Research Report

KSTS/RR-87/011
16 Dec. 1987

The use of a CGS method for the convective diffusion problem

by

Takashi Nodera

Takashi Nodera

Department of Mathematics
Faculty of Science and Technology
Keio University

Hiyoshi 3-14-1, Kohoku-ku
Yokohama, 223 Japan

Department of Mathematics
Faculty of Science and Technology
Keio University

KSTS/RR-87/011
December 16, 1987

THE USE OF A CGS METHOD FOR THE CONVECTIVE DIFFUSION PROBLEM*

Takashi NODERA†

Department of Computer Science
Stanford University
Stanford, CA 94305

August 1, 2005

Abstract

This paper introduces a new algorithm which solves nonsymmetric sparse linear systems of equations, by the conjugate gradient squared (CGS) method combined with some preconditionings. The algorithm is dramatically superior to biconjugate gradient (BCG) algorithm, although it is a variant of the BCG algorithm. The algorithm is also amenable for implementing on parallel computers. We report on some numerical examples taken from a finite difference approximation to the convective diffusion equation.

1 Introduction

In the mathematical modeling of physical problems one often produces the boundary value problems of partial differential equation. We consider the convective diffusion equation of the form

$$-\bigtriangledown^T \mu \bigtriangledown u + \beta \bigtriangledown u = 0. \tag{1}$$

This equation arises from convective heat transport problem, fluid mechanics and most other transport problems. A standard technique for solving these problems is to approximate the solution of the differential system at a discrete set of points by the solution of linear systems of equations, whose systems are usually large, sparse, and nonsymmetric. We now consider a system of linear equations,

$$Ax = b \tag{2}$$

where the coefficient matrix A is a $n \times n$ nonsingular and nonsymmetric matrix and b is a known vector. It is usually possible to apply *iterative* solvers for this kind of problem. In making the choice of iterative methods, we have to consider any special properties of the coefficient matrix of the linear system which may be exploited, such as definiteness (i.e. positive definite, positive real or not), diagonal dominance, sparsity, and the nature of eigenvalue spectrum of the coefficient matrix. In addition, we must consider the memory requirements of the algorithm, the vectorization possibilities, and the easy implementation of the algorithm.

*This paper will be appeared in CTAC-87, Vol.1 [North Holland].

†On leave from Department of Mathematics, Faculty of Science and Technology, Keio University, 3-14-1 Hiyoshi Kohoku Yokohama 223, JAPAN.

The biconjugate gradient (BCG) algorithm, an iterative method based upon the quasi projection method, can be used to solve nonsymmetric linear systems. This algorithm is firstly introduced by Fletcher[4] for solving symmetric indefinite linear systems of equations and extended to complex BCG algorithm by Jacobs[8]. This scheme is also discussed by O'Leary[13] and Saad[15]. A simple description of the algorithm, as presented in Fletcher[4], is the following:

[Algorithm BCG]
Let x_0 be any initial approximation to the exact solution $\tilde{x}$, compute initial residual: $r_0 = b - Ax_0$ and put $\hat{p}_0 = \hat{r}_0 = p_0 = r_0$. Then for $k = 0, 1, 2, 3, \cdots$, compute

$$\begin{aligned}
x_{k+1} &= x_k + \alpha_k p_k && (3)\\
r_{k+1} &= r_k - \alpha_k A p_k && (4)\\
\hat{r}_{k+1} &= \hat{r}_k - \alpha_k A^T \hat{p}_k && (5)\\
p_{k+1} &= r_{k+1} + \beta_k p_k && (6)\\
\hat{p}_{k+1} &= \hat{r}_{k+1} + \beta_k \hat{p}_k && (7)\\
\alpha_k &= (r_k, \hat{r}_k)/(Ap_k, \hat{p}_k) && (8)\\
\beta_k &= (r_{k+1}, \hat{r}_{k+1})/(r_k, \hat{r}_k). && (9)
\end{aligned}$$

This algorithm is easily vectorizable and doesn't need any parameters to perform iteration. However, the algorithm doesn't have a minimization property as does the basic conjugate gradient algorithm. Moreover, the monotonic decrease of the error of the BCG algorithm is not certified and there exists the possibility of breakdown or numerical instability. Nevertheless, this algorithm is a very powerful iterative procedure for solving nonsymmetric and indefinite linear systems of equations, if utilized prudently.

