Conjugate Gradient Algorithms and the Galerkin Boundary Element Method

O.O. Ademoyero, M.C. Bartholomew-Biggs, A.J. Davies, S.C. Parkhurst

Numerical Optimisation Centre, Mathematics Department
University of Hertfordshire

Abstract

This paper deals with the symmetric linear systems of equations arising in the Galerkin boundary element method. In particular we consider the application of several variants of the conjugate-gradient method to these systems and present some numerical results which shows that a simple preconditioning strategy can lead to significant improvements in solution times.

1 Introduction

In a previous paper [1] we considered the application of conjugate gradient (CG) methods to the linear systems arising in the Galerkin boundary element method. These systems are symmetric but not necessarily positive definite. We observed in [1] that the iterative CG approach was often more efficient than direct solution techniques, such as Gaussian elimination: but, unfortunately, this good performance was not particularly consistent. At best, the CG solutions could be ten times faster than those given by Gaussian elimination; but we also found situations where CG was less than twice as fast. What we also noted in [1] was the limited effectiveness of preconditioning by diagonal scaling. Such preconditioning could sometimes make a modest improvement to CG performance but it seemed just as often to lead to an increase in the number of iterations needed for convergence.

In this paper our aim is to improve upon the rather unsatisfactory state of affairs just outlined. We shall consider a wider range of CG variants and - more significantly - we shall use an alternative form of diagonal preconditioning. Our implementation of CG methods with and without preconditioning is based on suggestions by Luksan and Vlcek [2]. The main significance of the work in [2] is that the algorithms are posed in a form which is computable whether or not the coefficient matrix is positive definite.

Before proceeding to the main body of the paper we shall, for completeness, give a brief overview of the Galerkin boundary element method.

1.1 Boundary integral equations

The two-dimensional mixed potential problem may be written in the form

$$\nabla^2 u = 0 \qquad \text{in} \quad D \tag{1. 1}$$

subject to the boundary conditions

$$u = u_0 \ \text{ on } \quad C_0 \quad \text{and} \quad q \equiv \frac{\partial u}{\partial n} = q_1 \ \text{ on } \quad C_1 \tag{1. 2}$$

where D is the region bounded by the closed curve, $C = C_0 + C_1$. For a point, P, in D we can write the potential at P in the form of a boundary integral [3], using the notation of Gray [4]:

$$\mathcal{P}(P) \equiv u(P) + \oint_C \left(u(Q) \frac{\partial G}{\partial n}(P,Q) - G(P,Q) q(Q) \right) dQ = 0 \tag{1. 3}$$

where

$$G(P,Q) = -\frac{1}{2\pi} \ln |Q - P| = -\frac{1}{2\pi} \ln R$$

is the fundamental solution (Green's function) and $\mathbf{n} = \mathbf{n}(Q)$ is the unit outward normal on C at Q.

For properly-posed problems only one of u or q is known on C so that equation (1. 3) is not directly of use as it stands. The usual approach [3] is to develop a boundary integral equation by considering P as a boundary point and then to obtain the boundary integral

equation by 'excluding' P with a small disc and taking the limit as the disc radius tends to zero to give

$$\alpha(P)u(P) + \oint_C \left(u(Q)\frac{\partial G}{\partial n}(P,Q) - G(P,Q)q(Q) \right) dQ = 0 \qquad (1.\ 4)$$

where $\alpha(P)$ is the so-called 'free-term' coefficient. Equation (1. 4) is often called the potential boundary integral equation. Similarly the flux boundary integral equation is developed in the form [5]

$$\beta(P)q(P) + \oint_C \left(u(Q)\frac{\partial^2 G}{\partial N \partial n}(P,Q) - \frac{\partial G}{\partial N}(P,Q)q(Q) \right) dQ = 0 \qquad (1.\ 5)$$

where $N = N(P)$ is the unit outward normal on C at P.

Equations (1. 4) and (1. 5) are well-established. However, there are considerable worries about the existence of the integrals. Equation (1. 4) involves a weakly singular part, due to G, and a strongly singular part, due to $\frac{\partial G}{\partial n}$. Equation (1. 5) includes a strongly singular part, due to $\frac{\partial G}{\partial N}$, and a hypersingular part, due to $\frac{\partial^2 G}{\partial N \partial n}$. In fact the strongly singular parts are handled in the Cauchy principal value sense and the hypersingular part is usually handled via a Hadamard finite-part integral with the assumption that divergent terms from neighbouring regions cancel. This causes particular difficulties when collocation is used to develop the system equations and so Gray [4] suggests an alternative approach via the Galerkin method. We develop our argument in just the same way. We consider the potential and flux integrals for points P in D as follows:

