An Augmented Conjugate Gradient Method for Solving Consecutive Symmetric Positive Definite Linear Systems

Jocelyne Erhel, Frédéric Guyomarc'H

SIAM J. MATRIX ANAL. APPL.
Vol. 21, No. 4, pp. 1279–1299

AN AUGMENTED CONJUGATE GRADIENT METHOD FOR SOLVING CONSECUTIVE SYMMETRIC POSITIVE DEFINITE LINEAR SYSTEMS*

JOCELYNE ERHEL† AND FRÉDÉRIC GUYOMARC'H†

Abstract. Many scientific applications require one to solve successively linear systems $Ax = b$ with different right-hand sides b and a symmetric positive definite matrix A. The conjugate gradient method applied to the first system generates a Krylov subspace which can be efficiently recycled thanks to orthogonal projections in subsequent systems. A modified conjugate gradient method is then applied with a specific initial guess and initial descent direction and a modified descent direction during the iterations. This paper gives new theoretical results for this method and proposes a new version. Numerical experiments show the efficacy of our method even for quite different right-hand sides.

Key words. conjugate gradient, Krylov subspace, orthogonal projection

AMS subject classifications. 65F10, 65F35

PII. S0895479897330194

1. Introduction.

In this paper we are concerned with the solution of several symmetric linear systems of the form

$$Ax^{(i)} = b^{(i)},$$

where A is a $N \times N$ symmetric positive definite matrix. If A is not too large, a direct method is competitive since it only requires one factorization for several triangular solves. However, if A is large, an iterative method such as the conjugate gradient (CG) or the preconditioned conjugate gradient (PCG) is usually necessary because of memory constraints [4, 25]. The aim of this paper is to speed up the convergence of CG or PCG in order to reduce the CPU cost. The method should keep memory requirements low and vectorize or parallelize easily. Here we assume that the different right-hand sides $b^{(i)}$ are computed sequentially. The main idea is to solve the first system by CG and to recycle in the subsequent systems the Krylov subspace $\mathcal{K}_m(A, s_0)$ generated in the seed system.

1.1. Some examples.

This situation arises, for instance, when a new right-hand side depends upon previous solutions. Many applications lead to such problems.

- The asymptotic development of solutions of thick plates problems requires solving successive symmetric positive definite linear systems. The matrix is the same, but each solution at a given order of development depends upon the previous order [15].
- Nonlinear systems are often solved by an approximate Newton method where the approximate Jacobian matrix is frozen during several iterations. Each iteration then requires solving a linear system with a matrix unchanged and with a varying right-hand side which depends upon the previous iteration. Such nonlinear systems appear, for example, in implicit discretizations of differential equations.

*Received by the editors November 14, 1997; accepted for publication (in revised form) by D. P. O'Leary September 10, 1999; published electronically April 18, 2000.
http://www.siam.org/journals/simax/21-4/33019.html
†INRIA–Rennes, IRISA, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France (Jocelyne.Erhel@inria.fr, Frederic.Guyomarch@inria.fr, http://www.irisa.fr/aladin/).

- To compute generalized eigenvalues close to a given value σ, one often uses the so-called shift-and-invert method. This implies solving at each iteration of the eigenvalue solver linear systems of the form $(A - \sigma B)x^{(i)} = b^{(i)}$.
- When nonlinear least squares problems are solved by gradient methods, each iteration requires the computation of the gradient or a preconditioned gradient. If the preconditioning is a given and fixed linear operator, then computing the preconditioned gradient will involve solving a new linear system with a fixed matrix (the preconditioning matrix) and a varying right-hand side. This example arises in a wave-scattering problem studied in [3], where the gradient is preconditioned by a Laplacian.

1.2. Previous work. This problem of solving successive linear systems has been studied by several authors.

When the different right-hand sides are known a priori, then block CG methods are efficient [20]. In [6], block CG is combined with a seed projection method which generates a Krylov subspace for the so-called seed system and projects the residuals of other systems onto this subspace. Similar ideas are developed in [13, 21, 28]. This method is also extended to varying matrices [5].

Here we assume that the different right-hand sides $b^{(i)}$ are computed sequentially. A first idea is to use previous systems to derive an initial guess for the current system. In [12], the current right-hand side is A-projected onto the subspace spanned by the l previous approximate solutions, where l is a user-chosen parameter. When A is symmetric and possibly indefinite, a similar idea to the seed projection is introduced in [22] and further analyzed in [24]. This method computes an initial approximation by using a projection onto the Krylov subspace $\mathcal{K}_m(A, s_0)$. Saad [24] shows that this so-called restarted Lanczos–Galerkin method is in some sense equivalent to a block Lanczos method. Van der Vorst [30] gives a more stable formulation to compute the initial guess and extends the method to problems of the form $f(A)x = b$.

A second idea is to use information from previous systems in the iterations. In [22] and [24], the classical Lanczos procedure is modified so that the new Lanczos vector is orthogonal to $\mathcal{K}_m(A, s_0)$. This modified Lanczos method starts with the initial approximation defined previously. It appears sufficient to orthogonalize the current Lanczos vector against the last vector in $\mathcal{K}_m(A, s_0)$. This method is no longer equivalent to a block Lanczos method.

If the matrix is symmetric positive definite, the same approach can be applied to the CG method. It has the usual advantage of CG compared to Lanczos implementation. Indeed, the projected matrix is implicitly factored by recurrences at each iteration rather than explicitly at convergence. This modified CG method is applied in [23, 11, 2] in a domain decomposition framework. The small interface problem is solved by CG and several systems are solved successively because the initial problem is nonlinear. This modified CG method implements the two ideas presented above, since it uses the first Krylov subspace $\mathcal{K}_m(A, s_0)$ by computing an initial approximation and by forcing orthogonality against $\mathcal{K}_m(A, s_0)$ during iterations. But here, the current direction is orthogonalized against all previous directions to avoid loss of orthogonality due to rounding errors. Numerical experiments show a significant improvement over the classical CG method.

This idea is somewhat related to deflation methods. A deflation has been applied to CG in [19] for boundary value problems, and more recently in [26] for general symmetric positive definite systems. Similar ideas have been also applied to nonsymmetric problems using the GMRES algorithm [17, 7, 8, 1, 10, 9, 18, 16].

1.3. Contribution. In this paper, we present two algorithms called InitCG and AugCG. They are both CG forms for use when the symmetric matrix is positive definite and are derived from Lanczos-based algorithms described in [24].

InitCG computes an initial approximate solution as in [22] and [24] but uses the stable formulation proposed in [30] and uses the CG method instead of the Lanczos method. As in [24] for Lanczos, we prove that during the first iterations InitCG is in some sense equivalent to a block CG algorithm. Our numerical experiments show that InitCG is efficient if the second right-hand side is close to the first one.

AugCG includes a Krylov correction at each iteration. It is similar to the modified Lanczos method of [24] but here too uses the CG method. It is also similar to the modified CG method used in [11] for domain decomposition methods but uses a recurrence property rather than a full orthogonalization.

The main contribution of the paper is to prove an error bound giving the asymptotic rate of convergence of our AugCG algorithm. As far as we know, this result is new, although some lines of the proof can be found in [19].

Here we use the framework from [14] to design a balanced projection method defined by a solution space condition and a Petrov–Galerkin condition. Thanks to this formalism, it is straightforward to prove the convergence of the method. Then we prove that this balanced projection method possesses a four-term recurrence, so that the overhead introduced is very small.

Our AugCG method introduces at each step k the subspace $\mathcal{K}_{m,k}(A, s_0, r_0) = \mathcal{K}_m(A, s_0) + Span\{r_0, r_1, \ldots, r_k\}$, where $r_0, r_1, \ldots, r_k$ are the previous residuals. Our objective is to characterize this subspace, which is not a Krylov subspace. To achieve this, we introduce the matrix H of the A-orthogonal projection onto $\mathcal{K}_m(A, s_0)^{\perp_A}$. The main result, which is new, is that $Span\{r_0, r_1, \ldots, r_k\}$ is the Krylov subspace $\mathcal{K}_k(H^*AH, r_0)$. This allows us to derive an error bound involving the condition number of the matrix H^*AH.

To prove this result, we use a polynomial formalism with polynomials in the two variables A and H. We give another proof, which was first outlined in [19] and developed in [26], based on the fact that AugCG is equivalent to the CG method preconditioned by the symmetric positive semidefinite matrix HH^* and with the same initial approximation. Since the preconditioning is here singular, the usual results on PCG do not directly apply but must be proved in this context.

Our numerical experiments illustrate the behavior of AugCG algorithm. We observe that AugCG has a better asymptotic convergence than CG or InitCG. We did not investigate the numerical stability of AugCG. It may happen that full orthogonalization is required, as in domain decomposition methods.

The paper is organized as follows. Section 2 reviews the properties of CG needed in the new algorithms. Section 3 defines the algorithm InitCG and studies its convergence. Section 4 presents the algorithm AugCG and studies its convergence thanks to a polynomial formalism and to a preconditioning formalism. Section 5 deals with some practical considerations, such as complexity, memory requirements, numerical stability, and preconditioning. Finally, section 6 presents our numerical results and section 7 gives concluding remarks.

