Rokko チュートリアル

第 62 回お試しアカウント付き講習会「ライブラリ利用:科学技術計算の効率化入門

2016-09-07

講習資料: https://github.com/cmsi/rokko-tutorial/releases/latest

Rokko チュートリアル

- 講習会担当
  - 五十嵐亮 (東大情基セ) rigarash@cc.u-tokyo.ac.jp
- 主催
  - 東京大学情報基盤センター
    http://www.itc.u-tokyo.ac.jp/

チュートリアルの流れ

- 座学: 固有値問題の解法・固有値ソルバ/線形計算ライブラリ
- 座学: Rokko の概要と内部構造 (15 分)
- 実習: Rokko のインストール
- 座学・実習: 密行列向け MPI 並列ソルバの基本概念、サンプル実行
- 座学・実習: 疎行列向け MPI 並列ソルバの基本概念、サンプル実行
- 実習: 量子スピン系の対角化のサンプル実行

固有値問題の解法・固有値ソルバ/線形計算ライブラリ

- 行列の対角化
- 固有値問題の解法
- 固有値ソルバ/線形計算ライブラリ

行列の対角化

- 行列の種類
  - 実対称行列, 実非対称行列, エルミート行列, 非エルミート行列
- 行列の格納方法
  - 密行列, CRS(Compressed Row Storage) 形式, MatFree 形式 (それぞれ TITPACK2 の「小規模」, 「中規模」, 「大規模」に対応)
  - H 行列
- 必要な固有値
  - 全て, 絶対値の大きな (小さな) 順にいくつか, ある範囲内
- 固有ベクトル
  - 要/不要

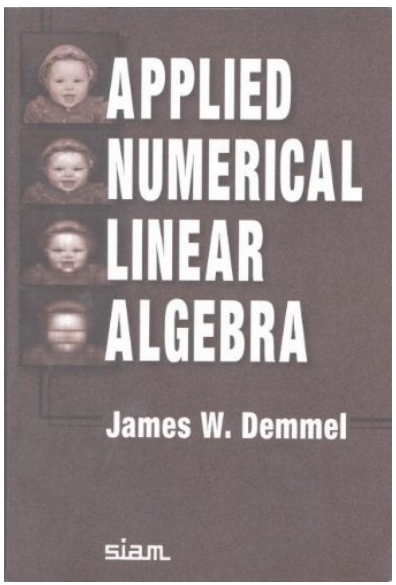
用語の定義

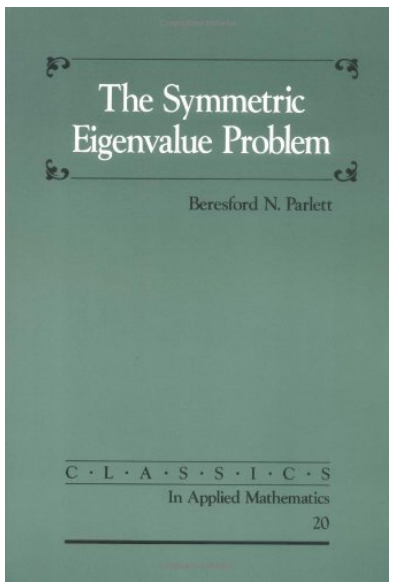
- 固有値問題の解法 (Eigenvalue algorithm)
  - 固有値問題を解くためのアルゴリズム
- 固有値ソルバ (Eigensolver, Eigenvalue problem solver)
  - 固有値解法の実装
- 固有値ソルバライブラリ (Eigensolver Library)
  - 固有値ソルバのみを含むライブラリ
- 線形計算ライブラリ (Linear Algebra Library)
  - 固有値ソルバや他の線形計算ソルバの集合体
- 厳密対角化パッケージ (Exact diagonalization package)
  - 量子格子模型のハミルトニアンの固有値問題を扱うソフトウェア

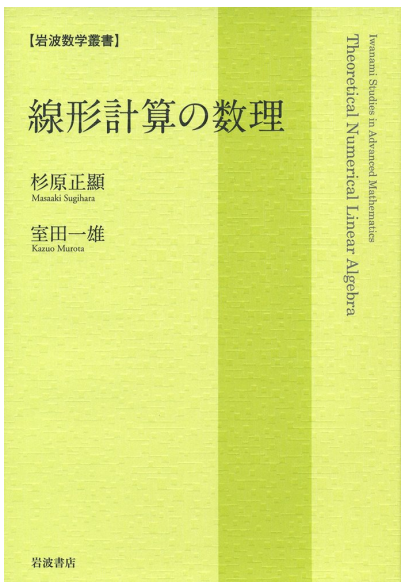
固有値問題の解法 (一部)

