SCASY Users' Guide*

Release 1.0

Robert Granat and Bo Kågström[1]

March 20, 2010.

Abstract

The functionality, the contents, some implementation issues, and the proper usage of the latest release of the SCASY software library are presented.

Keywords: SCASY, parallel software, ScaLAPACK, Sylvester-type matrix equations

1 Introduction

The SCASY library is a high performance software library for solving Sylvester-type matrix equations on distributed memory platforms. The focus of SCASY is on robustness, speed and scalability (see also the ACM TOMS Algorithms Policy [1]). This SCASY Users' Guide is a supplement to the ACM Transactions on Mathematical Software articles [17, 16].

2 Algorithms for Sylvester-type matrix equations

In this section, we give a brief overview of the algorithms underlying the implementations available through SCASY.

The algorithms in SCASY for Sylvester-type matrix equations are presented in [17, 16] and in the technical reports [14, 15]. Table 1 summarizes the considered Sylvester-type matrix equations.

The algorithms are based on the classical Bartels–Stewart's method [6] which utilizes standard and generalized Schur decompositions (see, e.g., [13]) to reduce the matrix equations into a triangular Schur form. These reduced matrix equations are solved using combined forward- and backward substitution techniques. For illustration, consider SYCT:

- Transform A and B to real Schur form (see, e.g., [13]), i.e., compute the factorizations $T_A = Q^T A Q$ and $T_B = P^T B P$, where $Q \in \mathbb{R}^{m \times m}$ and $P \in \mathbb{R}^{n \times n}$ are orthogonal and T_A and T_B are upper quasi-triangular, i.e., having 1×1 and 2×2 diagonal blocks corresponding to real and complex conjugate pairs of eigenvalues, respectively.

[1]Department of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden. E-mail: {granat,bokg}@cs.umu.se

*__Technical Report UMINF-09.10.__ This research was conducted using the resources of the High Performance Computing Center North (HPC2N). Financial support has been provided by the *Swedish Research Council* under grant VR 70625701 and by the *Swedish Foundation for Strategic Research* under grant A3 02:128.

Name	Matrix Equation	Acronym
Standard CT Sylvester	$\mathrm{op}(A)X \pm X\mathrm{op}(B) = sC \in \mathbb{R}^{m\times n}$	SYCT
Standard DT Lyapunov	$\mathrm{op}(A)X + X\mathrm{op}(A^T) = sC \in \mathbb{R}^{m\times m}$	LYCT
Standard DT Sylvester	$\mathrm{op}(A)X\mathrm{op}(B) \pm X = sC \in \mathbb{R}^{m\times n}$	SYDT
Standard DT Lyapunov	$\mathrm{op}(A)X\mathrm{op}(A^T) - X = sC \in \mathbb{R}^{m\times m}$	LYDT
Generalized Coupled Sylvester	$\begin{cases} \mathrm{op}(A)X \pm Y\mathrm{op}(B) = sC \in \mathbb{R}^{m\times n} \\ \mathrm{op}(D)X \pm Y\mathrm{op}(E) = sF \in \mathbb{R}^{m\times n} \end{cases}$	GCSY
Generalized Sylvester	$\mathrm{op}(A)X\mathrm{op}(B) \pm \mathrm{op}(C)X\mathrm{op}(D) = sE \in \mathbb{R}^{m\times n}$	GSYL
Generalized CT Lyapunov	$\mathrm{op}(A)X\mathrm{op}(E^T) + \mathrm{op}(E)X\mathrm{op}(A^T) = sC \in \mathbb{R}^{m\times m}$	GLYCT
Generalized DT Lyapunov	$\mathrm{op}(A)X\mathrm{op}(A^T) - \mathrm{op}(E)X\mathrm{op}(E^T) = sC \in \mathbb{R}^{m\times m}$	GLYDT

Table 1: Sign and transpose variants of the Sylvester-type matrix equations considered in SCASY. CT and DT denote the continuous-time and discrete-time variants, respectively. The scalar s, where $0 < s \leq 1.0$, is a scaling factor used to prevent from overflow in the solution process.

- Update C with respect to the Schur decompositions by the two-sided transformation $\tilde{C} = Q^T C P$.
- Solve the resulting reduced triangular matrix equation for the intermediate solution matrix $\tilde{X}$, see below.
- Transform the intermediate solution back to the original coordinate system by a second two-sided transformation $X = Q\tilde{X}P^T$.

The triangular matrix equation is solved by several layers of explicit and/or recursive blocking, leading to algorithms which are very rich in GEMM-updates in the right hand side(s); on the leaf nodes of the blocking, we consider small-sized matrix equations in an equivalent linear systems of equations $Zx = y$, where $Z = I_n \otimes A - B^T \otimes I_m$ is a Kronecker product representation of the corresponding Sylvester-type operator ($AX - XB$ for SYCT). This reformulation is also the key observation which enables condition estimation using the technique developed in [21, 23, 28].

3 Software

In this section, we present the parallel software that implements the SCASY algorithms.

3.1 Implementation notes

The algorithms in SCASY are implemented using Fortran 90/95 following the coding conventions that are typical for LAPACK [5] and ScaLAPACK [7]. The underlying parallel computing paradigm follows the ScaLAPACK standard:

- The parallel processes are organized into a rectangular $P_r \times P_c$ mesh labelled from $(0, 0)$ to $(P_r - 1, P_c - 1)$ according to their specific position indices in the mesh.
- The matrices are distributed over the mesh using 2-dimensional (2D) block cyclic mapping with the block sizes m_b and n_b in the row and column dimensions, respectively, i.e. each distributed block is of size $m_b \times n_b$.

The codes in the SCASY package have been tested extensively with the Portland Group Compiler suite and the Pathscale Compiler suite, and also with the GNU Fortran compiler and the NAG Fortran Compiler.

3.2 Download location

The stable version of SCASY published for ACM TOMS is available on CALGO [10]. Other SCASY releases (including bug-fixes and new features) are available via the library homepage [33] in either `.zip` or `.tar.gz` format with the name `scasy_latest`. The latest version can also be retrieved via correspondence with the authors.

3.3 Library installation

In what follows, we assume that the user works in a Linux-like environment and has downloaded the latest SCASY release in the `.tar.gz` format. (The software package has the same structure in the .zip format, but the accompanying Makefiles are most likely irrelevant in a Windows-like environment.)

