OPTIMIZATION

Class Notes by Bob Parker

1. Mathematical Optimization and Geophysical Modeling

The subject of optimization theory in mathematics usually posed as follows. We are given a real function $f(\mathbf{x})$ of the real vector argument $\mathbf{x} \in \mathbb{R}^n$; that is, in mathematical notation $f : \mathbb{R}^n \to \mathbb{R}$. We wish to find the vector $\mathbf{x}_*$ such that $f(\mathbf{x}_*)$ is as small as possible. This is written in the literature as

$$\mathbf{x}_* = \arg \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) . \tag{1}$$

The function f is called the **objective** or the **penalty** function. If f is smooth, say continuously differentiable, then the classic solution to this problem is to look for the stationary points, the places where the gradient vanishes:

$$\frac{\partial f}{\partial x_*^j} = 0, 1, 2, \cdots n; \quad \text{or} \quad \nabla f(\mathbf{x}_*) = 0 . \tag{2}$$

So (2) defines n (possibly nonlinear) equations in n unknowns, and if we can solve that system we have the stationary points, which we can check for local behavior. But numerically this not how (2) is solved, unless the equations are linear (**linear least squares**), and even then not always, as we will see. We can already anticipate a possible problem with (2): f may not be smooth enough to have a derivative, for example suppose $f(x_1, x_2) = | 1 + x_1 + x_2 |$; then (2) won't work.

Equation (1) is an example of an **unconstrained** optimization problem. Another important class of problems says that the coordinates $\mathbf{x}$ are not free, but must themselves satisfy certain conditions, say for the moment

$$g(\mathbf{x}) = 0 . \tag{3}$$

This is called a **constraint** equation. So that the optimization now reads

$$\mathbf{x}_* = \arg \min_{g(\mathbf{x}) = 0} f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n \tag{4}$$

and this is an example of a **constrained optimization problem**. For smooth functions you will recall this is solved by introducing a **Lagrange multiplier** λ and generating a new function F defined by

$$F(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda\, g(\mathbf{x}) . \tag{5}$$

Then the classic theory says the stationary points of (5) where

$$\frac{\partial F}{\partial x_j} = 0, 1, 2, .. n; \quad \text{and} \quad \frac{\partial F}{\partial \lambda} = 0 \tag{6}$$

are also stationary points of the constrained system. naturally we can generalize (3) to be a whole collection of functions. Then a Lagrange multiplier is introduced for each constraint. Obviously we just differentiate now on all the λ s.

Another way in which constraints can be applied is through an **inequality**, for example

$$\mathbf{x}_* = \arg \min_{0 \le g(\mathbf{x})} f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n . \tag{7}$$

In this case a **John multiplier** is added, which looks just like λ in (5). There are then two possibilities to be considered: either the constraint is *active*, and then $g(\mathbf{x}_*) = 0$ and we solve as before with the Lagrange multiplier; or it is *inactive*, meaning $g(\mathbf{x}_*) > 0$, and then we can solve by setting $\lambda = 0$ and differentiating. Both cases have to be investigated. Things become quite serious when many inequalities are included rather than just one. We will not discuss this more difficult case any further here, but even when g and f are linear, it is the subject of a huge literature about **linear programming**; for nonlinear f or g we have what is called a mathematical programming problem.

What does this have to do with geophysics? The reason geophysicists are always using numerical optimization is because they want to fit models with unknown parameters. For example, there is a model earth with layers of different seismic velocity and travel time data to be match. Suppose the model parameters are $x_1, x_2, \cdots x_n$ and the measured numbers are $d_1, d_2, \cdots d_m$ and the model theory says that we expect

$$\begin{aligned} d_1 &= f_1(x_1, x_2, \cdots x_n) \\ d_2 &= f_2(x_1, x_2, \cdots x_n) \\ &\cdots \\ d_m &= f_m(x_1, x_2, \cdots x_n) \end{aligned} \tag{8}$$

where the f_j are functions modeling the behavior of the earth. We adopt an obvious vector notation now:

$$\mathbf{d} = \mathbf{f}(\mathbf{x}) . \tag{9}$$

Usually the number of data exceeds the number of parameters ($m > n$), but it doesn't have to. In either case one is never going to be able to satisfy (9) exactly with real field measurements. Why is that? So instead we settle for the parameter set that *best approximates a solution in some sense*. The almost universally adopted strategy is to replace (9) with

$$\mathbf{x}_* = \arg \min_{\mathbf{x} \in \mathbb{R}^n} \| \mathbf{d} - \mathbf{f}(\mathbf{x}) \| . \tag{10}$$

And (10) is an example of an optimization problem! The traditional choice of norm here is

$$\| \mathbf{y} \| = \left(\sum_{k=1}^{m} y_k^2 \right)^{1/2} \tag{11}$$

known as the 2-norm or the **Euclidean norm**. Then (11) is a **Least Squares** (LS) problem. If the function **f** is just a linear map (that is represented by a matrix multiply) then (11) is a linear least squares problem, the easiest kind. But most often **f** is not a linear map. And you can see that if the parameters are velocities or layer thicknesses, it will be very natural to insist that they are positive, leading to problems like (7), but of a more general kind with a constraint on every parameter.

In the following notes we concentrate on unconstrained problems, although we will occasionally use Lagrange multipliers. And least squares problems will be our main topic.

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. Philip Gill is a UCSD Math Professor.

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

The one reference for numerical matrix linear algebra.

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

Old but still worth a look.

2. Simple Least Squares Problems

Least squares problems are examples of optimization problems that involve the simplest of norms. We are going to solve these problems in several ways, to illustrate a the use of use of Lagrange multipliers and a few other things. The bible for the numerical aspects of LS is the ancient Lawson and Hanson, 1974. Informally, a norm is a real number that measures the size of an element in a linear vector space. Assigning a norm to a linear vector space is said to **equip** the space with a norm. Most linear vector spaces can be so equipped (spaces of functions, operator, matrices, etc), but here we will consider only the simplest norm for $\mathbb{R}^m$, the Euclidean length 1(11). Note from here on I will not use bold **x** for a vector, but just use $x \in \mathbb{R}^n$ as the math literature does. Here is an approximation problem often encountered in geophysics, the classical least squares problem. We will state it as a problem in linear algebra.

