Spring 2014:
Computational and Variational Methods for Inverse Problems
CSE 397/GEO 391/ME 397/ORI 397
Assignment 4 (due 14 April 2014)

The first problem in this assignment is a paper-and-pencil exercise in which you work out expressions for the gradient and the Hessian action (in a given direction) for a frequency domain inverse wave propagation problem. This is an industrially-important problem: it's how we image the subsurface in search of energy resources, and how we non-destructively evaluate materials for internal defects.

In the second problem, you will be asked to solve an inverse problem governed by an advection-diffusion equation (this is the prototype inverse problem we discussed in class) using COMSOL. For this assignment we'll use just a steepest descent method; the next assignment will generalize the solver to a Newton method. You will be given an existing implementation of an inverse solver (for a Poisson equation) and will be asked to modify it.

1. Frequency-domain inverse wave propagation problem

Here we formulate and solve an inverse acoustic wave propagation problem in the frequency domain, which is governed by the Helmholtz equation. Let $\Omega \subset \mathbb{R}^d$ be a bounded domain ($d \in \{2, 3\}$). The idea is to propagate harmonic waves from multiple sources $f_j(\boldsymbol{x})$ ($j = 1, \ldots, N_s$) at multiple frequencies ω_i ($i = 1, \ldots, N_f$) into a medium and measure the amplitude $u_{ij}(\boldsymbol{x})$ of the reflected wavefields at points $\boldsymbol{x}_k$ ($k = 1, \ldots, N_d$) for each source and frequency, with the goal of inferring the soundspeed of the medium, $c(\boldsymbol{x})$. We can formulate this inverse problem as follows:

$$\min_m \mathcal{J}(m) := \frac{1}{2} \sum_i^{N_f} \sum_j^{N_s} \sum_k^{N_d} \int_\Omega (u_{ij}(m) - u_{ij}^{obs})^2 \, \delta(\boldsymbol{x} - \boldsymbol{x}_k) \, d\boldsymbol{x} + \frac{\beta}{2} \int_\Omega \nabla m \cdot \nabla m \, d\boldsymbol{x}$$

where $u_{ij}(\boldsymbol{x})$ depends on the medium parameter field $m(\boldsymbol{x})$, which is equal to $1/c(\boldsymbol{x})^2$, through the solution of Helmholtz equations

$$\begin{aligned} -\Delta u_{ij} - \omega_i^2 \, m \, u_{ij} &= f_j \quad \text{in } \Omega, \quad i = 1, \ldots, N_f, \quad j = 1, \ldots, N_s, \\ u_{ij} &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

In the problem above, $u_{ij}^{obs}(\boldsymbol{x})$ denotes given measurements (for frequency i and source j), u_{ij} is the amplitude of the acoustic wavefield, and $\beta > 0$ is the regularization parameter.

1. Derive an expression for the (infinite dimensional) gradient of $\mathcal{J}$ with respect to the medium m using the Lagrangian method for a single source and frequency, i.e., for $N_f = N_s = 1$.

2. Derive an expression for the (infinite dimensional) action of the Hessian in a direction $\tilde{m}$ in the single source and frequency case.

3. Derive an expression for the (infinite dimensional) gradient for an arbitrary number of sources and frequencies.[1] How many state and adjoint equations have to be solved for a single gradient computation?

2. Inverse advection-diffusion problem

We would like to solve the inverse problem for the advection-diffusion equation we discussed in class, on the domain $\Omega = [0,1] \times [0,1]$:

$$\min_m \mathcal{J}(m) := \frac{1}{2}\int_\Omega (u(m) - u^{obs})^2 \, d\boldsymbol{x} + \frac{\beta}{2}\int_\Omega \nabla m \cdot \nabla m \, d\boldsymbol{x}, \tag{1}$$

where the field $u(\boldsymbol{x})$ depends on the diffusivity $m(\boldsymbol{x})$ through the solution of

$$\begin{aligned} -\nabla \cdot (m \nabla u) + \boldsymbol{v} \cdot \nabla u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{2}$$

where $\boldsymbol{v}(\boldsymbol{x})$ is the advective velocity vector with components (v_x, v_y), $\beta > 0$ is the regularization parameter, and $f(\boldsymbol{x})$ is the source term. We synthesize the measurement data $u^{obs}(\boldsymbol{x})$ by solving the forward advection-diffusion equation with $m(x,y) = 4$ for $(x-0.5)^2 + (y-0.5)^2 \leq 0.2^2$ and otherwise $m(x,y) = 8$, for a source $f = 1$. Noise is added to this "data" to simulate actual instrument noise.

On the class website (`http://users.ices.utexas.edu/~omar/inverse_probs`), we provide the COMSOL code `elliptic_sd_ip`, which is an implementation of the steepest descent method[2] for this problem with advective velocity $\boldsymbol{v} = \boldsymbol{0}$ (i.e., for a pure diffusion problem). This code is also described at the end of this document. The COMSOL report that was referenced in Assignment 3 will be useful here as well. Please turn in your modified code along with numerical results for the following:

1. Report the solution of the inverse problem and the number of required iterations for the following cases:

   (a) Noise level of 0.01 (roughly 1% noise), and regularization $\beta = 5 \times 10^{-10}$.

