f p\, \mathrm{d}\boldsymbol{x}$$

where the weak form of the PDE has been derived from integration by parts against p, and the boundary terms are nil due to the homogeneous boundary condition. We then proceed to derive the gradient as follows

1. Forward problem:

$$\text{Find } u \in H_0^1(\Omega) : \delta_p \mathcal{L} = \int_\Omega k\nabla u \cdot \nabla \hat{p} + \hat{p}\boldsymbol{v} \cdot \nabla u + 100\exp(m)u^3\hat{p} - f\hat{p}\,\mathrm{d}\boldsymbol{x} = 0 \quad \forall \hat{p} \in H_0^1(\Omega) \tag{2.5}$$

2. Adjoint problem:

$$\text{Find } p \in H_0^1(\Omega) : \delta_u \mathcal{L} = \int_\Omega \hat{u}(u - u^{obs})\,\mathrm{d}\boldsymbol{x} + \int_\Omega k\nabla\hat{u} \cdot \nabla p + pv \cdot \nabla\hat{u} + 300\exp(m)u^2p\hat{u}\,\mathrm{d}\boldsymbol{x} = 0 \quad \forall \hat{u} \in H_0^1(\Omega) \tag{2.6}$$

3. Gradient evaluation w.r.t. m:

$$\mathcal{G}(m;\hat{m}) = \delta_m \mathcal{L} = \beta \int_\Omega \nabla m \cdot \nabla \hat{m} + \int_\Omega 100\exp(m)\hat{m}u^3p\,\mathrm{d}\boldsymbol{x} \tag{2.7}$$

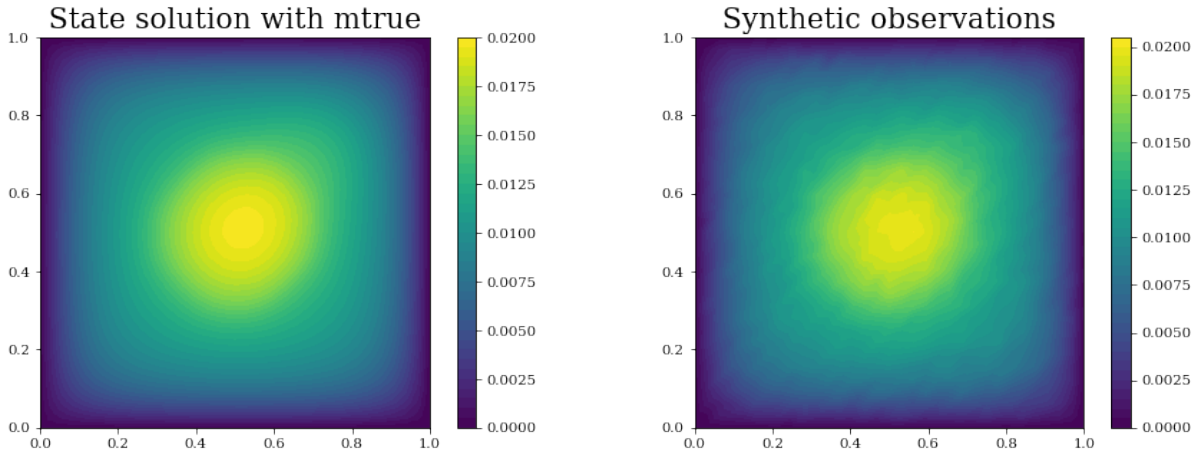

Figure 1: The true solution and noisy solution after adding 0.01% noise.

The jupyter notebook 'HW4_Q2.ipynb' is provided for the solution codes.

