# Installation and usage of this library

Souji Koikari

September 26, 2008

Thank you very much for your interest on this library.

## 1 Description

This is a FORTRAN 95 library that consists of the following two parts:

- the modified scaling and squaring method for computing matrix values of $\varphi$-functions defined by

$$\varphi_n(hA) := \sum_{k=0}^{\infty} \frac{(hA)^k}{(k+n)!}, \quad A \in \mathbb{C}^{N\times N}, \;\; h \in \mathbb{C}, \;\; 0 \leq \forall n \leq 5, \tag{1}$$

- the modified block Schur–Parlett algorithm for $\varphi$-functions based on the sep-inverse estimate.

All functions included in the library accept, as their arguments, real or complex matrices in one of four precisions. The assumed unit roundoff is $2^{-24}$, $2^{-53}$, $2^{-64}$, and $2^{-113}$, respectively. The modified scaling and squaring method may be applied to diagonal, upper quasi-triangular, or square matrices, while the modified block Schur–Parlett algorithm can be used only for upper quasi-triangular matrices. The details of the algorithm are described in the accompanying paper published in the journal, "ACM transactions on mathematical software". The usage of the library is described in the following, where we assume a UNIX-compatible system.

## 2 Install

### Requirement

The only requirement is a FORTRAN 95 compiler that supports a C-style preprocessor. The preprocessor is usually invoked by a command-line option such as `-fpp` or `-cpp`. If your compiler does not support the preprocessor, please use the `cpp` command directly, and eliminate all non-FORTRAN statements from the output. We have tested the library with `f95` of Sun Microsystems [6], `ifort` of Intel [4], and `g95` developed by Dr. Andy Vaught [7]. Unfortunately, the library may not work with the current release of `gfortran`.

### Directories

Six directories are found after uncompressing and extracting the source archive. The top directory `matrixcfgphi` of the archive is assumed to be a current directory.

`./src/` includes all source files of the library.

`./examples/` includes a test program and two examples.

`./lapack/` includes partially the source files of LAPACK [1]. The files in this directory are required to use Hager–Higham estimator `xLACON` [2, 3] in the example programs.

`./Makefiles/` includes makefiles for several FORTRAN compilers.

`./lib/` is empty. The above example programs assume that library archives, `libmtrcfgphi.so` and `libxlacon.so`, are stored in this directory. These libraries will be created in the directories, `./src/` and `./lapack/`, respectively. When the shared object is unavailable, the extension "`.a`" is used instead of "`.so`".

`./modules/` is empty. The above example programs assume that the modules files, `*.mod`, which will be created in `./src/`, are stored here.

## Make and test with `f95`, `ifort`, `g95`, or NAGWare F95

If your compiler is one of `f95` [6], `ifort` [4], `g95` [7], and NAGWare F95, please use one of the makefiles in `./Makefiles/`. The corresponding makefiles are named as `Makefile.f95sun`, `Makefile.ifort`, `Makefile.g95`, and `Makefile.nagware`, respectively. For example, to build the library with `f95` of Sun Microsystems,

```
% cat ./Makefiles/Makefile.f95sun > Makefile
% make all
```

Then, all module files and the library archives are created and stored in `./modules/` and `./lib/`, respectively. The root permission is not required. Example programs in `./examples` are also compiled, and three executable commands, `selftestall.out`, `examplesquare.out`, and `exampleparlett.out`, are generated. The first one is the test program and the others are examples, all of which may be executed without command line options. When you use `ifort`, it may be necessary to add the full pathname of `./lib/` to your environment variable, `LD_LIBRARY_PATH`.

If the above procedure does not work well, or you want to modify command-line options, please edit the following three files:

- `./src/Makefiles/Makefile.xxxxx`,
- `./lapack/Makefiles/Makefile.xxxxx`, and
- `./examples/Makefiles/Makefile.xxxxx`,

where `xxxxx` is one of `f95sun`, `ifort`, `g95`, and `nagware`. These four files are implicitly used in the procedure described in the previous paragraph.

## Make and test with the other compilers

For the other compilers, please edit the first few lines of the following three files:

- `./src/Makefiles/Makefile.generic`,
- `./lapack/Makefiles/Makefile.generic`,
- `./examples/Makefiles/Makefile.generic`.

It is necessary to specify the name of your compiler and its command-line option to invoke C-style preprocessor. In the last file of the above list, the name of the directory storing module files must also be specified as, e.g., `-M../modules`, `-I../modules`, or `-module ../modules`. After editing all these files, execute the following commands in the top directory, `matrixcfgphi`, of the source archive.

```
% cat ./Makefiles/Makefile.generic > Makefile
% make all
```

All module files and the library archives are created and stored in `./modules/` and `./lib/`, respectively. The root permission is not required. Example programs in `./examples` are also compiled, and three executable commands, `selftestall.out`, `examplesquare.out`, and `exampleparlett.out`, are generated. The first one is the test program and the others are examples, all of which may be executed without command line options.

