RECSY — A High Performance Library for Sylvester-Type Matrix Equations

Isak Jonsson and Bo Kågström

Department of Computing Science and HPC2N, Umeå University,
SE-901 87 Umeå, Sweden.
{isak,bokg}@cs.umu.se

Abstract. RECSY is a library for solving triangular Sylvester-type matrix equations. Its objectives are both speed and reliability. In order to achieve these goals, RECSY is based on novel recursive blocked algorithms, which call high-performance kernels for solving small-sized leaf problems of the recursion tree. In contrast to explicit standard blocking techniques, our recursive approach leads to an automatic variable blocking that has the potential of matching the memory hierarchies of today's HPC systems. The RECSY library comprises a set of Fortran 90 routines, which uses recursion and OpenMP for shared memory parallelism to solve eight different matrix equations, including continuous-time as well as discrete-time standard and generalized Sylvester and Lyapunov equations. Uniprocessor and SMP parallel performance results of our recursive blocked algorithms and corresponding routines in state-of-the-art libraries LAPACK and SLICOT are presented. The performance improvements of our recursive algorithms are remarkable, including 10-fold speedups compared to standard algorithms.

Keywords: Sylvester-type matrix equations, recursion, automatic blocking, superscalar, GEMM-based, level 3 BLAS, LAPACK, SLICOT, RECSY

1 Introduction

In [8, 9], we describe recursive blocked algorithms for solving different Sylvester-type matrix equations. We differentiate between one-sided and two-sided matrix equations. The notation one-sided matrix equations is used when the solution is only involved in matrix products of two matrices, e.g., $\mathrm{op}(A)X$ or $X\mathrm{op}(A)$, where $\mathrm{op}(A)$ can be A or A^T. In two-sided matrix equations, the solution is involved in matrix products of three matrices, both to the left and to the right, e.g., $\mathrm{op}(A)X\mathrm{op}(B)$. Table 1 lists eight different types of matrix equations considered together with the acronyms used (one-sided equations in the top and two-sided equations in the bottom part).

1.1 Triangular Matrix Equations

The classical method of solution of the Sylvester-type matrix equations is based on the Bartels–Stewart method [2], which includes three major steps. First,

Name	Matrix equation	Acronym
Standard Sylvester (CT)	$AX - XB = C$	SYCT
Standard Lyapunov (CT)	$AX + XA^T = C$	LYCT
Generalized Coupled Sylvester	$(AX - YB, DX - YE) = (C, F)$	GCSY
Standard Sylvester (DT)	$AXB^T - X = C$	SYDT
Standard Lyapunov (DT)	$AXA^T - X = C$	LYDT
Generalized Sylvester	$AXB^T - CXD^T = E$	GSYL
Generalized Lyapunov (CT)	$AXE^T + EXA^T = C$	GLYCT
Generalized Lyapunov (DT)	$AXA^T - EXE^T = C$	GLYDT

Table 1. One-sided (top) and two-sided (bottom) matrix equations.

the matrix (or matrix pair) is transformed to a Schur (or generalized Schur) form. This leads to a reduced triangular matrix equation. For example, the coefficient matrices A and B in the Sylvester equation $AX - XB = C$ are in upper triangular or upper quasi-triangular form. Finally, the solution of the reduced matrix equation is transformed back to the originally coordinate system.

Reliable and efficient algorithms for the reduction step can be found in LAPACK [1] for the standard case, and in [3] for the generalized case, where a blocked variant of the QZ method is presented.

Triangular matrix equations also appear naturally in estimating the condition numbers of matrix equations and different eigenspace computations, including block-diagonalization of matrices and matrix pairs and computation of functions of matrices. Related applications include the direct reordering of eigenvalues in the real (generalized) Schur form and the computation of additive decompositions of a (generalized) transfer function (see [8, 9] for more information and references).

1.2 Motivation

Our goal is to produce a state-of-the-art library which solves the triangular Sylvester-type matrix equations listed in Table 1. The library should be easy to use, provide excellent performance, and it should be expandable and tunable for new routines and different platforms.

In this contribution, we present a new library, RECSY, which contains sequential and parallel implementations of all triangular matrix equations listed in Table 1. Also, the routines feature several different transpose and sign options, so all in all the RECSY routines are able to solve 42 different variants.