$$\mathcal{P}(P) \equiv u(P) + \oint_C \left(u(Q)\frac{\partial G}{\partial n}(P,Q) - G(P,Q)q(Q) \right) dQ = 0 \qquad (1.\ 6)$$

$$\mathcal{F}(P) \equiv q(P) + \oint_C \left(u(Q)\frac{\partial^2 G}{\partial N \partial n}(P,Q) - \frac{\partial G}{\partial N}(P,Q)q(Q) \right) dQ = 0 \qquad (1.\ 7)$$

where equation (1. 7) is obtained by direct differentiation of equation (1. 6) using the fact that we can reverse the order of integration and differentiation since the integrals in (1. 6) are well-behaved. We now consider limiting values as the point P approaches the curve C.

1.2 Galerkin formulation

We use the usual boundary element approximation in which the curve C is approximated by a piecewise curve, C_N. In our case C_N is taken as a polygon and the boundary values of u and q are approximated by

$$\tilde{u}(Q) = \sum_{j=1}^{N} w_j(Q)u_j \quad \text{and} \quad \tilde{q}(Q) = \sum_{j=1}^{N} w_j(Q)q_j \qquad (1.\ 8)$$

where $\{w_j(Q) : j = 1, 2, \cdots, N\}$ is a set of linearly independent basis functions. We shall consider linear elements in which the $w_j(Q)$ are the usual 'hat' functions. The boundary element formulation of equations (1. 6) and (1. 7) takes the form

$$\tilde{\mathcal{P}}_{C_N}(P) \equiv u(P) + \oint_{C_N} \left(\tilde{u}(Q)\frac{\partial G}{\partial n}(P,Q) - G(P,Q)\tilde{q}(Q) \right) dQ = 0 \qquad (1.\ 9)$$

$$\tilde{\mathcal{F}}_{C_N}(P) \equiv q(P) + \oint_{C_N} \left(\tilde{u}(Q)\frac{\partial^2 G}{\partial N \partial n}(P,Q) - \frac{\partial G}{\partial N}(P,Q)\tilde{q}(Q) \right) dQ = 0 \qquad (1.\ 10)$$

The limiting process proposed by Gray sets equations (1. 9) and (1. 10) in the form

$$\lim_{\epsilon\to 0}\tilde{\mathcal{P}}_{C_N}(P_\epsilon)=0 \quad \text{and} \quad \lim_{\epsilon\to 0}\tilde{\mathcal{F}}_{C_N}(P_\epsilon)=0 \tag{1. 11}$$

where, as $\epsilon\to 0$, $P_\epsilon\to P_0$ on C_N. Finally, the Galerkin formulation is taken as

$$\oint_{C_N} w_k(P_0)\lim_{\epsilon\to 0}\tilde{\mathcal{P}}_{C_N}(P_\epsilon)dP_0 = 0 \tag{1. 12}$$

$$\oint_{C_N} w_l(P_0)\lim_{\epsilon\to 0}\tilde{\mathcal{F}}_{C_N}(P_\epsilon)dP_0 = 0 \tag{1. 13}$$

There is a variety of different types of integral in equations (1. 12) and (1. 13) depending on the elements in which P and Q are situated. The integrals may be either non-singular, weakly singular, strongly singular or hypersingular. The details of how these are handled are given by Gray [4] and we shall not repeat them here.

The importance of the Galerkin approach is that if equation (1. 6) is used on that part of the boundary on which a Dirichlet condition holds and the *negative* of equation (1. 7) is used on that part of the boundary on which a Neumann condition holds then the resulting boundary element algebraic equations are symmetric. The equations generated from (1. 12), (1. 13) may be written as

$$Hu-Gq=0 \tag{1. 14}$$

which, in block form, are

$$\begin{bmatrix} h^{dd} & h^{dn} \\ -h^{nd} & -h^{nn} \end{bmatrix}\begin{bmatrix} u^d \\ u^n \end{bmatrix}-\begin{bmatrix} g^{dd} & g^{dn} \\ -g^{nd} & -g^{nn} \end{bmatrix}\begin{bmatrix} q^d \\ q^n \end{bmatrix}=\mathbf{0}.$$

Here the partition superscripts indicate the distinction between the parts of the boundary on which Dirichlet and Neumann conditions hold. Since $\mathbf{u}^d$ and $\mathbf{q}^n$ are known, we can re-arrange the system so that only the unknown boundary values appear on the left. Thus if x denotes $(q^d,\ u^n)^T$ we can obtain the overall system of equations in the form $Ax=b$ where

$$A=\begin{bmatrix} -g^{dd} & h^{dn} \\ g^{nd} & -h^{nn} \end{bmatrix}. \tag{1. 15}$$

Green's function has the properties

$$\begin{aligned} G(P,Q) &= G(Q,P) \\ \frac{\partial G}{\partial n}(P,Q) &= \frac{\partial G}{\partial n}(Q,P)=-\frac{\partial G}{\partial N}(P,Q)=\frac{\partial G}{\partial N}(Q,P) \\ \frac{\partial^2 G}{\partial N\partial n}(P,Q) &= \frac{\partial^2 G}{\partial N\partial n}(Q,P) \end{aligned}$$

and by virtue of these it follows that A is symmetric.

