Chapter 7: Large-Scale Unconstrained Optimization

Outline

Outline

Sparse Quasi-Newton Updates

- We demand that the quasi-Newton approximations B_k have the same (or similar) sparsity pattern as the true Hessian explicitly. This would reduce the storage requirements of the algorithm and perhaps give rise to more accurate Hessian approximations.
- Suppose that we know which components of the Hessian may be nonzero at some point in the domain of interest. That is, we know the contents of the set Ω defined by

$$\Omega \equiv \{(i,j) | [\nabla^2 f(x)]_{ij} \neq 0 \text{ for some } x \text{ in the domain of } f\}.$$

Sparse Quasi-Newton Updates

- Suppose that the current Hessian approximation B_k mirrors the nonzero structure of the exact Hessian, i.e., $(B_k)_{ij} = 0$ for (i, j) not in Ω.
- In updating B_k to B_{k+1}, then, we could try to find the matrix B_{k+1} that satisfies the secant condition, has the same sparsity pattern, and is as close as possible to B_k.
- Specifically, we define B_{k+1} to be the solution of the following quadratic program:

$$\min_{B} \quad \|B - B_k\|_F^2 \tag{1.1a}$$

$$s.t. \quad Bs_k = y_k, B = B^T, B_{ij} = 0 \ \forall (i, j) \in \bar{\Omega} \tag{1.1b}$$

- We note that B_{k+1} is not guaranteed to be positive definite. We omit further details of this approach because its practical performance has been disappointing. The fundamental weakness of this approach is that (1.1) is an inadequate model and can produce poor Hessian approximations.

Sparse Quasi-Newton Updates

- An alternative approach is to relax the secant equation, making sure that it is approximately satisfied along the last few steps rather than requiring it to hold strictly on the latest step.
- To do so, we define $S_k = [s_{k-m}, \cdots, s_{k-1}]$ and $Y_k = [y_{k-m}, \cdots, y_{k-1}]$ so that they contain them most recent difference pairs. Define the new Hessian approximation B_{k+1} to be the solution of

$$\min_{B} \quad \|BS_k - Y_k\|_F^2 \tag{1.2a}$$

$$s.t. \quad B = B^T, B_{ij} = 0 \ \forall (i,j) \in \bar{\Omega} \tag{1.2b}$$

- This convex optimization problem has a solution, but it is not easy to compute. Moreover, this approach can produce singular or poorly conditioned Hessian approximations. Even though it frequently outperforms methods based on (1.1), its performance on large problems has not been impressive.

Outline

Limited-Memory BFGS

- Because the approximations to the Hessian or its inverse are usually dense in quasi-Newton mehod, even when the true Hessian is sparse. The storage and computational requirements grow in proportion to n^2, and become excessive for large n. The quasi-Newton methods are not directly applicable to large optimization problems.
- We need to modify and extend quasi-Newton methods to make them suitable for large problems.
- Limited-memory quasi-Newton methods modify the techniques described in the quasi-Newton methods to obtain Hessian approximations that can be stored compactly in just a few vectors of length n.

Derivation of L-BFGS

Various limited-memory methods have been proposed; we will focus mainly on an algorithm known as L-BFGS, which as its name suggests, is based on the BFGS updating formula.

The main idea of this method is to **use curvature information from only the most recent iterations to construct the Hessian approximation**. Curvature information from earlier iterations, which is less likely to be relevant to the actual behavior of the Hessian at the current iteration, is discarded in the interests of saving storage.

Reformulation of BFGS

Each step of the BFGS method has the form

$$x_{k+1} = x_k - \alpha_k H_k \nabla f_k, \qquad k = 0, 1, 2, \cdots,$$

where α_k is the step length, and H_k is updated at every iteration by means of the formula

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T, \tag{2.1}$$

where

$$\rho_k = \frac{1}{y_k^T s_k}, \qquad V_k = I - \rho_k y_k s_k^T,$$

and

$$s_k = x_{k+1} - x_k, \qquad y_k = \nabla f_{k+1} - \nabla f_k.$$

We say that the matrix H_{k+1} is obtained by updating H_k using the pair $\{s_k, y_k\}$.

Derivation of L-BFGS

- By storing a certain number (say m) of the vector pairs $\{s_i, y_i\}$ that are used in the formula (2.1), we store a modified version of H_k implicitly.
- The product $H_k \nabla f_k$ can be obtained by performing a sequence of inner products and vector summations involving ∇f_k and the pairs $\{s_i, y_i\}$.
- After the new iterate is computed, the oldest vector pair in the set of pairs $\{s_i, y_i\}$ is deleted and replaced by the new pair $\{s_k, y_k\}$ obtained from the current step. In this way, the set of vector pairs includes curvature information from the m most recent iterations.
- Practical experience has shown that modest values of m (between 3 and 20, say) often produce satisfactory results.