Before we go into details of the library, we outline the rest of the paper. In Section 2, we describe the routines for solving one-sided triangular equations. We do this by reviewing our recursive approach, and how it gives better performance. Furthermore, we describe the fast and highly optimized kernels used to solve small problems. We also show how we easily provide parallel routines for SMP/OpenMP-aware systems. The routines for solving two-sided triangular equations are listed in Section 3. The differences between the one-sided and the

```
Algorithm 1: SYCT function [X] = recsyct(A, B, C, uplo, blks)
if 1 ≤ M, N ≤ 4 then
    X = trsyct(A, B, C, uplo);
else
    if 1 ≤ N ≤ M/2   % Case 1: Split A (by rows and colums), C (by rows only)
        X_2 = recsyct(A_22, B, C_2, 1, blks);
        C_1 = gemm(−A_12, X_2, C_1);
        X_1 = recsyct(A_11, B, C_1, 1, blks);
        X = [X_1; X_2];
    elseif 1 ≤ M ≤ N/2   % Case 2: Split B (by rows and colums), C (by columns only)
        X_1 = recsyct(A, B_11, C_1, 1, blks);
        C_2 = gemm(X_1, B_12, C_2);
        X_2 = recsyct(A, B_22, C_2, 1, blks);
        X = [X_1, X_2];
    else   % M, N ≥ blks, Case 3: Split A, B and C (all by rows and colums)
        X_21 = recsyct(A_22, B_11, C_21, 1, blks);
        C_22 = gemm(X_21, B_12, C_22);  C_11 = gemm(−A_12, X_21, C_11);
        X_22 = recsyct(A_22, B_22, C_22, 1, blks);  X_11 = recsyct(A_11, B_11, C_11, 1, blks);
        C_12 = gemm(−A_12, X_22, C_12);
        C_12 = gemm(X_11, B_12, C_12);
        X_12 = recsyct(A_11, B_22, C_12, 1, blks);
        X = [X_11, X_12;  X_21, X_22];
    end
end
```

Algorithm 1: Recursive blocked algorithm for solving the triangular continuous-time Sylvester equation (SYCT).

two-sided equations are explained, and we show how the library takes care of these issues. In Section 4, we give a short guide on how to use the library. Some performance results are given in Section 5.

2 One-Sided Triangular Matrix Equations

In [6], we present pseudo-code algorithms for three one-sided triangular matrix equations. These are the continuous-time standard Sylvester (SYCT), the standard Lyapunov (LYCT), and the generalized coupled Sylvester (GCSY) equations, together with the mathematical derivation of the algorithms. For reference, we show a compressed version of one of the algorithms, SYCT, in Algorithm 1. The equation is solved by recursion, where the problem is split into two or four new, smaller, problems of approximately the same size. The aim is to split the matrix exactly as close to the middle as possible. If the splitting point appears at a 2×2 diagonal block, the matrices are split just below the splitting point.

2.1 Performance due to recursive level 3 blocking and updates

The recursive approach provides automatic cache blocking through great temporal locality. Furthermore, it reveals level 3 BLAS updates, shown as **gemm** calls in Algorithm 1. The same observations hold for the LYCT and GCSY equations.

For LYCT, the updates are done by symmetric rank-k and symmetric rank-$2k$ operations instead, due to its symmetry characteristics. However, most of the work in LYCT is done in SYCT, as LYCT uses SYCT to solve off-diagonal blocks in the solution X.

It has been observed that for some platforms, the vendor's DGEMM does not perform very well for small matrices. This is due to overhead from parameter checking and temporary buffer setup. Therefore, the library supplies its own matrix-matrix multiply which uses 4×4 outer loop unrolling, but without blocking for memory cache. It shows good performance on different types of processors. This routine is used instead of the regular DGEMM when the problem fits into level 1 cache. Note that recursion itself provides good temporal locality.

