JOURNAL OF COMPUTATIONAL PHYSICS 80, 489–497 (1989)

Note

Iterative Algorithms for the Solution of Nonsymmetric Systems in the Modelling of Weak Plasma Turbulence

1. INTRODUCTION

Experiments performed over the last decade in the domain of controlled nuclear fusion have demonstrated the ability of radio-frequency waves not only to maintain steady state currents in the absence of any applied ohmic voltage but even to increase them in the presence of an opposing electric field ("ramp-up"). The quantitative analysis of this regime requires intensive numerical simulations aiming to compute the large distortions induced by the injected waves in the electron distribution function. Along this line, a two-dimensional finite element code (ADLER) has been developed at CRPP (Centre Recherches en Physique des Plasmas) Lausanne [20], which solves for the simultaneous evolution of the electron distribution function and the wave spectrum. The computationally most intensive section of ADLER is the solution of a large number of non-symmetric linear systems using a banded Gauss elimination scheme. Because the memory and computing time required by a banded Gauss solver become prohibitively large as the problem size increases, we analyze four iterative algorithms for the solution of the non-symmetric systems that arise in this problem.

The preconditioned conjugate gradient algorithm has proven very successful in solving sparse symmetric positive-definite linear systems (see among others [5, 7, 8, 11]). Several generalizations of this method have been proposed for solving *nonsymmetric* linear systems. However, as reported by Saad and Schultz [16] none of these generalizations emerge as a clear winner, so that users face a difficult choice when trying to select the "best" algorithm to solve a particular problem.

In this note, we compare the efficiency of four conjugate-gradient-like algorithms to solve the nonsymmetric systems generated by this problem in a production-type code: the normal equation algorithm (CGN), the bi-conjugate gradient (BCG), the conjugate gradient squared (CGS), and the generalized minimum residual method (GMRES(k)). For all these methods, we precondition the system using an incomplete factorization [11, 8] or a modified incomplete factorization [6] of the coefficient matrix. We show that the iterative solvers we test provide a viable alternative to band Gauss methods, particularly for very large problems. We have tested the solvers with non-symmetric matrices arising from different physical parameters (the electric field) and we show that the iterative solvers perform well also in the

0021-9991/89 $3.00

presence of a large skew symmetric part if the symmetric part is positive. Our performance data suggest that CGS and GMRES(k) yield the best performances. The matrices we examine are typical of bilinear finite-element discretization of convection-diffusion equations, so that we expect the conclusions drawn in this note to apply to a wider class of problems. All the numerical experiments were performed on the IBM 3090 with vector facility [1].

2. Finite Element Modelling of Weak Plasma Turbulence

The quantitative analysis of the *ramp up* regime relies upon a detailed description of the interaction between the injected waves and the plasma electrons in the presence of an opposing electric field. In particular, the large distortions of the electron distribution function resulting from the application of the RF power have to be carefully evaluated to compute the total driven current.

To handle this problem a finite element code, ADLER, has been developed, which solves for the simultaneous evolution of the electron distribution function and the wave spectral distribution in velocity and wavenumber space respectively [18–20]. From the computational point of view, the problem consists in advancing in time a pair of nonlinearly coupled two-dimensional fields, $f(u, v)$ and $W(k_x, k_y)$, where u, v and k_x, k_y denote the components of the electron velocity and the wave number parallel and perpendicular to the ambient magnetic field respectively. The task of following a complete evolution requires several hundreds of time-steps.

The quasilinear model of current-drive consists in the following pair of kinetic equations [18]:

$$\frac{\partial f}{\partial t} = \nabla(\mathscr{R} f + \mathscr{D} \nabla f) \tag{1}$$

$$\frac{\partial W}{\partial t} = (2\gamma - \nu_0 Z)\, W + S, \tag{2}$$

where f and W are the electron and wave spectral distributions in velocity and wave number space, respectively. In Eq. (1) $\mathscr{R} = \mathscr{R}_c + E$ and $\mathscr{D} = \mathscr{D}_c + \mathscr{D}(W)$, where the subscript c denotes the Coulomb collisions, E the electric field, and $\mathscr{D}(W)$ the quasilinear diffusion coefficient.