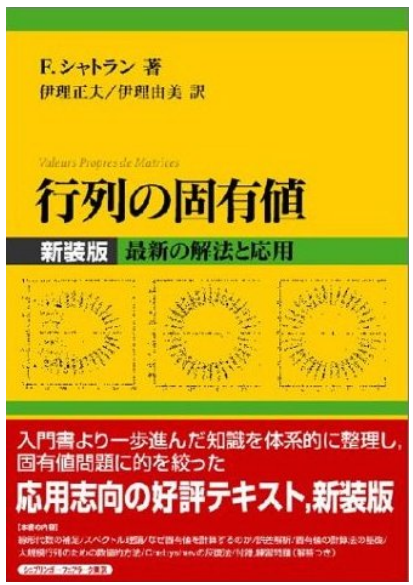
- 三重対角行列に対する固有値問題の解法
  - 二分法, QR 法, MR3, 分割統治法+QR 法
- 密行列の直接対角化
  - Jacobi 法
- 密行列の三重対角化
  - Householder 法
- 疎行列の直接対角化
  - べき乗法, 逆べき乗法, レイリー商反復法, Jacobi-Davidson 法, LOBPCG, Krylov-Schur 法
- 疎行列の三重対角化
  - Lanczos 法, Arnoldi 法, リスタート付き Lanczos 法 (Restart Lanczos), Thick-restart Lanczos 法
- その他の方法
  - Sakurai-Sugiura 法

固有値問題の解法


APPLIED
NUMERICAL
LINEAR
ALGEBRA
James W. Demmel
siam


The Symmetric
Eigenvalue Problem
Beresford N. Parlett
C・L・A・S・S・I・C・S
In Applied Mathematics
20


【岩波数学叢書】
線形計算の数理
杉原正顯
Masaaki Sugihara
室田一雄
Kazuo Murota
Iwanami Studies in Advanced Mathematics
Theoretical Numerical Linear Algebra
岩波書店


F. シャトラン 著
伊理正夫/伊理由美 訳
Valeurs Propres de Matrices
行列の固有値
新装版 最新の解法と応用
入門書より一歩進んだ知識を体系的に整理し,
固有値問題に的を絞った
応用志向の好評テキスト,新装版

固有値ソルバライブラリ（密行列向け）

- EigenExa: 密行列ソルバ
  - Householder (3 重対角化, 5 重対角化)+分割統治法+QR
- ELPA
  - Householder+分割統治法+QR

固有値ソルバライブラリ（疎行列向け）

- Anasazi: 反復法ソルバ中心
  - Krylov-Schur, Jacobi-Davidson, XXX-Davidson, LOBPCG, Implicit Riemannian Trust Region Method
- ARPACK
  - Implicit Restarted Lanczos
- BLOPEX
  - Locally Optimal Block Preconditioned Conjugate Gradient Method (LOBPCG)
- SLEPc: 反復法ソルバ中心, ビルド時に逐次か MPI 並列かを選ぶ必要あり
  - Krylov-Schur, Generalized Davidson, Jacobi-Davidson, Rayleigh Quotient Conjugate Gradient, Contour integral Sakurai-Sugiura, Power method, Subspace Itertation, Arnoldi (explicit restart), Lanczos (explicit restart)
- IETL: ALPS に含まれる反復法ソルバ
  - Lanczos, 他

線形計算ライブラリ（密行列向け）

- LAPACK, ScaLAPACK のベンダ実装
  - Apple Accelerate Framework(BLAS/vecLib): LAPACK
  - Fujitsu SSLII: LAPACK, ScaLAPACK(の一部), 他
  - Intel MKL: LAPACK, ScaLAPACK
  - ACML(AMD Core Math Library): LAPACK
  - OpenBLAS: BLAS+一部の LAPACK（GotoBLAS の後継）
- Netlib LAPACK: LAPACK のリファレンス実装
  - Householder+QR, Householder+分割統治法+QR, Householder+二分法, Householder+MR3
- Netlib ScaLAPACK: ScaLAPACK のリファレンス実装
  - Householder+QR, Householder+分割統治法+QR, Householder+二分法, Householder+MR3
- Eigen3（逐次版、行列・行列積のみスレッド並列化）
  - Householder+QR
- Elemental：含まれる固有値ソルバ MRRR は、プロセス数が平方数の場合のみ確実に動く
  - Householder+MR3

線形計算ライブラリ (疎行列向け)

- Trilinos：固有値ソルバライブラリ Anasazi を含む

おすすめ順

- 密行列・逐次
  - LAPACK (のベンダ実装) > Eigen3
- 密行列・MPI 並列
  - EigenExa > ELPA > ScaLAPACK (のベンダ実装) > Elemental
- 疎行列（逐次、MPI 並列）
  - Anasazi > SLEPc