Note that there is currently no installation routine; after the library has been built, it should be moved to a proper location (e.g., `/usr/local/lib`).

3.3.1 A look inside the SCASY tar-ball

To unpack the tar-ball, move to a proper working directory and try

```
host:~> tar zxvf scasy_latest.tar.gz
```

to unpack the library. This will reveal the SCASY root directory, which is called `scasy[x.y]`, where `[x.y]` denotes the current release and version numbers[1]. Inside this root directory, the `ls` command will reveal the following files and directories:

```
docs/ lib/ make/ Makefile Make_include README test/.
```

In what follows, we give an overview of the interior of each item above:

`docs/` This directory contains the documentation necessary to understand how to install and use the SCASY library, e.g., the latest version of this Users' Guide [18] as well some material related to the SCASY homepage [33].

`lib/` This directory contains all the Fortran source code as `.f` files (which implies that explicit pre-processing is necessary, see Section 3.3.2), and a few C (`.c`) files which are compiled and attached to the library; these files are used mostly for debugging purposes and most users can ignore them, as long as a proper C compiler is chosen. The directory also includes a local `Makefile` which compiles and builds the library via a call from the root directory `Makefile` (which is described below). The library will also be built in this directory as the archive `libscasy.a`.

`make/` For the user's convenience, this directory contains template `Make_include` files (see below) which are specific to each of the following compilers: Portland Group, Pathscale, GNU Fortran and NAG Fortran Compiler.

[1] Current release is 1.0.

`Makefile` This `Makefile` is responsible for building the library and the attached test program (see below) from the root directory, when called by the user.

`Make_include` In principle, *this file is the only one the user needs to modify* to build the library and the attached test program correctly. By using the available templates, see above, the modification can be carried out easily. Necessary modifications include choice of Fortran (and C) compilers, the associated Fortran compiler flags (the C compiler only uses `-c`), pre-processing options (see below), linking options, paths to external libraries, etc.

`README` This file contains some information quite similar to this Users' Guide, but is much shorter and serves more like a Quick Start Guide to the library.

`test/` This directory contains the Fortran file `TESTSCASY.f` and another local `Makefile`, which is responsible for compiling and linking the test program with the library. We give more information on the usage of the test program in Section 3.3.3.

3.3.2 Pre-processing options

The following pre-processing options are available for the SCASY library:

`USE_DYNAMIC` When this option is activated, the test program uses dynamic allocation (which is recommended). If this option is not activated, all allocations are done statically in the test program.

`USE_INTEGER8` When this option is activated, the test program uses 8 byte integers to specify the size of the memory areas to be allocated (recommended, especially in cases when large matrices are considered).

`LOOPGRID` When this option is activated, the test program loops through the possible process grid dimensions specified by its local variables `[ACRO]_NPROW_MIN`, `[ACRO]_NPROW_MAX`, `[ACRO]_NPCOL_MIN`, `[ACRO]_NPCOL_MAX` using the step variables `[ACRO]_NPROW_STEP` and `[ACRO]_NPCOL_STEP`. The user is responsible for allocating enough MPI processes when executing the test program such that

```
# MPI processes >= [ACRO]_NPROW_MAX*[ACRO]_NPCOL_MAX
```

otherwise the test program will abort with an error message. If this option is not activated, the test program ignores the specified process grid dimensions listed above and simply creates an "as-square-as-possible" rectangular process grid such that `NPROW*NPCOL` is as close to the number of allocated MPI processes as possible.

`USE_OMP` When this option is activated, both the library and the test program expand OpenMP directives for building an application that can be executed in a distributed memory environment with multi-threaded nodes. The number of nodes in the process grid is specified as usual by allocating a number of MPI processes in combination with/without the `LOOPGRID` option. The number of threads per node is specified by the environment variable `OMP_NUM_THREADS`.

`USE_NEWPQR` When this option is activated, the current ScaLAPACK implementation of the unsymmetric QR algorithm is replaced with a new preliminary multishift version with advanced deflation techniques (see [19] for some preliminary results), which should give the general solvers for the standard equations a considerable speed boost.

`USE_AED_RES` When this option is activated together with `USE_NEWPQR`, the new parallel QR algorithm used in SCASY checks the residual of each Schur decomposition computed in each stage of aggressive early deflation and saves the maximum results to an output array. This option is mostly used for debugging purposes, and is not recommended for non-expert users.

We remark that these pre-processing options are included in the template `Make_include` files for the convenience of the user.

3.3.3 How to use the attached test program

Detailed instructions on how to use the attached test program are given in the 736 first lines of the corresponding source file `test/TESTSCASY.f`. By default, the program is tuned for testing all driver routines listed in Table 2 in sequence for matrices of size 2048, and when `LOOPGRID` is defined it will be using up to 4 MPI processes. From a user's point of view, the attached test program could be considered as a starting point for a simple installation test. From portability aspects across different parallel systems, the authors have refrained from making this test automatic.

As long as the user allocates the correct number of MPI processes with `mpiexec` or `mpirun` (or equivalent), and specifies the number of available bytes of double precision and integer memory[2] on lines 236–242, the test program should work fine.

3.3.4 Building the library and the test program

As stated above, the library is most easily built using the `Make_include` file (and the templates) located in the root directory `scasy[x.y]`. When `Make_include` has been properly modified, try

```
host:~> make all
```

to build the library in `/lib` and the test program in `/test`. To build only the library in the directory `/lib` try

```
host:~> make libscasy
```

3.3.5 Dependencies on external libraries

To be able to link SCASY with a test program (e.g., the accompanying test program in `/test`), the following libraries are needed: LAPACK [30], BLAS [11], ScaLAPACK/PBLAS

[2]The variables `NODEMEM` and `INODEMEM` located at lines 236–242 of the file `test/TESTSCASY.f` specifies the amount of double precision memory and integer memory allocated by the test program before the tests are conducted. The sum of these variables should be smaller than or equal to the available memory in bytes on each node of the target parallel platform. Their default ratio (5:1), makes sense when all condition estimators are turned on (which is default); with all condition estimators turned off, the amount of integer memory can be cut down considerably.

