PEXSI Documentation

Lin Lin, Mathias Jacquelin, Weile Jia

Jan 06, 2019

Contents:

CHAPTER 1

Introduction

1.1 Overview

The Pole EXpansion and Selected Inversion (PEXSI) method is a fast method for electronic structure calculation based on Kohn-Sham density functional theory. It efficiently evaluates certain *selected elements* of matrix functions, e.g., the Fermi-Dirac function of the KS Hamiltonian, which yields a density matrix. It can be used as an alternative to diagonalization methods for obtaining the density, energy and forces in electronic structure calculations. The PEXSI library is written in C++, and uses message passing interface (MPI) to parallelize the computation on distributed memory computing systems and achieve scalability on more than 10,000 processors.

From numerical linear algebra perspective, the PEXSI library can be used as a general tool for evaluating certain *selected elements* of a matrix function, and therefore has application beyond electronic structure calculation as well.

Given a sparse Hermitian matrix A and a certain function $f(\cdot)$, the basic idea of PEXSI is to expand $f(A)$ using a small number of rational functions (pole expansion)

$$f(A) \approx \sum_{l=1}^{P} \omega_l (A - z_l I)^{-1}$$

and to efficiently evaluate $f(A)_{i,j}$ by evaluating selected elements $(A - z_l I)^{-1}_{i,j}$ (selected inversion).

The currently supported form of $f(\cdot)$ include:

- $f(z) = z^{-1}$: Matrix inversion. Since the matrix inversion is already represented as a single term of rational function (pole), no pole expansion is needed. The selected inversion method can be significantly faster than directly inverting the matrix and then extract the selected elements of the inverse. For only using PEXSI to evaluate selected elements of A^{-1}, see *Selected Inversion of complex* for an example.
- $f(z) = \frac{2}{1+e^{\beta(z-\mu)}}$: Fermi-Dirac function. This can be used as a "smeared" matrix sign function at $z = \mu$, without assuming a spectral gap near $z = \mu$. This can be used for evaluating the electron density for electronic structure calculation. See *Solving Kohn-Sham DFT: I* for an example using PEXSI for electronic structure calculation.

For sparse matrices, the PEXSI method can be more efficient than the widely used *Diagonalization method* for evaluating matrix functions, especially when a relatively large number of eigenpairs are needed to be computed in the

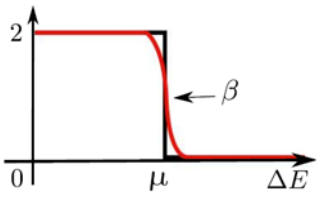

Fig. 1: Red: Fermi-Dirac function. Black: Matrix sign function

diagonalization method. PEXSI can also be used to compute the matrix functions associated with generalized eigenvalue problems, i.e.

$$f(A,B) := Vf(\Lambda)V^{-1} \approx \sum_{l=1}^{P} \omega_l (A - z_l B)^{-1}$$

where V, Λ are defined through the generalized eigenvalue problem $AV = BV\Lambda$.

PEXSI is most advantageous when a large number of processors are available, due to the two-level parallelism. It is most advantageous to use PEXSI when at least 1000 cores are available, and for many problems PEXSI can scale to tens of thousands of cores.

For details of the implementation of parallel selected inversion used in PEXSI, please see *TOMS2016* below.

Diagonalization method

- The diagonalization method evaluates a matrix function $f(A)$ by $f(A) = Vf(\Lambda)V^{-1}$, where the orthonormal matrix $V = [v_1, \ldots, v_N]$, and the diagonal matrix $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$ are defined through the eigenvalue problem $AV = V\Lambda$ It is often the case that not all eigenvalues λ_i are needed to be computed, depending on the value of $f(\lambda_i)$.

Selected elements

- For (real or complex) symmetric matrices (i.e. $A_{i,j} \neq 0$ implies $A_{j,i} \neq 0$), we define the selected elements of a matrix B with respect to a matrix A as the set $\{B_{i,j} | A_{i,j} \neq 0\}$.
- A commonly used case in PEXSI is the selected elements of A^{-1} with respect to a (real or complex) symmetric matrix A, or simply the selected elements of A^{-1}, which corresponds to the set $\{A^{-1}_{i,j} | A_{i,j} \neq 0\}$.
- For general non-symmetric matrices, the selected elements are $\{B_{i,j} | A_{j,i} \neq 0\}$. **NOTE**: this means that the matrix elements computed corresponding to the sparsity pattern of A^T (for more detailed explanation of the mathematical reason, please see the paper *PC2018* below). However, storing the matrix elements $\{A^{-1}_{i,j} | A_{j,i} \neq 0\}$ is practically cumbersome, especially in the context of distributed computing. Hence we choose to store the selected elements for A^{-T}, i.e. $\{A^{-T}_{i,j} | A_{i,j} \neq 0\}$. These are the values obtained from the non-symmetric version of PSelInv.
- One particular problem arising from electronic structure theory is when the Hamiltonian matrix and the overlap matrix H, S are Hermitian matrices, and the matrix to be inverted are of the form $(H - zS)^{-1}$. This matrix is only structurally symmetric, and we need to call the non-symmetric version of PSelInv. From the discussion above, the computed matrix entries are actually $\{(H - zS)^{-T}_{i,j} | A_{i,j} \neq 0\}$. Combining with the pole expansion, we obtain P^T, which is the transpose of the density matrix. Since the density matrix Hermitian, we have $P = P^* = \overline{P^T}$. Hence we perform an extra conjugation as a post-processing step to obtain the correct density matrix.

1.2 PEXSI used in external packages

Please let us know if PEXSI is useful for your application or software package!

- BigDFT
  - http://bigdft.org/Wiki/index.php?title=BigDFT_website
- CP2K
  - Tutorials for compiling CP2K-PEXSI: http://www.cp2k.org/howto:compile
- DFTB+
  - https://www.dftbplus.org/
- ESL Bundle
  - https://github.com/ElectronicStructureLibrary/esl-bundle
- FHI-aims
  - https://aimsclub.fhi-berlin.mpg.de/
- Quantumwise ATK
  - User manual: http://docs.quantumwise.com/manuals/Types/PEXSISolver/PEXSISolver.html
- SIESTA
  - Download link for SIESTA-PEXSI: https://launchpad.net/siesta
- Electronic Structure Infrastructure (ELSI) Project
  - http://www.elsi-interchange.org/webpage/
  - In the future, the support of PEXSI for electronic structure software packages will be provided mainly through the ELSI project.

1.3 License

You are under no obligation whatsoever to provide any bug fixes, patches, or upgrades to the features, functionality or performance of the source code ("Enhancements") to anyone; however, if you choose to make your Enhancements available either publicly, or directly to Lawrence Berkeley National Laboratory, without imposing a separate written license agreement for such Enhancements, then you hereby grant the following license: a non-exclusive, royalty-free perpetual license to install, use, modify, prepare derivative works, incorporate into other computer software, distribute, and sublicense such enhancements or derivative works thereof, in binary and source code form.

