Automatic Preconditioning by Limited Memory Quasi-Newton Updating

José Luis Morales* Jorge Nocedal†

September 6, 1999

Abstract

The paper proposes a preconditioner for the conjugate gradient method (CG) that is designed for solving systems of equations $Ax = b_i$ with different right hand side vectors, or for solving a sequence of slowly varying systems $A_k x = b_k$. The preconditioner has the form of a limited memory quasi-Newton matrix and is generated using information from the CG iteration. The automatic preconditioner does not require explicit knowledge of the coefficient matrix A and is therefore suitable for problems where only products of A times a vector can be computed. Numerical experiments indicate that the preconditioner has most to offer when these matrix-vector products are expensive to compute, and when low accuracy in the solution is required. The effectiveness of the preconditioner is tested within a Hessian-free Newton method for optimization, and by solving certain linear systems arising in finite element models.

Key words: 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 Asociación Mexicana de Cultura, A.C.

†ECE Department, Northwestern University, Evanston Il 60208. nocedal@ece.nwu.edu. 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

We describe a technique for automatically generating preconditioners for the conjugate gradient (CG) method. It is designed for solving either a sequence of linear systems

$$Ax = b_i \qquad i = 1, \ldots, t \tag{1.1}$$

in which the coefficient matrix is constant but the right hand side varies, or for solving a sequence of systems

$$A_k x = b_k \qquad k = 1, \ldots, t \tag{1.2}$$

where the matrices A_k vary slowly and the right hand sides b_k are arbitrary. We assume in both cases that the coefficient matrices are symmetric and positive definite.

The automatic preconditioner makes use of quasi-Newton updating techniques. It requires that the first problem in (1.1) or (1.2) be solved by the unpreconditioned CG method, and based on the information generated during this run, generates a preconditioner for solving the next linear system in the sequence. More precisely, if $\{x_i\}$ and $\{r_i\}$ denote the sequence of iterates and residuals generated by the CG method when applied to the first of the systems in (1.1) or (1.2), we compute and store the vectors

$$s_i = x_{i+1} - x_i, \qquad y_i = r_{i+1} - r_i, \qquad i = l_1, \ldots, l_m \tag{1.3}$$

corresponding to m iterates of the CG process, where m is an integer selected by the user. We then use these vectors to define a limited memory BFGS matrix H, which we call the *quasi-Newton preconditioner*, and which will be used to precondition the CG method when applied the next problem in the sequence (1.1) or (1.2). The parameter m determines the amount of memory in the preconditioner, and is normally chosen to be much smaller than the number of variables, so that the cost of applying the preconditioner is not too large.

The first question is how to select the m vectors (1.3) to be used in the definition of the quasi-Newton matrix. The two strategies that have performed best in our tests are to select the *last* m vectors generated during the CG iteration, or to take a *uniform* sample of them. In this paper we will concentrate on the second strategy: we will save m vectors that are approximately evenly distributed throughout the CG run. A detailed description of the quasi-Newton preconditioner will be given in the next section, after we have reviewed the main ideas of limited memory BFGS updating.

Our main interest is in accelerating the CG iteration used in Hessian-free Newton methods for nonlinear optimization. There one needs to solve systems of the form (1.2) where A_k is the Hessian of the objective function at the current iterate. Hessian-free Newton methods assume that the Hessian of the objective function is not known explicitly, but that products of A_k with a vector can be approximated by finite-differences of gradients, or by means of automatic differentiation. In either case these products can be very expensive to compute. After showing that the automatic preconditioner appears to be quite useful in a Hessian-free Newton method, we explore its behavior on a different context by testing it in the solution of linear systems arising in finite element models. In these tests we consider both problems of the form (1.1) and (1.2).

The idea of saving information from the CG iteration in the form of a quasi-Newton matrix is not new. Nash [15] constructs a limited memory matrix with memory $m = 2$, which is different from the one proposed here, to precondition the linear system of equations arising in the Hessian-free Newton method for optimization. O'Leary and Yeremin [21] explore the use of (full-memory) quasi-Newton matrices as preconditioners for the solution of closely related linear systems. Byrd, Nocedal and Zhu [6] propose an optimization algorithm in which information corresponding to the last m iterations of the CG method is used to update a limited memory matrix. However, in that algorithm the limited memory matrix is used only to compute a search direction and not as a preconditioner for the CG method. The motivation for the automatic preconditioner proposed in this paper arose while performing numerical tests with a Hessian-free Newton method for large scale optimization. We observed that the 2-step preconditioner of Nash was effective only in a few test problems, but that the technique proposed here gave improvements over a wide range of problems. The objective of this paper is to suggest that the automatic preconditioner is well suited not only in optimization, but in a wider context. Therefore we present our discussion in the framework of the general problems (1.1)-(1.2).

2. The Quasi-Newton Preconditioner

In the BFGS updating formula for minimizing a function f (see e.g. [7, 8, 10]) we are given a symmetric and positive definite $n \times n$ matrix H_k that approximates the inverse of the Hessian of f, and a pair of n-vectors $s_k = x_{k+1} - x_k$, and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ satisfying the condition $s_k^T y_k > 0$. Using this we compute a new inverse Hessian approximation H_{k+1} by means of the updating formula

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

where

$$\rho_k = 1/y_k^T s_k, \qquad V_k = I - \rho_k y_k s_k^T. \tag{2.2}$$

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

Even if H_k is sparse, the new BFGS matrix H_{k+1} will generally be dense, so that storing and manipulating it is prohibitive when the number of variables is large. To circumvent this problem, the limited memory approach makes use of an alternative representation of the updating process in which the quasi-Newton matrices are not explicitly formed.

It follows from (2.1)-(2.2) that if an initial matrix $\bar{H}$ is updated m times using the BFGS formula and the m pairs $\{s_i, y_i\}$, $i = k-1, \ldots, k-m$, then the resulting matrix $H(m)$ can be written as

$$\begin{aligned} H(m) &= \left(V_{k-1}^T \cdots V_{k-m}^T\right) \bar{H} \left(V_{k-m} \cdots V_{k-1}\right) \\ &+ \rho_{k-m} \left(V_{k-1}^T \cdots V_{k-m+1}^T\right) s_{k-m} s_{k-m}^T \left(V_{k-m+1} \cdots V_{k-1}\right) \\ &+ \rho_{k-m+1} \left(V_{k-1}^T \cdots V_{k-m+2}^T\right) s_{k-m+1} s_{k-m+1}^T \left(V_{k-m+2} \cdots V_{k-1}\right) \end{aligned}$$

$$\vdots$$

$$+ \quad \rho_{k-1} s_{k-1} s_{k-1}^T. \tag{2.3}$$

Thus instead of forming $H(m)$ we can store the scalars ρ_i and the vectors $\{s_i, y_i\}$, $i = k-1, \ldots, k-m$ which determine the matrices V_i. A recursive formula described in [13, 19] takes advantage of the symmetry in (2.3) to compute the product $H(m)v$, for any vector v, with only $4mn$ floating point operations.