EigenExa による超巨大密行列の対角化

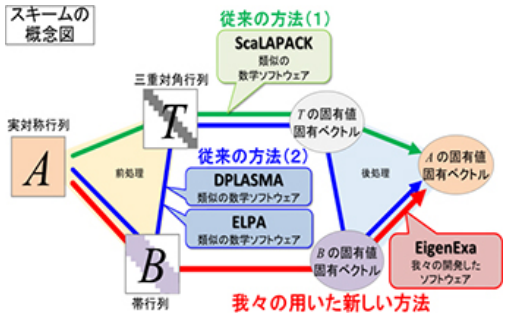
「京」を使い世界最高速の固有値計算に成功
ー超巨大行列の固有値を1時間で計算ー

報道発表資料

高校時代の数学の先生いわく「こんなにエレガントな学問は数学のほかに無いと思わないか」。公式通りに計算していけばスッキリとした答えが出るので、そうかなあと思っていましたが、行列とベクトルで挫折した苦い経験があります。もう出会わないと高をくくっていたら、固有値計算の研究成果のサマリーを書くことに…。

行列の計算では、行列の固有値を求めること、固有ベクトルを求めること、が求められます。「固有値、固有ベクトルって何？」ということになるのですが、ごく単純化するとこうなります。

ある行列Aが与えられている時、そのAに対して「$Ax=\lambda x$」という式があるとき、それを成立させるλの値が固有値で、xの値が固有ベクトルになります。固有ベクトルと固有値は密接につながり、固有ベクトルの大きさは固有値の倍数になります。つまり、固有ベクトルごとに分けて考えれば力や方向の問題が単純化されて扱いやすくなります。



従来の方法（緑線、青線）と、eigenexaの方法（赤線）との関係

計算科学の分野では、量子物理や量子科学の現象を行列で表し、その固有値を求める（行列の対角化ともいいます）ことによって問題を解いています。この手法は新しい電子材料の発見や新薬の設計などのための大規模なコンピュータシミュレーションやデータの相関関係の解析に使われています。しかし、行列の固有値計算では計算量が行列の次元の3乗に比例して増加するため、これまでのコンピュータでは能力不足でした。一方で、理研のスーパーコンピュータ「京」のような超並列スパコン向けの効率的な固有値計算用の数学ソフトも存在しませんでした。

行列の固有値計算では行列を簡単な形式（形状）に変換し、それを中間形式として取り扱います。理研の研究チームは、帯行列（ゼロでない要素が対角線上に帯状に分布する行列）を中間形式に採用することによって、前処理の時間の削減を図った新しい計算アルゴリズムを考案し、それを基にした数学ソフト「EigenExa（アイゲンエクサ）」を開発しました。「京」の全プロセッサを用いて計算した結果、世界最大規模の100万×100万の行列の固有値計算が1時間以内で可能なこと確認しました。これまでの地球シミュレーターの記録（40万×40万の行列で3時間半）を大幅に上回りました。「京」の高い計算能力とEigenExaの利用により、数十万から100万程度の固有値を求める問題は1時間以内にできることが立証されました。今後、シミュレーションの規模を大幅に拡大することが可能になります。なお、EigenExaはオープンソフトウエアとして公開され、理研計算科学研究機構研究部門のホームページからダウンロードできます。

独立行政法人理化学研究所
計算科学研究機構 研究部門 大規模並列数値計算技術研究チーム
チームリーダー 今村 俊幸 （いまむら としゆき）

最新の固有値ソルバ

- ハイブリッド並列 (MPI+OpenMP) 対応のソルバも増えてきた
- 超並列環境に対応した並列ソルバをユーザ自身が実装・最適化するのはもはや不可能
- 最新の並列固有値ソルバを積極的に活用すべき

物性理論向け厳密対角化パッケージの状況

- 代表的なパッケージ
  - ALPS (fulldiag / sparsediag): LAPACK, IETL を利用
  - KOBEPACK: 独自の固有値ソルバを実装
  - SPINPACK: LAPACK を利用
  - TITPACK2: 独自の固有値ソルバを実装
  - Hϕ: 独自の並列固有値ソルバを実装 (おすすめ)
- 逐次 or スレッド並列のみ, MPI 並列なし
- (Hϕ を除き) 最新の超並列スパコン環境を活かしきれていない

固有値解法とソルバ

- GitHub Wiki に, 固有値問題の解法, 固有値ソルバ, 固有値ソルバ/線形計算ライブラリ, 厳密対角化パッケージに関するリファレンス・マニュアルを作成中
  - 固有値解法とソルバ: https://github.com/t-sakashita/rokko/wiki/EigenvalueAlgorithms
- ボランティア募集中!