In section 2, we give one another different algorithm, which is called conjugate gradient squared (CGS) algorithm. This algorithm can be easily derived from the above biconjugate gradient algorithm. The attempts to obtain such an algorithm are described in Sonneveld[16]. We show that the CGS algorithm converges more rapidly rather than the BCG algorithm. Although its work and storage cost of performing one iteration slightly increase, the CGS method converges almost twice as fast as the BCG method. In consequence, it is quite important that we save the computational time (the total number of multiplications).

In section 3, we discuss some preconditionings which we have used for the CGS algorithm. It is well known that the ultimate success of the solver depends on the quality of the preconditioning. The modified incomplete *LU* factorization combined with the CGS method becomes preferable for the required work.

In section 4, the comparisons with other methods are given, along with some numerical experiments of the convective diffusion problem discretized by the modified upwind difference scheme ([1,9]). Our numerical experiments show that the CGS method has almost the fastest convergence among CG-like methods.

In section 5, we describe conclusions based upon the numerical experiments.

2 CGS Algorithm

We consider the systems of linear equations $Ax = b$, where the coefficient matrix A is a nonsingular and nonsymmetric matrix.

In the biconjugate gradient algorithm, the residual vector $\hat{r}_k$ and the direction vector $\hat{p}_k$ are only used for the determination of the step sizes α_k and β_k. Moreover, the matrix-vector product $A^T\hat{p}_k$ is just used for the computation of the inner product. We don't use a lot of information related to these vectors. In order to improve the efficiency of the BCG algorithm, we will consider reconstructing the BCG algorithm.

The conjugate gradient (CG) and biconjugate gradient algorithm are widely known polynomial-based iterative algorithms. One key to reconstructing the BCG algorithm is to consider the matrix polynomial. In the BCG algorithm as well as the CG algorithm, the approximate solution x_k is rewritten to the following polynomial form:

$$x_k = x_0 + q_{k-1}(A)r_0. \tag{10}$$

where x_0 is some initial approximation for the exact solution $\tilde{x}$, $r_0(= b - Ax_0)$ is the initial residual, and q_{k-1} is a real polynomial of degree $k-1$. The residual vector r_k is such that

$$r_k = (I - Aq_{k-1}(A))r_0 = R_k(A)r_0, \tag{11}$$

where $R_k(A)$ is a real matrix-polynomial of degree k, called a residual polynomial, which satisfies the constraint $R_k(0) = 1$. In the same way as above, we can rewrite $\hat{r}_k, p_k$ and $\hat{p}_k$ as the following matrix-polynomial forms.

$$\hat{r}_k = R_k(A^T)r_0, \;\; p_k = P_k(A)r_0, \;\; \hat{p}_k = P_k(A^T)r_0, \tag{12}$$

where $P_k(A)$ is also a real matrix-polynomial of degree k which satisfies the constraint $P_k(0) = 1$.

For (8),(9), by using the above matrix-polynomial froms, it can be easily verified that

$$\begin{aligned} (r_k, \hat{r}_k) &= (R_k^2(A)r_0, r_0), & (13)\\ (\hat{p}_k, Ap_k) &= (r_0, AP_k^2(A)r_0). & (14) \end{aligned}$$

Now let

$$\begin{aligned} \bar{p}_k &\equiv P_k^2(A)r_0, \\ \bar{r}_k &\equiv R_k^2(A)r_0, \\ e_k &\equiv P_k(A)R_k(A)r_0, \\ h_{k+1} &\equiv P_k(A)R_{k+1}(A)r_0. \end{aligned}$$

In the derivation of a new algorithm, we define the following residual:

$$\bar{r}_k = b - A\bar{x}_k. \tag{15}$$

In order to derive new recurrence relation of the residual, by using (11) and (12), we rewrite (4) in the following form:

$$R_{k+1}(A)r_0 = R_k(A)r_0 - \alpha_k Ap_k(A)r_0. \tag{16}$$

Premultiplying by $R_{k+1}(A)$, we have

$$R_{k+1}^2(A)r_0 = R_k(A)R_{k+1}(A)r_0 - \alpha_k AP_k(A)R_{k+1}(A)r_0. \tag{17}$$

Substituting (16) into (17), and using the above definition, we get the new recurrence relation for the residual:

$$\begin{aligned} \bar{r}_{k+1} &= R_k(A)\{R_k(A)r_0 - \alpha_k AP_k(A)r_0\} - \alpha_k Ah_{k+1} & (18)\\ &= \bar{r}_k - \alpha_k A(e_k + h_{k+1}). & (19) \end{aligned}$$

By using a similar simple calculation, we can obtain the following algorithm which is known as conjugate gradient squared method. This algorithm is firstly proposed by Sonneveld[16], and is presented to be equivalent to the biconjugate gradient algorithm.

