

ユーザズ・マニュアル (日本語版)

EigenExa

Version 2.13

1) EigenExa開発グループ
大規模並列数値計算技術研究チーム
2) 数値計算ライブラリ開発 WG
FS2020 アーキテクチャ開発チーム (2021年3月迄)

理化学研究所
計算科学研究センター^{*1*2}

発行 v2.6: 2020年 11月 1日
改訂 v2.9: 2021年 9月 24日
改訂 v2.10: 2021年 10月 17日
改訂 v2.11: 2021年 12月 1日
改訂 v2.13: 2024年 7月 23日

^{*1} 開発責任著者: 今村俊幸 (imamura.toshiyuki@riken.jp)

^{*2} 連絡先: EigenExa開発グループ (EigenExa@ml.riken.jp)

目次

第1章	はじめに	7
1.1	EigenExaとその開発経緯	7
1.2	現実装について	7
1.3	利用許諾/Copyright	9
第2章	利用の前に	13
2.1	EigenExaのインストールのために必要なソフトウェア	13
2.2	EigenExaの入手方法	13
2.3	コンパイルとインストール手順	14
第3章	クイックチュートリアル	17
3.1	基本呼び出し構造	17
3.2	コミュニケーター	18
3.3	カウンタとインデックスの取り扱い	19
3.4	ScaLAPACKとの連携	21
3.5	コンパイル時の注意	22
第4章	API	23
4.1	eigen_init	23
4.2	eigen_free	24
4.3	eigen_get_blacs_context	24
4.4	eigen_sx	25
4.5	eigen_s	26
4.6	eigen_h	27
4.7	eigen_get_version	28
4.8	eigen_show_version	28
4.9	eigen_get_matdims	29
4.10	eigen_memory_internal	29
4.11	eigen_get_comm	30
4.12	eigen_get_procs	31

4.13	<code>eigen_get_id</code>	31
4.14	<code>eigen_get_errinfo</code>	32
4.15	<code>eigen_loop_start</code>	32
4.16	<code>eigen_loop_end</code>	33
4.17	<code>eigen_loop_info</code>	34
4.18	<code>eigen_translate_l2g</code>	35
4.19	<code>eigen_translate_g2l</code>	36
4.20	<code>eigen_owner_node</code>	37
4.21	<code>eigen_owner_index</code>	38
4.22	<code>eigen_convert_ID_xy2w</code>	39
4.23	<code>eigen_convert_ID_w2xy</code>	39
4.24	<code>KMATH_EIGEN_GEV</code>	40
第5章	その他の注意事項	43
5.1	互換性の注意	43
5.2	他の言語との結合	43
5.3	エラー発生時の振舞い	43
5.4	バージョン1.xにおけるシェアード・ライブラリの扱い	44
5.5	既知の問題点と推奨回避策	44
5.5.1	Intelコンパイラ+Intel MPI利用時の数値再現性について	44
付録		
A章	アルゴリズムの概要	49
A.1	はじめに	49
A.2	様々なアプローチと関連プロジェクト	49
A.3	<code>eigen_s</code>	50
A.4	<code>eigen_sx</code>	51
A.5	<code>eigen_s</code> と <code>eigen_sx</code> の違い	52
A.6	おわりに	54
B章	Release notes	55
B.1	Version 2.13 (July 23, 2024)	55
B.2	Version 2.12 (January 10, 2023)	55
B.3	Version 2.11 (December 1, 2021)	55
B.4	Version 2.10 (October 17, 2021)	56
B.5	Version 2.9 (September 24, 2021)	56
B.6	Version 2.8 (August 20, 2021)	56
B.7	Version 2.7 (April 1, 2021)	57

B.8	Version 2.6 (November 1, 2020)	57
B.9	Version 2.5 (August 1, 2019)	57
B.10	Version 2.4b (August 20, 2018)	57
B.11	Version 2.3m (August 20, 2018)	57
B.12	Version 2.4p1 (May 25, 2017)	58
B.13	Version 2.4 (April 18, 2017)	58
B.14	Version 2.3k2 (April 12, 2017)	58
B.15	Version 2.3d (July 07, 2015)	58
B.16	Version 2.3c (April 23, 2015)	58
B.17	Version 2.3b (April 15, 2015)	59
B.18	Version 2.3a (April 14, 2015)	59
B.19	Version 2.3 (April 12, 2015)	59
B.20	Version 2.2d (March 20, 2015)	59
B.21	Version 2.2c (March 10, 2015)	59
B.22	Version 2.2b (October 30, 2014)	59
B.23	Version 2.2a (June 20, 2014)	60
B.24	Version 2.2 (April 10, 2014)	60
B.25	Version 2.1a (Feb 23, 2014)	60
B.26	Version 2.1 (Feb 10, 2014)	60
B.27	Version 2.0 (Dec 13, 2013)	61
B.28	Version 1.3a (Sep 21, 2013)	61
B.29	Version 1.3 (Sep 20, 2013)	61
B.30	Version 1.2 (Sep 17, 2013)	61
B.31	Version 1.1 (Aug 30, 2013)	61
B.32	Version 1.0 (Aug 1, 2013)	61
	謝辭	63
	参考文献	65

第1章

はじめに

1.1 EigenExaとその開発経緯

EigenExa(/aigen-éksə/)は高性能固有値ソルバである。EigenExaの歴史は、2002年に世界一位を記録した地球シミュレータ上で開発されたEigenES(正式名ではなくコードネーム)[1]まで遡る。EigenESの成果はSC2006でのゴードンベル賞にノミネートされ、その後大規模なPCクラスタ上での固有値ソルバーとして継続されている[2]。EigenExaの直前のライブラリ実装であるEigenK[3, 4]の開発が2008年頃に開始され、京[5, 6]の運用開始後の2013年8月に名称をEigenExaに改名し一般公開が開始された。EigenExaは将来登場するであろうポストペタスケール計算機システム(所謂「エクサ」($= 10^{15}$)または「エクストリーム」システム)でスケラブルに動作する固有値ライブラリ実現を目標に開発が継続されている。2014年から理研により開発が主導されているスーパーコンピュータ「富岳」[7]では、大規模並列数値計算技術研究チームとFS2020プロジェクトアーキテクチャ開発チームとの連携で「富岳」に向けたEigenExa開発と整備が進められ、EigenExaは「富岳」上の数値計算ライブラリの一部として整備される。

1.2 現実装について

これまでのリリース (バージョン2.3c, 2.4b, 2.6以降, 2.13まで)では、標準固有値問題と一般化固有値問題のいずれに対しても全ての固有対 (「固有値」と「対応する固有ベクトル」の組) を計算するという最もシンプルな機能を提供する。バージョン2.11からは実対称行列に加えて、念願のエルミート行列の対応を開始した。また、文献[2, 3, 4]で報告されているように、EigenK同様EigenExaもまた古典的なアルゴリズムと先進的なアルゴリズムの両者を採用して対角化に要する計算時間を削減している。「富岳」上での実装技術課題の一つは、「京」以降のスパコンではプロセッサ性能の向上に反して、ネットワーク性能の伸びが鈍化することによるハードウェアのアンバランス化である。実際、近年の小規模スパコンでも「高並列」化において通信が最大のボトルネックとなっており、「富岳」規模のスケールでは無視できないことが開発以前から認識されていた。

本リリース(バージョン2.13)では、ハウスホルダー三重対角化に通信回避技術を応用するとともに[15], 分割統治法のロードバランス改善に注意を払った新しいプロセスマッピング方式を採用している。これら技術は、高並列環境における性能改善に大きく貢献する。

EigenExaは開発当初から、MPI, OpenMP, 高性能BLAS, 更にSIMDベクトル化Fortran90コンパイラ技術など様々な並列プログラミング言語とライブラリを用いて開発されている。「京」それに続く「富岳」においても、次に挙げる項目が複数同時に機能して、高性能計算を実現することが期待される。

1. MPIによる分散メモリ型のノード間並列性
2. OpenMPによる共有メモリ型並列計算機もしくはマルチコアプロセッサでの並列性
3. ベンダにより高度に最適化されたBLASを用いた高い並列性
4. ベンダ提供の高性能コンパイラを用いたSIMDもしくは疎粒度並列性

EigenExaにはFortran90/95/2003以降のよい特徴も積極的に取り込まれている。当然、EigenExaのAPIはFortran77による実装ライブラリよりも柔軟であり、モジュールインターフェイスや省略可能引数によりユーザーフレンドリーなインターフェイスが提供される。データ分散は2次元サイクリック分割に限定されるが、プロセッサマップはほぼ任意形状が指定でき、ScaLAPACKが提供するデータ再分散関数を利用すれば既存の数値計算ライブラリとの親和性・整合性も保証されている。更に、実行性能を左右するブロックパラメータを利用者側から指定できる(省略も可能)など、性能向上のためのインターフェイスも提供している。

ライブラリ単体の並列性能の観点から見てみると、EigenKの通信オーバーヘッドを削減することで高い性能向上を達しており、多くの場合にEigenExaはEigenKやScaLAPACKなどの最高水準の数値計算ライブラリより高性能であることが確認されている[4]。

現在、EigenExaは「京」コンピュータ、スーパーコンピュータ「富岳」をはじめとして、その商用機であるFujitsu PRIMEHPC FXシリーズ、Intel x86系プロセッサやAMD64プロセッサを搭載する各種クラスタシステム、IBM Blue/Gene Qシステム、NECのベクトルコンピュータSXシリーズなど、多くのHPCプラットフォームで動作する。残念ながら、現時点ではGPUクラスタ上での動作バージョンは存在しない。EigenExaに関する学会報告は多数に上る[8, 9, 10, 11, 12, 13, 14, 15, 16, 17]。内部実装、アルゴリズム、性能評価に興味がある読者は必要に応じて参照されたい。

本ドキュメントはEigenExa version 2.13のユーザズ・マニュアルである^{*1}。導入開始から実際の使用までの内容を記している。特に、導入、コンパイル、クイックチュートリアル、APIリスト、EigenK, EigenExa version 2.3以前との互換性の注意が選択的に記載されている。EigenExa開発グループの全ての関係者は、本ドキュメントが多くの利用者に対して並列シミュレーションを効率よく走らせるための手助けになることを期待している。

^{*1} 本ユーザーガイドは英語版を優先的に更新しており、日本語版の更新作業は不定期、かつ、英語版の翻訳であるため対訳が適切でない可能性がある。最新英語版を合わせて参照していただくと幸いである

1.3 利用許諾/Copyright

EigenExa version 2.13は2条項BSDライセンス(The BSD 2-Clause License)に基づき利用を許諾する (ライブラリ内のLICENCE.txtに記載).

LICENCE.txt

```
Copyright (C) 2012- 2024 RIKEN.  
Copyright (C) 2011- 2012 Toshiyuki Imamura  
Graduate School of Informatics and Engineering,  
The University of Electro-Communications.  
Copyright (C) 2011- 2014 Japan Atomic Energy Agency.
```

Copyright notice is from here

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

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

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 THE COPYRIGHT HOLDERS OR CONTRIBUTORS 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.

なお、一部コードはLAPACK-3.4.2 and ScaLAPACK-2.0.2ならびにそれらの二次作成物を使用する。そのため、それらコードについては元のソフトウェアパッケージで許諾されているソフトウェアライセンスに従う。

[LAPACK 3.4.2]

Copyright (c) 1992-2011 The University of Tennessee and The University of Tennessee Research Foundation. All rights reserved.

Copyright (c) 2000-2011 The University of California Berkeley. All rights reserved.

Copyright (c) 2006-2012 The University of Colorado Denver. All rights reserved.

\$COPYRIGHT\$

Additional copyrights may follow

\$HEADER\$

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

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

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 THE COPYRIGHT OWNER OR CONTRIBUTORS 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.

[ScaLAPACK 2.0.2]

Copyright (c) 1992-2011 The University of Tennessee and The University of Tennessee Research Foundation. All rights reserved.

Copyright (c) 2000-2011 The University of California Berkeley. All rights reserved.

Copyright (c) 2006-2011 The University of Colorado Denver. All rights reserved.