1.4. Notation. We introduce some notation which will be useful throughout the paper. We denote (x, y) the scalar product between the two vectors x and y of $\mathbb{R}^N$; the transpose of the matrix B is denoted by B^*. $Span(B)$ denotes the subspace spanned by the column vectors of B. $\mathbb{R}[X]$ denotes the set of polynomials in one variable; $\mathbb{R}_j[X]$ denotes the set of polynomials in one variable of degree at most j;

$\mathbb{R}\langle X, Y\rangle$ denotes the set of polynomials in two noncommutative variables.

2. Background. Let A be a symmetric positive definite matrix of $\mathbb{R}^{N,N}$ and let

$$Ay = c, \tag{2.1}$$

$$Ax = b \tag{2.2}$$

be two successive linear systems to solve.

We assume that a classical CG algorithm is used to solve the first system (2.1). We do the analysis for CG, but the same applies for PCG. To set the notation, the algorithm is given below.

ALGORITHM 1. CG1.

* iterative solution of $Ay = c$;
* Initialization;
choose y_0;
$s_0 = c - Ay_0$;
$w_0 = s_0$;
* Iteration;
for $j = 0, 1, \ldots$ **until** convergence **do**
  $\gamma_j = (s_j, s_j)/(w_j, Aw_j)$;
  $y_{j+1} = y_j + \gamma_j w_j$;
  $s_{j+1} = s_j - \gamma_j A w_j$;
  $\delta_{j+1} = (s_{j+1}, s_{j+1})/(s_j, s_j)$;
  $w_{j+1} = s_{j+1} + \delta_{j+1} w_j$;
end do

Let $S_j = (s_0, s_1, \ldots, s_j)$ and $W_j = (w_0, w_1, \ldots, w_j)$ be the set of residuals and conjugate directions generated. Recall that

$$S_j^* S_j = \Delta_j,\ W_j^* A W_j = D_j,\ Span(S_j) = Span(W_j) = \mathcal{K}_j(A, s_0), \tag{2.3}$$

where Δ_j and D_j are diagonal matrices and $\mathcal{K}_j(A, s_0)$ is the Krylov subspace of dimension $j + 1$ generated by the initial residual s_0.

We also recall the following result, which introduces the projections H_j and H_j^*.

DEFINITION 2.1. *Let $H_j = I - W_j D_j^{-1} (AW_j)^*$ be the matrix of the A-orthogonal projection onto $\mathcal{K}_j(A, s_0)^{\perp_A}$; $H_j^* = I - (AW_j) D_j^{-1} W_j^*$ is the A^{-1}-orthogonal projection onto $\mathcal{K}_j(A, s_0)^{\perp}$.*

THEOREM 2.2. *The residual s_j satisfies the relations $s_{j+1} = H_j^* s_0 \quad (j \geq 0)$ and $H_m^* s_j = s_j$ for $m < j$.*

Proof. $s_1 = H_0^* s_0$ since $w_0 = s_0$. We prove the first relation by induction. We have $s_{j+1} = s_j - \gamma_j A w_j$ and $(s_j, s_j) = (w_j, s_j) = (w_j, s_0)$ so that $s_{j+1} = H_{j-1}^* s_0 - Aw_j\ 1/(w_j, Aw_j)\ w_j^* s_0 = H_j^* s_0$.

For $m < j$, $W_m^* s_j = 0$ so that $H_m^* s_j = s_j$. ∎

3. InitCG algorithm. We now want to use this information to speed up the solution of the second system (2.2). We will use m vectors, with m given, and from now on omit the index m in S, W, D, H, H^*.

A first idea, introduced in [22, 24, 30] for the Lanczos method, is to choose an initial guess x_0 such that the initial residual $r_0 = b - Ax_0$ is orthogonal to the Krylov subspace $\mathcal{K}_m(A, s_0)$. We thus want to enforce the orthogonality conditions

$$W^* r_0 = 0. \tag{3.1}$$

PROPOSITION 3.1. *Let x_{-1} be any initial approximate solution and $r_{-1} = b - Ax_{-1}$. To guarantee* (3.1)*, we must choose*

$$x_0 = x_{-1} + WD^{-1}W^*r_{-1}, \quad r_0 = b - Ax_0 = H^*r_{-1}. \tag{3.2}$$

Proof. $W^*r_0 = 0$ is equivalent to $r_0 \in Im(H^*)$; in other words

$$W^*r_0 = 0 \Leftrightarrow \exists u, \quad r_0 = H^*u.$$

Thus (3.1) implies that $r_0 = b - Ax_0 = u - AWD^{-1}W^*u$. Multiplying by A^{-1}, we get $x_0 = A^{-1}(b-u) + WD^{-1}W^*u$. Let $x_{-1} = A^{-1}(b-u)$, then $u = b - Ax_{-1}$, so that

$$x_0 = x_{-1} + WD^{-1}W^*r_{-1}, \text{ with } r_{-1} = b - Ax_{-1}.$$

Conversely, if x_0 satisfies (3.2), then $r_0 = H^*r_{-1}$. ∎

As far as x_{-1} is concerned, we can choose, for example, $x_{-1} = y_0$. Then $r_{-1} = b - Ax_{-1} = b - c + s_0$ and $r_0 = H^*r_{-1} = H^*(b-c) + s_m$.

We can also choose $x_{-1} = y_j$, where $j > m$ is the number of iterations in the first system. Then $r_{-1} = b - Ax_{-1} = b - c + s_j$ and $r_0 = H^*r_{-1} = H^*(b-c) + s_j$. This second solution is interesting because $\|s_j\|_{A^{-1}} \leq \|s_m\|_{A^{-1}}$. Anyway, the quality of the initial solution will depend upon $\|H^*(b-c)\|_{A^{-1}}$.

We then use the CG algorithm as usual, starting with $p_0 = r_0$. We get the following algorithm, which we call InitCG.

ALGORITHM 2. INITCG.

* iterative solution of $Ax = b$;
* Initialization;
choose x_{-1};
$r_{-1} = b - Ax_{-1}$;
$x_0 = x_{-1} + WD^{-1}W^*r_{-1}$;
$r_0 = b - Ax_0$;
$p_0 = r_0$;
* Iteration;
for $k = 0, 1, \ldots$ **until** convergence **do**
  $\alpha_k = (r_k, r_k)/(p_k, Ap_k)$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $r_{k+1} = r_k - \alpha_k Ap_k$;
  $\beta_{k+1} = (r_{k+1}, r_{k+1})/(r_k, r_k)$;
  $p_{k+1} = r_{k+1} + \beta_{k+1}p_k$;
end do

InitCG is still a CG method. However, during the first $m/2$ iterations, this algorithm is equivalent to a block CG method.

THEOREM 3.2. *Let n be the number of iterations in the second resolution. As long as $n \leq m/2$, the CG algorithm applied to the linear system $Ax = b$, started with the initial guess x_0 and initial residual r_0 given by* (3.2)*, where x_{-1} is any vector, is mathematically equivalent to the block CG algorithm with block size of* 2 *started with the block $[s_0, r_0]$.*

Of course, this result has no practical interest since r_0 is not known before W is built, so that the block CG should start after the first system has been solved. Anyway, it has a theoretical interest. For the first $m/2$ iterations, we get an accelerated scheme compared to a classical CG method. To prove this theorem, we follow and detail the proof given in [24] for the Lanczos method.

Proof. Let $S_j = (s_0, s_1, \ldots, s_j)$ and $R_k = (r_0, r_1, \ldots, r_k)$. We first show that $Span(S_j)$ is orthogonal to $Span(R_k)$ for all (j,k) such that $j + k \leq m$.

Let s and r be two vectors of $Span(S_j)$ and $Span(R_k)$, respectively. Thanks to (2.3), there exist a polynomial $Q \in \mathbb{R}_j[X]$ and a polynomial $P \in \mathbb{R}_k[X]$ such that $s = Q(A)s_0$ and $r = P(A)r_0$. Therefore,

$$s^* r = (Q(A)s_0)^*(P(A)r_0)$$

and we only have to prove the orthogonality for monomials such as $A^i s_0$ and $A^l r_0$ with $i \leq j$ and $l \leq k$. This property comes from the choice of the initial residual r_0; indeed, since A is symmetric,

$$(A^i s_0)^*(A^l r_0) = (A^{i+l} s_0)^* r_0.$$

Now, since $i + l \leq j + k \leq m$, $A^{i+l} s_0 \in \mathcal{K}_m(A, s_0) = Span(W)$, and by construction $W^* r_0 = 0$, so we conclude that $(A^i s_0)^*(A^i r_0) = 0$ and consequently $s^* r = 0$.

Now, we consider a block CG algorithm started with the block $[s_0, r_0]$, with r_0 defined by (3.2).

