ELSI Interface Users' Guide
v2.10.0

The ELSI Team
https://elsi-interchange.org

October 22, 2023

Contents

1 Introduction

1.1 ELSI: ELectronic Structure Infrastructure

Computer simulations based on electronic structure theory, particularly Kohn-Sham density-functional theory (KS-DFT), are facilitating scientific discoveries across a broad range of disciplines such as chemistry, physics, and materials science. Despite its remarkable success, routine application of KS-DFT to systems consisting of thousands of atoms is still difficult. The major computational bottleneck is an eigenvalue problem

$$\boldsymbol{HC} = \boldsymbol{SC}\boldsymbol{\Sigma}, \tag{1.1}$$

where $\boldsymbol{H}$ and $\boldsymbol{S}$ are the so-called Hamiltonian and overlap matrices, $\boldsymbol{C}$ and $\boldsymbol{\Sigma}$ are the eigenvectors and eigenvalues of this eigensystem. The direct solution of Eq. 1.1 scales cubically with respect to the problem size. To overcome this bottleneck, researchers and software developers are actively improving the efficiency of eigensolvers and developing alternative algorithms that circumvent the explicit solution of the eigenproblem. The open-source ELSI library features a unified software interface that connects electronic structure codes to various high-performance solver libraries ranging from conventional cubic scaling eigensolvers to linear scaling density matrix solvers [1]. To date, it is adopted by four electronic structure packages (DFTB+ [2], DGDFT [3], FHI-aims [4], and SIESTA [5]).

1.2 Solver Libraries Supported by ELSI

Distributed-memory solvers supported in the current version of ELSI are: ELPA [6, 7, 8, 9], libOMM [10], PEXSI [11, 12, 13, 14, 15], EigenExa [16, 17], SLEPc-SIPc [18, 19, 20, 21], NTPoly [22], BSEPACK [23] and ChASE [24, 25]. Shared-memory solvers supported in the current version of ELSI are: LAPACK [26], MAGMA [27, 28] and ChASE [24, 25].

What follows is a brief summary of the solvers supported in ELSI. For technical descriptions of the solvers, the reader is referred to the original publications of the solvers, e.g., those in the reference list of this document.

ELPA: The massively parallel dense eigensolver ELPA facilitates the solution of symmetric or Hermitian eigenproblems on high-performance computers. It features an efficient two-stage tridiagonalization algorithm which is better suited for parallel computing than the conventional one-stage algorithm.

libOMM: The orbital minimization method (OMM) bypasses the explicit solution of the Kohn-Sham eigenproblem by efficient iterative algorithms which directly minimize an unconstrained energy functional using a set of auxiliary Wannier functions. The Wannier functions are defined on the occupied subspace of the system, reducing the size of the problem. The density matrix is then obtained directly, without calculating the Kohn-Sham orbitals.

PEXSI: PEXSI is a Fermi operator expansion (FOE) based method which expands the density matrix in terms of a linear combination of a small number of rational functions (pole expansion). Evaluation of these rational functions exploits the sparsity of the Hamiltonian and overlap matrices using selected inversion to enable scaling to 100,000+ of MPI tasks for calculation of the electron density, energy, and forces in electronic structure calculations.

EigenExa: The EigenExa library consists of two massively parallel implementations of direct, dense eigensolver. Its eigen_sx method features an efficient transformation from full to pentadiagonal matrix. Eigenvalues and eigenvectors of the pentadiagonal matrix are directly solved with a divide-and-conquer algorithm. This method is particularly efficient when a large part of the eigenspectrum is of interest.

SLEPc-SIPs: SLEPc-SIPs is a parallel sparse eigensolver for real symmetric generalized eigenvalue problems. It implements a distributed spectrum slicing method and it is currently available through the SLEPc library built on top of the

PETSc framework.

NTPoly: NTPoly is a massively parallel library for computing the functions of sparse, symmetric matrices based on polynomial expansions. For sufficiently sparse matrices, most of the matrix functions can be computed in linear time. Distributed memory parallelization is based on a communication avoiding sparse matrix multiplication algorithm. Various density matrix purification algorithms which compute the density matrix as a function of the Hamiltonian matrix are implemented in NTPoly.

BSEPACK: BSEPACK is a parallel ScaLAPACK-style library for solving the Bethe-Salpeter eigenvalue problem on distributed-memory high-performance computers.

LAPACK: LAPACK provides routines for solving linear systems, least squares problems, eigenvalue problems, and singular value problems. In order to promotes high efficiency on present-day computers, LAPACK routines are written to exploit BLAS, particularly level-3 BLAS, as much as possible. In ELSI, the tridiagonalization and the corresponding back-transformation routines in LAPACK are combined with the efficient divide-and-conquer tridiagonal solver in ELPA.

MAGMA: The MAGMA project aims to develop a dense linear algebra framework for heterogeneous architectures consisting of manycore and GPU systems. MAGMA incorporates the latest advances in synchronization-avoiding and communication-avoiding algorithms, and uses a hybridization methodology where algorithms are split into tasks of varying granularity and their execution scheduled over the available hardware components.

ChASE: ChASE is a modern and scalable library based on subspace iteration with polynomial acceleration to solve dense Hermitian (Symmetric) algebraic eigenvalue problems. Novel to ChASE is the computation of the spectral estimates that enter in the filter and an optimization of the polynomial degree that further reduces the necessary floating-point operations. When solving sequences of Hermitian eigenproblems for a portion of their extremal spectrum, ChASE greatly benefits from the sequence's spectral properties and outperforms direct solvers in many scenarios. The library ships with both shared-memory and distributed-memory implementations with the acceleration of GPUs, if available.

1.3 Citing ELSI

Key concepts of ELSI and the first version of its implementation are described in the following paper [1]:

V. W-z. Yu, F. Corsetti, A. García, W. P. Huhn, M. Jacquelin, W. Jia, B. Lange, L. Lin, J. Lu, W. Mi, A. Seifitokaldani, Á. Vázquez-Mayagoitia, C. Yang, H. Yang, and V. Blum, ELSI: A Unified Software Interface for Kohn-Sham Electronic Structure Solvers, Computer Physics Communications, 222, 267-285 (2018).

In addition, an incomplete list of publications describing the solvers supported in ELSI may be found in the bibliography of this document. Please consider citing these articles when publishing results obtained with ELSI.

1.4 Acknowledgments

ELSI is a National Science Foundation Software Infrastructure for Sustained Innovation - Scientific Software Integration (SI2-SSI) supported software infrastructure project. The ELSI Interface software and this User's Guide are based upon work supported by the National Science Foundation under Grant Number 1450280. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the National Science Foundation.

2 Installation of ELSI

2.1 Prerequisites

The ELSI package contains the ELSI interface software as well as redistributed source code for the solver libraries ELPA (versions 2020.05.001, 2021.11.001 and 2023.05.001), libOMM (version 1.0.0), PEXSI (version 1.2.0), NTPoly (version 2.7.0), and ChASE (version 1.4.0). The installation of ELSI makes use of the CMake software. Minimum requirements include:

```
CMake [minimum version 3.0; newer version recommended] Fortrancompiler [Fortran 2003 compliant] C compiler [C
MPI [MPI-3]
```

The PEXSi, EigenExa, SLEPc-SIPs, BSEPACK, MAGMA, and ChASE solvers are not enabled by default. Enabling the PEXSI and ChASE solvers requires:

```
C++ compiler  [C++ 11 compliant]
```

Enabling the EigenExa solver requires:

```
EigenExa  [version 2.11 or newer]
```

Enabling the SLEPc-SIPs solver requires:

```
SLEPc  [version 3.9 or newer]
PETSc  [version 3.9 or newer, with MUMPS and ParMETIS enabled]
```

Enabling the MAGMA solver requires:

```
MAGMA  [version 2.5 or newer]
CUDA  [version 10.0 or newer]
```

Enabling the CUDA-based GPU acceleration in the ELPA and ChASE solvers requires:

```
CUDA  [version 10.0 or newer recommended]
```

Optionally, if "`ENABLE_CHASE_NCCL`" is set to be "`ON`',' ChASE solver with GPU acceleration requires:

```
NCCL  [version 2.0 or newer recommended]
```

Linear algebra libraries should be provided for ELSI to link against:

```
BLAS, LAPACK, BLACS, ScaLAPACK
```

2.2 Quick Start

We recommend preparing configuration settings in a toolchain file that can be read by CMake. Edit one of the templates provided in the "`toolchains`" directory of the ELSI package. As an example, a minimal Intel toolchain looks like

```
# Modify contents in red if necessary
set(CMAKE_Fortran_COMPILER "mpiifort" CACHE STRING "MPI Fortran compiler")
set(CMAKE_Fortran_FLAGS "-O3 -ip -fp-model precise" CACHE STRING "Fortran flags")
set(CMAKE_C_COMPILER "mpiicc" CACHE STRING "MPI C compiler")
set(CMAKE_C_FLAGS "-O3 -ip -fp-model precise -std=c99" CACHE STRING "C flags")
set(LIB_PATHS "$ENVMKLROOT/lib/intel64" CACHE STRING "External library paths")
set(LIBS "mkl_scalapack_lp64 mkl_blacs_intelmpi_lp64 mkl_intel_lp64 mkl_sequential mkl_core"
    CACHE STRING "External libraries")
```

This will build ELSI with the redistributed ELPA, libOMM, and NTPoly solvers. A complete list of configure options may be found in Sec. 2.3.4.

Once a toolchain file is ready, follow the steps below:

```
$ cd elsi-interface
$ ls

  CMakeLists.txt  external/  src/  test/  ...

$ mkdir build
$ cd build
$ cmake -DCMAKE_TOOLCHAIN_FILE=YOUR_TOOLCHAIN_FILE ..

  ...
  ...
  -- Generating done
  -- Build files have been written to: /current/dir

$ make [-j np]
$ [make install]
```

"`YOUR_TOOLCHAIN_FILE`" should be the user's toolchain file. Commands in square brackets are optional.

If the compilation succeeds, the next step would be reading the code examples in the "`test`" directory of the ELSI package, which showcase the use of ELSI in C and Fortran programs.

2.3 Configuration

2.3.1 Compilers

CMake automatically detects compilers. The choices made by CMake often work, but they do not necessarily lead to the optimal performance. In some cases, the compilers picked up by CMake may not be the ones desired by the user. To build ELSI, it is mandatory that the user explicitly sets the identification of the compilers in `CMAKE_Fortran_COMPILER`, `CMAKE_C_COMPILER`, and `CMAKE_CXX_COMPILER`. Please note that the C++ compiler is not needed when building ELSI without PEXSI or ChASE.

In addition, it is highly recommended to specify the compiler flags in `CMAKE_Fortran_FLAGS`, `CMAKE_C_FLAGS`, and `CMAKE_CXX_FLAGS`.

2.3.2 Solvers

The ELPA, libOMM, PEXSI, NTPoly, BSEPACK and ChASE solver libraries, as well as the SuperLU_DIST and PT-SCOTCH libraries (both required by PEXSI), are redistributed with the current ELSI package.

The redistributed version of ELPA comes with a few "kernels" written to take advantage of architecture-specific instruction sets, which may be chosen by the `ELPA2_KERNEL` keyword. Available options are `AVX`, `AVX2`, and `AVX512`, for architectures supporting Intel AVX, AVX2, and AVX512 instruction sets, respectively. In ELPA, these kernels are employed to accelerate the calculation of eigenvectors, which is often a bottleneck when calculating a large portion of the eigenspectrum. In addition, ELPA supports GPU acceleration on NVIDIA devices via the CUDA programming model and the cuBLAS library. This feature may be enabled by the keyword `USE_GPU_CUDA`, which requires the CUDA runtime and cuBLAS libraries. Different versions of internally built ELPA can be enabled by the cmake options `USE_ELPA_2021` and `USE_ELPA_2023`.

The PEXSI, EigenExa, SLEPc-SIPs, BSEPACK, MAGMA, and ChASE solvers are not enabled by default. They may be activated by the keywords `ENABLE_PEXSI`, `ENABLE_EIGENEXA`, `ENABLE_SIPS`, `ENABLE_BSEPACK`, `ENABLE_MAGMA` and `ENABLE_CHASE`, respectively. PEXSI 1.2.0, EigenExa 2.11, SLEPc 3.9, 3.10, 3.11, 3.12, 3.13, BSEPACK 0.1, and MAGMA 2.5 have been tested with this version of ELSI. Older/newer versions may or may not be compatible. The PETSc library, required by SLEPc, must be compiled with MPI support, and (at least) with MUMPS and ParMETIS enabled.