Routine	Functionality
SCASY drivers	
PGESYCTD	Solve general/triangular SYCT equation
PGELYCTD	Solve general/triangular LYCT equation
PGESYDTD	Solve general/triangular SYDT equation
PGELYDTD	Solve general/triangular LYDT equation
PGEGCSYD	Solve general/triangular GCSY equation
PGEGSYLD	Solve general/triangular GSYL equation
PGEGLY[C,D]TD	Solve general GLY[C,D]T equation
SCASY condition estimators	
PSYCTCON	Perform condition estimation on SYCT
PLYCTCON	Perform condition estimation on LYCT
PSYDTCON	Perform condition estimation on SYDT
PLYDTCON	Perform condition estimation on LYDT
PGSYLCON	Perform condition estimation on GSYL
PGCSYCON	Perform condition estimation on GSYL
PGLY[C,D]TCON	Perform condition estimation on GLY[C,D]T

Table 2: Summary of available driver and condition estimation routines and their functionality.

[35], BLACS [8] and RECSY [32] (which in turn depends on a few routines in SLICOT [34]). All these libraries are specified in the used `Make_include` file by their search paths and their respective names. For the download locations of these libraries, see the references, the README file or the library homepage [33].

4 Invoking the routines in SCASY

In this section, we describe in detail how the individual driver routines in SCASY are invoked properly. We also recommend the user to take a look at how the attached test program uses each specific driver.

4.1 Fortran interfaces for matrix equation drivers

In Table 2, we present the available high level driver routines in SCASY with their specific purposes. SCASY is designed and documented to be integrated in "state-of-the-art" software libraries like ScaLAPACK [7] and PSLICOT [34, 9]. Below we present the Fortran interfaces of the matrix equation driver routines. For the available routines, we also refer to the documentation available in the beginning of each individual source file; following the conventions of ScaLAPACK, all driver routines include an extensive documentation in the beginning of each source file. This documentation explains the purpose of each specific routine and how the routine should be invoked. Users of this package should be familiar with the ScaLAPACK philosophy of how to run parallel ScaLAPACK-style programs, see, e.g., Chapter 2 of the ScaLAPACK Users' Guide [7]. Expert users may also use the low level routines for convenience.

SCASY is designed to work with submatrices of globally distributed matrices. In the following, sub(X) denotes a submatrix *X(IX:IX+M,JX:JX+N)* (using MATLAB notation), where IX and JX point to the first row and column, and M and N are the number of rows

and columns of the considered submatrix, respectively.

4.1.1 Standard matrix equations

Below, the subroutine headings of our four general driver routines for the unreduced standard Sylvester-type matrix equations are listed.

- `SUBROUTINE PGESYCTD( JOB, ASCHUR, BSCHUR, TRANSA, TRANSB, ISGN, COMM, M, N, A, IA, JA, DESCA, B, IB, JB, DESCB, C, IC, JC, DESCC, MBNB2, DWORK, LDWORK, IWORK, LIWORK, NOEXSY, SCALE, INFO )`
  Solves the general continuous-time Sylvester (SYCT) equation, where sub(A) is $M \times M$, sub(B) is $N \times N$, and sub(C) and sub(X) (which overwrites sub(C)) are $M \times N$.

- `SUBROUTINE PGELYCTD( JOB, SYMM, OP, ASCHUR, M, A, IA, JA, DESCA, C, IC, JC, DESCC, NB2, DWORK, LDWORK, IWORK, LIWORK, NOEXSY, SCALE, INFO )`
  Solves the general continuous-time Lyapunov (LYCT) equation with general or symmetric right hand side C, where sub(A) is $M \times M$, and sub(C) and sub(X) (which overwrites sub(C)) are $M \times M$.

- `SUBROUTINE PGESYDTD( JOB, ASCHUR, BSCHUR, TRANSA, TRANSB, ISGN, COMM, M, N, A, IA, JA, DESCA, B, IB, JB, DESCB, C, IC, JC, DESCC, MB2, DWORK, LDWORK, IWORK, LIWORK, NOEXSY, SCALE, INFO )`
  Solves the general discrete-time Sylvester (SYDT) equation, where sub(A) is $M \times M$, sub(B) is $N \times N$, and sub(C) and sub(X) (which overwrites sub(C)) are $M \times N$.

- `SUBROUTINE PGELYDTD( JOB, SYMM, OP, ASCHUR, M, A, IA, JA, DESCA, C, IC, JC, DESCC, NB2, DWORK, LDWORK, IWORK, LIWORK, NOEXSY, SCALE, INFO )`
  Solves the general discrete-time Lyapunov equation (LYDT) with general or symmetric right hand side C, where sub(A) is $M \times M$, and sub(C) and sub(X) (which overwrites sub(C)) are $M \times M$.

The interfaces to the corresponding triangular solvers are not discussed since they are accessed through the corresponding general solvers by default (see, e.g., `ASCHUR` and `BSCHUR` below). In the following, we briefly describe the different interface arguments associated with the routines above. A summary of these arguments is listed in Table 3.

4.1.2 Mode parameters

The following mode parameters apply to the standard driver routines:

`JOB` The character mode parameter `JOB`, which takes the value `'R'` or `'S'`, chooses between *reduction mode* and *solving mode*. For the latter mode (`'S'`), the actual equation is reduced to triangular form *and* solved; for the former mode (`'R'`) the solver returns after the reduction. Such a stand-alone reduction is motivated by that it simplifies a subsequent call to the corresponding condition estimator in case the user does not specify a right hand side.

_SCHUR The character mode parameters ASCHUR and BSCHUR, which take the values 'N' or 'S', specify whether A and/or B are already in Schur form. For example, ASCHUR='S' and BSCHUR='S' correspond to a fully reduced triangular problem and no work related to the reduction will be performed. Expert users may instead call the triangular solver directly, but will then have to do more error checking in their calling program since SCASY has almost all error checking in the general routines.

TRANS_ The character mode parameters TRANSA and TRANSB (SYCT and SYDT) or OP (LYCT and LYDT), which take the values 'N' or 'T', switch between the transpose modes of the considered equation (see also Table 1).

SYMM For LYCT and LYDT, the character mode parameter SYMM, which takes the values 'N' or 'S', switches between the symmetric and nonsymmetric cases. SYMM='S' corresponds to symmetric right hand side and solution matrices and halves the number of flops needed for computing the solution. Moreover, with SYMM='S' only the lower or upper part of the right hand side matrix has to contain the symmetric matrix on input to the routine (see the software documentation for details).

ISGN The integer input parameter ISGN signals the sign ($+1$ or -1) in the actual equation (see Table 1).