   (b) Same, but with $\beta = 0$, i.e., no regularization[3].

   (c) No noise and $\beta = 0$, and use $m \equiv 4$ and $m \equiv 8$ as intitial guesses for the parameter field m. Do you find the same solution? Explain the behavior.

[1] Hint: Every state equation with solution u_{ij} has a corresponding adjoint variable p_{ij}. The Lagrangian functional involves the sum over all state equations.

[2] Note that we do not want to advocate the use of the steepest descent method for variational inverse problems, since its convergence can be very slow. For all tests, please use a maximum number of 300 iterations—you will need it! In the next assignment, we will solve this problem using an inexact Newton-CG method, which is more efficient.

[3] Inverse problems can also be "iteratively regularized", which means that iterative methods are terminated early in an iterative process. This can allow a reasonable solution also without regularization. The basic idea is that iterative methods such as the conjugate gradient method capture the most important modes determined by the data early in the iterations. A critical issue in iterative regularization is to find appropriate termination criteria such that the important information is found before the iterative method has brought in the noisy modes. See, for instance, the book by Kaipio and Somersalo for more on iterative regularization methods.

2. Add the advective term $\boldsymbol{v} = (30, 0)$ to the inverse problem and its COMSOL implementation and plot the resulting reconstruction of m for a noise level of 0.01 and for a reasonably chosen regularization parameter.

3. Since the coefficient m is discontinuous, a better choice of regularization is total variation rather than Tikhonov regularization, to prevent an overly smooth reconstruction. Modify the implementation and plot the result for a reasonably chosen regularization parameter[4].

Line-by-line description of steepest descent implementation

First, note that the coefficient m is called a in the implementation below. Our implementation starts with specifying the geometry and the mesh (lines 2 and 3), and with defining names for the finite element basis functions, their polynomial order, and the regularization parameter (lines 4–8). In this example, we use linear finite elements functions on a quadrilateral mesh for the coefficient a (and for the gradient), and linear[5] finite element functions for the state u, the adjoint p, and the measured data u_d. For our purposes, the latter are actually synthetic data computed by solving the forward problem given a "truth" coefficient field, `atrue` (defined in line 6).

```
2 fem.geom = rect2(0,1,0,1);
3 fem.mesh = meshmap(fem,'Edgelem',{1,40,2,40});
4 fem.dim = {'a' 'grad' 'p' 'u' 'ud'};
5 fem.shape = [1 1 1 1 1];
6 fem.equ.expr.atrue = '4+4*(sqrt((x-0.5)^2+(y-0.5)^2)>0.2)';
7 fem.equ.expr.f = '1';
8 fem.equ.expr.gamma = '1e-9';
```

Homogeneous Dirichlet boundary conditions for the state[6] and adjoint equations are used. In line 14 the conditions $u = 0$, $p = 0$, and $u_d = 0$ on the boundary $\partial\Omega$ are enforced. If different boundary conditions on the different sides of the domain should be specified, this can be done. The weak form for the state equation with the "truth" coefficient `atrue`, which is used to compute the synthetic measurements, is given in line 15, followed by the state equation with the unknown, to-be-reconstructed coefficient a (line 16). Lines 17–19 define the weak forms of the adjoint and the gradient equations, respectively. Note that in COMSOL Multiphysics, test functions (also known as variations) are denoted by adding "`_test`" to the variable name. Note also that in the code listing, the term "control" is used to refer to the equation that sets the gradient to zero.[7]

```
14 fem.bnd.r = {{'u' 'p' 'ud'}};
15 fem.equ.expr.goal = '-(atrue*(udx*udx_test+udy*udy_test)-f*ud_test)';
16 fem.equ.expr.state = '-(a*(ux*ux_test+uy*uy_test)-f*u_test)';
17 fem.equ.expr.adjoint = '-(a*(px*px_test+py*py_test)-(ud-u)*p_test)';
18 fem.equ.expr.control = ['(grad*grad_test-gamma*(ax*gradx_test '...
19                         '+ay*grady_test)-(px*ux+py*uy)*grad_test)'];
```

[4]Use $\varepsilon = 0.1$ to regularize the non-differentiable TV regularization term; also, use `realsqrt` instead of `sqrt` in weak forms. Do not forget to adjust the cost function appropriately.

[5]We could have used quadratic elements, for instance.

[6]Another term commonly used for the forward equation.

[7]The use of the term "control function" to refer to the inversion parameter a and the term "control equation" to refer to the vanishing-of-the-gradient equation comes from optimal control theory, where the framework for optimization of systems governed by PDEs originated.

