

Lecture XI

Quasi-Newton Methods of Optimization

I. A Baseline Scenario

A. In this lecture, we develop several alternatives to the Newton-Raphson algorithm. As starting point, I want to discuss a prototype algorithm. Algorithm U (Model algorithm for n-dimensional unconstrained minimization). Let x_k be the current estimate of x^* (the point which minimizes the objective function $f(x)$).

- U1. [Test for convergence] If the conditions for convergence are satisfied, the algorithm terminates with x_k as the solution.
- U2. [Compute a search direction] Compute a non-zero n -vector p_k , the direction of the search.
- U3. [Compute a step length] Compute a scalar a_k , the step length, for which $f(x_k + a_k p_k) < f(x_k)$.
- U4. [Update the estimate of the minimum] Set $x_{k+1} = x_k + a_k p_k$, $k=k+1$, and go back to step U1.

B. Given the steps to the prototype algorithm, I want to develop a sample problem that we can compare the various algorithms against. Notebook Algorithm3.ma includes the numeric problem:

$$\begin{aligned} \max_x & x_1^2 x_2^3 x_3^4 x_4^1 \\ \text{st } & x_1 = 100 - 2x_2 - 3x_3 - x_4 \end{aligned}$$

Using Newton-Raphson, the optimal point for this problem is found in 10 iterations using 1.23 seconds on the DEC Alpha.

II. An Overview of Newton and Quasi-Newton Algorithms

A. The Newton-Raphson methodology can be used in U2 in the prototype algorithm. Specifically, the search direction can be determined by:

$$p_k = -(\nabla_{xx}^2 f(x_k))^{-1} \nabla_x f(x_k)$$

B. Quasi-Newton algorithms involve an approximation to the Hessian matrix. For example, we could replace the Hessian matrix with the negative of the identity matrix for the maximization problem. In this case the search direction would be:

$$p_k = -(-I(n)) \nabla_x f(x_k)$$

where $I(n)$ is the identity matrix conformable with the gradient vector. This replacement is referred to as the steepest descent method. In our sample problem, this methodology requires 990 iterations and 29.28 seconds on the DEC Alpha. This result highlights some interesting features regarding the Quasi-Newton methods:

1. The steepest descent method requires more overall iterations. In this example, the steepest descent method requires 99 times as many iterations as the Newton-Raphson method.
2. Typically, the time spent on each iteration is reduced. Again, in the current comparison each the steepest descent method requires .123

seconds per iteration while Newton-Raphson requires .030 seconds per iteration.

- C. Obviously substituting the identity matrix uses no real information from the Hessian matrix. An alternative to this drastic reduction would be to systematically derive a matrix H_k which uses curvature information akin to the Hessian matrix. The projection could then be derived as:

$$p_k = -H_k^{-1} \nabla_x f(x_k).$$

III. Conjugate Gradient Methods

- A. One class of Quasi-Newton methods are the conjugate gradient methods which “build” up information on the Hessian matrix.

1. From our standard starting point, we take a Taylor series expansion around the point $x_k + s_k$

$$\nabla_x f(x_k + s_k) = \nabla_x f(x_k) + \nabla_{xx}^2 f(x_k) s_k$$

for some $s_k \in N(x_k, \delta)$. Solving this expression for the term involving the Hessian yields

$$\nabla_x f(x_k + s_k) - \nabla_x f(x_k) = \nabla_{xx}^2 f(x_k) s_k$$

$$s_k' (\nabla_x f(x_k + s_k) - \nabla_x f(x_k)) = s_k' \nabla_{xx}^2 f(x_k) s_k$$

Thus, the change in the gradient contains the same curvature information as the original Hessian.

2. Thus,

$$y_k = \nabla_x f(x_k + s_k) - \nabla_x f(x_k)$$

which amounts to

$$y_k = \nabla_{xx}^2 f(x_k) s_k$$

So we want to approximate the information in the Hessian using the some matrix B_{k+1}

$$y_k = B_{k+1} s_k$$

3. One way to generate B_{k+1} is to start with the current B_k and add new information on the current solution:

$$B_{k+1} = B_k + u v'$$

$$y_k = (B_k + u v') s_k$$

where u and v are $n \times 1$ vectors. Carrying through the vector multiplication

$$u(v' s_k) = y_k - B_k s_k$$

where $v' s_k$ is a scalar. Thus,

$$u = \frac{1}{v' s_k} (y_k - B_k s_k)$$

Substituting for u into the expression for B_{k+1} yields

$$B_{k+1} = B_k + \frac{1}{v' s_k} (y_k - B_k s_k) v'$$

Given this derivation, a logical choice for v would be $y_k + B_k s_k$. This approximation yields an updated Hessian of

$$B_{k+1} = B_k + \frac{1}{(y_k - B_k s_k)' s_k} (y_k - B_k s_k)(y_k - B_k s_k)'$$