Derivation of L-BFGS

- In detail, at iteration k, the current iterate is x_k and the set of vector pairs contains $\{s_i, y_i\}$ for $i = k - m, \cdots, k - 1$.
- Choose some initial Hessian approximation H_k^0 (in contrast to the standard BFGS iteration, this initial approximation is allowed to vary from iteration to iteration).
- Repeatedly applying the formula (2.1), the L-BFGS approximation H_k satisfies the following formula:

$$\begin{aligned} H_k \quad = \quad & (V_{k-1}^T \cdots V_{k-m}^T) H_k^0 (V_{k-m} \cdots V_{k-1}) \\ & + \rho_{k-m} (V_{k-1}^T \cdots V_{k-m+1}^T) s_{k-m} s_{k-m}^T (V_{k-m+1} \cdots V_{k-1}) \\ & + \rho_{k-m} (V_{k-1}^T \cdots V_{k-m+2}^T) s_{k-m} s_{k-m}^T (V_{k-m+2} \cdots V_{k-1}) \\ & + \cdots \\ & + \rho_{k-1} s_{k-1} s_{k-1}^T. \end{aligned} \tag{2.2}$$

- From this expression we can derive a recursive procedure to compute the product $H_k \nabla f_k$ efficiently.

Algorithm 1: L-BFGS two-loop recursion

$q \leftarrow \nabla f_k$;
for $i = k-1, k-2, \ldots, k-m$
$\quad \alpha_i \leftarrow \rho_i s_i^T q$;
$\quad q \leftarrow q - \alpha_i y_i$;
end (for)
$r \leftarrow H_k^0 q$;
for $i = k-m, k-m+1, \ldots, k-1$
$\quad \beta \leftarrow \rho_i y_i^T r$;
$\quad r \leftarrow r + s_i(\alpha_i - \beta)$
end (for)
stop with result $H_k \nabla f_k = r$.

Computational Complexity of Algorithm 1

- Without considering the multiplication $H_k^0 q$, the two-loop recursion scheme requires $4mn$ multiplications;
- If H_k^0 is diagonal, then n additional multiplications are needed.
- Apart from being inexpensive, this recursion has the advantage that the multiplication by the initial matrix H_k^0 is isolated from the rest of the computations, allowing this matrix to be chosen freely and to vary between iterations. We may even use an implicit choice of H_k^0 by defining some initial approximation B_k^0 to the Hessian (not its inverse), and obtaining r by solving the system $B_k^0 r = q$.

Choices for H_k^0

- A method for choosing H_k^0 that has proved to be effective in practice is to set $H_k^0 = \gamma_k I$, where

$$\gamma = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}. \tag{2.3}$$

  As discussed before, γ_k is the scaling factor that attempts to estimate the size of the true Hessian matrix along the most recent search direction.

- It is important that the line search be based on the Wolfe conditions or strong Wolfe conditions.

Algorithm 2: L-BFGS

Choose starting point x_0, integer $m > 0$;
$k \leftarrow 0$;
repeat
  Choose H_k^0 (for example, by using (7.20));
  Compute $p_k \leftarrow -H_k \nabla f_k$ from Algorithm 7.4;
  Compute $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where α_k is chosen to
    satisfy the Wolfe conditions;
  if $k > m$
    Discard the vector pair $\{s_{k-m}, y_{k-m}\}$ from storage;
  Compute and save $s_k \leftarrow x_{k+1} - x_k$, $y_k = \nabla f_{k+1} - \nabla f_k$;
  $k \leftarrow k + 1$;
until convergence.

Relationship with BFGS

- During its first $m-1$ iterations, L-BFGS is equivalent to the BFGS algorithm if the initial matrix H^0 is the same in both methods, and if L-BFGS chooses $H_k^0 = H_0$ at each iteration.
- In fact, we could reimplement the standard BFGS method by setting m to some large value in L-BFGS algorithm(larger than the number of iterations required to find the solution).
- However, as m approaches n (specifically, $m > n/2$), this approach would be more costly in terms of computer time and storage than the approach of Algorithm BFGS.

Computational Experiences

We test the behavior of L-BFGS for various levels of memory m. The following table gives the number of function and gradient evaluations (nfg) and the total CPU time on some test problems from the CUTE collection. The number of variables is indicated by n, and the termination criterion $\|\nabla f_k\| \leq 10^{-5}$ is used.

Problem	n	L-BFGS $m = 3$		L-BFGS $m = 5$		L-BFGS $m = 17$		L-BFGS $m = 29$	
		nfg	time	nfg	time	nfg	time	nfg	time
DIXMAANL	1500	146	16.5	134	17.4	120	28.2	125	44.4
EIGENALS	110	821	21.5	569	15.7	363	16.2	168	12.5
FREUROTH	1000	>999	—	>999	—	69	8.1	38	6.3
TRIDIA	1000	876	46.6	611	41.4	531	84.6	462	127.1

Results show that the algorithm tends to be less robust when m is small. As the amount of storage increases, the number of function evaluations tends to decrease, but since the cost of each iteration increases with the amount of storage, the best CPU time is often obtained for small values of m. Clearly, **the optimal choice of m is problem-dependent**.