1.4 Citing PEXSI

If you use PEXSI for electronic structure calculation in general, **please cite the following two papers.**:

```
@Article{CMS2009,
  Title                    = {Fast algorithm for extracting the diagonal of the␣
↪inverse matrix with application to the electronic structure analysis of metallic␣
↪systems},
  Author                   = {Lin, L. and Lu, J. and Ying, L. and Car, R. and E, W.},
  Journal                  = {Comm. Math. Sci.},
  Year                     = {2009},
  Pages                    = {755},
  Volume                   = {7}
}

@Article{JCPM2013,
  Title                    = {Accelerating atomic orbital-based electronic structure␣
↪calculation via pole expansion and selected inversion},
  Author                   = {Lin, L. and Chen, M. and Yang, C. and He, L.},
  Journal                  = {J. Phys. Condens. Matter},
  Year                     = {2013},
  Pages                    = {295501},
  Volume                   = {25}
}
```

If you use PEXSI for selected inversion (PSelInv), **please also cite the following paper.**:

```
@Article{TOMS2016,
  Title                    = {{PSelInv}--A distributed memory parallel algorithm for␣
↪selected inversion: the symmetric case},
  Author                   = {Jacquelin, M. and Lin, L. and Yang, C.},
  Journal                  = {ACM Trans. Math. Software},
  Year                     = {2016},
  Pages                    = {21},
  Volume                   = {43}
}
```

If you use the non-symmetric version of PSelInv, **please also cite the following paper.**:

```
@Article{PC2018,
  Title                    = {{PSelInv}--A distributed memory parallel algorithm for␣
↪selected inversion: the non-symmetric case},
  Author                   = {Jacquelin, M. and Lin, L. and Yang, C.},
  Journal                  = {Parallel Comput.},
  Year                     = {2018},
  Pages                    = {74},
  Volume                   = {84}
}
```

More references on method development:

M. Jacquelin, L. Lin and C. Yang, PSelInv ? A distributed memory parallel algorithm for selected inversion: the non-symmetric case, Parallel Comput. 74, 84, 2018 link.

M. Jacquelin, L. Lin, W. Jia, Y. Zhao and C. Yang, A left-looking selected inversion algorithm and task parallelism on shared memory systems, HPC Asia, 54, 2018

V. Wen-zhe Yu, F. Corsetti, A. Garcia, W. Huhn, M. Jacquelin, W. Jia, B. Lange, L. Lin, J. Lu, W. Mi, A. Seifitokaldani, A. Vazquez-Mayagoitia, C. Yang, H. Yang and V. Blum, ELSI: A Unified Software Interface for Kohn-Sham Electronic Structure Solvers, Commun. Phys. Comput. 222, 267, 2018 link.

W. Jia and L. Lin, Robust Determination of the Chemical Potential in the Pole Expansion and Selected Inversion Method for Solving Kohn-Sham density functional theory, J. Chem. Phys. 147, 144107, 2017 link.

M. Jacquelin, L. Lin, N. Wichmann and C. Yang, Enhancing the scalability and load balancing of the parallel selected inversion algorithm via tree-based asynchronous communication, IEEE IPDPS, 192, 2016 link.

M. Jacquelin, L. Lin and C. Yang, PSelInv ? A distributed memory parallel algorithm for selected inversion : the symmetric case, ACM Trans. Math. Software 43, 21, 2016 link.

L. Lin, A. Garcia, G. Huhs and C. Yang, SIESTA-PEXSI: Massively parallel method for efficient and accurate ab initio materials simulation without matrix diagonalization, J. Phys. Condens. Matter 26, 305503, 2014 link.

L. Lin, M. Chen, C. Yang and L. He, Accelerating atomic orbital-based electronic structure calculation via pole expansion and elected inversion, J. Phys. Condens. Matter 25, 295501, 2013 link.

L. Lin, C. Yang, J. Meza, J. Lu, L. Ying and W. E, SelInv – An algorithm for selected inversion of a sparse symmetric matrix, ACM Trans. Math. Software 37, 40, 2011 link.

L. Lin, C. Yang, J. Lu, L. Ying and W. E, A Fast Parallel algorithm for selected inversion of structured sparse matrices with application to 2D electronic structure calculations, SIAM J. Sci. Comput. 33, 1329, 2011 link.

L. Lin, J. Lu, L. Ying, R. Car and W. E, Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems, Commun. Math. Sci. 7, 755, 2009 link.

L. Lin, J. Lu, L. Ying and W. E, Pole-based approximation of the Fermi-Dirac function, Chin. Ann. Math. 30B, 729, 2009 link.

Some references on applications of PEXSI

W. Hu, L. Lin, C. Yang, J. Dai and J. Yang, Edge-modified phosphorene nanoflake heterojunctions as highly efficient solar cells, Nano Lett. 16 1675, 2016

W. Hu, L. Lin and C. Yang, DGDFT: A massively parallel method for large scale density functional theory calculations, J. Chem. Phys. 143, 124110, 2015

W. Hu, L. Lin and C. Yang, Edge reconstruction in armchair phosphorene nanoribbons revealed by discontinuous Galerkin density functional theory, Phys. Chem. Chem. Phys. 17, 31397, 2015

W. Hu, L. Lin, C. Yang and J. Yang, Electronic structure of large-scale graphene nanoflakes, J. Chem. Phys. 141, 214704, 2014

1.5 Developer's documentation

This document is generated with Sphinx. For more detailed API routines in C/C++/FORTRAN see the developer's documentation generated by doxygen. To obtain this document, type *doxygen* under the pexsi directory, and the document will appear in the *developerdoc* directory.

1.6 PEXSI version history

- **v1.0.3 (7/20/2018)**
  - Bug fix: consistency problem in the data type of fortran examples (contributed by Calvin Anderson).
- v1.0.2 (7/17/2018)
  - Bug fix: When H and S are Hermitian matrices, return the proper density matrix and energy density matrix from CalculateFermiOperatorComplex, instead of its transpose. This is done by transposing the e.g. density matrix due to the Hermitian property. (contributed by Victor Yu)
  - Clarify the documentation for the treatment of selected elements of non-symmetric matrices.
  - Fix the naming of SRealMat and SComplexMat in ppexsi.cpp.
- **v1.0.1 (6/20/2018)**
  - Bug fix: initialization error in driver_fermi_complex, and uninitialized variables in CalculateFermiOperatorComplex
- **v1.0 (10/22/2017)**
  - Introduce PPEXSIDFTDriver2. This reduces the number of user-defined parameters, and improves the robustness over PPEXSIDFTDriver.
  - Compatible with the ELSI software package.
  - Migrate from doxygen to sphinx for documentation. The original doxygen format is still kept for the purpose of developers.
  - symPACK replaces SuperLU_DIST as the default solver for factorizing symmetric matrices. SuperLU_DIST is still the default solver for factorizing unsymmetric matrices. Currently supported version of SuperLU_DIST is v5.1.3.
  - PT-Scotch replaces ParMETIS as the default matrix ordering package. ParMETIS is still supported. Currently supported version of PT-Scotch is v6.0
  - Support Moussa's optimization based pole expansion.

