Chapter 5

Optimization Techniques

5.1 Unconstrained optimization formulation

Maybe the most common approach to parameter estimation problem is the unconstrained optimization approach. Its advantage and disadvantage is in the separation of the solution of the forward and the inverse problem.

Given the constraint (PDE) $c(m, u) = 0$ and a candidate model m, we can "solve" for $u = u(m)$ and then solve the unconstrained optimization problem

$$\min_m \mathcal{J}(m) = \frac{1}{2}\|Qu(m) - d\|^2 + \alpha R(m)$$

A necessary condition for a minimum is

$$\nabla_m \mathcal{J}(m) = J(m)^\top (Qu(m) - d) + \alpha \nabla R(m) = 0 \tag{5.1}$$

that is a nonlinear system of equations for m. We now discuss different solution methodologies for the solution of the problem and discuss their properties.

5.2 Methods that use gradient information only

To solve the nonlinear system Eq. (5.1) we consider first methods that use gradient information only. Maybe the simplest is the steepest descent method, where each iteration can be summarized by the simple following lines

```
function[x] = ucSDOPT(x0,para,maxit)
% [x] = ucSDOPT(x,para)

gamma = 1e-5;
x = x0;
iter = 1;

while 1

    [f,df] = feval(para.obj,x,para);
    s = -para.H\df;
    if iter == 1, muls = 1; f0 = f; n0 = norm(df);end

    fprintf('%3d.0   %3.2e   %3.2e\n',iter,f/f0,norm(df)/n0);

    lsiter = 1;
    % Armijo line search
    while 1
        xt = x + muls*s;
        ft = feval(para.obj,xt,para);
        fprintf('%3d.%d   %3.2e\n',iter,lsiter,ft/f0);

        if ft < f + muls*gamma*s'*df, break; end

        muls = muls/2;
        lsiter = lsiter+1;
        if lsiter > 6, disp('Line Serach Break'); return; end
    end
    x = xt;
    if lsiter == 1, muls = muls*1.5; end
    iter = iter+1;
    if iter > maxit, return; end

end
```

For inverse problems steepest descent is known to slowly converge due to the ill-conditioning. Much better results can be obtained by using the regularizer and change the direction s_k to

$$s_k = -(\nabla_m^2 R(m_k))^{-1} \nabla \mathcal{J}(m_k).$$

For quadratic regularization this implies simply solving at each iteration the PDE

$$(\nabla_m^2 R(m_k))\, s_k = \nabla \mathcal{J}(m_k)$$

which can be done by using a multigrid method.

A better alternative than steepest descent is to use limited memory BFGS [15]. In LBFGS we save a number of the previous steps $\{s_k, s_{k-1}, \ldots, s_{k-\ell}\}$ and previous gradients $\{\nabla \mathcal{J}(m_k), \nabla \mathcal{J}(m_{k-1}), \ldots, \nabla \mathcal{J}(m_{k-\ell})\}$ in order to approximate the inverse of the Hessian. The method has a slightly large storage requirement compared with steepest descent but it typically yields much better convergence.

An important implementation note in LBFGS is that one needs to choose an initial Hessian. For inverse problem *it is crucial* to choose

$$H_0 = \nabla_m^2 R(m_0)$$

for the method to work well. Other quasi-Newton methods can be selected [**?**] for problems that have special structure.

From an implementation point of view, using simple descent methods requires the solution of the forward problem (to evaluate $\mathcal{J}(m)$) and the solution of the

adjoint problem (to evaluate $\nabla_m \mathcal{J}(m)$). In some cases, where these methods tend to work well, they are unbeatable. However, in many cases, using higher order information can lead to better and faster algorithms.