\$COPYRIGHT\$

Additional copyrights may follow

\$HEADER\$

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

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

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 THE COPYRIGHT OWNER OR CONTRIBUTORS 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.

第2章

利用の前に

2.1 EigenExaのインストールのために必要なソフトウェア

EigenExaライブラリをコンパイルするためには幾つかのソフトウェアパッケージを準備しなければならない。BLAS, LAPACK, ScaLAPACK更にMPIはEigenExaのコンパイルの前にシステムにインストールされていなくてはならない。現在のところ, EigenExaは以下に示すライブラリでコンパイルできることが確認されている。

BLAS	Intel MKL, GotoBLAS, OpenBLAS, ATLAS Fujitsu SSL II, IBM ESSL, NEC MathKeisan
LAPACK	Version 3.4.0 以降
ScaLAPACK	Version 1.8.0 以降
MPI	MPICH2 version 1.5 以降, MPICH version 3.0.2 以降 OpenMPI version 1.6.4 以降, IntelMPIやFujitsuMPI, MPI/SX などのベンダーMPI
コンパイラ	Fortran2003もしくは2008互換コンパイラ (Fortran 95以降の機能 とFortran 2003の技術拡張を利用しているため。なお, 2.6版以前 はFortran 90/95 を推奨。)と C/C++コンパイラ GNU compiler 9.0 or later for x86, Intel compiler 19, or openAPI compiler DPC++を含む, Fujitsu Software Technical Computing Suite V4.0, その他ベンダー提供のコンパイラ

2.2 EigenExaの入手方法

EigenExaに関する全ての情報は次のURLから入手可能である. :

<https://www.r-ccs.riken.jp/labs/lpnctrtr/projects/eigenexa/>

tarball(拡張子はtgzもしくはtar.gz)の配布も上記サイトからされている。EigenExaに関

するその他の情報も提供されていく予定である。本稿では、バージョン2.13のtarball(拡張子はtgzもしくはtar.gz)であるEigenExa-2.13.tar.gzをダウンロードしてからの作業について以下で述べていく。

2.3 コンパイルとインストール手順

EigenExaライブラリのコンパイルには幾つかの手順が必要である。次のインストレーション手引きに従ってほしい。

■解凍と展開 まず、tarball(拡張子はtgzもしくはtar.gz)をワーキングディレクトリ上で展開する。そして、ディレクトリEigenExa-2.13に移動する。

```
% tar zxvf EigenExa-2.13.tar.gz
% cd EigenExa-2.13
```

■環境設定 次に、bootstrapスクリプト、configureスクリプトの順に実行し、システム環境に合わせたMakefile等を自動生成する。

```
% ./bootstrap
% ./configure
```

BLAS, LAPACK, ScaLAPACKはシステムによって異なるディレクトリに格納されており、複数のライブラリから適切なものを選択しなくてはならない場合もありうる。その場合には、以下の環境変数を適切に設定することで対応する。configureスクリプトのオプションは./configure --helpで確認することができる。

- FC MPI用fortranコンパイラコマンド (mpif90などを指定する)
- CC MPI用Cコンパイラコマンド (mpiccなどを指定する)
- LAPACK_PATH LAPACKなど数値ライブラリの格納ディレクトリ
- LAPACK_LIBS LAPACKなど数値ライブラリのリンク時の指定方法

クロスコンパイルのオプションは--host=**hostname**を渡すことで利用できる。「富岳」でARM64v8.2aのバイナリをビルドしたい場合は、ログインノード上で--host=login(ここで、(**hostname**)は任意文字列でよい)を付ければよい。なお、configureスクリプトが適切に動作しない場合は、スクリプト内部で使用するソフトウェアバージョンの違いによることも考えられる。次のように、cleanupスクリプトとbootstrapスクリプトを実行しconfigureスクリプトの再作成から始めてほしい。

```
% ./cleanup
% ./bootstrap
```

■メイク 3番目に, makeを実行する. その結果スタティック・ライブラリlibEigenExa.aならびにシェアドライブラリlibEigenExa.soが作成される.

```
% make
```

■インストール 最後に, ライブラリ自身であるlibEigenExa.a (シェアドライブラリの場合はlibEigenExa.so)と幾つかのfortranモジュール(eigen_libs_mod.mod, eigen_libs0_mod.mod, eigen_blacs_mod.mod, comm_mod.mod, and fs_libs_mod.mod)をインストールディレクトリにコピーし終了である.

```
% make prefix=(インストールするtopディレクトリ) install
```

手動でインストールする場合, 例えば /usr/local/libにインストールするときは, 次のようにする(行末のバックスラッシュ('\')は継続行を意味するもので, 実際の入力時には不要).

```
% cp cp libEigenExa.a libEigenExa.so eigen_libs_mod.mod \  
eigen_libs0_mod.mod \  
eigen_blacs_mod.mod comm_mod.mod fs_libs_mod.mod /usr/local/lib/
```

■一般化固有値計算ドライバルーチン 旧版version 2.4までは, 一般化固有値ドライバルーチンKMATH_EIGEN_GEVを別個にコンパイルする必要があったが, version 2.6からはEigenExaに取り込まれている. makeすると一般化固有値用のドライバモジュールKMATH_EIGEN_GEV.oがlibEigenExa.aならびにlibEigenExa.soに収納されており, プログラムリンク時に一般化固有値計算が可能となっている.

第3章

クイックチュートリアル

3.1 基本呼び出し構造

ワーキングディレクトリに移り, 'make benchmark' を実行すると標準ベンチマークコードが得られる. ソースコード中の 'main2.F' と 'Makefile' はコード作成に役立つはずである.

main2.fの核部分を取り出したものが次のようになる.

```
main2.f
use MPI
use eigen_libs_mod
...
call MPI_Init_thread( MPI_THREAD_MULTIPLE, i, ierr )
call eigen_init( )

N=10000; mtype=0

call eigen_get_matdims( N, nm, ny )
allocate ( A(nm,ny), Z(nm,ny), w(N) )
call mat_set( N, a, nm, mtype )
call eigen_sx( N, N, a, nm, w, z, nm, m_forward=32, m_backward=128 )
deallocate ( A, Z, w )
...
call eigen_free( )
call MPI_Finalize( ierr )
end
```

上のコードは骨格部分を示したのみであり実際の動作はしないが, 「初期化」 → 「配列確保」 → 「固有値計算」 → 「終了手続き」の流れを示すには十分なものである. ^{*1}

^{*1} MPI_THREADの選択は本質的ではない. 実際, EigenExaは任意のMPI_THREADモードで動作する.

3.2 コミュニケータ

前節の例(main2.F)では初期化関数eigen_init()を引数省略型呼び出しで実行している. eigen_init()には固有値計算を実施するグループをコミュニケータとしてcomm=XXXの形で指定できる. 複数のグループで同時に固有値計算を並列実行したいときにはMPI_Comm_split()等で作成されたコミュニケータを渡すことで並列計算可能となる. eigen_init()は集団操作であるので, commに属する全プロセスに同時に呼び出されなくてはならないという制約がある. 個々のプロセスごとに異なるコミュニケータを指定できるので, 固有値計算に参加しないプロセスはMPI_COMM_NULLをeigen_init()に指定して, 固有値計算ドライバeigen_sx()自身の呼び出しをスキップさせることができる. つまり, eigen_sx()の処理以外にeigen_sx()を含む様々な演算を同時実行することができる.

```
MPI_Comm_splitとMPI_COMM_NULL
color = 0; key = my_rank / 4
call MPI_Comm_split( MPI_COMM_WORLD, color, key, comm_new, ierr )
if ( my_rank < 16 ) then
    comm = comm_new
else if ( my_rank < 32 ) then
    comm = MPI_COMM_SELF
else
    comm = MPI_COMM_NULL
endif
call eigen_init( comm )
if ( comm /= MPI_COMM_NULL ) then
    call eigen_sx( .... )
else
    ... (他の処理. 仕様ではeigen_sxを呼び出しても直ちにreturnする)
endif
```

EigenExaではeigen_init()で指定されたコミュニケータに属するプロセスを2次元プロセスグリッドに配置して使用する. できるだけ通信量が削減できるように正方に近い形のプロセスグリッドを採用するよう設計されている. また, EigenExaは利用者の便宜を高めるという観点から, MPIで採用されている2次元カーテシアンをcommに指定することができるように開発されている. 原理的にはカーテシアンの形状が2次元であれば任意のプロセス配置に対してEigenExaを呼び出して計算することができるので, 上述の複数種類のコミュニケータとの組み合わせにより複雑な並列処理を実行することができる. なお, カ

ーテーションは基本的にプロセスグリッドがRow-majorになるため, `order='C'` 指定と矛盾するときはカーテシアンを優先して扱うことになっている. なお, EigenExaは歴史的経緯からデフォルトのプロセスグリッドはColumn-Majorを採用している.

`eigen_sx()`の呼び出しの直前に呼び出している`mat_set()`において行列データの生成を行っている. 行列データは指定された2次元プロセスグリッド上に2次元サイクリック分割のスタイルで分散されており, ローカル配列として各プロセスに格納されている. 各プロセスは行列の一部のデータのみを格納するため, 行列要素のアクセスをする場合にはグローバルインデックスとローカルインデックス間の変換ルールが必要である.

3.3 カウンタとインデックスの取り扱い

次のプログラムは`mat_set()`からの抜粋であり, グローバルカウンタによるループ構成のFrank行列生成プログラムをローカルカウンタによるループに変換した両者を対比として示している.

```
matset(before parallelization) —————  
!  
! Global loop program to compute a Frank matrix  
do i = 1, n  
  do j = 1, n  
    a(j, i) = DBLE(n+1-Max(n+1-i,n+1-j))  
  end do  
end do
```

↓↓↓↓↓↓

```

matset (after parallelization)

! Translated local loop program to compute a Frank matrix
use MPI
use eigen_libs_mod

call eigen_loop_info( j_2, j_3, 1, n, 'X' )
call eigen_loop_info( i_2, i_3, 1, n, 'Y' )

do i_1 = i_2, i_3
  i = eigen_translate_l2g( i_1, 'Y' )
  do j_1 = j_2, j_3
    j = eigen_translate_l2g( j_1, 'X' )
    a(j_1, i_1) = DBLE(n+1-Max(n+1-i,n+1-j))
  end do
end do

```

```

matset (after parallelization using 2.4 or prior)

! Translated local loop program to compute a Frank matrix
use MPI
use eigen_libs_mod

call eigen_get_procs( nnod, x_nnod, y_nnod )
call eigen_get_id   ( inod, x_inod, y_inod )

j_2 = eigen_loop_start( 1, x_nnod, x_inod )
j_3 = eigen_loop_end  ( n, x_nnod, x_inod )
i_2 = eigen_loop_start( 1, y_nnod, y_inod )
i_3 = eigen_loop_end  ( n, y_nnod, y_inod )

do i_1 = i_2, i_3
  i = eigen_translate_l2g( i_1, y_nnod, y_inod )
  do j_1 = j_2, j_3
    j = eigen_translate_l2g( j_1, x_nnod, x_inod )
    a(j_1, i_1) = DBLE(n+1-Max(n+1-i,n+1-j))
  end do
end do

```

ループ範囲の変換は`eigen_loop_start()`もしくは`eigen_loop_end()`を使用するが、ここでは両者を同時に使用する関数`eigen_loop_info()`を使って例示している。第三、第四引数にはグローバルカウンタ値、第五引数には分散方向を示す文字を指定する。(旧版での例では、第二、第三引数は分散方向を示すコミュニケータから派生されるプロセス数とプロセスIDを指定する)。本ドキュメントでは常に「行(第一インデックス)」→”x”、「列(第二インデックス)」→”y”の対応になっている(参加プロセス全体のコミュニケータの場合は、”x”や”y”の部分を無文字にしている。ここで、重大な注意として「EigenExaではプロセスIDを1から始まる整数で管理している」。そのため、問合せ関数`eigen_get_id()`によって取得したプロセスIDはMPIのランクと1だけずれている。MPIのランクが必要な場合はプロセスIDから1減じる必要がある。

