# StarNEig Library

version v0.1-beta.5

Generated by Doxygen 1.8.13

Thu Apr 2 2020 16:01:10

# Contents


**1 Introduction** 1

**2 Installation** 6

**3 Initialization and shutdown** 10

**4 Distributed memory** 11

**5 Standard eigenvalue problem** 15

**6 Generalized eigenvalue problem** 17

**7 Expert functions** 20

**8 Test program** 21

**9 License and authors** 26

**10 Todo List** 27

**11 Module Documentation** 28

11.1 Library configuration . . . . . . . . . . 28

11.1.1 Detailed Description . . . . . . . . . . 28

11.2 Error codes . . . . . . . . . . 29

11.2.1 Detailed Description . . . . . . . . . . 29

11.3 Intra-node execution environment . . . . . . . . . . 30

11.3.1 Detailed Description . . . . . . . . . . 31

11.3.2 Macro Definition Documentation . . . . . . . . . . 31

11.3.3 Function Documentation . . . . . . . . . . 33

11.4 Shared Memory / Standard EVP . . . . . . . . . . 36

11.4.1 Detailed Description . . . . . . . . . . 36

11.4.2 Function Documentation . . . . . . . . . . 37

11.5 Shared Memory / Generalized EVP . . . . . . . . . . 45

11.5.1 Detailed Description . . . . . . . . . . 45

11.5.2 Function Documentation . . . . . 45
11.6 Distributed Memory / Distributed matrices . . . . . 56
11.6.1 Detailed Description . . . . . 57
11.6.2 Data Structure Documentation . . . . . 57
11.6.3 Enumeration Type Documentation . . . . . 57
11.6.4 Function Documentation . . . . . 58
11.7 Distributed Memory / Standard EVP . . . . . 67
11.7.1 Detailed Description . . . . . 67
11.7.2 Function Documentation . . . . . 67
11.8 Distributed Memory / Generalized EVP . . . . . 75
11.8.1 Detailed Description . . . . . 75
11.8.2 Function Documentation . . . . . 76
11.9 Expert configuration structures . . . . . 83
11.9.1 Detailed Description . . . . . 85
11.9.2 Data Structure Documentation . . . . . 85
11.9.3 Enumeration Type Documentation . . . . . 89
11.9.4 Function Documentation . . . . . 93
11.10 ScaLAPACK compatibility / BLACS matrices . . . . . 95
11.10.1 Detailed Description . . . . . 95
11.10.2 Data Structure Documentation . . . . . 95
11.10.3 Function Documentation . . . . . 96
11.11 ScaLAPACK compatibility / BLACS helpers . . . . . 101
11.11.1 Detailed Description . . . . . 101
11.11.2 Function Documentation . . . . . 101

**12 File Documentation** 106
12.1 blacs_helpers.h File Reference . . . 106
12.1.1 Detailed Description . . . 106
12.1.2 LICENSE . . . 107
12.2 blacs_matrix.h File Reference . . . 107
12.2.1 Detailed Description . . . 108
12.2.2 LICENSE . . . 108
12.3 configuration.h File Reference . . . 109
12.3.1 Detailed Description . . . 109
12.3.2 LICENSE . . . 109
12.4 distr_matrix.h File Reference . . . 110
12.4.1 Detailed Description . . . 111
12.4.2 LICENSE . . . 111
12.5 error.h File Reference . . . 112
12.5.1 Detailed Description . . . 112
12.5.2 LICENSE . . . 113
12.6 expert.h File Reference . . . 113
12.6.1 Detailed Description . . . 115
12.6.2 LICENSE . . . 115
12.7 gep_dm.h File Reference . . . 116
12.7.1 Detailed Description . . . 117
12.7.2 LICENSE . . . 117
12.8 gep_sm.h File Reference . . . 117
12.8.1 Detailed Description . . . 118
12.8.2 LICENSE . . . 119
12.9 node.h File Reference . . . 119
12.9.1 Detailed Description . . . 120
12.9.2 LICENSE . . . 121
12.10 sep_dm.h File Reference . . . 121
12.10.1 Detailed Description . . . 122
12.10.2 LICENSE . . . 122
12.11 sep_sm.h File Reference . . . 123
12.11.1 Detailed Description . . . 123
12.11.2 LICENSE . . . 124
12.12 starneig.h File Reference . . . 124
12.12.1 Detailed Description . . . 124
12.12.2 LICENSE . . . 125

**13 Example Documentation** 125

13.1 gep_dm_full_chain.c . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 125

13.2 gep_sm_eigenvectors.c . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 128

13.3 gep_sm_full_chain.c . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 130

13.4 sep_dm_full_chain.c . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 131

13.5 sep_sm_eigenvectors.c . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 133

13.6 sep_sm_full_chain.c . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 134

**Index** 137

# 1 Introduction

StarNEig library aims to provide a complete task-based software stack for solving **dense nonsymmetric** (generalized) eigenvalue problems. The library is built on top of the StarPU runtime system and targets both shared memory and distributed memory machines. Some components of the library support GPUs.

The four main components of the library are:

- **Hessenberg(-triangular) reduction**: A dense matrix (or a dense matrix pair) is reduced to upper Hessenberg (or Hessenberg-triangular) form.
- **Schur reduction**: A upper Hessenberg matrix (or a Hessenberg-triangular matrix pair) is reduced to (generalized) Schur form. The (generalized) eigenvalues can be determined from the diagonal blocks.
- **Eigenvalue reordering**: Reorders a user-selected set of (generalized) eigenvalues to the upper left corner of an updated (generalized) Schur form.
- **Eigenvectors**: Computes (generalized) eigenvectors for a user-selected set of (generalized) eigenvalues.

**A brief summary of the StarNEig library** can be found from a recent poster: *Task-based, GPU-accelerated and Robust Algorithms for Solving Dense Nonsymmetric Eigenvalue Problems*, Swedish eScience Academy, Lund, Sweden, October 15-16, 2019 (download)

The library has been developed as a part of the NLAFET project. The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671633. Support has also been received from eSSENCE, a collaborative e-Science programme funded by the Swedish Government via the Swedish Research Council (VR).



Please cite the following article when refering to StarNEig:

> Mirko Myllykoski, Carl Christian Kjelgaard Mikkelsen: *Introduction to StarNEig — A Task-based Library for Solving Nonsymmetric Eigenvalue Problems*, In Parallel Processing and Applied Mathematics, 13th International Conference, PPAM 2019, Bialystok, Poland, September 8–11, 2019, Revised Selected Papers, Part I, Lecture Notes in Computer Science, Vol. 12043, Wyrzykowski R., Deelman E., Dongarra J., Karczewski K. (eds), Springer International Publishing, pp. 70-81, 2020, doi: 10.1007/978-3-030-43229-4_7

### Current status

The library is currently in a beta state and only real arithmetic is supported. In addition, some interface functions are implemented as LAPACK and ScaLAPACK wrappers.

Current status:

| Component | Shared memory | Distributed memory | Accelerators (GPUs) |
|---|---|---|---|
| Hessenberg reduction | **Complete** | ScaLAPACK wrapper | **Single GPU** |
| Schur reduction | **Complete** | **Complete** | *Experimental* |
| Eigenvalue reordering | **Complete** | **Complete** | *Experimental* |
| Eigenvectors | **Complete** | Waiting integration | Not planned |
| Hessenberg-triangular reduction | LAPACK wrapper / Planned | ScaLAPACK wrapper | Not planned |
| Generalized Schur reduction | **Complete** | **Complete** | *Experimental* |
| Generalized eigenvalue re-ordering | **Complete** | **Complete** | *Experimental* |
| Generalized eigenvectors | **Complete** | Waiting integration | Not planned |

### Known problems

- Some older OpenMPI versions (pre summer 2017, e.g. <= 2.1.1) have a bug that might lead to a segmentation fault during a parallel AED.
- OpenBLAS version 0.3.1 has a bug that might lead to an incorrect result.
- OpenBLAS versions 0.3.3-0.3.5 might lead to poor scalability.
- Some MKL versions might lead to poor scalability. The problem appears to be related to Intel's OpenMP library. Setting the `KMP_AFFINITY` environmental variable to `disabled` fixes the problem in all known cases.
- StarPU versions 1.2.4 - 1.2.8 and some StarPU 1.3 snapshots cause poor CUDA performance. The problem can be fixed by compiling StarPU with `--disable-cuda-memcpy-peer`. It is possible that newer versions of StarPU are also effected by this problem.
- The `STARPU_MINIMUM_AVAILABLE_MEM` and `STARPU_TARGET_AVAILABLE_MEM` environmental variables can be used to fix some GPU related memory allocation problems:

```
STARPU_MINIMUM_AVAILABLE_MEM=10 STARPU_TARGET_AVAILABLE_MEM=15 ...
```

- The library has an unsolved memory leak problem with OpenMPI. Only large problem sizes are effected. It is not known whether this problem is related to StarNEig, StarPU, OpenMPI or something else. A memory leak is sometimes accompanied by the following warning:

```
mpool.c:38   UCX  WARN  object 0x2652000 was not returned to mpool ucp_requests
```

  The problem is known to occur with PMIx 2.2.1, UCX 1.5.0, OpenMPI 3.1.3, and StarPU 1.2.8.
- If the GPU support is enabled, then the starneig_SEP_SM_Hessenberg() interface function cannot always handle problems that do not fit into GPU's memory. The cause of this problem is is not known.
- The outputs of the starneig_GEP_SM_Schur() and starneig_GEP_DM_Schur() interface functions are not in the so-called standard format. It is possible that some diagonal entries in the right-hand side output matrix are negative. This will be fixed in the next version of the library.
- The starneig_GEP_SM_Eigenvectors() interface function may scale the input matrices. This will be fixed in the next version of the library.

### Related publications

#### Research papers

- Mirko Myllykoski, Carl Christian Kjelgaard Mikkelsen: *Task-based, GPU-accelerated and Robust Library for Solving Dense Nonsymmetric Eigenvalue Problems*, Invited article submitted to Concurrency and Computation: Practice and Experience, arXiv:2002.05024
- Mirko Myllykoski, Carl Christian Kjelgaard Mikkelsen: *Introduction to StarNEig — A Task-based Library for Solving Nonsymmetric Eigenvalue Problems*, In Parallel Processing and Applied Mathematics, 13th International Conference, PPAM 2019, Bialystok, Poland, September 8–11, 2019, Revised Selected Papers, Part I, Lecture Notes in Computer Science, Vol. 12043, Wyrzykowski R., Deelman E., Dongarra J., Karczewski K. (eds), Springer International Publishing, pp. 70-81, 2020, doi: 10.1007/978-3-030-43229-4_7
- Carl Christian Kjelgaard Mikkelsen, Mirko Myllykoski: *Parallel Robust Computation of Generalized Eigenvectors of Matrix Pencils*, presented at PPAM 2019, In Parallel Processing and Applied Mathematics, 13th International Conference, PPAM 2019, Bialystok, Poland, September 8–11, 2019, Revised Selected Papers, Part I, Lecture Notes in Computer Science, Vol. 12043, Wyrzykowski R., Deelman E., Dongarra J., Karczewski K. (eds), Springer International Publishing, pp. 58-69, 2020, doi: 10.1007/978-3-030-43229-4_6
- Carl Christian Kjelgaard Mikkelsen, Angelika Schwarz, and Lars Karlsson: *Parallel Robust Solution of Triangular Linear Systems*, Concurrency and Computation: Practice and Experience, 31 (19), 2019, doi: 10.1016/j.parco.2019.04.001
- Mirko Myllykoski: *A Task-Based Algorithm for Reordering the Eigenvalues of a Matrix in Real Schur Form*, In Parallel Processing and Applied Mathematics, 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017, Revised Selected Papers, Part I, Lecture Notes in Computer Science, Vol. 10777, Wyrzykowski R., Dongarra J., Deelman E., Karczewski K. (eds), Springer International Publishing, pp. 207-216, 2018, doi: 10.1007/978-3-319-78024-5_19
- Carl Christian Kjelgaard Mikkelsen, Lars Karlsson. *Blocked Algorithms for Robust Solution of Triangular Linear Systems*, In Parallel Processing and Applied Mathematics, 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017, Revised Selected Papers, Part I, Lecture Notes in Computer Science, Vol. 10777, Wyrzykowski R., Dongarra J., Deelman E., Karczewski K. (eds), Springer International Publishing, pp. 207-216, 2018, doi: 10.1007/978-3-319-78024-5_7

#### Reports, deliverables etc

- Angelika Schwarz, Carl Christian Kjelgaard Mikkelsen, Lars Karlsson: *Robust Parallel Eigenvector Computation For the Non-Symmetric Eigenvalue Problem*, Report UMINF 20.02, Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden, 2020 (download)
- Angelika Schwarz: *Towards efficient overflow-free solvers for systems of triangular type*, Licentiate thesis, Department of computing science, Umeå University, ISSN: 0348-0542, 2019
- Mirko Myllykoski, Carl Christian Kjelgaard Mikkelsen, Angelika Schwarz, Bo Kågström: *D2.7 Eigenvalue solvers for nonsymmetric problems*, public NLAFET deliverable, 2019 (download)
- Lars Karlsson, Mahmoud Eljammaly, Mirko Myllykoski: *D6.5 Evaluation of auto-tuning techniques*, public NLAFET deliverable, 2019 (download)
- Bo Kågström et al.: *D7.8 Release of the NLAFET library*, public NLAFET deliverable, 2019 (download)
- Mirko Myllykoski, Lars Karlsson, Bo Kågström, Mahmoud Eljammaly, Srikara Pranesh, Mawussi Zounon: *D2.6 Prototype Software for Eigenvalue Problem Solvers*, public NLAFET deliverable, 2018 (download)
- Mirko Myllykoski, Carl Christian Kjelgaard Mikkelsen, Lars Karlsson, Bo Kågström: *Task-Based Parallel Algorithms for Reordering of Matrices in Real Schur Forms*, NLAFET Working Note WN-11, 2017. Also as Report UMINF 17.11, Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden (download)
- Carl Christian Kjelgaard Mikkelsen, Mirko Myllykoski, Björn Adlerborn, Lars Karlsson, Bo Kågström: *D2.5 Eigenvalue Problem Solvers*, public NLAFET deliverable, 2017 (download)

#### The standard eigenvalue problem

Given a square matrix $A$ of size $n \times n$, the *standard eigenvalue problem* (SEP) consists of finding *eigenvalues* $\lambda_i \in \mathbb{C}$ and associated *eigenvectors* $0 \neq v_i \in \mathbb{C}^n$ such that

$$Av_i = \lambda_i v_i, \text{ for } i = 1, 2, \dots, n.$$

The eigenvalues are the $n$ (potentially complex) roots of the polynomial det$(A - \lambda I) = 0$ of degree $n$. There is often a full set of $n$ linearly independent eigenvectors, but if there are *multiple* eigenvalues (i.e., if $\lambda_i = \lambda_j$ for some $i \neq j$) then there might not be a full set of independent eigenvectors.

#### Reduction to Hessenberg form

The dense matrix $A$ is condensed to *Hessenberg form* by computing a *Hessenberg decomposition*

$$A = Q_1 H Q_1^H,$$

where $Q_1$ is unitary and $H$ is upper Hessenberg. This is done in order to greatly accelerate the subsequent computation of a Schur decomposition since when working on $H$ of size $n \times n$, the amount of work in each iteration of the QR algorithm is reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ flops.

#### Reduction to Schur form

Starting from the Hessenberg matrix $H$ we compute a *Schur decomposition*

$$H = Q_2 S Q_2^H,$$

where $Q_2$ is unitary and $S$ is upper triangular. The eigenvalues of $A$ can now be determined as they appear on the diagonal of $S$, i.e., $\lambda_i = s_{ii}$. For real matrices there is a similar decomposition known as the *real Schur decomposition*

$$H = Q_2 S Q_2^T,$$

where $Q_2$ is orthogonal and $S$ is upper quasi-triangular with $1 \times 1$ and $2 \times 2$ blocks on the diagonal. The $1 \times 1$ blocks correspond to the real eigenvalues and each $2 \times 2$ block corresponds to a pair of complex conjugate eigenvalues.

#### Eigenvalue reordering and invariant subspaces

Given a subset consisting of $m \leq n$ of the eigenvalues, we can *reorder the eigenvalues* on the diagonal of the Schur form by constructing a unitary matrix $Q_3$ such that

$$S = Q_3 \begin{bmatrix} \hat{S}_{11} & \hat{S}_{12} \\ 0 & \hat{S}_{22} \end{bmatrix} Q_3^H$$

and the eigenvalues of the $m \times m$ block $\hat{S}_{11}$ are the selected eigenvalues. The first $m$ columns of $Q_3$ span an *invariant subspace* associated with the selected eigenvalues.

#### Computation of eigenvectors

Given a subset consisting of $m \leq n$ of the eigenvalues $\lambda_i$ for $i = 1, 2, \dots, m$ and a Schur decomposition $A = QSQ^H$, we can compute for each $\lambda_i$ an *eigenvector* $v_i \neq 0$ such that $Av_i = \lambda_i v_i$ by first computing an eigenvector $w_i$ of $S$ and then transform it back to the original basis by pre-multiplication with $Q$.

#### The generalized eigenvalue problem

Given a square matrix pencil $A - \lambda B$, where $A, B \in \mathbb{C}^{n \times n}$, the *generalized eigenvalue problem* (GEP) consists of finding *generalized eigenvalues* $\lambda_i \in \mathbb{C}$ and associated *generalized eigenvectors* $0 \neq v_i \in \mathbb{C}^n$ such that

$$Av_i = \lambda_i B v_i, \text{ for } i = 1, 2, \ldots, n.$$

The eigenvalues are the $n$ (potentially complex) roots of the polynomial $\det(A - \lambda B) = 0$ of degree $n$. There is often a full set of $n$ linearly independent generalized eigenvectors, but if there are *multiple eigenvalues* (i.e., if $\lambda_i = \lambda_j$ for some $i \neq j$) then there might not be a full set of independent eigenvectors.

At least in principle, a GEP can be transformed into a SEP provided that $B$ is invertible, since

$$Av = \lambda Bv \Leftrightarrow (B^{-1}A)v = \lambda v.$$

However, in finite precision arithmetic this practice is not recommended.

#### Reduction to Hessenberg-triangular form

The dense matrix pair $(A, B)$ is condensed to *Hessenberg-triangular form* by computing a *Hessenberg-triangular decomposition*

$$A = Q_1 H Z_1^H, \quad B = Q_1 Y Z_1^H,$$

where $Q_1, Z_1$ are unitary, $H$ is upper Hessenberg, and $Y$ is upper triangular. This is done in order to greatly accelerate the subsequent computation of a generalized Schur decomposition.

#### Reduction to generalized Schur form

Starting from the Hessenberg-triangular pencil $H - \lambda Y$ we compute a *generalized Schur decomposition*

$$H = Q_2 S Z_2^H, \quad Y = Q_2 T Z_2^H,$$

where $Q_2, Z_2$ are unitary and $S, T$ are upper triangular. The eigenvalues of $A - \lambda B$ can now be determined from the diagonal element pairs $(s_{ii}, t_{ii})$, i.e., $\lambda_i = s_{ii}/t_{ii}$ (if $t_{ii} \neq 0$). If $s_{ii} \neq 0$ and $t_{ii} = 0$, then $\lambda_i = \infty$ is an *infinite eigenvalue* of the matrix pair $(S, T)$. (If both $s_{ii} = 0$ and $t_{ii} = 0$ for some $i$, then the pencil is *singular* and the eigenvalues are undetermined; all complex numbers are eigenvalues). For real matrix pairs there is a similar decomposition known as the *real generalized Schur decomposition*

$$H = Q_2 S Z_2^T, \quad Y = Q_2 T Z_2^T,$$

where $Q_2, Z_2$ are orthogonal, $S$ is upper quasi-triangular with $1 \times 1$ and $2 \times 2$ blocks on the diagonal, and $T$ is upper triangular. The $1 \times 1$ blocks on the diagonal of $S - \lambda T$ correspond to the real generalized eigenvalues and each $2 \times 2$ block corresponds to a pair of complex conjugate generalized eigenvalues.

#### Eigenvalue reordering and deflating subspaces

Given a subset consisting of $m \leq n$ of the generalized eigenvalues, we can *reorder the generalized eigenvalues* on the diagonal of the generalized Schur form by constructing unitary matrices $Q_3$ and $Z_3$ such that

$$S - \lambda T = Q_3 \begin{bmatrix} \hat{S}_{11} - \lambda \hat{T}_{11} & \hat{S}_{12} - \lambda \hat{T}_{12} \\ 0 & \hat{S}_{22} - \lambda \hat{T}_{22} \end{bmatrix} Z_3^H$$

and the eigenvalues of the $m \times m$ block pencil $\hat{S}_{11} - \lambda \hat{T}_{11}$ are the selected generalized eigenvalues. The first $m$ columns of $Z_3$ spans a right *deflating subspace* associated with the selected generalized eigenvalues.

#### Computation of generalized eigenvectors

Given a subset consisting of $m \leq n$ of the eigenvalues $\lambda_i$ for $i = 1, 2, \ldots, m$ and a generalized Schur decomposition $A - \lambda B = Q(S - \lambda T)Z^H$, we can compute for each $\lambda_i$ a *generalized eigenvector* $v_i \neq 0$ such that $Av_i = \lambda_i Bv_i$ by first computing a generalized eigenvector $w_i$ of $S - \lambda_i T$ and then transform it back to the original basis by pre-multiplication with $Z$.

# 2 Installation

#### Documentation

The user manual can be generated independently from the rest of the library.

Documentation dependencies:

- CMake 3.3 or newer
- Doxygen
- Latex + pdflatex

It is recommended that a user builds the documentation in a separate build directory:

```
$ cd path_to_the_root_directory/
$ mkdir build_doc
$ cd build_doc/
$ cmake ../doc/
$ make
```

The PDF documentation is copied to `build_doc/starneig_manual.pdf`. The HTML documentation is available at `build_doc/html` directory.

#### Dependencies

Library dependencies:

- Linux (not tested in Window or Mac OS X)
- CMake 3.3 or newer
- Portable Hardware Locality (hwloc)
- Starpu 1.2 or 1.3 (newer versions require minor changes to `src/CMakeLists.txt`; `SUPPORTED_STARPU`)
- BLAS (preferably a multi-threaded variant that has an option to change the thread count)
- LAPACK
- MPI (optional)
- CUDA (optional)
- ScaLAPACK (optional)

Test program and example code dependencies:

- pkg-config
- GNU Scientific Library (optional)
- MAGMA (optional)

#### StarPU 1.2.8 installation

1. Download StarPU 1.2.8 (or newer) from http://starpu.gforge.inria.fr/files/
2. Unzip the package and create/enter directory `starpu-1.2.8/build`
3. Configure: `$ ../configure`
4. Compile: `$ make`
5. Install: `$ sudo make install`

The default installation path is `/usr/local` but this can be changed during the configuration phase (`$ ../configure --prefix=...`). It is something necessary to append the `CPATH`, `LIBRARY_PATH`, and `LD_LIBRARY_PATH` environmental variables by adding the following to `~/.profile`:

```
export CPATH=$CPATH:/usr/local/include/
export LIBRARY_PATH=$LIBRARY_PATH:/usr/local/lib/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/
```

See the StarPU handbook for further instructions: http://starpu.gforge.inria.fr/doc/html/BuildingAndInstallingStarPU.html

#### Configuration

It is recommended that a user builds the library in a separate build directory:

```
$ cd path_to_the_root_directory/
$ mkdir build
$ cd build
```

The library is configured with the `cmake` command. In most cases, it is not necessary to give this command any additional arguments:

```
$ cmake ../
...
-- Configuring done
-- Generating done
-- Build files have been written to: /.../build
```

However, the library can be customized with various options. For example, the example codes and documentation generation can be enabled by setting the `STARNEIG_ENABLE_EXAMPLES` and `STARNEIG_ENABLE_DOCS` options:

```
$ cmake -DSTARNEIG_ENABLE_EXAMPLES=ON -DSTARNEIG_ENABLE_DOCS=ON ../
```

The installation path can be changed during the configuration phase:

```
$ cmake -DCMAKE_INSTALL_PREFIX=/path/to/somewhere/ ../
```

**Remarks**

It may sometimes be necessary to compile CUDA source files with a different compiler than what `cmake` uses by default. For example, some CUDA version do not support GCC compilers that are newer than GCC 5 release series. In that case `cmake` can be configured to use GCC 5:

```
$ cmake -DCUDA_HOST_COMPILER=/usr/bin/gcc-5 -DCUDA_PROPAGATE_HOST_FLAGS=OFF ../
```

List of StarNEig library specific configuration options:

- `STARNEIG_ENABLE_OPTIMIZATION`: Enables compiler optimizations (`ON` by default).
- `STARNEIG_ENABLE_EXAMPLES`: Enables examples (`OFF` by default).
- `STARNEIG_ENABLE_DOCS`: Enables documentation generation (`OFF` by default).
- `STARNEIG_ENABLE_TESTS`: : Enables test program (`ON` by default).
- `STARNEIG_ENABLE_FULL_TESTS`: Enables additional tests (`OFF` by default).
- `STARNEIG_ENABLE_REFERENCE`: : Enables reference MPI implementations (`OFF` by default).
- `STARNEIG_DISABLE_MPI`: Explicitly disables the MPI support even when the system would support it (`OFF` by default).
- `STARNEIG_DISABLE_CUDA`: Explicitly disables the CUDA support even when the system would support it (`OFF` by default).
- `STARNEIG_DISABLE_BLACS`: Explicitly disables the ScaLAPACK/BLACS support even when the system would support it (`OFF` by default).
- `STARNEIG_ENABLE_MESSAGES`: Enable basic verbose messages (`ON` by default).
- `STARNEIG_ENABLE_VERBOSE`: Enable additional verbose messages (`OFF` by default).
- `STARNEIG_ENABLE_EVENTS`: Enable event traces (`OFF` by default).
- `STARNEIG_ENABLE_EVENT_PARSER`: Enable event parser (`OFF` by default).
- `STARNEIG_ENABLE_SANITY_CHECKS`: Enables additional satiny checks. These checks are very expensive and should not be enabled unless absolutely necessary (`OFF` by default).
- `STARNEIG_ENABLE_PRUNING`: Enable task graph pruning (`ON` by default).
- `STARNEIG_ENABLE_MRM`: Enable multiple linear regression performance models (`OFF` by default).
- `STARNEIG_ENABLE_CUDA_REORDER_WINDOW`: Enable CUDA-based reorder_window codelet (`OFF` by default).
- `STARNEIG_ENABLE_INTEGER_SCALING`: Enable integer-based scaling factors (`ON` by default).

The following **environmental variables** can be used to configure the used libraries:

- `BLAS_LIBRARIES`: BLAS library.
- `LAPACK_LIBRARIES`: LAPACK library.
- `HWLOC_LIBRARIES`: Portable Hardware Locality (hwloc) library.
- `MPI_LIBRARIES`: C MPI library.
- `MPI_Fortran_LIBRARIES`: Fortran MPI library.
- `SCALAPACK_LIBRARIES`: ScaLAPACK library.

- BLACS_LIBRARIES: BLACS library.
- STARPU_LIBRARIES_BASE: StarPU library.
- STARPU_LIBRARIES_MPI: StarPU-MPI library.
- GSL_LIBRARIES: GNU Scientific Library.
- MAGMA_LIBRARIES: MAGMA library.
- MISC_LIBRARIES: Miscellaneous libraries.

For example, if a user has a custom build ATLAS BLAS library and a matching LAPACK library that are not detected by the build system, then the user might define BLAS_LIBRARIES=/usr/local/atlas/lib/libsatlas.so and LAPACK_LIBRARIES=/usr/local/atlas/lib/liblapack.so before calling cmake.

The following environmental variables can be used to configure include paths for the used libraries:

- OMP_INCLUDE_PATH: OpenMP include path.
- BLAS_INCLUDE_PATH: BLAS include path.
- MKL_INCLUDE_PATH: MKL include path.
- HWLOC_INCLUDE_PATH: Portable Hardware Locality (hwloc) include path.
- MPI_INCLUDE_PATH: MPI include path.
- STARPU_INCLUDE_PATH: StarPU include path.
- GSL_INCLUDE_PATH: GNU Scientific Library include path.
- MAGMA_INCLUDE_PATH: MAGMA include path.
- MISC_INCLUDE_PATH: Miscellaneous include paths.

#### Compile

The library (and other components) are compiled with the make command:

```
$ make
Scanning dependencies of target starneig
[  1%] Building C object src/CMakeFiles/starneig.dir/common/combined.c.o
[  2%] Building C object src/CMakeFiles/starneig.dir/common/common.c.o
...
```

#### Test

The automated tests can be executed as follows:

```
$ make test
Running tests...
Test project /.../build
      Start  1: simple-hessenberg
 1/18 Test  #1: simple-hessenberg ..................   Passed   15.19 sec
      Start  2: simple-hessenberg-mpi
 2/18 Test  #2: simple-hessenberg-mpi ..............   Passed   49.59 sec
...
      Start 17: simple-full-chain-generalized
17/18 Test #17: simple-full-chain-generalized .......   Passed  180.50 sec
      Start 18: simple-full-chain-generalized-mpi
18/18 Test #18: simple-full-chain-generalized-mpi ...   Passed  195.39 sec

100% tests passed, 0 tests failed out of 18

Total Test time (real) = 1219.47 sec
```

The STARNEIG_ENABLE_FULL_TESTS cmake option can be used to enable additional tests.

#### Install

The library and the related header files are installed by executing:

```
$ sudo make install
```

This also installs `starneig.pc` configuration file.

## 3 Initialization and shutdown

The initialization and shutdown interface functions can be found from the starneig/node.h header file. The library provides separate header files for shared memory (starneig/sep_sm.h, starneig/gep_sm.h) and distributed memory (starneig/sep_dm.h, starneig/gep_dm.h). However, a user may simply include all header files as follows:

```
#include <starneig/starneig.h>
```

Certain header files and interface functions exist only when the library is compiled with MPI and ScaLAPACK / BLACS support. The configuration of the installed library can be found from the starneig/configuration.h header file. See module Library configuration for further information.

Each node must call the starneig_node_init() interface function to initialize the library and the starneig_node_finalize() interface function to shutdown the library:

```
starneig_node_init(cores, gpus, flags);

...

starneig_node_finalize();
```

The starneig_node_init() interface function initializes StarPU (and cuBLAS) and pauses all worker threads. The `cores` argument specifies the total number of used CPU cores. In distributed memory mode, one of these CPU cores is automatically allocated for the StarPU-MPI communication thread. The `gpus` argument specifies the total number of used GPUs. One or more CPU cores are automatically allocated for GPU devices. The `flags` (starneig_flag_t) argument can provide additional configuration information.

A node can also be configured with default values:

```
starneig_node_init(-1, -1, STARNEIG_DEFAULT);
```

This tells the library to use all available CPU cores and GPUs. See module Intra-node execution environment for further information.

Most interface functions return one of the following values:

- STARNEIG_SUCCESS (0): The interface function was executed successfully.
- A negative number `-i`: The `i`'th interface function argument was invalid.
- A positive number `i`: The interface function encountered an error or a warning was raised. See module Error codes for further information.

All return values (starneig_error_t) are defined in the starneig/error.h header file.

#### Remarks

The library may call the `exit()` and `abort()` functions if an interface function encounters a fatal error from which it cannot recover.
The StarPU performance models must be calibrated before the software can function efficiently on heterogeneous platforms (CPUs + GPUs). The calibration is triggered automatically if the models are not calibrated well enough for a given problem size. This may impact the execution time negatively during the first run. Please see the StarPU handbook for further information: `http://starpu.gforge.inria.fr/doc/html/Scheduling.html`

# 4 Distributed memory

The STARNEIG_HINT_DM initialization flag tells the library to configure itself for distributed memory computation. The flag is indented to be only a hint and the library will automatically reconfigure itself for the correct computation mode. A user is allowed to mix shared memory and distributed memory functions without reninitializing the library. The library is intended to be run in a **hybrid configuration** (each MPI rank is mapped to several CPU cores). Failing to do so leads to CPU core oversubscription. It is generally a good idea to map each MPI rank to a full node or a NUMA island / CPU socket:

```
# OpenMPI / one rank per node:
$ mpirun -n RANKS --map-by ppr:1:node --bind-to none ...

# OpenMPI / one rank per socket:
$ mpirun -n RANKS --map-by ppr:1:socket --bind-to socket ...

# Intel MPI / one rank per node:
$ mpirun -n RANKS -binding "pin=on;domain=node" ...

# Intel MPI / one rank per socket:
$ mpirun -n RANKS -binding "pin=on;domain=socket" ...
```

#### Attention

StarPU attempts to bind the worker threads to the available CPU cores. This may sometimes conflict with the MPI library and/or the batch system CPU core allocation. StarNEig library attempts to correct for by factoring in the CPU core binding mask. However, if neither the MPI library nor the batch system enforces such a binding mask, it is possible that several StarPU worker threads end up bound to a same CPU core. In such a situation, it is recommended that a user disables the StarPU thread binding explicitly:

```
STARPU_WORKERS_NOBIND=1 mpirun ...
```

This is particularly important when several ranks / processes are mapped to a same node.

The library assumes that the MPI library is already initialized when the starneig_node_init() interface function is called with the STARNEIG_HINT_DM flag or when the library reconfigures itself for distributed memory after a user has called a distributed memory interface function. The MPI library should be initialized either in the serialized mode:

```
int thread_support;
MPI_Init_thread(
    &argc, (char ***)&argv, MPI_THREAD_SERIALIZED, &thread_support);

if (thread_support < MPI_THREAD_SERIALIZED) {
    fprintf(stderr,
        "MPI_THREAD_SERIALIZED is not supported. Aborting...\n");
    abort();
}
```

Or in the multi-threaded mode:

```
int thread_support;
MPI_Init_thread(
    &argc, (char ***)&argv, MPI_THREAD_MULTIPLE, &thread_support);

if (thread_support < MPI_THREAD_SERIALIZED) {
    fprintf(stderr,
        "MPI_THREAD_SERIALIZED is not supported. Aborting...\n");
    abort();
} else if (thread_support < MPI_THREAD_MULTIPLE) {
    fprintf(stderr,
        "Warning: MPI_THREAD_MULTIPLE is not supported.\n");
}
```

A user is allowed to change the library MPI communicator with the starneig_mpi_set_comm() interface function. This interface function should be called **before** the library is initialized.