2 Linear solvers for the Galerkin method

The $N\times N$ linear system $Ax=b$ arising in the Galerkin method may be solved by a variety of methods. Of the direct approaches (i.e. those which transform or factorise the given

system) the Gaussian elimination or Gauss-Jordan methods are always applicable. Choleski factorization is a more efficient method (requiring only about half as much arithmetic and storage), but it requires A to be both symmetric and positive definite. While the symmetry of (1. 15) is assured there is no guarantee, unfortunately, that it will be positive definite. However, there are iterative methods which can be used to exploit the symmetry of the system without assuming positive definiteness. These can be developed from the method of conjugate gradients.

2.1 Conjugate gradient methods

The conjugate gradient (CG) approach, as first proposed by Hestenes and Stiefel [6], was intended for definite, rather than indefinite systems of equations. It proceeds via iterations of the form

$$x^{(k+1)} = x^{(k)} + \alpha p^{(k)} \tag{2. 1}$$

where the *search directions* $p^{(0)},\ p^{(1)}, ...p^{(k)}, ..$ are constructed to satisfy the conjugacy property

$$p^{(i)T} A p^{(j)} = 0 \quad \text{when } i \neq j. \tag{2. 2}$$

If α in (2. 1) is calculated by

$$\alpha = -\frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}, \tag{2. 3}$$

where r denotes the residual $Ax - b$, then it follows that

$$p^{(k)T} r^{(k+1)} = 0. \tag{2. 4}$$

By using the conjugacy property (2. 2), it can be proved that, after k iterations,

$$p^{(j)T} r^{(k+1)} = 0, \quad \text{for } j = 0, 1, .., k. \tag{2. 5}$$

This result implies *finite termination* — i.e. CG methods solve $Ax = b$ in at most N iterations.

There are a number of ways of generating the conjugate sequence $\{p^{(k)}\}$. The original CG algorithm takes $p^{(0)} = -r^{(0)}$ and then uses the two-term recurrence relation

$$p^{(k+1)} = -r^{(k+1)} + \beta p^{(k)} \quad \text{with } \beta = \frac{r^{(k+1)T} r^{(k+1)}}{r^{(k)T} r^{(k)}}. \tag{2. 6}$$

This is only guaranteed to be stable when A is a positive- (or negative-) definite matrix. To deal with the indefinite case we can replace (2. 6) by a three-term recurrence relation [7]

$$p^{(k+1)} = -Ap^{(k)} + \hat{\alpha} p^{(k)} + \hat{\beta} p^{(k-1)} \tag{2. 7}$$

where $\hat{\alpha},\ \hat{\beta}$ are chosen so that

$$p^{(k+1)T} A p^{(j)} = 0 \quad \text{for } j = k, k-1.$$

We will consider other extensions of the CG method to cover the indefinite case in a later section. (A fuller account of CG methods in the context of boundary integral methods is also given in [8].)

The finite termination property ensures that the CG method is an $O(N^3)$ process, since the work on each iteration is chiefly the N^2 multiplications needed to form the matrix-vector product $Ap^{(k)}$. This workload can be reduced if A is sparse; but unfortunately the Galerkin approach typically generates dense matrices. However, even in the dense case, the CG method becomes more competitive if the eigenvalues of A are "bunched". Specifically it can be proved that, if A has only K distinct eigenvalues then convergence occurs in, at most, K iterations.

The last remark motivates *pre-conditioned* CG methods. In this approach we seek matrices C and M where C is symmetric and M is such that $MM^T = C^{-1}$. The intention is that MAM^T has eigenvalues more tightly bunched than the orginal matrix A. The basic CG method can then be applied to solve the system $MAM^T y = Mb$ after which we set $x = M^T y$. In the context of the original CG method for positive-definite A it was expected that C would also be positive-definite so that M has real elements. For instance, when A is positive-definite its diagonal terms a_{ii} are strictly positive and a simple choice for C is the so-called *diagonal pre-conditioner*

$$C = \mathbf{diag}(a_{ii}). \tag{2. 8}$$

The corresonding M is then $\mathbf{diag}(a_{ii}^{-1/2})$. If we use (2. 8) the pre-conditioned matrix MAM^T has all diagonal elements equal to one which can sometimes cause the eigenvalues also to be clustered around 1. More sophisticated preconditioners for the positive-definite case can be obtained by finding M by means of "incomplete" Cholesky factorizations of A.