### LAPACK

The directory `./lapack` is included only for linking the example programs. If the library LAPACK is already installed on your computer, and you are not interested in the example programs, the directories, `./lapack` and `./examples`, and the archive `libxlacon.so` are not necessary. In this case, the installation can be done in the following way.

```
% \rm -fr ./lapack/ ./examples/
% cd ./src/
% cat ./Makefiles/Makefile.xxxxx > Makefile
% make all install
```

where `xxxxx` is one of `f95sun`, `ifort`, `g95`, and `nagware`. The module files are created and stored in `./modules`, and the library archive `libmtrcfgphi.so` are created and stored in `./lib`.

# 3 Usage

## 3.1 Generic interface

This subsection describes the usage of the library under its default configuration. Each function for real or complex matrices in one of the four precisions can be called by a common generic name independent of data types and floating-point precisions. The contents of the arguments are not modified. In the following description, "an upper quasi-triangular matrix" means a block upper-triangular matrix whose diagonal blocks are at most two-by-two and each diagonal block stores a conjugate pair of eigenvalues.

### 3.1.1 Scaling and squaring method

The generic name for computing $\varphi$-functions of a square or a diagonal matrix by the modified scaling and squaring method is `sasmtrphif`, and that for an upper quasi-triangular matrix is `sasqtrphif`. The following is an example for a square matrix.

```
use scalesquare
double precision :: hci, A(N,N), phi(N,N,0:5)
phi = sasmtrphif(5,hci,A)              ! after defining hci and A.
```

Of course, the matrix `A` and the scalar `hci` must be defined before calling the function. The next example treats a diagonal matrix.

```
use scalesquare
complex :: hci, A(N), phi(N,0:2)       ! A is an N-by-N diagonal matrix.
phi = sasmtrphif(2,hci,A)              ! after defining hci and A.
```

As is shown in the above example, a one dimensional array (a "rank one array" in the FORTRAN terminology) is regarded as a diagonal matrix. In these two examples, the functions, $\varphi_n$(`hci*A`), for $0 \leq \forall n \leq 5$ and for $0 \leq \forall n \leq 2$ are computed, respectively. The next example shows an upper quasi-triangular case.

```
use scalesquare
real :: hci, T(N,N), phi(N,N,0:3)      ! T should be upper quasi-triangular.
phi = sasqtrphif(3,hci,T)              ! after defining hci and T.
```

Real matrices should be upper quasi-triangular, and complex matrices must be strictly upper triangular. Any unnecessary entries in the lower triangle should be exactly zero.

### 3.1.2 Modified block Schur–Parlett algorithm

The interface of the module implementing the modified block Schur–Parlett algorithm consists of two functions. The function `bspqtr1dcmpf` determine a block decomposition based on the sep-inverse estimate. The other function, `bspqtrphif`, computes $\varphi$-functions according to the blocking produced by the above function. The following code shows a typical example of usage.

```
use blockdecomp
use schurparlett
double precision :: hci, T(N,N), phi(N,N,0:4)
type(bspblock) :: info
                                        ! After defining hci and T,
info = bspqtr1dcmpf(T)                  ! determines a block decomposition.
phi = bspqtrphif(4,hci,T,info)          ! computes phi-functions up to phi_4.
```

In the above example, a block decomposition is determined based on the sep-inverse estimate provided by Hager–Higham estimator. A decomposition once determined by `bspqtr1dcmpf` and stored in a variable of the type `bspblock` may be used repeatedly for the other values of the scaling, `hci`. Real matrices should be upper quasi-triangular, and complex matrices must be strictly upper triangular. Any unnecessary entries in the lower triangle should be exactly zero.

## 3.2 Other optional features

### 3.2.1 BLAS

When the intrinsic function, `matmul`, automatically invokes `xGEMM` or its equivalence, the best efficiency of this library is already available as it is. If this is not the case and the BLAS library is installed on your computer, please add another option, `-Dpure= -D__USE_BLAS`, literally to the command line argument of your FORTRAN compiler. The space after `pure=` is mandatory. The makefiles provided here already include this additional option as commented lines. By this option, `xGEMM` is explicitly called from the library instead of `matmul`, and efficiency may be improve significantly, though the pure attribute is lost for all functions. This modification is not necessary for `f95` command of Sun Microsystems, while it is recommended for `ifort` command of Intel corporation.

