NEWTON'S METHOD FOR LARGE-SCALE OPTIMIZATION

Ali Bouaricha, Jorge J. Moré, and Zhijun Wu

1 Introduction

Algorithms for the solution of large-scale unconstrained minimization problems include conjugate gradient methods, limited memory variable metric methods, and Newton methods. In this paper we compare and contrast these algorithms, and show that the efficient and reliable solution of large-scale unconstrained minimization problems requires a Newton method. Moreover, we describe a Newton method that has proved to be efficient and reliable on optimization problems from applications.

Conjugate gradient methods for the minimization of a nonlinear function $f : \mathbb{R}^n \mapsto \mathbb{R}$ generates x_{k+1} from x_k by setting

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

where the scalar α_k is chosen by a line search, and the direction d_k is generated by a recurrence of the form

$$d_k = -\nabla f(x_k) + \beta_k d_{k-1}$$

for some $\beta_k \geq 0$. These methods only require a small number (5 is typical) of vectors of storage, but tend to require a large number of iterations for convergence. In most cases, better performance is obtained by a limited memory variable metric method.

Limited memory variable metric methods differ from conjugate gradient methods in that the user is allowed to choose the number of vector of storage, and in each iteration,

$$x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k)$$

where the symmetric, positive definite matrix H_k is determined by m vectors obtained during the previous m iterations, and stored in compact form so that the product $H_k \nabla f(x_k)$ only requires order mn flops (floating point operations). The number of iterations usually decreases as m increases, but the cost per iteration increases. In practice a choice of $m = 5$ is reasonable.

Limited memory variable metric methods have several advantages. For example, they only require the user to provide code for the evaluation of the function and the gradient, and the required storage is fixed. On the other hand, they tend to fail on difficult problems.

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the National Science Foundation, through the Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008.

Even if they converge, their rate of convergence can be too slow to obtain an accurate solution, if compared with a Newton's method. We will come back to this point when we discuss numerical results at the end of this paper.

An important difference between variations of Newton's method is how they obtain the Newton step. Implementations of Newton's method that use a direct method to obtain the step are not suitable for many large problems because of their cost in terms of computing time and storage. We prefer a variation of Newton's method that uses an iterative method to obtain the step because if properly preconditioned, this variation is efficient and reliable.

In this paper we propose a trust region version of Newton's method that uses an incomplete Cholesky decomposition and the conjugate gradient method to obtain the trust region step. In this algorithm the step s_k between iterates is an approximate solution to the trust region subproblem

$$\min \{q_k(w) : \|D_k w\| \le \Delta_k\},$$

where $q_k : \mathbb{R}^n \mapsto \mathbb{R}$ is a quadratic model of the function at the current iterate, D_k is a scaling matrix, Δ_k is the trust region radius, and $\|\cdot\|$ is the l_2 norm.

Other codes that are suitable for the solution of large-scale unconstrained minimization problems include LBFGS (Liu and Nocedal [16]), TN (Nash [20]), TNPACK (Folgeson and Schlick [24, 25]), SBMIN (Conn, Gould, and Toint [6, 5, 7]), and STENMIN (Bouaricha [3, 4]).

LBFGS is a limited memory variable metric method, TN and TNPACK are truncated Newton methods, and STENMIN is a tensor code that bases each iteration on a tensor model of the objective function. Of these codes, only TN and TNPACK use preconditioners. Both TN and TNPACK algorithms use a preconditioned conjugate gradient method to obtain the Newton step. The preconditioner used in TN is a scaled two-step limited memory BFGS method. In TNPACK, however, the user must supply a preconditioner; if the preconditioner is not positive definite, a modified Cholesky decomposition is used to obtain a positive definite preconditioner. The success of TNPACK clearly depends on the preconditioner. If the user supplies the Hessian matrix as a preconditioner, TNPACK reduces to a line search version of a modified Newton's method. With the Hessian as a preconditioner TNPACK would not be suitable for large problems, so the preconditioner must be chosen with care.

Numerical results for limited memory variable metric method and truncated Newton methods have been obtained by Phua [23], Nash and Nocedal [21], Gilbert and Lemarechal [9], Zou *et al.* [28], and Papadrakakis and Pantazopoulos [22].

The only other implementation of Newton's method that uses preconditioners is the Newton's method (SBMIN) in the LANCELOT package. Although our algorithm and the SBMIN algorithm rely on the trust region philosophy, both codes are quite different. We detail some of the differences when we describe the computation of the Newton's step in Section 3, and in the numerical results.

The test results obtained on the MINPACK-2 test problem collection [1] indicate that Newton's method is superior than a limited memory variable metric method (vmlm) in computing time, and usually requires far fewer function and gradient evaluations. We emphasize that the cost of the function and gradient evaluations of the MINPACK-2 test problems is moderate ($O(n)$). Therefore, on problems with expensive function and gradient evaluations, which is the case in many real-life applications, Newton's method is likely to be the winner because of its small number of function and gradient evaluations requirement.

The remainder of the paper is organized as follows. In Section 2 we describe the main steps of the trust region Newton method for large-scale problems. In Section 3 we describe the computation of the step within the trust region framework. A preconditioner based on the incomplete Cholesky factorization is described in Section 4. In Section 5 we give a brief description of the MINPACK-2 test problems which we will use in the comparison between Newton's method and vmlm. In Section 6 we compare the performance of the trust region Newton method with that of vmlm. A comparison with Lancelot is made in Section 7. Finally, in Section 8, we present our conclusions.

2 Newton's Method

At each iteration of a trust region Newton method for the minimization of $f : \mathbb{R}^n \mapsto \mathbb{R}$ we have an iterate x_k, a bound Δ_k, a scaling matrix D_k, and a quadratic model

$$q_k(w) = \nabla f(x_k)^T w + \tfrac{1}{2} w^T B_k w$$

of the possible reduction $f(x_k + w) - f(x_k)$ for $\|D_k w\| \leq \Delta_k$. Given a step s_k, the test for acceptance of the trial point $x_k + s_k$ depends on a parameter $\eta_0 > 0$. The following algorithm summarizes the main computational steps:

For $k = 0, 1, \ldots,$ maxiter
    Compute the quadratic model q_k.
    Compute a scaling matrix D_k.
    Compute an approximate solution s_k to the trust region subproblem.
    Compute the ratio ρ_k of actual to predicted reduction.
    Set $x_{k+1} = x_k + s_k$ if $\rho_k \geq \eta_0$; otherwise set $x_{k+1} = x_k$. Update Δ_k.

In this section we elaborate on this outline and on our implementation of the trust region Newton method for large-scale problems.

The computation of the model q_k requires the gradient and the approximate Hessian matrix B_k. In this work, B_k is usually either the Hessian matrix $\nabla^2 f(x_k)$, or a symmetric approximation to the Hessian matrix obtained by differences of the gradient.