2.1 Solution of the inverse problem (H^1 regularization)

(a) $\beta = 10^{-8}$, $m(\boldsymbol{x}) = 8$

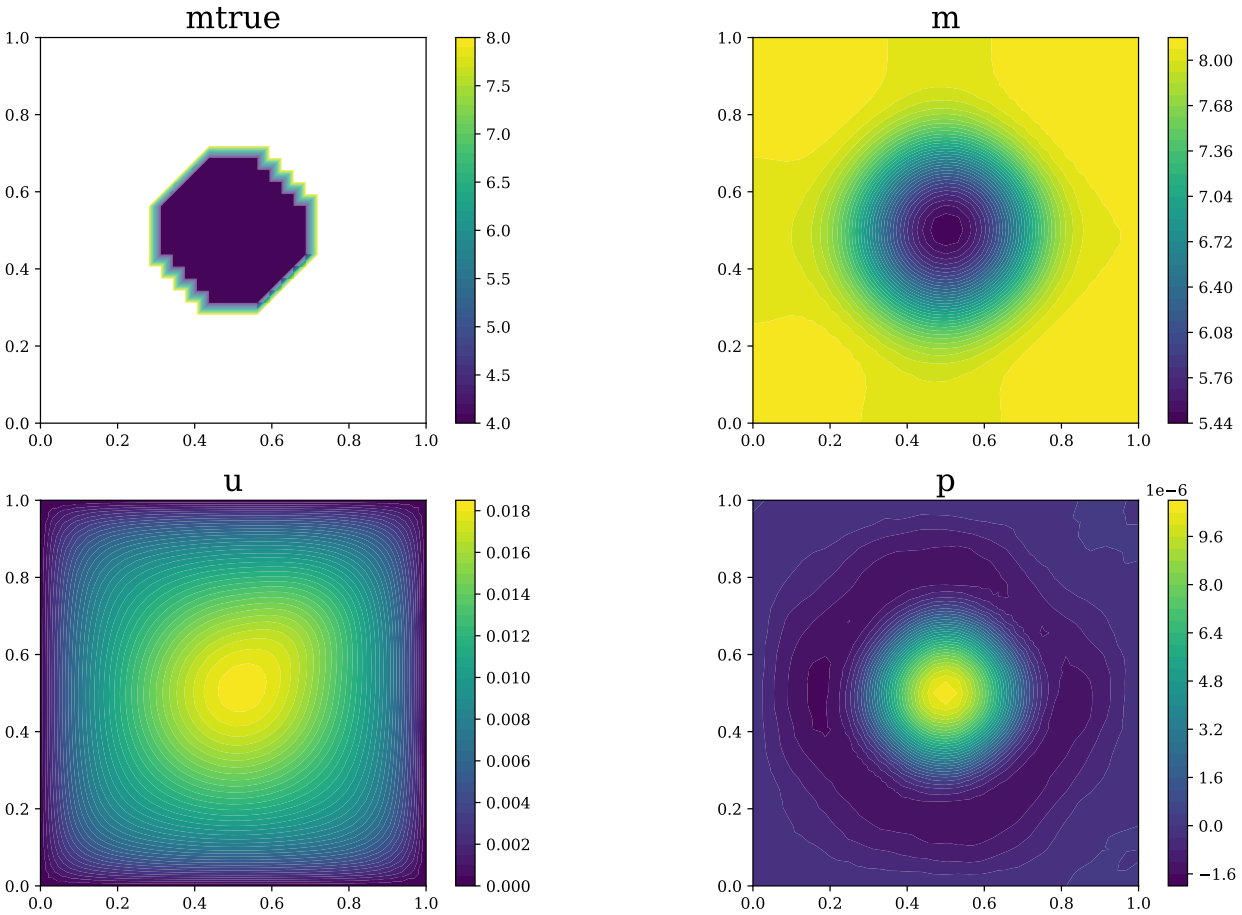

Figure 2: The inversion results for the case $\beta = 10^{-8}$, $m(\boldsymbol{x}) = 8$.

The steepest descent does not converge in 1000 iterations. The parameters are recovered reasonably well. The inverted m is diffused due to Tikhonov (H^1) regularization.