We denote by $[s_k, r_k]$ the block of size 2 of residuals and by $[w_k, p_k]$ the block of size 2 of descent directions. The block CG algorithm, with no preconditioning, gives

$$[s_{k+1}, r_{k+1}] = [s_k, r_k] - A[w_k, p_k]\eta_k,$$

where

$$\eta_k = ([w_k, p_k]^* A[w_k, p_k])^{-1}[s_k, r_k]^*[s_k, r_k].$$

As long as $k + 1 \leq m/2$, so that $2k + 1 \leq m$, we have $w_k^* A p_k = p_k^* A w_k = 0$ since $Aw_k \in Span(S_{k+1})$ and $p_k \in Span(R_k)$. Hence

$$[w_k, p_k]^* A[w_k, p_k] = \begin{pmatrix} w_k^* A w_k & 0 \\ 0 & p_k^* A p_k \end{pmatrix}.$$

Also $s_k^* r_k = 0$ since $s_k \in Span(S_k)$ and $r_k \in Span(R_k)$, so

$$[s_k, r_k]^*[s_k, r_k] = \begin{pmatrix} s_k^* s_k & 0 \\ 0 & r_k^* r_k \end{pmatrix}.$$

Hence we get

$$\eta_k = \begin{pmatrix} s_k^* s_k / w_k^* A w_k & 0 \\ 0 & r_k^* r_k / p_k^* A p_k \end{pmatrix} = \begin{pmatrix} \gamma_k & 0 \\ 0 & \alpha_k \end{pmatrix}$$

and

$$\begin{cases} s_{k+1} = s_k - \gamma_k A w_k, \\ r_{k+1} = r_k - \alpha_k A p_k, \end{cases}$$

which is exactly what is given by Algorithms 1 and 2. The proof is similar for the computation of $[w_k, p_k]$. ∎

4. AugCG algorithm.

4.1. Construction of AugCG. Another way to improve the convergence is to keep the orthogonality condition throughout the iterations. This method was developed for the Lanczos algorithm in [22, 24]. It was also defined for the CG algorithm in [19, 23, 11].

We still denote by x_k the current approximate solution, by r_k the current residual $r_k = b - Ax_k$, by $e_k = x - x_k$ the current error where x is the exact solution, and by p_k the current descent direction.

The idea here is to build the method with two subspaces: the Krylov subspace $\mathcal{K}_m(A, s_0)$ already computed and the subspace $Span(r_0, r_1, \ldots, r_k)$. A condition to satisfy is that the residual r_{k+1} must be orthogonal to both subspaces and that the descent direction p_{k+1} must be A-conjugate to both subspaces. These orthogonality conditions will appear to be satisfied by a recurrence relation.

We thus start with x_0 and r_0 as defined by (3.2). We must start with a descent direction A-orthogonal to $Span(W)$, so that we must choose

$$p_0 = (I - WD^{-1}(AW)^*)r_0 = Hr_0. \tag{4.1}$$

Our method is a projection method onto a subspace which is not a Krylov subspace. This subspace is defined by

$$\mathcal{K}_{m,k}(A, s_0, r_0) = \mathcal{K}_m(A, s_0) + Span(r_0, r_1, \ldots, r_k) = Span(W) + Span(R_k). \tag{4.2}$$

Our projection method is then defined by the solution space condition

$$x_{k+1} - x_k \in \mathcal{K}_{m,k}(A, s_0, r_0) \tag{4.3}$$

and by the Petrov–Galerkin condition

$$(r_{k+1}, z) = 0 \quad \text{for all } z \in \mathcal{K}_{m,k}(A, s_0, r_0). \tag{4.4}$$

The algorithm, which we call augmented conjugate gradient (AugCG), defined by (3.2), (4.1), (4.2), (4.3), and (4.4), is a balanced projection method (BPM) based on the matrix $B = A$ as defined in [14]. Since A is assumed to be HPD, we therefore have immediately the following result.

THEOREM 4.1. *Let A be a symmetric positive definite matrix. The algorithm AugCG applied to the linear system $Ax = b$ will not break down at any step. The approximate solution x_k is the unique minimizer of the error $\|e_k\|_A$ over the solution space $\mathcal{K}_{m,k}(A, s_0, r_0)$ and there exists $\epsilon > 0$ independent of e_0 such that for all k*

$$\|e_k\|_A \leq (1 - \epsilon)\ \|e_{k-1}\|_A.$$

Proof. See Theorems 2.4, 2.6, and 2.7 in [14]. ∎

We will now prove that AugCG can be implemented with a four-term recurrence. The solution space condition (4.3) will be satisfied as usual in the CG. The Petrov–Galerkin condition (4.4) will be satisfied simply thanks to three orthogonality conditions: the current residual r_{k+1} is orthogonal to r_k, and the current descent direction p_{k+1} is A-conjugate to p_k and to the last vector w_m. This recurrence means that the complexity of the inner loop is similar to the classical CG. Our new method adds merely one dot-product and one vector update at each iteration. We will prove that the following algorithm implements the projection method AugCG.

ALGORITHM 3. AUGCG.
* iterative solution of $Ax = b$;
* Initialization;
choose x_{-1};
$r_{-1} = b - Ax_{-1}$;
$x_0 = x_{-1} + WD^{-1}W^* r_{-1}$;
$r_0 = b - Ax_0$;
$p_0 = (I - WD^{-1}(AW)^*)r_0$;
* Iteration;
for $k = 0, 1, \ldots$ **until** convergence **do**
$\alpha_k = (r_k, r_k)/(p_k, Ap_k)$;
$x_{k+1} = x_k + \alpha_k p_k$;
$r_{k+1} = r_k - \alpha_k Ap_k$;
$\beta_{k+1} = (r_{k+1}, r_{k+1})/(r_k, r_k)$;
$\mu_{k+1} = (r_{k+1}, Aw_m)/(w_m, Aw_m)$;
$p_{k+1} = r_{k+1} + \beta_{k+1} p_k - \mu_{k+1} w_m$;
end do

As shown in Algorithm 3, we introduce a new scalar μ_{k+1} and a new update for the descent direction p_{k+1}. The initialization step guarantees the relations (3.2) and (4.1).

THEOREM 4.2. *Algorithm* 3 *implements the BPM AugCG defined by* (3.2), (4.1), (4.2), (4.3), *and* (4.4). *More precisely,*

$$z^* r_{k+1} = 0 \quad \text{and} \quad z^* Ap_{k+1} = 0 \quad \textit{for all } z \in \mathcal{K}_{m,k}(A, s_0, r_0).$$

Proof. $p_0 \in Span(W) + Span(R_0)$ and by induction $p_k \in Span(W) + Span(R_k)$ so that Algorithm 3 satisfies the solution space condition (4.3). Indeed,

$$\mathcal{K}_{m,k}(A, s_0, r_0) = Span(W) + Span(R_k) = Span(W) + Span(P_k).$$

As for the orthogonality conditions, we first prove by induction that $W^* r_k = 0$ and $W^* Ap_k = 0$. This is true for $k = 0$. Now we assume it is true at step k. So

$$W^* r_{k+1} = W^* r_k - \alpha_k W^* Ap_k = 0$$

and

$$W^* Ap_{k+1} = W^* A(r_{k+1} - \mu_{k+1} w_m) + \beta_{k+1} W^* Ap_k$$

so that we have to prove that $W^* A(r_{k+1} - \mu_{k+1} w_m) = 0$. To get this, we show that

$$Hr_{k+1} = r_{k+1} - \mu_{k+1} w_m.$$

Indeed,

$$Hr_{k+1} = r_{k+1} - \mu_{k+1} w_m - \sum_{j=1}^{j=m-1} \frac{(r_{k+1}, Aw_j)}{(w_j, Aw_j)} w_j.$$

For $j \leq m-1$, $Aw_j \in Span(w_0, \ldots, w_{j+1}) \subset Span(W)$ and $W^* r_{k+1} = 0$ so that $(r_{k+1}, Aw_j) = 0$. Consequently, $Hr_{k+1} = r_{k+1} - \mu_{k+1} w_m$ and $W^* Ap_{k+1} = 0$.

Now we have to prove by induction that $r_j^* r_{k+1} = 0$ and $p_j^* Ap_{k+1} = 0$ for all $j \leq k$.

This is true for $k = 0$; assuming it is true for k, then

$$r_j^* r_{k+1} = r_j^* r_k - \alpha_k r_j^* A p_k.$$

We first assume that $j \leq k-1$. Then $r_j^* r_k = 0$ by induction. Also $r_j \in Span(W) + Span(P_j)$ so that $r_j^* A p_k = 0$ by induction and by the property $W^* A p_k = 0$ already proved, so that $r_j^* r_{k+1} = 0$. For $j = k$, we have

$$r_k^* r_{k+1} = r_k^* r_k - \alpha_k r_k^* A p_k,$$

but $r_k^* A p_k = p_k^* A p_k - \beta_k p_{k-1}^* A p_k + \mu_k w_m^* A p_k = p_k^* A p_k$ so that by definition of α_k we get $r_k^* r_{k+1} = 0$.