The iterate x_k and the bound Δ_k are updated according to rules that are standard in trust region methods. Given a step s_k such that $\|D_k s_k\| \leq \Delta_k$ and $q_k(s_k) < 0$, these rules depend on the ratio

$$\rho_k = \frac{f(x_k + s_k) - f(x_k)}{q_k(s_k)} \tag{2.1}$$

of the actual reduction in the function to the predicted reduction in the model. Since the step s_k is chosen so that $q_k(s_k) < 0$, a step with $\rho_k > 0$ yields a reduction in the function. Given $\eta_0 > 0$, the iterate x_k is updated as in the basic algorithm, that is, $x_{k+1} = x_k + s_k$ if $\rho_k \geq \eta_0$, otherwise $x_{k+1} = x_k$. The updating rules for Δ_k depend on constants η_1 and η_2 such that

$$0 < \eta_0 < \eta_1 < \eta_2 < 1,$$

while the rate at which Δ_k is either increased or decreased depend on constants σ_1, σ_2, and σ_3 such that

$$0 < \sigma_1 < \sigma_2 < 1 < \sigma_3.$$

The trust region bound Δ_k is updated by setting

$$\begin{array}{lll} \Delta_{k+1} \in [\sigma_1 \Delta_k, \sigma_2 \Delta_k] & \text{if} & \rho_k \leq \eta_1 \\ \Delta_{k+1} \in [\sigma_1 \Delta_k, \sigma_3 \Delta_k] & \text{if} & \rho_k \in (\eta_1, \eta_2) \\ \Delta_{k+1} \in [\Delta_k, \sigma_3 \Delta_k] & \text{if} & \rho_k \geq \eta_2. \end{array}$$

We choose a step s_k that provides an approximate solution s_k to the trust region subproblem

$$\min \left\{ q_k(w) : \|D_k w\| \leq \Delta_k \right\}.$$

There is no need to compute an accurate minimizer; the main requirement is that the step s_k give as much reduction in the model q_k as the Cauchy step s_k^C, that is, a solution to the problem

$$\min \left\{ q_k(w) : \|D_k w\| \leq \Delta_k, \ w = -\nu \nabla f(x_k), \ \nu \in \mathbb{R} \right\}.$$

This requirement guarantees global convergence of the trust region method under suitable conditions.

The above outline of a trust region Newton method is standard. The main differences appear in the method used to compute the step s_k and the use of the scaling matrix D_k. We discuss the algorithm used to compute the step s_k in Section 3, and the scaling matrix in Section 4.

3 Computation of the step

The computation of the step in an unconstrained trust region method requires an approximate solution of the subproblem

$$\min \left\{ q(s) : \|Ds\| \leq \Delta \right\}, \tag{3.1}$$

where D is the scaling matrix and $q : \mathbb{R}^n \mapsto \mathbb{R}$ is the quadratic

$$q(s) = g^T s + \tfrac{1}{2} s^T B s.$$

In this formulation g is the gradient at the current iterate and B is the approximate Hessian matrix. In this section we describe the method used to compute the step, and contrast our approach with others that have appeared in the literature.

In our approach we transform the ellipsoidal trust region into a spherical trust region and then apply the conjugate gradient method to the transformed problem. Thus, given the problem (3.1), we transform this problem into

$$\min \left\{ \hat{q}(w) : \|w\| \leq \Delta \right\}, \tag{3.2}$$

where

$$\hat{q}(w) = \hat{g}^T w + \tfrac{1}{2} w^T \hat{B} w, \qquad \hat{g} = D^{-1} g, \qquad \hat{B} = D^{-1} B D^{-T}.$$

Given an approximate solution of (3.2), the corresponding solution of (3.1) is $s = D^{-T} w$.

We are interested in the solution of large-scale problems, and thus the conjugate gradient method, with suitable modifications that take into account the trust region constraint and the possible indefiniteness of $\hat{B}$, is a natural choice for computing a step w. The global convergence theory of the trust region method only requires that

$$\hat{q}(w) \leq \beta \hat{q}(s^C), \tag{3.3}$$

where s^C is the Cauchy step and β is a positive constant, but for fast local convergence we need

$$\|\nabla \hat{q}(w)\| \leq \xi \|\nabla \hat{q}(0)\|, \tag{3.4}$$

where $\xi \in (0,1)$ is a constant that may depend on the iteration.

At the moment we leave the matrix D unspecified. The main requirement on D is that the resulting $\hat{B}$ must have clustered eigenvalues. As we shall see in Section 4, this requirement is satisfied by choosing D from an incomplete Cholesky factorization of $\hat{B}$.

The classical conjugate gradient method for minimizing the quadratic $\hat{q}$ generates a sequence of iterates $\{w_k\}$ as follows:

> Let $w_0 \in \mathbb{R}^n$ be given. Set $r_0 = -(\hat{g} + \hat{B} w_0)$ and $d_0 = r_0$.
> For $k = 0, 1, \ldots,$
> > Compute $\alpha_k = \|r_k\|^2 / (d_k^T \hat{B} d_k)$.
> > Update the iterate: $w_{k+1} = w_k + \alpha_k d_k$.
> > Update the residual: $r_{k+1} = r_k - \alpha_k \hat{B} d_k$.
> > Compute $\beta_k = \|r_{k+1}\|^2 / \|r_k\|^2$.
> > Update the direction: $d_{k+1} = r_{k+1} + \beta_k d_k$.

Since the trust region (3.2) is centered at the origin, we choose $w_0 = 0$. With this choice, the conjugate gradient method generates iterates $w_1, \ldots, w_m$ where m is the largest integer such that $d_k^T \hat{B} d_k > 0$ for $0 \le k < m$, such that

$$\hat{q}(w_k) = \min \left\{ \hat{q}(w) : w \in \operatorname{span}\{r_0, \hat{B} r_0, \ldots, \hat{B}^{k-1} r_0\} \right\}, \qquad 0 \le k \le m.$$

The conjugate gradient method terminates with $r_m = 0$ or with $d_m^T \hat{B} d_m \le 0$. These properties of the conjugate gradient method are well-known, see, for example, Hestenes and Steifel [14] or Hackbusch [13, Section 9.5]. We also need to know that since we have chosen $w_0 = 0$, the iterates satisfy

$$\|w_k\| < \|w_{k+1}\|, \qquad 0 \le k < m.$$

This result was first obtained by Steihaug [26] from the inequality

$$(w_{k+1} - w_k)^T (w_{j+1} - w_j) > 0, \qquad j \le k.$$

As we shall see, this result is of importance to our development.

We claim that as the conjugate gradient method generates iterates for the trust region problem (3.2), one of the following three conditions will hold for some index k with $k \le m$:

$$\begin{array}{ll} \|w_k\| \le \Delta\ , & \|r_k\| \le \xi \|g\|. \\ \|w_k\| \le \Delta\ , & d_k^T \hat{B} d_k \le 0. \\ \|w_k\| \le \Delta, & \|w_{k+1}\| > \Delta. \end{array} \tag{3.5}$$

Clearly, if we continue to generate iterates with $\|w_k\| \le \Delta$ then the one of the first two conditions in (3.5) will hold for some $k \le m$; otherwise, we must exit the trust region at some iterate, and then the third condition holds.

