Algorithm PREQN: Fortran 77 Subroutines for Preconditioning the Conjugate Gradient Method

José Luis Morales* Jorge Nocedal†

March 14, 2000

Abstract

PREQN is a package of Fortran 77 subroutines for automatically generating preconditioners for the conjugate gradient method. It is designed for solving a sequence of linear systems $A_i x = b_i,\ i = 1, \ldots, t$, where the coefficient matrices A_i are symmetric and positive definite and vary slowly. Problems of this type arise, for example, in nonlinear optimization. The preconditioners are based on limited memory quasi-Newton updating and are recommended for problems in which: i) the coefficient matrices are not explicitly known and only matrix-vector products of the form $A_i v$ can be computed; or ii) the coefficient matrices are not sparse. PREQN is written so that a single call from a conjugate gradient routine performs the preconditioning operation and stores information needed for the generation of a new preconditioner.

Categories and Subject Descriptors: G.1 [**Numerical Analysis**]: G 1.3 Numerical Linear Algebra—*Linear systems*; G 1.6 Optimization—*Unconstrained optimization*; G.4 [**Mathematical Software**]:—*Efficiency; Reliability and robustness.*

General Terms: Algorithms.

Additional Key Words and Phrases: Preconditioning, conjugate gradient method, quasi-Newton method, Hessian-free Newton method, limited memory method.

*Departamento de Matemáticas, Instituto Tecnológico Autónomo de México, Río Hondo 1, Col Tizapán San Angel, México D.F. CP 01000, México. jmorales@gauss.rhon.itam.mx. This author was supported by CONACyT grant 25710-A and by Asociación Mexicana de Cultura AC.

†ECE Department, Northwestern University, Evanston Il 60208, USA. nocedal@ece.nwu.edu; www.ece.nwu.edu∼nocedal. This author was supported by National Science Foundation grants CCR-9625613 and INT-9416004 and by Department of Energy grant DE-FG02-87ER25047-A004.

1 Introduction

In this paper we describe Fortran 77 subroutines for preconditioning the conjugate gradient method. They are designed for solving a sequence of linear systems,

$$A_i x = b_i, \quad i = 1, 2, \ldots, t, \tag{1.1}$$

where the matrices A_i vary slowly and the right hand side vectors are arbitrary. We assume that each matrix A_i is symmetric and positive definite and of size n. A special case of (1.1), known as the multiple right hand sides problem, occurs when the matrices $\{A_i\}$ are all the same.

The preconditioners are defined by means of quasi-Newton updating and do not require explicit knowledge of the matrices A_i, but only make use of products $A_i v$. They are particularly well suited for problems in which these matrix-vector products are expensive to compute, or when low accuracy in the solution of some of the systems (1.1) is acceptable. Problems with these characteristics arise, for example, in Newton methods for nonlinear optimization [10, 6, 8, 11, 12], in the numerical solution of differential equations [9], and in statistics [1].

We denote the quasi-Newton preconditioner for the i-th system as H_i, so that (1.1) is solved by applying the conjugate gradient (CG) method to

$$H_i A_i x = H_i b_i, \quad i = 1, 2, \ldots, t. \tag{1.2}$$

The preconditioners are constructed by means of a limited memory quasi-Newton updating procedure described below; see also [5]. That paper also presents numerical tests illustrating the performance of the preconditioner on a wide variety of problems.

PREQN is a package of Fortran 77 subroutines that implements these quasi-Newton preconditioners. A salient feature of our implementation is that a single call to PREQN applies the preconditioner and transmits the information that is required in the generation of the next preconditioner. This allows the user to easily invoke PREQN from any conjugate gradient routine.

2 Overview of the Quasi-Newton Preconditioners

We start by reformulating the i-th subproblem $A_i x = b_i$ in the sequence (1.1), as the following optimization problem

$$\text{minimize} \qquad q(x) = \frac{1}{2} x^T A x - b^T x, \tag{2.3}$$

where, for notational convenience, the subscript i has been ignored.

