17. Steepest Descent Optimization

In geophysical inverse theory, and in many other geophysical contexts, we need to find the minimum of a real function F of many variables. (Let us say $F : \mathbb{R}^N \to \mathbb{R}$.) This is an example of an **optimization** problem, because the function F is regarded as a penalty of some kind, and minimizing it represents doing the best possible job in some sense. We have already come across a simple example of this kind in section 3, the least-squares problem, where $F = || Ax - y ||^2$, and we are minimizing the distance between a data vector y and model predictions Ax. Sometimes the minimization must be carried out subject to side conditions, or **constraints**, and then the problem is a **constrained optimization**. Again the underdetermined least squares systems is an example of this kind. Now we consider briefly general methods for solving these problems, though in fact we will look only at the unconstrained system for simplicity. The standard reference for numerical techniques is by Gill, P. E., Murray, W. and M. H. Wright, *Practical Optimization,* Academic Press, New York, 1981; Philip Gill is on the UCSD math department faculty. Our books on reserve, by Strang and by Golub and Van Loan also provide a lot of information about the topic too.

We shall assume the function F is smooth, at least twice differentiable, for otherwise things get messy. We will also assume that we can obtain an analytic expression for the derivatives of F, the gradient of F:

$$g = \nabla F = \left[\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, \cdots \frac{\partial F}{\partial x_n} \right]^T . \tag{1}$$

Then the condition that we are at a local minimum is of course that the gradient vanishes, that is all the components:

$$\nabla F(x) = 0 . \tag{2}$$

This is a system of n equations in n unknowns, and unless F is quadratic (which it is for least-squares problems) (2) may be impossible to solve in terms of elementary functions; when F is quadratic (2) is a set of linear equations. We will return to the quadratic case later, because it is surprisingly important. Notice (2) locates any local minimum, maximum or saddle; if there are multiple minima, they will all satisfy the system, and we must pick the best one, a fundamental difficulty in general optimization problems.

We examine first a very simple-minded idea called **steepest descent**. Suppose we are the vicinity of a local minimum, at the point $x^0 \in \mathbb{R}^N$ (Because subscripts denote components of a vector, we have to use superscripts, which do not mean powers of x; there should be no confusion since we normally do not exponentiate vectors). Then as usual we write the local behavior of F using a Taylor expansion:

$$F(x) = F(x^0 + s) = F(x^0) + s^T \nabla F(x^0) + O \| s \|^2 . \tag{3}$$

If we take $s = -\gamma \nabla F(x^0)$ with small γ, then

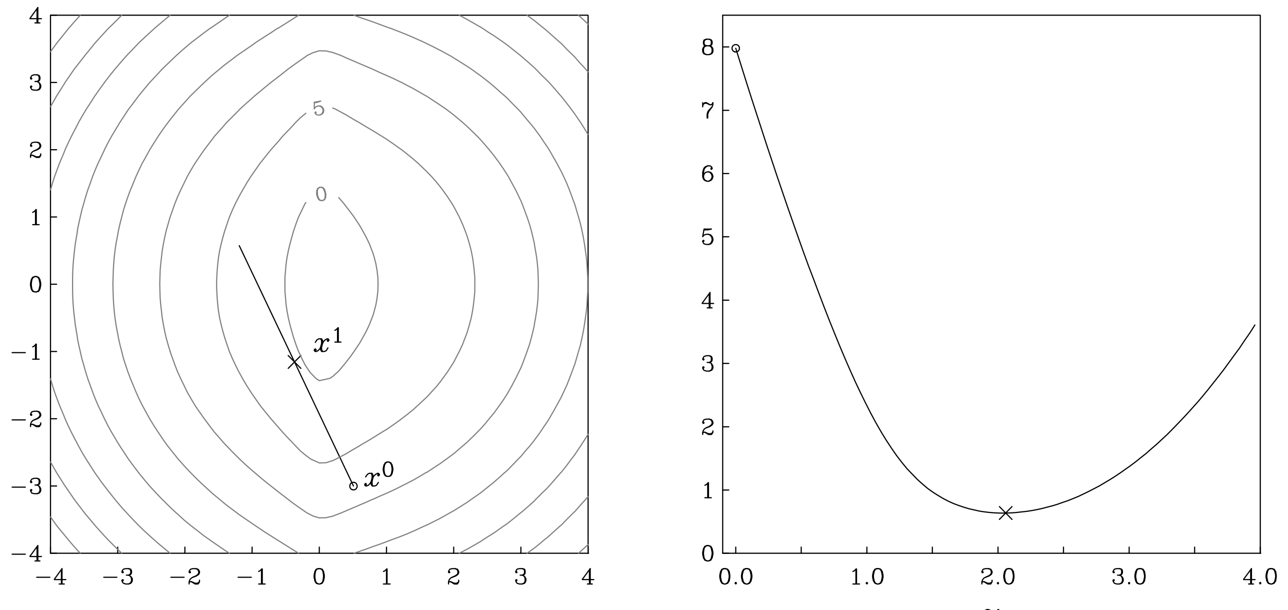
$$F(x) = F(x^0) - \gamma \| \nabla F(x^0) \|^2 + O(\gamma^2) \tag{4}$$