Suppose we are given a collection of n vectors $a_k \in \mathbb{R}^m$ and we wish to approximate a target vector y by forming a linear combination of the a_k; when $n < m$, as we shall assume, we will not expect to be able to do this exactly, and so there will be an error, called in statistics the **residual**:

$$r = y - \sum_{k=1}^{n} x_k a_k \,. \tag{1}$$

In data analysis, straight-line regression is in this form, or fitting any simple linear model to a data set. In numerical analysis you might want to approximate a complicated function by a polynomial. To get the *best approximation* in some sense, we want the size of the vector $r \in \mathbb{R}^m$ to be as *small* as possible. Once we've picked a way to measure the size, we have a minimization problem. The simplest norm for computational purposes is the Euclidean length, and this leads to the **overdetermined least squares problem**. If we can rewrite (1) in matrix notation:

$$r = y - Ax \tag{2}$$

where $x \in \mathbb{R}^n$ and the matrix $A \in \mathbb{R}^{m \times n}$ is built from columns that are the a_k:

$$A = [a_1, a_2, \cdots a_n] \,. \tag{3}$$

So the minimization problem is to solve

$$\min_{x \in \mathbb{R}^n} \; f(x) \tag{4}$$

where

$$f(x) = \| r \|^2 = r^T r = (y - Ax)^T (y - Ax) \,. \tag{5}$$

Obviously we can square the norm if it simplifies the algebra.

I will offer you several solutions to this problem, some of which may be unfamiliar. First the classical approach, which is to multiply descend into subscripts, and differentiate:

$$f = \sum_{j=1}^{m} r_j^2 . \tag{6}$$

Then

$$\frac{\partial f}{\partial x_k} = 2 \sum_{j=1}^{m} r_j \frac{\partial r_j}{\partial x_k} \tag{7}$$

$$= 2 \sum_{j=1}^{m} (y_j - \sum_{i=1}^{n} A_{ji} x_i) \times (-A_{jk}) \tag{8}$$

$$= -2 \sum_{j=1}^{n} A_{jk} y_j + 2 \sum_{i=1}^{n} (\sum_{j=1}^{m} A_{jk} A_{ji}) x_i \tag{9}$$

and this is true for each value of k. At the minimum we set all the derivatives to zero, which leads to:

$$\sum_{i=1}^{n} (\sum_{j=1}^{m} A_{jk} A_{ji}) x_i = \sum_{j=1}^{n} A_{jk} y_j, \quad k = 1, 2, \cdots n . \tag{10}$$

Translated into matrix language these are the so-called **normal equations**:

$$A^T A x = A^T y . \tag{11}$$

Note that $A^T A$ is a square n by n matrix, and the left side is a column n-vector. So the unknown expansion coefficients are found by solving this system of linear equations, formally by writing

$$x = (A^T A)^{-1} A^T y \tag{12}$$

a result that should be familiar to you.

This answer looks ugly and seems to have no intuitive content. But a geometrical interpretation can help a lot. Suppose we assume that the vectors a_k are linearly independent (which they must be if we can write (12)). Then the collection of all vectors that can be formed from linear combinations of them is a **subspace** of $\mathbb{R}^m$ which we will call $\mathcal{A}$; it is the column, or **range space** of the matrix A, and so $\mathcal{A} = \mathcal{R}(A)$. The approximation problem we are solving can be stated as finding the vector in $\mathcal{A}$ that comes as close to y as possible. We rewrite (11) as

$$0 = A^T (A x - y) = A^T r \tag{13}$$

$$= \begin{bmatrix} a_1^T r \\ a_2^T r \\ \vdots \\ a_n^T r \end{bmatrix} . \tag{14}$$

Remember the zero on the left is the vector $0 \in \mathbb{R}^n$. So what this equation is saying is that the residual vector, the error in the approximation to y, is orthogonal to every one of the basis vectors of the space $\mathcal{A}$ (because

$a_1^T r$ is the dot product between a_1 and r). And that is what you might expect from a geometrical interpretation as shown in Figure 5.1.

Let us give a name to the approximation we have created, let $Ax = \tilde{y}$. Then $\tilde{y}$ is called the **orthogonal projection** of y into the subspace $\mathcal{A}$. The idea of a projection relies on the **Projection Theorem** for Hilbert spaces. The theorem says, that given a subspace like $\mathcal{A}$, every vector can be written uniquely as the sum of two parts, one part that lies in $\mathcal{A}$ and a second part orthogonal to the first. The part lying in $\mathcal{A}$ is orthogonal projection of the vector onto $\mathcal{A}$. Here we have

$$y = \tilde{y} + r\,. \tag{15}$$

There is a linear operator, $P_{\mathcal{A}}$ the projection matrix, that acts on y to generate $\tilde{y}$, and we can see that

$$P_{\mathcal{A}} = A\,(A^T A)^{-1} A^T\,. \tag{16}$$

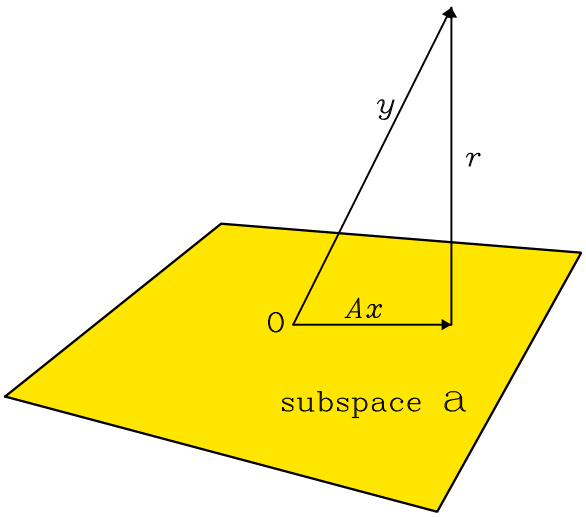
The general definition of a Projection Matrix is that $P = P^T$ and $P^2 = P$. The second property is natural for a projection, because acting once creates a vector falling into the given subspace, acting again leaves it there. Verify these properties for $P_{\mathcal{A}}$.