Suppose that we apply the BFGS method (see e.g. [2]) to compute the solution of (2.3). Starting from initial approximations $x^{(0)}$ and $H^{(0)}$, to the solution x^* and to A^{-1} respectively, the BFGS method generates new approximations $x^{(j)}$ and $H^{(j)}$ by means of the formulae:

$$\bar{p}^{(j)} = -H^{(j)} \nabla q(x^{(j)}), \qquad x^{(j+1)} = x^{(j)} + \bar{\alpha}_j \bar{p}^{(j)}, \tag{2.4}$$

where $\bar{\alpha}_j > 0$ is the step length that minimizes q along the direction $\bar{p}^{(j)}$. The quasi-Newton matrices are updated by means of the BFGS formula,

$$H^{(j+1)} = H^{(j)} + V_j^T H^{(j)} V_j + \rho_j y_j s_j^T, \tag{2.5}$$

where

$$V_j = I - \rho_j y_j s_j^T, \qquad \rho_j = 1/y_j^T s_j, \tag{2.6}$$

$$s_j = x^{(j+1)} - x^{(j)}, \quad y_j = \nabla q(x^{(j+1)}) - \nabla(x^{(j)}). \tag{2.7}$$

The pair of vectors (s_j, y_j) is called a *correction pair* and satisfies $s_j^T y_j > 0$ because of our assumption that A is positive definite.

Note that we can also write

$$s_j = \bar{\alpha}_j \bar{p}^{(j)}, \qquad y_j = \bar{\alpha}_j A \bar{p}^{(j)}. \tag{2.8}$$

and since the scalars $\bar{\alpha_j}$ will cancel out in the computation that follows, we prefer to define the correction pair as

$$(s^{(j)}, y^{(j)}) \leftarrow (\bar{p}^{(j)}, A\bar{p}^{(j)}). \tag{2.9}$$

From (2.5) it is clear that the matrices $H^{(j)}$ will usually be dense, and therefore, their storage and the computation of the matrix-vector product (2.4) are impractical. To overcome this difficulty, we will not form the matrices $H^{(j)}$, but only store the vectors s_j, y_j and the scalars ρ_j that form them. More specifically, we can express the update as

$$\begin{aligned} H^{(j)} \;=\; & (V_{j-1}^T \cdots V_0^T) H^{(1)} (V_0 \cdots V_{j-1}) \\ & + \rho_0 (V_{j-1}^T \cdots V_1^T) s_0 s_0^T (V_1 \cdots V_{j-1}) \\ & + \rho_1 (V_{j-1}^T \cdots V_2^T) s_1 s_1^T (V_2 \cdots V_{j-1}) \\ & \vdots \\ & + \rho_j s_{j-1} s_{j-1}^T. \end{aligned} \tag{2.10}$$

A recursive formula described in [7] makes use of the structure of (2.10) to compute the product (2.4) in approximately $4jn$ floating point operations. The storage requirements for this computation are $2jn$ spaces to hold the set of correction pairs, and $2j$ spaces to hold the scalars ρ_j and intermediate results. It is now clear that the computational resources can be kept to a reasonable level by limiting the number of pairs that participate in (2.10). This gives rise to a limited memory quasi-Newton method.

In summary, the set of correction pairs

$$\mathcal{S} = \{(\bar{p}^{(j)}, A\bar{p}^{(j)}) \mid j = 0, \ldots\}$$

along with the BFGS formula implicitly define an approximation to A^{-1}. In practice we wish to store only a small subset of $\mathcal{S}$ which defines the inverse of A on a small subspace of R^n. This poses the question of how many and which pairs must be chosen, an issue we discuss next.

2.1 Preconditioning the CG Method

The role of limited memory quasi-Newton matrices as preconditioners for the CG iteration becomes clear by noting that, when applied to positive definite quadratic functions, the BFGS and CG methods are equivalent. Therefore, as we solve a linear system by means of the CG method, we can save the correction pairs (2.9) that determine an approximation to A^{-1}, and use this approximation to precondition the next system in (1.1).

