SIAM J. Optim.
Vol. 23, No. 4, pp. 2150–2168

THE LIMITED MEMORY CONJUGATE GRADIENT METHOD*

WILLIAM W. HAGER† AND HONGCHAO ZHANG‡

Abstract. In theory, the successive gradients generated by the conjugate gradient method applied to a quadratic should be orthogonal. However, for some ill-conditioned problems, orthogonality is quickly lost due to rounding errors, and convergence is much slower than expected. A limited memory version of the nonlinear conjugate gradient method is developed. The memory is used to both detect the loss of orthogonality and to restore orthogonality. An implementation of the algorithm is presented based on the CG_DESCENT nonlinear conjugate gradient method. Limited memory CG_DESCENT (L-CG_DESCENT) possesses a global convergence property similar to that of the memoryless algorithm but has much better practical performance. Numerical comparisons to the limited memory BFGS method (L-BFGS) are given using the CUTEr test problems.

Key words. nonlinear conjugate gradients, CG_DESCENT, unconstrained optimization, limited memory, BFGS, limited memory BFGS, L-BFGS, reduced Hessian method, L-RHR, adaptive method

AMS subject classifications. 90C06, 90C26, 65Y20

DOI. 10.1137/120898097

1. Introduction.

A new implementation of the nonlinear conjugate gradient method is developed for the unconstrained optimization problem

$$\min\{f(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^n\}, \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. The nonlinear conjugate gradient method is very effective for a wide range of unconstrained optimization problems with first-order derivatives available. However, in some ill-conditioned cases, the convergence can be extremely slow, even for very small problems. As an illustration, let us consider the CUTEr [3] test problem PALMER1C, a positive definite quadratic optimization problem of dimension 8 with a condition number around 10^{12}; the eigenvalues range from 0.0002 up to 2×10^8. In theory, the conjugate gradient method should reach the minimum in 8 iterations. In actuality, with an exact line search, it takes the PRP+ conjugate gradient method [9, 23] 26,563 iterations to reduce the Euclidean norm of the gradient to 10^{-6}. Figure 1.1 plots the base 10 logarithm of the Euclidean norm of the gradient, as a function of the iteration number k. In contrast, with an exact line search and with memory of 8, the unscaled limited memory BFGS algorithm (L-BFGS) [16, 20] is able to reduce the Euclidean norm of the gradient beneath 10^{-6} in 16 iterations.

In theory, for this quadratic test problem, the conjugate gradient method and L-BFGS should yield exactly the same iterates. In Table 1.1 we give the Euclidean norm

*Received by the editors November 7, 2012; accepted for publication (in revised form) August 19, 2013; published electronically November 5, 2013. The authors gratefully acknowledge support by the National Science Foundation under grant 1016204, by the Office of Naval Research under grant N00014-11-1-0068, and by the Defense Advanced Research Project Agency under contract HR0011-12-C-0011. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Approved for public release, distribution unlimited.

http://www.siam.org/journals/siopt/23-4/89809.html

†Department of Mathematics, University of Florida, Gainesville, FL 32611-8105 (hager@math.ufl.edu, http://www.math.ufl.edu/∼hager).

‡Department of Mathematics, Louisiana State University, Baton Rouge, LA 70803-4918 (hozhang@math.lsu.edu, http://www.math.lsu.edu/∼hozhang).

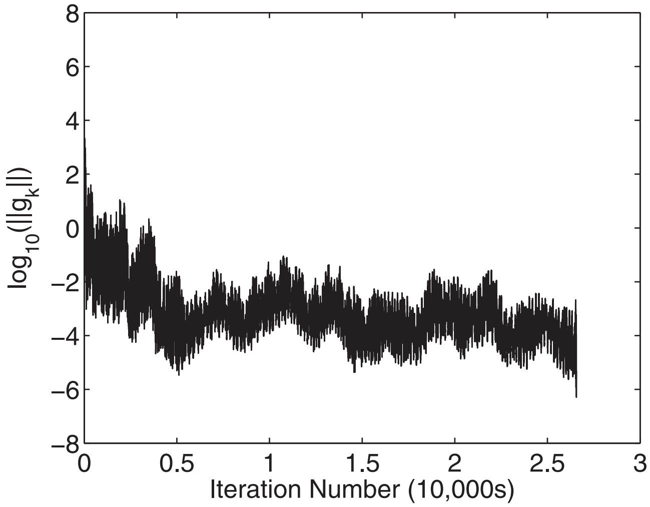

FIG. 1.1. *A plot of* $\log_{10}(\|\mathbf{g}_k\|)$ *versus the iteration number* k *for the test problem PALMER1C and the PRP+ conjugate gradient algorithm.*

TABLE 1.1
Comparison between L-BFGS and the conjugate gradient method for test problem PALMER1C.

Iteration	Euclidean norm of gradient		distance$\{\mathbf{g}_k, \mathcal{S}_k\}/\|\mathbf{g}_k\|$	
Number	L-BFGS	CG	L-BFGS	CG
0	1.236601e+04	1.236601e+04	1.000000e+00	1.000000e+00
1	5.963288e+02	5.963288e+02	1.000000e+00	1.000000e+00
2	7.407602e+02	7.407604e+02	1.000000e+00	1.000000e+00
3	1.522621e+02	4.295004e+03	1.000000e+00	1.000000e+00
4	1.854969e+01	1.522737e+02	1.000000e+00	6.071837e-02
5	8.570299e-01	1.855053e+01	1.000000e+00	2.970627e-09
6	1.916191e-02	2.735232e+05	1.000000e+00	1.547388e-03
7	5.895725e-01	5.585652e+03	9.999946e-01	1.291663e-06
8	5.077823e-02	8.561509e-01	9.959666e-01	4.454090e-08
9	1.578722e-02	4.375147e+01	8.544348e-01	9.995882e-01
10	1.229866e-02	9.167281e+02	7.709504e-01	3.273471e-07
11	8.741490e-03	6.924029e+01	6.475927e-01	5.993251e-06
12	1.179530e-03	2.844638e+02	9.789403e-01	3.497028e-05
13	1.704901e-05	9.924564e-01	9.999620e-01	4.165132e-08
14	4.185954e-06	7.273669e+02	9.999951e-01	3.995330e-04
15	1.096675e-08	2.769453e+01	1.000000e+00	1.225716e-05

of the gradient versus the iteration number for these two algorithms. In iterations 0 and 1, the errors are identical, while at iteration 2, the errors agree to at least 6 significant digits. At iteration 3, the errors differ by more than a factor of 10, and by iteration 6, the errors differ by more than a factor of 10^7. The sharp contrast between the theoretically predicted performance and the observed performance is due to the propagation of numerical errors in inexact 64-bit (double precision) floating point arithmetic.

In theory, for this quadratic test problem, the gradient at each iterate of either the conjugate gradient method or L-BFGS should be orthogonal to the space spanned by the previous search directions. Table 1.1 also gives the distance between $\mathbf{g}_k/\|\mathbf{g}_k\|$ and the space $\mathcal{S}_k$ spanned by the previous search directions up to a maximum of 6 directions. If $\mathbf{g}_k$ is orthogonal to $\mathcal{S}_k$, then this distance is 1, while if $\mathbf{g}_k \in \mathcal{S}_k$, then this distance is 0. Observe that L-BFGS preserves the orthogonality of $\mathbf{g}_k$ to $\mathcal{S}_k$, while the conjugate gradient method loses orthogonality at about the same time that the iterate error grows substantially. Moreover, the gradients in the conjugate gradient method not only lose orthogonality, but by iteration 5, the gradient essentially lies in the space spanned by the first 5 search directions.

The performance of the conjugate gradient method depends both on the problem conditioning and on the propagation of arithmetic errors in finite precision arithmetic. If the problem is quadratic, then there is an easy way to correct for the loss of orthogonality associated with the propagation of arithmetic errors. For a quadratic optimization problem of the form

$$\min \frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} - \mathbf{b}^{\mathsf{T}}\mathbf{x},$$

where $\mathbf{A}$ is a symmetric, positive definite matrix, the following code corresponds to an implementation of the Fletcher–Reeves [8] conjugate gradient method.

```
Fletcher–Reeves conjugate gradient method for a quadratic.
$\mathbf{g}_0 = \mathbf{A}\mathbf{x}_0 - \mathbf{b}, \quad \mathbf{d}_0 = -\mathbf{g}_0$
for $k = 0, 1, \ldots$
      $\mathbf{p}_k = \mathbf{A}\mathbf{d}_k$
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{d}_k, \quad \alpha_k = \|\mathbf{g}_k\|^2/\mathbf{d}_k^{\mathsf{T}}\mathbf{p}_k$
    $\mathbf{g}_{k+1} = \mathbf{g}_k + \alpha_k\mathbf{p}_k$
    $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k\mathbf{d}_k, \quad \beta_k = \|\mathbf{g}_{k+1}\|^2/\|\mathbf{g}_k\|^2$
end
```

When this code is applied to PALMER1C, the norm of the gradient is reduced below 10^{-9} by iteration 39, and below 10^{-12} by iteration 54.

This formulation of the conjugate gradient only applies to a quadratic since the gradient is updated by the formula $\mathbf{g}_{k+1} = \mathbf{g}_k + \alpha_k\mathbf{A}\mathbf{d}_k$. For a general nonlinear function, we treat the gradient as a black box with input $\mathbf{x}$ and output $\nabla f(\mathbf{x})$. If the gradient update was replaced by the gradient formula $\mathbf{g}_{k+1} = \mathbf{A}\mathbf{x}_{k+1} - \mathbf{b}$ in the test problem PALMER1C, then for double precision arithmetic, the conjugate gradient algorithm cannot reduce $\|\mathbf{g}_k\|$ below 10^{-8}.