Given an iterate w_k that satisfies the conditions in (3.5), we can compute a suitable step w. If for some $\xi \in (0,1)$ and $0 \le k \le m$ we have $\|w_k\| \le \Delta$ and $\|r_k\| \le \xi \|g\|$, then we accept $w = w_k$ as the step. In this case w is a truncated Newton step. If $\|w_k\| \le \Delta$ and $d_k^T \hat{B} d_k \le 0$, or if $\|w_k\| \le \Delta$ and $\|w_{k+1}\| > \Delta$, let $\tau_k > 0$ be the unique positive solution to the quadratic equation

$$\|w_k + \tau d_k\| = \Delta,$$

and set $w = w_k + \tau_k d_k$. In all cases we have

$$\hat{q}(w) \le \min\{\hat{q}(w_k), \hat{q}(s^C)\},$$

for the final step w, and thus this choice of step satisfies the sufficient decrease condition (3.3) of trust region methods.

Steihaug [26] suggested the trust region step described above, but his description was in terms of the preconditioned conjugate gradient method applied to the problem

$$\min \left\{ q(s) : (s^T C s)^{\frac{1}{2}} \le \Delta \right\},$$

where C is a positive definite matrix. The preconditioned conjugate gradient method for minimizing q takes the following form:

> Let $s_0 \in \mathbb{R}^n$ be given. Set $r_0 = -(g + Bs_0)$ and $d_0 = q_0$ where $Cq_0 = r_0$.
> For $k = 0, 1, \dots,$
>   Compute $\alpha_k = (r_k^T q_k)/(d_k^T B d_k)$.
>   Update the iterate: $s_{k+1} = s_k + \alpha_k d_k$.
>   Update the residual: $r_{k+1} = r_k - \alpha_k B d_k$.
>   Solve $Cq_{k+1} = r_{k+1}$.
>   Compute $\beta_k = (r_{k+1}^T q_{k+1})/(r_k^T q_k)$.
>   Update the direction: $d_{k+1} = q_{k+1} + \beta_k d_k$.

The relationship between this algorithm for generating $\{s_k\}$, and our algorithm for generating $\{w_k\}$ is that if $C = D^T D$ then $Ds_k = w_k$. This can be verified by an induction argument. Thus, the two methods are equivalent. However, our approach based on the subproblem (3.2) is more direct. Moreover, we deal with the scaling matrix D directly, and not through the matrix $C = D^T D$.

Conn, Gould, and Toint [6, 5] and [7, Sections 3.2 and 3.3] also use the preconditioned conjugate gradient method to compute steps in a trust region method. In their approach, the trust region subproblem

$$\min \{ q(s) : \|s\|_\infty \le \Delta \}$$

has no scaling matrix and uses the l_∞ norm. A preconditioned conjugate gradient method is used to generate the steps $\{s_k\}$; the preconditioner is a diagonal matrix in [6, 5], while more general preconditioners are used in [7].

A disadvantage of the Conn, Gould, and Toint [7] approach is that they cannot rely on the property that $\{\|s_k\|\}$ is monotonically increasing; instead, they have that $\{s_k^T C s_k\}$ is monotonically increasing, where C is the preconditioner. As a consequence iterates may enter and leave the trust region several times. This does not destroy the global convergence properties of the algorithm, but can lead to poor behavior because we would expect later iterates to produce better steps. In particular, if the first iterate exits the trust region, then their approach generates a steepest descent iterate.

4 Incomplete Cholesky factorizations as preconditioners

The efficiency and reliability of the algorithm for computing the trust region step depends on the preconditioner. There are many widely used preconditioners, ranging from diagonal to full-matrix preconditioners. However, choosing a good preconditioner for a given problem has always been a difficult task. On one hand, preconditioners such as diagonal or band preconditioners may not be successful if the essential part of the Hessian matrix is not contained within the diagonal or the band, respectively. Full-matrix preconditioners, on the other hand, may be prohibitive if it is too expensive to factor or store the Hessian matrix. A reasonable compromise in efficiency and cost for finding and storing the factorization of the Hessian matrix are the so-called incomplete factorization preconditioners. In this section we describe a preconditioner based on the incomplete Cholesky factorization.

Given a symmetric sparsity pattern $\mathcal{S}$, the incomplete Cholesky factorization proposed by Meijerink and Van Der Vorst [18] computes a lower triangular matrix L such that

$$B = LL^T + R, \qquad l_{i,j} = 0 \text{ if } (i,j) \notin \mathcal{S}, \quad r_{i,j} = 0 \text{ if } (i,j) \in \mathcal{S}. \tag{4.1}$$

The modified incomplete Cholesky factorization of Gustafsson [10] replaces the requirement that $r_{i,j} = 0$ by $Re = 0$, where e is the vector of all ones. For additional information on modified incomplete Cholesky factorizations, see Gustafsson [11, 12] and Hackbusch [13].

A difficulty with an incomplete Cholesky factorization is that it is not clear how to choose the sparsity pattern $\mathcal{S}$ for the numerical factorization. We could choose $\mathcal{S}$ to be the sparsity pattern of B, but it is usually advantageous to allow some fill. Several approaches for defining $\mathcal{S}$ have been proposed. Fixed fill strategies (Meijerink and Van Der Vorst [18]) fix the nonzero structure of the incomplete factor prior to the factorization. Drop-tolerance strategies (Munksgaard [19], for example) include nonzeros in the incomplete factor if they are larger than some threshold parameter. Thus, the number of nonzeros in the incomplete factor is unknown prior to the factorization.

Another difficulty with an incomplete Cholesky factorization is that the factorization may fail or be unstable for a general positive definite matrix. The standard solution for this situation is to increase the size of the elements in the diagonal until a satisfactory factorization is obtained. See, for example, Manteuffel [17] and Munksgaard [19].

The strategies described above require that the user impose a fixed sparsity pattern on the Cholesky factor or specify a drop-tolerance. Jones and Plassmann [15] have proposed an incomplete Cholesky factorization that avoids these requirements. In their approach the m_k largest nonzeroes in the off-diagonal part of the k-th column of L (plus the k-th diagonal element of L) are retained, where m_k is the number of nonzero off-diagonal elements in the k-th column of the lower triangular part of B , and thus the number of nonzeros in the incomplete factor is the same as in the original matrix. The numerical results of Jones and Plassmann [15] show that significant improvements in performance are obtained on problems generated from finite element models, and on problems from the Harwell-Boeing sparse matrix collection. Their algorithm can be summarized as follows:

Algorithm 4.1 Let B be a symmetric matrix, and let m_k be the number of nonzero off-diagonal elements in the k-th column of the lower triangular part of B.

For $k = 0, 1, \ldots,$

Compute the off-diagonal elements in the k-th column of L.

Update the last $n - k$ diagonal elements of L.

Select the m_k nonzeros of the off-diagonal part of the k-th column of L with the largest magnitude.

Algorithm 4.1 is a modified Cholesky factorization in the sense that (4.1) holds. The sparsity pattern $\mathcal{S}$, however, is generated dynamically.