From a numerical computational perspective, it turns out that (12) may suffer unnecessarily from a loss of precision. We won't go into detail now, but we describe a completely different way of looking at the least-squares (LS) problem. Let us return to Householder's **QR factorization** of a matrix, mentioned briefly in the previous section: every matrix $A \in \mathbb{R}^{m\times n}$ where A is tall (meaning $m \geq n$) can be written as the product:

$$A = Q\,R \tag{17}$$

where $Q \in \mathbb{R}^{m\times m}$, $R \in \mathbb{R}^{m\times n}$, and Q is *orthogonal*, and R is *upper*

Figure 2.1: Orthogonal projection of y onto the column space of A.



triangular; recall that upper triangular means all zeros below the diagonal:

$$R = \begin{bmatrix} R_1 \\ O \end{bmatrix} \tag{18}$$

where $R_1 \in \mathbb{R}^{n \times n}$. For how to the QR is found in practice and why QR is numerically stable, see GIT 1.13 and the references there. To solve the LS problem we look to the Euclidean norm of r:

$$\| r \| = \| y - Ax \| = \| y - QRx \| . \tag{19}$$

Recall that $Q\,Q^T = I$, so

$$\| r \| = \| QQ^T y - QRx \| = \| Q(Q^T y - Rx) \| . \tag{20}$$

Now recall that the length of a vector is unchanged under mapping with an orthogonal matrix: $\| z \| = \| Qz \|$. So

$$\| r \| = \| Q^T y - Rx \| = \| \hat{y} - Rx \| . \tag{21}$$

Next square the norm and partition the arrays inside the norm into two parts, the top one with n rows:

$$\| r \|^2 = \left\| \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} - \begin{bmatrix} R_1 \\ O \end{bmatrix} x \right\|^2 = \| \hat{y}_1 - R_1 x \|^2 + \| \hat{y}_2 \|^2 . \tag{22}$$

The second term $\| \hat{y}_2 \|^2$ in the sum is indifferent to the choice of x; but we can reduce the first term to zero by solving

$$R_1 x = \hat{y}_1 . \tag{23}$$

So this must be the solution to finding the smallest norm of r. Because R_1 is upper triangular, (23) is solved by **back substitution**, starting at the bottom and working upwards, which is very simple. This doesn't look like a very efficient way to find the LS answer, but it can be made very efficient: for example, there is no need to store the matrix Q, because one can calculate the vector $\hat{y} = Q^T y$ without it. The QR factorization is competitive with the normal equations for execution times (it is slightly slower), but it is numerically much more stable against the accumulation of numerical error. Therefore, for not too large systems, QR is the proper way to go. In MATLAB, while you can get the QR factors with the call

```
[Q R] = qr(A);
```

the LS problem is *solved automatically* for you by the QR method if you simply write

```
x = A\y;
```

Finally, suppose you substitute the QR factors into the expression for the projection matrix. We find after some algebra that

$$P_{\mathcal{A}} = Q^T \begin{bmatrix} I_n & 0 \\ 0 & O \end{bmatrix} Q \tag{24}$$

where $I_n \in \mathbb{R}^{n\times n}$ is the unit matrix and the rest of the entries are zero. Numerically this way of finding the projection is very stable because one never needs to solve a linear system. But (24) also shows that if one imagines rotating the data space onto new coordinates with Q, the projection operator then becomes the matrix in the middle of (24), which is the projection that simply zeros out all the components of a vector after the nth one.

Exercises

2.1 Show that the last $m-n$ columns in the factor Q of the QR factorization are never used in the LS calculation.

2.2 The *Gram-Schmidt* process is a method of creating a set of orthonormal vectors from a given ordered set of linearly independent vectors by forming linear combinations of one, then two, then three, etc, of the given vectors. Show how the QR process does the same thing.
Hint: First show that the inverse of an right triangular matrix is also right triangular.

2.3 Show how Singular Value Decomposition can be used to solve the over-determined least squares problem.

3. Lagrange Multipliers and More Least Squares

There is more. We now consider solving minimization problems with **Lagrange Multipliers**. For proofs see GIT 1.14 and the references mentioned there. The minimization we solved in (5) was an example of an **unconstrained minimization** in which we found the smallest possible value of a function. But suppose there is a side condition, called a **constraint**, that must hold for all solutions. The Figure 6.1, taken from GIT, show the general idea for single condition. If the constraint condition is expressed the form:

$$g(x) = 0 \tag{1}$$

then the minimum of the constrained problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{with} \quad g(x) = 0 \tag{2}$$

occurs at a stationary point of the *unconstrained* function

$$u(x, \mu) = f(x) - \mu g(x) \tag{3}$$

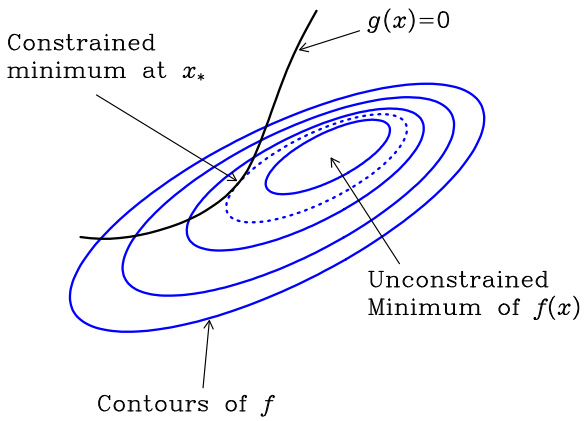
where we must consider variations of x and μ; of course μ is called a Lagrange multiplier. If there are q constraint conditions in the from $g_k(x) = 0,\ k = 1, 2, \cdots q$, each would be associated with its own Lagrange multiplier:

$$u(x, \mu_1, \mu_2, \cdots \mu_n) = f(x) - \sum_{k=1}^{q} \mu_k g_k(x)\,. \tag{4}$$