4.1.3 Input/output arguments

The following input/output arguments apply to the standard driver routines:

COMM The character input/output argument COMM gives the user the opportunity to choose one out of two communication schemes in the triangular solver, the default *on demand* scheme (COMM='D') or the non-default *matrix block shift* scheme (COMM='S'). Matrix block shifting cannot be employed for all problems, so the corresponding triangular solver may switch to the *on demand* scheme if necessary. For this reason, COMM is also output from the routines showing which scheme that was actually used in the routine.

M,N The integer input arguments M and N specify the dimensions of the involved submatrices.

A,B,C The input/output two-dimensional double precision arrays A, B and C correspond to the globally distributed matrices A, B and C stored using column-major layout. On return from the routines, A and B are in real Schur form and in the solving mode (see above) C is overwritten with the solution matrix X. Notice that SCASY treats all matrices as *one-dimensional* arrays internally in the subroutines.

I_,J_ The integer input arguments IA, JA, IB, JB, IC and JC specify start rows and columns for the corresponding submatrices to operate on. These arguments must obey some alignment requirements to not violate the ScaLAPACK conventions. [3]

DESC_ The input integer descriptor arrays DESC_ correspond to the ScaLAPACK distributed matrix descriptors. These arrays consist of nine elements each and describe how the corresponding matrix is distributed across the process mesh, as follows:

[3]We remark that the current release of SCASY can not work on submatrices for the unreduced matrix equations because of missing functionality in the routines used in the reduction to triangular form, e.g., PDLAHQR from ScaLAPACK.

1. DESC_(1) contains the descriptor type. For dense matrices, DESC_(1) = 1.
2. DESC_(2) contains the identity of the corresponding BLACS *context*, which is very much like an MPI Communicator, over which the corresponding matrix is distributed.
3. DESC_(3) and DESC_(4) contain the total number of rows and columns of the corresponding globally distributed matrix, respectively. These dimensions should not be confused with M or N above.
4. DESC_(5) and DESC_(6) contain the blocking factors MB_ and NB_ used in the block-cyclic data layout of the corresponding globally distributed matrix in the row and column dimensions, respectively.
5. DESC_(7) and DESC_(8) contain the process row and column over which the first row and column of the corresponding globally distributed matrix is distributed, respectively.
6. DESC_(9) contains the leading dimension of the local part of the corresponding globally distributed matrix.

The first eight elements of a descriptor must be globally consistent for the corresponding context. For more information on the descriptor arrays, see, e.g., [7].

MBNB2 The input arguments MBNB2, which is an integer array of size 2, and MB2 and NB2, which are integer scalars, contain the internal blocking factors used in the multiple pipelining approach [17, 16] utilized in the corresponding triangular solvers. Multiple pipelining is turned off by using the same blocking factors for the pipelining as in the data distribution (i.e., in the matrix descriptors, see above).

4.1.4 Workspace

The following workspace arguments apply to the standard driver routines:

DWORK One-dimensional double precision workspace array

IWORK One-dimensional integer workspace array

LDWORK The length of the array DWORK

LIWORK The length of the array IWORK

4.1.5 Output information

The following output information arguments apply to the standard driver routines:

NOEXSY The triangular solvers handle 2×2 blocks shared by multiple data layout blocks by an *implicit redistribution* (see [20, 15] and Section 4.4) of the involved matrices (matrix pairs). This causes some subsystems to be of slightly different size and the integer output argument NOEXSY counts the number of such systems solved during the execution of the corresponding triangular solver.

SCALE The output double precision argument SCALE is a global scaling factor in the interval $(0, 1]$ for the right hand side used in the parallel solver to avoid overflow in the solution. SCALE corresponds to the scalar s in Table 1.

4.1.6 Error handling

The output integer argument `INFO` gives error messages, including overflow warnings, on output from the calling routine, as follows:

- If `INFO` < 0, some of the argument passed to the routine had an illegal value and `INFO` is set pointing to that particular argument.
- If `INFO` $= 0$, the routine was invoked successfully and returned without any error messages.
- If `INFO` $= 1$, there was no valid BLACS context [8] in the call and the call was aborted.
- If `INFO` $= 2$, the problem was very ill-conditioned and a perturbed nearly singular system was used to solve the corresponding matrix equations.
- If `INFO` $= 3$, the problem was badly scaled and the right hand side(s) was scaled by a factor `SCALE` to avoid overflow in the solution.
- If `INFO` $= 99$, the current call was to an empty wrapper to a non-existent reduction routine, see Section 4.4.2; no computations were performed and the invoked routine returned immediately.

4.1.7 Generalized matrix equations

Below, the subroutine headings of our four general driver routines for the unreduced generalized Sylvester-type matrix equations are listed.

- `SUBROUTINE PGEGCSYD( JOB, TRANZ, ADSCHR, BESCHR, TRANAD, TRANBE, ISGN, COMM, M, N, A, IA, JA, DESCA, B, IB, JB, DESCB, C, IC, JC, DESCC, D, ID, JD, DESCD, E, IE, JE, DESCE, F, IF, JF, DESCF, MBNB2, DWORK, LDWORK, IWORK, LIWORK, NOEXSY, SCALE, INFO )`
  Solves the unreduced generalized coupled Sylvester (GCSY) equation, where sub(A) and sub(D) are $M \times M$, sub(B) and sub(E) are $N \times N$, and sub(C), sub(X), sub(F) and sub(Y) are $M \times N$. Notice that sub(X) and sub(Y) overwrite sub(C) and sub(F) on output.

- `SUBROUTINE PGEGSYLD( JOB, ACSCHR, BDSCHR, TRANAC, TRANBD, ISGN, COMM, M, N, A, IA, JA, DESCA, B, IB, JB, DESCB, C, IC, JC, DESCC, D, ID, JD, DESCD, E, IE, JE, DESCE, MB2, DWORK, LDWORK, IWORK, LIWORK, NOEXSY, SCALE, INFO )`
  Solves the unreduced generalized Sylvester (GSYL) equation, where sub(A) and sub(C) are $M \times M$, sub(B) and sub(D) are $N \times N$, and sub(E) and sub(X) (which overwrites sub(E)) are $M \times N$.