上記プログラムは、ローカルなループカウンタ値から対応するグローバルカウンタ値に変換して使用する一例でもある。その変換には`eigen_translate_l2g()`を使用する。第二、第三引数は`eigen_loop_start()`などと同様に指定するとよい。逆に、グローバルカウンタ値をローカルカウンタ値に変換するには`eigen_translate_g2l()`を使用する。ただし、`eigen_translate_g2l()`はグローバルカウンタ値をループカウンタとして見たときに、当該カウンタ値のオーナープロセス(そのグローバルカウンタ値に対応するローカルカウンタ値をループ内に含むプロセス)となるプロセス上で対応するローカルカウンタ値を返すこととする(関数を呼び出したプロセスがオーナープロセスであるかとは関係なく同一値を返す)。

指定したグローバルループカウンタのオーナープロセスを知るには`eigen_owner_node()`や`eigen_owner_index()`を使用する。次のプログラムで例示するように、特定の行列要素(ここでは(j,i)の要素)を参照したりブロードキャストする際に利用するとよい。

```
Broadcast
! a(j,i) をブロードキャスト
i_1=eigen_owner_index( i, 'Y' )
j_1=eigen_owner_index( j, 'X' )
if ( i_1 > 0 .and . j_1 > 0 ) then
    v = a(j_1, i_1)
endif
i_0=eigen_owner_node( i, 'Y' )
j_0=eigen_owner_node( j, 'X' )
root=eigen_convert_ID_xy2w( j_0, i_0 )
call MPI_Bcast( v, 1, MPI_DOUBLE_PRECISION, root, TRD_COMM_WORLD, ierr )
```

3.4 ScaLAPACKとの連携

更に、ScaLAPACKと連携した上級者向けの計算を進めたいときは、補助関数

`eigen_get_blacs_context()`によりEigenExaとScaLAPACKとで互換なプロセスグリッドコンテキスト情報を取得すればよい。`mat_set()`関数の`mtype=2`の部分が良い例である(次プログラムは、行列ASの転置を行列Aに格納するPDTRAN()呼び出しの核部分である)。

```
pdtran
! Cooperation with ScaLAPACK
NPROW = x_nnod; NPCOL = y_nnod

ICTXT = eigen_get_blacs_context( )
CALL DESCINIT( DESCA, n, n, 1, 1, 0, 0, ICTXT, nm, INFO )

! A ← AST
CALL PDTRAN( n, n, 1D0, as, 1, 1, DESCA, 1D0, a, 1, 1, DESCA )
```

3.5 コンパイル時の注意

コンパイル時にはMPI用fortranコンパイラ(例えばmpif90)を使用するとともに、`eigen_libs_mod.mod`などモジュールへのアクセスが必要となるためパスの設定をしておく必要がある(多くの場合は-Iオプションである)。また、EigenExaライブラリをリンクするには、MPI, OpenMP, ScaLAPACK(バージョンが1.8以前の場合はBLACSも)などを同時にリンクする必要がある。GNUコンパイラベースのOSS MPIの場合は以下のようにする(ScaLAPACKやBLASまわりのライブラリパスや名前は環境によって異なる)。

```
% mpif90 -c a.f -fopenmp -I/usr/local/include -I/usr/local/lib
% mpif90 -o exe a.o -fopenmp -L/usr/local/lib -lEigenExa \
-lscalapack -llapack -lblas
```

第4章

API

本章では fortran モジュール 'eigen_libs_mod.mod' 中で public 属性を与えられた関数をリストアップする。始めの三ルーチンはメインドライバーであり、その他はユーティリティ関数である。必要なモジュール(基本的なユーザーの場合は eigen_libs_mod.mod)をUSE文で指定すれば、総称名関数と Optional 属性のついた引数の機能を使用できる。Optional 属性のついた引数(斜体で記述している)は、省略もできるし、Fortran のフォーマット形式である TERM='variable' or 'constant value' でも指定可能である。

浮動小数点数の変数は、倍精度と単精度を区別するためにreal(8)の記法を使用しているが(複素数も同様)、整数変数は原則として32ビット形式であり、integer(8)の明示的な指示がなければ、integer=integer(4)を想定する。

EigenExaはスレッドセーフではない。したがって、以下に示す関数でマルチスレッド利用に対する注意書きがないものについては OpenMP の OMP リージョンの外側でのみ使用しなくてはならない。

4.1 eigen_init

EigenExaの機能を初期化する。プロセスグリッドマッピングを引数 'comm' や 'order' を通じて指定することができる。本手続きは集団的であるため、EigenExaの計算に参加する全プロセスは本関数を同時に呼び出さなくてはならない。また、本関数でサブコミュニケータを最大5つ作成するため、大規模並列実行の際には多くの内部メモリと処理時間の消費が見込まれる。さらに、作成したサブコミュニケータでの通信性能のサンプリングを実施するため、相当のオーバーヘッドがあることを注意されたい。

本関数を呼び出した後は、次項の eigen_free() による終了手続きを済ませた後でなければ、再度 eigen_init() を呼び出してはいけない。例えば、基盤コミュニケータ comm を変更するときは、eigen_free() を一旦呼び出し、再度 eigen_init() でコミュニケータを変更しなくてはならない。

commは各プロセスグループごとに異なる値を指定でき、異なるプロセスグループが同時にドライバ関数(eigen_sx() もしくは eigen_s())を呼び出した際には、ドライバ

関数単位で並列動作する。comm が MPI_COMM_NULL の場合、ハンドラ `eigen_sx()` や `eigen_s()` が呼ばれた際には内部で何も行わず直ちにリターンする。インターコミュニケーターは使用できない。

EigenExaは内部でOpenMPによるマルチスレッド処理を行うが、全てのプロセスでスレッド数が一致していなくてはならない。本関数呼び出し時に、スレッド数が異なるプロセスの存在が検出された際にはプログラムがアボートする(`MPI_Abort()` が呼び出される)。また、`eigen_init()`を起動したときのデフォルトのスレッド数でのスレッド並列が唯一の推奨利用方法であり、中途 `omp_set_num_threads()` による動的なスレッド数変更はEigenExa内部に悪影響を及ぼす可能性がある。

```
subroutine eigen_init( comm, order )
```

1. integer, optional, intent(IN) :: comm = MPI_COMM_WORLD

基盤となるコミュニケーター。

comm が2次元カーテシアンの場合はプロセスグリッドマッピングが有効になる。

注意：省略時は MPI_COMM_WORLD

2. character(*), optional, intent(IN) :: order = 'C'

'R' (Row) もしくは 'C' (Column)

注意：省略時は 'C' として扱われる。グリッドメジャーがカーテシアンcommの指定と矛盾するときは適切なmajorが採用される。

4.2 eigen_free

EigenExaの機能を終了する。

```
subroutine eigen_free( flag )
```

1. integer, optional, intent(IN) :: flag = 0

タイマープリンタのフラグ

本引数は開発用途のためのものであり通常は指定しない。省略時は0である。

4.3 eigen_get_blacs_context

EigenExaで規定されるプロセスグリッド情報に対応するScaLAPACK(BLACS)のコンテキストを返す。EigenExaとScaLAPACK間でデータのやり取りをする際に必要となる。


```
integer function eigen_get_blacs_context( )
```

4.4 eigen_sx

EigenExaの主たるドライバルーチンである。五重対角行列への変換を経て固有対を計算する。本ドライバは集団操作であり、呼び出しを行うプロセスグループに属する全てのプロセスが呼び出しに参加しなくてはならない。

```
subroutine eigen_sx( n, nvec, a, lda, w, z, ldz, \
                    m_forward, m_backward, mode )
```

1. integer, intent(IN) :: n
行列・ベクトルの次元
2. integer, intent(IN) :: nvec
計算する固有値・対応する固有ベクトル(固有モード)の本数
正值の場合は、最小から指定された個数の固有モードを計算する。
0の場合は、全固有値を昇順に計算する。
一方、負値の場合は、最大から指定された個数の固有モードを計算する
(負値使用は現バージョンで未対応)。
3. real(8), intent(INOUT) :: a(1:lda,*)
対角化される対称行列(上三角行列要素のみ有効) で、サイズは
eigen_get_matdims を呼び出して得られた値以上でなければならない。
サブルーチン終了時には配列の内容は破壊される。
ただし、a(1, 1) にはflopsカウントが格納される。
4. integer, intent(IN) :: lda
配列aの整合寸法 (リーディングディメンジョン)
eigen_get_matdims を呼び出して得られた値以上でなければならない。
5. real(8), intent(OUT) :: w(1:n)
昇順の固有値
6. real(8), intent(OUT) :: z(1:ldz,*)
行列aの直交固有ベクトル
7. integer, intent(IN) :: ldz
配列zの整合寸法 (リーディングディメンジョン)
eigen_get_matdims を呼び出して得られた値以上でなければならない。
8. integer, optional, intent(IN) :: m_forward = 48
ハウスホルダー変換のブロック係数(偶数でなければいけない) 省略時は48。
9. integer, optional, intent(IN) :: m_backward = 128

ハウスホルダー逆変換のブロック係数. 省略時は128.

10. `character(*), optional, intent(IN) :: mode = 'A'`
 'A' : 全ての固有値と対応する固有ベクトル(default)
 'N' : 固有値のみ
 'X' : モードAに加えて固有値の精度改善を行う

4.5 eigen_s

EigenExaのドライバルーチンである. 集団操作であり, 呼び出したプロセスグループに属するすべてのプロセスが呼び出しに参加しなければならない.

```
subroutine eigen_s( n, nvec, a, lda, w, z, ldz, \
                   m_forward, m_backward, mode )
```

1. `integer, intent(IN) :: n`
 行列・ベクトルの次元
2. `integer, intent(IN) :: nvec`
 計算する固有値・対応する固有ベクトル(固有モード)の本数
 正值の場合は, 最小から指定された個数の固有モードを計算する.
 0の場合は, 全固有値を昇順に計算する.
 一方, 負値の場合は, 最大から指定された個数の固有モードを計算する
 (負値使用は現バージョンで未対応).
3. `real(8), intent(INOUT) :: a(1:lda,*)`
 対角化される対称行列 (上三角行列要素のみ有効), サイズは
 `eigen_get_matdims` を呼び出して得られた値以上でなければならない.
 サブルーチン終了時には配列の内容は破壊される.
 ただし, `a(1, 1)` にはflopsカウントが格納される.
4. `integer, intent(IN) :: lda`
 配列aの整合寸法 (リーディングディメンジョン)
 `eigen_get_matdims` を呼び出して得られた値以上でなければならない.
5. `real(8), intent(OUT) :: w(1:n)`
 昇順の固有値
6. `real(8), intent(OUT) :: z(1:ldz,*)`
 行列aの直交固有ベクトル
7. `integer, intent(IN) :: ldz`
 配列zの整合寸法 (リーディングディメンジョン)
 `eigen_get_matdims` を呼び出して得られた値以上でなければならない.

