SECTION: CONTINUOUS OPTIMISATION
LECTURE 4: QUASI-NEWTON METHODS

HONOUR SCHOOL OF MATHEMATICS, OXFORD UNIVERSITY
HILARY TERM 2006, DR RAPHAEL HAUSER
WITH A FEW ADDITIONS FROM DR. NICK GOULD

1. The Quasi-Newton Idea. In this lecture we will discuss unconstrained minimisation methods that form a tradeoff between the advantages and disadvantages of the steepest descent and Newton-Raphson methods: we aim at methods that converge faster than steepest descent but have a lower operation count per iteration than Newton-Raphson. The family of algorithms we will consider are called *quasi-Newton* methods, as justified by the following motivation.

In Problem 2 of this week's exercises you will learn that an alternative derivation of the Newton–Raphson method is via the replacement of the minimisation problem $\min_{x\in\mathbb{R}^n} f(x)$ by $\min_{x\in\mathbb{R}^n} q(x)$, where $q(x)$ is the second order Taylor approximation of f around x_k, that is,

$$q(x) = f(x_k) + \langle \nabla f(x_k), x - x_k\rangle + \frac{1}{2}(x - x_k)^{\mathrm{T}} D^2 f(x_k)(x - x_k).$$

If $D^2 f(x_k)$ is positive definite, then the minimisation of $q(x)$ yields

$$x_{k+1} = x_k - \big(D^2 f(x_k)\big)^{-1} \nabla f(x_k)$$

as the optimal solution, and this is taken to be the next iterate in the Newton-Raphson process.

If instead of the Hessian $D^2 f(x_k)$ we use an approximation $B_k \approx D^2 f(x_k)$, we can generalise this idea and consider the minimisation problem $\min_{x\in\mathbb{R}^n} p(x)$, where $p(x)$ is a quadratic model

$$p(x) = f(x_k) + \langle \nabla f(x_k), x - x_k\rangle + \frac{1}{2}(x - x_k)^{\mathrm{T}} B_k (x - x_k)$$

of the objective function f. The minimisation of $p(x)$ yields the iterate

$$x^* = x_k - B_k^{-1} \nabla f(x_k).$$

This update is well-defined when B_k is nonsingular, and in particular when B_k is positive definite symmetric. Since B_k is only an approximation of $D^2 f(x_k)$, the update $d_k = x^* - x_k$ is used as a search direction rather than an exact update. A line-search then yields a new Quasi-Newton iterate

$$x_{k+1} = x_k + \alpha_k d_k.$$

Suppose we are given a rule for cheaply computing B_{k+1} as a function of the previously computed gradients $\nabla f(x_0), \dots, \nabla f(x_{k+1})$, or as a function of the previously computed quantities $B_k, \nabla f(x_k)$ and $\nabla f(x_{k+1})$. Then a generic quasi Newton algorithm proceeds as follows:

Algorithm 1.1 (Generic quasi-Newton).

S0 *Choose a starting point $x_0 \in \mathbb{R}^n$, a nonsingular $B_0 \in S^n$ (often the choice is $B_0 = I$), and a termination tolerance $\epsilon > 0$. Set $k = 0$.*

S1 *If $\|\nabla f(x_k)\| \le \epsilon$ then stop and output x_k as an approximate local minimiser of f. Else go to* **S2**.

S2 *Compute the quasi-Newton search direction $d_k = -B_k^{-1}\nabla f(x_k)$.*

S3 *Perform a practical line-search for the minimisation of $\phi(\alpha) = f(x_k + \alpha d_k)$: find a step length α_k that satisfies the Wolfe conditions and compute the new iterate $x_{k+1} = x_k + \alpha_k d_k$.*

S4 *Compute the new approximate Hessian B_{k+1} according to the specified rule.*

S5 *Replace k by $k+1$ and go to* **S1**.