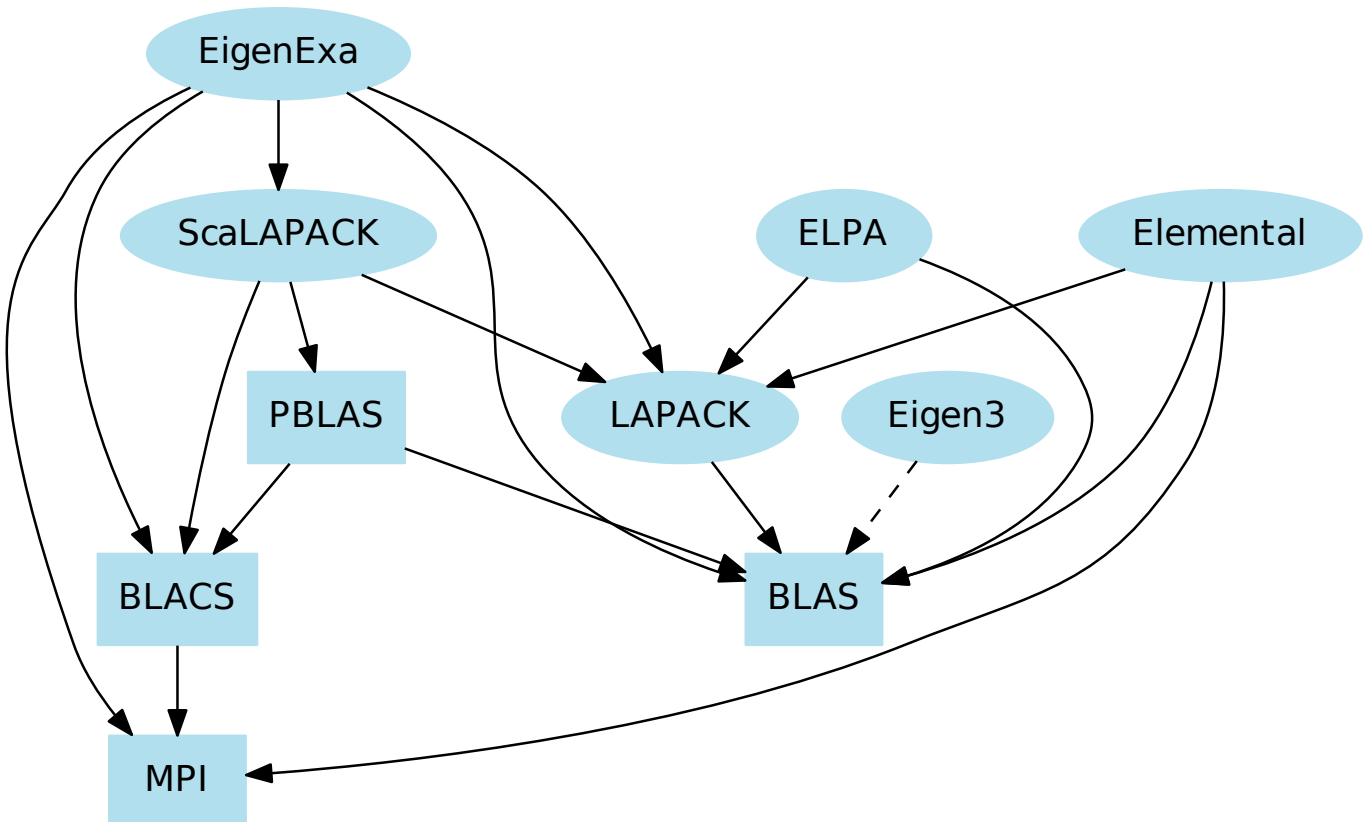
Rokko の概要と内部構造

- 既存のライブラリの問題点
- Rokko の概要
- 並列ソルバの基本概念
- Rokko の内部構造

既存のライブラリの問題点

- ソルバ毎に異なるデザイン
- インストール方法もライブラリ毎に異なる
- ドキュメントが不十分な場合も多い
- コンピュータのアーキテクチャ毎に異なるコンパイル・リンクオプションが必要
- C++/C/Fortran 相互リンクの問題
- ライブラリ間の依存関係が複雑
- 実際に試す前に大まかな性能比較が欲しい

ライブラリ間の依存関係（密行列向け）



Rokko の開発者

- 坂下達哉 (名大工) tatsuya.sakashita@apchem.nagoya-u.ac.jp
- 五十嵐 亮 (東大情基セ) rigarash@cc.u-tokyo.ac.jp
- 本山裕一 (東大物性研) y-motoyama@issp.u-tokyo.ac.jp
- 大久保 毅 (東大物性研) t-okubo@issp.u-tokyo.ac.jp
- 藤堂眞治 (東大院理/物性研) wistaria@phys.s.u-tokyo.ac.jp

Rokko の概要

- 使用言語
  - コア部分: C++
  - 言語バインディング: C, Fortran90
  - ベンチマークスクリプト: Python
- ライセンス
  - Boost ライセンス (ほぼ自由に使える)
- ソースコード
  - GitHub で公開
    https://github.com/t-sakashita/rokko/

Rokko の設計方針

- 共通のベクトルや行列クラス
- ソルバの違いはラッパーで吸収
  - 個々のソルバに仕様変更があっても Rokko が吸収
- 再コンパイルなしに, 実行時にソルバを選択可能
- 仮想関数とテンプレートを組み合わせることで, オーバーヘッドの少ないラッパーを実装
- C++, C, Fortran90 から使用可能に