If we want to extend the idea of pre-conditioning to the case when A is indefinite then, in order for M to be computable in real arithmetic, the diagonal scaling matrix would have to be defined as

$$C = \mathbf{diag}(|a_{ii}|). \tag{2. 9}$$

The preconditioned matrix MAM^T would then have diagonals equal to ± 1, which can still imply that it has eigenvalues grouped close to 1 and -1. In [1] it was observed that preconditioning based on (2. 9) was sometimes successful, but not consistently so.

In the present paper we follow some ideas given in [2] which show that the preconditoned conjugate gradient method can be implemented *without making use of M explicitly*. The algorithm can be written in such a way that it only involves multiplications by C^{-1} and hence it is possible to use indefinite preconditioners. Specifically, the diagonal preconditioner (2. 8) is practicable (except, of course, in the case when any a_{ii} is zero). In the numerical results presented below we shall show that (2. 8) can be much more successful than (2. 9).

In the following sections we consider variants of the conjugate-gradient approach which can be applied to indefinite matrices and which can use indefinite preconditioners. The algorithms quoted are based on those given by [2].

2.1.1 A preconditioned CG algorithm to solve $Ax = b$

Set $x^{(0)} = 0, \quad r^{(0)} = b$
Calculate $\tilde{r}^{(0)} = C^{-1} r^{(0)}, \ \gamma^{(0)} = \tilde{r}^{(0)T} r^{(0)}, \ \ p^{(0)} = \tilde{r}^{(0)}$
For $k = 0, 1, 2, ...$ calculate

$\alpha = \gamma^{(k)}/p^{(k)T}Ap^{(k)},\ \ x^{(k+1)} = x^{(k)} + \alpha p^{(k)}$
$r^{(k+1)} = r^{(k)} - \alpha Ap^{(k)},\ \ \tilde{r}^{(k+1)} = C^{-1}r^{(k+1)},\ \ \gamma^{(k+1)} = \tilde{r}^{(k+1)T}r^{(k+1)}$
$\beta = \gamma^{(k+1)}/\gamma^{(k)},\ \ p^{(k+1)} = \tilde{r}^{(k+1)} + \beta p^{(k)}$
until $||r^{(k+1)}|| <$ specified tolerance

This algorithm implements the basic conjugate gradient technique, based on (2. 1) - (2. 6). As mentioned already, the algorithm is designed for the case when A is positive definite; and if it is applied when A is indefinite (and possibly with an indefinite preconditioner) then it can break down with division by zero in the calculation of $\alpha^{(k+1)}$ or $\beta^{(k+1)}$. Luksan [2] observes that this happens very rarely - as does the related possibility of loss of accuracy in the recurrence relations due to division by near-zero quantities. Our own experiments, described below, do not seem to have suffered from any such difficulties.

2.1.2 A precondioned and smoothed CG algorithm to solve $Ax = b$

Set $x^{(0)} = y^{(0)} = 0,\ \ r^{(0)} = s^{(0)} = b$
Calculate $\tilde{s}^{(0)} = C^{-1}s^{(0)},\ \gamma^{(0)} = \tilde{s}^{(0)T}s^{(0)},\ \ p^{(0)} = \tilde{s}^{(0)}$
For $k = 0, 1, 2, ...$ calculate
$\alpha = \gamma^{(k)}/p^{(k)T}Ap^{(k)},\ \ y^{(k+1)} = y^{(k)} + \alpha p^{(k)}$
$s^{(k+1)} = s^{(k)} - \alpha Ap^{(k)},\ \ \tilde{r}^{(k+1)} = C^{-1}r^{(k+1)}$
$\lambda = r^{(k)T}(s^{(k+1)} - r^{(k)})/||s^{(k+1)} - r^{(k)}||^2$
$x^{(k+1)} = x^{(k)} + \lambda(y^{(k+1)} - x^{(k)}),\ \ r^{(k+1)} = r^{(k)} + \lambda(s^{(k+1)} - r^{(k)})$
$\tilde{s}^{(k+1)} = C^{-1}s^{(k+1)},\ \ \gamma^{(k+1)} = \tilde{s}^{(k+1)T}s^{(k+1)}$
$\beta = \gamma^{(k+1)}/\gamma^{(k)},\ \ p^{(k+1)} = \tilde{s}^{(k+1)} + \beta p^{(k)}$
until $||r^{(k+1)}|| <$ specified tolerance

The smoothed CG method was proposed by Weiss [9] and the iteratates $y^{(k+1)}$ are calculated in the same way as the $x^{(k+1)}$ in the original CG algorithm. In order to get the solution estimates $x^{(k+1)}$ used by the smoothed CG method a further line search along the direction $y^{(k+1)} - x^{(k)}$ is used to yield a one-dimensional minimum of the residual vector $r^{(k+1)}$. This extra step ensures that the residuals decrease monotonically.