In this paper, we develop a limited memory conjugate gradient method that corrects for the loss of orthogonality that can occur in ill-conditioned optimization problems. In our method, we check the distance between the current gradient and the space $\mathcal{S}_k$ spanned by the recent prior search directions. When the distance becomes sufficiently small, the orthogonality property has been lost, and in this case, we optimize the objective function over $\mathcal{S}_k$ until achieving a gradient that is approximately orthogonal to $\mathcal{S}_k$. This approximate orthogonality condition is eventually fulfilled by the first-order optimality conditions for a local minimizer in the subspace. The algorithm continues to operate in this same way: We apply the conjugate gradient iteration until the distance between the current gradient and $\mathcal{S}_k$ becomes sufficiently small, and then we solve a subspace problem to obtain an iterate for which the gradient is approximately orthogonal to $\mathcal{S}_k$.

Our limited memory algorithm has connections with both L-BFGS of Nocedal [20] and Liu and Nocedal [16], and with the reduced Hessian method of Gill and Leonard [10, 11]. Unlike either of these limited memory approaches, we do not always use the memory to construct the new search direction. The memory is used to monitor the orthogonality of the search directions; and when orthogonality is lost, the memory is used to generate a new orthogonal search direction. Our rational for not using the memory to generate the current search when orthogonality holds is that conjugate gradients can be very efficient and the benefit from using old search directions in the computation of the current search direction can be small when the successive search directions are locally orthogonal.

Our limited memory conjugate gradient algorithm is related to the reduced Hessian algorithm of Gill and Leonard in that both algorithms use the recent search directions to build the memory, not the recent gradients. On the other hand, our algorithm differs from the reduced Hessian method in the way that the memory enters into the computation of the search direction. When our algorithm detects a loss of orthogonality, it operates in an affine space associated with the memory until orthogonality is restored. In contrast, the reduced Hessian algorithm uses the memory in each iteration to update the search direction.

In the L-BFGS method, the formula for the new search direction is expressed in terms of both the recent gradients and the recent search directions. In our limited memory conjugate gradient algorithm, we only retain the recent search directions, not the gradients so the storage associated with the memory is cut in half. On the other hand, when the loss of orthogonality is detected, we utilize an L-BFGS iteration in the subspace in order to generate a gradient orthogonal to the subspace. And when the iterate leaves the subspace, we exploit an L-BFGS-based preconditioner. In our implementation of the subspace L-BFGS method, we save both the subspace direction and the subspace gradient vectors. But since the dimension of the subspace problem is small, the L-BFGS memory requirements in this subspace context are insignificant. Moreover, L-BFGS usually solves the subspace problem quickly due to its small dimension.

The paper is organized as follows. In section 2, we present the preconditioned version of CG_DESCENT. Section 3 discusses the composition of subspace and the criteria for switching between the subspace and the full-space conjugate gradient algorithms. Section 4 presents a quasi-Newton based preconditioner that can be used when returning to the full space after solving the subspace problem. Section 5 summarizes the three components of the limited memory conjugate gradient algorithm: the standard conjugate gradient iteration, the subspace iteration, and the preconditioner when leaving the subspace. Section 6 provides a convergence analysis for the limited memory algorithm. Section 7 discusses the implicit implementation of the subspace techniques, while section 8 gives numerical comparisons with L-BFGS and with (memoryless) CG_DESCENT 5.3 using the unconstrained CUTEr test problems [3].

Notation. $\nabla f(\mathbf{x})$ denotes the gradient of f, a row vector. The gradient of $f(\mathbf{x})$, arranged as a column vector, is $\mathbf{g}(\mathbf{x})$. The subscript k often represents the iteration number in an algorithm; for example, $\mathbf{x}_k$ is the kth $\mathbf{x}$ iterate while $\mathbf{g}_k$ stands for $\mathbf{g}(\mathbf{x}_k)$. $\|\cdot\|$ denotes the Euclidean norm. If $\mathbf{P} \in \mathbb{R}^{n\times n}$, then $\mathbf{P}^{-1}$ denotes the pseudoinverse of $\mathbf{P}$. If $\mathbf{P}$ is invertible, then $\mathbf{P}^{-1}$ is the ordinary inverse. If $\mathbf{x} \in \mathbb{R}^n$ and $\mathcal{S} \subset \mathbb{R}^n$, then

$$\text{dist}\{\mathbf{x}, \mathcal{S}\} = \inf\{\|\mathbf{y} - \mathbf{x}\| : \mathbf{y} \in \mathcal{S}\}.$$

2. Preconditioned CG_DESCENT.

The development of our limited memory conjugate gradient algorithm will be done in the context of the CG_DESCENT nonlinear conjugate gradient algorithm [13, 14, 15]. In this algorithm, the search directions are updated by the formula

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k, \tag{2.1}$$

where

$$\beta_k = \frac{\mathbf{y}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} - \theta_k \frac{\mathbf{y}_k^{\mathsf{T}}\mathbf{y}_k}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}. \tag{2.2}$$

Here $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$ and $\theta_k > 1/4$ is a parameter associated with the CG_DESCENT family. In the first papers [13, 14] analyzing CG_DESCENT, θ_k was 2. Our limited memory conjugate gradient algorithm uses a preconditioned version of (2.1)–(2.2). The idea behind preconditioning is to make a change of variables $\mathbf{x} = \mathbf{U}\mathbf{y}$, where $\mathbf{U} \in \mathbb{R}^{n\times n}$ is nonsingular, in order to improve the condition number of the objective function. The Hessian of $f(\mathbf{U}\mathbf{y})$ is

$$\nabla_y^2 f(\mathbf{U}\mathbf{y}) = \mathbf{U}^{\mathsf{T}}\nabla^2 f(\mathbf{x})\mathbf{U}.$$

In a perfectly conditioned problem, the eigenvalues of the Hessian are all the same. Hence, the goal in preconditioning is to choose $\mathbf{U}$ so that the eigenvalues of $\mathbf{U}^{\mathsf{T}}\nabla^2 f(\mathbf{x})\mathbf{U}$ are roughly the same. Since $\mathbf{U}^{\mathsf{T}}\nabla^2 f(\mathbf{x})\mathbf{U}$ is similar to $\nabla^2 f(\mathbf{x})\mathbf{U}\mathbf{U}^{\mathsf{T}}$, the product $\mathbf{U}\mathbf{U}^{\mathsf{T}}$ is usually chosen to approximate the inverse Hessian $\nabla^2 f(\mathbf{x})^{-1}$.

In the $\mathbf{y}$ variable, the conjugate gradient algorithm takes the form

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \widehat{\alpha}_k\widehat{\mathbf{d}}_k, \tag{2.3}$$

$$\widehat{\mathbf{d}}_{k+1} = -\widehat{\mathbf{g}}_{k+1} + \widehat{\beta}_k\widehat{\mathbf{d}}_k, \quad \widehat{\mathbf{d}}_0 = -\widehat{\mathbf{g}}_0, \tag{2.4}$$

where $\widehat{\alpha}_k$ is the stepsize, which is often computed to satisfy a line search condition such as the Wolfe conditions [27, 28]. Here the hats remind us that all the derivatives are computed with respect to $\mathbf{y}$. In practice, it is more convenient to perform the equivalent iteration in the original $\mathbf{x}$ variable. Multiplying (2.3) and (2.4) by $\mathbf{U}$ and substituting $\widehat{\mathbf{g}}_k = \mathbf{U}^{\mathsf{T}}\mathbf{g}_k$ and $\mathbf{x} = \mathbf{U}\mathbf{y}$, we obtain

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \widehat{\alpha}_k\mathbf{d}_k, \tag{2.5}$$

$$\mathbf{d}_{k+1} = -\mathbf{U}\mathbf{U}^{\mathsf{T}}\mathbf{g}_{k+1} + \widehat{\beta}_k\mathbf{d}_k, \quad \mathbf{d}_0 = -\mathbf{U}\mathbf{U}^{\mathsf{T}}\mathbf{g}_0. \tag{2.6}$$

The product $\mathbf{P} = \mathbf{U}\mathbf{U}^{\mathsf{T}}$ is usually called the preconditioner. This matrix also enters into the formula for $\widehat{\beta}_k$; for example, $\widehat{\mathbf{y}}_k^{\mathsf{T}}\widehat{\mathbf{g}}_{k+1} = \mathbf{y}_k^{\mathsf{T}}\mathbf{P}\mathbf{g}_{k+1}$ since $\widehat{\mathbf{y}}_k = \mathbf{U}^{\mathsf{T}}\mathbf{y}_k$ and $\widehat{\mathbf{g}}_{k+1} = \mathbf{U}^{\mathsf{T}}\mathbf{g}_{k+1}$. On the other hand, $\widehat{\mathbf{d}}_k^{\mathsf{T}}\widehat{\mathbf{y}}_k = \mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k$, so $\mathbf{P}$ only appears in some of terms forming $\widehat{\beta}_k$. For a Wolfe line search, we have $\widehat{\alpha}_k = \alpha_k$.