- `SUBROUTINE PGEGLYCTD( JOB, SYMM, OP, AESCHR, M, A, IA, JA, DESCA, E, IE, JE, DESCE, C, IC, JC, DESCC, NB2, DWORK, LDWORK, IWORK, LIWORK, NOEXSY, SCALE, INFO )`
  Solves the unreduced generalized continuous-time Lyapunov (GLYCT) equation, with general or symmetric right hand side C, where sub(A) and sub(E) are $M \times M$, and

sub(C) and sub(X) (which overwrites sub(C)) are $M \times M$.

- `SUBROUTINE PGEGLYDTD( JOB, SYMM, OP, AESCHR, M, A, IA, JA, DESCA, E, IE, JE, DESCE, C, IC, JC, DESCC, NB2, DWORK, LDWORK, IWORK, LIWORK, NOEXSY, SCALE, INFO )`
  Solves the unreduced generalized discrete-time Lyapunov (GLYDT) equation, with general or symmetric right hand side C, where sub(A) and sub(E) are $M \times M$, and sub(C) and sub(X) (which overwrites sub(C)) are $M \times M$.

Below, we give a description of the additional arguments that do not exist for the standard matrix equations. A summary of these is listed in Table 3.

4.1.8 Mode parameters

The following additional mode parameters apply to the generalized driver routines:

`__SCHR` The character mode parameter `__SCHR`, which takes the values `'N'` or `'S'`, specifies whether the corresponding matrix pair is already in generalized Schur form or not[4].

`TRAN__` The character mode parameters `TRAN__` and `OP`, which take the values `'N'` or `'T'`, give the transpose mode for the corresponding matrix pair in the considered equation (see Table 1). The algorithms in SCASY are limited to the cases where the corresponding matrix pairs formed by the left and right multiplying matrices (see [17, 16]) have the same transpose mode.

`TRANZ` In condition estimation of GCSY, we need to solve transpose variants of the Kronecker product matrix representation Z_{GCSY} of the generalized coupled Sylvester operator which can not be expressed by changing the transpose modes of the involved left hand side coefficient matrices (see [17, 16]). Therefore, the corresponding transpose mode is specified by the character mode argument `TRANZ`, which takes the values `'N'` or `'T'`, in the routine `PGEGCSYD`.

4.1.9 Input/output arguments

The following additional input/output arguments apply to the generalized driver routines:

`D,E,F` The input/output two-dimensional double precision arrays `A`, `B`, `C`, `D`, `E` and `F` correspond to the matrices A, B, C, D, E and F. On return from the routines, the involved matrix pairs (excluding the right hand side pair (E, F) in GCSY) are in generalized real Schur form. In solving mode (see above), the right hand side matrix (or matrix pair) is overwritten with the solution matrix (pair). As mentioned before, SCASY treats all matrices as *one-dimensional* arrays internally in the subroutines.

`I_,J_` The input integer arguments `IA`, `JA`, `IB`, `JB`, `IC`, `JC`, `ID`, `JD`, `IE`, `JE`, `IF` and `JF`, specify start rows and columns for the corresponding submatrices of A, B, C, D, E, and F to operate on. These arguments must obey some alignment requirements to not violate ScaLAPACK conventions.

[4] In the current release, no reduction to generalized Schur form is performed, see also Section 4.4.2.

Table 3: Parameters to Fortran interfaces

Mode parameters	Type	Description
JOB	CHARACTER*1	Specifies solving mode ('S') or reduction mode ('R').
SYMM	CHARACTER*1	Specifies symmetric right hand side ('S') or not ('N')).
OP	CHARACTER*1	Specifies transpose mode ('N' or 'T'). Only for Lyapunov equations.
TRANZ	CHARACTER*1	Specifies transpose mode for Kronecker product matrix representation ('N' or 'T'). Only applies to GCSY.
_SCHUR	CHARACTER*1	Specifies if the matrix is in real Schur form ('S') or not ('N'). Only for standard matrix equations.
__SCHR	CHARACTER*1	Specifies if the matrix pair is in generalized real Schur form ('S') or not ('N'). Only for generalized matrix equations.
TRANS_	CHARACTER*1	Specifies transpose mode for specific matrix ('N' or 'T').
TRAN__	CHARACTER*1	Specifies transpose mode for specific matrix pair ('N' or 'T').
ISGN	INTEGER	Specifies the sign variant of the equation (-1 or 1).
Input/Output arguments		
COMM	CHARACTER*1	Sets communication scheme used in PTR[ACRO]D.
M,N	INTEGER	(Sub)matrix dimensions.
A,B,C,D,E,F	DOUBLE PRECISION(*)	Two-dimensional arrays corresponding to the local parts of the globally distributed matrices. On output, each left hand side matrix (pair) is returned in Schur (generalized Schur) form. On output and in solving mode, right hand side is overwritten with the solution. In reduction mode, the right hand side is not referenced.
IA,IB,IC,ID,IE,IF	INTEGER	Row starting indices for submatrices to operate on.
JA,JB,JC,JD,JE,JF	INTEGER	Column starting indices for submatrices to operate on.
DESC_	INTEGER(*)	ScaLAPACK matrix descriptor arrays.
MBNB2	INTEGER(2)	Blocking factors for multiple pipelining in one-sided Sylvester equations.
MB2	INTEGER	Blocking factor for multiple pipelining in two-sided Sylvester equations.
NB2	INTEGER	Blocking factor for multiple pipelining in Lyapunov equations. Currently used for LYCT.
Workspace		
DWORK	DOUBLE PRECISION(*)	Double precision workspace.
LDWORK	INTEGER	Length of DWORK.
IWORK	INTEGER(*)	Integer workspace.
LIWORK	INTEGER	Length of IWORK.
Output information		
NOEXSY	INTEGER	Counts the number of extended/diminished subsystems solved in the call to the triangular solver.
SCALE	DOUBLE PRECISION	Right hand side scaling factor, $0 <$ SCALE ≤ 1.0.
EST	DOUBLE PRECISION	A 1-norm based lower bound estimate of sep^{-1}[ACRO] (condition estimator only).
NOITER	INTEGER	Number of performed iterations in computing sep^{-1}[ACRO] (condition estimator only).
Error handling		
INFO	INTEGER	Returns error information to the calling program.