5.3 Gauss-Newton and Newton like methods

Methods that use second order information can also be used for PDE optimization problems. In these methods one generates an approximation to the Hessian and (approximately) solves the linear system

$$\nabla_m^2 \mathcal{J}(m)\, s = -\nabla_m \mathcal{J}(m)$$

Let us first compute the Hessian of the problem. Differentiating Eq. (5.1) with respect to m we obtain

$$\nabla_m^2 \mathcal{J}(m) \;=\; J(m)^\top J(m) + \nabla_m (J(m)^\top r) + \alpha \nabla_m^2 R(m).$$

where $r = Qu - d$ is taken as a constant vector that do not depend on m. The Hessian contains three different parts. Clearly, the term

$$H_{GN}(m) = J(m)^\top J(m) + \alpha \nabla_m^2 R(m)$$

is symmetric positive semidefinite (and likely SPD). The extra term $\nabla_m (J(m)^\top r)$ is symmetric (why?) but may not be positive. It is therefore common to ignore this term in practice.

It is important to note that although the second term tends to be ignored, it is actually possible to use it with relative easy without adding much cost to the problem. This is in contrast to many other least squares problems where the second order terms are ignored because of the cost associated with computing them. We will discuss the computation of the term in the next chapter.

Clearly, using direct methods for the solution of the Gauss-Newton system is out of the question and therefore iterative methods are used. Since the Gauss-Newton system in PSD it is common to use conjugate gradient (CG), conjugate gradient least squares (CGLS) or least square QR (LSQR) for the solution of the problem. If the number of CG iterations is large then it can be rather expensive to solve the problem. It is therefore common to use a fix, large tolerance when approximately solving the linear system. It is rare to set the constant below say 10^{-2} and in most cases, rather ill-advised. Newton like methods with inexact solve of the linear system are often refer to as inexact Newton and Gauss-Newton methods. The convergence of this methods is linear but the constant of the linear convergence is typically much better than the constant of the steepest descent method.

5.4 More on solving the linear systems

As discussed above for any Newton like iteration one requires the solution of the system

$$(J(m)^\top J(m) + \alpha \nabla_m^2 R(m))\, s \;=\; -\nabla_m \mathcal{J}(m).$$

That is done by, say, the CG method. Clearly, the main cost is the computation of the matrix $J(m)$ and a vector and the computation of $J(m)^\top$ and a vector. Since in general, $J(m) = -c_u^{-1}c_m$ the computation of a matrix vector problem is equivalent to the solution of a (linearized) forward problem. It is therefore natural the measure the "cost" of the solution of the linear system by considering how many $J(m)$ and $J(m)^\top$ mat-vecs are computed. If the forward problem is linear with respect to u then the cost is equivalent to the cost of the forward problem.

To reduce the cost associated with the linear system preconditioning is needed. To recall, a preconditioner is a symmetric positive definite matrix M that is used in order to solve an equivalent problem

$$M^{-1}H_{GN}s = M^{-1}\nabla_m \mathcal{J}$$

where the condition number of the preconditioned system $M^{-1}H_{GN}$ is much smaller than the condition number of H_{GN} leading to faster convergence of the conjugate gradient method. The problem of preconditioning the Gauss-Newton system in an effective way is an open research problem. The difficulty in finding an appropriate preconditioner stems from the point that $J(m)$ can be thought of as a (discretization of) an integral operator that is, it is a compact operator with singular values that cluster at 0. To add to this difficulty, the operator $\nabla_m^2 R$ can be thought of as a differential operator with eigenvalues that cluster at infinity. Dealing with this combination has proved to be a tough problem.

We now briefly discuss a number of techniques that have been used with some success to precondition the system.

- Maybe the most obvious preconditioner that is used is to simply use the regularization term as a preconditioner, that is, setting $M = \nabla_m^2 R(m)$. Since $\nabla_m^2 R(m)$ is sparse and can be considered as a discretization of a PDE, multigrid methods can be often used for the solution of the preconditioner system.

- A more complex preconditioner (at least in terms of implementation) can be obtained by combining quasi-Newton methods with the Gauss-Newton method. Recall that in the L-BFGS method we build an approximation to the inverse of the Hessian using previous steps $\{s_k, s_{k-1}, \ldots, s_{k-\ell}\}$ and previous gradients $\{\nabla\mathcal{J}(m_k), \nabla\mathcal{J}(m_{k-1}), \ldots, \nabla\mathcal{J}(m_{k-\ell})\}$. The inverse of the preconditioner, M^{-1} is supposed to approximate exactly that same quantity. It is therefore possible to use the *Gauss Newton steps* in combination with the previous gradients and build the L-BFGS approximation to the inverse of the Hessian. However, rather than using this inverse in order to obtain a step, we use this inverse to *precondition the linear system*. In my experience this simple preconditioner can be highly effective in many cases.