Computational Experiences

- L-BFGS is often the approach of choice for large problems in which **the true Hessian is not sparse**, because some rival algorithms become inefficient.
  - In particular, a Newton method, in which the true Hessian is computed and factorized, is not practical in such circumstances. The L-BFGS approach may even outperform Hessian-free Newton methods such as Algorithm Newton-CG, in which Hessian-vector products are calculated by finite differences or automatic differentiation.
  - When the Hessian is dense but the objective function has partially separable structure, the methods that exploit this structure normally outperform L-BFGS by a wide margin in terms of function evaluations. In terms of computing time, however, L-BFGS can be more efficient due to the low cost of its iteration.
- Computational experience to date also indicates that L-BFGS is more rapid and robust than nonlinear conjugate gradient methods.

Weakness of L-BFGS

The main weaknesses of the L-BFGS method are that it often converges slowly, which usually leads to a relatively large number of function evaluations, and that it is inefficient on highly ill-conditioned problems. Specifically, on problems where the Hessian matrix contains a wide distribution of eigenvalues.

Relationship with CG Methods

Limited-memory methods evolved gradually as an attempt to improve nonlinear conjugate gradient methods, and early implementations resembled CG methods more than quasi-Newton methods. The relationship between the two classes is the basis of a *memoryless BFGS iteration*.

Relationship with CG Methods

Consider the Hestenes-Stiefel form of the nonlinear CG method. Since $s_k = \alpha_k p_k$, the search direction for this method can be written as

$$p_{k+1} = -\nabla f_{k+1} + \frac{\nabla f_{k+1}^T y_k}{y_k^T p_k} p_k = -\left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) \nabla f_{k+1} \equiv -\hat{H}_{k+1} \nabla f_{k+1}. \tag{2.4}$$

This formula resembles a quasi-Newton iteration, but the matrix $\hat{H}_{k+1}$ is neither symmetric nor positive definite. We could symmetrize it as $\hat{H}_{k+1}^T \hat{H}_{k+1}$, but this matrix does not satisfy the secant equation $\hat{H}_{k+1} y_k = s_k$, and is, in any case, singular.

Relationship with CG Methods

- An iteration matrix that is symmetric, positive definite, and satisfies the secant equation is given by

$$H_{k+1} = (I - \frac{s_k y_k^T}{y_k^T s_k})(I - \frac{y_k s_k^T}{y_k^T s_k}) + \frac{s_k s_k^T}{y_k^T s_k}, \tag{2.5}$$

  which is exactly the one obtained by applying a single BFGS update to the identity matrix.

- By using the notation of L-BFGS, we can rewrite (2.5) as

$$H_{k+1} = V_k^T V_k + \rho_k s_k s_k^T.$$

- An algorithm whose search direction is given by $p_{k+1} = -H_{k+1}\nabla f_{k+1}$, with H_{k+1} defined by (2.5), can be thought of as a "memoryless" BFGS method in which the previous Hessian approximation is always reset to the identity matrix before updating, and where only the most recent correction pair (s_k, y_k) is kept at every iteration. Alternatively, we can view the method as a variant of Algorithm 2 in which $m = 1$ and $H_k^0 = I$ at each iteration.

- In this sense, **L-BFGS is a natural extension of the memoryless method that uses extra memory to store additional curvature information**.

Relationship with CG Methods

- Consider the memoryless BFGS formula (2.5) in conjunction with an exact line search, for which $\nabla f_{k+1}^T p_k = 0$ for all k.
- We then obtain

$$p_{k+1} = -H_{k+1}\nabla f_{k+1} = -\nabla f_{k+1} + \frac{\nabla f_{k+1}^T y_k}{y_k^T p_k} p_k \tag{2.6}$$

  which is none other than the Hestenes-Stiefel CG method. Moreover, it is easy to verify that when $\nabla f_{k+1}^T p_k = 0$, the HS formula reduces to the Polak-Ribière formula.
- Even though the assumption of exact line searches is unrealistic, it is intriguing that the BFGS formula, which is considered to be the most effective quasi-Newton update, is related in this way to the PR and HS methods, which are the most efficient nonlinear CG methods.

General Limited-Memory Updating

$$B_{k+1} = B_k - a_k a_k^T + b_k b_k^T$$

A limited-memory method then can proceed by defining the basic matrix B_k^0 at each iteration and then updating according to the formula

$$B_k = B_k^0 + \sum_{i=k-m}^{k-1} [b_i b_i^T - a_i a_i^T].$$

The vector pairs $\{a_i, b_i\}$, $i = k-m, \cdots, k-1$, would then be recovered from the stored vector pairs $\{s_i, y_i\}$, $i = k-m, \cdots, k-1$. We can reduce the computational cost further by using the Compact representation of the quasi-Newton updating formula.

Compact Representation of BFGS Updating

We now describe an approach to limited-memory updating that is based on representing quasi-Newton matrices in outer-product form.