#### Data distribution

Distributed matrices are represented using two opaque objects:

- *Data distribution* (starneig_distr_t)
- *Distributed matrix* (starneig_distr_matrix_t)

Each matrix is divided into rectangular blocks of uniform size (excluding the last block row and column):

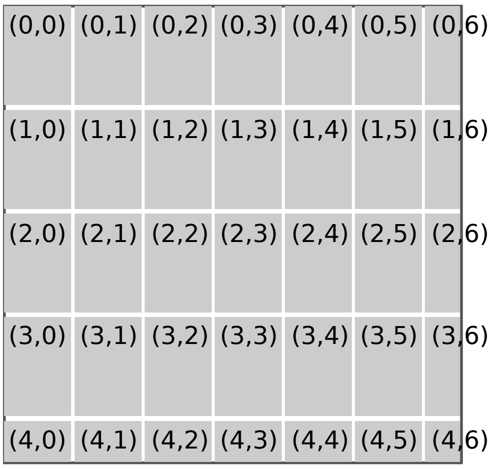


**Figure 1 A matrix divided into rectangular blocks of uniform size.**

The blocks are indexed using a two-dimensional index space. A data distribution encapsulates an arbitrary mapping from this two-dimensional block index space to the one-dimensional MPI rank space:

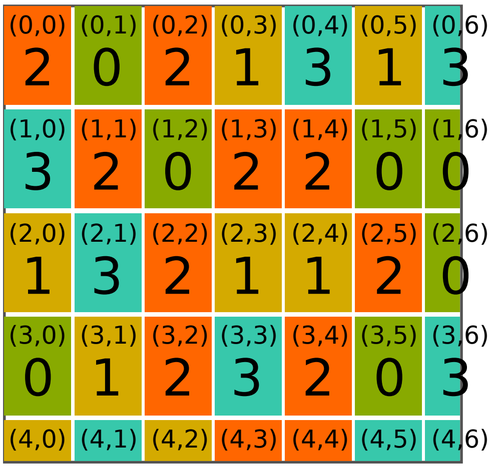


**Figure 2 An example of a block mapping.**

In the above example, the rank 0 owns the blocks (0,1), (1,2), (1,5), (1,6), (2,6), (3,0) and (3,5). Naturally, a data distribution can describe a two-dimensional block cyclic distribution that is very common with ScaLAPACK subroutines:

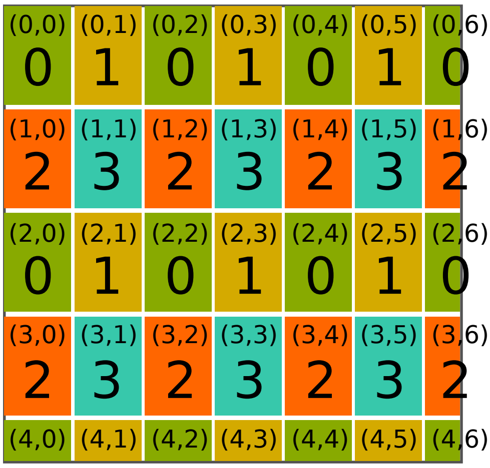


**Figure 3 An example of a row-major ordered two-dimensional block cyclic mapping.**

A data distribution can be created using one of the following interface functions:

- starneig_distr_init() creates a default distribution (row-major ordered two-dimensional block cyclic distribution with a squarish mesh).
- starneig_distr_init_mesh() creates a row-major or column-major ordered two-dimensional block cyclic distribution with desired number of rows and columns in the mesh.
- starneig_distr_init_func() creates an arbitrary distribution defined by a function.

Fox example,

```
starneig_distr_t distr = starneig_distr_init_mesh(4, 6,
      STARNEIG_ORDER_DEFAULT);
```

would create a two-dimensional block cyclic distribution with 4 rows and 6 columns in the mesh. Alternatively, a user can create an equivalent data distribution using the starneig_distr_init_func() interface function:

```
// additional distribution function argument structure
struct block_cyclic_arg {
    int rows;   // the number of rows in the process mesh
    int cols;   // the number of columns in the process mesh
};

// distribution function (row-major ordered 2D block cyclic distribution)
int block_cyclic_func(int i, int j, void *arg)
{
    struct block_cyclic_arg *mesh = (struct block_cyclic_arg *) arg;
    return (i % mesh->rows) * mesh->cols + j % mesh->cols;
}

void func(...)
{
    ...

    struct block_cyclic_arg arg = { .rows = 4, .cols = 6 };
    starneig_distr_t distr =
        starneig_distr_init_func(&block_cyclic_func, &arg, sizeof(arg));

    ...
}
```

A data distribution is destroyed with the starneig_distr_destroy() interface function.

**Remarks**

Certain interface functions (starneig_SEP_DM_Hessenberg(), starneig_SEP_DM_Reduce(), starneig_GEP_DM_HessenbergTriangular(), and starneig_GEP_DM_Reduce()) are wrappers for / use several ScaLAPACK subroutines. The involved matrices should thus have a two-dimensional block cyclic data distribution. The library will automatically convert the matrices to a compatible format but this requires extra memory.

### Distributed matrix

A distributed matrix is created using the starneig_distr_matrix_create() interface function. The function call will automatically allocate the required local resources. For example,

```
starneig_distr_t distr =
    starneig_distr_init_mesh(4, 6,
      STARNEIG_ORDER_DEFAULT);
starneig_distr_matrix_t dA =
    starneig_distr_matrix_create(m, n, bm, bn,
      STARNEIG_REAL_DOUBLE, distr);
```

would create a $m \times n$ double-precision real matrix that is distributed in a two-dimensional block cyclic fashion in $bm \times bn$ blocks. Or,

```
starneig_distr_matrix_t dB =
    starneig_distr_matrix_create(n, n, -1, -1,
      STARNEIG_REAL_DOUBLE, NULL);
```

would create a $n \times n$ double-precision real matrix with a default data distribution (`NULL` argument) and a default block size (`-1, -1`).

**Attention**

StarNEig library is designed to use much larger distributed blocks than ScaLAPACK. Selecting a too small distributed block size will be detrimental to the performance.

A user may access the locally owned blocks using the starneig_distr_matrix_get_blocks() interface function. A distributed matrix is destroyed using the starneig_distr_matrix_destroy() interface function. This will deallocate all local resources. See module Distributed Memory / Distributed matrices for further information.

**Remarks**

Certain interface functions (starneig_SEP_DM_Hessenberg(), starneig_SEP_DM_Reduce(), starneig_GEP↩_DM_HessenbergTriangular(), and starneig_GEP_DM_Reduce()) are wrappers for / use several ScaLAPACK subroutines. The involved matrices should thus be distributed in square blocks. In addition, the ScaLAPACK subroutines usually perform better when the block size is relatively small. The library will automatically convert the matrices to a compatible format but this requires extra memory.

### Copy, scatter and gather

An entire distributed matrix can be copied with the starneig_distr_matrix_copy() interface function:

```
starneig_distr_matrix_t dA, dB;

...

starneig_distr_matrix_copy(dB, dA);
```

This copies distributed matrix `dB` to a distributed matrix `dA`. A region (submatrix) of a distributed matrix can be copied to a second distributed matrix using the starneig_distr_matrix_copy_region() interface function.

A local matrix can be converted to a "single owner" distributed matrix with the starneig_distr_matrix_create_local() interface function:

```
int owner = 0; // MPI rank that owns the local matrix
double *A;     // local pointer
int ldA;       // matching leading dimension

...

starneig_distr_matrix_t lA = starneig_distr_matrix_create_local(
    n, n, STARNEIG_REAL_DOUBLE, owner, A, ldA);
```

This creates a wrapper object, i.e., the pointer `A` and the distributed matrix `lA` point to the same data on the `owner` node. The created distributed matrix is associated with a data distribution that indicated that the whole matrix is owned by the node `owner`. The used block size is $n \times n$.

Copying from a "single owner" distributed matrix to a distributed matrix performs a *scatter* operation and copying from a distributed matrix to a "single owner" distributed matrix performs a *gather* operation.

### ScaLAPACK compatibility layer

The library provides a ScaLAPACK compatibility layer:

- BLACS contexts are encapsulated inside starneig_blacs_context_t objects.
- BLACS descriptors are encapsulated inside starneig_blacs_descr_t objects.

A two-dimensional block cyclic data distribution can be converted to a BLACS context and vice versa using the starneig_distr_to_blacs_context() and starneig_blacs_context_to_distr() interface functions, respectively. Similarly, a distributed matrix that uses a two-dimensional block cyclic data distribution can be converted to a BLACS descriptor (and a local buffer) and vice versa using the starneig_distr_matrix_to_blacs_descr() and starneig_blacs_descr_to_distr_matrix() interface functions, respectively. The conversion is performed in-place and a user is allowed to mix StarNEig interface functions with ScaLAPACK style subroutines/functions without reconversion.

For example,

```
starneig_distr_matrix_t dA = starneig_distr_matrix_create
      (...);

...

// convert the data distribution to a BLACS context
starneig_distr_t distr = starneig_distr_matrix_get_distr(A);
starneig_blacs_context_t context =
      starneig_distr_to_blacs_context(distr);

// convert the distributed matrix to a BLACS descriptor and a local buffer
starneig_blacs_descr_t descr_a;
double *local_a;
starneig_distr_matrix_to_blacs_descr(dA, context, &descr_a, (void **)&
      local_a);

...

// a ScaLAPACK subroutine for reducing general distributed matrix to upper
// Hessenberg form
extern void pdgehrd_(int const *, int const *, int const *, double *,
    int const *, int const *, starneig_blacs_descr_t const *, double *,
    double *, int const *, int *);

pdgehrd_(&n, &ilo, &ihi, local_a, &ia, &ja, &descr_a, tau, ...);
```

converts a distributed matrix `dA` to a BLACS descriptor `descr_a` and a local pointer `local_a`. The descriptor and the local array are then fed to a ScaLAPACK subroutine. A user must make sure that the live time of the distributed matrix `dA` is at least as long as the live time of the matching BLACS descriptor `descr_a`. See modules ScaLAPACK compatibility / BLACS helpers and ScaLAPACK compatibility / BLACS matrices for further information.

# 5 Standard eigenvalue problem

The library provides 12 interface functions for the standard case:

### Hessenberg reduction

Given a general matrix $A$, the starneig_SEP_SM_Hessenberg() and starneig_SEP_DM_Hessenberg() interface functions compute a Hessenberg decomposition

$$A = U * H * U^T,$$

where $H$ is upper Hessenberg and $U$ is orthogonal. On exit, $A$ is overwritten by $H$ and $Q$ (which is an orthogonal matrix on entry) is overwritten by

$$Q \leftarrow Q * U.$$

#### Schur reduction

Given a Hessenberg decomposition

$$A = Q * H * Q^T,$$

of a general matrix $A$, the starneig_SEP_SM_Schur() and starneig_SEP_DM_Schur() interface functions compute a Schur decomposition

$$A = Q * (U * S * U^T) * Q^T$$

where $S$ is upper quasi-triangular with $1 \times 1$ and $2 \times 2$ blocks on the diagonal (Schur matrix) and $U$ is orthogonal. On exit, $H$ is overwritten by $S$ and $Q$ is overwritten by

$$Q \leftarrow Q * U.$$

#### Eigenvalue reordering

Given a Schur decomposition

$$A = Q * S * Q^T$$

of a general matrix $A$ and a selection of eigenvalues, the starneig_SEP_SM_ReorderSchur() and starneig_SEP_DM_ReorderSchur() interface functions attempt to reorder the selected eigenvalues to the top left corner of an updated Schur matrix $\hat{S}$ by an orthogonal similarity transformation

$$A = Q * (U * \hat{S} * U^T) * Q^T.$$

On exit, $S$ is overwritten by $\hat{S}$ and $Q$ is overwritten by

$$Q \leftarrow Q * U.$$

Reordering may in rare cases fail. In such cases the output is guaranteed to be a Schur decomposition and all (if any) selected eigenvalues that are correctly placed are marked in the selection array on exit. Reordering may perturb the eigenvalues and the eigenvalues after reordering are returned.

#### Combined reduction to Schur form and eigenvalue reordering

Given a general matrix $A$, the starneig_SEP_SM_Reduce() and starneig_SEP_DM_Reduce() interface functions compute a (reordered) Schur decomposition

$$A = U * S * U^T,$$

where $S$ is upper quasi-triangular with $1 \times 1$ and $2 \times 2$ blocks on the diagonal (Schur matrix) and $U$ is orthogonal. Optionally, the interface functions attempt to reorder selected eigenvalues to the top left corner of the Schur matrix $S$.

On exit, $A$ is overwritten by $S$ and $Q$ (which is an orthogonal matrix on entry) is overwritten by

$$Q \leftarrow Q * U.$$

Reordering may in rare cases fail. In such cases the output is guaranteed to be a Schur decomposition and all (if any) selected eigenvalues that are correctly placed are marked in the selection array on exit. Reordering may perturb the eigenvalues and the eigenvalues after reordering are returned.

#### Eigenvectors

Given a Schur decomposition

$$A = Q * S * Q^T$$

of a general matrix $A$ and a selection of eigenvalues, the starneig_SEP_SM_Eigenvectors() and starneig_SEP_DM_Eigenvectors() interface functions compute and return an eigenvector for each of the selected eigenvalues.

The eigenvectors are stored as columns in the output matrix $X$ in the same order as their corresponding eigenvalues appear in the selection array. A real eigenvector is stored as a single column. The real and imaginary parts of a complex eigenvector are stored as consecutive columns.

For a selected pair of complex conjugate eigenvalues, an eigenvector is computed only for the eigenvalue with positive imaginary part. Thus, every selected eigenvalue contributes one column to the output matrix and thus the number of selected eigenvalues is equal to the number of columns of $X$.

#### Eigenvalue selection helper

Given a Schur matrix and a predicate function, the starneig_SEP_SM_Select() and starneig_SEP_DM_Select() interface functions conveniently generate a correct selection array and count the number of selected eigenvalues. The count is useful when allocating storage for the eigenvector matrix computed by the starneig_SEP_SM_Eigenvectors() and starneig_SEP_DM_Eigenvectors() interface functions.

```
// a predicate function that selects all eigenvalues that have a real
// part that is larger than a given value
static int predicate(double real, double imag, void *arg)
{
    double value = * (double *) arg;

    if (value < real)
        return 1;
    return 0;
}

void func(...)
{
    double *S; int ldS;

    ...

    double value = 0.5;
    int num_selected, *selected = malloc(n*sizeof(int));
    starneig_SEP_SM_Select(
        n, S, ldS, &predicate, &value, selected, &num_selected);

    ...
}
```

See modules Shared Memory / Standard EVP and Distributed Memory / Standard EVP for further information. See also examples sep_sm_full_chain.c, sep_dm_full_chain.c and sep_sm_eigenvectors.c.

# 6 Generalized eigenvalue problem

The library provides 12 interface functions for the generalized case:

#### Hessenberg-triangular reduction

Given a general matrix $(A, B)$, the starneig_GEP_SM_HessenbergTriangular() and starneig_GEP_DM_HessenbergTriangular() interface functions compute a Hessenberg-triangular decomposition

$$(A, B) = U_1 * (H, T) * U_2^T,$$

where $H$ is upper Hessenberg, $T$ is upper triangular, and $U_1$ and $U_2$ are orthogonal. On exit, $A$ is overwritten by $H$, $B$ is overwritten by $T$, and $Q$ and $Z$ (which are orthogonal matrices on entry) are overwritten by

$$Q \leftarrow Q * U_1 \text{ and } Z \leftarrow Z * U_2.$$

#### Generalized Schur reduction

Given a Hessenberg-triangular decomposition

$$(A, B) = Q * (H, T) * Z^T$$

of a general matrix pencil $(A, B)$, the starneig_GEP_SM_Schur() and starneig_GEP_DM_Schur() interface functions function compute a generalized Schur decomposition

$$(A, B) = Q * (U_1 * (S, \hat{T}) * U_2^T) * Z^T,$$

where $S$ is upper quasi-triangular with $1 \times 1$ and $2 \times 2$ blocks on the diagonal, $\hat{T}$ is a upper triangular matrix, and $U_1$ and $U_2$ are orthogonal.

On exit, $H$ is overwritten by $S$, $T$ is overwritten by $\hat{T}$, and $Q$ and $Z$ are overwritten by

$$Q \leftarrow Q * U_1 \text{ and } Z \leftarrow Z * U_2.$$

The computed generalized eigenvalues are returned as a pair of values $(\alpha, \beta)$ such that $\alpha/\beta$ gives the actual generalized eigenvalue. The quantity $\alpha/\beta$ may overflow.

#### Generalized eigenvalue reordering

Given a generalized Schur decomposition

$$(A, B) = Q * (S, T) * Z^T$$

of a general matrix pencil $(A, B)$ and a selection of generalized eigenvalues, the starneig_GEP_SM_ReorderSchur() and starneig_GEP_DM_ReorderSchur() interface functions attempt to reorder the selected generalized eigenvalues to the top left corner of an updated generalized Schur decomposition by an orthogonal similarity transformation

$$(A, B) = Q * (U_1 * (\hat{S}, \hat{T}) * U_2^T) * Z^T.$$

On exit, $S$ is overwritten by $\hat{S}$, $T$ is overwritten by $\hat{T}$, and $Q$ and $Z$ are overwritten by

$$Q \leftarrow Q * U_1 \text{ and } Z \leftarrow Z * U_2.$$

Reordering may in rare cases fail. In such cases the output is guaranteed to be a Schur decomposition and all (if any) selected generalized eigenvalues that are correctly placed are marked in the selection array on exit.

Reordering may perturb the generalized eigenvalues and the generalized eigenvalues after reordering are returned. The computed generalized eigenvalues are returned as a pair of values $(\alpha, \beta)$ such that $\alpha/\beta$ gives the actual generalized eigenvalue. The quantity $\alpha/\beta$ may overflow.

#### Combined reduction to generalized Schur form and eigenvalue reordering

Given a general matrix pencil $(A, B)$, the starneig_GEP_SM_Reduce() and starneig_GEP_DM_Reduce() interface functions compute a (reordered) generalized Schur decomposition

$$(A, B) = U_1 * (S, T) * U_2^T,$$

where $S$ is upper quasi-triangular with $1 \times 1$ and $2 \times 2$ blocks on the diagonal, $T$ is a upper triangular matrix, and $U_1$ and $U_2$ are orthogonal. Optionally, the interface functions attempt to reorder selected generalized eigenvalues to the top left corner of the generalized Schur decomposition.

On exit, $A$ is overwritten by $S$, $B$ is overwritten by $T$, and $Q$ and $Z$ (which are orthogonal matrices on entry) are overwritten by

$$Q \leftarrow Q * U_1 \text{ and } Z \leftarrow Z * U_2.$$

The computed generalized eigenvalues are returned as a pair of values $(\alpha, \beta)$ such that $\alpha/\beta$ gives the actual generalized eigenvalue. The quantity $\alpha/\beta$ may overflow.

Reordering may in rare cases fail. In such cases the output is guaranteed to be a Schur-triangular decomposition and all (if any) selected generalized eigenvalues that are correctly placed are marked in the selection array on exit.

#### Generalized eigenvectors

Given a generalized Schur decomposition

$$(A, B) = Q * (S, T) * Z^T$$

of a general matrix pencil $(A, B)$ and a selection of generalized eigenvalues, the starneig_GEP_SM_Eigenvectors() and starneig_GEP_DM_Eigenvectors() interface functions compute and return a generalized eigenvector for each of the selected generalized eigenvalues.

The generalized eigenvectors are stored as columns in the output matrix $X$ in the same order as their corresponding generalized eigenvalues appear in the selection array. A real generalized eigenvector is stored as a single column. The real and imaginary parts of a complex generalized eigenvector are stored as consecutive columns.

For a selected pair of complex conjugate generalized eigenvalues, a generalized eigenvector is computed only for the generalized eigenvalue with positive imaginary part. Thus, every selected generalized eigenvalue contributes one column to the output matrix and thus the number of selected generalized eigenvalues is equal to the number of columns of $X$.

#### Eigenvalue selection helper

Given a Schur-triangular matrix pencil $(S, T)$ and a predicate function, the starneig_GEP_SM_Select() and starneig_GEP_DM_Select() interface functions conveniently generate a correct selection array and count the number of selected generalized eigenvalues. The count is useful when allocating storage for the generalized eigenvector matrix computed by starneig_GEP_DM_Eigenvectors().

```
// a predicate function that selects all finite generalized eigenvalues that
// have a real part that is larger than a given value
static int predicate(double real, double imag, double beta, void *arg)
{
    double value = * (double *) arg;

    if (beta != 0.0 && value < real/beta)
        return 1;
    return 0;
}
```

```
void func(...)
{
    ...

    double value = 0.5;
    int num_selected, *selected = malloc(n*sizeof(int));
    starneig_GEP_SM_Select(
        n, S, ldS, T, ldT, &predicate, &value, selected, &num_selected);

    ...
}
```

See modules Shared Memory / Generalized EVP and Distributed Memory / Generalized EVP for further information. See also examples gep_sm_full_chain.c, gep_dm_full_chain.c and gep_sm_eigenvectors.c.

# 7 Expert functions

The library provides a set of configuration structures:

- starneig_hessenberg_conf : A configuration structure for Hessenberg reduction related expert interface functions.
- starneig_schur_conf : A configuration structure for Schur reduction related expert interface functions.
- starneig_reorder_conf : A configuration structure for eigenvalue reordering related interface functions.
- starneig_eigenvectors_conf : A configuration structure for eigenvector computation related interface functions.

The default parameters can generated with the following interface functions:

- starneig_hessenberg_init_conf() : Generates default parameters for Hessenberg reduction related expert interface functions.
- starneig_schur_init_conf() : Generates default parameters for Schur reduction related expert interface functions.
- starneig_reorder_init_conf() : Generates default parameters for eigenvalue reordering related interface functions.
- starneig_eigenvectors_init_conf() : Generates default parameters for eigenvector computation related interface functions.

A user is allowed to modify these default values before passing them to the expert interface function.

Only certain interface functions have expert version:

- starneig_SEP_SM_Hessenberg_expert()
- starneig_SEP_SM_Schur_expert()
- starneig_SEP_SM_ReorderSchur_expert()
- starneig_SEP_SM_Eigenvectors_expert()
- starneig_SEP_DM_Schur_expert()
- starneig_SEP_DM_ReorderSchur_expert()
- starneig_GEP_SM_Schur_expert()
- starneig_GEP_SM_ReorderSchur_expert()
- starneig_GEP_SM_Eigenvectors_expert()
- starneig_GEP_DM_Schur_expert()
- starneig_GEP_DM_ReorderSchur_expert()

See module Expert configuration structures for further information.

# 8 Test program

The test program provides an unified interface for testing the entire library. Most command line arguments have default values so only few of them have to be set in most situations. General usage information can be printed as follows:

```
$ ./starneig-test
Usage: ./starneig-test (options)

Global options:
  --mpi -- Enable MPI
  --mpi-mode [serialized,multiple] -- MPI mode
  --seed (num) -- Random number generator seed
  --experiment (experiment) -- Experiment module
  --test-workers [(num),default] -- Test program StarPU worker count
  --blas-threads [(num),default] -- Test program BLAS thread count
  --lapack-threads [(num),default] -- LAPACK solver thread count
  --scalapack-threads [(num),default] -- ScaLAPACK solver thread count

Available experiment modules:
    'hessenberg' : Hessenberg reduction experiment
    'schur' : Schur reduction experiment
    'reorder' : Eigenvalue reordering experiment
    'eigenvectors' : Eigenvectors experiment
    'full-chain' : Full chain experiment
    'partial-hessenberg' : Partial Hessenberg reduction experiment
    'validator' : Validation experiment
```

The `--mpi` option enables the MPI support and `--seed (num)` option initializes the random number generator with a given seed. Available experiment modules are listed below the global options and the desired experiment module is selected with the `--experiment (experiment)` option. For example, the Hessenberg reduction specific experiment module usage information can be printed as follows:

```
$ ./starneig-test --experiment hessenberg
Usage: ./starneig-test (options)

Global options:
  --mpi -- Enable MPI
  --mpi-mode [serialized,multiple] -- MPI mode
  --seed (num) -- Random number generator seed
  --experiment (experiment) -- Experiment module
  --test-workers [(num),default] -- Test program StarPU worker count
  --blas-threads [(num),default] -- Test program BLAS thread count
  --lapack-threads [(num),default] -- LAPACK solver thread count
  --scalapack-threads [(num),default] -- ScaLAPACK solver thread count

Available experiment modules:
    'hessenberg' : Hessenberg reduction experiment
    'schur' : Schur reduction experiment
    'reorder' : Eigenvalue reordering experiment
    'eigenvectors' : Eigenvectors experiment
    'full-chain' : Full chain experiment
    'partial-hessenberg' : Partial Hessenberg reduction experiment
    'validator' : Validation experiment

Experiment module (hessenberg) specific options:
  --data-format (format) -- Data format
  --init (initializer) -- Initialization module
  --solver (solver) -- Solver module
  --hooks (hook:mode, ...) -- Hooks and modes
  --no-reinit -- Do not reinitialize after each repetition
  --repeat (num) -- Repeated experiment
  --warmup (num) -- Perform "warmups"
  --keep-going -- Try to recover from a solver failure
  --abort -- Call abort() in failure
...
```

The overall design of the test program is modular. Each experiment module is built on *initializers*, *solvers* and *hooks*. Each experiment module contains several of each, thus allowing a user to initialize the inout data in various ways and compare different solvers with each other. Each of these building blocks can be configured with various command line arguments. However, in most situations only the problem dimension `--n (num)` needs to be specified.

Hooks are used to test and validate the output of the solver. For example, `--hooks hessenberg residual print` option enables hooks that

- check whether the output matrix is in upper Hessenberg form;
- computes the related residuals using Frobenius norm (reported in terms of unit roundoff error) and checks that they are within the permissible limits; and
- prints the output matrices.

Certain general purpose initializers allow a user to read the input data from a disk (`read-mtx` and `read-raw`) and output data can be stored to a disk using a suitable post-processing hook (`store-raw`).

The test program supports various data formats. For example, shared memory experiments are usually performed using the `pencil-local` data format which stores the matrices continuously in the main memory. Distributed memory experiments are performed using either a StarNEig library specific distributed data format (`pencil-starneig`) and the BLACS data format (`pencil-starneig-blacs`). The test program will detect the correct data format automatically. The test program is also is capable of converting the data between various data formats. The related data converter modules can be in most cases configured using additional command line arguments. For example, the distributed data formats distribute the data in rectangular sections and the section size can be set with command line arguments `--section-height (num)` and `--section-width (num)`).

#### Performance models

The StarPU performance models must be calibrated before the software can function efficiently on heterogeneous platforms (CPUs+GPUs). The calibration is triggered automatically if the models are not calibrated well enough for a given problem size. This can impact the execution time negatively. The test program provides an easy-to-use solution for this problem:

```
$ ./starneig-test ... --warmup 3
```

The `--warmup (number)` argument causes the test program to perform a number of warm-up runs before the actual execution time measurement takes place.

Please see the StarPU handbook for further instructions: http://starpu.gforge.inria.fr/doc/html/↩ Scheduling.html

#### Examples

- Reorder a 4000 x 4000 matrix using the StarNEig implementation:

```
$ ./starneig-test --experiment reorder --n 4000
TEST: --seed 1585762840 --experiment reorder --test-workers default --blas-threads default --lapack-threads
       default --scalapack-threads default --data-format pencil-local --init default --n 4000 --complex-distr
       uniform --complex-ratio 0.500000 --zero-ratio 0.010000 --inf-ratio 0.010000 --data-distr default
       --section-height default --section-width default --select-ratio 0.350000 --solver starneig --cores default --gpus defau
       --tile-size default --window-size default --values-per-chain default --small-window-size default
       --small-window-threshold default --update-width default --update-height default --plan default --blueprint default
       --hooks schur:normal eigenvalues:normal analysis:normal reordering:normal residual:normal
       --eigenvalues-fail-threshold 10000 --eigenvalues-warn-threshold 1000 --reordering-fail-threshold 10000 --reordering-wa
       1000 --residual-fail-threshold 10000 --residual-warn-threshold 500 --repeat 1 --warmup 0
THREADS: Using 6 StarPU worker threads during initialization and validation.
THREADS: Using 6 BLAS threads during initialization and validation.
THREADS: Using 6 BLAS threads in LAPACK solvers.
THREADS: Using 1 BLAS threads in ScaLAPACK solvers.
INIT...
PREPARE...
PROCESS...
[starneig][message] Setting tile size to 192.
[starneig][message] Using multi-part task insertion plan.
[starneig][message] Using two-pass backward dummy blueprint.
[starneig][message] Using "rounded" window size.
EXPERIMENT TIME = 1701 MS
FINALIZE...
EIGENVALUES CHECK: mean = 0 u, min = 0 u, max = 0 u
```

```
EIGENVALUES ANALYSIS: zeros = 36, infinities = 0, indefinites = 0
EIGENVALUES ANALYSIS: close zeros = 0, close infinities = 0, close indefinites = 0
REORDERING CHECK: Checking selected eigenvalues...
REORDERING CHECK: Checking other eigenvalues...
REORDERING CHECK: mean = 71 u, min = 0 u, max = 526 u
|Q ~A Q^T - A| / |A| = 314 u
|Q Q^T - I| / |I| = 140 u
================================================================
TIME = 1701 MS [avg 1701 MS, cv 0.00, min 1701 MS, max 1701 MS]
NO FAILED SCHUR FORM TESTS
EIGENVALUES CHECK (WARNINGS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES CHECK (FAILS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES CHECK (MEANS): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES CHECK (MIN): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES CHECK (MAX): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES ANALYSIS (ZEROS): [avg 36.0, cv 0.00, min 36, max 36]
EIGENVALUES ANALYSIS (CLOSE ZEROS): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (INFINITIES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE INFINITIES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (INDEFINITES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE INDEFINITES): [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (WARNINGS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (FAILS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (MEANS): [avg 71 u, cv 0.00, min 71 u, max 71 u]
REORDERING CHECK (MIN): [avg 0 u, cv 0.00, min 0 u, max 0 u]
REORDERING CHECK (MAX): [avg 526 u, cv 0.00, min 526 u, max 526 u]
|Q ~A Q^T - A| / |A| = [avg 314 u, cv 0.00, min 314 u, max 314 u]
|Q Q^T - I| / |I| = [avg 140 u, cv 0.00, min 140 u, max 140 u]
```

- Reorder a 4000 x 4000 matrix stencil $(A, B)$ using the StarNEig implementation, initialize the random number generator using the seed `1480591971`, fortify the matrix stencil $(A, B)$ against failed swaps, and disable GPUs:

```
$ ./starneig-test --experiment reorder --n 4000 --generalized --seed 1480591971 --fortify --gpus 0
TEST: --seed 1480591971 --experiment reorder --test-workers default --blas-threads default --lapack-threads
       default --scalapack-threads default --data-format pencil-local --init default --n 4000 --generalized
       --complex-distr uniform --fortify --data-distr default --section-height default --section-width default
       --select-ratio 0.350000 --solver starneig --cores default --gpus 0 --tile-size default --window-size default
       --values-per-chain default --small-window-size default --small-window-threshold default --update-width default
       --update-height default --plan default --blueprint default --hooks schur:normal eigenvalues:normal analysis:normal
       reordering:normal residual:normal --eigenvalues-fail-threshold 10000 --eigenvalues-warn-threshold 1000
       --reordering-fail-threshold 10000 --reordering-warn-threshold 1000 --residual-fail-threshold 10000
       --residual-warn-threshold 500 --repeat 1 --warmup 0
THREADS: Using 6 StarPU worker threads during initialization and validation.
THREADS: Using 6 BLAS threads during initialization and validation.
THREADS: Using 6 BLAS threads in LAPACK solvers.
THREADS: Using 1 BLAS threads in ScaLAPACK solvers.
INIT...
PREPARE...
PROCESS...
[starneig][message] Setting tile size to 192.
[starneig][message] Using multi-part task insertion plan.
[starneig][message] Using two-pass backward dummy blueprint.
[starneig][message] Using "rounded" window size.
EXPERIMENT TIME = 7472 MS
FINALIZE...
EIGENVALUES CHECK: mean = 0 u, min = 0 u, max = 0 u
EIGENVALUES ANALYSIS: zeros = 0, infinities = 0, indefinites = 0
EIGENVALUES ANALYSIS: close zeros = 0, close infinities = 0, close indefinites = 0
REORDERING CHECK: Checking selected eigenvalues...
REORDERING CHECK: Checking other eigenvalues...
REORDERING CHECK: mean = 23 u, min = 0 u, max = 169 u
|Q ~A Z^T - A| / |A| = 46 u
|Q ~B Z^T - B| / |B| = 63 u
|Q Q^T - I| / |I| = 44 u
|Z Z^T - I| / |I| = 43 u
================================================================
TIME = 7472 MS [avg 7472 MS, cv 0.00, min 7472 MS, max 7472 MS]
NO FAILED SCHUR FORM TESTS
EIGENVALUES CHECK (WARNINGS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES CHECK (FAILS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES CHECK (MEANS): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES CHECK (MIN): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES CHECK (MAX): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES ANALYSIS (ZEROS): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE ZEROS): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (INFINITIES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE INFINITIES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (INDEFINITES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE INDEFINITES): [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (WARNINGS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (FAILS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
```

```
REORDERING CHECK (MEANS): [avg 23 u, cv 0.00, min 23 u, max 23 u]
REORDERING CHECK (MIN): [avg 0 u, cv 0.00, min 0 u, max 0 u]
REORDERING CHECK (MAX): [avg 169 u, cv 0.00, min 169 u, max 169 u]
|Q ~A Z^T - A| / |A| = [avg 46 u, cv 0.00, min 46 u, max 46 u]
|Q ~B Z^T - B| / |B| = [avg 63 u, cv 0.00, min 63 u, max 63 u]
|Q Q^T - I| / |I| = [avg 44 u, cv 0.00, min 44 u, max 44 u]
|Z Z^T - I| / |I| = [avg 43 u, cv 0.00, min 43 u, max 43 u]
```