Earlier remarks about the possibility of breakdown due to division by zero when A is indefinite are also applicable to this algorithm.

2.1.3 A precondioned conjugate residual algorithm to solve $Ax = b$

Set $x^{(0)} = 0,\quad r^{(0)} = b$
Calculate $\tilde{r}^{(0)} = C^{-1}r^{(0)},\ \tilde{s}^{(0)} = A\tilde{r}^{(0)},\ \gamma^{(0)} = \tilde{s}^{(0)T}\tilde{r}^{(0)}$,
$p^{(0)} = \tilde{r}^{(0)},\ q^{(0)} = \tilde{s}^{(0)}$
For $k = 0, 1, 2, ...$ calculate
$\tilde{q}^{(k)} = C^{-1}q^{(k)},\ \alpha = \gamma^{(k)}/q^{(k)T}\tilde{q}^{(k)}$
$x^{(k+1)} = x^{(k)} + \alpha p^{(k)}$
$r^{(k+1)} = r^{(k)} - \alpha q^{(k)},\ \ \tilde{r}^{(k+1)} = \tilde{r}^{(k)} - \alpha\tilde{q}^{(k)}$
$s^{(k+1)} = A\tilde{r}^{(k+1)},\ \gamma^{(k+1)} = \tilde{s}^{(k+1)T}\tilde{r}^{(k+1)}$
$\beta = \gamma^{(k+1)}/\gamma^{(k)},\ \ p^{(k+1)} = \tilde{r}^{(k+1)} + \beta p^{(k)},\ \tilde{q}^{(k+1)} = \tilde{s}^{(k+1)} + \beta q^{(k)}$
until $||r^{(k+1)}|| <$ specified tolerance

The conjugate residual method was proposed by Stiefel [10] and, like the smoothed CG method, it is based on the minimization of the residual norm $||Ax - b||$ and ensures a monotonic decrease in this quantity. Division by zero can occur in the indefinite case, but this possibility has not arisen in our numerical tests (or in the extensive tests reported in [2] which use the same algorithm).

2.1.4 A precondioned symmetric QMR algorithm to solve $Ax = b$

Set $x^{(0)} = 0,\ \ r^{(0)} = s^{(0)} = b$, $u^{(0)} = v^{(0)} = 0,\ \theta^{(0)} = 0,\ \tau^{(0)} = s^{(0)T}s^{(0)}$
Calculate $\tilde{s}^{(0)} = C^{-1}s^{(0)},\ \gamma^{(0)} = \tilde{s}^{(0)T}s^{(0)},\ \ p^{(0)} = \tilde{s}^{(0)}$
For $k = 0, 1, 2, ...$ calculate
$\alpha = \gamma^{(k)}/p^{(k)T}Ap^{(k)},\ \ s^{(k+1)} = s^{(k)} - \alpha Ap^{(k)}$
$\sigma = s^{(k+1)T}s^{(k+1)}/tau^{(k)}$
$u^{(k+1)} = (\theta^{(k)}u + \alpha p)/(1+\sigma),\ \ v^{(k+1)} = (\theta^{(k)}v + \alpha Ap^{(k)})/(1+\sigma)$
$x^{(k+1)} = x^{(k)} + u^{(k+1)},\ \ r^{(k+1)} = r^{(k)} - v^{(k+1)},$
$theta^{(k+1)} = \sigma,\ \ \tau^{(k+1)} = \tau^{(k)}\sigma/(1+\sigma)$
$\tilde{s}^{(k+1)} = C^{-1}s^{(k+1)},\ \ \gamma^{(k+1)} = \tilde{s}^{(k+1)T}s^{(k+1)}$
$\beta = \gamma^{(k+1)}/\gamma^{(k)},\ \ p^{(k+1)} = \tilde{s}^{(k+1)} + \beta p^{(k)}$
until $||r^{(k+1)}|| <$ specified tolerance

The symmetric QMR algorithm was derived in [11] from the more general non-symmetric algorithm.

