PDHSEQR User's Guide

By

Robert Granat, Bo Kågström, Daniel Kressner, and Meiyue Shao

UMINF-14/24

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
Sweden

PDHSEQR User's Guide

Robert Granat[*], Bo Kågström[*], Daniel Kressner[†], and Meiyue Shao[*,†]

November 29, 2014

1 Introduction

PDHSEQR is a parallel ScaLAPACK-style library for solving nonsymmetric standard eigenvalue problems. The library is written in Fortran 90 and targets distributed memory HPC systems. Using the small-bulge multishift QR algorithm with aggressive early deflation, it computes the real Schur decomposition $H = ZTZ^T$ of an upper Hessenberg matrix $H \in \mathbb{R}^{n\times n}$, such that Z is orthogonal and T is quasi-upper triangular. This document concerns the usage of PDHSEQR and is a supplement to the article [11]. For the description of the algorithm and implementation, we refer to [11] and the references therein (especially, [7, 8, 9, 10, 12]).

2 Installation

In the following, an installation guide is provided. It is assumed that the user is working in a Unix-like system.

2.1 Prerequisites

To build the library, the following software is required.

- A Fortran 90/95 compiler.
- The MPI library, e.g., OpenMPI or MPICH.
- An optimized BLAS library, e.g., ATLAS or OpenBLAS.
- The LAPACK library.
- The ScaLAPACK library (including BLACS and PBLAS).

[*]Department of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden. Email: `{granat,bokg,myshao}@cs.umu.se`.

[†]MATHICSE ANCHP, EPF Lausanne, CH-1015 Lausanne, Switzerland. Email: `daniel.kressner@epfl.ch`.

2.2 How to compile the library

Download location.
The software version published by ACM TOMS can be downloaded from CALGO [2]. The latest version of the source code (with bug fixes) as well as the associated documents are available on the PDHSEQR homepage [3].

Files in the tar-ball.
The following command unpacks the tar-ball and creates a directory `pdhseqr/`, which is the root directory of the library.

```
tar xzfv pdhseqr.tar.gz
```

Inside the root directory, there are several files and directories:

```
EXAMPLES/ MAKE_INC/ Makefile make.inc README SRC/ TESTING/ TOOLS/ TUNING/ ug.pdf
```

Below is an overview of these items.

- `EXAMPLES/` This directory contains two simple drivers.
- `MAKE_INC/` This directory contains several templates of `make.inc` for GNU, Intel, and PathScale compilers.
- `Makefile` The Makefile for building the library. This file does *not* need to be modified.
- `make.inc` *This is the only file which requires modifications when building the library.* It contains compiler settings and external libraries for the Makefile. The user is required to modify this file according to the target computational environment before compiling the library. Several templates of this file are provided in the directory `MAKE_INC`.
- `README` A shorter version of this document containing a quick installation guide.
- `SRC/` This directory contains source code for all computational routines of the library.
- `TESTING/` This directory contains testing examples.
- `TOOLS/` This directory contains several auxiliary routines (e.g., random number/matrix generators, input/output routines).
- `TUNING/` This directory contains auto-tuning scripts.
- `ug.pdf` The User's Guide of PDHSEQR (i.e., this document).

Build the library.
Once `make.inc` is properly modified according to the computational environment, the library can be built by

```
make all
```

in the root directory of PDHSEQR. This generates the library archive `libpdhseqr.a` in the root directory, two examples in `EXAMPLES/`, and test programs in `TESTING/`. The script `quicktest.sh`

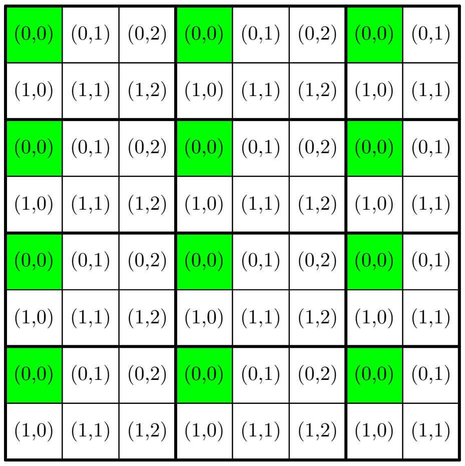

Figure 1: The 2D block-cyclic data layout across a 2×3 processor grid. For example, processor $(0, 0)$ owns all highlighted blocks. Picture from [11].

in `TESTING`, which performs twelve quick tests, needs to be run after the compilation. Hopefully the following result will be displayed on the screen:

```
% 7 out of 7 tests passed!
% 5 out of 5 tests passed!
```

This means that the Schur decomposition has been successfully computed for seven random matrices and five benchmark matrices, indicating that the compilation has been successful. We recommend that the script `runquick.sh` in `TESTING` is also run once to make sure that the parallel code works properly. You may need to modify the MPI execution command in this script according to your system (e.g., `mpirun`, `mpiexec`, etc.). If everything works out, 20 lines of information summarizing the 120 tests will be displayed and written to the file `summary.txt`. We also provide `runall.sh` with many large test cases in the same directory. (Running this set of tests may take *very long*!)