Experienced users are encouraged to link ELSI against externally installed, better optimized solver libraries. The keywords `USE_EXTERNAL_ELPA`, `USE_EXTERNAL_OMM`, `USE_EXTERNAL_PEXSI`, `USE_EXTERNAL_NTPOLY`, `USE_EXTERNAL_BSEPACK`, and `USE_EXTERNAL_CHASE` control the usage of externally compiled ELPA, libOMM, PEXSI, NTPoly, BSEPACK, and ChASE, respectively.

All external libraries and include paths should be set via `INC_PATHS`, `LIB_PATHS`, and `LIBS`, each of which is a list of items separated with " " (space) or ";" (semicolon). If an external library depends on additional libraries, `LIBS` should include all the relevant dependencies. For instance, `LIBS` should include the MAGMA library and CUDA libraries when enabling MAGMA support.

2.3.3 Tests

Building ELSI test programs may be enabled by `ENABLE_TESTS`. Then, the compilation of ELSI may be verified by "`make test`" or "`ctest`". Note that the tests may not run if launching MPI jobs is prohibited on the user's working platform.

2.3.4 List of All Configure Options

The options accepted by the ELSI CMake build system are listed here in alphabetical order. Some additional explanations are made below the table.

Option	Type	Default	Explanation
`ADD_UNDERSCORE`	boolean	ON	Suffix C functions with an underscore
`BUILD_SHARED_LIBS`	boolean	OFF	Build ELSI as a shared library
`CMAKE_C_COMPILER`	string	none	MPI C compiler
`CMAKE_C_FLAGS`	string	none	C flags
`CMAKE_CUDA_COMPILER`	string	none	CUDA compiler (nvcc)
`CMAKE_CUDA_FLAGS`	string	none	CUDA flags
`CMAKE_CXX_COMPILER`	string	none	MPI C++ compiler
`CMAKE_CXX_FLAGS`	string	none	C++ flags
`CMAKE_Fortran_COMPILER`	string	none	MPI Fortran compiler
`CMAKE_Fortran_FLAGS`	string	none	Fortran flags
`CMAKE_INSTALL_PREFIX`	path	/usr/local	Path to install ELSI
`USE_ELPA_2021`	boolean	OFF	Enable ELPA 2021
`USE_ELPA_2023`	boolean	OFF	Enable ELPA 2023
`ELPA2_KERNEL`	string	none	ELPA2 kernel
`ENABLE_BSEPACK`	boolean	OFF	Enable BSEPACK support

ENABLE_C_TESTS	boolean	OFF	Build C test programs
ENABLE_EIGENEXA	boolean	OFF	Enable EigenExa support
ENABLE_MAGMA	boolean	OFF	Enable MAGMA support
ENABLE_PEXSI	boolean	OFF	Enable PEXSI support
ENABLE_SIPS	boolean	OFF	Enable SLEPc-SIPs support
ENABLE_CHASE	boolean	OFF	Enable ChASE support
ENABLE_CHASE_NCCL	boolean	OFF	Enable ChASE support with NCCL
ENABLE_TESTS	boolean	OFF	Build Fortran test programs
INC_PATHS	string	none	Include directories of external libraries
LIB_PATHS	string	none	Directories containing external libraries
LIBS	string	none	External libraries
MPIEXEC_NP	string	mpirun -n 4	Command to run tests in parallel with MPI
MPIEXEC_1P	string	mpirun -n 1	Command to run tests in serial with MPI
SCOTCH_LAST_RESORT	string	none	Command to invoke PT-SCOTCH header generator
USE_EXTERNAL_BSEPACK	boolean	OFF	Use external BSEPACK
USE_EXTERNAL_ELPA	boolean	OFF	Use external ELPA
USE_EXTERNAL_NTPOLY	boolean	OFF	Use external NTPoly
USE_EXTERNAL_OMM	boolean	OFF	Use external libOMM and MatrixSwitch
USE_EXTERNAL_PEXSI	boolean	OFF	Use external PEXSI (if PEXSI enabled)
USE_GPU_CUDA	boolean	OFF	Use CUDA-based GPU acceleration in ELPA
USE_MPI_MODULE	boolean	OFF	Use MPI module instead of "mpif.h" in Fortran code

Remarks

(1) ADD_UNDERSCORE: In the redistributed PEXSI and SuperLU_DIST code, there are calls to functions from the linear algebra libraries, e.g. "dgemm". If ADD_UNDERSCORE is "ON", the code will call "dgemm_" instead of "dgemm". Turn this keyword off if routines are not suffixed with "_" in the linear algebra libraries.

(2) CMAKE_INSTALL_PREFIX: ELSI may be installed to the location specified in CMAKE_INSTALL_PREFIX by "make install".

(3) ELPA2_KERNEL: There are a number of computational kernels available with the ELPA solver. Choose from "AVX" (Intel AVX), "AVX2" (Intel AVX2), and "AVX512" (Intel AVX512). See Sec. 2.3.2 for more information.

(4) SCOTCH_LAST_RESORT: The compilation of PT-SCOTCH is a multi-step process. First, two auxiliary executables are created. Then, some header files are generated on-the-fly by the two executables. Finally, the main source files are compiled with the generated header files included. The header generation step may fail on platforms where directly running an executable is prohibited, e.g. login nodes of a supercomputer. Often this can be circumvented by requesting an interactive session to a compute node and compiling the code there, or by submitting the compilation as a job to the queuing system. However, this may still fail on platforms where an executable compiled with MPI must be launched by an MPI job launcher (aprun, mpirun, srun, etc). If the standard compilation of PT-SCOTCH fails due to this reason, the user may set SCOTCH_LAST_RESORT to the command that starts an MPI job with one MPI task, e.g. "mpirun -n 1". This command is then used to launch the auxiliary executables to generate necessary header files for PT-SCOTCH.

(5) External libraries: ELSI redistributes source code of ELPA, libOMM, NTPoly, PEXSI, SuperLU_DIST, PT-SCOTCH and ChASE libraries, which are built by default together with the ELSI interface. Experienced users are encouraged to link the ELSI interface against external, better optimized solver libraries. See Sec. 2.3.2 for more information.

2.4 Importing ELSI into Third-Party Code Projects

2.4.1 Linking against ELSI: CMake

A CMake configuration file called elsiConfig.cmake should be generated after ELSI is successfully installed. This file contains all the information about how the ELSI library and its dependencies should be included in an external project. For a project using CMake, only two lines are required to find and link to ELSI:

```
find_package(elsi REQUIRED)
target_link_libraries(my_project PRIVATE elsi::elsi)
```

If a minimum version of ELSI is required, this information may be passed to "`find_package`" by, e.g.:

```
find_package(elsi 2.0 REQUIRED)
```

2.4.2 Linking against ELSI: Makefile

For a project using makefiles, an example set of compiler flags to link against ELSI would be:

```
ELSI_INCLUDE = -I/PATH/TO/BUILD/ELSI/include
ELSI_LIB     = -L/PATH/TO/BUILD/ELSI/lib -lelsi \
               -lfortjson -lOMM -lMatrixSwitch -lelpa \
               -lNTPoly -lpexsi -lsuperlu_dist \
               -lptscotchparmetis -lptscotch -lptscotcherr \
               -lscotchmetis -lscotch -lscotcherr
```

Enabling/disabling PEXSI, EigenExa, SLEPc-SIPs, BSEPACK, MAGMA, ChASE or linking ELSI against externally installed solver libraries requires the user modify these flags accordingly.

2.4.3 Using ELSI

ELSI may be used in an electronic structure code by importing the appropriate header file. For codes written in Fortran, this is done by using the ELSI module

```
use ELSI
```

For codes written in C, the ELSI wrapper may be imported by including the header file

```
#include <elsi.h>
```

3 The ELSI API

3.1 Overview of the ELSI API

In this chapter, we present the public-facing API for the ELSI Interface. We anticipate that fine details of this interface may change slightly in the future, but the fundamental structure of the interface layer is expected to remain consistent. While this chapter serves as a reference to the ELSI subroutines, the user is encouraged to explore the demonstration pseudo-codes of ELSI in Sec. 3.9.

To allow multiple instances of ELSI to co-exist within a single calling code, we define an `elsi_handle` data type to encapsulate the state of an ELSI instance, i.e., all runtime parameters associated with the ELSI instance. An `elsi_handle` instance is initialized with the `elsi_init` subroutine and is subsequently passed to all other ELSI subroutine calls.

ELSI provides a C interface in addition to the native Fortran interface. The vast majority of this chapter, while written from a Fortran standpoint, applies equally to both interfaces. Information specifically about the C wrapper for ELSI may be found in Sec. 3.8.

In the source code of ELSI, there may exist subroutines that are not documented as public API here. Usage of those undocumented subroutines is not recommended, as they are usually experimental and subject to modification or removal without notice.

3.2 Setting Up ELSI

3.2.1 Initializing ELSI

The ELSI interface must be initialized via the `elsi_init` subroutine before any other ELSI subroutine may be called.

`elsi_init`(handle, solver, parallel_mode, matrix_format, n_basis, n_electron, n_state)

Argument	Data Type	in/out	Explanation
`handle`	elsi_handle	out	ELSI handle.
`solver`	integer	in	0: `AUTO`. 1: `ELPA`. 2: `libOMM`. 3: `PEXSI`. 4: `EigenExa`. 5: `SLEPc-SIPs`. 6: `NTPoly`. 7: `MAGMA`. 8: `BSEPACK`. 9: `ChASE`. See remark 1.
`parallel_mode`	integer	in	0: `SINGLE_PROC`. 1: `MULTI_PROC`. See remark 4.
`matrix_format`	integer	in	0: `BLACS_DENSE`. 1: `PEXSI_CSC`. 2: `SIESTA_CSC`. 3: `GENERIC_COO`. See remark 2.
`n_basis`	integer	in	Number of basis functions, i.e. global size of Hamiltonian.
`n_electron`	real double	in	Number of electrons.
`n_state`	integer	in	Number of states. See remark 3.

Remarks

(1) `solver`: The "`AUTO`" option attempts to automate the solver selection procedure based on benchmarks performed and experiences gained in the ELSI project. User-supplied information may assist in finding the optimal solver. In particular, see `elsi_set_dimensionality` and `elsi_set_energy_gap` in Sec. 3.5. Simply put, the solver selection favors ELPA for small-and-medium-sized problems, PEXSI for large, sparse, low-dimensional problems, and NTPoly for extra-large, sparse systems with a decent energy gap.

When the ELPA solver is chosen for the "SINGLE_PROC" parallel mode, the tridiagonalization and back-transformation routines in LAPACK and the divide-and-conquer tridiagonal solver routine in ELPA are used.

The table below summarizes the supported problem types for each solver.

Solver	Parallel Mode	Matrix Format	Data Type	Problem Type
ELPA	SINGLE_PROC	BLACS_DENSE	Real/complex	KS-EV
ELPA	MULTI_PROC	All	Real/complex	KS-EV/KS-DM
libOMM	MULTI_PROC	All	Real/complex	KS-DM
PEXSI	MULTI_PROC	All	Real/complex	KS-DM
EigenExa	MULTI_PROC	All	Real/complex	KS-EV/KS-DM
SLEPc-SIPs	MULTI_PROC	All	Real	KS-EV/KS-DM
NTPoly	MULTI_PROC	All	Real/complex	KS-DM
MAGMA	SINGLE_PROC	BLACS_DENSE	Real/complex	KS-EV
BSEPACK	MULTI_PROC	BLACS_DENSE	Real/complex	BSE-EV
ChASE	SINGLE_PROC	BLACS_DENSE	Real/complex	KS-EV
ChASE	MULTI_PROC	BLACS_DENSE	Real/complex	KS-EV

(2) `matrix_format`: "BLACS_DENSE" refers to a dense matrix format in a 2-dimensional block-cyclic distribution, i.e. the BLACS standard. "PEXSI_CSC" refers to a compressed sparse column (CSC) matrix format in a 1-dimensional block distribution. "SIESTA_CSC" refers to a compressed sparse column (CSC) matrix format in a 1-dimensional block-cyclic distribution. As the Hamiltonian, overlap, and density matrices are symmetric (Hermitian), compressed sparse row (CSR) matrix format is effectively supported. "GENERIC_COO" refers to a coordinate (COO) sparse matrix format in an arbitrary distribution. Please refer to Sec. 3.2.3 for specifications of these matrix formats.