(b) $\beta = 0$**,** $m(\boldsymbol{x}) = 8$

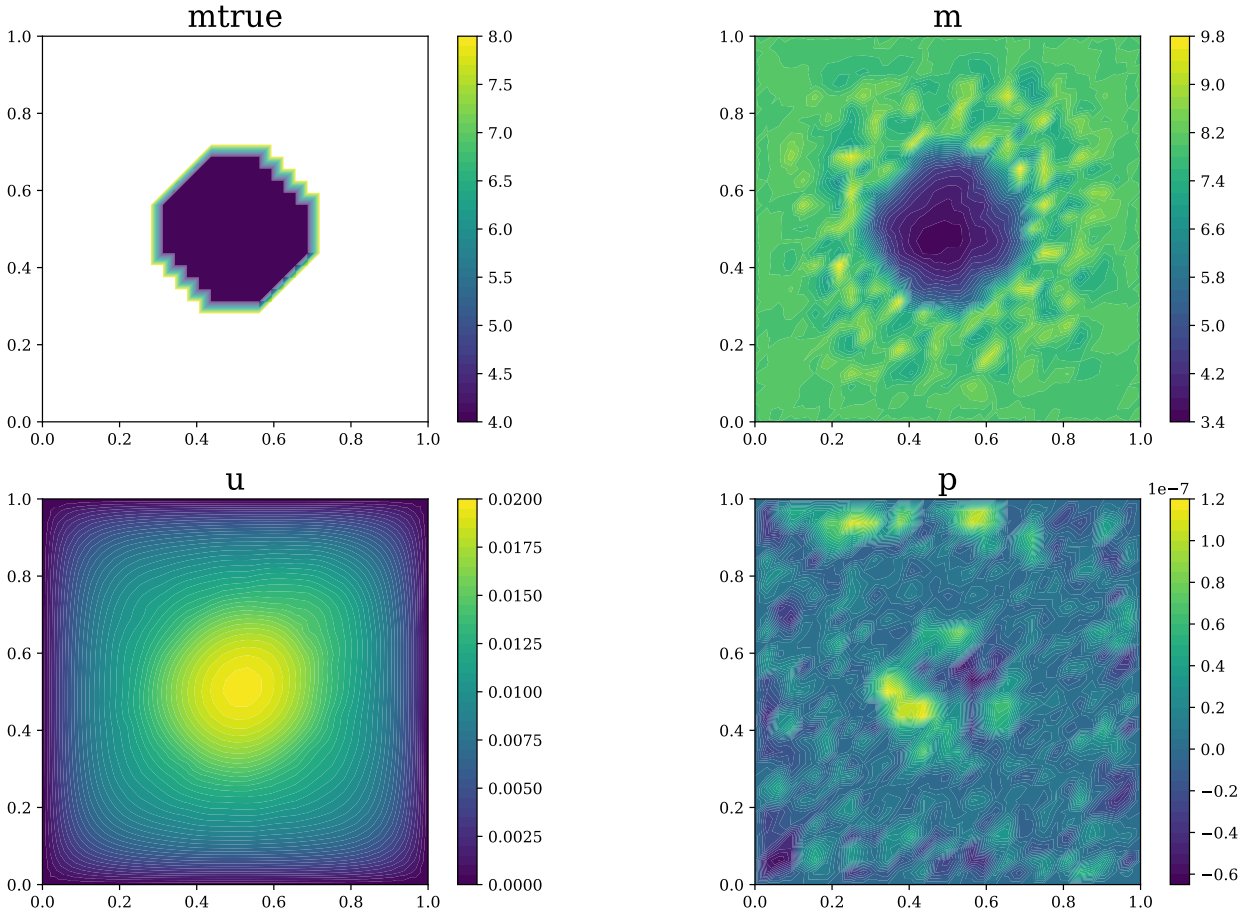

Figure 3: The inversion results for the case $\beta = 0$, $m(\boldsymbol{x}) = 8$.

The steepest descent does not converge in 1000 iterations. The parameters are recovered reasonably well. But the inverted m and p have higher oscillations compared with the case (a) $\beta = 0$ as there is no regularization in this case. The maximum m goes to 9.8 compared with 8.2 for $\beta = 10^{-8}$. Also, the range of inverted m in this case has increased, i.e., $3.4 - 9.8$.

