10.5 Direction Set (Powell's) Methods in Multidimensions

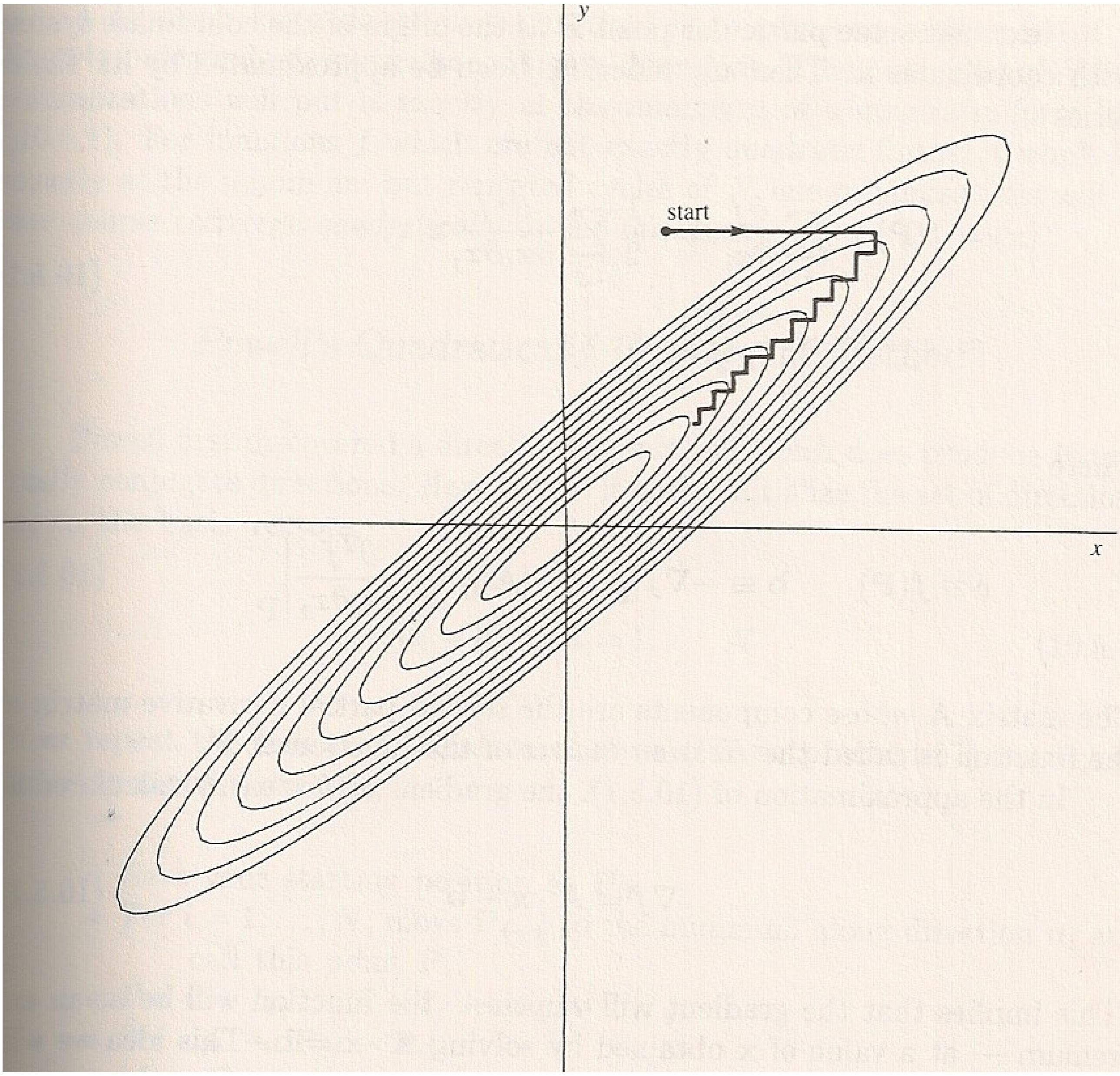
We know (§1O.1-§1O.3) how to minimize a function of one variable. If we start at a point P in N-dimensional space, and proceed from there in some vector direction n, then any function of N variables f(P) can be minimized along the line n by our one-dimensional methods. One can dream up various multidimensional minimization methods which consist of sequences of such line minimizations. Different methods will differ only by how, at each step, they choose the next direction n to try. All such methods presume the existence of a "black-box" sub-algorithm, which we might call LINMIN (given an explicit routine at the end of this section), whose definition can be taken for now as

> **LIMIN**: Given as input the vectors P and n, and
> the function f, find the scalar À that minimizes
> f(P + lambda*n). Replace P by P + lambda*n,
> Replace n by lambda*n. Done.

All the minimization methods in this section and in the two sections following fall under this general schema of successive line minimizations. In this section we consider a class of methods whose choice of successive directions does not involve explicit computation of the function's gradient; the next two sections do require such gradient calculations. You will note that we need not specify whether LINMIN uses gradient information or not. That choice is up to you, and its optimization depends on your particular function. You would be crazy, however, to use gradients in LINMIN and not use them in the choice of directions, since in this latter role they can drastically reduce the total computational burden.