We start by solving the first system $A_1 x = b_1$ with the unpreconditioned CG method, so that $H_1 = I$. During the course of the CG iteration, we collect m correction pairs

$$\{p^{(j)}, A_1 p^{(j)}\}, \quad j = l_1, l_2, \ldots, l_m, \tag{2.11}$$

where $\{p^{(j)}\}_{j=1,2,\ldots}$ is the sequence of search directions produced by the CG method. (The choice of the indices $l_1, l_2, \ldots, l_m$ will be discussed below.) The user determines the number m of pairs to be collected. When the CG method has completed the solution of the first system $A_1 x = b_1$, the m pairs $\{p^{(j)}, A_1 p^{(j)}\}$ are used to construct a limited memory quasi-Newton matrix $H_2(m)$ which is the preconditioner for the second system $A_2 x = b_2$. While solving this second system with the preconditioned CG method, we collect new pairs $\{p^{(j)}, A_2 p^{(j)}\}$ to define the next preconditioner. We proceed in this manner until all the linear systems have been solved.

An important question in the design of the quasi-Newton preconditioners is how to select the m correction pairs $\{p^{(j)}, A_i p^{(j)}\}$ during the CG iteration. In the companion paper [5] we proposed two strategies that have performed well on a wide range of problems: a (nearly) uniform sample of m pairs collected during the course of the CG iteration, and the set formed by the last m pairs generated by the CG iteration. These two strategies are implemented in PREQN.

We use the notation $H_i(m)$ to indicate the amount of information used in the preconditioner. Values of m, in the range $[4, 20]$, are recommended for most problems, independently of the value of n. Of course, the available memory may impose a further restriction on m. As mentioned earlier, the multiplication of $H_i(m)$ with a vector can be performed by a sequence of inner products involving the correction pairs, and requires approximately $4mn$ floating point operations. The application of the preconditioner may therefore be expensive compared, say, with incomplete Cholesky preconditioning [4], and we advocate its use for problems in which the coefficient matrices A_i are not known, are not sparse, or cannot be computed cheaply. We should also note that, if a good preconditioner is known, it can be used to precondition the first problem in (1.1), and the rest of the problems can use the quasi-Newton preconditioner.

We have coded the quasi-Newton preconditioners so that they can be incorporated easily in any CG routine. In Section 3 we provide an algorithmic description of the main routine in PREQN. In Section 4 we describe the parameters of the main routine, and in Section 5 we present a sample of numerical results illustrating the performance of the software.

3 Implementation of the Routines

We begin by stating the CG method (cf. [3]) when applied to the i-th system $A_i x = b_i$ in the sequence (1.1).

Preconditioned CG Method

`input:` $A_i, b_i, m, H_i(m), x^{(0)}$,

`output:` x_i^*

compute $r^{(0)} = A_i x^{(0)} - b_i$

for $j = 1, 2, \ldots$

1. compute $z^{(j-1)} = H_i(m) r^{(j-1)}$ [PREQN a,c]
2. if convergence test is satisfied, set $x_i^* = x^{(j-1)}$ and **stop**
3. $\rho_{j-1} = r^{(j-1)^T} z^{(j-1)}$
4. **if** $j = 1$ **then**
   $p^{(1)} = -z^{(0)}$
   else
   $\beta_{j-1} = \rho_{j-1}/\rho_{j-2}$
   $p^{(j)} = -z^{(j-1)} + \beta_{j-1} p^{(j-1)}$
   end if [PREQN b]
5. $\alpha_j = \rho_{j-1}/p^{(j)^T} A_i p^{(j)}$ [PREQN c]
6. $x^{(j)} = x^{(j-1)} + \alpha_j p^{(j)}$
7. $r^{(j)} = r^{(j-1)} + \alpha_j A_i p^{(j)}$

end for

As mentioned earlier, one of our goals is to make only one call to PREQN at each CG iteration. Let us consider the tasks that are required to construct and apply the preconditioner.

a. We need to compute the product $H_i(m) r^{(j-1)}$ in step 1 of the preconditioned CG method. This will be done by means of a call to PREQN.

b. We must decide if the most recently generated pair $\{p^{(j)}, A_i p^{(j)}\}$ should be saved and used to define the new preconditioner $H_{i+1}(m)$. This could be done by calling PREQN near the end of the CG iteration, after the new correction pair has been computed in steps 4 and 5. To facilitate the use of the software, however, we can delay this selection process until the next CG iteration, just before the application of the preconditioner in step 1. Thus the call to PREQN requesting the application of the residual is accompanied by the transmission of a correction pair $\{p^{(j)}, A_i p^{(j)}\}$. The

decision to incorporate this pair into the new preconditioner $H_{i+1}(m)$ is delegated to the appropriate routine, and depends on the value of the parameter IOP; see next section.

c. Once a linear system $A_i x = b_i$ has been solved, we must remove the old preconditioner, and replace it by the new preconditioner $H_{i+1}(m)$ for the next system $A_{i+1}x = b_{i+1}$. This could be done by means of a call to PREQN at the end of step 2, when all the information from the CG run has been collected. Instead, we inform PREQN during the call in step 1 that the solution of a new linear system has commenced. The formation of the new preconditioner thus takes place just before it is applied to the first residual vector.