(3) `n_state`: If ELPA, EigenExa, SLEPc-SIPs, MAGMA or ChASE is the chosen solver, this parameter specifies the number of eigenstates to solve. EigenExa internally computes all the eigenstates unless `n_state` is 0. When `n_state` is larger than 0 and smaller than `n_basis`, ELSI simply discards the unwanted solutions. libOMM, PEXSI and NTPoly do not make use of this parameter.

(4) `parallel_mode`: The two parallelization modes, "SINGLE_PROC" and "MULTI_PROC", allow for two parallelization strategies commonly employed by electronic structure codes. See below.

4a) "SINGLE_PROC": Solves the KS eigenproblem following a LAPACK-like fashion. This option may only be selected when ELPA, MAGMA or ChASE is chosen as the solver. Every MPI task independently handles a group of k-points uniquely assigned to it. Example: 16 k-points, 4 MPI tasks.

- MPI task 0 handles k-points 1, 2, 3, 4 sequentially;
- MPI task 1 handles k-points 5, 6, 7, 8 sequentially;
- MPI task 2 handles k-points 9, 10, 11, 12 sequentially;
- MPI task 3 handles k-points 13, 14, 15, 16 sequentially.

```
call elsi_init (eh, ..., parallel_mode=0, ...)
...
do i_kpt = 1, n_kpt_local
   call elsi_ev_{real|complex} (eh, ham_this_kpt, ovlp_this_kpt, eval_this_kpt, evec_this_kpt)
end do
```

4b) "MULTI_PROC": Solves the KS eigenproblem following a ScaLAPACK-like fashion. Groups of MPI tasks coordinate to handle the same k-point, uniquely assigned to that group. Example: 4 k-points, 16 MPI tasks.

- MPI tasks 0, 1, 2, 3 cooperatively handle k-point 1;
- MPI tasks 4, 5, 6, 7 cooperatively handle k-point 2;
- MPI tasks 8, 9, 10, 11 cooperatively handle k-point 3;
- MPI tasks 12, 13, 14, 15 cooperatively handle k-point 4.

```
call elsi_init (eh, ..., parallel_mode=1, ...)
call elsi_set_mpi (eh, my_mpi_comm)
call elsi_set_kpoint (eh, n_kpt, my_kpt, my_weight)
call elsi_set_mpi_global (eh, mpi_comm_global)
...
call elsi_{ev|dm}_{real|complex} (eh, my_ham, my_ovlp, ...)
```

Please note that when there is more than one k-point, a global MPI communicator must be provided for inter-k-point communications. See Sec. 3.2.4 for `elsi_set_kpoint`, `elsi_set_spin`, and `elsi_set_mpi_global`, which are used to set up a calculation with two spin channels and/or multiple k-points.

3.2.2 Setting Up MPI

The MPI communicator used by ELSI is passed into ELSI by the calling code via the `elsi_set_mpi` subroutine. When there is more than one k-point and/or spin channel, this communicator is used only for solving one problem corresponding to one k-point and one spin channel. See Sec. 3.2.4 for details.

`elsi_set_mpi`(handle, comm)

Argument	Data Type	in/out	Explanation
`handle`	elsi_handle	inout	ELSI handle.
`comm`	integer	in	MPI communicator.

3.2.3 Setting Up Matrix Formats

Four matrix formats are supported by ELSI, namely the 2D block-cyclic distributed dense matrix format ("`BLACS_DENSE`"), the 1D block distributed compressed sparse column format ("`PEXSI_CSC`"), the 1D block-cyclic distributed compressed sparse column format ("`SIESTA_CSC`"), and the arbitrarily distributed coordinate sparse format ("`GENERIC_COO`").

When using the "`BLACS_DENSE`" format, BLACS parameters are passed into ELSI via the `elsi_set_blacs` subroutine. The matrix format used internally in the ELSI interface and the ELPA solver requires the block sizes of the 2-dimensional block-cyclic distribution are the same in the row and column directions. It is necessary to call this subroutine before calling any solver interface that makes use of the "`BLACS_DENSE`" format.

`elsi_set_blacs`(handle, blacs_ctxt, block_size)

Argument	Data Type	in/out	Explanation
`handle`	elsi_handle	inout	ELSI handle.
`blacs_ctxt`	integer	in	BLACS context.
`block_size`	integer	in	Block size of the 2D block-cyclic distribution, specifying both row and column directions.

When using the "`PEXSI_CSC`" or "`SIESTA_CSC`" format, the sparsity pattern should be passed into ELSI via the `elsi_set_csc` subroutine. It is necessary to call this subroutine before calling any solver interface that makes use of the CSC sparse matrix formats.

`elsi_set_csc`(handle, global_nnz, local_nnz, local_col, row_idx, col_ptr)

Argument	Data Type	in/out	Explanation
`handle`	elsi_handle	inout	ELSI handle.
`global_nnz`	integer	in	Global number of non-zeros.
`local_nnz`	integer	in	Local number of non-zeros.
`local_col`	integer	in	Local number of matrix columns.
`row_idx`	1D integer array	in	Local row index array. Dimension: local_nnz.
`col_ptr`	1D integer array	in	Local column pointer array. Dimension: local_col+1.

The block size of the "`PEXSI_CSC`" format cannot be set by the user. This is because the PEXSI solver requires that the block size must be floor(N_{basis}/N_{procs}), where floor(x) is the greatest integer less than or equal to x, N_{basis} and N_{procs}

are the number of basis functions and the number of MPI tasks, respectively. The block size of the "SIESTA_CSC" format must be explicitly set by calling elsi_set_csc_blk.

elsi_set_csc_blk(handle, block_size)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
global_nnz	integer	in	Block size of the 1D block-cyclic distribution.

In most cases, input and output matrices should be distributed across all MPI tasks. The only exception is when using the PEXSI solver, the sparse density matrix interface elsi_dm_{real|complex}_sparse, and the "PEXSI_CSC" matrix format. In this case, an additional parameter, pexsi_np_per_pole, must be set by the user. Input and output matrices should be 1D-block-distributed among the first pexsi_np_per_pole MPI tasks (not all the MPI tasks). Please see Sec. 3.5.4 for more information.

When using the "GENERIC_COO" format, the sparsity pattern should be passed into ELSI via the elsi_set_coo subroutine. It is necessary to call this subroutine before calling any solver interface that makes use of the COO sparse matrix format. The distribution of matrix elements in the "GENERIC_COO" format is arbitrary. Both sorted and unsorted inputs are supported.

elsi_set_coo(handle, global_nnz, local_nnz, row_idx, col_idx)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
global_nnz	integer	in	Global number of non-zeros.
local_nnz	integer	in	Local number of non-zeros.
row_idx	1D integer array	in	Local row index array. Dimension: local_nnz.
col_idx	1D integer array	in	Local column index array. Dimension: local_nnz.

3.2.4 Setting Up Multiple *k*-points and/or Spin Channels

When there is more than one k-point and/or spin channel in the physical system being simulated, the ELSI interface can be set up to support parallel calculation of the k-points and/or spin channels. The base case is a system isolated in space, e.g. free atoms, molecules, clusters, without spin-polarization. In this case, there is one eigenproblem in each iteration of an SCF cycle. When a spin-polarized periodic system is considered, there is an index α denoting the spin channel, and an index k denoting points in reciprocal space. In total, there are $N_{kpt} \times N_{spin}$ eigenproblems that can be solved in an embarrassingly parallel fashion. In ELSI, these eigenproblems are considered as equivalent "unit tasks". The available computer processes are divided into $N_{kpt} \times N_{spin}$ groups, each of which is responsible for one unit task.

To set up the ELSI interface for a calculation with more than one k-point and/or more than one spin channel, the elsi_set_kpoint and/or elsi_set_spin subroutines are called to pass the required information into ELSI. The MPI communicator for each unit task is passed into ELSI by calling elsi_set_mpi. In addition, a global MPI communicator for all tasks is passed into ELSI by calling elsi_set_mpi_global. Note that the current ELSI interface only supports the case where the eigenproblems for all the k-points and spin channels are fully parallelized, i.e., there is no MPI task handling more than one k-point and/or more than one spin channel. In ELSI, the two spin channels are always coupled by a uniform chemical potential. The distribution of electrons among the two channels, and thus the net spin moment of the system, cannot be specified. Calculations with a fixed, user-specified spin moment can be performed by initializing two independent ELSI instances for the two spin channels.

In this version of ELSI, the SLEPc-SIPs eigensolver is not supported in spin-polarized and/or periodic calculations.

elsi_set_kpoint(handle, n_kpt, i_kpt, weight)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
n_kpt	integer	in	Total number of k-points.
i_kpt	integer	in	Index of the k-point handled by this MPI task.
weight	integer	in	Weight of the k-point handled by this MPI task.

elsi_set_spin(handle, n_spin, i_spin)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
n_spin	integer	in	Total number of spin channels.
i_spin	integer	in	Index of the spin channel handled by this MPI task.

elsi_set_mpi_global(handle, comm_global)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
comm_global	integer	in	Global MPI communicator used for communications among all k-points and spin channels.

3.2.5 Re-initializing ELSI

When a geometry update takes place in geometry optimization or molecular dynamics calculations, the overlap matrix changes due to the movement of localized basis functions. Calling **elsi_reinit** instructs ELSI to flush geometry-related variables and arrays that cannot be used in the new geometry step, e.g., the overlap matrix and its sparsity pattern. Other runtime parameters are kept within the ELSI instance and reused throughout multiple geometry steps. Note that the chemical potential determination in PEXSI must be restarted for every new geometry. See Sec. 3.5.4 for details.

elsi_reinit(handle)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.

3.2.6 Finalizing ELSI

When an ELSI instance is no longer needed, its associated handle should be cleaned up by calling elsi_finalize.

elsi_finalize(handle)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.

3.3 Solving Eigenvalues and Eigenvectors

elsi_ev_{real|complex}{_sparse} returns all the eigenvalues (except ChASE is selected which returns only a subset of required extremal eigenvalues) and a subset of eigenvectors of a generalized eigenproblem defined in Eq. 1.1. See elsi_set_unit_ovlp in Sec. 3.5.1 for standard eigenproblems. ELPA, EigenExa, SLEPc-SIPs, MAGMA or ChASE may be selected as the solver when using these subroutines.

elsi_ev_real(handle, ham, ovlp, eval, evec)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ham	2D real double array	inout	Hamiltonian matrix in "BLACS_DENSE" format. See remark 1.
ovlp	2D real double array	inout	Overlap matrix (or its Cholesky factorization) in "BLACS_DENSE" format. See remark 1.
eval	1D real double array	inout	Eigenvalues. See remark 2.
evec	2D real double array	out	Eigenvectors in "BLACS_DENSE" format. See remark 3.

elsi_ev_complex(handle, ham, ovlp, eval, evec)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ham	2D complex double array	inout	Hamiltonian matrix in "BLACS_DENSE" format. See remark 1.
ovlp	2D complex double array	inout	Overlap matrix (or its Cholesky factorization) in "BLACS_DENSE" format. See remark 1.
eval	1D real double array	inout	Eigenvalues. See remark 2.
evec	2D complex double array	out	Eigenvectors in "BLACS_DENSE" format. See remark 3.

elsi_ev_real_sparse(handle, ham, ovlp, eval, evec)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ham	1D real double array	inout	Hamiltonian matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
ovlp	1D real double array	inout	Overlap matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
eval	1D real double array	inout	Eigenvalues. See remark 2.
evec	2D real double array	out	Eigenvectors in "BLACS_DENSE" format. See remark 3.

elsi_ev_complex_sparse(handle, ham, ovlp, eval, evec)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ham	1D complex double array	inout	Hamiltonian matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
ovlp	1D complex double array	inout	Overlap matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
eval	1D real double array	inout	Eigenvalues. See remark 2.
evec	2D complex double array	out	Eigenvectors in "BLACS_DENSE" format. See remark 3.

Remarks

(1) When using elsi_ev_{real|complex} with ELPA or EigenExa, the Hamiltonian matrix is destroyed during the computation, the overlap matrix is used to store its Cholesky factorization, which can be reused until the overlap matrix changes. When using elsi_ev_{real|complex}_sparse, the Cholesky factorization is stored internally in the "BLACS_DENSE" format.

(2) The dimension of eval should always be n_basis, regardless of the choice of n_state specified in elsi_init.