- Reduce a dense matrix to upper Hessenberg form, validate the output and store the output matrices to the disk:

```
$ ./starneig-test --experiment hessenberg --n 4000 --hooks hessenberg residual store-raw --store-raw-output
       hessenberg_%s.dat
TEST: --seed 1585762935 --experiment hessenberg --test-workers default --blas-threads default
       --lapack-threads default --scalapack-threads default --data-format pencil-local --init default --n 4000 --data-distr
       default --section-height default --section-width default --solver starneig --cores default --gpus default
       --tile-size default --panel-width default --hooks hessenberg:normal residual:normal store-raw:normal
       --residual-fail-threshold 10000 --residual-warn-threshold 500 --store-raw-output hessenberg_%s.dat --repeat 1 --warmup
THREADS: Using 6 StarPU worker threads during initialization and validation.
THREADS: Using 6 BLAS threads during initialization and validation.
THREADS: Using 6 BLAS threads in LAPACK solvers.
THREADS: Using 1 BLAS threads in ScaLAPACK solvers.
INIT...
PREPARE...
PROCESS...
[starneig][message] Setting tile size to 336.
[starneig][message] Setting panel width to 288.
EXPERIMENT TIME = 13121 MS
FINALIZE...
|Q ~A Q^T - A| / |A| = 15 u
|Q Q^T - I| / |I| = 11 u
WRITING TO hessenberg_A.dat...
WRITING TO hessenberg_Q.dat...
WRITING TO hessenberg_CA.dat...
================================================================
TIME = 13121 MS [avg 13121 MS, cv 0.00, min 13121 MS, max 13121 MS]
NO FAILED HESSENBERG FORM TESTS
|Q ~A Q^T - A| / |A| = [avg 15 u, cv 0.00, min 15 u, max 15 u]
|Q Q^T - I| / |I| = [avg 11 u, cv 0.00, min 11 u, max 11 u]
```

- Read an upper Hessenberg matrix from the disk, reduce it to Schur form and set tile size to `128`:

```
$ ./starneig-test --experiment schur --init read-raw --input hessenberg_%s.dat --tile-size 128
TEST: --seed 1585762972 --experiment schur --test-workers default --blas-threads default --lapack-threads
       default --scalapack-threads default --data-format pencil-local --init read-raw --input hessenberg_%s.dat
       --data-distr default --section-height default --section-width default --solver starneig --cores default --gpus
       default --iteration-limit default --tile-size 128 --small-limit default --aed-window-size default
       --aed-shift-count default --aed-nibble default --aed-parallel-soft-limit default --aed-parallel-hard-limit default
       --window-size default --shifts-per-window default --update-width default --update-height default
       --left-threshold default --right-threshold default --inf-threshold default --hooks schur:normal eigenvalues:normal
       known-eigenvalues:normal analysis:normal residual:normal --eigenvalues-fail-threshold 10000
       --eigenvalues-warn-threshold 1000 --known-eigenvalues-fail-threshold 1000000 --known-eigenvalues-warn-threshold 10000
       --residual-fail-threshold 10000 --residual-warn-threshold 500 --repeat 1 --warmup 0
THREADS: Using 6 StarPU worker threads during initialization and validation.
THREADS: Using 6 BLAS threads during initialization and validation.
THREADS: Using 6 BLAS threads in LAPACK solvers.
THREADS: Using 1 BLAS threads in ScaLAPACK solvers.
INIT...
READING FROM hessenberg_A.dat...
READING A 4000 X 4000 MATRIX ...
READING FROM hessenberg_Q.dat...
READING A 4000 X 4000 MATRIX ...
READING FROM hessenberg_CA.dat...
READING A 4000 X 4000 MATRIX ...
PREPARE...
PROCESS...
[starneig][message] Using AED windows size 320.
[starneig][message] Using 240 shifts.
EXPERIMENT TIME = 9479 MS
FINALIZE...
EIGENVALUES CHECK: mean = 0 u, min = 0 u, max = 0 u
KNOWN EIGENVALUES CHECK: The stored pencil does not contain the known eigenvalues. Skipping.
EIGENVALUES ANALYSIS: zeros = 0, infinities = 0, indefinites = 0
EIGENVALUES ANALYSIS: close zeros = 0, close infinities = 0, close indefinites = 0
|Q ~A Q^T - A| / |A| = 68 u
|Q Q^T - I| / |I| = 89 u
================================================================
TIME = 9479 MS [avg 9479 MS, cv 0.00, min 9479 MS, max 9479 MS]
NO FAILED SCHUR FORM TESTS
```

```
EIGENVALUES CHECK (WARNINGS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES CHECK (FAILS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES CHECK (MEANS): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES CHECK (MIN): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES CHECK (MAX): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES ANALYSIS (ZEROS): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE ZEROS): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (INFINITIES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE INFINITIES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (INDEFINITES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE INDEFINITES): [avg 0.0, cv 0.00, min 0, max 0]
|Q ~A Q^T - A| / |A| = [avg 68 u, cv 0.00, min 68 u, max 68 u]
|Q Q^T - I| / |I| = [avg 89 u, cv 0.00, min 89 u, max 89 u]
```

#### Distributed memory example

- Reorder a 4000 x 4000 matrix using the StarNEig implementation, use two MPI ranks, use three CPU cores per rank, distribute the matrix in 1024 x 1024 sections, and use tile size 256:

```
$ mpirun -n 2 --map-by :PE=3 ./starneig-test --mpi --experiment reorder --n 4000 --section-height 1024
       --section-width 1024 --tile-size 256
```

Rank 0 output:

```
MPI INIT...
TEST: --mpi --mpi-mode serialized --seed 1585763077 --experiment reorder --test-workers default
       --blas-threads default --lapack-threads default --scalapack-threads default --data-format pencil-starneig-blacs --init
       default --n 4000 --complex-distr uniform --complex-ratio 0.500000 --zero-ratio 0.010000 --inf-ratio 0.010000
       --data-distr default --section-height 1024 --section-width 1024 --select-ratio 0.350000 --solver starneig
       --cores default --gpus default --tile-size 256 --window-size default --values-per-chain default
       --small-window-size default --small-window-threshold default --update-width default --update-height default --plan
       default --blueprint default --hooks schur:normal eigenvalues:normal analysis:normal reordering:normal
       residual:normal --eigenvalues-fail-threshold 10000 --eigenvalues-warn-threshold 1000 --reordering-fail-threshold 10000
       --reordering-warn-threshold 1000 --residual-fail-threshold 10000 --residual-warn-threshold 500 --repeat 1
       --warmup 0
THREADS: Using 3 StarPU worker threads during initialization and validation.
THREADS: Using 3 BLAS threads during initialization and validation.
THREADS: Using 3 BLAS threads in LAPACK solvers.
THREADS: Using 1 BLAS threads in ScaLAPACK solvers.
INIT...
PREPARE...
PROCESS...
[starneig][message] Attempting to set tile size to 256.
[starneig][message] Setting tile size to 256.
[starneig][message] Using multi-part task insertion plan.
[starneig][message] Using two-pass backward dummy blueprint.
[starneig][message] Using "rounded" window size.
EXPERIMENT TIME = 3320 MS
FINALIZE...
EIGENVALUES CHECK: mean = 0 u, min = 0 u, max = 0 u
EIGENVALUES ANALYSIS: zeros = 37, infinities = 0, indefinites = 0
EIGENVALUES ANALYSIS: close zeros = 0, close infinities = 0, close indefinites = 0
REORDERING CHECK: Checking selected eigenvalues...
REORDERING CHECK: Checking other eigenvalues...
REORDERING CHECK: mean = 70 u, min = 0 u, max = 527 u
|Q ~A Q^T - A| / |A| = 314 u
|Q Q^T - I| / |I| = 139 u
================================================================
TIME = 3319 MS [avg 3320 MS, cv 0.00, min 3320 MS, max 3320 MS]
NO FAILED SCHUR FORM TESTS
EIGENVALUES CHECK (WARNINGS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES CHECK (FAILS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES CHECK (MEANS): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES CHECK (MIN): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES CHECK (MAX): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES ANALYSIS (ZEROS): [avg 37.0, cv 0.00, min 37, max 37]
EIGENVALUES ANALYSIS (CLOSE ZEROS): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (INFINITIES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE INFINITIES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (INDEFINITES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE INDEFINITES): [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (WARNINGS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (FAILS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (MEANS): [avg 70 u, cv 0.00, min 70 u, max 70 u]
REORDERING CHECK (MIN): [avg 0 u, cv 0.00, min 0 u, max 0 u]
REORDERING CHECK (MAX): [avg 527 u, cv 0.00, min 527 u, max 527 u]
|Q ~A Q^T - A| / |A| = [avg 314 u, cv 0.00, min 314 u, max 314 u]
|Q Q^T - I| / |I| = [avg 139 u, cv 0.00, min 139 u, max 139 u]
```

Rank 1 output:

```
MPI INIT...
TEST: --mpi --mpi-mode serialized --seed 1585763077 --experiment reorder --test-workers default
       --blas-threads default --lapack-threads default --scalapack-threads default --data-format pencil-starneig-blacs --init
       default --n 4000 --complex-distr uniform --complex-ratio 0.500000 --zero-ratio 0.010000 --inf-ratio 0.010000
       --data-distr default --section-height 1024 --section-width 1024 --select-ratio 0.350000 --solver starneig
       --cores default --gpus default --tile-size 256 --window-size default --values-per-chain default
       --small-window-size default --small-window-threshold default --update-width default --update-height default --plan
       default --blueprint default --hooks schur:normal eigenvalues:normal analysis:normal reordering:normal
       residual:normal --eigenvalues-fail-threshold 10000 --eigenvalues-warn-threshold 1000 --reordering-fail-threshold 10000
       --reordering-warn-threshold 1000 --residual-fail-threshold 10000 --residual-warn-threshold 500 --repeat 1
       --warmup 0
THREADS: Using 3 StarPU worker threads during initialization and validation.
THREADS: Using 3 BLAS threads during initialization and validation.
THREADS: Using 3 BLAS threads in LAPACK solvers.
THREADS: Using 1 BLAS threads in ScaLAPACK solvers.
INIT...
PREPARE...
PROCESS...
[starneig][message] Attempting to set tile size to 256.
[starneig][message] Setting tile size to 256.
[starneig][message] Using multi-part task insertion plan.
[starneig][message] Using two-pass backward dummy blueprint.
[starneig][message] Using "rounded" window size.
EXPERIMENT TIME = 3320 MS
FINALIZE...
EIGENVALUES CHECK: mean = 0 u, min = 0 u, max = 0 u
EIGENVALUES ANALYSIS: zeros = 37, infinities = 0, indefinites = 0
EIGENVALUES ANALYSIS: close zeros = 0, close infinities = 0, close indefinites = 0
REORDERING CHECK: Checking selected eigenvalues...
REORDERING CHECK: Checking other eigenvalues...
REORDERING CHECK: mean = 70 u, min = 0 u, max = 527 u
|Q ~A Q^T - A| / |A| = 314 u
|Q Q^T - I| / |I| = 139 u
================================================================
TIME = 3319 MS [avg 3320 MS, cv 0.00, min 3320 MS, max 3320 MS]
NO FAILED SCHUR FORM TESTS
EIGENVALUES CHECK (WARNINGS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES CHECK (FAILS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES CHECK (MEANS): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES CHECK (MIN): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES CHECK (MAX): [avg 0 u, cv 0.00, min 0 u, max 0 u]
EIGENVALUES ANALYSIS (ZEROS): [avg 37.0, cv 0.00, min 37, max 37]
EIGENVALUES ANALYSIS (CLOSE ZEROS): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (INFINITIES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE INFINITIES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (INDEFINITES): [avg 0.0, cv 0.00, min 0, max 0]
EIGENVALUES ANALYSIS (CLOSE INDEFINITES): [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (WARNINGS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (FAILS): 0 runs effected [avg 0.0, cv 0.00, min 0, max 0]
REORDERING CHECK (MEANS): [avg 70 u, cv 0.00, min 70 u, max 70 u]
REORDERING CHECK (MIN): [avg 0 u, cv 0.00, min 0 u, max 0 u]
REORDERING CHECK (MAX): [avg 527 u, cv 0.00, min 527 u, max 527 u]
|Q ~A Q^T - A| / |A| = [avg 314 u, cv 0.00, min 314 u, max 314 u]
|Q Q^T - I| / |I| = [avg 139 u, cv 0.00, min 139 u, max 139 u]
```

# 9 License and authors

The following people have contributed to the library:

- Angelika Schwarz (angies@cs.umu.se)
  - Standard eigenvectors
- Bo Kågström (bokg@cs.umu.se)
  - Coordinator and scientific director for the NLAFET project
  - Documentation
- Carl Christian Kjelgaard Mikkelsen (spock@cs.umu.se)
  - Generalized eigenvectors
- Lars Karlsson (larsk@cs.umu.se)
  - Miscellaneous user interface functions
  - Documentation
- Mirko Myllykoski (mirkom@cs.umu.se)
  - Hessenberg reduction
  - Schur reduction (standard and generalized)
  - Eigenvalue reordering (standard and generalized)
  - Miscellaneous user interface functions
  - Test program
  - Documentation

# 10 Todo List

**Global starneig_GEP_DM_Eigenvectors (int selected[], starneig_distr_matrix_t S, starneig_distr_matrix_t T, starneig_distr_matrix_t Z, starneig_distr_matrix_t X)**

This interface function is not implemented.

**Global starneig_GEP_DM_Eigenvectors_expert (struct starneig_eigenvectors_conf ∗conf, int selected[], starneig_distr_matrix_t S, starneig_distr_matrix_t T, starneig_distr_matrix_t Z, starneig_distr_matrix_t X)**

This interface function is not implemented.

**Global starneig_SEP_DM_Eigenvectors (int selected[], starneig_distr_matrix_t S, starneig_distr_matrix_t Q, starneig_distr_matrix_t X)**

This interface function is not implemented.

**Global starneig_SEP_DM_Eigenvectors_expert (struct starneig_eigenvectors_conf ∗conf, int selected[], starneig_distr_matrix_t S, starneig_distr_matrix_t Q, starneig_distr_matrix_t X)**

This interface function is not implemented.

# 11 Module Documentation

## 11.1 Library configuration

Configuration of the installed library.

#### Macros

- #define STARNEIG_ENABLE_MPI

    *Defined if the library was compiled with MPI support.*
- #define STARNEIG_ENABLE_CUDA

    *Defined if the library was compiled with CUDA support.*
- #define STARNEIG_ENABLE_BLACS

    *Defined if the library was compiled with BLACS support.*
- #define STARNEIG_SEP_DM_HESSENBERG

    *Defined if the starneig_SEP_DM_Hessenberg() function exists.*
- #define STARNEIG_GEP_DM_HESSENBERGTRIANGULAR

    *Defined if the starneig_GEP_DM_HessenbergTriangular() function exists.*
- #define STARNEIG_SEP_DM_REDUCE

    *Defined if the starneig_SEP_DM_Reduce() function exists.*
- #define STARNEIG_GEP_DM_REDUCE

    *Defined if the starneig_GEP_DM_Reduce() function exists.*

### 11.1.1 Detailed Description

Configuration of the installed library.

## 11.2 Error codes

Interface function return values and error codes.

### Macros

- #define STARNEIG_SUCCESS 0

  *The interface function was executed successfully.*
- #define STARNEIG_GENERIC_ERROR 1

  *The interface function encountered a generic error.*
- #define STARNEIG_NOT_INITIALIZED 2

  *The library was not initialized when the interface function was called.*
- #define STARNEIG_INVALID_CONFIGURATION 3

  *The interface function encountered an invalid configuration argument.*
- #define STARNEIG_INVALID_ARGUMENTS 4

  *The interface function encountered an invalid argument.*
- #define STARNEIG_INVALID_DISTR_MATRIX 5

  *One or more of the involved distributed matrices have an invalid distribution, invalid dimensions and/or an invalid distributed block size.*
- #define STARNEIG_DID_NOT_CONVERGE 6

  *The interface function encountered a situation where the QR/QZ algorithm did not converge. The matrix (pencil) may be partially in Schur form.*
- #define STARNEIG_PARTIAL_REORDERING 7

  *The interface function failed to reorder the (generalized) Schur form. The (generalized) Schur form may be partially reordered.*
- #define STARNEIG_CLOSE_EIGENVALUES 8

  *The interface function encountered a situation where two selected eigenvalues were close to each other.*

### Typedefs

- typedef int starneig_error_t

  *Interface function return value data type.*

### 11.2.1 Detailed Description

Interface function return values and error codes.

## 11.3 Intra-node execution environment

Interface to configure the intra-node execution environment.

### Functions

- void starneig_node_init (int cores, int gpus, starneig_flag_t flags)
  *Initializes the intra-node execution environment.*
- int starneig_node_initialized ()
  *Checks whether the intra-node execution environment is initialized.*
- int starneig_node_get_cores ()
  *Returns the number of cores (threads) per MPI rank.*
- void starneig_node_set_cores (int cores)
  *Changes the number of CPUs cores (threads) to use per MPI rank.*
- int starneig_node_get_gpus ()
  *Returns the number of GPUs per MPI rank.*
- void starneig_node_set_gpus (int gpus)
  *Changes the number of GPUs to use per MPI rank.*
- void starneig_node_finalize ()
  *Deallocates resources associated with the intra-node configuration.*

### Library initialization flags

- typedef unsigned starneig_flag_t
  *Library initialization flag data type.*
- #define STARNEIG_DEFAULT 0x0
  *Default initialization flag.*
- #define STARNEIG_HINT_SM 0x0
  *Initializes the library for shared memory computation.*
- #define STARNEIG_HINT_DM 0x1
  *Initializes the library for distributed memory computation.*
- #define STARNEIG_FXT_DISABLE 0x2
  *Disables FXT traces.*
- #define STARNEIG_AWAKE_WORKERS 0x4
  *Keeps worker threads awake.*
- #define STARNEIG_AWAKE_MPI_WORKER 0x8
  *Keeps StarPU-MPI communication thread awake.*
- #define STARNEIG_FAST_DM (STARNEIG_HINT_DM | STARNEIG_AWAKE_WORKERS | STARNEIG_AWAKE_MPI_WORKER)
  *Enables fast StarPU-MPI mode.*
- #define STARNEIG_NO_VERBOSE 0x10
  *Disables verbose messages.*
- #define STARNEIG_NO_MESSAGES (STARNEIG_NO_VERBOSE | 0x20)
  *Disables messages.*

### Pinned host memory

- void starneig_node_enable_pinning ()
  *Enable CUDA host memory pinning.*
- void starneig_node_disable_pinning ()
  *Disables CUDA host memory pinning.*

**Distributed memory**

- void starneig_mpi_set_comm (MPI_Comm comm)
    *Sets a MPI communicator for the library.*
- MPI_Comm starneig_mpi_get_comm ()
    *Returns the library MPI communicator.*

### 11.3.1 Detailed Description

Interface to configure the intra-node execution environment.

### 11.3.2 Macro Definition Documentation

#### 11.3.2.1 STARNEIG_DEFAULT

```
#define STARNEIG_DEFAULT 0x0
```

Default initialization flag.

The library defaults to the shared memory mode.

#### 11.3.2.2 STARNEIG_HINT_SM

```
#define STARNEIG_HINT_SM 0x0
```

Initializes the library for shared memory computation.

The library will automatically reconfigure itself for distributed memory computation.

**Examples:**

gep_sm_eigenvectors.c, gep_sm_full_chain.c, sep_sm_eigenvectors.c, and sep_sm_full_chain.c.

#### 11.3.2.3 STARNEIG_HINT_DM

```
#define STARNEIG_HINT_DM 0x1
```

Initializes the library for distributed memory computation.

The library will automatically reconfigure itself for shared memory computation.

**Examples:**

sep_dm_full_chain.c.

#### 11.3.2.4 STARNEIG_FXT_DISABLE

```
#define STARNEIG_FXT_DISABLE 0x2
```

Disables FXT traces.

This flag does not work reliably with all StarPU versions.

#### 11.3.2.5 STARNEIG_AWAKE_WORKERS

```
#define STARNEIG_AWAKE_WORKERS 0x4
```

Keeps worker threads awake.

Keeps the StarPU worker threads awake between interface function calls.

**Examples:**

gep_sm_full_chain.c.

#### 11.3.2.6 STARNEIG_AWAKE_MPI_WORKER

```
#define STARNEIG_AWAKE_MPI_WORKER 0x8
```

Keeps StarPU-MPI communication thread awake.

Keeps the StarPU-MPI communication thread awake between interface function calls.

#### 11.3.2.7 STARNEIG_FAST_DM

```
#define STARNEIG_FAST_DM (STARNEIG_HINT_DM | STARNEIG_AWAKE_WORKERS | STARNEIG_AWAKE_MPI_WORK↩
ER)
```

Enables fast StarPU-MPI mode.

Keeps the worker threads and StarPU-MPI communication thread awake between interface function calls.

**Examples:**

gep_dm_full_chain.c.

#### 11.3.2.8 STARNEIG_NO_VERBOSE

```
#define STARNEIG_NO_VERBOSE 0x10
```

Disables verbose messages.

Disables all additional verbose messages.

#### 11.3.2.9 STARNEIG_NO_MESSAGES

```
#define STARNEIG_NO_MESSAGES (STARNEIG_NO_VERBOSE | 0x20)
```

Disables messages.

Disables all messages (including verbose).

### 11.3.3 Function Documentation

#### 11.3.3.1 starneig_node_init()

```
void starneig_node_init (
             int cores,
             int gpus,
             starneig_flag_t flags )
```

Initializes the intra-node execution environment.

The interface function initializes StarPU (and cuBLAS) and pauses all worker The `cores` argument specifies the **total number of used CPU cores**. In distributed memory mode, one CPU core is automatically allocated for the StarPU-MPI communication thread. One or more CPU cores are automatically allocated for GPU devices.

**Parameters**

| | | |
|---|---|---|
| in | *cores* | The number of cores (threads) to use per MPI rank. Can be set to -1 in which case the library determines the value. |
| in | *gpus* | The number of GPUs to use per MPI rank. Can be set to -1 in which case the library determines the value. |
| in | *flags* | Initialization flags. |

**Examples:**

gep_dm_full_chain.c, gep_sm_eigenvectors.c, gep_sm_full_chain.c, sep_dm_full_chain.c, sep_sm_↩ eigenvectors.c, and sep_sm_full_chain.c.

#### 11.3.3.2 starneig_node_initialized()

```
int starneig_node_initialized ( )
```

Checks whether the intra-node execution environment is initialized.

**Returns**

Non-zero if the environment is initialized, 0 otherwise.

#### 11.3.3.3 starneig_node_get_cores()

```
int starneig_node_get_cores ( )
```

Returns the number of cores (threads) per MPI rank.

**Returns**

The number of cores (threads) per MPI rank.

#### 11.3.3.4 starneig_node_set_cores()

```
void starneig_node_set_cores (
            int cores )
```

Changes the number of CPUs cores (threads) to use per MPI rank.

**Parameters**

| | |
|---|---|
| *cores* | The number of CPUs to use per MPI rank. |

#### 11.3.3.5 starneig_node_get_gpus()

```
int starneig_node_get_gpus ( )
```

Returns the number of GPUs per MPI rank.

**Returns**

The number of GPUs per MPI rank.

#### 11.3.3.6 starneig_node_set_gpus()

```
void starneig_node_set_gpus (
            int gpus )
```

Changes the number of GPUs to use per MPI rank.

**Parameters**

| | |
|---|---|
| *gpus* | The number of GPUs to use per MPI rank. |

#### 11.3.3.7 starneig_node_enable_pinning()

```
void starneig_node_enable_pinning ( )
```

Enable CUDA host memory pinning.

Should be called before any memory allocations are made.

#### 11.3.3.8 starneig_node_disable_pinning()

```
void starneig_node_disable_pinning ( )
```

Disables CUDA host memory pinning.

Should be called before any memory allocations are made.

#### 11.3.3.9 starneig_mpi_set_comm()

```
void starneig_mpi_set_comm (
             MPI_Comm comm )
```

Sets a MPI communicator for the library.

Should be called before the starneig_node_init() interface function.

**Parameters**

| | | |
|---|---|---|
| in | *comm* | The library MPI communicator. |

#### 11.3.3.10 starneig_mpi_get_comm()

```
MPI_Comm starneig_mpi_get_comm ( )
```

Returns the library MPI communicator.

**Returns**

The library MPI communicator.

## 11.4 Shared Memory / Standard EVP

Functions for solving non-symmetric standard eigenvalue problems on shared memory systems.

### Computational functions

- starneig_error_t starneig_SEP_SM_Hessenberg (int n, double A[ ], int ldA, double Q[ ], int ldQ)

  *Computes a Hessenberg decomposition of a general matrix.*
- starneig_error_t starneig_SEP_SM_Schur (int n, double H[ ], int ldH, double Q[ ], int ldQ, double real[ ], double imag[ ])

  *Computes a Schur decomposition given a Hessenberg decomposition.*
- starneig_error_t starneig_SEP_SM_ReorderSchur (int n, int selected[ ], double S[ ], int ldS, double Q[ ], int ldQ, double real[ ], double imag[ ])

  *Reorders selected eigenvalues to the top left corner of a Schur decomposition.*
- starneig_error_t starneig_SEP_SM_Reduce (int n, double A[ ], int ldA, double Q[ ], int ldQ, double real[ ], double imag[ ], int(∗predicate)(double real, double imag, void ∗arg), void ∗arg, int selected[ ], int ∗num_↩ selected)

  *Computes a (reordered) Schur decomposition of a general matrix.*
- starneig_error_t starneig_SEP_SM_Eigenvectors (int n, int selected[ ], double S[ ], int ldS, double Q[ ], int ldQ, double X[ ], int ldX)

  *Computes an eigenvector for each selected eigenvalue.*

### Helper functions

- starneig_error_t starneig_SEP_SM_Select (int n, double S[ ], int ldS, int(∗predicate)(double real, double imag, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)

  *Generates a selection array for a Schur matrix using a user-supplied predicate function.*

### Expert computational functions

- starneig_error_t starneig_SEP_SM_Hessenberg_expert (struct starneig_hessenberg_conf ∗conf, int n, int begin, int end, double A[ ], int ldA, double Q[ ], int ldQ)

  *Computes a Hessenberg decomposition of a general matrix.*
- starneig_error_t starneig_SEP_SM_Schur_expert (struct starneig_schur_conf ∗conf, int n, double H[ ], int ldH, double Q[ ], int ldQ, double real[ ], double imag[ ])

  *Computes a Schur decomposition given a Hessenberg decomposition.*
- starneig_error_t starneig_SEP_SM_ReorderSchur_expert (struct starneig_reorder_conf ∗conf, int n, int selected[ ], double S[ ], int ldS, double Q[ ], int ldQ, double real[ ], double imag[ ])

  *Reorders selected eigenvalues to the top left corner of a Schur decomposition.*
- starneig_error_t starneig_SEP_SM_Eigenvectors_expert (struct starneig_eigenvectors_conf ∗conf, int n, int selected[ ], double S[ ], int ldS, double Q[ ], int ldQ, double X[ ], int ldX)

  *Computes an eigenvector for each selected eigenvalue.*

### 11.4.1 Detailed Description

Functions for solving non-symmetric standard eigenvalue problems on shared memory systems.

### 11.4.2 Function Documentation

#### 11.4.2.1 starneig_SEP_SM_Hessenberg()

```
starneig_error_t starneig_SEP_SM_Hessenberg (
            int n,
            double A[ ],
            int ldA,
            double Q[ ],
            int ldQ )
```

Computes a Hessenberg decomposition of a general matrix.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $A$ and $Q$. |
| in,out | *A* | On entry, the general matrix $A$. On exit, the upper Hessenberg matrix $H$. |
| in | *ldA* | The leading dimension of $A$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| in | *ldQ* | The leading dimension of $Q$. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**Examples:**

sep_sm_full_chain.c.

#### 11.4.2.2 starneig_SEP_SM_Schur()

```
starneig_error_t starneig_SEP_SM_Schur (
            int n,
            double H[ ],
            int ldH,
            double Q[ ],
            int ldQ,
            double real[ ],
            double imag[ ] )
```

Computes a Schur decomposition given a Hessenberg decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $H$ and $Q$. |
| in,out | *H* | On entry, the upper Hessenberg matrix $H$. On exit, the Schur matrix $S$. |
| in | *ldH* | The leading dimension of $H$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| in | *ldQ* | The leading dimension of $Q$. |
| out | *real* | An array of the same size as $H$ containing the real parts of the computed eigenvalues. |
| out | *imag* | An array of the same size as $H$ containing the imaginary parts of the computed eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_DID_NOT_CONVERGE if the QR algorithm failed to converge.

**Examples:**

sep_sm_full_chain.c.

#### 11.4.2.3 starneig_SEP_SM_ReorderSchur()

```
starneig_error_t starneig_SEP_SM_ReorderSchur (
            int n,
            int selected[ ],
            double S[ ],
            int ldS,
            double Q[ ],
            int ldQ,
            double real[ ],
            double imag[ ] )
```

Reorders selected eigenvalues to the top left corner of a Schur decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $S$ and $Q$. |
| in,out | *selected* | The selection array. On entry, the initial positions of the selected eigenvalues. On exit, the final positions of all correctly placed selected eigenvalues. In case of failure, the number of 1's in the output may be less than the number of 1's in the input. |
| in,out | *S* | On entry, the Schur matrix $S$. On exit, the updated Schur matrix $\hat{S}$. |
| in | *ldS* | The leading dimension of $S$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| in | *ldQ* | The leading dimension of $Q$. |
| out | *real* | An array of the same size as $S$ containing the real parts of the computed eigenvalues. |
| out | *imag* | An array of the same size as $S$ containing the imaginary parts of the computed eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_PARTIAL_REORDERING if the Schur form is not fully reordered.

**See also**

starneig_SEP_SM_Select

**Examples:**

sep_sm_full_chain.c.

#### 11.4.2.4 starneig_SEP_SM_Reduce()

```
starneig_error_t starneig_SEP_SM_Reduce (
            int n,
            double A[ ],
            int ldA,
            double Q[ ],
            int ldQ,
            double real[ ],
            double imag[ ],
            int(*)(double real, double imag, void *arg) predicate,
            void * arg,
            int selected[ ],
            int * num_selected )
```

Computes a (reordered) Schur decomposition of a general matrix.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $A$ and $Q$. |
| in,out | *A* | On entry, the general matrix $A$. On exit, the Schur matrix $S$. |
| in | *ldA* | The leading dimension of $A$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| in | *ldQ* | The leading dimension of $Q$. |
| out | *real* | An array of the same size as $A$ containing the real parts of the computed eigenvalues. |
| out | *imag* | An array of the same size as $A$ containing the imaginary parts of the computed eigenvalues. |
| in | *predicate* | A function that takes a (complex) eigenvalue as input and returns non-zero if it should be selected. For complex conjugate pairs of eigenvalues, the predicate is called only for the eigenvalue with positive imaginary part and the corresponding $2 \times 2$ block is either selected or deselected. The reordering step is skipped if the argument is a NULL pointer. |
| in | *arg* | An optional argument for the predicate function. |
| out | *selected* | The final positions of all correctly placed selected eigenvalues. |
| out | *num_selected* | The number of selected eigenvalues (a complex conjugate pair is counted as two selected eigenvalues). |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_DID_NOT_CONVERGE if the QR algorithm failed to converge. STARNEIG_PARTIAL_REORDERING if the Schur form is not fully reordered.

**Examples:**

sep_sm_eigenvectors.c.

#### 11.4.2.5 starneig_SEP_SM_Eigenvectors()

```
starneig_error_t starneig_SEP_SM_Eigenvectors (
            int n,
            int selected[ ],
            double S[ ],
            int ldS,
            double Q[ ],
            int ldQ,
            double X[ ],
            int ldX )
```

Computes an eigenvector for each selected eigenvalue.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $S$ and $Q$ and the number of rows of $X$. |
| in | *selected* | The selection array specifying the locations of the selected eigenvalues. The number of 1's in the array is the same as the number of columns in $X$. |
| in | *S* | The Schur matrix $S$. |
| in | *ldS* | The leading dimension of $S$. |
| in | *Q* | The orthogonal matrix $Q$. |
| in | *ldQ* | The The leading dimension of $Q$. |
| out | *X* | A matrix with $n$ rows and one column for each selected eigenvalue. The columns represent the computed eigenvectors as previously described. |
| in | *ldX* | The leading dimension of $X$. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_SEP_SM_Select

**Examples:**

sep_sm_eigenvectors.c.

#### 11.4.2.6 starneig_SEP_SM_Select()

```
starneig_error_t starneig_SEP_SM_Select (
            int n,
            double S[ ],
            int ldS,
            int(*)(double real, double imag, void *arg) predicate,
            void * arg,
            int selected[ ],
            int * num_selected )
```

Generates a selection array for a Schur matrix using a user-supplied predicate function.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $S$. |
| in | *S* | The Schur matrix $S$. |
| in | *ldS* | The leading dimension of $S$. |
| in | *predicate* | A function that takes a (complex) eigenvalue as input and returns non-zero if it should be selected. For complex conjugate pairs of eigenvalues, the predicate is called only for the eigenvalue with positive imaginary part and the corresponding $2 \times 2$ block is either selected or deselected. |
| in | *arg* | An optional argument for the predicate function. |
| out | *selected* | The selection array. Both elements of a selected complex conjugate pair are set to 1. |
| out | *num_selected* | The number of selected eigenvalues (a complex conjugate pair is counted as two selected eigenvalues). |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**Examples:**

sep_sm_eigenvectors.c, and sep_sm_full_chain.c.

