

General Optimization Methods

February 24, 2006

1 Introduction

In the past section we discussed iterative solutions to finding the minimum of the general Tikhonov functional.

$$T_\alpha(m; d) = \|d - g(m)\|^2 + \alpha \|\hat{L}(m)\|^2$$

so that in the case of affine $g(m) = Gm - a$ and $\hat{L}(m) = Lm - b$, m satisfies the linear equations

$$(G^A G + \alpha L^A L) m = r_0,$$

where

$$r_0 = G^A (d + a) + L^A b.$$

This is a set of linear equations that must be solved. In this linear situation we discussed a few methods that are useful for solving this problem iteratively.

In this chapter, the goal is to outline methods for solving the general minimization problem, when either g is non-linear or the functional $J(m) \equiv T_\alpha(m; d)$ is not derived from a Gaussian likelihood and prior (or both). This is the most general situation that arises in practice and it occurs quite frequently. As a result, to be proficient in solving inverse problems, some familiarity with general-purpose optimization methods must be obtained.

2 Projection onto Convex Sets (POCS)

The method of projection onto convex sets is not really an optimization method, but it does allow for finding solutions to inverse problems that are not simply minima of some quadratic Tikhonov functional. The motivation comes from the projected Landweber method which defined an iterative method which can converge to a constrained regularized solution.

A nonlinear approach that is related is to simply specify the solution as the intersection of a several convex sets so that

$$m_o \in \mathcal{C}_1 \cap \cdots \cap \mathcal{C}_N.$$

Obviously, if this solution is to have some validity then the sets \mathcal{C} should be defined as tightly as possible. Some examples of convex sets (some of these have been discussed before) are

- $\|Lm\| < B_1$
- finite support (in space or frequency domain)
- $\|g - Am\| < B_2$

Define P_i to be the projection operator onto \mathcal{C}_i . Then an element from the intersection clearly satisfies the fixed point equation:

$$m = P_1 \circ \cdots \circ P_N (m).$$

The iteration,

$$m_{k+1} = P_1 \circ \cdots \circ P_N (m_k)$$

will converge to an element of the intersection (assuming the intersection is not the empty set). Because of its simplicity, this algorithm can be quite useful. Which element of the intersection will be selected depends on the starting position. Notice that the total projection operator must be a contraction operator to guarantee convergence. If at least one of the operators can be shown to a contraction operator, then the fact that projection operators onto convex sets are always non-expansive is enough to guarantee convergence of the iteration.

This algorithm is called “projection onto convex sets.” It does not need specification of the generalized Tikhonov functional (though in practice it is buried in the definition of one of the convex sets). It is a generally useful, simple algorithm anytime you are looking for an object that you know sits in the intersection of several convex sets.

3 Non-linear Landweber method

The landweber method can be adjusted to the non-linear case but the convergence analysis that allowed a suitable choice for τ doesn't apply and so it is better to use non-linear steepest descent or conjugate gradient. As a result, we will not discuss a non-linear extension to the Landweber method.

4 Non-linear steepest descent

To introduce the terminology used in optimization theory, we will revisit the steepest-descent algorithm discussed in the last chapter. The idea of the steepest-descent algorithm was to move in the direction of the negative gradient (this is the physical gradient, or steepest-descent vector) at each iteration with an optimal step size. When the Tikhonov functional is quadratic, the negative gradient can be computed analytically in terms of the residuals

$$-\nabla J = r(m) = [G(m)]^A (d - g(m)) - \alpha [L(m)]^A \hat{L}(m).$$

We use

$$r_k \equiv r(m_k).$$

The optimal distance to move in that direction (assuming it is small enough that an affine approximation for g and \hat{L} work):

$$\tau_k = \frac{\Re(r_k, r_k)}{(r_k, \bar{A}r_k)},$$

where $\bar{A} = G^A G + \alpha L^A L$.