In Eq. (2) $\gamma \equiv \gamma(f)$ is the quasilinear wave damping, ν_0 the collisional damping, Z the ionic charge state, and S is the source of RF power. Length and time are normalized by the electron Debye length and inverse electron plasma frequency, respectively. At each time-step the code yields the electron distribution function and the wave spectral distribution.

Using bilinear finite elements in velocity space and piecewise constants in wavenumber space (see [20]) and a Crank–Nicolson scheme to march in time, at each time step we have to solve an algebraic problem of the form

$$\left(A - B_n \frac{\Delta t}{2}\right) f_{n+1} = \left(A + B_n \frac{\Delta t}{2}\right) f_n \tag{3}$$

$$\left(G - C_n \frac{\Delta t}{2}\right) w_{n+1} = \left(G + C_n \frac{\Delta t}{2}\right) w_n + S_n \Delta t, \tag{4}$$

where A, G, B, and C are the mass and force matrices arising from the discretization of the Eqs. (1)–(2), and the subscript n labels the discrete time lattice t_n.

A and B_n are banded matrices of size $N_u N_v$ and bandwidth $2N_v + 3$, with at most nine non-zero elements per row, while G and C_n are diagonal. The code has a switch to change from an explicit to a fully implicit time integration scheme. We have not experimented lumping the matrix A, so that there is no advantage in using an explicit scheme. It is worth noting that the linear system in Eq. (3) is unsymmetric: it can be split into a symmetric positive definite part $A - B_s \Delta t/2$, where B_s is generated by the diffusive terms in the Eq. (1), and a skew-symmetric part $-B_{ss}\Delta t/2$ generated by the convective terms. In particular, for each value of the electric field E, one defines a critical speed $v_c = \sqrt{2v_0/E}$ such that for $v > v_c$ the skew-symmetric matrix B_{ss} becomes dominant and can cause numerical instabilities.

The algebraic problem represented by the Eqs. (3)–(4) has to be solved at each time-step and iteration cycle. As a result, we must solve several hundred linear systems with the same sparsity structure but different coefficients. When a direct Gauss solver is adopted the solution of these systems costs up to 90% of the total CPU time and requires a large amount of memory. These two factors limit the size of the modelization grid (about 100×50 gridpoints in velocity space) and motivate the analysis of iterative schemes.

3. The Algorithms

Consider the linear system of equations $Ax = b$, where A is a real, sparse, non-singular, non-symmetric matrix of order m, and b is a vector with m components. We consider four iterative algorithms to solve this problem, namely CGN, BCS, CGS, and GMRES. The four algorithms share the same general structure, and the same basic computational kernels.

In our comparisons we consider vectorized versions of the four algorithms. To represent the sparse matrices, we use a general storage format that allows the sparse matrix-vector product to be efficiently vectorized [9]. The storage representation is quite general and sparse matrices generated by irregular grids can be efficiently represented: the only constraint is that all the rows of the sparse matrix must have roughly the same number of non-zero elements to not degrade performance. Using this representation the solution of the two sparse triangular systems in the preconditioning step is not vectorized. If the sparse matrix is generated by a regular grid and has a regular diagonal structure, it is possible to use more efficient storage representations. We do not consider such storage schemes in this note.

The CGN and the BCG algorithms perform operations that involve the transpose of the original matrix. In our implementation of these two algorithms we store the transpose, because on the IBM 3090 VF this is cheaper than scanning the original matrix to search for elements to perform operations with the transpose. This would not be necessary if the matrix were stored by diagonals.