3 Numerical experiments

In this section we compare the performance of algorithms 2. 8 - 2.1.4. In our tests we use the linear equations arising in the Galerkin solutions to Laplace's equation with various boundary conditions. It is worth noting that the algorithms stated in the previous section all have a stopping rule based on the *actual* residual $r = Ax - b$, even when pre-conditioning is used. In the numerical tests reported below we have used the convergence test

$$||r|| \leq 10^{-5}||b||.$$

3.1 Example 1

Example 1 involves the problem $\nabla^2 u = 0$ subject to the boundary conditions

$$\begin{aligned} u(0,y) &= 0 \qquad 0 \leq y \leq 1 \\ u(x,0) &= x \qquad 0 \leq x \leq 1 \\ u(1,y) &= 1 \qquad 0 \leq y \leq 1 \\ u(x,1) &= x \qquad 0 \leq x \leq 1 \end{aligned}$$

Results for Example 1 are given in the tables below. We consider different numbers of points N in the discretization of the boundary and for each value of N we show the numbers of it-

erations needed by the algorithms 2.1.1 - 2.1.4. We show the results with no preconditioning and also with preconditioners C given by (2. 8) and (2. 9).

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	27	17	17
2.1.2	25	17	17
2.1.3	26	17	17
2.1.4	25	17	17

Table 1: Numbers of CG iterations for Example 1 with $N = 128$

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	35	24	24
2.1.2	31	20	21
2.1.3	31	20	21
2.1.4	31	20	21

Table 2: Numbers of CG iterations for Example 1 with $N = 256$

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	92	93	93
2.1.2	88	74	75
2.1.3	87	73	74
2.1.4	88	74	75

Table 3: Numbers of CG iterations for Example 1 with $N = 512$

These first results show that performance of the basic conjugate algorithm is consistently inferior to that of the more sophisticated algorithms 2.1.2 - 2.1.4. For the larger problem, preconditioning seems to have no effect for algorithm 2.1.1 and it remains slower than the best of its competitors by about 20%.

3.2 Example 2

Example 2 involves the solution of $\nabla^2 u = 0$ subject to the boundary conditions

$$\begin{aligned} q(0,y) &= 0 \qquad 0 \le y \le 1 \\ u(x,0) &= 0 \qquad 0 \le x \le 1 \\ q(1,y) &= 0 \qquad 0 \le y \le 1 \\ u(x,1) &= 1 \qquad 0 \le x \le 1 \end{aligned}$$

Results for Example 2 are as follows.

To some extent, the results for Example 2 are similar to those for Example 1. Without preconditioning the basic CG method is less efficient than the smoothed CG, conjugate

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	94	41	68
2.1.2	86	38	68
2.1.3	85	38	68
2.1.4	86	36	68

Table 4: Numbers of CG iterations for Example 2 with $N = 128$

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	162	55	137
2.1.2	121	50	136
2.1.3	123	50	135
2.1.4	121	55	136

Table 5: Numbers of CG iterations for Example 2 with $N = 256$

residual and QMR algorithms. When preconditioning is used, however, the behaviour of all four methods is more uniform. What is most striking is that the indefinite preconditioner (2. 8) produces by far the best results for all the algorithms. By contrast, the positive definite preconditioner (2. 9) causes a deterioration in performance in some cases when $N = 256$.

3.3 Example 3

In Example 3 we solve $\nabla^2 u = 0$ subject to the boundary conditions

$$\begin{aligned} u(0,y) &= 300 \qquad 0 \le y \le 6 \\ q(x,0) &= 0 \qquad 0 \le x \le 6 \\ u(6,y) &= 0 \qquad 0 \le y \le 6 \\ q(x,6) &= 0 \qquad 0 \le x \le 6 \end{aligned}$$

Results for this problem are as follows

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	136	54	160
2.1.2	126	54	160
2.1.3	126	56	162
2.1.4	126	54	160

Table 6: Numbers of CG iterations for Example 3 with $N = 256$

These results reinforce the observations made on Example 2. The indefinite preconditioner (2. 8) gives the fastest convergence, and all the methods 2.1.1 - 2.1.4 use similar numbers of iterations. The positive definite preconditioner (2. 9), however is consistently counter-productive. In the absence of preconditioning, the basic CG method needs about 10% more iterations than the other techniques considered.

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	232	94	370
2.1.2	208	93	369
2.1.3	209	94	408
2.1.4	208	94	369

Table 7: Numbers of CG iterations for Example 3 with $N = 512$

3.4 Example 4

Example 4 is the Motz problem which involves solving $\nabla^2 u = 0$ on a rectangular domain $\overline{D\{x, y : 0 \leq x \leq 14,\ 0 \leq y \leq 7\}}$ subject to the boundary conditions

$$\begin{aligned} u(x,0) &= 500 \qquad 0 \leq x \leq 7 \\ q(x,0) &= 0 \qquad 7 \leq x \leq 14 \\ q(x,7) &= 0 \qquad 0 \leq x \leq 14 \\ u(0,y) &= 1000 \qquad 0 \leq y \leq 7 \\ q(14,y) &= 0 \qquad 0 \leq y \leq 7 \end{aligned} \tag{3. 1}$$

Results obtained with Example 4 are given below.