In the general non-linear case, this optimal distance to move might not be the best choice because it was derived assuming a Taylor series on both $g(m)$ and $\hat{L}(m)$. We can still implement the steepest descent algorithm if we choose the step-size using a one-dimensional search:

$$\tau_k = \arg \min_{\tau > 0} \phi(\tau)$$

where $\phi(\tau) = J(m_k + \tau p_k)$ and p_k is the search direction (in this case $p_k = r_k$). Notice that

$$\phi'(\tau) = \left\langle p_k, dJ|_{m_k + \tau p_k} \right\rangle = -(p_k, r(m_k + \tau p_k))$$

In practice, the line search does not have to be accurate, we just need it to actually move to a lower value for the function and satisfy some further conditions (e.g. Wolfe) that can be somewhat algorithm dependent. Some considerations with regard to the line search are given in the overview section below.

5 Non-linear conjugate gradient

The non-linear Conjugate Gradient (CG) algorithm extends the conjugate gradient algorithm to the case of minimizing general non-quadratic functions. It is usually preferable to the steepest descent algorithm as it

generally converges more quickly and require only a bit more storage. It can be seen as a pre-conditioned steepest-descent method since the search direction is replaced with the conjugate-gradient direction.

Recall that the linear CG algorithm updates the search direction using a direction that points to the minimum of the discrepancy functional projected onto the current Krylov subspace. This direction is built up recursively using the current negative gradient. In the linear case, this negative gradient is proportional to the residuals, but in the nonlinear case it must be computed (or estimated). In addition, how far to go in the current update step must also be estimated using a line search (in the linear case it could be directly calculated). Consequently, the (Fletcher-Reeves) non-linear CG algorithm is similar to the linear case except the update-step is

$$\alpha_k = \arg \min_{\alpha > 0} J(m_k + \alpha p_k).$$

In reality, α_k does not have to be an accurate minimum but only has to improve the estimate. One method to choose α is to find a value of α_k that satisfies the strong Wolfe conditions:

$$\begin{aligned} J(m_k + \alpha_k p_k) &\leq J(m_k) + c_1 \alpha_k (\nabla J, p_k) \\ |(\nabla J(m_k + \alpha_k p_k), p_k)| &\leq c_2 |(\nabla J, p_k)|, \end{aligned}$$

where $0 < c_1 < c_2 < \frac{1}{2}$. Any line search that yields an α_k satisfying the above properties will ensure that all search directions p_k are descent directions for J . Notice that if $\phi(\tau) = J(m_k + \tau p_k)$ then these Wolfe conditions can be stated as

$$\begin{aligned} \phi(\alpha_k) &\leq \phi(0) + c_1 \alpha_k \phi'(0) \\ |\phi'(\alpha_k)| &\leq c_2 |\phi'(0)|. \end{aligned}$$

As these conditions are useful for any line-search algorithm, more discussion on these Wolfe conditions will be given below.

5.1 Fletcher-Reeves

For reference, the Fletcher-Reeves version of the nonlinear CG method which makes the two changes described above can be written as

- Start with m_0
- $q_0 = \nabla J(m_0)$; $p_0 = -q_0$
- $\delta_0 = \|q_0\|^2$
- Start iterations
 - Find α_k to satisfy Wolfe conditions.
 - $m_{k+1} = m_k + \alpha_k p_k$
 - $q_{k+1} = \nabla J(m_{k+1})$
 - $\delta_{k+1} = \|q_{k+1}\|^2$
 - $\beta_k = \frac{\delta_{k+1}}{\delta_k}$
 - $p_{k+1} = -q_{k+1} + \beta_k p_k$
- End iterations

5.2 Polak-Ribiere

There are several variations of the non-linear CG algorithm that differ only in their choice of β_k . The previously quoted algorithm uses the definition of β_k useful in the linear case. An important variant of the nonlinear CG method was introduced by Polak and Ribere. It defines the beta parameter as

$$\beta_k^{PR} = \max \left\{ \frac{(q_{k+1} - q_k, q_{k+1})}{(q_k, q_k)}, 0 \right\}.$$

Notice what happens if we ever need to choose $\beta_k = 0$ (which will occur if $(q_k, q_{k+1}) > (q_{k+1}, q_{k+1})$). In this situation, the conjugate search direction is reset to the current negative gradient just as it was in the beginning and we can say that the search has restarted. In the linear case, of course, all gradients are orthogonal and $\beta_k^{PR} = \beta_k^{FR}$ but in the non-linear case with inexact line-searches, the two different choices for β_k can lead to different convergence results. Apparently, the Polak-Ribiere choice has seen more success.

