Splitting Correction Preconditioner for Linear Systems that Arise from Periodic Boundary Problems

Shoji ITOH* Shao-Liang ZHANG†
Yoshio OYANAGI‡ Makoto NATORI§

Jun 1, 2000

ISE-TR-00-170

*itosho@nalab.is.tsukuba.ac.jp:
Doctoral Program in Engineering, on Information Sciences and Electronics,
University of Tsukuba.
Ten-no dai, Tsukuba, Ibaraki, 305-8573, Japan.
†zhang@zzz.t.u-tokyo.ac.jp:
Department of Applied Physics,
University of Tokyo.
Hongo, Bunkyo, Tokyo, 113-8656, Japan.
‡oyanagi@is.s.u-tokyo.ac.jp:
Department of Information Science,
University of Tokyo.
Hongo, Bunkyo, Tokyo, 113-0033, Japan.
§natori@is.tsukuba.ac.jp:
Institute of Information Sciences and Electronics,
University of Tsukuba.
Ten-no dai, Tsukuba, Ibaraki, 305-8573 Japan.

Splitting Correction Preconditioner for Linear Systems that Arise from Periodic Boundary Problems*

Shoji ITOH† Shao-Liang ZHANG‡
Yoshio OYANAGI§ Makoto NATORI¶

Jun 1, 2000

Abstract

In this paper, a new preconditioner for solving periodic boundary systems based on block factorization is proposed. When the two-dimensional partial differential equations (briefly PDEs) with periodic boundary conditions for x-direction are discreted by the finite difference method, the linear systems with periodic boundary matrix elements (briefly PBEs) in diagonal blocks are obtained. For these systems, the basic ideas of this preconditioner are splitting PBEs from the diagonal block matrix by a rank-1 correction with the Sherman-Morrison formula. Several numerical results show this preconditioner gives faster convergence than the conventional one.

AMS subject classificaion: 65F10.
Key words: block preconditioning, Sherman-Morrison formula.

1 Introduction

The discretization of PDEs by finite difference method leads to large and sparse linear systems

$$Ax = b. \tag{1.1}$$

Especially, under the periodic boundary conditions, the coefficient matrix A has PBEs (In this research, we call so) . The PBEs means the matrix elements corresponding to the periodic boundary conditions generated by discreting the PDEs. To solve these systems, various preconditioned iterative methods are applied to. For example, the preconditioned conjugate gradient (PCG) method [8] is used for symmetric systems and the preconditioned BiCGSTAB [11] for nonsymmetric systems. Here, preconditioners should be chosen to attain fast convergence. The "preconditioning" means the transformation of a linear system to a nearby system which is easier to solve than the given one [2][6]. When deciding on which preconditioner K to use, several issues must be considered, of which the most important are (i) resemblance between K^{-1} and A^{-1}, (ii) cost of construction K and (iii) cost of computing $r' = K^{-1}r$ [2]. Here, the decided preconditioner $K = K_L K_R$ is applied to (1.1), such that the preconditioned system

$$(K_L^{-1} A K_R^{-1})(K_R x) = (K_L^{-1} b) \tag{1.2}$$

is converged faster than the given system (1.1).

In this paper, a new preconditioner for linear systems that arise from PDEs with periodic boundary conditions is proposed. This preconditioner is based on the block preconditionings [1][4], and algebraically equivalent to the complete factorization to block diagonal matrices. Consequently, the

*ISE-TR-00-170: ISE-Technical Report, University of Tsukuba, Japan. Preprint of the International Conference on NUMERICAL MATHEMATICS on the 40th Anniversary of the journal BIT, August 9-12, 2000, Lund, Sweden.
†Doctoral Program in Engineering, on Information Sciences and Electronics, University of Tsukuba, Japan.
‡Department of Applied Physics, University of Tokyo, Japan.
§Department of Information Science, University of Tokyo, Japan.
¶Institute of Information Sciences and Electronics, University of Tsukuba, Japan.

convergence of the iterative methods with this preconditioner is improved much more than the incomplete factorization to diagonal block. Nevertheless, its calculating cost increases only a little for the incomplete one.

In section 2, physical model, the linear system and some typical solvers are presented.

In section 3, the conventional preconditioners are presented.

In section 4, a new preconditioner for linear systems that arise from periodic boundary problems is proposed, and in section 5, some numerical results show that this preconditioner improves the convergence and reduces the calculating time.

2 Model Problem and Numerical Solving

In this research, the diffusion-convection equation (2.1) in two-dimensional unit square domain $\Omega = [0,1] \times [0,1]$

$$-\Delta u + v_x \frac{\partial u}{\partial x} + v_y \frac{\partial u}{\partial y} = f, \qquad (x,y) \in [0,1] \times (0,1), \tag{2.1}$$

$$u(0,y) = u(1,y), \qquad \text{(periodic boundary conditions)}, \tag{2.2}$$

$$u(x,0) = g_0, \qquad \text{(Dirichlet conditions)}, \tag{2.3}$$

$$u(x,1) = g_1, \qquad \text{(Dirichlet conditions)} \tag{2.4}$$

is discussed. Here, v_x is the x-direction's convection term and v_y is the y's one.

2.1 Coefficient Matrix

Eq.(2.1) is discreted by five-points central differences, with stepsize $h = 1/(n+1)$ in each direction. Then, the coefficient matrix A is a sparse nonsymmetric matrix of size $n(n+1) \times n(n+1)$ with PBEs.

$$A = \begin{bmatrix} P_1 & C_1 & & & 0 \\ B_2 & P_2 & C_2 & & \\ & \ddots & \ddots & \ddots & \\ & & B_{n-1} & P_{n-1} & C_{n-1} \\ 0 & & & B_n & P_n \end{bmatrix}, \tag{2.5}$$

where P_l, B_l, C_l $(l = 1, 2, \cdots, n)$ are block matrices whose size is $(n+1) \times (n+1)$.

The block diagonal matrix is

$$P_l = \begin{bmatrix} d_1^{(l)} & b_1^{(l)} & & 0 & p_1^{(l)} \\ a_2^{(l)} & d_2^{(l)} & b_2^{(l)} & & \\ & \ddots & \ddots & \ddots & \\ & & a_{m-1}^{(l)} & d_{m-1}^{(l)} & b_{m-1}^{(l)} \\ q_m^{(l)} & 0 & & a_m^{(l)} & d_m^{(l)} \end{bmatrix}, \tag{2.6}$$

which is tridiagonal matrix with PBEs. Here, $p_1^{(l)}, q_m^{(l)}$ are PBEs of the l-th block, and $m = n+1$. B_l, C_l are diagonal matrices.

2.2 Iterative Methods

To these systems, the iterative methods with preconditioning are often used. When the coefficient matrix is symmetric, usually the PCG method is adopted [8, 9] (Algorithm 1).

x_0 : is initial guess,
$r_0 = b - Ax_0$, $p_0 = K^{-1}r_0$,
for $k = 0, 1, ...,$ until $\|r_k\| \leq \varepsilon \| b \|$ do:
begin

$$\begin{aligned}
\alpha_k &= \frac{(K^{-1}r_k, r_k)}{(p_k, Ap_k)}, \\
x_{k+1} &= x_k + \alpha_k p_k, \\
r_{k+1} &= r_k - \alpha_k A p_k, \\
\beta_k &= \frac{(K^{-1}r_{k+1}, r_{k+1})}{(K^{-1}r_k, r_k)}, \\
p_{k+1} &= K^{-1}r_{k+1} + \beta_k p_k,
\end{aligned}$$

end

Algorithm 1 : Preconditioned CG

When the coefficient matrix is nonsymmetric, the PBiCGSTAB method is one of the most effective[11] (Algorithm 2).

x_0 : is initial guess,
$r_0^* = K^{-1}(b - Ax_0)$,
$p_0 = t_0 = r_0 = r_0^*$,
for $k = 0, 1, ...,$ until $\|r_k\| \leq \varepsilon \| b \|$ do:
begin

$$\begin{aligned}
\alpha_k &= \frac{(r_0^*, r_k)}{(r_0^*, K^{-1}Ap_k)}, \\
t_k &= r_k - \alpha_k K^{-1} A p_k, \\
\zeta_k &= \frac{(K^{-1}At_k, t_k)}{(K^{-1}At_k, K^{-1}At_k)}, \\
x_{k+1} &= x_k + \alpha_k p_k + \zeta_k t_k, \\
r_{k+1} &= t_k - \zeta_k K^{-1} A t_k, \\
\beta_k &= \frac{\alpha_k (r_0^*, r_{k+1})}{\zeta_k (r_0^*, r_k)}, \\
p_{k+1} &= r_{k+1} + \beta_k (p_k - \zeta_k K^{-1} A p_k),
\end{aligned}$$

end

Algorithm 2 : Preconditioned Bi-CGSTAB

In these algorithms, the matrix K is the preconditioner. Usually, the preconditioning step is calculated as solving the system, $s' = K^{-1}s$ in each iteration step. This nearby problem is solved repeatedly (with appropriately chosen right-hand-side vectors b) in such a way that the solution to the original problem can be obtained in the limit[3].