Theorem 1

Let B_0 be symmetric and positive definite and assume that the k vector pairs $\{s_i, y_i\}_{i=0}^{k-1}$ satisfy $s_i^T y_i > 0$. Let B_k be obtained by applying k BFGS updates with these vector pairs to B_0. We then have that

$$B_k = B_0 - \begin{bmatrix} B_0 S_k & Y_k \end{bmatrix} \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix} \tag{2.7}$$

where S_k and Y_k are the $n \times k$ matrices defined by

$$S_k = [s_0, \cdots, s_{k-1}], \text{ and } Y_k = [y_0, \cdots, y_{k-1}], \tag{2.8}$$

while L_k and D_k are the $k \times k$ matrices

$$(L_k)_{i,j} = \begin{cases} s_{i-1}^T y_{j-1} & \text{if } i > j, \\ 0 & \text{otherwise,} \end{cases} \qquad D_k = \operatorname{diag}[s_0^T y_0, \cdots, s_{k-1}^T y_{k-1}]. \tag{2.9}$$

Compact Representation of BFGS Updating

As in the L-BFGS algorithm, we will keep them most recent correction pairs $\{s_i, y_i\}$ and refresh this set at every iteration by removing the oldest pair and adding a newly generated pair. During the first m iterations, the update procedure described in above theorem can be used without modification, except that usually we make the specific choice $B_k^0 = \delta_k I$ for the basic matrix, where $\delta_k = 1/\gamma_k$ and γ_k is defined by (2.3). At subsequent iterations $k > m$, the update procedure needs to be modified slightly to reflect the changing nature of the set of vector pairs $\{s_i, y_i\}$ for $i = k - m, \cdots, k - 1$. Defining the $n \times m$ matrices S_k and Y_k by

$$S_k = [s_{k-m}, \cdots, s_{k-1}], \text{ and } Y_k = [y_{k-m}, \cdots, y_{k-1}], \tag{2.10}$$

Compact Representation of BFGS Updating

the matrix B_k resulting from m updates to the basic matrix $B_0^k = \delta_k I$ is given by

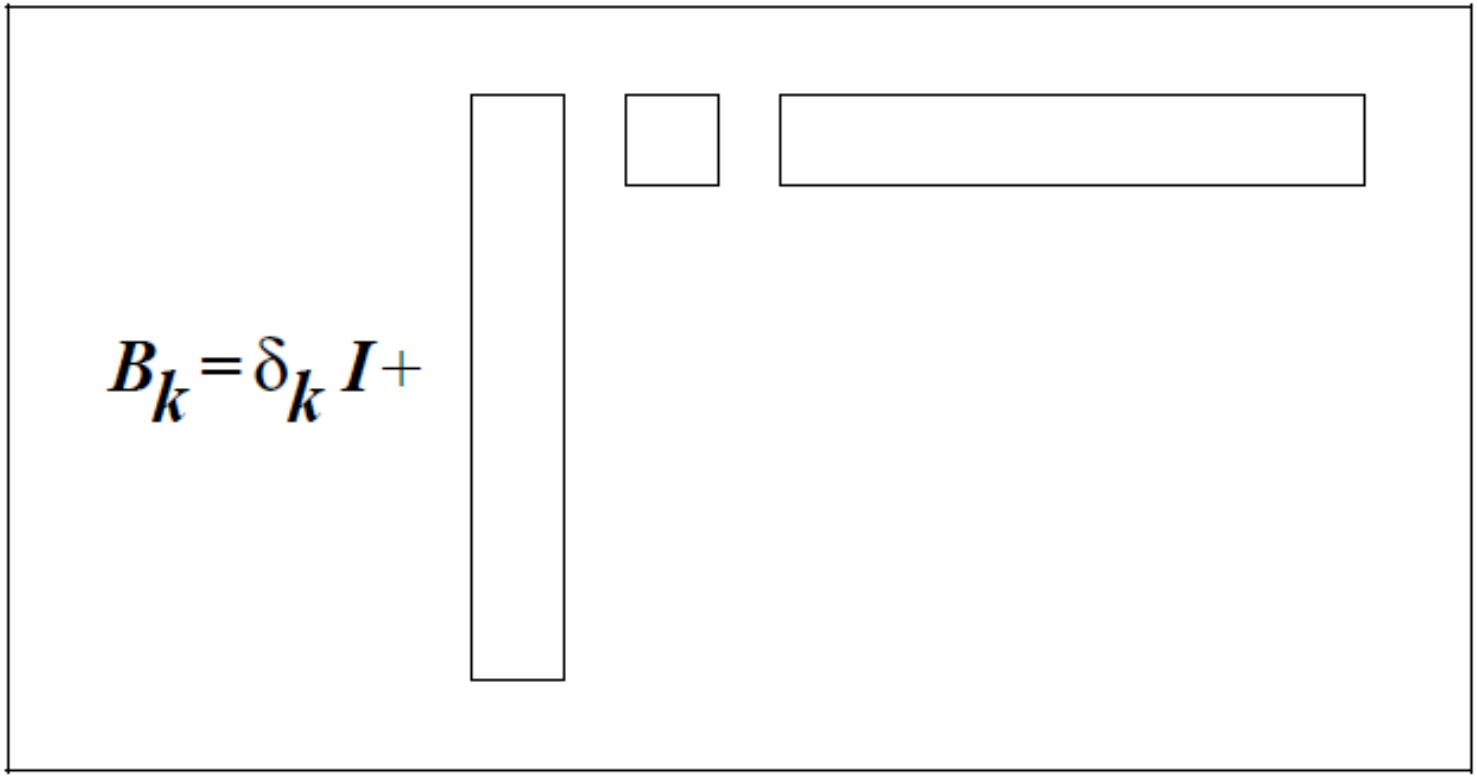
$$B_k = \delta_k I - \begin{bmatrix} \delta_k S_k & Y_k \end{bmatrix} \begin{bmatrix} \delta_k S_k^T S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} \delta_k S_k^T \\ Y_k^T \end{bmatrix} \tag{2.11}$$

