Optimization with the Quasi-Newton Method

by
Ronald Schoenberg

September 5, 2001

Aptech Systems, Inc.
Maple Valley, WA

Contents

Chapter 1

Optimization with the Quasi-Newton Method

One of the first problems to which Sir Isaac Newton applied calculus was the optimization of a function. He observed that the extremum of a function is characterized by its derivatives being equal to zero. For example, for the ordinary least squares problem

$$f(B) = y^t y - 2Bx^t y + B^t x^t xB$$

is a multivariate quadratic function of a vector of coefficients. The extremum, i.e., the value of B for which $f(B)$ is either maximum or minimum, is found by setting the derivative of $f(B)$

$$f'(B) = -2x'y + x'xB$$

with respect to B to zero and solving for B

$$B_m = (x^t x)^{-1} x^t y$$

Finding such an extremum for nonquadratic functions is not so easy. In general a simple closed form solution is not available as it is in the least squares problem. For this kind of problem Newton proposed an iterative solution: first we look at a local quadratic approximation to the nonlinear function and find its extremum, and then generate a new local approximation and so on. For the local approximation we use a Taylor series approximation about some given point x˙k on the function's surface,

$$f(x) = f(x_m) + f'(x_m)(x - x_m) + \frac{1}{2}(x - x_m)f''(x_m)(x - x_m)$$

In the same manner as above we calculate the derivatives, set to zero, and solve for x:

$$\begin{aligned} f'(x) &= f'(x_m) + f''(x_m)(x - x_m) = 0 \\ x &= x_m - [f''(x_m)]^{-1} f'(x_m) \end{aligned}$$

If the function is quadratic, we arrive at the extremum in a singel step, i.e., x is the solution. If the function is not quadratic, we must solve for the solution iteratively, that is we set x_m equal to x and compute a new x

$$x_{m+1} = x_m - \delta_m$$

where

$$\delta_m = [f''(x_m)]^{-1} f'(x_m) = H_m^{-1} g_m$$

is called the direction. The direction is a vector describing a segment of a path from the starting point to the solution were the inverse of the Hessian, H_m determines the "angle" of the direction and the gradient, g_m determines its "size".

When the approximation is good, the Hessian is well-conditioned and the convergence quadratic (Dennis and Schnabel, 1989, Theorem 2.1.1) which roughly speaking means that the number of places of accuracy doubles at each step (Gill, Murray, and Wright, 1981). Quite often, however, the function being optimized is not particularly well behaved in the region of x˙m. This point might be far from the optimum and the surface in that region might be poorly approximated by the quadratic function. An additional step is introduced in the iterations to deal with this. The Newton step is re-defined as

$$x_{m+1} = x_m - \alpha_m \delta_m$$

where α_m is called the step length. The step length is determined by a local optimization of the function, called a line search, that is given the direction and the starting point

$$f(x_m - \alpha_m \delta_m)$$

is a scalar function of the step length. Depending on the type of line search, this function will be either minimized or some value of α_m is found such that $f(x_m - \alpha_m \delta_m) < f(x_m)$.

The Newton method is simple and straightforward to describe, but there are a number of issues that arise in actual application. The first issue devolves from the fact that a function for computing an analytical Hessian is almost hever available.

1.0.1 The Quasi-Newton Method

Since a function for computing the Hessian used in computing the direction is rarely available, attention has focused on computing it numerically. The calculation of the Hessian is very expensive computationally, however, and efforts were made to find a way to produce the Hessian more cheaply. The critical insight from which came the current quasi-Newton methods was made by Broyden (1969): use informatoin from the current iteration to compute the new Hessian. Let

$$s_k = x_{m+1} - x_m = \alpha_m \delta_m$$

be the chane in the parameters in the current iteration, and

$$\eta_m = g_{m+1} - g_m$$

be the change in the gradients. Then a natural estimate of the Hessian at the next iteration H_{m+1} would be the solution of the system of linear equations

$$H_{m+1} s_m = eta_m$$

that is, H_{m+1} is the ratio of the change in the gradient to the change in the parameters. This is called the quasi-Newton condition. There are many solutions to this set of equations. Broyden suggested a solution in the form of a secand update

$$H_{m+1} = H_m + uv^t$$

Further work has developed other types of secant updates, the most important of which are the DFP (for Davidon, 1959, and Fletcher and Powell, 1963), and the BFGS (for Broyden, 1969, Fletcher, 1970, Goldfarb, 1970, and Shanno, 1970). The BFGS is generally regarded as the best performing method:

$$\begin{aligned} H_{m+1} &= H_m + \frac{\eta_m \eta_m^t}{\eta_m^t s_m} - \frac{H_m s_m s_m^t H_m}{s_m^t H_m s_m} \\ &= H_m + \frac{\eta_m \eta_m^t}{\eta_m^t s_m} - \frac{g_m g_m^t}{\delta_m^t g_m} \end{aligned}$$

taking advantage of the fact that $H_m s_m = \alpha_m H_m \delta_m = \alpha_m g_m$. The BFGS method is used in the **GAUSS** function **QNEWTON**. However, the update is made to the Cholesky factorization of H rather than to the Hessian itself, that is to R where $H = R^t R$. In **QNEWTON** H itself is not computed anywhere in the iterations. The direction δ_m is computed using $CHOLSOL$, that is as a solution to

$$R_m^t R_m \delta_m = g_m$$

where R_m and g_m are its arguments, and R_m is the Cholesky factorization of H_m. Then R_{m+1} is computed as an update and a downdate to R_m using the **GAUSS** functions **CHOLUP** and **CHOLDN**.

CHOLUP and CHOLDN

Suppose one was using a Cholesky factorization of a moment matrix computed on a data set in some analytical method. Then suppose you have acquired an additional observation. The Cholesky factorization could be re-computed from the original data set augmented by the additional row. Such a complex and time consuming computation is not necessary. Numerical methods have been worked out to compute the new updated factorizatoin with the new observation without having to re-compute the moment matrix and factorization from scratch. This update is handled by the **GAUSS CHOLUP** function. In a similar way a Cholesky factorization can be downdated, i.e., a new factorization is computed from a data set with one observation removed. In either of these functions there are two arguments, the original factorization and the observation to be added or removed.

Now back to the BFGS secant update in the quasi-Newton method. The factorization R_{m+1} is generated by first adding the "observation",

$$\eta_m / \sqrt{\eta_m^t s_m}$$

and then removing the "observation"

$$g_m / \sqrt{\delta_m^t g_m}$$

Line Search

Line search is the method by which some value for α_m is found such that $f(x_m - \alpha_m \delta_m) < f(x_m)$. Some common methods are step halving, golden section search, random, and polynomial fit. There are known example where optimizations will fail by setting $\alpha_m = 1$ for all iterations (Dennis and Schnabel, 1983, page 163) and thus the line search is an essential part of the quasi-Newton optimization. In step halving, α_m is set first to 1, and $f(x_m - \alpha_m \delta_m)$ is tested for a decrease. If the test fails, α_m is divided by 2 and the test is tried again. This continues until a decrease in the function occurs and the final value of α_m is then the required step length. For quick and dirty optimization, this method can work quite well. In general, however, a method that finds some kind of optimum value will help the optimization best.

In the golden section method (Gill, Murray, and Wright, 1981), an interval is defined for $\alpha_m \in [0, 1]$, then trial values are selected at t and $(1 - t)$ where $t = .618$ (t is the solution of the quadratic function $t^2 + t - 1$. A new interval is then defined at $[0, 1 - t]$ if $f(x_m - (1 - t)\delta_m)$ is less than $f(x_m - t\delta_m)$ and $[t, 1]$ otherwise. This is repeated in the next interval until some criterion is met such as $f(x_m - \alpha_m \delta_m) < f(x_m)$ or $f(x_m - t\delta_m)$ and $f(x_m - (1 - t)\delta_m)$ are sufficiently close together.

The primary line search in **QNEWTON** is one developed by Dennis and Schnabel, 1981, called STEPBT. It is a polynomial search method. First, $\alpha_m = 1$ is tried, and if the reduction in the function is sufficient, that is, if

$$[f(x_m - \delta) - f(x_m)] / \delta_m^t g_m < 1e - 4$$

then the step length is set to one and the iterations continued. If not a quadratic function is fit to $f(x_m - \alpha_m \delta_m)$ using $f(x_m)$, g_m, $f(x_m - \delta_m)$, i.e., using the falues of $f()$ where $\alpha_m = 0$, $\alpha_m = 1$, and the gradient evaluated where $\alpha_m = 0$. If the value of α_m at the minimum of that quadratic passes a criterion, that value is selected for the step length, but if not, a cubic is fit to the above information plus an additional evaluation of the function with α_m set to a specified value. If that fails, new evaluations of the function with trial values of the step length are set to the value found in the cubic fitting. This continues either until a step length is found that satisfies the criteria, or until 40 attempts have been made. If 40 attempts have been made without finding a satisfactory step length, QNEWTON tries a random search. In the random line search, random trial values of the step length from within a set radius are tried. If any reduce the function, it is selected as the step length.