and from this we see that for some choice of $\gamma > 0$ we must be able to find a value of $F(x)$ smaller than $F(x^0)$, and therefore better, because from small enough γ the linear term will dominate the quadratic one. (This is provided $\nabla F(x^0)$ does not vanish, but then we would be at a stationary point.) Looking at the 2-dimensional example in the figure below we see that taking the new $x = x^0 - \gamma \nabla F(x^0)$ is to select a value on the line perpendicular to the local contour, that is to head downhill as rapidly as possible, hence the name steepest descent). But what value of γ should be selected?

The answer to this question is fairly obvious: we keep going until F starts to increase again. In other words we go to the minimum along the direction of initial steepest gradient, as illustrated below. In general this point must be determined by numerical experiment: in a systematic way we try various values of γ in a **line search** until the least value has been found in the direction of $\nabla F(x^0)$. Clearly it is most unlikely that the result will be the true minimum, and so the process will have to be repeated. But the analysis guarantees an improvement will be obtained for every iteration, and so if there is a local minimum, the procedure will converge to it.

So the algorithm is as follows: at the k-th step we compute the next approximation from

Figure 17.1: Contours of F from (6), and values of F on the line of steepest descent starting at x_0.



$$x^{k+1} = x^k - \gamma\,\nabla F(x^k),\ \ k = 0, 1, 2, \cdots \tag{5}$$

where $\gamma > 0$ is chosen to minimize $F(x^{k+1})$, by a line search; x^0 is an arbitrary initial guess vector.

Below we illustrate the continuation of the iterations, using the solution from the previous line search as a starting point for another, and repeating for several steps. Incidentally, the function used for illustration is:

$$F(x_1, x_2) = (x_1 - 0.5)^2 + x_2^2 + \ln(x_1^2 + 0.1) \tag{6}$$

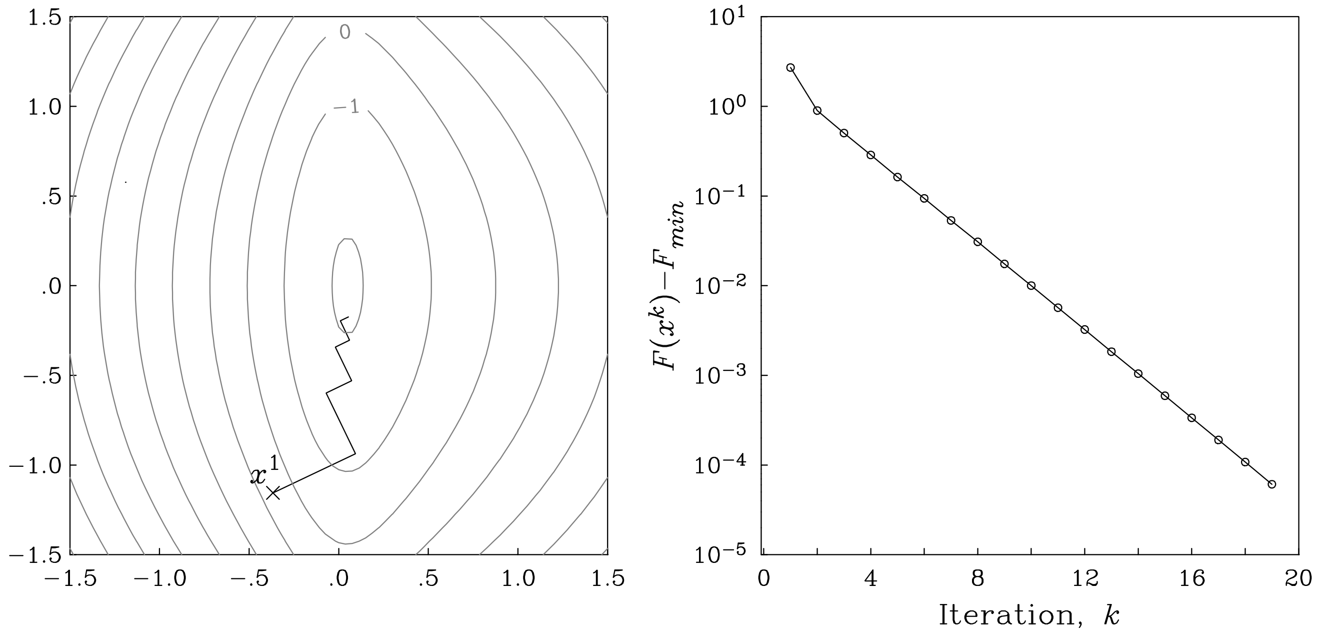
$$\nabla F = [2x_1 - 1 + 2x_1/(x_1^2 + 0.1),\ \ 2x_2]^T\ . \tag{7}$$