To synthesize measurement data, the state equation with the given coefficient `atrue` is solved (lines 20–22).

```
20 fem.equ.weak = 'goal';
21 fem.xmesh = meshextend(fem);
22 fem.sol = femlin(fem, 'Solcomp', {'ud'});
```

COMSOL allows the user to access its internal finite element structures, such as the degrees of freedom for each finite element function. Our implementation of the steepest descent iteration works on the finite element coefficients, and the indices for the degrees of freedom are extracted from the finite element data structure in the lines 23–29. Note that internally the unknowns are ordered alphabetically (independently from the order given in line 4). Thus, the indices for the finite element function a can be extracted from the first column of `dofs`; see line 25. If different order element functions are used in the same finite element structure, COMSOL pads the list of indices for the lower-order function with zeros. These zero indices are removed by only choosing the positive indices (lines 25–29). The index vectors are used to access the entries in `X`, the vector containing all finite element coefficients, which can be accessed as shown in line 30.

```
23 nodes = xmeshinfo(fem ,'out', 'nodes');
24 dofs = nodes.dofs';
25 AI = dofs(dofs(:,1)>0,1);
26 GI = dofs(dofs(:,2)>0,2);
27 PI = dofs(dofs(:,3)>0,3);
28 UI = dofs(dofs(:,4)>0,4);
29 UDI = dofs(dofs(:,5)>0,5);
30 X = fem.sol.u;
```

We add noise to our synthetic data (line 32) to lessen the "inverse crime," which refers to the situation where the same numerical method is used to both create the synthetic data as well as to solve the inverse problem.[8]

```
31 randn('seed', 0);
32 X(UDI) = X(UDI) + datanoise * max(abs(X(UDI))) * randn(length(UDI),1);
```

We initialize the coefficient a (line 33; the initial guess is a constant function) and (re-)define the weak form as the sum of the state, adjoint, and control equations (line 34). Note that since the test functions for these three weak forms differ, one can regain the individual equations by setting the appropriate test functions to zero. To compute the initial value of the cost functional, in line 36 we solve the system with respect to u only. For the variables not solved for, the finite element functions specified in `X` are used. Then, the solution of the state equation is copied into `X`, and is used in the evaluation of the cost functional (lines 37 and 38).

```
33 X(AI) = 8.0;
34 fem.equ.weak = 'state + adjoint + control';
35 fem.xmesh = meshextend(fem);
36 fem.sol = femlin(fem, 'Solcomp', {'u'}, 'U', X);
```

[8]This so-called inverse crime is often committed by those doing theoretical work on inverse problems. Using the same forward model to both compute the synthetic data as well as to solve the inverse problem can make inversion results look better than they would were real data used, or even if synthetic data generated from a different forward model were used, because the effect of modeling error is not accounted for. (Here, *forward model* refers both to a given mathematical model, as well as its discretization.) Nevertheless, synthesizing data is popular when analyzing the accuracy of inversion methods, since one then knows the "truth" parameter field that gave rise to the data.

```
37 X(UI) = fem.sol.u(UI);
38 cost_old = evaluate_cost (fem, X);
```

Next, we iteratively update a in the steepest descent direction. For a current iterate of the coefficient a, we first solve the state equation (line 40) for u. Given the state solution, we solve the adjoint equation (line 42) and compute the gradient from the control equation (line 44). A line search to satisfy the Armijo descent criterion is used (line 56). If, for a step length α, the cost is not sufficiently decreased, backtracking is performed, i.e., the step length is reduced (line 61). In each backtracking step, the state equation has to be solved to evaluate the cost functional (lines 53-54).

```
39 for iter = 1:maxiter
40   fem.sol = femlin(fem, 'Solcomp', {'u'}, 'U', X);
41   X(UI) = fem.sol.u(UI);
42   fem.sol = femlin(fem, 'Solcomp', {'p'}, 'U', X);
43   X(PI) = fem.sol.u(PI);
44   fem.sol = femlin(fem, 'Solcomp', {'grad'}, 'U', X);
45   X(GI) = fem.sol.u(GI);
46   grad2 = postint (fem, 'grad * grad');
47   Xtry = X;
48   alpha = alpha * 1.2;
49   descent = 0;
50   no_backtrack = 0;
51   while (~descent && no_backtrack < 10)
52       Xtry(AI) = X(AI) - alpha * X(GI);
53       fem.sol = femlin(fem, 'Solcomp', {'u'}, 'U', Xtry);
54       Xtry(UI) = fem.sol.u(UI);
55       [cost, misfit, reg] = evaluate_cost (fem, Xtry);
56       if (cost < cost_old - alpha * c * grad2)
57           cost_old = cost;
58           descent = 1;
59       else
60           no_backtrack = no_backtrack + 1;
61           alpha = 0.5 * alpha;
62       end
63   end
64   X = Xtry;
65   fem.sol = femsol(X);
66   if (sqrt(grad2) < tol && iter > 1)
67        fprintf (['Gradient method converged after %d iterations.'...
68                  '\n\n'], iter);
69   break;
70   end
71 end
```

The steepest descent iteration is terminated when the norm of the gradient is sufficiently small.