(3) The number of eigenvectors to be computed by elsi_ev_{real|complex}{_sparse} is specified by n_state in elsi_init. However, the local evec array should always be initialized to correspond to a global array of size n_basis by n_basis, whose extra part is used as work space. When using elsi_ev_{real|complex}_sparse, the eigenvectors are returned in a dense format ("BLACS_DENSE"), as they are in general not sparse.

elsi_bse_{real|complex} solves the Bethe-Salpeter eigenproblem

$$\boldsymbol{H}_{\mathrm{BS}}\boldsymbol{C} = \boldsymbol{C}\boldsymbol{\Sigma}. \tag{3.1}$$

The BSE Hamiltonian $\boldsymbol{H}_{\mathrm{BS}}$ has the following structure

$$\boldsymbol{H}_{\mathrm{BS}} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ -\boldsymbol{B}^{\mathrm{H}} & \boldsymbol{A}^{\mathrm{T}} \end{bmatrix}, \tag{3.2}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are N by N matrices, $\boldsymbol{H}_{\mathrm{BS}}$ and $\boldsymbol{C}$ are therefore 2N by 2N matrices. $^{\mathrm{H}}$ and $^{\mathrm{T}}$ denote the conjugate transpose and the transpose of a matrix, respectively. $\boldsymbol{A} = \boldsymbol{A}^{\mathrm{H}}$ and $\boldsymbol{B} = \boldsymbol{B}^{\mathrm{T}}$. BSEPACK must be selected as the solver when using these subroutines.

elsi_bse_real(handle, A, B, eval, evec)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
A	2D real double array	inout	Matrix A in "BLACS_DENSE" format.
B	2D real double array	in	Matrix B in "BLACS_DENSE" format.
eval	1D real double array	out	Eigenvalues.
evec	2D real double array	out	Eigenvectors in "BLACS_DENSE" format. See remark 1.

elsi_bse_complex(handle, A, B, eval, evec)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
A	2D complex double array	inout	Matrix A in "BLACS_DENSE" format.
B	2D complex double array	in	Matrix B in "BLACS_DENSE" format.
eval	1D real double array	out	Eigenvalues.
evec	2D complex double array	out	Eigenvectors in "BLACS_DENSE" format. See remark 1.

Remarks

(1) The global dimension of evec should be 2N by 2N.

3.4 Computing Density Matrices

elsi_dm_{real|complex}{_sparse} returns the density matrix computed from the provided H and S matrices, as well as the band structure energy.

elsi_dm_real(handle, ham, ovlp, dm, e_bs)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ham	2D real double array	inout	Hamiltonian matrix in "BLACS_DENSE" format.
ovlp	2D real double array	inout	Overlap matrix (or Cholesky factorization) in "BLACS_DENSE" format. See remark 1.
dm	2D real double array	out	Density matrix in "BLACS_DENSE" format.
e_bs	real double	out	Band structure energy.

elsi_dm_complex(handle, ham, ovlp, dm, e_bs)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ham	2D complex double array	inout	Hamiltonian matrix in "BLACS_DENSE" format.
ovlp	2D complex double array	inout	Overlap matrix (or its Cholesky factorization) in "BLACS_DENSE" format. See remark 1.
dm	2D complex double array	out	Density matrix in "BLACS_DENSE" format.
e_bs	real double	out	Band structure energy.

elsi_dm_real_sparse(handle, ham, ovlp, dm, e_bs)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ham	1D real double array	inout	Hamiltonian matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
ovlp	1D real double array	inout	Overlap matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
dm	1D real double array	out	Density matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
e_bs	real double	out	Band structure energy.

elsi_dm_complex_sparse(handle, ham, ovlp, dm, e_bs)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ham	1D complex double array	inout	Hamiltonian matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
ovlp	1D complex double array	inout	Overlap matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
dm	1D complex double array	out	Density matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
e_bs	real double	out	Band structure energy.

Remarks

(1) When using elsi_dm_{real|complex} with ELPA, libOMM, or EigenExa, the Hamiltonian matrix is destroyed during the computation. The overlap matrix is used to store its Cholesky factorization, which can be reused until the overlap matrix changes.

3.5 Customizing ELSI

ELSI provides reasonable default values for parameters used in the ELSI interface and the supported solvers. However, no set of default parameters can adequately cover all use cases. Parameters that can be adjusted are described in the following subsections.

3.5.1 Customizing the ELSI Interface

In all the subroutines listed below, the first argument (input and output) is an elsi_handle. The second argument (input) of each subroutine is the name of parameter to set. Note that logical variables are not used in ELSI API. Integers are used to represent logical, with 0 being false and any positive integer being true.

elsi_set_output(handle, output_level)
elsi_set_output_unit(handle, output_unit)
elsi_set_output_log(handle, output_log)
elsi_set_save_ovlp(handle, save_ovlp)
elsi_set_unit_ovlp(handle, unit_ovlp)
elsi_set_zero_def(handle, zero_def)
elsi_set_sparsity_mask(handle, sparsity_mask)
elsi_set_illcond_check(handle, illcond_check)
elsi_set_illcond_tol(handle, illcond_tol)
elsi_set_spin_degeneracy(handle, spin_degeneracy)
elsi_set_energy_gap(handle, energy_gap)
elsi_set_spectrum_width(handle, spectrum_width)
elsi_set_dimensionality(handle, dimensionality)
elsi_set_mu_broaden_scheme(handle, mu_broaden_scheme)
elsi_set_mu_broaden_width(handle, mu_broaden_width)
elsi_set_mu_tol(handle, mu_tol)
elsi_set_mu_mp_order(handle, mu_mp_order)
elsi_set_n_frozen(handle, n_frozen)
elsi_set_frozen_idx(handle, frozen_idx)

Argument	Data Type	Default	Explanation
output_level	integer	0	0: No output. 1: Informative output from ELSI. 2: Informative output from ELSI and the solvers. 3: Informative and debugging output from ELSI and the solvers.
output_unit	integer	6	Unit used by ELSI to write out information.

`output_log`	integer	0	If not 0, a separate JSON log file will be written out.
`save_ovlp`	integer	0	If not 0, the overlap matrix will be saved for density matrix extrapolation.
`unit_ovlp`	integer	0	If not 0, the overlap matrix is treated as an identity (unit) matrix. See remark 1.
`zero_def`	real double	10^{-15}	When converting a matrix from dense to sparse format, values below this threshold are discarded.
`sparsity_mask`	integer	2	Which sparsity pattern to use when converting a matrix from dense to sparse format. 0: The union of the sparsity patterns of the Hamiltonian and overlap matrices. 1: The sparsity pattern of the Hamiltonian matrix. 2: The sparsity pattern of the overlap matrix.
`illcond_check`	integer	0	If not 0, the eigenvalues of the overlap matrix will be calculated to check if the overlap matrix is ill-conditioned. See remark 2.
`illcond_tol`	real double	10^{-5}	Eigenfunctions of the overlap matrix with eigenvalues smaller than this threshold will be removed to avoid ill-conditioning. See remark 2.
`spin_degeneracy`	real double	2.0/n_spin	Spin degeneracy that controls the maximum number of electrons on a state.
`energy_gap`	real double	0	Energy gap. See remark 3.
`spectrum_width`	real double	10^3	Width of the eigenspectrum. See remark 3.
`dimensionality`	integer	3	Dimensionality (1, 2, or 3) of the physical system being simulated. Only used for automatic solver selection.
`mu_broaden_scheme`	integer	0	Broadening scheme employed to compute the occupation numbers and the Fermi level. 0: Gaussian. 1: Fermi-Dirac. 2: Methfessel-Paxton. 4: Marzari-Vanderbilt. See remark 4.
`mu_broaden_width`	real double	0.01	Broadening width employed to compute the occupation numbers and the Fermi level. See remark 5.
`mu_tol`	real double	10^{-13}	Convergence tolerance (in terms of the absolute error in electron count) of the bisection algorithm employed to compute the occupation numbers and the Fermi level.
`mu_mp_order`	integer	0	Order of the Methfessel-Paxton broadening scheme. No effect if Methfessel-Paxton is not used.
`n_frozen`	integer	0	Number of states to be treated as "frozen" when using `elsi_ev_{real|complex}` with ELPA or LAPACK.
`frozen_idx`	1D integer array	-	List of indices of states to be frozen. Only relevant if `n_frozen` is set. By default, the first `n_frozen` states are frozen.

Remarks

(1) If the overlap matrix is an identity matrix, all settings related to the singularity (ill-conditioning) check are ignored. The `ovlp` passed into `elsi_{ev|dm}_{real|complex}{_sparse}` is not referenced.

(2) If the ill-conditioning check is not disabled, in the first iteration of each SCF cycle, all eigenvalues of the overlap matrix are computed. If there is any eigenvalue smaller than `illcond_tol`, the matrix is considered to be ill-conditioned.

(3) `spectrum_width` and `energy_gap` refer to the width and the gap of the eigenspectrum. Simply use the default values if there is no better estimate.

(4) `mu_broaden_scheme`, `mu_broaden_width`, and `mu_tol` are only referenced when using `elsi_dm_{real|complex}{_sparse}` and an eigensolver. They are ignored when using `elsi_ev_{real|complex}{_sparse}`, or `elsi_dm_{real|complex}{_sparse}` with a density matrix solver.

(5) In all supported broadening schemes, there is a term $(\epsilon - E_\mathrm{F})/W$ in the distribution function, where ϵ is the energy of an eigenstate, and E_F is the Fermi level. `broadening_width` should be W in the same unit of ϵ and E_F.

3.5.2 Customizing the ELPA Solver

elsi_set_elpa_solver(handle, elpa_solver)
elsi_set_elpa_n_single(handle, elpa_n_single)
elsi_set_elpa_gpu(handle, elpa_gpu)
elsi_set_elpa_autotune(handle, elpa_autotune)
elsi_set_elpa_gpu_kernels(handle, elpa_gpu_kernels)

Argument	Data Type	Default	Explanation
elpa_solver	integer	2	1: One-stage solver. 2: Two-stage solver (recommended).
elpa_n_single	integer	0	Number of SCF steps using single precision ELPA to solve standard eigenproblems. See remark 1.
elpa_gpu	integer	0	If not 0, enable GPU-acceleration in ELPA. See remark 2.
elpa_autotune	integer	1	If not 0, enable auto-tuning of runtime parameters in ELPA. Not compatible with illcond_check.

Remarks

(1) elpa_n_single: If single precision arithmetic is available in an externally complied ELPA library, it may be enabled by setting elpa_n_single to a positive integer, then the standard eigenproblems in the first elpa_n_single SCF steps are solved with single precision. The transformations between generalized eigenproblem and the standard form are always performed with double precision. Although this keyword accelerates the solution of standard eigenproblems, the overall SCF convergence may be slower, depending on the physical system and the SCF settings used in the electronic structure code.

(2) elpa_gpu: If ELPA is compiled with GPU support, GPU acceleration may be enabled by setting elpa_gpu to a nonzero integer. This keyword is ignored if no GPU support is available.

3.5.3 Customizing the libOMM Solver

elsi_set_omm_flavor(handle, omm_flavor)
elsi_set_omm_n_elpa(handle, omm_n_elpa)
elsi_set_omm_tol(handle, omm_tol)

Argument	Data Type	Default	Explanation
omm_flavor	integer	0	0: Direct minimization of a generalized eigenproblem. 2: Cholesky factorization of the overlap matrix transforming the generalized eigenproblem to the standard form.
omm_n_elpa	integer	6	Number of SCF steps using ELPA. See remark 1.
omm_tol	real double	10^{-12}	Convergence tolerance of orbital minimization. See remark 2.

Remarks

(1) omm_n_elpa: It has been demonstrated that OMM is optimal at later stages of an SCF cycle where the electronic structure is closer to its local minimum, requiring only one CG iteration to converge the minimization of the OMM energy functional. It is therefore recommended to use ELPA for omm_n_elpa SCF steps before switching to libOMM.

(2) omm_tol: A large minimization tolerance leads to a faster convergence, at the price of a lower accuracy. omm_tol should be tested and chosen to balance the desired accuracy and computation time.