while L_k and D_k are the $m \times m$ matrices

$$(L_k)_{i,j} = \begin{cases} s_{k-m-1+i}^T y_{k-m-j+1} & \text{if } i > j, \\ 0 & \text{otherwise,} \end{cases} \tag{2.12}$$

$$D_k = \mathrm{diag}[s_{k-m}^T y_{k-m}, \cdots, s_{k-1}^T y_{k-1}]. \tag{2.13}$$

Figure 1: Compact(or outer product) representation of B_k



SR1 Matrices

Theorem 2

Suppose that k updates are applied to the symmetric matrix B_0 using the vector pairs $\{s_i, y_i\}_{i=0}^{k-1}$ and the SR1 formula, and assume that each update is well-defined in the sense that $(y_i - B_i s_i)^T s_i = 0$, $i = 0, \cdots, k-1$. The resulting matrix B_k can be expressed as

$$B_k = B_0 + (Y_k - B_0 S_k)(D_k + L_k + L_k^T - S_k^T B_0 S_k)^{-1}(Y_k - B_0 S_k)^T, \tag{2.14}$$

where S_k, Y_k, D_k, and L_k are as defined in (2.8), (2.13), and (2.13).

Since the SR1 method is self-dual, the inverse formula H_k can be obtained simply by replacing B, s, and y by H, y, and s, respectively. Limited-memory SR1 methods can be derived in the same way as for the BFGS method. We replace B_0 with the basic matrix B_k^0 at the kth iteration, and we redefine S_k and Y_k to contain the m most recent corrections, as in (2.10).

General Limited-Memory Updating

- In fact, by representing quasi-Newton matrices in a compact (or outer product) form, we can derive efficient implementations of all popular quasi-Newton update formulae, and their inverses.
- These compact representations will also be useful in designing limited-memory methods for constrained optimization, where approximations to the Hessian or reduced Hessian of the Lagrangian are needed.
- These methods are fairly robust, inexpensive, and easy to implement, but they do not converge rapidly(linear).

Outline

Inexact Newton Methods

- The basic Newton step p_k^N is obtained by solving the symmetric $n \times n$ linear system

$$\nabla^2 f_k p_k^N = -\nabla f_k. \tag{3.1}$$

  The line search and trust-region Newton algorithms requires matrix factorizations of the Hessian matrices $\nabla^2 f_k$.

- In the large scale case, these factorizations can be carried out using **sparse elimination** techniques. If the computational cost and memory requirements of these sparse factorization methods are affordable for a given application, and if the Hessian matrix can be formed explicitly, Newton methods based on sparse factorizations constitute an effective approach for solving such problems.

Inexact Newton Methods

- Often, however, the cost of factorizing the Hessian is prohibitive, and it is preferable to compute approximations to the Newton step using iterative linear algebra techniques.
- We will discuss *inexact Newton methods* that use these techniques, in both line search and trust-region frameworks. The resulting algorithms have attractive global convergence properties and may be superlinearly convergent for suitable choices of parameters.
- They find effective search directions when the Hessian $\nabla^2 f_k$ is indefinite, and may even be implemented in a "Hessian-free" manner, without explicit calculation or storage of the Hessian.

Inexact Newton Methods

- Most rules for terminating the iterative solver for (3.1) are based on the residual

$$r_k = \nabla^2 f_k p_k + \nabla f_k, \tag{3.2}$$

  where p_k is the inexact Newton step.

- Since the size of the residual changes if f is multiplied by a constant (i.e., r is not invariant to scaling of the objective function), we consider its size relative to that of the right-hand-side vector in (3.1), namely $\nabla f(x_k)$.

- We therefore terminate the iterative solver when

$$\|r_k\| = \eta_k \|\nabla f_k\|, \tag{3.3}$$

  where the sequence $\{\eta_k\}$ (with $0 < \eta_k < 1$ for all k) is called the *forcing sequence*.

Local Convergence of Inexact Newton Methods

We now study how the rate of convergence of inexact Newton methods based on (3.1), (3.2) and (3.3) is affected by the choice of the forcing sequence. Our first result says that local convergence is obtained simply by ensuring that k is bounded away from 1.