1.1. A Wish List. To turn the generic procedure outlined in Algorithm 1.1 into a practical algorithm, we have to specify how to update the approximate Hessian B_k. Many variants of such updating rules have been proposed, and we will discuss the two most widely used choices in Sections 2 and 3.

But before doing that, we draw up a "wish list" of properties we would like to find in B_k. These properties serve as a motivation for almost all quasi-Newton methods, not merely the ones discussed in Sections 2 and 3.

P1: B_k should be nonsingular, so that **S2** *is well-defined.*

P2: B_k should be such that d_k is a descent direction, so that **S3** *is well-defined.*

P3: B_k should be symmetric, because Hessians are symmetric matrices.

Note that properties P1–P3 can be satisfied for example by requiring that B_k be positive definite symmetric, as P1 and P3 are then trivially true, and P2 follows from

$$\langle \nabla f(x_k), d_k\rangle = -\nabla f(x_k)^{\mathrm{T}} B_k^{-1} \nabla f(x_k) < 0,$$

unless $\nabla f(x_k) = 0$, in which case Algorithm 1.1 already stopped and the line-search is unnecessary.

One might argue that a positive definite matrix B_k is a poor choice as a Hessian approximation when $D^2 f(x_k)$ has at least one negative eigenvalue. However, such iterates x_k are not near a local minimiser, and at such points one is primarily concerned with finding a descent direction, and d_k is then a valid choice for any B_k positive definite. This avoids that the quasi-Newton method gets attracted to any point but a local minimiser.

P4: B_{k+1} should be computable by "recycling" the quantities

$$\nabla f(x_{k+1}), \nabla f(x_k), \dots, \nabla f(x_0), d_k, \alpha_k$$

which have to be computed anyway.

In order to meet this requirement while computing a B_{k+1} that is a credible Hessian approximation, it is useful to observe that the gradient change

$$\gamma_k := \nabla f(x_{k+1}) - \nabla f(x_k)$$

yields information about the Hessian change $D^2 f(x_{k+1}) - D^2 f(x_k)$: let us write $\delta_k := \alpha_k d_k$ for the update chosen by Algorithm 1.1. Recall that the search direction

d_k computed in **S2** is motivated by the fact that the gradient change predicted by the quadratic model

$$p(x) = f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2}(x - x_k)^{\mathrm{T}} B_k (x - x_k)$$

is

$$\begin{aligned} \nabla f(x_k + d_k) - \nabla f(x_k) &\approx \nabla p(x_k + d_k) - \nabla p(x_k) \\ &= \nabla f(x_k) + B_k d_k - \nabla f(x_k) \\ &= -\nabla f(x_k). \end{aligned} \tag{1.1}$$

In other words, it is predicted that $x_k + d_k$ is exactly a stationary point of f.

But of course, p is only a locally valid model of f, and since the new iterate x_{k+1} is obtained via a line search, the true gradient change $\gamma_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ is different from the prediction (1.1). A clever way to incorporate γ_k into the Hessian approximations is therefore to choose B_{k+1} so that the quadratic model

$$h(x) = f(x_k) + \langle \nabla f(x_k), (x - x_k) \rangle + \frac{1}{2}(x - x_k) B_{k+1} (x - x)$$

would have correctly predicted the observed gradient change:

$$\gamma_k = \nabla f(x_{k+1}) - \nabla f(x_k) = \nabla h(x_{k+1}) - \nabla h(x_k) = \nabla f(x_k) + B_{k+1}\delta_k - \nabla f(x_k).$$

In other words, B_{k+1} should be chosen such that

$$B_{k+1}\delta_k = \gamma_k \tag{1.2}$$

holds true. (1.2) is called the *secant condition.*

> *P5: B_{k+1} should be close to B_k in a well-defined sense, so that B_k can converge to $D^2 f(x^*)$ and d_k is allowed to become the Newton-Raphson step asymptotically.*

A straightforward idea to define a notion of closeness is by use of a matrix norm: $d(B_{k+1}, B_k) = \|B_{k+1} - B_k\|$. However, it is often more useful to characterise closeness by keeping the rank of $B_{k+1} - B_k$ as low as possible. This choice automatically guarantees that the last property on our wish list is satisfied too:

> *P6: The choice of B_k should be such that the overall work per iteration is at most of order $O(n^2)$, to gain a substantial speed-up over the $O(n^3)$ computer operations needed to perform a Newton-Raphson step.*

2. Symmetric Rank 1 Updates. Let us now describe a simple method that satisfies some but not all of the properties P1–P6. P3 and P5 can be satisfied by requiring that B_{k+1} is a rank-1 update of B_k, that is, we want to select some vector u and set

$$B_{k+1} = B_k + uu^{\mathrm{T}}, \tag{2.1}$$

where the uu^{T} is the usual matrix product with u seen as a $n \times 1$ matrix. The matrix uu^{T} – called the *outer product* of u with itself – is clearly symmetric, and all its columns are scalar multiples of u. Therefore, the rank of this matrix is 1, which explains the the name "rank-1 update". Thus, if we start out with a symmetric matrix B_0, then B_k will be symmetric for all k, as all the updates are symmetric. Moreover, the rank-1 condition (2.1) guarantees that B_k and B_{k+1} are "close" to each other in the rank sense.

So far we did not specify the choice of u. This choice is fixed when the requirements of P4 are satisfied through the secant condition (1.2): multiplying (2.1) by δ_k and substituting the result into (1.2), we find

$$(u^{\mathrm{T}}\delta_k)u = \gamma_k - B_k\delta_k. \tag{2.2}$$

Multiplying the transpose of this equation by δ_k, we obtain

$$(u^{\mathrm{T}}\delta_k)^2 = (\gamma_k - B_k\delta_k)^{\mathrm{T}}\delta_k. \tag{2.3}$$

Equation (2.2) shows that

$$u = \frac{\gamma_k - B_k\delta_k}{u^{\mathrm{T}}\delta_k},$$

thus, (2.1) and (2.3) imply that the updating rule should be as follows,

$$\begin{aligned} B_{k+1} &= B_k + \frac{(\gamma_k - B_k\delta_k)(\gamma_k - B_k\delta_k)^{\mathrm{T}}}{(u^{\mathrm{T}}\delta_k)^2} \\ &= B_k + \frac{(\gamma_k - B_k\delta_k)(\gamma_k - B_k\delta_k)^{\mathrm{T}}}{(\gamma_k - B_k\delta_k)^{\mathrm{T}}\delta_k}. \end{aligned} \tag{2.4}$$

Note that since $\gamma_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $\delta_k = \alpha_k d_k$, we can compute the SR1 update from the "recycled" information referred to in P4.

When B_{k+1} is computed via the updating rule (2.4) Algorithm 1.1 is called the *symmetric rank 1 method* (or SR1). This method was independently suggested by Broyden, Davidson, Fiacco-McCormick, Murtagh-Sargent, and Wolfe in 1967-69. The updates of the SR1 method are very simple to compute, but they have the drawback that B_k is not always positive definite and d_k might not always be defined or be a descent direction. Moreover, $(\gamma_k - B_k\delta_k)^{\mathrm{T}}\delta$ can be close to zero which leads to very large updates.

It remains to address property P6. Note that once d_k is known, computing $\alpha_k, x_{k+1}, \nabla f(x_{k+1}, \gamma_k$ and δ_k is very cheap. Moreover, since an outer product of two $n \times 1$ vectors necessitates the computation of n^2 entries, and since adding two $n \times n$ matrices takes n^2 additions, the total work for computing the updated matrix B_{k+1} from B_k and d_k is of order $O(n^2)$.