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	253	87	345
2.1.2	240	87	335
2.1.3	239	84	326
2.1.4	240	87	335

Table 8: Numbers of CG iterations for Example 4 with $N = 256$

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	464	139	756
2.1.2	397	139	711
2.1.3	398	141	693
2.1.4	397	139	711

Table 9: Numbers of CG iterations for Example 4 with $N = 512$

The spread of eigenvalues in the coefficient matrix of the linear system for this example appears to be wider than in Examples 1-3, since the number of iterations needed by the unpreconditioned methods (and particularly the basic CG method 2.1.1) is much closer to N. Nevertheless, even though the problem is seems more difficult, we see similar behaviour to that in the preceding subsections. Preconditioner (2. 8) is very effective while (2. 9)is extremely ineffective.

3.5 Example 5

In Example 5 we solve $\nabla^2 u = 0$ subject to the boundary conditions

$$\begin{aligned} u(x,0) &= 0 \qquad 0 \le x \le 100 \\ q(100,y) &= 0 \qquad 0 \le y \le 8 \\ u(x,8) &= -16 \qquad 95 \le x \le 100 \\ q(95,y) &= 0 \qquad 8 \le y \le 20 \\ u(x,20) &= -4 \qquad 0 \le x \le 95 \\ q(0,y) &= 0 \qquad 0 \le y \le 20 \end{aligned}$$

The results for this problem are shown in the tables below.

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	729	203	738
2.1.2	654	203	710
2.1.3	672	209	699
2.1.4	654	204	710

Table 10: Numbers of CG iterations for Example 5 with $N = 256$

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	2062	191	971
2.1.2	1878	189	922
2.1.3	1947	179	936
2.1.4	1878	191	922

Table 11: Numbers of CG iterations for Example 5 with $N = 512$

The linear systems for this example prove to be the most difficult yet for the unpreconditioned algorithms and we see iteration counts of between $2.5N$ and $4N$. Positive definite preconditoning using (2. 9) helps in the larger case but not when $N = 256$. However the indefinite preconditioner (2. 8) makes a substantial difference in all cases, accelerating convergence by a factor of 10 when $N = 512$.

3.6 Example 6

The final problem Example 6 introduces a non-rectangular boundary. We solve $\nabla^2 u = 0$ subject to the boundary conditions

$$\begin{aligned} u(x,0) &= 0 \qquad 0 \le x \le 3 \\ u(x,2) &= 2 \qquad 0 \le x \le 4 \\ u(0,y) &= y \qquad 0 \le y \le 2 \\ q(4,y) &= 0 \qquad 1 \le y \le 2 \\ u(x,y) &= 0 \qquad 3 \le x \le 4, y = \sqrt{(}1 - (x-4)^2) \end{aligned}$$

Results for this example are summarised below.

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	190	74	137
2.1.2	152	71	132
2.1.3	153	70	131
2.1.4	152	71	132

Table 12: Numbers of CG iterations for Example 6 with $N = 256$

algorithm	no preconditioner	preconditioner (2. 8)	preconditioner (2. 9)
2.1.1	339	101	266
2.1.2	241	101	264
2.1.3	243	104	259
2.1.4	241	99	264

Table 13: Numbers of CG iterations for Example 6 with $N = 512$

The introduction of a curved boundary does not seem to have made the problem any more difficult than the previous examples and the relative performances of the methods are much the same as we have seen before.

4 Discussion and Conclusions

Based on the example problems in the previous section we can make the following comments

• without preconditioning, the basic conjugate gradient method is consistently less efficient than the more sophisticated algorithms 2.1.2, 2.1.3 and 2.1.4. On the other hand, there is relatively little to choose between the performances of the smoothed CG method, the conjugate residual method and the symmetric QMR algorithm.

• use of the very simple *indefinite* peconditioner (2. 8) produces significant savings in numbers of iterations for all the methods tested. Indeed, with this preconditioner, all the methods have near-identical performance.

• as we observed in [1], the positive definite preconditioner (2. 9) is only sometimes successful. It never yields faster convergence than (2. 8) and in about half the tests it actually causes an increase in the number of iterations.

The point we now need to consider is how these CG solutions compare, in terms of efficiency, with the use of a direct method such as Gauss-Jordan or Gaussian elimination.