(c) $\beta = 10^{-8}$**,** $m(\boldsymbol{x}) = 4$

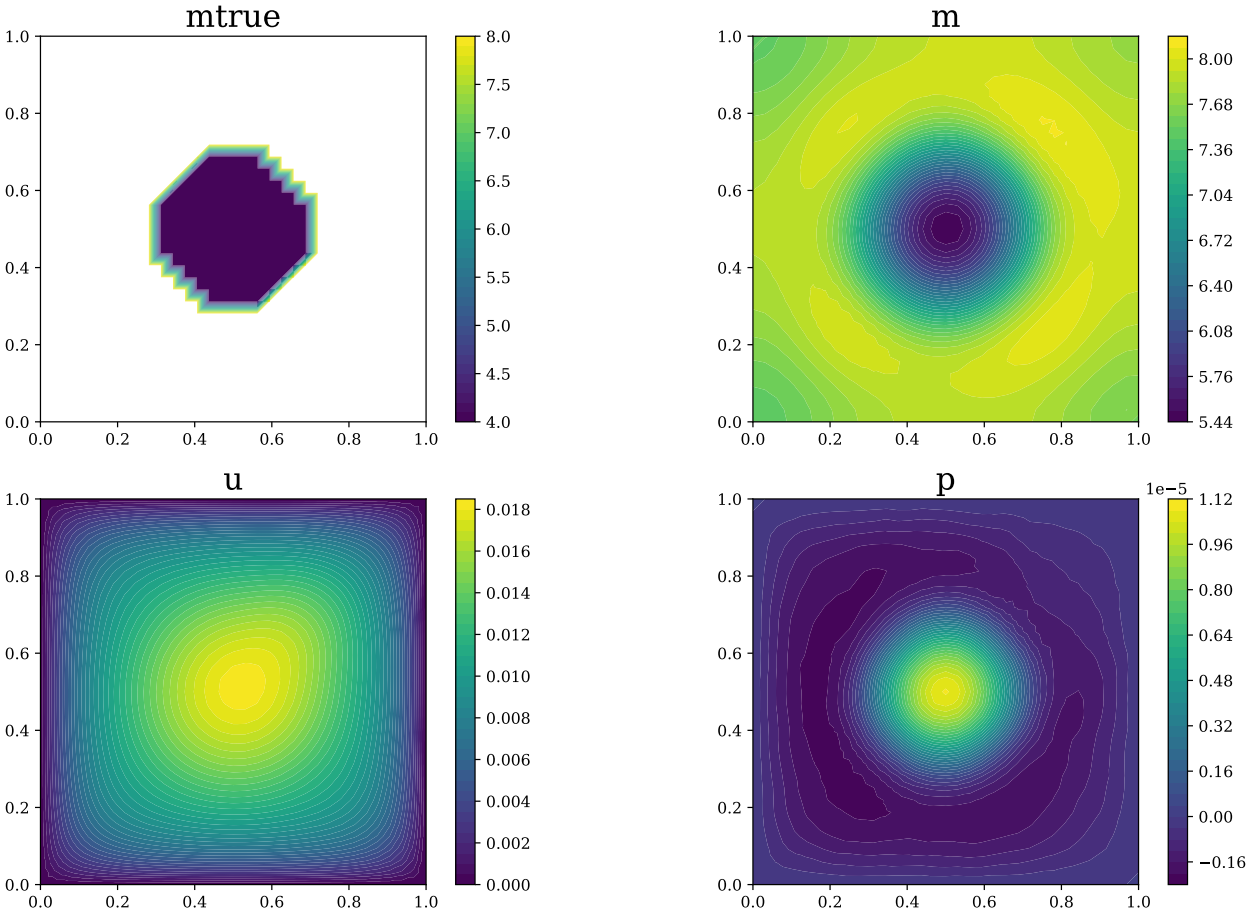

Figure 4: The inversion results for the case $\beta = 10^{-8}$, $m(\boldsymbol{x}) = 4$.

The steepest descent does not converge in 1000 iterations. The parameters m and p are recovered similar to the case (a). The maximum $m(\boldsymbol{x})$ goes lies in range for $5.44 - 8.2$.

But the issue is the reaction term $100\exp(m)u^3$ which is insignificant due to the coefficient term '$100\exp(4)$' for $m = 4$ compared with '$100\exp(8)$' for $m = 8$. Due to the homogeneous Dirichlet boundary condition $u = 0$, reaction term at the boundary condition is zero and therefore can't be inverted for accurately. So, in this case of inaccurate initialization $m = 4$, the inversion is poor close to the boundaries.