The so-called L-BFGS method described in [19, 12, 9] updates Hessian approximations as follows. We first choose a sparse (usually diagonal) initial Hessian approximation $\bar{H}$, and define the first m approximations through (2.3) as $H(1), \ldots, H(m)$. At this stage the storage is full, and to construct the new Hessian approximation, we first delete the oldest correction pair from the set $\{s_i, y_i\}$ to make room for the newest one, $\{s_k, y_k\}$. The new Hessian approximation $H(m+1)$ is defined by (2.3), using the new set of pairs $\{s_i, y_i\}$. This process is repeated during all subsequent iterations: the oldest correction pair is removed to make space for the newest one.

In this paper we are interested in solving positive definite linear systems $Ax = b$, and therefore the function to be minimized is the quadratic $\frac{1}{2}x^T Ax - b^T x$, whose gradient is equal to the residual $r(x) = Ax - b$. Therefore when using the BFGS updating formula to minimize this quadratic, it is appropriate to define s_k and y_k by (1.3). To find a preconditioner for solving a sequence of problems of the form (1.1), with a constant coefficient matrix but different right hand sides, we proceed as follows. We solve the first of the systems using the unpreconditioned CG method. We save m correction pairs $\{s_i, y_i\}$ generated during this CG iteration and use (2.3) to define the preconditioner to be $H(m)$. We solve the rest of the linear systems in (1.1) using the preconditioned CG method with this fixed preconditioner.

A similar approach can be used for solving the sequence of slowly varying linear systems (1.2). An alternative, in this case, is to generate a new preconditioner during the solution of every linear system, so that the preconditioner is always based on the most recently solved system in the sequence (1.2). We will report results using both approaches.

We have experimented with various strategies for selecting the correction pairs to be saved. In analogy with nonlinear optimization we can simply save the last m pairs. But a strategy that is more effective in some cases is to save the correction pairs at regular intervals. Suppose that $m > 1$ and that ncg denotes the number of CG iterations performed during the solution of the first linear system. If we define $\nu = \lfloor ncg/(m-1) \rfloor$, then we would like to save the pairs $\{s_k, y_k\}$, for $k = 0, \nu, 2\nu, , \ldots, (m-1)\nu$. Even though this cannot be done in practice since the number ncg of CG iterations is not known beforehand, in Appendix 1 we describe an algorithm that dynamically stores the correction pairs so that they are as evenly distributed as possible. This algorithm requires no extra storage or computation, and in our tests gives essentially the same results as saving the correction pairs at exactly uniform intervals.

Following the L-BFGS algorithm, we will always choose the initial matrix $\bar{H}$ in (2.3) to be

$$\bar{H} = \frac{s_l^T y_l}{y_l^T y_l} I, \tag{2.4}$$

where l denotes the last correction pair generated in the CG cycle.

We conclude this section by noting that limited memory updating is flexible enough to accommodate information generated at any stage during the solution of the sequence of problems (1.1) or (1.2). In particular the preconditioner could contain correction pairs corresponding to different linear systems, but we will not explore this possibility here.

3. Application to the Hessian-free Newton Method

In this section we investigate the effectiveness of the automatic preconditioner within a Hessian-free Newton method for solving the unconstrained optimization problem

$$\text{minimize} \quad f(x). \tag{3.1}$$

Here f is a twice continuously differentiable function of n variables. Our experiments will be performed using Nash's implementation [16, 17] of the Hessian-free Newton algorithm, which we now briefly review.

Given the current estimate x_k of the optimal solution of (3.1), we generate a search direction p_k by approximately minimizing the quadratic model

$$Q_k(p_k) = \nabla f(x_k)^T p_k + \frac{1}{2} p_k^T \nabla^2 f(x_k) p_k. \tag{3.2}$$

The new iterate is then defined to be $x_{k+1} = x_k + \alpha_k p_k$, where the step size α_k is computed by means of a line search procedure; in our code we used the line search routine developed by Moré and Thuente [14].

The approximate solution of (3.2) is obtained by applying the CG method to the system

$$\nabla^2 f(x_k) p = -\nabla f(x_k), \tag{3.3}$$

starting from the initial guess $p_k^{(0)} = 0$, and terminating if a direction of negative curvature is detected or if the following stopping test is satisfied

$$i \left(1 - \frac{Q_k(p_k^{(i-1)})}{Q_k(p_k^{(i)})} \right) \leq 0.5, \tag{3.4}$$

where $\{p^{(i)}\}$ denotes the sequence of CG iterates. This test aims to terminate the CG iteration when the reduction in the quadratic model is judged to be so small that the improvement in the quality of the search direction is not likely to offset the cost of computing it.

In the Hessian-free Newton method [20, 17] it is assumed that the elements of the Hessian matrix $\nabla^2 f$ are not available. One must therefore compute the matrix-vector products required by the CG iteration by automatic differentiation, or as will be done in our tests, approximate them by finite-differences,

$$\nabla^2 f(x_k) v \approx \frac{\nabla f(x_k + hv) - \nabla f(x_k)}{h}, \tag{3.5}$$

where $h = (1 + ||x_k||_2)\sqrt{\epsilon_M}$, and ϵ_M denotes unit roundoff. The computational cost of a matrix-vector product in the CG iteration therefore equals the cost of a gradient evaluation. (Current software for automatic differentiation will normally be at least as expensive as finite-differences).

We make use of the automatic preconditioner as follows. During the first iteration of the Hessian-free Newton method we apply the unpreconditioned CG method to compute the first search direction, and build a quasi-Newton preconditioner $H(m)$, as discussed in §2, and using the uniform sampling strategy described in the Appendix. This preconditioner is used to compute the next search direction, and during this second iteration we construct a new preconditioner $H(m)$. This process is repeated every iteration of the Hessian-free Newton method: the search direction is always computed by means of the preconditioned CG method using the preconditioner constructed at the previous iteration. The starting point for every CG run is $p_k^{(0)} = 0$.

3.1. Experiments with Selected Problems

We begin by focusing on the 5 problems listed in Table 1 whose Hessian matrices possess 5 distinct classes of eigenvalue distributions. Liu, Marazzi and Nocedal [11], describe these eigenvalue distributions and how they evolve as the iterates approach the solution. Other characteristics of the 5 problems are discussed in Nash and Nocedal [18]. The number of variables in all these test problems is $n = 100$. All the numerical results reported in this paper were performed on a DEC ALPHA2100 workstation with 128 Mb of main memory, and using double precision FORTRAN; machine accuracy is approximately 10^{-16}.