2.2 Performance due to fast kernels

At the bottom of the recursion tree for the Sylvester equations (SYCT, GCSY), where the problem size is very small ($\leq 4\times4$), the recursion is terminated and the problem is instead solved by a very fast kernel. Now, the small matrix equations are represented as a linear system of equations $Zx = y$, where Z is a Kronecker product representation of the Sylvester-type operator, and x and the right hand side y are vectors representing the solution and the the right hands side(s). For example, $Z_{\mathrm{SYCT}} = I_n \otimes A - B^T \otimes I_m$ (see also [8, 9]). This kernel features unrolling for the updates, and uses partial pivoting for solving small matrix equations. These equations lead to Kronecker product matrix representations Z of size 1×1 – 4×4 for SYCT and 2×2 – 8×8 for GCSY. If a good pivot candidate cannot be found (problem is nearly singular), or if the solution procedure is close to produce an overflow, the fast kernel is aborted. On abortion, the recursive procedure backtracks and we construct a new Kronecker product representation, now leading to a matrix Z of size 16×16 for SYCT and 32×32 for GCSY. The system is then solved using complete pivoting with perturbed pivot elements if necessary and right hand scaling to avoid overflow.

For LYCT, the recursion continues until subproblems of size 1×1 or 2×2. Such a problem, with Kronecker product matrix Z of size 1×1 or 3×3, is solved using the fast kernel solver (partial pivoting), or with complete pivoting for very ill–conditioned cases. The reason for not optimizing LYCT as much as SYCT and GCSY is that most of the solving takes part in SYCT. The number of small problems solved in LYCT by the Lyapunov small problem solver is $O(n)$, whereas the number of problems solved in LYCT by the SYCT solver is $O(n^2)$.

2.3 Performance due to parallelism

First of all, it is possible to get a fair amount of speedup on systems using only SMP versions of level 3 BLAS routines. This is due to fact that the recursive algorithms call the level 3 BLAS routines with large, squarish blocks. The routines can thus do each update in parallel with good efficiency.

As can be observed in Case 3 of Algorithm 1, there are two recursive calls that can be executed simultaneously. This is used in our implementation of SYCT and

GCSY to create OpenMP versions of our algorithms for SMP systems, where not only the updates, but also the solves are done in parallel. The OpenMP versions take an additional argument, PROC, the number of processors available. Whenever PROC > 1, and the problem is a Case 3 problem, the second and third recursive calls (and corresponding updates) are done in parallel, giving a diamond-shaped execution graph. For more details, see [6]. For LYCT, there is no explicit parallelism in the Lyapunov solver. However, as most of the work takes place in the Sylvester solver SYCT, the LYCT routine gains performance from the OpenMP parallelism as well.

In order for this scheme to work well, two conditions must hold. First, there should not be any degradation in the total performance when the SMP BLAS 3 routines are called simultaneously (multiple parallelism). This has been a problem on a small number of machines, where OpenMP and SMP BLAS did not work together. Second, if there are more than two processors in the system, the OpenMP compiler must support nested parallelism in order to fully utilize the system, which most modern OpenMP compilers do.

The overhead of SMP BLAS routines can be pretty large, especially when solving too small problems. Fortunately, the library's own matrix-matrix multiply routine alleviates this problem. As it is not parallelized, it is perfectly suited for small problems, where the parallel divide-and-conquer splitting already has taken place at higher levels of the recursion tree.

2.4 Fortran interfaces

The routines are implemented in Fortran 90, which greatly simplifies the recursive calls. Also, this makes the routines' signatures easier, as some arguments can be declared optional. Below, the six routines for solving one-sided triangular matrix equations are listed.

- `RECSYCT (UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, INFO, MACHINE)`
- `RECSYCT_P (PROCS, UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, INFO, MACHINE)`
  Solves the triangular continuous-time Sylvester equation. A and B are upper quasitriangular of size $M \times M$ and $N \times N$, C is rectangular $M \times N$.

`UPLOSIGN`	SYCT equation
0	$AX + XB = scale\,C, C \leftarrow X$
1	$AX - XB = scale\,C, C \leftarrow X$
2	$AX + XB^T = scale\,C, C \leftarrow X$
3	$AX - XB^T = scale\,C, C \leftarrow X$
4	$A^TX + XB = scale\,C, C \leftarrow X$
5	$A^TX - XB = scale\,C, C \leftarrow X$
6	$A^TX + XB^T = scale\,C, C \leftarrow X$
7	$A^TX - XB^T = scale\,C, C \leftarrow X$

- `RECLYCT (UPLO, SCALE, M, A, LDA, C, LDC, INFO, MACHINE)`
- `RECLYCT_P (PROCS, UPLO, SCALE, M, A, LDA, C, LDC, INFO, MACHINE)`
  Solves the symmetric triangular continuous-time Lyapunov equation. A is upper quasitriangular $M \times M$, C is symmetric $M \times M$, only upper part of C is accessed.