2.2 Solution of the inverse problem (TV regularization)

The TV regularization is implemented instead of H^1 regularization in the Lagrangian

$$\mathcal{R}^{\delta}_{TV} := \beta \int_{\Omega} (\nabla m \cdot \nabla m + \delta)^{1/2} \tag{2.8}$$

with the first variation of the TV functional

$$\delta_u \mathcal{R}_{TV}^{\delta}(m, \tilde{m}) = \beta \int_{\Omega} \frac{\nabla m \cdot \nabla \tilde{m}}{(\nabla m \cdot \nabla m + \delta)^{1/2}} \, \mathrm{d}\boldsymbol{x} \tag{2.9}$$

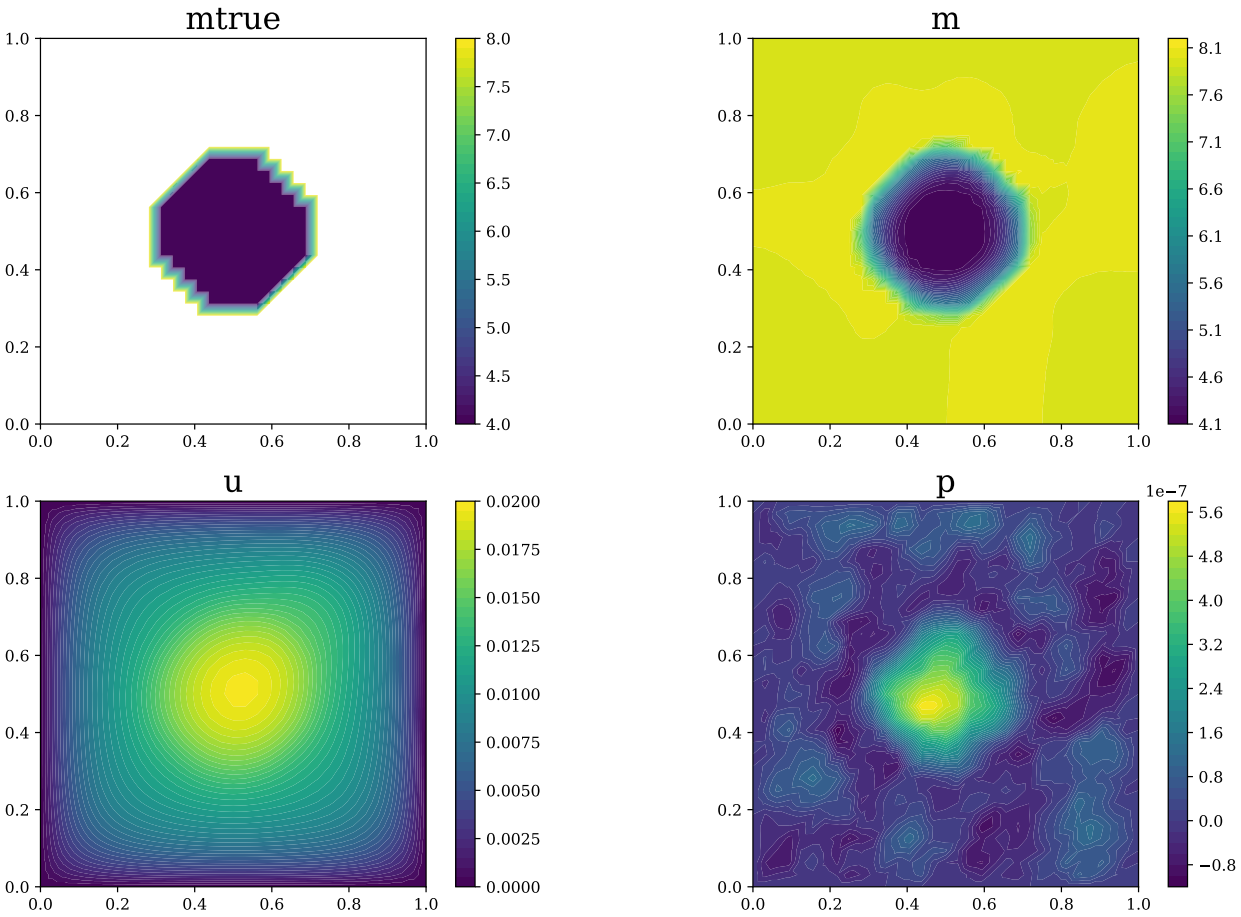

Figure 5: The inversion results for the case $\beta = 10^{-6}$, $m(\boldsymbol{x}) = 8$ using TV regularization.