So it remains to prove $p_j^* A p_{k+1} = 0$. We first assume that $j \leq k-1$. Then

$$p_j^* A p_{k+1} = p_j^* A r_{k+1} + \beta_{k+1} p_j^* A p_k - \mu_{k+1} p_j^* A w_m,$$

but $p_j^* A r_{k+1} = r_{k+1}^* A p_j = r_{k+1}^* (r_j - r_{j+1})/\alpha_j = 0$ by the result proved just above, $p_j^* A p_k = 0$ by induction, and $p_j^* A w_m = 0$ by the previous result, so that $p_j^* A p_{k+1} = 0$. For $j = k$ we have

$$p_k^* A p_{k+1} = r_{k+1}^* A p_k + \beta_{k+1} p_k^* A p_k - \mu_{k+1} p_k^* A w_m,$$

but $r_{k+1}^* A p_k = r_{k+1}^* (r_k - r_{k+1})/\alpha_{k+1} = -r_{k+1}^* r_{k+1}/\alpha_{k+1}$ and $p_k^* A w_m = 0$, so that by definition of α_{k+1} and β_{k+1} we get $p_k^* A p_{k+1} = 0$.

This completes the proof. ∎

4.2. Polynomial formalism. A polynomial version of the AugCG algorithm will enable us to characterize the subspace $\mathcal{K}_{m,k}$. We first express r_k and p_k with polynomials in two variables, A and H. More precisely, we have the following result.

THEOREM 4.3. *For $k \geq 0$, there exist two polynomials in two variables $\Phi_k(X,Y)$ and $\Psi_k(X,Y)$ in $\mathbb{R}\langle X,Y\rangle$ such that $r_k = \Phi_k(A,H)r_0$ and $p_k = \Psi_k(A,H)r_0$.*

Furthermore $\Phi_0(X,Y) = 1$, $\Psi_0(X,Y) = Y$, and we have the following recurrence relations in the polynomial space $\mathbb{R}\langle X,Y\rangle$:

$$\begin{aligned} \Phi_{k+1}(X,Y) &= \Phi_k(X,Y) - \alpha_k X \Psi_k(X,Y), \\ \Psi_{k+1}(X,Y) &= Y\Phi_{k+1}(X,Y) + \beta_{k+1}\Psi_k(X,Y). \end{aligned}$$

Proof. Since $p_0 = Hr_0$, we define immediately $\Phi_0(X,Y) = 1$ and $\Psi_0(X,Y) = Y$. The recurrences are deduced from Algorithm 3: it is easy to rewrite the definition of r_{k+1} as $\Phi_{k+1}(A,H)r_0 = \Phi_k(A,H)r_0 - \alpha_k A\Psi_k(A,H)r_0$ which gives the recurrence relation for Φ_k. Recalling that $Hr_{k+1} = r_{k+1} - \mu_{k+1}w_m$ we rewrite the definition of p_{k+1} as

$$\Psi_{k+1}(A,H)r_0 = H\Phi_{k+1}(A,H)r_0 + \beta_{k+1}\Psi_k(A,H)r_0$$

which gives the recurrence relation for Ψ_k. ∎

We can simplify the polynomial expression by introducing polynomials in only one variable, XY. Indeed, we have the following result.

THEOREM 4.4. *For $k \geq 0$, there exist polynomials of degree k in one variable $\Xi_k(Z)$ and $\Omega_k(Z)$ in $\mathbb{R}_k[Z]$ such that*

$$\Phi_k(X,Y) = \Xi_k(XY) \quad \textit{and} \quad \Psi_k(X,Y) = Y\Omega_k(XY). \tag{4.5}$$

Proof. We will prove the theorem by induction: It is true for $k = 0$, since $\Phi_0(X,Y) = 1$ and $\Psi_0(X,Y) = Y$. Assuming it is true at step k, then

$$\begin{aligned}
\Phi_{k+1}(X,Y) &= \Phi_k(X,Y) - \alpha_k X\Psi_k(X,Y) \\
&= \Xi_k(XY) - \alpha_k XY\Omega_k(XY) \\
&= \Xi_{k+1}(XY), \\
\Psi_{k+1}(X,Y) &= Y\Phi_{k+1}(X,Y) + \beta_{k+1}\Psi_k(X,Y) \\
&= Y\Xi_{k+1}(XY) + \beta_{k+1}Y\Omega_k(XY) \\
&= Y(\Xi_{k+1}(XY) + \beta_{k+1}\Omega_k(XY)) \\
&= Y\Omega_{k+1}(XY).
\end{aligned}$$

Clearly, by induction, $\Xi_{k+1}(Z)$ and $\Omega_{k+1}(Z)$ are of degree $k+1$. ∎

This simple polynomial formulation leads to a characterization of the subspace $Span(R_k)$ as a Krylov subspace. This result, which is new as far as we know, is important since it allows us to derive an asymptotic error bound.

THEOREM 4.5. *For $k \geq 0$, $Span(r_0, \ldots, r_k) = \mathcal{K}_k(H^*AH, r_0)$ and*

$$\mathcal{K}_{m,k}(A, s_0, r_0) = \mathcal{K}_m(A, s_0) \overset{\perp_A}{\oplus} \mathcal{K}_k(H^*AH, r_0).$$

Proof. Since the polynomial $\Xi_k(Z)$ is of degree k, polynomials $(\Xi_i(Z))_{i=0\ldots k}$ are independent and build a basis of $\mathbb{R}_k[Z]$. Therefore, since $r_k = \Xi_k(AH)r_0$,

$$Span(r_0, \ldots, r_k) = \mathcal{K}_k(AH, r_0).$$

It is easy to show that $H^*AH = AH$ so that by induction $(H^*AH)^i r_0 = (AH)^i r_0$ and

$$Span(r_0, \ldots, r_k) = \mathcal{K}_k(H^*AH, r_0).$$

The characterization of $\mathcal{K}_{m,k}$ follows immediately. ∎

We can use classical results of conjugate gradient theory: algorithm AugCG computes the minimum over $\mathcal{K}_{m,k}$ which is no larger than the minimum over $\mathcal{K}_k(H^*AH, r_0)$.

COROLLARY 4.6. *Let κ_1 be the condition number of H^*AH; the error at iteration k in algorithm AugCG satisfies*

$$\|e_k\|_A \leq 2\|e_0\|_A \left(\frac{\sqrt{\kappa_1} - 1}{\sqrt{\kappa_1} + 1}\right)^k.$$

This result means that the asymptotic rate of convergence of AugCG is the same as or better than that for the classical CG because the condition number κ_1 of H^*AH is no larger than the condition number κ_0 of A.

4.3. Preconditioned formalism. We briefly present another way to get the same result, which is developed in [19] for the orthodir version of CG and in [26] for any W. We still consider the projections H and H^*. We will show that algorithm AugCG is equivalent to CG preconditioned by the symmetric positive semidefinite matrix HH^* and started by (3.2).

Indeed, AugCG can be rewritten in the following form.

ALGORITHM 4. AUGCG—PRECONDITIONED FORM.

* iterative solution of $Ax = b$;
* Initialization;
choose x_{-1};
$r_{-1} = b - Ax_{-1}$;
$x_0 = x_{-1} + WD^{-1}W^*r_{-1}$;
$r_0 = b - Ax_0 = H^*r_{-1}$;
$z_0 = Hr_0 = HH^*r_0$;
$p_0 = z_0$;
* Iteration;
for $k = 0, 1, \ldots$ **until** convergence **do**
  $\alpha_k = (r_k, z_k)/(p_k, Ap_k)$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $r_{k+1} = r_k - \alpha_k Ap_k$;
  $z_{k+1} = Hr_{k+1} = HH^*r_{k+1}$;
  $\beta_{k+1} = (r_{k+1}, z_{k+1})/(r_k, z_k)$;
  $p_{k+1} = z_{k+1} + \beta_{k+1}p_k$;
end do

Compared to Algorithm 3, we introduced the matrix H using the fact that $H^*r_k = r_k$, $p_0 = Hr_0$, and $r_{k+1} - \mu_{k+1}w_m = Hr_{k+1}$; we also used the equality $(r_k, z_k) = (r_k, r_k)$, which comes from $(r_k, z_k) = (r_k, Hr_k) = (H^*r_k, r_k) = (r_k, r_k)$.

Therefore, algorithm AugCG is nothing else than PCG with the symmetric positive semidefinite preconditioning matrix HH^* and with the initial guess x_0. This is not classical since usually the preconditioning matrix is nonsingular.

As for any PCG, even with a singular preconditioner, we can prove by recurrence that $r_k \in \mathcal{K}_k(AH, r_0)$ and $p_k \in H\mathcal{K}_k(AH, r_0)$. These properties are stated in Theorem 4.4 in a polynomial form.

Though the preconditioner is semidefinite, the algorithm is well defined, converges and does not break down, because r_0 is chosen orthogonal to W. These properties are stated in theorem 3.2. Therefore, the theory on PCG applies, the algorithm can be rewritten as CG applied to $H^*AHy = H^*b$, and the asymptotic convergence rate is governed by the condition number of H^*AH. This result is stated in Corollary 4.6.

5. Practical considerations.

5.1. Complexity and memory requirements of InitCG and AugCG. We first consider operations count. The initial guess computation in InitCG and AugCG is the most time-consuming overhead. The computation of x_0 in InitCG and AugCG is a BLAS2 operation (projection onto a subspace of size m). The computation of p_0 in AugCG is also a BLAS2 operation.

On the other hand, iterations in InitCG have the same cost as iterations of CG and each iteration of AugCG adds merely one dot-product and one vector update (BLAS1 operations), both of which are very cheap. We emphasize that neither InitCG nor AugCG induces new matrix-vector products. Hence the global overhead introduced should be easily balanced by the reduction in the number of iterations. Moreover, all operations are of type BLAS2 or BLAS1 so they easily vectorize or parallelize.