- Other preconditioners that are problem dependent can be developed. For example, if it is possible to approximate c_u^{-1} with a sparse approximation then the matrix $c_u^{-1}c_m$ can be sparse and sparse linear algebra techniques can be used for the solution of the preconditioned system

It is important to remember that since each product of $J(m)$ and a vector requires an equivalent to the solution of the forward problem (often with multiple right hand sides) using a rather less conventional and more expensive preconditioners may have significant cost effectiveness.

Another important aspect of the solution of the system is our ability to factorize the matrix c_u. If this is possible then the computation of the forward, gradient and the computation in each CG iteration are dramatically reduced. For many right hand sides or for time dependent problems with implicit methods decomposing c_u is highly recommended. In many cases this requires specialize software that allows for the effective decomposition of large systems and a reasonable computational platform that can perform parallel decomposition. Examples for such decompositions are MUMPS [**?**], PARADISO [**?**] and superLU [**?**].

5.5 Discussion - optimization method selection

Although it is often desirable to work with methods that tend to converge faster than steepest descent of LBFGS, it is important to realize some of the advantages of these methods, especially in the context of parameter estimation in PDE's. Although most books tend to emphasize the advantage of not solving linear system (and therefore defaulting to LBFGS for large scale problems) this is not the real advantage in our case. As discussed, solving linear systems is typically done using conjugate gradient, that is, all we need is the evaluation of Jv and $J^\top w$ at each iteration and therefore, each CG iteration is equivalent to one steepest descent step. If the number of CG iterations is small, and the reduction in the function in a Gauss-Newton step is much better than an equivalent number of steepest descent (or LBFGS) steps then Gauss-Newton method and its equivalent would be preferable.

The main difficulty in applying the Gauss-Newton method is especially evident for problems with multiple right hand sides, where for any single model m we have many fields $u_1, \ldots, u_{ns}$. In this case, it is possible to compute (and we will show in a later chapter) that the computation of the function and gradient require that we solve (and hold in memory) only a single field. While for Newton like methods we require to hold the full set of fields. For problems with many sources memory can be a serious issue and can prohibit the use of second order methods.

5.6 Box constraints

For many problems box constraints on the solution exists, that is, we wish to solve

$$\min_m \quad \mathcal{J}(m) = \frac{1}{2}\|Qu(m) - d\|^2 + \alpha R(m) \tag{5.2a}$$

$$\text{s.t} \quad m_{\text{low}} \leq m \leq m_{\text{high}} \tag{5.2b}$$

It is straight forward to verify that the conditions for a minimum are

$$\begin{cases} m_i = [m_{\text{low}}]_i & (\mathcal{J}(m))_i \geq 0 \\ [m_{\text{low}}]_i < m_i < [m_{\text{high}}]_i & (\mathcal{J}(m))_i = 0 \\ m_i = [m_{\text{high}}]_i & (\mathcal{J}(m))_i \geq 0 \end{cases} \tag{5.3}$$

which can be written in compact notation

$$(m - m_{\text{low}}) \odot (m - m_{\text{high}}) \odot \mathcal{J}(m) = 0 \quad m_{\text{low}} \leq m \leq m_{\text{high}} \quad 0 \leq \mathcal{J}(m)$$

One way to obtain this point is to use the projected gradient method. The projected gradient method use the steepest descent method and projects every step to be feasible. First, we define the projection $\mathcal{P}$ to the feasible set

$$\mathcal{P}(t) = \begin{cases} t & \text{if } m_{\text{low}} \leq t \leq m_{\text{high}} \\ m_{\text{low}} & \text{if } m_{\text{low}} > t \\ m_{\text{high}} & \text{otherwise} \end{cases}$$

The projected steepest descent algorithm is a simple variation of the steepest descent algorithm that reads

- Compute $\mathcal{J}(m_k)$ and $\nabla\mathcal{J}(m_k)$
- Set $s_k = -\nabla\mathcal{J}(m_k)$
- Set $m_{k+1} = \mathcal{P}(m_k + \mu_k s_k)$

The line search parameter μ is chosen such that the objective function decreases. The main difference between the steepest descent without constraints to steepest descent with is that the termination criteria is slightly different, noting that we cannot expect to obtain that the gradient will shrink to 0.