Rokko の全体像

- 固有値ソルバ/線形演算ライブラリのインストールスクリプト
- 共通基本クラス (分散行列, プロセスグリッド他)
- 固有値ソルバラッパー (C++)
- 固有値ソルバ・ファクトリ (C++)
- C/Fortran ラッパー
- テスト・サンプルプログラム
- ベンチマークスクリプト

サードパーティーの固有値ソルバ/線形計算ライブラリのインストール

- Eigen3 (ベクトル, 行列クラスを内部で利用), LAPACKE (LAPACK の C インターフェース) は, Rokko に同梱されている
- インストールスクリプト: 3rd-party/install
  - ライブラリ: Anasazi, EigenExa, Elemental, ELPA, PETSc, ScaLAPACK, SLEPc
  - 対応アーキテクチャ: 京/FX10, x86 スパコン・クラスタ (Intel コンパイラ/GCC), Mac OS X (GCC), 他

Rokko のインストール

- 下準備 (reedbush-u.cc.u-tokyo.ac.jp の場合)

```
echo "export LUSTRE=/lustre/gt00/t002XX" >> $HOME/.bash_profile
echo "module load boost/1.61" >> $HOME/.bash_profile
echo "module unload intel-mpi && module load mpt" >> $HOME/.bash_profile
echo "PREFIX_ROKKO=/lustre/gt00/t002XX/rokko" >> $HOME/.rokkoinstaller
echo "   BUILD_DIR=/lustre/gt00/t002XX/build" >> $HOME/.rokkoinstaller
echo "  SOURCE_DIR=/lustre/gt00/share/source" >> $HOME/.rokkoinstaller
. $HOME/.bash_profile
```

- ソースコードのダウンロード・展開

```
cd $LUSTRE && mkdir rokko && mkdir build
mkdir src && cd src
git clone https://github.com/t-sakashita/rokko.git
```

- CMake の実行 ()

```
mkdir rokko-build && cd rokko-build && . /lustre/gt00/share/rokko/trillinos-11.14.2-2/
      trillinosvars.sh
cmake $LUSTRE/src/rokko -DCMAKE_CXX_COMILER=icpc -DCMAKE_C_COMPILER=icc \
 -DCMAKE_Fortran_COMPILER=ifort -DCMAKE_INSTALL_PREFIX=$LUSTRE/rokko \
 -DBoost_INCLUDE_DIR=/lustre/app/boost/1.61/include
```

Rokko のインストール（続き）

- Make, テスト

```
make -j 4
make test
make install
```

インストールの成否の確認

Rokko で使用できる固有値ルーチン一覧の表示 tool/rokko_solvers.cpp

```
cd rokko/tool
./rokko_solvers
```

出力結果

```
[serial dense solvers]
  eigen3
  lapack
[parallel dense solvers]
  scalapack
[parallel sparse solvers]
  anasazi
```

Rokko のディレクトリ構造

- rokko ··· Rokko 本体（ヘッダファイル・ソースファイル）
- example ··· サンプルプログラム
  - cxx
    - dense ··· 密行列用（逐次版、MPI 版）
    - sparse ··· 疎行列用（MPI 版）
  - c
    - dense ··· 密行列用（逐次版、MPI 版）
    - sparse ··· 疎行列用（MPI 版）
  - fortran
    - dense ··· 密行列用（逐次版、MPI 版）
    - sparse ··· 疎行列用（MPI 版）
- tutorial ··· チュートリアル
  - titpack ··· 厳密対角化パッケージの書き換え

rokko::localized_matrix クラステンプレート

rokko/localized_matrix.hpp

```
namespace rokko {
template<typename T, typename MATRIX_MAJOR = rokko::matrix_col_major>
class localized_matrix {
public:
  typedef MATRIX_MAJOR major_type;
  localized_matrix();
  localized_matrix(int rows, int cols);
  template <typename U>
  localized_matrix(U const& other);
  template <typename U>
  matrix_type& operator=(U const& other);
  double operator[](int i, int j) const;
  double& operator[](int i, int j);
  int get_m_global() const;
  int get_n_global() const;
  int get_m_local() const;
  int get_n_local() const;
  bool is_gindex_myrow(const int& global_i) const;
};
}
```

rokko::localized_vector と rokko::localized_matrix の利用例

- test/localized_matrix.cpp

```
int dim = 3;
rokko::localized_matrix<double> M(dim,dim);
M << 1,2,3,4,5,6,7,8,9;
double a = 5.0;
rokko::localized_vector u(dim);
u << 1,2,3;
rokko::localized_vector v(dim);
v << 4,5,6;
rokko::localized_vector w = a*u+M*v;
```

rokko::serial_dense_solver クラス

- ソルバの初期化

```
rokko::serial_dense_solver solver(name);
solver.initialize(argc, argv);
```

- ソルバの終了

```
solver.finalize();
```

- 対角化 (行列は破壊される)

```
rokko::localized_matrix<double, matrix_col_major> mat(dim, dim);
...
rokko::localized_vector evals(dim);
rokko::localized_matrix<double, matrix_col_major> evecs(dim, dim);
solver.diagonalize(mat, eigval, eigvec, params);
```