Another variant due to Hestenes and Stiefel that gives algorithmic performance similar to that of Polak-Ribiere but which has some additional theoretical justification is to choose β_k so that successive vectors p_{k+1} and p_k are conjugate with respect to the average Hessian (matrix of second derivatives of a function) over the line-segment from m_k to m_{k+1} . The average Hessian is

$$\bar{H}_k \equiv \int_0^1 \text{Hess} [J(m_k + \tau \alpha_k p_k)] d\tau.$$

Enforcing $(p_{k+1}, \bar{H} p_k) = 0$ requires that

$$\beta_k^{HS} = \frac{(q_{k+1}, q_{k+1} - q_k)}{(q_{k+1} - q_k, p_k)}.$$

In practice, either β_k^{PR} or β_k^{HS} seem to give similar performance. There is some theoretical value to the β_k^{HS} definition however which will become clear in the next section.

6 Broyden-Fletcher-Goldfarb-Shanno (BFGS)

Another pre-conditioned steepest-descent algorithm can be obtained by using a different weighting operator to map from the partial-derivative dual-vector to the direction vector. These algorithms are based on Newton's method and use a different weighting scheme than the inverse of the covariance operator W_M .

6.1 Newton Methods

Newton's method for minimizing a function locally approximates the function using the second order Taylor-series.

$$J(m_0 + h) = J(m_0) + J'(m_0)h + \frac{1}{2}(J''(m_0)h)h.$$

where $J'(m_0)$ is the first-order Frechet derivative and $J''(m_0)$ is the second-order Frechet derivative. Taking the first Frechet derivative of this functional (with respect to h) (and using the symmetry of $J''(m_0)$) gives

$$J'(m_0) + J''(m_0)h = 0.$$

Notice, that in this equation $J''(m_0)$ can be seen as a linear operator mapping \mathbb{M} to \mathbb{M}^* . If $J''(m_0)$ is positive definite, then it can be a weighting operator. For finite dimensional spaces, this equation becomes

$$dJ + \mathbf{H}h = 0,$$

where \mathbf{H} is a symmetric matrix of second-order derivatives:

$$H_{ij} = \frac{\partial^2 J}{\partial m^i \partial m^j}.$$

so that the update to m_0 that could potentially optimize the local quadratic is

$$p = h = -H^{-1}dJ.$$

In order that this update step be an approximate minimum, it is required that H be positive definite (i.e. $\mathbf{x}^T \mathbf{H} \mathbf{x} > 0$ so that H is a true weighting operator). Aside from the fact that inverting the Hessian matrix at each iteration can be costly, if the Hessian is not positive definite (we are near a maximum, or saddle point), Newton iterations can fail to converge to a local minimum.

Convergence was not a problem when J was a quadratic (affine G and L) because the Hessian matrix was a constant and always positive definite. The conjugate gradient iteratively built up the inverse Hessian matrix so that the minimum was not found in one step but as the limit of a sequence of steps.

6.2 Quasi-Newton Methods

Quasi-Newton methods build up an approximation to the (inverse of) the Hessian matrix and use that approximation at each iteration. Line-search quasi-Newton methods, for example, use a model for the J that satisfies

$$\hat{J}_k(m) = J_k + (\nabla J_k, p) + \frac{1}{2}(H_k p, p)$$

where $B_k = H_k^{-1}$ is symmetric positive-definite matrix that is updated at every iteration. The new search direction is the minimizer of this model:

$$p_k = -B_k dJ_k.$$

The new iterate is

$$m_{k+1} = m_k + \alpha_k p_k$$

where α_k is chosen to satisfy the Wolfe conditions using a line-search algorithm.