[Algorithm CGS]
Let $\bar{x}_0$ be any initial approximation to the exact solution $\tilde{x}$, compute initial residual: $\bar{r}_0 = b - A\bar{x}_0$ and put $\bar{p}_0 = r_0 = e_0 = \bar{r}_0$. Then for $k = 0, 1, 2, 3, \cdots$, compute

$$\begin{aligned}
\bar{x}_{k+1} &= \bar{x}_k + \alpha_k(e_k + h_{k+1}) && (20)\\
h_{k+1} &= e_k - \alpha_k A\bar{p}_k && (21)\\
\bar{r}_{k+1} &= \bar{r}_k - \alpha_k A(e_k + h_{k+1}) && (22)\\
e_{k+1} &= \bar{r}_{k+1} + \beta_k h_{k+1} && (23)\\
\bar{p}_{k+1} &= e_{k+1} + \beta_k(h_{k+1} + \beta_k\bar{p}_k) && (24)\\
\alpha_k &= (r_0, \bar{r}_k)/(r_0, A\bar{p}_k) && (25)\\
\beta_k &= (r_0, \bar{r}_{k+1})/(r_0, \bar{r}_k). && (26)
\end{aligned}$$

The CGS algorithm has a basically good property of convergence, although it can be simply derived from the BCG algorithm. In order to study the goodness of convergence, we now consider the residual polynomial of the CGS algorithm. From (11) and (15), we can expect that for many cases, if $\| r_k \| = \| R_k(A)r_0 \| \ll b$ is almost satisfied, then $\| \bar{r}_k \| = \| R_k^2(A)r_0 \|$ will become even smaller. In consequence, we will suggest that the CGS algorithm converges almost twice as fast as the corresponding BCG algorithm. However, we don't have any theoretical results on the rate of convergence of above CGS algorithm.

The CGS algorithm, as well as the BCG algorithm, may have the possibility of breakdown or numerical instability, since it doesn't have certified convergence properties as does the CG algorithm. However, our numerical experiments show that the CGS algorithm combined with a good preconditioning converges very fast, even if it does not have the property of strict minimization.

In Table 2.1, we summerize a comparison of the work per iteration and the storage requirement (excluding storage for A and b) for several algorithms. The matrix-vector products are denoted by Av and A^Tv. From this table, we can see that the storage and work per iteration for the CGS algorithm slightly increases. However, the rapid convergence of the CGS algorithm will overcome these difficulties.

3 Art of Preconditioning

As is well known, the preconditioning has been found to be very effective in solving a large sparse systems of equations. In this section, we consider the application of preconditioning to speed up the convergence of the conjugate gradient squated algorithm. Generally, the preconditioning is important not only for the better rate of convergence, but also the resulting matrix is closer to symmetric than the original matrix.

The basic idea of preconditioning consists of modifying the original system of equations in such a way that iteration on the modified system of equations can proceed with a fast rate of convergence. Namely, the preconditioning works to accelerate the iteration process. Usually, the preconditioning consists of replacing the system of linear equations (2) by an equivalent system of linear equations:

$$B^{-1}Ax = B^{-1}b. \qquad (27)$$

The operator B is called preconditioner, and it is produced by approximate factorization of the coefficient matrix A.

The incomplete LU factorization has been used quite successfully in combination with the conjugate gradient-like method (see [1],[2],[7],[11],[15],[17],[18]). Recently, some advance has occurred and the preconditionings for nonsymmetric matrices are being completely studied. However, most of the preconditionings used for nonsymmetric matrices are generalizations of techniques for symmetric matrices. That is

$$A = LU - R = B - R, \tag{28}$$

where the preconditioner B is the product of a lower triangular matrix L and an upper triangular matrix U which approximate the elements obtained from incomplete Gauss elimination, and R is called the error matrix. The rate of convergence depends on how well the preconditioner B approximates the original coefficient matrix A.

For our examples, we apply both the incomplete $LU(ILU)$ factorization and modified incomplete $LU(MILU)$ factorization with no extra fill in (see [1],[5],[7],[10],[11],[15] for referring to these techniques).