It is well documented that projected steepest descent tend to converge rather slowly, exactly like steepest descent. Nonetheless, the method has one remarkable property. Let us define two set of points $m_{\mathcal{I}}$, where if $m_{\text{low}} \leq m \leq m_{\text{high}}$ and $m_{\mathcal{A}}$ where $m = m_{\text{high}}$ or $m = m_{\text{low}}$. Projected steepest descent tends to identify the set of the active constraints much before its final convergence. This leads to a natural idea to combine the method with a Newton-like method. That can be done by separating the iteration into two parts. In the first part, we use steepest descent and in the second we use the Gauss-Newton method on the set of points that is inactive. The algorithm can be summarized as follows

- Compute $\mathcal{J}(m_k)$ and $\nabla\mathcal{J}(m_k)$
- Set $s_k = -\nabla\mathcal{J}(m_k)$
- Set $m_{k+1} = \mathcal{P}(m_k + \mu_k s_k)$
- Identify the set $\mathcal{I}$ where if $m_{\text{low}} \leq m_k \leq m_{\text{high}}$

- Solve the system $P_{\mathcal{I}}^{\top}(J(m)^{\top}J(m) + \alpha\nabla_m^2 R)P_{\mathcal{I}}m = P_{\mathcal{I}}^{\top}\nabla_m\mathcal{J}$

where $P_{\mathcal{I}}$ is a selection matrix such that $m_{\mathcal{I}} = P_{\mathcal{I}}m$ The additional Newton like step speeds up convergence when the correct set of constraint is found.

The Gauss-Newton system that is solved at each iteration is slightly changed. Let $P_{\mathcal{I}}$ be a projection matrix that chooses the inactive set at each iteration. Then, the projected Gauss-Newton step is computed by (approximately) solving the system

$$P^{\top}(J(m)^{\top}J(m) + \alpha\nabla_m^2 R)P\,s = -\nabla_m\mathcal{J}(m).$$

Again, the solution of the system can be done using some iterative method.

5.7 Constraint optimization formulation

In the above discussion we have assumed that the forward problem is solved for each model exactly, that is $u = u(m)$. In this section we discuss a different formulation to the problem where the forward problem is not eliminated to obtain an unconstrained optimization problem.

The motivation to work on the constrained approach is simple. Why solve for $u(m)$ accurately when we are far from the solution? In more technical terminology, we look at the optimality, that is, how small is the objective function and feasibility, that is, how accurate we solve the forward problem. The goal is to solve the forward problem and the optimization problem in tandem that is, converge for the solution of the forward problem as we converge for the solution of the inverse problem.

We thus look at the constrained optimization formulation

$$\begin{aligned} \min \quad & \mathcal{J}(m,u) && \text{(5.4a)} \\ \text{subject to} \quad & c(m,u) = 0 && \text{(5.4b)} \end{aligned}$$

To solve such problems one requires to compute the stationary point of the Lagrangian

$$\mathcal{L}(m,u,\lambda) = \mathcal{J}(m,u) + \lambda^{\top}c(m,u)$$

We now discuss a Newton-like method to achieve this goal.