#### 11.4.2.7 starneig_SEP_SM_Hessenberg_expert()

```
starneig_error_t starneig_SEP_SM_Hessenberg_expert (
            struct starneig_hessenberg_conf * conf,
            int n,
            int begin,
            int end,
            double A[ ],
            int ldA,
            double Q[ ],
            int ldQ )
```

Computes a Hessenberg decomposition of a general matrix.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in | *n* | The order of $A$ and $Q$. |
| in | *begin* | First column to be reduced. |
| in | *end* | Last column to be reduced. |
| in,out | *A* | On entry, the general matrix $A$. On exit, the upper Hessenberg matrix $H$. |
| in | *ldA* | The leading dimension of $A$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| in | *ldQ* | The leading dimension of $Q$. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_SEP_SM_Hessenberg
starneig_hessenberg_conf
starneig_hessenberg_init_conf

#### 11.4.2.8 starneig_SEP_SM_Schur_expert()

```
starneig_error_t starneig_SEP_SM_Schur_expert (
            struct starneig_schur_conf * conf,
            int n,
            double H[ ],
            int ldH,
            double Q[ ],
            int ldQ,
            double real[ ],
            double imag[ ] )
```

Computes a Schur decomposition given a Hessenberg decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in | *n* | The order of $H$ and $Q$. |
| in,out | *H* | On entry, the upper Hessenberg matrix $H$. On exit, the Schur matrix $S$. |
| in | *ldH* | The leading dimension of $H$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| in | *ldQ* | The leading dimension of $Q$. |
| out | *real* | An array of the same size as $H$ containing the real parts of the computed eigenvalues. |
| out | *imag* | An array of the same size as $H$ containing the imaginary parts of the computed eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_SEP_SM_Schur
starneig_schur_conf
starneig_schur_init_conf

#### 11.4.2.9 starneig_SEP_SM_ReorderSchur_expert()

```
starneig_error_t starneig_SEP_SM_ReorderSchur_expert (
            struct starneig_reorder_conf * conf,
            int n,
            int selected[ ],
            double S[ ],
            int ldS,
            double Q[ ],
            int ldQ,
            double real[ ],
            double imag[ ] )
```

Reorders selected eigenvalues to the top left corner of a Schur decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in | *n* | The order of $S$ and $Q$. |
| in,out | *selected* | The selection array. |
| in,out | *S* | On entry, the Schur matrix $S$. On exit, the updated Schur matrix $\hat{S}$. |
| in | *ldS* | The leading dimension of $S$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| in | *ldQ* | The leading dimension of $Q$. |
| out | *real* | An array of the same size as $S$ containing the real parts of the computed eigenvalues. |
| out | *imag* | An array of the same size as $S$ containing the imaginary parts of the computed eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_SEP_SM_ReorderSchur
starneig_SEP_SM_Select
starneig_reorder_conf
starneig_reorder_init_conf

#### 11.4.2.10 starneig_SEP_SM_Eigenvectors_expert()

```
starneig_error_t starneig_SEP_SM_Eigenvectors_expert (
            struct starneig_eigenvectors_conf * conf,
            int n,
            int selected[ ],
            double S[ ],
            int ldS,
            double Q[ ],
            int ldQ,
            double X[ ],
            int ldX )
```

Computes an eigenvector for each selected eigenvalue.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in | *n* | The order of $S$ and $Q$ and the number of rows of $X$. |
| in | *selected* | The selection array specifying the locations of the selected eigenvalues. The number of 1's in the array is the same as the number of columns in $X$. |
| in | *S* | The Schur matrix $S$. |
| in | *ldS* | The leading dimension of $S$. |
| in | *Q* | The orthogonal matrix $Q$. |
| in | *ldQ* | The The leading dimension of $Q$. |
| out | *X* | A matrix with $n$ rows and one column for each selected eigenvalue. The columns represent the computed eigenvectors as previously described. |
| in | *ldX* | The leading dimension of $X$. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_SEP_SM_Select

## 11.5 Shared Memory / Generalized EVP

Functions for solving non-symmetric generalized eigenvalue problems on shared memory systems.

### Computational functions

- starneig_error_t starneig_GEP_SM_HessenbergTriangular (int n, double A[ ], int ldA, double B[ ], int ldB, double Q[ ], int ldQ, double Z[ ], int ldZ)

  *Computes a Hessenberg-triangular decomposition of a general matrix pencil.*
- starneig_error_t starneig_GEP_SM_Schur (int n, double H[ ], int ldH, double T[ ], int ldT, double Q[ ], int ldQ, double Z[ ], int ldZ, double real[ ], double imag[ ], double beta[ ])

  *Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.*
- starneig_error_t starneig_GEP_SM_ReorderSchur (int n, int selected[ ], double S[ ], int ldS, double T[ ], int ldT, double Q[ ], int ldQ, double Z[ ], int ldZ, double real[ ], double imag[ ], double beta[ ])

  *Reorders selected generalized eigenvalues to the top left corner of a generalized Schur decomposition.*
- starneig_error_t starneig_GEP_SM_Reduce (int n, double A[ ], int ldA, double B[ ], int ldB, double Q[ ], int ldQ, double Z[ ], int ldZ, double real[ ], double imag[ ], double beta[ ], int(∗predicate)(double real, double imag, double beta, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)

  *Computes a (reordered) generalized Schur decomposition given a general matrix pencil.*
- starneig_error_t starneig_GEP_SM_Eigenvectors (int n, int selected[ ], double S[ ], int ldS, double T[ ], int ldT, double Z[ ], int ldZ, double X[ ], int ldX)

  *Computes a generalized eigenvector for each selected generalized eigenvalue.*

### Helper functions

- starneig_error_t starneig_GEP_SM_Select (int n, double S[ ], int ldS, double T[ ], int ldT, int(∗predicate)(double real, double imag, double beta, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)

  *Generates a selection array for a Schur-triangular matrix pencil using a user-supplied predicate function.*

### Expert computational functions

- starneig_error_t starneig_GEP_SM_Schur_expert (struct starneig_schur_conf ∗conf, int n, double H[ ], int ldH, double T[ ], int ldT, double Q[ ], int ldQ, double Z[ ], int ldZ, double real[ ], double imag[ ], double beta[ ])

  *Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.*
- starneig_error_t starneig_GEP_SM_ReorderSchur_expert (struct starneig_reorder_conf ∗conf, int n, int selected[ ], double S[ ], int ldS, double T[ ], int ldT, double Q[ ], int ldQ, double Z[ ], int ldZ, double real[ ], double imag[ ], double beta[ ])

  *Reorders selected eigenvalues to the top left corner of a generalized Schur decomposition.*
- starneig_error_t starneig_GEP_SM_Eigenvectors_expert (struct starneig_eigenvectors_conf ∗conf, int n, int selected[ ], double S[ ], int ldS, double T[ ], int ldT, double Z[ ], int ldZ, double X[ ], int ldX)

  *Computes a generalized eigenvector for each selected generalized eigenvalue.*

### 11.5.1 Detailed Description

Functions for solving non-symmetric generalized eigenvalue problems on shared memory systems.

### 11.5.2 Function Documentation

#### 11.5.2.1 starneig_GEP_SM_HessenbergTriangular()

```
starneig_error_t starneig_GEP_SM_HessenbergTriangular (
            int n,
            double A[ ],
            int ldA,
            double B[ ],
            int ldB,
            double Q[ ],
            int ldQ,
            double Z[ ],
            int ldZ )
```

Computes a Hessenberg-triangular decomposition of a general matrix pencil.

**Remarks**

This function is a wrapper for several LAPACK subroutines.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $A$, $B$, $Q$ and $Z$. |
| in,out | *A* | On entry, the general matrix $A$. On exit, the upper Hessenberg matrix $H$. |
| in | *ldA* | The leading dimension of $A$. |
| in,out | *B* | On entry, the general matrix $B$. On exit, the upper triangular matrix $T$. |
| in | *ldB* | The leading dimension of $B$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| in | *ldQ* | The leading dimension of $Q$. |
| in,out | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| in | *ldZ* | The leading dimension of $Z$. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**Examples:**

gep_sm_full_chain.c.

#### 11.5.2.2 starneig_GEP_SM_Schur()

```
starneig_error_t starneig_GEP_SM_Schur (
            int n,
            double H[ ],
            int ldH,
            double T[ ],
            int ldT,
            double Q[ ],
            int ldQ,
```

```
            double Z[],
            int ldZ,
            double real[],
            double imag[],
            double beta[] )
```

Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $H$, $T$, $Q$ and $Z$. |
| in,out | *H* | On entry, the upper Hessenberg matrix $H$. On exit, the Schur matrix $S$. |
| in | *ldH* | The leading dimension of $H$. |
| in,out | *T* | On entry, the upper triangular matrix $T$. On exit, the upper triangular matrix $\hat{T}$. |
| in | *ldT* | The leading dimension of $T$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| in | *ldQ* | The leading dimension of $Q$. |
| in,out | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| in | *ldZ* | The leading dimension of $Z$. |
| out | *real* | An array of the same size as $H$ containing the real parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *imag* | An array of the same size as $H$ containing the imaginary parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *beta* | An array of the same size as $H$ containing the $\beta$ values of computed generalized eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_DID_NOT_CONVERGE if the QZ algorithm failed to converge.

**Examples:**

gep_sm_full_chain.c.

#### 11.5.2.3 starneig_GEP_SM_ReorderSchur()

```
starneig_error_t starneig_GEP_SM_ReorderSchur (
            int n,
            int selected[],
            double S[],
            int ldS,
            double T[],
            int ldT,
            double Q[],
            int ldQ,
            double Z[],
            int ldZ,
            double real[],
            double imag[],
            double beta[] )
```

Reorders selected generalized eigenvalues to the top left corner of a generalized Schur decomposition.

**Parameters**

| | | |
|---|---|---|
| `in` | *n* | The order of $H$, $T$, $Q$ and $Z$. |
| `in,out` | *selected* | The selection array. On entry, the initial positions of the selected generalized eigenvalues. On exit, the final positions of all correctly placed selected generalized eigenvalues. In case of failure, the number of 1's in the output may be less than the number of 1's in the input. |
| `in,out` | *S* | On entry, the Schur matrix $S$. On exit, the updated Schur matrix $\hat{S}$. |
| `in` | *ldS* | The leading dimension of $S$. |
| `in,out` | *T* | On entry, the upper triangular $T$. On exit, the updates upper triangular matrix $\hat{T}$. |
| `in` | *ldT* | The leading dimension of $T$. |
| `in,out` | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| `in` | *ldQ* | The leading dimension of $Q$. |
| `in,out` | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| `in` | *ldZ* | The leading dimension of $Z$. |
| `out` | *real* | An array of the same size as $S$ containing the real parts of the $\alpha$ values of the computed generalized eigenvalues. |
| `out` | *imag* | An array of the same size as $S$ containing the imaginary parts of the $\alpha$ values of the computed generalized eigenvalues. |
| `out` | *beta* | An array of the same size as $S$ containing the $\beta$ values of computed generalized eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_PARTIAL_REORDERING if the generalized Schur form is not fully reordered.

**See also**

starneig_GEP_SM_Select

**Examples:**

gep_sm_full_chain.c.

#### 11.5.2.4 starneig_GEP_SM_Reduce()

```
starneig_error_t starneig_GEP_SM_Reduce (
            int n,
            double A[ ],
            int ldA,
            double B[ ],
            int ldB,
            double Q[ ],
            int ldQ,
            double Z[ ],
            int ldZ,
            double real[ ],
            double imag[ ],
```

```
double beta[],
int(*)(double real, double imag, double beta, void *arg) predicate,
void * arg,
int selected[],
int * num_selected )
```

Computes a (reordered) generalized Schur decomposition given a general matrix pencil.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $A$, $B$, $Q$ and $Z$. |
| in,out | *A* | On entry, the general matrix $A$. On exit, the Schur matrix $S$. |
| in | *ldA* | The leading dimension of $A$. |
| in,out | *B* | On entry, the general matrix $B$. On exit, the upper triangular matrix $T$. |
| in | *ldB* | The leading dimension of $B$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| in | *ldQ* | The leading dimension of $Q$. |
| in,out | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| in | *ldZ* | The leading dimension of $Z$. |
| out | *real* | An array of the same size as $A$ containing the real parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *imag* | An array of the same size as $A$ containing the imaginary parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *beta* | An array of the same size as $A$ containing the $\beta$ values of computed generalized eigenvalues. |
| in | *predicate* | A function that takes a (complex) generalized eigenvalue as input and returns non-zero if it should be selected. For complex conjugate pairs of generalized eigenvalues, the predicate is called only for the generalized eigenvalue with positive imaginary part and the corresponding $2 \times 2$ block is either selected or deselected. The reordering step is skipped if the argument is a NULL pointer. |
| in | *arg* | An optional argument for the predicate function. |
| out | *selected* | The final positions of all correctly placed selected generalized eigenvalues. |
| out | *num_selected* | The number of selected generalized eigenvalues (a complex conjugate pair is counted as two selected generalized eigenvalues). |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_DID_NOT_CONVERGE if the QZ algorithm failed to converge. STARNEIG_PARTIAL_REORDERING if the generalized Schur form is not fully reordered.

**Examples:**

gep_sm_eigenvectors.c.

### 11.5.2.5 starneig_GEP_SM_Eigenvectors()

```
starneig_error_t starneig_GEP_SM_Eigenvectors (
            int n,
            int selected[ ],
            double S[ ],
            int ldS,
            double T[ ],
            int ldT,
            double Z[ ],
            int ldZ,
            double X[ ],
            int ldX )
```

Computes a generalized eigenvector for each selected generalized eigenvalue.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $S$ and $Q$ and the number of rows of $X$. |
| in | *selected* | The selection array specifying the locations of the selected generalized eigenvalues. The number of 1's in the array is the same as the number of columns in $X$. |
| in | *S* | The Schur matrix $S$. |
| in | *ldS* | The leading dimension of $S$. |
| in | *T* | The upper triangular matrix $T$. |
| in | *ldT* | The leading dimension of $T$. |
| in | *Z* | The orthogonal matrix $Z$. |
| in | *ldZ* | The leading dimension of $Z$. |
| out | *X* | A matrix with $n$ rows and one column for each selected generalized eigenvalue. The columns represent the computed generalized eigenvectors as previously described. |
| in | *ldX* | The leading dimension of $X$. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_GEP_SM_Select

**Examples:**

gep_sm_eigenvectors.c.

#### 11.5.2.6 starneig_GEP_SM_Select()

```
starneig_error_t starneig_GEP_SM_Select (
            int n,
            double S[ ],
            int ldS,
            double T[ ],
            int ldT,
            int(*)(double real, double imag, double beta, void *arg) predicate,
            void * arg,
            int selected[ ],
            int * num_selected )
```

Generates a selection array for a Schur-triangular matrix pencil using a user-supplied predicate function.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The order of $S$ and $T$. |
| in | *S* | The Schur matrix $S$. |
| in | *ldS* | The leading dimension of $S$. |
| in | *T* | The upper triangular matrix $T$. |
| in | *ldT* | The leading dimension of $T$. |

**Parameters**

| | | |
|---|---|---|
| in | *predicate* | A function that takes a (complex) generalized eigenvalue as input and returns non-zero if it should be selected. For complex conjugate pairs of generalized eigenvalues, the predicate is called only for the generallized eigenvalue with positive imaginary part and the corresponding $2 \times 2$ block is either selected or deselected. |
| in | *arg* | An optional argument for the predicate function. |
| out | *selected* | The selection array. Both elements of a selected complex conjugate pair are set to 1. |
| out | *num_selected* | The number of selected generalized eigenvalues (a complex conjugate pair is counted as two selected generalized eigenvalues). |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**Examples:**

gep_sm_eigenvectors.c, and gep_sm_full_chain.c.

#### 11.5.2.7 starneig_GEP_SM_Schur_expert()

```
starneig_error_t starneig_GEP_SM_Schur_expert (
            struct starneig_schur_conf * conf,
            int n,
            double H[ ],
            int ldH,
            double T[ ],
            int ldT,
            double Q[ ],
            int ldQ,
            double Z[ ],
            int ldZ,
            double real[ ],
            double imag[ ],
            double beta[ ] )
```

Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in | *n* | The order of $H$, $T$, $Q$ and $Z$. |
| in,out | *H* | On entry, the upper Hessenberg matrix $H$. On exit, the Schur matrix $S$. |
| in | *ldH* | The leading dimension of $H$. |
| in,out | *T* | On entry, the upper triangular matrix $T$. On exit, the upper triangular matrix $\hat{T}$. |
| in | *ldT* | The leading dimension of $T$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| in | *ldQ* | The leading dimension of $Q$. |
| in,out | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| in | *ldZ* | The leading dimension of $Z$. |

**Parameters**

| | | |
|---|---|---|
| out | *real* | An array of the same size as $H$ containing the real parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *imag* | An array of the same size as $H$ containing the imaginary parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *beta* | An array of the same size as $H$ containing the $\beta$ values of computed generalized eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_GEP_SM_Schur
starneig_schur_conf
starneig_schur_init_conf

#### 11.5.2.8 starneig_GEP_SM_ReorderSchur_expert()

```
starneig_error_t starneig_GEP_SM_ReorderSchur_expert (
            struct starneig_reorder_conf * conf,
            int n,
            int selected[ ],
            double S[ ],
            int ldS,
            double T[ ],
            int ldT,
            double Q[ ],
            int ldQ,
            double Z[ ],
            int ldZ,
            double real[ ],
            double imag[ ],
            double beta[ ] )
```

Reorders selected eigenvalues to the top left corner of a generalized Schur decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in | *n* | The order of $H$, $T$, $Q$ and $Z$. |
| in,out | *selected* | The selection array. |
| in,out | *S* | On entry, the Schur matrix $S$. On exit, the updated Schur matrix $\hat{S}$. |
| in | *ldS* | The leading dimension of $S$. |
| in,out | *T* | On entry, the upper triangular $T$. On exit, the updates upper triangular matrix $\hat{T}$. |
| in | *ldT* | The leading dimension of $T$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |

**Parameters**

| | | |
|---|---|---|
| in | *ldQ* | The leading dimension of $Q$. |
| in,out | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| in | *ldZ* | The leading dimension of $Z$. |
| out | *real* | An array of the same size as $S$ containing the real parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *imag* | An array of the same size as $S$ containing the imaginary parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *beta* | An array of the same size as $S$ containing the $\beta$ values of computed generalized eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_GEP_SM_ReorderSchur
starneig_GEP_SM_Select
starneig_reorder_conf
starneig_reorder_init_conf

#### 11.5.2.9 starneig_GEP_SM_Eigenvectors_expert()

```
starneig_error_t starneig_GEP_SM_Eigenvectors_expert (
            struct starneig_eigenvectors_conf * conf,
            int n,
            int selected[ ],
            double S[ ],
            int ldS,
            double T[ ],
            int ldT,
            double Z[ ],
            int ldZ,
            double X[ ],
            int ldX )
```

Computes a generalized eigenvector for each selected generalized eigenvalue.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in | *n* | The order of $S$ and $Q$ and the number of rows of $X$. |
| in | *selected* | The selection array specifying the locations of the selected generalized eigenvalues. The number of 1's in the array is the same as the number of columns in $X$. |
| in | *S* | The Schur matrix $S$. |
| in | *ldS* | The leading dimension of $S$. |
| in | *T* | The upper triangular matrix $T$. |
| in | *ldT* | The leading dimension of $T$. |

**Parameters**

| | | |
|---|---|---|
| in | *Z* | The orthogonal matrix $Z$. |
| in | *ldZ* | The leading dimension of $Z$. |
| out | *X* | A matrix with $n$ rows and one column for each selected generalized eigenvalue. The columns represent the computed generalized eigenvectors as previously described. |
| in | *ldX* | The leading dimension of $X$. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_GEP_SM_Select

## 11.6 Distributed Memory / Distributed matrices

Data types and functions for distributed matrices.

### Data Structures

- struct starneig_distr_block

  *Distributed block. More...*

### Data distributions

- enum starneig_distr_order_t { STARNEIG_ORDER_DEFAULT, STARNEIG_ORDER_ROW_MAJOR, ST↩ ARNEIG_ORDER_COL_MAJOR }

  *Process mapping order.*
- typedef struct starneig_distr * starneig_distr_t

  *Data distribution.*
- starneig_distr_t starneig_distr_init ()

  *Creates a default data distribution.*
- starneig_distr_t starneig_distr_init_mesh (int rows, int cols, starneig_distr_order_t order)

  *Creates a two-dimensional block cyclic data distribution.*
- starneig_distr_t starneig_distr_init_func (int(*func)(int row, int col, void *arg), void *arg, size_t arg_size)

  *Creates a distribution using a data distribution function.*
- starneig_distr_t starneig_distr_duplicate (starneig_distr_t distr)

  *Duplicates a data distribution.*
- void starneig_distr_destroy (starneig_distr_t distr)

  *Destroys a data distribution.*

### Distributed matrices

- enum starneig_datatype_t { STARNEIG_REAL_DOUBLE }

  *Distributed matrix element data type.*
- typedef struct starneig_distr_matrix * starneig_distr_matrix_t

  *Distributed matrix.*
- starneig_distr_matrix_t starneig_distr_matrix_create (int rows, int cols, int row_blksz, int col_blksz, starneig↩ _datatype_t type, starneig_distr_t distr)

  *Creates a distributed matrix with uninitialized matrix elements.*
- starneig_distr_matrix_t starneig_distr_matrix_create_local (int rows, int cols, starneig_datatype_t type, int owner, double *A, int ldA)

  *Creates a single-owner distributed matrix from a local matrix.*
- void starneig_distr_matrix_destroy (starneig_distr_matrix_t matrix)

  *Destroys a distributed matrix.*
- void starneig_distr_matrix_copy (starneig_distr_matrix_t source, starneig_distr_matrix_t dest)

  *Copies the contents of a distributed matrix to a second distributed matrix.*
- void starneig_distr_matrix_copy_region (int sr, int sc, int dr, int dc, int rows, int cols, starneig_distr_matrix_t source, starneig_distr_matrix_t dest)

  *Copies region of a distributed matrix to a second distributed matrix.*

#### Query functions

- void starneig_distr_matrix_get_blocks (starneig_distr_matrix_t matrix, struct starneig_distr_block **blocks, int *num_blocks)

  *Returns the locally owned distributed blocks.*
- starneig_distr_t starneig_distr_matrix_get_distr (starneig_distr_matrix_t matrix)

  *Returns the distribution that is associated with a distributed matrix.*
- starneig_datatype_t starneig_distr_matrix_get_datatype (starneig_distr_matrix_t matrix)

  *Returns the matrix element data type.*
- size_t starneig_distr_matrix_get_elemsize (starneig_distr_matrix_t matrix)

  *Returns the matrix element size.*
- int starneig_distr_matrix_get_rows (starneig_distr_matrix_t matrix)

  *Returns the number of (global) rows.*
- int starneig_distr_matrix_get_cols (starneig_distr_matrix_t matrix)

  *Returns the number of (global) columns.*
- int starneig_distr_matrix_get_row_blksz (starneig_distr_matrix_t matrix)

  *Returns the number of rows in a distribution block.*
- int starneig_distr_matrix_get_col_blksz (starneig_distr_matrix_t matrix)

  *Returns the number of columns in a distribution block.*

#### Helpers

- void starneig_broadcast (int root, size_t size, void *buffer)

  *Broadcast a buffer.*

### 11.6.1 Detailed Description

Data types and functions for distributed matrices.

### 11.6.2 Data Structure Documentation

#### 11.6.2.1 struct starneig_distr_block

Distributed block.

#### Data Fields

| | | |
|---|---|---|
| int | row_blksz | The number of rows in the block. |
| int | col_blksz | The number of columns in the block. |
| int | glo_row | The topmost global row that belong to the block. |
| int | glo_col | The leftmost global column that belong to the block. |
| int | ld | The leading dimension of the local array. |
| void * | ptr | A pointer to the local array. |

### 11.6.3 Enumeration Type Documentation

#### 11.6.3.1 starneig_distr_order_t

```
enum starneig_distr_order_t
```

Process mapping order.

**Enumerator**

| | |
|---|---|
| STARNEIG_ORDER_DEFAULT | Default ordering. |
| STARNEIG_ORDER_ROW_MAJOR | Row-major natural ordering. |
| STARNEIG_ORDER_COL_MAJOR | Column-major natural ordering. |

#### 11.6.3.2 starneig_datatype_t

```
enum starneig_datatype_t
```

Distributed matrix element data type.

**Enumerator**

| | |
|---|---|
| STARNEIG_REAL_DOUBLE | Double precision real numbers. |

### 11.6.4 Function Documentation

#### 11.6.4.1 starneig_distr_init()

```
starneig_distr_t starneig_distr_init ( )
```

Creates a default data distribution.

**Returns**

A new data distribution.

#### 11.6.4.2 starneig_distr_init_mesh()

```
starneig_distr_t starneig_distr_init_mesh (
              int rows,
              int cols,
              starneig_distr_order_t order )
```

Creates a two-dimensional block cyclic data distribution.

**Parameters**

| | | |
|---|---|---|
| in | *rows* | The number of rows in the mesh. Can be set to -1 in which case the library decides the value. |
| in | *cols* | The number of columns in the mesh. Can be set to -1 in which case the library decides the value. |
| in | *order* | The process mapping order. |

**Returns**

A new data distribution.

**Examples:**

gep_dm_full_chain.c, and sep_dm_full_chain.c.

#### 11.6.4.3 starneig_distr_init_func()

```
starneig_distr_t starneig_distr_init_func (
            int(*)(int row, int col, void *arg) func,
            void * arg,
            size_t arg_size )
```

Creates a distribution using a data distribution function.

The distribution function maps each block to it's owner. The function takes three arguments: block's row index, blocks's column index and an optional user defined argument.

```
struct block_cyclic_arg {
    int rows;
    int cols;
};

int block_cyclic_func(int i, int j, void *arg)
{
    struct block_cyclic_arg *mesh = (struct block_cyclic_arg *) arg;
    return (i % mesh->rows) * mesh->cols + j % mesh->cols;
}

void func(...)
{
    ...

    // create a custom two-dimensional block cyclic distribution with 4 rows
    // and 6 columns in the mesh
    struct block_cyclic_arg arg = { .rows = 4, .cols = 6 };
    starneig_distr_t distr =
        starneig_distr_init_func(&block_cyclic_func, &arg, sizeof(arg));

    ...
}
```

**Parameters**

| | | |
|---|---|---|
| in | *func* | The data distribution function. |
| in | *arg* | An optional data distribution function argument. |
| in | *arg_size* | The size of the optional data distribution function argument. |

**Returns**

A new data distribution.

#### 11.6.4.4 starneig_distr_duplicate()

```
starneig_distr_t starneig_distr_duplicate (
            starneig_distr_t distr )
```

Duplicates a data distribution.

**Parameters**

| | | |
|---|---|---|
| in | *distr* | The data distribution to be duplicated. |

**Returns**

A duplicated data distribution.

#### 11.6.4.5 starneig_distr_destroy()

```
void starneig_distr_destroy (
            starneig_distr_t distr )
```

Destroys a data distribution.

**Parameters**

| | | |
|---|---|---|
| in,out | *distr* | The data distribution to be destroyed. |

**Examples:**

gep_dm_full_chain.c, and sep_dm_full_chain.c.

#### 11.6.4.6 starneig_distr_matrix_create()

```
starneig_distr_matrix_t starneig_distr_matrix_create (
            int rows,
            int cols,
            int row_blksz,
            int col_blksz,
            starneig_datatype_t type,
            starneig_distr_t distr )
```

Creates a distributed matrix with uninitialized matrix elements.

```c
// create a m X n double-precision real matrix that is distributed in a
// two-dimensional block cyclic fashion in bm X bn blocks
starneig_distr_t distr = starneig_distr_init();
starneig_distr_matrix_t dA =
    starneig_distr_matrix_create(m, n, bm, bn,
      STARNEIG_REAL_DOUBLE, distr);
```

**Attention**

StarNEig library is designed to use much larger distributed blocks than ScaLAPACK. Selecting a too small distributed block size will be detrimental to the performance.

**Parameters**

| | | |
|---|---|---|
| in | *rows* | The number of (global) rows in the matrix. |
| in | *cols* | The number of (global) columns in the matrix. |
| in | *row_blksz* | The number of rows in a distribution block. Can be set to -1 in which case the library decides the value. |
| in | *col_blksz* | The number of columns in a distribution block. Can be set to -1 in which case the library decides the value. |
| in | *type* | The matrix element data type. |
| in | *distr* | The data distribution. Can be left to NULL in which case the library decides the distribution. |

**Returns**

A new distributed matrix.

**Examples:**

gep_dm_full_chain.c, and sep_dm_full_chain.c.

#### 11.6.4.7 starneig_distr_matrix_create_local()

```c
starneig_distr_matrix_t starneig_distr_matrix_create_local (
            int rows,
            int cols,
            starneig_datatype_t type,
            int owner,
            double * A,
            int ldA )
```

Creates a single-owner distributed matrix from a local matrix.

This creates a wrapper. The contents of the local matrix may be modified by the functions that use the wrapper. The starneig_distr_matrix_destroy() function does not free the local matrix.

```c
int m = 1000, n = 1000;
double *A = NULL; size_t ldA = 0;

// rank 3 initialized the local matrix
if (my_rank = 3) {
    A = initialize_matrix(m, n, &ldA);
}

// all ranks initialize the distributed matrix
starneig_distr_matrix_t lA =
      starneig_distr_matrix_create_local(
    m, n, STARNEIG_REAL_DOUBLE, 3, A, ldA);
```

**Parameters**

| | | |
|---|---|---|
| in | *rows* | The number of rows in the matrix. |
| in | *cols* | The number of columns in the matrix. |
| in | *type* | Matrix element data type. |
| in | *owner* | MPI rank that owns the distributed matrix. |
| in | *A* | A pointer to the local matrix. This argument is ignored the calling rank is not the same as the owner. |
| in | *ldA* | The leading dimension of the local matrix. This argument is ignored the calling rank is not the same as the owner. |

**Returns**

A new distributed matrix.

**Examples:**

gep_dm_full_chain.c, and sep_dm_full_chain.c.

#### 11.6.4.8 starneig_distr_matrix_destroy()

```
void starneig_distr_matrix_destroy (
            starneig_distr_matrix_t matrix )
```

Destroys a distributed matrix.

**Parameters**

| | | |
|---|---|---|
| in,out | *matrix* | The distributed matrix to be destroyed. |

**Examples:**

gep_dm_full_chain.c, and sep_dm_full_chain.c.

#### 11.6.4.9 starneig_distr_matrix_copy()

```
void starneig_distr_matrix_copy (
            starneig_distr_matrix_t source,
            starneig_distr_matrix_t dest )
```

Copies the contents of a distributed matrix to a second distributed matrix.

**Parameters**

| | | |
|---|---|---|
| in | *source* | The source matrix. |
| out | *dest* | The destination matrix. |

**Examples:**

gep_dm_full_chain.c, and sep_dm_full_chain.c.

#### 11.6.4.10 starneig_distr_matrix_copy_region()

```
void starneig_distr_matrix_copy_region (
            int sr,
            int sc,
            int dr,
            int dc,
            int rows,
            int cols,
            starneig_distr_matrix_t source,
            starneig_distr_matrix_t dest )
```

Copies region of a distributed matrix to a second distributed matrix.

**Parameters**

| | | |
|---|---|---|
| in | *sr* | The first source matrix row to be copied. |
| in | *sc* | The first source matrix column to be copied. |
| in | *dr* | The first destination matrix row. |
| in | *dc* | The first destination matrix column. |
| in | *rows* | The number of rows to copy. |
| in | *cols* | The number of columns to copy. |
| in | *source* | The source matrix. |
| out | *dest* | The destination matrix. |

#### 11.6.4.11 starneig_distr_matrix_get_blocks()

```
void starneig_distr_matrix_get_blocks (
            starneig_distr_matrix_t matrix,
            struct starneig_distr_block ** blocks,
            int * num_blocks )
```

Returns the locally owned distributed blocks.

**Attention**

A user is allowed to modify the contents of the locally owned blocks but the the returned array itself should not be modified.

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |
| out | *blocks* | An array that contains all locally owned distributed blocks. |
| out | *num_blocks* | The total number of locally owned distributed blocks. |

#### 11.6.4.12 starneig_distr_matrix_get_distr()

```
starneig_distr_t starneig_distr_matrix_get_distr (
            starneig_distr_matrix_t matrix )
```

Returns the distribution that is associated with a distributed matrix.

**Attention**

The distributed matrix maintains the ownership of the returned data distribution. A user must duplicate the data distribution if necessary.

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |

**Returns**

The associated distribution.

#### 11.6.4.13 starneig_distr_matrix_get_datatype()

```
starneig_datatype_t starneig_distr_matrix_get_datatype (
            starneig_distr_matrix_t matrix )
```

Returns the matrix element data type.

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |

**Returns**

The matrix element data type.

#### 11.6.4.14 starneig_distr_matrix_get_elemsize()

```
size_t starneig_distr_matrix_get_elemsize (
            starneig_distr_matrix_t matrix )
```

Returns the matrix element size.

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |

**Returns**

The matrix element size.

#### 11.6.4.15 starneig_distr_matrix_get_rows()

```
int starneig_distr_matrix_get_rows (
            starneig_distr_matrix_t matrix )
```

Returns the number of (global) rows.

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |

**Returns**

The number of (global) rows.

#### 11.6.4.16 starneig_distr_matrix_get_cols()

```
int starneig_distr_matrix_get_cols (
            starneig_distr_matrix_t matrix )
```

Returns the number of (global) columns.

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |

**Returns**

The number of (global) columns.

#### 11.6.4.17 starneig_distr_matrix_get_row_blksz()

```
int starneig_distr_matrix_get_row_blksz (
            starneig_distr_matrix_t matrix )
```

Returns the number of rows in a distribution block.

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |

**Returns**

The number of rows in a distribution block.

#### 11.6.4.18 starneig_distr_matrix_get_col_blksz()

```
int starneig_distr_matrix_get_col_blksz (
            starneig_distr_matrix_t matrix )
```

Returns the number of columns in a distribution block.

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |

**Returns**

The number of columns in a distribution block.

#### 11.6.4.19 starneig_broadcast()

```
void starneig_broadcast (
            int root,
            size_t size,
            void * buffer )
```

Broadcast a buffer.

**Parameters**

| | | |
|---|---|---|
| in | *root* | The rank that is going to broadcast the buffer. |
| in | *size* | The size of the buffer. |
| in,out | *buffer* | A pointer to the buffer. |

## 11.7 Distributed Memory / Standard EVP

Functions for solving non-symmetric standard eigenvalue problems on distributed memory systems.