4 Numerical Experiments

In this section, we compare the performances of CGS algorithm with BCG algorithm and ORTHOMIN(1) in solving the systems of linear equations of arising from the convective diffusion problems. All numerical experiments were run on the Vax 11/780 of the Department of Computer Science, Stanford University, in double precision(55 bit mantissa). The initial approximation x_0 in all runs was set to equal to zero. We now consider the following two examples.

Example 1. (Problem with Dirichlet boundary conditions)
The first example is the Dirichlet problem on the unite square Ω with boundary $\partial\Omega$.

$$\begin{aligned} \triangle u + \beta u_x &= f, \ (x,y) \in \Omega = (0,1) \times (0,1), \\ u &= xy(1-x)(1-y), \ (x,y) \in \partial\Omega, \end{aligned}$$

where β is constant and $f = 2x(x-1) + y(y-1)\{2 - \beta(1-2x)\}$. This simple problem has been chosen since it has been widely used by Kincaid and Young[9]. In this problem, a modified upwind difference scheme is used on a square mesh ($h = 1/40$) with a total of $n = 1521$ unknown mesh values with natural ordering to obtain the corresponding discrete problem of the form (2) (see [1],[9] for a simple derivation of this scheme).

We consider three cases of numerical experiments, corresponding to $\beta = 10$, 100 and 1000. The convergence criterion used for stopping the iterative process is $\epsilon \ (=\parallel r_k \parallel / \parallel r_0 \parallel) \leq 10^{-14}$. In Table 4.1 and 4.2, we summarize the total number of iterations and total number of multiplications for different values of β and for different preconditionings, when the several CG-like methods are applied to example 1. We have not taken the work of approximate factorizations into account. From these results, we can find out that the CGS process is stopped as soon as the residual norm was reduced by a factor of 10^{-14}. Moreover, $MILU$ factorization seems to be more effective preconditioning than the ILU factorization. In fact, CGS method, when applied to $MILU$ preconditioning, performs very well and takes the least work among these methods for satisfying the convergence criterion. We also find out that BCG method breaks down at the point of $\epsilon = 10^{-12}$ except for $\beta = 10$. In Fig.4.1 and 4.2, we draw the graphs of norm of residual versus number of iterations for each methods in order to see the change of residual during each iterative process. From these figures, we can see that the CGS and BCG method have some peaks during the iterative process, but both of the methods turn out to be stable gradually.

Example 2. (Problem with Neumann boundary conditions)
The second example is the steady state problem on unite square Ω with Neumann boundary $\partial\Omega$.

$$\begin{aligned} \triangle u + \beta u_x = 0,\ (x,y) &\in \Omega \\ u(x,0) = 0,\ u(x,1) &= 1, \\ u(0,y) = 1,\ u_x(1,y) &= 0,\ (x,y) \in \partial\Omega \end{aligned}$$

We also use a modified upwind difference scheme for the discretization on a square mesh and the same initial guess, mesh size and stopping criterion as same as for the example 1.

Here again, we consider the three cases of numerical experiments. The total number of iterations and multiplications taken to converge are compared in Table 4.3 and 4.4. In Fig.4.3, we compare the behavior of residual versus iterations for $\beta = 10$ when we don't use preconditioning. In this figure, we can see that the residual of CGS method, as well as the BCG method, doesn't always decrease, especially at the beginning of the iterative process, but the CGS method converges faster than the BCG method. On the other hand, the residual of ORTHOMIN(1) converges very slow. In Fig.4.4, we also show the behavior of convergence for $\beta = 100$ when applied to ILU preconditioning. Comparing these results, we find out that for most cases the CGS method combined with $MILU$ preconditioning is a very effective procedure.

5 Conclusions

We have considered the conjugate gradient squared algorithm for solving a nonsymmetric system of linear equations, which arose from the numerical solution of the convective diffusion problem by finite difference and finite element schemes.

A number of numerical experiments were run in order to test the behavior of the convergence for the CGS method. From these results, we conclude that the CGS method combined with $MILU$ preconditioning is quite effective for the solution of this kind of nonsymmetric problem among these methods. However, as stated earlier, a lot of care has to be taken in more general problems, since the CGS algorithm is originally numerically unstable just like the BCG algorithm.