The optimization iteration was terminated when

$$||\nabla f(x_k)||_2 \leq 10^{-5} \max\{1, ||x_k||_2\} \tag{3.6}$$

The results are summarized in Table 1, for various values of the memory parameter m in the preconditioner. We report the number of iterations (iter) of the Hessian-free Newton method, the number of function and gradient evaluations (fg) performed during the line search, and the number of CG iterations (cg). Recall that every iteration of the CG method requires one gradient evaluation.

Our main interest in these results lies in the number of CG iterations; the number of function/gradient evaluations in the line search and the number of iterations of the Hessian-free Newton method vary somewhat randomly due to the nonlinearities in the problem and due to the inner termination test (3.4). We observe from Table 1 that a substantial reduction in the number of CG iterations was obtained, in all problems, for $m = 8$.

No further gains were achieved by increasing m to 16 (or beyond). The reason for this is partly explained by Table 2 which reports the average number of CG iterations per Newton iteration. Note that since the preconditioner makes use of the correction pairs generated by the CG method, and since Table 2 shows that the average number of CG iterations is small, increasing the storage beyond 10 corrections will have no effect most of the time. This explains, in particular, why for several problems the results for $m = 8$ and $m = 16$ are identical.

	m=0			m=4			m=8			m=16		
problem	iter	fg	cg	iter	fg	cg	iter	fg	cg	iter	fg	cg
cvar-2	51	52	871	51	52	760	48	49	578	41	42	610
penalty-3	25	29	142	21	25	71	20	24	72	20	24	72
tridiag	24	25	166	24	25	119	20	21	83	20	21	83
QOR	13	14	52	13	14	32	13	14	32	13	14	32
SQRT(2)	37	43	640	34	43	416	31	37	380	36	42	359
Total	150	163	1871	143	159	1398	132	145	1145	130	143	1156

Table 1. Performance of the Hessian-free Newton method for various values of the memory parameter m in the preconditioner.

problem	m=0	m=4	m=8	m= 16
cvar-2	17.0	14.9	12.0	14.9
penalty-3	5.7	3.4	3.6	3.6
tridiag	6.9	5.0	4.2	4.2
QOR	4.0	2.5	2.5	2.5
SQRT(2)	17.3	12.2	12.3	10.0

Table 2. Average number of CG iterations per Newton step for the results of Table 1.

Table 1 suggests that the preconditioner is successful. To quantify its effectiveness in a more controlled setting, we perform the following tests using problems cvar-2 and penalty-3 (similar results are obtained with the other test problems). For each function we select an intermediate iterate generated by the Hessian-free Newton method, and at that point compute the Hessian matrix using finite-differences. This iterate is selected so that the Hessians are positive definite at that point. For each of the two problems, we solved the 51 linear systems

$$Ax = b_i, \quad i = 0, \ldots, 50, \tag{3.7}$$

where A denotes the Hessian matrix and where the right hand side vectors b_i were randomly generated with components in the interval $[0, 1]$. We solve the first system $Ax = b_0$ using unpreconditioned CG, and construct preconditioners $H(m)$ for various values of m. We then solved the remaining systems $Ax = b_i$, $i = 1, \ldots, 50$ using the preconditioned CG method. In all cases, the starting point was $x_0 = 0$ and the CG iteration was terminated by means of the residual test recommended in [2]:

$$||r_k||_\infty \leq (||A||_\infty ||x_k||_\infty + ||b||_\infty)\text{TOL}. \tag{3.8}$$

In Table 3 we report the results for two values of the parameter TOL.

We observe that for a tight tolerance, $\text{TOL}_1 = 10^{-7}$, the benefit of the preconditioner can be modest, as in the problem cvar-2, but that for the relaxed tolerance $\text{TOL}_2 = 10^{-3}$ the savings in the number of CG iterations are substantial. These results are typical of

	cvar-2		penalty-3	
m	TOL_1	TOL_2	TOL_1	TOL_2
0	61	37	26	12
4	71	7	22	5
8	54	6	15	2
12	52	3	15	2
18	49	3	15	2
20	46	1	15	2

Table 3. Solving systems with a fixed coefficient matrix and multiple right hand sides. Number of CG iterations for two tolerances, $\text{TOL}_1 = 10^{-7}$, $\text{TOL}_2 = 10^{-3}$.

what we have observed using other coefficient matrices and right hand side vectors. They suggest that the quasi-Newton preconditioner is well suited in settings similar to that of the Hessian-free Newton method, where the stopping test for the CG iteration often demands low accuracy.

In all these tests we have reported only the number of CG iterations, and not computing times. This is because our objective in introducing the automatic preconditioner is to reduce the number of gradient evaluations which often render Hessian-free Newton methods impractical. We should mention, however, that the cost of applying the preconditioner, which is $4mn$ floating point operations, may constitute a substantial portion of the optimization process if the evaluation of the gradient is inexpensive. We will return to this point in the next section.

As mentioned in §2, the preconditioner saves the correction pairs at uniform intervals throughout the CG run. If instead we build the preconditioner by using the last m pairs of the CG iteration, the results described in this section would not be quite as good, but overall similar, to the ones obtained with the uniform sampling technique. In the next section, however, we will report experiments in which saving the last m correction pairs is a significantly inferior strategy.

3.2. Extensive Tests

We now test the efficacy of the automatic preconditioner by solving a set of unconstrained problems from the CUTE collection [4]. We will use this experiment to report on one of the many variants of the sampling techniques we have tried. In addition to collecting m correction pairs during the CG cycle using the sampling technique, we will also store the correction pair produced by the outer iteration of the optimization algorithm,

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

The results obtained with this strategy are shown in Table 4. Without storing the outer correction pair the results are slightly less successful, and will not be reported here.

		m=0			m=4			m=8		
problem	n	iter	fg	cg	iter	fg	cg	iter	fg	cg
ARWHEAD	1000	6	7	15	6	7	12	6	7	12
BDQRTIC	100	16	17	69	15	16	52	17	18	52
BROYDN7D	1000	114	267	1061	105	273	634	106	273	712
CRAGGLVY	1000	20	20	98	22	22	68	23	23	74
DIXMAANA	1500	6	6	14	6	6	13	6	6	13
DIXMAANE	1500	22	23	266	22	23	198	23	24	186
DIXMAANG	1500	21	21	209	27	29	200	29	31	182
DIXMAANH	1500	21	21	207	26	26	184	24	24	158
DIXMAANI	1500	-2	-2	-2	64	65	2616	63	64	2572
DIXMAANL	1500	-2	-2	-2	227	229	2749	213	215	2586
DQDRTIC	1000	6	6	16	6	6	13	6	6	13
DQRTIC	500	-2	-2	-2	22	24	45	22	24	45
EIGENALS	110	38	39	233	36	39	218	29	32	147
EIGENBLS	110	-2	-2	-2	124	193	1020	135	198	1039
EIGENCLS	462	-2	-2	-2	128	177	1491	121	172	1528
ENGVAL1	1000	11	11	25	9	9	18	9	9	18
FREUROTH	1000	11	17	28	11	14	22	11	14	22
GENROSE	500	-2	-2	-2	366	669	1982	365	646	2000
MOREBV	1000	5	6	70	5	6	67	5	6	68
NONDQUAR	100	56	67	323	62	101	392	56	91	320
PENALTY1	1000	-3	-3	-3	41	46	84	41	46	84
PENALTY3	100	-3	-3	-3	23	31	59	23	31	59
QUARTC	1000	-3	-3	-3	24	27	49	24	27	49
SINQUAD	1000	67	84	248	22	35	58	22	35	58
SROSENBR	1000	9	10	22	9	10	19	9	10	19
TQUARTIC	1000	3	3	8	3	3	7	3	3	7
TRIDIA	1000	46	46	1306	35	35	570	34	34	575