This update results in a symmetric approximation to the Hessian. In general, any approximation to the Hessian can be made symmetric by

$$B^{(2)} = \frac{1}{2} (B^{(1)} + B^{(1)'})$$

Substituting such an expression into the general updating procedure results in

$$B_{k+1} = B_k + \frac{1}{v' s_k} \left((y_k - B_k s_k) v' + v (y_k - B_k s_k)' \right) - \frac{(y_k - B_k s_k)' s_k}{(v' s_k)^2} v v'$$

The advantage of this formulation is that the approximation of the Hessian matrix will be symmetric for any vector v .

B. Two prominent conjugate gradient methods are the Davidon-Fletcher-Powell (DFP) update and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update.

1. In the DFP update v is set equal to y_k yielding

$$B_{k+1} = B_k - \frac{1}{s_k' B_k s_k} B_k s_k s_k' B_k + \frac{1}{y_k' s_k} y_k y_k' + (s_k' B_k s_k) w_k w_k'$$

$$w_k = \frac{1}{y_k' s_k} y_k - \frac{1}{s_k' B_k s_k} B_k s_k$$

2. The BFGS update is then

$$B_{k+1} = B_k - \frac{1}{s_k' B_k s_k} B_k s_k s_k' B_k + \frac{1}{y_k' s_k} y_k y_k'$$

IV. A Numerical Example

A. Using the previously specified problem and starting with an identity matrix as the original Hessian matrix, each algorithm was used to maximize the utility function. Each of the procedures will have the same first iteration because they have the same original Hessian matrix and analytical gradient. The initial gradient for all the Quasi-Newton procedures was

$$B_t = \begin{pmatrix} -1 & 0 & 0 \\ & -1 & 0 \\ & & -1 \end{pmatrix}$$

compared with an analytical gradient of

$$B_t^* = \begin{pmatrix} -5.275 & .2885 & .0718 \\ & -.6085 & .0954 \\ & & -.2244 \end{pmatrix}$$

This substitution implies a step of

$$s_t = (.7337 \quad .9766 \quad .2428)$$

Compared with an optimal step of

$$s_t^* = (4.3559 \quad 4.3476 \quad 4.3226)$$

In discussing the difference in step, I will focus on two attributes. The first attribute is the relative length of the step (the 2-norm). If the conjugate gradient routines can be made identical with Newton-Raphson with the appropriate step length algorithm, we could then focus on designing step lengths. The second attribute is the direction of the step. As we discussed in lecture 9, one of gains to Newton-Raphson is the refinement of the search direction through the information in the Hessian. The value of this refinement can be seen in the relative gain of the Newton-Raphson over the steepest descent method. Dividing each vector by its 2-norm yields

	x_1	x_2	x_3		x_1	x_2	x_3
Newton-Raphson	4.3559	4.3476	4.3226	7.5207	0.579191	0.578087	0.574763
Conjugate Gradient	0.7337	0.9766	0.2428	1.2454	0.589129	0.784167	0.194958

1. The Rank One Approximation

a. Iteration 1

$$B_t = \begin{pmatrix} -0.6690 & 0.3842 & 0.1536 \\ & -0.5540 & 0.1783 \\ & & -0.9287 \end{pmatrix}$$

$$B_t^* = \begin{pmatrix} -0.2780 & 0.1276 & 0.0511 \\ & -0.2501 & 0.0592 \\ & & -0.2280 \end{pmatrix}$$

	x_1	x_2	x_3		x_1	x_2	x_3
Newton-Raphson	6.7466	7.6270	4.8254	11.2682	0.5987	0.6769	0.4282
Conjugate Gradient	4.3187	5.0130	2.0042	6.9136	0.6247	0.7251	0.2899

b. Iteration 2

$$B_t = \begin{pmatrix} -0.6315 & 0.4256 & 0.1776 \\ & -0.5083 & 0.2048 \\ & & -0.9134 \end{pmatrix}$$

$$B_t^* = \begin{pmatrix} -0.0628 & 0.0171 & 0.0106 \\ & -0.0612 & 0.0113 \\ & & -0.0846 \end{pmatrix}$$

	x_1	x_2	x_3		x_1	x_2	x_3
Newton-Raphson	10.8541	11.6909	6.0789	17.0717	0.6358	0.6848	0.3561
Conjugate Gradient	7.7298	8.9067	3.7912	12.3876	0.6240	0.7190	0.3060