As an example consider again the overdetermined LS problem. We wish to find the minimum of the function $f(r) = \|r\|^2$, with $r \in \mathbb{R}^m$. As an unconstrained problem the answer is obviously zero. But we have the following m conditions on r:

$$0 = y - Ax - r \tag{5}$$

Figure 3.1: An optimization problem with one constraint.



where $y \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$ are known, while the vector $x \in \mathbb{R}^n$ is unknown. So, writing (5) out in components and giving each row its own Lagrange multiplier, (4) becomes for this problem

$$u(x, \mu_l) = \sum_{j=1}^{m} r_j^2 - \sum_{j=1}^{m} \mu_j (y_j - \sum_{k=1}^{n} a_{jk}\, x_k - r_j)\,. \tag{6}$$

Differentiating over r_i, x_i and μ_i, the stationary points of u occur when

$$\frac{\partial u}{\partial r_i} = 0 = 2r_i + \mu_i \tag{7}$$

$$\frac{\partial u}{\partial x_i} = 0 = -\sum_{j=1}^{m} a_{ji}\mu_j \tag{8}$$

$$\frac{\partial u}{\partial \mu_i} = 0 = y_i - \sum_{k=1}^{n} a_{ik}\, x_k - r_i\,. \tag{9}$$

Equation (7) says the vector of Lagrange multipliers $\mu = -2r$; then using this fact and translating (8), (9) into matrix notation:

$$A^T \mu = -2A^T r = 0 \tag{10}$$

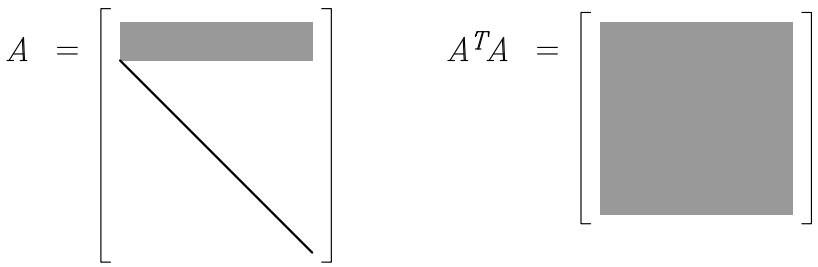
$$Ax - r = y\,. \tag{11}$$

If we multiply (11) from the left with A^T and use (10) we get the normal equations 2(11) again. But let us do something else: we combine (10) and (11) into a single linear system in which the unknown consists of both x and r:

$$\begin{bmatrix} -I_m & A \\ A^T & O_n \end{bmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} y \\ 0 \end{pmatrix} \tag{12}$$

where $I_m \in \mathbb{R}^{m \times m}$ is the unit matrix, where $O_n \in \mathbb{R}^{n \times n}$ is square matrix of all zeros. This system has the same content as the normal equations, but solves for the residual and the coefficients at the same time. If A is sparse, (12) can be a better way to solve the LS problem than by the

Figure 3.2: Loss of sparseness in forming the normal equations.



normal equations or by QR, particularly as QR does not have a good adaptation to sparse systems. The situation is illustrated for a common form of overdetermined problem in Figure 6.2.

We turn next to the so-called **undetermined least-squares** problem. While the overdetermined LS problem occurs with monotonous regularity in statistical parameter estimation problems, the underdetermined LS problem looks quite a lot like an *inverse problem*. Instead of trying to approximate the known y by a vector in the column space of A , we can match it exactly: we have

$$y = Ax\,. \tag{13}$$

where $A \in \mathbb{R}^{m\times n}$, but now $m < n$ and A is of full rank. This is a finite-dimensional version of the linear forward problem, in which the number of measurements, y , is *less* than the number of parameters in the model x . So instead of looking for the smallest error in (13), which is now zero, we ask instead for the *smallest model, x*. We are performing a simplified regularization, in which size, here represented by the Euclidean length, stands for simplicity. This problem is solved just as the last one, with a collection of m Lagrange multipliers to supply the constraints given by (13), but with $\|x\|^2$ being minimized instead of $\|r\|^2$. I will skip the details and report the well-known result: we must solve

$$A\,A^T\mu = y \tag{14}$$

for μ, where $\mu \in \mathbb{R}^m$ is the Lagrange multiplier vector; then the solution vector is

$$x = A^T\mu\,. \tag{15}$$

The x of (15) has the smallest 2-norm of any of the infinitely many solutions to (13).

Like the normal equations, (14) too suffers from poor conditioning numerically, however. And as before QR comes to the rescue, but in a cute way. Recall that the classic QR factorization works only if $m \geq n$, here that is violated. So we write instead that

$$A^T = QR \quad \text{or} \quad A = R^TQ^T\,. \tag{16}$$

Then (13) can be written

$$y = R^TQ^Tx = R^T\hat{x} \tag{17}$$

where $\hat{x} \in \mathbb{R}^m$ is just Q^Tx. Then, since Q is an orthogonal matrix

$$x = Q\hat{x} \tag{18}$$

and it follows that x and $\hat{x}$ have the same norm, ie. Euclidean length. Recall that R is upper triangular; so (17) is

$$y = [R_1^T \;\; O]\begin{pmatrix}\hat{x}_1\\ \hat{x}_2\end{pmatrix} \tag{19}$$

where $R_1 \in \mathbb{R}^{n\times n}$, and $\hat{x}_1 \in \mathbb{R}^n$. If we multiply out the partitioned matrix we see that

$$y = R_1^T \hat{x}_1 + O\,\hat{x}_2 = R_1^T \hat{x}_1\,. \tag{20}$$

Because the second term vanishes, (20) shows that we can choose $\hat{x}_2$ (the bottom part of $\hat{x}$) in any way we like and it will not affect the match to the data: only $\hat{x}_1$ influences that. So we match the data exactly by solving the system

$$R_1^T \hat{x}_1 = y\,. \tag{21}$$

Now observe that

$$\|\,x\,\|^2 = \|\,\hat{x}\,\|^2 = \|\,\hat{x}_1\,\|^2 + \|\,\hat{x}_2\,\|^2\,. \tag{22}$$