However, in order to compute d_k we need to solve the linear system of equations

$$B_k d_k = -\nabla f(x_k). \tag{2.5}$$

Gaussian elimination takes $2n^3/3$ operations to compute d_k. There are better methods to solve the system (2.5) numerically, but all practical methods take $O(n^3)$ operations (although some theoretical methods reduce this complexity slightly). No known method can solve (2.5) in the $O(n^2)$ operations apparently asked for in P6. Intuitively, it should be clear that this is the case, because even if we guessed the solution, verifying that d_k satisfies the system necessitates the computation of n inner products – one for each column of B_k multiplied with d_k – each of which takes n multiplications and $n-1$ additions, and hence this takes $O(n^2)$ operations in total. (Our analysis is crude, as we perform only a simple operations count without taking into account facts such as that performing a multiplication on a computer takes much more time than an addition.) Thus, the SR1 method, it seems, does not satisfy the complexity requirements of P6. Luckily, a way out of the dilemma is given by the Sherman–Morrison–Woodbury formula:

THEOREM 2.1 (Sherman–Morrison–Woodbury). *If $B \in \mathbb{R}^{n\times n}$ and $U, V \in \mathbb{R}^{n\times p}$ are matrices then*

$$(B + UV^{\mathrm{T}})^{-1} = B^{-1} - B^{-1}U(I + V^{\mathrm{T}}B^{-1}U)^{-1}V^{\mathrm{T}}B^{-1}.$$

Proof. See problem set. □

The usefulness of this formula is quickly understood when we apply it in the context of SR1 updates: suppose we knew $H_k = B_k^{-1}$. Then, applying the Sherman-Morrison-Woodbury formula to $B_+ = B_{k+1}$, $B = B_k$, $U = u = (\gamma_k - B_k\delta_k)$ and $V = U^{\mathrm{T}}$ (that is, $p = 1$ in this case), we find

$$\begin{aligned} H_{k+1} &= (B_+)^{-1} \\ &= B^{-1} - B^{-1}u\bigl(1 + u^{\mathrm{T}}B^{-1}u\bigr)^{-1}u^{\mathrm{T}}B^{-1} \\ &= H_k + \frac{(\delta_k - H_k\gamma_k)(\delta_k - H_k\gamma_k)^{\mathrm{T}}}{(\delta_k - H_k\gamma_k)^{\mathrm{T}}\gamma_k}. \end{aligned}$$

Thus, H_{k+1} is just a rank 1 update of H_k. But since we have assumed that H_k is known, computing d_k takes only $O(n^2)$ work because this is obtained via the matrix-vector multiplication $d_k = -H_k\nabla f(x_k)$. Hence, computing δ_k and γ_k takes only $O(n^2)$ operations. Finally, because outer products of vectors take $O(n^2)$ operations, H_{k+1} can be computed from H_k in $O(n^2)$ time.

Of course, this saving was only possible because we assumed that $B_k^{-1} = H_k$ is known. But if we start Algorithm 1.1 with a matrix B_0 whose inverse is know, for example with $B_0 = \mathrm{I}$, then we need never form the matrices B_k and can work directly with the updates H_k in every iteration. The SR1 method then takes only $O(n^2)$ work per iteration, which is a considerable saving over the Newton-Raphson method which takes $O(n^3)$ work due to the solving of a linear system in each iteration.

It is possible to analyse the local convergence of the SR1 method and show that the method converges superlinearly in a neighbourhood of a local minimiser of f. Thus, if the SR1 method is properly implemented, it can combine convergence speeds similar to those of the Newton-Raphson method with a lower complexity.

3. BFGS Updates.

SR1 updates are easy to use, but they have the disadvantage that B_k is not guaranteed to be positive definite, and d_k is not guaranteed to be a descent direction.

We are now going to describe the most widely used optimisation algorithm in unconstrained optimisation: the BFGS (Broyden– Fletcher–Goldfarb–Shanno) method, which is based on an updating rule for B_k that satisfies all six properties P1–P6 on our wish list. Thus, this method overcomes the weaknesses of SR1 while retaining its strengths.

Maintaining positive definiteness comes at the slight price of requiring rank 2 updates: B_{k+1} is of the form

$$B_{k+1} = B_k + uu^{\mathrm{T}} + vv^{\mathrm{T}},$$

that is, the update consists of a sum of two symmetric rank 1 matrices, and such matrices are of rank 2 if u and v are linearly independent.