With $\delta = 0.1$ and $\beta = 10^{-9}$, we initialize with an initial guess of $m = 8$. From the results, it is clear that the true reaction coefficient field is recovered very well with least oscillations. The range of m is well captured, i.e., $4.1 - 8.1$. The β value can be reduced more precisely using Morozov discrepency criterion but I have used visual method to reach to the optimal regularization parameter. Also, the edges are preserved due to preferential diffusion compared with isotropic diffusion of H^1 regularization.

Code starts here:

```
from __future__ import print_function, division, absolute_import

import matplotlib.pyplot as plt
import numpy as np
import dolfin as dl

plt.rcParams.update({'font.family': "Serif"})
plt.rcParams.update({'font.size': 22})

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

TINY_SIZE = 10
SMALL_SIZE = 20
MEDIUM_SIZE = 30
BIGGER_SIZE = 40
plt.rc('font', size=SMALL_SIZE)
plt.rc('axes', titlesize=SMALL_SIZE)
plt.rc('axes', labelsize=SMALL_SIZE)
plt.rc('xtick', labelsize=TINY_SIZE)
plt.rc('ytick', labelsize=TINY_SIZE)
plt.rc('legend', fontsize=SMALL_SIZE)
plt.rc('figure', titlesize=MEDIUM_SIZE)

np.random.seed(seed=1)

def AdrInverseProblem(nx, ny, gamma, noise_level, regularization = 'h1', plotting=False,
                                            starting_location='low', run_name = "test")
                                           :
    np.random.seed(seed=1)
    mesh = dl.UnitSquareMesh(nx, ny)
    Vm = dl.FunctionSpace(mesh, 'Lagrange', 1)
    Vu = dl.FunctionSpace(mesh, 'Lagrange', 2)

    # The true and initial guess for inverted parameter
    mtrue = dl.interpolate(dl.Expression('8. - 4.*(pow(x[0] - 0.5,2) + pow(x[1] - 0.5,2)
                                              < pow(0.2,2) )', degree=5), Vm)

    # define function for state and adjoint
    u = dl.Function(Vu)
    m = dl.Function(Vm)
    p = dl.Function(Vu)

    # define Trial and Test Functions
    u_trial, m_trial, p_trial = dl.TrialFunction(Vu), dl.TrialFunction(Vm), dl.
                                                TrialFunction(Vu)
    u_test,  m_test,  p_test  = dl.TestFunction(Vu),  dl.TestFunction(Vm),  dl.
                                                TestFunction(Vu)

    # initialize input functions
    f = dl.interpolate(dl.Expression('std::max(0.5, exp(-25*(x[0]-0.7)*(x[0]-0.7) - 25 *
                                                (x[1]-0.7)*(x[1]-0.7)))', degree=2), Vu
                                                )
    v  = dl.Constant((1.0, 0.0))#advective velocity vector
    u0 = dl.Constant(0.0)       #BC
    k  = dl.Constant(1.0)       #Diffusion parameter

    # set up dirichlet boundary conditions
    def boundary(x,on_boundary):
        return on_boundary

    bc_state = dl.DirichletBC(Vu, u0, boundary)
    bc_adj   = dl.DirichletBC(Vu, dl.Constant(0.), boundary)

    # weak form of non-linear ADR: Forward problem
    utrue = dl.Function(Vu)

    J_true = dl.inner(k*dl.grad(utrue), dl.grad(u_test))*dl.dx \
        + dl.inner(u_test*v, dl.grad(utrue))*dl.dx \
        + 100.0 * dl.exp(mtrue) * utrue**3 * u_test*dl.dx \
```

```
        - f*u_test*dl.dx

    dl.solve(J_true == 0, utrue, bc_state, solver_parameters={"newton_solver":{"
                                                relative_tolerance":1e-6},
    "newton_solver":{"maximum_iterations":2000}})

    Avarf = dl.inner(u_trial, u_test)*dl.dx
    A_true = dl.assemble(Avarf)

    ud = dl.Function(Vu)
    ud.assign(utrue)

    # perturb state solution and create synthetic measurements ud
    # ud = u + ||u||/SNR * random.normal
    MAX = ud.vector().norm("linf")
    noise = dl.Vector()