There are different variations of the Quasi-Newton methods which differ in how the H_k^{-1} matrix is updated. The Broyden-Fletcher-Goldfarb-Shanno (BFGS) update for $B_k \equiv H_k^{-1}$ is considered to be the most effective of the Quasi-Newton methods. The update is derived by choosing B_{k+1} as

$$B_{k+1} = \arg \min_B \|B - B_k\|$$

subject to

$$B = B^T \quad B y_k = s_k$$

where

$$\begin{aligned} y_k &= dJ_{k+1} - dJ_k \\ s_k &= m_{k+1} - m_k. \end{aligned}$$

The matrix norm used is the weighted Frobenius norm defined as

$$\|A\|_W = \left\| W^{1/2} A W^{1/2} \right\|_F$$

where

$$\|C\|_F = \sum_{i,j} c_{ij}^2.$$

The weight can be chosen as any matrix satisfying $W s_k = y_k$ (the secant equation), and $W^{1/2} W^{1/2} = W$ (matrix square-root). For concreteness assume that $W = \bar{G}_k$, where

$$\bar{G}_k = \int_0^1 \text{Hess}(J(m_k + \tau \alpha_k p_k)) d\tau$$

is the average Hessian. The unique solution to this problem gives the BFGS update

$$B_{k+1} = (I - \rho_k s_k y_k^T) B_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

where

$$\rho_k = \frac{1}{y_k^T s_k}.$$

Notice that if B_k is positive definite, then B_{k+1} will be positive definite as well. In order to start the algorithm, a first estimate of the inverse Hessian is needed. Often the identity is chosen or a scaled version of the identity (to account for different scales in the components of m). For particular problems, other guesses for the inverse Hessian could be more effective.

To summarize the above algorithm. The BFGS algorithm is

- Start with a guess m_0 and an inverse Hessian approximation B_0 (could be just I)
- Find dJ_0
- Begin iterations (could end when $\|\nabla J\| < \epsilon$)
 - Find search direction $p_k = -B_k dJ_k$
 - Find α_k using a line search that satisfies Wolfe conditions
 - $m_{k+1} = m_k + \alpha_k p_k$
 - Compute dJ_{k+1}
 - Define $s_k = m_{k+1} - m_k = \alpha_k p_k$ and $y_k = dJ_{k+1} - dJ_k$
 - $B_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k} \right) B_k \left(I - \frac{y_k s_k^T}{y_k^T s_k} \right) + \frac{s_k s_k^T}{y_k^T s_k}$
- End iterations

6.3 Limited-memory BFGS and relationship to nonlinear-conjugate gradient methods

The biggest drawback of the quasi-Newton methods is that they require the storage of the $N \times N$ Hessian matrix where N is the number of unknowns being recovered, while the non-linear conjugate gradient algorithms do not require that storage. There is a useful relationship however to the non-linear conjugate gradient algorithms that shows how a limited-memory BFGS algorithm could be implemented

Recall that the Hestenes-Stiefel form of the non-linear conjugate gradient algorithm used

$$\beta_k = \frac{(q_{k+1}, q_{k+1} - q_k)}{(q_{k+1} - q_k, p_k)} = \frac{(\nabla J_{k+1}, y_k)}{(p_k, y_k)}$$

so that (recall that $s_k = \alpha_k p_k$)

$$\begin{aligned} p_{k+1} &= -\nabla J_{k+1} + \beta_k p_k \\ &= -\nabla J_{k+1} + \frac{(\nabla J_{k+1}, y_k)}{(p_k, y_k)} p_k \\ &= -\nabla J_{k+1} + \frac{s_k y_k^T}{(s_k, y_k)} \nabla J_{k+1} \\ &= -\left(I - \frac{s_k y_k^T}{y_k^T s_k} \right) \nabla J_{k+1} \\ &\equiv -\hat{H}_{k+1} \nabla J_{k+1}. \end{aligned}$$

In this form, the non-linear conjugate gradient algorithm resembles a quasi-Newton algorithm, except \hat{H}_{k+1} is not symmetric nor positive definite. A related form for H_{k+1} that is symmetric and positive definite can be found as

$$H_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k} \right) \left(I - \frac{y_k s_k^T}{y_k^T s_k} \right) + \frac{s_k s_k^T}{y_k^T s_k}.$$

This is just the BFGS update equation with $H_k = I$.

Thus a limited-memory BFGS algorithm can be conceived up by using the quasi-Newton update but starting over with the identity matrix at each iteration. Other limited-memory forms use only m of the previous s_k and y_k pairs at each iteration for improved approximation to the Hessian.