3 Using the package

3.1 ScaLAPACK data layout convention

In ScaLAPACK, the $p = p_r p_c$ processors are usually arranged into a $p_r \times p_c$ grid. Matrices are distributed across the rectangular processor grid in a *2D block-cyclic layout* with block size $m_b \times n_b$ (see Figure 1 for an example). The information regarding the data layout is stored in an *array descriptor* so that the mapping between the entries of the global matrix and their corresponding locations in the memory hierarchy can be established. We adopt ScaLAPACK's data layout convention and require that the $n \times n$ input matrices H and Z have identical data layout with square data blocks (i.e., $m_b = n_b$). The processor grid, however, does not need to be square. A distributed matrix H is referenced by two arrays `H` (local matrix entries) and `DESCH` (array descriptor). A typical setting of `DESCH` is listed below.

- `DESCH(1)`: Type of the matrix. In our case, `DESCH(1) = 1` since H is stored as a dense matrix.

- `DESCH(2)`: The handle of the BLACS context.
- `DESCH(3)`, `DESCH(4)`: The size of H, i.e., `DESCH(5) = DESCH(6) =` n.
- `DESCH(5)`, `DESCH(6)`: Blocking factors m_b and n_b. We require that `DESCH(5) = DESCH(6)`.
- `DESCH(7)`, `DESCH(8)`: The process row and column that contain h_{11}. Usually, `DESCH(7) = DESCH(8) = 0`.
- `DESCH(9)`: Leading dimension of the local part of H on the current processor. This value needs to be at least one, even if the local part is empty.

3.2 Calling sequence

The main functionality of this package is to compute the real Schur decomposition of an upper Hessenberg matrix using the routine `PDHSEQR`. The ScaLAPACK routine `PDGEHRD` can be used to transform a general square matrix to Hessenberg form, see the test programs in `TESTING/` for examples. The interface of `PDHSEQR` displayed below follows the convention of LAPACK/ScaLAPACK routines [4, 6].

```
      SUBROUTINE PDHSEQR( JOB, COMPZ, N, ILO, IHI, H, DESCH, WR, WI, Z,
     $                    DESCZ, WORK, LWORK, IWORK, LIWORK, INFO )
*
*     .. Scalar Arguments ..
      INTEGER            IHI, ILO, INFO, LWORK, LIWORK, N
      CHARACTER          COMPZ, JOB
*     ..
*     .. Array Arguments ..
      INTEGER            DESCH( * ) , DESCZ( * ), IWORK( * )
      DOUBLE PRECISION   H( * ), WI( N ), WORK( * ), WR( N ), Z( * )
```

For comparison, the (nearly identical) interface of the LAPACK routine `DHSEQR`.

```
      SUBROUTINE DHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH, WR, WI, Z,
     $                   LDZ, WORK, LWORK, INFO )
```

Also, the interface of the ScaLAPACK auxiliary routine `PDLAHQR` is similar.

```
      SUBROUTINE PDLAHQR( WANTT, WANTZ, N, ILO, IHI, A, DESCA, WR, WI,
     $                    ILOZ, IHIZ, Z, DESCZ, WORK, LWORK, IWORK,
     $                    ILWORK, INFO )
```

Therefore, it may not require much effort to switch existing code calling `PDLAHQR` to `PDHSEQR`.

An example for calling `PDHSEQR` is provided in the test program (`TESTING/driver.f`). We advice that `PDHSEQR` is called twice—the first call for performing a workspace query (by setting `LWORK = -1`) and the second call for actually performing the computation.

Below is a detailed list of the arguments.

- JOB: (global input) CHARACTER*1.
  JOB = 'E': Compute eigenvalues only;
  JOB = 'S': Compute eigenvalues and the Schur form T.

- COMPZ (global input) CHARACTER*1.
  COMPZ = 'N': Schur vectors (i.e., Z) are not computed;
  COMPZ = 'I': Z is initialized to the identity matrix and both H and Z are returned;
  COMPZ = 'V': Z must contain an orthogonal matrix Q on entry, and the product QZ is returned.

- N: (global input) INTEGER.
  The order of the Hessenberg matrix H (and Z).

- ILO, IHI: (global input) INTEGER.
  It is assumed that H is already upper triangular in rows and columns (1:ILO-1) and (IHI+1:N). They are normally set by a previous call to PDGEBAL, and then passed to PDGEHRD when the matrix output by PDGEBAL is reduced to Hessenberg form. Otherwise ILO = 1 and IHI = N should be used, which guarantees that Z is orthogonal on exit of PDHSEQR.