### Computational functions

- starneig_error_t starneig_SEP_DM_Hessenberg (starneig_distr_matrix_t A, starneig_distr_matrix_t Q)

  *Computes a Hessenberg decomposition of a general matrix.*
- starneig_error_t starneig_SEP_DM_Schur (starneig_distr_matrix_t H, starneig_distr_matrix_t Q, double real[ ], double imag[ ])

  *Computes a Schur decomposition given a Hessenberg decomposition.*
- starneig_error_t starneig_SEP_DM_ReorderSchur (int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t Q, double real[ ], double imag[ ])

  *Reorders selected eigenvalues to the top left corner of a Schur decomposition.*
- starneig_error_t starneig_SEP_DM_Reduce (starneig_distr_matrix_t A, starneig_distr_matrix_t Q, double real[ ], double imag[ ], int(*predicate)(double real, double imag, void *arg), void *arg, int selected[ ], int *num_selected)

  *Computes a (reordered) Schur decomposition of a general matrix.*
- starneig_error_t starneig_SEP_DM_Eigenvectors (int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t Q, starneig_distr_matrix_t X)

  *Computes an eigenvector for each selected eigenvalue.*

### Helper functions

- starneig_error_t starneig_SEP_DM_Select (starneig_distr_matrix_t S, int(*predicate)(double real, double imag, void *arg), void *arg, int selected[ ], int *num_selected)

  *Generates a selection array for a Schur matrix using a user-supplied predicate function.*

### Expert computational functions

- starneig_error_t starneig_SEP_DM_Schur_expert (struct starneig_schur_conf *conf, starneig_distr_matrix_t H, starneig_distr_matrix_t Q, double real[ ], double imag[ ])

  *Computes a Schur decomposition given a Hessenberg decomposition.*
- starneig_error_t starneig_SEP_DM_ReorderSchur_expert (struct starneig_reorder_conf *conf, int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t Q, double real[ ], double imag[ ])

  *Reorders selected eigenvalues to the top left corner of a Schur decomposition.*
- starneig_error_t starneig_SEP_DM_Eigenvectors_expert (struct starneig_eigenvectors_conf *conf, int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t Q, starneig_distr_matrix_t X)

  *Computes an eigenvector for each selected eigenvalue.*

### 11.7.1 Detailed Description

Functions for solving non-symmetric standard eigenvalue problems on distributed memory systems.

### 11.7.2 Function Documentation

#### 11.7.2.1 starneig_SEP_DM_Hessenberg()

```
starneig_error_t starneig_SEP_DM_Hessenberg (
            starneig_distr_matrix_t A,
            starneig_distr_matrix_t Q )
```

Computes a Hessenberg decomposition of a general matrix.

**Attention**

This function is a wrapper for several ScaLAPACK subroutines. The function exists if STARNEIG_SEP_DM_HESSENBERG is defined.

**Parameters**

| | | |
|---|---|---|
| in,out | *A* | On entry, the general matrix $A$. On exit, the upper Hessenberg matrix $H$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**Examples:**

sep_dm_full_chain.c.

#### 11.7.2.2 starneig_SEP_DM_Schur()

```
starneig_error_t starneig_SEP_DM_Schur (
            starneig_distr_matrix_t H,
            starneig_distr_matrix_t Q,
            double real[ ],
            double imag[ ] )
```

Computes a Schur decomposition given a Hessenberg decomposition.

**Parameters**

| | | |
|---|---|---|
| in,out | *H* | On entry, the upper Hessenberg matrix $H$. On exit, the Schur matrix $S$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| out | *real* | An array of the same size as $H$ containing the real parts of the computed eigenvalues. |
| out | *imag* | An array of the same size as $H$ containing the imaginary parts of the computed eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_DID_NOT_CONVERGE if the QR algorithm failed to converge.

**Examples:**

sep_dm_full_chain.c.

#### 11.7.2.3 starneig_SEP_DM_ReorderSchur()

```
starneig_error_t starneig_SEP_DM_ReorderSchur (
            int selected[],
            starneig_distr_matrix_t S,
            starneig_distr_matrix_t Q,
            double real[],
            double imag[] )
```

Reorders selected eigenvalues to the top left corner of a Schur decomposition.

**Parameters**

| | | |
|---|---|---|
| in,out | *selected* | The selection array. On entry, the initial positions of the selected eigenvalues. On exit, the final positions of all correctly placed selected eigenvalues. In case of failure, the number of 1's in the output may be less than the number of 1's in the input. |
| in,out | *S* | On entry, the Schur matrix $S$. On exit, the updated Schur matrix $\hat{S}$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| out | *real* | An array of the same size as $S$ containing the real parts of the computed eigenvalues. |
| out | *imag* | An array of the same size as $S$ containing the imaginary parts of the computed eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_PARTIAL_REORDERING if the Schur form is not fully reordered.

**See also**

starneig_SEP_DM_Select

**Examples:**

sep_dm_full_chain.c.

#### 11.7.2.4 starneig_SEP_DM_Reduce()

```
starneig_error_t starneig_SEP_DM_Reduce (
            starneig_distr_matrix_t A,
            starneig_distr_matrix_t Q,
            double real[],
            double imag[],
            int(*)(double real, double imag, void *arg) predicate,
            void * arg,
            int selected[],
            int * num_selected )
```

Computes a (reordered) Schur decomposition of a general matrix.

**Attention**

This function uses several ScaLAPACK subroutines. The function exists if STARNEIG_SEP_DM_REDUCE is defined.

**Parameters**

| | | |
|---|---|---|
| in,out | *A* | On entry, the general matrix $A$. On exit, the Schur matrix $S$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| out | *real* | An array of the same size as $A$ containing the real parts of the computed eigenvalues. |
| out | *imag* | An array of the same size as $A$ containing the imaginary parts of the computed eigenvalues. |
| in | *predicate* | A function that takes a (complex) eigenvalue as input and returns non-zero if it should be selected. For complex conjugate pairs of eigenvalues, the predicate is called only for the eigenvalue with positive imaginary part and the corresponding $2 \times 2$ block is either selected or deselected. The reordering step is skipped if the argument is a NULL pointer. |
| in | *arg* | An optional argument for the predicate function. |
| out | *selected* | The final positions of all correctly placed selected eigenvalues. |
| out | *num_selected* | The number of selected eigenvalues (a complex conjugate pair is counted as two selected eigenvalues). |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_DID_NOT_CONVERGE if the QR algorithm failed to converge. STARNEIG_PARTIAL_REORDERING if the Schur form is not fully reordered.

#### 11.7.2.5 starneig_SEP_DM_Eigenvectors()

```
starneig_error_t starneig_SEP_DM_Eigenvectors (
            int selected[ ],
            starneig_distr_matrix_t S,
            starneig_distr_matrix_t Q,
            starneig_distr_matrix_t X )
```

Computes an eigenvector for each selected eigenvalue.

**Parameters**

| | | |
|---|---|---|
| in | *selected* | The selection array specifying the locations of the selected eigenvalues. The number of 1's in the array is the same as the number of columns in $X$. |
| in | *S* | The Schur matrix $S$. |
| in | *Q* | The orthogonal matrix $Q$. |
| out | *X* | A matrix with $n$ rows and one column for each selected eigenvalue. The columns represent the computed eigenvectors as previously described. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_SEP_DM_Select

**Todo** This interface function is not implemented.

#### 11.7.2.6 starneig_SEP_DM_Select()

```
starneig_error_t starneig_SEP_DM_Select (
            starneig_distr_matrix_t S,
            int(*)(double real, double imag, void *arg) predicate,
            void * arg,
            int selected[ ],
            int * num_selected )
```

Generates a selection array for a Schur matrix using a user-supplied predicate function.

**Parameters**

| | | |
|---|---|---|
| in | *S* | The Schur matrix $S$. |
| in | *predicate* | A function that takes a (complex) eigenvalue as input and returns non-zero if it should be selected. For complex conjugate pairs of eigenvalues, the predicate is called only for the eigenvalue with positive imaginary part and the corresponding $2 \times 2$ block is either selected or deselected. |
| in | *arg* | An optional argument for the predicate function. |
| out | *selected* | The selection array. Both elements of a selected complex conjugate pair are set to 1. |
| out | *num_selected* | The (global) number of selected eigenvalues (a complex conjugate pair is counted as two selected eigenvalues). |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**Examples:**

sep_dm_full_chain.c.

#### 11.7.2.7 starneig_SEP_DM_Schur_expert()

```
starneig_error_t starneig_SEP_DM_Schur_expert (
            struct starneig_schur_conf * conf,
            starneig_distr_matrix_t H,
```

```
starneig_distr_matrix_t Q,
double real[],
double imag[] )
```

Computes a Schur decomposition given a Hessenberg decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in,out | *H* | On entry, the upper Hessenberg matrix $H$. On exit, the Schur matrix $S$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| out | *real* | An array of the same size as $H$ containing the real parts of the computed eigenvalues. |
| out | *imag* | An array of the same size as $H$ containing the imaginary parts of the computed eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_SEP_DM_Schur
starneig_schur_conf
starneig_schur_init_conf

#### 11.7.2.8 starneig_SEP_DM_ReorderSchur_expert()

```
starneig_error_t starneig_SEP_DM_ReorderSchur_expert (
            struct starneig_reorder_conf * conf,
            int selected[],
            starneig_distr_matrix_t S,
            starneig_distr_matrix_t Q,
            double real[],
            double imag[] )
```

Reorders selected eigenvalues to the top left corner of a Schur decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in,out | *selected* | The selection array. |
| in,out | *S* | On entry, the Schur matrix $S$. On exit, the updated Schur matrix $\hat{S}$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U$. |
| out | *real* | An array of the same size as $S$ containing the real parts of the computed eigenvalues. |
| out | *imag* | An array of the same size as $S$ containing the imaginary parts of the computed eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_SEP_DM_ReorderSchur
starneig_SEP_DM_Select
starneig_reorder_conf
starneig_reorder_init_conf

#### 11.7.2.9 starneig_SEP_DM_Eigenvectors_expert()

```
starneig_error_t starneig_SEP_DM_Eigenvectors_expert (
            struct starneig_eigenvectors_conf * conf,
            int selected[ ],
            starneig_distr_matrix_t S,
            starneig_distr_matrix_t Q,
            starneig_distr_matrix_t X )
```

Computes an eigenvector for each selected eigenvalue.

**Parameters**

| | | |
|---|---|---|
| in | *selected* | The selection array specifying the locations of the selected eigenvalues. The number of 1's in the array is the same as the number of columns in $X$. |
| in | *S* | The Schur matrix $S$. |
| in | *Q* | The orthogonal matrix $Q$. |
| out | *X* | A matrix with $n$ rows and one column for each selected eigenvalue. The columns represent the computed eigenvectors as previously described. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_SEP_DM_Select

**Todo** This interface function is not implemented.

## 11.8 Distributed Memory / Generalized EVP

Functions for solving non-symmetric generalized eigenvalue problems on distributed memory systems.

### Computational functions

- starneig_error_t starneig_GEP_DM_HessenbergTriangular (starneig_distr_matrix_t A, starneig_distr_matrix_t B, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z)

  *Computes a Hessenberg-triangular decomposition of a general matrix pencil.*
- starneig_error_t starneig_GEP_DM_Schur (starneig_distr_matrix_t H, starneig_distr_matrix_t T, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z, double real[ ], double imag[ ], double beta[ ])

  *Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.*
- starneig_error_t starneig_GEP_DM_ReorderSchur (int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t T, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z, double real[ ], double imag[ ], double beta[ ])

  *Reorders selected generalized eigenvalues to the top left corner of a generalized Schur decomposition.*
- starneig_error_t starneig_GEP_DM_Reduce (starneig_distr_matrix_t A, starneig_distr_matrix_t B, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z, double real[ ], double imag[ ], double beta[ ], int(∗predicate)(double real, double imag, double beta, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)

  *Computes a (reordered) generalized Schur decomposition given a general matrix pencil.*
- starneig_error_t starneig_GEP_DM_Eigenvectors (int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t T, starneig_distr_matrix_t Z, starneig_distr_matrix_t X)

  *Computes a generalized eigenvector for each selected generalized eigenvalue.*

### Helper functions

- starneig_error_t starneig_GEP_DM_Select (starneig_distr_matrix_t S, starneig_distr_matrix_t T, int(∗predicate)(double real, double imag, double beta, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)

  *Generates a selection array for a Schur-triangular matrix pencil using a user-supplied predicate function.*

### Expert computational functions

- starneig_error_t starneig_GEP_DM_Schur_expert (struct starneig_schur_conf ∗conf, starneig_distr_matrix_t H, starneig_distr_matrix_t T, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z, double real[ ], double imag[ ], double beta[ ])

  *Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.*
- starneig_error_t starneig_GEP_DM_ReorderSchur_expert (struct starneig_reorder_conf ∗conf, int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t T, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z, double real[ ], double imag[ ], double beta[ ])

  *Reorders selected generalized eigenvalues to the top left corner of a generalized Schur decomposition.*
- starneig_error_t starneig_GEP_DM_Eigenvectors_expert (struct starneig_eigenvectors_conf ∗conf, int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t T, starneig_distr_matrix_t Z, starneig_distr_matrix_t X)

  *Computes a generalized eigenvector for each selected generalized eigenvalue.*

### 11.8.1 Detailed Description

Functions for solving non-symmetric generalized eigenvalue problems on distributed memory systems.

### 11.8.2 Function Documentation

#### 11.8.2.1 starneig_GEP_DM_HessenbergTriangular()

```
starneig_error_t starneig_GEP_DM_HessenbergTriangular (
            starneig_distr_matrix_t A,
            starneig_distr_matrix_t B,
            starneig_distr_matrix_t Q,
            starneig_distr_matrix_t Z )
```

Computes a Hessenberg-triangular decomposition of a general matrix pencil.

**Attention**

This function is a wrapper for several ScaLAPACK subroutines. The function exists if STARNEIG_GEP_DM_HESSENBERGTRIANGULAR is defined.

**Parameters**

| | | |
|---|---|---|
| in,out | A | On entry, the general matrix $A$. On exit, the upper Hessenberg matrix $H$. |
| in,out | B | On entry, the general matrix $B$. On exit, the upper triangular matrix $T$. |
| in,out | Q | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| in,out | Z | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**Examples:**

gep_dm_full_chain.c.

#### 11.8.2.2 starneig_GEP_DM_Schur()

```
starneig_error_t starneig_GEP_DM_Schur (
            starneig_distr_matrix_t H,
            starneig_distr_matrix_t T,
            starneig_distr_matrix_t Q,
            starneig_distr_matrix_t Z,
            double real[ ],
            double imag[ ],
            double beta[ ] )
```

Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.

**Parameters**

| | | |
|---|---|---|
| in,out | *H* | On entry, the upper Hessenberg matrix $H$. On exit, the Schur matrix $S$. |
| in,out | *T* | On entry, the upper triangular matrix $T$. On exit, the upper triangular matrix $\hat{T}$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| in,out | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| out | *real* | An array of the same size as $H$ containing the real parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *imag* | An array of the same size as $H$ containing the imaginary parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *beta* | An array of the same size as $H$ containing the $\beta$ values of computed generalized eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_DID_NOT_CONVERGE if the QZ algorithm failed to converge.

**Examples:**

gep_dm_full_chain.c.

#### 11.8.2.3 starneig_GEP_DM_ReorderSchur()

```
starneig_error_t starneig_GEP_DM_ReorderSchur (
            int selected[],
            starneig_distr_matrix_t S,
            starneig_distr_matrix_t T,
            starneig_distr_matrix_t Q,
            starneig_distr_matrix_t Z,
            double real[],
            double imag[],
            double beta[] )
```

Reorders selected generalized eigenvalues to the top left corner of a generalized Schur decomposition.

**Parameters**

| | | |
|---|---|---|
| in,out | *selected* | The selection array. On entry, the initial positions of the selected generalized eigenvalues. On exit, the final positions of all correctly placed selected generalized eigenvalues. In case of failure, the number of 1's in the output may be less than the number of 1's in the input. |
| in,out | *S* | On entry, the Schur matrix $S$. On exit, the updated Schur matrix $\hat{S}$. |
| in,out | *T* | On entry, the upper triangular $T$. On exit, the updates upper triangular matrix $\hat{T}$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| in,out | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| out | *real* | An array of the same size as $S$ containing the real parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *imag* | An array of the same size as $S$ containing the imaginary parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *beta* | An array of the same size as $S$ containing the $\beta$ values of computed generalized eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_PARTIAL_REORDERING if the generalized Schur form is not fully reordered.

**See also**

starneig_GEP_DM_Select

**Examples:**

gep_dm_full_chain.c.

#### 11.8.2.4 starneig_GEP_DM_Reduce()

```
starneig_error_t starneig_GEP_DM_Reduce (
              starneig_distr_matrix_t A,
              starneig_distr_matrix_t B,
              starneig_distr_matrix_t Q,
              starneig_distr_matrix_t Z,
              double real[],
              double imag[],
              double beta[],
              int(*)(double real, double imag, double beta, void *arg) predicate,
              void * arg,
              int selected[],
              int * num_selected )
```

Computes a (reordered) generalized Schur decomposition given a general matrix pencil.

**Attention**

This function uses several ScaLAPACK subroutines. The function exists if STARNEIG_GEP_DM_REDUCE is defined.

**Parameters**

| | | |
|---|---|---|
| in,out | *A* | On entry, the general matrix $A$. On exit, the Schur matrix $S$. |
| in,out | *B* | On entry, the general matrix $B$. On exit, the upper triangular matrix $T$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| in,out | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| out | *real* | An array of the same size as $A$ containing the real parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *imag* | An array of the same size as $A$ containing the imaginary parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *beta* | An array of the same size as $A$ containing the $\beta$ values of computed generalized eigenvalues. |
| in | *predicate* | A function that takes a (complex) generalized eigenvalue as input and returns non-zero if it should be selected. For complex conjugate pairs of generalized eigenvalues, the predicate is called only for the generalized eigenvalue with positive imaginary part and the corresponding $2 \times 2$ block is either selected or deselected. The reordering step is skipped if the argument is a NULL pointer. |
| in | *arg* | An optional argument for the predicate function. |
| out | *selected* | The final positions of all correctly placed selected generalized eigenvalues. |
| out | *num_selected* | The number of selected generalized eigenvalues (a complex conjugate pair is counted as two selected generalized eigenvalues). |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise. STARNEIG_DID_NOT_CONVERGE if the QZ algorithm failed to converge. STARNEIG_PARTIAL_REORDERING if the generalized Schur form is not fully reordered.

#### 11.8.2.5 starneig_GEP_DM_Eigenvectors()

```
starneig_error_t starneig_GEP_DM_Eigenvectors (
            int selected[ ],
            starneig_distr_matrix_t S,
            starneig_distr_matrix_t T,
            starneig_distr_matrix_t Z,
            starneig_distr_matrix_t X )
```

Computes a generalized eigenvector for each selected generalized eigenvalue.

**Parameters**

| | | |
|---|---|---|
| in | *selected* | The selection array specifying the locations of the selected generalized eigenvalues. The number of 1's in the array is the same as the number of columns in $X$. |
| in | *S* | The Schur matrix $S$. |
| in | *T* | The upper triangular matrix $T$. |
| in | *Z* | The orthogonal matrix $Z$. |
| out | *X* | A matrix with $n$ rows and one column for each selected generalized eigenvalue. The columns represent the computed generalized eigenvectors as previously described. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_GEP_DM_Select

**Todo** This interface function is not implemented.

#### 11.8.2.6 starneig_GEP_DM_Select()

```
starneig_error_t starneig_GEP_DM_Select (
            starneig_distr_matrix_t S,
            starneig_distr_matrix_t T,
            int(*)(double real, double imag, double beta, void *arg) predicate,
            void * arg,
            int selected[ ],
            int * num_selected )
```

Generates a selection array for a Schur-triangular matrix pencil using a user-supplied predicate function.

**Parameters**

| | | |
|---|---|---|
| in | *S* | The Schur matrix $S$. |
| in | *T* | The upper triangular matrix $T$. |
| in | *predicate* | A function that takes a (complex) generalized eigenvalue as input and returns non-zero if it should be selected. For complex conjugate pairs of generalized eigenvalues, the predicate is called only for the generallized eigenvalue with positive imaginary part and the corresponding $2 \times 2$ block is either selected or deselected. |
| in | *arg* | An optional argument for the predicate function. |
| out | *selected* | The selection array. Both elements of a selected complex conjugate pair are set to 1. |
| out | *num_selected* | The number of selected generalized eigenvalues (a complex conjugate pair is counted as two selected generalized eigenvalues). |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**Examples:**

gep_dm_full_chain.c.

#### 11.8.2.7 starneig_GEP_DM_Schur_expert()

```
starneig_error_t starneig_GEP_DM_Schur_expert (
            struct starneig_schur_conf * conf,
            starneig_distr_matrix_t H,
            starneig_distr_matrix_t T,
            starneig_distr_matrix_t Q,
            starneig_distr_matrix_t Z,
            double real[ ],
            double imag[ ],
            double beta[ ] )
```

Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in,out | *H* | On entry, the upper Hessenberg matrix $H$. On exit, the Schur matrix $S$. |
| in,out | *T* | On entry, the upper triangular matrix $T$. On exit, the upper triangular matrix $\hat{T}$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| in,out | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| out | *real* | An array of the same size as $H$ containing the real parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *imag* | An array of the same size as $H$ containing the imaginary parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *beta* | An array of the same size as $H$ containing the $\beta$ values of computed generalized eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_GEP_DM_Schur
starneig_schur_conf
starneig_schur_init_conf

#### 11.8.2.8 starneig_GEP_DM_ReorderSchur_expert()

```
starneig_error_t starneig_GEP_DM_ReorderSchur_expert (
            struct starneig_reorder_conf * conf,
            int selected[ ],
            starneig_distr_matrix_t S,
            starneig_distr_matrix_t T,
            starneig_distr_matrix_t Q,
            starneig_distr_matrix_t Z,
            double real[ ],
            double imag[ ],
            double beta[ ] )
```

Reorders selected generalized eigenvalues to the top left corner of a generalized Schur decomposition.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in,out | *selected* | The selection array. |
| in,out | *S* | On entry, the Schur matrix $S$. On exit, the updated Schur matrix $\hat{S}$. |
| in,out | *T* | On entry, the upper triangular $T$. On exit, the updates upper triangular matrix $\hat{T}$. |
| in,out | *Q* | On entry, the orthogonal matrix $Q$. On exit, the product matrix $Q * U_1$. |
| in,out | *Z* | On entry, the orthogonal matrix $Z$. On exit, the product matrix $Z * U_2$. |
| out | *real* | An array of the same size as $S$ containing the real parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *imag* | An array of the same size as $S$ containing the imaginary parts of the $\alpha$ values of the computed generalized eigenvalues. |
| out | *beta* | An array of the same size as $S$ containing the $\beta$ values of computed generalized eigenvalues. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_GEP_DM_ReorderSchur
starneig_GEP_DM_Select
starneig_reorder_conf
starneig_reorder_init_conf

#### 11.8.2.9 starneig_GEP_DM_Eigenvectors_expert()

```
starneig_error_t starneig_GEP_DM_Eigenvectors_expert (
            struct starneig_eigenvectors_conf * conf,
            int selected[],
            starneig_distr_matrix_t S,
            starneig_distr_matrix_t T,
            starneig_distr_matrix_t Z,
            starneig_distr_matrix_t X )
```

Computes a generalized eigenvector for each selected generalized eigenvalue.

**Parameters**

| | | |
|---|---|---|
| in | *conf* | Configuration structure. |
| in | *selected* | The selection array specifying the locations of the selected generalized eigenvalues. The number of 1's in the array is the same as the number of columns in $X$. |
| in | *S* | The Schur matrix $S$. |
| in | *T* | The upper triangular matrix $T$. |
| in | *Z* | The orthogonal matrix $Z$. |
| out | *X* | A matrix with $n$ rows and one column for each selected generalized eigenvalue. The columns represent the computed generalized eigenvectors as previously described. |

**Returns**

STARNEIG_SUCCESS (0) on success. Negative integer -i when i'th argument is invalid. Positive error code otherwise.

**See also**

starneig_GEP_DM_Select

**Todo** This interface function is not implemented.

## 11.9 Expert configuration structures

Configuration structures and functions for the expert interface functions.

### Data Structures

- struct starneig_hessenberg_conf

    *Hessenberg reduction configuration structure. More...*
- struct starneig_schur_conf

    *Schur reduction configuration structure. More...*
- struct starneig_reorder_conf

    *Eigenvalue reordering configuration structure. More...*
- struct starneig_eigenvectors_conf

    *Eigenvector computation configuration structure. More...*

### Hessenberg reduction

- void starneig_hessenberg_init_conf (struct starneig_hessenberg_conf ∗conf)

    *Initializes a Hessenberg reduction configuration structure with default parameters.*
- #define STARNEIG_HESSENBERG_DEFAULT_TILE_SIZE -1

    *Default tile size.*
- #define STARNEIG_HESSENBERG_DEFAULT_PANEL_WIDTH -1

    *Default panel width.*

### Schur reduction

- void starneig_schur_init_conf (struct starneig_schur_conf ∗conf)

    *Initializes a Schur reduction configuration structure with default parameters.*
- #define STARNEIG_SCHUR_DEFAULT_INTERATION_LIMIT -1

    *Default iteration limit.*
- #define STARNEIG_SCHUR_DEFAULT_TILE_SIZE -1

    *Default tile size.*
- #define STARNEIG_SCHUR_DEFAULT_SMALL_LIMIT -1

    *Default sequential QR limit.*
- #define STARNEIG_SCHUR_DEFAULT_AED_WINDOW_SIZE -1

    *Default AED window size.*
- #define STARNEIG_SCHUR_DEFAULT_AED_SHIFT_COUNT -1

    *Default AED shift count.*
- #define STARNEIG_SCHUR_DEFAULT_AED_NIBBLE -1

    *Default nibble value.*
- #define STARNEIG_SCHUR_DEFAULT_AED_PARALLEL_SOFT_LIMIT -1

    *Default soft sequential AED limit.*
- #define STARNEIG_SCHUR_DEFAULT_AED_PARALLEL_HARD_LIMIT -1

    *Default hard sequential AED limit.*
- #define STARNEIG_SCHUR_DEFAULT_WINDOW_SIZE -1

    *Default bulge chasing window size.*
- #define STARNEIG_SCHUR_ROUNDED_WINDOW_SIZE -2

    *Rounded bulge chasing window.*
- #define STARNEIG_SCHUR_DEFAULT_SHIFTS_PER_WINDOW -1

*Default number of shifts per bulge chasing window.*

- #define STARNEIG_SCHUR_DEFAULT_UPDATE_WIDTH -1

  *Default left-hand side update width.*
- #define STARNEIG_SCHUR_DEFAULT_UPDATE_HEIGHT -1

  *Default right-hand side update height.*
- #define STARNEIG_SCHUR_DEFAULT_THRESHOLD -1

  *Default deflation threshold.*
- #define STARNEIG_SCHUR_NORM_STABLE_THRESHOLD -2

  *Norm stable deflation threshold.*
- #define STARNEIG_SCHUR_LAPACK_THRESHOLD -3

  *LAPACK-style deflation threshold.*

### Eigenvalue reordering

- enum starneig_reorder_plan_t { STARNEIG_REORDER_DEFAULT_PLAN = 1, STARNEIG_REORDER_↩ ONE_PART_PLAN = 2, STARNEIG_REORDER_MULTI_PART_PLAN = 3 }

  *Reordering plan enumerator.*
- enum starneig_reorder_blueprint_t {
  STARNEIG_REORDER_DEFAULT_BLUEPRINT = 1, STARNEIG_REORDER_DUMMY_INSERT_A = 2,
  STARNEIG_REORDER_DUMMY_INSERT_B = 3, STARNEIG_REORDER_CHAIN_INSERT_A = 4,
  STARNEIG_REORDER_CHAIN_INSERT_B = 5, STARNEIG_REORDER_CHAIN_INSERT_C = 6, STAR↩ NEIG_REORDER_CHAIN_INSERT_D = 7, STARNEIG_REORDER_CHAIN_INSERT_E = 8,
  STARNEIG_REORDER_CHAIN_INSERT_F = 9 }

  *Task insertion blueprint.*
- void starneig_reorder_init_conf (struct starneig_reorder_conf ∗conf)

  *Initializes an eigenvalue reordering configuration structure with default parameters.*
- #define STARNEIG_REORDER_DEFAULT_UPDATE_WIDTH -1

  *Default left-hand side update task width.*
- #define STARNEIG_REORDER_DEFAULT_UPDATE_HEIGHT -1

  *Default right-hand side update task height.*
- #define STARNEIG_REORDER_DEFAULT_TILE_SIZE -1

  *Default tile size.*
- #define STARNEIG_REORDER_DEFAULT_VALUES_PER_CHAIN -1

  *Default number of selected eigenvalues per window.*
- #define STARNEIG_REORDER_DEFAULT_WINDOW_SIZE -1

  *Default default window size.*
- #define STARNEIG_REORDER_ROUNDED_WINDOW_SIZE -2

  *Default rounded window size.*
- #define STARNEIG_REORDER_DEFAULT_SMALL_WINDOW_SIZE -1

  *Default small window size.*
- #define STARNEIG_REORDER_DEFAULT_SMALL_WINDOW_THRESHOLD -1

  *Default small window threshold.*

### Eigenvectors

- void starneig_eigenvectors_init_conf (struct starneig_eigenvectors_conf ∗conf)

  *Initializes an eigenvectors configuration structure with default parameters.*
- #define STARNEIG_EIGENVECTORS_DEFAULT_TILE_SIZE -1

  *Default tile size.*

#### 11.9.1 Detailed Description

Configuration structures and functions for the expert interface functions.

#### 11.9.2 Data Structure Documentation

##### 11.9.2.1 struct starneig_hessenberg_conf

Hessenberg reduction configuration structure.

**Data Fields**

| | | |
|---|---|---|
| int | tile_size | The matrices are divided into square tiles. This parameter defines the used tile size. If the parameter is set to STARNEIG_HESSENBERG_DEFAULT_TILE_SIZE, then the implementation will determine a suitable tile size automatically. |
| int | panel_width | The reduction is performed one panel at a time. This parameter defines the used panel width. If the parameter is set to STARNEIG_HESSENBERG_DEFAULT_PANEL_WIDTH, then the implementation will determine a suitable panel width automatically. |

##### 11.9.2.2 struct starneig_schur_conf

Schur reduction configuration structure.

**Data Fields**

| | | |
|---|---|---|
| int | iteration_limit | The QR/QZ is an iterative algorithm. This parameter defines the maximum number of iterations the algorithm is allowed to perform. If the parameter is STARNEIG_SCHUR_DEFAULT_INTERATION_LIMIT, then the implementation will determine a suitable iteration limit automatically. |
| int | tile_size | The matrices are divided into square tiles. This parameter defines the used tile size. If the parameter is set to STARNEIG_SCHUR_DEFAULT_TILE_SIZE, then the implementation will determine a suitable tile size automatically. |
| int | small_limit | As the QR/QZ algorithm progresses, the size of the active region shrinks. Once the size of the active region is small enough, then the remaining problem is solved in a sequential manner. This parameter defines the transition point where the implementation switches to a sequential QR algorithm. If the parameter is set to STARNEIG_SCHUR_DEFAULT_SMALL_LIMIT, then the implementation will determine a suitable switching point automatically. |
| int | aed_window_size | The implementation relies on a so-called Aggressive Early Deflation (AED) technique to accelerate the convergence of the algorithm. Each AED is performed inside a small diagonal window. This parameter defines used AED window size. If the parameter is set to STARNEIG_SCHUR_DEFAULT_AED_WINDOW_SIZE, then the implementation will determine a suitable AED window size automatically. |
| int | aed_shift_count | The QR/QZ algorithm chases a set of $3 \times 3$ bulges across the diagonal of the Hessenberg(-triangular) decomposition. Two shifts (eigenvalue estimates) are required to generate each bulge. This parameter defines the number of shifts to use. If the parameter is set to STARNEIG_SCHUR_DEFAULT_AED_SHIFT_COUNT, then the implementation will determine a suitable shift count automatically. |

**Data Fields**

| | | |
|---|---|---|
| int | aed_nibble | The implementation relies on a so-called Aggressive Early Deflation (AED) technique to accelerate the convergence of the algorithm. Each AED is performed inside a small diagonal window. If the number deflated (converged) eigenvalues is larger than `aed_nibble` $\times$ size of AED window, then the next bulge chasing step is skipped. If the parameter is set to STARNEIG_SCHUR_DEFAULT_AED_NIBBLE, then the implementation will determine a suitable value automatically. |
| int | aed_parallel_soft_limit | The implementation relies on a so-called Aggressive Early Deflation (AED) technique to accelerate the convergence of the algorithm. Each AED is performed inside a small diagonal window. An AED can be performed sequentially or in parallel. This parameter defines the transition point where the implementation allowed to switch to a sequential AED algorithm. The decision is made based on the size of the AED window. If the parameter is set to STARNEIG_SCHUR_DEFAULT_AED_PARALLEL_SOFT_LIMIT, then the implementation will determine a suitable switching point automatically. |
| int | aed_parallel_hard_limit | The implementation relies on a so-called Aggressive Early Deflation (AED) technique to accelerate the convergence of the algorithm. Each AED is performed inside a small diagonal window. An AED can be performed sequentially or in parallel. This parameter defines the transition point where the implementation switches to a sequential AED algorithm. The decision is made based on the size of the AED window. If the parameter is set to STARNEIG_SCHUR_DEFAULT_AED_PARALLEL_HARD_LIMIT, then the implementation will determine a suitable switching point automatically. |
| int | window_size | The QR/QZ algorithm chases a set of $3 \times 3$ bulges across the diagonal of the Hessenberg(-triangular) decomposition. The bulges are chased in batches. The related similarity transformations are initially restricted to inside a small diagonal window and the accumulated transformation are applied only later as BLAS-3 updates. This parameter defines the used bulge chasing window size. If the parameter is set to STARNEIG_SCHUR_ROUNDED_WINDOW_SIZE, then<br>• maximum window size is set to 2 $*$ tile_size and<br>• the windows are placed such that their lower right corners respect the boundaries of the underlying data tiles.<br>If the parameter is set to STARNEIG_SCHUR_DEFAULT_WINDOW_SIZE, then the implementation will determine a suitable window size automatically. |
| int | shifts_per_window | The QR/QZ algorithm chases a set of $3 \times 3$ bulges across the diagonal of the Hessenberg(-triangular) decomposition. The bulges are chased in batches. This parameter defines the used batch size. If the parameter is set to STARNEIG_SCHUR_DEFAULT_SHIFTS_PER_WINDOW then the implementation will determine a suitable batch size automatically. |
| int | update_width | The similarity similarity transformations are initially restricted to inside a small diagonal window and the accumulated transformation are applied only later as BLAS-3 updates. This parameter defines the width of each left-hand side update task. The value should be multiple of the tile size. If the parameter is set to STARNEIG_SCHUR_DEFAULT_UPDATE_WIDTH, then the implementation will determine a suitable width automatically. |

**Data Fields**

| | | |
|---|---|---|
| int | update_height | The similarity similarity transformations are initially restricted to inside a small diagonal window and the accumulated transformation are applied only later as BLAS-3 updates. This parameter defines the height of each right-hand side update task. The value should be multiple of the tile size. If the parameter is set to STARNEIG_SCHUR_DEFAULT_UPDATE_HEIGHT, then the implementation will determine a suitable height automatically. |
| double | left_threshold | The QR/QZ algorithm is allowed to set tiny matrix entires to zero as long as their magnitudes are smaller that a given threshold. This parameter defines the threshold for the left-hand side matrix ( $H$). If the parameter is set to STARNEIG_SCHUR_DEFAULT_THRESHOLD, then the implementation will determine a suitable threshold automatically. If the parameter is set to STARNEIG_SCHUR_NORM_STABLE_THRESHOLD, then the implementation will use the threshold $u|H|_F$, where $u$ is the unit roundoff and $|H|_F$ is the Frobenius norm of the matrix $H$. If the parameter is set to STARNEIG_SCHUR_LAPACK_THRESHOLD, then the implementation will use a deflation threshold that is compatible with LAPACK. |
| double | right_threshold | The QZ algorithm is allowed to set tiny matrix entires to zero as long as their magnitudes are smaller that a given threshold. This parameter defines the threshold for the right-hand side matrix ( $R$) off-diagonal entires. If the parameter is set to STARNEIG_SCHUR_DEFAULT_THRESHOLD, then the implementation will determine a suitable threshold automatically. If the parameter is set to STARNEIG_SCHUR_NORM_STABLE_THRESHOLD, then the implementation will use the threshold $u|R|_F$, where $u$ is the unit roundoff and $|H|_F$ is the Frobenius norm of the matrix $R$. If the parameter is set to STARNEIG_SCHUR_LAPACK_THRESHOLD, then the implementation will use a deflation threshold that is compatible with LAPACK. |
| double | inf_threshold | The QZ algorithm is allowed to set tiny matrix entires to zero as long as their magnitudes are smaller that a given threshold. This parameter defines the threshold for the right-hand side matrix ( $R$) diagonal entries. If the parameter is set to STARNEIG_SCHUR_DEFAULT_THRESHOLD, then the implementation will determine a suitable threshold automatically. If the parameter is set to STARNEIG_SCHUR_NORM_STABLE_THRESHOLD, then the implementation will use the threshold $u|R|_F$, where $u$ is the unit roundoff and $|R|_F$ is the Frobenius norm of the matrix $R$. |

#### 11.9.2.3 struct starneig_reorder_conf

Eigenvalue reordering configuration structure.