After a little thought it should be clear that the steepest descent path must be orthogonal to the previous one, and therefore the trajectory consists of a zig-zag path downhill. The error in the minimum decreases geometrically as you can see from the right panel, and while this looks fairly impressive, it is not very good for a simple two-dimensional minimum; recall every point on the graph requires a separate line search.

It is easy to write a crude line-search program, as you can imagine. But a fair amount of care is needed to be efficient and avoid blunders. See *Numerical Recipes* for a classic algorithm.

To understand, and then perhaps correct, this poor behavior we study the simplest system, the quadratic form. If the second derivative $\nabla\nabla F = H$ does not vanish at x^*, the local minimum, then the behavior of any smooth function of many variables is quadratic and is modeled by

Figure 17.2: Steepest descent path and convergence of objective function for (6).



$$F(x) = F(x^*) + \tfrac{1}{2}(x - x^*)^T H(x - x^*) \tag{8}$$

where we have dropped the terms from the third and higher derivative in a Taylor series expansion. If x^* truly is the site of a local minimum then H is positive definite. Then it can be proved that the error in the estimate of the minimum value at the k-th step behaves like $c\,[1 - 1/\kappa_2(H)]^k$ where c is a constant, and κ_2 is the condition number in the 2-norm (Recall $\kappa_2 = \lambda_{\max}/\lambda_{\min}$). For example, condition numbers greater than 1,000 are commonplace, and then the convergence would be as 0.999^k. This is very poor behavior.

Before discussing the most popular remedy for this ailment, we should notice that (8) is essentially identical to the function minimization arising from the underdetermined least-squares problem: we must minimize

$$G(x) = \|Ax - b\|^2 = (Ax - b)^T(Ax - b) \tag{9}$$

$$= b^T b - 2b^T Ax + x^T A^T Ax \tag{10}$$

$$= b^T b - y^T y + (x - y)^T A^T A(x - y) \tag{11}$$

where $y = (A^T A)^{-1} A^T b$ (we arrived at y by completing the square). Comparing (9) with (11) we see they are the same (up to an additive constant), after identifying $A^T A$ with $\tfrac{1}{2}H$ and y with x^*. For sparse systems, in which lots of elements in A might be zero, QR is unable to take much advantage of sparsity. So when the system is large (> 1000 unknowns) it might be very useful to minimize G in (9) directly by an iterative method, since one evidently only needs to be able to perform the operation Ax a lot of times, and it is often possible to simply skip large chunks of the array, both in storage and in arithmetic, associated with zero entries. Furthermore, QR attempts to get an "exact" solution (up to limitations of round-off), but an iterative approach might find a less accurate, but for many purposes completely satisfactory, answer in a much shorter time. For these reasons, large linear systems, even the solution of $Ax = y$ for square matrices A, are converted to quadratic minimizations. But they cannot be efficiently solved by steepest descent; we need a better tool: conjugate gradients.