This is summarized in the procedure below which outlines the main tasks of PREQN.

```
% i = problem number;   j = CG iteration number

Procedure PREQN

    input: i, j, m, {p^(j-1), Ap^(j-1)}, r^(j-1),
    output: z^(j-1) ← H_i(m) r^(j-1),

    if j > 1 decide if {p^(j-1), Ap^(j-1)} is to be saved
    if i = 1 then
        z^(j-1) ← r^(j-1)                    % No preconditioning
    else
        if j = 1 build preconditioner H_{i+1}(m)
        z^(j-1) ← H_i(m) r^(j-1)
    endif
```

This simultaneous transmission of information greatly facilitates the use of the package. A code implementing the CG method can be easily modified so that the computation of the preconditioned residual is done by means of a call to PREQN in step 1.

4 Choice of certain parameters

The calling sequence is

```
CALL PREQN ( N, M, IOP, IPROB, JCG, S, Y, R, Z, W, LW, IW, LIW, BUILD,
             INFO, MSSG )
```

The meaning of the parameters is provided in the code documentation. We now discuss how to choose three parameters that have an important impact in the performance of the preconditioner.

The variable M specifies the maximum number m of pairs used in the definition of the quasi-Newton preconditioner. In our experience, good performance is observed with values of M in the range $[4, 20]$; smaller values normally do not provide adequate preconditioning, whereas values greater than 20 may result in high computing times.

The variable IOP determines the scheme for the selection of correction pairs. If IOP = 1, the pairs are selected as a uniform sample throughout the CG cycle. If IOP = 2, the last M computed pairs are selected.

When the uniform sampling scheme is chosen (IOP=1) an extra pair may be stored and used in the construction of the preconditioner, as we now explain. The first M pairs are selected by the uniform sampling scheme. A routine in the package checks if the last pair produced by the CG iteration was selected, and if not, saves it. In this case the number of correction pairs becomes M + 1. We have observed a slight improvement in performance if the last pair is always included, and since by incorporating it the storage requirements and computational effort increase only modestly, we have implemented this strategy in the uniform sampling scheme. The use of IOP = 1 requires M to be an even integer.

The variable BUILD indicates whether the preconditioner for the problem IPROB − 1 should be built. It should be set to .FALSE. before the first call to PREQN. This variable allows the reuse of a preconditioner for several problems. If BUILD = .TRUE. a preconditioner is computed using the information collected during the CG cycle for problem IPROB − 1. If BUILD = .FALSE. the information collected during the previous CG cycle is ignored and superseeded by the information collected during the current CG cycle.

In some situations, it may be useful to apply the same preconditioner for several problems in the sequence (1.1). One example of this is when the CG iteration performed a very small number of iterations (say, less than 5) to meet the stopping test for the current problem. A preconditioner based on so little information is not efficient, and it is preferable to revert to a previous preconditioner. Another case in which it may be appropriate to reuse the preconditioner is when the coefficient matrices in (1.1) are constant and only the right hand side vector varies. In the next section we report numerical tests with problems of this type. The option BUILD = .FALSE. must be used with caution; in general we recommend that the preconditioner be recomputed for each new problem in the sequence.

5 Numerical Results

In this section we illustrate the performance of the quasi-Newton preconditioners when solving a sequence of the form (1.1) where the coefficient matrix is constant and the right hand side vectors vary. We report results for three matrices arising in finite element computations, denoted by $A_{1_0}, A_{1_1}, A_{2_0}$, which are described in [5]. The tests reported here differ from those in [5] in that we use a different stopping test to terminate the CG iteration.