8. integer, optional, intent(IN) :: m_forward = 48
ハウスホルダー変換のブロック係数
9. integer, optional, intent(IN) :: m_backward = 128
ハウスホルダー逆変換のブロック係数
10. character(*), optional, intent(IN) :: mode = 'A'
'A' : 全ての固有値と対応する固有ベクトル(default)
'N' : 固有値のみ
'X' : モードAに加えて固有値の精度改善を行う

4.6 eigen_h

エルミート固有値問題を計算するためのEigenExaのドライバルーチンである。集団操作であり、呼び出したプロセスグループに属するすべてのプロセスが呼び出しに参加しなければならない。本ドライバルーチンはバージョン2.11から新たにサポートされている。

```
subroutine eigen_h( n, nvec, a, lda, w, z, ldz, \
                   m_forward, m_backward, mode )
```

1. integer, intent(IN) :: n
行列・ベクトルの次元
2. integer, intent(IN) :: nvec
計算する固有値・対応する固有ベクトル(固有モード)の本数
正值の場合は, 最小から指定された個数の固有モードを計算する。
0の場合は, 全固有値を昇順に計算する。
一方, 負値の場合は, 最大から指定された個数の固有モードを計算する
(負値使用は現バージョンで未対応)。
3. complex(8), intent(INOUT) :: a(1:lda,*)
対角化されるエルミート行列 (上三角行列要素のみ有効), サイズは
eigen_get_matdims を呼び出して得られた値以上でなければならない。
サブルーチン終了時には配列の内容は破壊される。
ただし, DBLE(a(1, 1)) にはflopsカウントが格納される。
4. integer, intent(IN) :: lda
配列aの整合寸法 (リーディングディメンジョン)
eigen_get_matdims を呼び出して得られた値以上でなければならない。
5. real(8), intent(OUT) :: w(1:n)
昇順の固有値
6. complex(8), intent(OUT) :: z(1:ldz,*)

行列aの直交固有ベクトル

7. integer, intent(IN) :: ldz
配列zの整合寸法 (リーディングディメンジョン)
eigen_get_matdims を呼び出して得られた値以上でなければならない.
8. integer, optional, intent(IN) :: m_forward = 48
ハウスホルダー変換のブロック係数
9. integer, optional, intent(IN) :: m_backward = 128
ハウスホルダー逆変換のブロック係数
10. character(*), optional, intent(IN) :: mode = 'A'
'A': 全ての固有値と対応する固有ベクトル(default)
'N': 固有値のみ
'X': モードAに加えて固有値の精度改善を行う

4.7 eigen_get_version

EigenExaのバージョン情報を返す. 引数versionの書式がversion 2.11から, 1桁ずつの参照から2桁ずつの表現方法に変更されている.

本サブルーチンはローカル操作である. (引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが, 引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる.

- ```
subroutine eigen_get_version(version, data, vcode)
```
1. integer, intent(OUT) :: version  
6桁のバージョン番号.  
上位から2桁毎にmajor version, minor version, patch levelを表す.
  2. character(\*), optional, intent(OUT) :: date  
リリース日.
  3. character(\*), optional, intent(OUT) :: vcode  
各バージョンに対応するコードネーム.

## 4.8 eigen\_show\_version

EigenExaのバージョン情報を標準出力(stdout)する.

```
subroutine eigen_show_version()
```

## 4.9 eigen\_get\_matdims

EigenExaで推奨する配列サイズを返す。ユーザーは本関数で取得した配列寸法(nx,ny)もしくはそれ以上の数値を使用してローカルな配列を動的に確保することが望ましい。行列全体は(CYCLIC,CYCLIC)分割されている。modeオプションにより配列サイズの使用メモリサイズに対する指定ができる(メモリ消費量はMinimal<LineAligned<Optimalの順)。本オプションはVersion 2.6以降で有効。

本サブルーチンはローカル操作である。(引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが、引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる。

```
subroutine eigen_get_matdims(n, nx, ny, mode)
```

```
1. integer, intent(IN) :: n
```

行列の次元

```
2. integer, intent(OUT) :: nx
```

2次元配列a(:, :), b(:, :)ならびにz(:, :)の整合寸法(リーディングディメンジョン)の下限值

```
3. integer, intent(OUT) :: ny
```

2次元配列a(:, :), b(:, :)ならびにz(:, :)の第二インデックスの下限值

```
4. character(*), optional, intent(IN) :: mode = '0'
```

配列の寸法の指定方法に対するオプション。

'M': Minimal, 最小限の寸法を返却

'L': LineAligned, キャッシュラインにアライメントされた寸法を返却

'0': Optimal, 内部でキャッシュスラッシングを回避する寸法を返却(default)

## 4.10 eigen\_memory\_internal

本サブルーチンはEigenExaが呼び出されている間に内部で動的に確保されるメモリサイズを返す。利用者は本関数の返却値を知り、メモリ不足に陥らないようにすべきである。バージョン2.3cより8バイト整数(integer(8))を返り値とする仕様変更がなされた。

返り値が負値(多くの場合-1)のときは行列サイズが大きすぎてEigenExa内部で使用する整数値(integer(4))がオーバーフローするおそれがあることを警告しており、注意が必

要である。

本サブルーチンはローカル操作である。(引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが、引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる。

```
integer(8) function eigen_memory_internal(n, lda, ldz, m1, m0)
```

1. integer, intent(IN) :: n  
行列の次元
2. integer, intent(IN) :: lda  
配列aの整合寸法 (リーディングディメンジョン)
3. integer, intent(IN) :: ldz  
配列zの整合寸法 (リーディングディメンジョン)
4. integer, intent(IN) :: m1  
ハウスホルダー変換のブロック係数(偶数でなければいけない)
5. integer, intent(IN) :: m2  
ハウスホルダー逆変換のブロック係数

#### 4.11 eigen\_get\_comm

eigen\_init()によって生成されたMPIコミュニケーターを返す。本サブルーチンはローカル操作である。(引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが、引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる。

```
subroutine eigen_get_comm(comm, x_comm, y_comm)
```

1. integer, intent(OUT) :: comm  
基盤となるコミュニケーター
2. integer, intent(OUT) :: x\_comm  
行コミュニケーター。行idが一致する全プロセス所属する。
3. integer, intent(OUT) :: y\_comm  
列コミュニケーター。列idが一致する全プロセスが所属する。

## 4.12 eigen\_get\_procs

eigen\_init()によって生成されたコミュニケータに関するプロセス数情報を返す. 本サブルーチンはローカル操作である. (引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが, 引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる.

```
subroutine eigen_get_procs(procs, x_procs, y_procs)
```

1. integer, intent(OUT) :: procs  
comm中のプロセス数
2. integer, intent(OUT) :: x\_procs  
x\_comm中のプロセス数
3. integer, intent(OUT) :: y\_procs  
y\_comm中のプロセス数

## 4.13 eigen\_get\_id

eigen\_init()によって生成されたコミュニケータに関するプロセスID情報を返す. ここでプロセスIDはMPIランクとは異なり1から開始する整数値で, MPIランク=プロセスID - 1の関係にある. 本サブルーチンはローカル操作である. (引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが, 引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる.

```
subroutine eigen_get_id(id, x_id, y_id)
```

1. integer, intent(OUT) :: id  
commで定義されたプロセスID
2. integer, intent(OUT) :: x\_id  
x\_commで定義されたプロセスID
3. integer, intent(OUT) :: y\_id  
y\_commで定義されたプロセスID

## 4.14 eigen\_get\_errinfo

内部エラー状態情報を返す。もしinfoが0であれば、エラーは起こっていない。非ゼロの場合は、内部で呼ばれている固有値ソルバールーチンで生じたエラー状態を示している(例えば、分割統治法など)。本関数は2.11版から利用可能になったものであり、引数や返却値の意味などについて今後仕様変更がなされることがある。本サブルーチンはローカル操作である。(引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが、引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる。

```
subroutine eigen_get_errinfo(info)
```

```
1. integer, intent(OUT) :: info
```

内部エラー状態情報

## 4.15 eigen\_loop\_start

指定されたグローバルループ開始値に対応するローカルなループ構造におけるループ開始値を返す。注意として、グローバルループ開始値は1以上でなくてはならない。 $1 \leq \text{inod} \leq \text{nnod}$ (pdir指定の場合はそのコミュニケータの参加プロセス数)でなくてはならない。pdirが適切な指定でない場合は、0を返す。

本関数は総称名インターフェイスを採用しており、引数に応じて適切な下位関数が呼び出される。

本関数はローカル操作である。(引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが、引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる。

```
integer function eigen_loop_start(istart, nnod, inod)
```

```
1. integer, intent(IN) :: istart
```

グローバルループ開始値

```
2. integer, intent(IN) :: nnod
```

プロセス数

```
3. integer, intent(IN) :: inod
```

プロセスID

```
integer function eigen_loop_start(istart, pdir, inod)
```



1. integer, intent(IN) :: istart  
グローバルループ開始値
2. character(\*), intent(IN) :: pdir  
システムで管理する基盤・列・行のコミュニケータを指す文字を指定  
'W' or 'T' : comm  
'X' or 'R' : x\_comm  
'Y' or 'C' : y\_comm
3. integer, intent(IN), optional :: inod  
プロセスID, 省力時は呼び出しプロセスのpdirに対応するID

例えば、次のような逐次ループをコミュニケータx\_commの中で並列化するとき、ローカルループの処理範囲を eigen\_loop\_startならびにeigen\_loop\_endで取得する。変換後のループ内で呼び出されている関数eigen\_translate\_l2gはローカルループカウンタが示すローカルインデックス値に対応するグローバルインデックス値を返すものである。

```
do i=J,K

enddo
```

↓↓↓↓↓↓

```
i_start=eigen_loop_start(J, 'X')
i_end =eigen_loop_end (K, 'X')
do i_local=i_start, i_end
 i=eigen_translate_l2g(i_local, 'X')

enddo
```

## 4.16 eigen\_loop\_end

指定されたグローバルループ終端値に対応するローカルなループ構造におけるループ終端値を返す。注意として、グローバルループ終端値は1以上でなくてはならない。 $1 \leq \text{inod} \leq \text{nnod}(\text{pdir}$ 指定の場合はそのコミュニケータの参加プロセス数)でなくてはならない。pdirが適切な指定でない場合は、-1を返す。

本関数は総称名インターフェイスを採用しており、引数に応じて適切な下位関数が呼び出される。

本関数はローカル操作である。(引数がfortranのポインタ渡しであるため)スレッドセー

フとは言えないが、引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる。

```
integer function eigen_loop_end(iend, nnod, inod)
```

1. integer, intent(IN) :: istart  
グローバルループ終端値
2. integer, intent(IN) :: nnod  
プロセス数
3. integer, intent(IN) :: inod  
プロセスID

```
integer function eigen_loop_end(iend, pdir, inod)
```

1. integer, intent(IN) :: iend  
グローバルループ終端値
2. character(\*), intent(IN) :: pdir  
システムで管理する基盤・列・行のコミュニケータを指す文字を指定  
'W' or 'T' : comm  
'X' or 'R' : x\_comm  
'Y' or 'C' : y\_comm
3. integer, intent(IN), optional :: inod  
プロセスID, 省力時は呼び出しプロセスのpdirに対応するID

## 4.17 eigen\_loop\_info

関数eigen\_loop\_startとeigen\_loop\_endを結合させたサブルーチンであり、開始値・終端値を同時に返す。仕様は両関数のものに従う。

```
subroutine eigen_loop_info(istart, iend, lstart, lend, nnod, inod)
```

1. integer, intent(IN) :: istart  
グローバルループ開始値
2. integer, intent(IN) :: iend  
グローバルループ終端値
3. integer, intent(OUT) :: lstart  
ローカルループ開始値
4. integer, intent(OUT) :: lend

ローカルループ終端値

5. integer, intent(IN) :: nnod

プロセス数

6. integer, intent(IN) :: inod

プロセスID

```
subroutine eigen_loop_info(istart, iend, lstart, lend, pdir, inod)
```

1. integer, intent(IN) :: istart

グローバルループ開始値

2. integer, intent(IN) :: iend

グローバルループ終端値

3. integer, intent(OUT) :: lstart

ローカルループ開始値

4. integer, intent(OUT) :: lend

ローカルループ終端値

5. character(\*), intent(IN) :: pdir

システムで管理する基盤・列・行のコミュニケータを指す文字を指定

'W' or 'T' : comm

'X' or 'R' : x\_comm

'Y' or 'C' : y\_comm

6. integer, intent(IN), optional :: inod

プロセスID, 省力時は呼び出しプロセスのpdirに対応するID

## 4.18 eigen\_translate\_l2g

ローカルカウンタが示すローカルインデックス値(1以上の値)に対応するグローバルインデックスを返す.  $1 \leq \text{inod} \leq \text{nnod}$  (pdir指定の場合はそのコミュニケータの参加プロセス数)でなくてはならない. pdirが適切な指定でない場合は, -1を返す.

本関数は総称名インターフェイスを採用しており, 引数に応じて適切な下位関数が呼び出される.