In our preconditioned version of CG_DESCENT, we allow the preconditioner to change in each iteration. If $\mathbf{P}_k$ denotes a symmetric, positive semidefinite preconditioner, then the search directions for preconditioned CG_DESCENT are updated by the formula

$$\mathbf{d}_{k+1} = -\mathbf{P}_k\mathbf{g}_{k+1} + \beta_k\mathbf{d}_k, \tag{2.7}$$

where

$$\beta_k = \frac{\mathbf{y}_k^{\mathsf{T}}\mathbf{P}_k\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} - \theta_k\frac{\mathbf{y}_k^{\mathsf{T}}\mathbf{P}_k\mathbf{y}_k}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}. \tag{2.8}$$

Note that $\mathbf{P}_k = \mathbf{I}$ corresponds to the original update formula (2.1)–(2.2). For the CG_DESCENT family associated with (2.8), it follows from [15, eq. (7.3)] that the search directions satisfy the sufficient descent condition

$$\mathbf{d}_{k+1}^{\mathsf{T}}\mathbf{g}_{k+1} \le -\left(1 - \frac{1}{4\theta_k}\right)\mathbf{g}_{k+1}^{\mathsf{T}}\mathbf{P}_k\mathbf{g}_{k+1}. \tag{2.9}$$

To ensure global convergence, we must truncate β_k when it becomes too small. We have found that a lower bound β_k^+ of the following form both guarantees global convergence and works well in practice:

$$\beta_k^+ = \max\{\beta_k, \eta_k\}, \quad \eta_k = \eta\left(\frac{\mathbf{d}_k^\mathsf{T}\mathbf{g}_k}{\mathbf{d}_k^\mathsf{T}\mathbf{P}_k^{-1}\mathbf{d}_k}\right), \tag{2.10}$$

where η is a positive parameter ($= 0.4$ in the numerical experiments) and $\mathbf{P}_k^{-1}$ denotes the pseudoinverse of $\mathbf{P}_k$. In our original CG_DESCENT paper [13], the proposed truncation formula was not scale invariant in the sense that β_k^+ changed if either the objective function was multiplied by a positive constant or the independent variable $\mathbf{x}$ was multiplied by a nonzero scalar. The truncation formula (2.10), on the other hand, is scale invariant. Unlike the PRP+ conjugate gradient algorithm [9, 23], $\eta_k < 0$ in (2.10) and $\beta_k < 0$ can be accepted. By permitting $\beta_k < 0$, there are more cases where $\beta_k^+ = \beta_k$, which implies that there are fewer cases where β_k is truncated to ensure convergence. A related truncation strategy is given in [4] in which the $\mathbf{d}_k^\mathsf{T}\mathbf{g}_k$ term in the numerator of η_k is replaced by $\mathbf{d}_k^\mathsf{T}\mathbf{g}_{k+1}$. With an exact line search, $\mathbf{d}_k^\mathsf{T}\mathbf{g}_{k+1} = 0$, and the resulting truncation strategy becomes the PRP+ truncation analyzed by Gilbert and Nocedal. In contrast, the truncation (2.10) always allows $\beta_k^+ < 0$ by (2.9). In numerical experiments with the CUTEr test problems, truncation (2.10) gave better performance than the scheme suggested by Dai and Kou [4], and hence, (2.10) was adopted in Version 4.0 of CG_DESCENT in 2011.

Taking into account the lower bound, the preconditioned search direction is

$$\mathbf{d}_{k+1} = -\mathbf{P}_k\mathbf{g}_{k+1} + \beta_k^+\mathbf{d}_k. \tag{2.11}$$

If $\theta_k = \theta > 1/4$ and the smallest and largest eigenvalues of $\mathbf{P}_k$ are uniformly bounded away from 0 and ∞, then the CG_DESCENT family has a global convergence property when implemented using the standard Wolfe line search. Dai and Yuan [5, 6] were the first to show that the standard Wolfe line search is sufficient for the global convergence of conjugate gradient methods.

In the preconditioned CG_DESCENT family, we now observe a connection between (2.7)–(2.8) with $\theta_k = 1$ and any quasi-Newton scheme.

PROPOSITION 2.1. *Let $\mathbf{H}_{k+1}$ denote a symmetric, positive semidefinite quasi-Newton approximation to the inverse Hessian at $\mathbf{x}_{k+1}$ that satisfies the standard secant condition:*

$$\mathbf{H}_{k+1}\mathbf{y}_k = \mathbf{s}_k := \mathbf{x}_{k+1} - \mathbf{x}_k = \alpha_k\mathbf{d}_k, \tag{2.12}$$

where $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$ is the gradient change and α_k is the stepsize in the direction $\mathbf{d}_k$ at iteration k. In the preconditioned CG_DESCENT scheme using $\mathbf{P}_k = \mathbf{H}_{k+1}$ and $\theta_k = 1$, we have

$$\beta_k = \beta_k^+ = 0.$$

Proof. Utilizing (2.8), (2.12), and the fact that $\theta_k = 1$, it follows that

$$\beta_k = \frac{\alpha_k\mathbf{d}_k^\mathsf{T}\mathbf{g}_{k+1}}{\mathbf{d}_k^\mathsf{T}\mathbf{y}_k} - \frac{\alpha_k\mathbf{d}_k^\mathsf{T}\mathbf{y}_k}{\mathbf{d}_k^\mathsf{T}\mathbf{y}_k}\frac{\mathbf{d}_k^\mathsf{T}\mathbf{g}_{k+1}}{\mathbf{d}_k^\mathsf{T}\mathbf{y}_k} = 0.$$

Moreover, by the sufficient descent condition (2.9), $\mathbf{d}_k\mathbf{g}_k \le 0$, and since $\mathbf{P}_k$ is symmetric and positive semidefinite, $\eta_k \le 0$. Hence, $\beta_k^+ = \beta_k = 0$. □

Proposition 2.1 implies that the CG_DESCENT scheme with $\theta_k = 1$ and a quasi-Newton preconditioner is equivalent to the quasi-Newton scheme itself.

Remark. In [4] Dai and Kou consider a family of conjugate gradient schemes depending on a parameter τ_k, where β_k is of the form

$$\beta_k = \frac{\mathbf{y}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} - \left(\tau_k + \frac{\|\mathbf{y}_k\|^2}{\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k} - \frac{\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k}{\|\mathbf{s}_k\|^2}\right)\frac{\mathbf{s}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}, \quad \mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k. \tag{2.13}$$

They obtain this formula by taking a multiple of the memoryless BFGS direction of Perry [22] and Shanno [24] and projecting into the manifold

$$\{-\mathbf{g}_{k+1} + s\mathbf{d}_k : s \in \mathbb{R}\}.$$

The parameter τ_k is a scaling parameter appearing in the memoryless BFGS scheme. When the scaling parameter is $\tau_k = \tau_k^B := \mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k/\|\mathbf{s}_k\|^2$, the formula (2.13) for β_k is the same as the CG_DESCENT formula associated with $\theta_k = 1$. Dai and Kou provide numerical experiments showing that $\tau_k = \tau_k^B$, or, equivalently, CG_DESCENT with $\theta_k = 1$, performed generally better than several other members of the family (2.13). Moreover, in Remark 2 of [4], the authors observe that for the range of scaling suggested by Oren in his thesis [21],

$$\tau_k = \nu\frac{\|\mathbf{y}_k\|^2}{\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k} + (1-\nu)\frac{\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k}{\|\mathbf{s}_k\|^2}, \quad \nu \in [0,1],$$

the corresponding range of θ_k in CG_DESCENT is

$$\theta_k \in \left[1, 2 - \frac{(\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k)^2}{\|\mathbf{s}_k\|^2\|\mathbf{y}_k\|^2}\right].$$

This suggests that an optimal range of θ_k in CG_DESCENT lies in the interval $[1, 2)$. Motivated by Dai and Kou's observations, Hager and Zhang evaluated performance profiles for the CUTEr test problems and for different constant values of θ_k, independent of k. It was observed that the best performance profile for CG_DESCENT was achieved by $\theta_k = 1$ for this test set. Hence, CG_DESCENT has used $\theta_k = 1$ by default since Version 4.0 in 2011.

3. Optimization in the subspace.

Let $m > 0$ denote the number of vectors in the memory of our limited memory conjugate gradient algorithm, and let $\mathcal{S}_k$ denote the subspace spanned by the previous m search directions:

$$\mathcal{S}_k = \operatorname{span}\{\mathbf{d}_{k-1}, \mathbf{d}_{k-2}, \ldots, \mathbf{d}_{k-m}\}.$$

If $\mathbf{g}_k$ is nearly contained in $\mathcal{S}_k$, then we feel that the algorithm has lost its orthogonality property and is returning to a previously explored subspace. In this case, we interrupt the conjugate gradient iterations and consider the problem

$$\min_{\mathbf{z}\in\mathcal{S}_k} f(\mathbf{x}_k + \mathbf{z}). \tag{3.1}$$

If $\mathbf{z}_k$ is a solution of this problem and $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{z}_k$, then by the first-order optimality conditions for (3.1), we have $\mathbf{d}^{\mathsf{T}}\mathbf{g}_{k+1} = \mathbf{0}$ for all $\mathbf{d} \in \mathcal{S}_k$. Hence, the solution to the subspace problem will lead us to an iterate with an improved function value and to a search direction that takes us out of the subspace.

We use two parameters η_0 and η_1 to implement the subspace process, where $0 < \eta_0 < \eta_1 < 1$. If the condition

$$\text{dist}\{\mathbf{g}_k, \mathcal{S}_k\} \le \eta_0 \|\mathbf{g}_k\| \tag{3.2}$$

is satisfied, then we switch to the subspace problem (3.1). We continue to perform iterations inside the subspace until the gradient becomes sufficiently orthogonal to the subspace to satisfy the condition

$$\text{dist}\{\mathbf{g}_{k+1}, \mathcal{S}_k\} \ge \eta_1 \|\mathbf{g}_{k+1}\|. \tag{3.3}$$

If $\mathbf{Z}$ is a matrix whose columns are an orthonormal basis for $\mathcal{S}_k$, then these two conditions can be expressed as

$$(1-\eta_0^2)\|\mathbf{g}_k\|^2 \le \|\mathbf{g}_k^{\mathsf{T}}\mathbf{Z}\|^2 \quad \text{and} \quad (1-\eta_1^2)\|\mathbf{g}_{k+1}\|^2 \ge \|\mathbf{g}_{k+1}^{\mathsf{T}}\mathbf{Z}\|^2. \tag{3.4}$$

The numerical results given in this paper are based on using a quasi-Newton method to solve the subspace problem. According to Proposition 2.1, if the conjugate gradient algorithm (2.7)–(2.8) with $\theta_k = 1$ is preconditioned by the Hessian approximation gotten from a quasi-Newton method, then $\beta_k^+ = 0$ and the resulting scheme is the quasi-Newton method itself. Consequently, we can view the quasi-Newton iteration applied to the subspace problem as a special case of CG_DESCENT with a preconditioner of the form

$$\mathbf{P}_k = \mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}},$$

where $\widehat{\mathbf{P}}_k = \widehat{\mathbf{H}}_{k+1}$ and $\widehat{\mathbf{H}}_{k+1}$ is the quasi-Newton matrix in the subspace. The search direction $\widehat{\mathbf{d}}_{k+1}$ in the subspace is given by $\widehat{\mathbf{d}}_{k+1} = -\widehat{\mathbf{H}}_{k+1}\widehat{\mathbf{g}}_{k+1}$.