2. PSB

a. Iteration 1

$$B_t = \begin{pmatrix} -.6703 & .3860 & .1504 \\ & -.5565 & .1827 \\ & & -.9365 \end{pmatrix}$$

$$B_t^* = \begin{pmatrix} -.2780 & .1276 & .0511 \\ & -.2501 & .0592 \\ & & -.2280 \end{pmatrix}$$

	x_1	x_2	x_3		x_1	x_2	x_3
Newton-Raphson	6.7466	7.6270	4.8255	11.2682	0.5987	0.6769	0.4282
Conjugate Gradient	4.3118	5.0129	1.9948	6.9065	0.6243	0.7258	0.2888

b. Iteration 2

$$B_t = \begin{pmatrix} -.6328 & .4274 & .1745 \\ & -.5109 & .2096 \\ & & -.9223 \end{pmatrix}$$

$$B_t^* = \begin{pmatrix} -.0629 & .0171 & .0106 \\ & -.0612 & .0114 \\ & & -.0850 \end{pmatrix}$$

	x_1	x_2	x_3		x_1	x_2	x_3
Newton-Raphson	10.8525	11.7000	6.0686	17.0732	0.6356	0.6853	0.3554
Conjugate Gradient	7.7228	8.9130	3.7768	12.3834	0.6236	0.7198	0.3050

3. DFP

a. Iteration 1

$$B_t = \begin{pmatrix} -.7187 & .4517 & .0326 \\ & -.6455 & .3424 \\ & & -1.2232 \end{pmatrix}$$

$$B_t^* = \begin{pmatrix} -.2780 & .1276 & .0511 \\ & -.2501 & .0592 \\ & & -.2280 \end{pmatrix}$$

	x_1	x_2	x_3		x_1	x_2	x_3
Newton-Raphson	6.7466	7.6270	4.8255	11.2682	0.5987	0.6769	0.4282
Conjugate Gradient	4.1395	5.0091	1.7613	6.7327	0.6148	0.7440	0.2616

b. Iteration 2

$$B_t = \begin{pmatrix} -.6788 & .4945 & .0589 \\ & -.6021 & .3766 \\ & & -1.2194 \end{pmatrix}$$

$$B_t^* = \begin{pmatrix} -.0653 & .0177 & .0119 \\ & -.0602 & .0124 \\ & & -.0971 \end{pmatrix}$$

	x_1	x_2	x_3		x_1	x_2	x_3
Newton-Raphson	10.8069	11.9291	5.8015	17.1099	0.6316	0.6972	0.3391
Conjugate Gradient	7.5224	9.0598	3.3966	12.2557	0.6138	0.7392	0.2771

4. BFGS

a. Iteration 1

$$B_t = \begin{pmatrix} -0.6771 & 0.3952 & 0.1338 \\ & -0.5690 & 0.2051 \\ & & -0.9768 \end{pmatrix}$$

$$B_t^* = \begin{pmatrix} -0.2780 & 0.1276 & 0.0511 \\ & -0.2501 & 0.0593 \\ & & -0.2280 \end{pmatrix}$$

	x_1	x_2	x_3		x_1	x_2	x_3
Newton-Raphson	6.7466	7.6270	4.8255	11.2682	0.5987	0.6769	0.4282
Conjugate Gradient	4.2788	5.0121	1.9501	6.8726	0.6226	0.7293	0.2838

b. Iteration 2

$$B_t = \begin{pmatrix} -0.6391 & 0.4369 & 0.1585 \\ & -0.5238 & 0.2333 \\ & & -0.9644 \end{pmatrix}$$

$$B_t^* = \begin{pmatrix} -0.0634 & 0.0172 & 0.0109 \\ & -0.0610 & 0.0115 \\ & & -0.0871 \end{pmatrix}$$

	x_1	x_2	x_3		x_1	x_2	x_3
Newton-Raphson	10.8447	11.7436	6.0193	17.0807	0.6349	0.6875	0.3524
Conjugate Gradient	7.6884	8.9427	3.7078	12.3625	0.6219	0.7234	0.2999