本関数はローカル操作である. (引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが, 引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる.

```
integer function eigen_translate_l2g(ictr, nnod, inod)
```

1. integer, intent(IN) :: ictr  
ローカルカウンタ
2. integer, intent(IN) :: nnod  
プロセス数
3. integer, intent(IN):: inod  
プロセスID

```
integer function eigen_translate_l2g(ictr, pdir, inod)
```

1. integer, intent(IN) :: ictr  
ローカルカウンタ
2. character(\*), intent(IN) :: pdir  
システムで管理する基盤・列・行のコミュニケータを指す文字を指定  
'W' or 'T' : comm  
'X' or 'R' : x\_comm  
'Y' or 'C' : y\_comm
3. integer, intent(IN), optional :: inod  
プロセスID, 省力時は呼び出しプロセスのpdirに対応するID

## 4.19 eigen\_translate\_g2l

グローバルカウンタが示すグローバルインデックス値(1以上の値)に対応するローカルインデックスを返す(ただし, 当該プロセスがオーナーであるとは限らない).  $1 \leq \text{inod} \leq \text{nnod}$  (pdir指定の場合はそのコミュニケータの参加プロセス数)でなくてはならない. pdirが適切な指定でない場合は, -1を返す.

本関数は総称名インターフェイスを採用しており, 引数に応じて適切な下位関数が呼び出される.

本関数はローカル操作である. (引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが, 引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる.

```
integer function eigen_translate_g2l(ictr, nnod, inod)
```

1. integer, intent(IN) :: ictr  
グローバルカウンタ
2. integer, intent(IN) :: nnod

プロセス数

3. integer, intent(IN) :: inod

プロセスID

```
integer function eigen_translate_g2l(ictr, pdir, inod)
```

1. integer, intent(IN) :: ictr

グローバルカウンタ

2. character(\*), intent(IN) :: pdir

システムで管理する基盤・列・行のコミュニケータを指す文字を指定

'W' or 'T' : comm

'X' or 'R' : x\_comm

'Y' or 'C' : y\_comm

3. integer, intent(IN), optional :: inod

プロセスID, 省力時は呼び出しプロセスのpdirに対応するID

## 4.20 eigen\_owner\_node

指定されたグローバルインデックス値(1以上の値)に対応するオーナープロセスのIDを返す.  $1 \leq \text{inod} \leq \text{nnod}(\text{pdir指定の場合はそのコミュニケータの参加プロセス数})$ でなくてはならない. pdirが適切な指定でない場合は, -1を返す.

本関数は総称名インターフェイスを採用しており, 引数に応じて適切な下位関数が呼び出される.

本関数はローカル操作である. (引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが, 引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる.

```
integer function eigen_owner_node(ictr, nnod, inod)
```

1. integer, intent(IN) :: ictr

グローバルカウンタ

2. integer, intent(IN) :: nnod

プロセス数

3. integer, intent(IN) :: inod

プロセスID

```
integer function eigen_owner_node(ictr, pdir, inod)
```

1. integer, intent(IN) :: ictr  
グローバルカウンタ
2. character(\*), intent(IN) :: pdir  
システムで管理する基盤・列・行のコミュニケータを指す文字を指定  
'W' or 'T' : comm  
'X' or 'R' : x\_comm  
'Y' or 'C' : y\_comm
3. integer, intent(IN), optional :: inod  
プロセスID, 省力時は呼び出しプロセスのpdirに対応するID

## 4.21 eigen\_owner\_index

当該プロセスが指定されたグローバルインデックス値(1以上の値)のオーナーの場合に、対応するローカルインデックスを返す。オーナーでない場合は  $-1$  を返す。 $1 \leq \text{inod} \leq \text{nnod}$ (pdir指定の場合はそのコミュニケータの参加プロセス数)でなくてはならない。pdirが適切な指定でない場合は、 $-1$ を返す。

本関数は総称名インターフェイスを採用しており、引数に応じて適切な下位関数が呼び出される。

本関数はローカル操作である。(引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが、引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる。

```
integer function eigen_index_node(ictr, nnod, inod)
```

1. integer, intent(IN) :: ictr  
グローバルカウンタ
2. integer, intent(IN) :: nnod  
プロセス数
3. integer, intent(IN) :: inod  
プロセスID

```
integer function eigen_index_node(ictr, pdir, inod)
```

1. integer, intent(IN) :: ictr  
グローバルカウンタ
2. character(\*), intent(IN) :: pdir

システムで管理する基盤・列・行のコミュニケータを指す文字を指定

'W' or 'T' : comm

'X' or 'R' : x\_comm

'Y' or 'C' : y\_comm

3. integer, intent(IN), optional :: inod

プロセスID, 省力時は呼び出しプロセスのpdirに対応するID

## 4.22 eigen\_convert\_ID\_xy2w

2次元プロセスIDを, グリッドメジャーに応じて基盤コミュニケータ上のプロセスIDに変換する. 入力されたプロセスIDや戻り値のプロセスIDが範囲内であるかの検査は行わない.

本関数はローカル操作である. (引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが, 引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる.

```
integer function eigen_convert_ID_xy2w(xinod, yinod)
```

1. integer, intent(IN) :: xinod

x\_commにおけるプロセスID

2. integer, intent(IN) :: yinod

y\_commにおけるプロセスID

## 4.23 eigen\_convert\_ID\_w2xy

基盤コミュニケータ上のプロセスIDを, グリッドメジャーに応じて2次元プロセスIDに変換する. 入力されたプロセスIDや戻り値のプロセスIDが範囲内であるかの検査は行わない.

本サブルーチンはローカル操作である. (引数がfortranのポインタ渡しであるため)スレッドセーフとは言えないが, 引数の適切な排他制御が行われていればマルチスレッド処理中に呼び出すことができる.

```
subroutine eigen_convert_ID_w2xy(inod, xinod, yinod)
```

1. integer, intent(IN) :: inod

基盤コミュニケーターにおけるプロセスID

2. integer, intent(OUT) :: xinod  
x\_commにおけるプロセスID
3. integer, intent(OUT) :: yinod  
y\_commにおけるプロセスID

## 4.24 KMATH\_EIGEN\_GEV

EigenExaを固有値計算エンジンとして利用する一般化実対称固有計算ドライバルーチンである。version2.6以降からEigenExaに収納されている(それ以前のversionでは別途パッケージをmakeする必要がある)。本ドライバの内部で五重対角行列への変換を経て固有対を計算するeigen\_sxを次のような計算過程で呼び出す:  $Cy = \lambda y$ , where  $B = X_B \Lambda_B X_B^T \Rightarrow Y_B = X_B \Lambda_B^{1/2} \Rightarrow C = Y_B^{-1} A Y_B^{-T}, y = Y_B^T x$ 。本ドライバはeigen\_sxと同様の制約を受ける。

```
subroutine kmath_eigen_gev(n, nvec, a, lda, b, ldb, w, z, ldz)
```

1. integer, intent(IN) :: n  
行列・ベクトルの次元
2. integer, intent(IN) :: nvec  
計算する固有値・対応する固有ベクトル(固有モード)の本数  
正值の場合は, 最小から指定された個数の固有モードを計算する。  
一方, 負値の場合は, 最大から指定された個数の固有モードを計算する  
(負値使用は現バージョンで未対応)。
3. real(8), intent(INOUT) :: a(1:lda,\*)  
計算対象のペンシル( $A - \lambda B$ )の行列A (上三角行列要素のみ有効), サイズは  
eigen\_get\_matdims を呼び出して得られた値以上でなければならない。  
サブルーチン終了時には配列の内容は破壊される
4. integer, intent(IN) :: lda  
配列 a の整合寸法 (リーディングディメンジョン)  
eigen\_get\_matdims を呼び出して得られた値以上でなければならない。
5. real(8), intent(INOUT) :: b(1:ldb,\*)  
計算対象のペンシル( $A - \lambda B$ )の行列B (上三角行列要素のみ有効), サイズは  
eigen\_get\_matdims を呼び出して得られた値以上でなければならない。  
サブルーチン終了時には標準固有値問題への変換行列が格納される。
6. integer, intent(IN) :: ldb  
配列 b の整合寸法 (リーディングディメンジョン)



`eigen_get_matdims` を呼び出して得られた値以上でなければならない.

7. `real(8), intent(OUT) :: w(1:n)`

昇順の固有値

8. `real(8), intent(OUT) :: z(1:ldz,*)`

一般化固有値問題の  $B$  直交固有ベクトル, サイズは

`eigen_get_matdims` を呼び出して得られた値以上でなければならない.

9. `integer, intent(IN) :: ldz`

配列  $z$  の整合寸法 (リーディングディメンジョン)

`eigen_get_matdims` を呼び出して得られた値以上でなければならない.



## 第5章

# その他の注意事項

### 5.1 互換性の注意

EigenExaはEigenKの後継で多くの機能を継承している。しかしながら両者の間には完全な互換性を保証していない。それは内部実装の詳細が異なることに起因しており、主に関数や変数の命名則、コモン領域の管理方法の違いによるものである。また、同様の理由によりEigenExaとEigenKを同時にリンクすることを勧めない。

EigenExaのバージョン2.3から2.4にアップデートする際に、モジュールの管理方法や命名ルールが変更されている。したがって、バージョン2.3以前の利用者が上位バージョンにアップデートする際には、ライブラリのみならずモジュールを読み込んでいるアプリケーション側の各種ソースコードも再コンパイルが必要となる。

### 5.2 他の言語との結合

Fortran90以外からのEigenExaの呼び出し方法は利用者の環境に大きく依存する。コンパイラマニュアルを引き、「言語結合(language bindings)」や「複数プログラミング言語とのリンク方法」を参照してほしい。なお、Python言語からの呼び出しを可能とする"Python binding of EigenExa" プロジェクト[\[18\]](#)の成果も存在する。それらを参考にしてほしい。

### 5.3 エラー発生時の振舞い

EigenExaは初期化時に適切な条件のもとで実行されているかのチェックを行うが、実行時にはエラー検出を行っていない。リンクしたBLASやLAPACKなど下位のライブラリがエラーを吐いてライブラリ強制終了される場合がある。つまり、これらのエラー発生は、メインの呼び出しルーチンを再開するのではなく、メインプログラムを終了させるので注意が必要である。

不具合並びにバグ発見の情報はライブラリの品質向上には欠かすことができないものである。バグ発見時には、ライブラリ公開HPや本ドキュメントに書かれている開発者のメイ

ルアドレス(EigenExa@ml.riken.jp)まで一報してほしい。

## 5.4 バージョン1.xにおけるシェアード・ライブラリの扱い

旧バージョン1.xではシェアード・ライブラリをサポートしていない。バージョン1.x開発当時、シェアード・ライブラリ使用時に関数名の解決が衝突なく完全にできることを保証できなかったからである（gccのあるバージョンでは実行時に関数名の解決ができず異常終了する場合があった）。バージョン1.xをシェアード・ライブラリとして利用する場合はあくまでも利用者の責任の元実行して欲しい。

EigenExaのバージョン2.0からは、理化学研究所計算科学研究機構の高度利用化チーム前田俊行チームリーダー(2015年当時)らの技術協力を受けシェアード・ライブラリ化が実現し、バージョン2.4以降からはスタティック・ライブラリとシェアード・ライブラリの両者をビルドする仕様になった。実行時に適切な環境変数(LD\_LIBRARY\_PATHなど)の設定を忘れずに行えば、適切なライブラリが選択され動作するはずである。

## 5.5 既知の問題点と推奨回避策

### 5.5.1 Intelコンパイラ+Intel MPI利用時の数値再現性について

いくつかのMPI実装系では、通信性能を向上するため集団通信とくに算術演算を含むリダクション操作に対して、再現性を保証しないアルゴリズムを採用するものが存在する。Intel MPIのドキュメント[19]には、ネットワークポロジを意識したアルゴリズムを採用しているために、実数のリダクションにおいて(MPI\_SUMなど)ビットレベルの計算再現性が保証されないと明記されている。EigenExaでは内部でいくつかのプロセスグループにわけ冗長な計算を行う部分が存在している。それらプロセスグループ間では、計算結果のビットレベルでの一致が必要となるため、計算再現性が破綻するとEigenExaの動作そのものが異常となる可能性がある。そのため、内部実装においてできる限りそのような状況を避けるアルゴリズムに切り替えているが、現在のアルゴリズム実装が問題の本質部分を解決しているわけではない。