4. A quasi-Newton based preconditioner when departing subspace.

In this section we present a preconditioner that can be used when terminating the subspace problem and returning to the full space. Again, let $\widehat{\mathbf{P}}_k$ denote the preconditioner in the subspace; we think of $\widehat{\mathbf{P}}_k$ as an approximation to the inverse Hessian in the subspace. If $\mathbf{Z}$ denotes a matrix whose columns are an orthonormal basis for the subspace $\mathcal{S}_k$, then we consider the following preconditioner for the conjugate gradient iteration (2.11):

$$\mathbf{P}_k = \mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}} + \sigma_k \overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^{\mathsf{T}}, \tag{4.1}$$

where $\overline{\mathbf{Z}}$ is a matrix whose columns are an orthonormal basis for the complement of $\mathcal{S}_k$, and $\sigma_k\mathbf{I}$ is the safe-guarded Barzilai-Borwein (BB) approximation [2] to the inverse Hessian given by

$$\sigma_k = \max\left\{\sigma_{\min}, \min\left[\sigma_{\max}, \frac{\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k}{\mathbf{y}_k^{\mathsf{T}}\mathbf{y}_k}\right]\right\}, \quad 0 < \sigma_{\min} \le \sigma_{\max} < \infty. \tag{4.2}$$

In other words, σ_k is the projection of $\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k/\mathbf{y}_k^{\mathsf{T}}\mathbf{y}_k$ onto the interval $[\sigma_{\min}, \sigma_{\max}]$. Since $\widehat{\mathbf{P}}_k$ is an approximation to the inverse Hessian in the subspace, we can think of $\mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}}$ as the analogous approximation to the full Hessian restricted to the subspace. Since there is no information concerning the Hessian outside the subspace, we use a BB approximation $\sigma_k\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^{\mathsf{T}}$ in the complement of $\mathcal{S}_k$. Since $\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^{\mathsf{T}} = \mathbf{I} - \mathbf{Z}\mathbf{Z}^{\mathsf{T}}$, the preconditioned search direction $\mathbf{d}_{k+1}$ in (2.11) can be expressed as follows:

$$\begin{aligned}
\mathbf{d}_{k+1} &= -\mathbf{P}_k\mathbf{g}_{k+1} + \beta_k^+\mathbf{d}_k \\
&= -\mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}}\mathbf{g}_{k+1} - \sigma_k(\mathbf{I} - \mathbf{Z}\mathbf{Z}^{\mathsf{T}})\mathbf{g}_{k+1} + \beta_k^+\mathbf{d}_k \\
&= -\mathbf{Z}(\widehat{\mathbf{P}}_k - \sigma_k\mathbf{I})\widehat{\mathbf{g}}_{k+1} - \sigma_k\mathbf{g}_{k+1} + \beta_k^+\mathbf{d}_k. \qquad (4.3)
\end{aligned}$$

Here $\widehat{\mathbf{g}}_{k+1} = \mathbf{Z}^{\mathsf{T}}\mathbf{g}_{k+1}$ is the gradient in the subspace. The first term in (4.3) is the subspace contribution, while the remaining terms are a scaled conjugate gradient direction.

In the case where $\theta_k = 1$ and $\widehat{\mathbf{P}}_k = \widehat{\mathbf{H}}_{k+1}$, a quasi-Newton matrix, we can express β_k in terms of easily computed subspace and full space quantities without explicitly forming either $\mathbf{Z}$ or $\overline{\mathbf{Z}}$. Since $\mathbf{P}_k$ in (4.1) is the sum of two terms, β_k in (2.8) is the sum of two terms, a subspace term containing $\mathbf{Z}\widehat{\mathbf{H}}_{k+1}\mathbf{Z}^{\mathsf{T}}$ and another term containing $\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^{\mathsf{T}}$. We first show that the subspace term vanishes when $\theta_k = 1$.

Since $\mathbf{s}_k \in \mathcal{S}_k$, we have

$$\mathbf{s}_k = \mathbf{Z}\mathbf{Z}^{\mathsf{T}}\mathbf{s}_k = \mathbf{Z}\widehat{\mathbf{s}}_k, \tag{4.4}$$

where $\widehat{\mathbf{s}}_k = \mathbf{Z}^{\mathsf{T}}\mathbf{s}_k$. By the quasi-Newton condition in the subspace, $\widehat{\mathbf{y}}_k^{\mathsf{T}}\widehat{\mathbf{H}}_{k+1} = \widehat{\mathbf{s}}_k^{\mathsf{T}}$. Consequently, we have

$$\mathbf{y}_k^{\mathsf{T}}\mathbf{Z}\mathbf{H}_{k+1}\mathbf{Z}^{\mathsf{T}}\mathbf{g}_{k+1} = \widehat{\mathbf{y}}_k^{\mathsf{T}}\mathbf{H}_{k+1}\mathbf{Z}^{\mathsf{T}}\mathbf{g}_{k+1} = \widehat{\mathbf{s}}_k^{\mathsf{T}}\mathbf{Z}^{\mathsf{T}}\mathbf{g}_{k+1} = \mathbf{s}_k^{\mathsf{T}}\mathbf{g}_{k+1}. \tag{4.5}$$

In a similar fashion,

$$\mathbf{y}_k^{\mathsf{T}}\mathbf{Z}\mathbf{H}_{k+1}\mathbf{Z}^{\mathsf{T}}\mathbf{y}_k = \mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k. \tag{4.6}$$

Taking $\theta_k = 1$, focusing on the $\mathbf{Z}\widehat{\mathbf{H}}_{k+1}\mathbf{Z}^{\mathsf{T}}$ terms in β_k, and exploiting the identities (4.5) and (4.6) yields

$$\begin{aligned}\frac{\mathbf{y}_k^{\mathsf{T}}\mathbf{Z}\mathbf{H}_{k+1}\mathbf{Z}^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} - \frac{\mathbf{y}_k^{\mathsf{T}}\mathbf{Z}\mathbf{H}_{k+1}\mathbf{Z}^{\mathsf{T}}\mathbf{y}_k}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} &= \frac{\mathbf{s}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} - \frac{\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} \\ &= \alpha_k\left(\frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} - \frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\right) = 0.\end{aligned}$$

Since the subspace term in β_k vanishes, we are left with the complementary term:

$$\begin{aligned}\beta_k &= \sigma_k\left(\frac{\mathbf{y}_k^{\mathsf{T}}\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} - \frac{\mathbf{y}_k^{\mathsf{T}}\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^{\mathsf{T}}\mathbf{y}_k}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\right) \\ &= \sigma_k\left(\frac{\mathbf{y}_k^{\mathsf{T}}(\mathbf{I}-\mathbf{Z}\mathbf{Z}^{\mathsf{T}})\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} - \frac{\mathbf{y}_k^{\mathsf{T}}(\mathbf{I}-\mathbf{Z}\mathbf{Z}^{\mathsf{T}})\mathbf{y}_k}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\right) \\ &= \sigma_k\left(\frac{\mathbf{y}_k^{\mathsf{T}}\mathbf{g}_{k+1} - \widehat{\mathbf{y}}_k^{\mathsf{T}}\widehat{\mathbf{g}}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k} - \left[\frac{\|\mathbf{y}_k\|^2 - \|\widehat{\mathbf{y}}_k\|^2}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\right]\frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_{k+1}}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\right). \qquad (4.7)\end{aligned}$$

Hence, β_k can be expressed in terms of easily computed quantities involving either vectors in the subspace or in the full space.

The expression (2.10) for β_k^+ also can be simplified. The inverse of the preconditioner $\mathbf{P}_k$ in (4.1) is

$$\mathbf{P}_k^{-1} = \mathbf{Z}\widehat{\mathbf{H}}_{k+1}^{-1}\mathbf{Z}^{\mathsf{T}} + \sigma_k^{-1}\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^{\mathsf{T}}.$$

Since $\mathbf{d}_k \in \mathcal{S}_k$, it follows that $\overline{\mathbf{Z}}^{\mathsf{T}}\mathbf{d}_k = \mathbf{0}$ and

$$\mathbf{d}_k^{\mathsf{T}}\mathbf{P}_k^{-1}\mathbf{d}_k = \mathbf{d}_k^{\mathsf{T}}(\mathbf{Z}\mathbf{H}_{k+1}^{-1}\mathbf{Z}^{\mathsf{T}})\mathbf{d}_k = \frac{\mathbf{s}_k^{\mathsf{T}}(\mathbf{Z}\mathbf{H}_{k+1}^{-1}\mathbf{Z}^{\mathsf{T}})\mathbf{s}_k}{\alpha_k^2} = \frac{\widehat{\mathbf{s}}_k^{\mathsf{T}}\mathbf{H}_{k+1}^{-1}\widehat{\mathbf{s}}_k}{\alpha_k^2} = \frac{\widehat{\mathbf{s}}_k^{\mathsf{T}}\widehat{\mathbf{y}}_k}{\alpha_k^2}. \tag{4.8}$$

We utilize (4.4) to obtain

$$\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k = (\mathbf{Z}\mathbf{Z}^{\mathsf{T}}\mathbf{s}_k)^{\mathsf{T}}\mathbf{y}_k = (\mathbf{Z}^{\mathsf{T}}\mathbf{s}_k)^{\mathsf{T}}\mathbf{Z}^{\mathsf{T}}\mathbf{y}_k = \widehat{\mathbf{s}}_k^{\mathsf{T}}\widehat{\mathbf{y}}_k. \tag{4.9}$$

Combining (4.8) and (4.9) yields

$$\mathbf{d}_k^{\mathsf{T}}\mathbf{P}_k^{-1}\mathbf{d}_k = \frac{\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k}{\alpha_k^2} = \frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}{\alpha_k}.$$

Hence, we have

$$\beta_k^+ = \max\{\beta_k, \eta_k\}, \quad \eta_k = \eta\left(\frac{\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_k}{\mathbf{d}_k^{\mathsf{T}}\mathbf{P}_k^{-1}\mathbf{d}_k}\right) = \eta\left(\frac{\mathbf{s}_k^{\mathsf{T}}\mathbf{g}_k}{\mathbf{d}_k^{\mathsf{T}}\mathbf{y}_k}\right), \tag{4.10}$$

where β_k is given in (4.7).