The details of the preconditioner are explained in the next section.

3 Conventional Preconditioning

3.1 Block Preconditioner

The block preconditioner K is defined as follows:

$$K = \begin{bmatrix} K_1 & & & 0 \\ & K_2 & & \\ & & \ddots & \\ 0 & & & K_n \end{bmatrix}, \tag{3.1}$$

where K_l's are block matrices with size $(n+1) \times (n+1)$.

By block preconditioning for (2.5), K_l is assigned for diagonal block (2.6) [4]. Usually K_l is generated with the incomplete Cholesky (IC) factorization for symmetric, or the incomplete LU (ILU) factorization for nonsymmetric in order to decrease its calculating costs[5]. In other words, the incomplete factorization without *fill-in* is done. Fill-in is the behavior that the zero elements of original matrix change into non-zero by an exact factorization. That is to say,

$$K_l^{-1} \approx P_l^{-1}. \tag{3.2}$$

Then, the factorization is as follows:

$$K_l = \begin{bmatrix} \delta_1^{(l)} & & & & \\ a_2^{(l)} & \delta_2^{(l)} & & 0 & \\ & \ddots & \ddots & & \\ & & a_{m-1}^{(l)} & \delta_{m-1}^{(l)} & \\ q_m^{(l)} & & & a_m^{(l)} & \delta_m^{(l)} \end{bmatrix} \begin{bmatrix} \delta_1^{(l)} & b_1^{(l)} & & & p_1^{(l)} \\ & \delta_2^{(l)} & b_2^{(l)} & & \\ & & \ddots & \ddots & \\ 0 & & & \delta_{m-1}^{(l)} & b_{m-1}^{(l)} \\ & & & & \delta_m^{(l)} \end{bmatrix}. \tag{3.3}$$

On the other hand, from a convergence point of view,

$$K_l^{-1} = P_l^{-1} \tag{3.4}$$

brings better convergence than (3.2) [4]. Namely this is algebraically equivalent to the complete factorization to diagonal blocks.

$$K_l = \begin{bmatrix} \delta_1^{(l)} & & & & \\ a_2^{(l)} & \delta_2^{(l)} & & 0 & \\ & \ddots & \ddots & & \\ & & a_{m-1}^{(l)} & \delta_{m-1}^{(l)} & \\ q_1^{(l)} & \star & \star & \varrho_m^{(l)} & \delta_m^{(l)} \end{bmatrix} \begin{bmatrix} \delta_1^{(l)} & b_1^{(l)} & & & p_1^{(l)} \\ & \delta_2^{(l)} & b_2^{(l)} & & \star \\ & & \ddots & \ddots & \star \\ 0 & & & \delta_{m-1}^{(l)} & \rho_{m-1}^{(l)} \\ & & & & \delta_m^{(l)} \end{bmatrix}, \tag{3.5}$$

where "$\star$" means the non-zero element by fill-in.

However, this costs more flops.

4 New Preconditioner

In this research, a new preconditioner is proposed. This preconditioner satisfies eq.(3.4), but the cost of this new preconditioner increases little to one of the incomplete factorization.

On one-dimensional periodic boundary problems, the way for the direct method had always been proposed[12]. In this research, this way is developed as a preconditioner of two-dimensional systems. The detail is as follows:

4.1 Splitting PBEs as the Correction Term

Firstly, PBEs are split from the original K_l. Then the first and the last elements of the diagonal are corrected.

$$
\begin{aligned}
K_l &= \begin{bmatrix} d_1^{(l)} - p_1^{(l)} & b_1^{(l)} & & & 0 \\ a_2^{(l)} & d_2^{(l)} & b_2^{(l)} & & \\ & \ddots & \ddots & \ddots & \\ & & a_{m-1}^{(l)} & d_{m-1}^{(l)} & b_{m-1}^{(l)} \\ 0 & & & a_m^{(l)} & d_m^{(l)} - q_m^{(l)} \end{bmatrix} + \begin{bmatrix} p_1^{(l)} & 0 \dots 0 & p_1^{(l)} \\ 0 & 0 \dots 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 \dots 0 & 0 \\ q_m^{(l)} & 0 \dots 0 & q_m^{(l)} \end{bmatrix} \\
&= T_l + \begin{bmatrix} p_1^{(l)} \\ 0 \\ \vdots \\ 0 \\ q_m^{(l)} \end{bmatrix} \begin{bmatrix} 1\, 0 \cdots 0\, 1 \end{bmatrix} \\
&= T_l + u_l v_l^T. \qquad (4.1)
\end{aligned}
$$

Consequently, K_l is split as tridiagonal matrix and the rank-1 matrix made up with PBEs. The latter is represented as the correction term by two vectors' product.

By using (4.1), the residual vector r_l updated by preconditioning is represented as r'_l. That is

$$
r'_l = K_l^{-1} r_l = (T_l + u_l v_l^T)^{-1} r_l. \qquad (4.2)
$$

4.2 Applying to the Sherman-Morrison Formula

For solving eq.(4.2), the Sherman-Morrison formula [7]

$$
\begin{aligned}
(T + uv^T)^{-1} &= T^{-1} - T^{-1} u (1 + v^T T^{-1} u)^{-1} v^T T^{-1}, \qquad (4.3) \\
& \quad T \in R^{m \times m}, \quad u, v \in R^m
\end{aligned}
$$

is applied to. Then, eq.(4.2) can be rewritten as follows:

$$
\begin{aligned}
r'_l &= T_l^{-1} r_l - T_l^{-1} u_l (1 + v_l^T T_l^{-1} u_l)^{-1} v_l^T T_l^{-1} r_l \\
&= y_l - z_l \left(1 + v_l^T z_l\right)^{-1} v_l^T y_l \\
&= y_l - z'_l \begin{bmatrix} 1\, 0 \cdots 0\, 1 \end{bmatrix} \begin{bmatrix} y_1^{(l)} \\ \vdots \\ \vdots \\ \vdots \\ y_m^{(l)} \end{bmatrix} \\
&= y_l - (y_1^{(l)} + y_m^{(l)}) z'_l. \qquad (4.4)
\end{aligned}
$$

Here,

$$
\begin{aligned}
y_l &= T_l^{-1} r_l \qquad (4.5) \\
&\equiv \left[y_1^{(l)} y_2^{(l)} \cdots y_m^{(l)} \right]^T, \\
z_l &= T_l^{-1} u_l, \qquad (4.6) \\
z'_l &= z_l \left(1 + v_l^T z_l\right)^{-1}. \qquad (4.7)
\end{aligned}
$$

From the fact mentioned above, solving (4.2) is changed into solving (4.4). This calculating cost is less than the complete factorization for the diagonal block. Because, on the linear equations (4.5) (4.6),

those coefficient matrix T_l is tridiagonal without PBEs. Furthermore, eq.(4.6)(4.7) are sufficient to compute only one time at the first iteration step, because the u_l, v_l are invariant vectors for iterations. Therefore each iteration step's computation is to solve (4.5), and (4.4) with slightly amount of flop.