UPLOSIGN	LYCT equation
0	$AX + XA^T = scale\,C, C \leftarrow X$
1	$A^TX + XA = scale\,C, C \leftarrow X$

- `RECGCSY (UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE, F, LDF, INFO, MACHINE)`
- `RECGCSY_P (PROCS, UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE, F, LDF, INFO, MACHINE)`
  Solves the generalized coupled triangular Sylvester equation. A is upper quasitriangular $M \times M$, D is upper triangular $M \times M$, B is upper quasitriangular $N \times N$, E is upper triangular $N \times N$, C and F are rectangular $M \times N$.

UPLOSIGN	GCSY equation
0	$(AX + YB = scale\,C, DX + YE = scale\,F), (C, F) \leftarrow (X, Y)$
1	$(AX - YB = scale\,C, DX - YE = scale\,F), (C, F) \leftarrow (X, Y)$
2	$(AX + YB^T = scale\,C, DX + YE^T = scale\,F), (C, F) \leftarrow (X, Y)$
3	$(AX - YB^T = scale\,C, DX - YE^T = scale\,F), (C, F) \leftarrow (X, Y)$
4	$(A^TX + YB = scale\,C, D^TX + YE = scale\,F), (C, F) \leftarrow (X, Y)$
5	$(A^TX - YB = scale\,C, D^TX - YE = scale\,F), (C, F) \leftarrow (X, Y)$
6	$(A^TX + YB^T = scale\,C, D^TX + YE^T = scale\,F), (C, F) \leftarrow (X, Y)$
7	$(A^TX - YB^T = scale\,C, D^TX - YE^T = scale\,F), (C, F) \leftarrow (X, Y)$

3 Two-Sided Triangular Matrix Equations

The recursive templates for solving two-sided triangular matrix equations are similar to the templates for solving one-sided triangular matrix equations. However, the two-sidedness of the matrix updates leads to several different implementation choices. In the RECSY library, two internal routines are used: RECSY_MULT_LEFT and RECSY_MULT_RIGHT. These routines compute $C = \beta C + \alpha \text{op}(A)X$ and $C = \beta C + \alpha X \text{op}(A)$, respectively, where $\text{op}(A)$ is A or A^T, and A is rectangular (DGEMM), upper triangular, upper quasitriangular, upper trapezoidal, or upper quasitrapezoidal (i.e., trapezoidal with 1×1 and 2×2 diagonal blocks). C and X are rectangular matrices. The actual computation is carried out by calls to DGEMM, DTRMM, the internal small matrix-matrix multiply, and level 1 BLAS routines for subdiagonal elements. Note that these routines require extra workspace when DTRMM is used, as DTRMM only computes the product ($C = \text{op}(A)X$) and not the multiply-and-add ($C = C + \text{op}(A)X$).

Combining RECSY_MULT_LEFT and RECSY_MULT_RIGHT give the routine RECSY_AXB. It computes the update $C = \beta C + \alpha \text{op}(A) X \text{op}(B)$, where A and B can independently be any of the five matrix structures listed above. There are two ways of computing the update, either by doing the left multiplication first or by doing the right multiplication first. RECSY_AXB chooses the ordering which uses the least amount of floating point operations. RECSY_AXB also requires workspace to store the intermediate product.

However, for symmetric two-sided updates $C = \beta C + \alpha \text{op}(A) X \text{op}(A^T)$, where C and X are symmetric, the SLICOT [12] routine MB01RD is used. MB01RD uses a series of level 3 updates to calculate C without requiring workspace and with fewer floating point operations than RECSY_AXB.

3.1 Fortran interfaces

As shown above, the two-sided matrix equation solvers require extra workspace. As Fortran 90 allows dynamic memory allocation, this facilitates the subroutine calls. The user now has the option of either supplying workspace to the routine, or, preferably, leave this task to the subroutine itself. Below, the ten routines for solving two-sided triangular matrix equations are shown.