We outline here the computational procedure:

1. Scale the matrix A and the right hand side b by a diagonal matrix D such that all the diagonal elements of the matrix $(D^{-1}A)(D^{-1}A)^t$ are equal to 1 (such a scaling is introduced to improve numerical stability, see, e.g., [12]).

2. For CGN and BCG, transpose matrix A and store A^t.

3. Compute and store the incomplete factors L, D, and U of the incomplete (ILU) decomposition $A = LDU - R$ [8, 11]. We used a standard incomplete factorization scheme. L and U are required to have the same sparsity structure as A: an entry is accepted in L or U during the factorization process only if the corresponding entry of A is non-zero. When a modified incomplete factorization (MILU) [6] is used, the factorization procedure must satisfy the following additional row-sum criterion: the sums of the elements in a given row of A and LDU must be equal.

4. For CGN and BCG, transpose L and U and store the two transpose factors.

5. Iterate until convergence. The main kernels of the iteration routine are matrix-vector products, the solution of sparse triangular systems, and vector updates.

We outline below the iteration schemes. $K = LDU$ denotes either the standard or the modified incomplete decomposition of A. The iteration procedure is terminated when the relative norm of the residual is less than a specified precision, typically 10^{-6}.

Normal Equations CGN [2, 3]

The normal equations algorithm is conjugate gradient applied to the matrix AA^t, which is symmetric and positive definite. Although the algorithm theoretically converges for any non singular matrix, the convergence rate is slow and the round-off error builds up very fast in many cases of practical interest:

- Initialization
  - choose u_0, compute $r_0 = K^{-1}(b - Au_0)$, and $\eta_0 = (r_0, r_0)$
  - set $p_0 = A^t K^{-t} r_0$
- For $i = 0, 1, \ldots,$ do until $\|r_i\|_2 / \|u_i\|_2 < \varepsilon$
  - $\alpha_i = \eta_i / (p_i, p_i)$
  - $u_{i+1} = u_i + \alpha_i p_i$
  - $r_{i+1} = r_i - \alpha_i K^{-1} A p_i$

— $\eta_{i+1} = (r_{i+1}, r_{i+1})$
— $\beta_{i+1} = \eta_{i+1}/\eta_i$
— $p_{i+1} = A^t K^{-t} r_{i+1} + \beta_{i+1} p_i$.

Bi-conjugate Gradient BCG [3, 10, 13, 14, 15]

The BCG algorithm has been used successfully by Mikic and Morse [13] and Koniges and Anderson [10] for problems similar to the one we consider:

- Initialization
  — choose u_0, compute $r_0 = K^{-1}(b - Au_0)$
  — set $\bar{r}_0 = p_0 = \bar{p}_0 = r_0$, and compute $\eta_0 = (\bar{r}_0, r_0)$
- For $i = 0, 1, \ldots$, do until $\|r_i\|_2 / \|u_i\|_2 < \varepsilon$
  — $\alpha_i = \eta_i / (\bar{p}_i, K^{-1} A p_i)$
  — $u_{i+1} = u_i + \alpha_i p_i$
  — $r_{i+1} = r_i - \alpha_i K^{-1} A p_i$
  — $\bar{r}_{i+1} = \bar{r}_i - \alpha_i A^t K^{-t} \bar{p}_i$
  — $\eta_{i+1} = (\bar{r}_{i+1}, r_{i+1})$
  — $\beta_{i+1} = \eta_{i+1}/\eta_i$
  — $p_{i+1} = r_{i+1} + \beta_{i+1} p_i$
  — $\bar{p}_{i+1} = \bar{r}_{i+1} + \beta_{i+1} \bar{p}_i$.

The vectors r_i and $\bar{r}_i$ are such that they verify a biorthorgonality property: $(r_i, \bar{r}_j) = 0$ for $i \neq j$, and p_i and $\bar{p}_i$ verify a biconjugacy property: $(Ap_i, \bar{p}_j) = 0$ for $i \neq j$. The algorithm converges in at most n iterations if it does not break down: as far as we know it is difficult to establish a priori if this condition is satisfied.