5. Overview of the limited memory algorithm.

Our proposed limited memory conjugate gradient algorithm has three parts:

1. Standard conjugate gradient iteration. Perform the conjugate gradient algorithm (2.11) with $\mathbf{P}_k = \mathbf{I}$ as long as $\text{dist}\{\mathbf{g}_k, \mathcal{S}_k\} > \eta_0\|\mathbf{g}_k\|$. When the subspace condition $\text{dist}\{\mathbf{g}_k, \mathcal{S}_k\} \le \eta_0\|\mathbf{g}_k\|$ is satisfied, branch to the subspace iteration.
2. Subspace iteration. Solve the subspace problem (3.1) by CG_DESCENT with preconditioner $\mathbf{P}_k = \mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}}$, where $\mathbf{Z}$ is a matrix whose columns are an orthonormal basis for the subspace $\mathcal{S}_k$ and $\widehat{\mathbf{P}}_k$ is a preconditioner in the subspace. Stop at the first iteration where $\text{dist}\{\mathbf{g}_k, \mathcal{S}_k\} \ge \eta_1\|\mathbf{g}_k\|$, and then branch to the preconditioning step.
3. Preconditioning step. When the subspace iteration terminates and we return to the full space standard conjugate gradient iteration, we have found that the convergence can be accelerated by performing a single preconditioned iteration. In the special case $\widehat{\mathbf{P}}_k = \widehat{\mathbf{H}}_{k+1}$, where $\widehat{\mathbf{H}}_{k+1}$ is a quasi-Newton matrix, an appropriate preconditioned step corresponds to the search direction (4.3), where σ_k is given by the BB formula (4.2), $\mathbf{Z}$ is a matrix whose columns are an orthonormal basis for the subspace $\mathcal{S}_k$, and β_k^+ is given by (4.10). After completing the preconditioning iteration, return to the standard conjugate gradient iteration (step 1).

Potentially three different preconditioners could arise during the limited memory conjugate gradient algorithm corresponding to the three parts of the algorithm:

1. $\mathbf{P}_k = \mathbf{I}$.
2. $\mathbf{P}_k = \mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}}$, where $\widehat{\mathbf{P}}_k$ is the subspace preconditioner and $\mathbf{Z}$ is a matrix whose columns are an orthonormal basis for the subspace $\mathcal{S}_k$.
3. $\mathbf{P}_k = \mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}} + \sigma_k\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^{\mathsf{T}}$, where $\overline{\mathbf{Z}}$ is a matrix whose columns are an orthonormal basis for the complement of $\mathcal{S}_k$ and $\sigma_k\mathbf{I}$ is the safe-guarded BB approximation [2] to the inverse Hessian given by (4.2).

6. Convergence analysis.

As mentioned in the introduction, it is well known [17] that the conjugate gradient algorithm with an exact line search should reach the minimum of a strongly convex quadratic in at most n iterations, and in each iteration, the gradient is orthogonal to the space spanned by the previous search directions. Hence, for an exact line search and for any value of the memory m, the subspace condition (3.2) is never satisfied in theory (since $\text{dist}\{\mathbf{g}_k, \mathcal{S}_k\} = 1$ and $\eta_0 < 1$). Consequently, the limited memory conjugate gradient algorithm of section 5 reduces to the standard linear conjugate gradient method. And hence it reaches the global minimum of a strongly convex quadratic in at most n iterations. We summarize this as follows.

Proposition 6.1. *If f is a strongly convex quadratic, then the limited memory conjugate gradient algorithm of section* 5 *reaches the global minimum in at most n iterations when implemented with an exact line search.*

Next, we consider the convergence of the preconditioned conjugate gradient scheme given by (2.8), (2.10), and (2.11) for a more general nonlinear function. In [13] we give a convergence result in the case that $\mathbf{P}_k = \mathbf{I}$ for each k. With small modifications in the assumptions and the analysis, we obtain the following result for the preconditioned algorithm.

Theorem 6.2. *Suppose that the preconditioned conjugate gradient algorithm given by* (2.8), (2.10), *and* (2.11) *satisfies the following conditions:*

(C1) $\theta_k = \theta > 1/4$, *where θ_k appears in* (2.8).

(C2) *The line search satisfies the standard Wolfe conditions, that is,*

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) - f(\mathbf{x}_k) \le \delta \alpha_k \mathbf{g}_k^{\mathsf{T}} \mathbf{d}_k \quad \text{and} \quad \mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{d}_k \ge \sigma \mathbf{g}_k^{\mathsf{T}} \mathbf{d}_k,$$

where $0 < \delta \le \sigma < 1$.

(C3) *The level set*

$$\mathcal{L} = \{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) \le f(\mathbf{x}_0)\}$$

is bounded, and ∇f is Lipschitz continuous on $\mathcal{L}$.

(C4) *The preconditioner $\mathbf{P}_k$ satisfies the conditions*

$$\|\mathbf{P}_k\| \le \gamma_0, \quad \mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{P}_k \mathbf{g}_{k+1} \ge \gamma_1 \|\mathbf{g}_{k+1}\|^2, \quad \text{and} \quad \mathbf{d}_k^{\mathsf{T}} \mathbf{P}_k^{-1} \mathbf{d}_k \ge \gamma_2 \|\mathbf{d}_k\|^2$$

for all k, where γ_0, γ_1, and γ_2 are positive constants.

Then either $\mathbf{g}_k = \mathbf{0}$ for some k, or

$$\liminf_{k\to\infty} \|\mathbf{g}_k\| = 0.$$

Proof. Conditions (C1)–(C3) appeared in the original convergence proof [13, Thm. 3.2] for the unconditioned algorithm. The modifications needed to account for the preconditioner $\mathbf{P}_k$ are relatively minor. In particular, the descent condition

$$\mathbf{d}_k^{\mathsf{T}} \mathbf{g}_k \le -\left(1 - \frac{1}{4\theta}\right) \|\mathbf{g}_k\|^2$$

is replaced by

$$\mathbf{d}_k^{\mathsf{T}} \mathbf{g}_k \le -\left(1 - \frac{1}{4\theta}\right) \mathbf{g}_k^{\mathsf{T}} \mathbf{P}_k \mathbf{g}_k \le -\gamma_1 \left(1 - \frac{1}{4\theta}\right) \|\mathbf{g}_k\|^2,$$

where γ_1 appears in (C4). The denominator $\mathbf{d}_k^{\mathsf{T}} \mathbf{P}_k^{-1} \mathbf{d}_k$ of η_k in (2.10) is bounded from below by $\gamma_2 \|\mathbf{d}_k\|^2$, which leads to the lower bound

$$\eta_k \ge \eta \left(\frac{\mathbf{d}_k^{\mathsf{T}} \mathbf{g}_k}{\gamma_2 \|\mathbf{d}_k\|^2}\right)$$

since $\mathbf{d}_k^{\mathsf{T}}\mathbf{g}_k \le 0$. Finally, during the proof of [13, Thm. 3.2], we bound $\|\mathbf{g}_k\|$ due to fact that the level set $\mathcal{L}$ is bounded and ∇f is Lipschitz continuous. For the preconditioned scheme, the product $\mathbf{P}_k\mathbf{g}_k$ plays the role of $\mathbf{g}_k$ and the analogous bound is $\|\mathbf{P}_k\mathbf{g}_k\| \le \|\mathbf{P}_k\|\|\mathbf{g}_k\| \le \gamma_0\|\mathbf{g}_k\|$ (by (C4)), where $\|\mathbf{g}_k\|$ is again bounded due to fact that the level set $\mathcal{L}$ is bounded and ∇f is Lipschitz continuous. □

Next, let us suppose that the CG_DESCENT algorithm is implemented using the framework of section 5, where $\mathbf{P}_k$ is expressed further in terms of a subspace matrix $\widehat{\mathbf{P}}_k$ and a matrix $\mathbf{Z}$ with orthonormal columns that forms a basis for the subspace $\mathcal{S}_k$: $\mathbf{P}_k = \mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}}$.

THEOREM 6.3. *Suppose that the preconditioned conjugate gradient algorithm given by* (2.8), (2.10), *and* (2.11) *satisfies* (C1)–(C3). *Moreover, suppose that the subspace preconditioner $\widehat{\mathbf{P}}_k$ has the following properties:*

($\widehat{\text{C}}$4) *There are positive constants $\widehat{\gamma}_0$, $\widehat{\gamma}_1$, and $\widehat{\gamma}_2$ such that for all k, we have*

$$\|\widehat{\mathbf{P}}_k\| \le \widehat{\gamma}_0, \quad \widehat{\mathbf{g}}_{k+1}^{\mathsf{T}}\widehat{\mathbf{P}}_k\widehat{\mathbf{g}}_{k+1} \ge \widehat{\gamma}_1\|\widehat{\mathbf{g}}_{k+1}\|^2, \quad \textit{and} \quad \widehat{\mathbf{d}}_k^{\mathsf{T}}\widehat{\mathbf{P}}_k^{-1}\widehat{\mathbf{d}}_k \ge \widehat{\gamma}_2\|\widehat{\mathbf{d}}_k\|^2.$$

Then either $\mathbf{g}_k = \mathbf{0}$ for some k, or

$$\liminf_{k\to\infty} \|\mathbf{g}_k\| = 0.$$

Proof. We show that condition (C4) of Theorem 6.2 is satisfied. According to section 5, there are three different choices for $\mathbf{P}_k$ to consider. If $\mathbf{P}_k = \mathbf{I}$, then (C4) holds trivially.