For each matrix A, we solve the sequence of problems

$$Ax = b_i, \quad i = 1, \ldots, 51, \tag{5.12}$$

where the right hand side vectors b_i, $i = 2, \ldots, 51$ are obtained as perturbations of an initial vector b_1; see [5]. In order to solve (5.12) we proceed as follows: i) the quasi-Newton preconditioner is computed with the information gathered by the CG method while solving the first system $Ax = b_1$; ii) the remaining 50 problems are solved using this preconditioner. We report in Table 1 the average number of CG iterations (rounded to the nearest integer) required to solve the 50 preconditioned systems, for various values of m. Both strategies for forming the preconditioner were used: uniform sampling (IOP = 1) and the last correction pairs (IOP = 2). The stopping test for the CG method was

$$||r^{(j)}||_\infty \leq 10^{-7} ||r^{(0)}||_\infty. \tag{5.13}$$

	A_{1_0}		A_{1_1}		A_{2_0}	
m	IOP=1	IOP=2	IOP=1	IOP=2	IOP=1	IOP=2
m	iter	iter	iter	iter	iter	iter
0	49	49	449	449	66	66
4	46	46	399	446	79	89
8	32	42	237	442	59	88
12	21	38	191	438	53	87
16	17	34	116	433	47	86
20	16	30	117	430	43	81

Table 1. Results for 3 test matrices using multiple right hand side vectors. The table reports average number of CG iterations for 50 runs, using different values of the memory parameter m and different storage schemes. The initial point for every CG iteration is $x^{(0)} = 0$.

Whereas in these tests the uniform sampling technique performs much better than the strategy of using the last m pairs, in most of the experiments we have performed, the two strategies are comparable in performance. Observe that the preconditioner is not always beneficial, and that its effectiveness tends to increase with the amount of information stored in it. As is the case in this example, we have observed that the preconditioner is usually capable of significantly reducing the number of CG iterations, but it is recommended to experiment with various values of m to find an appropriate setting for the application at hand.

To give an idea of the types of problems for which the quasi-Newton preconditioner may be effective, we can estimate the amount of work involved. Consider, for example, the A_{1_0} problem with IOP=1 and $m = 16$. Since the unpreconditioned algorithm requires $10n$ flops per iteration, in addition to the matrix-vector product, the preconditioned algorithm will be more efficient if the matrix-vector multiplication requires more than $24n$ operations. Therefore the preconditioner is unlikely to be useful for very sparse problems. Nevertheless in optimization , the matrix-vector product is roughly as expensive as the cost of evaluating a gradient, which can be a large multiple of n in many applications.

Acknowledgements. We thank two referees for several useful suggestions on how to improve the description of the preconditioner.

References

[1] T. F. Chan and M. K. Ng, *Galerkin Projection Methods for Solving Multiple Linear Systems*, Technical Report (96-31), Department of Mathematics, University of Calif. at Los Angeles, Los Angeles CA 90024., 1996.

[2] J.E. Dennis, Jr. and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[3] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.

[4] M. T. Jones and P. E. Plassmann, *An Improved Incomplete Cholesky Factorization*, Preprint MCS-P206-0191, MCS Division, Argonne National Laboratory, Argonne, Ill., 1991.

[5] J.L. Morales and J. Nocedal, *Automatic Preconditioning by Limited Memory Quasi-Newton Updating*, to appear in SIAM Journal on Optimization.

[6] S. G. Nash, *Newton-type minimization via the Lanczos method*, SIAM Journal on Numerical Analysis, 21 (1984), pp. 553–572.

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

[8] D. P. O'Leary, *A discrete Newton algorithm for minimizing a function of many variables*, Mathematical Programming, 23 (1982), pp. 20–33.

[9] J. M. Ortega and W. C. Rheinboldt, *Iterative solution of nonlinear equations in several variables*, Academic Press, New York and London, 1970.

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

[11] P. L. Toint, *Towards an efficient sparsity exploiting Newton method for minimization*, in Sparse Matrices and Their Uses, Academic Press, New York, 1981, pp. 57–87.

[12] D. Xie and T. T. Schlick, *Efficient Implementation of the Truncated Newton Method for Large-Scale Chemistry Applications*, to appear in SIAM Journal on Optimization.