The performance of Algorithm 4.1 depends on the size of the elements of B. For a general positive definite matrix B, Jones and Plassmann [15] recommended computing a scaled matrix

$$\hat{B} = D^{-1/2} B D^{-1/2}, \qquad D = \text{diag}(b_{i,i}),$$

and using Algorithm 4.1 on $\hat{B}_k = \hat{B} + \alpha_k I$ for some $\alpha_k \geq 0$. If the incomplete Cholesky factor of $\hat{B}_k$ is $\hat{L}_k$, then $L_k = D^{1/2} \hat{L}_k$ is the factor of $B + \alpha_k D$. The algorithm used to generate α_k was to start with $\alpha_0 = 0$, and increment α_k by a constant factor (0.01) until Algorithm 4.1 succeeded.

This approach does not extend to general indefinite matrices. In the general case B may have zero or small diagonal elements, and then the scaling above is almost certain to produce a badly scaled matrix. We also note that the strategy of adding a constant factor to the diagonal is not likely to be efficient in general.

In our approach we scale the initial matrix by the l_2 norm of the columns of B, and use a more aggressive update for α_k. The following algorithm specifies our strategy in detail.

Algorithm 4.2 Let B be a symmetric matrix.

Compute $\hat{B} = D^{-1/2} B D^{-1/2}$ where $D = \text{diag}(\|Be_i\|_2)$.

Let $\beta = \|\hat{B}\|_\infty$ and set $\alpha_0 = 0$ if $\min(b_{i,i}) > 0$; otherwise $\alpha_0 = \beta/2$.

For $k = 0, 1, \ldots,$

Use Algorithm 4.1 on $\hat{B}_k = \hat{B} + \alpha_k I$; exit if successful.

Set $\alpha_{k+1} = \max(2\alpha_k, \frac{1}{2}\beta)$

For any α_0, this algorithm is guaranteed to generate an α_2 with $\alpha_2 \geq \beta$, and thus $\hat{B}_2$ is diagonally dominant. Hence, the incomplete Cholesky factorization exists. The choice of $\alpha_0 = 0$ is certainly reasonable if B is positive definite, or more generally, if B has positive diagonal elements. A reasonable initial choice for α_0 is not clear if B is an indefinite matrix, but our choice of $\alpha_0 = \beta/2$ seems to be adequate.

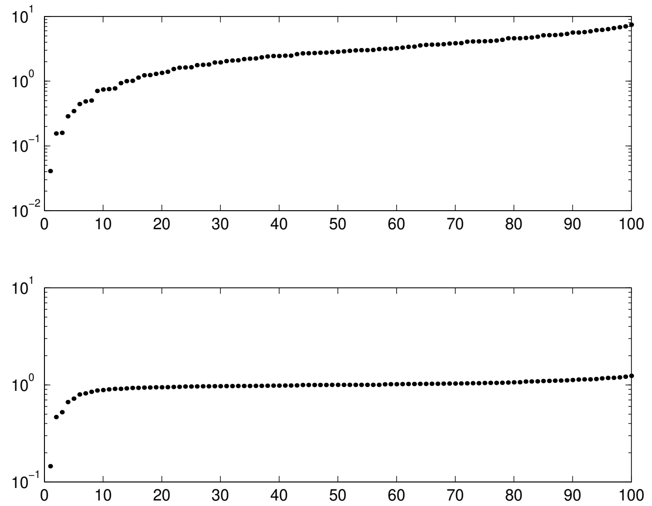

Figure 4.1: Eigenvalues of B (top) and $L_k^{-1} B L_k^{-T}$ (bottom) for the MSA problem

Algorithm 4.2 produces the incomplete Cholesky factor $\hat{L}_k$ of the final $\hat{B}_k$, and thus $L_k = D^{1/2} \hat{L}_k$ is the factor of $B + \alpha_k D$. Since we are interested in using L_k as a preconditioner for B in the conjugate gradient method, we need the spectrum of the preconditioned matrix

$$\hat{L}_k^{-1} \hat{B} \hat{L}_k^{-T} = L_k^{-1} B L_k^{-T}$$

to have clustered eigenvalues. We give an idea of the behavior of L_k as a preconditioner by plotting the eigenvalues of B and of the preconditioned matrix $L_k^{-1} B L_k^{-T}$ for two problems.

Figure 4.1 is a plot of the eigenvalues of B and of the preconditioned matrix for the MSA problem with $n = 100$ variables. In this problem the Hessian matrix is positive definite and Algorithm 4.2 produces the incomplete Cholesky factorization with $\alpha_0 = 0$. Note that the eigenvalues of the preconditioned matrix are clustered and that the condition number of the preconditioned matrix is reduced. The important effect is the clustering of the eigenvalues.

The plot in Figure 4.1 is fairly typical of the positive definite problems that we have tried, and thus we expect good behavior from the conjugate gradient method. For more information on the effect of clustering on the convergence behavior of the conjugate gradient method, see Van der Vorst [27]. We now consider indefinite problems.

Figure 4.2 has the plots of the eigenvalues of B and of the preconditioned matrix for a randomly generated sparse symmetric matrix of order 100. We used the `sprandsym` function in MATLAB with a density of 0.05 and a distribution of eigenvalues defined by

$$\lambda_k = \rho_k 10^{s_k}, \qquad s_k = \text{ncond} \left(\frac{k-1}{n-1} \right)$$

where ρ_k is uniformly distributed in $[-1, 1]$ and ncond $= 8$. For this problem Algorithm 4.2 produced the incomplete Cholesky factorization with $\alpha_2 = \beta$.

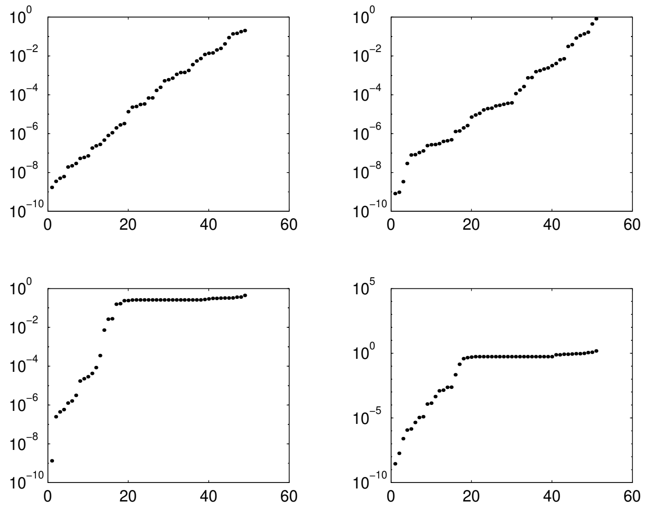

Figure 4.2: Magnitude of the eigenvalues of B and $L_k^{-1}BL_k^{-T}$ for a randomly generated sparse indefinite matrix of order 100. The positive and negative eigenvalues of B are on the top left and top right, respectively. Similarly, the positive and negative eigenvalues of $L_k^{-1}BL_k^{-T}$ are on the bottom left and bottom right, respectively.

Since the Hessian matrix is indefinite, we have plotted the magnitude of both the positive and the negative eigenvalues. The plots of the eigenvalues of the original matrix are on the top of Figure 4.2, while the plots of the eigenvalues of the preconditioned matrix are on the bottom. These plots show that Algorithm 4.2 tends to cluster the eigenvalues of largest magnitude, and tends to reduce the condition number of the preconditioned problem. In general we do not get the almost spectacular clustering shown in Figure 4.2, but clustering of the largest positive eigenvalues does seem to be a general trend. In a trust region method we would want to have clustering of the positive eigenvalues because the algorithm for computing the step in Section 3 terminates successfully once the conjugate gradient method determines a direction of negative curvature.