**Data Fields**

| | | |
|---|---|---|
| starneig_reorder_plan_t | plan | This parameter plan defines the used reordering plan. If the parameter is set to STARNEIG_REORDER_DEFAULT_PLAN, then the implementation will determine a suitable reordering plan automatically. |
| starneig_reorder_blueprint_t | blueprint | This parameter defines the used task insertion blueprint. If the parameter is set to STARNEIG_REORDER_DEFAULT_BLUEPRINT, then the implementation will determine a suitable task insertion blueprint automatically. |

### Data Fields

| | | |
|---|---|---|
| int | tile_size | The matrices are divided into square tiles. This parameter defines the used tile size. If the parameter is set to STARNEIG_REORDER_DEFAULT_TILE_SIZE, then the implementation will determine a suitable tile size automatically. |
| int | values_per_chain | The selected eigenvalues are processed in batches and each batch is assigned a window chain. This parameter defines the number of selected eigenvalues processed by each window chain. If the parameter is set to STARNEIG_RE↩ORDER_DEFAULT_VALUES_PER_CHAIN, then the implementation will determine a suitable value automatically. |
| int | window_size | The similarity similarity transformations are initially restricted to inside a small diagonal window and the accumulated transformation are applied only later as BLAS-3 updates. This parameter defines the size of the window. If the parameter is set to STA↩RNEIG_REORDER_ROUNDED_WINDOW_SIZE, then<br>• maximum window size is set to $2 * $ tile_size,<br>• the windows are placed such that their upper left corners respect the boundaries of the underlying data tiles, and<br>• the parameter values_per_chain is ignored.<br>If the parameter is set to STARNEIG_REORDER↩_DEFAULT_WINDOW_SIZE, then the implementation will determine a suitable window size automatically. |
| int | small_window_size | Larger diagonal window are processed using even smaller diagonal windows in a recursive manner. This parameter defines the used small window size. If the parameter is set to STARNEIG_REO↩RDER_DEFAULT_SMALL_WINDOW_SIZE, then the implementation will determine a suitable small window size automatically. |
| int | small_window_threshold | Larger diagonal window are processed using even smaller diagonal windows in a recursive manner. This parameter defines the largest diagonal window that is processed in a scalar manner. If the parameter is set to STARNEIG_REORDER_DE↩FAULT_SMALL_WINDOW_THRESHOLD, then the implementation will determine a suitable threshold automatically. |

**Data Fields**

| | | |
|---|---|---|
| int | update_width | The similarity similarity transformations are initially restricted to inside a small diagonal window and the accumulated transformation are applied only later as BLAS-3 updates. This parameter defines the width of each left-hand side update task. The value should be multiple of the tile size. If the parameter is set to STARNEIG_REORDER_DEFAULT_UP↩ DATE_WIDTH, then the implementation will determine a suitable width automatically. |
| int | update_height | The similarity similarity transformations are initially restricted to inside a small diagonal window and the accumulated transformation are applied only later as BLAS-3 updates. This parameter defines the height of each right-hand side update task. The value should be multiple of the tile size. If the parameter is set to STARNEIG_REORDER_DE↩ FAULT_UPDATE_HEIGHT, then the implementation will determine a suitable height automatically. |

#### 11.9.2.4 struct starneig_eigenvectors_conf

Eigenvector computation configuration structure.

**Data Fields**

| | | |
|---|---|---|
| int | tile_size | The matrices are divided into tiles. This parameter defines the used tile size. If the parameter is set to STARNEIG_EIGENVECTORS_DEFAULT_TILE_SIZE, then the implementation will determine a suitable tile size automatically. |

### 11.9.3 Enumeration Type Documentation

#### 11.9.3.1 starneig_reorder_plan_t

```
enum starneig_reorder_plan_t
```

Reordering plan enumerator.

Eigenvalues that fall within a diagonal computation *window* are reordered such that all selected eigenvalues are moved to the upper left corner of the window. The corresponding orthogonal transformations are accumulated to separate accumulator matrix / matrices.

```
+---------+        +---------+
|¤ x x x x|        |$ x x x x|       $ selected eigenvalue
|  $ x x x|        |  $ x x x|       ¤ deselected eigenvalue
|    ¤ x x| ==> Q  |    ¤ x x| Q^T    x non-zero entry
|      ¤ x|        |      ¤ x|
|        $|        |        ¤|
+---------+        +---------+
```

A *window chain* comprises from multiple overlapping diagonal computation windows that are intended to be processed in a particular order. More precisely, the windows are placed such that the overlap between two windows is big enough to accommodate all selected eigenvalues that fall within the preceding windows. In this way, the windows can be processed in sequential order, starting from the bottom window, such that the reordering that takes place in one window always moves the preceding selected eigenvalues to the lower right corner of the next window. In the end, all selected that fall within the combined computation area of the chain are moved to the upper left corner of the topmost window.

An example showing how an eigenvalue can be moved six entries upwards by using three diagonal windows:

```
+-------+         +-------+         +-------+         +-------+
| x x x | ===>    | x x x | ===>    | x x x | ===>    | #<--+ |
|   x-x-+---+     |   x-x-+---+     |   x-x-+---+     |   ¤-|-+---+
|   | x x x |     |   | x x x |     |   | #<--+ |     |   | ¤ ¤ ¤ |
+---+---x-x-+---+ +---+---x-x-+---+ +---+---¤-|-+---+ +---+---¤-¤-+---+
    |   | x x x |     |   | #<--+ |     |   | ¤ ¤ ¤ |     |   | ¤ ¤ ¤ |
    +---+---x x |     +---+---¤ | |     +---+---¤ ¤ |     +---+---¤ ¤ |
        |     # |         |     ¤ |         |     ¤ |         |     ¤ |
        +-------+         +-------+         +-------+         +-------+
```

The number of selected eigenvalues that can be moved by a single window chain is limited by the windows size. Thus, the whole reordering procedure usually involves multiple chains that must be be processed in a particular order. A *chain list* describes a list of chains that are intended to be processed together. Window chains that belong to different chain lists are processed separately.

A *plan* consists from one or more chain lists that are intended to be processed in a particular order.

**STARNEIG_REORDER_ONE_PART_PLAN:**

The first chain is placed in the upper left corner of the matrix and its size is chosen such that it contains a desired number of selected eigenvalues (values_per_chain parameter). The next chain is places such that its upper left corner is located one entry after the location where the last selected eigenvalue, that falls within the first chain, would be after the reordering. The chain is sized such that the part of the chain, that does not intersect the first chain, contain the desired number of selected eigenvalues. This same procedure is repeated until all selected eigenvalues have been accounted for. All chains belong to the same chain lists and are intended to be processed sequentially.

An example showing the placement of the chains in a case where each chain wields two selected eigenvalues:

```
                                                      before:      after:
.-------+                                             .            0
| 0     | <-- chain 0                                 0 _          1_
|   .-----------+                                     . .   ====>  2
|   | . |       | <-- chain 1                         . . _        3_
+---|---1---------------+ . . . . . . . . . . . . . . 1_. .        4
    |   | .     |       | <-- chain 2                   . . _      5_
    |   |   2-------------------+                       2 . .      6
    |   |   | . |       |       | <-- chain 3           . . . _    7_
    +---|---|---3-----------------------+ . . . . . . . 3_. . .    8
        |   |   | 4     |       |       | <-- chain 4     4 . .    9
        |   |   |   .   |       |       |                 . . .    .
        |   |   |     . |       |       |                 . . .    .
        +---|---|-------5       |       | . . . . . . . . 5_. .    .
            |   |         6     |       |                   6 .    .
            |   |           .   |       |                   . .    .
            |   |             . |       |                   . .    .
            +---|---------------7       | . . . . . . . . . 7_.    .
                |                 .     |                     .    .
                |                   .   |                     .    .
                |                     8 |                     8    .
                +-----------------------9 . . . . . . . . . . 9    .
```

An example showing what happens when the first three chains are processed:

```
.-------+                        >0<------+
| 0     |                         |>1<    |
|   .-----------+                 |   #-----------+
|   | . |       |                 |   | # |       |
+---|---1---------------+         +---|---#---------------+
    |   | .     |       |             |   | .     |       |
    |   |   2   |       |  ===>       |   |   2   |       |  ===>
    |   |     . |       |             |   |     . |       |
    +---|-------3       |             +---|-------3       |
        |         4     |                 |         4     |
        |           .   |                 |           .   |
        |             . |                 |             . |
        +---------------5                 +---------------5

0-------+                         0-------+
| 1     |                         | 1     |
|  >2<----------+                 |   2-----------+
|   |>3<|       |                 |   | 3 |       |
+---|---#---------------+         +---|-->4<--------------+
    |   | #     |       |             |   |>5<    |       |
    |   |   #   |       |   ===>      |   |   #   |       |
    |   |     # |       |             |   |     # |       |
    +---|-------#       |             +---|-------#       |
        |         4     |                 |         #     |
        |           .   |                 |           #   |
        |             . |                 |             # |
        +---------------5                 +---------------#
```

If necessary, each chain is re-sized to avoid splitting any $2 \times 2$ tiles.

Windows are placed such that the first window is located in the lower right corner of the computation area of the window chain. The last window is correspondingly placed in the upper left corner of the computation area.

If necessary, each window is re-sized to avoid splitting any $2 \times 2$ tiles.

**STARNEIG_REORDER_MULTI_PART_PLAN:**

A multi-part reordering plan is derived from an one-part reordering plan by splitting the chains into sub-chains as shown below:

```
Initial one-part plan:
  Chain 0: aaaaaa
  Chain 1:  bbbbbbbbbb               a,b,c,d,e diagonal computation window
  Chain 2:   cccccccccccccc
  Chain 3:    dddddddddddddddddd
  Chain 4:     eeeeeeeeeeeeeeeeeeeeee

Resulting multi-part plan:
  Chain 0: aaaaaa
  Chain 1:  ......bbbb
  Chain 2:   ..........cccc               chain list 0
  Chain 3:    ..............dddd
  Chain 4:     ..................eeee
  -----------------------------------------------------
  Chain 0:  bbbbbb....
  Chain 1:   ......cccc....               chain list 1
  Chain 2:    ..........dddd....
  Chain 3:     ..............eeee....
  -----------------------------------------------------
  Chain 0:   cccccc........
  Chain 1:    ......dddd........          chain list 2
  Chain 2:     ..........eeee........
  -----------------------------------------------------
  Chain 0:    dddddd............          chain list 3
  Chain 1:     ......eeee............
  -----------------------------------------------------
  Chain 0:     eeeeee................     chain list 4
```

Note that the chains that belong to the same chain list are independent from each other and can therefore be processed in an arbitrary order.

**Enumerator**

| | |
|---|---|
| STARNEIG_REORDER_DEFAULT_PLAN | Default plan. |
| STARNEIG_REORDER_ONE_PART_PLAN | One part plan. |
| STARNEIG_REORDER_MULTI_PART_PLAN | Multi part plan. |

#### 11.9.3.2 starneig_reorder_blueprint_t

`enum starneig_reorder_blueprint_t`

Task insertion blueprint.

A task insertion blueprint defines how a reordering plan is carried out.

**Enumerator**

| | |
|---|---|
| STARNEIG_REORDER_DEFAULT_BLUEPRINT | Default blueprint. |
| STARNEIG_REORDER_DUMMY_INSERT_A | One-pass forward dummy blueprint. Processes the window chains in order starting from the topmost chain. All update tasks are inserted right after each window reordering task. |
| STARNEIG_REORDER_DUMMY_INSERT_B | Two-pass backward dummy blueprint. Processes the window chains in two phases starting from the bottommost chain. The window reordering tasks and the right-hand side update tasks are inserted during the first phase. Other update tasks are inserted during the second phase. |
| STARNEIG_REORDER_CHAIN_INSERT_A | One-pass forward chain blueprint. Processes the window chains in order starting from the topmost chain. The window reordering tasks and high priority right-hand side update tasks are inserted first. Other update tasks are inserted after them. |
| STARNEIG_REORDER_CHAIN_INSERT_B | Two-pass forward chain blueprint. Processes the window chains in two phases starting from the topmost chain. The window reordering tasks and high priority right-hand side update tasks are inserted first. The left- hand side updates are inserted after them. Other updates are inserted during the second phase. |
| STARNEIG_REORDER_CHAIN_INSERT_C | One-pass backward chain blueprint. Processes the window chains in order starting from the bottommost chain. The window reordering tasks and high priority right-hand side update tasks are inserted first. Other update tasks are inserted later. |
| STARNEIG_REORDER_CHAIN_INSERT_D | Two-pass backward chain blueprint. Processes the window chains in two phases starting from the bottommost chain. The window reordering tasks and high priority right-hand side update tasks are inserted first. Update tasks that are related to the Schur matrix are inserted later. Update tasks that are related to the orthogonal matrices are inserted during the second phase. |
| STARNEIG_REORDER_CHAIN_INSERT_E | Two-pass delayed backward chain blueprint. Processes the window chains in order starting from the bottommost chain. The window reordering tasks and high priority right-hand side update tasks are inserted first. Update tasks that are related to the Schur matrix are inserted later. Update tasks that are related to the orthogonal matrices are inserted only after all chain list have been processed. |

**Enumerator**

| | |
|---|---|
| STARNEIG_REORDER_CHAIN_INSERT_F | Three-pass delayed backward chain blueprint. Processes the window chains in two phases starting from the bottommost chain. The window reordering tasks and high priority right-hand side update tasks are inserted during the first phase. Update tasks that are related to the Schur matrix are inserted during the second phase. Update tasks that are related to the orthogonal matrices are inserted only after all chain list have been processed. |

### 11.9.4 Function Documentation

#### 11.9.4.1 starneig_hessenberg_init_conf()

```
void starneig_hessenberg_init_conf (
            struct starneig_hessenberg_conf * conf )
```

Initializes a Hessenberg reduction configuration structure with default parameters.

**Parameters**

| | | |
|---|---|---|
| out | *conf* | The Hessenberg reduction configuration structure. |

#### 11.9.4.2 starneig_schur_init_conf()

```
void starneig_schur_init_conf (
            struct starneig_schur_conf * conf )
```

Initializes a Schur reduction configuration structure with default parameters.

**Parameters**

| | | |
|---|---|---|
| out | *conf* | The Schur reduction configuration structure. |

#### 11.9.4.3 starneig_reorder_init_conf()

```
void starneig_reorder_init_conf (
            struct starneig_reorder_conf * conf )
```

Initializes an eigenvalue reordering configuration structure with default parameters.

**Parameters**

| | | |
|---|---|---|
| out | *conf* | The eigenvalue reordering configuration structure. |

#### 11.9.4.4 starneig_eigenvectors_init_conf()

```
void starneig_eigenvectors_init_conf (
            struct starneig_eigenvectors_conf * conf )
```

Initializes an eigenvectors configuration structure with default parameters.

**Parameters**

| | | |
|---|---|---|
| out | *conf* | The eigenvectors configuration structure. |

## 11.10 ScaLAPACK compatibility / BLACS matrices

Data types and functions for BLACS formatted distributed matrices.

### Data Structures

- struct starneig_blacs_descr
  *BLACS descriptor. More...*

### BLACS contexts

- typedef int starneig_blacs_context_t
  *BLACS context.*
- starneig_blacs_context_t starneig_distr_to_blacs_context (starneig_distr_t distr)
  *Convers a data distribution to a BLACS context.*
- starneig_distr_t starneig_blacs_context_to_distr (starneig_blacs_context_t context)
  *Convers a BLACS context to a data distribution.*
- int starneig_distr_is_blacs_compatible (starneig_distr_t distr)
  *Checks whether a data distribution is BLACS compatible.*
- int starneig_distr_is_compatible_with (starneig_distr_t distr, starneig_blacs_context_t context)
  *Checks whether a data distribution is compatible with a given BLACS context.*

### BLACS descriptors

- typedef struct starneig_blacs_descr starneig_blacs_descr_t
  *BLACS descriptor.*
- void starneig_create_blacs_matrix (int rows, int cols, int row_blksz, int col_blksz, starneig_datatype_t type, starneig_blacs_context_t context, starneig_blacs_descr_t ∗descr, void ∗∗local)
  *Creates a BLACS matrix with uninitialized matrix elements.*
- void starneig_destroy_blacs_matrix (starneig_blacs_descr_t ∗descr, void ∗∗local)
  *Destroyes a BLACS matrix.*
- void starneig_distr_matrix_to_blacs_descr (starneig_distr_matrix_t matrix, starneig_blacs_context_t context, starneig_blacs_descr_t ∗descr, void ∗∗local)
  *Convers a distributed matrix to a BLACS descriptor and a matching local array.*
- starneig_distr_matrix_t starneig_blacs_descr_to_distr_matrix (starneig_datatype_t type, starneig_distr_↩ t distr, starneig_blacs_descr_t ∗descr, void ∗local)
  *Convers a BLACS descriptor and a matching local array to a distributed matrix.*
- int starneig_distr_matrix_is_blacs_compatible (starneig_distr_matrix_t matrix)
  *Checks whether a distributed matrix is BLACS compatible.*
- int starneig_distr_matrix_is_compatible_with (starneig_distr_matrix_t matrix, starneig_blacs_context_t context)
  *Checks whether a distributed matrix is compatible with a given BLACS context.*

### 11.10.1 Detailed Description

Data types and functions for BLACS formatted distributed matrices.

### 11.10.2 Data Structure Documentation

#### 11.10.2.1 struct starneig_blacs_descr

BLACS descriptor.

#### Data Fields

| | | |
|---|---|---|
| int | type | The descriptor type. |
| starneig_blacs_context_t | context | The related BLACS context. |
| int | m | The number of (global) rows in the matrix. |
| int | n | The number of (global) columns in the matrix. |
| int | sm | The number of rows in a distribution block. |
| int | sn | The number of columns in a distribution block. |
| int | rsrc | The process grid row over which the first row is distributed. |
| int | csrc | The process grid column over which the first column is distributed. |
| int | lld | The leading dimension of the local array. |

### 11.10.3 Function Documentation

#### 11.10.3.1 starneig_distr_to_blacs_context()

```
starneig_blacs_context_t starneig_distr_to_blacs_context (
            starneig_distr_t distr )
```

Convers a data distribution to a BLACS context.

**Attention**

The data distribution must describe a two-dimensional block cyclic distribution.

**Parameters**

| | | |
|---|---|---|
| in | *distr* | The data distribution. |

**Returns**

The BLACS context.

#### 11.10.3.2 starneig_blacs_context_to_distr()

```
starneig_distr_t starneig_blacs_context_to_distr (
            starneig_blacs_context_t context )
```

Convers a BLACS context to a data distribution.

**Parameters**

| | | |
|---|---|---|
| in | *context* | The BLACS context. |

**Returns**

The data distribution.

#### 11.10.3.3 starneig_distr_is_blacs_compatible()

```
int starneig_distr_is_blacs_compatible (
            starneig_distr_t distr )
```

Checks whether a data distribution is BLACS compatible.

**Parameters**

| | | |
|---|---|---|
| in | *distr* | The data distribution. |

**Returns**

Non-zero if the data distribution matrix is BLACS compatible.

#### 11.10.3.4 starneig_distr_is_compatible_with()

```
int starneig_distr_is_compatible_with (
            starneig_distr_t distr,
            starneig_blacs_context_t context )
```

Checks whether a data distribution is compatible with a given BLACS context.

**Parameters**

| | | |
|---|---|---|
| in | *distr* | The data distribution. |
| in | *context* | The BLACS context. |

**Returns**

Non-zero if the data distribution compatible with the BLACS context.

#### 11.10.3.5 starneig_create_blacs_matrix()

```
void starneig_create_blacs_matrix (
            int rows,
            int cols,
            int row_blksz,
            int col_blksz,
            starneig_datatype_t type,
            starneig_blacs_context_t context,
            starneig_blacs_descr_t * descr,
            void ** local )
```

Creates a BLACS matrix with uninitialized matrix elements.

**Parameters**

| | | |
|---|---|---|
| in | *rows* | The number of (global) rows in the matrix. |
| in | *cols* | The number of (global) columns in the matrix. |
| in | *row_blksz* | The number of rows in a distribution block. Can be set to -1 in which case the library decides the value. |
| in | *col_blksz* | The number of columns in a distribution block. Can be set to -1 in which case the library decides the value. |
| in | *type* | The matrix element data type. |
| in | *context* | The BLACS context. |
| out | *descr* | The BLACS descriptor. |
| out | *local* | A pointer to the local array. |

#### 11.10.3.6 starneig_destroy_blacs_matrix()

```
void starneig_destroy_blacs_matrix (
            starneig_blacs_descr_t * descr,
            void ** local )
```

Destroyes a BLACS matrix.

**Parameters**

| | | |
|---|---|---|
| in,out | *descr* | The BLACS descriptor. |
| in,out | *local* | A pointer to the local array. |

#### 11.10.3.7 starneig_distr_matrix_to_blacs_descr()

```
void starneig_distr_matrix_to_blacs_descr (
            starneig_distr_matrix_t matrix,
            starneig_blacs_context_t context,
            starneig_blacs_descr_t * descr,
            void ** local )
```

Convers a distributed matrix to a BLACS descriptor and a matching local array.

This function creates a wrapper object. The contents of the distributed matrix may be modified by the functions that use the wrapper object.

```
starneig_distr_matrix_t dA = starneig_distr_matrix_create
      (...);

...

starneig_distr_t distr = starneig_distr_matrix_get_distr(A);
starneig_blacs_context_t context =
      starneig_distr_to_blacs_context(distr);

starneig_blacs_descr_t descr_a;
double *local_a;
starneig_distr_matrix_to_blacs_descr(
    dA, context, &descr_a, (void **)&local_a);
```

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |
| in | *context* | The BLACS context. The context must have been converted from the same data distribution the distributed matrix is using or vice versa. |
| out | *descr* | The BLACS descriptor. |
| out | *local* | A pointer to the local array. |

#### 11.10.3.8 starneig_blacs_descr_to_distr_matrix()

```
starneig_distr_matrix_t starneig_blacs_descr_to_distr_matrix (
            starneig_datatype_t type,
            starneig_distr_t distr,
            starneig_blacs_descr_t * descr,
            void * local )
```

Convers a BLACS descriptor and a matching local array to a distributed matrix.

This function creates a wrapper object. The contents of the local array may be modified by the functions that use the wrapper object. The starneig_distr_matrix_destroy() function does not de-initilize the BLACS descriptor nor free the local array.

```
starneig_blacs_context_t context;
starneig_blacs_descr_t descr_a;
double *local_a;

...

starneig_distr_t distr = starneig_blacs_context_to_distr(context);
starneig_distr_matrix_t dA =
      starneig_blacs_descr_to_distr_matrix(
    STARNEIG_REAL_DOUBLE, distr, descr_a, (void *)local_a);
```

**Parameters**

| | | |
|---|---|---|
| in | *type* | The matrix element data type. |
| in | *distr* | The data distribution. The data distribution must have been converted from the same BLACS context the BLACS descriptor is using or vice versa. |
| in | *descr* | The BLACS descriptor. |
| in | *local* | A pointer to the local array. |

**Returns**

The distributed matrix.

#### 11.10.3.9 starneig_distr_matrix_is_blacs_compatible()

```
int starneig_distr_matrix_is_blacs_compatible (
            starneig_distr_matrix_t matrix )
```

Checks whether a distributed matrix is BLACS compatible.

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |

**Returns**

Non-zero if the distributed matrix is BLACS compatible.

#### 11.10.3.10 starneig_distr_matrix_is_compatible_with()

```
int starneig_distr_matrix_is_compatible_with (
            starneig_distr_matrix_t matrix,
            starneig_blacs_context_t context )
```

Checks whether a distributed matrix is compatible with a given BLACS context.

**Parameters**

| | | |
|---|---|---|
| in | *matrix* | The distributed matrix. |
| in | *context* | The BLACS context. |

**Returns**

Non-zero if the distributed matrix compatible with the BLACS context.

## 11.11 ScaLAPACK compatibility / BLACS helpers

Data types and helper functions for BLACS.

### Functions

- void starneig_blacs_pinfo (int ∗my_rank, int ∗rank_count)

  *Queries process rank information.*
- int starneig_blacs_get (starneig_blacs_context_t context, starneig_blacs_query_id_t query)

  *Returns BLACS context's internal defaults.*
- starneig_blacs_context_t starneig_blacs_gridinit (starneig_blacs_context_t system_context, char ∗order, int rows, int cols)

  *Initializes a BLACS process grid.*
- void starneig_blacs_gridinfo (starneig_blacs_context_t context, int ∗rows, int ∗cols, int ∗row, int ∗col)

  *Queries BLACS process grid information.*
- void starneig_blacs_pcoord (starneig_blacs_context_t context, int process, int ∗row, int ∗col)

  *Queries BLACS process grid coordinates.*
- void starneig_blacs_gridexit (starneig_blacs_context_t context)

  *Releases process grid specific resources.*
- void starneig_blacs_exit (int cont)

  *Releases all contexts and related resources.*
- int starneig_numroc (int n, int nb, int iproc, int isrcproc, int nprocs)

  *Computes the number of matrix rows/columns owned by a given process.*
- int starneig_descinit (struct starneig_blacs_descr ∗descr, int m, int n, int sm, int sn, int irsrc, int icsrc, starneig_blacs_context_t context, int ld)

  *Initializes a BLACS descriptor.*

### Query indeces

- typedef int starneig_blacs_query_id_t

  *Data type for blacs_get() function query id.*
- #define STARNEIG_BLACS_GET_DEFAULT_CONTEXT 0

  *Query id for getting the default system context.*

### 11.11.1 Detailed Description

Data types and helper functions for BLACS.

### 11.11.2 Function Documentation

#### 11.11.2.1 starneig_blacs_pinfo()

```
void starneig_blacs_pinfo (
            int * my_rank,
            int * rank_count )
```

Queries process rank information.

**Parameters**

| | | |
|---|---|---|
| out | *my_rank* | An unique process id (rank). |
| out | *rank_count* | The total number of processes (ranks) available. |

#### 11.11.2.2 starneig_blacs_get()

```
int starneig_blacs_get (
            starneig_blacs_context_t context,
            starneig_blacs_query_id_t query )
```

Returns BLACS context's internal defaults.

**Parameters**

| | | |
|---|---|---|
| in | *context* | The BLACS context. |
| in | *query* | The query id. |

**Returns**

The internal default value that matches the given query id.

#### 11.11.2.3 starneig_blacs_gridinit()

```
starneig_blacs_context_t starneig_blacs_gridinit (
            starneig_blacs_context_t system_context,
            char * order,
            int rows,
            int cols )
```

Initializes a BLACS process grid.

**Parameters**

| | | |
|---|---|---|
| in | *system_context* | The system BLACS context to be used in creating the process grid. |
| in | *order* | The process mapping order. "R" : Use row-major natural ordering. "C" : Use column-major natural ordering. ELSE: Use row-major natural ordering. |
| in | *rows* | The number of rows in the process grid. |
| in | *cols* | The number of columns in the process grid. |

**Returns**

A handle to the created BLACS context.

#### 11.11.2.4 starneig_blacs_gridinfo()

```
void starneig_blacs_gridinfo (
            starneig_blacs_context_t context,
            int * rows,
            int * cols,
            int * row,
            int * col )
```

Queries BLACS process grid information.

**Parameters**

| | | |
|---|---|---|
| in | *context* | The BLACS context. |
| out | *rows* | The number of rows in the process grid. |
| out | *cols* | The number of columns in the process grid. |
| out | *row* | The row coordinate of the calling process. |
| out | *col* | The column coordinate of the calling process. |

#### 11.11.2.5 starneig_blacs_pcoord()

```
void starneig_blacs_pcoord (
            starneig_blacs_context_t context,
            int process,
            int * row,
            int * col )
```

Queries BLACS process grid coordinates.

**Parameters**

| | | |
|---|---|---|
| in | *context* | The BLACS context. |
| in | *process* | The process id (rank). |
| out | *row* | The row coordinate of the process. |
| out | *col* | The column coordinate of the process. |

#### 11.11.2.6 starneig_blacs_gridexit()

```
void starneig_blacs_gridexit (
            starneig_blacs_context_t context )
```

Releases process grid specific resources.

**Parameters**

| | | |
|---|---|---|
| in | *context* | The BLACS context. |

#### 11.11.2.7 starneig_blacs_exit()

```
void starneig_blacs_exit (
            int cont )
```

Releases all contexts and related resources.

**Parameters**

| | | |
|---|---|---|
| in | *cont* | The continue flag. |

#### 11.11.2.8 starneig_numroc()

```
int starneig_numroc (
            int n,
            int nb,
            int iproc,
            int isrcproc,
            int nprocs )
```

Computes the number of matrix rows/columns owned by a given process.

**Parameters**

| | | |
|---|---|---|
| in | *n* | The number of rows/columns in the distributed matrix. |
| in | *nb* | The block size. |
| in | *iproc* | The coordinate of the process whose local array row or column is to be determined. |
| in | *isrcproc* | The coordinate of the process that possesses the first row or column of the distributed matrix. |
| in | *nprocs* | The total number processes over which the matrix is distributed. |

**Returns**

The number of rows/columns owned by the process.

#### 11.11.2.9 starneig_descinit()

```
int starneig_descinit (
            struct starneig_blacs_descr * descr,
            int m,
            int n,
            int sm,
            int sn,
            int irsrc,
            int icsrc,
            starneig_blacs_context_t context,
            int ld )
```

Initializes a BLACS descriptor.

**Parameters**

| | | |
|---|---|---|
| out | *descr* | The matrix descriptor. |
| in | *m* | The number of rows in the matrix. |
| in | *n* | The number of columns in the matrix. |
| in | *sm* | The number of rows in a distributed block. |
| in | *sn* | The number of columns in a distributed block. |
| in | *irsrc* | The process grid row over which the first row is distributed. |
| in | *icsrc* | The process grid column over which the first column is distributed. |
| in | *context* | The BLACS context. |
| in | *ld* | The local array leading dimension. |

**Returns**

Zero if the initialization was successful, non-zero otherwise.

# 12 File Documentation

## 12.1 blacs_helpers.h File Reference

This file contains various BLACS helper functions.

```
#include <starneig/configuration.h>
#include <starneig/blacs_matrix.h>
```

### Functions

- void starneig_blacs_pinfo (int *my_rank, int *rank_count)

  *Queries process rank information.*
- int starneig_blacs_get (starneig_blacs_context_t context, starneig_blacs_query_id_t query)

  *Returns BLACS context's internal defaults.*
- starneig_blacs_context_t starneig_blacs_gridinit (starneig_blacs_context_t system_context, char *order, int rows, int cols)

  *Initializes a BLACS process grid.*
- void starneig_blacs_gridinfo (starneig_blacs_context_t context, int *rows, int *cols, int *row, int *col)

  *Queries BLACS process grid information.*
- void starneig_blacs_pcoord (starneig_blacs_context_t context, int process, int *row, int *col)

  *Queries BLACS process grid coordinates.*
- void starneig_blacs_gridexit (starneig_blacs_context_t context)

  *Releases process grid specific resources.*
- void starneig_blacs_exit (int cont)

  *Releases all contexts and related resources.*
- int starneig_numroc (int n, int nb, int iproc, int isrcproc, int nprocs)

  *Computes the number of matrix rows/columns owned by a given process.*
- int starneig_descinit (struct starneig_blacs_descr *descr, int m, int n, int sm, int sn, int irsrc, int icsrc, starneig_blacs_context_t context, int ld)

  *Initializes a BLACS descriptor.*

### Query indeces

- #define STARNEIG_BLACS_GET_DEFAULT_CONTEXT 0

  *Query id for getting the default system context.*
- typedef int starneig_blacs_query_id_t

  *Data type for blacs_get() function query id.*

### 12.1.1 Detailed Description

This file contains various BLACS helper functions.