Table 4. Performance of the Hessian-free Newton method on a set of problems from the CUTE collection. The code -2 indicates that more than 3000 CG iterations were performed. The code -3 indicates that the line search routine performed more than 20 iterations without decreasing the objective function.

		m=0				m=4				m=8			
problem	n	iter	fg	cg	cpu	iter	fg	cg	cpu	iter	fg	cg	cpu
MinSurA	2500	16	19	178	18.9	15	19	101	14.1	15	18	103	15.5
G-L 2D	400	136	149	2948	82.9	76	93	1716	54.8	58	65	1308	45.9

Table 5. Performance of the Hessian-free Newton method on two problems of MINPACK-2 collection of problems. CPU time is reported in seconds.

In these experiments the preconditioner is successful, not only in reducing the total number of CG iterations, but also in improving the reliability of our optimization method.

We conclude our numerical study in the optimization setting by considering two problems from the MINPACK-2 collection [1]. The preconditioner was the same as the one used to generate the results in Table 4. We now also report CPU time to illustrate the effect of the preconditioner on the Newton iteration. The results are presented in Table 5.

4. Experiments with Finite Element Matrices

Our numerical experiments with nonlinear optimization test problems suggest that the quasi-Newton preconditioner holds much promise. To continue our evaluation of its performance, we would like to test it on matrices that have different eigenvalue distributions from the ones studied so far, and that are representative of an important class of applications. To this end we have selected several linear systems arising in the finite element models of Belytschko et al [3]. The first two matrices used in our experiments, A_{1_0}, A_{1_1}, were obtained from a 1-dimensional model consisting of a line of 2-node elements with support conditions at both ends, and a linearly varying body force. A_{1_0} has dimension $n = 50$ and A_{1_1} has dimension $n = 451$. The right hand side vector in these systems, which we denote by c_0, is defined by

$$c_0^1 = c_0^n = 0, \quad c_0^i = i/(n-1) \times 10^2, \quad i = 2, \ldots, n-1, \tag{4.1}$$

where superscripts indicate components of a vector.

The third matrix used in our tests, A_{2_0}, is the stiffness matrix from a two-dimensional finite element model of a cantilever beam. The beam is fixed at one end, and a shear load is applied at the other end. The finite element mesh consists of an even array of elements in the x and y-coordinates [3]. The right hand side vector for this two-dimensional model will be denoted by d_0; it has zeros in all positions except that

$$d_0^{34} = d_0^{68} = d_0^{102} = d_0^{136} = d_0^{170} = -8000. \tag{4.2}$$

We also generated 5 matrices $A_{2_1}, \ldots, A_{2_5}$ by perturbing the mesh for the cantilever model A_{2_0} along one of the coordinate directions. The size of the perturbation increases linearly with every new matrix in the sequence: it is 1% in A_{2_1} and 10% in A_{2_5}.

The characteristics of the matrices are shown in Table 6, where λ_{min} and λ_{max} denote their extreme eigenvalues.

problem	origin	n	λ_{min}	λ_{max}
A_{1_0}	1D	50	1.0	$.20 \times 10^{10}$
A_{1_1}	1D	451	1.0	$.18 \times 10^{11}$
A_{2_0}	2D	170	1.0	$.13 \times 10^{9}$
A_{2_1}	2D pert	170	1.0	$.13 \times 10^{9}$
A_{2_2}	2D pert	170	1.0	$.14 \times 10^{9}$
A_{2_3}	2D pert	170	1.0	$.14 \times 10^{9}$
A_{2_4}	2D pert	170	1.0	$.14 \times 10^{9}$
A_{2_5}	2D pert	170	1.0	$.15 \times 10^{9}$

Table 6. Characteristics of the finite element test problems.

The first matrix, A_{1_0}, has one eigenvalue of size $\lambda = 1$, one of size $\lambda = .2 \times 10^7$, and the rest are distributed in a wide gap and cluster near the largest eigenvalue, $\lambda = .2 \times 10^{10}$. The second matrix A_{1_1} has a similar eigenvalue distribution, except that the smallest eigenvalue $\lambda = 1$ has multiplicity two. The matrix A_{2_0} from the two dimensional model has a cluster of 10 eigenvalues at $\lambda = 1$; the next eigenvalue is located at $\lambda = .54 \times 10^4$, and the rest form several clusters between $\lambda = 10^7$ and $\lambda = .13 \times 10^9$. We illustrate the eigenvalue distributions of these test matrices in Figure 1.

In the experiments with finite element matrices reported next, the CG iteration was terminated using the residual test (3.8), where the value of the parameter TOL will be specified later on. The preconditioner was constructed using the uniform sampling strategy described in the Appendix.

4.1. Multiple Right Hand Sides

We first test the efficiency of the quasi-Newton preconditioner in solving a sequence of problems (1.1), in which the coefficient matrix is constant but the right hand side varies. To do so, we applied the unpreconditioned CG method to the first system $Ax = b_0$. The information generated during this run was used to construct 5 quasi-Newton preconditioners $H(m)$, for $m = 4, 8, 12, 16, 20$, as described in §2. For each preconditioner $H(m)$, we solved the remaining systems $Ax = b_i$, $i = 1, \ldots, 50$, using the preconditioned CG method. The first right hand side vector b_0 was defined as c_0 or d_0, depending on whether the matrices correspond to the one or two-dimensional models (see (4.1)-(4.2)). The other 50 right hand sides, $b_1, \ldots, b_{50}$ were obtained according to the following recursion, which starts with $j = 0$. Using b_j as a "seed", we obtain b_{j+1} by introducing perturbations of size $\pm 5\%$, with random signs, to each of the nonzero components of b_j.

The results are presented in Table 7. We report the average number of CG iterations (rounded to the nearest integer) needed to meet the stopping test (3.8) with TOL $= 10^{-7}$. We present results for two initial points, $x_0 = 0$ and $x_0 = 10^2 e$, where $e = (1, 1, \ldots, 1)^T$.