4.2 Condition estimators

Parallel implementations of the condition estimators presented in [17, 16] are available in SCASY as the routines P[ACRO]CON. The Fortran interfaces to the different estimators are derived from the corresponding solvers. For example, the SYCT condition estimator has the following Fortran interface:

- SUBROUTINE PSYCTCON(TRANSA, TRANSB, ISGN, COMM, M, N, A, IA, JA, DESCA, B, IB, JB, DESCB, MBNB2, DWORK, LDWORK, IWORK, LIWORK, EST, NOITER, INFO)
  Computes a 1-norm based lower bound estimate EST of $\mathrm{sep}^{-1}(\mathrm{sub}(A), \mathrm{sub}(B))$, where $\mathrm{sub}(A)$ is $M \times M$ and $\mathrm{sub}(B)$ is $N \times N$, using NOITER iterations and calls to PTRSYCTD (via PGESYCTD with ASCHUR='S' and BSCHUR='S').

The differences from the general SYCT interface are that it is assumed that the matrix equation is in reduced form, there is no argument for the right hand side C since it is generated internally by the estimator (ScaLAPACK's PDLACON), and the two output arguments EST and NOITER which give the computed estimate and the number of iterations (the number of triangular matrix equations solved) needed to compute the estimate.

The implemented condition estimators assume that the corresponding matrix equations are in reduced (triangular) form. If this is not the case, the user must perform the reduction step by one single call to the corresponding general solver with the mode parameter JOB set to 'R'.

4.3 Test example generators

SCASY includes two problem generator routines which generate matrices or matrix pairs with specified standard or generalized eigenvalues, with the following interfaces (see the software documentation for details regarding the arguments):

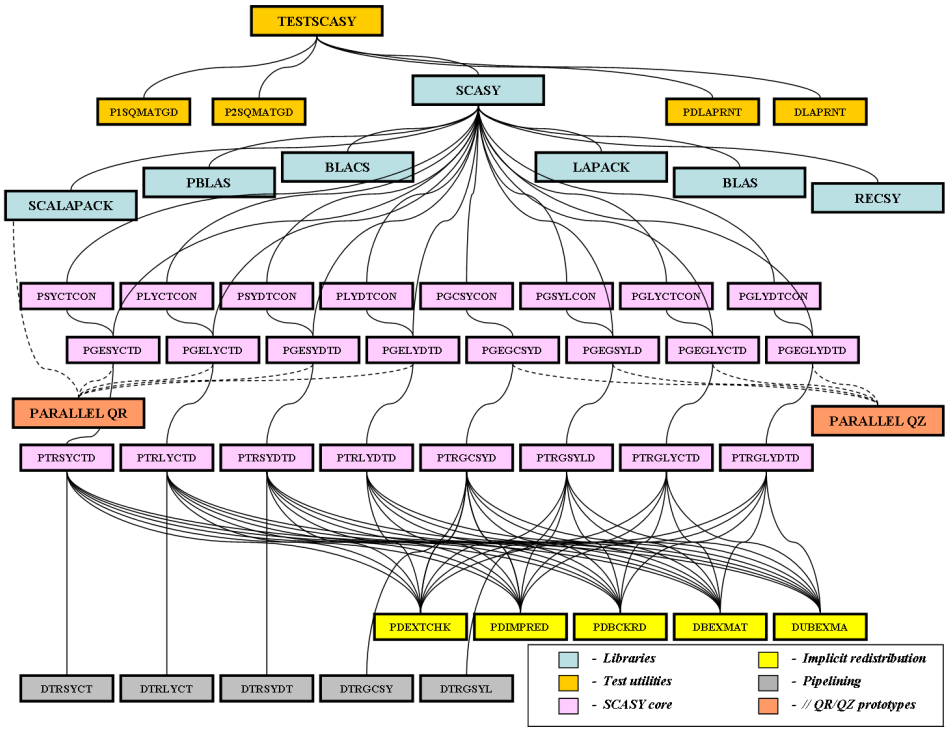
- SUBROUTINE P1SQMATGD(DIAG, SDIAG, UPPER, M, A, DESCA, DW, MW, DDIAG, SUBDIAG, NQTRBL, ASEED, DWORK, LDWORK, INFO)
  Generates the $M \times M$ matrix A with eigenvalues specified by DDIAG and SUBDIAG.

- SUBROUTINE P2SQMATGD(ADIAG, ASDIAG, BDIAG, UPPER, M, A, DESCA, ADW, AMW, ADDIAG, ASUBDIAG, ANQTRBL, ASEED, B, DESCB, BDW, BMW, BDDIAG, BSEED, DWORK, LDWORK, INFO).
  Generates the $M \times M$ matrix pair (A, B) with generalized eigenvalues specified by ADDIAG, ASUBDIAG and BDDIAG.

P1SQMATGD generates coefficient matrices for the standard matrix equations as follows: Consider a matrix $A \in \mathbb{R}^{m \times m}$ in the form $A = Q(\alpha_A D_A + \beta_A M_A)Q^T$, where D_A is block diagonal with 1×1 and 2×2 blocks, M_A is strictly upper triangular with zeros in the first superdiagonal where D_A has 2×2 blocks, Q is a random orthogonal matrix and α_A and β_A are real scalars. We choose M_A as a random matrix with uniformly distributed elements in the interval $[0, 1]$ and prescribe the eigenvalues of A by specifying the elements of D_A, where the 2×2 blocks correspond to complex conjugate pairs of eigenvalues.

P2SQMATGD generates test matrix pairs for the generalized matrix equations in a similar way by specifying the generalized eigenvalues for a given diagonal matrix pair $(D_A, D_B) \in \mathbb{R}^{(m \times m) \times 2}$ (2×2 blocks are only allowed in D_A) and performing the equivalence transformation $(A, B) = X^T(D_A, D_B)Y$, where X and Y are invertible matrices which are constructed as follows: we specify their corresponding singular values $\Sigma_X = \mathrm{diag}(\sigma_1, \ldots, \sigma_m)$ and $\Sigma_Y = \mathrm{diag}(\rho_1, \ldots, \rho_m)$ and generate four random orthogonal matrices U_1, U_2, V_1 and V_2 such that $X = U_1 \Sigma_U V_1^T$ and $Y = U_2 \Sigma_U V_2^T$. In practice, the matrices Σ_X and Σ_Y are not generated

Figure 1: Subroutine call graph of SCASY.



explicitly, but the singular values are used to scale the corresponding rows and columns in the congruence transformations above. The conditioning of X and Y are controlled by Σ_X and Σ_Y.

In the current release, the matrix generation routines do only support fully distributed testmatrices (and not submatrices of a global matrix).