**Author**

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University

#### 12.1.2 LICENSE



## 12.2 blacs_matrix.h File Reference

This file contains data types and functions for BLACS formatted distributed matrices.

```
#include <starneig/configuration.h>
#include <starneig/distr_matrix.h>
```

#### Data Structures

- struct starneig_blacs_descr

  *BLACS descriptor. More...*

#### BLACS contexts

- typedef int starneig_blacs_context_t

  *BLACS context.*
- starneig_blacs_context_t starneig_distr_to_blacs_context (starneig_distr_t distr)

  *Convers a data distribution to a BLACS context.*
- starneig_distr_t starneig_blacs_context_to_distr (starneig_blacs_context_t context)

  *Convers a BLACS context to a data distribution.*
- int starneig_distr_is_blacs_compatible (starneig_distr_t distr)

  *Checks whether a data distribution is BLACS compatible.*
- int starneig_distr_is_compatible_with (starneig_distr_t distr, starneig_blacs_context_t context)

  *Checks whether a data distribution is compatible with a given BLACS context.*

**BLACS descriptors**

- typedef struct starneig_blacs_descr starneig_blacs_descr_t

    *BLACS descriptor.*
- void starneig_create_blacs_matrix (int rows, int cols, int row_blksz, int col_blksz, starneig_datatype_t type, starneig_blacs_context_t context, starneig_blacs_descr_t *descr, void **local)

    *Creates a BLACS matrix with uninitialized matrix elements.*
- void starneig_destroy_blacs_matrix (starneig_blacs_descr_t *descr, void **local)

    *Destroyes a BLACS matrix.*
- void starneig_distr_matrix_to_blacs_descr (starneig_distr_matrix_t matrix, starneig_blacs_context_t context, starneig_blacs_descr_t *descr, void **local)

    *Convers a distributed matrix to a BLACS descriptor and a matching local array.*
- starneig_distr_matrix_t starneig_blacs_descr_to_distr_matrix (starneig_datatype_t type, starneig_distr_↩ t distr, starneig_blacs_descr_t *descr, void *local)

    *Convers a BLACS descriptor and a matching local array to a distributed matrix.*
- int starneig_distr_matrix_is_blacs_compatible (starneig_distr_matrix_t matrix)

    *Checks whether a distributed matrix is BLACS compatible.*
- int starneig_distr_matrix_is_compatible_with (starneig_distr_matrix_t matrix, starneig_blacs_context_t context)

    *Checks whether a distributed matrix is compatible with a given BLACS context.*

### 12.2.1 Detailed Description

This file contains data types and functions for BLACS formatted distributed matrices.

**Author**

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University

### 12.2.2 LICENSE

## 12.3 configuration.h File Reference

This file contains StarNEig library configuration.

### Macros

- #define STARNEIG_ENABLE_MPI

  *Defined if the library was compiled with MPI support.*
- #define STARNEIG_ENABLE_CUDA

  *Defined if the library was compiled with CUDA support.*
- #define STARNEIG_ENABLE_BLACS

  *Defined if the library was compiled with BLACS support.*
- #define STARNEIG_SEP_DM_HESSENBERG

  *Defined if the starneig_SEP_DM_Hessenberg() function exists.*
- #define STARNEIG_GEP_DM_HESSENBERGTRIANGULAR

  *Defined if the starneig_GEP_DM_HessenbergTriangular() function exists.*
- #define STARNEIG_SEP_DM_REDUCE

  *Defined if the starneig_SEP_DM_Reduce() function exists.*
- #define STARNEIG_GEP_DM_REDUCE

  *Defined if the starneig_GEP_DM_Reduce() function exists.*

### 12.3.1 Detailed Description

This file contains StarNEig library configuration.

**Author**

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University

### 12.3.2 LICENSE

## 12.4 distr_matrix.h File Reference

This file contains data types and functions for distributed matrices.

```
#include <starneig/configuration.h>
#include <stddef.h>
```

#### Data Structures

- struct starneig_distr_block

    *Distributed block. More...*

#### Functions

**Query functions**

- void starneig_distr_matrix_get_blocks (starneig_distr_matrix_t matrix, struct starneig_distr_block **blocks, int *num_blocks)

    *Returns the locally owned distributed blocks.*
- starneig_distr_t starneig_distr_matrix_get_distr (starneig_distr_matrix_t matrix)

    *Returns the distribution that is associated with a distributed matrix.*
- starneig_datatype_t starneig_distr_matrix_get_datatype (starneig_distr_matrix_t matrix)

    *Returns the matrix element data type.*
- size_t starneig_distr_matrix_get_elemsize (starneig_distr_matrix_t matrix)

    *Returns the matrix element size.*
- int starneig_distr_matrix_get_rows (starneig_distr_matrix_t matrix)

    *Returns the number of (global) rows.*
- int starneig_distr_matrix_get_cols (starneig_distr_matrix_t matrix)

    *Returns the number of (global) columns.*
- int starneig_distr_matrix_get_row_blksz (starneig_distr_matrix_t matrix)

    *Returns the number of rows in a distribution block.*
- int starneig_distr_matrix_get_col_blksz (starneig_distr_matrix_t matrix)

    *Returns the number of columns in a distribution block.*

**Helpers**

- void starneig_broadcast (int root, size_t size, void *buffer)

    *Broadcast a buffer.*

#### Data distributions

- enum starneig_distr_order_t { STARNEIG_ORDER_DEFAULT, STARNEIG_ORDER_ROW_MAJOR, ST↩ ARNEIG_ORDER_COL_MAJOR }

    *Process mapping order.*
- typedef struct starneig_distr * starneig_distr_t

    *Data distribution.*
- starneig_distr_t starneig_distr_init ()

    *Creates a default data distribution.*
- starneig_distr_t starneig_distr_init_mesh (int rows, int cols, starneig_distr_order_t order)

    *Creates a two-dimensional block cyclic data distribution.*
- starneig_distr_t starneig_distr_init_func (int(*func)(int row, int col, void *arg), void *arg, size_t arg_size)

    *Creates a distribution using a data distribution function.*
- starneig_distr_t starneig_distr_duplicate (starneig_distr_t distr)

    *Duplicates a data distribution.*
- void starneig_distr_destroy (starneig_distr_t distr)

    *Destroys a data distribution.*

#### Distributed matrices

- enum starneig_datatype_t { STARNEIG_REAL_DOUBLE }

    *Distributed matrix element data type.*
- typedef struct starneig_distr_matrix * starneig_distr_matrix_t

    *Distributed matrix.*
- starneig_distr_matrix_t starneig_distr_matrix_create (int rows, int cols, int row_blksz, int col_blksz, starneig↩_datatype_t type, starneig_distr_t distr)

    *Creates a distributed matrix with uninitialized matrix elements.*
- starneig_distr_matrix_t starneig_distr_matrix_create_local (int rows, int cols, starneig_datatype_t type, int owner, double *A, int ldA)

    *Creates a single-owner distributed matrix from a local matrix.*
- void starneig_distr_matrix_destroy (starneig_distr_matrix_t matrix)

    *Destroys a distributed matrix.*
- void starneig_distr_matrix_copy (starneig_distr_matrix_t source, starneig_distr_matrix_t dest)

    *Copies the contents of a distributed matrix to a second distributed matrix.*
- void starneig_distr_matrix_copy_region (int sr, int sc, int dr, int dc, int rows, int cols, starneig_distr_matrix_t source, starneig_distr_matrix_t dest)

    *Copies region of a distributed matrix to a second distributed matrix.*

### 12.4.1 Detailed Description

This file contains data types and functions for distributed matrices.

#### Author

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University
Lars Karlsson (larsk@cs.umu.se), Umeå University

### 12.4.2 LICENSE

## 12.5 error.h File Reference

This file contains the library error codes.

```
#include <starneig/configuration.h>
```

#### Macros

- #define STARNEIG_SUCCESS 0

  *The interface function was executed successfully.*
- #define STARNEIG_GENERIC_ERROR 1

  *The interface function encountered a generic error.*
- #define STARNEIG_NOT_INITIALIZED 2

  *The library was not initialized when the interface function was called.*
- #define STARNEIG_INVALID_CONFIGURATION 3

  *The interface function encountered an invalid configuration argument.*
- #define STARNEIG_INVALID_ARGUMENTS 4

  *The interface function encountered an invalid argument.*
- #define STARNEIG_INVALID_DISTR_MATRIX 5

  *One or more of the involved distributed matrices have an invalid distribution, invalid dimensions and/or an invalid distributed block size.*
- #define STARNEIG_DID_NOT_CONVERGE 6

  *The interface function encountered a situation where the QR/QZ algorithm did not converge. The matrix (pencil) may be partially in Schur form.*
- #define STARNEIG_PARTIAL_REORDERING 7

  *The interface function failed to reorder the (generalized) Schur form. The (generalized) Schur form may be partially reordered.*
- #define STARNEIG_CLOSE_EIGENVALUES 8

  *The interface function encountered a situation where two selected eigenvalues were close to each other.*

#### Typedefs

- typedef int starneig_error_t

  *Interface function return value data type.*

### 12.5.1 Detailed Description

This file contains the library error codes.

#### Author

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University

#### 12.5.2 LICENSE



## 12.6 expert.h File Reference

This file contains configuration structures and functions for the expert interface functions.

```
#include <starneig/configuration.h>
```

#### Data Structures

- struct starneig_hessenberg_conf

  *Hessenberg reduction configuration structure. More...*
- struct starneig_schur_conf

  *Schur reduction configuration structure. More...*
- struct starneig_reorder_conf

  *Eigenvalue reordering configuration structure. More...*
- struct starneig_eigenvectors_conf

  *Eigenvector computation configuration structure. More...*

#### Hessenberg reduction

- #define STARNEIG_HESSENBERG_DEFAULT_TILE_SIZE -1

  *Default tile size.*
- #define STARNEIG_HESSENBERG_DEFAULT_PANEL_WIDTH -1

  *Default panel width.*
- void starneig_hessenberg_init_conf (struct starneig_hessenberg_conf *conf)

  *Initializes a Hessenberg reduction configuration structure with default parameters.*

### Schur reduction

- #define STARNEIG_SCHUR_DEFAULT_INTERATION_LIMIT -1
  *Default iteration limit.*
- #define STARNEIG_SCHUR_DEFAULT_TILE_SIZE -1
  *Default tile size.*
- #define STARNEIG_SCHUR_DEFAULT_SMALL_LIMIT -1
  *Default sequential QR limit.*
- #define STARNEIG_SCHUR_DEFAULT_AED_WINDOW_SIZE -1
  *Default AED window size.*
- #define STARNEIG_SCHUR_DEFAULT_AED_SHIFT_COUNT -1
  *Default AED shift count.*
- #define STARNEIG_SCHUR_DEFAULT_AED_NIBBLE -1
  *Default nibble value.*
- #define STARNEIG_SCHUR_DEFAULT_AED_PARALLEL_SOFT_LIMIT -1
  *Default soft sequential AED limit.*
- #define STARNEIG_SCHUR_DEFAULT_AED_PARALLEL_HARD_LIMIT -1
  *Default hard sequential AED limit.*
- #define STARNEIG_SCHUR_DEFAULT_WINDOW_SIZE -1
  *Default bulge chasing window size.*
- #define STARNEIG_SCHUR_ROUNDED_WINDOW_SIZE -2
  *Rounded bulge chasing window.*
- #define STARNEIG_SCHUR_DEFAULT_SHIFTS_PER_WINDOW -1
  *Default number of shifts per bulge chasing window.*
- #define STARNEIG_SCHUR_DEFAULT_UPDATE_WIDTH -1
  *Default left-hand side update width.*
- #define STARNEIG_SCHUR_DEFAULT_UPDATE_HEIGHT -1
  *Default right-hand side update height.*
- #define STARNEIG_SCHUR_DEFAULT_THRESHOLD -1
  *Default deflation threshold.*
- #define STARNEIG_SCHUR_NORM_STABLE_THRESHOLD -2
  *Norm stable deflation threshold.*
- #define STARNEIG_SCHUR_LAPACK_THRESHOLD -3
  *LAPACK-style deflation threshold.*
- void starneig_schur_init_conf (struct starneig_schur_conf ∗conf)
  *Initializes a Schur reduction configuration structure with default parameters.*

### Eigenvalue reordering

- #define STARNEIG_REORDER_DEFAULT_UPDATE_WIDTH -1
  *Default left-hand side update task width.*
- #define STARNEIG_REORDER_DEFAULT_UPDATE_HEIGHT -1
  *Default right-hand side update task height.*
- #define STARNEIG_REORDER_DEFAULT_TILE_SIZE -1
  *Default tile size.*
- #define STARNEIG_REORDER_DEFAULT_VALUES_PER_CHAIN -1
  *Default number of selected eigenvalues per window.*
- #define STARNEIG_REORDER_DEFAULT_WINDOW_SIZE -1
  *Default default window size.*
- #define STARNEIG_REORDER_ROUNDED_WINDOW_SIZE -2

*Default rounded window size.*

- #define STARNEIG_REORDER_DEFAULT_SMALL_WINDOW_SIZE -1

  *Default small window size.*
- #define STARNEIG_REORDER_DEFAULT_SMALL_WINDOW_THRESHOLD -1

  *Default small window threshold.*
- enum starneig_reorder_plan_t { STARNEIG_REORDER_DEFAULT_PLAN = 1, STARNEIG_REORDER_←ONE_PART_PLAN = 2, STARNEIG_REORDER_MULTI_PART_PLAN = 3 }

  *Reordering plan enumerator.*
- enum starneig_reorder_blueprint_t {
  STARNEIG_REORDER_DEFAULT_BLUEPRINT = 1, STARNEIG_REORDER_DUMMY_INSERT_A = 2,
  STARNEIG_REORDER_DUMMY_INSERT_B = 3, STARNEIG_REORDER_CHAIN_INSERT_A = 4,
  STARNEIG_REORDER_CHAIN_INSERT_B = 5, STARNEIG_REORDER_CHAIN_INSERT_C = 6, STAR←NEIG_REORDER_CHAIN_INSERT_D = 7, STARNEIG_REORDER_CHAIN_INSERT_E = 8,
  STARNEIG_REORDER_CHAIN_INSERT_F = 9 }

  *Task insertion blueprint.*
- void starneig_reorder_init_conf (struct starneig_reorder_conf ∗conf)

  *Initializes an eigenvalue reordering configuration structure with default parameters.*

**Eigenvectors**

- #define STARNEIG_EIGENVECTORS_DEFAULT_TILE_SIZE -1

  *Default tile size.*
- void starneig_eigenvectors_init_conf (struct starneig_eigenvectors_conf ∗conf)

  *Initializes an eigenvectors configuration structure with default parameters.*

### 12.6.1 Detailed Description

This file contains configuration structures and functions for the expert interface functions.

**Author**

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University
Angelika Schwarz (angies@cs.umu.se), Umeå University

### 12.6.2 LICENSE

## 12.7 gep_dm.h File Reference

This file contains distributed memory interface functions for generalized eigenvalue problems.

```
#include <starneig/configuration.h>
#include <starneig/error.h>
#include <starneig/expert.h>
#include <starneig/distr_matrix.h>
```

### Functions

#### Computational functions

- starneig_error_t starneig_GEP_DM_HessenbergTriangular (starneig_distr_matrix_t A, starneig_distr_matrix_t B, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z)
  *Computes a Hessenberg-triangular decomposition of a general matrix pencil.*
- starneig_error_t starneig_GEP_DM_Schur (starneig_distr_matrix_t H, starneig_distr_matrix_t T, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z, double real[ ], double imag[ ], double beta[ ])
  *Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.*
- starneig_error_t starneig_GEP_DM_ReorderSchur (int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t T, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z, double real[ ], double imag[ ], double beta[ ])
  *Reorders selected generalized eigenvalues to the top left corner of a generalized Schur decomposition.*
- starneig_error_t starneig_GEP_DM_Reduce (starneig_distr_matrix_t A, starneig_distr_matrix_t B, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z, double real[ ], double imag[ ], double beta[ ], int(∗predicate)(double real, double imag, double beta, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)
  *Computes a (reordered) generalized Schur decomposition given a general matrix pencil.*
- starneig_error_t starneig_GEP_DM_Eigenvectors (int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t T, starneig_distr_matrix_t Z, starneig_distr_matrix_t X)
  *Computes a generalized eigenvector for each selected generalized eigenvalue.*

#### Helper functions

- starneig_error_t starneig_GEP_DM_Select (starneig_distr_matrix_t S, starneig_distr_matrix_t T, int(∗predicate)(double real, double imag, double beta, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)
  *Generates a selection array for a Schur-triangular matrix pencil using a user-supplied predicate function.*

#### Expert computational functions

- starneig_error_t starneig_GEP_DM_Schur_expert (struct starneig_schur_conf ∗conf, starneig_distr_matrix_t H, starneig_distr_matrix_t T, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z, double real[ ], double imag[ ], double beta[ ])
  *Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.*
- starneig_error_t starneig_GEP_DM_ReorderSchur_expert (struct starneig_reorder_conf ∗conf, int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t T, starneig_distr_matrix_t Q, starneig_distr_matrix_t Z, double real[ ], double imag[ ], double beta[ ])
  *Reorders selected generalized eigenvalues to the top left corner of a generalized Schur decomposition.*
- starneig_error_t starneig_GEP_DM_Eigenvectors_expert (struct starneig_eigenvectors_conf ∗conf, int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t T, starneig_distr_matrix_t Z, starneig_distr_matrix_t X)
  *Computes a generalized eigenvector for each selected generalized eigenvalue.*

#### 12.7.1 Detailed Description

This file contains distributed memory interface functions for generalized eigenvalue problems.

**Author**

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University
Lars Karlsson (larsk@cs.umu.se), Umeå University

#### 12.7.2 LICENSE



## 12.8 gep_sm.h File Reference

This file contains shared memory interface functions for generalized eigenvalue problems.

```
#include <starneig/configuration.h>
#include <starneig/error.h>
#include <starneig/expert.h>
```

### Functions

#### Computational functions

- starneig_error_t starneig_GEP_SM_HessenbergTriangular (int n, double A[ ], int ldA, double B[ ], int ldB, double Q[ ], int ldQ, double Z[ ], int ldZ)
  *Computes a Hessenberg-triangular decomposition of a general matrix pencil.*
- starneig_error_t starneig_GEP_SM_Schur (int n, double H[ ], int ldH, double T[ ], int ldT, double Q[ ], int ldQ, double Z[ ], int ldZ, double real[ ], double imag[ ], double beta[ ])
  *Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.*
- starneig_error_t starneig_GEP_SM_ReorderSchur (int n, int selected[ ], double S[ ], int ldS, double T[ ], int ldT, double Q[ ], int ldQ, double Z[ ], int ldZ, double real[ ], double imag[ ], double beta[ ])
  *Reorders selected generalized eigenvalues to the top left corner of a generalized Schur decomposition.*
- starneig_error_t starneig_GEP_SM_Reduce (int n, double A[ ], int ldA, double B[ ], int ldB, double Q[ ], int ldQ, double Z[ ], int ldZ, double real[ ], double imag[ ], double beta[ ], int(∗predicate)(double real, double imag, double beta, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)
  *Computes a (reordered) generalized Schur decomposition given a general matrix pencil.*
- starneig_error_t starneig_GEP_SM_Eigenvectors (int n, int selected[ ], double S[ ], int ldS, double T[ ], int ldT, double Z[ ], int ldZ, double X[ ], int ldX)
  *Computes a generalized eigenvector for each selected generalized eigenvalue.*

#### Helper functions

- starneig_error_t starneig_GEP_SM_Select (int n, double S[ ], int ldS, double T[ ], int ldT, int(∗predicate)(double real, double imag, double beta, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)
  *Generates a selection array for a Schur-triangular matrix pencil using a user-supplied predicate function.*

#### Expert computational functions

- starneig_error_t starneig_GEP_SM_Schur_expert (struct starneig_schur_conf ∗conf, int n, double H[ ], int ldH, double T[ ], int ldT, double Q[ ], int ldQ, double Z[ ], int ldZ, double real[ ], double imag[ ], double beta[ ])
  *Computes a generalized Schur decomposition given a Hessenberg-triangular decomposition.*
- starneig_error_t starneig_GEP_SM_ReorderSchur_expert (struct starneig_reorder_conf ∗conf, int n, int selected[ ], double S[ ], int ldS, double T[ ], int ldT, double Q[ ], int ldQ, double Z[ ], int ldZ, double real[ ], double imag[ ], double beta[ ])
  *Reorders selected eigenvalues to the top left corner of a generalized Schur decomposition.*
- starneig_error_t starneig_GEP_SM_Eigenvectors_expert (struct starneig_eigenvectors_conf ∗conf, int n, int selected[ ], double S[ ], int ldS, double T[ ], int ldT, double Z[ ], int ldZ, double X[ ], int ldX)
  *Computes a generalized eigenvector for each selected generalized eigenvalue.*

### 12.8.1 Detailed Description

This file contains shared memory interface functions for generalized eigenvalue problems.

**Author**

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University
Lars Karlsson (larsk@cs.umu.se), Umeå University

#### 12.8.2 LICENSE



## 12.9 node.h File Reference

This file contains interface to configure the intra-node execution environment.

```
#include <starneig/configuration.h>
#include <mpi.h>
```

#### Functions

- void starneig_node_init (int cores, int gpus, starneig_flag_t flags)

  *Initializes the intra-node execution environment.*
- int starneig_node_initialized ()

  *Checks whether the intra-node execution environment is initialized.*
- int starneig_node_get_cores ()

  *Returns the number of cores (threads) per MPI rank.*
- void starneig_node_set_cores (int cores)

  *Changes the number of CPUs cores (threads) to use per MPI rank.*
- int starneig_node_get_gpus ()

  *Returns the number of GPUs per MPI rank.*
- void starneig_node_set_gpus (int gpus)

  *Changes the number of GPUs to use per MPI rank.*
- void starneig_node_finalize ()

  *Deallocates resources associated with the intra-node configuration.*

#### Pinned host memory

- void starneig_node_enable_pinning ()
  *Enable CUDA host memory pinning.*
- void starneig_node_disable_pinning ()
  *Disables CUDA host memory pinning.*

#### Distributed memory

- void starneig_mpi_set_comm (MPI_Comm comm)
  *Sets a MPI communicator for the library.*
- MPI_Comm starneig_mpi_get_comm ()
  *Returns the library MPI communicator.*

#### Library initialization flags

- #define STARNEIG_DEFAULT 0x0
  *Default initialization flag.*
- #define STARNEIG_HINT_SM 0x0
  *Initializes the library for shared memory computation.*
- #define STARNEIG_HINT_DM 0x1
  *Initializes the library for distributed memory computation.*
- #define STARNEIG_FXT_DISABLE 0x2
  *Disables FXT traces.*
- #define STARNEIG_AWAKE_WORKERS 0x4
  *Keeps worker threads awake.*
- #define STARNEIG_AWAKE_MPI_WORKER 0x8
  *Keeps StarPU-MPI communication thread awake.*
- #define STARNEIG_FAST_DM (STARNEIG_HINT_DM | STARNEIG_AWAKE_WORKERS | STARNEIG_AWAKE_MPI_WORKER)
  *Enables fast StarPU-MPI mode.*
- #define STARNEIG_NO_VERBOSE 0x10
  *Disables verbose messages.*
- #define STARNEIG_NO_MESSAGES (STARNEIG_NO_VERBOSE | 0x20)
  *Disables messages.*
- typedef unsigned starneig_flag_t
  *Library initialization flag data type.*

### 12.9.1 Detailed Description

This file contains interface to configure the intra-node execution environment.

#### Author

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University
Lars Karlsson (larsk@cs.umu.se), Umeå University

#### 12.9.2 LICENSE



## 12.10 sep_dm.h File Reference

This file contains distributed memory interface functions for standard eigenvalue problems.

```
#include <starneig/configuration.h>
#include <starneig/error.h>
#include <starneig/expert.h>
#include <starneig/distr_matrix.h>
```

#### Functions

**Computational functions**

- starneig_error_t starneig_SEP_DM_Hessenberg (starneig_distr_matrix_t A, starneig_distr_matrix_t Q)

  *Computes a Hessenberg decomposition of a general matrix.*
- starneig_error_t starneig_SEP_DM_Schur (starneig_distr_matrix_t H, starneig_distr_matrix_t Q, double real[ ], double imag[ ])

  *Computes a Schur decomposition given a Hessenberg decomposition.*
- starneig_error_t starneig_SEP_DM_ReorderSchur (int selected[ ], starneig_distr_matrix_t S, starneig_← distr_matrix_t Q, double real[ ], double imag[ ])

  *Reorders selected eigenvalues to the top left corner of a Schur decomposition.*
- starneig_error_t starneig_SEP_DM_Reduce (starneig_distr_matrix_t A, starneig_distr_matrix_t Q, double real[ ], double imag[ ], int(∗predicate)(double real, double imag, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)

  *Computes a (reordered) Schur decomposition of a general matrix.*
- starneig_error_t starneig_SEP_DM_Eigenvectors (int selected[ ], starneig_distr_matrix_t S, starneig_← distr_matrix_t Q, starneig_distr_matrix_t X)

*Computes an eigenvector for each selected eigenvalue.*

### Helper functions

- starneig_error_t starneig_SEP_DM_Select (starneig_distr_matrix_t S, int(*predicate)(double real, double imag, void *arg), void *arg, int selected[ ], int *num_selected)
  *Generates a selection array for a Schur matrix using a user-supplied predicate function.*

### Expert computational functions

- starneig_error_t starneig_SEP_DM_Schur_expert (struct starneig_schur_conf *conf, starneig_distr_matrix_t H, starneig_distr_matrix_t Q, double real[ ], double imag[ ])
  *Computes a Schur decomposition given a Hessenberg decomposition.*
- starneig_error_t starneig_SEP_DM_ReorderSchur_expert (struct starneig_reorder_conf *conf, int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t Q, double real[ ], double imag[ ])
  *Reorders selected eigenvalues to the top left corner of a Schur decomposition.*
- starneig_error_t starneig_SEP_DM_Eigenvectors_expert (struct starneig_eigenvectors_conf *conf, int selected[ ], starneig_distr_matrix_t S, starneig_distr_matrix_t Q, starneig_distr_matrix_t X)
  *Computes an eigenvector for each selected eigenvalue.*

#### 12.10.1 Detailed Description

This file contains distributed memory interface functions for standard eigenvalue problems.

**Author**

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University
Lars Karlsson (larsk@cs.umu.se), Umeå University

#### 12.10.2 LICENSE

## 12.11 sep_sm.h File Reference

This file contains shared memory interface functions for standard eigenvalue problems.

```
#include <starneig/configuration.h>
#include <starneig/error.h>
#include <starneig/expert.h>
```

### Functions

#### Computational functions

- starneig_error_t starneig_SEP_SM_Hessenberg (int n, double A[ ], int ldA, double Q[ ], int ldQ)
  *Computes a Hessenberg decomposition of a general matrix.*
- starneig_error_t starneig_SEP_SM_Schur (int n, double H[ ], int ldH, double Q[ ], int ldQ, double real[ ], double imag[ ])
  *Computes a Schur decomposition given a Hessenberg decomposition.*
- starneig_error_t starneig_SEP_SM_ReorderSchur (int n, int selected[ ], double S[ ], int ldS, double Q[ ], int ldQ, double real[ ], double imag[ ])
  *Reorders selected eigenvalues to the top left corner of a Schur decomposition.*
- starneig_error_t starneig_SEP_SM_Reduce (int n, double A[ ], int ldA, double Q[ ], int ldQ, double real[ ], double imag[ ], int(∗predicate)(double real, double imag, void ∗arg), void ∗arg, int selected[ ], int ∗num_↩ selected)
  *Computes a (reordered) Schur decomposition of a general matrix.*
- starneig_error_t starneig_SEP_SM_Eigenvectors (int n, int selected[ ], double S[ ], int ldS, double Q[ ], int ldQ, double X[ ], int ldX)
  *Computes an eigenvector for each selected eigenvalue.*

#### Helper functions

- starneig_error_t starneig_SEP_SM_Select (int n, double S[ ], int ldS, int(∗predicate)(double real, double imag, void ∗arg), void ∗arg, int selected[ ], int ∗num_selected)
  *Generates a selection array for a Schur matrix using a user-supplied predicate function.*

#### Expert computational functions

- starneig_error_t starneig_SEP_SM_Hessenberg_expert (struct starneig_hessenberg_conf ∗conf, int n, int begin, int end, double A[ ], int ldA, double Q[ ], int ldQ)
  *Computes a Hessenberg decomposition of a general matrix.*
- starneig_error_t starneig_SEP_SM_Schur_expert (struct starneig_schur_conf ∗conf, int n, double H[ ], int ldH, double Q[ ], int ldQ, double real[ ], double imag[ ])
  *Computes a Schur decomposition given a Hessenberg decomposition.*
- starneig_error_t starneig_SEP_SM_ReorderSchur_expert (struct starneig_reorder_conf ∗conf, int n, int selected[ ], double S[ ], int ldS, double Q[ ], int ldQ, double real[ ], double imag[ ])
  *Reorders selected eigenvalues to the top left corner of a Schur decomposition.*
- starneig_error_t starneig_SEP_SM_Eigenvectors_expert (struct starneig_eigenvectors_conf ∗conf, int n, int selected[ ], double S[ ], int ldS, double Q[ ], int ldQ, double X[ ], int ldX)
  *Computes an eigenvector for each selected eigenvalue.*

### 12.11.1 Detailed Description

This file contains shared memory interface functions for standard eigenvalue problems.

**Author**

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University
Lars Karlsson (larsk@cs.umu.se), Umeå University

#### 12.11.2 LICENSE



### 12.12 starneig.h File Reference

This file includes most StarNEig header files.

```
#include <starneig/configuration.h>
#include <starneig/node.h>
#include <starneig/gep_sm.h>
#include <starneig/sep_sm.h>
#include <starneig/gep_dm.h>
#include <starneig/sep_dm.h>
#include <starneig/blacs_helpers.h>
#include <starneig/blacs_matrix.h>
```

#### 12.12.1 Detailed Description

This file includes most StarNEig header files.