Differentiating the Lagrangian we obtain the necessary conditions for a minimum sometimes refer to as the Kush-Kun-Tucker (KKT) conditions

$$\nabla\mathcal{L} = \begin{cases} \nabla_u\mathcal{J}(m,u) + (\nabla_u c)^{\top}\lambda = 0 \\ \nabla_m\mathcal{J}(m,u) + (\nabla_m c)^{\top}\lambda = 0 \\ c(m,u) = 0 \end{cases} \tag{5.5}$$

This is a nonlinear system for m, u and λ that can be solved using Newton's method. In each Newton iteration one computes the Hessian $H = \nabla^2\mathcal{L}$ and solves the Newton system

$$Hs = -\nabla\mathcal{L}$$

where $s = [s_u^{\top},\ s_m^{\top},\ s_{\lambda}^{\top}]^{\top}$. When the system is solved we update.

$$u \leftarrow u + \mu s_u \quad m \leftarrow m + \mu s_m \quad \lambda \leftarrow \lambda + \mu s_{\lambda}$$

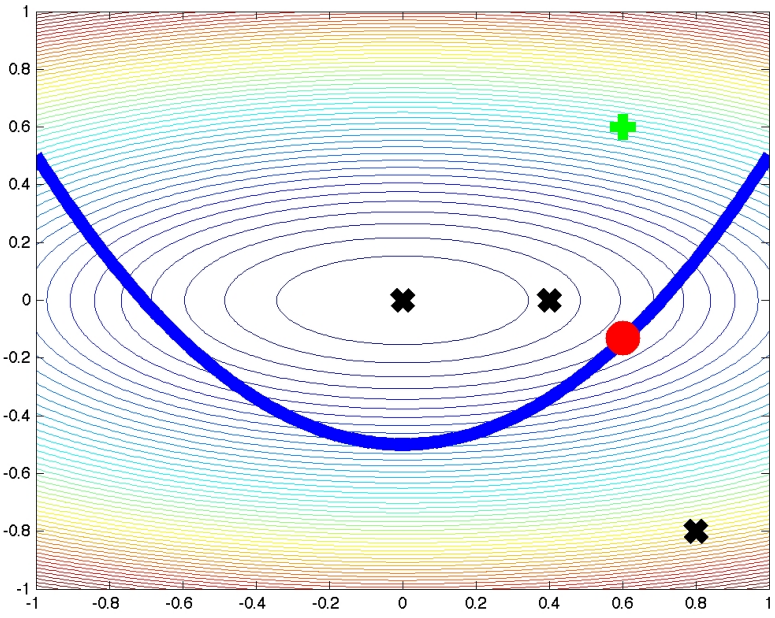

Figure 5.1. *Constrained optimization - the goal is to minimize the function and to stay on the constraint (blue line).*

where μ is a parameter that is guarantied to decrees a merit function.

The difficulty in constrained optimization is that the objective function cannot be used as a merit function since it may not decrease if we wish to be feasible. This is demonstrated in Figure 5.1. Assume that at some iteration we are at the green point. The red point is a local minimizer. It is obvious that one can decrease the value of the objective function without getting closer to the minimum. At the same time, one can get closer to the constraint and get further from the solution. A good step is obtain by a combination of both. It is possible to show that a good descent direction decreases the merit function

$$\mathcal{J}_{\text{merit}} = f(m,u) + \gamma |c(m,u)|_1$$

where the parameter γ needs to be chosen judicially, for more details see [15]. In my experience in many cases, using the merit function $\|\nabla \mathcal{L}\|^2$ can work well in practice

Example: DC resistivity
To be a bit more specific we return to the DC resistivity example. Here we have that

$$c(m,u) = D^\top \,\mathsf{diag}(A_v m)\, Du - q$$

and

$$\mathcal{J}(m,u) = \frac{1}{2}\|Qu-d\|^2 + \frac{\alpha}{2}\|Lm\|^2$$

Differentiating we obtain

$$\begin{array}{cc} \nabla_u \mathcal{J} = Q^\top(Qu-d) & \nabla_m \mathcal{J} = \alpha\, L^\top Lm \\ \nabla_m c = D^\top\, \mathsf{diag}(Du)\, A_v & \nabla_u c = D^\top\, \mathsf{diag}(A_v m)\, D \end{array}$$

The Hessian is

$$H = \begin{pmatrix} Q^\top Q & D^\top \mathsf{diag}(A_v\lambda) D & D^\top\, \mathsf{diag}(A_v m)\, D \\ D^\top \mathsf{diag}(A_v\lambda) D & \alpha L^\top L + D^\top \mathsf{diag}(A_v\lambda) D & A_v^\top \mathsf{diag}(Du) D \\ D^\top\, \mathsf{diag}(A_v m)\, D & D^\top \mathsf{diag}(Du) A_v & 0 \end{pmatrix}$$