If $\mathbf{P}_k = \mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}}$, where $\widehat{\mathbf{P}}_k$ is the subspace preconditioner and $\mathbf{Z}$ is a matrix whose columns are an orthonormal basis for the subspace $\mathcal{S}_k$, then by ($\widehat{\text{C}}$4), we have

$$\|\mathbf{P}_k\| = \|\mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}}\| = \|\widehat{\mathbf{P}}_k\| \le \widehat{\gamma}_0,$$

and

$$\mathbf{g}_{k+1}^{\mathsf{T}}\mathbf{P}_k\mathbf{g}_{k+1} = \mathbf{g}_{k+1}^{\mathsf{T}}\mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}}\mathbf{g}_{k+1} = \widehat{\mathbf{g}}_{k+1}^{\mathsf{T}}\widehat{\mathbf{P}}_k\widehat{\mathbf{g}}_{k+1} \ge \widehat{\gamma}_1\|\widehat{\mathbf{g}}_{k+1}\|^2. \tag{6.1}$$

Moreover, since the search direction $\mathbf{d}_{k+1}$ is computed by the subspace iteration in section 5, it follows from (3.4) that

$$\|\widehat{\mathbf{g}}_{k+1}\|^2 \ge (1-\eta_1^2)\|\mathbf{g}_{k+1}\|^2. \tag{6.2}$$

We combine (6.1) and (6.2) to obtain

$$\mathbf{g}_{k+1}^{\mathsf{T}}\mathbf{P}_k\mathbf{g}_{k+1} \ge \widehat{\gamma}_1(1-\eta_1^2)\|\mathbf{g}_{k+1}\|^2.$$

Finally,

$$\mathbf{d}_k^{\mathsf{T}}\mathbf{P}_k^{-1}\mathbf{d}_k = \mathbf{d}_k^{\mathsf{T}}\mathbf{Z}\widehat{\mathbf{P}}_k^{-1}\mathbf{Z}^{\mathsf{T}}\mathbf{d}_k = \widehat{\mathbf{d}}_k^{\mathsf{T}}\widehat{\mathbf{P}}_k^{-1}\widehat{\mathbf{d}}_k \ge \widehat{\gamma}_2\|\widehat{\mathbf{d}}_k\|^2 = \widehat{\gamma}_2\|\mathbf{d}_k\|^2$$

since $\mathbf{d}_k \in \mathcal{S}_k$. Hence, the preconditioner $\mathbf{P}_k = \mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^{\mathsf{T}}$ satisfies the conditions of (C4).

For the preconditioner $\mathbf{P}_k = \mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^\mathsf{T} + \sigma_k\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^\mathsf{T}$, we have

$$\|\mathbf{P}_k\| = \|\mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^\mathsf{T} + \sigma_k\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^\mathsf{T}\| \le \|\widehat{\mathbf{P}}_k\| + \sigma_{\max} \le \widehat{\gamma}_0 + \sigma_{\max}.$$

For the preconditioning step of section 5, we have

$$(1-\eta_1^2)\|\mathbf{g}_{k+1}\|^2 \ge \|\mathbf{g}_{k+1}^\mathsf{T}\mathbf{Z}\|^2$$

since the condition for exiting the subspace was fulfilled. By ($\widehat{\text{C}}$4), it follows that

$$\begin{aligned}\mathbf{g}_{k+1}^\mathsf{T}\mathbf{P}_k\mathbf{g}_{k+1} &= \mathbf{g}_{k+1}^\mathsf{T}(\mathbf{Z}\widehat{\mathbf{P}}_k\mathbf{Z}^\mathsf{T} + \sigma_k\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^\mathsf{T})\mathbf{g}_{k+1}\\ &\ge \widehat{\gamma}_1\|\mathbf{Z}^\mathsf{T}\mathbf{g}_{k+1}\|^2 + \sigma_k\|\overline{\mathbf{Z}}^\mathsf{T}\mathbf{g}_{k+1}\|^2\\ &\ge \min\{\widehat{\gamma}_1, \sigma_{\min}\}\|\mathbf{g}_{k+1}\|^2.\end{aligned}$$

In a similar manner, we have

$$\begin{aligned}\mathbf{d}_k^\mathsf{T}\mathbf{P}_k^{-1}\mathbf{d}_k &= \mathbf{d}_k^\mathsf{T}(\mathbf{Z}\widehat{\mathbf{P}}_k^{-1}\mathbf{Z}^\mathsf{T} + \sigma_k^{-1}\overline{\mathbf{Z}}\,\overline{\mathbf{Z}}^\mathsf{T})\mathbf{d}_k\\ &\ge \widehat{\gamma}_2\|\mathbf{Z}^\mathsf{T}\mathbf{d}_k\|^2 + \sigma_k^{-1}\|\overline{\mathbf{Z}}^\mathsf{T}\mathbf{d}_k\|^2\\ &\ge \min\{\widehat{\gamma}_2, 1/\sigma_{\max}\}\|\mathbf{d}_k\|^2.\end{aligned}$$

Hence, all the preconditioners in section 5 satisfy (C4), and the proof is complete. □

We now remove the assumption ($\widehat{\text{C}}$4) by introducing a strong convexity assumption and assuming that the subspace preconditioner is gotten by applying the L-BFGS algorithm [16, 20] to the subspace problem.

Corollary 6.4. *Suppose that the preconditioned conjugate gradient algorithm given by* (2.8)*,* (2.10)*, and* (2.11) *satisfies* C1 *and* C2 *and that objective function f is twice continuously differentiable and strongly convex. If the subspace preconditioner is implemented by applying the L-BFGS algorithm, as described in* [16]*, with a starting matrix in the L-BFGS update whose eigenvalues lie on an interval $[a,b] \subset (0,\infty)$, then either $\mathbf{g}_k = \mathbf{0}$ for some k, or*

$$\liminf_{k\to\infty} \|\mathbf{g}_k\| = 0.$$

Proof. Since f is strongly convex and twice continuously differentiable, (C3) is satisfied. Let $\mathbf{Z}_k$ denote a matrix whose columns are an orthonormal basis for the subspace at iteration k. Since f is strongly convex, the subspace Hessian $\mathbf{Z}_k^\mathsf{T}\nabla^2 f(\mathbf{x})\mathbf{Z}_k$ is positive definite with largest and smallest eigenvalues between the largest and smallest eigenvalues of $\nabla^2 f(\mathbf{x})$. In [16, Thm. 7.1], the authors show that for a strongly convex objective function, the eigenvalues of the L-BFGS matrices are uniformly bounded away from 0 and $+\infty$. Hence, ($\widehat{\text{C}}$4) is satisfied, and Theorem 6.3 completes the proof. □

Using the techniques of [13, Thm. 2.2], the conclusion of Corollary 6.4 can be strengthened to $\lim_{k\to\infty}\mathbf{g}_k = \mathbf{0}$.

7. Implementation details.

Our strategies for implementing a standard Wolfe line search are given in [13, 14]. We solve the subspace problem (3.1) by the scaled L-BFGS method using a standard implementation [16, 19], which is stated below for completeness.

```
Limited memory BFGS (L-BFGS).
$\mathbf{d} = -\widehat{\mathbf{g}}_k$
for $j = k-1, k-2, \ldots, k-m$
    $\alpha_j \quad \leftarrow \quad \rho_j \widehat{\mathbf{s}}_j^{\mathsf{T}} \mathbf{d}, \quad \rho_j = 1/\widehat{\mathbf{y}}_j^{\mathsf{T}} \widehat{\mathbf{s}}_j$
    $\mathbf{d} \quad \leftarrow \quad \mathbf{d} - \alpha_j \widehat{\mathbf{y}}_j$
end
$\mathbf{d} \leftarrow (\widehat{\mathbf{s}}_{k-1}^{\mathsf{T}} \widehat{\mathbf{y}}_{k-1} / \widehat{\mathbf{y}}_{k-1}^{\mathsf{T}} \widehat{\mathbf{y}}_{k-1}) \mathbf{d}$
for $j = k-m, k-m+1, \ldots, k-1$
    $\beta \quad \leftarrow \quad \rho_j \widehat{\mathbf{y}}_j^{\mathsf{T}} \mathbf{d}$
    $\mathbf{d} \quad \leftarrow \quad \mathbf{d} + \widehat{\mathbf{s}}_j (\alpha_j - \beta)$
end
$\widehat{\mathbf{d}}_k \leftarrow \mathbf{d}$
```

The variables with hats here pertain to the subspace; by the chain rule, the relation between a gradient $\widehat{\mathbf{g}}$ in the subspace and the corresponding gradient $\mathbf{g}$ in the full space is $\widehat{\mathbf{g}} = \mathbf{Z}^{\mathsf{T}} \mathbf{g}$, where the columns of $\mathbf{Z}$ form an orthonormal basis for the subspace. The cost of the L-BFGS update is $O(m^2)$ since the subspace vectors have length m, the dot products and the saxpy operations involve $O(m)$ flops, and the j index has m values. Hence, when m is much smaller than n, the linear algebra overhead associated with the L-BFGS iteration can be much less than the cost of evaluating a gradient in $\mathbb{R}^n$ or updating the iterate $\mathbf{x}_k$ in $\mathbb{R}^n$. After computing a subspace search direction $\widehat{\mathbf{d}}_k$, we perform a Wolfe linesearch, as in [13, 14], to obtain the new iterate

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k = \mathbf{x}_k + \alpha_k \mathbf{Z} \widehat{\mathbf{d}}_k.$$

To help reduce the linear algebra overhead in implementing the subspace techniques, we use the implicit factorization techniques proposed by Siegel [25] and by Gill and Leonard [10, 11]. The implicit factorization techniques are based on the following idea: Let $\mathbf{S}$ denote a matrix whose columns are a basis for the subspace. The columns of $\mathbf{S}$ will be formed from the previous search directions. Suppose that $\mathbf{S} = \mathbf{ZR}$ is the factorization of $\mathbf{S}$ into the product of an n by m matrix $\mathbf{Z}$ with orthonormal columns and an m by m upper triangular matrix $\mathbf{R}$ with positive diagonal entries (for example, see Golub and Van Loan [12] or Trefethen [26]). Since $\mathbf{Z} = \mathbf{S}\mathbf{R}^{-1}$, any occurrence of $\mathbf{Z}$ can be replaced by $\mathbf{S}\mathbf{R}^{-1}$. For example, to compute an expression such as $\mathbf{Zy}$, we solve $\mathbf{Rz} = \mathbf{y}$ for $\mathbf{z}$ and then

$$\mathbf{Zy} = (\mathbf{S}\mathbf{R}^{-1})\mathbf{y} = \mathbf{S}(\mathbf{R}^{-1}\mathbf{y}) = \mathbf{Sz}.$$

We now observe that the implicit algorithm where $\mathbf{Z}$ is replaced by $\mathbf{S}\mathbf{R}^{-1}$ is more efficient than the explicit algorithm where $\mathbf{Z}$ is stored and updated in each iteration. Before we enter the subspace, $\mathbf{S}$ changes in each iteration as we add a new column (new search direction) until the memory is full, and then after adding the new column, an old column (old search direction) will be deleted. If a new column is added to $\mathbf{S}$ and an old column is deleted from $\mathbf{S}$, then the update of $\mathbf{Z}$ takes about $10mn$ flops ($4mn$ flops to add the new column to $\mathbf{Z}$ through a Gram–Schmidt process, and $6mn$ flops to remove the column from $\mathbf{Z}$ using a series of plane rotations [1]). On the other hand, the update of $\mathbf{R}$ can be done in about $3m^2$ flops by a series of plane rotations. Hence, if m is much smaller than n, it is much more efficient to store $\mathbf{S}$ and update $\mathbf{R}$ rather than update $\mathbf{Z}$.