We observe that the preconditioner is successful in reducing the number of CG iterations in both the 1-dimensional and 2-dimensional models. To illustrate how the preconditioner

	A_{1_0}		A_{1_1}		A_{2_0}	
	$x_0 = 0$	$x_0 = 10^2 e$	$x_0 = 0$	$x_0 = 10^2 e$	$x_0 = 0$	$x_0 = 10^2 e$
m	iter	iter	iter	iter	iter	iter
0	49	25	447	225	56	93
4	43	22	291	185	48	68
8	23	12	126	89	26	56
12	16	6	125	80	28	36
16	12	4	62	41	27	30
20	12	5	63	41	21	24

Table 7. Results for finite element matrices using multiple right hand sides. The table reports average number of CG iterations for 50 runs, using different values of the memory parameter m, and for two different initial points.

transforms the spectrum of the coefficient matrix A_{1_0}, we plot in Figure 1 the eigenvalues of $H(m)A_{1_0}$, for $m = 16, 49$, as well as the spectrum of the original matrix A_{1_0}. We note that even though $H(16)A_{1_0}$ is only slightly better conditioned than A_{1_0}, its eigenvalues are more tightly clustered. We also observe that the condition number of $H(49)A_{1_0}$ is $.52 \times 10^7$ with just one eigenvalue $\lambda = 0.19 \times 10^{-8}$ and a cluster of 49 eigenvalues $\lambda = 1$; this is expected, given the properties of quasi-Newton updating and the fact that A_{1_0} is of dimension 50.

It is natural to ask whether the preconditioner provides a reduction in cpu time – and not just in CG iterations – in these finite element test problems. It turns out that since our test matrices are very sparse, the cost of applying the preconditioner is too high to offset the reduction in the number of CG iterations in these experiments. More specifically, the product Av, which is the most computationally expensive part of the unpreconditioned CG method, requires approximately $3n$ multiplications for the 1-dimensional model and $14n$ multiplications for the 2-dimensional model. In contrast, the product of the preconditioner $H(m)$ and a vector requires $4mn$ multiplications independently of the matrix structure. As a result, one is not able to obtain reductions in cpu time for any of the values of m listed in Table 7. Nevertheless these results indicate that for matrices having the same eigenvalue distribution as our test matrices, but with a substantial number of nonzero elements, significant reductions in computing time can be achieved with the quasi-Newton preconditioner. For the rest of the paper we will continue to assume that the cost of computing Av is much higher than the cost of applying the quasi-Newton preconditioner, and will report only the number of CG iterations.

4.2. Slowly Varying Systems

Next we consider the family of problems $A_{2_k} x = b_k$, for $k = 0, 1, \ldots, 5$, using the perturbations of the 2-dimensional finite element matrix A_{2_0}. We solve the first system $A_{2_0} x = b_0$ using unpreconditioned CG, and construct 5 quasi-Newton preconditioners $H(m)$ for $m = 4, 8, 12, 16, 20$. Each of these is used to solve the five remaining systems using the

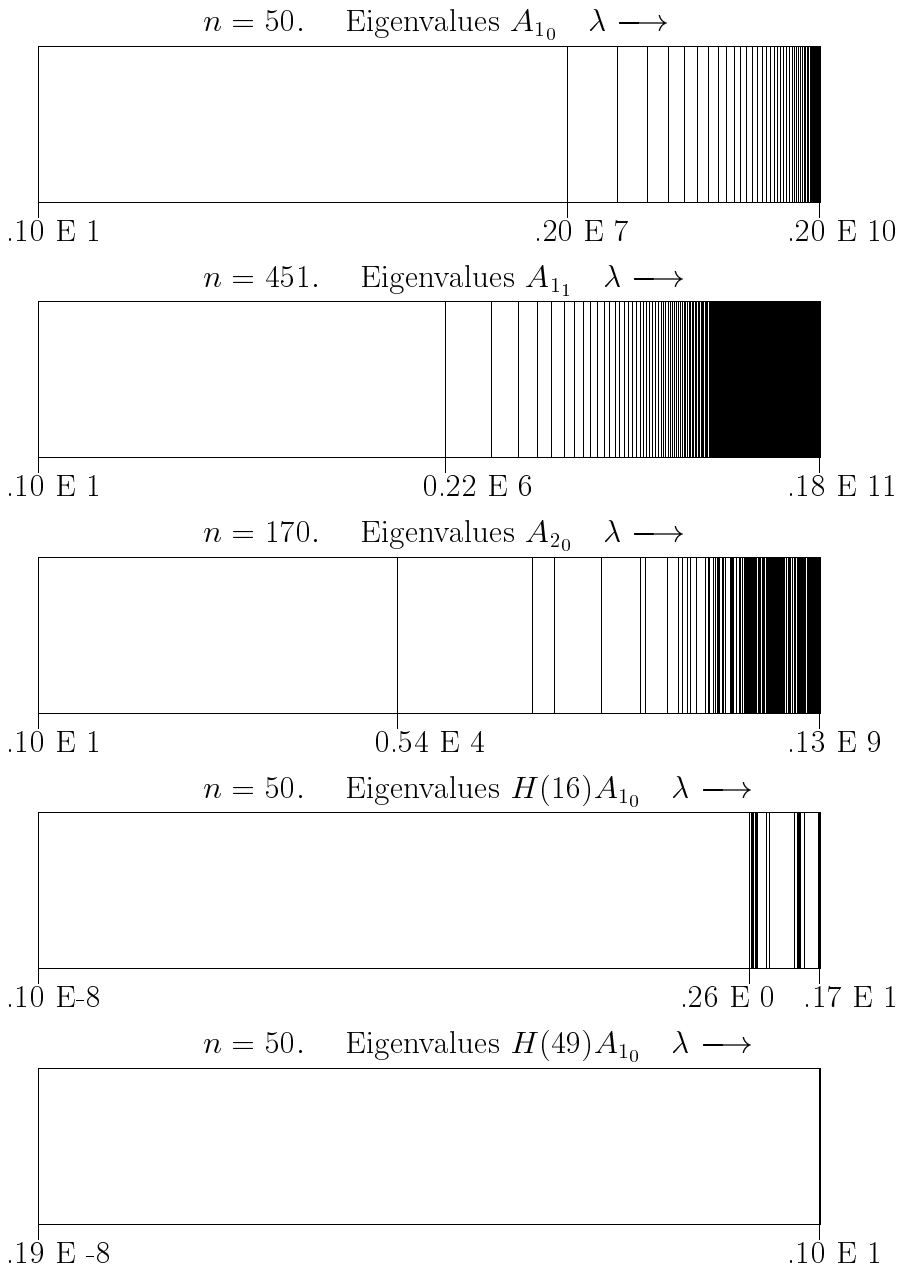

Figure 1: Eigenvalue distributions of three finite element matrices, and the effect of preconditioning on A_{1_0}

	$x_0 = 0$						$x_0 = 10^2 e$					
m	A_{2_0}	A_{2_1}	A_{2_2}	A_{2_3}	A_{2_4}	A_{2_5}	A_{2_0}	A_{2_1}	A_{2_2}	A_{2_3}	A_{2_4}	A_{2_5}
4	56	50	51	51	52	53	93	70	71	72	73	75
8	56	27	28	29	31	31	93	58	59	61	62	63
12	56	20	22	22	26	27	93	37	37	38	39	40
16	56	19	21	22	23	24	93	32	32	32	34	36
20	56	18	19	20	26	28	93	25	26	26	28	30

Table 8. Results on a sequence of slowly varying linear systems arising from the 2-dimensional finite element model. The table presents the number of iterations of the preconditioned CG method to solve each of the systems. The starting point for solving all the systems is given by x_0.