    Moussa, J., Minimax rational approximation of the Fermi-Dirac distribution, J. Chem. Phys. 145, 164108 (2016)
  - Pole expansion given by src/getPole.cpp generated by a utility file. This allows types of pole expansions other than discretization of the contour integral to be implemented in the same fashion.
  - Compatible with spin-polarized and k-point parallelized calculations.
- **v0.10.1 (11/8/2016)**
  - Bug fix: matrix pattern for nonzero overlap matrices and missing option in fortran interface (contributed by Victor Yu)
- v0.10.0 (11/6/2016)

  - Combine LoadRealSymmetricMatrix / LoadRealUnsymmetricMatrix into one single function LoadRealMatrix. Similar change for LoadComplexMatrix. The driver routines and output are updated as well.
  - Updated makefile (contributed by Patrick Seewald)
  - Compatible with SuperLU_DIST_v5.1.2
  - Replace the debugging with PushCallStack / PopCallStack debugging by Google's coredumper.
  - A number of new example driver rouintes in examples/ and fortran/
  - Experimental feature: Add CalculateFermiOperatorComplex function. The implementation corresponds to CalculateFermiOperatorReal, but is applicable to the case when H and S are complex Hermitian matrices. This feature will facilitates the future integration with the Electronic Structure Infrastructure (ELSI) project.
  - Experimental feature: integration with symPACK for LDLT factorization.
  - Bug fix: Initialization variable pstat in interface with SuperLU_DIST
  - Bug fix: Add (void*) in MPI_Allgather of sparseA.nnzLocal in utility_impl.hpp.

- **v0.9.2 (2/29/2016)**
  - Add support for SuperLU_DIST v4.3. Starting from v0.9.2, the SuperLU_DIST v3.3 version is **NO LONGER SUPPORTED**.
  - Change the compile / installation to the more standard make / make install commands.
  - Add pole expansion C/FORTRAN interfaces that can be called separately.
  - Bug fix: remove a const attribute in CSCToCSR since it is modified by MPI. Add (void*) to MPI_Allgather for some compilers.
  - Bug fix: Mathjax is upgraded to v2.6 to support chrome rendering.
  - Add DFTDriver2 which allows only one PEXSI iteration per SCF iteration. This requires a careful setup of the inertia counting procedure.
  - In DFTDriver2, the muMinInertia and muMaxInertia are updated to avoid the true chemical potential to be at the edge of an interval.

- **v0.9.0 (07/15/2015)**
  - Add parallel selected inversion (PSelInv) for asymmetric matrices. The asymmetric matrix can be either structurally symmetric or fully asymmetric.
  - Add the example routines and fortran interfaces for asymmetric selected inversion.
  - Simplify the interface for installation.
  - (Contributed by Patrick Seewald) Bug fix: output string for SharedWrite utility routine.

- **v0.8.0 (05/11/2015)**
  - Improve the data communication pattern for PSelInv. The parallel scalability of PSelInv is much improved when more than 1000 processors are used. The variation of running time among different instances is also reduced.

    For more details of the improvement see

    M. Jacquelin, L. Lin, N. Wichmann and C. Yang, Enhancing the scalability and load balancing of the parallel selected inversion algorithm via tree-based asynchronous communication, submitted [arXiv]
  - Templated implementation of a number of classes including SuperLUMatrix.

  - Update the structure of the include/ folder to avoid conflict when PEXSI is included in other software packages.
  - Update the configuration files. Remove the out-of-date profile options.
  - Bug fix: MPI communicator in f_driver_ksdft.f90.

- v0.7.3 (11/27/2014) - Multiple patches suggested by Alberto Garcia.
  - Fix a bug in the "lateral expansion" for locating the bracket for the chemical potential.
  - Search for band edges of the chemical potential, which serve both for metals and for systems with a gap.
  - Add a paramter (mu0 in in PPEXSIOptions) to provide the starting guess of chemical potential. This can be used for the case in which the PEXSI solver is invoked directly, without an inertia-counting phase.
  - Update the example drivers accordingly to these bug fixes.
- v0.7.2 (08/27/2014) - Bug fix: Two temporary variables were not initialized during the computation of the number of electrons and its derivatives. - Add test matrices to the fortran/ folder as well. - Update the configuration files.
- v0.7.1 (07/01/2014) - Bug fix: PPEXSIPlanInitialize specifics the input according to mpirank instead of outputFileIndex. - Bug fix: PPEXSIPlanFinalize gives floating point error due to the double deallocation of SuperLU_Grid.
- v0.7.0 (05/24/2014) - Use PPEXSIPlan to coordinate the computation, and allows the code to be used for C/C++/FORTRAN. - Templated implementation and support for both real and complex arithmetic. - New interface routines for FORTRAN based on ISO_C_BINDING (FORTRAN 2003 and later). - Basic interface for KSDFT calculation, with a small number of input parameters and built-in heuristic strategies. - Expert interface for KSDFT calculation, providing full-control of the heuristics. - Symbolic factorization can be reused for multiple calculations. - Enhanced error estimate for the pole expansion using energy as a guidance.
- v0.6.0 (03/11/2014) - Version integrated with the SIESTA package for Kohn-Sham density functional theory (KSDFT) calculation. - Parallel selected inversion for complex symmetric matrices. - Estimate the density of state profile via inertia counting. - Compute the density of states and local density of states.

CHAPTER 2

Download

v1.0.3:

- Code: pexsi_v1.0.3.tar.gz.

Older version of PEXSI, as well as snapshots of documentation for major releases:

NOTE: Due to insufficient manpower (as there will never be in science!), older version of PEXSI is no longer supported

v1.0.2:

- Code: pexsi_v1.0.2.tar.gz.

v1.0.1:

- Code: pexsi_v1.0.1.tar.gz.

v1.0:

- Code: pexsi_v1.0.tar.gz.

v0.10.2:

- Code: pexsi_v0.10.2.tar.gz.
- Documentation: doc_v0.10.2.

v0.9.2:

- Code: pexsi_v0.9.2.tar.gz.
- Documentation: doc_v0.9.2.

v0.9.0:

- Code: pexsi_v0.9.0.tar.gz.
- Documentation: doc_v0.9.0.

v0.8.0:

- Code: pexsi_v0.8.0.tar.gz.