Suppose that at iteration k, we enter the subpace. In order to perform the L-BFGS iteration given above, we need both $\widehat{\mathbf{s}}_j$ and $\widehat{\mathbf{y}}_j$ for $k - m \le j \le k-1$. Note the columns of the upper triangular matrix $\mathbf{R}$ that is computed when iterates are outside the subspace are precisely the vectors $\widehat{\mathbf{s}}_j$ for $k-m \le j \le k-1$. In order to test the subspace condition (3.2), we need to compute $\widehat{\mathbf{g}}_k$ in each iteration. Hence, it is relatively cheap to also form and update the m-component vectors $\widehat{\mathbf{y}}_j$ for $k-m \le j \le k-1$; consequently, when the iterates enter the subspace, both $\widehat{\mathbf{s}}_j$ and $\widehat{\mathbf{y}}_j$ for $k-m \le j \le k-1$ are available. Also, when performing the L-BFGS iteration, we can exploit the sparsity of these vectors since nearly half the components are zero when we first enter the subspace. More precisely, for $k-m \le j \le k-1$, $\widehat{\mathbf{s}}_j$ has a triangular structure, while $\widehat{\mathbf{y}}_j$ has an upper Hessenberg structure except for $\widehat{\mathbf{y}}_{j-m}$, which could be dense.

When we are inside the subspace, the most costly operations are the computations of

$$\mathbf{Z}\widehat{\mathbf{d}}_k = \mathbf{S}\mathbf{R}^{-1}\widehat{\mathbf{d}}_k \quad \text{and} \quad \widehat{\mathbf{g}}_k = \mathbf{Z}^{\mathsf{T}}\mathbf{g}_k = \mathbf{R}^{-\mathsf{T}}\mathbf{S}^{\mathsf{T}}\mathbf{g}_k.$$

The first operation occurs when we map the subspace search direction to the full space, and the second operation occurs when we map the full space gradient into the subspace. Each of these operations involve about $2mn$ flops, assuming m is much smaller than n, when we multiply $\mathbf{S}$ from the right or from the left by a vector. This is the same as the cost of multiplying $\mathbf{Z}$ by a vector from the left or the right.

8. Numerical results. A new version of the CG_DESCENT algorithm has been developed, Version 6.0, that implements the limited memory techniques developed in this paper. We refer to this limited memory conjugate gradient algorithm as L-CG_DESCENT. This code can be downloaded from the following web sites:

www.math.ufl.edu/∼hager or www.math.lsu.edu/∼hozhang.

We compare the performance of L-CG_DESCENT to both L-BFGS [16, 20] and to CG_DESCENT Version 5.3. All three algorithms, L-CG_DESCENT, L-BFGS, and CG_DESCENT Version 5.3, correspond to different parameter settings in Version 6.0 of CG_DESCENT. When the memory is zero, CG_DESCENT 6.0 reduces to CG_DESCENT 5.3 (except for minor changes to the default parameter values). Setting the LBFGS parameter in CG_DESCENT 6.0 to TRUE yields L-BFGS. Hence, all three algorithms employ the same CG_DESCENT line search developed in [13, 14]. This line search performs better than the line search [18] used in the L-BFGS Fortran code on Jorge Nocedal's web page. For example, the line search in the Fortran code fails in 33 of the 145 test problems used in this paper, while the version of L-BFGS contained in CG_DESCENT solves all 145 test problems but one. Hence, the L-BFGS performance shown in this paper should be better than the performance of the L-BFGS Fortran code.

On the web site given above, we have posted the performance results of the algorithms for the test set consisting of the 145 unconstrained problems in CUTEr [3] that could be solved with the sup-norm convergence tolerance $\|\mathbf{g}_k\|_\infty \le 10^{-6}$. We used the default dimensions provided with each of the problems. The remaining 15 unconstrained problems in CUTEr for which this particular convergence tolerance was not achieved were simply skipped. The names of the omitted problems are provided on the web site above.

In the performance profiles comparing L-CG_DESCENT and L-BFGS, we restrict the problem dimension to be at least 50; 79 out of the 145 problems satisfy this constraint. The minimum problem dimension is 50, the maximum dimension is 10,000, and the mean dimension is 3783. The reason for adding this constraint to the problem dimension is that L-CG_DESCENT reduces to L-BFGS for the smaller problems and

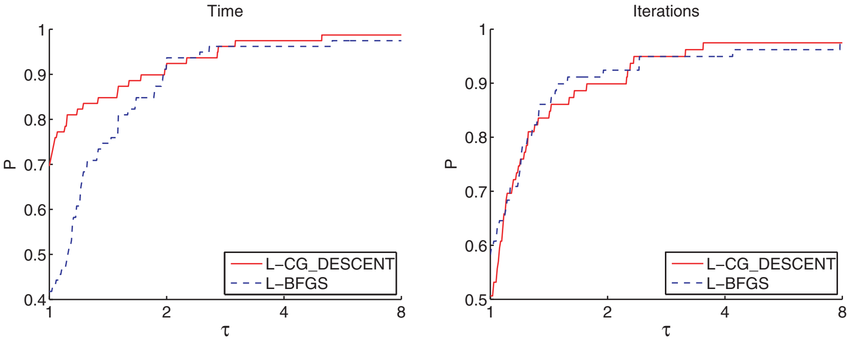

Fig. 8.1. *Performance profiles for L-CG_DESCENT and L-BFGS based on time (left) and number of gradient iterations (right).*

the performance for these small problems is identical. In the analysis, we eliminate these small problems where the performance is identical. In our experiments, we took $m = 11$ for the memory in both L-CG_DESCENT and L-BFGS since this value seemed to give the best performance for both methods. This amounts to storing the 11 previous $\mathbf{s}_k$ in L-CG_DESCENT, and the 11 previous $\mathbf{s}_k$ and $\mathbf{y}_k$ in L-BFGS. Hence, the memory requirement for L-BFGS is double that of L-CG_DESCENT.

The experiments were performed on a Dell Precision T7500 with 96 GB memory and dual six core Intel Xeon Processors (3.46 GZ). Only one core was used for the experiments. Note that CG_DESCENT 6.0 includes a BLAS interface that can exploit additional cores, which could be beneficial for really large problems; however, the BLAS interface was turned off for the experiments.

In Figure 8.1, we show performance profiles [7] based on CPU time and number of iterations. The vertical axis gives the fraction P of problems for which any given method is within a factor τ of the best performance. In the CPU time performance profile plot, the top curve is the method that solved the most problems in a time that was within a factor τ of the best time. The percentage of the test problems for which a method is fastest is given on the left axis of the plot. The right side of the plot gives the percentage of the test problems that were successfully solved by each of the methods. In essence, the right side is a measure of an algorithm's robustness. Here, the number of iterations for the limited memory conjugate gradient algorithm is the total number of iterations both inside and outside the subspace.

In Figure 8.2, we show the performance of L-CG_DESCENT and L-BFGS based on number of function and gradient evaluations. Comparing Figures 8.1 and 8.2, we see that the limited memory conjugate gradient algorithm is faster then the limited memory BFGS algorithm on this test set, while the number of iterations, function evaluations, and gradient evaluations are comparable for the two algorithms.

In Figures 8.3 and 8.4, we give performance profiles comparing CG_DESCENT Version 5.3 and L-CG_DESCENT (labeled CG_DESCENT 6.0 in the figures). These experiments correspond to running Version 6.0 twice, with memory = 11 and memory = 0. When the memory is zero in CG_DESCENT 6.0, the code reduces to CG_DESCENT 5.3 (except for minor changes in default parameters). The performance profiles correspond to all 145 test problems. The plots indicate that L-CG_DESCENT typically performs substantially better than the memory-free version.