In a sense, the SR1 rule describes the "best possible" updates we can achieve with rank 1 updates, while the BFGS formula describes the "best possible" updates achievable with rank 2 updates.

BFGS updates are described by the relation

$$B_{k+1} = B_k - \frac{B_k\delta_k\delta_k^{\mathrm{T}}B_k}{\delta_k^{\mathrm{T}}B_k\delta_k} + \frac{\gamma_k\gamma_k^{\mathrm{T}}}{\gamma_k^{\mathrm{T}}\delta_k}, \tag{3.1}$$

where $\gamma_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $\delta_k = x_{k+1} - x_k = \alpha_k d_k$ are defined as in Section 1.1. We will see shortly how it is motivated mathematically.

The same comment applies to (3.1) as to the complexity of computing the update (2.4): if d_k was known, computing the update would take $O(n^2)$ operations, but the computation of d_k requires the solution of the linear system $B_k d_k = -\nabla f(x_k)$. Of course, the Sherman–Morrison–Woodbury formula could save us once again (see problem set), but here we will pursue a slightly different path to achieving a complexity reduction: maintaining B_k as a Cholesky factorisation $B_k = L_k L_k^{\mathrm{T}}$. In fact, as a by-product of our approach we will find that B_{k+1} inherits the positive definiteness property from B_k, a fact that automatically guarantees that the requirements P1 and P2 are met.

DEFINITION 3.1. *Let $X \in \mathbb{R}^{n\times n}$ be a symmetric matrix. We say that X has a* Cholesky factorisation *if there exists a lower-triangular matrix $L \in \mathbb{R}^{n\times n}$ with positive diagonal, that is, $L_{ij} = 0$ if $i < j$ and $L_{ii} > 0$ for all i, and such that $X = LL^{\mathrm{T}}$.*

PROPOSITION 3.2. *$B \in \mathbb{R}^{n\times n}$ has a Cholesky factorisation $B = LL^{\mathrm{T}}$ if and only if X is positive definite symmetric. Moreover, if the Cholesky factorisation exists, then L is unique.*

Proof. See e.g. Golub-Van Loan "Matrix Computations", 3rd edition, Johns Hopkins University Press. You may also attempt to prove this result by induction on n, which is not too difficult and makes you discover an actual algorithm for the computation of L. □

Suppose B_k is positive definite symmetric and that we know its Cholesky factorisation $B_k = L_k L_k^{\mathrm{T}}$. Then solving the linear system $B_k d_k = -\nabla f(x_k)$ is the same as solving the two triangular systems

$$L_k g_k = -\nabla f(x_k), \tag{3.2}$$

$$L_k^{\mathrm{T}} d_k = g_k. \tag{3.3}$$

Triangular systems are great because they can be solved by direct back-substitution. For example, the system

$$\begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

can be solved as follows: the value of x_1 can be read off the first equation, yielding $x_1 = 3/2$. After substituting this value into the second equation, we can simply read off the value of x_2 and find $x_2 = (4 - 3/2)/3$. If we generalise this procedure, we find that solving (3.2) for g_k takes $\sum_{l=1}^{n}[2(l-1)+1] = n^2$ operations, and likewise solving (3.3) for d_k takes n^2 operations.

Thus, if the Cholesky factorisation of B_k is available, the computation of d_k incurs an $O(n^2)$ cost and the BFGS method takes $O(n^2)$ work per iteration. The main idea of the BFGS method then becomes to seek a sensible updating rule $L_k \leftarrow L_{k+1}$ for the Cholesky factor and to take $B_{k+1} = L_{k+1}L_{k+1}^{\mathrm{T}}$, which by Proposition 3.2 automatically guarantees that B_{k+1} is symmetric positive definite. In order to address the requirement P5, we will attempt to minimise the distance of L_k and L_{k+1} in a well-defined sense. Furthermore, in order to meet the requirement P4, we will choose L_{k+1} so that B_{k+1} satisfies the secant condition (1.2).