We emphasize that Algorithm 4.2 is invariant to changes of scale in the matrix B, that is, if we replace B by D_0BD_0, where D_0 is a nonsingular diagonal matrix, then the preconditioned matrix $L_k^{-1}BL_k^{-T}$ is unchanged. Scaling by the l_2 norm of the columns of B guarantees this invariance and works better than setting

$$D = \operatorname{diag}\{|b_{i,i}|, \epsilon\}, \quad \text{or} \quad D = \operatorname{diag}\{b_{i,i}, \epsilon\},$$

for some $\epsilon > 0$, which are scalings that are commonly used in optimization (see, for example, Gay [8], Conn, Gould, and Toint [7, page 125].

5 Large-Scale Optimization Problems

Most of our test problems come from the MINPACK-2 test problem collection [1] since this collection is representative of large-scale optimization problems arising from applications. Below we give brief descriptions of the infinite dimensional version of these problems, and of the finite dimensional formulations. The MINPACK-2 report contains additional information on these problems; in particular, parameter values are chosen as in this report.

The optimization problems that we consider arise from the need to minimize a function f of the form

$$f(v) = \int_{\mathcal{D}} \Phi(x, v, \nabla v)\, dx, \tag{5.1}$$

where $\mathcal{D}$ is some domain in either $\mathbb{R}$ or $\mathbb{R}^2$, and Φ is defined by the application. In all cases f is well defined if $v : \mathcal{D} \mapsto \mathbb{R}^p$ belongs to $H^1(\mathcal{D})$, the Hilbert space of functions such that v and $\|\nabla v\|$ belong to $L^2(\mathcal{D})$. This is the proper setting for examining existence and uniqueness questions for the infinite-dimensional problem.

Finite element approximations to these problems are obtained by minimizing f over the space of piecewise linear functions v with values $v_{i,j}$ at $z_{i,j}$, $0 \le i \le n_y + 1$, $0 \le j \le n_x + 1$, where $z_{i,j} \in \mathbb{R}^2$ are the vertices of a triangulation of $\mathcal{D}$ with grid spacings h_x and h_y. The vertices $z_{i,j}$ are chosen to be a regular lattice so that there are n_x and n_y interior grid points in the coordinate directions, respectively.

Lower triangular elements T_L are defined by vertices $z_{i,j}, z_{i+1,j}, z_{i,j+1}$, while upper triangular elements T_U are defined by vertices $z_{i,j}, z_{i-1,j}, z_{i,j-1}$. The values $v_{i,j}$ are obtained by solving the minimization problem

$$\min \left\{ \sum \left(f_{i,j}^L(v) + f_{i,j}^U(v) \right) : v \in \mathbb{R}^n \right\},$$

where $f_{i,j}^L$ and $f_{i,j}^U$ are the finite element approximation to the integrals in the elements T_L and T_U, respectively.

The elastic plastic torsion (EPT) and the journal bearing problem (JBP) are quadratic problems of the form

$$\min\{f(v) : v \in K\}, \tag{5.2}$$

where $f : K \mapsto \mathbb{R}$ is the quadratic

$$f(v) = \int_{\mathcal{D}} \left\{ \tfrac{1}{2} w_q(x) \|\nabla v(x)\|^2 - w_l(x) v(x) \right\} dx,$$

where $w_q : \mathcal{D} \mapsto \mathbb{R}$ and $w_l : \mathcal{D} \mapsto \mathbb{R}$ are functions defined on the rectangle $\mathcal{D}$. In the EPT problem $w_q \equiv 1$ and $w_l \equiv c$. For our numerical results we use $c = 5$. In the JBP problem

$$w_q(\xi_1, \xi_2) = (1 + \epsilon \cos \xi_1)^3, \quad w_l(\xi_1, \xi_2) = \epsilon \sin \xi_1$$

for some constant ϵ in $(0, 1)$, and $\mathcal{D} = (0, 2\pi) \times (0, 2b)$ for some constant $b > 0$. For our numerical results we use $\epsilon = 0.1$ and $b = 10$.

The steady state combustion (SSC) problem and the optimal design with composites (ODC) are formulated in terms of a family of minimization problems of the form

$$\min\{f_\lambda(v) : v \in H_0^1(\mathcal{D})\}.$$

For the SSC problem $f_\lambda : H_0^1(\mathcal{D}) \mapsto \mathbb{R}$ is the functional

$$f_\lambda(v) = \int_{\mathcal{D}} \left\{ \tfrac{1}{2}\|\nabla v(x)\|^2 - \lambda \exp[v(x)] \right\} dx,$$

and $\lambda \geq 0$ is a parameter. For the ODC problem

$$f_\lambda(v) = \int_{\mathcal{D}} \left\{ \psi_\lambda\left(\|\nabla v(x)\|\right) + v(x) \right\} dx,$$

and $\psi_\lambda : \mathbb{R} \mapsto \mathbb{R}$ is the piecewise quadratic

$$\psi_\lambda(t) = \begin{cases} t^2, & 0 \leq t \leq t_1, \\ 2t_1(t - \frac{1}{2}t_1), & t_1 \leq t \leq t_2, \\ \frac{1}{2}(t^2 - t_2^2) + 2t_1(t_2 - \frac{1}{2}t_1), & t_2 \leq t, \end{cases}$$

with the breakpoints t_1 and t_2 defined by $t_1^2 = \lambda$, $t_2^2 = 2\lambda$. In our numerical results we consider the problem of minimizing f_λ for a fixed value of λ; for the SSC problem $\lambda = 2$, while for the ODC problem $\lambda = 0.008$.

The minimal surface area (MSA) problem is an optimization problem of the form (5.2) where $f : K \mapsto \mathbb{R}$ is the functional

$$f(v) = \int_{\mathcal{D}} \left(1 + \|\nabla v(x)\|^2\right)^{1/2} dx,$$

and the set K is defined by

$$K = \left\{ v \in H^1(\mathcal{D}) : v(x) = v_D(x) \text{ for } x \in \partial\mathcal{D} \right\}$$

for the boundary data function $v_D : \partial\mathcal{D} \mapsto \mathbb{R}$ that specifies the Enneper minimal surface.

In all the problems so far v has been defined in some domain in $\mathbb{R}^2$. For the one-dimensional Ginzburg-Landau problem (GL1) v is defined in a subset of $\mathbb{R}$. This problem is defined by

$$\min\{f(v) : v(-d) = v(d),\ v \in C^1[-d, d]\},$$

where $2d$ is a constant (the width of the superconducting material), and

$$f(v) = \frac{1}{2d} \int_{-d}^{d} \left\{ \alpha(\xi)|v(\xi)|^2 + \tfrac{1}{2}\beta(\xi)|v(\xi)|^4 + \gamma|v'(\xi)|^2 \right\} d\xi.$$

The functions α and β are piecewise constant, and γ is a (universal) constant.