Theorem 3

Suppose that $\nabla f(x)$ is continuously differentiable in a neighborhood of a minimizer x^*, and assume that $\nabla^2 f(x^*)$ is positive definite. Consider the iteration $x_{k+1} = x_k + p_k$ where p_k satisfies (3.3), and assume that $\eta_k \leq \eta$ for some constant $\eta \in [0, 1)$. Then, if the starting point x_0 is sufficiently near x^*, the sequence $\{x_k\}$ converges to x^* linearly. That is, for all k sufficiently large, we have

$$\|x_{k+1} - x^*\| \leq c\|x_k - x^*\|,$$

for some constant $0 < c < 1$.

Local Convergence of Inexact Newton Methods

Note that the condition on the forcing sequence $\{\eta_k\}$ is not very restrictive, in the sense that if we relaxed it just a little, it would yield an algorithm that obviously does not converge. Specifically, if we allowed $\eta_k \geq 1$, the step $p_k = 0$ would satisfy (3.3) at every iteration, but the resulting method would set $x_k = x_0$ for all k and would not converge to the solution.

Local Convergence of Inexact Newton Methods

Theorem 4

Suppose that the conditions of above theorem hold and assume that the iterates $\{x_k\}$ *generated by the inexact Newton method converge to* x^*. *Then the rate of convergence is superlinear if* $\eta_k \to 0$ *and quadratic if* $\eta_k = O(\|\nabla f(x_k)\|)$.

To obtain superlinear convergence we can set, for example, $\eta_k = \min(0.5, \sqrt{\|\nabla f(x_k)\|})$, while the choice $\eta_k = \min(0.5, \|\nabla f(x_k)\|)$ would yield quadratic convergence.

Proof Sketch

Proof Sketch. Since $r_k = \nabla^2 f_k d_k + \nabla f_k$, we have

$$d_k = [\nabla^2 f_k]^{-1}(r_k - \nabla f_k).$$

Due to the positive definiteness of $\nabla f(x^*)$ and $x_k \to x^*$, for all sufficiently large k,

$$\|d_k\| = O(\|r_k\| + \|\nabla f_k\|) = O(\|\nabla f_k\|).$$

Then

$$\begin{aligned}\|\nabla f(x_{k+1})\| &\le \|\nabla f(x_k) + \nabla^2 f_k d_k\| + O(\|d_k\|^2) \\ &= \|r_k\| + O(\|d_k\|^2) \\ &\le \eta_k \|\nabla f_k\| + O(\|\nabla f_k\|^2).\end{aligned}$$

Following from $\nabla f(x_{k+1}) \sim x_{k+1} - x^*$ and $\nabla f_{x_k} \sim x_k - x^*$, the theorems can be proved.

Local Convergence of Inexact Newton Methods

- All the results presented until now about the inexact Newton Methods are local in nature: They assume that the sequence $\{x_k\}$ eventually enters the near vicinity of the solution x^*.
- They also assume that the unit step length $\alpha_k = 1$ is taken, and hence that globalization strategies (line search, trust-region) do not get in the way of rapid convergence.
- In the next, we will show that inexact Newton strategies can, in fact, be incorporated in practical line search and trust-region implementations of Newton's method, yielding algorithms with good local and global convergence properties.

Line Search Newton-CG Method

Given initial point x_0;

for $k = 0, 1, 2, \ldots$

Compute a search direction p_k by applying the CG method to $\nabla^2 f(x_k)p = -\nabla f_k$, starting from $x^{(0)} = 0$. Terminate when $\|r_k\| \leq \min(0.5, \sqrt{\|\nabla f_k\|})\|\nabla f(x_k)\|$, or if negative curvature is encountered;

Set $x_{k+1} = x_k + \alpha_k p_k$, where α_k satisfies the Wolfe, Goldstein, or Armijo backtracking conditions;

end

Line Search Newton-CG Method

- It requires only that we can supply Hessian-vector products of the form $\nabla^2 f_k$ for any given vector d(do not require explicit knowledge of the Hessian). When the user cannot easily supply code to calculate second derivatives, or where the Hessian requires too much storage, the automatic differentiation and finite differencing techniques can be used to calculate these Hessian-vector products. Methods of this type are known as *Hessian-free* Newton methods.
- To illustrate the finite-differencing technique briefly, we use the approximation

$$\nabla^2 f_k d \approx \frac{\nabla f(x_k + hd) - \nabla f(x_k)}{h},$$

for some small differencing interval h. It is easy to prove that the accuracy of this approximation is $O(h)$. The price we pay for bypassing the hessian is one new gradient evaluation per CG iteration.

Line Search Newton-CG Method

- the specific starting point 0;
- the use of a positive tolerance allows the CG iterations to terminate at an inexact solution;
- the negative curvature test ensures that p_k is a descent direction for f at x_k;
- we can modify the CG iterations by introducing preconditioning;
- Weakness:
  - When $\nabla^2 f_k$ is nearly singular, it may lead to long direction;
  - Set a threshold to the negative curvature test $d_j^T B_k d_j \leq 0$, but good choices of the threshold are difficult to determine.