So to match the data we solve (21), then to get the smallest norm we simply put $\hat{x}_2 = 0$. Thus $\hat{x}$ has been found that minimizes the norm, and the corresponding x is recovered from (18).

The undetermined LS problem is artificial in the sense that (13) the condition that the model fit the data *exactly* is unrealistic: if there is noise in the data y, we must not demand an exact fit. It is more realistic to say that we would be satisfied with a reasonably close fit, as measured by the Euclidean norm; so replace (13) with

$$\|\,Ax - y\,\| = \gamma \tag{23}$$

where we get choose γ from a statistical criterion that depends on the noise in y. Now we need a single Lagrange multiplier to apply (23). To complicate things slightly more, instead of minimizing the norm of x, we will minimize

$$f(x) = \|\,Px\,\|^2 \tag{24}$$

where $P \in \mathbb{R}^{p\times n}$ is a matrix that suppresses undesirable properties, for example, it might difference x to minimize slopes instead of magnitudes. Now we have the unconstrained function

$$u(x,\mu) = \|\,P\,x\,\|^2 - \mu(\gamma^2 - \|\,Ax - y\,\|^2) \tag{25}$$

where I have squared the condition factor because it will simplify things later. A trivial rearrangement gives:

$$u(x,\mu) = \|\,P\,x\,\|^2 + \mu\,\|\,Ax - y\,\|^2 - \mu\gamma^2\,. \tag{26}$$

It can be shown (see GIT, Chapter 3) that $\mu > 0$. Then for a fixed value of μ, the function u can be interpreted as finding a compromise between two undesirable properties, large $P\,x$, and large data misfit. If we minimize over x with a small μ we give less emphasis to misfit and find models that keep $P\,x$ very small; and conversely, large μ causes minimization of u to yield x with small misfit. This is an example of a **trade-off** between two incompatible quantities: it is shown in GIT that decreasing the misfit always increases the penalty norm, and vice versa.

We could solve this problem by differentiating in the usual tedious way. Instead we will be a bit more clever. As usual, differentiating by μ just gives the constraint (23). The derivatives on x don't see the γ term in (26) so we can drop that term when we consider the stationary points of u with respect to variations in x:

$$\hat{u}(x) = \| P\,x \|^2 + \mu \| Ax - y \|^2 \tag{27}$$

$$= \| Px - 0 \|^2 + \| \mu^{½} Ax - \mu^{½} y \|^2 . \tag{28}$$

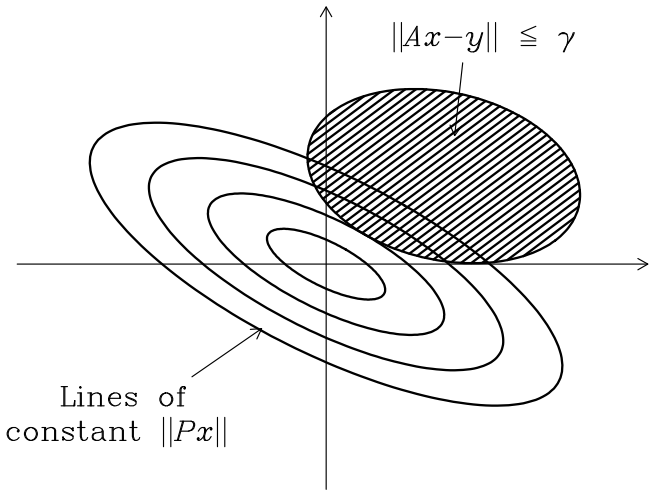
Both of the terms are norms acting on a vector; we can make the sum into a single squared norm of a longer vector, the reverse of what we did on equation 3(12):

$$\hat{u}(x) = \left\| \begin{bmatrix} P \\ \mu^{½} A \end{bmatrix} x - \begin{pmatrix} 0 \\ \mu^{½} y \end{pmatrix} \right\|^2 \tag{29}$$

$$= \| Cx - d \|^2 . \tag{30}$$

The matrix $C \in \mathbb{R}^{p+m \times n}$ must be tall, that is $p+m > n$, or the original problem has a trivial solution (Why?), (30) is just an ordinary *overdetermined least squares problem*. So for any given value of μ, we can find the corresponding x through our standard LS solution. But this doesn't take care of (23). The only way to satisfy this misfit criterion is by solving a series of versions of (30) for different guesses of μ in an iterative way, because unlike all the other systems we have met so far, this equation is nonlinear.

Figure 3.3: Minimization of $\| Px \|$ subject to $\| Ax - y \| \leq \gamma$ for $x \in \mathbb{R}^2$.



4. Lightly Parameterized Nonlinear Models

First a little digression on the penalty function. The objective in modeling is to find a function that matches observation adequately, and so we need to agree upon a measure of misfit between the predictions of the forward calculations and the observations. The traditional choice is based on the weighted 2-norm:

$$\chi^2 = \sum_{j=1}^{m} \frac{[d_j - p_j(x_1, x_2, \cdots x_n)]^2}{\sigma_j^2} \tag{1}$$

where d_j are the measured values, m of them, and σ_j are the estimated uncertainties at the 1 standard deviation level, p_j is the function that predicts values for the measurements based on the n parameters x_k. If the noise in the measurements is Gaussian, zero mean, and uncorrelated (a tall order) the minimum χ^2 has a statistical distribution like χ^2_{m-n}. Then the expected size of χ^2 is $m-n$, and that is normally the target level for model construction. If the value discovered by minimizing χ^2 is still much larger that would expected statistically, we have to reject the model.