In this research, this preconditioner is named the "Splitting Correction (SC)".

On the ratio of flop per iteration, the incomplete factorization for the diagonal block is set `1.00`, then the SC is `1.18` and the complete factorization for the diagonal block is `1.55`.

5 Numerical Results

In numerical examination, some linear systems based on physical problem (2.1) was examined and the analysis solution was $u(x, y) = \sin(2\pi(x + y))$ in all cases.

For the coefficient matrix by discreting this problem, the comparison between the conventional incomplete factorization and the SC was done by evaluating of convergence and CPU time.

Experiments were carried out in double precision and executed on IBM RS/6000 SP 1PE RISC processor. Compiler is `xlf` based on Fortran77. In all cases, the iterations were started with $\boldsymbol{x}_0$=$\mathbf{0}$, and stopped when $\|\boldsymbol{r}_k\|_2/\|\boldsymbol{r}_0\|_2 \leq 10^{-12}$.

5.1 Symmetric Problems

The Poisson's equation was discussed. The respective parameters of eq.(2.1) were set as $v_x = 0.0$, $v_y = 0.0$. Since this coefficient matrix was symmetric, the CG method was adopted as solver and the IC factorization was compared as the conventional preconditioner. Namely, the effect of block preconditioning by the conventional IC and the SC was compared. Calculating models were $65 \times 64, 97 \times 96$ and 129×128 (grid size).

Figure 1$\sim$ 3 are the performance of *log* scaled relative residual 2-norm.

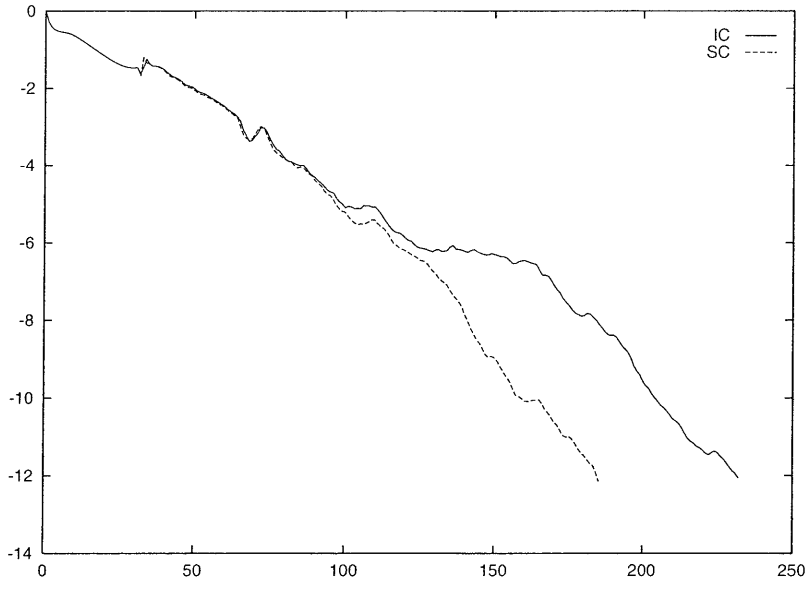

Figure 1: Peformance of convergence between IC and SC preconditioner (65×64).

Table 1 shows a summary of the iteration number and CPU time for each model.

These results show the using SC had faster convergence than the using conventional IC preconditioner. On the calculating time, the using SC is about 20% shorter.

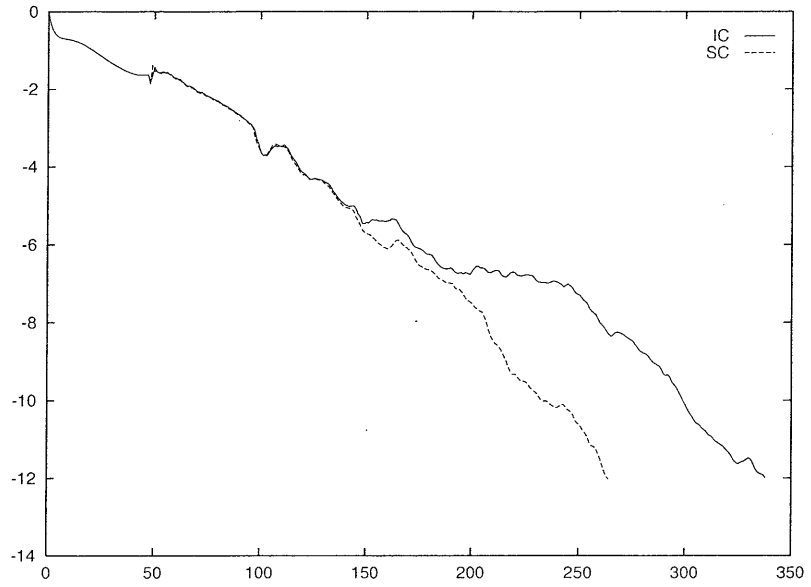

Figure 2: Peformance of convergence between IC and SC preconditioner (97×96).

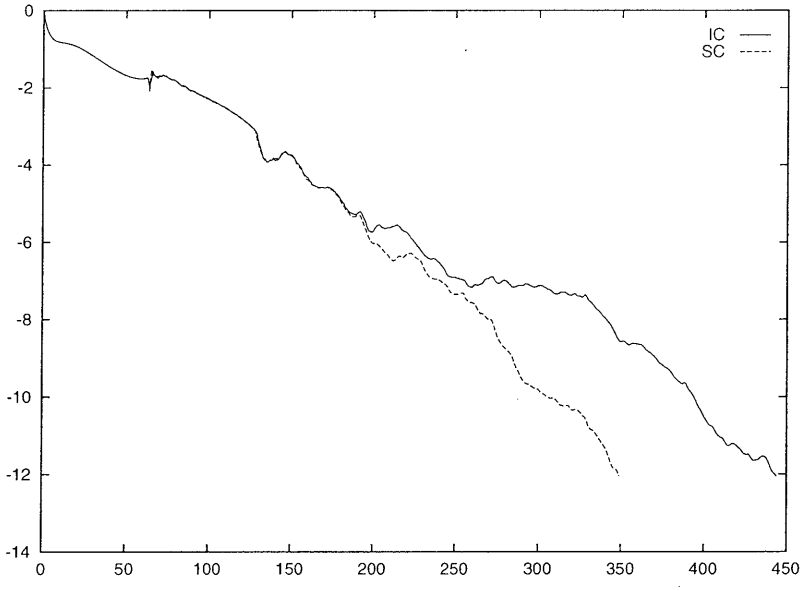

Figure 3: Peformance of convergence between IC and SC preconditioner (129×128).

Table 1: The number of iterations until convergence for symmetric problem (CPU time[sec]).

grid size	65×64	97×96	129×128
IC	232(1.43)	338(4.53)	444(10.23)
SC	185(1.15)	264(3.56)	349(8.32)
SC/IC [%]	79.7(80.4)	78.1(78.6)	78.6(81.3)

5.2 Nonsymmetric Problems

The diffusion-convection equation was discussed. The respective parameters of eq.(2.1) were set as v_x = (0.0, 0.1, 0.5, 1.0, 5.0, 10.0) , v_y = (0.0, 0.1, 0.5, 1.0, 5.0, 10.0). Since these coefficient matrices were nonsymmetric, the BiCGSTAB method was adopted as solver and the ILU factorization was compared as the conventional preconditioner. Namely, the effect of block preconditioning by the conventional ILU and the SC was compared. Calculating models were 65×64 (grid size) in all cases.

Figure 4$\sim$ 6 are the performance of *log* scaled relative residual 2-norm about (v_x, v_y) = (0.0, 1.0), (1.0, 0.0), (1.0, 1.0) cases and others are shown at Appendix A.