	$x_0 = 0$						$x_0 = 10^2 e$					
m	A_{2_0}	A_{2_1}	A_{2_2}	A_{2_3}	A_{2_4}	A_{2_5}	A_{2_0}	A_{2_1}	A_{2_2}	A_{2_3}	A_{2_4}	A_{2_5}
4	56	41	41	46	47	46	93	4	5	4	4	5
8	56	20	24	25	26	26	93	4	4	7	4	5
12	56	17	18	19	19	20	93	5	6	8	9	9
16	56	17	18	15	21	18	93	5	7	8	9	10
20	56	17	18	15	19	17	93	5	5	7	8	8

Table 9. A variation of the results in Table 8. The initial point for solving system $A_{2_k} x = b_k$ is now taken as the solution of previous system, $A_{2_{k-1}} x = b_{k-1}$. The initial point for the first system $A_{2_0} x = b_0$ is given by x_0.

preconditioned CG method. The CG iteration was stopped by (3.8) with TOL$= 10^{-7}$. The first right hand side vector is defined to be d_0 (see (4.2)), and the remaining right hand sides $d_1, \ldots, d_5$ were constructed as before by adding, every time, perturbations of $\pm 5\%$ to the non-zero elements in each of the vectors in the sequence $\{d_j\}$.

In the first set of experiments, reported in Table 8, the same starting point x_0 was used for all the systems $A_{2_k} x = b_k$. We experimented with two choices for this starting point, $x_0 = 0$ and $x_0 = 10^2 e$. In the second set of experiments, reported in Table 9, the initial point for solving each system $A_{2_k} x = b_k$ was chosen to be the solution of previous system, $A_{2_{k-1}} x = b_{k-1}$. Recall that the system A_{2_0} is always solved by unpreconditioned CG.

Table 8 shows that the preconditioner is effective. The fact that the number of iterations increases slightly as we move along a row of the table is not surprising. Since the preconditioner was generated from the first matrix A_{2_0}, and the matrices A_{2_k} differ more and more from it as the subscript k increases, the preconditioner becomes "older" for each new system. Table 9 indicates that using the solution of $A_{2_{k-1}} x = b_{k-1}$ as the initial point for the new system $A_{2_k} x = b_k$, has been advantageous.

We repeated the tests of Tables 8 and 9 refreshing the preconditioner after every solution. To be more precise during the solution of each system $A_{2_k} x = b_k$ we construct a

	$x_0 = 0$						$x_0 = 10^2 e$					
m	A_{2_0}	A_{2_1}	A_{2_2}	A_{2_3}	A_{2_4}	A_{2_5}	A_{2_0}	A_{2_1}	A_{2_2}	A_{2_3}	A_{2_4}	A_{2_5}
4	56	50	44	51	55	49	93	70	73	67	73	68
8	56	27	34	41	33	33	93	58	49	52	47	51
12	56	20	31	37	24	31	93	37	41	42	43	37
16	56	19	31	21	37	27	93	32	32	32	34	36
20	56	18	26	23	32	22	93	25	43	30	31	32

Table 10. A variation of the results given in Table 8. A new preconditioner is now computed after every solution. The right hand side vectors were the vectors b_i. The starting point for solving all the systems is given by x_0.

	$x_0 = 0$						$x_0 = 10^2 e$					
m	A_{2_0}	A_{2_1}	A_{2_2}	A_{2_3}	A_{2_4}	A_{2_5}	A_{2_0}	A_{2_1}	A_{2_2}	A_{2_3}	A_{2_4}	A_{2_5}
4	56	41	38	48	40	59	93	4	3	1	1	2
8	56	20	37	32	34	33	93	4	2	1	2	1
12	56	17	26	32	27	39	93	5	5	5	4	2
16	56	17	24	24	28	26	93	5	4	6	3	2
20	56	17	19	14	19	16	93	5	4	4	4	2

Table 11. A variation of the results given in Table 9. A new preconditioner is now computed after every solution. The initial point for solving system $A_{2_k} x = b_k$ is taken as the solution of system $A_{2_{k-1}} x = b_{k-1}$. The initial point for the first system $A_{2_0} x = b_0$ is given by x_0.

preconditioner, and use it to solve the next system $A_{2_{k+1}} x = b_{k+1}$ (as in the Hessian-free Newton method). We have made an exception to this strategy when the CG method required only one or two iterations to meet the stopping test, since building a preconditioner with $m = 1, 2$ is not useful. In this case we use the preconditioner most recently generated. The results are given in Tables 10 and 11.

The results of Tables 10 and 11 are better than those of Tables 8 and 9, particularly in that there is no longer a trend for the number of CG iterations to increase as we move along a row of the table. Nevertheless the gains are less significant than one would expect. We should note that when the preconditioner is built during an unpreconditioned CG run, the number of CG iterations is larger, and the pairs $\{s_k, y_k\}$ represent a better sample than that obtained during a preconditioned CG run. Indeed, if the preconditioner is so effective that the number of CG iterations is very small, then collecting information from this run may not be advantageous, as we mentioned above. Our conclusion is that the decision of when to refresh the preconditioner is not simple, and dynamic strategies that balance the currency of the information with the amount of information available could be quite effective. We will, however, not pursue this question here.

Tables 8-11 indicate that using the previous solution as the starting point for a new run

	b random		b random, $b^1 = b^n = 0$	
m	TOL $= 10^{-7}$	TOL $= 10^{-9}$	TOL $= 10^{-7}$	TOL $= 10^{-9}$
0	50	50	48	49
4	7	39	45	46
8	6	34	41	42
12	6	29	37	38
16	6	27	34	34
20	5	25	29	30

Table 12. Constructing the preconditioner using the last m correction pairs. Number of CG iterations for two types of right hand side vectors b and for two levels of accuracy TOL.

(a "hot start") sometimes, but not always, leads to a substantial reduction of CG iterations. We should also point out that the results for $x_0 = 0$ in Table 11 show that the hot start benefits from preconditioning, as can be seen by reading the results one column at a time. But for $x_0 = 10^2 e$ in Table 9, preconditioning does not help the hot start strategy.