Now we come to the optimization problem: finding the parameter values that minimize χ^2. In linear least squares problems the penalty function is **convex**, which means that $f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha) f(y)$ for all $0 \le \alpha \le 1$. Then a local minimum, one found by differentiation, say, is the smallest. Obviously for an arbitrary function that will not necessarily be true: there may be many local minima. There is no general recipe for ensuring the minimum you have found is indeed the best one, however.

The reason the optimization problem is nonlinear is that all the p_j are not linear functionals, and so they cannot be represented exactly by a matrix. However, linear least squares problems are so easily solved, the general approach is to make a **linear approximation** to the actual problem, and iterate. Here is an efficient algorithm, the **Levenberg-Marquardt** (LM) method. First for convenience, we normalize the data and the model predictions by the uncertainties to create nondimensional data and prediction functions:

$$y_j = d_j/\sigma_j; \quad g_j = p_j/\sigma_j \,. \tag{2}$$

We assume the functions g_j are twice continuously differentiable, and that we guess an initial solution with the parameters x_k^0. The Taylor series expansion for the forward problem is then

$$g_j(x_k^0 + \Delta_k) = g_j(x_k^0) + \sum_{k=1}^{n} \Delta_k \frac{\partial g_j}{\partial x_k^0} + O(\sum_k \Delta_k^2) \tag{3}$$

which can be more compactly written as

$$g(x^0 + \Delta) = g(x^0) + J\,\Delta + O\,||\,\Delta\,||^2 \,. \tag{4}$$

Here $x^0, \Delta \in \mathbb{R}^n$ and $g: \mathbb{R}^n \to \mathbb{R}^m$. The matrix $J \in \mathbb{R}^{m \times n}$, called the

Jacobian matrix, is explicitly

$$J_{jk} = [\nabla g(x^0)]_{jk} = \frac{\partial g_j}{\partial x_k^0} . \tag{5}$$

Next we substitute (4) into the nondimensional form of (2):

$$\chi^2(x^0 + \Delta) = || y - (g(x^0) + J\Delta) ||^2 + O||\Delta||^2 \tag{6}$$

$$= ||(y - g(x^0)) - J\Delta||^2 + O||\Delta||^2 \tag{7}$$

$$= ||\tilde{y} - J\Delta||^2 + O||\Delta||^2 \tag{8}$$

where we introduce the abbreviation $\tilde{y} = y - g(x^0)$. Neglecting quadratic terms, we differentiate χ^2 and obtain the standard linear LS result, obtained by translating 2(8) into vector language, that

$$\nabla\chi^2 = -2J^T(\tilde{y} - J\Delta) . \tag{9}$$

If the second-order terms are truly negligible we can find the minimum χ^2 by setting the gradient $\nabla\chi^2 = 0$; doing this in (9) we obtain the usual least-squares (LS) result, 2(11), which in our current problem gives

$$\Delta = (J^T J)^{-1} J^T \tilde{y} . \tag{10}$$

Then the approximate solution is

$$x' = x^0 + \Delta . \tag{11}$$

But suppose at the proposed solution, one cannot neglect the omitted terms in (8). Then another, more cautious approach is to evaluate a different Taylor series, the one for χ^2 itself. Using (9) we see that:

$$\chi^2(x^0 + \Delta) = \chi^2(x^0) + \Delta^T \nabla\chi^2(x^0) + O||\Delta||^2 \tag{12}$$

$$= \chi^2(x^0) - 2\Delta^T J^T \tilde{y} + O||\Delta||^2 . \tag{13}$$

In equation (13) we choose $\Delta = \mu J^T \tilde{y}$ for some scalar $\mu > 0$; then

$$\chi^2(\Delta) = \chi^2(0) - 2\mu||J^T\tilde{y}||^2 + O(\mu^2) \tag{14}$$

which shows there if the value of μ is small enough (but positive) the misfit $\chi^2(\Delta)$ is less than it is at the starting point. Solutions of the form $\Delta(\mu) = \mu J^T \tilde{y}$ are said to be on the **steepest descent** path away from the current guess x^0 , and will discuss this class of solutions in more detail in the next Section. Notice that in the linear LS problem, the true minimizer x_* does **not** generally lie on the path of steepest descent.

Finally, Levenberg-Marquardt combines the steepest descent approach (14) and the linearized LS solution (10) in the following clever way. We consider a family of vectors

$$x(\lambda) = x^0 + \Delta(\lambda) = x^0 + (\lambda I + J^T J)^{-1} J^T \tilde{y} . \tag{15}$$

When λ is small we see from (10) that $x(\lambda)$ is close to the linear LS solution; when λ is large $x(\lambda)$ lies on the steepest descent path but with a small $\mu \sim 1/\lambda$. An iterative strategy, called a **line search**, is pursued:

(o) For a starting guess x^0 evaluate $\chi^2(x^0)$.

(i) For x^0 fixed vary $\lambda > 0$ seeking the minimum of $\chi^2(x(\lambda))$; call the $x(\lambda)$ achieving the minimum misfit x'.

(ii) If $| \chi^2(x^0) - \chi^2(x') | \leq \eta$ where η is a small number (~ 0.01) stop and accept x' as the minimizing vector. If this convergence criterion is not met, set $x^0 = x'$ and go to (i).

We are guaranteed each time through the loop that χ^2 will decrease, and since its value is bounded below by zero, the process must converge. In practice the rate of converge is very satisfactory.