Let us put that programme into practice: In order to avoid cluttering our equations with indices we write B, L, α d δ and γ for B_k, L_k, α_k, d_k, δ_k and γ_k respectively, and B_+, L_+ etc. instead of B_{k+1}, L_{k+1} and so forth. At first, we will neglect the condition that L is lower triangular and replace it by an arbitrary nonsingular matrix J such that $JJ^{\mathrm{T}} = B$ and $J_+J_+^{\mathrm{T}} = B_+$.

We propose to solve the minimisation problem

$$\min_{J_+} \|J_+ - J\|_F \tag{3.4}$$

$$\text{s.t. } J_+g = \gamma, \tag{3.5}$$

where $\|\cdot\|_F$ is the Frobenius norm and $g \in \mathbb{R}^n$ is a parameter vector, and then we choose g so that

$$J_+^{\mathrm{T}}\delta = g. \tag{3.6}$$

The result will be that $B_+ = J_+J_+^{\mathrm{T}}$ satisfies the quasi-Newton equation (1.2), and that $\|J_+ - J\|_F$, a measure of distance between B_+ and B, is minimised under this condition.

The minimisation problem (3.4) is clearly the same as the smooth strictly convex optimisation problem

$$\begin{aligned} &\min_{J_+} \operatorname{tr}\big((J_+ - J)(J_+ - J)^{\mathrm{T}}\big) \qquad (3.7)\\ &\text{s.t. } J_+g = \gamma, \end{aligned}$$

where $\operatorname{tr}(A) = \sum_i A_{ii}$ denotes the trace of a square matrix A. This problem can be reformulated as

$$\min_{J_+} \langle J_+ - J, J_+ - J\rangle \tag{3.8}$$

$$\text{s.t. } \langle J_+, e_ig^{\mathrm{T}}\rangle = \gamma_i \quad (i = 1, \dots, n), \tag{3.9}$$

where e_i is the i-th coordinate vector and

$$\begin{aligned} \langle\cdot,\cdot\rangle &: \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \to \mathbb{R}\\ (A, B) &\mapsto \langle A, B\rangle := \operatorname{tr}(AB^{\mathrm{T}}) = \sum_{i,j} a_{ij}b_{ij} \end{aligned}$$

is a Euclidean inner product on the vector space $\mathbb{R}^{n\times n}$ of $n \times n$ real matrices. It is easy to check that the Euclidean norm that corresponds to this inner product is the usual Frobenius norm, defined by

$$\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}.$$

Thus, $(\mathbb{R}^{n\times n}, \langle\cdot,\cdot\rangle)$ is a Euclidean space of dimension n^2. The problem (3.8) is then the task of finding the point closest to J in the affine subspace defined by the constraints (3.9). The minimum is achieved at the unique point J_+^* where $(J_+^* - J)$ is orthogonal to the affine subspace, that is, $(J_+^* - J) \in \operatorname{span}(e_1g^{\mathrm{T}}, \dots, e_ng^{\mathrm{T}})$.

Another way to express this is to say that J_+^* is characterised by the existence of a vector $\lambda^* \in \mathbb{R}^n$ such that

$$\begin{aligned} 2(J_+^* - J) - \lambda^*g^{\mathrm{T}} &= 0,\\ J_+g &= \gamma. \end{aligned}$$

Both conditions are satisfied for

$$\begin{aligned} J_+^* &= J + \frac{(\gamma - Jg)g^{\mathrm{T}}}{g^{\mathrm{T}}g} \qquad \text{and} \qquad (3.10)\\ \lambda^* &= 2\frac{\gamma - Jg}{g^{\mathrm{T}}g}, \end{aligned}$$

and hence J_+^* is the global minimiser of (3.4).