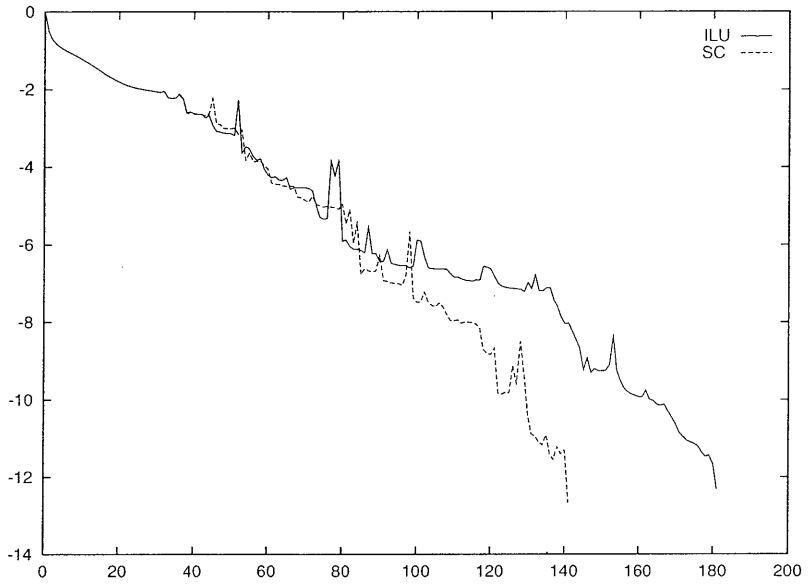

Figure 4: Peformance of convergence between ILU and SC preconditioner ($v_x = 0.0, v_y = 1.0$).

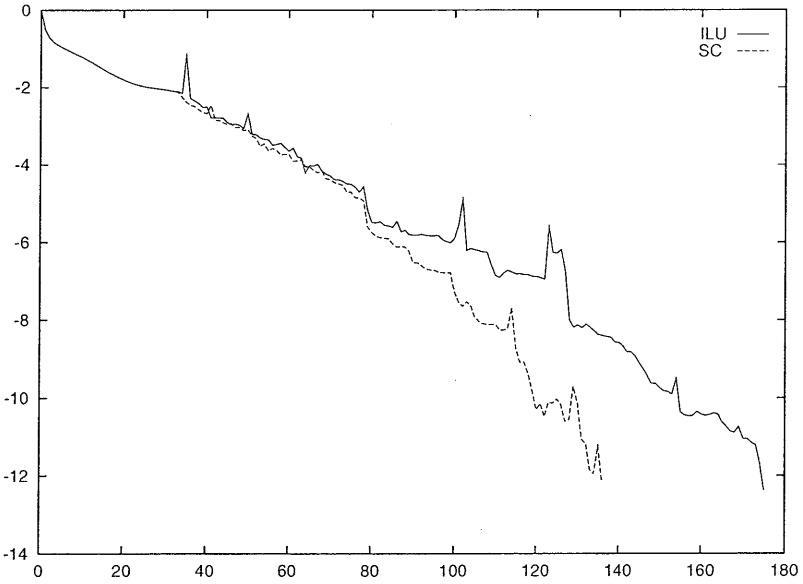

Figure 5: Peformance of convergence between ILU and SC preconditioner ($v_x = 1.0, v_y = 0.0$).

Table 2 shows a summary of the iteration number and CPU time for each model.

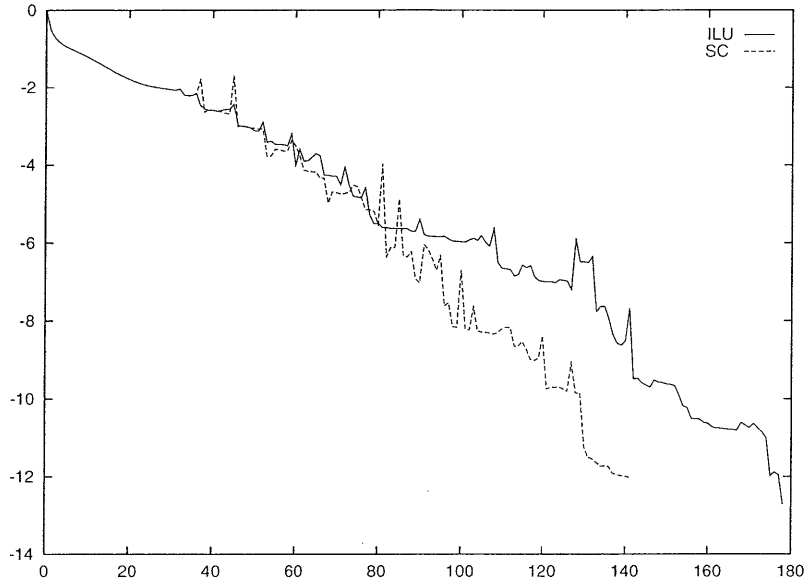

Figure 6: Peformance of convergence between ILU and SC preconditioner ($v_x = 1.0, v_y = 1.0$).

Table 2: The number of iterations until convergence for nonsymmetric problem (CPU time[sec]).

grid size	65×64	65×64	65×64
(v_x, v_y)	$(0.0, 1.0)$	$(1.0, 0.0)$	$(1.0, 1.0)$
ILU	181(4.37)	175(4.21)	178(4.32)
SC	141(3.43)	136(3.37)	141(3.49)
SC/ILU[%]	77.9(78.5)	77.7(80.0)	79.2(80.8)

These results show the using SC had faster convergence than the using conventional ILU preconditioner. On the calculating time, the using SC is about 20% shorter.

6 Conclusion

In this paper, we have proposed a new preconditioner "Splitting Correction (SC)" that is for solving the linear systems that arise from periodic boundary problems.

This preconditioner is based on the idea of the blockwise complete factorization. This method costs less than the blockwise complete one and increases only a little more than the blockwise incomplete one. Some numerical results show the SC improves the convergence than the using incomplete Cholesky factorization.

Acknowledgements

This work is supported by Center for Computational Science and Energy of JAERI (Japan Atomic Energy Research Institute) in 1997 – 1999.

References

[1] O. Axelsson, Incomplete Block Matrix Factorization Preconditioning Methods. The Ultimate Answer?, *J. Comp. Appl. Math.*, 12&13, pp. 3–18 (1985).

[2] A. M. Bruaset, *A Survey of Preconditioned Iterative Methods*, Longman Scientific & Technical (1995).

[3] T. F. Chan and H. A. Van der Vorst, Approximate and Incomplete Factorizations., In D. E. Keyes, A. Sameh, and V. Venkatakrishnan, editors, *Parallel Numerical Algorithms* , ICASE/LaRC Interdisciplinary Series in Science and Engineering, pp. 167–202. Kluwer, Dordrecht, (1997).

[4] P. Concus, G. H. Golub and G. Meurant, Block Preconditioning for the Conjugate Gradient Method,*SIAM J. Sci. Stat. Comput.*, 6, pp. 220–252 (1985).

[5] S. C. Eisenstat, Efficient Implementation of a Class of Preconditioned Conjugate Gradient Methods, *SIAM J. Sci. Stat. Comput.*, 2, pp. 1–4 (1981).

[6] G. H. Golub and D. P. O'Leary, Some History of the Conjugate Gradient and Lanczos Algorithms: 1948–1976,*SIAM Review*, 31, pp. 50–102 (1989).

[7] G. H. Golub and D. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press (1996).

[8] M. R. Hestenes and E. Stiefel, Methods of Conjugate Gradients for Solving Linear Systems, *J. Res. Nat. Bur. Standards* , 49, pp. 409–435 (1952).

[9] J. A. Meijerink and H. A. Van der Vorst, An Iterative Solution Method for Linear Systems of Which the Coefficient Matrix is a Symmetric *M*-Matrix, *Math. Comput.*, 31, pp. 148–162, (1977).

[10] A. M. Turing, Rounding-off Errors in Matrix Process, *Quart. J. of Mech. and Appl. Math.*, 1, pp. 287–308 (1948).

[11] H. A. Van der Vorst, Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comput.*, 13, pp. 631–644 (1992).

[12] M. Yarrow, Solving Periodic Block Tridiagonal Systems Using the Sherman-Morrison-Woodbury Formula, *AIAA 9th Computational Fluid Dynamics Conf.*, Jun. 13–15, pp. 188–196 (1989).