Whenever $n \leq m$, the L-CG_DESCENT algorithm theoretically reduces to L-BFGS. Hence, when $n \leq m$, the code automatically uses the L-BFGS search direc-

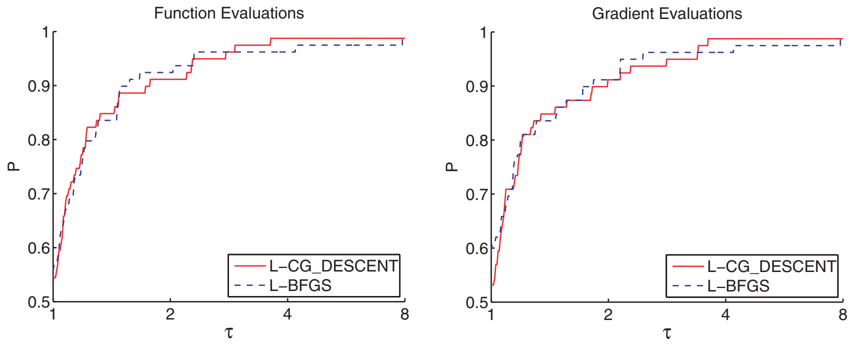

Fig. 8.2. *Performance profiles for L-CG_DESCENT and L-BFGS based on number of function evaluations (left) and number of gradient evaluations (right).*

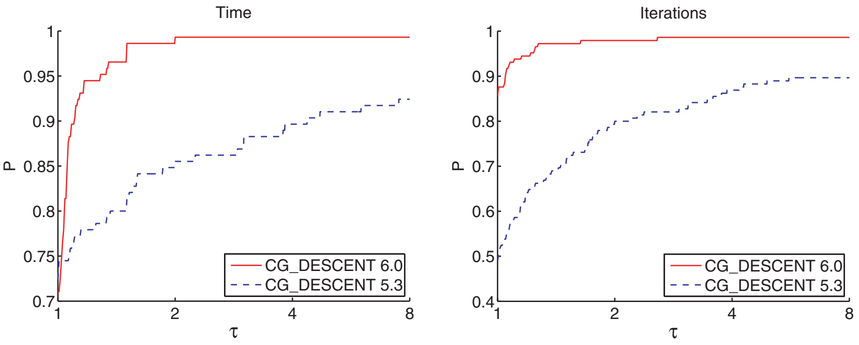

Fig. 8.3. *Performance profiles for L-CG_DESCENT (Version* 6.0*) and CG_DESCENT Version* 5.3 *based on CPU time (left) and number of iterations (right).*

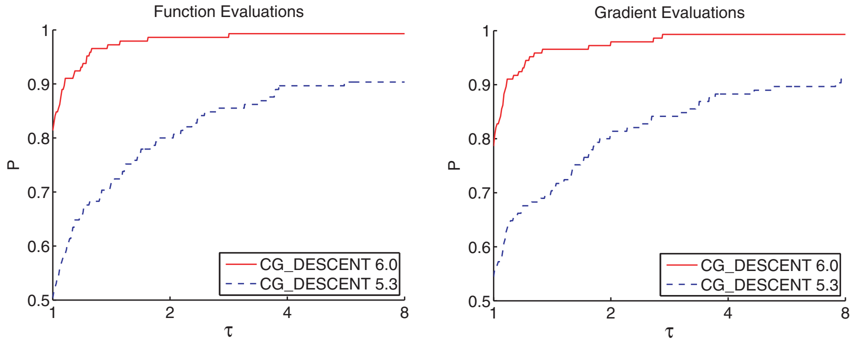

Fig. 8.4. *Performance profiles for L-CG_DESCENT (Version* 6.0*) and CG_DESCENT Version* 5.3 *based on number of function evaluations (left) and number of gradient evaluations (right).*

tions. This led to a tremendous improvement in the performance on the PALMER test problems discussed in the section 1. For example, the number of iterations used by Version 5.3 for PALMER1C was 126,827, while the number of iterations used by L-CG_DESCENT was 11. In the problems with $n > m$, there were 31 cases where L-CG_DESCENT solved at least one subspace problem. For 19 problems, exactly 1 subspace problem was solved, for 3 problems, 2 subspace problems were solved, and for 9 problems, 5 or more subspace problems were solved. The iterations and running

Table 8.1

Comparison between CG_DESCENT Version 5.3 (no memory) and L-CG_DESCENT (with memory). "# Sub" denotes the number of subspace problems that were solved, "#Sub Iter" denotes the number of subspace iterations, and "Tot Iter" denotes the total number of iterations.

Problem	——— L-CG_DESCENT ———				— Version 5.3 —	
Name	# Sub	# Sub Iter	Tot Iter	CPU (s)	Tot Iter	CPU (s)
BDQRTIC	9	51	136	0.155	761	0.597
ERRINROS	18	130	380	0.003	637	0.004
EXTROSNB	74	750	3808	0.323	6879	0.494
NCB20B	187	756	2935	50.365	4595	78.554
NCB20	273	1294	4437	54.876	391	4.783
NONDQUAR	104	787	1942	0.701	2059	0.640
PARKCH	32	368	700	34.130	1597	42.758
PENALTY3	5	33	99	0.814	117	0.860
TOINTPSP	7	56	143	0.001	136	0.000

time for these 9 problems are shown in Table 8.1. In 5 out of these 9 problems, the limited memory code gave significantly better performance than Version 5.3. In 3 problems, there were almost no difference in the codes, while in 1 problem, NCB20, Version 5.3 was much faster. This particular problem is very unstable with respect to the stepsize in an initial iteration. For a small modification in the initial stepsize, the iterates take a much slower path to the optimum and the CPU time can increase by a factor of 10.

9. Conclusions.

A new limited memory conjugate gradient algorithm has been introduced and analyzed. It was implemented within the framework of the conjugate gradient algorithm [13, 14, 15] CG_DESCENT. Unlike previous limited memory algorithms [10, 11, 16, 20], the memory is mostly used to monitor convergence, and the memory is only used to compute the search direction when the gradient vectors lose orthogonality. When the loss of orthogonality is detected, a subspace problem is solved to restore orthogonality. If the subspace problem is solved by a preconditioned version of the CG_DESCENT algorithm and the preconditioning matrices possess the properties stated in $(\widehat{\mathrm{C}}4)$, then the iterates possess the same global convergence property established previously for the CG_DESCENT algorithm.

In [14] it was observed that the memoryless version of CG_DESCENT was faster than L-BFGS, but L-BFGS had better performance relative to the number of iterations, function evaluations, and gradient evaluations. As seen in Figures 8.1 and 8.2, L-CG_DESCENT is able to match L-BFGS with respect to the number of iterations, function evaluations, and gradient evaluations. It is faster than L-BFGS due to the reduced amount of linear algebra within each iteration. In L-CG_DESCENT, each iteration where orthogonality is monitored requires on the order of $4mn$ flops at most since we need to multiply both a gradient and a search direction by the vectors in memory. In theory, this can be reduced to $2mn$ flops by exploiting the known relationship between the gradient, the new search direction, and the previous search direction in the conjugate gradient method. And if the orthogonality is preserved for enough iterations, then we turn off the orthogonality test for a number of iterations. On the other hand, the L-BFGS algorithm (section 7) involves about $8mn$ flops in each iteration. Hence, L-CG_DESCENT is able to monitor orthogonality relatively cheaply, and the memory is only used when necessary. The algorithm is able to match L-BFGS with respect to the number of iterations, function evaluations, and gradient evaluations, while reducing CPU time by performing fewer operations on the memory in each iteration.

REFERENCES

[1] R. H. Bartels, G. H. Golub, and M. A. Saunders, *Numerical techniques in mathematical programming*, in Nonlinear Programming, J. B. Rosen, O. L. Mangasarian, and K. Ritter, eds., Academic Press, New York, 1970, pp. 123–176.

[2] J. Barzilai and J. M. Borwein, *Two point step size gradient methods*, IMA J. Numer. Anal., 8 (1988), pp. 141–148.

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

[4] Y. H. Dai and C. X. Kou, *A nonlinear conjugate gradient algorithm with an optimal property and an improved Wolfe line search*, SIAM J. Optim., 23 (2013), pp. 296–320.

[5] Y. H. Dai and Y. Yuan, *A nonlinear conjugate gradient method with a strong global convergence property*, SIAM J. Optim., 10 (1999), pp. 177–182.

[6] Y. H. Dai and Y. Yuan, *An efficient hybrid conjugate gradient method for unconstrained optimization*, Ann. Oper. Res., 103 (2001), pp. 33–47.

[7] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.

[8] R. Fletcher and C. Reeves, *Function minimization by conjugate gradients*, Comput. J., 7 (1964), pp. 149–154.

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

[10] P. E. Gill and M. W. Leonard, *Reduced-Hessian quasi-Newton methods for unconstrained optimization*, SIAM J. Optim., 12 (2001), pp. 209–237.

[11] P. E. Gill and M. W. Leonard, *Limited memory reduced-Hessian methods for large-scale unconstrained optimization*, SIAM J. Optim., 14 (2003), pp. 380–401.

[12] G. H. Golub and C. Van Loan, *Matrix Computations*, 2nd ed., Johns Hopkins University Press, Baltimore, MD, 1989.

[13] W. W. Hager and H. Zhang, *A new conjugate gradient method with guaranteed descent and an efficient line search*, SIAM J. Optim., 16 (2005), pp. 170–192.

[14] W. W. Hager and H. Zhang, *Algorithm* 851: *CG_DESCENT, a conjugate gradient method with guaranteed descent*, ACM Trans. Math. Software, 32 (2006), pp. 113–137.

[15] W. W. Hager and H. Zhang, *A survey of nonlinear conjugate gradient methods*, Pacific J. Optim., 2 (2006), pp. 35–58.

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

[17] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*, Springer, Berlin, 2008.

[18] J. J. Moré and D. J. Thuente, *Line search algorithms with guaranteed sufficient decrease*, ACM Trans. Math. Software, 20 (1994), pp. 286–307.

[19] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, 1999.

[20] J. Nocedal, *Updating quasi-Newton matrices with limited storage*, Math. Comp., 35 (1980), pp. 773–782.

[21] S. S. Oren, *Self-scaling Variable Metric Algorithms for Unconstrained Minimization*, Ph.D. thesis, Department of Engineering Economic Systems, Stanford University, Stanford, CA, 1972.

[22] J. M. Perry, *A Class of Conjugate Gradient Algorithms with a Two Step Variable Metric Memory*, Technical report 269, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL, 1977.

[23] M. J. D. Powell, *Convergence properties of algorithms for nonlinear optimization*, SIAM Rev., 28 (1986), pp. 487–500.

[24] D. F. Shanno, *On the convergence of a new conjugate gradient algorithm*, SIAM J. Numer. Anal., 15 (1978), pp. 1247–1257.

[25] D. Siegel, *Modifying the BFGS update by a new column scaling technique*, Math. Program., 66 (1993), pp. 48–78.

[26] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[27] P. Wolfe, *Convergence conditions for ascent methods*, SIAM Rev., 11 (1969), pp. 226–235.

[28] P. Wolfe, *Convergence conditions for ascent methods* II: *Some corrections*, SIAM Rev., 13 (1971), pp. 185–188.