If we combine the limited-memory BFGS algorithm implied by the equation for H_{k+1} given above with an exact line-search (so that $\phi'(\alpha) = 0$ which is equivalent to $s_k^T \nabla J_{k+1} = 0$) then we get for the update search direction

$$\begin{aligned} p_{k+1} &= -H_{k+1} \nabla J_{k+1} \\ &= -\nabla J_{k+1} + \frac{p_k y_k^T \nabla J_{k+1}}{y_k^T p_k} \\ &= -\nabla J_{k+1} + \frac{(\nabla J_{k+1}, y_k)}{(p_k, y_k)} p_k \\ &= -\nabla J_{k+1} + \beta_k^{HS} p_k \end{aligned}$$

which is equivalent to the Hestenes-Stiefel formula. In addition, if $s_k^T \nabla J_{k+1} = 0$ then $\beta_k^{HS} = \beta_k^{PR}$. This connection between the non-linear conjugate gradient algorithms and the quasi-Newton BFGS algorithm is intriguing and could help explain the success of the non-linear conjugate gradient algorithms.

If the limited memory BFGS algorithm is not combined with an exact line search then

$$\begin{aligned} p_{k+1} &= -H_{k+1} \nabla J_{k+1} \\ &= -q_{k+1} + \frac{p_k y_k^T q_{k+1}}{y_k^T p_k} + \frac{y_k p_k^T q_{k+1}}{y_k^T p_k} - \frac{p_k y_k^T y_k p_k^T q_{k+1}}{(y_k^T p_k)^2} - \alpha_k \frac{p_k p_k^T q_{k+1}}{y_k^T p_k} \\ &= -\gamma_k \nabla J_{k+1} - \delta_k \nabla J_k + \beta_k^O p_k \end{aligned}$$

where

$$\begin{aligned} \gamma_k &= 1 - \delta_k \\ \delta_k &= \frac{(q_{k+1}, p_k)}{(p_k, q_{k+1} - q_k)} \\ \beta_k^O &= \frac{(q_{k+1}, y_k)}{(p_k, y_k)} - \frac{(q_{k+1}, p_k)}{(p_k, y_k)} \left(\frac{(y_k, y_k)}{(p_k, y_k)} - \alpha_k \right). \end{aligned}$$

I do not know if this version of the limited-memory BFGS algorithm performs any differently than the Polak-Ribiere conjugate-gradient algorithm.

7 Overview of unconstrained optimization methods

We have reviewed some of the most commonly used unconstrained optimization algorithms. The entire field of unconstrained optimization is a large forest with many varieties of trees. Some of the patterns in the forest can be understood however. This section will be a view of the forest of unconstrained optimization from a few hundred feet up.

7.1 Non-linear (Nelder-Mead) simplex algorithm

This algorithm attempts to find a minimum of a function using only function calls. It does not attempt to specify gradients. This algorithm may be appropriate when the gradient is hard to calculate, or when the function is not smooth. For an n -dimensional problem, this method maintains a simplex of $n+1$ points (*e.g.* a triangle in two dimensions, or a pyramid in three dimensions). The simplex moves, expands, contracts, and distorts its shape as it attempts to find a minimizer. This method is slow and can be applied only to problems in which n is small. Despite its potentially poor convergence, it is quite popular because it requires the user to supply only function values, not derivatives.

7.2 Non-linear Conjugate gradients

As described above these codes extend the success of linear conjugate gradient methods to minimizing non-quadratic functionals. They are (somewhat) related to quasi-Newton methods but have a different history of development. They do not require much storage but do require computation of the gradient of the functional. The most important variant is the Polak-Ribiere variant for the calculation of β_k .

These methods have little advantage over the BFGS method except they don't require storage of the intermediate inverse Hessian approximation.

7.3 Newton Methods

Newton's methods are derived from the second-order Taylor series of the functional around the current estimate of the minimum:

$$J(f_k + s_k) = J(f_k) + \nabla J(f_k) s_k + \frac{1}{2} s_k^T H(f_k) s_k$$

where $H(f_k)$ is the Hessian matrix composed of the second partial derivatives of the functional with respect to (the components of) f_k . At each step of the iteration, the new update is

$$f_{k+1} = f_k + s_k$$

where s_k is the solution to the system of equations

$$H(f_k) s_k = -\nabla J(f_k).$$

Because the Hessian may not be positive definite, this iteration may not converge to a minimum. As a result, practical Newton methods must modify the basic idea in some way. There are two commonly used variants: line-search and trust-region methods. These variants are also found for Quasi-Newton methods discussed next. If the Hessian is not known, it can be approximated using gradient differences.