3.5.4 Customizing the PEXSI Solver

elsi_set_pexsi_method(handle, pexsi_method)
elsi_set_pexsi_n_pole(handle, pexsi_n_pole)
elsi_set_pexsi_n_mu(handle, pexsi_n_mu)
elsi_set_pexsi_np_per_pole(handle, pexsi_np_per_pole)
elsi_set_pexsi_np_symbo(handle, pexsi_np_symbo)
elsi_set_pexsi_temp(handle, pexsi_temp)
elsi_set_pexsi_mu_min(handle, pexsi_mu_min)
elsi_set_pexsi_mu_max(handle, pexsi_mu_max)
elsi_set_pexsi_inertia_tol(handle, pexsi_inertia_tol)

Argument	Data Type	Default	Explanation
pexsi_method	integer	3	1: Contour integral [12]. 2: Minimax rational approximation [29]. 3: Adaptive Antoulas-Anderson (AAA) [30]. See remark 1.
pexsi_n_pole	integer	30	Number of poles used by PEXSI. See remark 1.
pexsi_n_mu	integer	2	Number of mu points used by PEXSI. See remark 2.
pexsi_np_per_pole	integer	-	Number of MPI tasks assigned to one pole. See remark 3.
pexsi_np_symbo	integer	1	Number of MPI tasks for symbolic factorization. See remark 4.
pexsi_temp	real double	0.002	Electronic temperature. See remark 5.
pexsi_inertia_tol	real double	0.05	Stopping criterion of inertia counting. See remark 6.
pexsi_mu_min	real double	-10.0	Lower bound of mu. See remark 7.
pexsi_mu_max	real double	10.0	Upper bound of mu. See remark 7.

Remarks

(1) When using the pole expansion method based on contour integral, allowed numbers for pexsi_n_pole are: 10, 20, 30, ..., 110, 120. 60 to 100 poles are typically needed to get an accuracy that is comparable with the result obtained from diagonalization. When using the minimax rational approximation or the Adaptive Antoulas-Anderson method, allowed numbers for pexsi_n_pole are: 10, 15, 20, ..., 35, 40. 20 to 30 poles are typically needed to get an accuracy that is comparable with the result obtained from diagonalization. PEXSI outputs an error message when it detects an unsupported choice of number of poles.

The electronic entropy can only be computed with the contour integral method and the Adaptive Antoulas-Anderson method. It may be accessed via elsi_get_entropy.

(2) PEXSI determines the chemical potential by performing Fermi operator expansion at several chemical potential values (referred to as "points") in an SCF step, then interpolating the results at all points to the final answer. The pexsi_n_mu parameter controls the number of chemical potential "points" to be evaluated. Two points followed by a simple linear interpolation often yield reasonable results.

(3) pexsi_np_per_pole: PEXSI has three levels of parallelism: the first level evaluates the Fermi operator at all the chemical potential points in parallel; at each chemical potential point, the second level handles the poles in parallel; finally, for each pole, parallel selected inversion is performed as the third level. The value of pexsi_np_per_pole is the number of MPI tasks assigned to one pole at one chemical potential point for the parallel selected inversion. Ideally, the total number of MPI tasks should be pexsi_np_per_pole × pexsi_n_mu × pexsi_n_pole, i.e., all the three levels of parallelism are fully exploited. In case that this is not feasible, PEXSI can also process the poles in serial, whereas all the chemical potential points must be evaluated simultaneously. The user should make sure that the total number of MPI tasks is divisible by the product of the number of MPI tasks per pole and the number of points.

When not using the "PEXSI_CSC" matrix format, pexsi_np_per_pole can be automatically determined to balance the three levels of parallelism in PEXSI. Please note that when using the "PEXSI_CSC" matrix format together with the PEXSI solver, input and output matrices should be distributed among the first pexsi_np_per_pole MPI tasks (not all) in a 1D block distribution. The block size of the distribution must be floor($N_{basis}/N_{procs\_per\_pole}$), where floor(x) is the greatest integer less than or equal to x, N_{basis} and $N_{procs\_per\_pole}$ are the number of basis functions and the value of pexsi_np_per_pole, respectively. When using the "PEXSI_CSC" matrix format with the ELPA, libOMM, EigenExa, SLEPc-SIPs, or NTPoly solver, input and output matrices should be distributed across all the MPI tasks in a 1D block distribution. Again, the block size of the distribution must be floor(N_{basis}/N_{procs}).

(4) `pexsi_np_symbo`: Unless there is a memory bottleneck, using 1 MPI task for matrix reordering and symbolic factorization is favorable. When running in serial, the matrix reordering in PT-SCOTCH or ParMETIS introduces a minimal number of "fill-ins" to the factorized matrices. Using more MPI tasks introduces more fill-ins. As the matrix reordering and symbolic factorization are performed only once per SCF cycle (with a fixed overlap matrix), using 1 MPI task should not affect the overall timing too much. On the other hand, more fill-ins lead to slower numerical factorization in every SCF step. In addition, the number of MPI tasks used for matrix reordering and symbolic factorization cannot be too large. Otherwise, the symbolic factorization may fail. Therefore, the default number of MPI tasks for symbolic factorization is 1. It is worth testing and increasing this number for large-scale calculations.

(5) `pexsi_temp`: This value corresponds to the $k_{\mathrm{B}}T$ term (not T) in the Fermi-Dirac distribution function.

(6) The chemical potential determination in PEXSI relies on an inertia counting step to narrow down the chemical potential searching interval in the first few SCF steps. The inertia counting step is skipped if the difference between `pexsi_mu_min` and `pexsi_mu_max` becomes smaller than `pexsi_inertia_tol`.

(7) PEXSI performs Fermi operator calculations at a number of points within the chemical potential search interval, based on which the chemical potential is determined. In the first SCF iteration of each geometry step, `pexsi_mu_min` and `pexsi_mu_max` should be set to safe values that guarantee the true chemical potential lies in this interval. Then, for the n^{th} SCF step, `pexsi_mu_min` should be set to $(\mu_{\mathrm{min}}^{\mathrm{n\text{-}1}} + \Delta V_{\mathrm{min}})$, `pexsi_mu_max` should be set to $(\mu_{\mathrm{max}}^{\mathrm{n\text{-}1}} + \Delta V_{\mathrm{max}})$. Here, $\mu_{\mathrm{min}}^{\mathrm{n\text{-}1}}$ and $\mu_{\mathrm{max}}^{\mathrm{n\text{-}1}}$ are the lower bound and the upper bound of the chemical potential, determined by PEXSI in the $(\mathrm{n\text{-}1})^{\mathrm{th}}$ SCF step. They can be retrieved by calling `elsi_get_pexsi_mu_min` and `elsi_get_pexsi_mu_max`, respectively (see Sec. 3.6.2). Suppose the effective potential (Hartree potential, exchange-correlation potential, and external potential) is stored in an array V, whose dimension is the number of grid points. From one SCF iteration to the next, ΔV denotes the potential change, and ΔV_{min} and ΔV_{max} are the minimum and maximum values in the array ΔV, respectively. The whole process is summarized in the pseudo-code below. The (re-)initialization and finalization of ELSI are omitted.

```
do geometry update
  mu_min = -10.0
  mu_max = 10.0
  delta_V_min = 0.0
  delta_V_max = 0.0

  do SCF cycle
    Update Hamiltonian

    call elsi_set_pexsi_mu_min (eh, mu_min + delta_V_min)
    call elsi_set_pexsi_mu_max (eh, mu_max + delta_V_max)

    call elsi_dm_{real|complex} (eh, ham, ovlp, dm, bs_energy)

    call elsi_get_pexsi_mu_min (eh, mu_min)
    call elsi_get_pexsi_mu_max (eh, mu_max)

    Update electron density
    Update potential

    delta_V_min = minval (V_new - V_old)
    delta_V_max = maxval (V_new - V_old)

    Check SCF convergence
  end do
end do
```

(8) `pexsi_gap`: The PEXSI method does not require an energy gap. Use the default value if no knowledge is available.

3.5.5 Customizing the EigenExa Solver

elsi_set_eigenexa_method(handle, eigenexa_method)

Argument	Data Type	Default	Explanation
eigenexa_method	integer	2	1: Tridiagonalization solver eigen_s. 2: Pentadiagonalization solver eigen_sx. The latter is usually faster and more scalable.

3.5.6 Customizing the SLEPc-SIPs Solver

elsi_set_sips_ev_min(handle, sips_ev_min)
elsi_set_sips_ev_max(handle, sips_ev_max)
elsi_set_sips_n_elpa(handle, sips_n_elpa)
elsi_set_sips_n_slice(handle, sips_n_slice)
elsi_set_sips_interval(handle, sips_lower, sips_upper)

Argument	Data Type	Default	Explanation
sips_ev_min	real double	-2.0	Lower bound of eigenspectrum. See remark 1.
sips_ev_max	real double	2.0	Upper bound of eigenspectrum. See remark 1.
sips_n_elpa	integer	0	Number of SCF steps using ELPA. See remark 2.
sips_n_slice	integer	1	Number of slices. See remark 3.

Remarks

(1) sips_ev_min and sips_ev_max: SLEPc-SIPs relies on an inertia counting step to estimate the lower and upper bounds of the eigenspectrum. Only eigenvalues within this interval, and their associated eigenvectors, are solved. The inertia counting starts from the interval determined by sips_ev_min and sips_ev_max. This interval may expand or shrink to make sure that it encloses the 1st to the n_stateth eigenvalues. If a good estimate of the lower or upper bounds of the eigenspectrum is available, it should be set by elsi_set_sips_ev_min or elsi_set_sips_ev_max.

(2) sips_n_elpa: The performance of SLEPc-SIPs mainly depends on the load balance across slices. Optimal performance is expected if the desired eigenvalues are evenly distributed across slices. In an SCF calculation, eigenvalues obtained in one SCF step can be used as an approximate distribution of eigenvalues in the next SCF step. This approximation should become better as the SCF cycle approaches its convergence. Using the direct eigensolver ELPA in the first sips_n_elpa SCF steps can circumvent the load imbalance of spectrum slicing in the initial SCF steps.

(3) sips_n_slice: SLEPc-SIPs partitions the eigenspectrum into slices and solves the slices in parallel. The number of slices is controlled by sips_n_slice. The default value, 1, should always work, but by no means leads to the optimal performance of the solver. There are some general rules to set this parameter. First, as a requirement of the SLEPc library, the total number of MPI tasks must by divisible by sips_n_slice. Second, setting sips_n_slice to the number of compute nodes usually yields better performance, as the inter-node communication is minimized. The optimal value of sips_n_slice depends on the actual problem as well as the hardware.

3.5.7 Customizing the NTPoly Solver

elsi_set_ntpoly_method(handle, ntpoly_method)
elsi_set_ntpoly_filter(handle, ntpoly_filter)
elsi_set_ntpoly_tol(handle, ntpoly_tol)

Argument	Data Type	Default	Explanation
ntpoly_method	integer	2	0: Canonical purification [31]. 1: 2nd order trace resetting purification [32]. 2: 4th order trace resetting purification [32]. 3: Generalized hole-particle canonical purification [33]. 1 and 2 are recommended.
ntpoly_filter	real double	10^{-15}	When performing sparse matrix multiplications, values below this filter are discarded. See remark 1.
ntpoly_tol	real double	10^{-8}	Convergence tolerance of purification. See remark 1.

Remarks

(1) `ntpoly_filter` and `ntpoly_tol` control the accuracy and computational cost of a density matrix purification method. Tight choices of `ntpoly_filter` and `ntpoly_tol`, e.g. the default values here, lead to highly accurate results that are comparable to the results obtained from diagonalization. However, linear scaling can only be achieved with a relatively large `ntpoly_filter` such as 10^{-6}. Note that the purification may not converge if `ntpoly_filter` is too large relative to `ntpoly_tol`. Setting `ntpoly_filter` to be $\leq 10^{-3}\times$ `ntpoly_tol` is safe in most cases.

3.5.8 Customizing the MAGMA Solver

`elsi_set_magma_solver`(handle, magma_solver)

Argument	Data Type	Default	Explanation
`magma_solver`	integer	1	1: One-stage solver. 2: Two-stage solver.

Remarks

(1) MAGMA can use multiple GPUs, controlled by the environment variable `MAGMA_NUM_GPUS`. Refer to the users' guide of MAGMA for more information.

3.5.9 Customizing the ChASE Solver

`elsi_set_chase_tol`(handle, resid_tol)
`elsi_set_chase_filter_deg`(handle, deg)
`elsi_set_chase_extra_space`(handle, extra_ratio)
`elsi_set_chase_min_extra_space`(handle, min_s)
`elsi_set_chase_deg_opt`(handle, is_deg_opt)
`elsi_set_chase_evecs_recycl`(handle, is_recycl)
`elsi_set_chase_cholqr`(handle, is_cholqr)

Argument	Data Type	Default	Explanation
`resid_tol`	real double	10^{-8}	Convergence tolerance of the residuals of the desired eigenpairs.
`deg`	integer	20	Initial degree of Chebyshev polynomial in ChASE.
`extra_ratio`	real double	0.25	Percentage of extra eigenvalues added to the searching space filtered by the Chebyshev polynomial. The size of the search space is then ((1+`extra_ratio`)× `n_state`).
`min_s`	integer	10	Minimal number of extra eigenvalues to be added to the search space filtered by the Chebyshev polynomial.
`is_deg_opt`	integer	1	Controls the activation of the optimization mechanism of the degree of the Cheby. polynomimal. 0: Disabled. 1: Enabled.
`is_recycl`	integer	1	Controls the use of approximate eigenvectors when they are available. 0: No. 1: Yes.
`is_cholqr`	integer	1	Controls the use of flexible Cholesky QR in ChASE. 0: use Householder QR. 1: Yes.