対角化のサンプル

- LAPACK dsyev を直接使う example/cxx/dense/dsyev.cpp

```
./example/cxx/dense/dsyev
```

逐次密行列ソルバを使う

- C++版 example/cxx/dense/frank.cpp

```
./example/cxx/dense/frank lapack 5
```

- C 版 example/c/dense/frank.c

```
./example/c/dense/frank lapack 5
```

- Fortran 版 example/fortran/dense/frank.f90

```
./example/fortran/dense/frank lapack 5
```

固有値ソルバへのパラメータの渡し方

- パラメータクラス rokko/parameters.hpp を利用
  - example/cxx/dense/frank.cpp の例：

```
rokko::parameters params;
params.set("routine", "tri");
params.set("upper_value", 1.2);
params.set("lower_value", 0.1);
params.set("uplow", "lower");
params.set("verbose", true);
```

密行列逐次ソルバに渡すパラメータ

Eigen3

Rokko ルーチン名	元のルーチン名	解法	入力パラメータ（オプション）	出力パラメータ	備考
eigen3:qr	SelfAdjointEigensolver	QR 法	なし	なし	デフォルト

密行列逐次ソルバに渡すパラメータ

LAPACK

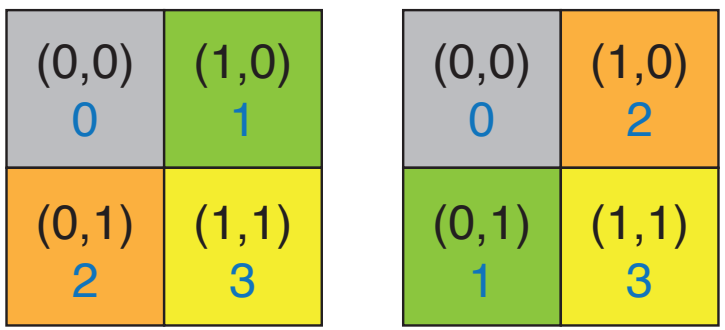
Rokko ルーチン名	元のルーチン名	解法	入力パラメータ（オプション）	出力パラメータ	備考
lapack:dsyev lapack:qr	dsyev	QR 法	uplow	なし	引数なしのデフォルト
lapack:dsyevx	dsyevx	2 分法 QR 法	upper と lower の組, abstol, uplow	m, ifail	*abstol* >0 のとき、2 分法, *abstol* <0 のとき、QR 法
lapack:bisection	dsyevx	2 分法	upper と lower の組, abstol, uplow	m, fail	dsyevx を *abstol* >0 で使用
lapack:dsyevr lapack:mr3	dsyevr	MR3 法	upper と lower の組, abstol, uplow	m, isuppz	固有値範囲指定ありのデフォルト
lapack:dsyevd lapack:dc	dsyevd	分割統治法	uplow	なし	

LAPACK 向け入力パラメータ

Rokko の パラメータ名	ScaLAPACK の パラメータ名	型	意味	値	備考
matrix_part	uplow	std::string	ソルバに読まれる部分 （上三角/下三角）	upper(U), lower(L)	最初の一文字で判断。 大文字、小文字のどちらも可。 デフォルトは、'U'
abstol	abstol	double	固有値の収束判定に用いる		
upper_index	iu	int	求めたい固有値の番号の上限		番号は昇順につけられる
upper_value	vu	double	求めたい固有値の範囲の上限		
lower_index	il	int	求めたい固有値の番号の下限		番号は昇順につけられる
lower_value	vl	double	求めたい固有値の範囲の下限		
verbose	なし	bool	Rokko で用意したフラグ。 ソルバに設定した入力パラメータ、エラーの詳細を表示。		デフォルトは false

2 次元プロセスグリッド

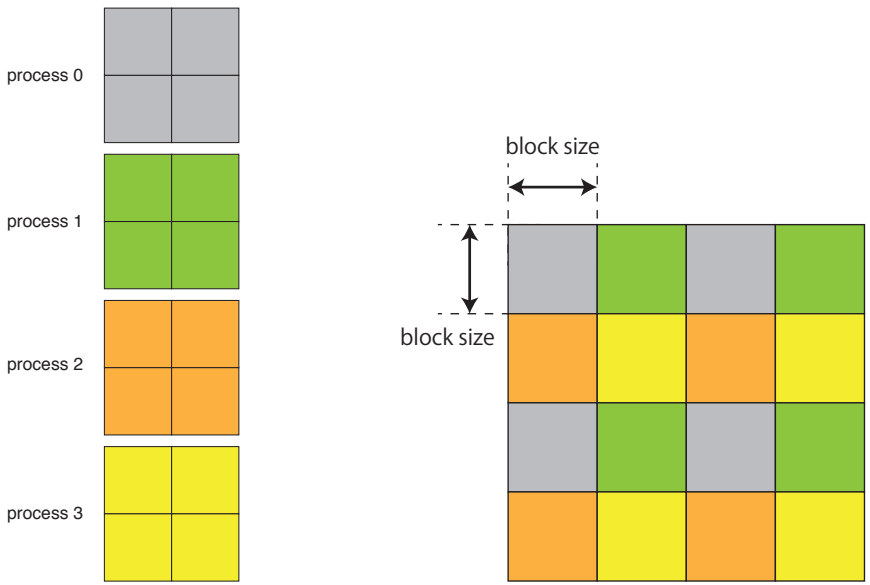
- MPI プロセスの 2 次元割付けを指定
- Row-major と column-major の二種類
- 4 プロセスの例 (左:row-major, 右:column-major)