Condition (3.6) now becomes

$$J^{\mathrm{T}}\delta = \left(1 - \frac{(\gamma - Jg)^{\mathrm{T}}}{g^{\mathrm{T}}g}\delta\right)g, \tag{3.11}$$

and since $J^{\mathrm{T}}\delta \neq 0$, this can only be satisfied for $g = \beta J^{\mathrm{T}}\delta$ with $\beta \neq 0$. Substituting into (3.11), we obtain

$$1 = \left(1 - \frac{\beta(\gamma^{\mathrm{T}}\delta - \alpha\delta^{\mathrm{T}}B\delta)}{\beta^2\delta^{\mathrm{T}}B\delta}\right)\beta,$$

or

$$\beta = \pm\sqrt{\frac{\gamma^{\mathrm{T}}\delta}{\delta^{\mathrm{T}}B\delta}}. \tag{3.12}$$

Note that if α satisfies the Wolfe conditions of Lecture 2, then

$$\gamma^{\mathrm{T}}\delta = \big(\nabla f(x + \alpha d) - \nabla f(x)\big)^{\mathrm{T}}\alpha d \geq (c_2 - 1)\phi'(0) > 0.$$

Therefore, the square-root in (3.12) is real.

Expressing the update of J in terms of β, we find

$$J_+ = J + \frac{(\gamma - \beta B\delta)\delta^{\mathrm{T}}J}{\beta\delta^{\mathrm{T}}B\delta}. \tag{3.13}$$

The corresponding update of B is

$$\begin{aligned}
B_+ &= J_+ J_+^{\mathrm{T}} \\
&= JJ^{\mathrm{T}} + \frac{(\gamma - \beta B\delta)\delta^{\mathrm{T}} B}{\beta\delta^{\mathrm{T}} B\delta} + \frac{B\delta(\gamma - \beta B\delta)^{\mathrm{T}}}{\beta\delta^{\mathrm{T}} B\delta} + \frac{(\gamma - \beta B\delta)(\gamma - \beta B\delta)^{\mathrm{T}}}{\beta^2\delta^{\mathrm{T}} B\delta} \\
&= \ldots \\
&= B - \frac{B\delta\delta^{\mathrm{T}} B}{\delta^{\mathrm{T}} B\delta} + \frac{\gamma\gamma^{\mathrm{T}}}{\beta^2\delta^{\mathrm{T}} B\delta}.
\end{aligned}$$

Using (3.12), this yields the BFGS formula 3.1:

$$B_+ = B - \frac{B\delta\delta^{\mathrm{T}} B}{\delta^{\mathrm{T}} B\delta} + \frac{\gamma\gamma^{\mathrm{T}}}{\gamma^{\mathrm{T}}\delta}.$$

Note that this formula is independent of the sign of β, and more importantly of J, that is to say, we could have chosen *any* factorisation $B = JJ^{\mathrm{T}}$ of B into a nonsingular matrix J and its transpose and we would have ended up with the same update for B! This remarkable property makes it possible to choose $J = L$, where L is the Cholesky factor of B. With this choice the transpose of Equation (3.13) becomes

$$J_+^{\mathrm{T}} = L^{\mathrm{T}} + \frac{L^{\mathrm{T}}\delta(\gamma - \beta B\delta)^{\mathrm{T}}}{\beta\delta^{\mathrm{T}} B\delta}.$$

This is an upper triangular matrix plus a rank 1 update.

PROPOSITION 3.3. *Let $X \in \mathbb{R}^{n\times n}$ be a square matrix. Then there exist unique matrices $Q, R \in \mathbb{R}^{n\times n}$ such that Q is orthogonal (that is, $Q^{\mathrm{T}} Q = \mathrm{I}$) and R is upper triangular (that is $R_{ij} = 0$ if $i > j$) and with nonnegative diagonal entries (that is, $R_{ii} \geq 0$ for all i), and such that $X = QR$.*