### 3.2.2 Block size

The block size used for all block algorithms in the library is determined by a public parameter, `mpt_blksize`, defined in a file, `matrixpwrtag.f95`, as

```
integer, parameter :: mpt_blksize = 50
```

If you want to use another block size, please edit this line, and recompile all files to build a new library archive. The value of this parameter must be greater than 3. This parameter is defined as a compile-time constant, because it helps significantly the compiler's optimization.

### 3.2.3 Higher precision

The precisions higher than double ($u = 2^{-53}$) are disabled in the default configuration. If these precisions are necessary, please add another option, `-D__USE_TPREC` or `-D__USE_QPREC`, to the command line argument of your FORTRAN compiler. The former enables the triple precision ($u = 2^{-64}$) and the latter enables the quadruple precision ($u = 2^{-113}$). The availability of these higher precisions depends on a compiler and a computer. For example, `g95` on x86 machines supports the triple precision, and `ifort` supports the quadruple precision. Sun's `f95` for x86 architecture does not support these precisions, while on a SPARC-based system, the quadruple precision is supported. Please do not enable an unsupported precision, because it may cause a compile-time error. When the test program, `selftestall.f95`, is compiled with the same option, the enabled higher precision is also tested.

### 3.2.4 Modified block Schur–Parlett algorithm

*Tuning parameter P.* The tuning parameter $P$ that bounds the estimated $\tau_d$ and $\tau_o$ may be chosen freely by setting a public variable, `blksp1maxcondition`, as

```
use blockdecomp
blksp1maxcondition = 1500.0
```

The default value is 150, which usually works well within our experience. When efficiency is more important than accuracy, a larger value may be chosen as in the above example. We do not recommend to choose an excessively large value, because such a choice may result in the overflow of computation.

*Block-wise computation.* In the default configuration, the functions of each diagonal block are computed separately from other diagonal blocks. The advantage of this approach is that we can expect high accuracy even if the spectral radius of the argument is large. Hence, the default choice is suited for implementing exponential integrators. However, efficiency may be unsatisfactory for certain cases, because we must repeat equivalent operations several times in the computation of the functions of diagonal blocks. When the spectral radius of the argument is small ($O(10) \sim O(100)$), and the relative departure from normality ("dep($A$)" defined in the paper) is large ($1 \sim 4$), we recommend to use the technique described in Section 2.3 of the accompanying paper. To enable the technique, please set the module variable `blockwisefunc` as `blockwisefunc=.false.`. This variable is defined in the module `schurparlett`, and its default value is `.true.`.

*Simultaneous computation.* The other run-time variable is `blkspfuncstride` defined in the module `blockdecomp`. The value of this variable must be one or two. If the value is two, which is the default choice, two $\varphi$-functions are computed simultaneously at the cost of one recursion by utilizing the relation,

$$\varphi_{j-1}(A) = I/(j-1)! + A\varphi_j(A). \tag{2}$$

When we compute, for example, $\varphi_j(A)$ for $0 \leq \forall j \leq 5$, the functions with odd indices are computed by the recurrence, and the functions with even indices are simultaneously obtained according to the above equation. If this approach fails, i.e., the error of $\varphi$-functions with even (or odd) indices is obviously larger, please set `blkspfuncstride=1`, then all functions are computed by Parlett's recurrence, and the problem may be avoided, though the computational cost may increase.

### 3.2.5 Interface to external libraries

*LAPACK.* If the LAPACK library is already available on your computer, please remove the library archive `./lib/libxlacon.so`, and use your own LAPACK with your programs. If you want to enable the interface to LAPACK subroutines other than `xLACON`, please add another option `-Dpure= -D__USE_LAPACK` to the command-line options of your compiler. Although the pure attribute is lost, the subroutines in LAPACK will be used as many as possible.

*RECSY.* The library has its interface to the library RECSY developed by Jonsson and Kågström [5]. It is enabled by adding `-Dpure= -D__USE_RECSY` to the command-line options of your compiler. Although the pure attribute is lost, all Sylvester equations defined by real matrices in double precision are solved by `RECSYCT`.

■

# References


[1] E. Anderson, Z. Bai, C. H. Bishof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, third edition, 1999.

[2] W. W. Hager. Condition estimates. *SIAM Journal on Scientific and Statistical Computations*, 5:311–316, 1984.

[3] Nicholas J. Higham. FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM transactions on mathematical software*, 14(4):381–396, 1988.

[4] Intel corporation. Intel Fortran compiler. http://developer.intel.com/software/products/, 2008.

[5] Isak Jonsson and Bo Kågström. Recursive blocked algorithms for solving triangular systems. I. One-sided and coupled Sylvester-type matrix equations. *ACM Transactions on Mathematical Software*, 28((4)):392–415, 2002.

[6] Sun Microsystems. Sun Fortran 95 compiler. http://developers.sun.com/sunstudio/, 2008.

[7] Andy Vaught. G95 Manual. http://www.g95.org/, 2006.