- Documentation: doc_v0.8.0.

v0.7.3: - Code: pexsi_v0.7.3.tar.gz. - Documentation: doc_v0.7.3.

v0.7.2:

- Code: pexsi_v0.7.2.tar.gz.

v0.7.1:

- Code: pexsi_v0.7.2.tar.gz.
- Documentation: doc_v0.7.1.

v0.6.0:

- Code: pexsi_v0.6.0.tar.gz.
- Documentation: doc_v0.6.0.

CHAPTER 3

Installation

3.1 Dependencies

PEXSI requires an external parallel LU factorization or LDL^T factorization routine, and an external parallel matrix reordering routine to reduce the fill-in of the factorization routine.

Starting from v1.0, PEXSI requires both symPACK and SuperLU_DIST. symPACK is the default option for the LDL^T factorization of symmetric matrices, and use SuperLU_DIST as the default option for the LU factorization of unsymmetric matrices. SuperLU_DIST can also be used for symmetric matrices, by means of treating the matrix as a general matrix but use symmetric reordering.

Starting from v1.0, PEXSI uses the PT-Scotch as the default package for matrix reordering. The ParMETIS package can also be used.

The installation procedure and dependencies of every version of the PEXSI package may be slightly different. Please follow the documentation of the version of the PEXSI package you are working with. (provided in the *Download Page*)

3.2 Build PT-Scotch

PT-Scotch can be downloaded from (latest version 6.0.0) https://gforge.inria.fr/frs/download.php/31831/scotch_6.0.0.tar.gz

**PT-Scotch 6.0.5 seems to be incompatible with PEXSI. For the moment please use 6.0.0 (contributed by Victor Yu, 6/20/2018) **

Follow the installation step to install PT-Scotch. **In INSTALL.TXT, pay special attention to the following sections in order to compile PT-Scotch correctly.**

2.3) Integer size issues

2.5) Threads issues

PT-Scotch is also METIS-Compatible. See the following section in INSTALL.TXT for more information.

2.9) MeTiS compatibility library

In *src/* directory, you need

```
make ptscotch
```

to compile PT-Scotch.

NOTE: Just typing *make* will generate the Scotch library but not PT-Scotch. Then all libraries will be given in *lib/* directory.**

3.3 Build symPACK

symPACK is a sparse symmetric matrix direct linear solver. More information can be found at http://www.sympack.org/.

To use symPACK, first, download the package as follows

```
git clone https://github.com/symPACK/symPACK.git  /path/to/sympack
```

Several environment variables can be set before configuring the build:

```
`SCOTCH_DIR` = Installation directory for **SCOTCH** and **PT-SCOTCH**
```

Then, create a build directory, enter that directory and type:

```
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/path/to/install/sympack
...OPTIONS... /path/to/sympack
```

The *...OPTIONS...* can be one of the following:

```
* `-DENABLE_METIS=ON|OFF`   to make **MeTiS** ordering available in **symPACK**␣
↪(`METIS_DIR` must be set in the environment)

* `-DENABLE_PARMETIS=ON|OFF`   to make **ParMETIS** ordering available in **symPACK**␣
↪(`PARMETIS_DIR` must be set in the environment, `METIS_DIR` is required as well)

* `-DENABLE_SCOTCH=ON|OFF`   to make **SCOTCH** / **PT-SCOTCH** orderings available␣
↪in **symPACK** (`SCOTCH_DIR` must be set in the environment)
```

Some platforms have preconfigured toolchain files which can be used by adding the following option to the *cmake* command:

```
-DCMAKE_TOOLCHAIN_FILE=/path/to/sympack/toolchains/edison.cmake
(To build on NERSC Edison machine for instance)
```

A sample toolchain file can be found in */path/to/sympack/toolchains/build_config.cmake* and customized for the target platform.

The *cmake* command will configure the build process, which can now start by typing:

```
make
make install
```

Additionally, a standalone driver for **symPACK** can be built by typing *make examples*

Note Since cmake also compiles UPCxx and GASNET, the compilation time may be long especially on certain clusters.

3.4 Build SuperLU_DIST

Download SuperLU_DIST (latest version 5.2.1) from

http://crd-legacy.lbl.gov/~xiaoye/SuperLU/superlu_dist_5.2.1.tar.gz

Follow the installation step to install SuperLU_DIST.

Our experience shows that on some machines it may be better to build SuperLU_DIST with -O2 option than the more aggresive optimization options provided by vendors.

- In SuperLU_DIST v5.1.3, some functions conflict when both real and complex arithmetic factorization is needed. This can be temporarily solved by adding *-Wl,–allow-multiple-definition* in the linking option.
- In SuperLU_DIST v5.1.3, there could be some excessive outputs. This can be removed by going to the SRC/ directory of superlu, and comment out the line starting with *printf(".. dQuery_Space* in dmemory_dist.c. Do the same thing for the line starting with *printf(".. zQuery_Space..)* in zmemory_dist.c.
- Please note that the number of processors for symbolic factorization cannot be too large when PARMETIS is used together with SuperLU. The exact number of processors for symbolic factorization is unfortunately a **magic parameter**. See *FAQ page*.

3.5 (Optional) Build ParMETIS

Download ParMETIS (latest version 4.0.3) from

http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/parmetis-4.0.3.tar.gz

Follow the installation step to install ParMETIS.

After untar the ParMETIS package, in Install.txt

> Edit the file metis/include/metis.h and specify the width (32 or 64 bits) of the elementary data type used in ParMetis (and METIS). This is controled by the IDXTYPEWIDTH constant.
>
> For now, on a 32 bit architecture you can only specify a width of 32, whereas for a 64 bit architecture you can specify a width of either 32 or 64 bits.

In our experience for most cases, the following setup work fine.:

```
#define IDXTYPEWIDTH 32
```

3.6 Build PEXSI

3.7 Edit make.inc

Configuration of PEXSI is controlled by a single *make.inc* file. Examples of the *make.inc* file are given under the *config/* directory.

Find *make.inc* with the most similar architecture, and copy to the main PEXSI directory (using Edison for example, the latest Intel computer at NERSC, a CRAY X30 machine). *${PEXSI_DIR}* stands for the main directory of PEXSI.

```
cd ${PEXSI_DIR}
cp config/make.inc.CRAY_XC30.intel make.inc
```

Edit the variables in make.inc.

```
PEXSI_DIR     = Main directory for PEXSI
DSUPERLU_DIR  = Main directory for SuperLU_DIST
PARMETIS_DIR  = Main directory for ParMETIS
PTSCOTCH_DIR  = Main directory for PT-Scotch
```

Edit the compiler options, for instance

```
CC            = cc
CXX           = CC
FC            = ftn
LOADER        = CC
```

The *USE_SYMPACK* option can be set to use the symPACK solver in PEXSI. It is set to 0 by default. When set to 1, the *SYMPACK_DIR* variable must be pointing to symPACK's installation directory.