Intel MPIではそのような、再現性の不具合を解消するために、環境変数I\_MPI\_CBWRが用意されている。EigenExaは、内部的に同環境変数を指定した場合と同じ機能を達成するように、コミュニケータに実行時属性を付与する処理をしている。しかしながら、ユーザーが陽に通信アルゴリズムを指定した場合に、同機能が完全に働くとは言い切れない。したがって、**Intel MPI**を使用する場合は、環境変数I\_MPI\_CBWRを2に指定することを推奨する。<sup>\*1</sup>

```
% export I_MPI_CBWR=2 (Bashなど)
```

<sup>\*1</sup> ただし、この対策は同時にリンクしている他の数値計算ライブラリやユーザーのソースセグメントでのビット単位の完全な数値再現性を保証するものではない

```
% setenv I_MPI_CBWR 2 (Cシェルなど)
```

また、コンパイラの最適化レベルによる不具合を避けるため、**Intel compiler**を使用する場合は、configureスクリプトの環境変数オプション(CFLAGSやFFLAGS)に`-fp-model=strict`以下の計算精度に影響を与えるオプションを指定しないことを推奨する。なお、configureスクリプトはintel compilerが検出された場合には、同オプション(`-fp-model=strict`)を自動的に指定するため、利用者が別途追加指定する必要はない。



## 付録





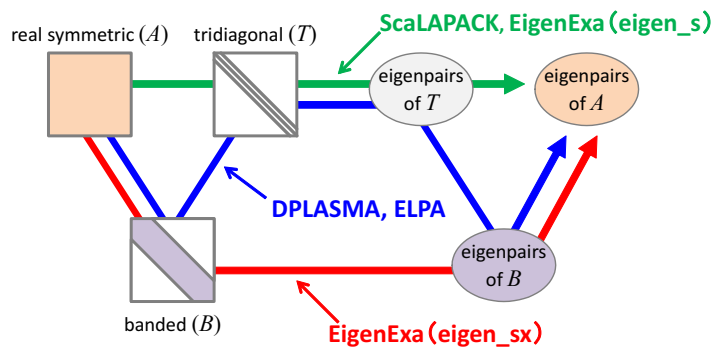
## A章

# アルゴリズムの概要

### A.1 はじめに

本章では、EigenExaで採用されている固有値計算アルゴリズムの概要を述べる。EigenExaでは、実対称密行列の全ての固有値と固有ベクトルを計算する場合を想定しており、これを実現する二つのドライバルーチン（`eigen_s`と`eigen_sx`）を提供する。本章では、主にこの二つのルーチンの違いに主眼を置いて、両者で採用されているアルゴリズムを概観する。なお、密行列向け固有値計算アルゴリズムの一般的な内容は[20, 21, 22, 23, 24, 25, 26, 27, 28]などを参照されたい。

### A.2 様々なアプローチと関連プロジェクト



図A.1 実対称密行列向け固有値計算の様々なアプローチ。

最初に、実対称密行列向け固有値計算の手順については簡単に紹介する。一般的な行列計算の教科書に記載されているのは、入力行列の三重対角化に基づくアプローチ（図A.1の緑色のパス）であり、ScaLAPACK[29]（やLAPACK）で採用されている。しかし、このアプローチでは、最初のステップ（三重対角化）がメモリバンド幅に律速され、近年の計算機システム上では高い性能が期待できないことが問題となっている。

この問題を踏まえて、ELPA[30]やDPLASMA[31]と呼ばれるプロジェクトでは、帯行列を経由する二段階の三重対角化に基づくアプローチ（図A.1の青色のパス）が採用されている。この二段階の三重対角化における主要コストは一段階目の帯行列化の部分であり、その部分の要求byte/flop値が直接三重対角化する場合よりも小さくなるため、実効性能の向上が期待できる。ただし、固有ベクトルの逆変換も二段階必要（こちらはコストが単純に二倍）になり、その一段階目（ $T$ から $B$ への逆変換）の高性能実装が困難となっている。そのため、求める固有ベクトルの本数が増えると逆変換のコストが膨大となってしまう問題を抱えている。

この状況を踏まえて、EigeExaでは従来の（一段階の）三重対角化に基づくアプローチ（図A.1の緑色のパス）を採用したルーチン（eigen\_s）と、帯行列の固有値・固有ベクトルを直接計算するアプローチ（図A.1の赤色のパス）を採用したルーチン（eigen\_sx）の二つを開発・提供している。これら二つのアプローチについては、次節以降でもう少し詳しく述べる。

### A.3 eigen\_s

EigenExaで提供されるドライバルーチンの一つであるeigen\_sは、上述のようにScaLAPACK等で採用されている従来の（一段階の）三重対角化に基づくアプローチを採用している。具体的には、以下の三つのステップにより、固有値問題 $Ax_i = \lambda_i x_i$  ( $i = 1, \dots, N$ )を解く。

1. ハウスホルダー変換による入力行列の三重対角化： $Q^T A Q \rightarrow T$
2. 分割統治法による三重対角行列の固有値・固有ベクトルの計算： $Ty_i = \lambda_i y_i$
3. 固有ベクトルの逆変換： $Qy_i \rightarrow x_i$

最初のステップでは、ハウスホルダー変換を両側から

$$H_{N-2}^T \cdots H_1^T A H_1 \cdots H_{N-2} \rightarrow T, \quad H_i = I - u_i \beta_i u_i^T \quad (A.1)$$

と作用させて、入力行列を一行（一行）ごとに三重対角行列に変形していく（図A.2(a)）。なお、 $\beta$ はスカラーであるが、後述の内容との対応を明確にするために、式中の位置を配慮している。詳細は割愛するが、一つのハウスホルダー変換による変形の計算は、

$$(I - u\beta u^T)^T A (I - u\beta u^T) = A - uv^T - vu^T, \quad v = (w - \frac{1}{2}u\beta^T(w^T u))\beta, \quad w = Au \quad (A.2)$$

として $A$ の対称性を利用して行われることが一般的となっている。また、Dongarraの手法を用いることで、複数のハウスホルダー変換による $A$ の更新を以下のように行列積の形で一度に行うことが可能となる。

$$(I - u_K \beta_K u_K^T)^T \cdots (I - u_1 \beta_1 u_1^T)^T A (I - u_1 \beta_1 u_1^T) \cdots (I - u_K \beta_K u_K^T) = A - UV^T - VU^T. \quad (A.3)$$

しかし、行列ベクトル積はそのまま残り（行列 $V$ の計算のため）、この部分がメモリバンド幅律速となり、高性能化の際に大きなボトルネックとなる。

二番目のステップでは、Cuppenにより提案された分割統治法[32]により、三重対角行列の固有値と固有ベクトルを計算する。この手法は、図A.2(b)に示すように三重対角行列をブロック対角行列とランク1の摂動に分解し、ブロック対角行列の各ブロックの固有値分解結果を利用して元の（ランク1の摂動が加わった）行列の固有値分解を効率的に計算する、という原理に基づいている。

三番目のステップでは、最初のステップで得られたハウスホルダー変換を逆の順番に作用させて固有ベクトルの逆変換を行う。実際には、複数個のハウスホルダー変換が、

$$H_1 \cdots H_K = (I - \mathbf{u}_1 \beta_1 \mathbf{u}_1^\top) \cdots (I - \mathbf{u}_K \beta_K \mathbf{u}_K^\top) \rightarrow I - U S U^\top, \quad U = [\mathbf{u}_1 \cdots \mathbf{u}_K] \quad (\text{A.4})$$

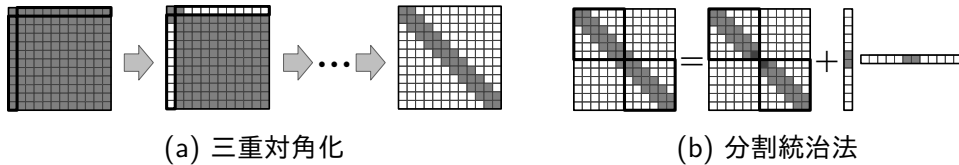
と少ないコスト（ $S$ の計算のみ）で行列を使った表現に合成できる（compact-WY表現）ことを用いて、

$$H_1 \cdots H_{N-2} Y = (I - U_1 S_1 U_1^\top) \cdots (I - U_M S_M U_M^\top) Y \rightarrow X \quad (\text{A.5})$$

として行列積（Level-3 BLAS）で計算するため、高性能が期待できる。なお、ここで

$$Y = [\mathbf{y}_1 \cdots \mathbf{y}_N], \quad X = [\mathbf{x}_1 \cdots \mathbf{x}_N] \quad (\text{A.6})$$

である。



図A.2 eigen\_sの固有値計算の概略図。

eigen\_sでは、一番目と三番目のステップは独自の実装を採用しており、適切なスレッド並列化等を施すことで高性能化を図っている。一方、二番目のステップについては、ScaLAPACKのコードをベースとした実装となっている。

## A.4 eigen\_sx

EigenExaで提供されるもう一つのドライバルーチンeigen\_sxは帯行列の固有値・固有ベクトルを直接計算するアプローチを採用している。現状、後述の理由により帯行列として五重対角行列を採用している。具体的には、以下の三つのステップにより固有値問題を解く。

1. ブロック版のハウスホルダー変換による入力行列の五重対角化： $\tilde{Q}^\top A \tilde{Q} \rightarrow B$
2. 分割統治法による五重対角行列の固有値・固有ベクトルの計算： $B \mathbf{y}_i = \lambda_i \mathbf{y}_i$

### 3. 固有ベクトルの逆変換： $\tilde{Q}y_i \rightarrow x_i$

最初のステップでは、ブロック版のハウスホルダー変換を両側から

$$\tilde{H}_{N/2-1}^\top \cdots \tilde{H}_1^\top A \tilde{H}_1 \cdots \tilde{H}_{N/2-1} \rightarrow P, \quad \tilde{H}_i = I - \tilde{u}_i \tilde{\beta}_i \tilde{u}_i^\top \quad (\text{A.7})$$

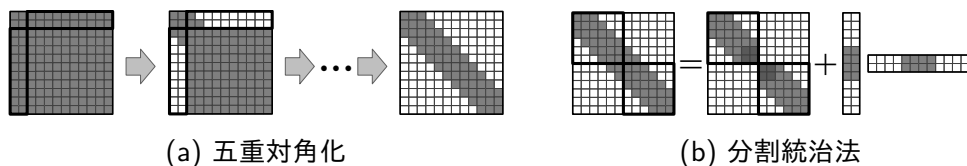
と作用させて、入力行列を二列（二行）ごとに三重対角行列に変形していく（図A.3(a)）。ただし、

$$\tilde{u}_i = [\mathbf{u}_1^{(i)} \ \mathbf{u}_1^{(i)}], \quad \tilde{\beta}_i = \begin{pmatrix} \beta_{11}^{(i)} & \beta_{12}^{(i)} \\ \beta_{21}^{(i)} & \beta_{22}^{(i)} \end{pmatrix} \quad (\text{A.8})$$

である。あとは、式(A.2)と全く同じ形式で計算を行うことができ、Dongarraの手法も同様に適用が可能となっている。その結果として、最終的にボトルネックになるのは、 $A\tilde{u}$ の部分となる。

二番目のステップでは、図A.3(b)に示すように五重対角行列をブロック対角行列とランク2の摂動に分解し、三重対角行列に対する分割統治法と同様の原理を繰り返す（ランク2の摂動を二つのランク1の摂動として処理する）ことにより、五重対角行列の固有値・固有ベクトルを計算する[33]。

最後のステップでは、最初のステップで用いたブロック版ハウスホルダー変換を逆順で作用させることで、固有ベクトルの逆変換を計算する。ブロック版のハウスホルダー変換に関しても、式(A.4)と同様の形で行列を使った表現に合成することが可能であるため、同様の計算手順により行列積が利用可能で高性能が期待できる。