A Other Numerical Results for the Nonsymmetric Problems

In this section, the several numerical results for eq.(2.1) are shown and are size of 65×64 (grid size). In these case, the parameters were set as v_x = 0.0, 0.1, 0.5, 1.0, 5.0, 10.0, v_y = 0.0, 0.1, 0.5, 1.0, 5.0, 10.0.

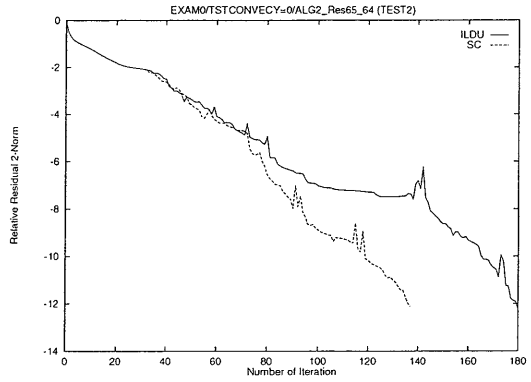

Figure 7: $65\times64 : v_x = 0.0, v_y = 0.0$

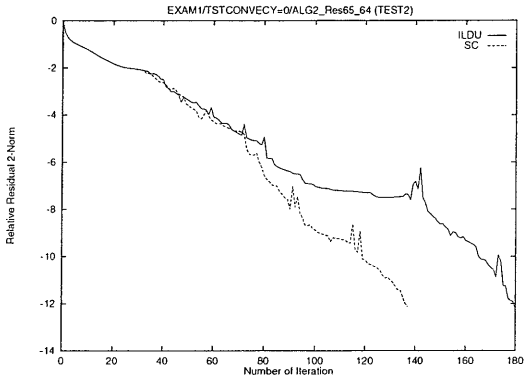

Figure 8: $65\times64 : v_x = 0.1, v_y = 0.0$

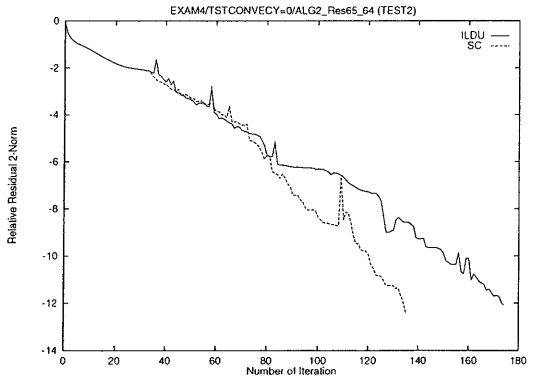

Figure 9: $65\times64 : v_x = 0.5, v_y = 0.0$

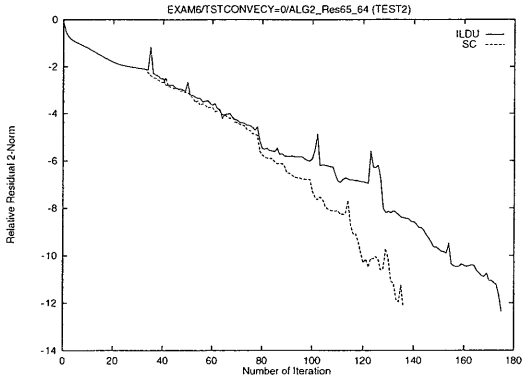

Figure 10: $65\times64 : v_x = 1.0, v_y = 0.0$

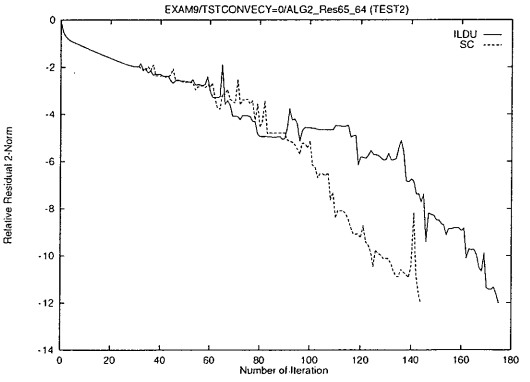

Figure 11: $65\times64 : v_x = 5.0, v_y = 0.0$

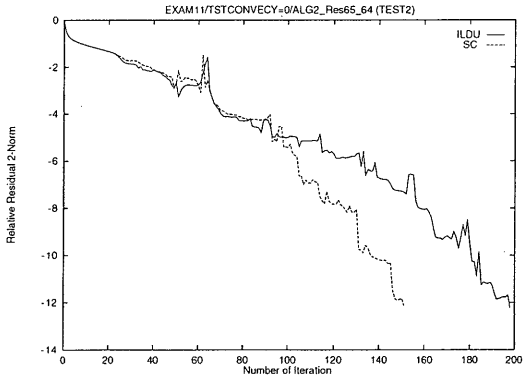

Figure 12: $65\times64 : v_x = 10.0, v_y = 0.0$

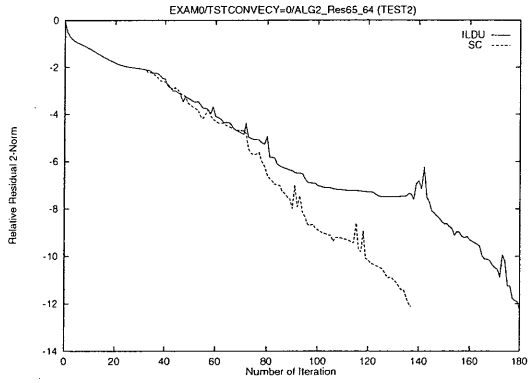

Figure 13: $65\times64 : v_x = 0.0, v_y = 0.0$

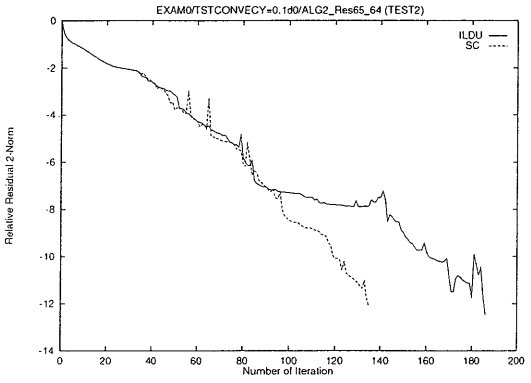

Figure 14: $65\times64 : v_x = 0.0, v_y = 0.1$