Remarks

(1) `resid_tol` is an optional parameter controlling the minimal value for the residual after which the corresponding eigenpair is declared converged. As a rule of thumb a minimum value of 10^{-8} and 10^{-4} are suggested for Double Precision and Single Precision, respectively. The smallest safe value for this parameter is 10^{-14}. For smaller values, the ChASE algorithm may fail to converge.

(2) `deg` controls the polynomial degree for the Chebyshev filter. When the value of `is_deg_opt` = 1, this is the initial polynomial degree used solely in the first subspace iteration. When `is_deg_opt` = 0 the same value `deg` is used for every vector for each filter call. When `is_deg_opt` = 1, it is advisable to set `deg` to a value not larger than 10. If `is_deg_opt` = 0, then it is advised to select a value not smaller than 15 but not larger than 30.

(3) `extra_ratio` specifies the search space increment such that the overall size of the search space is `n_state` + `extra_ratio` × `n_state`. In most cases 25% of the value of `n_state` is more than sufficient. The best case sce-

nario for optimal convergence is when there is a large spectrum gap between `n_state` and `n_state` + `extra_ratio` × `n_state`.

(4) `is_deg_opt` is a parameter controlling that the filter uses a vector of degrees optimized for each single filtered vector. It is advisable to keep the degree optimization enabled in order to avoid wasting extra FLOPs to reach convergence. In some cases where the spectrum of the problem is particularly challenging, the optimization could be disabled to avoid undesired fluctuation in the eigenpairs residuals.

(5) `is_recycl` controls those cases when the user can provide ChASE with approximate eigenvectors. This situation is typical for eigenproblems that are part of a Self-Consistent Field (SCF) cycle of a Density Functional Theory computation. In these cases, ChASE can take advantage of the correlation between eigenpairs of problems generated in adjacent SCF cycles. When it is enabled, ChASE can use the eigenvactors solution of an eigenproblem at a given SCF cycle as input for the problem at the next SCF cycle. By solving the entire sequence of eigenproblems in this way, ChASE can gain substantial speedups sometimes even halfing the time to solution. When `is_recycl` is set to 1, ChASE expects to receive in input two arrays holding approximate vectors and values.

(6) `is_cholqr` can be used to enable/disable a flexible Cholesky QR in ChASE. Based on an estimation of condition number of filtered vectors, this flexible Cholesky QR is able to select different variants of CholeskyQR on the fly to attain the best performance while maintain a good numerical stability. This mechanism is enabled by default, and it can be disabled by setting `is_cholqr` to 0. Then a numerical stable Housheulder QR factorization (either from LAPACK or ScaLAPACK) would be used.

3.6 Getting Additional Results from ELSI

In Sec. 3.3 and Sec. 3.4, the interfaces to compute and return the eigensolutions and the density matrices have been introduced. ELSI and the solvers may perform additional calculations whose results are useful at a certain stage of a calculation. One example is the energy-weighted density matrix that is employed to evaluate the Pulay forces during a geometry optimization calculation. The subroutines introduced in the following subsections are used to retrieve such additional results from ELSI.

3.6.1 Getting Results from the ELSI Interface

In all the subroutines listed below, the first argument (input and output) is an `elsi_handle`. The second argument (output) of each subroutine is the name of the parameter to get.

`elsi_get_version`(major, minor, patch)
`elsi_get_datestamp`(date_stamp)
`elsi_get_initialized`(handle, handle_init)
`elsi_get_n_illcond`(handle, n_illcond)
`elsi_get_ovlp_ev_min`(handle, ev_min)
`elsi_get_ovlp_ev_max`(handle, ev_max)
`elsi_get_mu`(handle, mu)
`elsi_get_entropy`(handle, ts)
`elsi_get_edm_real`(handle, edm_real)
`elsi_get_edm_complex`(handle, edm_complex)
`elsi_get_edm_real_sparse`(handle, edm_real_sparse)
`elsi_get_edm_complex_sparse`(handle, edm_complex_sparse)
`elsi_get_eval`(handle, eval)
`elsi_get_evec_real`(handle, evec_real)
`elsi_get_evec_complex`(handle, evec_complex)
`elsi_get_occ`(handle, occ)

Argument	Data Type	Explanation
`major`	integer	Major version number.
`minor`	integer	Minor version number.
`patch`	integer	Patch level.

date_stamp	integer	Date stamp of ELSI (yyyymmdd).
handle_init	integer	0 if the ELSI handle has not been initialized; 1 if initialized.
n_illcond	integer	Number of eigenvalues of the overlap matrix that are smaller than the ill-conditioning tolerance. See Sec. 3.5.1.
ovlp_ev_min	real double	Lowest eigenvalue of the overlap matrix. See remark 1.
ovlp_ev_max	real double	Highest eigenvalue of the overlap matrix. See remark 1.
mu	real double	Chemical potential. See remark 2.
ts	real double	Entropy. See remark 2.
edm_real	2D real double array	Energy-weighted density matrix in "BLACS_DENSE" format. See remark 3.
edm_complex	2D complex double array	Energy-weighted density matrix in "BLACS_DENSE" format. See remark 3.
edm_real_sparse	1D real double array	Energy-weighted density matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format. See remark 3.
edm_complex_sparse	1D complex double array	Energy-weighted density matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format. See remark 3.
eval	1D real double array	Eigenvalues. See remark 4.
evec_real	2D real double array	Eigenvectors in "BLACS_DENSE" format. See remark 4.
evec_complex	2D complex double array	Eigenvectors in "BLACS_DENSE" format. See remark 4.
occ	1D real double array	Occupation numbers. See remark 4.

Remarks

(1) Ill-conditioning check of the overlap matrix is enabled by default when ELPA is the chosen solver. It may be disabled by calling elsi_set_illcond_check, and is automatically disabled when the chosen solver is not ELPA. ovlp_ev_min and ovlp_ev_max are computed only if ill-conditioning check is enabled. Otherwise the return value may be zero.

(2) The chemical potential is available only if elsi_dm_{real|complex}{_sparse} has been called, with ELPA, PEXSI, SLEPc-SIPs, EigenExa, or NTPoly being the chosen solver. The entropy is available only if elsi_dm_{real|complex}{_sparse} has been called with ELPA, PEXSI (see also elsi_set_pexsi_method), SLEPc-SIPs, or EigenExa being the chosen solver. ELSI may return zero when the chemical potential or the entropy is not available.

(3) In general, the energy-weighted density matrix is only needed in a late stage of an SCF cycle to evaluate forces. It is, therefore, not calculated when any of the density matrix solver interface is called. When the energy-weighted density matrix is actually needed, it can be requested by calling elsi_get_edm_{real|complex}{_sparse}. These subroutines have the requirement that the corresponding elsi_dm subroutine must have been invoked. For instance, elsi_get_edm_real_sparse only makes sense if elsi_dm_real_sparse has been successfully executed.

(4) When using elsi_dm_{real|complex}{_sparse} with an eigensolver, ELSI computes and stores the eigenvalues, eigenvectors, and occupation numbers. They may be accessed by calling elsi_get_eval, elsi_get_evec_{real|complex}, and elsi_get_occ. The dimension of eval and occ should be equal to the value of n_states set in elsi_init. Even with elsi_dm_{real|complex}_sparse, the eigenvectors are returned in a dense format ("BLACS_DENSE"), as they are in general not sparse. The size of evec_{real|complex} should always correspond to a global array of size n_basis by n_basis, regardless of the value of n_states.

3.6.2 Getting Results from the PEXSI Solver

elsi_get_pexsi_mu_min(handle, pexsi_mu_min)
elsi_get_pexsi_mu_max(handle, pexsi_mu_max)

Argument	Data Type	Explanation
pexsi_mu_min	real double	Minimum value of mu. See remark 1.
pexsi_mu_max	real double	Maximum value of mu. See remark 1.

Remarks

(1) Please refer to Sec. 3.5.4 for the chemical potential determination algorithm in PEXSI and ELSI.

3.6.3 Extrapolation of Wavefunctions and Density Matrices

In geometry optimization and molecular dynamics calculations, the initial guess of the electron density in the $(n+1)^{th}$ geometry step can be constructed from the wavefunctions or density matrix calculated in the n^{th} geometry step. However, due to the movement of atoms and localized basis functions around them, wavefunctions obtained in the n^{th} geometry step are no longer orthonormalized in the $(n+1)^{th}$ geometry step. `elsi_orthonormalize_ev_{real|complex}{_sparse}` orthonormalizes eigenvectors (coefficients of wavefunctions) in the n^{th} geometry step with respect to the overlap matrix in the $(n+1)^{th}$ geometry step with a Gram-Schmidt algorithm.

`elsi_orthonormalize_ev_real`(handle, ovlp, evec)

Argument	Data Type	in/out	Explanation
`handle`	elsi_handle	inout	ELSI handle.
`ovlp`	2D real double array	in	Overlap matrix in "`BLACS_DENSE`" format.
`evec`	2D real double array	inout	Eigenvectors in "`BLACS_DENSE`" format.

`elsi_orthonormalize_ev_complex`(handle, ovlp, evec)

Argument	Data Type	in/out	Explanation
`handle`	elsi_handle	inout	ELSI handle.
`ovlp`	2D complex double array	in	Overlap matrix in "`BLACS_DENSE`" format.
`evec`	2D complex double array	inout	Eigenvectors in "`BLACS_DENSE`" format.

`elsi_orthonormalize_ev_real_sparse`(handle, ovlp, evec)

Argument	Data Type	in/out	Explanation
`handle`	elsi_handle	inout	ELSI handle.
`ovlp`	1D real double array	in	Overlap matrix in "`PEXSI_CSC`", "`SIESTA_CSC`", or "`GENERIC_COO`" sparse format.
`evec`	2D real double array	inout	Eigenvectors in "`BLACS_DENSE`" format. See remark 1.

`elsi_orthonormalize_ev_complex_sparse`(handle, ovlp, evec)

Argument	Data Type	in/out	Explanation
`handle`	elsi_handle	inout	ELSI handle.
`ovlp`	1D complex double array	in	Overlap matrix in "`PEXSI_CSC`", "`SIESTA_CSC`", or "`GENERIC_COO`" sparse format.
`evec`	2D complex double array	inout	Eigenvectors in "`BLACS_DENSE`" format. See remark 1.

Remarks

(1) Even when using `elsi_orthonormalize_ev_{real|complex}_sparse`, the eigenvectors are still stored in a dense format ("`BLACS_DENSE`"), as they are in general not sparse.

`elsi_extrapolate_dm_{real|complex}{_sparse}` extrapolates density matrix in the n^{th} geometry step to the overlap matrix in the $(n+1)^{th}$ geometry step. `elsi_set_save_ovlp` must have been called to store the relevant matrices in the n^{th} geometry step within ELSI.

`elsi_extrapolate_dm_real`(handle, ovlp, dm)

Argument	Data Type	in/out	Explanation
`handle`	elsi_handle	inout	ELSI handle.
`ovlp`	2D real double array	in	New overlap matrix in "`BLACS_DENSE`" format.
`dm`	2D real double array	out	New density matrix in "`BLACS_DENSE`" format.

elsi_extrapolate_dm_complex(handle, ovlp, dm)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ovlp	2D complex double array	in	New overlap matrix in "BLACS_DENSE" format.
dm	2D complex double array	out	New density matrix in "BLACS_DENSE" format.

elsi_extrapolate_dm_real_sparse(handle, ovlp, dm)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ovlp	1D real double array	in	New overlap matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
dm	1D real double array	out	New density matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.

elsi_extrapolate_dm_complex_sparse(handle, ovlp, dm)

Argument	Data Type	in/out	Explanation
handle	elsi_handle	inout	ELSI handle.
ovlp	1D complex double array	in	New overlap matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.
dm	1D complex double array	out	New density matrix in "PEXSI_CSC", "SIESTA_CSC", or "GENERIC_COO" sparse format.

3.7 Parallel Matrix I/O

The ELSI interface is able to read and write distributed matrices in parallel. There exist a number of libraries for high-performance parallel I/O that are particularly capable of reading and writing a large amount of data with hierarchical structures and complex metadata. However, the data structure in ELSI is simply arrays that represent matrices, with a few integers to define the dimension of the matrices. In order to circumvent the development and performance overhead associated with a high level I/O library, ELSI directly uses the parallel I/O functionality defined in the MPI standard.