図A.3 eigen\_sxの固有値計算の概略図。

eigen\_sxでも、一番目と三番目のステップは独自の実装を採用しており、適切なスレッド並列化等を施すことで高性能化を図っている。一方、二番目のステップについては、ScaLAPACKのコードをベースとして、五重対角行列向けに拡張した実装を行っている。

## A.5 eigen\_sとeigen\_sxの違い

先の二つの節の説明から、eigen\_sとeigen\_sxは同じような三つのステップで構成されており、特に固有ベクトルの逆変換のステップに関しては、ほぼ同様の計算内容となり両者の間に大きな差はない。本節では、この二つのステップにおける両者の差を述べる。

### 三重（五重）対角化

このステップにおける違いは、eigen\_sでは一本のベクトルを単位として処理をするの

に対して、eigen\_sxでは二本のベクトルを単位として処理をする部分である。具体的には、

$$\boldsymbol{w} = A\boldsymbol{u} \text{ (in eigen\_s)} \quad \rightarrow \quad [\boldsymbol{w}_1 \ \boldsymbol{w}_2] = A[\boldsymbol{u}_1 \ \boldsymbol{u}_2] \text{ (in eigen\_sx)} \quad (\text{A.9})$$

といった部分である。まず、ステップ全体としての演算量に関しては、(少なくとも最高次の項に関しては)両者は同程度となる。これは、一回当たり演算量はeigen\_sの方が少ない(約半分)が、処理の回数は二列(二行)ずる進めるeigen\_sxの方が少ない(約半分)からである。同様に、分散並列計算時に通信するデータの総量に関しても、両者は同程度となる。この理由も演算量の場合と同じで、一回当たりのデータ量と回数の関係にある。

両者の間で生じる一つ目の差は、浮動小数点演算(特に行列ベクトル積)の実効性能である。 $A\boldsymbol{u}$ を計算する場合に対して $A[\boldsymbol{u}_1 \ \boldsymbol{u}_2]$ を計算する場合、行列 $A$ のデータの再利用性が向上する。つまり、要求byte/flop値が減少することになる。これにより、メモリバンド幅に律速される影響が小さく(理論的には約半分)なり、演算の実効性能の向上が期待できる。

二つ目の差は、通信回数の違いから生じる、通信のレイテンシの差である。eigen\_sxは一回の通信データ量が多いが、通信回数はeigen\_sよりも少なく(約半分)なる。近年、通信のレイテンシが大きな問題となっているため、この通信回数が少ない(Communication-Avoidance)という特徴は、特に並列数が多くなった場合に非常に大きな差となる。

このように、eigen\_sxはeigen\_sに比べて、演算性能と通信のレイテンシの面で有利であり、並列数に対して問題サイズが十分に大きい(演算時間が支配的な)場合は前者の効果が大きく、逆に並列数が多い(通信時間が支配的な)場合は後者の効果が大きくなることが予想される。

#### 分割統治法

分割統治法では、eigen\_sが毎回ランク1の摂動を処理するのに対して、eigen\_sxではランク2の摂動を処理する必要があるため、その分の演算量と通信コストが両者の差として生じる。さらに、五重対角行列に対する分割統治法では、ランク2の摂動を二つのランク1の摂動として処理するが、二回目のランク1の摂動を処理する場合に、行列の構造を利用することが出来なくなり(一回目はブロック対角という構造を活用して演算量を削減できる)、その結果として、演算量が三倍程度となってしまう。

ただし、分割統治法では、デフレーションと呼ばれる技法により、演算量を大幅に削減することが可能となる場合がある。デフレーションの発生頻度は問題依存であり、三重対角行列を経由した場合と五重対角行列を経由した場合では、同じ入力行列でも、デフレーションの起こり方に差が生じる。そのため、両者のコストの差を理論的に見積もることは容易ではない。

## A.6 おわりに

本章では、EigenExaで提供されている二つのルーチン`eigen_s`と`eigen_sx`のアルゴリズムの概要とその違いを説明した。一般に経由する帯行列の帯幅を大きくすると、最初の変換するステップでは有利になり、二番目の分割統治法のステップでは不利になるというトレードオフがあり、現状では、五重対角行列が妥当であるとの判断から、`eigen_sx`では五重対角化を採用している。今後のシステムの性能や分割統治法の実装の改善次第（現状のScaLAPACKベースのコードの性能は十分とはいえない）で、より大きな帯行列（例えば七重対角行列）が有望となる可能性もある。逆に、従来の三重対角化（`eigen_s`）を使う場合に最善となるケースも状況次第では在り得ると思われる。ぜひ、本章の内容を把握した上で、ユーザが適切なルーチンを選択してもらえると幸いである。

## B章

# Release notes

これまでのリリースの履歴については `ReleaseNotes.txt` にも示されている。

### B.1 Version 2.13 (July 23, 2024)

- Fix some internal errors occurring in a  $>32$ bit integer expression. Still, unexpected overflow may happen in huge calculation cases.
- Fix binding mismatches or instantiation of the modules of the compiler versions in C, C++, F9X, and F2XXX.
- Fix Intel oenAPI compiler command from `mpiicc` to `mpiicx`, even though they are not called in this version...
- Experimental implementation of a minimal set of caller APIs from C/C++ language environment (confirmed gcc, icc, and DPC++/c++).

### B.2 Versiob 2.12 (January 10, 2023)

- **[Serious]** Bug fix for a very rare condensed clustering case of eigenmodes. Modified the deflation counting correctly.
- Modify the internal integer treatment partly from `int32` to `int64` to avoid overfull of the huge memory assignment and indexing.
- Experimental support of a MacOS environments.
- Modify the benchmark options to check the specific kernel.
- Modify the collective communication algorithms.

### B.3 Version 2.11 (December 1, 2021)

- **[Upgrade]** Experimental support for Hermite solver, namely, `eigen_h`.

- **[Serious]** Fix the numerical error happened to be included in v2.10, where the tall-skinny QR coded in `eigen_prd_t4x.F` of `eigen_sx` was too sensitive to treat tiny values and forced-double truncation. But, it might depend on the compiler version and the code generator.
- Modify pointer attribution to 'allocatable' to avoid automatic deallocation on the exit of callee routines.
- Some code modifications are applied to pass strong debugging tools with respect to Fortran 95 and some of Fortran 2003 extensions.
- Fix some bugs, for exmaple, missing private attribution to some variables for OpenMP.

## B.4 Version 2.10 (Octorber 17, 2021)

- **[Serious]** Bug fix for violation of the result of allreduce in DC. It happened very rarely when data to be transferred was shorter than the number of processes participating.
- **[Serious]** Bug fix for inconsistent API interpretation of DLAED4, when  $K$  is less than or equal to 2. This bug happened when a lot of deflations are carried out, and sub-matrices are shrunk tiny as 1 or 2. So, it is infrequent to see.
- **[Serious]** Bug fix for non-deterministic behavior of the DC branch, which happened if an uninitialized variable referred in the brach condition, and is affected by the side-effects of other modules, etc. It was fixed when 2.9 was released but noted in the release.
- Reduce the internal data capacity in the TRD and DC routines.
- Fix the installation of Fortran modules.

## B.5 Version 2.9 (September 24, 2021)

- Modify the flops count precise in DC kernels.
- Modify `trbak` not to multiply  $D^{-1}$  and `TRSM`.
- Add enable/disable-switch for building the shared library.
- Modify to detect the memory allocation fault.

## B.6 Version 2.8 (August 20, 2021)

- **[Minor]** Modify the DC kernel to reduce intermediate buffer storage.
- Bug fix on a t1 loop structure



- Updated the error check routine
- Fixed on Makefile to add the missing fortran module.

## B.7 Version 2.7 (April 1, 2021)

- **[Minor]** Modify the compilation rules corresponding to static/shared libraries defined in `src/Makefile.am`.
- Performance tweak with a modification of the compilation options not to use `-fPIC` when build a static library.
- License document is packed as an independent file (the license notice was stated in User's manual for version 2.6).

## B.8 Version 2.6 (November 1, 2020)

- **[Upgrade]** This version applies a communication avoidance technique to the householder tridiagonalization together with, new process mapping for the load balance of the divide and conquer method.

## B.9 Version 2.5 (August 1, 2019)

- **[Internal]** Refine the data distribution in the divide and conquer algorithm routine.
- This version is only internal management version and not published.

## B.10 Version 2.4b (August 20, 2018)

- **[Serious]** Bug fix for incorrect data redistribution, which might violate allocated memory. The bug might have happened in the case that the number of processes,  $P = P_x * P_y$ , is large, and  $P_x$  and  $P_y$  are not equal but nearly equal.
- This version is for only bug fix for the serious one.

## B.11 Version 2.3m (August 20, 2018)

- **[Serious]** Bug fix for incorrect data redistribution, which might violate allocated memory. The bug might have happened in the case that the number of processes,  $P = P_x * P_y$ , is large, and  $P_x$  and  $P_y$  are not equal but nearly equal.
- This version is for only bug fix for the serious one.

## B.12 Version 2.4p1 (May 25, 2017)

- **[Serious]** Bug fix for incorrect data redistribution in `eigen_s`.
- Major change with Autoconf -and- Automake framework.

## B.13 Version 2.4 (April 18, 2017)

- **[Upgrade]** Major change with Autoconf -and- Automake framework

## B.14 Version 2.3k2 (April 12, 2017)

- Communication Avoiding algorithms to the `eigen_s` driver.
- The optional argument `nvec` is available, which specifies the number of eigenvectors to be computed from the smallest. This version does not employ the special algorithm to reduce the computational cost. It only drops off the unnecessary eigenmodes in the backtransformation.

## B.15 Version 2.3d (July 07, 2015)

- Tuned up the parameters according to target architectures.
- Introduce a sort routine in `bisect.F` and `bisect2.F` for eigenvalues.
- Modify the algorithm to create reflector vectors in `eigen_prd_t4x.F`
- Modify the matrix setting routine to load the `mtx` (Matrix Market) format file via both '`A.mtx`' and '`B.mtx`'.
- Re-format the source code by the `fortran-mode` of `emacs` and extra rules.

## B.16 Version 2.3c (April 23, 2015)

- Fix bug on flops count of `eigen_s` which returned incorrect value due to missing initialization in `dc2.F`.
- This bug is found in version 2.3a and version 2.3b.
- Minor change on timer routines.
- Minor change on broadcast algorithm in `comm.F`.

## B.17 Version 2.3b (April 15, 2015)

- Minor change to manage the real constants.
- Minor change to use Level 1 and 2 BLAS routines.
- Minor change to preserve invalid or oversized matrices.
- Minor change of Makefile to allow '-j' option.

## B.18 Version 2.3a (April 14, 2015)

- Minor change on thread parallelization of `eigen_s`.
- Minor change of the API's for timer routines.
- Fix the unexpected optimization of rounding errors in `eigen_dcx()`.

## B.19 Version 2.3 (April 12, 2015)

- Bug fix on the benchmark program.
- Refine the race condition in the backtransformation routine.
- Introduce Communication Avoiding algorithms to the `eigen_s` driver.

## B.20 Version 2.2d (March 20, 2015)

- Bug fix on the timer print part in `trbakwy4.F` not to do zero division.
- Modify the synchronization point in `eigen_s`.
- Modify thread parallelization in `eigen_dc2()` and `eigen_dcx()`.

## B.21 Version 2.2c (March 10, 2015)

- Bug fix on the benchmark program.
- Add the `make_inc` file for an NEC SX platform.

## B.22 Version 2.2b (October 30, 2014)

- Introduce new API to query the current version.
- Introduce the constant `eigen_NB=64`, which refers to the block size for cooperative work with the ScaLAPACK routines.

- Correct the required array size in `eigen_mat_dims()`.
- Improve the performance of test matrix generator routine `mat_set()`.
- Add the listing option of test matrices in `eigenexa_benchmark/`.

### B.23 Version 2.2a (June 20, 2014)

- Fix minor bug of `Makefile`, `miscC.c` and etc for BG/Q.
- Modify the initialization process not to use invalid communicators.
- Comment out the calling `BLACS_EXIT()` in `eigen_free()`.