    A_true.init_vector(noise,1)
    noise.set_local( noise_level * MAX * np.random.normal(0, 1, len(ud.vector().
                                                get_local())) )
    bc_adj.apply(noise)

    ud.vector().axpy(1., noise)

    if plotting:
        plt.figure(figsize = (15,5))

        plt.subplot(121)
        pl = dl.plot(utrue, title = "State solution with mtrue")
        plt.colorbar(pl)

        plt.subplot(122)
        pl = dl.plot(ud, title = "Synthetic observations")
        plt.colorbar(pl)
#         nb.multi1_plot([utrue, ud], ["State solution with mtrue", "Synthetic
                                            observations"])
        plt.show()

    # Define cost function
    def cost(u, ud, m, gamma):
        # reg = 0.5* gamma * dl.assemble( dl.inner(dl.grad(m), dl.grad(m))*dl.dx )
        reg = 0.5* gamma * dl.assemble(dl.sqrt(dl.inner(dl.grad(m), dl.grad(m)) + dl.
                                                Constant(0.1))*dl.dx)
        misfit = 0.5 * dl.assemble( (u-ud)**2*dl.dx)
        return [reg + misfit, misfit, reg]

    # weak form for setting up the state equation
    J_state = dl.inner(k * dl.grad(u) , dl.grad(p_test)) * dl.dx + \
          dl.inner(v, dl.grad(u)) * p_test * dl.dx + \
          dl.Constant(100)*dl.exp(m) * u * u * u * p_test * dl.dx - \
          f * p_test * dl.dx

    # weak form for adjoint equations
    F_adj = (u - ud) * u_test * dl.dx + k * dl.inner(dl.grad(u_test), dl.grad(p)) * dl.
                                                dx + \
    dl.inner(v, dl.grad(u_test)) * p * dl.dx + dl.Constant(300) * dl.exp(m) * u * u *
                                                u_test * p * dl.dx

    '''
    a_adj = dl.inner(k*dl.grad(p_trial), dl.grad(p_test))*dl.dx \
            + dl.inner(p_trial * v, dl.grad(p_test))* dl.dx \
            + 300.0 * dl.exp(m)*u**2 * p_trial * p_test * dl.dx

    L_adj = - dl.inner(u - ud, p_test) * dl.dx
    '''
    # weak form for gradient
    delta = 0.1
    CTvarf = dl.Constant(100.0) * dl.exp(m)*m_test * u**3 * p * dl.dx

    if regularization == 'h1':
        gradRvarf = dl.Constant(gamma) * dl.inner(dl.grad(m), dl.grad(m_test))*dl.dx

    if regularization == 'tv':
        gradRvarf = dl.Constant(gamma) * dl.inner(dl.grad(m), dl.grad(m_test)) \
```

```
                /dl.sqrt(dl.inner(dl.grad(m), dl.grad(m)) + dl.Constant(delta))*dl.
                                                                    dx

# Mass matrix in parameter space
Mvarf = dl.inner(m_trial, m_test) * dl.dx
M = dl.assemble(Mvarf)

if starting_location == 'low':
    m0 = dl.interpolate(dl.Constant(4.05), Vm)
else:
    m0 = dl.interpolate(dl.Constant(8.), Vm)

m.assign(m0)

# solve state equation
dl.solve(J_state == 0, u, bc_state, solver_parameters={"newton_solver":{"
                                        relative_tolerance":1e-6},
"newton_solver":{"maximum_iterations":2000}})

# evaluate cost
[cost_old, misfit_old, reg_old] = cost(u, ud, m, gamma)

if plotting:
    plt.figure(figsize=(15,5))
    plt.subplot(121)
    dl.plot(m,title="m0", vmin=mtrue.vector().min(), vmax=mtrue.vector().max())
    plt.subplot(122)
    dl.plot(u, title="u(m0)")
    plt.show()

tol = 1e-6
maxiter = 1000
backtrack_maxiter = 20
c_armijo = 1e-5

# initialize iter counters
count = 0
converged = False

# initializations
g = dl.Vector()
M.init_vector(g,0)

m_prev = dl.Function(Vm)

while count <  maxiter and not converged:

    # solve the adoint problem
    dl.solve(F_adj == 0, p, bc_adj, solver_parameters={"newton_solver": {"
                                            relative_tolerance": 1e-6}})

    # evaluate the  gradient