- 全てのソルバは両方の種類をサポート

2 次元ブロック・サイクリック行列 (distributed_matrix)

- ほとんどの並列固有値ソルバにおいて, 密行列は「2 次元ブロック・サイクリック形式」でデータ分割される
- 4 プロセスでの例 (左 ローカルビュー, 右 グローバルビュー)



- ScaLAPACK と ELPA は任意のブロックサイズをサポート
- EigenExa と Elemental は 1×1 のみをサポート

rokko::grid クラス

rokko/grid.hpp

```
namespace rokko {
extern struct grid_row_major_t {} grid_row_major;
extern struct grid_col_major_t {} grid_col_major;
class grid {
public:
  explicit grid(MPI_Comm comm_in = MPI_COMM_WORLD);
  template <typename GRID_MAJOR>
  grid(MPI_Comm comm_in, GRID_MAJOR const& grid_major);
  MPI_Comm get_comm() const { return comm; }
  int get_nprocs() const;
  int get_nprow() const;
  int get_npcol() const;
  int get_myrank() const;
  int get_myrow() const;
  int get_mycol() const;
  bool is_row_major() const;
  bool is_col_major() const;
  int calculate_grid_row(int proc_rank) const;
  int calculate_grid_col(int proc_rank) const;
};
}
```

rokko::distributed_matrix クラステンプレート

rokko/distributed_matrix.hpp

```
namespace rokko {
template<typename T, typename MATRIX_MAJOR = rokko::matrix_col_major>
class distributed_matrix {
public:
  template<typename SOLVER>
  distributed_matrix(mapping_bc<MATRIX_MAJOR> const& map_in);
  template<typename SOLVER>
  ...
  bool is_gindex_myrow(int global_i) const; // プロセスグリッドのうち、自プロセスの行方向の番号を返す
  bool is_gindex_mycol(int global_j) const; // プロセスグリッドのうち、自プロセスの列方向の番号を返す
  bool is_gindex(int global_i, int global_j) const; // 自プロセスが (global_i, global_j)を持っているか？
  void set_local(int local_i, int local_j, double value);  // 自プロセスのlocal storage の(local_i, local_j)に値value を格納
  double get_local(int local_i, int local_j) const;  // 自プロセスのlocal storage の(local_i, local_j)の値を返す
  void update_local(int local_i, int local_j, double value); // 自プロセスのlocal storage の(local_i, local_j)に値value を足し込む
  void set_global(int global_i, int global_j, double value); // 自プロセスが (global_i, global_j)を受け持っていたら、値value を格納
  double get_global(int global_i, int global_j) const; // 自プロセスが (global_i, global_j)を受け持っていたら、その値を返す
};
}
```

distributed_matrix の操作

- 行列の掛け算 $C = \alpha AB + \beta C$

```
...
rokko::mapping_bc<matrix_major> map(dim, g, solver);  // 正方行列と与えられた
    grid と solver に適切なブロックサイズを返す。
rokko::distributed_matrix<double, rokko::matrix_col_major> matA(map);
rokko::distributed_matrix<double, rokko::matrix_col_major> matB(map);
rokko::distributed_matrix<double, rokko::matrix_col_major> matC(map);
...
rokko::product(alpha, matA, transA, matB, transB, beta, matC);
```

- distributed_matrix の scatter と gather

```
rokko::localized_matrix<double, LOC_MAT_MAJOR> lmat(dim, dim);
rokko::distributed_matrix<double, DIST_MAT_MAJOR> mat(map);
rokko::scatter(lmat, mat, root);
rokko::gather(mat, lmat, root);
```

distributed_matrix の生成

- 要素毎に代入 (global な添字)

```
for(int global_i=0; global_i < mat.get_m_global(); ++global_i) {
  for(int global_j=0; global_j < mat.get_n_global(); ++global_j) {
    mat.set_global(global_i, global_j, f(global_i, global_j));
  }
}
```

- 要素毎に代入 (local な添字)

```
for(int local_i = 0; local_i < mat.get_m_local(); ++local_i) {
  for(int local_j = 0; local_j < mat.get_n_local(); ++local_j) {
    int global_i = mat.translate_l2g_row(local_i);
    int global_j = mat.translate_l2g_col(local_j);
    mat.set_local(local_i, local_j, f(global_i, global_j));
  }
}
```