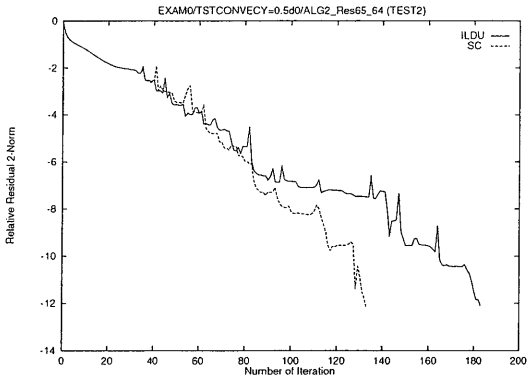

Figure 15: $65\times64 : v_x = 0.0, v_y = 0.5$

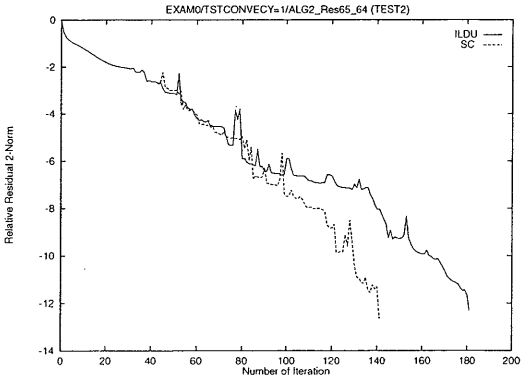

Figure 16: $65\times64 : v_x = 0.0, v_y = 1.0$

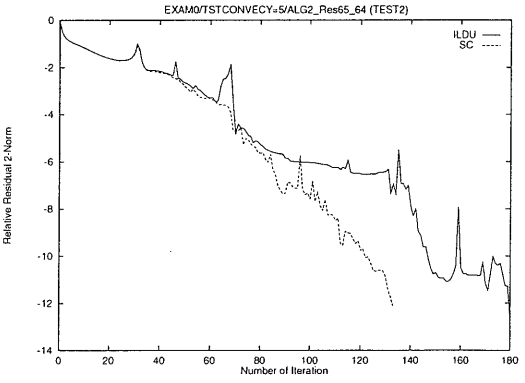

Figure 17: $65\times64 : v_x = 0.0, v_y = 5.0$

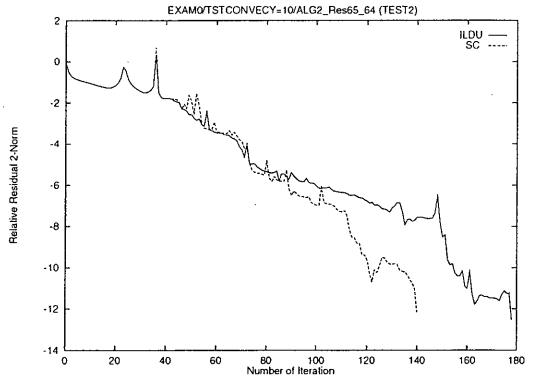

Figure 18: $65\times64 : v_x = 0.0, v_y = 10.0$

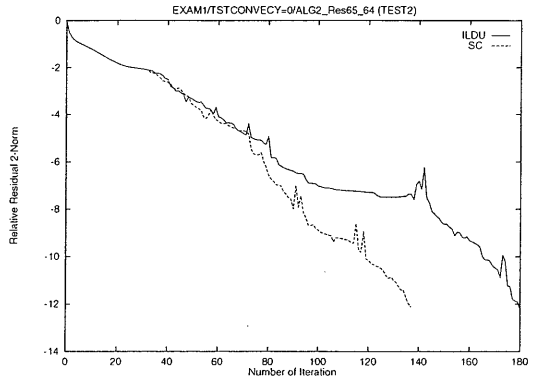

Figure 19: $65\times64 : v_x = 0.1, v_y = 0.0$

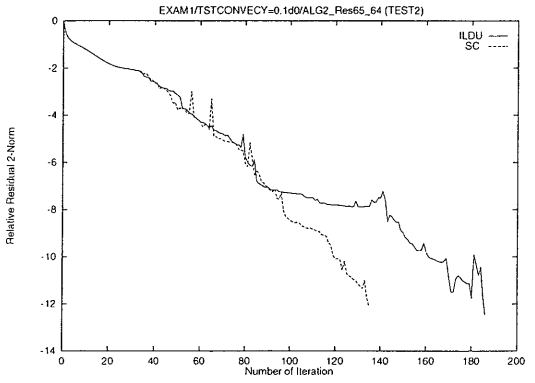

Figure 20: $65\times64 : v_x = 0.1, v_y = 0.1$

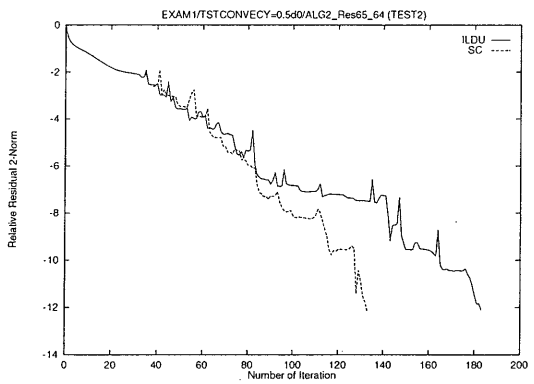

Figure 21: $65\times64 : v_x = 0.1, v_y = 0.5$

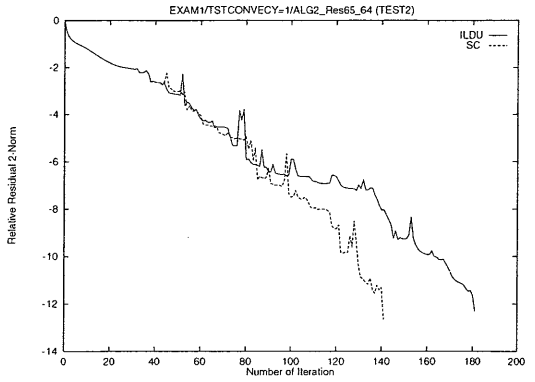

Figure 22: $65\times64 : v_x = 0.1, v_y = 1.0$

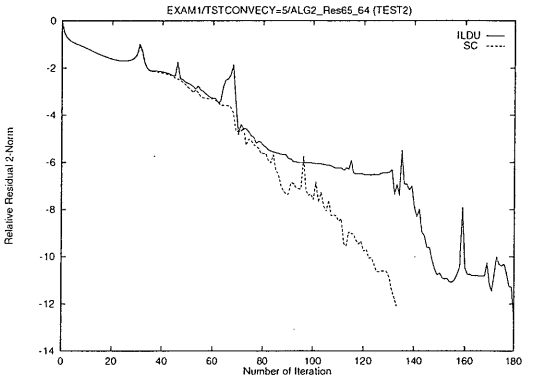

Figure 23: $65\times64 : v_x = 0.1, v_y = 5.0$

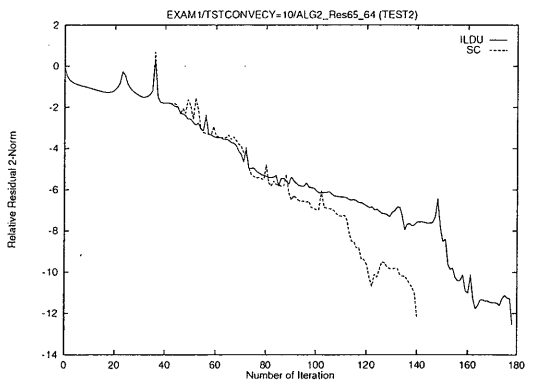

Figure 24: $65\times64 : v_x = 0.1, v_y = 10.0$

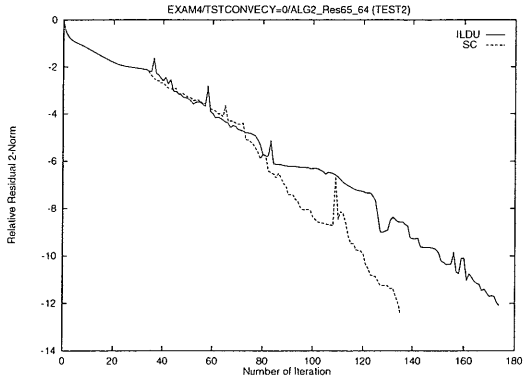

Figure 25: $65\times64 : v_x = 0.5, v_y = 0.0$

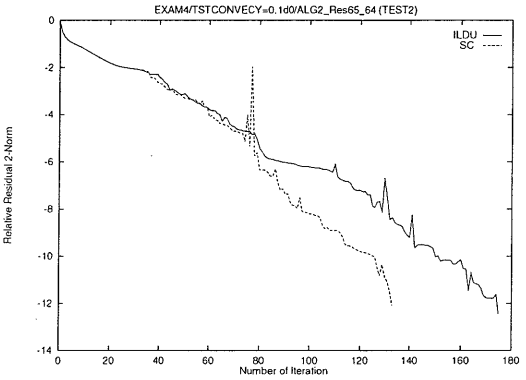