7.3.1 Line search

Line search Newton methods do not solve the system of equations for the inverse Hessian completely at each iteration. Instead, some iterative method (such as conjugate gradients) is applied to solve the system which then terminates whenever the residuals are "small enough" or when negative curvature is detected. Stopping early provides a direction, and then a step length is chosen using a line search. The step-length is selected as before to satisfy conditions like the Wolfe conditions.

This Newton-Conjugate-Gradient (Newton-CG) can converge in fewer iterations than other algorithms. However, the cost per iteration can be higher depending on the application. For this reason quasi-Newton methods are used quite often.

7.3.2 Trust region search

We have discussed mainly line-search variants of our optimization algorithms. There is another approach called a trust-region search. The biggest difference with the line search methods is that instead of picking a direction and then choosing a "good enough" step distance, the trust-region search methods, pick a step distance and then find a direction to step in that is "good enough" for this iteration. Thus, the trust-region iteration requires that we find

$$\min_s Q_k(s) \quad \text{subject to} \quad \|s\| \leq \Delta_k$$

where $Q_k(s)$ is a quadratic approximation to the functional. These methods can be more difficult to code but can also be more robust in practice.

7.4 Quasi-Newton Methods

Many of the advantages of Newton's methods (quick convergence, robustness) can be obtained using the BFGS Quasi-Newton method. This method chooses the search direction using an estimate of the inverse Hessian matrix that is updated at each iteration. A line search finds a step-length to complete the iteration. Trust-region variants of the the quasi-Newton method have also been described.

8 Line search

A critical part of most of the algorithms described here is the line search algorithm that determines how far the step should proceed in the direction chosen. As mentioned, this algorithm does not need to return an exact minimizer to the one-dimensional equation, but just a step-length that is large enough resulting in a small enough slope. A common set of conditions is the strong Wolfe conditions. If we define

$$\begin{aligned}\phi(\alpha) &= J(m_k + \alpha p_k) \\ \phi'(\alpha) &= (\nabla J(m_k + \alpha p_k), p_k)\end{aligned}$$

then the strong Wolfe conditions state that the step-length α_k must satisfy

$$\begin{aligned}\phi(\alpha_k) &\leq \phi(0) + c_1 \alpha_k \phi'(0) \\ |\phi'(\alpha_k)| &\leq c_2 |\phi'(0)|\end{aligned}$$

where c_1 and c_2 are constants with $0 < c_1 < c_2 < 1$ or sometimes (for the non-linear conjugate gradient method) $c_2 < \frac{1}{2}$.

The first of these conditions is sometimes called the sufficient decrease condition, or the *Armijo condition*. It's intent is to ensure that the step-length chosen decreases the function value sufficiently notice that $\alpha_k \phi'(0)$ gives the decrease in the function value approximated by a straight-line fit. Typically values of c_1 are chosen to be quite small, *e.g.* $c_1 = 10^{-4}$ or $c_1 = 10^{-3}$.

The second Wolfe condition is sometimes called the *curvature condition*. It ensures that the slope of the functional at the step-length chosen is closer to zero than the original slope (at the minimum $\phi'(\alpha_k) = 0$). Typical values of c_2 are 0.9 when the search direction is chosen using a Newton or quasi-Newton method and $c_2 = 0.3$ when p_k is chosen using a non-linear conjugate gradient method.

An algorithm that finds a suitable step length to satisfy these conditions usually consists of two parts:

- A bracketing phase in which the interval is expanded (often in powers of two) until either an acceptable α_k is found or an interval in which an acceptable α_k should be found is identified.
- A selection phase in which interpolation is used to estimate a value of α_k which is then checked to see if the Wolfe conditions are satisfied.

An algorithm that implements a good line search can be difficult to write and it is often best to call a pre-written algorithm. Open-source Line-search algorithms can be found for MATLAB and SciPy and are also available in commercial optimization codes. For an outline of a line-search algorithm see Nocedal and Wright pg. 59-60.