Convergence

From the formula for the direction δ_m, we see that progress in the iterations becomes indescernable when the gradient approaches zero. This could be due either to the iterations having converged or to poor scaling. The convergence criterion in **QNEWTON** thus is a relative gradient, a gradient adjusted for scaling:

$$max|f(x_{m+1}) \cdot g_{i,m+1}/x_{i,m+1}| < gradtol$$

Condition

Computer arithmetic is fundamentally flawed by the fact that the computer number is finite (see Higham, 1996, for a general discussion). The standard double precision number in PC's carries about 16 decimal significant places. A simple operation can destroy nearly all of those places. For example, consider the Box-Cox transformation $(x^\lambda - 1)/\lambda$. With infinite precision this calculation approaches $ln(x)$ as λ approaches zero. But observe what happens with finite precision for $x = 2$:

λ	2^λ	$(2^\lambda - 1)/\lambda$
1e-05	1.0000069314958282e+00	6.9314958281996286e-01
1e-06	1.0000006931474208e+00	6.9314742079384928e-01
1e-07	1.0000000693147204e+00	6.9314720407831487e-01
1e-08	1.0000000069314718e+00	6.9314718409430043e-01
1e-09	1.0000000006931471e+00	6.9314709527645846e-01
1e-10	1.0000000000693148e+00	6.9314776141027323e-01
1e-11	1.0000000000069316e+00	6.9315664319447023e-01
1e-12	1.0000000000006932e+00	6.9322325657594774e-01
1e-13	1.0000000000000693e+00	6.9277916736609768e-01
1e-14	1.0000000000000069e+00	6.8833827526759706e-01
1e-15	1.0000000000000007e+00	6.6613381477509392e-01

Until λ reaches about $1e-11$ the function is converging nicely to $ln(2) = 0.693147180559945$. After that point, however, the convergence clearly falls apart. Observe that as λ gets smaller the informative decimal places in 2^{λ} slip away until by $\lambda = 1e-16$ there's only one informative decimal place left and its rounded.

This problem is due to the finiteness of the computer number, not to the implementation of the operators. The calcualtion of 2^{λ} is equivalent to adding a progressively smaller number to 1. Suppose we have a very small number known to 15 places of precision, $.999999999999999e-15$. Adding 1 to that number we get 1.000000000000001, in other words, 14 of the 15 places in the smaller number are completely lost. It is an inherent problem in all computers and the only solution, adding more bits to the computer number, is only temporary because sooner or later a problem will arise where that quantity of bits won't be enough. The first lesson to be learned from this is to avoid operations combining very small numbers with relatively large numbers. And for very small numbers, 1 can be a large number.

The key quantity in the Newton methods, including the quasi-Newton, are the derivatives. The calculation of the direction involves an inversion and a matrix multiplication. A direct inversion is avoided by using a solve algorithm, but that only alleviates the problem, it doesnt do away with it. The standard method for evaluating the precision lost in computing a matrix inverse is the ratio of the largest to the smallest eigenvalue of the matrix. This quantity is sometimes called the condition number. The log of the condition number to the base 10 is approximately the number of decimal places lost in computing the inverse. A condition number greater than $1e16$ therefore indicates that all of the 16 decimal places are lost that are available in the standard double precision floating point number.

The quasi-Newton optimization method has been successful primarily because its method of generating an approximation to the Hessian encourages better conditioning. Nearly all implementations of the Newton method involve a numerical calculation of the Hessian. A numerical Hessian, like all numerical derivatives, are computed by dividing a difference by a very small quantity, a very unfavorable computational method on a computer. In general, when using double precision with 16 places of accuracy, about four places are lost in calculating a first derivative and another four with the second derivative. The numerical hessian therefore begins with a loss of eight places of precision. If there are any problems computing the function itself, or if the model itself contains any problems of condition, there may be nothing left at all.

The BFGS method implemented in **QNEWTON** avoids much of the problems in computing a numerical Hessian. It produces an approximation by building information slowly with each iteration. Initially the Hessian is set tot he identity matrix, the matrix with the best conditoin but the least information. Information is increased at each iteration with a method that guarantees a positive definite result. This provides for stabler, though slower, progress towards convergence.

The implemenation of **QNEWTON** has been designed to minimize the damage to the precision of the optimization problem. The BFGS method avoids a direct calculation of

the numerical Hessian, and uses sophisticated techmiques for calculating the direction that preserve as much precision as possible. However, all of this can be defeated by a poorly scaled problem or a poorly specified model. When the objective function being optimized is a log-likelihood, the inverse of the Hessian is an estimate of the covariance matrix of the sampling distribution of the parameters. The condition of the Hessian is related to (i) the scaling of the parameters, and (ii) the degree with which there are linear dependencies in the sampling distribution of the parameters.