Figure 26: $65\times64 : v_x = 0.5, v_y = 0.1$

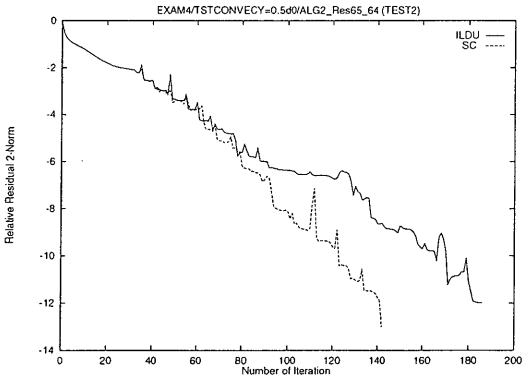

Figure 27: $65\times64 : v_x = 0.5, v_y = 0.5$

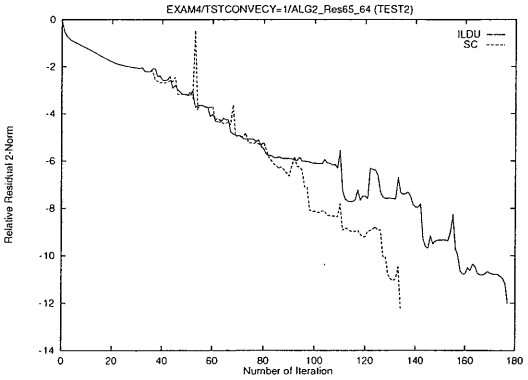

Figure 28: $65\times64 : v_x = 0.5, v_y = 1.0$

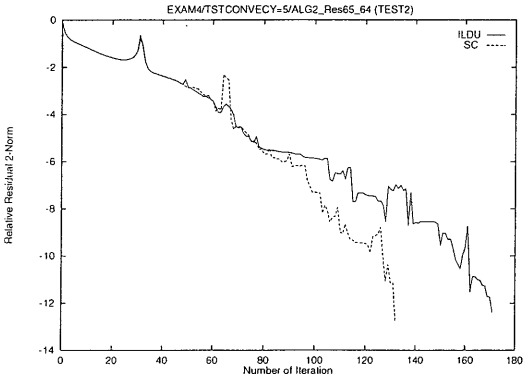

Figure 29: $65\times64 : v_x = 0.5, v_y = 5.0$

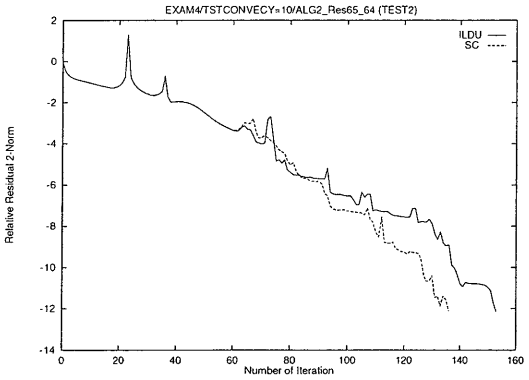

Figure 30: $65\times64 : v_x = 0.5, v_y = 10.0$

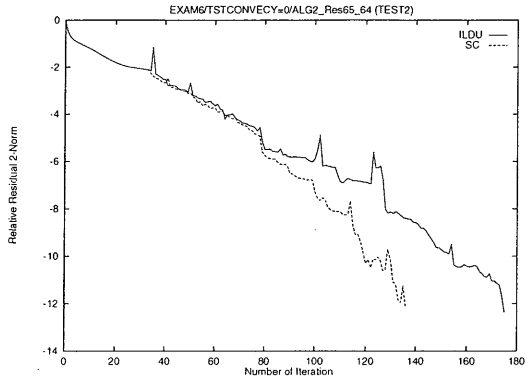

Figure 31: $65\times64 : v_x = 1.0, v_y = 0.0$

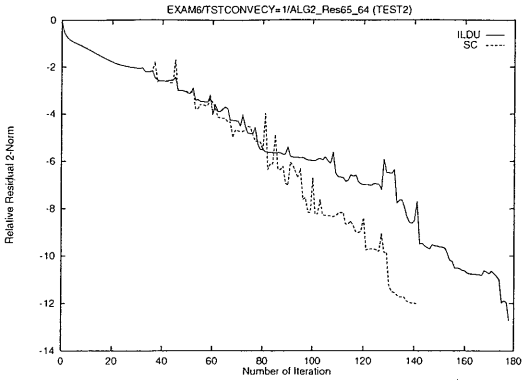

Figure 34: $65\times64 : v_x = 1.0, v_y = 1.0$

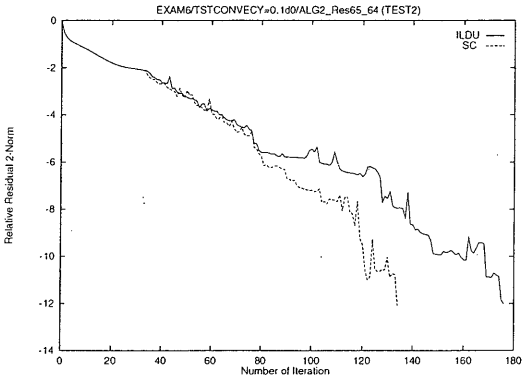

Figure 32: $65\times64 : v_x = 1.0, v_y = 0.1$

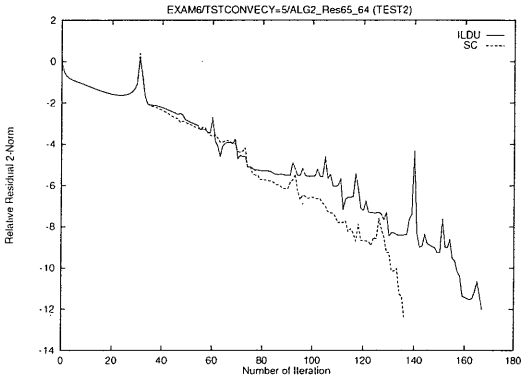

Figure 35: $65\times64 : v_x = 1.0, v_y = 5.0$

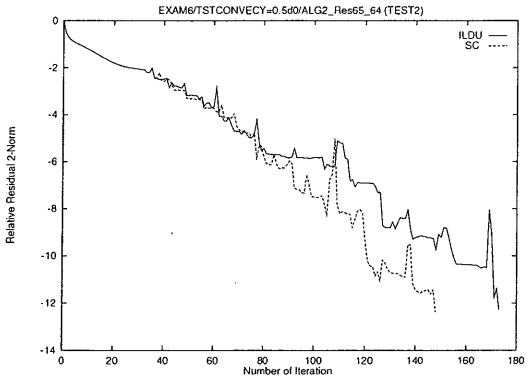

Figure 33: $65\times64 : v_x = 1.0, v_y = 0.5$

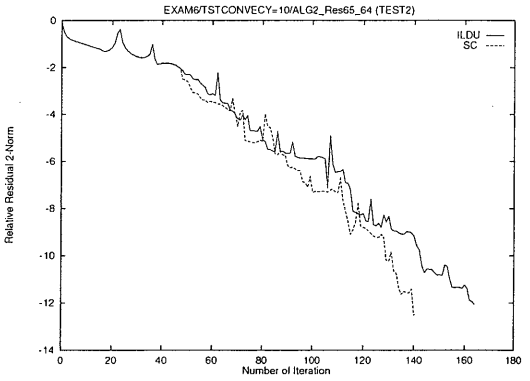

Figure 36: $65\times64 : v_x = 1.0, v_y = 10.0$

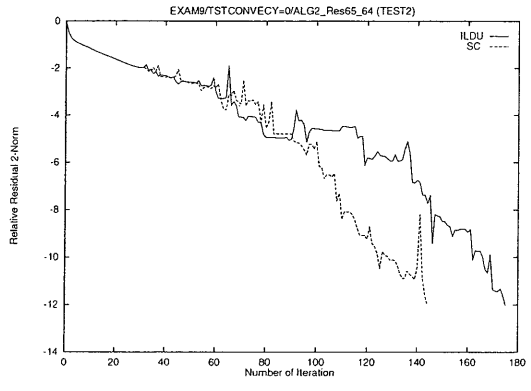

Figure 37: $65\times64 : v_x = 5.0, v_y = 0.0$