The Ginzburg-Landau problem (GL2) can be phrased as an optimization problem of the general form (5.1), where v is defined in $\mathbb{R}^4$. The first two components represent a complex-valued function ψ (the order parameter), and the other two components a vector-valued function (the vector potential). The dimensionless form of this problem is

$$\min\{f_1(\psi) + f_2(\psi, A) : \psi, A \in H_0^1(\mathcal{D})\},$$

where $\mathcal{D}$ is a two-dimensional region,

$$f_1(\psi) = \int_{\mathcal{D}} \left\{ -|\psi(x)|^2 + \tfrac{1}{2}|\psi(x)|^4 \right\} \, dx,$$

$$f_2(\psi, A) = \int_{\mathcal{D}} \left\{ \left\| [\nabla - iA(x)]\, \psi(x) \right\|^2 + \kappa^2 \left\| (\nabla \times A)(x) \right\|^2 \right\} \, dx,$$

and κ is the Ginzburg-landau constant.

6 Numerical Results

Our aim in this section is to analyze the performance of the **nmtrs** code that implements a trust region version of Newton's method. In particular, we compare the performance of the trust region code with a limited memory variable metric code (**vmlm**) since several numerical studies have shown that algorithms of this type compare well with other codes for the solution of large-scale problems.

We consider the behavior of the algorithm as the number of variables n varies between $2,500$ and $40,000$ since one of our purposes is to study the behavior of these algorithms as the number of variables increases and to show that the trust region code can easily produce solutions for problems where the number of variables is in this range.

For most of our results we imposed a limit of one hour of computing time per problem, and a limit of $5,000$ gradient evaluations. Our computations were performed on a Sun SPARC 10 workstation using double precision arithmetic. The termination test used in these results was

$$\|\nabla f(x)\| \le \tau \|\nabla f(x_0)\|, \qquad \tau = 10^{-5}.$$

This test is scale invariant, and scales as the number of variables increases, but we do not recommend this test as a general termination test for optimization algorithms. The choice of $\tau = 10^{-5}$ tests the ability to solve optimization problems to moderate accuracy.

In the trust region algorithm we accept x_{k+1} as a next iterate if the ratio ρ_k in (2.1) is bigger than 10^{-4}; otherwise we update the trust region radius as follows:

$$\begin{array}{lll}
\Delta_{k+1} = 0.5\ \Delta_k & \text{if} & \rho_k < 0.25 \\
\Delta_{k+1} = \Delta_k & \text{if} & \rho_k \in (0.25, 0.5] \\
\Delta_{k+1} = 2.0\ \Delta_k & \text{if} & \rho_k \in (0.5, 0.9] \\
\Delta_{k+1} = 4.0\ \Delta_k & \text{if} & \rho_k \ge 0.9
\end{array}$$

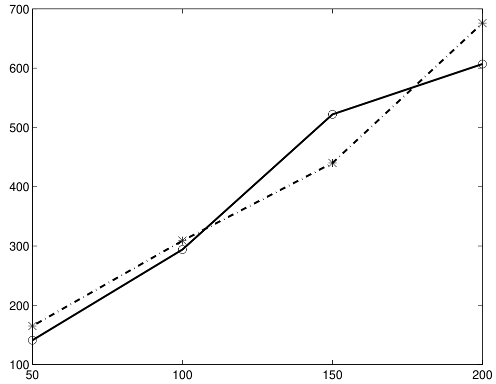

Figure 6.1: Number of iterations of vmlm as a function of $n^{1/2}$

We chose $\Delta_0 = \min(1000.0 \parallel g(x_0) \parallel, 1000.0)$, where $g(x_0)$ is the gradient value at the initial guess x_0. We selected this value of Δ_0 based on the experimental test results. In the conjugate gradient iterative algorithm, the value of ξ in (3.4) is set to 10^{-2} for all our tests.

We have to change the way Δ is chosen; the above choice stinks!

Table 6.1 contains a summary of the results for the nmtrs and vmlm codes on the MINPACK-2 large-scale problems [1]. In this table `iters` is the number of iterations, `nfev` is the number of function and gradient evaluations, `nhev` is the number of Hessian evaluations, `ncg` is the number of conjugate gradient iterations, and `time` is the computing time (in seconds).

An important difference between a limited memory variable metric method and a Newton method is that for the Newton method the number of iterations required to solve a variational problem (such as those described in Section 5) can be independent of n. This can be seen in Table 6.1 for most of the problems. In contrast, the number of iterations required by a limited memory variable metric method is likely to grow with n. Table 6.1 suggests that for these problems the number of iterations grows like $n^{1/2}$. This is verified in Figure 6.1 by plotting the number of gradient evaluations required to solve the SSC and MSA problems.

Since the computing time for evaluating the function and gradient of these problems grows linearly with n, and the number of iterations grows like $n^{1/2}$, the computing time to solve these problems with vmlm grows like $n^{3/2}$. In particular, this implies that if we quadruple the number of variables in vmlm, then the computing time grows by a factor of 8. This is verified by the results in Table 6.1.

For the Newton method the computing time depends on the cost of the four main components of the Newton iteration: the function and gradient evaluation, the evaluation of the Hessian matrix, the incomplete Cholesky factor of the Hessian matrix, and the conjugate gradient iteration. In general we cannot expect a linear growth in the computing cost for

Table 6.1: Performance of **nmtrs** versus **vmlm** on the MINPACK-2 test problems

		NMTRS					vmlm		
Problem	n	`iters`	`nfev`	`nhev`	`ncg`	`time`	`iters`	`nfev`	`time`
EPT	2500	3	4	4	27	0.8	124	133	2.8
EPT	10000	3	4	4	46	3.9	280	293	25.1
EPT	40000	3	4	4	88	22.4	610	629	217.0
PJB	2500	3	4	4	32	0.8	226	236	5.1
PJB	10000	3	4	4	61	4.6	423	445	38.2
PJB	40000	3	4	4	120	28.0	823	855	310.2
GL2	2500	8	15	9	439	25.6	2710	2786	50.1
GL2	10000	11	17	12	1109	171.5	†4831	†5000	†368.0
GL2	40000	7	13	8	1052	506.9	†4839	†5000	†1535.0
MSA	2500	6	7	7	68	2.8	138	144	4.7
MSA	10000	6	7	7	98	12.4	269	277	34.8
MSA	40000	10	14	11	229	91.2	609	623	307.0
SSC	2500	3	4	4	33	1.4	167	176	7.5
SSC	10000	3	4	4	59	6.8	345	358	60.8
SSC	40000	3	4	4	113	35.6	588	615	414.8
GL1	2500	14	23	14	62	1.5	†4844	†5000	†61.7
GL1	10000	15	17	15	44	5.5	†4847	†5000	†246.3
GL1	40000	21	30	21	61	30.9	†4854	†5000	†933.0
ODC	2500	40	61	41	253	20.3	260	268	11.3
ODC	10000	187	274	188	858	362.4	410	415	69.3
ODC	40000	882	1217	883	3946	7114.0	1209	1228	815.9

†Limit of 5000 function evaluations reached.