One practical drawback is the need to find the derivatives $\partial g_j / \partial x_k$ in (3), which can be very messy if the calculation involves recursion, as it does for both MT and resistivity sounding. Derivatives can be found approximately by finite differences, although then the converge rate near the end of the process may not be quite as good as with analytic derivatives.

5. Large-Scale Nonlinear Systems – Steepest Descents

We now study numerical optimization methods that, unlike LM, are not specifically tied to least-squares penalty functions, although of course they can be applied to them if desired. The function to be minimized is the general real-valued $f(x) = f(x_1, x_2, \cdots x_n)$, and so $x \in \mathbb{R}^n$. As in the last Section we will assume that we can obtain an analytic expression for the derivatives of f, the gradient of f:

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

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, the normal equations 2(11). Suppose now that m and n are both so large that exact solution of the normal equations is rather slow, typically on the order of $n^3/3$ flops when $n < m$. The Levenberg-Marquardt scheme requires such a solution *for every step of its line search*. If we could find a scheme that avoids the necessity of solving a linear system of equations we could speed things up and tackle problems with many thousands of unknowns, which might be impractical otherwise. Even linear least squares might benefit. 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 re-examine first the simple-minded **steepest descent**. Suppose we are in 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, repeating the argument in the previous Section, 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?

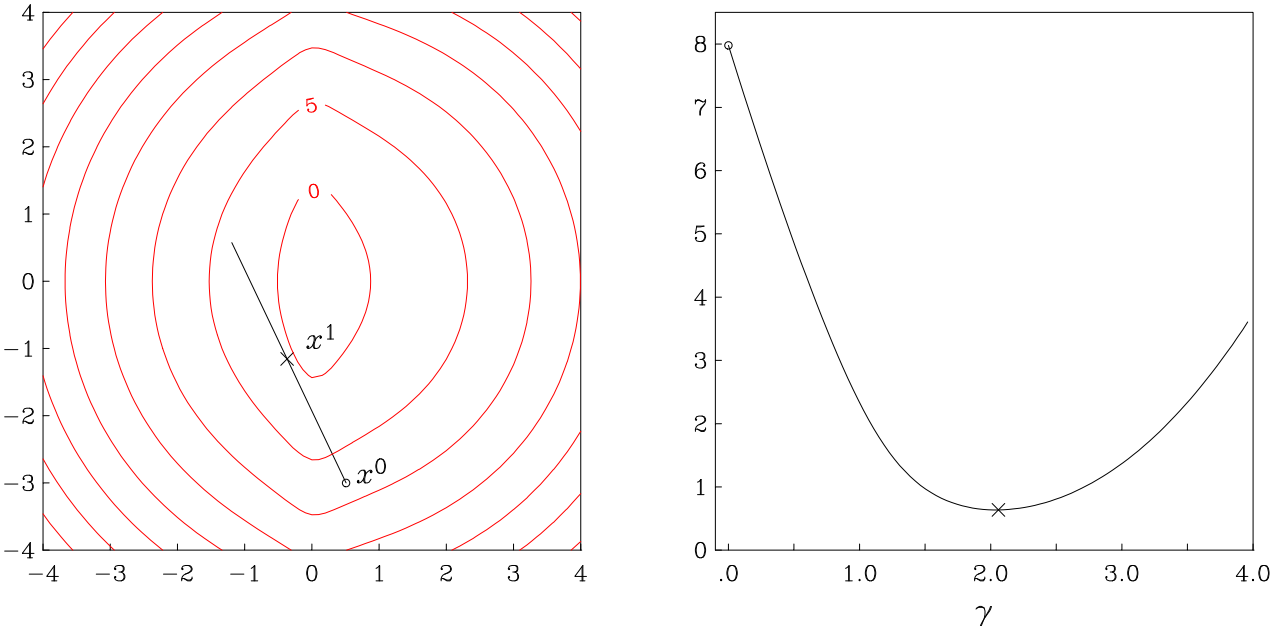
The answer to this question is fairly obvious: we use a **line search** on γ; we keep increasing γ until f starts to increase again. In other words we go to the minimum along the direction of initial steepest gradient, as illustrated below. In general this point must be determined by numerical experiment: in a systematic way we try various values of γ until the least value has been found in the direction of $\nabla f(x^0)$. 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

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

where $\gamma_k > 0$ is chosen to minimize $f(x^{k+1})$, by a line search; x^0 is an arbitrary initial guess vector. Notice that no large matrix (like the Jacobian $J \in \mathbb{R}^{m \times n}$) is even computed here, just the gradient vector $\nabla f \in \mathbb{R}^n$. And no $n \times n$ linear system of equations is solved. Thus *each step* of the line search in LM is much more expensive than *the whole line search* of steepest descents. However, steepest descents is far less efficient at locating the minimum. If the function f is a quadratic form, then LM locates the

Figure 5.1: Contours of f from (6), and values of f on the first line of steepest descent starting at x_0.



answer after only one line search because the linearized system is exact, and the normal equations give the exact answer. But steepest descent requires infinitely many iterations, because it is an iterative scheme even for quadratic objective functions.