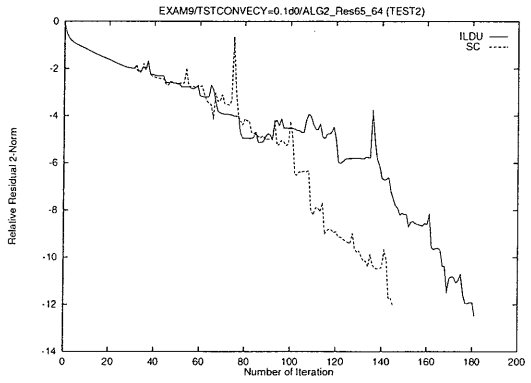

Figure 38: $65\times64 : v_x = 5.0, v_y = 0.1$

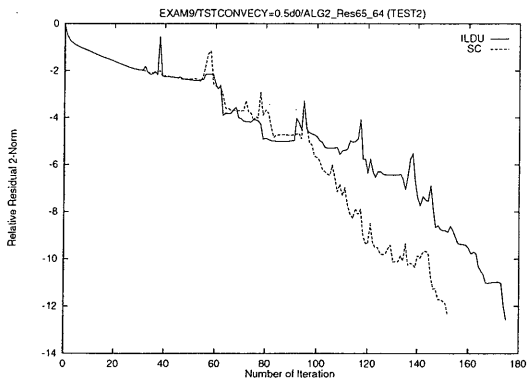

Figure 39: $65\times64 : v_x = 5.0, v_y = 0.5$

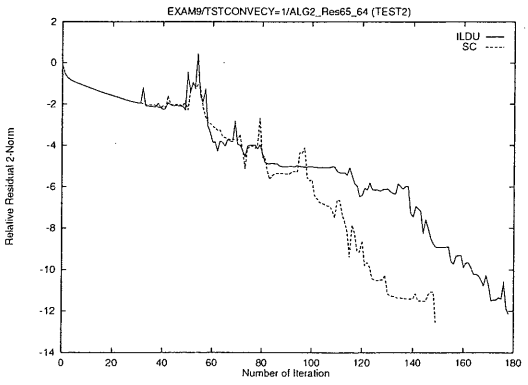

Figure 40: $65\times64 : v_x = 5.0, v_y = 1.0$

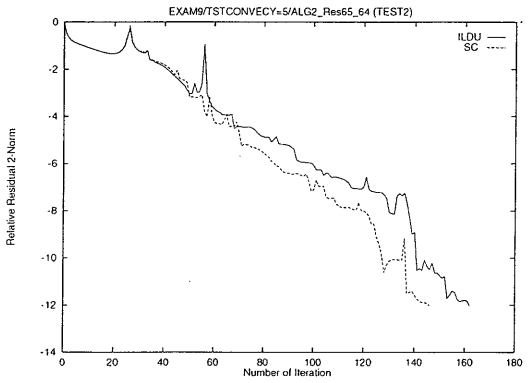

Figure 41: $65\times64 : v_x = 5.0, v_y = 5.0$

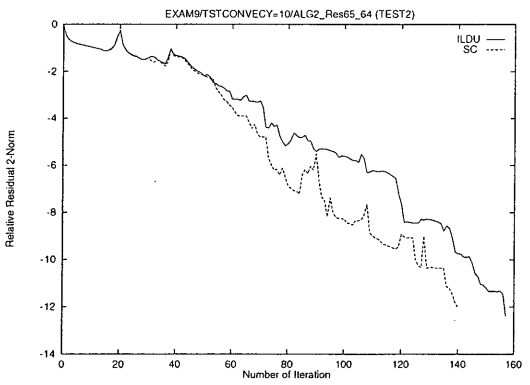

Figure 42: $65\times64 : v_x = 5.0, v_y = 10.0$

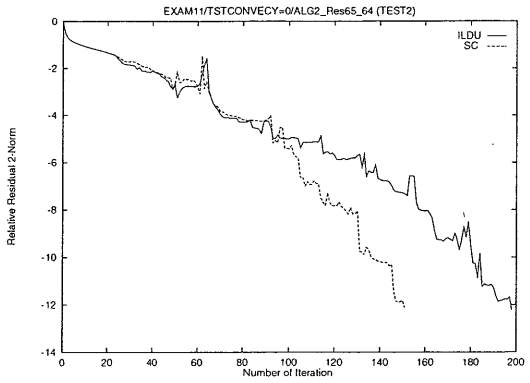

Figure 43: $65\times64 : v_x = 10.0, v_y = 0.0$

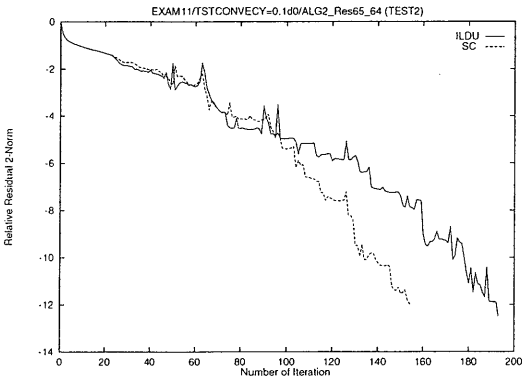

Figure 44: $65\times64 : v_x = 10.0, v_y = 0.1$

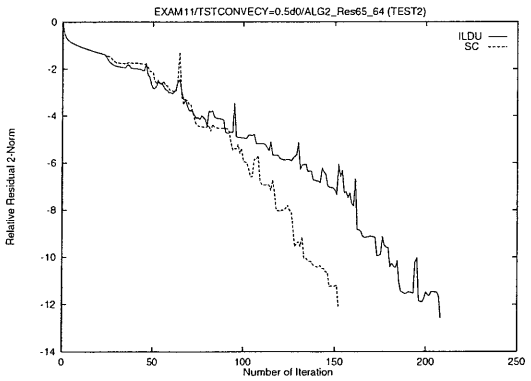

Figure 45: $65\times64 : v_x = 10.0, v_y = 0.5$

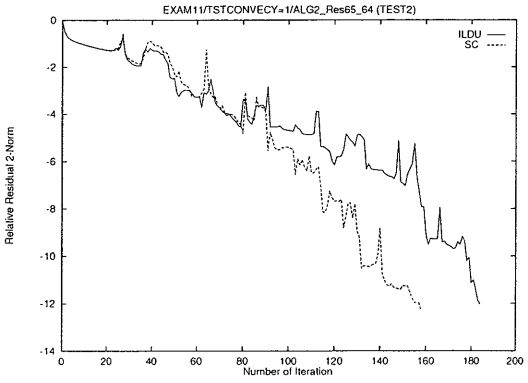

Figure 46: $65\times64 : v_x = 10.0, v_y = 1.0$

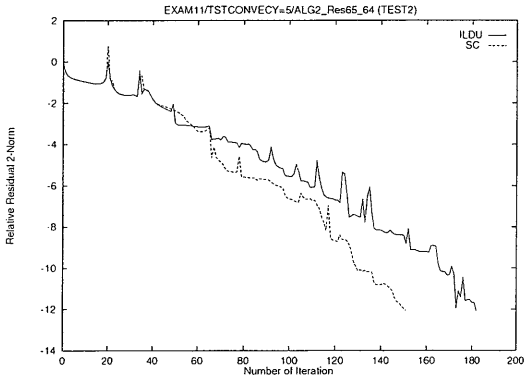

Figure 47: $65\times64 : v_x = 10.0, v_y = 5.0$

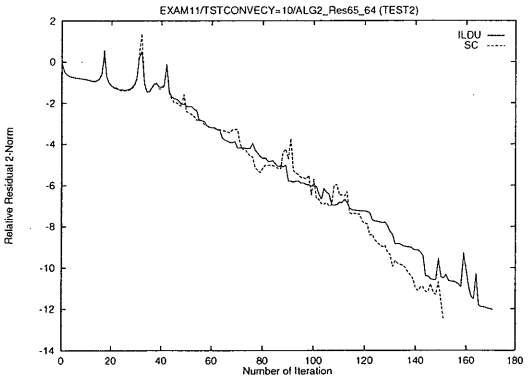

Figure 48: $65\times64 : v_x = 10.0, v_y = 10.0$