Writing the distributed matrices into N_{procs} separate files, where N_{procs} is the number of MPI tasks, is not preferred, as manipulating a large number of files would be difficult. The implementation of matrix I/O in ELSI adopts collective MPI I/O routines to write data to (read data from) a single binary file, as if the data was gathered onto a single MPI task then written to one file (read from one file by one MPI task then scattered to all tasks). The optimal I/O performance, both with MPI I/O and in general, is achieved by making large and contiguous requests to access the file system. Therefore, ELSI always redistributes the matrices to a 1D block distribution before writing it to file. This guarantees that each MPI task writes a contiguous chunk of data to a contiguous piece of file. Similarly, matrices read from file are in a 1D block distribution, and can be redistributed automatically if needed. ELSI always stores matrices in a sparse CSC format. The conversion between dense and sparse formats is handled automatically.

3.7.1 Setting Up Matrix I/O

An elsi_rw_handle must be initialized via the elsi_init_rw subroutine before any other matrix I/O subroutine may be called. This elsi_rw_handle must be passed to all other matrix I/O subroutine calls.

elsi_init_rw(handle, task, parallel_mode, n_basis, n_electron)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	out	ELSI matrix I/O handle.
task	integer	in	0: READ_MATRIX. 1: WRITE_MATRIX.
parallel_mode	integer	in	0: SINGLE_PROC. 1: MULTI_PROC. See elsi_init.
n_electron	real double	in	Number of electrons. See remark 1.
n_basis	integer	in	Number of basis functions, i.e. global size of matrix.

Remarks

(1) `n_electron`: Matrices written out with ELSI matrix I/O are usually from actual electronic structure calculations. Having the number of electrons available makes the matrix file useful for testing density matrix solvers such as PEXSI. Therefore, it is recommended to set the correct number of electrons when initializing an matrix I/O handle, although setting it to an arbitrary number does not affect the matrix I/O operation.

(2) `n_basis`: This can be set to an arbitrary value if `task` is "`READ_MATRIX`". Its value is read from file when calling `elsi_read_mat_dim` or `elsi_read_mat_dim_sparse`.

The MPI communicator which encloses the MPI tasks to perform the matrix I/O operation needs to be passed into ELSI via the `elsi_set_rw_mpi` subroutine.

`elsi_set_rw_mpi`(handle, comm)

Argument	Data Type	in/out	Explanation
`handle`	elsi_rw_handle	inout	ELSI matrix I/O handle.
`comm`	integer	in	MPI communicator.

When reading or writing a dense matrix, BLACS parameters are passed into ELSI via the `elsi_set_rw_blacs` subroutine.

`elsi_set_rw_blacs`(handle, blacs_ctxt, block_size)

Argument	Data Type	in/out	Explanation
`handle`	elsi_rw_handle	inout	ELSI matrix I/O handle.
`blacs_ctxt`	integer	in	BLACS context.
`block_size`	integer	in	Block size of the 2D block-cyclic distribution, specifying both row and column directions.

When writing a sparse matrix, its dimensions are passed into ELSI via the `elsi_set_rw_csc` subroutine. The only sparse matrix format currently supported by ELSI matrix I/O is the "`PEXSI_CSC`" format. When reading a sparse matrix, there is no need to call this subroutine. The relevant parameters are read from file when calling `elsi_read_mat_dim` or `elsi_read_mat_dim_sparse`.

`elsi_set_rw_csc`(handle, global_nnz, local_nnz, local_col)

Argument	Data Type	in/out	Explanation
`handle`	elsi_rw_handle	inout	ELSI matrix I/O handle.
`global_nnz`	integer	in	Global number of non-zeros.
`local_nnz`	integer	in	Local number of non-zeros.
`local_col`	integer	in	Local number of matrix columns.

When a matrix I/O instance is no longer needed, its associated handle should be cleaned up by calling `elsi_finalize_rw`.

`elsi_finalize_rw`(handle)

Argument	Data Type	in/out	Explanation
`handle`	elsi_rw_handle	inout	ELSI matrix I/O handle.

3.7.2 Writing Matrices

`elsi_write_mat_{real|complex}` writes a dense matrix to file. Before writing a dense matrix, MPI and BLACS should be set up properly using `elsi_set_rw_mpi` and `elsi_set_rw_blacs`.

`elsi_write_mat_real`(handle, filename, mat)

Argument	Data Type	in/out	Explanation
`handle`	elsi_rw_handle	in	ELSI matrix I/O handle.
`filename`	string	in	Name of file to write.
`mat`	2D real double array	in	Matrix in "`BLACS_DENSE`" format.

elsi_write_mat_complex(handle, filename, mat)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	in	ELSI matrix I/O handle.
filename	string	in	Name of file to write.
mat	2D complex double array	in	Matrix in "BLACS_DENSE" format.

elsi_write_mat_{real|complex}_sparse writes a sparse matrix to file. Before writing a sparse matrix, MPI and CSC matrix format should be set up properly using elsi_set_rw_mpi and elsi_set_rw_csc.

elsi_write_mat_real_sparse(handle, filename, row_idx, col_ptr, mat)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	in	ELSI matrix I/O handle.
filename	string	in	Name of file to write.
row_idx	1D integer array	in	Local row index array.
col_ptr	1D integer array	in	Local column pointer array.
mat	1D real double array	in	Matrix in "PEXSI_CSC" format.

elsi_write_mat_complex_sparse(handle, filename, row_idx, col_ptr, mat)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	in	ELSI matrix I/O handle.
filename	string	in	Name of file to write.
row_idx	1D integer array	in	Local row index array.
col_ptr	1D integer array	in	Local column pointer array.
mat	1D complex double array	in	Matrix in "PEXSI_CSC" format.

When writing a dense matrix to file, values smaller than a predefined threshold are discarded. The default value of this threshold is 10^{-15}. It can be overridden via elsi_set_rw_zero_def.

elsi_set_rw_zero_def(handle, zero_def)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	inout	ELSI matrix I/O handle.
zero_def	real double	in	When writing a dense matrix to file, values below this threshold are discarded.

An array of eight user-defined integers can be optionally set up via elsi_set_rw_header. This array is attached to the matrix file written out by elsi_write_mat_{real|complex}{_sparse}. When reading a matrix file, this array may be retrieved via elsi_get_rw_header.

elsi_set_rw_header(handle, header)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	inout	ELSI matrix I/O handle.
header	1D integer array	in	An array of eight integers.

3.7.3 Reading Matrices

elsi_real_mat_{real|complex}{_sparse} reads a dense or sparse matrix from file. While writing a matrix to file can be done in one step, it is easier to read a matrix from file in two steps, i.e., first read the dimension of the matrix and allocate memory accordingly, then read the actual data of the matrix.

Before reading a dense matrix, MPI and BLACS should be set up properly using elsi_set_rw_mpi and elsi_set_rw_blacs. elsi_read_mat_dim is used to read the dimension of a matrix, including the number of electrons in the physical system (for testing purpose), the global size of the matrix, and the local size of the matrix. Memory needs to be allocated according to the return values of local_row and local_col. Then elsi_read_mat_{real|complex} may be called to read a real or complex matrix, respectively.

elsi_read_mat_dim(handle, filename, n_electron, n_basis, local_row, local_col)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	inout	ELSI matrix I/O handle.
filename	string	in	Name of file to read.
n_electron	real double	out	Number of electrons.
n_basis	integer	out	Number of basis functions, i.e. global size of matrix.
local_row	integer	out	Local number of matrix rows.
local_col	integer	out	Local number of matrix columns.

elsi_read_mat_real(handle, filename, mat)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	inout	ELSI matrix I/O handle.
filename	string	in	Name of file to read.
mat	2D real double array	out	Matrix in "BLACS_DENSE" format.

elsi_read_mat_complex(handle, filename, mat)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	inout	ELSI matrix I/O handle.
filename	string	in	Name of file to read.
mat	2D complex double array	out	Matrix in "BLACS_DENSE" format.

Before reading a sparse matrix, MPI should be set up properly using elsi_set_rw_mpi. elsi_read_mat_dim_sparse is used to read the dimension of a matrix, including the number of electrons in the physical system (for testing purpose), the global size of the matrix, and the local size of the matrix. Memory needs to be allocated according to the return values of local_nnz and local_col. Then elsi_read_mat_{real|complex}_sparse may be called to read a real or complex matrix, respectively.

elsi_read_mat_dim_sparse(handle, filename, n_electron, n_basis, global_nnz, local_nnz, local_col)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	inout	ELSI matrix I/O handle.
filename	string	in	Name of file to read.
n_electron	real double	out	Number of electrons.
n_basis	integer	out	Number of basis functions, i.e. global size of matrix.
global_nnz	integer	out	Global number of non-zeros.
local_nnz	integer	out	Local number of non-zeros.
local_col	integer	out	Local number of matrix columns.

elsi_read_mat_real_sparse(handle, filename, row_idx, col_ptr, mat)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	inout	ELSI matrix I/O handle.
filename	string	in	Name of file to read.
row_idx	1D integer array	out	Local row index array.
col_ptr	1D integer array	out	Local column pointer array.
mat	1D real double array	out	Matrix in "PEXSI_CSC" format.

elsi_read_mat_complex_sparse(handle, filename, row_idx, col_ptr, mat)

Argument	Data Type	in/out	Explanation
handle	elsi_rw_handle	inout	ELSI matrix I/O handle.
filename	string	in	Name of file to read.
row_idx	1D integer array	out	Local row index array.
col_ptr	1D integer array	out	Local column pointer array.
mat	1D complex double array	out	Matrix in "PEXSI_CSC" format.

An array of eight user-defined integers can be optionally set up via `elsi_set_rw_header`. This array is attached to the matrix file written out by `elsi_write_mat_{real|complex}{_sparse}`. When reading a matrix file, this array may be retrieved via `elsi_get_rw_header`.

`elsi_get_rw_header`(handle, header)

Argument	Data Type	in/out	Explanation
`handle`	elsi_rw_handle	inout	ELSI matrix I/O handle.
`header`	1D integer array	out	An array of eight integers.

3.8 C/C++ Interface

ELSI is written in Fortran. A C interface around the core Fortran code is provided, which can be called from a C or C++ program. Each C wrapper function corresponds to a Fortran subroutine, where we have prefixed the original Fortran subroutine name with `c_` for clarity and consistency. Argument lists are identical to the associated native Fortran subroutine. For the complete definition of the C interface, the user is encouraged to look at the `elsi.h` header file directly.

3.9 Example Pseudo-Code

Typical workflow of ELSI within an electronic structure code is demonstrated by the following pseudo-code. In the "`test`" directory of the ELSI package, there are also examples that showcase the usage of ELSI in C and Fortran.

2D Block-Cyclic Distributed Dense Matrix + ELSI Eigensolver Interface

```
SCF initialize

call elsi_init (eh, ELPA, MULTI_PROC, BLACS_DENSE, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)

do SCF cycle
  Update Hamiltonian

  call elsi_ev_{real|complex} (eh, ham, ovlp, eval, evec)

  Update electron density
  Check SCF convergence
end do

call elsi_finalize (eh)
```

1D Block-Cyclic Distributed CSC Sparse Matrix + ELSI Eigensolver Interface

```
SCF initialize

call elsi_init (eh, ELPA, MULTI_PROC, SIESTA_CSC, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)
call elsi_set_csc (eh, global_nnz, local_nnz, local_col, row_idx, col_ptr)
call elsi_set_csc_blk (eh, block_size_csc)

do SCF cycle
  Update Hamiltonian

  call elsi_ev_{real|complex}_sparse (eh, ham, ovlp, eval, evec)

  Update electron density
  Check SCF convergence
end do

call elsi_finalize (eh)
```

Remarks

(1) Eigenvectors are returned in the "BLACS_DENSE" format, which is required to be properly set up.

Arbitrarily Distributed COO Sparse Matrix + ELSI Eigensolver Interface

```
SCF initialize

call elsi_init (eh, ELPA, MULTI_PROC, GENERIC_COO, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)
call elsi_set_coo (eh, global_nnz, local_nnz, row_idx, col_idx)

do SCF cycle
  Update Hamiltonian

  call elsi_ev_{real|complex}_sparse (eh, ham, ovlp, eval, evec)

  Update electron density
  Check SCF convergence
end do

call elsi_finalize (eh)
```

Remarks

(1) Eigenvectors are returned in the "BLACS_DENSE" format, which is required to be properly set up.

