Computing Gradients and Hessian from Objective Values

(Nonlinear Optimization without Gradients)

John Kormylo

Our goal is to minimize objective function $J(\mathbf{x})$ with respect to parameter vector $\mathbf{x}$ for the case where computing gradients costs as much as computing n objective functions, where n is the number of parameters in $\mathbf{x}$. We assume that $J(\mathbf{x})$ is relatively well behaved over the region of interest (continuous to the third derivative). Consequently we would like to use Newton's method [1] or a variant thereof, which requires a gradient and Hessian.

The approach here will be to estimate the gradient and Hessian at some location $\mathbf{x}_0$ using a least squares fit over a list of previously generated values. The idea is that by using far more points than needed to get a solution, an occasional oddball objective function value will not entirely unhinge the algorithm, nor need we overly concern ourselves whether the given vectors form a basis.

Assume a model of the form

$$J(\mathbf{x}) \approx J(\mathbf{x}_0) + \mathbf{g}'(\mathbf{x} - \mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)' H (\mathbf{x} - \mathbf{x}_0) \tag{1}$$

where $\mathbf{g}$ and H are the gradient and Hessian at $\mathbf{x}_0$, the current best set of parameters. We want estimates of $\mathbf{g}$ and H so as to minimize error function

$$\sum_{i=1}^{N} \left[J(\mathbf{x}_i) - J(\mathbf{x}_0) - \mathbf{g}'(\mathbf{x}_i - \mathbf{x}_0) - (\mathbf{x}_i - \mathbf{x}_0)' H (\mathbf{x}_i - \mathbf{x}_0) \right]^2$$

where the $\mathbf{x}_i$ for $i = 1, 2, \ldots, N$ are a list of previously used parameters. This can be rewritten as

$$\sum_{i=1}^{N} \left[\Delta J(i) - \mathbf{g}'\Delta\mathbf{x}(i) - \Delta\mathbf{x}'(i) H \Delta\mathbf{x}(i) \right]^2 \tag{2}$$

using the notation $\Delta J(i) = J(\mathbf{x}_i) - J(\mathbf{x}_0)$ and $\Delta\mathbf{x}(i) = \mathbf{x}_i - \mathbf{x}_0$.

Optimizing with respect to $\mathbf{g}$ we get

$$2 \sum_{i=1}^{N} \left[\Delta J(i) - \mathbf{g}'\Delta\mathbf{x}(i) - \Delta\mathbf{x}'(i) H \Delta\mathbf{x}(i) \right] \Delta\mathbf{x}(i) = 0$$

and therefore

$$\sum_{i=1}^{N} \left[\mathbf{g}\Delta\mathbf{x}(i) + \Delta\mathbf{x}'(i) H \Delta\mathbf{x}(i) \right] \Delta\mathbf{x}(i) = \sum_{i=1}^{N} \Delta J(i) \Delta\mathbf{x}(i) \quad . \tag{3}$$

Optimizing with respect to H we get

$$2 \sum_{i=1}^{N} \left[\Delta J(i) - \mathbf{g}'\Delta\mathbf{x}(i) - \Delta\mathbf{x}'(i) H \Delta\mathbf{x}(i) \right] \Delta\mathbf{x}(i)\Delta\mathbf{x}'(i) = 0$$

and therefore

$$\sum_{i=1}^{N} \left[\mathbf{g}'\Delta\mathbf{x}(i) + \Delta\mathbf{x}'(i) H \Delta\mathbf{x}(i) \right] \Delta\mathbf{x}(i)\Delta\mathbf{x}'(i) = \sum_{i=1}^{N} \Delta J(i) \Delta\mathbf{x}(i)\Delta\mathbf{x}'(i) \quad . \tag{4}$$

There is no simple matrix solution for H, but (4) does represent n^2 scalar equations to match the n^2 scalar unknowns in H. One simply has to map the elements of $\mathbf{g}$ and H into a single vector.

For $n = 2$ we can define this vector as

$$\mathbf{v}' = \left[\begin{array}{ccccc} g_1 & g_2 & h_{1,1} & h_{1,2} + h_{2,1} & h_{2,2} \end{array} \right] \tag{5}$$

taking advantage of symmetry. If we then define vector $\mathbf{u}(i)$ as

$$\mathbf{u}'(i) = \left[\begin{array}{ccccc} \Delta x_1(i) & \Delta x_2(i) & \Delta x_1^2(i) & \Delta x_1(i)\Delta x_2(i) & \Delta x_2^2(i) \end{array} \right] \tag{6}$$

one can show that

$$\mathbf{u}'(i)\mathbf{v} = \mathbf{g}'\Delta\mathbf{x}(i) + \Delta\mathbf{x}'(i)H\Delta\mathbf{x}(i) \tag{7}$$

for $i = 1, 2, \ldots, N$. Extending this for $n > 2$ is fairly straight forward.

We can now combine equations (3) and (4) as

$$\left(\sum_{i=1}^{N} \mathbf{u}(i)\mathbf{u}'(i) \right) \mathbf{v} = \sum_{i=1}^{N} \Delta J(i)\mathbf{u}(i) \tag{8}$$

and solve for $\mathbf{v}$. Note that since H is symmetrical, from (5) we have

$$h_{1,2} = h_{2,1} = v_4/2 \quad .$$

The minimum number of points $\mathbf{x}_i$ (including $\mathbf{x}_0$) needed for a solution is $n + n(n+1)/2 + 1$ or more simply $(n+1)(n+2)/2$. Generally one would like to have 3 points in every direction, rather than just the single best point from each line search. It should be noted that using Newton's method with these estimates when $n = 1$ is equivalent to performing a line search by quadratic curve fitting.

References

[1] David G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley Publishing, 1973.