### B.24 Version 2.2 (April 10, 2014)

- Arrange the structure of source directory.
- Reversion of the DC routines back to version 1.3a to avoid bug.
- Hack `miscC.c` to be called from IBM BG/Q.
- Fix bug on the benchmark program for exceptional case of `MPI_COMM_NULL`.
- Fix bug on `eigen_s` with splitted communicator.
- Update machine depended configuration files.
- Experimental support of building a shared library

### B.25 Version 2.1a (Feb 23, 2014)

- Fix bug on the benchmark program.

### B.26 Version 2.1 (Feb 10, 2014)

- Fix bug on `eigen_sx`: it gave wrong results when  $N=3$ .
- Modify the `bisect2` by a pivoting algorithm.
- Update the test program '`eigenexa_benchmark`' in order to check accuracy with several test matrices and computing modes.
- Tune performance for K computer and Fujitsu FX10 platforms.
- Add `make_inc` file for a BlueGeneQ platform, but it is not official support, just an experimental.

## B.27 Version 2.0 (Dec 13, 2013)

- **[Upgrade]** Add `eigen_s`, which adopts the conventional 1-stage algorithm.
- Add optional modes to compute only eigenvalues and to improve the accuracy of eigenvalues.
- Modify to support a thread mode with any number of threads.
- Tune performance for K computer and Fujitsu FX10 platforms.

## B.28 Version 1.3a (Sep 21, 2013)

- Fix bug on synchronization mechanism of `eigen_trbakwyx()`.

## B.29 Version 1.3 (Sep 20, 2013)

- Fix bug on `eigen_init()` in initialization with `MPI_Cart`'s or `MPI_COMM_NULL`'s.
- Add test programs to check several process patterns.

## B.30 Version 1.2 (Sep 17, 2013)

- Fix bug on benchmark code in making a random seed.
- Modify to support upto 64-thread running.

## B.31 Version 1.1 (Aug 30, 2013)

- Fix bug on data-redistribution row vector to column vector when  $P = p * q$  and  $p$  and  $q$  have common divisor except themselves.
- Optimize data redistribution algorithm in `dc_redist[12].F`.

## B.32 Version 1.0 (Aug 1, 2013)

- **[Release]** This is the first release
- Standard eigenvalue problem for a dense symmetric matrix by a novel one-stage algorithm



# 謝辞

EigenExa開発グループ全員の真摯な取り組み, 精力的な活動, 各方面の協力, 特に, 理学研究所計算科学研究センター(旧計算科学研究機構)各位の支援に対して感謝する。また, 多くの利用者からバグ情報や品質向上のためのフィードバックを頂いている。関係した多くのメンバーの努力なしにはEigenExaを公開することは叶わなかったであろう。最後に, EigenExa開発には幾つかの外部資金ならびに開発プロジェクトの援助を受けている。それらは以下の通りである。併せてここに謝意を表したい。

1. 科学技術振興機構 戦略的創造研究推進事業CREST 「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」 (平成23年度～27年度), ならびにESSEX-II project in SPPEXA-phase2 (2016-18)
2. 文部科学省 科学研究費補助(科研費): 基盤研究(B)課題番号21300013 (平成24年度), 基盤研究(A)課題番号23240005 (平成23年度～25年度), 基盤研究(B)課題番号15H02709 (平成27年度～29年度), 基盤研究(B)課題番号19H04127 (平成31, 令和元年～3年度)
3. HPCI利用研究課題, 「京」一般利用: 課題番号hp140069 (平成26年度), hp120170(平成24～25年度)
4. 「京」調整高度化枠利用: 課題番号ra000005 (平成25年度～31, 令和元年度)
5. FX10 スーパーコンピュータシステム「大規模 HPC チャレンジ」, 東京大学情報基盤センター (2013年)
6. FS2020ポスト「京」開発プロジェクト (2014年度～2020年度)
7. 「「富岳」共用前評価環境」, poroject ID ra000006 (2020年度)
8. 「「富岳」試行的利用課題(利用準備課題)」, project ID hp200263 (2020年度), ならびに hp200313 (2021年度上半期)
9. 「「富岳」高度化・利用拡大枠」, poroject ID ra000006 (2021年度～)

また, EigenExa以前のEigenKの時代には以下の外部資金からの支援を受けていた。共同研究者である国立研究開発法人日本原子力研究開発機構システム計算科学センターの皆様にごにこにこ合わせて謝意を示す。

10. 科学技術振興機構 戦略的創造研究推進事業CREST 「マルチスケール・マルチフィジックス現象の統合シミュレーション」 (平成18年度～24年度)





## 参考文献

- [1] S. Yamada, T. Imamura, T. Kano and M. Machida, “High-Performance Computing for Exact Numerical Approaches to Quantum Many-Body Problems on the Earth Simulator”, Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC '06), November 2006. Tampa USA.  
<http://doi.acm.org/10.1145/1188455.1188504>
- [2] 今村俊幸, “T2Kスパコンにおける固有値ソルバの開発”, 東京大学スーパーコンピューティングニュース, Vol.11, No.6, pp.12-32 (2009).  
<http://www.cc.u-tokyo.ac.jp/support/press/news/VOL11/No6/200911imamura.pdf>
- [3] T. Imamura, S. Yamada and M. Machida, “Development of a High Performance Eigensolver on the Peta-Scale Next Generation Supercomputer System”, Progress in Nuclear Science and Technology, the Atomic Energy Society of Japan, Vol. 2, pp.643–650 (2011) .
- [4] T. Imamura, S. Yamada and M. Machida, “Eigen-K: high performance eigenvalue solver for symmetric matrices developed for K computer” , 7th International Workshop on. Parallel Matrix Algorithms and Applications (PMAA2012), June 2012, London UK.
- [5] 「京」について | 理化学研究所計算科学研究機構 (AICS) ,  
<http://www.aics.riken.jp/jp/k/>
- [6] スーパーコンピュータ「京」 — Fujitsu, Japan,  
<http://www.fujitsu.com/jp/about/businesspolicy/tech/k/>
- [7] スーパーコンピュータ「富岳」プロジェクト,  
<https://www.r-ccs.riken.jp/jp/post-k/project.html> (日本語) <https://www.r-ccs.riken.jp/en/fugaku/project> (英語)
- [8] T. Imamura and Y. Yamamoto, “CREST: Dense Eigen-Engine Groups”, International Workshop on Eigenvalue Problems: Algorithms; Software and Applications, in Petascale Computing (EPASA 2014), Tsukuba, March 7–9, 2014 (poster).  
[http://www.aics.riken.jp/labs/lpnctr/EPASA2014\\_dense\\_poster\\_ImamuraT\\_only.pdf](http://www.aics.riken.jp/labs/lpnctr/EPASA2014_dense_poster_ImamuraT_only.pdf)

- [9] T. Imamura, “The EigenExa Library – High Performance & Scalable Direct Eigensolver for Large-Scale Computational Science”, HPC in Asia, Leipzig, Germany, June 22–26, 2014.
- [10] T. Imamura, Y. Hirota, T. Fukaya, S. Yamada and M. Machida, “EigenExa: high performance dense eigensolver, present and future”, 8th International Workshop on Parallel Matrix Algorithms and Applications (PMAA14), Lugano, Switzerland, July 2–4, 2014.
- [11] 深谷猛, 今村俊幸, “FX10 4800ノードを用いた密行列向け固有値ソルバ EigenExa の性能評価”, 東京大学スーパーコンピューティングニュース, Vol.16 No.3, pp.20-27 (2014). [http://www.cc.u-tokyo.ac.jp/support/press/news/VOL16/No3/09\\_User201405-1.pdf](http://www.cc.u-tokyo.ac.jp/support/press/news/VOL16/No3/09_User201405-1.pdf)
- [12] T. Fukaya and T. Imamura, “Performance evaluation of the EigenExa eigensolver on the Oakleaf-FX supercomputing system”, Annual Meeting on Advanced Computing System and Infrastructure (ACSI) 2015, Tsukuba, January 26–28, 2015.
- [13] Takeshi Fukaya, and Toshiyuki Imamura, “Performance Evaluation of the Eigen Exa Eigensolver on Oakleaf-FX: Tridiagonalization Versus Pentadiagonalization”, Proceedings of the Parallel and Distributed Processing Symposium Workshop (IPDPSW, PDSEC 2015), p.960-969, doi:10.1109/IPDPSW.2015.128, 2015.
- [14] Yusuke Hirota, Daichi Mukunoki, Susumu Yamada, Narimasa Sasa, Toshiyuki Imamura, and Masahiko Machida, “Performance of Quadruple Precision Eigenvalue Solver Libraries QPEigenK & QPEigenG on the K Computer”, poster presentation, ISC2016, Best poster award in ‘HPC in Asia’.
- [15] Toshiyuki Imamura, Takeshi Fukaya, Yusuke Hirota, Susumu Yamada, and Masahiko Machida: “CAHTR: Communication-Avoiding Householder Tridiagonalization”, Proceedings of ParCo2015, Advances in Parallel Computing, Vol.27: Parallel Computing: On the Road to Exascale, p.381-390, doi:10.3233/978-1-61499-621-7-381, 2016
- [16] Yusuke Hirota, and Toshiyuki Imamura, “Performance Analysis of a Dense Eigenvalue Solver on the K Computer”, Proceedings of the 36th JSST Annual International Conference on Simulation Technology, Oct. 2017.
- [17] Takeshi Fukaya, Toshiyuki Imamura, and Yusaku Yamamoto, “A Case Study on Modeling the Performance of Dense Matrix Computation: Tridiagonalization in the EigenExa Eigensolver on the K Computer”. Proceedings of the Parallel and Distributed Processing Symposium Workshop (IPDPSW, iWAPT 2018), pp. 1113-1122, doi:10.1109/IPDPSW.2018.00171, 2018
- [18] Python binding of EigenExa, HPC Usability Research Team, RIKEN AICS, <http://www.hpcu.aics.riken.jp/>
- [19] Intel MPI Library Developer Reference for Linux OS, Developer Reference, 特に, 以下の集団通信アルゴリズムの利用者指定方法について詳細がある. <https://>

[//software.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/environment-variable-reference/i-mpi-adjust-family-environment-variables.html](https://software.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/environment-variable-reference/i-mpi-adjust-family-environment-variables.html)

- [20] 一般社団法人日本計算工学会編, 長谷川秀彦, 今村俊幸, 山田進, 櫻井鉄也, 荻田武史, 相島健助, 木村欣司, 中村佳正著計算力学レクチャーコース “固有値計算と特異値計算”, 丸善出版 (2019)
- [21] 杉原正顯, 室田一雄, “線形計算の数理”, 岩波書店 (2009).
- [22] B. Parlett, “The Symmetric Eigenvalue Problem”, SIAM (1987).
- [23] J. Demmel, “Applied Numerical Linear Algebra”, SIAM (1997).
- [24] L. Trefethen and D. Bau, III, “Numerical Linear Algebra”, SIAM (1997).
- [25] W. Press, S. Teukolsky, W. Vetterling and B. Flannery, “Numerical Recipes: The Art of Scientific Computing”, 3rd ed., Cambridge University Press (2007).
- [26] G. Golub and C. Van Loan, “Matrix Computations”, 4th ed., Johns Hopkins University Press (2012).
- [27] 山本有作, “密行列固有値解法の最近の発展(I) : Multiple Relatively Robust Representationsアルゴリズム”, 日本応用数理学会論文誌, Vol.15, No.2, pp.181–208 (2005).
- [28] 山本有作, “密行列固有値解法の最近の発展(II) : マルチシフトQR法”, 日本応用数理学会論文誌, Vol.16, No.4, pp.507–534 (2006).
- [29] ScaLAPACK, <http://www.netlib.org/scalapack/>
- [30] ELPA, <http://elpa.rzg.mpg.de/>
- [31] DPLASMA, <http://icl.cs.utk.edu/dplasma/>
- [32] J. Cuppen, “A divide and conquer method for the symmetric tridiagonal eigenproblem”, Numer. Math, Vol.36, pp.177–195 (1981).
- [33] P. Arbenz, “Divide and conquer algorithms for the bandsymmetric eigenvalue problem”, Parallel Computing, Vol.18, No.10, pp.1105–1128 (1992).