Table 6.2: Percentages of the computing time for NMTRS and $n = 10,000$

Problem	$f(x)$, $\nabla f(x)$	$\nabla^2 f(x)$	ICF	CG
EPT	6	28	14	49
PJB	5	26	12	55
GL2	1	19	47	33
MSA	7	50	9	33
SSC	11	44	8	36
GL1	4	28	30	33
ODC	9	68	10	11

Table 6.3: Performance of vmlm as a function of m for $n = 10,000$

	$m = 5$			$m = 10$			$m = 15$			$m = 20$		
Problem	`iters`	`nfev`	`time`	`iters`	`nfev`	`time`	iters	nfev	time	iters	nfev	time
EPT	280	293	25	244	249	26	239	248	30	191	198	27
PJB	423	445	38	394	404	43	392	400	48	339	350	48
GL2	4831	5000	368	3344	3441	322	3537	3622	395	2700	2780	355
MSA	269	277	35	221	226	32	217	226	36	219	225	40
SSC	345	358	60	278	289	54	236	242	49	240	250	55
ODC	410	415	69	475	489	90	395	403	81	472	480	106

Newton's method because the number of conjugate gradient iterations is likely to grow with n; this holds even for simple model problems like EPT.

The results in Table 6.1 show that the computing time of NMTRS grows almost like $n^{3/2}$. The reason for this is that for these problems the cost of the conjugate gradient iterations tends to dominate, and the number of conjugate gradient iterations tends to grow like $n^{1/2}$.

An analysis of the computing time reveals where the computing time is spent. In Table 6.2 we present the percentages of the overall computing time spent by the various components of the Newton method: the function and gradient evaluation, the Hessian evaluation, the incomplete Cholesky factorization, and the conjugate gradient iteration. In this table $n = 10,000$.

Table 6.2 shows that, with the exception of the ODC and MSA problems, the cost of the conjugate gradient iterations dominates the computing time. Elaborate on this.

We tried to improve the performance of the vmlm code by increasing the number of vectors saved. The results of a series of runs with $m = 5, 10, 15, 20$ appear in Table 6.3.

These results show that the number of function and gradient evaluations decreases as we increase m from $m = 5$ to $m = 10$. The 50% reduction for the GL2 problem and the 25% reduction for the MSA problem are dramatic, but the reductions are not as dramatic for the other problems.

Note that in almost all cases the computing time increases as we increase m. We would have expected a decrease in computing time if the cost of evaluating the function and gradient dominated, but this is not the case for these problems. The **vmlm** algorithm requires $(8m + 12)n$ operations per iteration; the number of operations for the function and gradient is harder to estimate since most of the problems require the evaluation of intrinsic functions. The cost of the function and gradient evaluations relative to the internal arithmetic of the algorithm can be measured by computing the ratio $t_f/t_a^{(m)}$, where t_f is the time required to evaluate the function and gradient, and $t_a^{(m)}$ is the time required for the $(8m + 12)n$ operations. These ratios are shown in Table 6.4 for $m = 5$.

Table 6.4: Ratios $t_f/t_a^{(m)}$ of **vmlm** for $n = 10,000$ and $m = 5$

Problem	EPT	PJB	GL2	MSA	SSC	ODC
$t_f/t_a^{(m)}$	1.0	1.2	1.3	1.9	2.3	2.5

Given these ratios, we could have predicted that we would need roughly a 20% reduction in the number of function and gradient evaluations before we would see an improvement in performance for **vmlm**. We establish this claim by first noting that if k_m is the number of function and gradient evaluations required for convergence then

$$t_m = (t_f + t_a^{(m)})k_m$$

is the time required for convergence. We have ignored the time required by the algorithm when the line search requires more than one function and gradient evaluation since the number of operations in this part of the code is a small multiple of n. Now note that since $t_a^{(2m)} \geq 1.7t_a^{(m)}$ for $m \geq 5$, we have

$$t_{2m} \geq (t_f + 1.7t_a^{(m)})k_{2m}.$$

Thus, $t_{2m} \leq t_m$ implies that we must have

$$\frac{k_{2m}}{k_m} \leq \frac{t_f + t_a^{(m)}}{t_f + 1.7t_a^{(m)}} \equiv \frac{1 + \nu_m}{1.7 + \nu_m},$$

where ν_m is the ratio $t_f/t_a^{(m)}$. This shows that for $v_5 \leq 2.5$ we have $t_{10} \leq t_5$ only if we get a reduction of 18% in the number of function and gradient evaluations. The results in Table

6.3 confirm this analysis. A similar analysis holds if we compare t_{15} with t_5. In this case $t_a^{(3m)} \geq 2.5 t_a^{(m)}$ for $m \geq 5$, and thus $t_{15} \leq t_5$ for any function with $v_5 \leq 2.5$ only if we get a reduction of 30% in the number of function and gradient evaluations. Once again, the results in Table 6.3 confirm this analysis.

The above analysis shows that if the cost of the function and gradient evaluation is moderate relative to the cost of the arithmetic in the algorithm, then it probably does not pay to increase the number of vectors saved. If the cost of the function and gradient evaluation increases, then it does pay to increase m. However, then the vmlm code requires storage comparable with Newton's method. Moreover, for problems with expensive function and gradient evaluations the cost of the overall algorithm will be determined by the number of function and gradient evaluations, and as we have already seen, Newton's method is likely to require a much smaller number of function and gradient evaluations.

Of course, there is no guarantee that increasing m leads to a reduction in the number of function and gradient evaluations. This can be seen in Table 6.3 by comparing the results of $m = 5$ with those for $m = 20$.

7 Comparison of NMTRS with LANCELOT

We have also tested the nmtrs code on a set of test problems from the CUTE collection [2], and compared the results with those obtained with sbmin [7]. All sbmin runs were performed with the default options—exact second derivatives and a bandsolver preconditioned conjugate gradient solver were used. In these test we used

$$\|\nabla f(x)\| \leq \tau, \qquad \tau = 10^{-5}$$

as the termination test. We only present results for problems on which sbmin took at least 100 function evaluations; we did run tests on a large number of test problems from CUTE, but we feel that useful comparisons can only be made on difficult problems.

The results in Table 7.1 indicate that, with the exception of the GENROSE problem, nmtrs requires less function and gradient evaluations than sbmin. In terms of computing times, nmtrs is in general more efficient than sbmin. We observe that eventhough nmtrs requires about half the number of function and gradient evaluations of that required by sbmin on the SINQUAD problem, its computing time is approximately four times worse. This can be explained by noting that the first column of the Hessian matrix of the SINQUAD problem is dense. As a result, the incomplete Cholesky factorization requires an $O(n^2)$ operations, which causes each iteration of nmtrs to be quite expensive. The solve this problem, we reordered the SINQUAD Hessian matrix by using the reverse Cuthill-Mckee ordering (rcm). The new test result is given in Table 7.2. Notice that the use of rcm has led to a much better performance of nmtrs. This is possible because the rcm ordering produces an entirely different preconditioner, which could be more effective than that obtained without

Table 7.1: Performance of NMTRS with LANCELOT on the CUTE test problems

		NMTRS				LANCELOT			
Problem	n	`iters`	`ngev`	`ncg`	`time`	`iters`	`ngev`	`ncg`	`time`
BROYDN7D	1000	86	131	269	9.5	126	100	134	9.2
GENROSE	500	522	820	1392	15.9	585	500	593	15.4
LMINSURF	900	86	122	394	12.0	294	230	557	27.5
NLMSURF	900	366	533	2032	50.5	864	741	1816	85.4
NONMSQRT	529	280	375	441	357.7	-	-	-	‡
SINQUAD	1000	58	72	206	56.8	131	112	341	13.7

‡Time limit of one hour reached.