4.4 Software hierarchy

In total, the current release of the SCASY library consists of over 70 routines whose design depends on the functionality of a number of external libraries. The call graph in Figure 1 shows the subroutine hierarchy in SCASY. The six types of software included in the call graph are presented in detail in [15]. Below we give a glance of the internals of the SCASY software by presenting three types of the software in Figure 1: *library dependencies*, *parallel QR/QZ prototypes* and the routines used for performing *implicit redistribution*.

4.4.1 Libraries

The following external libraries are used in SCASY:

- ScaLAPACK [7, 35] including the PBLAS [31] and BLACS [8],
- LAPACK and BLAS [5],
- RECSY [27], which provides almost all node solvers except for one transpose case of the GCSY equation. Notice that RECSY in turn calls a small set of subroutines from SLICOT (*Software Library in Control*) [34, 12].

For example, the routines for the standard matrix equations utilize the ScaLAPACK routines PDGEHRD, which performs a parallel Hessenberg reduction, PDLAHQR, which is the parallel unsymmetric QR algorithm presented in [22], and PDGEMM, the PBLAS parallel implementation of the level-3 BLAS GEMM-operation. The triangular solvers employ the RECSY node solvers [25, 26, 27] and LAPACK's DTGSYL (see also [29]) for solving (small) matrix equations on the nodes and the BLAS for the level 3 updates (DGEMM, DTRMM and DSYR2K operations). To perform explicit communication and coordination in the triangular solvers we use the BLACS library.

4.4.2 Parallel QR/QZ prototypes

The reduction to standard Schur form in the non-generalized matrix equations are in the current release performed using existing ScaLAPACK software. However, as pointed out and demonstrated in [17], the current QR algorithm in ScaLAPACK represents a time-consuming bottleneck of the reduction step in the standard driver routines. A proposed revision of the parallel QR algorithm with level 3 node performance and advanced deflation techniques is presented in [19]. Preliminary versions of these algorithms are attached to the software package, and are accessed through wrappers to the existing ScaLAPACK wrappers, provided the pre-processing option USE_NEWPQR is defined on compile time (see also Section 3.3.2). The following is the assumed Fortran interface of the improved standard Schur reduction algorithm in case of a direct call:

- SUBROUTINE PDHSEQR(JOB, COMPZ, N, ILO, IHI, H, DESCH, WR, WI, Z, DESCZ, WORK, LWORK, IWORK, LIWORK, INFO)
  Computes the eigenvalues of the Hessenberg matrix H and, optionally, the matrices T and Z from the Schur decomposition $T = Z^T H Z$, where T is the Schur form and Z is the orthogonal matrix of Schur vectors.

The reduction of the involved matrix pairs to generalized real Schur form is performed by linking with and invoking the new prototype ScaLAPACK-style implementations of the Hessenberg-triangular reduction and the parallel multi-shift QZ algorithm presented in [3, 4]. Presently, all associated QZ routines are provided as empty wrappers which when invoked immediately return with the error code INFO=99, signalling that no computations were performed. The following are the assumed Fortran interfaces of the generalized Schur reduction algorithms:

- SUBROUTINE PDGGHRD(COMPQ, COMPZ, N, ILO, IHI, A, DESCA, B, DESCB, Q, DESCQ, Z, DESCZ, WORK, LWORK, INFO)
  Computes the Hessenberg-triangular form (H, T) by the orthogonal equivalence transformation $(H, T) = Q^T(A, B)Z$, where the matrix pair (A, B) is regular and B is assumed to be upper triangular by an initial QR factorization.

- SUBROUTINE PDHGEQZ(JOB, COMPQ, COMPZ, N, ILO, IHI, A, DESCA, B, DESCB, ALPHAR, ALPHAI, BETA, Q, DESCQ, Z, DESCZ, ILOQ, IHIQ, ILOZ, IHIZ, MXBLGS, WORK, LWORK, INFO)
  Computes the generalized eigenvalues of the regular Hessenberg-triangular matrix pair (H, T) and, optionally, the matrix pairs (S, T) and (Q, Z) from the generalized Schur decomposition $(S, T) = Q^T(H, T)Z$, where (S, T) is the generalized Schur form and Q and Z are orthogonal matrices of generalized Schur vectors.

These Schur reduction algorithms are still under development and are not a part of the current SCASY contribution, which is emphasized using dashes line in the subroutine hierarchy in Figure 1. When production software for this software is available they will be included in future SCASY releases. For an explanation of the arguments of these Fortran interfaces, we refer to the software documentation, which follows the standard (Sca)LAPACK source documentation style.

4.4.3 Implicit redistribution

The handling of the implicit redistribution, caused by 2×2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues), shared by multiple data layout blocks (and processors) in the left hand side quasi-triangular matrices of the matrix equations, is concentrated to a few routines, as follows:

- `PDEXTCHK` searches the diagonal blocks of a given matrix, say A, looking for any 2×2 blocks shared by several data layout blocks. The routine returns the following *redistribution* information which decides which $mb_A \times nb_A$ diagonal blocks A_{ii} that should be extended or diminished one row and column to include all elements of a shared 2×2 block:

$$EXT\_INFO\_A(i) = \begin{cases} 0 & \text{if } A_{ii} \text{ is unchanged} \\ 1 & \text{if } A_{ii} \text{ is extended} \\ 2 & \text{if } A_{ii} \text{ is diminished} \\ 3 & \text{if } A_{ii} \text{ is extended } \textit{and} \text{ diminished} \end{cases} .$$

  This information is broadcasted to all processors and is used in

- `PDIMPRED`, in which data is exchanged between the processors via message passing to build up local arrays of extra elements which are used while constructing and decomposing the "correct" submatrices locally on the nodes by invoking the serial routines `DBEXMAT` and `DUBEXMA`.

- `PDBCKRD` is called right before returning from the corresponding triangular solver and sends back the redistributed parts of the solution matrix (pair) to their original owner processes such that the solution matrix (pair) is correctly distributed over the process mesh on output.

These routines are implemented in a general manner such that they are easy to reuse in other applications involving distributed quasi-triangular matrices.

5 Terms of usage

Any use of ACM Algorithms is subject to the ACM Software Copyright and License Agreement [2]. Moreover, use of the SCASY library should also be acknowledged by citing the corresponding papers [17, 16] and referring to the SCASY webpage [33].