Let us examine now the memory requirements. Here too the initial guess computation in InitCG and AugCG is the most expensive. It requires storing the m vectors W. In contrast, the iterations of AugCG need no more than two extra vectors (w_m

and Aw_m). Hence it seems reasonable to use secondary storage for W if the main memory is not large enough.

If we do not rely on secondary storage, the question of the choice of m arises. This will be analyzed in numerical experiments.

5.2. Numerical stability of InitCG and AugCG. Now, we must also analyze the effect of rounding errors. Quite often in Lanczos-type methods, rounding errors lead to a loss of orthogonality, which implies poor convergence or divergence. This situation is analyzed for modified Lanczos method in [22] and [30]. In InitCG and AugCG, we enforce the initial orthogonality conditions (3.1). In AugCG, we also enforce two orthogonality conditions during the iterations.

As far as (3.1) is concerned, we follow the scheme proposed in [30] for the initial residual r_0 which merely implements a modified Gram–Schmidt process. We extend this scheme to the initial descent direction p_0. This ensures numerical stability for both r_0 and p_0.

During iterations of AugCG, the new descent direction p_{k+1} is enforced to be A-orthogonal to the last vector in the previous system w_m and to the previous descent direction p_k. Here too we apply a modified Gram–Schmidt process by first A-orthogonalizing against w_m then A-orthogonalizing against p_k. Anyway, loss of orthogonality can appear during the iterations of AugCG, as observed in [26]. A full orthogonalization, as in [26] and [2], can remedy this phenomenon in most situations. We do not investigate full orthogonalization further here.

5.3. Preconditioned versions InitPCG and AugPCG. We also analyze the application of AugCG to PCG, which we call AugPCG, since in practice we will use a preconditioned version of the algorithm. We assume that PCG has been applied in the first system with the symmetric positive definite preconditioning matrix M^{-1}. So now

$$Span(W) = M^{-1}Span(S) = M^{-1}\mathcal{K}_m(AM^{-1}, s_0).$$

The method AugPCG is defined by the subspace

$$\mathcal{K}_{m,k}(A, s_0, r_0) = M^{-1}(Span(S) + Span(R_k)) = Span(W) + Span(P_k)$$

with the subspace condition (4.3) and the Petrov–Galerkin condition (4.4). The results proved above still hold, with the following characterization of $Span(R_k)$:

$$Span(R_k) = \mathcal{K}_k(AHM^{-1}, r_0).$$

Therefore, AugPCG can be viewed as CG preconditioned by both HH^* and M^{-1}, and we have the following implementation.

ALGORITHM 5. AUGPCG.

* iterative solution of $Ax = b$;
* save vectors w_j and Aw_j in secondary storage $j = 1, \ldots, m-1$;
* save vectors w_m and Aw_m in main memory;
* save quantities (w_j, Aw_j) in a vector of length m;
* Initialization;
choose x_{-1};
$x_0 = x_{-1}$;
$r_0 = b - Ax_0$;
for $j = 1$ **to** m **do**

$\sigma_j = (r_0, w_j)/(w_j, Aw_j)$;
$x_0 = x_0 + \sigma_j w_j$;
$r_0 = r_0 - \sigma_j A w_j$;
end do;
$z_0 = M^{-1} r_0$;
for $j = 1$ **to** m **do**
$z_0 = z_0 - (z_0, Aw_j)/(w_j, Aw_j) w_j$;
end do;
$p_0 = z_0$;
* Iteration;
for $k = 0, 1, \ldots$ **until** convergence **do**
$\alpha_k = (r_k, z_k)/(p_k, Ap_k)$;
$x_{k+1} = x_k + \alpha_k p_k$;
$r_{k+1} = r_k - \alpha_k A p_k$;
$z_{k+1} = M^{-1} r_{k+1}$;
$\mu_{k+1} = (z_{k+1}, Aw_m)/(w_m, Aw_m)$;
$z_{k+1} = z_{k+1} - \mu_{k+1} w_m$;
$\beta_{k+1} = (r_{k+1}, z_{k+1})/(r_k, z_k)$;
$p_{k+1} = z_{k+1} + \beta_{k+1} p_k$;
end do

InitPCG starts with the same x_0, r_0 but with $p_0 = M^{-1} r_0$. During iterations of InitPCG, $z_{k+1} = M^{-1} r_{k+1}$, the same formula as in PCG.

6. Numerical experiments. We present some numerical experiments which study the efficiency of our InitCG and AugCG methods. We report here results on some test problems; other examples can be found in [26].

We implemented the method in Matlab, so we give only convergence results, since CPU times would not give realistic comparisons. We solve a first system $Ay = c$, then a second system $Ax = b$. We give numerical experiments for different matrices A.

In all our examples, we choose

$$c = Ae, \text{ where } e = (1, 1, \ldots, 1),$$
$$y_0 = 0, \text{ hence } s_0 = c,$$
$$x_{-1} = y_j, \text{ where } j \text{ is the total number of iterations done in the first system,}$$
$$x_0 = x_{-1} + WD^{-1}W^* r_{-1}.$$

We then study two cases for the right-hand side b. In the first case, $b^{(1)}$ is "close" to c:

$$b^{(1)} = \nu c + \epsilon e + AWe,$$

so that $H^* b^{(1)} = \nu H^* c + \epsilon H^* e = \nu s_m + \epsilon H^* e$ and

$$r_0^{(1)} = (\nu - 1) s_m + s_j + \epsilon H^* e.$$

In the second case, $b^{(2)}$ is "far" from c:

$$b^{(2)} = e,$$

so that

$$r_0^{(2)} = H^*(e - Ae) + s_j.$$

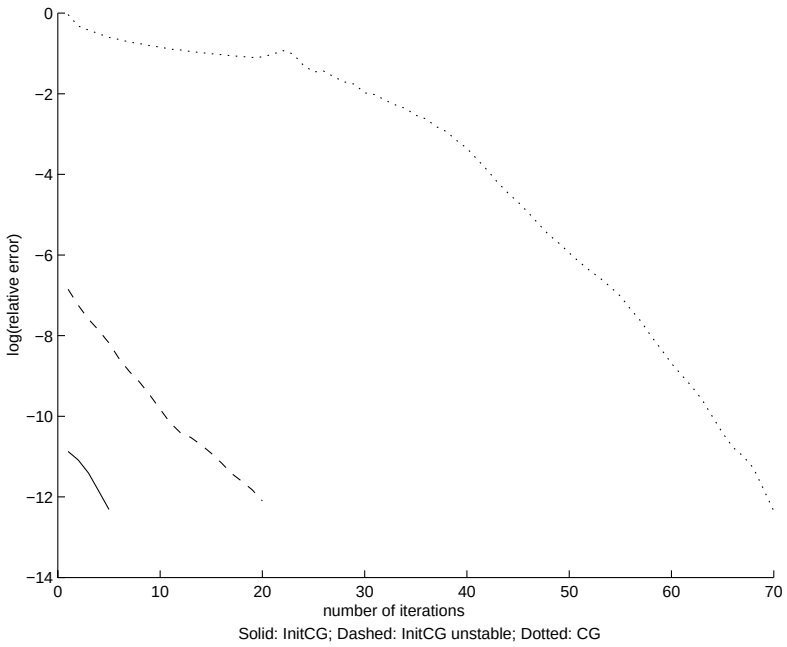

FIG. 6.1. *Results for the Laplacian matrix of order* 900 *with right-hand side* $b^{(1)}$ *(Example* 1*).*

The stopping criterion for the second system is

$$\|r_k\| \leq tol \quad \|b\|,$$

where *tol* is specified for each example.

All figures plot the relative residuals $\|r_k\|/\|b\|$ versus the number of iterations.

Figure 6.1 compares CG, InitCG-unstable, and InitCG. Figures 6.2–6.6 compare CG, InitCG, and AugCG or PCG, InitPCG, and AugPCG. In Figures 6.2–6.6,

- the dotted line plots CG or PCG started with $x_{-1}, r_{-1} = b - Ax_{-1}$,
- the dashed line plots InitCG or InitPCG started with $x_0, r_0 = b - Ax_0 = H^* r_{-1}$,
- and the solid line plots AugCG or AugPCG started with x_0.

6.1. Example 1: Efficiency and numerical stability of InitCG. We choose first the matrix of the Laplacian with Dirichlet boundary conditions discretized by a centered 5-point finite difference scheme on a square grid of size 32×32 so that the matrix is of size 900.

We take here $b = b^{(1)}$, with $\nu = 10$ and $\epsilon = 10^{-2}$. We choose $tol = 10^{-12}$. Here $m = 65$ and $j = 68$, which corresponds to $\|s_j\| \leq 10^{-12}\|c\|$.

We compare CG and InitCG, using either classical Gram–Schmidt (CGS) or modified Gram–Schmidt (MGS) to implement the projection H^*. Figure 6.1 shows the convergence curves to solve $Ax = b$ for CG started with x_{-1} (dotted); for InitCG with CGS, called unstable (dashed); and for InitCG with MGS (solid).