But what if, in your application, calculation of the gradient is out of the question. You might first think of this simple method: Take the unit vectors el, e2, ... eN as a set of directions. Using LINMIN, move along the first direction to its minimum, then from there along the second direction to its minimum, and so on, cycling through the whole set of directions as many times as necessary, until the function stops decreasing.

This dumb method is actually not too bad for many functions. even more interesting is why it is bad, i.e. very inefficient, for some other functions. Consider a function of two dimensions whose contour map (level lines) happens to define a long, narrow valley at some angle to the coordinate basis vectors (see Figure 10.5.1). Then the only way "down the length of the valley" going along the basis vectors at each stage is by a series of many tiny steps. More generally, in N dimensions, if the function's second derivative are much larger in magnitude in some directions than in others, then many cycles through all N basis vectors will be required in order to get anywhere. This condition is not all that unusual; by Murphy's Law, you should count on it.



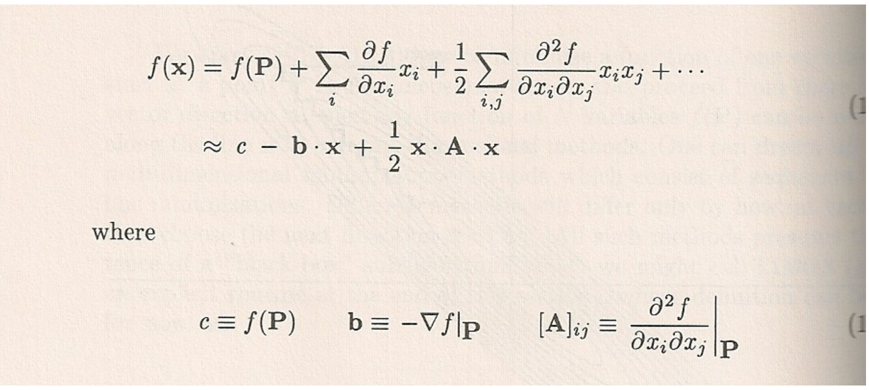
F**igure 10.5.1.** *Successive minimizations along coordinate directions in a long, narrow "valley" (shown as contour lines). Unless the valley is optimally oriented, this method is exinefficient, taking many tiny steps to get to the minimum, crossing and re-crossing the principal axis.*

Obviously what we need is a better set of directions than the ei's. All direction set methods consist of prescriptions for updating the set of directions as the method proceeds, attempting to come up with a set which either (i) includes some very good directions that will take us far along narrow valleys, or (more subtly) (ii) includes some number of "non-interfering" directions with the special property that minimization along one is not "spoiled" by subsequent minimization along another, so that interminable cycling through the set of directions can be avoided.

Conjugate Directions

This concept of "non-interfering" directions, more conventionally called conjugate directions, is worth making mathematically explicit.
First, note that if we minimize a function along sorne direction u, then the gradient of the function must be perpendicular to u at the line minimum; If not, then there would still be a nonzero directional derivative along u.

Next take some particular point P as the origin of the coordinate system with coordinates x. Then any function f can be approximated by its Taylor series



$$f(\mathbf{x}) = f(\mathbf{P}) + \sum_i \frac{\partial f}{\partial x_i} x_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j + \cdots$$
$$\approx c - \mathbf{b} \cdot \mathbf{x} + \frac{1}{2} \mathbf{x} \cdot \mathbf{A} \cdot \mathbf{x}$$

(1

where

$$c \equiv f(\mathbf{P}) \qquad \mathbf{b} \equiv -\nabla f|_{\mathbf{P}} \qquad [\mathbf{A}]_{ij} \equiv \left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_{\mathbf{P}}$$

(1

The matrix **A** whose components are the second partial derivative matrix of the function is called the Hessian matrix of the function at **P**.

In the approximation of (10.5.1), the gradient of f is easily calculated as

$\overline{V}$ f = A·x-b (10.5.3)

(This implies that the gradient will vanish - the function will be at an extremum - at a value of x obtained by solving A . x = b. This idea we will return to in §10.7!)

How does the gradient $\overline{V}$ f change as we move along some direction? Evidently

d($\overline{V}$ f) = A . (dx) (10.5.4)

Suppose that we have moved along some direction u to a minimum and now propose to move along some new direction v. The condition that motion along v not spoil our minimization along u is just that the gradient stay perpendicular to u, i.e. that the change in the gradient be perpendicular to u. By equation (10.5.4) this is just

0 = u d($\overline{V}$f) = u· A· v (10.5.5)

When (10.5.5) holds for two vectors u and v, they are said to be conjugate. When the relation holds pairwise for all members of a set of vectors, they are said to be a conjugate set. If you do successive line minimization of a function along a conjugate set of directions, then you don't need to redo any of those directions (unless, of course, you spoil things by minimizing along a direction that they are not conjugate to).

A triumph for a direction set method is to come up with a set of N linearly independent, mutually conjugate directions. Then, one pass of N line minimizations will put it exactly at the minimum of a quadratic form like (10.5.1). For functions f which are not exactly quadratic forms, it won't be exactly at the minimum; but repeated cycles of N line minimizations will in due course converge quadratically to the minimum.