Trust-Region Newton-CG Method

Given $\epsilon > 0$;
Set $p_0 = 0, r_0 = g, d_0 = -r_0$;
if $\|r_0\| < \epsilon$
    return $p = p_0$;
for $j = 0, 1, 2, \ldots$
    if $d_j^T B d_j \leq 0$
        Find τ such that $p = p_j + \tau d_j$ minimizes $m(p)$
            and satisfies $\|p\| = \Delta$;
        return p;
    Set $\alpha_j = r_j^T r_j / d_j^T B d_j$;
    Set $p_{j+1} = p_j + \alpha_j d_j$;
    if $\|p_{j+1}\| \geq \Delta$
        Find $\tau \geq 0$ such that $p = p_j + \tau d_j$ satisfies $\|p\| = \Delta$;
        return p;
    Set $r_{j+1} = r_j + \alpha_j B d_j$;
    if $\|r_{j+1}\| < \epsilon \|r_0\|$
        return $p = p_{j+1}$;
    Set $\beta_{j+1} = r_{j+1}^T r_{j+1} / r_j^T r_j$;
    Set $d_{j+1} = r_{j+1} + \beta_{j+1} d_j$;
end (for).

Outline

Separable

In a *separable* unconstrained optimization problem, the objective function can be decomposed into a sum of simpler functions that can be optimized independently. For example, if we have

$$f(x) = f_1(x_1, x_3) + f_2(x_2, x_4, x_6) + f_3(x_5),$$

we can find the optimal value of x by minimizing each function f_i, $i = 1, 2, 3$, independently, since **no variable appears in more than one function**. The cost of performing n lower-dimensional optimizations is much less in general than the cost of optimizing an n-dimensional function.

Partially Separable

In many large problems the objective function $f : \Re^n \to \Re$ is not separable, but it can still be written as the sum of simpler functions, known as *element functions*. **Each element function has the property that it is unaffected when we move along a large number of linearly independent directions**. If this property holds, we say that f is *partially separable*. We will see that **all functions whose Hessians $\nabla^2 f$ are sparse are partially separable, but so are many functions whose Hessian is not sparse**. Partial separability allows for economical problem representation, efficient automatic differentiation, and effective quasi-Newton updating.

Partially Separable

- The simplest form of partial separability arises when the objective function can be written as

$$f(x) = \sum_{i=1}^{n_e} f_i(x), \tag{4.1}$$

where each of the element functions f_i depends on only a few components of x.

- It follows that the gradients ∇f_i and Hessians $\nabla^2 f_i$ of each element function contain just a few nonzeros. By differentiating (4.1), we obtain

$$\nabla f(x) = \sum_{i=1}^{n_e} \nabla f_i(x), \qquad \nabla^2 f(x) = \sum_{i=1}^{n_e} \nabla^2 f_i(x),$$

and it is natural to ask whether it is more effective to maintain quasi-Newton approximations to each of the element Hessians $\nabla^2 f_i(x)$ separately, rather than approximating the entire Hessian $\nabla^2 f(x)$.

- The answer is affirmative, provided that the quasi-Newton approximation fully exploits the structure of each element Hessian.

A Simple Example

- Consider the objective function

$$\begin{aligned} f(x) &= (x_1 - x_3^2)^2 + (x_2 - x_4^2)^2 + (x_3 - x_2^2)^2 + (x_4 - x_1^2)^2 \\ &\equiv f_1(x) + f_2(x) + f_3(x) + f_4(x), \end{aligned} \tag{4.2}$$

- Note that all f_i have exactly the same form. Let us focus on f_1. Even though f_1 is formally a function of all components of x, it depends only on x_1 and x_3, that we call the *element variables* for f_1.
- We assemble the element variables into a vector that we call $x_{[1]}$, that is,

$$x_{[1]} = \begin{bmatrix} x_1 \\ x_3 \end{bmatrix},$$

an note that

$$x_{[1]} = U_1 x \text{ with } U_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

A Simple Example

- If we define the function ϕ_1 by

$$\phi_1(z_1, z_2) = (z_1 - z_2^2)^2,$$

then we can write $f_1(x) = \phi_1(U_1 x)$.

- By applying the chain rule to this representation, we obtain

$$\nabla f_1(x) = U_1^T \nabla \phi_1(U_1 x), \qquad \nabla^2 f_1(x) = U_1^T \nabla^2 \phi_1(U_1 x) U_1, \tag{4.3}$$

- In our case, we have

$$\nabla^2 \phi_1(U_1 x) = \begin{bmatrix} 2 & -4x_3 \\ -4x_3 & 12x_3^2 - 4x_1 \end{bmatrix}, \nabla^2 f_1(x) = \begin{bmatrix} 2 & 0 & -4x_3 & 0 \\ 0 & 0 & 0 & 0 \\ -4x_3 & 0 & 12x_3^2 - 4x_1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- The matrix U_1, known as a *compactifying matrix*, allow us to map the derivative information for the low-dimensional function ϕ_1 into the derivative information for the element function f_1.