Note

- Starting from PEXSI v0.8.0, *-std=c++11* is required in *CXXFLAGS*.
- Starting from PEXSI v0.9.2, *-std=c99* is required in *CFLAGS* to be compatible with SuperLU_DIST starting from v4.3.
- For **FORTRAN** users, *CPP_LIB=-lstdc++ -lmpi -lmpi_cxx* is often needed. Check this if there is link error.
- PEXSI can be compiled using *debug* or *release* mode in by the variable *COMPILE_MODE* in *make.inc*. This variable mainly controls the compiling flag *-DRELEASE*. The *debug* mode introduces tracing of call stacks at all levels of functions, and may significantly slow down the code. For production runs, use *release* mode.
- The *USE_PROFILE* option is for internal test purpose. Usually set this to 0.

3.8 Build the PEXSI library

The installation procedure and dependencies of every version of the PEXSI package may be different. Please follow the documentation of the version of the PEXSI package you are working with (provided in the *Download Page*)

If make.inc is configured correctly,:

```
make
make install
```

Should build the PEXSI library under the *build* directory ready to be used in an external package. If the FORTRAN interface is needed, type:

```
make finstall
```

If examples are needed (not necessary if you use PEXSI in an external package), type

```
make examples
```

which will generate C examples in *examples/* directory and FORTRAN examples in *fortran/* directory, respectively.:

```
make all
```

will make the library and the examples.

For more information on the examples, see *Tutorial Page*.

3.9 Tests

After example files are compiled, go to the *examples/* directory, and:

```
examples$ mpirun -n 1 ./driver_pselinv_complex_(suffix)
```

should return the diagonal of the matrix $(A+iI)^{-1}$ saved on the 0-th processor, where A is the five-point discretization of a Laplacian operator on a 2D domain. The result can be compared with *examples/driver_pselinv_complex.out* to check the correctness of the result. For more examples see *Tutorial Page*.

CHAPTER 4

Tutorial

4.1 Using plans and generating log files

PEXSI is written in C++, and the subroutines cannot directly interface with other programming languages such as C or FORTRAN. To solve this problem, the PEXSI internal data structure is handled using a datatype *PPEXSIPlan*. The idea and the usage of PPEXSIPlan is similar to *fftw_plan* in the FFTW (http://www.fftw.org/~fftw/fftw3_doc/Using-Plans.html) package.

In PEXSI, a matrix (or more accurately, its inverse) is generally referred to as a "pole". The factorization and selected inversion procedure for a pole is computed in parallel using *numProcRow * numProcCol* processors.

When only selected inversion (PSelInv) is used, it is recommended to set the *mpisize* of the communicator *comm* to be just *numProcRow * numProcCol*.

When PEXSI is used to evaluate a large number of inverse matrices such as in the electronic structure calculation, it is best to set *mpisize* to be *numPole*numProcRow*numProcCol*, where *numPole* is the number of poles can be processed in parallel.

Starting from v1.0, when *PPEXSIDFTDriver2* is used, it is best set *mpisize* to be *numPoint*numPole*numProcRow*numProcCol*, where *numPoint* is the number of PEXSI evaluations that can be performed in parallel. When *mpisize* < *numPoint*numPole*numProcRow*numProcCol*, *PPEXSIDFTDriver2* will first parallel over the *numProcRow*numProcCol* and *numPoint*.

The output information is controlled by the *outputFileIndex* variable, which is a local variable for each processor. For instance, if this index is 1, then the corresponding processor will output to the file *logPEXSI1*. If outputFileIndex is negative, then this processor does NOT output logPEXSI files.

Note

- Each processor must output to a **different** file if outputFileIndex is non-negative.
- When many processors are used, it is **not recommended** for all processors to output the log files. This is because the IO takes time and can be the bottleneck on many architecture. A good practice is to let the master processor output information (generating *logPEXSI0*) or to let the master processor of each pole to output the information.

4.2 Parallel selected inversion for a real symmetric matrix

The parallel selected inversion routine for a real symmetric matrix can be used as follows. This assumes that the mpisize of *MPI_COMM_WORLD* is *nprow * npcol*.

```
#include  "c_pexsi_interface.h"
...
{
  /* Setup the input matrix in distributed compressed sparse column (CSC) format */
  ...;
  /* Initialize PEXSI.
   * PPEXSIPlan is a handle communicating with the C++ internal data structure */
  PPEXSIPlan   plan;

  plan = PPEXSIPlanInitialize(
      MPI_COMM_WORLD,
      nprow,
      npcol,
      mpirank,
      &info );

  /* Tuning parameters of PEXSI. The default options is reasonable to
   * start, and the parameters in options can be changed.  */
  PPEXSIOptions  options;
  PPEXSISetDefaultOptions( &options );

  /* Load the matrix into the internal data structure */
  PPEXSILoadRealHSMatrix(
      plan,
      options,
      nrows,
      nnz,
      nnzLocal,
      numColLocal,
      colptrLocal,
      rowindLocal,
      AnzvalLocal,
      1,     // S is an identity matrix here
      NULL,  // S is an identity matrix here
      &info );

  /* Factorize the matrix symbolically */
  PPEXSISymbolicFactorizeRealSymmetricMatrix(
      plan,
      options,
      &info );

  /* Main routine for computing selected elements and save into AinvnzvalLocal */
  PPEXSISelInvRealSymmetricMatrix (
      plan,
      options,
      AnzvalLocal,
      AinvnzvalLocal,
      &info );

  ...;
  /* Post processing AinvnzvalLocal */
```

(continues on next page)

(continued from previous page)

```
  ...;

  PPEXSIPlanFinalize(
      plan,
      &info );
}
```

This routine computes the selected elements of the matrix $A^{-1} = (H - zS)^{-1}$ in parallel. The input matrix H follows the *Distribute CSC format*, defined through the variables *colptrLocal*,'rowindLocal', *HnzvalLocal*. The input matrix S can be omitted if it is an identity matrix and by setting *isSIdentity=1*. If S is not an identity matrix, the nonzero sparsity pattern is assumed to be the same as the nonzero sparsity pattern of H. Both *HnzvalLocal* and *SnzvalLocal* are double precision arrays.

An example is given in *examples/driver_pselinv_real.c*, which evaluates the selected elements of the inverse of the matrix saved in *examples/lap2dr.matrix*. See also *PEXSI Real Symmetric Matrix* for detailed information of its usage.