18. Conjugate Gradients

The steepest descent path is clearly the best one can do if one is permitted only a single operation. But each stage of the scheme behaves as though we have been given a completely new problem — it doesn't use any information from the earlier steps, and as the Figure 17.2 shows, the procedure seems condemned to repeat itself, zig-zagging back and forth instead of heading down the axis of the valley in F. The conjugate gradient method takes advantage of earlier steps. It modifies the steepest descent direction in the light of previous history, and achieves remarkable gains, as we shall soon see. First let me simply describe the algorithm without attempting to justify it.

The **conjugate gradient** algorithm chooses a search direction s for a line search based on the local gradient, and on previous search directions like this:

$$x^{k+1} = x^k + \gamma\, p^k, \quad k = 0, 1, 2, \cdots \tag{1}$$

where

$$p^k = -\nabla F(x^k) + \beta p^{k-1}, \quad k > 0 \tag{2}$$

and

$$\beta = \frac{\nabla F(x^k)^T (\nabla F(x^k) - \nabla F(x^{k-1}))}{\|\nabla F(x^{k-1})\|^2}\,. \tag{3}$$

For the initial iteration, $k = 0$, when there is no previous search direction, $p^0 = -\nabla F(x^0)$, the steepest descent direction. At each step γ in (1) is determined as before by minimizing $F(x^{k+1})$ along the line.

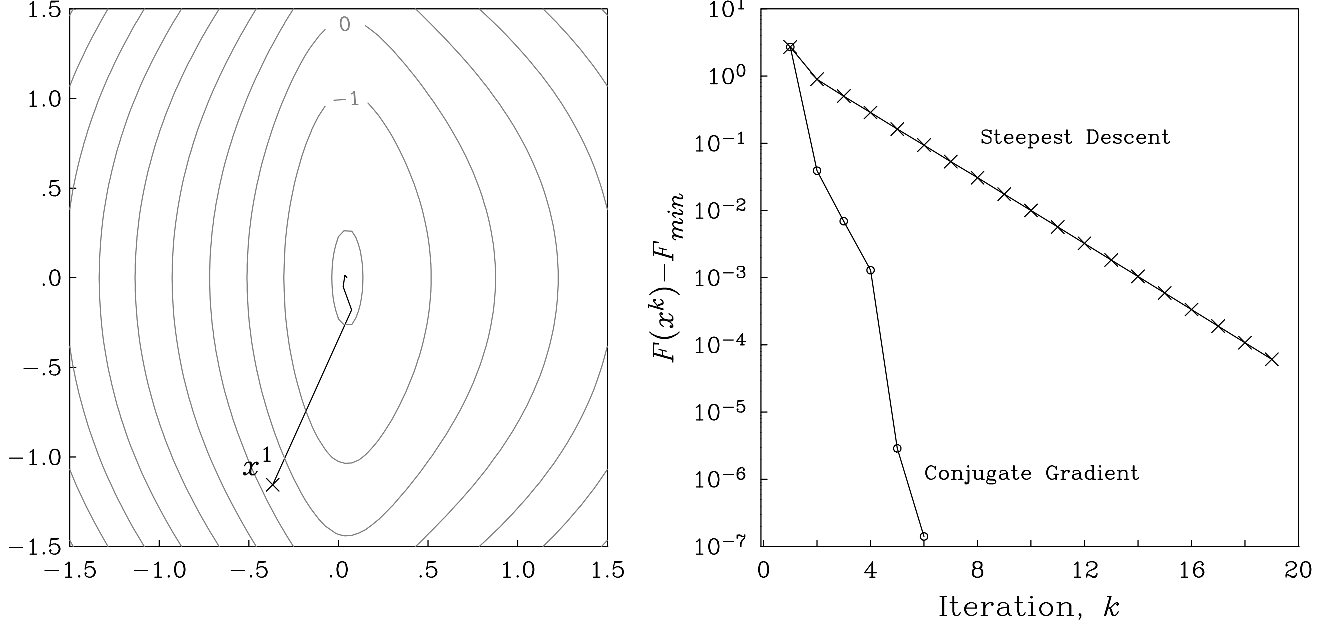


Figure 18.1: Conjugate gradient and objective convergence for (6).

On the prevous page we show the results of the application to the minimization of (5). The improvement on steepest descent is extraordinary, as the right panel shows. Notice also that the convergence is not a steady exponential decline in error; the rate varies. This is a feature of conjugate gradient optimization: it may chug along reducing the penalty only modestly, then make a huge gain, then settle back to slower progress. Such behavior makes a termination strategy difficult to devise, because one cannot tell when the penalty has been reduced to its minimum value. Remember, the convergence plots in these notes are cheats, because I know the real answer, something obviously not normally available.