4.3. Comparing Sampling Strategies

We will now perform some tests to compare the strategy of saving correction pairs at uniform intervals during the CG run, with that of saving the last m pairs. In the first experiment we use the matrix A_{1_0} from the 1-dimensional finite element model, and solve systems of the form (1.1) where the matrix is fixed and the right hand sides vary. The initial point was $x_0 = 0$ and the right hand sides were chosen to have random components in the interval $[0, 1]$. The preconditioner is first constructed using the last m iterations of the CG method. The results are presented in the 2nd and 3rd columns of Table 12, for two values of the tolerance TOL in (3.8). It is remarkable that the preconditioner is extremely effective when TOL$= 10^{-7}$, which is a fairly tight accuracy, but that it gives only modest gains when TOL$= 10^{-9}$. We then modified the right hand sides by setting their first and last components to zero. The results, which are markedly different, are given in the last two columns of Table 12.

We can explain these results by considering the properties of the matrix A_{1_0}, which is given by

$$A_{1_0} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & a & -a/2 & 0 & \cdots & 0 & 0 \\ 0 & -a/2 & a & -a/2 & \cdots & 0 & 0 \\ 0 & 0 & -a/2 & a & \cdots & 0 & 0 \\ & & & & \vdots & & \\ 0 & 0 & 0 & 0 & \cdots & -a/2 & a \end{bmatrix},$$

where $a = 10^{-9}$. Since the first row is e_1^T, the first component of the solution x equals the first component b^1 of the right hand side vector. It is not difficult to show that since all the

	b random		b random, $b^1 = b^n = 0$	
m	TOL $= 10^{-7}$	TOL $= 10^{-9}$	TOL $= 10^{-7}$	TOL $= 10^{-9}$
0	50	50	48	49
4	10	34	36	44
8	25	35	37	33
12	17	22	19	24
16	12	18	14	19
20	15	18	13	16

Table 13. Constructing the preconditioner sampling m correction pairs. Number of CG iterations for two types of right hand side vectors b and for two levels of accuracy TOL.

entries in b are not greater than 1, all other components of x are of order 10^{-6}. Therefore for these random right hand side vectors we can expect the solutions to be closely aligned with the first coordinate direction e_1. Since the preconditioner is able to incorporate the curvature along e_1, it forces the CG iteration to immediately point towards the solution. As a result the CG iteration will terminate quickly if the required accuracy is not too high. These are the most favorable conditions for the automatic preconditioner. But if the tolerance is set to be TOL$= 10^{-9}$, it will be necessary that the components of the solution along the other coordinate directions be estimated well, and the limited memory preconditioner is only able to provide some of the needed information.

The solution will no longer be closely aligned with e_1 if the first component of the right hand side vector is set to zero. One can show that in this case the solution will have significant components along *all* the coordinate directions, except for the first component which is zero. The problem thus becomes particularly difficult for limited memory preconditioning. This is confirmed by the last two columns of Table 12 which show very modest gains in performance. Note also that the performance is now insensitive to the stopping tolerance.

In Table 13 we repeat the tests reported in Table 12, but using a uniform sampling strategy. The latter clearly performs better than saving the last m pairs, except for the first case (b random TOL$= 10^{-7}$), which as we have explained, represents a special case.

To continue our comparison of sampling strategies, we repeat in Table 14 the experiments of Table 7 with the two-dimensional finite element matrix A_{2_0}, using two different starting points. We compare the strategy of saving the last m pairs ("last") with that of uniform sampling. It is clear that the latter performs much better in this experiment.

Our computational experience, both in the optimization setting and in finite element calculations, is that saving the last m corrections usually gives comparable performance to the uniform sampling technique. But as we have just shown, there are cases when uniform sampling is superior. It is difficult to provide theoretical arguments in favor of either strategy, but we now report the results of controlled tests that further support the uniform sampling technique.

	$x_0 = 0$		$x_0 = 10^2 e$	
m	last	uniform	last	uniform
0	56	56	93	93
4	74	48	90	68
8	71	26	87	56
12	56	28	83	36
16	44	27	77	30
20	43	21	72	24

Table 14. Average number of iterations of the CG method for the matrix A_{2_0} and multiple right hand sides. Comparison of two sampling strategies in the formation of the preconditioner: saving the last m iterations and sampling at uniform intervals. Results for two starting points are given.

4.4. On the Sample Size.

When the number of correction pairs available to form the preconditioner is small, the two strategies (uniform sampling and using the last m corrections) will clearly give similar results. Therefore in the following tests we will force the CG algorithm to perform an increasingly large number of iterations, and observe the effect that this has on the quality of the preconditioner.

More specifically we study whether the preconditioner benefits from having a larger sample of corrections to choose from, for a given amount of memory m. In the tests described next, we will consider the solution of a sequence of finite element systems with multiple right hand sides. We will fix the value of m, apply the unpreconditioned CG for a fixed number `maxCG` of CG iterations to the first system in the sequence, and build the preconditioner using the sampling technique. We then solve the rest of the linear systems using this preconditioner, terminating the CG iteration by means of (3.8). To study the benefit of a larger sample size, we repeat this test for various values of `maxCG`.

The results are given in Tables 15-17. Note that, for a given value of m, the preconditioners differ in that they use an increasingly wider sample of CG iterations. We observe that if the amount of memory is small ($m = 4$) the quality of the preconditioner appears to be independent of the sample size `maxCG`. But for larger values of m the sample size has a beneficial effect.

5. Final Remarks

We have presented a quasi-Newton preconditioner for accelerating the conjugate gradient method, when this is applied to a sequence of linear systems with positive definite coefficient matrices. Our numerical experiments indicate that the preconditioner may be useful when the coefficient matrices A are not very sparse, or when A is not explicitly available and products of A times vectors are expensive to compute. The motivation for this work arose from the desire to accelerate the CG iteration used in a Hessian-free Newton

maxCG	$m = 4$	$m = 8$	$m = 16$
10	42	41	41
20	38	33	32
30	31	26	22
40	31	22	16
50	43	24	13

Table 15. Results for test matrix A_{1_0} using multiple right hand sides. The table reports the number of iterations to achieve convergence for 50 runs, using different values of the memory parameter m and of the CG iteration limit maxCG. Initial point $x^{(0)} = 0$.

maxCG	$m = 4$	$m = 8$	$m = 16$
250	245	209	198
300	254	192	159
350	246	160	113
400	275	114	69
450	287	125	60

Table 16. The experiment reported in Table 15 using the test matrix A_{1_1}.

maxCG	$m = 4$	$m = 8$	$m = 16$
20	57	52	50
30	49	43	41
40	32	25	22
50	48	26	19
60	48	26	19

Table 17. The experiment reported in Table 15 using the test matrix A_{2_0}.

method for nonlinear optimization, and in that context the new preconditioner appears to provide substantial savings. Our experiments with finite element models suggest that the preconditioner may prove to be useful in other areas of application, but more research is required to establish this firmly.