- `RECSYDT (UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, INFO, MACHINE, WORKSPACE, WKSIZE)`
- `RECSYDT_P (PROCS, UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, INFO, MACHINE, WORKSPACE, WKSIZE)`
  Solves the triangular discrete-time Sylvester equation. A is upper quasitriangular $M \times M$, B is upper quasitriangular $N \times N$, C is rectangular $M \times N$.

UPLOSIGN	SYDT equation
0	$AXB + X = scale\, C, C \leftarrow X$
1	$AXB - X = scale\, C, C \leftarrow X$
2	$AXB^T + X = scale\, C, C \leftarrow X$
3	$AXB^T - X = scale\, C, C \leftarrow X$
4	$A^T XB + X = scale\, C, C \leftarrow X$
5	$A^T XB - X = scale\, C, C \leftarrow X$
6	$A^T XB^T + X = scale\, C, C \leftarrow X$
7	$A^T XB^T - X = scale\, C, C \leftarrow X$

- `RECLYDT (UPLO, SCALE, M, A, LDA, C, LDC, INFO, MACHINE, WORKSPACE, WKSIZE)`
- `RECLYDT_P (PROCS, UPLO, SCALE, M, A, LDA, C, LDC, INFO, MACHINE, WORKSPACE, WKSIZE)`
  Solves the symmetric triangular discrete-time Lyapunov equation. A is upper quasitriangular $M \times M$, C is symmetric $M \times M$, only upper part of C is accessed.

UPLOSIGN	LYDT equation
0	$AXA^T - X = scale\, C, C \leftarrow X$
1	$A^T XA - X = scale\, C, C \leftarrow X$

- `RECGSYL (UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE, INFO, MACHINE,WORKSPACE, WKSIZE)`
- `RECGSYL_P (UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE, INFO, MACHINE, WORKSPACE, WKSIZE)`
  Solves the triangular generalized Sylvester equation. A is upper quasitriangular $M \times M$, C is upper triangular $M \times M$, B is is upper quasitriangular $N \times N$, D is upper triangular $N \times N$, C and F are rectangular $M \times N$.

UPLOSIGN	GSYL equation
0	$AXB + CXD = scale\, E, E \leftarrow X$
1	$AXB - CXD = scale\, E, E \leftarrow X$
2	$AXB^T + CXD^T = scale\, E, E \leftarrow X$
3	$AXB^T - CXD^T = scale\, E, E \leftarrow X$
4	$A^T XB + C^T XD = scale\, E, E \leftarrow X$
5	$A^T XB - C^T XD = scale\, E, E \leftarrow X$
6	$A^T XB^T + C^T XD^T = scale\, E, E \leftarrow X$
7	$A^T XB^T - C^T XD^T = scale\, E, E \leftarrow X$
10	$AXB^T + CXD^T = scale\, E, E \leftarrow X, B$ is upper triangular $N \times N$, D is upper quasitriangular $N \times N$ (used by RECGLYCT)
12	$A^T XB + C^T XD = scale\, E, E \leftarrow X, B$ is upper triangular $N \times N$, D is upper quasitriangular $N \times N$ (used by RECGLYCT)

- RECGLYDT (UPLO, SCALE, M, A, LDA, E, LDE, C, LDC, INFO, MACHINE, WORKSPACE, WKSIZE)
- RECGLYDT_P (PROCS, UPLO, SCALE, M, A, LDA, E, LDE, C, LDC, INFO, MACHINE, WORKSPACE, WKSIZE)
- RECGLYCT (UPLO, SCALE, M, A, LDA, E, LDE, C, LDC, INFO, MACHINE, WORKSPACE, WKSIZE)
- RECGLYCT_P (PROCS, UPLO, SCALE, M, A, LDA, E, LDE, C, LDC, INFO, MACHINE, WORKSPACE, WKSIZE)

  Solves the triangular generalized discrete-time and continuous-time Lyapunov equation, respectively. A is upper quasitriangular $M \times M$, E is upper triangular $M \times M$, C is symmetric $M \times M$, only upper part of C is accessed.