Because m is rather large, the initial error is very small in InitCG. Although CG starts with $x_{-1} = y_j$, InitCG converges much faster than CG thanks to the projection H^*, with $r_0 = H^* r_{-1}$.

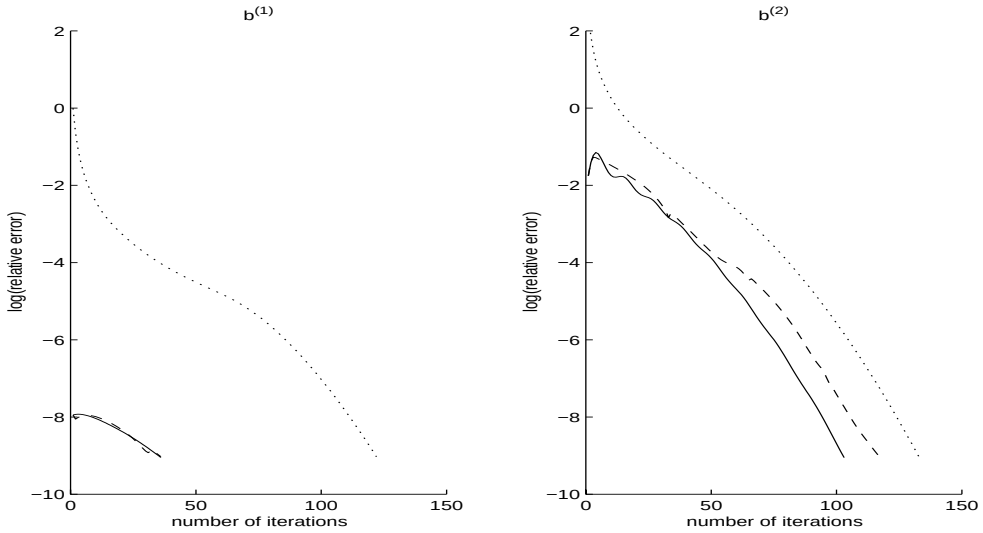

FIG. 6.2. *Results for the matrix* diag([1 : 500]) *(Example 2). Dotted line: CG; dashed line: InitCG; solid line: AugCG.*

If m is small, then both formulations CGS and MGS yield the same result. But here, with $m = 65$, CGS is unstable, as expected, and we will use MGS in what follows, in both InitCG and AugCG.

6.2. Example 2: Impact of right-hand side. Since we study only the convergence behavior, we choose here a diagonal matrix for which we can easily choose the eigenvalues. This is quite usual, as noted in [30, 29]. We choose $A = \text{diag}([1 : 500])$.

Figure 6.2 shows the results for $b^{(1)}$ and $b^{(2)}$. For both right-hand sides, $m = 30$ and $tol = 10^{-9}$. For $b = b^{(1)}$, $\nu = 1$ and $\epsilon = 10^{-2}$.

InitCG (dashed line) and AugCG (solid line) are similar for the "close" right-hand side $b^{(1)}$ and much faster than CG (dotted line); they save about 90 iterations thanks to $\|r_0\|/\|b\|$, which is much smaller than $\|r_{-1}\|/\|b\|$. In some cases, in particular when tol is not too small, AugCG can converge in 1 or 2 iterations [2].

InitCG is not as efficient as AugCG for the "far" right-hand side $b^{(2)}$ because $\|H^*(b-c)\|$ is too large. InitCG and AugCG start with a better approximation than CG, because $b^{(2)}$ is probably related somehow to c. InitCG and AugCG are similar during the first iterations, as expected, then AugCG has a better asymptotic rate than InitCG. Also AugCG converges faster than CG, as expected. In this example, AugCG saves about m iterations.

6.3. Example 3: Impact of size m. We study here the impact of the size m on the efficiency of AugPCG. We use the matrix S1RMQ4M1, from the Cylshell group of independent sets and generators.[1] The matrix is of size $N = 5489$, with 133950 nonzero entries and a condition number equal to 1.81×10^6. The matrix is preconditioned by incomplete Cholesky IC(1).

We choose $b = b^{(2)}$ and $tol = 10^{-9}$.

Figure 6.3 plots the convergence curve of PCG (dotted line) and AugPCG (solid lines) for m varying from 10 to 50.

[1] http://math.nist.gov/MatrixMarket/data

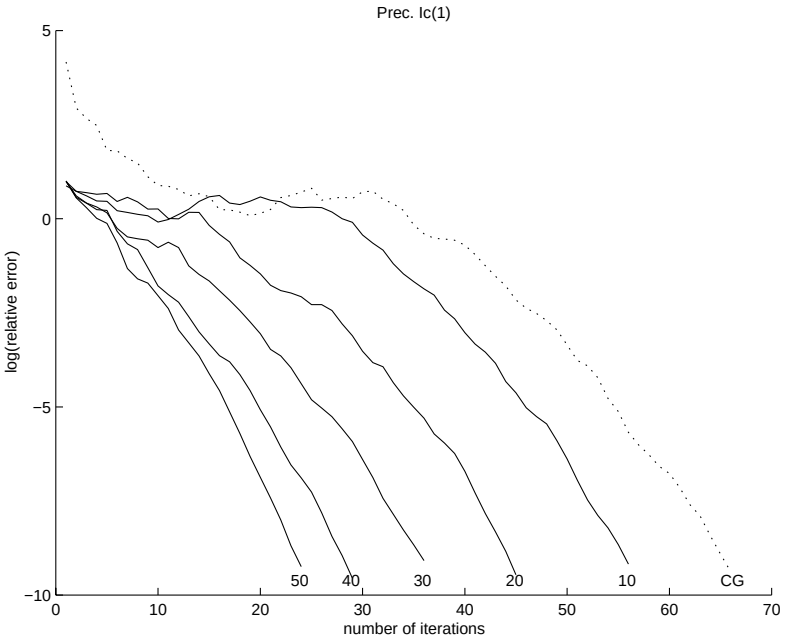

FIG. 6.3. *Results for the matrix S1RMQ4M1 with the right-hand side $b^{(2)}$ (Example 3). Dotted line: PCG; solid line: AugPCG with varying m.*

Up to $m = 40$, AugPCG saves about m iterations, then it saves slightly less than m iterations.

6.4. Example 4: Impact of preconditioning with the close right-hand side $b^{(1)}$. We take in this example the matrix S2RMQ4M1 from the same collection. The matrix is of size $N = 5,489$ with $134,420$ nonzero entries and with a condition number equal to 1.77×10^8.

We choose $b = b^{(1)}$ with $\nu = 1$ and $\epsilon = 10^{-3}$. We choose $m = 20$ and $tol = 10^{-6}$. Figure 6.4 shows the results with four incomplete Cholesky preconditioners with a varying fill-in: IC(1), IC(2), IC(3), and IC(4).

Here InitPCG is quite efficient because both right-hand sides are "close." The initial relative residual $\|r_0\|/\|b\|$ in InitPCG and AugPCG is much smaller than the relative residual $\|r_{-1}\|/\|b\|$ in PCG. It is almost equal to $tol = 10^{-6}$.

AugPCG saves about 36 iterations with IC(1) and about 20 iterations for the other three preconditioners. For IC(1), the gain in the initial residual is relatively more important because PCG converges slowly.

With all four preconditioners, the initial large gain is partly lost afterward because the residual increases. It seems that the convergence curves of AugPCG and PCG have similar patterns, with a shift of about m iterations.

6.5. Example 5: Impact of preconditioning with the far right-hand side $b^{(2)}$. We choose the same matrix S2RMQ4M1 and the parameters $b = b^{(2)}$, $m = 20$, and $tol = 10^{-6}$. Figure 6.5 shows the results for the preconditioners IC(1), IC(2), IC(3), and IC(4).

InitPCG is not efficient because the right-hand side $b^{(2)}$ is too "far" from c.