    MG = dl.assemble(CTvarf + gradRvarf)
    dl.solve(M, g, MG)

    # calculate the norm of the gradient
    grad_norm2 = g.inner(MG)
    gradnorm = np.sqrt(grad_norm2)

    if count == 0:
        gradnorm0 = gradnorm

    # linesearch
    it_backtrack = 0
    m_prev.assign(m)
    alpha = 1.e5
    backtrack_converged = False
    for it_backtrack in range(backtrack_maxiter):

        m.vector().axpy(-alpha, g)

        # print("solving forward prob")
        dl.solve(J_state == 0, u, bc_state, solver_parameters={"newton_solver":{"
```

```
                                                    relative_tolerance":1e-8},
            "newton_solver":{"maximum_iterations":4000}})

            # evaluate cost
            [cost_new, misfit_new, reg_new] = cost(u, ud, m, gamma)

            # check if Armijo conditions are satisfied
            if cost_new < cost_old - alpha * c_armijo * grad_norm2:
                cost_old = cost_new
                backtrack_converged = True
                break
            else:
                alpha *= 0.5
                m.assign(m_prev)  # reset a

        if backtrack_converged == False:
            print( "Backtracking failed. A sufficient descent direction was not found" )
            converged = False
            break
        if count%50==0: print("Iteration %d"%count)

        # check for convergence
        if gradnorm < tol*gradnorm0 and count > 0:
            converged = True
            print ("Steepest descent converged in ",count,"  iterations")

        count += 1
    if not converged:
        print ( "Steepest descent did not converge in ", maxiter, " iterations")

    if plotting:
        plt.figure(figsize = (15,10))

        plt.subplot(221)
        pl = dl.plot(mtrue, title = "mtrue")
        plt.colorbar(pl)

        plt.subplot(222)
        pl = dl.plot(m, title = "m")
        plt.colorbar(pl)

        plt.subplot(223)
        pl = dl.plot(u, title = "u")
        plt.colorbar(pl)

        plt.subplot(224)
        pl = dl.plot(p, title = "p")
        plt.colorbar(pl)
        plt.show()

    if plotting:
        plt.figure()
        plt.subplot(121)

    plt.figure(figsize = (15,10))

    plt.subplot(221)
    pl = dl.plot(mtrue, title = "mtrue")
    plt.colorbar(pl)

    plt.subplot(222)
    pl = dl.plot(m, title = "m")
    plt.colorbar(pl)

    plt.subplot(223)
    pl = dl.plot(u, title = "u")
    plt.colorbar(pl)

    plt.subplot(224)
    pl = dl.plot(p, title = "p")
    plt.colorbar(pl)
    plt.savefig("Results/%s.pdf" %(run_name))
```

```
    Mstate = dl.assemble(u_trial*u_test*dl.dx)
    noise_norm2 = noise.inner(Mstate*noise)
    return Vm.dim(), count, noise_norm2, cost_new, misfit_new, reg_new


n = 32
noise_level = 0.01
plotting = True

# Cases - with regularization
beta = 1e-8
casename = "2_1part_a"
AdrInverseProblem(n, n, beta, noise_level, regularization='h1', plotting=plotting,
                                            starting_location='high', run_name=casename)

beta = 0.0
casename = "2_1part_b"
AdrInverseProblem(n, n, beta, noise_level, regularization='h1', plotting=plotting,
                                            starting_location='high', run_name=casename)

beta = 1e-8
casename = "2_1part_c"
AdrInverseProblem(n, n, beta, noise_level, regularization='h1', plotting=plotting,
                                            starting_location='low', run_name=casename)

n = 32
noise_level = 0.01
plotting = True

beta = 1e-9
casename = "2_2_1e-9"
AdrInverseProblem(n, n, beta, noise_level, regularization='tv', plotting=plotting,
                                            starting_location='hi', run_name=casename)
```