4.3 Parallel selected inversion for a complex symmetric matrix

The parallel selected inversion routine for a complex symmetric matrix is very similar to the real symmetric case. An example is given in *examples/driver_pselinv_complex.c*. See also *PEXSI Real Symmetric Matrix* for detailed information of its usage.

4.4 Parallel selected inversion for a real unsymmetric matrix

The parallel selected inversion routine for a real unsymmetric matrix can be used as follows. This assumes that the size of *MPI_COMM_WORLD* is *nprow * npcol*.

```
#include  "c_pexsi_interface.h"
...
{
  /* Setup the input matrix in distributed compressed sparse column (CSC) format */
  ...;
  /* Initialize PEXSI.
   * PPEXSIPlan is a handle communicating with the C++ internal data structure */
  PPEXSIPlan   plan;

  plan = PPEXSIPlanInitialize(
      MPI_COMM_WORLD,
      nprow,
      npcol,
      mpirank,
      &info );

  /* Tuning parameters of PEXSI. The default options is reasonable to
   * start, and the parameters in options can be changed.  */
  PPEXSIOptions  options;
  PPEXSISetDefaultOptions( &options );


  /* Load the matrix into the internal data structure */
  PPEXSILoadRealHSMatrix(
      plan,
```

(continues on next page)

(continued from previous page)

```
      options,
      nrows,
      nnz,
      nnzLocal,
      numColLocal,
      colptrLocal,
      rowindLocal,
      AnzvalLocal,
      1,     // S is an identity matrix here
      NULL,  // S is an identity matrix here
      &info );

  /* Factorize the matrix symbolically */
  PPEXSISymbolicFactorizeRealUnsymmetricMatrix(
      plan,
      options,
      &info );

  /* Main routine for computing selected elements and save into AinvnzvalLocal */
  PPEXSISelInvRealUnsymmetricMatrix (
      plan,
      options,
      AnzvalLocal,
      AinvnzvalLocal,
      &info );

  ...;
  /* Post processing AinvnzvalLocal */
  ...;

  PPEXSIPlanFinalize(
      plan,
      &info );
}
```

This routine computes the selected elements of the matrix $A^{-1} = (H - zS)^{-1}$ in parallel. The input matrix H follows the *Distribute CSC format*, defined through the variables *colptrLocal*, *rowindLocal*, *HnzvalLocal*. The input matrix S can be omitted if it is an identity matrix and by setting *isSIdentity=1*. If S is not an identity matrix, the nonzero sparsity pattern is assumed to be the same as the nonzero sparsity pattern of H. Both *HnzvalLocal* and *SnzvalLocal* are double precision arrays.

NOTE: As discussed in *selected elements*, for general non-symmetric matrices, the selected elements are the elements such that $\{A_{j,i} \neq 0\}$. This means that the matrix elements computed corresponding to the sparsity pattern of A^T. However, storing the matrix elements $\{A^{-1}_{i,j} | A_{j,i} \neq 0\}$ is practically cumbersome, especially in the context of distributed computing. Hence we choose to store the selected elements for A^{-T}, i.e. $\{A^{-T}_{i,j} | A_{i,j} \neq 0\}$. These are the values obtained from the non-symmetric version of PSelInv.

An example is given in *examples/driver_pselinv_real_unsym.c*, which evaluates the selected elements of the inverse of the matrix saved in *examples/big.unsym.matrix*. See also *PPEXSISelInvRealUnsymmetricMatrix* for detailed information of its usage.

4.5 Parallel selected inversion for a complex unsymmetric matrix

The parallel selected inversion routine for a complex unsymmetric matrix is very similar to the real unsymmetric case. An example is given in *examples/driver_pselinv_complex_unsym.c*. See also *PPEXSISelInvComplexUnsymmetricMa-*

trix for detailed information of its usage.

Similar to the case of real unsymmetric matrices, the values $\{A^{-T}_{i,j} | A_{i,j} \neq 0\}$ are the values obtained from the non-symmetric version of PSelInv.

4.6 Solving Kohn-Sham density functional theory: I

The simplest way to use PEXSI to solve Kohn-Sham density functional theory is to use the *PPEXSIDFTDriver2* routine. This routine uses built-in heuristics to obtain values of some parameters in PEXSI and provides a relatively small set of adjustable parameters for users to tune. This routine estimates the chemical potential self-consistently using a combined approach of inertia counting procedure and Newton's iteration through PEXSI. Some heuristic approach is also implemented in this routine for dynamic adjustment of the chemical potential and some stopping criterion.

An example routine is given in *examples/driver_ksdft.c*, which solves a fake DFT problem by taking a Hamiltonian matrix from *examples/lap2dr.matrix*.

Here is the structure of the code using the simple driver routine.

```
#include  "c_pexsi_interface.h"
...
{
  /* Setup the input matrix in distributed compressed sparse column (CSC) format */
  ...;
  /* Initialize PEXSI.
   * PPEXSIPlan is a handle communicating with the C++ internal data structure */
  PPEXSIPlan   plan;

  /* Set the outputFileIndex to be the pole index */
  /* The first processor for each pole outputs information */
  if( mpirank % (nprow * npcol) == 0 ){
    outputFileIndex = mpirank / (nprow * npcol);
  }
  else{
    outputFileIndex = -1;
  }

  plan = PPEXSIPlanInitialize(
      MPI_COMM_WORLD,
      nprow,
      npcol,
      outputFileIndex,
      &info );

  /* Tuning parameters of PEXSI. See PPEXSIOption for explanation of the
   * parameters */
  PPEXSIOptions  options;
  PPEXSISetDefaultOptions( &options );

  options.temperature  = 0.019; // 3000K
  options.muPEXSISafeGuard  = 0.2;
  options.numElectronPEXSITolerance = 0.001;
  /* method = 1: Contour integral ; method = 2: Moussa optimized poles; default is 2*/
  options.method = 2;
  /* typically 20-30 poles when using method = 2; 40-80 poles when method = 1 */
  options.numPole  = 20;
```

(continues on next page)

(continued from previous page)

```
  /* 2 points parallelization is set as default. */
  options.nPoints = 2;

  /* Load the matrix into the internal data structure */
  PPEXSILoadRealHSMatrix(
      plan,
      options,
      nrows,
      nnz,
      nnzLocal,
      numColLocal,
      colptrLocal,
      rowindLocal,
      HnzvalLocal,
      isSIdentity,
      SnzvalLocal,
      &info );

  /* Call the simple DFT driver2 using PEXSI */
  PPEXSIDFTDriver2(
      plan,
      options,
      numElectronExact,
      &muPEXSI,
      &numElectronPEXSI,
      &numTotalInertiaIter,
      &info );

  /* Retrieve the density matrix and other quantities from the plan */
  if(mpirank < nprow * npcol ) {

  PPEXSIRetrieveRealDM(
      plan,
      DMnzvalLocal,
      &totalEnergyH,
      &info );

  PPEXSIRetrieveRealEDM(
      plan,
      options,
      EDMnzvalLocal,
      &totalEnergyS,
      &info );
  }

  /* Clean up */
  PPEXSIPlanFinalize(
      plan,
      &info );
}
```

4.7 Solving Kohn-Sham density functional theory: II

In a DFT calculation, the information of the symbolic factorization can be reused for different (H, S) matrix pencil if the sparsity pattern does not change. An example routine is given in *examples/driver2_ksdft.c*, which solves a fake

DFT problem by taking a Hamiltonian matrix from *examples/lap2dr.matrix*.