Powell's Quadratically Convergent Method

Powell first discovered a direction set method which does produce N mutually conjugate directions. Here is how it goes: Initialize the set of directions u, to the basis vectors,

$$u_i = e_i \quad i= 1, \ldots ,N \qquad (10.5.6)$$

Now repeat the following sequence of steps ("basic procedure") until your function stops decreasing:

- Save your starting position as Po.
- For i = 1,.,. ,N, move Pi-1 to the minimum along direction u, and call this point Pi.
- For i = 1, ... ,N - 1, set u, t- Ui+l.
- Set UN t- P N - Po.
- Move PN to the minimum along direction UN and cali this point Po.

Powell, in 1964, showed that, for a quadratic form like (lO.5.1), k iterations of the above basic procedure produce a set of directions u, whose last k members are mutually conjugate. Therefore, N iterations of the basic procedure, amounting to N(N + 1) line minimizations in all, will exactly minimize a quadratic form. Brent (1973) gives proofs of these statements in accessible form.

Unfortunately, there is a problem with Powell's quadratically convergent algorithm. The procedure of throwing away, at each stage, **Ui** in favor of **Pn - Po** tends to produce sets of directions that "fold up on each other" and become linearly dependent. Once this happens, then the procedure finds the minimum of the function f only over a subspace of the full N-dimensional case; in other words, it gives the wrong answer. Therefore, the algorithm must not be used in the form given above.

There are a number of ways to fix up the problem of linear dependence in Powell's algorithm, among them:

1· You can reinitialize the set of directions **ui,** to the basis vectors ei, after every N or N + 1 iterations of the basic procedure. This produces a serviceable method, which we commend to you if quadratic convergence is important for your application (i.e. if your functions are close to quadratic forms and if you desire high accuracy).

2. Brent points out that the set of directions can equally well be reset to the columns of any orthogonal matrix. Rather than throw away the information on conjugate directions already built up, he resets the direction set to calculated principal directions of the matrix A (which he gives a procedure for determining). The calculation is essentially a singular value decomposition algorithm (see §2.9). Brent has a number of other cute tricks up his sleeve, and his modification of Powell's method is probably the best presently known. Consult his book for a detailed description and listing of the program. Unfortunately it is rather too elaborate for us to include here.

3. You can give up the property of quadratic convergence in favor of a more heuristic scheme (due to Powell) which tries to find a few good directions along narrow valleys instead of N necessarily conjugate directions. This is the method which we now implement. (It is also the version of Powell's method given in Acton, from which parts of the following discussion are drawn).

Powell's Method Discarding the Direction of Largest Decrease

The fox and the grapes: Now that we are going to give up the property of quadratic convergence, was it so important after ail? That depends on the function that you are minimizing. Some applications produce functions with long, twisty valleys. Quadratic convergence is of no particular advantage to a program which must slalom down the length of a valley floor that twists one way and another (and another, and another, ... - there are N dimensions!). Along the long direction, a quadratically convergent method is trying to extrapolate to the minimum of a parabola which just isn't (yet) there; while the conjugacy of the N - 1 transverse directions keeps getting spoiled by the twists.

Sooner or later, however, we do arrive at an approximately ellipsoidal minimum (cf. equation 10.5.1 when b, the gradient, is zero). Then, depending on how much accuracy we require, a method with quadratic convergence can save us several times N2 extra line minimizations, since quadratic convergence doubles the number of significant figures at each iteration.

The basic idea of our now-modified Powell's method is still to take **Pn - P0** as a new direction; it is, after all, the average direction moved after trying all N possible directions. For a valley whose long direction is twisting slowly, this direction is likely to give us a good run along the new long direction. The change is to discard the old direction along which the function f made its largest decrease. This seems paradoxical, since that direction was the best of the previous iteration. However, it is also likely to be a major component of the new direction that we are adding, so dropping it gives us the best chance of avoiding a buildup of linear dependence.

There are a couple of exceptions to this basic idea. Sometimes it is better not to add a new direction at aIl. Define

f0 ~ f(Po)   fN ~ f(PN)   fE ~ f(2PN - Po)   (10.5.7)

Here fE is the function value at an "extrapolated" point somewhat further along the proposed new direction. Also define deltaf to be the magnitude of the largest decrease along one particular direction of the present basic procedure iteration. (deltaf is a positive number.) Then:

1. If fE >= f0, then keep the old set of directions for the next basic procedure, because the average direction P N - Po is all played out.

2. If 2 (f0-2fN+fE) [(fo- fN)-deltafj^2 >= (fo- fE)^2/deltaf, then keep the old set of directions for the next basic procedure, because either (i) the decrease along the average direction was not primarily due to any single direction's decrease, or (ii) there is a substantial second derivative along the average direction and we seem to be near to the bottom of its minimum.

The following routine (powell) implements Powell's method in the version just described. ln the routine, XI is the matrix whose columns are the set of directions Ni; otherwise the correspondence of notation should be self-evident.

[From BIBLI08].