With all four preconditioners, AugPCG saves about 15 iterations, slightly less

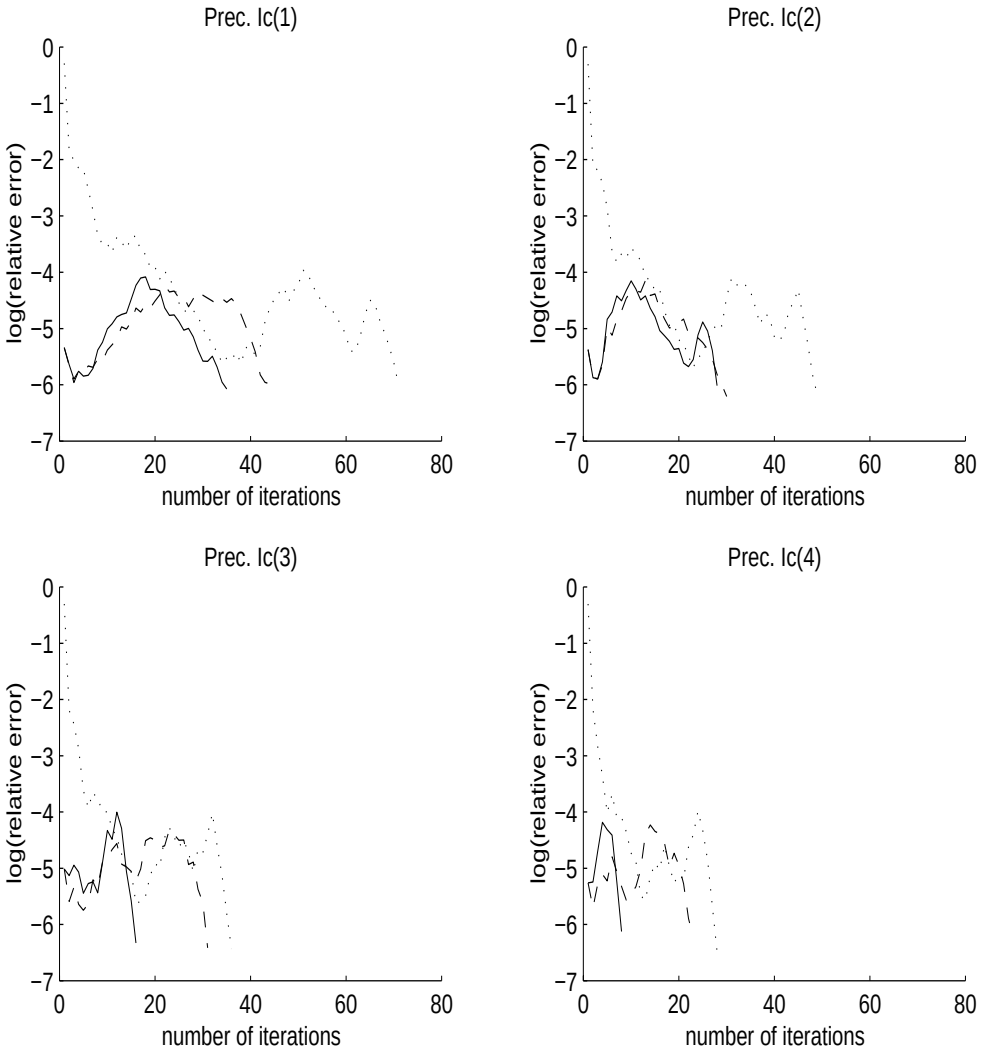

FIG. 6.4. *Results for the matrix S2RMQ4M1 with the right-hand side* $b^{(1)}$ *(Example* 4*). Dotted line: PCG; dashed line: InitPCG; solid line: AugPCG.*

than m equal to 20. So, as in Example 4, AugPCG remains efficient if the preconditioner improves. As in Example 4, AugPCG and PCG have similar patterns with a shift of almost m iterations.

6.6. Example 6: Relative impact of preconditioning and size m. We study here an example where CG converges slowly. We show the impact of preconditioning and of m. We choose the matrix S3RMT3M1 from the same collection. This matrix is of size $N = 5,489$ with $111,579$ nonzero entries. The condition number is 2.48×10^{10}.

We choose $b = b^{(2)}$ and $tol = 10^{-6}$. Figure 6.6 shows the results for $m = 20$ with the preconditioner IC(1), $m = 20$ with the preconditioner IC(2), $m = 50$ and IC(1), $m = 100$ and IC(1).

In all cases, InitPCG is not efficient at all because the right-hand side is too far.

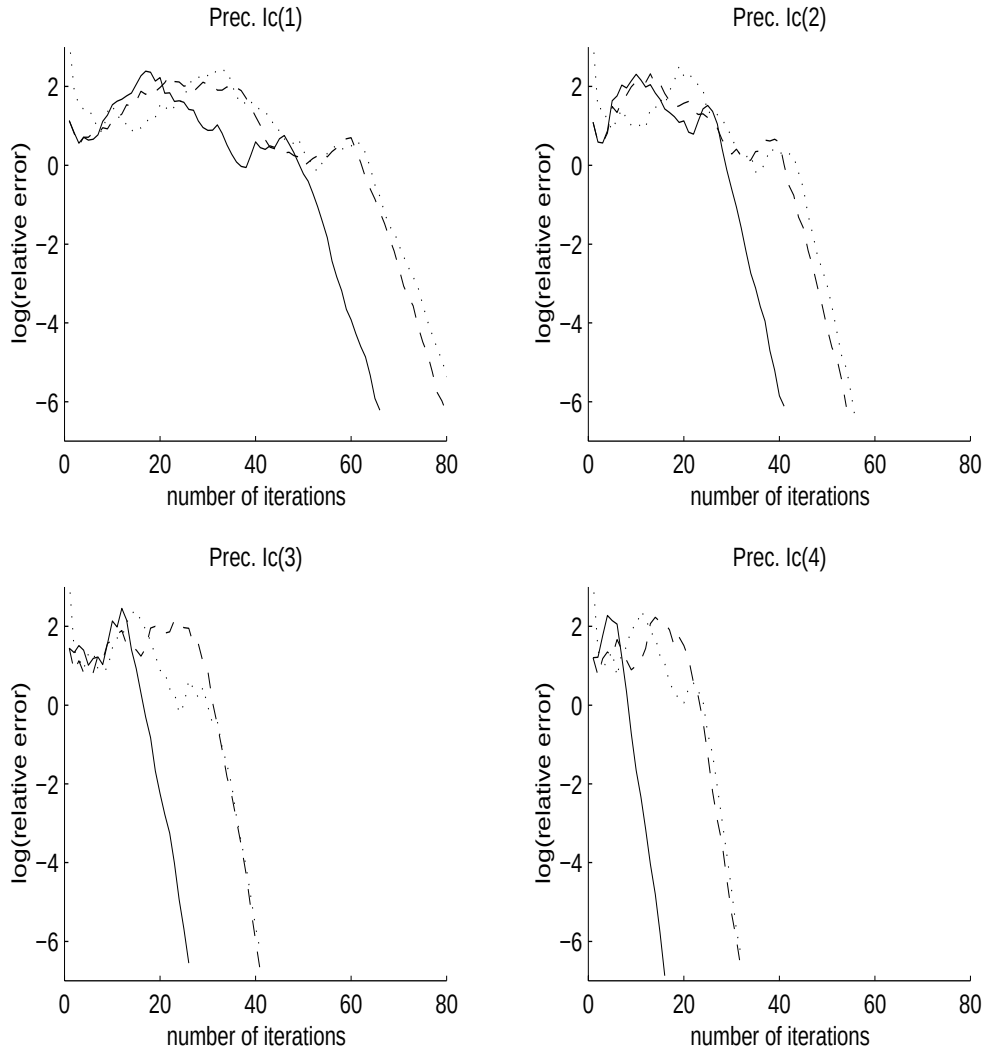

FIG. 6.5. *Results for the matrix S2RMQ4M1 with the right-hand side* $b^{(2)}$ *(Example* 5*). Dotted line: PCG; dashed line: InitPCG; solid line: AugPCG.*

Here, the initial error is of the same order as the error in CG.

With IC(1) and $m = 20$, AugPCG is not very efficient because PCG did not converge well in the seed system.

There are two ways to get a better efficiency. We first improve the preconditioner. With IC(2) and $m = 20$, AugPCG saves 15 iterations compared to PCG.

We then increase the number of vectors retained, while keeping the poor preconditioning IC(1). Whereas $m = 20$ does not improve convergence, $m = 50$ and $m = 100$ are quite efficient.

If the seed system converges fast, then AugPCG is efficient and m can be small. If the seed system converges slowly, then m must be large for an efficient acceleration, but m is restricted by memory constraints. Indeed, the initial guess computation requires the storage of m full vectors of length N. If secondary storage is used, the