Finally, for more complicated problems, a natural generalization of ILU factorization to block case will produce a class of effective preconditioning, because the CGS method turns out to be more stable, when applied to more accurate factorization.

Acknowledgement.
The author wishes to thank Prof. Gene H. Golub for the invitation to visit Stanford University, for providing nice working conditions and for his helpful comments. The author also wishes to thank for Prof. M. Natori, University of Tsukuba, for his helpful comments and friendship.

References

[1] O. Axelsson and I. Gustafsson, *A modified upwind scheme for convective transport equations and the use of a conjugate gradient method for the solution of nonsymmetric system of equations,* J. of Inst. Math. Appl. No.23, pp. 321-337 (1979).

[2] P. Concus and G. H. Golub, *A generalized conjugate gradient method for nonsymmetric systems of linear equations,* Lecture Notes in Economics and Mathematical Systems 134, Springer-Verlag, pp. 56-65 (1976).

[3] S. C. Eisenstat, H. C. Elman and M. H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations,* SIAM J. Numer. Anal. 20, pp. 345-357 (1983).

[4] R. Flecher, *Conjugate gradient methods for indefinite systems,* Lecture Notes in Math. 506, pp. 73-89 (1976).

[5] I. Gustafsson, *A class of first order factorizations,* BIT 18, pp. 142-156 (1978).

[6] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems,* J. Research Nat. Bur. Standards 49, pp. 409-435 (1952).

[7] J. M. Hyman and T. A. Manteuffel, *High-order sparse factorization methods for elliptic boundary value problems,* Advances in Computational Methods for PDE-V, IMACS, pp. 551-555 (1984).

[8] D. A. H. Jacobs, *The exploitation of sparsity by iterative methods,* in "Sparse Matrices and their Uses", Academic Press, pp. 191-222 (1981).

[9] D. M. Kincaid and D. M. Young, *Adaptive iterative algorithm developed for symmetric systems to nonsymmetric systems,* Elliptic Problem Solvers, Academic Press, pp. 353-359 (1981).

[10] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is symmetric M-matrix,* Math. Comp. 31, pp. 148-162 (1977).

[11] T. Nodera, *PCG methods for a large sparse matrix* (in Japanese), Seminar on Mathematical Science No.7, Keio Univ. (1983).

[12] T. Nodera, *New variant of BCG method for solving nonsymmetric systems,* Advances in Computer Methods for PDE-VI, IMACS, pp. 130-135 (1987).

[13] D. P. O'Leary, *The block conjugate gradient algorithm and related methods,* Linear Algebra and its Appl. 29, pp. 293-322 (1980).

[14] C. C. Page and M. A. Saundars, *Solution of sparse indefinite systems of linear equations,* SIAM J. Numer. Anal. 12, pp. 617-624 (1975).

[15] Y. Saad, *The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems,* SIAM J. Numer. Anal. 19, pp. 485-506 (1982).

[16] P. Sonneveld, *CGS, a fast Lanczos type solver for nonsymmetric linear systems,* Rep. 84-16, Delft University, Dept. of Math., Delft, The Netherlands (1984).

[17] H. Takahasi and T. Nodera, *New variants of conjugate gradient algorithm,* Numerical Methods for Engineering Vol.1, Dunod, pp. 209-219 (1980).

[18] D. M. Young and K. C. Jea, *Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods,* Linear Algebra and its Appl. 34, pp. 159-194 (1980).

Table 2.1 Comparisons of Work per Iteration and Storage Requirement of Different Algorithm.

• *type*	*ORTHOMIN(q)*	*BCG*	*CGS*
work/iteration	(3q+4)n, Av	7n, Av, A^Tv	8n, $2 \times Av$
storage	(2q+4)n	6n	7n

Table 4.1 *Example 1.* Number of Iterations and Number of Multiplications for Convergence ($\epsilon = 10^{-14}$) When Applied to ILU Preconditioning ($h = 1/40$).

β	*algorithm*					
	BCG		CGS		ORTHOMIN(1)	
	Iters.	Mults.	Iters.	Mults.	Iters.	Mults.
10	63	2547909	40	1678560	119	3039855
100	33*	1334619	19	797316	39	996255
1000	14*	566202	9	377676	18	459810

*The iteration of BCG method broke down at the point of $\epsilon = 10^{-12}$.

Table 4.2 *Example 1.* Number of Iterations and Number of Multiplications for Convergence ($\epsilon = 10^{-14}$) When Applied to MILU Preconditioning ($h = 1/40$).