Conjugate Gradient Squared CGS [17]

CGS was introduced recently by Sonneveld, Wesseling, and de Zeeuv [17]. It is derived from BCG to circumvent the need to store the transposes of the coefficient matrix and of the preconditioner. From the equations describing the BCG algorithm, we see that r_i and $\bar{r}_i$ can be expressed as polynomials Φ_i of degree i in the matrices $B = K^{-1}A$ and B^t applied to the original residual vectors r_0 and $\bar{r}_0$:

$$r_i = \Phi_i(B)\, r_0, \qquad \bar{r}_i = \Phi_i(B^t)\, \bar{r}_0.$$

As a consequence the scalar product η_i can be computed as

$$\eta_i = (r_i, \bar{r}_i) = (\Phi_i(B)\, r_0, \Phi_i(B^t)\, \bar{r}_0) = (\Phi_i(B)^2\, r_0, \bar{r}_0).$$

Note that the transpose matrix does not appear in the last term. CGS is based on using some simple algebraic manipulations on Φ to obtain a recursive relation for $\Phi_i(B)^2 r_0$, the *square* of the residual of BCG, which is the residual for the new

algorithm. If $\|\Phi_i(B)\, r_0\|_2$ converges to 0, so does $\|\Phi_i(B)^2 r_0\|_2$. Hence CGS converges whenever BCG does and has a faster convergence rate:

- Initialization

— choose u_0, compute $r_0 = K^{-1}(b - Au_0)$, set $\bar{r}_0 = g_0 = r_0$, $h_0 = 0$, $\beta_0 = 0$, and compute $\eta_0(\bar{r}_0, r_0)$

- For $i = 0, 1, \ldots$, do until $\|r_i\|_2 / \|u_i\|_2 < \varepsilon$

— $\alpha_i = \eta_i / (\bar{r}_0, K^{-1} A g_i)$

— $h_{i+1} = r_i + \beta_i h_i + \alpha_i K^{-1} A g_i$

— $p_i = r_i + \beta_i h_i + h_{i+1}$

— $u_{i+1} = u_i + \alpha_i p_i$

— $r_{i+1} = r_i - \alpha_i K^{-1} A p_i$

— $\eta_{i+1} = (\bar{r}_0, r_{i+1})$

— $\beta_{i+1} = \eta_{i+1} / \eta_i$

— $g_{i+1} = r_{i+1} + 2\beta_{i+1} h_{i+1} + \beta_{i+1}^2 g_i$.

GMRES [16]

Several iterative schemes for non-symmetric matrices are based on minimizing the residual r_i over the Krylov subspace spanned by $r_0, Ar_0, \ldots, A^{i-1} r_0$. Here we describe GMRES(k). This method is attractive because it cannot break down, it is stable, and it only requires storing a small number of search directions along which to minimize the residual before restarting the process. In a practical implementation it is generally sufficient to generate 5 to 10 Krylov vectors before restarting the process:

- choose x_0, and compute $r_0 = K^{-1}(b - Au_0)$.

- generate an orthonormal basis for the k-dimensional Krylov subspace K_k spanned by $r_0, \ldots, K^{-1} A^{k-1} r_0$ using a modified Gram–Schmidt procedure.

- solve a least squares problem to find z in K_k so that $x_{i+k} = x_i + z$ has minimum residual.

- if $\|r_{i+k}\|_2 / \|u_{i+k}\|_2 \leqslant \varepsilon$ exit, else restart.