- H: (global input/output) DOUBLE PRECISION array of dimension (DESCH(9),*).
  DESCH: (global and local input) INTEGER array descriptor of dimension 9.
  H and DESCH define the distributed matrix H.
  On entry, H contains the upper Hessenberg matrix H.
  On exit, if JOB = 'S', H is quasi-upper triangular in rows and columns (ILO:IHI), with 1×1 and 2×2 blocks on the main diagonal. The 2×2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues) are returned in standard form, with $h_{ii} = h_{i+1,i+1}$ and $h_{i+1,i}h_{i,i+1} < 0$. If INFO = 0 and JOB = 'E', the contents of H are unspecified on exit.

- WR, WI: (global output) DOUBLE PRECISION array of dimension N.
  The eigenvalues of H(ILO:IHI,IHO:IHI) are stored in WR(ILO:IHI) and WI(ILO:IHI) where WR contains the real parts and WI contains the imaginary parts, respectively.
  If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)-th, with WI(i) > 0 and WI(i+1) < 0.
  If JOB = 'S', the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H.

- Z: (global input/output) DOUBLE PRECISION array of dimension (DESCZ(9),*).
  DESCZ: (global and local input) INTEGER array descriptor of dimension 9.
  Z and DESCZ define the distributed matrix Z.
  If COMPZ = 'V', on entry Z must contain the current matrix Z of accumulated transformations from, e.g., PDGEHRD, and on exit Z has been updated.
  If COMPZ = 'N', Z is not referenced.
  If COMPZ = 'I', on entry Z does not need be set and on exit, if INFO = 0, Z contains the orthogonal matrix Z of the Schur vectors of H.

- WORK: (local workspace) DOUBLE PRECISION array of dimension LWORK.
  LWORK: (local input) INTEGER.
  In case LWORK = -1, a workspace query will be performed and on exit, WORK(1) is set to the required length of the double precision workspace. No computation is performed in this case.

- `IWORK`: (local workspace) `INTEGER` array of dimension `LIWORK`.
  `LIWORK`: (local input) `INTEGER`.
  In case `LIWORK = -1`, a workspace query will be performed and on exit, `IWORK(1)` is set to the required length of the integer workspace. No computation is performed in this case.

- `INFO`: (global output) `INTEGER`.
  If `INFO = 0`, `PDHSEQR` returns successfully.
  If `INFO < 0`, let $i = -\texttt{INFO}$, then the i-th argument had an illegal value.
  (See below for exceptions with $i = 7777$ or $i = 8888$.)
  If `INFO > 0`, then `PDHSEQR` failed to compute all of the eigenvalues. (This is a rare case.) Elements `(1:ILO-1)` and `(INFO+1:N)` of `WR` and `WI` contain the eigenvalues which have been successfully computed. Let U be the orthogonal matrix logically produced in the computation (regardless of `COMPZ`, i.e., no matter whether it is explicitly formulated or not). Then on exit,

$$\begin{cases} H_{in}U = U^T H_{out},\ Z = U, & \text{if } \texttt{INFO > 0},\ \texttt{COMPZ = 'I'}, \\ H_{in}U = U^T H_{out},\ Z_{out} = Z_{in}U, & \text{if } \texttt{INFO > 0},\ \texttt{COMPZ = 'V'}. \end{cases}$$

  If `INFO = 7777` or `INFO = 8888`, please send a bug report to the authors.

3.3 Example programs

We provide two simple examples in the directory `EXAMPLES/`. The program `example1.f` generates a 500×500 random matrix and computes its Schur decomposition, while `example2.f` reads the benchmark matrix OLM500[1] in the Matrix Market format [5].

To compute eigenvalues of other matrices, the following lines of the example program need to be adjusted:

- Line 40: Matrix size and the block factor.
- Lines 50–51: Make sure to provide sufficient memory.
- Line 222: Replace the matrix generator `PDMATGEN2/PQRRMMM` by your own matrix.

4 Tuning of parameters*

The instructions below are intended for experienced users. Other users may want to skip reading this section.

In `SRC/pilaenvx.f` and `SRC/piparmq.f`, there are several machine-dependent parameters, see [11] for details. On contemporary architectures, we expect that most of the default values provided in the source code yield reasonable performance. However, for `PILAENVX(ISPEC=12, 14, 23)` some fine tuning might be helpful. The package offers two scripts in the directory `TUNING/` that aim at tuning these three parameters.

Before starting the tuning procedure, you first need to choose a frequently used block factor (n_b) and modify the corresponding value in `tune1.in`, `tune2_1.in`, `tune2_2.in`, and `tune2_3.in`. You also need to adjust the MPI execution command according to your system in the scripts `tune1.sh` and `tune2.sh`.