5.8 Solving the linear systems

When solving the constrained formulation, every Newton iteration we require to solve a large sparse system of equations. We now briefly review the solution of the system. The system has the form

$$H = \begin{pmatrix} \mathcal{L}_{uu} & \mathcal{L}_{um} & c_u^\top \\ \mathcal{L}_{um}^\top & \mathcal{L}_{mm} & c_m^\top \\ c_u & c_m & 0 \end{pmatrix}$$

The system is indefinite (has positive and negative eigenvalues) and is referred to as a saddle point system, or a KKT system. The interesting aspect of this system is that it represents a system of tightly coupled PDE's. A few approaches have been proposed for the solution of the system. Here we follow two possible approaches.

First, we consider an optimization based approach that is often referred as the reduced Hessian method. The idea here is to eliminate the constraint and obtain a locally unconstrained approach. The (linearized) constrained reads

$$c_u \delta u + c_m \delta m = -c(m,u)$$

Recall that c_u is the Jacobian of the forward problem and therefore can be inverted (at least in principle). We can therefore, eliminate δu from the equations

$$\delta u = c_u^{-1}(-c(m,u) - c_m \delta m$$

We then substitute δu in the first equation obtaining

$$\mathcal{L}_{uu}(c_u^{-1}(-c(m,u) - c_m\delta m) + \mathcal{L}_{um}\delta m + c_u^\top \delta\lambda = -\mathcal{L}_u$$

Using this equation we can solve for $\delta\lambda$ and obtain an equation for m alone.

$$H\delta m = (c_m^\top c_u^{-\top} \mathcal{L}_{uu} c_u^{-1} c_m + \mathcal{L}_{mm} - \mathcal{L}_{um} c_u^{-1} c_m - c_m^\top c_u^{-\top} \mathcal{L}_{um}^\top)\delta m = \mathbf{rhs}$$

It is possible to verify that to compute the product of this matrix and a vector only to linear systems need to be solved. Now, consider the case that we drop the term

$\mathcal{L}_{um}$ from the Hessian. Note that this leads to the Gauss-Newton matrix that was introduced for the unconstrained problem. Indeed, it is possible to verify that the reduced Hessain above is the true Hessain of the unconstrained problem.

As previously discussed, the term $\mathcal{L}_{um}c_u^{-1}c_m - c_m^\top c_u^{-\top}\mathcal{L}_{um}^\top$ may not be positive and therefore is dropped in many practical setting. The linear system is typically solved using a conjugate-gradient like method and δu and $\delta\lambda$ are obtained from δm.

A second approach altogether for the solution of the system is to use the underline properties of the system. Recall that this system represents a tightly coupled system of PDE's and therefore is amendable to PDE solvers. In particular, multigrid methods have been proposed for the solution of the problem [2]. While the discussion about multigrid is beyond the scope of this course we refer the interested reader to a resent review paper on the topic [**?**].

5.9 Discussion - Constrained vs Unconstrained formulation

At this point we would like to comment about the two different approaches for the solution of the problem. Research on the constrained formulation has been in focus in the last 10 years. Using the constrained approach is algorithmically much more difficult compared with the unconstrained approach. However, there is no doubt that by using specialized solvers and sophisticated optimization techniques it is possible to obtain faster constrained algorithms than the constrained ones. The advantage of the constrained methods is that one does not waste expensive iterations, solving the forward problem exactly when we are far from the solution. Nonetheless, this advantage is also the Achilles heal of constrained methods. In the unconstrained methods, every iteration is feasible and therefore, they tend to be more robust to the solution of the problem given say, an inappropriate regularization parameters or when the iteration is terminated early. Another important issue is that for problems with many sources, when working with the constrained approach, one requires to store u, m and λ at each iteration which can be prohibitively expensive.

My recommendation is to use unconstrained approach in general and to invest time and effort in the constrained approach when a problem is repeatedly solved and a more special software is required.