4. Numerical Experiments

The test problem consists in computing the current driven by the radio frequency waves in opposition to the electric field E. This requires evolving the code for a physical time of about a thousand collision times $t_{el} \sim 10^3 \nu_0^{-1}$, which is achieved in 300 Crank–Nicolson steps. From a computational point of view, this requires solving 300 times the sparse, non-symmetric linear system (3). We have solved this problem for many different values of the electric field E, which means solving

TABLE I

Total CPU Seconds to Solve the Problem of Size 64×32

Method	$E/v_0 = 0.01$	$E/v_0 = 0.20$
ILU-CGN	952	1140
ILU-BCG	422	492
ILU-CGS	327	389
ILU-GMRES	303	348
MILU-CGN	1275	xxx
MILU-BCG	549	601
MILU-CGS	560	630
MILU-GMRES	xxx	xxx

systems with very different skew symmetric part B_{ss} generated by the convective term. We recall that the linear systems generated by this problem are positive real, i.e., the symmetric part is positive definite. The skew-symmetric part B_{ss} generated by $E = 0.2 v_0$ is fairly large; in fact in this case the critical speed $v_c = \sqrt{10}$ is well inside the computational domain which extends up to 15 electron thermal speed units.

We report in Table I the total CPU time required to solve the problem for two distinct values of the electric field. With "*xxx*" we indicate that the method has not converged after 400 iterations at one particular time step. The CPU time includes the solution of 300 linear systems of size 2048, the time to compute the array w_n (200 components) and the time to generate the time-dependent matrices. We report the times obtained using the four different algorithms, using both ILU and MILU preconditioning schemes. All the performance data were obtained on the IBM 3090 with the vector facility at ECSEC [4]. We point out that the values of the main physical quantities obtained using the iterative schemes that converged and the Gauss method deviate by less than 1%, which is well within the accuracy required for this kind of physical problem.

Table II summarizes the storage needed by the four algorithms. Each double-precision real number is stored in 64 bits. *Vectors* in the table refers to the storage for the right-hand side, the initial guess, and some work arrays needed for the iteration process; note that this is independent of the sparsity structure of the matrix.

TABLE II

Storage Requirements (in Double Words)

Method	Vectors	Matrices	Total
CGN	$7*M$	$54*M$	$61*M$
BCG	$9*M$	$54*M$	$63*M$
CGS	$9*M$	$27*M$	$36*M$
GMRES(10)	$(10+4)*M$	$27*M$	$(10+31)*M$

TABLE III

Comparing CGS and the Banded Gauss Scheme

Size	CGS		GAUSS	
	CPU time	Memory	CPU time	Memory
64×32	$3.3\ 10^2$	0.071	$2.6\ 10^2$	0.13
128×64	$1.4\ 10^3$	0.29	$3.9\ 10^3$	1.07
192×96	$4.8\ 10^3$	0.66	$1.9\ 10^4$	3.59

Matrices refers to A, the preconditioner, and, where needed, their transposes; these figures are peculiar to the sparsity structure we consider in this example.

We remark that ILU-BCG, ILU-CGS, and ILU-GMRES perform successfully, while CGN requires considerably more CPU time. Second we note that the MILU preconditioner is less efficient, although we do not have a theoretical explanation for this. The data reported in Tables I and II suggest that CGS and GMRES are more efficient both in terms of CPU time and need about half as much memory as the other algorithms.

Because of the small size of the problem (64×32) reported here, the Gauss solver is still competitive with the iterative schemes. However, this situation is rapidly reversed on finer grids as shown in Table III which reports the CPU time and the storage required by CGS and by the banded Gauss solver for three problems of increasing size. Time is measured in seconds and storage in units of 10^6 double-words.

The data we report were obtained with a relative precision of 10^{-9} in the stopping criterion. This was chosen to be comparable to the precision yielded by the Gauss scheme. For these particular physical problems, this is far too restrictive a criterion. Choosing a relative precision of 10^{-6} with ILU-GMRES, 64×32, $E = 0.2v_0$ we obtained a global solve time of 200 s, instead of the 348 reported in Table I. The results of the two runs are indistinguishable from the physical point of view. This shows that a careful tuning, which was not within the scope of this note, of the parameters of the iterative schemes may yield significant performance improvements.