Table 7.2: Performance of NMTRS using reverse Cuthill-Mckee ordering

Problem	n	`iters`	`nfev`	`ncg`	`time`
SINQUAD	1000	11	12	33	4.5

ordering, as in the SINQUAD case. The sparsity structure of the remaining problems is tridiagonal for the GENROSE problem, banded for the LMINSURF and NLMSURF problems, and block diagonal for the NONMSQRT problem. Consequently, the rcm ordering is not effective on such problems. One possible reason for the failure of sbmin on the NONMSQRT problem is that the default semi-bandwidth is not large enough to produce an effective band preconditioner for the SBMIN conjugate gradient solver.

In summary, the comparison study of nmtrs with sbmin appears to indicate that nmtrs is very competitive with sbmin in function and gradient evaluations, and in computing time. We believe that nmtrs is likely to outperform sbmin on general unstructured problems. In order to firmly establish this conclusion, additional testing is required.

8 Concluding Remarks

The test results in Table 6.1 show that Newton's method is more robust than the VMLM algorithm. Newton's method solved all the test problems whereas the LMVM method failed to solve the GL1 problem for all sizes of n and the GL2 problem for sizes of $n \geq 10,000$. Note: I want to work more on this aspect of robustness.

We have shown that the NMTRS code is generally more efficient than the VMLM code in terms of function and gradient evaluations, and in terms of total computing time. These conclusions were drawn based on the MINPACK-2 large-scale problems. For these problems, the cost of evaluating the function and gradient is moderate. For many applications we expect the cost of the function and gradient evaluation to dominate, and for these problems the improvements over the VMLM code are likely to be higher.

A possible advantage of the limited memry algorithms is that they require a small amount of storage. Investigate this issue, and obtain the cross-over points.

The advantage of limited memory codes is that they are easy to use because they only require the user to provide the function and gradient evaluation codes, while the Newton code requires more. And now a plug for the environments work ...

Of course, for many problems we can expect a linear growth with n if the function and gradient evaluations are expensive enough. Expand on this.

References

[1] B. M. Averick, R. G. Carter, J. J. Moré, and G.-L. Xue, *The MINPACK-2 test problem collection*, Preprint MCS-P153-0694, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[2] I. Bongartz, A. R. Conn, N. I. M. Gould, and P. L. Toint, CUTE*: Constrained and Unconstrained Testing Environment*, ACM Trans. Math. Software, 21 (1995), pp. 123–160.

[3] A. Bouaricha, *STENMIN: A software package for large, sparse unconstrained optimization using tensor methods*, Preprint MCS-P451-0794, Argonne National Laboratory, Argonne, Illinois, 1994.

[4] ——, *Tensor methods for large, sparse unconstrainted optimization using tensor methods*, Preprint MCS-P452-0794, Argonne National Laboratory, Argonne, Illinois, 1994.

[5] A. R. Conn, N. I. M. Gould, M. Lescrenier, and P. L. Toint, *Performance of a multifrontal scheme for partially separable optimization*, Report 88-4, Namur University, Namur, Belgium, 1988.

[6] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, Math. Comp., 50 (1988), pp. 399–430.

[7] ——, *LANCELOT*, Springer Series in Computational Mathematics, Springer-Verlag, 1992.

[8] D. M. Gay, *Subroutines for unconstrained minimization using a model/trust region approach*, ACM Trans. Math. Software, 9 (1983), pp. 503–524.

[9] J. C. Gilbert and J. Nocedal, *Global convergence properties of conjugate gradient methods for optimization*, SIAM J. Optimization, 2 (1992), pp. 21–42.

[10] I. Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.

[11] ——, *Modified incomplete Cholesky (MIC) methods*, in Preconditioning Methods: Theory and Applications, D. Evans, ed., Gordon and Breach, 1983, pp. 265–293.

[12] ——, *A class of preconditioned conjugate gradient methods applied to the finite element equations*, in Preconditioning Conjugate Gradient Methods, O. Axelsson and L. Y. Koltilina, eds., Springer-Verlag, 1990, pp. 44–57.

[13] W. Hackbusch, *Iterative Solution of Large Sparse Systems of Equations*, Applied Mathematical Sciences 95, Springer-Verlag, 1994.

[14] M. R. Hestenes and E. Steifel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.

[15] M. T. Jones and P. E. Plassmann, *An improved incomplete Cholesky factorization*, ACM Trans. Math. Software, 21 (1995), pp. 5–17.

[16] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Programming, 45 (1989), pp. 503–528.

[17] T. A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 307–327.

[18] J. A. Meijerink and H. A. Van der Vorst, *An iterative solution method for linear equations systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[19] N. Munksgaard, *Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients*, ACM Trans. Math. Software, 6 (1980), pp. 206–219.

[20] S. G. Nash, *Newton-like methods minimization via the Lanczos method*, SIAM J. Numer. Anal., 21 (1984), pp. 770–788.

[21] S. G. Nash and J. Nocedal, *A numerical study of the limited memory BFGS method and the truncated Newton method for large scale optimization*, SIAM J. Optimization, 1 (1991), pp. 358–372.

[22] M. Papadrakakis and G. Pantazopoulos, *A survey of quasi-Newton methods with reduced storage*, Internat. J. Numer. Methods Engrg., 36 (1993), pp. 1573–1596.

[23] K. H. Phua, *An evaluation of nonlinear optimization codes on supercomputers*, International Journal of High Speed Computing, 3 (1991), pp. 199–213.

[24] T. Schlick and A. Fogelson, *TNPACK – A truncated Newton minimization package for large-scale problems: I. Algorithms and usage*, ACM Trans. Math. Software, 18 (1992), pp. 46–70.

[25] ——, *TNPACK – A truncated Newton minimization package for large-scale problems: II. Implementations examples*, ACM Trans. Math. Software, 18 (1992), pp. 71–111.

[26] T. Steihaug, *The conjugate gradient method and trust regions in large scale optimization*, SIAM J. Numer. Anal., 20 (1983), pp. 626–637.

[27] H. A. Van der Vorst, *The convergence behavior of the preconditioned CG and CG-S in the presence of rounding errors*, in Preconditioning Conjugate Gradient Methods, L. Y. K. O. Axelsson, ed., Springer-Verlag, 1990, pp. 126–136.

[28] X. Zou, I. M. Navon, M. Berger, K. H. Phua, T. Schlick, and F. X. Le Dimet, *Numerical experience with limited-memory quasi-Newton and truncated Newton methods*, SIAM J. Optimization, 3 (1993), pp. 582–608.