**Author**

Mirko Myllykoski (mirkom@cs.umu.se), Umeå University

#### 12.12.2 LICENSE



# 13 Example Documentation

## 13.1 gep_dm_full_chain.c

```
#include "validate.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <mpi.h>
#include <starneig/starneig.h>

// a predicate function that selects all finate eigenvalues that have positive
// a real part
static int predicate(double real, double imag, double beta, void *arg)
{
    if (0.0 < real && beta != 0.0)
        return 1;
    return 0;
}

int main(int argc, char **argv)
{
    const int n = 3000; // matrix dimension
    const int root = 0; // root rank

    // initialize MPI

    int thread_support;
    MPI_Init_thread(
        &argc, (char ***)&argv, MPI_THREAD_MULTIPLE, &thread_support);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // the root node initializes the matrices locally

    int ldA = 0, ldB = 0, ldQ = 0, ldZ = 0, ldC = 0, ldD = 0;
    double *A = NULL, *B = NULL, *Q = NULL, *Z = NULL, *C = NULL, *D = NULL;
    if (world_rank == root) {
```

```
    srand((unsigned) time(NULL));

    // generate a full random matrix A and a copy C

    ldA = ((n/8)+1)*8, ldC = ((n/8)+1)*8;
    A = malloc(n*ldA*sizeof(double));
    C = malloc(n*ldC*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            A[j*ldA+i] = C[j*ldC+i] = 2.0*rand()/RAND_MAX - 1.0;

    // generate a full random matrix B and a copy D

    ldB = ((n/8)+1)*8, ldD = ((n/8)+1)*8;
    B = malloc(n*ldB*sizeof(double));
    D = malloc(n*ldD*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            B[j*ldB+i] = D[j*ldD+i] = 2.0*rand()/RAND_MAX - 1.0;

    // generate an identity matrix Q

    ldQ = ((n/8)+1)*8;
    Q = malloc(n*ldA*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            Q[j*ldQ+i] = i == j ? 1.0 : 0.0;

    // generate an identity matrix Z

    ldZ = ((n/8)+1)*8;
    Z = malloc(n*ldZ*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            Z[j*ldZ+i] = i == j ? 1.0 : 0.0;
}

// allocate space for the eigenvalues and the eigenvalue selection vector

double *real = malloc(n*sizeof(double));
double *imag = malloc(n*sizeof(double));
double *beta = malloc(n*sizeof(double));
int *select = malloc(n*sizeof(int));

// Initialize the StarNEig library using a default number of CPU cores and
// GPUs. The STARNEIG_FAST_DM flag indicates that the library should
// initialize itself for distributed memory computations and keep StarPU
// worker threads and StarPU-MPI communication thread awake between
// interface function calls.

starneig_node_init(-1, -1, STARNEIG_FAST_DM);

// create a two-dimensional block cyclic distribution with column-major
// ordering

starneig_distr_t distr = starneig_distr_init_mesh(
    -1, -1, STARNEIG_ORDER_COL_MAJOR);

// Convert the local matrix A to a distributed matrix lA that is owned by
// the root node. This is done in-place, i.e., the matrices A and lA point
// to the same data.

starneig_distr_matrix_t lA =
  starneig_distr_matrix_create_local(
    n, n, STARNEIG_REAL_DOUBLE, root, A, ldA);

// create a distributed matrix dA using the earlier created data
// distribution and default distributed block size

starneig_distr_matrix_t dA =
    starneig_distr_matrix_create(n, n, -1, -1,
  STARNEIG_REAL_DOUBLE, distr);

// copy the local matrix lA to the distributed matrix dA (scatter)

starneig_distr_matrix_copy(lA, dA);

// scatter the matrix B

starneig_distr_matrix_t lB =
  starneig_distr_matrix_create_local(
    n, n, STARNEIG_REAL_DOUBLE, root, B, ldB);
starneig_distr_matrix_t dB =
    starneig_distr_matrix_create(n, n, -1, -1,
  STARNEIG_REAL_DOUBLE, distr);
starneig_distr_matrix_copy(lB, dB);
```

```
    // scatter the matrix Q

    starneig_distr_matrix_t lQ =
      starneig_distr_matrix_create_local(
        n, n, STARNEIG_REAL_DOUBLE, root, Q, ldQ);
    starneig_distr_matrix_t dQ =
        starneig_distr_matrix_create(n, n, -1, -1,
      STARNEIG_REAL_DOUBLE, distr);
    starneig_distr_matrix_copy(lQ, dQ);

    // scatter the matrix Z

    starneig_distr_matrix_t lZ =
      starneig_distr_matrix_create_local(
        n, n, STARNEIG_REAL_DOUBLE, root, Z, ldZ);
    starneig_distr_matrix_t dZ =
        starneig_distr_matrix_create(n, n, -1, -1,
      STARNEIG_REAL_DOUBLE, distr);
    starneig_distr_matrix_copy(lZ, dZ);

    // reduce the dense-dense matrix pencil (A,B) to Hessenberg-triangular form

    printf("Hessenberg-triangular reduction...\n");
    starneig_GEP_DM_HessenbergTriangular(dA, dB, dQ, dZ);

    // reduce the Hessenberg-triangular matrix pencil (A,B) to generalized Schur
    // form

    printf("Schur reduction...\n");
    starneig_GEP_DM_Schur(dA, dB, dQ, dZ, real, imag, beta);

    // select eigenvalues that have positive a real part

    int num_selected;
    starneig_GEP_DM_Select(dA, dB, &predicate, NULL, select, &num_selected);
    printf("Selected %d eigenvalues out of %d.\n", num_selected, n);

    // reorder selected eigenvalues to the upper left corner of the generalized
    // Schur form (A,B)

    printf("Reordering...\n");
    starneig_GEP_DM_ReorderSchur(select, dA, dB, dQ, dZ, real, imag, beta);

    // copy the distributed matrix dA back to the local matrix lA (gather)

    starneig_distr_matrix_copy(dA, lA);

    // free the distributed matrix lA (matrix A is not freed)

    starneig_distr_matrix_destroy(lA);

    // free the distributed matrix dA (all local resources are freed)

    starneig_distr_matrix_destroy(dA);

    // gather the matrix B

    starneig_distr_matrix_copy(dB, lB);
    starneig_distr_matrix_destroy(lB);
    starneig_distr_matrix_destroy(dB);

    // gather the matrix Q

    starneig_distr_matrix_copy(dQ, lQ);
    starneig_distr_matrix_destroy(lQ);
    starneig_distr_matrix_destroy(dQ);

    // gather the matrix Z

    starneig_distr_matrix_copy(dZ, lZ);
    starneig_distr_matrix_destroy(lZ);
    starneig_distr_matrix_destroy(dZ);

    // free the data distribution

    starneig_distr_destroy(distr);

    // de-initialize the StarNEig library

    starneig_node_finalize();

    // de-initialize MPI

    MPI_Finalize();

    if (world_rank == root) {
```

```
        // check residual || Q A Z^T - C ||_F / || C ||_F

        check_residual(n, ldQ, ldA, ldZ, ldC, Q, A, Z, C);

        // check residual || Q B Z^T - D ||_F / || D ||_F

        check_residual(n, ldQ, ldB, ldZ, ldD, Q, B, Z, D);

        // check residual || Q Q^T - I ||_F / || I ||_F

        check_orthogonality(n, ldQ, Q);

        // check residual || Z Z^T - I ||_F / || I ||_F

        check_orthogonality(n, ldZ, Z);
    }

    // cleanup

    free(A);
    free(C);
    free(B);
    free(D);
    free(Q);
    free(Z);

    free(real);
    free(imag);
    free(beta);
    free(select);

    return 0;
}
```

## 13.2 gep_sm_eigenvectors.c

```
#include "validate.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <starneig/starneig.h>

// a predicate function that selects all finite eigenvalues that have positive
// a real part
static int predicate(double real, double imag, double beta, void *arg)
{
    if (0.0 < real && beta != 0.0)
        return 1;
    return 0;
}

int main()
{
    const int n = 3000; // matrix dimension

    srand((unsigned) time(NULL));

    // generate a full random matrix A and a copy C

    int ldA = ((n/8)+1)*8, ldC = ((n/8)+1)*8;
    double *A = malloc(n*ldA*sizeof(double));
    double *C = malloc(n*ldC*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            A[j*ldA+i] = C[j*ldC+i] = 2.0*rand()/RAND_MAX - 1.0;

    // generate a full random matrix B and a copy D

    int ldB = ((n/8)+1)*8, ldD = ((n/8)+1)*8;
    double *B = malloc(n*ldB*sizeof(double));
    double *D = malloc(n*ldD*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            B[j*ldB+i] = D[j*ldD+i] = 2.0*rand()/RAND_MAX - 1.0;

    // generate an identity matrix Q

    int ldQ = ((n/8)+1)*8;
    double *Q = malloc(n*ldA*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            Q[j*ldQ+i] = i == j ? 1.0 : 0.0;
```

```
    // generate an identity matrix Z

    int ldZ = ((n/8)+1)*8;
    double *Z = malloc(n*ldZ*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            Z[j*ldZ+i] = i == j ? 1.0 : 0.0;

    double *X = NULL; int ldX = 0;

    // allocate space for the eigenvalues and the eigenvalue selection vector

    double *real = malloc(n*sizeof(double));
    double *imag = malloc(n*sizeof(double));
    double *beta = malloc(n*sizeof(double));
    int *select = malloc(n*sizeof(int));

    // Initialize the StarNEig library using a default number of CPU cores and
    // GPUs. The STARNEIG_HINT_SM flag indicates that the library should
    // initialize itself for shared memory computations.

    starneig_node_init(-1, -1, STARNEIG_HINT_SM);

    // reduce the dense-dense matrix pencil (A,B) to generalized Schur form
    // (skip reordering)

    printf("Reduce...\n");
    starneig_GEP_SM_Reduce(
        n, A, ldA, B, ldB, Q, ldQ, Z, ldZ, real, imag, beta,
        NULL, NULL, NULL, NULL);

    // select eigenvalues that have positive a real part and allocate space for
    // the eigenvectors

    int num_selected;
    starneig_GEP_SM_Select(
        n, A, ldA, B, ldB, &predicate, NULL, select, &num_selected);
    printf("Selected %d eigenvalues out of %d.\n", num_selected, n);

    ldX = ((n/8)+1)*8;
    X = malloc(num_selected*ldX*sizeof(double));

    // compute a selected set of eigenvectors

    printf("Eigenvectors...\n");
    starneig_GEP_SM_Eigenvectors(n, select, A, ldA, B, ldB, Q, ldQ, X, ldX);

    // de-initialize the StarNEig library

    starneig_node_finalize();

    // check residual || Q A Z^T - C ||_F / || C ||_F

    check_residual(n, ldQ, ldA, ldZ, ldC, Q, A, Z, C);

    // check residual || Q B Z^T - D ||_F / || D ||_F

    check_residual(n, ldQ, ldB, ldZ, ldD, Q, B, Z, D);

    // check residual || Q Q^T - I ||_F / || I ||_F

    check_orthogonality(n, ldQ, Q);

    // check residual || Z Z^T - I ||_F / || I ||_F

    check_orthogonality(n, ldZ, Z);

    // cleanup

    free(A);
    free(C);
    free(B);
    free(D);
    free(Q);
    free(Z);
    free(X);

    free(real);
    free(imag);
    free(beta);
    free(select);

    return 0;
}
```

## 13.3 gep_sm_full_chain.c

```
#include "validate.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <starneig/starneig.h>

// a predicate function that selects all finate eigenvalues that have positive
// a real part
static int predicate(double real, double imag, double beta, void *arg)
{
    if (0.0 < real && beta != 0.0)
        return 1;
    return 0;
}

int main()
{
    const int n = 3000; // matrix dimension

    srand((unsigned) time(NULL));

    // generate a full random matrix A and a copy C

    int ldA = ((n/8)+1)*8, ldC = ((n/8)+1)*8;
    double *A = malloc(n*ldA*sizeof(double));
    double *C = malloc(n*ldC*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            A[j*ldA+i] = C[j*ldC+i] = 2.0*rand()/RAND_MAX - 1.0;

    // generate a full random matrix B and a copy D

    int ldB = ((n/8)+1)*8, ldD = ((n/8)+1)*8;
    double *B = malloc(n*ldB*sizeof(double));
    double *D = malloc(n*ldD*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            B[j*ldB+i] = D[j*ldD+i] = 2.0*rand()/RAND_MAX - 1.0;

    // generate an identity matrix Q

    int ldQ = ((n/8)+1)*8;
    double *Q = malloc(n*ldA*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            Q[j*ldQ+i] = i == j ? 1.0 : 0.0;

    // generate an identity matrix Z

    int ldZ = ((n/8)+1)*8;
    double *Z = malloc(n*ldZ*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            Z[j*ldZ+i] = i == j ? 1.0 : 0.0;

    // allocate space for the eigenvalues and the eigenvalue selection vector

    double *real = malloc(n*sizeof(double));
    double *imag = malloc(n*sizeof(double));
    double *beta = malloc(n*sizeof(double));
    int *select = malloc(n*sizeof(int));

    // Initialize the StarNEig library using a default number of CPU cores and
    // GPUs. The STARNEIG_HINT_SM flag indicates that the library should
    // initialize itself for shared memory computations and the
    // STARNEIG_AWAKE_WORKERS indicates that the library should should keep
    // StarPU worker threads awake between interface function calls.

    starneig_node_init(-1, -1, STARNEIG_HINT_SM |
      STARNEIG_AWAKE_WORKERS);

    // reduce the dense-dense matrix pencil (A,B) to Hessenberg-triangular form

    printf("Hessenberg-triangular reduction...\n");
    starneig_GEP_SM_HessenbergTriangular(n, A, ldA, B, ldB, Q, ldQ, Z,
      ldZ);

    // reduce the Hessenberg-triangular matrix pencil (A,B) to generalized Schur
    // form

    printf("Schur reduction...\n");
    starneig_GEP_SM_Schur(n, A, ldA, B, ldB, Q, ldQ, Z, ldZ, real, imag, beta);
```

```
    // select eigenvalues that have positive a real part

    int num_selected;
    starneig_GEP_SM_Select(
        n, A, ldA, B, ldB, &predicate, NULL, select, &num_selected);
    printf("Selected %d eigenvalues out of %d.\n", num_selected, n);

    // reorder selected eigenvalues to the upper left corner of the generalized
    // Schur form (A,B)

    printf("Reordering...\n");
    starneig_GEP_SM_ReorderSchur(
        n, select, A, ldA, B, ldB, Q, ldQ, Z, ldZ, real, imag, beta);

    // de-initialize the StarNEig library

    starneig_node_finalize();

    // check residual || Q A Z^T - C ||_F / || C ||_F

    check_residual(n, ldQ, ldA, ldZ, ldC, Q, A, Z, C);

    // check residual || Q B Z^T - D ||_F / || D ||_F

    check_residual(n, ldQ, ldB, ldZ, ldD, Q, B, Z, D);

    // check residual || Q Q^T - I ||_F / || I ||_F

    check_orthogonality(n, ldQ, Q);

    // check residual || Z Z^T - I ||_F / || I ||_F

    check_orthogonality(n, ldZ, Z);

    // cleanup

    free(A);
    free(C);
    free(B);
    free(D);
    free(Q);
    free(Z);

    free(real);
    free(imag);
    free(beta);
    free(select);

    return 0;
}
```

## 13.4 sep_dm_full_chain.c

```
#include "validate.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <mpi.h>
#include <starneig/starneig.h>

// a predicate function that selects all eigenvalues that have positive a real
// part
static int predicate(double real, double imag, void *arg)
{
    if (0.0 < real)
        return 1;
    return 0;
}

int main(int argc, char **argv)
{
    const int n = 3000; // matrix dimension
    const int root = 0; // root rank

    // initialize MPI

    int thread_support;
    MPI_Init_thread(
        &argc, (char ***)&argv, MPI_THREAD_MULTIPLE, &thread_support);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
```

```
// the root node initializes the matrices locally

int ldA = 0, ldQ = 0, ldC = 0;
double *A = NULL, *Q = NULL, *C = NULL;
if (world_rank == root) {
    srand((unsigned) time(NULL));

    // generate a full random matrix A and a copy C

    ldA = ((n/8)+1)*8; ldC = ((n/8)+1)*8;
    A = malloc(n*ldA*sizeof(double));
    C = malloc(n*ldC*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            A[j*ldA+i] = C[j*ldC+i] = 2.0*rand()/RAND_MAX - 1.0;

    // generate an identity matrix Q

    ldQ = ((n/8)+1)*8;
    Q = malloc(n*ldA*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            Q[j*ldQ+i] = i == j ? 1.0 : 0.0;
}

// allocate space for the eigenvalues and the eigenvalue selection vector

double *real = malloc(n*sizeof(double));
double *imag = malloc(n*sizeof(double));
int *select = malloc(n*sizeof(int));

// Initialize the StarNEig library using a default number of CPU cores and
// GPUs. The STARNEIG_HINT_DM flag indicates that the library should
// initialize itself for distributed memory computations.

starneig_node_init(-1, -1, STARNEIG_HINT_DM);

// create a two-dimensional block cyclic distribution with row-major
// ordering

starneig_distr_t distr = starneig_distr_init_mesh(
    -1, -1, STARNEIG_ORDER_ROW_MAJOR);

// Convert the local matrix A to a distributed matrix lA that is owned by
// the root node. This is done in-place, i.e., the matrices A and lA point
// to the same data.

starneig_distr_matrix_t lA =
  starneig_distr_matrix_create_local(
    n, n, STARNEIG_REAL_DOUBLE, root, A, ldA);

// create a distributed matrix dA using default data distribution and
// distributed block size

starneig_distr_matrix_t dA =
    starneig_distr_matrix_create(n, n, -1, -1,
  STARNEIG_REAL_DOUBLE, distr);

// copy the local matrix lA to the distributed matrix dA (scatter)

starneig_distr_matrix_copy(lA, dA);

// scatter the matrix Q

starneig_distr_matrix_t lQ =
  starneig_distr_matrix_create_local(
    n, n, STARNEIG_REAL_DOUBLE, root, Q, ldQ);
starneig_distr_matrix_t dQ =
    starneig_distr_matrix_create(n, n, -1, -1,
  STARNEIG_REAL_DOUBLE, distr);
starneig_distr_matrix_copy(lQ, dQ);

// reduce the full matrix dA to upper Hessenberg form

printf("Hessenberg reduction...\n");
starneig_SEP_DM_Hessenberg(dA, dQ);

// reduce the upper Hessenberg matrix dA to Schur form

printf("Schur reduction...\n");
starneig_SEP_DM_Schur(dA, dQ, real, imag);

// select eigenvalues that have positive a real part

int num_selected;
starneig_SEP_DM_Select(dA, &predicate, NULL, select, &num_selected);
```

```
    printf("Selected %d eigenvalues out of %d.\n", num_selected, n);

    // reorder the selected eigenvalues to the upper left corner of the matrix
    // dA

    printf("Reordering...\n");
    starneig_SEP_DM_ReorderSchur(select, dA, dQ, real, imag);

    // copy the distributed matrix dA back to the local matrix lA (gather)

    starneig_distr_matrix_copy(dA, lA);

    // free the distributed matrix lA (matrix A is not freed)

    starneig_distr_matrix_destroy(lA);

    // free the distributed matrix dA (all local resources are freed)

    starneig_distr_matrix_destroy(dA);

    // gather the matrix Q

    starneig_distr_matrix_copy(dQ, lQ);
    starneig_distr_matrix_destroy(lQ);
    starneig_distr_matrix_destroy(dQ);

    // free the data distribution

    starneig_distr_destroy(distr);

    // de-initialize the StarNEig library

    starneig_node_finalize();

    // de-initialize MPI

    MPI_Finalize();

    if (world_rank == root) {

        // check residual || Q A Q^T - C ||_F / || C ||_F

        check_residual(n, ldQ, ldA, ldQ, ldC, Q, A, Q, C);

        // check residual || Q Q^T - I ||_F / || I ||_F

        check_orthogonality(n, ldQ, Q);
    }

    // cleanup

    free(A);
    free(C);
    free(Q);

    free(real);
    free(imag);
    free(select);

    return 0;
}
```

## 13.5 sep_sm_eigenvectors.c

```
#include "validate.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <starneig/starneig.h>

// a predicate function that selects all eigenvalues that have positive a real
// part
static int predicate(double real, double imag, void *arg)
{
    if (0.0 < real)
        return 1;
    return 0;
}

int main()
{
    const int n = 3000; // matrix dimension
```

```
    srand((unsigned) time(NULL));

    // generate a full random matrix A and a copy C

    int ldA = ((n/8)+1)*8, ldC = ((n/8)+1)*8;
    double *A = malloc(n*ldA*sizeof(double));
    double *C = malloc(n*ldC*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            A[j*ldA+i] = C[j*ldC+i] = 2.0*rand()/RAND_MAX - 1.0;

    // generate an identity matrix Q

    int ldQ = ((n/8)+1)*8;
    double *Q = malloc(n*ldA*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            Q[j*ldQ+i] = i == j ? 1.0 : 0.0;

    double *X = NULL; int ldX = 0;

    // allocate space for the eigenvalues and the eigenvector selection vector

    double *real = malloc(n*sizeof(double));
    double *imag = malloc(n*sizeof(double));
    int *select = malloc(n*sizeof(int));

    // Initialize the StarNEig library using a default number of CPU cores and
    // GPUs. The STARNEIG_HINT_SM flag indicates that the library should
    // initialize itself for shared memory computations.

    starneig_node_init(-1, -1, STARNEIG_HINT_SM);

    // reduce the full matrix matrix A to Schur form (skip reordering)

    printf("Reduce...\n");
    starneig_SEP_SM_Reduce(
        n, A, ldA, Q, ldQ, real, imag, NULL, NULL, NULL, NULL);

    // select eigenvalues that have positive a real part and allocate space for
    // the eigenvectors

    int num_selected;
    starneig_SEP_SM_Select(n, A, ldA, &predicate, NULL, select, &num_selected);
    printf("Selected %d eigenvalues out of %d.\n", num_selected, n);

    ldX = ((n/8)+1)*8;
    X = malloc(num_selected*ldX*sizeof(double));

    // compute a selected set of eigenvectors

    printf("Eigenvectors...\n");
    starneig_SEP_SM_Eigenvectors(n, select, A, ldA, Q, ldQ, X, ldX);

    // de-initialize the StarNEig library

    starneig_node_finalize();

    // check residual || Q A Q^T - C ||_F / || C ||_F

    check_residual(n, ldQ, ldA, ldQ, ldC, Q, A, Q, C);

    // check residual || Q Q^T - I ||_F / || I ||_F

    check_orthogonality(n, ldQ, Q);

    // cleanup

    free(A);
    free(C);
    free(Q);
    free(X);

    free(real);
    free(imag);
    free(select);

    return 0;
}
```

## 13.6 sep_sm_full_chain.c

```c
#include "validate.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <starneig/starneig.h>

// a predicate function that selects all eigenvalues that have positive a real
// part
static int predicate(double real, double imag, void *arg)
{
    if (0.0 < real)
        return 1;
    return 0;
}

int main()
{
    const int n = 3000; // matrix dimension

    srand((unsigned) time(NULL));

    // generate a full random matrix A and a copy C

    int ldA = ((n/8)+1)*8, ldC = ((n/8)+1)*8;
    double *A = malloc(n*ldA*sizeof(double));
    double *C = malloc(n*ldC*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            A[j*ldA+i] = C[j*ldC+i] = 2.0*rand()/RAND_MAX - 1.0;

    // generate an identity matrix Q

    int ldQ = ((n/8)+1)*8;
    double *Q = malloc(n*ldA*sizeof(double));
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            Q[j*ldQ+i] = i == j ? 1.0 : 0.0;

    // allocate space for the eigenvalues and the eigenvalue selection vector

    double *real = malloc(n*sizeof(double));
    double *imag = malloc(n*sizeof(double));
    int *select = malloc(n*sizeof(int));

    // Initialize the StarNEig library using a default number of CPU cores and
    // GPUs. The STARNEIG_HINT_SM flag indicates that the library should
    // initialize itself for shared memory computations.

    starneig_node_init(-1, -1, STARNEIG_HINT_SM);

    // reduce the full matrix matrix A to upper Hessenberg form

    printf("Hessenberg reduction...\n");
    starneig_SEP_SM_Hessenberg(n, A, ldA, Q, ldQ);

    // reduce the upper Hessenberg matrix A to Schur form

    printf("Schur reduction...\n");
    starneig_SEP_SM_Schur(n, A, ldA, Q, ldQ, real, imag);

    // select eigenvalues that have positive a real part

    int num_selected;
    starneig_SEP_SM_Select(n, A, ldA, &predicate, NULL, select, &num_selected);
    printf("Selected %d eigenvalues out of %d.\n", num_selected, n);

    // reorder the selected eigenvalues to the upper left corner of the matrix A

    printf("Reordering...\n");
    starneig_SEP_SM_ReorderSchur(n, select, A, ldA, Q, ldQ, real, imag);

    // de-initialize the StarNEig library

    starneig_node_finalize();

    // check residual || Q A Q^T - C ||_F / || C ||_F

    check_residual(n, ldQ, ldA, ldQ, ldC, Q, A, Q, C);

    // check residual || Q Q^T - I ||_F / || I ||_F

    check_orthogonality(n, ldQ, Q);

    // cleanup

    free(A);
    free(C);
```

```
    free(Q);

    free(real);
    free(imag);
    free(select);

    return 0;
}
```

# Index

blacs_helpers.h, 106
blacs_matrix.h, 107

configuration.h, 109

distr_matrix.h, 110
Distributed Memory / Distributed matrices, 56
    starneig_broadcast, 66
    starneig_datatype_t, 58
    starneig_distr_destroy, 60
    starneig_distr_duplicate, 60
    starneig_distr_init, 58
    starneig_distr_init_func, 59
    starneig_distr_init_mesh, 58
    starneig_distr_matrix_copy, 62
    starneig_distr_matrix_copy_region, 63
    starneig_distr_matrix_create, 60
    starneig_distr_matrix_create_local, 61
    starneig_distr_matrix_destroy, 62
    starneig_distr_matrix_get_blocks, 63
    starneig_distr_matrix_get_col_blksz, 66
    starneig_distr_matrix_get_cols, 65
    starneig_distr_matrix_get_datatype, 64
    starneig_distr_matrix_get_distr, 64
    starneig_distr_matrix_get_elemsize, 64
    starneig_distr_matrix_get_row_blksz, 65
    starneig_distr_matrix_get_rows, 65
    starneig_distr_order_t, 57
Distributed Memory / Generalized EVP, 75
    starneig_GEP_DM_Eigenvectors, 79
    starneig_GEP_DM_Eigenvectors_expert, 82
    starneig_GEP_DM_HessenbergTriangular, 76
    starneig_GEP_DM_Reduce, 78
    starneig_GEP_DM_ReorderSchur, 77
    starneig_GEP_DM_ReorderSchur_expert, 81
    starneig_GEP_DM_Schur, 76
    starneig_GEP_DM_Schur_expert, 80
    starneig_GEP_DM_Select, 79
Distributed Memory / Standard EVP, 67
    starneig_SEP_DM_Eigenvectors, 70
    starneig_SEP_DM_Eigenvectors_expert, 74
    starneig_SEP_DM_Hessenberg, 67
    starneig_SEP_DM_Reduce, 69
    starneig_SEP_DM_ReorderSchur, 69
    starneig_SEP_DM_ReorderSchur_expert, 73
    starneig_SEP_DM_Schur, 68
    starneig_SEP_DM_Schur_expert, 71
    starneig_SEP_DM_Select, 71

Error codes, 29
error.h, 112
Expert configuration structures, 83
    starneig_eigenvectors_init_conf, 94
    starneig_hessenberg_init_conf, 93
    starneig_reorder_blueprint_t, 92
    starneig_reorder_init_conf, 93
    starneig_reorder_plan_t, 89
    starneig_schur_init_conf, 93
expert.h, 113

gep_dm.h, 116
gep_sm.h, 117

Intra-node execution environment, 30
    STARNEIG_AWAKE_MPI_WORKER, 32
    STARNEIG_AWAKE_WORKERS, 32
    STARNEIG_DEFAULT, 31
    STARNEIG_FAST_DM, 32
    STARNEIG_FXT_DISABLE, 31
    STARNEIG_HINT_DM, 31
    STARNEIG_HINT_SM, 31
    STARNEIG_NO_MESSAGES, 32
    STARNEIG_NO_VERBOSE, 32
    starneig_mpi_get_comm, 35
    starneig_mpi_set_comm, 35
    starneig_node_disable_pinning, 35
    starneig_node_enable_pinning, 34
    starneig_node_get_cores, 33
    starneig_node_get_gpus, 34
    starneig_node_init, 33
    starneig_node_initialized, 33
    starneig_node_set_cores, 34
    starneig_node_set_gpus, 34

Library configuration, 28

node.h, 119

STARNEIG_AWAKE_MPI_WORKER
    Intra-node execution environment, 32
STARNEIG_AWAKE_WORKERS
    Intra-node execution environment, 32
STARNEIG_DEFAULT
    Intra-node execution environment, 31
STARNEIG_FAST_DM
    Intra-node execution environment, 32
STARNEIG_FXT_DISABLE
    Intra-node execution environment, 31
STARNEIG_HINT_DM
    Intra-node execution environment, 31
STARNEIG_HINT_SM
    Intra-node execution environment, 31
STARNEIG_NO_MESSAGES
    Intra-node execution environment, 32
STARNEIG_NO_VERBOSE
    Intra-node execution environment, 32
ScaLAPACK compatibility / BLACS helpers, 101
    starneig_blacs_exit, 104
    starneig_blacs_get, 102
    starneig_blacs_gridexit, 103
    starneig_blacs_gridinfo, 102
    starneig_blacs_gridinit, 102

starneig_blacs_pcoord, 103
starneig_blacs_pinfo, 101
starneig_descinit, 104
starneig_numroc, 104
ScaLAPACK compatibility / BLACS matrices, 95
starneig_blacs_context_to_distr, 96
starneig_blacs_descr_to_distr_matrix, 99
starneig_create_blacs_matrix, 97
starneig_destroy_blacs_matrix, 98
starneig_distr_is_blacs_compatible, 97
starneig_distr_is_compatible_with, 97
starneig_distr_matrix_is_blacs_compatible, 99
starneig_distr_matrix_is_compatible_with, 100
starneig_distr_matrix_to_blacs_descr, 98
starneig_distr_to_blacs_context, 96
sep_dm.h, 121
sep_sm.h, 123
Shared Memory / Generalized EVP, 45
starneig_GEP_SM_Eigenvectors, 50
starneig_GEP_SM_Eigenvectors_expert, 54
starneig_GEP_SM_HessenbergTriangular, 45
starneig_GEP_SM_Reduce, 48
starneig_GEP_SM_ReorderSchur, 47
starneig_GEP_SM_ReorderSchur_expert, 53
starneig_GEP_SM_Schur, 46
starneig_GEP_SM_Schur_expert, 52
starneig_GEP_SM_Select, 51
Shared Memory / Standard EVP, 36
starneig_SEP_SM_Eigenvectors, 39
starneig_SEP_SM_Eigenvectors_expert, 43
starneig_SEP_SM_Hessenberg, 37
starneig_SEP_SM_Hessenberg_expert, 41
starneig_SEP_SM_Reduce, 38
starneig_SEP_SM_ReorderSchur, 38
starneig_SEP_SM_ReorderSchur_expert, 42
starneig_SEP_SM_Schur, 37
starneig_SEP_SM_Schur_expert, 42
starneig_SEP_SM_Select, 40
starneig.h, 124
starneig_GEP_DM_Eigenvectors
Distributed Memory / Generalized EVP, 79
starneig_GEP_DM_Eigenvectors_expert
Distributed Memory / Generalized EVP, 82
starneig_GEP_DM_HessenbergTriangular
Distributed Memory / Generalized EVP, 76
starneig_GEP_DM_Reduce
Distributed Memory / Generalized EVP, 78
starneig_GEP_DM_ReorderSchur
Distributed Memory / Generalized EVP, 77
starneig_GEP_DM_ReorderSchur_expert
Distributed Memory / Generalized EVP, 81
starneig_GEP_DM_Schur
Distributed Memory / Generalized EVP, 76
starneig_GEP_DM_Schur_expert
Distributed Memory / Generalized EVP, 80
starneig_GEP_DM_Select
Distributed Memory / Generalized EVP, 79
starneig_GEP_SM_Eigenvectors
Shared Memory / Generalized EVP, 50
starneig_GEP_SM_Eigenvectors_expert
Shared Memory / Generalized EVP, 54
starneig_GEP_SM_HessenbergTriangular
Shared Memory / Generalized EVP, 45
starneig_GEP_SM_Reduce
Shared Memory / Generalized EVP, 48
starneig_GEP_SM_ReorderSchur
Shared Memory / Generalized EVP, 47
starneig_GEP_SM_ReorderSchur_expert
Shared Memory / Generalized EVP, 53
starneig_GEP_SM_Schur
Shared Memory / Generalized EVP, 46
starneig_GEP_SM_Schur_expert
Shared Memory / Generalized EVP, 52
starneig_GEP_SM_Select
Shared Memory / Generalized EVP, 51
starneig_SEP_DM_Eigenvectors
Distributed Memory / Standard EVP, 70
starneig_SEP_DM_Eigenvectors_expert
Distributed Memory / Standard EVP, 74
starneig_SEP_DM_Hessenberg
Distributed Memory / Standard EVP, 67
starneig_SEP_DM_Reduce
Distributed Memory / Standard EVP, 69
starneig_SEP_DM_ReorderSchur
Distributed Memory / Standard EVP, 69
starneig_SEP_DM_ReorderSchur_expert
Distributed Memory / Standard EVP, 73
starneig_SEP_DM_Schur
Distributed Memory / Standard EVP, 68
starneig_SEP_DM_Schur_expert
Distributed Memory / Standard EVP, 71
starneig_SEP_DM_Select
Distributed Memory / Standard EVP, 71
starneig_SEP_SM_Eigenvectors
Shared Memory / Standard EVP, 39
starneig_SEP_SM_Eigenvectors_expert
Shared Memory / Standard EVP, 43
starneig_SEP_SM_Hessenberg
Shared Memory / Standard EVP, 37
starneig_SEP_SM_Hessenberg_expert
Shared Memory / Standard EVP, 41
starneig_SEP_SM_Reduce
Shared Memory / Standard EVP, 38
starneig_SEP_SM_ReorderSchur
Shared Memory / Standard EVP, 38
starneig_SEP_SM_ReorderSchur_expert
Shared Memory / Standard EVP, 42
starneig_SEP_SM_Schur
Shared Memory / Standard EVP, 37
starneig_SEP_SM_Schur_expert
Shared Memory / Standard EVP, 42
starneig_SEP_SM_Select
Shared Memory / Standard EVP, 40
starneig_blacs_context_to_distr
ScaLAPACK compatibility / BLACS matrices, 96
starneig_blacs_descr, 95

starneig_blacs_descr_to_distr_matrix
ScaLAPACK compatibility / BLACS matrices, 99
starneig_blacs_exit
ScaLAPACK compatibility / BLACS helpers, 104
starneig_blacs_get
ScaLAPACK compatibility / BLACS helpers, 102
starneig_blacs_gridexit
ScaLAPACK compatibility / BLACS helpers, 103
starneig_blacs_gridinfo
ScaLAPACK compatibility / BLACS helpers, 102
starneig_blacs_gridinit
ScaLAPACK compatibility / BLACS helpers, 102
starneig_blacs_pcoord
ScaLAPACK compatibility / BLACS helpers, 103
starneig_blacs_pinfo
ScaLAPACK compatibility / BLACS helpers, 101
starneig_broadcast
Distributed Memory / Distributed matrices, 66
starneig_create_blacs_matrix
ScaLAPACK compatibility / BLACS matrices, 97
starneig_datatype_t
Distributed Memory / Distributed matrices, 58
starneig_descinit
ScaLAPACK compatibility / BLACS helpers, 104
starneig_destroy_blacs_matrix
ScaLAPACK compatibility / BLACS matrices, 98
starneig_distr_block, 57
starneig_distr_destroy
Distributed Memory / Distributed matrices, 60
starneig_distr_duplicate
Distributed Memory / Distributed matrices, 60
starneig_distr_init
Distributed Memory / Distributed matrices, 58
starneig_distr_init_func
Distributed Memory / Distributed matrices, 59
starneig_distr_init_mesh
Distributed Memory / Distributed matrices, 58
starneig_distr_is_blacs_compatible
ScaLAPACK compatibility / BLACS matrices, 97
starneig_distr_is_compatible_with
ScaLAPACK compatibility / BLACS matrices, 97
starneig_distr_matrix_copy
Distributed Memory / Distributed matrices, 62
starneig_distr_matrix_copy_region
Distributed Memory / Distributed matrices, 63
starneig_distr_matrix_create
Distributed Memory / Distributed matrices, 60
starneig_distr_matrix_create_local
Distributed Memory / Distributed matrices, 61
starneig_distr_matrix_destroy
Distributed Memory / Distributed matrices, 62
starneig_distr_matrix_get_blocks
Distributed Memory / Distributed matrices, 63
starneig_distr_matrix_get_col_blksz
Distributed Memory / Distributed matrices, 66
starneig_distr_matrix_get_cols
Distributed Memory / Distributed matrices, 65
starneig_distr_matrix_get_datatype
Distributed Memory / Distributed matrices, 64
starneig_distr_matrix_get_distr
Distributed Memory / Distributed matrices, 64
starneig_distr_matrix_get_elemsize
Distributed Memory / Distributed matrices, 64
starneig_distr_matrix_get_row_blksz
Distributed Memory / Distributed matrices, 65
starneig_distr_matrix_get_rows
Distributed Memory / Distributed matrices, 65
starneig_distr_matrix_is_blacs_compatible
ScaLAPACK compatibility / BLACS matrices, 99
starneig_distr_matrix_is_compatible_with
ScaLAPACK compatibility / BLACS matrices, 100
starneig_distr_matrix_to_blacs_descr
ScaLAPACK compatibility / BLACS matrices, 98
starneig_distr_order_t
Distributed Memory / Distributed matrices, 57
starneig_distr_to_blacs_context
ScaLAPACK compatibility / BLACS matrices, 96
starneig_eigenvectors_conf, 89
starneig_eigenvectors_init_conf
Expert configuration structures, 94
starneig_hessenberg_conf, 85
starneig_hessenberg_init_conf
Expert configuration structures, 93
starneig_mpi_get_comm
Intra-node execution environment, 35
starneig_mpi_set_comm
Intra-node execution environment, 35
starneig_node_disable_pinning
Intra-node execution environment, 35
starneig_node_enable_pinning
Intra-node execution environment, 34
starneig_node_get_cores
Intra-node execution environment, 33
starneig_node_get_gpus
Intra-node execution environment, 34
starneig_node_init
Intra-node execution environment, 33
starneig_node_initialized
Intra-node execution environment, 33
starneig_node_set_cores
Intra-node execution environment, 34
starneig_node_set_gpus
Intra-node execution environment, 34
starneig_numroc
ScaLAPACK compatibility / BLACS helpers, 104
starneig_reorder_blueprint_t
Expert configuration structures, 92
starneig_reorder_conf, 87
starneig_reorder_init_conf
Expert configuration structures, 93
starneig_reorder_plan_t
Expert configuration structures, 89
starneig_schur_conf, 85
starneig_schur_init_conf
Expert configuration structures, 93