β	*algorithm*					
	BCG		CGS		ORTHOMIN(1)	
	Iters.	Mults.	Iters.	Mults.	Iters.	Mults.
10	37	1496391	22	923208	47	1200615
100	26	1051518	13	545532	28	715260
1000	14*	566202	8	335712	15	383175

*The iteration of BCG method broke down at the point of $\epsilon = 10^{-12}$.

Table 4.3 *Example 2.* Number of Iterations and Number of Multiplications for Convergence ($\epsilon = 10^{-14}$) When Applied to ILU Preconditioning ($h = 1/40$).

β	*algorithm*					
	BCG		CGS		ORTHOMIN(1)	
	Iters.	Mults.	Iters.	Mults.	Iters.	Mults.
10	64	2588352	41	1720524	251	6411795
100	41†	1658163	20	839280	48	1226160
1000	18	727974	10	419640	20	510900

†The iteration of BCG method broke down at the point of $\epsilon = 10^{-11}$.

Table 4.4 *Example 2.* Number of Iterations and Number of Multiplications for Convergence ($\epsilon = 10^{-14}$) When Applied to MILU Preconditioning ($h = 1/40$).

β	*algorithm*					
	BCG		CGS		ORTHOMIN(1)	
	Iters.	Mults.	Iters.	Mults.	Iters.	Mults.
10	39	1577277	24	1007136	84	2145780
100	29	1172847	14	587496	30	766350
1000	19	768417	8	335712	16	408720