Here is the structure of the code using the simple driver routine.

```
#include  "c_pexsi_interface.h"
...
{
  /* Perform DFT calculation as in the previous note */

  /* Update and obtain another set of H and S */

  /* Solve the problem once again without symbolic factorization */
  PPEXSILoadRealHSMatrix(
      plan,
      options,
      nrows,
      nnz,
      nnzLocal,
      numColLocal,
      colptrLocal,
      rowindLocal,
      HnzvalLocal,
      isSIdentity,
      SnzvalLocal,
      &info );

  // No need to perform symbolic factorization
  options.isSymbolicFactorize = 0;
  // Given a good guess of the chemical potential, no need to perform
  // inertia counting.
  options.isInertiaCount = 0;
  // Optional update mu0, muMin0, muMax0 in PPEXSIOptions

  PPEXSIDFTDriver2(
      plan,
      options,
      numElectronExact,
      &muPEXSI,
      &numElectronPEXSI,
      &numTotalInertiaIter,
      &info );

  /* Postprocessing */

}
```

Note: The built-in heuristics in *PPEXSIDFTDriver2* may not be optimal. It handles only one (H, S) pair at a time, and does not accept multiple matrix pairs $\{(H_l, S_l)\}$ as in the case of spin-orbit polarized calculations. For expert users and developers, it should be relatively easy to dig into the driver routine, and only use *PEXSI::PPEXSIData::SymbolicFactorizeRealSymmetricMatrix* (for symbolic factorization), *PEXSI::PPEXSIData::CalculateNegativeInertiaReal* (for inertia counting), and *PEXSI::PPEXSIData::CalculateFermiOperatorReal* (for one-shot PEXSI calculation) to improve heuristics and extend the functionalities.

4.8 Parallel computation of the Fermi operator for complex Hermitian matrices

The PPEXSIDFTDriver routine and PPEXSIDFTDriver2 routines are standalone routines for solving the density matrix with the capability of finding the chemical potential effectively. This can be used for Γ point calculation. For electronic structure calculations with k-points, multiple Hamiltonian operators may be needed to compute the number of electrons. The PEXSI package provides expert level routines for such purpose. See driver_fermi_complex.c for an example of the components.

CHAPTER 5

Further details

5.1 Basic

You should be able to skip most of this section if you only intend to use the driver routines, described in the *Tutorial Page* section and in *c_pexsi_interface.h*.

In such case for C/C++ programmers, include the interface file:

```
#include  "c_pexsi_interface.h"
```

For FORTRAN programmers, there is no interface routines such as *f_pexsi_interface.F90* yet. However, the FORTRAN routines can directly be used. See *Fortran Page* for more information.

The remaining section is mainly for C++ developers to have more detailed control of the PEXSI package. For C++ and usage beyond the driver routines, include the following file:

```
#include  "ppexsi.hpp"
```

For developers,

- For VI/VIM users, PEXSI seems to be best visualized with the following options concerning indentation.:

```
set tabstop=2
set shiftwidth=2
set expandtab
```

5.2 Data type

5.2.1 Basic data type

To enhance potential transplantability of the code, some basic data types are constants are defined in *environment.hpp*.

The basic data types *int* and *double* are redefined as *Int* and *Real*, in order to improve compatibility for different architecture especially on 64-bit machines (**not implemented yet**). The 64-bit long integer *int64_t* is also redefined as *LongInt*.

The complex arithmetic is treated using the standard C++ <*complex*> library. The complex data type is *std::complex*<*double*>, and is redefined as *Complex* in the implementation.:

```
typedef    int                   Int;
typedef    int64_t               LongInt;
typedef    double                Real;
typedef    std::complex<double>  Complex;
```

5.2.2 NumVec, NumMat, NumTns

The design of PEXSI tries to eliminate as much as possible the direct usage of pointers. This helps reducing memory leak. Commonly used pointers are wrapped into different classes.

The most commonly used are *PEXSI::NumVec* , *PEXSI::NumMat* , and *PEXSI::NumTns*, which are wrappers for 1D array (vector), 2D array (matrix) and 3D array (tensor), respectively. The arrays are always saved contiguously in memory as a 1D array. Column-major ordering is assumed for arrays of all dimensions, which makes the arrays directly compatible with BLAS/LAPACK libraries.

These wrapper classes can both actually own an array (by specifying *owndata_=true*), and just view an array (by specifying *owndata_=false*). Elements of arrays can be accessed directly as in FORTRAN convention, such as *A(i)* (NumVec), *A(i,j)* (NumMat), and *A(i,j,k)* (NumTns).

The underlying pointer can be accessed using the member function *Data()*.

Commonly used wrapper classes

NumVec:

```
typedef NumVec<bool>       BolNumVec;
typedef NumVec<Int>        IntNumVec;
typedef NumVec<Real>       DblNumVec;
typedef NumVec<Complex>    CpxNumVec;
```

NumMat:

```
typedef NumMat<bool>       BolNumMat;
typedef NumMat<Int>        IntNumMat;
typedef NumMat<Real>       DblNumMat;
typedef NumMat<Complex>    CpxNumMat;
```

NumTns:

```
typedef NumTns<bool>       BolNumTns;
typedef NumTns<Int>        IntNumTns;
typedef NumTns<Real>       DblNumTns;
typedef NumTns<Complex>    CpxNumTns;
```

5.2.3 Distributed compressed sparse column (CSC) format

We use the Compressed Sparse Column (CSC) format, a.k.a. the Compressed Column Storage (CCS) format for storing a sparse matrix. Click CSC link for the explanation of the format.

We adopt the following convention for distributed CSC format for saving a sparse matrix on distributed memory parallel machines. We assume the number of processor is P, the number of rows and columns of the matrix is N. The class for distributed memory CSC format matrix is *PEXSI::DistSparseMatrix*.