Proof. See e.g. Golub-Van Loan, “Matrix Computations”, 3rd Edition, Johns Hopkins University Press. This result is easy to prove because the column vectors of Q are the vectors obtained when a Gram-Schmidt orthogonalisation is applied to the column vectors of X, and the columns of R consist of the scalar weights needed to write the columns of X as linear combinations of the columns of Q. The result can also be easily extended to rectangular matrices. $\square$

Using the triangular structure of J_+^{T}, it is easy to show that the QR factorisation of $J_+^{\mathrm{T}} = Q_+ R_+$ can be computed in $O(n^2)$ operations. But then we have

$$B_+ = J_+ J_+^{\mathrm{T}} = R_+^{\mathrm{T}} Q_+^{\mathrm{T}} Q_+ R_+ = R_+^{\mathrm{T}} R_+,$$

and this shows that $L_+ := R_+^{\mathrm{T}}$ is the Cholesky factor of B_+.

Having derived the BFGS update and a rule for computing the updates of the Cholesky factor of the approximate Hessian B_k, we can now formulate the BFGS algorithm with all the pieces put together:

ALGORITHM 3.4 (BFGS method).

S0 *Choose a starting point x_0 and a lower triangular matrix L_0 with positive diagonal entries (the usual choice is $L_0 = I$, if no other information about the problem is known). Set a termination tolerance $\epsilon > 0$. Set $k = 0$.*

S1 *If $\|\nabla f(x_k)\| \leq \epsilon$ then stop and output x_k as an approximate local minimiser.*

S2 *Otherwise solve the triangular system $L_k g_k = -\nabla f(x_k)$ for g_k, and then the triangular system $L_k^{\mathrm{T}} d_k = g_k$ for d_k.*

S3 *Perform a line search to find a positive step length $\alpha_k > 0$ such that $f(x_k + \alpha_k d_k) < f(x_k)$, and such that α_k satisfies the Wolfe conditions.*

S4 *Set $\delta_k := \alpha_k d_k$, $x_{k+1} = x_k + \delta_k$. Compute $\gamma_k := \nabla f(x_{k+1}) - \nabla f(x_k)$ and $\beta_k = \pm\sqrt{\gamma_k^{\mathrm{T}}\delta_k / \delta_k^{\mathrm{T}} B_k \delta_k}$.*

S5 *Compute*

$$J_{k+1}^{\mathrm{T}} = L_k^{\mathrm{T}} + \frac{L_k^{\mathrm{T}}\delta_k(\gamma_k - \beta_k B_k\delta_k)^{\mathrm{T}}}{\beta_k\delta_k^{\mathrm{T}} B\delta_k},$$

and then compute the QR factorisation $J_{k+1}^{\mathrm{T}} = Q_{k+1} R_{k+1}$. Set $L_{k+1} := R_{k+1}^{\mathrm{T}}$ and return to **S1**.

The BFGS algorithm spends only $O(n^2)$ computation time per iteration, yet its convergence behaviour is not unlike that of the Newton-Raphson method: it can be shown that the BFGS method has local Q-superlinear convergence (but not Q-quadratic). It can also be shown that if the BFGS algorithm is used on a strictly convex quadratic function and in conjunction with exact line searches, then B_k becomes the exact (constant) Hessian after n iterations.

4. Final Comments. In the lectures on trust region algorithms we will see that quasi-Newton updates are useful not only in combination with line searches, but also in methods that rely on building local models of the objective function in the form of polynomials of degree 2.

The complexity per iteration and the convergence speed of quasi-Newton methods are an exact tradeoff between the advantages and disadvantages of the steepest descent and Newton-Raphson methods. We summarise this in the following table, where $\mathcal{C}(f)$ denotes the cost of one function evaluation of f:

	cost per iteration	convergence rate
Steepest descent	$O\big(n\mathcal{C}(f)\big)$	Q-linear
Quasi-Newton	$O\big(n^2 + n\mathcal{C}(f)\big)$	Q-superlinear
Newton-Raphson	$O\big(n^3 + n^2\mathcal{C}(f)\big)$	Q-quadratic