The data presented in this note show that for our problem, the iterative methods provide a viable alternative to the direct Gauss elimination scheme, particularly when the size of the problem is very large. We expect this conclusion to apply to a wider class of convection-diffusion equations in which the symmetric part of the linear system is positive definite.

ACKNOWLEDGMENTS

We wish to thank Vijay Sonnad, IBM Kingston, for providing us with an extensive bibliography and helping us in the early phases of our work. We also wish to acknowledge the referees for their helpful suggestions and comments on our manuscript.

REFERENCES

1. W. BUCHHOLZ, *IBM Systems J.* **25**, 51 (1986).
2. H. C. ELMAN, Thesis, Computer Science Dept. Report 229, Yale University, 1982 (unpublished).
3. R. FLETCHER, in *Numerical Analysis, Dundee 1975*, edited by A. Dold and B. Eckmann (Springer-Verlag, Berlin, 1976), p. 73.
4. P. FRANCHI, ECSEC Report G513-5020, 1986 (unpublished).
5. G. H. GOLUB AND G. A. MEURANT, *Résolution numérique des grands systèmes linéaires* (Eyrolles, Paris, 1983).
6. I. GUSTAFSSON, *BIT* **18**, 142 (1978).
7. L. HAGEMAN AND D. M. YOUNG, *Applied Iterative Methods* (Academic Press, New York, 1981).
8. D. S. KERSHAW, *J. Comput. Phys.* **26**, 43 (1978).
9. D. R. KINCAID, T. C. OPPE, J. R. RESPESS, AND D. M. YOUNG, *ITPACKV 2C User's Guide*, Center for Numerical Analysis Report CNA-191, The University of Texas at Austin, 1984 (unpublished).
10. A. E. KONIGES AND D. V. ANDERSON, *Comput. Phys. Commun.* **43**, 297 (1987).
11. J. A. MEIJERINK AND H. A. VAN DER VORST, *Math. Comput.* **31**, 148 (1977).
12. J. A. MEIJERINK AND H. A. VAN DER VORST, *J. Comput. Phys.* **44**, 134 (1981).
13. Z. MIKIC AND M. MORSE, *J. Comput. Phys.* **61**, 154 (1985).
14. D. P. O'LEARY, *Linear Algebra Appl.* **29**, 293 (1980).
15. Y. SAAD, *SIAM J. Num. Anal.* **19**, 485 (1982).
16. Y. SAAD AND M. H. SCHULTZ, *Math. Comput.* **44**, 417 (1985).
17. P. SONNEWELD, P. WESSELING, AND P. M. DE ZEEUV, in *Multigrid Methods, Bristol, 1983*, edited by D. J. Paddon and M. Holstein (Oxford Univ. Press, Clarendon, Oxford, 1985), p. 117.
18. S. SUCCI, K. APPERT, AND J. VACLAVIK, *Plasma Phys. Controlled Nucl. Fusion* **27**, 863 (1985).
19. S. SUCCI, K. APPERT., G. RADICATI, Y. ROBERT, AND J. VACLAVIK, *J. Comput. Meth. Appl. Mech. Eng.*, in press.
20. S. SUCCI, K. APPERT, AND J. VACLAVIK, in *Proceedings Eighth EPS Conference on Computational Physics, Eibsee, 1986*, Europhysics Conf. Abstracts, edited by J. Nuhrberg (EPS, Geneva, 1986), p. 113.

RECEIVED: June 3, 1987; REVISED: March 18, 1988

G. RADICATI

IBM ECSEC, via Giorgione 150, 00147 Rome, Italy

Y. ROBERT

IBM ECSEC, via Giorgione 159, 00147 Rome, Italy;
CNRS, Laboratoire TIM3-INPG, 38031 Grenoble, France

S. SUCCI

IBM ECSEC, via Giorgione 159, 00147 Rome, Italy