We have experimented with several other strategies for selecting the correction pairs. One idea that deserves to be mentioned is to use the m pairs with the smallest Rayleigh quotient,

$$\frac{s_i^T y_i}{\|s_i\|^2}.$$

Even though this strategy has not proved to be more successful in our tests than the other selection schemes described in the paper, it may be effective in some areas of application.

Acknowledgements. We would like to thank Ted Belytschko and Mark Flemming for providing a Matlab code that generates the finite element models used in §4, Eric Schwabe for a very useful discussion on the algorithm appearing in the Appendix, and Richard Waltz for running the tests reported in Table 5. We would also like to thank a referee for suggesting the experiments reported in Tables 15-17.

6. Appendix

We now present a formal description of the sampling algorithm (mentioned in §2) that collects the pairs $\{s_k, y_k\}$ as uniformly as possible, with the restriction that at most m pairs be stored at any stage. We denote the set of correction pairs that have been stored as $\mathcal{P}$. We will assume that m is an even number since this simplifies the algorithm and is not restrictive in practice.

The sampling algorithm runs parallel to the CG method. Once a pair $\{s_k, y_k\}$ has been computed by the CG method, the sampling algorithm examines the iteration index k and decides if the pair should be included in $\mathcal{P}$. When a new pair is accepted, the algorithm checks the available space, and if the number of pairs in $\mathcal{P}$ is m, then a pair is chosen to leave $\mathcal{P}$. The algorithm is started by inserting into $\mathcal{P}$ the first m pairs generated by the CG process. After this, the entering and leaving pairs are chosen to keep an *almost* uniform distribution at any time.

Algorithm SAMPLE

Choose an even number m; set $k \leftarrow 0$ and $cycle \leftarrow 1$.

REPEAT:

Starting

while $k < m$,

- **get** $\{s_k, y_k\}$
- Add $\{s_k, y_k\}$ to $\mathcal{P}$

- $k \leftarrow k + 1$

end while

Deletion/Insertion.

if k can be expressed as $k = (\frac{m}{2} + l - 1)2^{cycle}$ for an integer l of the form $l = 1, 2, \ldots, \frac{m}{2}$ **then**

- Store l
- Compute the subscript of the leaving pair as $k' = (2l - 1)2^{cycle-1}$
- Delete $\{s_{k'}, y_{k'}\}$ from $\mathcal{P}$
- Add $\{s_k, y_k\}$ to $\mathcal{P}$
- **if** $l = \frac{m}{2}$ **then** set $cycle \leftarrow cycle + 1$

end if

$k \leftarrow k + 1$

END REPEAT

Note that the first pair ($k = 0$) generated in the CG iteration always remains in $\mathcal{P}$. This has no particular significance, and it is easy to change the algorithm so that this is not the case.

We now discuss some properties of the sampling algorithm. After the initialization, in which the first m pairs are stored, the algorithm performs deletion an insertion operations controlled by the variable *cycle*. For a given value of *cycle*, the algorithm stores $\frac{m}{2}$ new pairs spaced by a distance of 2^{cycle}, and deletes the same of number of pairs. Deletion takes place in such a way that the space created between two consecutive pairs is 2^{cycle}. Therefore when *cycle* attains a new value, the distribution ceases to be uniform and there is a transition period during which a new uniform distribution is generated; this is achieved at the end of the second loop. It follows that the larger m is, the longer it will take to move from one uniform distribution to the next.

7. *

References

[1] B. Averick, R.G. Carter, and J.J.Moré (1991). The MINPACK-2 test problem collection. Preprint MCS-P153-0692, Argonne National Laboratory, Argonne.

[2] R. Barret, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Vorst (1994). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Philadelphia, SIAM.

[3] T. Belytschko, A. Bayliss, C. Brinson, S. Carr, W. Kath, S. Krishnaswamy, B. Moran, J. Nocedal and M. Peshkin (1997). Mechanics in the Engineering First Curriculum at Northwestern University, *Int J. Engineering Education*, vol. 13, No. 6, pp. 457-472.

[4] I. Bongartz, A.R. Conn, N.I.M. Gould, and Ph.L. Toint (1995). CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, Vol. 21, No. 1, pp. 123-160.

[5] R.H. Byrd, P. Lu, J. Nocedal and C. Zhu (1995). A limited memory algorithm for bound constrained optimization, *SIAM Journal on Scientific Computing*, 16, 5, pp. 1190–1208.

[6] R.H. Byrd, J. Nocedal and C. Zhu (1995). Towards a discrete Newton method with memory for large scale optimization, to appear in *Nonlinear Optimization and Applications*, G. Di Pillo and F. Giannessi, eds. Plenum.

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

[8] R. Fletcher (1987). *Practical Methods of Optimization*, 2nd Edition, New York, John Wiley & Sons.

[9] J.C. Gilbert and C. Lemaréchal (1989). Some numerical experiments with variable storage quasi-Newton algorithms, *Mathematical Programming* 45, pp. 407–436.

[10] P.E. Gill, W. Murray and M.H. Wright (1981). *Practical Optimization*, London, Academic Press.

[11] G. Liu, M. Marazzi and J. Nocedal. Incorporating Eigenvalue Information in Limited Memory Methods for Unconstrained Optimization, to appear.

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

[13] H. Matthies and G. Strang (1979). The solution of nonlinear finite element equations, *International Journal of Numerical Methods in Engineering* 14, pp. 1613–1626.

[14] J.J. Moré and D.J. Thuente (1994). Line search algorithms with guaranteed sufficient decrease, *ACM Transactions on Mathematical Software*, Vol. 20, No. 3, pp. 286–307.

[15] S.G. Nash (1984). Newton-type minimization via the Lanczos method, *SIAM. J. Numerical Analysis*, 21, 4, pp. 770–778

[16] S.G. Nash (1984). User's guide for TN/TNBC: FORTRAN routines for nonlinear optimization, *Report 397*, Mathematical Sciences Dept., The Johns Hopkins University.

[17] S.G. Nash (1985). Preconditioning of truncated-Newton methods, *SIAM Journal on Scientific and Statistical Computing*, 6, pp. 599–616.

[18] S.G. Nash and J. Nocedal (1991). A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization, *SIAM Journal on Optimization*, 1, 3, pp. 358-372.

[19] J. Nocedal (1980). Updating quasi-Newton matrices with limited storage, *Mathematics of Computation*, 35, pp. 773-782.

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

[21] D.P. O'Leary and A. Yeremin (1994), The linear algebra of block quasi-Newton algorithms, *Linear Algebra and Its Applications*, 212/213, pp. 153–168.