UPLOSIGN	GLYDT equation	GLYCT equation
0	$AXA^T - EXE^T = scale\, C, C \leftarrow X$	$AXE^T + EXA^T = scale\, C, C \leftarrow X$
1	$A^TXA - E^TXE = scale\, C, C \leftarrow X$	$A^TXE - E^TXA = scale\, C, C \leftarrow X$

4 Library Usage

The library, together with building instructions, can be downloaded from `http://www.cs.umu.se/~isak/recsy`. The library is easily built with any Fortran 90 compiler, both UNIX and Windows compilers are supported. By using the routines listed in Sections 2 and 3, the user gets access to all of the different options and tuning parameters of the library. The types of the parameters to the routines are listed in Table 2. The parameter MACHINE, which can be omitted – as the authors recommend, sets different tuning parameters. For details, see the library homepage.

4.1 SLICOT/LAPACK wrapper routines

Instead of using the native routines listed, the library also provides wrapper routines which overload SLICOT [12] and LAPACK [1] routines, see Table 3. By linking with the library, calls to SLICOT and LAPACK Sylvester-type matrix

Parameter	Type	Meaning
UPLOSIGN, UPLO	INTEGER	Defines different options (see above).
A, B, C, D, E, F	DOUBLE PRECISION(*)	Coefficient/right hand side matrices (see above).
SCALE	DOUBLE PRECISION	Scaling factor (output argument), to avoid overflow in the computed solution.
LDA, LDB, LDC, LDD, LDE, LDF	INTEGER	Leading dimension of matrices A to F.
INFO	INTEGER	Output parameter. INFO$= -100$: Out of memory, INFO$= 0$: No error, INFO> 0: The equation is (nearly) singular to working precision; perturbed values were used to solve the equation.
MACHINE	DOUBLE PRECISION(0:9)	Optional. Specify only if you want different parameters than the default parameters, which are set by RECSY_MACHINE.
WORKSPACE	DOUBLE PRECISION(*)	Optional. Specify only if you provide your own workspace.
WKSIZE	INTEGER	Optional. Specify only if you provide your own workspace.

Table 2. The parameters to the RECSY library routines, their type and meaning.

Routine	Native routine
SLICOT SB03MX	RECLYDT
SLICOT SB03MY	RECLYCT
SLICOT SB04PY	RECSYDT
SLICOT SG03AX	RECGLYDT
SLICOT SG03AY	RECGLYCT
LAPACK DTRSYL	RECSYCT
LAPACK DTGSYL [10]	RECGCSY

Table 3. SLICOT and LAPACK routines and their RECSY equivalents.

equations solvers will be replaced to a call to the optimized RECSY equivalent. We remark that neither LAPACK nor SLICOT provide a GSYL solver.

Note that not only the SLICOT and LAPACK routines listed above are affected. Any routine which uses any of these routines will also be using the RECSY library. In particular, this is true for the SLICOT routines which solve unreduced problems, e.g., SB04MD (SYCT solver) or SM03MD (LYDT/LYCT solver). SB04MD and SB03MD call DTRSYL and SB03MX/SB03MY, respectively, for solving the reduced equation. With the RECSY library, performance for SB04MD and SB03MD are also improved [7, 9].

5 Performance Results

We list a few results obtained with the RECSY library. In Table 4 a), results obtained for the continuous-time Lyapunov equation (LYCT) are shown. Here, the speedup is remarkable. The RECSY library is up to 83 times faster than the original SLICOT library. This is both due to faster kernels and multi-level blocking from recursion. The same results can be observed for LAPACK routines. For example, the RECSY routine RECSYCT is more than 20 times faster then the LAPACK routine DTRSYL for $M = N > 500$ on the IBM PowerPC 604e.

Timings for two-sided matrix equation examples and given in Table 4 b). For the largest example, the SLICOT library requires more than 45 minutes to solve the problem. The RECSY library solves the same problem in less than a half minute. The extra speedup from the OpenMP version of the library is also given. For further results, we refer to [8, 9] and the library homepage.

Acknowledgements

Financial support for this project has been provided by the *Swedish Research Council* under grants TFR 98-604, VR 621-2001-3284 and the *Swedish Foundation for Strategic Research* under grant A3 02:128.