2D Block-Cyclic Distributed Dense Matrix + ELSI Density Matrix Interface

```
SCF initialize

call elsi_init (eh, OMM, MULTI_PROC, BLACS_DENSE, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)

do SCF cycle
  Update Hamiltonian

  call elsi_dm_{real|complex} (eh, ham, ovlp, dm, bs_energy)

  Update electron density
  Check SCF convergence
end do

call elsi_finalize (eh)
```

1D Block-Cyclic Distributed CSC Sparse Matrix + ELSI Density Matrix Interface

```
SCF initialize

call elsi_init (eh, PEXSI, MULTI_PROC, SIESTA_CSC, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_csc (eh, global_nnz, local_nnz, local_col, row_idx, col_ptr)
call elsi_set_csc_blk (eh, block_size)

do SCF cycle
  Update Hamiltonian

  call elsi_dm_{real|complex}_sparse (eh, ham, ovlp, dm, bs_energy)
  call elsi_get_edm_{real|complex}_sparse (eh, edm)

  Update electron density
  Check SCF convergence
end do

call elsi_finalize (eh)
```

Remarks

(1) Refer to Sec. 3.5.4 for the chemical potential determination algorithm in PEXSI.

Arbitrarily Distributed COO Sparse Matrix + ELSI Density Matrix Interface

```
SCF initialize

call elsi_init (eh, PEXSI, MULTI_PROC, GENERIC_COO, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_coo (eh, global_nnz, local_nnz, row_idx, col_idx)

do SCF cycle
  Update Hamiltonian

  call elsi_dm_{real|complex}_sparse (eh, ham, ovlp, dm, bs_energy)
  call elsi_get_edm_{real|complex}_sparse (eh, edm)

  Update electron density
  Check SCF convergence
end do

call elsi_finalize (eh)
```

Remarks

(1) Refer to Sec. 3.5.4 for the chemical potential determination algorithm in PEXSI.

Multiple k-points Calculations

```
SCF initialize

call elsi_init (eh, NTPOLY, MULTI_PROC, BLACS_DENSE, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)
call elsi_set_kpoint (eh, n_kpt, i_kpt, i_wt)
call elsi_set_mpi_global (eh, mpi_comm_global)

do SCF cycle
  Update Hamiltonian

  call elsi_dm_{real|complex} (eh, ham, ovlp, dm, bs_energy)
  call elsi_get_edm_{real|complex} (eh, edm)

  Update electron density
  Check SCF convergence
end do

call elsi_finalize (eh)
```

Remarks

(1) When there are multiple k-points, there is no change in the way ELSI solver interfaces are called.

(2) The electronic structure code needs to assemble the real-space density from the density matrices returned for the k-points. The returned band structure energy, however, is already summed over all k-points with respect to the weight of each k-point. Refer to Sec. 3.2.4 for more information.

(3) Spin-polarized calculations may be set up similarly.

Geometry Relaxation Calculations

```
SCF initialize

call elsi_init (eh, ...)
call elsi_set_* (eh, ...)

do geometry
  do SCF cycle
    Update Hamiltonian

    call elsi_{ev|dm}_{real|complex} (eh, ham, ovlp, ...)

    Update electron density
    Check SCF convergence
  end do

  Update geometry (overlap)

  call elsi_reinit (eh)
end do

call elsi_finalize (eh)
```

Standard Eigenproblem

```
call elsi_init (eh, ELPA, MULTI_PROC, BLACS_DENSE, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)
call elsi_set_unit_overlap (eh, 1)
call elsi_ev_{real|complex} (eh, mat, dummy, eval, evec)
call elsi_finalize (eh)
```

Bethe-Salpeter Eigenproblem

```
call elsi_init (eh, BSEPACK, MULTI_PROC, BLACS_DENSE, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)
call elsi_bse_{real|complex} (eh, A, B, eval, evec)
call elsi_finalize (eh)
```

Static excitations through the Delta-SCF method

This subroutine allows the user to excite a `n_excited_electrons` number of integer or fractional electrons from the valence band maximum to the conduction band minimum.

```
call elsi_static_excitations (eh, n_electron, n_state, n_spin, n_kpt,
k_wt, eval, occ, mu, mu_lower, mu_upper, n_excited_electrons, excitation_type)
```

Remarks
(1) This subroutine should be called at each SCF iteration.
(2) `occ` outputs the occupation number matrix for the excited state with a number of `n_excited_electrons` excited electrons.
(3) `excitation_type` corresponds to either `scf` or `nscf`, the former for self-consistent calculations and the latter for non-self consistent calculations such as bandstructures, which should be followed by a converged SCF calculation.

Bibliography

[1] V. Yu, F. Corsetti, A. García, W. P. Huhn, M. Jacquelin, W. Jia, B. Lange, L. Lin, J. Lu, W. Mi, A. Seifitokaldani, A. Vázquez-Mayagoitia, C. Yang, H. Yang, V. Blum, ELSI: A unified software interface for Kohn-Sham electronic structure solvers, Computer Physics Communications 222 (2018) 267–285.

[2] B. Aradi, B. Hourahine, T. Frauenheim, DFTB+, a sparse matrix-based implementation of the DFTB method, The Journal of Physical Chemistry A 111 (2007) 5678–5684.

[3] W. Hu, L. Lin, C. Yang, DGDFT: A massively parallel method for large scale density functional theory calculations, The Journal of Chemical Physics 143 (2015) 124110.

[4] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, M. Scheffler, Ab initio molecular simulations with numeric atom-centered orbitals, Computer Physics Communications 180 (2009) 2175–2196.

[5] J. M. Soler, E. Artacho, J. D. Gale, A. García, J. Junquera, P. Ordejón, D. Sánchez-Portal, The SIESTA method for ab initio order-N materials simulation, Journal of Physics: Condensed Matter 14 (2002) 2745–2779.

[6] T. Auckenthaler, V. Blum, H. J. Bungartz, T. Huckle, R. Johanni, L. Kramer, B. Lang, H. Lederer, P. R. Willems, Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations, Parallel Computing 37 (2011) 783–794.

[7] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H. J. Bungartz, H. Lederer, The ELPA library: Scalable parallel eigenvalue solutions for electronic structure theory and computational science, Journal of Physics: Condensed Matter 26 (2014) 213201.

[8] P. Kůs, H. Lederer, A. Marek, GPU optimization of large-scale eigenvalue solver, in: Numerical Mathematics and Advanced Applications ENUMATH 2017, Springer International Publishing, 2019, pp. 123–131.

[9] P. Kůs, A. Marek, S. Koecher, H.-H. Kowalski, C. Carbogno, C. Scheurer, K. Reuter, M. Scheffler, H. Lederer, Optimizations of the eigensolvers in the ELPA library, Parallel Computing 85 (2019) 167–177.

[10] F. Corsetti, The orbital minimization method for electronic structure calculations with finite-range atomic basis sets, Computer Physics Communications 185 (2014) 873–883.

[11] L. Lin, J. Lu, L. Ying, R. Car, W. E, Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems, Communications in Mathematical Sciences 7 (2009) 755–777.

[12] L. Lin, M. Chen, C. Yang, L. He, Accelerating atomic orbital-based electronic structure calculation via pole expansion and selected inversion, Journal of Physics: Condensed Matter 25 (2013) 295501.

[13] L. Lin, A. García, G. Huhs, C. Yang, SIESTA-PEXSI: Massively parallel method for efficient and accurate ab initio materials simulation without matrix diagonalization, Journal of Physics: Condensed Matter 26 (2014) 305503.

[14] M. Jacquelin, L. Lin, C. Yang, PSelInv - A distributed memory parallel algorithm for selected inversion: The symmetric case, ACM Transactions on Mathematical Software 43 (2016) 21.

[15] W. Jia, L. Lin, Robust determination of the chemical potential in the pole expansion and selected inversion method for solving Kohn-Sham density functional theory, The Journal of Chemical Physics 147 (2017) 144107.

[16] T. Imamura, S. Yamada, M. Machida, Development of a high-performance eigensolver on a peta-scale next-generation supercomputer system, Progress in Nuclear Science and Technology (2011) 643–650.

[17] T. Fukaya, T. Imamura, Performance evaluation of the EigenExa eigensolver on Oakleaf-FX: Tridiagonalization versus pentadiagonalization, in: IEEE International Parallel and Distributed Processing Symposium Workshop, 2015, pp. 960–969.

[18] V. Hernandez, J. E. Roman, V. Vidal, SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems, ACM Transactions on Mathematical Software 31 (2005) 351–362.

[19] C. Campos, J. E. Roman, Strategies for spectrum slicing based on restarted Lanczos methods, Numerical Algorithms 60 (2012) 279–295.

[20] M. Keçeli, H. Zhang, P. Zapol, D. A. Dixon, A. F. Wagner, Shift-and-invert parallel spectral transformation eigensolver: Massively parallel performance for density-functional based tight-binding, Journal of Computational Chemistry 37 (2016) 448–459.

[21] M. Keçeli, F. Corsetti, C. Campos, J. E. Roman, H. Zhang, Álvaro Vázquez-Mayagoitia, P. Zapol, A. F. Wagner, SIESTA-SIPs: Massively parallel spectrum-slicing eigensolver for an ab initio molecular dynamics package, Journal of Computational Chemistry 39 (2018) 1806–1814.

[22] W. Dawson, T. Nakajima, Massively parallel sparse matrix function calculations with NTPoly, Computer Physics Communications 225 (2018) 154–165.

[23] M. Shao, H. Felipe, C. Yang, J. Deslippe, S. G. Louie, Structure preserving parallel algorithms for solving the Bethe-Salpeter eigenvalue problem, Linear Algebra and its Applications 488 (2016) 148–167.

[24] J. Winkelmann, P. Springer, E. D. Napoli, Chase: Chebyshev accelerated subspace iteration eigensolver for sequences of hermitian eigenvalue problems, ACM Transactions on Mathematical Software (TOMS) 45 (2) (2019) 1–34.

[25] X. Wu, D. Davidović, S. Achilles, E. Di Napoli, Chase: A distributed hybrid cpu-gpu eigensolver for large-scale hermitian eigenvalue problems, in: PASC '22: Proceedings of the Platform for Advanced Scientific Computing Conference, 2022.

[26] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK users' guide, SIAM, 1999.

[27] S. Tomov, J. Dongarra, M. Baboulin, Towards dense linear algebra for hybrid GPU accelerated manycore systems, Parallel Computing 36 (2010) 232–240.

[28] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, I. Yamazaki, Accelerating numerical dense linear algebra calculations with GPUs, in: Numerical Computations with GPUs, Springer, 2014, pp. 3–28.

[29] J. E. Moussa, Minimax rational approximation of the Fermi-Dirac distribution, The Journal of Chemical Physics 145 (2016) 164108.

[30] Y. Nakatsukasa, O. Sète, L. N. Trefethen, The AAA algorithm for rational approximation, SIAM Journal on Scientific Computing 40 (2018) A1494–A1522.

[31] A. H. R. Palser, D. E. Manolopoulos, Canonical purification of the density matrix in electronic-structure theory, Physical Review B 58 (1998) 12704–12711.

[32] A. M. N. Niklasson, Expansion algorithm for the density matrix, Physical Review B 66 (2002) 155115.

[33] L. A. Truflandier, R. M. Dianzinga, D. R. Bowler, Communication: Generalized canonical purification for density matrix minimization, The Journal of Chemical Physics 144 (2016) 091102.

License and Copyright

ELSI interface software is licensed under the 3-clause BSD license:

```
Copyright (c) 2015-2023, the ELSI team.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted
provided that the following conditions are met:

1) Redistributions of source code must retain the above copyright notice, this list of
   conditions and the following disclaimer.

2) Redistributions in binary form must reproduce the above copyright notice, this list of
   conditions and the following disclaimer in the documentation and/or other materials
   provided with the distribution.

3) Neither the name of the "ELectronic Structure Infrastructure (ELSI)" project nor the names
   of its contributors may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL COPYRIGHT HOLDER BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

The source code of ELPA 2020.05.001, 2021.11.001 and 2023.05.001 (LGPL3), libOMM 1.0.0 (BSD2), NTPoly 2.7.0 (MIT), PEXSI 1.2.0 (BSD3), PT-SCOTCH 6.1.0 (CeCILL-C), SuperLU_DIST 6.2.0 (BSD3), BSEPACK 0.1 (BSD3), ChASE 1.4.0 (BSD3) are redistributed through this version of ELSI. Individual license of each library can be found in the corresponding subfolder.