A Simple Example

- We maintain a 2×2 quasi-Newton approximation $B_{[1]}$ of $\nabla^2 \phi_1$.
- To update $B_{[1]}$ after a typical step from x to x^+, we record the information

$$s_{[1]} = x^+_{[1]} - x_{[1]}, \qquad y_{[1]} = \nabla \phi_1(x^+_{[1]}) - \nabla \phi_1(x_{[1]}) \tag{4.4}$$

  and use BFGS or SR1 updating to obtain the new approximation $B^+_{[1]}$.
- We therefore update small, dense quasi-Newton approximations with the property

$$B_{[1]} \approx \nabla^2 \phi_1(U_1 x) = \nabla^2 \phi_1(x_{[1]}). \tag{4.5}$$

- To obtain an approximation of the element Hessian $\nabla^2 f_1$, we use the transformation suggested by the relationship (4.3); that is,

$$\nabla^2 f_1(x) \approx U_1^T B_{[1]} U_1.$$

  This operation has the effect of mapping the elements of $B_{[1]}$ to the correct positions in the full $n \times n$ Hessian approximation.

A Simple Example

- The full objective function can now be written as

$$f(x) = \sum_{i=1}^{n_e} \phi_i(U_i x), \tag{4.6}$$

  and we obtain a quasi-Newton approximation $B_{[i]}$ for each one of ϕ_i.

- To obtain a complete approximation to the full Hessian $\nabla^2 f$, we simply sum the element Hessian approximations as follows:

$$B = \sum_{i=1}^{n_e} U_i^T B_{[i]} U_i. \tag{4.7}$$

- We may use this approximation Hessian in a trust-region algorithm, obtaining an approximation solution p_k of the system

$$B_k p_k = -\nabla f_k. \tag{4.8}$$

- We need not assemble B_k explicitly but rather use the CG method to solve (4.8), computing matrix-vector products of the form $B_k v$ by performing operations with the matrices U_i and $B_{[i]}$.

A Simple Example

To illustrate the usefulness of this element-by-element updating technique, let us consider a problem of the form (4.2) but this time involving 1000 variables, not just 4. The functions ϕ_i still depend on only two internal variables, so that each Hessian approximation $B_{[i]}$ is a 2×2 matrix. After just a few iterations, we will have sampled enough directions $s_{[i]}$ to make each $B_{[i]}$ an accurate approximation to $\nabla^2 \phi_i$. Hence the full quasi-Newton approximation (4.8) will tend to be a very good approximation to $\nabla^2 f(x)$.

By contrast, a quasi-Newton method that ignores the partially separable structure of the objective function of the element functions by approximating the 1000×1000 Hessian matrix. When the number of variables n is large, many iterations will be required before this quasi-Newton approximation is of good quality. Hence an algorithm of this type(for example, standard BFGS or L-BFGS) will require many more iteration than a method based on the partially separable approximation Hessian.

A Simple Example

- It is not always possible to use the BFGS formula to update the partial Hessian $B_{[i]}$, because there is no guarantee that the curvature condition $s_{[i]}^T y_{[i]} > 0$ will be satisfied.
- One way is to apply the SR1 update to each of the element Hessians. This approach has proved effective in the LANCELOT package, which is designed to take full advantage of partial separability.
- The main limitations of this quasi-Newton approach are the cost of the step computation(7.42), and the difficulty of identifying the partially separable structure of a function.

Outline

Perspective and Software: Newton-CG

Newton-CG methods have been used successfully to solve large problems in a variety of applications. Many of these implementations are developed by engineers and scientists and use **problem-specific preconditioners**.

- Freely available packages include TN/TNBC and TNPACK.
- Software for more general problems, such as LANCELOT, KNITRO/CG, and TRON, employ Newton-CG methods when applied to unconstrained problems.
- Other packages, such as LOQO implement Newton methods with a sparse factorization modified to ensure positive definiteness.
- GLTR offers a Newton-Lanczos method. There is insufficient experience to date to sat whether the Newton-Lanczos method is significantly better in practice than the Steihaug strategy.

Perspective and Software: Preconditioner

Software for computing incomplete Cholesky preconditioners includes the ICFS and MA57 packages. A preconditioner for Newton-CG based on limited-memory BFGS approximations is provided by PREQN.

Perspective and Software: L-BFGS

- Limited-memory BFGS methods are implemented in LBFGS and M1QN3.
- The compact limited-memory representations are used in LBFGS-B, IPOPT, and KNITRO.

Perspective and Software: Partial Separability

The LANCELOT package exploits partial separability. It provides SR1 and BFGS quasi-Newton options as well as a Newton methods. The step computation is obtained by a preconditioned conjugate gradient iteration using trust regions. If f is partially separable, a general affine transformation will not in general preserve the partially separable structure. The quasi-Newton method for partially separable functions is not invariant to affine transformations of the variables, but this is not a drawback because the method is invariant under transformations that preserve separability.

Thanks for your attention!