The design of the conjugate gradient method is centered on the goal of solving a problem with a quadratic penalty function, like (8), exactly after precisely n iterative steps, where n is the dimension of the space of unknowns. When solving a very large system (with $n > 1000$, say), one would not want to have to take so many steps, but it is often the case that an exact answer is not required, and a perfectly satisfactory reduction in the penalty function will have been achieved long before $k = n$. It also turns out that for very large systems, exact answers cannot be obtained in practice even after n iterations, because of the accumulation of round-off error in the computer arithmetic.

Here is an explanation of how conjugate gradients work, taken from Gill et al., Chap 4. Strang, and Golub and Van Loan, offer different derivations which are longer. As just remarked the procedure is set up to solve a quadratic problem, which we will take to be the minimization of

$$F(x) = c \cdot x + \tfrac{1}{2} x \cdot Gx \tag{4}$$

where $G \in \mathbb{R}^{n \times n}$ and is positive definite and symmetric. For this last proof we will use the familiar notation $x \cdot y$ to be the inner product of two vectors because it is much cleaner: so recall $x \cdot y = x^T y = y^T x$. Also, since G is symmetric, note that $x \cdot Gy = y \cdot Gx$.

The exact minimum of F is easily seen to be the point $x^* = -G^{-1} c$, so solution of (4) by conjugate gradients is equivalent to solving that linear system of equations. You will easily verify that

$$\nabla F(x) = c + Gx\,. \tag{5}$$

We look at the process at iterative step k; we assume we have an approximation for the minimizer x^k and we are going build the next approximation by a linear combination of vectors, $p^0,\ p^1,\ \cdots p^k$ collected over previous iterations, together with the current approximation to the solution; we will explain later how the vectors p^k are chosen. For now we assert:

$$x^{k+1} = x^k + \sum_{j=0}^{k} w_j p^j \tag{6}$$

$$= x^k + P_k w \tag{7}$$

where the matrix $P = [p^0, p^1, \cdots p^k]$ and the vector w contains the weights. Our first job is to find w so that x^{k+1} in (7) minimizes F. This is a straightforward least-squares problem, details omitted. We find

$$w = -(P_k^T G P_k)^{-1} P_k^T g^k \tag{8}$$

where the vector g^k is defined as the gradient:

$$g^k = \nabla F(x^k) = c + Gx^k \,. \tag{9}$$

Plugging (8) into (7) gives

$$x^{k+1} = x^k - P_k(P_k^T G P_k)^{-1} P_k^T g^k \,. \tag{10}$$

At this point we note a useful property of the process: the gradient at the $k+1$-st approximation is orthogonal to all the current vectors p^i. Proof — calculate:

$$P_k^T g^{k+1} = P_k^T \nabla F(x^{k+1}) = P_k^T (c + Gx^{k+1}) \tag{11}$$

$$= P_k^T (g^k - Gx^k + G(x^k - P_k(P_k^T G P_k)^{-1} P_k^T g^k)) \tag{12}$$

$$= P_k^T g^k - P_k^T G P_k (P_k^T G P_k)^{-1} P_k^T g^k \tag{13}$$

$$= 0. \tag{14}$$

By expanding P_k into column vectors we see this means:

$$P_k^T g^{k+1} = [p^0 \cdot g^{k+1},\ p^1 \cdot g^{k+1}, \cdots\ p^k \cdot g^{k+1}]^T = [0, 0, \cdots 0]^T \tag{15}$$

and therefore

$$p^i \cdot g^{k+1} = g^{k+1} \cdot p^i = 0,\ \ i = 0, 1, 2, \cdots k \,. \tag{16}$$

Now if we assert that all the x^k to this point have been found in the same way, it must be true that for $j = 1, 2, \cdots k$

$$p^i \cdot g^j = g^j \cdot p^i = 0,\ \ i < j \,. \tag{17}$$

Thus the gradient vector g^j is orthogonal to every earlier p^i vector, as advertised.

With this information let us calculate the product $P_k^T g^k$ at the end of (10):

$$P_k^T g^k = [p^0 \cdot g^k,\ p^1 \cdot g^k, \cdots\ p^k \cdot g^k]^T = [0, 0, \cdots 0, \alpha\,]^T \tag{18}$$

where $\alpha = p^k \cdot g^k$.