Scaling

Scaling is under the direct control of the investigator and should never be an issue in the optimization. It might not always be obvious how to do it, though. In estimation problems scaling of the parameters is usually implemented by scaling the data. in regression models this is simple to accomplish, but in more complicated models it might be more difficult to do. It might be necessary to experiment with different scaling to get it right. The goal is to optimize the condition of the hessian. The definition of the condition number implies that we endeavor to minimize the difference of the largest to the smallest eigenvalue of the Hessian. A rule of thumb for this is to scale the Hessian so that the diagonal elements are all about the same magnitude.

If the scaling of the Hessian proves too difficult, an alternative method is to scale the parameters directly in the procedure computing the log-likelihood. Multiply or divide the parameter values being passed to the procedure by setting quantities before their use in the calculation of the log-likelihood. Experiment with different values until the diagonal elements of the Hessian are all about the same magnitude.

Linear dependencies or nearly linear dependencies in the sampling distribution.

This is the most common difficulty in estimation and arises because of a discrepancy between the data and the model. If the data do not contain sufficient information to "identify" a parameter or set of parameters, a linear dependency is generated. A simple example occurs in regressors that cannot be distinquished from the constant because its variation is too small. When this happens, the sampling distribution of these two parameters becomes highly collinear. This collinearity will produce an eigenvalue approaching zero in the Hessian, increasing the number of places lost in the calculation of the inverse of the Hessian, degrading the optimization.

In the real world the data we have available will frequently fail to contain the information we need to estimate all of the parameters of our models. This means that it is a constant struggle to a well-conditioned estimation. When the condition sufficiently deteriorates to the point that the optimization fails, or the statistical inference fails through a failure to invert the Hessian, either more data must be found, or the model must be re-specified. Re-specification means either the direct reduction of the parameter space, that is, a parameter is deleted from the mdoel, or some sort of restriction is applied to the parameters.

Diagnosing the linear dependency.

At times it may be very difficult to determine the cause of the ill-conditioning. If the Hessian being computed at convergence for the covariance matrix of the parameters fails to invert or is very ill-conditioned, try the following: first generate the pivoted QR factorization of the Hessian,

```
{ R,E } = qre(H);
```

The linearly dependent columns of H are pivoted to the end of the R matrix. E contains the new order of the columns of H after pivoting. The number of linearly dependent columns is found by looking at the number of nearly zero elements at the end of the diagonal fo R.

We can compute a coefficient matrix of the linear relationship of the dependent columns on the remaining columns by computing $R_{11}^{-1}R_{12}$ where R_{11} is that portion of the R matrix associated with the independent columns and R_{12} the independent with dependent. Rather than use the inverse function in **GAUSS**, we use a special solve function that takes advantage of the triangular shape of R_{11}. Suppose that the last two elements of R are nearly zero, then

```
r0 = rows(R);
r1 = rows(R) - 1;
r2 = rows(R) - 2;
B = utrisol(R[1:r2,r1:r0],R[1:r2,1:r2);
```

B describes the linear dependencies among the columns of H and can be used to diagnose the ill-conditioning in the Hessian.

1.1 References

Broyden, C.G., 1969. "A new double-rank minimization algorithm", *Notices of the American Mathematical Society*, 16:670.

Davidon, W.C., 1970. "Variable metric methods for minimization", *Argonne National Labs Report ANDL-5990.*

Dennis, J.E., Jr., Schnabel, Robert B., 1983, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, New York.

Dennis, J.E., Jr., Schnabel, Robert B., 1989, "A view of unconstrained optimization", in *Handbook in Operations Research and Management Science, Volume 1, Optimization*, G.L. Nemhauser, A.H.G. Runnooy Kan and M.J. Todd (eds.). Elsevier, Amsterdam.

Fletcher, R., Powell, M.J.D., 1963. "A rapidly convergent descent method for minimization", *Computer Journal*, 7:149-154.

Fletcher, R., 1970. "A new approach to variable metric methods", *Computer Journal*, 13:317-322.

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

Goldfarb, D., 1970. "A family of variable metric methods derived by variational means", *Mathematics of Computation*, 24:23-26.

Higham, Nicholas, J., 1996. *Accuracy and Stablility of Numerical Algorithms.* SIAM, Philadelphia.

Luenberger, David G., 1984. *Linear and Nonlinear Programming, 2nd Edition.* Addison-Wesley, Reading MA.

Shanno, D.F., 1970. "Conditioning of quasi-Newton methods for function minimization", *Mathematics of Computation*, 24:145-160.