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. The function used for illustration is:

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

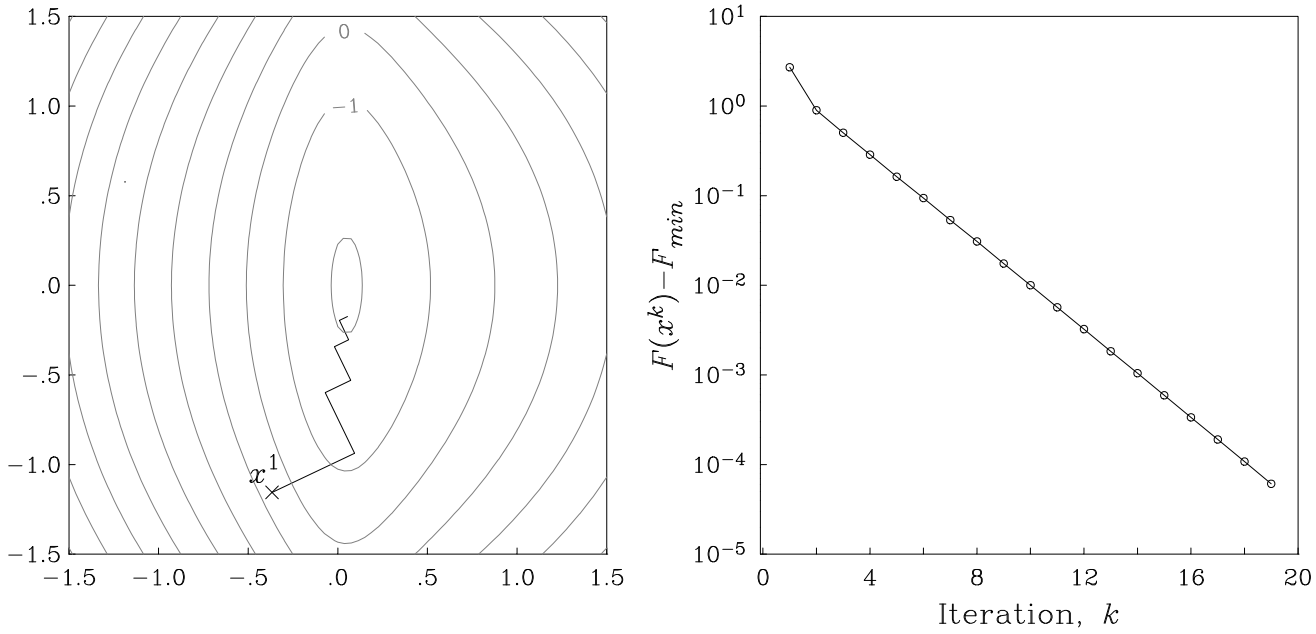
$$\nabla f = [2x_1 - 1 + 2x_1/(x_1^2 + 0.1),\ 2x_2]^T . \tag{7}$$

Obviously the penalty function here is not in the least-squares form. After a moment's thought it will be obvious to you that the new 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

Figure 5.2: Steepest descent path and convergence of penalty function for (6).



$\nabla\nabla f = H$ does not vanish at x^*, the local minimum, then the behavior of any smooth function of many variables is quadratic and is modeled by

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

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 the penalty function

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

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

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

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 roundoff), 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.

6. 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 5.2 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

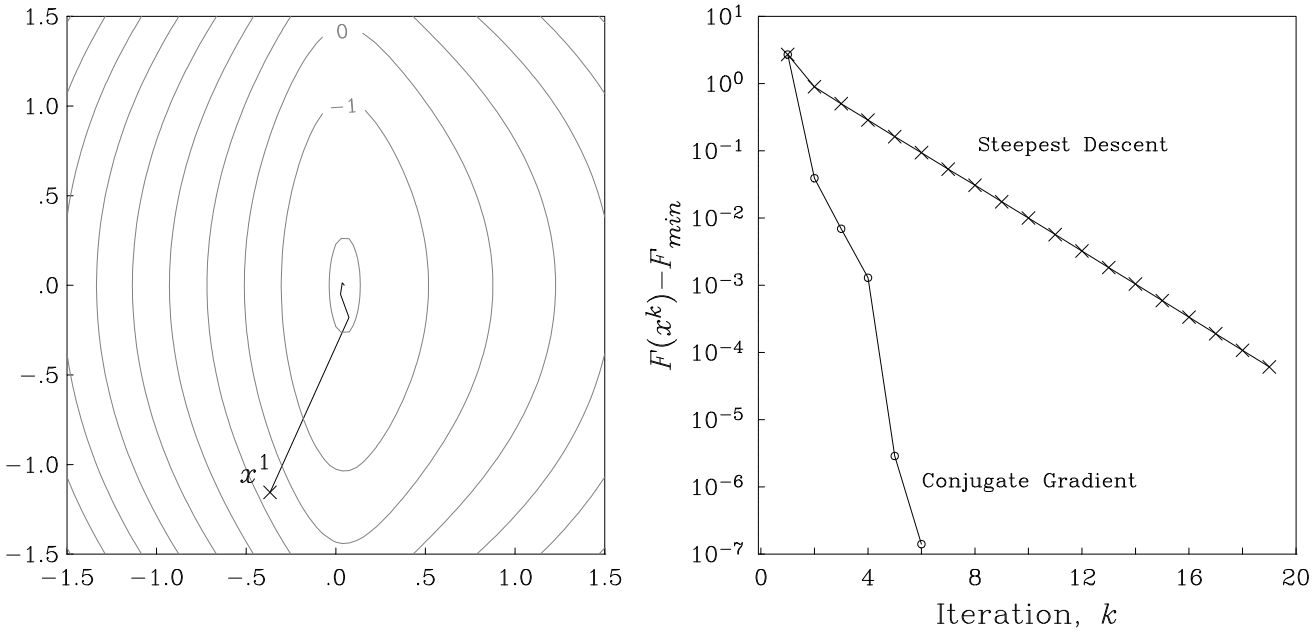
$$p^k = -\nabla f(x^k) + \beta p^{k-1}, \quad k > 0 \tag{2}$$

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

Figure 6.1: Conjugate gradient and objective convergence for (6).



determined as before by minimizing $f(x^{k+1})$ along the line.

Below 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 be if this is 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

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

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.

Parker, Robert, L., Geophysical Inverse Theory, Princeton University Press, 1994.

Referred to as GIT.

Press, W. H., Flannery, B. P., Teukolsky, S. A. & Vetterling, W. T., *Numerical Recipes in Fortran: The Art of Scientific Computing,* Cambridge University Press. 1986.

This (or for masochists, the C version) should be on every scientific programmer's shelf. One warning: the authors have sued people who distributed the code as part of their own programs.

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

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