So far the only property assumed of the p^i has been linear independence, needed for the inverse in (8). Let us now assert that we would like another property (which we will have to build into process somehow): let us propose that the vectors p^i are mutually **conjugate** under the action of G. This means that they orthogonal in the G inner product, or explicitly that

$$(p^i,\, p^j)_G = p^i \cdot Gp^j = 0,\;\; i \neq j\,. \tag{19}$$

Then the matrix $P_k^T G P_k$ in (10) becomes a diagonal matrix. Combining that fact with (18), which is always true, the expression x^{k+1} in (10) simplifies to

$$x^{k+1} = x^k + \gamma_k p^k \tag{20}$$

which is (1). In other words, when we started, the search for the minimum at step k was over the complete set of previous vectors p^j, but with conjugacy we find only the most recent vector need be searched over to achieve the optimal result. The parameter γ_k which we happen to know is

$$\gamma_k = -\frac{\alpha}{p^k \cdot Gp^k} = -\frac{p^k \cdot g^k}{p^k \cdot Gp^k} \tag{21}$$

could be found by a line search, and would have to be if this were a linearization of a nonquadratic system.

To summarize: if we can somehow arrange the vectors p^i to be mutually conjugate, they are the search directions at each iterative step, and at the end of that step, F has achieved its minimum over the space spanned by the vectors $p^0, p^1, \cdots p^k$. Since these vectors are linearly independent and at step $n-1$ there are n of them, they must span $\mathbb{R}^n$, and therefore at this last step we must have the global minimum of F over all vectors in $\mathbb{R}^n$. Our task is to set up a scheme for producing search direction vectors p^i with the property of conjugacy under G.

We set about building the p^i from the available gradients as follows. First we take $p^0 = -g^0$ (the steepest descent direction; why?). Subsequently we say

$$p^k = -g^k + \sum_{j=0}^{k-1} \beta_{kj}\, p^j \tag{22}$$

that is, the new direction is found from the current gradient and a linear combination of previous search directions. In what follows we work towards determining the values of the unknown coefficients β_{kj} in this expansion. By a simple rearrangement, it follows from the recipe (22) that g^k is a linear combination of the p^j up to $j = k$: Consider $i < k$ and dot a gradient vector with any earlier gradient vector:

$$g^k \cdot g^i = g^k \cdot \sum_{j=0}^{i} \sigma_j p^j = \sum_{j=0}^{i} \sigma_j g^k \cdot p^j \tag{23}$$

$$= 0 \tag{24}$$

because of (17). So the gradient vectors are mutually orthogonal too!

To discover the coefficients β_{kj} we make use of the mutual conjugacy of the p^i vectors — we pre-multiply (22) by G, then dot on the left with p^i:

$$p^i \cdot Gp^k = -\, p^i \cdot Gg^k + \sum_{j=0}^{k-1} \beta_{kj}\, p^i \cdot Gp^j\,. \tag{25}$$

Then for $i < k$, because of the conjugacy, (19), the left side vanishes and so do most of the terms in the sum:

$$0 = -\, p^i \cdot Gg^k + \beta_{ki}\, p^i \cdot Gp^i, \quad i < k \tag{26}$$

$$= -\, g^k \cdot Gp^i + \beta_{ki}\, p^i \cdot Gp^i\,. \tag{27}$$

From (9), the definition of g^i, and using (20) we see

$$g^{i+1} - g^i = G(x^{i+1} - x^i) = \gamma_i G p^i\,. \tag{28}$$

This allows us to substitute for Gp^i in (27):

$$0 = -\frac{1}{\gamma_i}\, g^k \cdot (g^{i+1} - g^i) + \beta_{ki}\, p^i \cdot Gp^i\,. \tag{29}$$