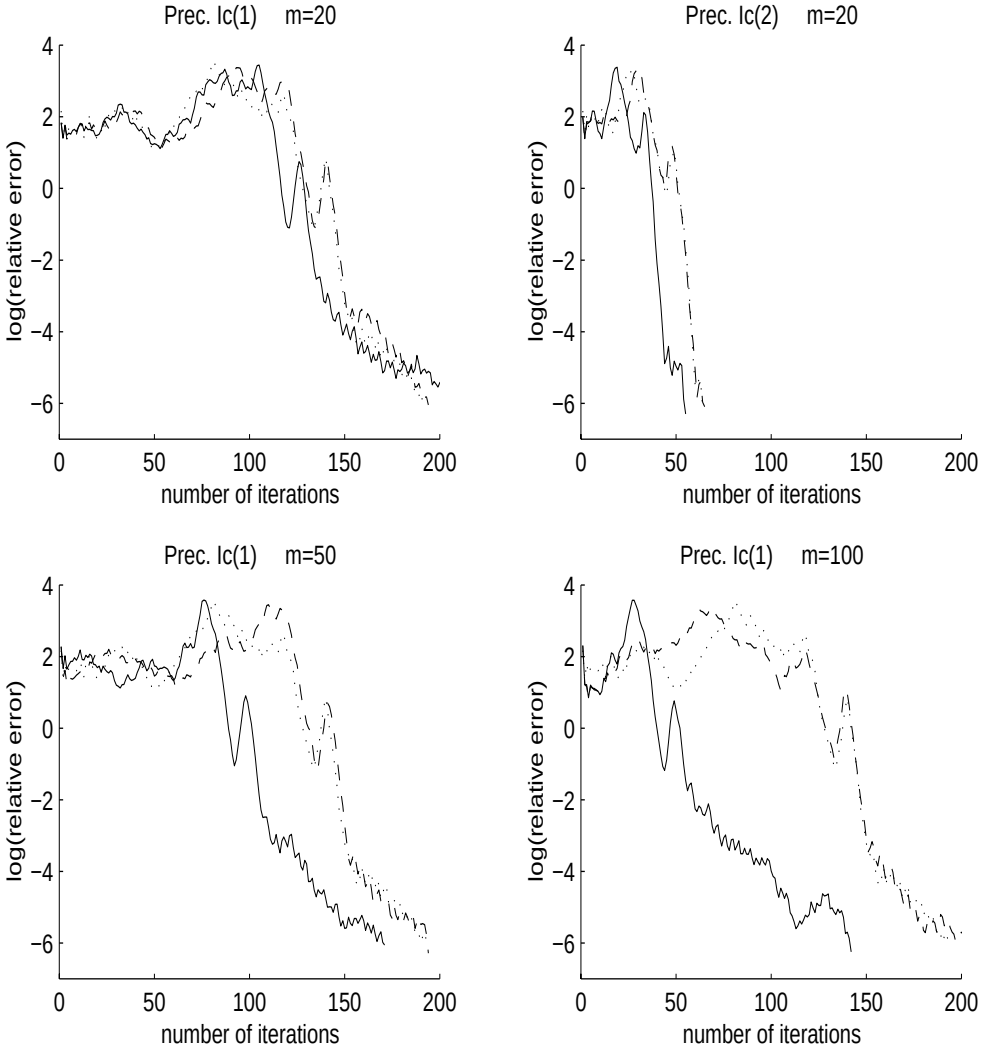

FIG. 6.6. *Results for the matrix S3RMT3M1 with the right-hand side* $b^{(2)}$ *(Example* 6*). Dotted line: CG; dashed line: InitCG; solid line: AugCG.*

gain in iterations must balance the overhead due to I/O [3]. As we mentioned in section 5, the initial guess computation involves BLAS2 operations of order m, so that the CPU cost increases with m. This overhead depends upon the relative cost of each iteration.

The optimal choice of m yielding a minimal CPU time is still an open question. It is probably better to use a good preconditioner with a moderate m. Also it seems that m must be large enough to capture the smallest eigenvalues of A, which mostly affect the convergence of CG [29, 27].

7. Conclusion. This papers considers the problem of solving $Ax = b$ once another system $Ay = c$ has already been solved, where A is symmetric positive definite. The CG applied to $Ay = c$ generates a Krylov subspace $K_m(A, s_0)$, which is used in the second system. The method InitCG computes an initial guess x_0 and residual r_0

using a stable formulation of the A^{-1}-orthogonal projection H^* onto $\mathcal{K}_m(A, s_0)^{\perp}$. The method AugCG not only starts with x_0, but also modifies the descent direction p_k for $k \geq 0$ using a cheap formulation of the A-orthogonal projection H onto $\mathcal{K}_m(A, s_0)^{\perp_A}$.

This paper shows that during the first $m/2$ iterations, InitCG is equivalent to a block CG method started with the 2-block $[s_0, r_0]$. It also shows that convergence of AugCG is governed by the condition number of H^*AH. Numerical experiments, using CG or PCG, demonstrate the efficiency of the method.

Though this method is described here for only two systems, it can be easily extended to more by involving all previous Krylov subspaces in the current system. However, the main drawback of this approach is the rapidly increasing memory requirement. But secondary storage can be used for large systems because it is used only in the initialization part of AugCG. We implemented AugCG with secondary storage in a wave-scattering problem [3]. Also, memory overhead is not crucial when AugCG is applied to the interface problem in a domain decomposition framework [2]. A generalization of AugCG based on a deflation approach was investigated in [26] as well as in [19].

Acknowledgment. The authors wish to thank the referees for their helpful comments and suggestions.

REFERENCES

[1] J. Baglama, D. Calvetti, G. H. Golub, and L. Reichel, *Adaptively preconditioned GMRES algorithms*, SIAM J. Sci. Comput., 20 (1998), pp. 243–269.

[2] M. Brieu, *Homogénéisation et endommagement de composites élastomère par techniques de calcul parallèle*, Ph.D. thesis, Ecole Normale Supérieure de Cachan, France, 1999.

[3] M.-O. Bristeau and J. Erhel, *Augmented conjugate gradient. Application in an iterative process for the solution of scattering problems*, Numer. Algorithms, 18 (1998), pp. 71–90.

[4] A. Bruaset, *A Survey of Preconditioned Iterative Methods*, Pitman Res. Notes Math. Ser. 328, Longman Scientific and Technical, Harlow, UK, 1995.

[5] T. Chan and M. Ng, *Galerkin Projection Methods for Solving Multiple Linear Systems*, Tech. Report 96-31, UCLA, Los Angeles, CA, 1996.

[6] T. F. Chan and W. L. Wan, *Analysis of projection methods for solving linear systems with multiple right-hand sides*, SIAM J. Sci. Comput., 18 (1997), pp. 1698–1721.

[7] A. Chapman and Y. Saad, *Deflated and augmented Krylov subspace techniques*, Numer. Linear Algebra Appl., 4 (1997), pp. 43–66.

[8] R. Choquet, *Etude de la methode de Newton-GMRES—Application aux equations de Navier-Stokes compressibles*, Ph.D. thesis, Université de Rennes 1, France, 1995.

[9] J. Erhel and K. Burrage, *On the performance of various adaptive preconditioned GMRES strategies*, Numer. Linear Algebra Appl., 5 (1998), pp. 101–121.

[10] J. Erhel, K. Burrage, and B. Pohl, *Restarted GMRES preconditioned by deflation*, J. Comput. Appl. Math., 69 (1996), pp. 303–318.

[11] C. Farhat, L. Crivelli, and F.-X. Roux, *Extending substructure based iterative solvers to multiple load and repeated analyses*, Comput. Methods Appl. Mech. Engrg., 117 (1994), pp. 195–209.

[12] P. F. Fisher, *Projection Techniques for Iterative Solution of $Ax = b$ with Successive Right-Hand Sides*, ICASE Tech. Report 93-90, NASA Langley Research Center, Hampton, VA, 1993.

[13] P. Joly, *Résolution de systèmes linéaires avec plusieurs seconds membres par la méthode du gradient conjugué*, Tech. Report R-91012, Université Pierre et Marie Curie, Paris, France, 1991.

[14] W. D. Joubert and T. A. Manteuffel, *Iterative Methods for Nonsymmetric Linear Systems*, Academic Press, New York, 1990, pp. 149–171.

[15] R. Kail, F. Léné, and Y.-H. D. Roeck, *An asymptotic model of laminated plates*, Rev. Européenne Élém. Finis, 6 (1997), pp. 23–42.

[16] C. Le Calvez, *Accélération de méthodes de Krylov pour la résolution de systèmes linéaires creux sur machines parallèles*, Ph.D. thesis, Université de Lille, France, 1998.

[17] R. B. Morgan, *A restarted GMRES method augmented with eigenvectors*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 1154–1171.
[18] R. B. Morgan, *Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations*, SIAM J. Matrix Anal. Appl., to appear.
[19] R. A. Nicolaides, *Deflation of conjugate gradients with applications to boundary value problems*, SIAM J. Numer. Anal., 24 (1987), pp. 355–365.
[20] D. P. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322.
[21] M. Papadrakakis and S. Smerou, *A new implementation of the Lanczos method in linear problems*, Internat. J. Numer. Methods in Engrg., 29 (1990), pp. 141–159.
[22] B. Parlett, *A new look at the Lanczos algorithm for solving symmetric systems of linear equations*, Linear Algebra Appl., 29 (1980), pp. 323–346.
[23] C. Rey, *Développement d'algorithmes parallèles de résolution en calcul non-linéaire de structures hétérogènes: Application au cas de la butée acier-élastomère.*, Ph.D. thesis, Ecole Normale Supérieure de Cachan, France, 1994.
[24] Y. Saad, *On the Lanczos method for solving symmetric linear systems with several right-hand sides*, Math. Comp., 178 (1987), pp. 651–662.
[25] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, 1996.
[26] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h, *A deflated version of the conjugate gradient algorithm*, SIAM J. Sci. Comput., to appear.
[27] G. Sleijpen and A. van der Sluis, *Further results on the convergence behavior of conjugate-gradients and Ritz values*, Linear Algebra Appl., 246 (1996), pp. 233–278.
[28] C. Smith, A. Peterson, and R. Mittra, *A conjugate gradient algorithm for the treatment of multiple incident electromagnetic fields*, IEEE Trans. Antennas and Propagation, 37 (1989), pp. 1490–1493.
[29] A. van der Sluis and H. van der Vorst, *The rate of convergence of conjugate gradients*, Numer. Math., 48 (1986), pp. 543–560.
[30] H. van der Vorst, *An iterative method for solving $f(A)x = b$ using Krylov subspace information obtained for the symmetric positive definite matrix A*, J. Comput. Appl. Math., 18 (1987), pp. 249–263.