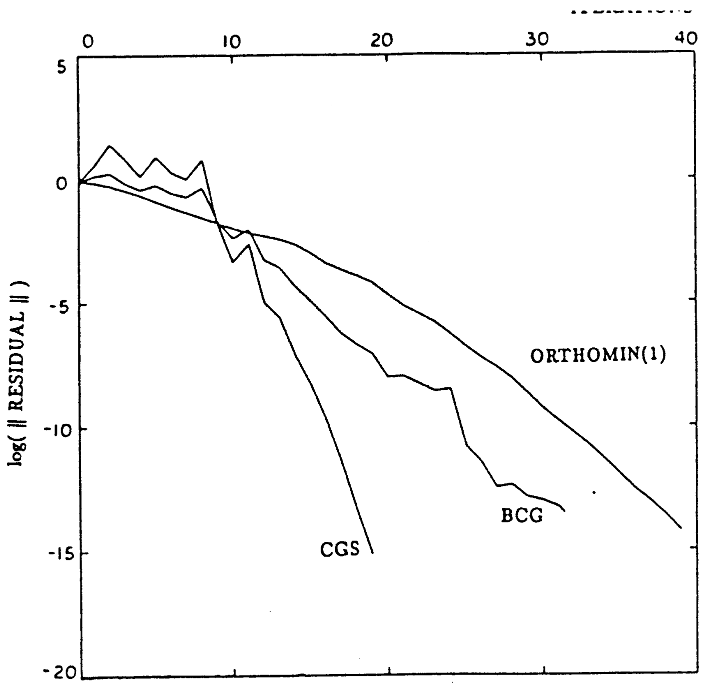

Fig.4.1 *Example 1*, $h = 1/40, \beta = 100$, *ILU* preconditioning

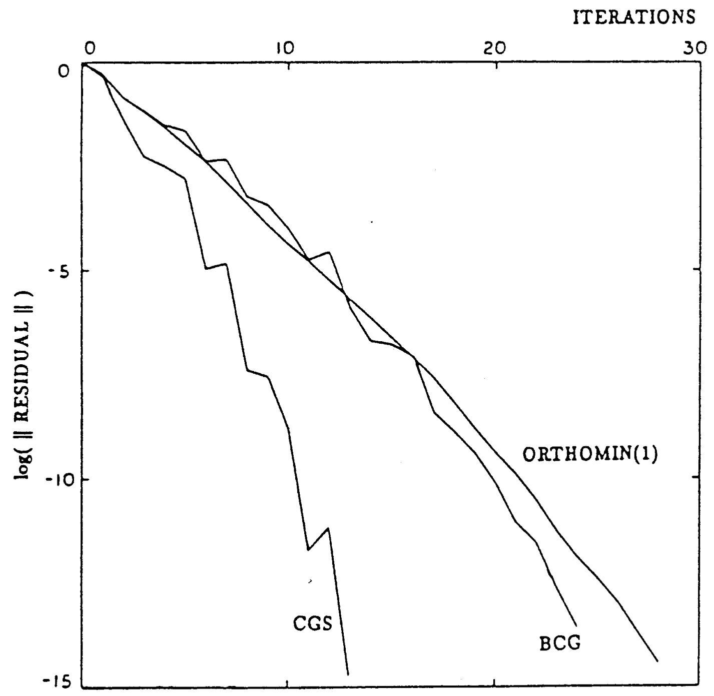

Fig.4.2 *Example 1*, $h = 1/40, \beta = 100$, *MILU* preconditioning

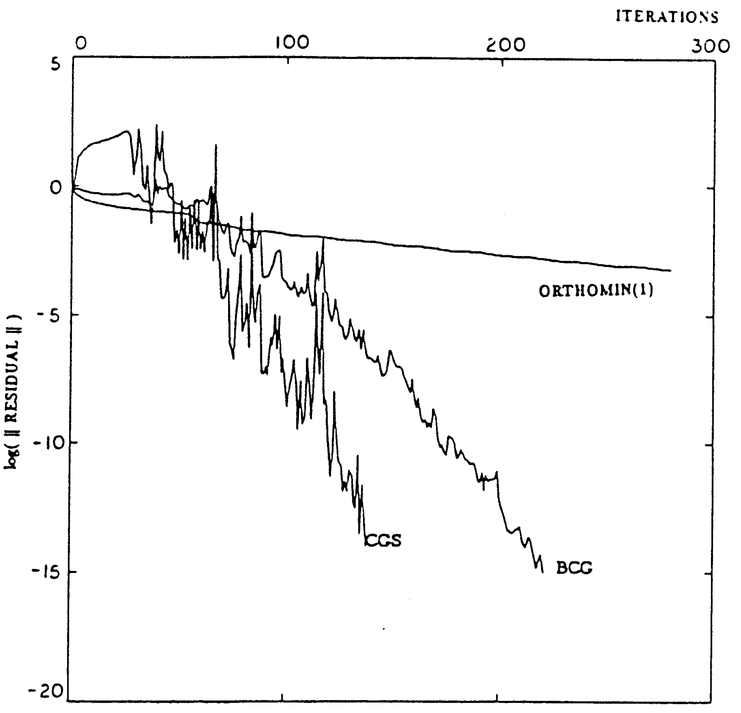

Fig.4.3 *Example 2*, $h = 1/40, \beta = 10$, nonpreconditioning

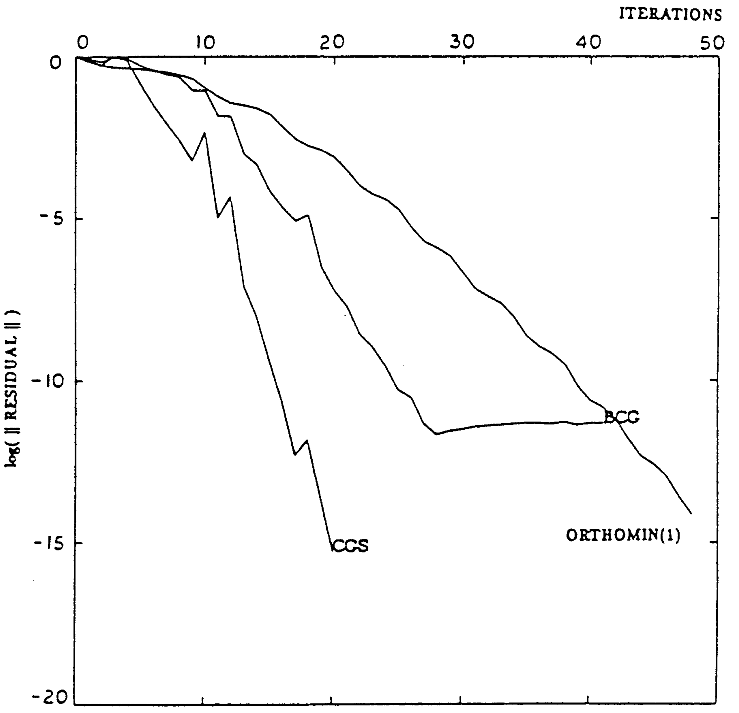

Fig.4.4 *Example 2*, $h = 1/40, \beta = 100$, *ILU* preconditioning