But now the orthogonality of the gradients, (24), means that when $i < k-1$ the first term on the right automatically vanishes too; since $p^i \cdot Gp^i$ must not be zero,

$$\beta_{ki} = 0, \quad i < k-1\,. \tag{30}$$

Hence we have just shown that to get conjugacy of search directions, the new search direction at each step involves the current gradient and the previous direction only; (22) has become:

$$p^k = -\, g^k + \beta_{k,k-1}\, p^{k-1} \tag{31}$$

which is of course (2). Finally we need to find the coefficient $\beta_{k,k-1}$ explicitly. Premultiply (31) by G then dot with p^{k-1}; conjugacy makes $p^{k-1} \cdot Gp^k$ on the left side vanish, and so, rearranging we find

$$\beta_{k,k-1} = \frac{p^{k-1} \cdot Gg^k}{p^{k-1} \cdot Gp^{k-1}} \tag{32}$$

$$= \frac{(g^k - g^{k-1}) \cdot g^k}{g^{k-1} \cdot g^{k-1}} \tag{33}$$

$$= \frac{g^k \cdot g^k}{g^{k-1} \cdot g^{k-1}} = \frac{\| g^k \|^2}{\| g^{k-1} \|^2} \tag{34}$$

where (33), (34) follow from applications of (16), (24) and (28). The form (33) is used in the nonquadratic application (3) rather than (34) because when the problem is not quadratic, orthogonality of the successive gradients is only approximate.

Powerful as CG certainly is, it still may require a lot of numerical work when the dimension of the system becomes very large. Then there a further tricks that can improve the convergence rate, but they are dependent on special structure a particular problem may exhibit, and are not generally available. The concept is called **preconditioning**, and is covered in Golub and Van Loan, Chapter 10.

Bibliography

Gill, P. E., Murray, W. and Wright, M. H., *Practical Optimization,* Academic Press, New York, 1981.

A treasure trove of numerical methods for every kind of optimization problem: linear, nonlinear, constrained, unconstrained, sparse, full, linear programming.

Golub, G., and Van Loan , C., *Matrix Computations*, 3rd Edition, Johns Hopkins Univ. Press, 1996.

The one reference for matrix linear algebra.

Lawson, C. L., and Hanson, R. J. , *Solving Least Squares Problems*, 1974.

Classic text for full analysis of QR and SVD in least squares.

Strang, G., *Introduction to Applied Mathematics*, Wellesley-Cambridge, 1986.

Readable treatment of many topics, though sometimes a little off base.

19. Application to the Gravity Anomaly Problem

The nonlinear gravity inversion problem in GIT was solved there by means of Occam and by B-G creeping. As a finale we will apply steepest descents and conjugate gradients to it. We have already computed (in GIT) the Fréchet Derivative for the problem, but as I want to stress here, those derivatives do not all need to be stored, or a Gram matrix inverted for SD or CG. This makes them more suitable for very large problems, which the gravity problem is not, of course.

First we need a scalar function to minimize. This immediately points up one of the disadvantages of these optimization methods: *they are designed for unconstrained optimization*. If we are seeking a regularized solution we will need to minimize something like

$$U(h, \lambda) = \sum_{j=1}^{N} (\hat{d}_j - F_j[h])^2 + \lambda \,||\, h \,||^2 \tag{1}$$

where λ is the unknown Lagrange multiplier whose value we discover by achieving the appropriate misfit. In nonlinear problems, however, the first order of business is to get *any* solution at all, and so initially we might choose λ to be very small, or zero. In case of the gravity problem we first want to fit the data almost exactly, and since the topography is order of magnitude unity, setting $\lambda = 0.01$ will insure most emphasis is placed on the first term. The derivative of U is just a vector in the discretized form: $\nabla U \in \mathbb{R}^L$, where $L =$ the number sample points in $h(x)$; while the Fréchet derivative is approximated by the matrix $D \in \mathbb{R}^{N \times L}$. We easily calculate that

$$\nabla U = 2D^T(\hat{d} - F) + 2\lambda h \tag{2}$$

where $\hat{d} - F \in \mathbb{R}^N$ is the vector of misfits. We should not forget to note that:

$$d_j = F_j[h] = \mathcal{G}\Delta\rho \int_0^a \ln\left[\frac{(x - x_j)^2 + h(x)^2}{(x - x_j)^2}\right] dx \tag{3}$$

$$D_j(x) = \frac{2\mathcal{G}\Delta\rho\, h(x)}{(x - x_j)^2 + h(x)^2} \tag{4}$$

The discretized form of (3) and (4) follows easily by replacing the function $h(x)$ with the vector of samples $[h_1, h_2, \cdots h_L]$. In the calculations that follow we take $L = 50$.