[1]Downloaded from `ftp://math.nist.gov/pub/MatrixMarket2/NEP/olmstead/olm500.mtx.gz`.

- The first script `tune1.sh` searches suitable settings for `PILAENVX(ISPEC=12, 23)`. It performs the tests described in `tune1.in` on 1×1, 2×2, 4×4, 8×8 processor grids, and analyzes the collected data by the code `tune1.f`. This procedure usually takes 1–4 hours. When completed, it reports suggestions on the parameters in the file `suggestion1.txt`. You should then modify the constants `NMIN` (Line 193 in `SRC/piparmq.f`) and `NTHRESH` (Line 648 in `SRC/pilaenvx.f`) in accordance with these suggestions.

- The second script `tune2.sh` searches suitable settings for `PILAENVX(ISPEC=14)`. This set of tests should only be done after running `tune1.sh` and modifying the parameters in `SRC/pilaenvx.f`, `SRC/piparmq.f`, and `TUNING/piparmq.f` correspondingly. Then the library should be compiled again with the new settings:

  ```
  make clean; make all; make tuning
  ```

  Finally, the script `tune2.sh` is executed. This set of tests takes a long time (up to 1–2 days). If all tests are completed successfully, `tune2.f` computes and reports the suggested settings for the parameters in `suggestions2.txt`. You should then update `SRC/piparmq.f` (Line 197) and rebuild the library. This completes the tuning procedure.

  If some of the tests are interrupted, due to an error, it may not be necessary to rerun the whole set of tests. You can also manually collect the execution times for each test into `summary2.txt` and apply `tune2.f` to determine the parameter suggestions.

It is possible to run both tuning scripts on other processor grids (besides the default 2×2, 4×4, 8×8). This, however, requires to not only adjust the scripts `tune*.sh` but also the programs `tune*.f` correspondingly.

5 Terms of Usage

Use of the ACM Algorithm is subject to the ACM Software Copyright and License Agreement [1]. Furthermore, any use of the PDHSEQR library should be acknowledged by citing the corresponding paper [11]. Depending on the context, the citation of the papers [9, 10, 12] is also encouraged.

6 Summary

The use of the software library PDHSEQR is presented in this document. Currently only the double precision version is provided; a complex version of the software (`PZHSEQR`) is planned for future releases. Comments, suggestions, and bug reports from users are welcome. The latest version of the software and documents are always available from the website of PDHSEQR [3].

Acknowledgements

The authors are grateful to Björn Adlerborn and Lars Karlsson for discussions and comments, and to Åke Sandgren for support at High Performance Computing Center North (HPC2N). They also thank David Guerrero, Rodney James, Julien Langou, Jack Poulson, Jose Roman, as well as anonymous users from IBM for helpful feedback.

References

[1] ACM software license agreement. See `http://www.acm.org/publications/policies/softwarecrnotice`.

[2] Collected algorithms of the ACM. See `http://calgo.acm.org/`.

[3] PDHSEQR homepage. See `http://www8.cs.umu.se/~myshao/software/pdhseqr/`.

[4] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. J. D. Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK User's Guide, 3rd Edition.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[5] Z. Bai, D. Day, J. W. Demmel, and J. J. Dongarra. A test matrix collection for non-Hermitian eigenvalue problems (release 1.0). Technical Report CS-97-355, Department of Computer Science, University of Tennessee, 1997. Also available online from http://math.nist.gov/MatrixMarket.

[6] 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 User's Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[7] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. Part I: Maintaining well-focused shifts and level 3 performance. *SIAM J. Matrix Anal. Appl.*, 23(4):929–947, 2002.

[8] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. Part II: Aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 23(4):948–973, 2002.

[9] R. Granat, B. Kågström, and D. Kressner. Parallel eigenvalue reordering in real Schur forms. *Concurrency and Computat.: Pract. Exper.*, 21(9):1225–1250, 2009.

[10] R. Granat, B. Kågström, and D. Kressner. A novel parallel QR algorithm for hybrid distributed memory HPC systems. *SIAM J. Sci. Comput.*, 32(4):2345–2378, 2010.

[11] R. Granat, B. Kågström, D. Kressner, and M. Shao. Algorithm xxx: Parallel library software for the multishift QR algorithm with aggressive early deflation. *ACM Trans. Math. Software.* (to appear).

[12] B. Kågström, D. Kressner, and M. Shao. On aggressive early deflation in parallel variants of the QR algorithm. In K. Jónasson, editor, *Applied Parallel and Scientific Computing (PARA 2010)*, volume 7133 of *Lecture Notes in Comput. Sci.*, pages 1–10, Berlin, 2012. Springer-Verlag.