A benchmark test shows that, for a problem with $N = 512$, the solution time needed by a Gauss-Jordan approach is about 240 times as long as a single unpreconditioned conjugate gradient iteration. This factor is likely to be relatively problem-independent since the number of arithmetic steps needed by the Gauss-Jordan approach is about $N^3/2$ multiplications

and does not depend on the values in the coefficient matrix. The main work in an iteration of algorithm 2.1.1 is a matrix-vector product which costs N^2 multiplications. Therefore we can deduce that a conjugate-gradient algorithm will be faster than Gauss-Jordan if it takes fewer than $N/2$ iterations.

The same remarks about cost per iteration also apply to the unpreconditioned forms of algorithms 2.1.2 - 2.1.4. Moreover, if we use a diagonal preconditioner like (2. 8) the cost is not significantly increased. Hence we can measure the performance of the Gauss-Jordan method against the algorithms discussed in this paper by considering the ratios

$$\rho_{CG} = \frac{\text{No of unpreconditioned CG iterations when } N = 512}{N/2}$$

$$\rho_{best} = \frac{\text{No of iterations by best preconditioned CG method when } N = 512}{N/2}.$$

The following table shows the values of these ratios for each of the test examples.

Example	ρ_{cg}	ρ_{best}
1	0.33	0.25
2	0.78	0.37
3	0.82	0.34
4	1.5	0.5
5	7.9	0.69
4	1.0	0.38

Table 14: CG methods vs Gauss-Jordan for problems with $N = 512$

In the first three cases the unpreconditoned CG method is competitive with the Gauss-Jordan technique; but it is of no advantage for the last three examples. However, with the simple indefinite preconditioner (2. 8), *all* the CG variants that we have considered seem to offer appreciably faster solutions than the Guass-Jordan approach.

Further reductions in the numbers of iterations might be expected if the algorithms 2.1.1 - 2.1.4 used a more advanced preconditoner C based on (say) an incomplete LDL^T factorization. It is worth remembering though that the algorithms would then need about $2N^2$ multiplications because a calculation like $\tilde{s}^{(k+1)} = C^{-1}s^{(k+1)}$ or $\tilde{r}^{(k+1)} = C^{-1}r^{(k+1)}$ would then involve a dense matrix rather than a diagonal one. Therefore the use of such a preconditioner would need to halve the number of iterations to solve $Ax = b$ in order to match the relative performance measures quoted in Table 14.

Our experience in this paper strengthens the remarks that we made in [1] suggesting that conjugate gradient methods could *sometimes* be advantageous for solving the linear systems arising the Galerkin boundary element method. We have now obtained a much more consistent advantage by using a simple indefinite preconditioner within an algorithmic framework given by [2]. This preconditoner seems to be equally successful when used with a number of different variants of the CG approach.

References

[1] Ademoyero O.O, Bartholomew-Biggs, M.C. and Davies A.J. *Computational Linear Algebra Issues in the Galerkin Boundary Element Method*, Computers and Mathematics with Applications, 42, pp 1267-1283, 2001.

[2] Luksan L. and Vlcek J. *Numerical experience with Iterative Methods for Equality Constrained Nonlinear programming Problems* Optimization Methods and Software, 16, pp 257-287, 2001.

[3] Brebbia C.A. and Dominquez J. *Boundary Elements, an Introductory Course,* Computational Mechanics Publications, 1992.

[4] Gray L.J. Evaluation of singular and hypersingular Galerkin integrals: direct limits and symbolic computation; *Singular Integrals in boundary element methods*, eds Sladek V. and Sladek J, Computational Mechanics Publications, pp 33-84, 1998.

[5] Sladek V, and Sladek J. Introductory Notes on Singular Integrals; *Singular Integrals in boundary element methods*, eds Sladek V. and Sladek J, Computational Mechanics Publications, pp 1-31, 1998.

[6] Hestenes M.R. and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bureau of Standards* 49, 409-436, 1952.

[7] Nazareth L. A conjugate gradient algorithm without line searches. *Journ. Opt. Theory Applics.* 23, 373-387, 1977.

[8] Romate J.E. On the use of conjugate gradient type methods for boundary integral equations. *Computational Mechanics* 12, 214-232, 1993.

[9] Weiss R. *Parameter-free Iterative Linear Solvers*, Akademie Verlag, Berlin, 1996,

[10] , Stiefel E. *Relaxationsmethoden bester Strategie zur Losung Linearer Gleichungssysteme* Comm. Math. Helv., 29, pp 157-179, 1955.

[11] Freund R.W. and Nachtigal N.M. *A New Krylov-subspace Method for Indefinite Linear Systems* Report ORNL/TM-12754, Oak Ridge National Laboratory, 1994.