First we perform the minimization using steepest descent steps, starting at the model that is a constant $h(x) = 1$ km; $\lambda = 0.01$. The squared norm of misfits is shown in Figure 19.1, plotted against step number: each step involves the line search along the steepest descent direction. At the end of 50 line searches the penalty function U in (1) was reduced to $0.327\,\mathrm{mGal}^2$ and the norm of misfits was 0.391 mGal. Next we perform the conjugate gradient minimization, starting at the same initial

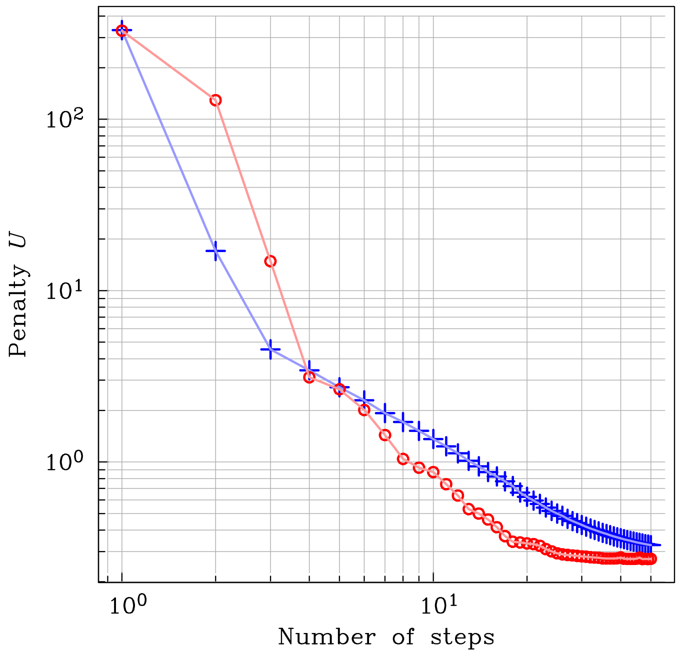


Figure 19.1: Penalty function U after n line searches with (+) steepest descents; (o) conjugate gradients.

guess. This is shown in the Figure too. Notice how the initial performance is worse than steepest descents, but then the method beats steepest descents: after 50 steps we have a penalty of $0.272\,\mathrm{mGal}^2$ and a misfit 2-norm of 0.300 mGal. The models are shown on the next page in Figure 19.2.

At first glance these methods appear to be at a severe disadvantage to Occam, which needed only four line searches to reach a similar misfit level. But that is not necessarily the case, as we now discuss. Each line search in Occam requires the solution of the linear system of equations

$$\left(\frac{1}{\mu} I + D\,D^T\right)\alpha = \hat{d} \qquad (5)$$

(This is (25) on p 315 of GIT.) This uses on the order of N^3 computer operations, since $D\,D^T \in \mathbb{R}^N$. Whereas the single line search in SD or CG involves nothing more than operations of addition and subtraction of vectors, for which the computer costs are only order N. In fact, the largest cost in each step is the evaluation of ∇U, which here takes order N^2 operations. For large data sets CG (or SD) have a much lower cost per step. In this light, the cost for 50 steps of CG is about the same as 4 steps of Occam, since $N = 12$. So when N is very large, say hundreds or even thousands, CG may be the only practical approach.

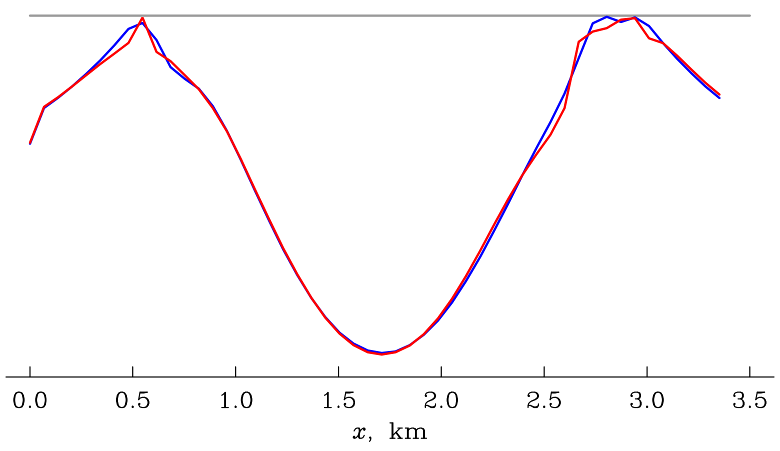


Figure 19.2: Valley profiles obtained by steepest descents (blue); conjugate gradients (red).