- *DistSparseMatrix* uses FORTRAN convention (1-based) indices for *colptrLocal* and *rowindLocal*, i.e. the first row and the first column indices are 1 instead of 0.
- *mpirank* follows the standard C convention, i.e. the *mpirank* for the first processor is 0.
- Each processor holds $\lfloor N/P \rfloor$ consequentive columns, with the exception that the last processor (*mpirank == P-1*) holds a all the remaining $N - (P-1)\lfloor N/P \rfloor$ columns. The first column holds by the i-th processor is $i\lfloor N/P \rfloor$. The number of columns on each local processor is usually denoted by *numColLocal*.
- *colptrLocal*, which is an integer array of type *IntNumVec* of dimension *numColLocal + 1*, stores the pointers to the nonzero row indices and nonzero values in *rowptrLocal* and *nzvalLocal*, respectively.
- *rowindLocal*, which is an integer array of type *IntNumVec* of dimension *nnzLocal*, stores the nonzero row indices in each column.
- *nzvalLocal*, which is an array of flexible type (usually *Real* or *Complex*) *NumVec* of dimension *nnzLocal*, stores the nonzero values in each column.

5.3 C/C++ interface

The main interface routines are given in *c_pexsi_interface.h*. The routines are callable from C/C++.

Note: C++ users also have the option of directly using the subroutines provided in *ppexsi.cpp*. The usage can be obtained from *interface.cpp*.

5.4 FORTRAN interface

The FORTRAN interface is based on the ISO_C_BINDING feature, which is available for FORTRAN 2003 or later. The usage of FORTRAN interface is very similar to the C interface as given in the *Tutorial Page* section.

In FORTRAN, the PPEXSIPlan data type is *c_intptr_t* (or equivalently *INTEGER*8*). The naming of the subroutines is similar to the C interface as in *c_pexsi_interface.h*. All FORTRAN interface routines are in *f_interface.f90*. For instance, the subroutine *PPEXSIPlanInitialize* (C/C++) corresponds to the subroutine *f_ppexsi_plan_initialize* (FORTRAN).

Example: Parallel selected inversion for a real symmetric matrix

```
integer(c_intptr_t)    :: plan
type(f_ppexsi_options) :: options

! Initialize PEXSI.
! PPEXSIPlan is a handle communicating with the C++ internal data structure

! Set the outputFileIndex to be the pole index.
! The first processor for each pole outputs information

if( mod( mpirank, nprow * npcol ) .eq. 0 ) then
  outputFileIndex = mpirank / (nprow * npcol);
else
  outputFileIndex = -1;
endif
```

(continues on next page)

(continued from previous page)

```
plan = f_ppexsi_plan_initialize(&
  MPI_COMM_WORLD,&
  nprow,&
  npcol,&
  outputFileIndex,&
  info )

! Tuning parameters of PEXSI. The default options is reasonable to
! start, and the parameters in options can be changed.
call f_ppexsi_set_default_options(&
  options )

! Load the matrix into the internal data structure
call f_ppexsi_load_real_hs_matrix(&
      plan,&
      options,&
      nrows,&
      nnz,&
      nnzLocal,&
      numColLocal,&
      colptrLocal,&
      rowindLocal,&
      HnzvalLocal,&
      1,&
      SnzvalLocal,&
      info )

! Factorize the matrix symbolically
call f_ppexsi_symbolic_factorize_real_symmetric_matrix(&
  plan,&
  options,&
  info)

! Main routine for computing selected elements and save into AinvnzvalLocal
call f_ppexsi_selinv_real_symmetric_matrix(& plan,&
  options,&
  AnzvalLocal,&
  AinvnzvalLocal,&
  info)

! Post processing step...

! Release the data saved in the plan
call f_ppexsi_plan_finalize( plan, info )
```

The examples of the FORTRAN interface can be found under *fortran/* directory, including

```
f_driver_pselinv_real.f90,
f_driver_pselinv_complex.f90,
f_driver_pselinv_real_unsym.f90,
f_driver_pselinv_complex_unsym.f90,
f_driver_ksdft.f90.
```

CHAPTER 6

Frequently asked questions

6.1 General questions

Q: How do I know whether PEXSI works for my application?

A: PEXSI may not necessarily be faster than the diagonalization method or other competitive methods. The simplest way to see whether PEXSI brings acceleration for your applications is to use PEXSI to compute the selected elements of the inverse for a typical matrix from your applications. See *Selected Inversion for Real Symmetric Matrix Page* and *driver_pselinv_real.c* for how to do this.

Q: Can I just use the selected inversion routine?

A: The parallel selected inversion (PSelInv) is a standalone routine. See *Selected Inversion for Real Symmetric Matrix Page* for an example.

Q: Does PEXSI accelerate dense matrix computation?

A: Not likely. The acceleration is based on the sparsity of the LU factor or the Cholesky factor. PEXSI should not be fast if the matrix is dense or nearly dense.

Q: Does PEXSI work for asymmetric matrices?

A: Yes! Starting from v0.9.0, PEXSI v0.9.0 supports PSelInv for asymmetric matrices, boh structurally symmetric and fully asymmetric. See *Selected Inversion for Real Unsymmetric Matrix Page* for example. This can already be used for Kohn-Sham DFT calculations for k-point etc with an "expert-mode". The full support similar to *PPEXSIDFTDriver* might be available in the future, but the interface might not be as straightforward.

Q: How to control the amount of output information and which processor outputs the log file?

See *PEXSI Plan section*. Also the *verbosity* option in *PPEXSIOptions* control the amount of information.

6.2 Installation

Q: The FORTRAN routine cannot compile.

A: For FORTRAN users, *CPP_LIB=-lstdc++ -lmpi -lmpi_cxx* is often needed. Check this first if there is link error.

6.3 Performance

Q: What if PEXSI is numerically unstable or inaccurate for my application?

A: If you are using the pole expansion, the expansion converges exponentially with respect to the number of poles. So first increase the number of poles until the error saturates. After this step, error only comes from the selected inversion phase. The selected inversion is in principle an exact method, but may possibly suffer from numerical instability issue due to the round off erros. This is possible due to the lack of dynamic pivoting strategies. If this problem persists, please contact us as in *Trouble Shooting Page*, with some description of you matrix and application.

Q: When using ParMETIS/PT-Scotch, I got segmentaiton fault in the factorization phase.

A: We have observed that when ParMETIS/PT-Scotch is used, the number of processors for symbolic factorization (*npSymbFact*) cannot be larger than a magic number depending on the matrix and the machine. This may to be related to the parallel symbolic factorization routine in SuperLU_DIST. If this problem happens, try to reduce *npSymbFact* to a small number (such as 4 or 16), or even try to use the sequential symbolic factorization if feasible. This should be more stable when symPACK is used for factorization.

CHAPTER 7

Troubleshooting

If you run into problems or would like to request additional features, please email to *linlin@math.berkeley.edu*.

If you are having problems building the package, please make sure to attach the *make.inc* file. An example of this file can be found in the *config* directory.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search