6 Conclusions and future work

We have presented the 1.0 release of the high performance software library SCASY. The latest version of the library along with updated information and documentation is always

available for download from the SCASY website [33]. We welcome bug-reports, comments and suggestions from users.

Acknowledgments

The authors are grateful to the support staff of HPC2N [24] for help with preparing the codes for this release.

SCASY's main contributors are Robert Granat and Bo Kågström, Department of Computing Science and HPC2N, Umeå University, Sweden. Collaborators and secondary (indirect) contributors have been Björn Adlerborn, Isak Jonsson and Daniel Kressner.

References

[1] ACM TOMS Algorithms Policy. See `http://toms.acm.org/AlgPolicy.html`.

[2] ACM Software Copyright and License Agreement. See `http://www.acm.org/publications/policies/softwarecrnotice`.

[3] B. Adlerborn, K. Dackland, and B. Kågström. Parallel and blocked algorithms for reduction of a regular matrix pair to Hessenberg-triangular and generalized Schur forms. In J. Fagerholm and et al., editors, *Applied Parallel Computing PARA 2002*, volume 2367 of *Lecture Notes in Computer Science*, pages 319–328. Springer-Verlag, 2002.

[4] B. Adlerborn, B. Kågström, and D. Kressner. Parallel Variants of the Multishift QZ Algorithm with Advanced Deflation Techniques . In B. Kågström et al., editor, *Applied Parallel Computing: State of the Art in Scientific Computing, PARA 2006*, Lecture Notes in Computer Science, LNCS 4699, pages 117–126. Springer, 2007.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

[6] R. H. Bartels and G. W. Stewart. Algorithm 432: The Solution of the Matrix Equation $AX - BX = C$. *Communications of the ACM*, 8:820–826, 1972.

[7] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. W. Demmel, I. Dhillon, J. J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997.

[8] BLACS - Basic Linear Algebra Communication Subprograms. See `http://www.netlib.org/blacs/index.html`.

[9] I. Blanquer, D. Guerrero, V. Hernandez, E. Quintana-Orti, and P. Ruiz. Parallel-SLICOT implementation and documentation standards, 1998.

[10] Collected Algorithms of the ACM. See `http://calgo.acm.org`.

[11] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Software*, 16:1–17, 1990.

[12] E. Elmroth, P. Johansson, B. Kågström, and D. Kressner. A web computing environment for the SLICOT library. In *The Third NICONET Workshop on Numerical Control Software*, pages 53–61, 2001.

[13] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[14] R. Granat and B. Kågström. Parallel Solvers for Sylvester-type Matrix Equations with Applications in Condition Estimation, Part I: Theory and Algorithms. Report UMINF-07.15, Dept. Computing Science, Umeå University, Sweden, 2007. Revised August 2009.

[15] R. Granat and B. Kågström. Parallel Solvers for Sylvester-type Matrix Equations with Applications in Condition Estimation, Part II: The SCASY Software. Report UMINF-07.16, Dept. Computing Science, Umeå University, Sweden, 2007. Revised August 2009.

[16] R. Granat and B. Kågström. ALGORITHM XXX: The SCASY Software Library – Parallel Solvers for Sylvester-type Matrix Equations with Applications in Condition Estimation, Part II. 2009. *ACM Transactions on Mathematical Software* (submitted July 2007, revised January and August 2009, and March 2010).

[17] R. Granat and B. Kågström. Parallel Solvers for Sylvester-type Matrix Equations with Applications in Condition Estimation, Part I: Theory and Algorithms. 2009. *ACM Transactions on Mathematical Software* (submitted July 2007, revised January and August 2009, and March 2010).

[18] R. Granat and B. Kågström. SCASY Users' Guide. Report UMINF-09.10, Dept. Computing Science, Umeå University, Sweden, 2010.

[19] R. Granat, B. Kågström, and D. Kressner. A novel parallel QR algorithm for hybrid distributed memory HPC systems. Submitted to *SIAM Journal on Scientific Computing*, 2009. From Technical report UMINF-09.06, also as LAPACK Working note #216.

[20] R. Granat, B. Kågström, and P. Poromaa. Parallel ScaLAPACK-style Algorithms for Solving Continuous-Time Sylvester Equations. In H. Kosch and et al, editors, *Euro-Par 2003 Parallel Processing*, volume 2790 of *Lecture Notes in Computer Science*, pages 800–809. Springer, 2003.

[21] W.W. Hager. Condition estimates. *SIAM J. Sci. Statist. Comput.*, (3):311–316, 1984.

[22] G. Henry, D. S. Watkins, and J. J. Dongarra. A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. *SIAM J. Sci. Comput.*, 24(1):284–311, 2002.

[23] N. J. Higham. Fortran codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Trans. of Math. Software*, 14(4):381–396, 1988.

[24] HPC2N - High Performance Computing Center North. See `http://www.hpc2n.umu.se`.

[25] 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. Software*, 28(4):392–415, 2002.

[26] 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. Software*, 28(4):416–435, 2002.

[27] I. Jonsson and B. Kågström. RECSY - A High Performance Library for Solving Sylvester-Type Matrix Equations. In Kosch H. et al, editor, *Euro-Par 2003 Parallel Processing*, volume 2790 of *Lecture Notes in Computer Science*, pages 810–819, 2003.

[28] B. Kågström and P. Poromaa. Distributed and shared memory block algorithms for the triangular Sylvester equation with sep^{-1} estimators. *SIAM J. Matrix Anal. Appl.*, 13(1):90–101, 1992.

[29] B. Kågström and P. Poromaa. Computing eigenspaces with specified eigenvalues of a regular matrix pair (A, B) and condition estimation: theory, algorithms and software. *Numer. Algorithms*, 12(3-4):369–407, 1996.

[30] LAPACK - Linear Algebra Package. See `http://www.netlib.org/lapack/`.

[31] PBLAS - Parallel Basic Linear Algebra Subprograms. See `http://www.netlib.org/scalapack/pblas`.

[32] RECSY - High Performance library for Sylvester-type matrix equations. See http://www8.cs.umu.se/research/parallel/recsy.

[33] SCASY - ScaLAPACK-style solvers for Sylvester-type matrix equations. See http://www8.cs.umu.se/research/parallel/scasy.

[34] SLICOT Library In The Numerics In Control Network (Niconet). See http://www.win.tue.nl/niconet/index.html.

[35] ScaLAPACK Users' Guide. See http://www.netlib.org/scalapack/slug/.