References

1. E. Anderson, Z. Bai, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenny, S. Ostrouchov, and D. Sorensen *LAPACK Users Guide, Third Edition.* SIAM Publications, 1999.

	a) $AX + XA^T = C$						b) $AXB - X = C$							
	IBM Power3			MIPS R10000			IBM Power3				Intel Pentium III			
	Mflops/s		Speedup	Mflops/s		Speedup	Time (sec)		Speedup		Time (sec)		Speedup	
$(M =)N$	A	B	B/A	A	B	B/A	C	D	D/C	E/D	C	D	D/C	E/D
100	77.0	166.5	2.16	82.0	123.5	1.51	1.73e-2	7.41e-3	2.33	1.16	2.20e-2	2.20e-2	1.00	1.22
250	85.3	344.5	4.04	88.7	224.5	2.53	6.20e-1	6.93e-2	8.95	0.98	6.83e-1	2.31e-1	2.96	1.33
500	10.6	465.0	43.85	42.2	277.8	6.58	2.32e+1	4.60e-1	50.50	1.48	7.82e+0	1.56e+0	5.00	1.35
1000	7.7	554.7	72.20	14.5	254.0	17.57	2.44e+2	3.26e+0	74.65	1.94	7.04e+1	1.10e+1	6.40	1.48
1500	7.0	580.5	83.19	9.7	251.0	25.81	9.37e+2	1.08e+1	86.66	2.04	2.78e+2	3.58e+1	7.76	1.53
2000							2.84e+3	2.41e+1	117.71	2.14	9.07e+2	8.03e+1	11.29	1.57
	A – SLICOT SB03MY B – RECLYCT						C – SLICOT SB04PY D – RECSYDT E – RECSYDT_P							

Table 4. a) Performance results for the triangular Lyapunov equation—IBM Power3, 200 MHz (left) and SGI Onyx2 MIPS R10000, 195 MHz (right). b) Performance results for the triangular discrete-time Sylvester equation—IBM Power3, 4×375 MHz (left) and Intel Pentium III, 2×550 MHz (right).

2. R.H. Bartels and G.W. Stewart. Algorithm 432: Solution of the Equation $AX + XB = C$, *Comm. ACM*, 15(9):820–826, 1972.
3. K. Dackland and B. Kågström. Blocked Algorithms and Software for Reduction of a Regular Matrix Pair to Generalized Schur Form. *ACM Trans. Math. Software*, 24(4):425–454, December 1999.
4. J.D. Gardiner, A.J. Laub, J.J. Amato, and C.B. Moler. Solution of the Sylvester Matrix Equation $AXB^T + CXD^T = E$, *ACM Trans. Math. Software*, 18(2):223–231, June 1992.
5. J.D. Gardiner, M.R. Wette, A.J. Laub, J.J. Amato, and C.B. Moler. A Fortran 77 Software Package for Solving the Sylvester Matrix Equation $AXB^T + CXD^T = E$, *ACM Trans. Math. Software*, 18(2):232–238, June 1992.
6. I. Jonsson and B. Kågström. Recursive Blocked Algorithms for Solving Triangular Matrix Equations—Part I: One-Sided and Coupled Sylvester-Type Equations, *SLICOT Working Note* 2001-4.
7. I. Jonsson and B. Kågström. Recursive Blocked Algorithms for Solving Triangular Matrix Equations—Part II: Two-Sided and Generalized Sylvester and Lyapunov Equations, *SLICOT Working Note* 2001-5.
8. I. Jonsson, and B. Kågström. Recursive blocked algorithms for solving triangular systems — Part I: One-sided and coupled Sylvester-type matrix equations. *ACM Trans. Math. Softw.*, 28(4):392–415, December 2002.
9. I. Jonsson, and B. Kågström. Recursive blocked algorithms for solving triangular systems — Part II: Two-sided and generalized Sylvester and Lyapunov matrix equations *ACM Trans. Math. Softw.*, 28(4):416–435, December 2002.
10. B. Kågström and P. Poromaa. LAPACK–Style Algorithms and Software for Solving the Generalized Sylvester Equation and Estimating the Separation between Regular Matrix Pairs. *ACM Trans. Math. Software*, 22(1):78–103, March 1996.
11. B. Kågström and L. Westin. Generalized Schur methods with condition estimators for solving the generalized Sylvester equation. *IEEE Trans. Autom. Contr.*, 34(7):745–751, July 1989.
12. SLICOT library and the Numerics in Control Network (NICONET) website: www.win.tue.nl/niconet/index.html