- 関数の値で行列を埋める

```
mat.generate(f);
```

rokko::parallel_dense_solver クラス

- ソルバの初期化

```
rokko::parallel_dense_solver solver(name);
solver.initialize(argc, argv);
```

- ソルバの終了

```
solver.finalize();
```

- 対角化 (行列は破壊される)

```
rokko::mapping_bc<double, matrix_major> map(dim, g, solver);
rokko::distributed_matrix<double, matrix_col_major> mat(map);
...
rokko::localized_vector evals(dim);
rokko::distributed_matrix<double, matrix_col_major> evecs(map);
solver.diagonalize(mat, evals, evecs, params);
```

対角化のサンプル

- C++ example/cxx/dense/frank_mpi.cpp

```
mpirun -np 4 ./example/cxx/dense/frank_mpi eigen_exa 5
```

- C 版 example/c/dense/frank_mpi.cpp

```
mpirun -np 4 ./example/c/dense/frank_mpi eigen_exa 5
```

- Fortran 版 example/fortran/dense/frank_mpi.cpp

```
mpirun -np 4 ./example/fortran/dense/frank_mpi eigen_exa 5
```

テスト行列：Frank 行列（定義）

定義

$$[a_{ij}]_{i,j=0,\ldots,n-1} = [n - \max(i,j)]$$

Example ($n = 5$)

$$[a_{ij}]_{i,j=0,\ldots,4} = \begin{bmatrix} 5 & 4 & 3 & 2 & 1 \\ 4 & 4 & 3 & 2 & 1 \\ 3 & 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

テスト行列：Frank 行列（性質）

理論固有値

$$\lambda_k = \frac{1}{2\left(1 - \cos\frac{2k+1}{2n+1}\pi\right)} \quad (k = 0, \ldots, n-1)$$

Frank matrix has an eigenvalue 1
$\iff \frac{2k+1}{2n+1} = \frac{1}{3}$
$\iff n - 1$ is a multiple of 3.

密行列 MPI 並列ソルバに渡すパラメータ

ScaLAPACK 向けルーチン

Rokko ルーチン名	元のルーチン名	解法	入力パラメータ（オプション）	出力パラメータ	備考
scalapack:pdsyev scalapack:qr	pdsyev	QR 法	uplow	なし	引数なしのデフォルト
scalapack:pdsyevx	pdsyevx	2 分法 QR 法	upper と lower の組, abstol, uplow	m, ifail	*abstol* > 0 のとき、2 分法, *abstol* < 0 のとき、QR 法
scalapack:bisection	pdsyevx	2 分法	upper と lower の組, abstol, uplow	m, ifail	pdsyevx を *abstol* > 0 として使用
scalapack:pdsyevr scalapack:mr3	pdsyevr	MR3 法	upper と lower の組, abstol, uplow, orfac	m, nz, ifail, iclustr, gap	固有値範囲指定ありのデフォルト
scalapack:pdsyevd scalapack:dc	pdsyevd	分割統治法	uplow	なし	

ScaLAPACK 向け入力パラメータ

Rokko の パラメータ名	ScaLAPACK の パラメータ名	型	意味	値	備考
matrix_part	uplow	std::string	ソルバに読まれる部分 （上三角/下三角）	upper(U), lower(L)	最初の一文字で判断。 大文字、小文字のどちらも可。 デフォルトは、'U'
abstol	abstol	double	固有値の収束判定に用いる		
upper_index	iu	int	求めたい固有値の番号の上限		番号は昇順につけられる
upper_value	vu	double	求めたい固有値の範囲の上限		
lower_index	il	int	求めたい固有値の番号の下限		番号は昇順につけられる
lower_value	vl	double	求めたい固有値の範囲の下限		
verbose	なし	bool	Rokko で用意したフラグ。 ソルバに設定した入力パラメータ、エラーの詳細を表示。		デフォルトは false

密行列 MPI 並列ソルバに渡すパラメータ

EigenExa

Rokko ルーチン名	元のルーチン名	解法	入力パラメータ（オプション）	出力パラメータ	備考
eigen_exa:tri eigen_exa:eigen_s	eigen_s	分割統治法（３重対角化を経由）	m_forward, m_backward	なし	
eigen_exa:penta eigen_exa:eigen_sx	eigen_sx	分割統治法（５重対角化を経由）	m_forward, m_backward	なし	デフォルト

CRS(Compressed Row Storage) 方式

各行ごとに非ゼロ成分とその列添字を格納する方法

例

$$\begin{bmatrix} 7.1 & 5.2 & 0 & 0 \\ 0 & 0 & 0 & 6.4 \\ 0.2 & 0 & 0 & 4.3 \\ 0 & 0 & 0.5 & 0 \end{bmatrix}$$

CRS 方式による表現：

- 各行ごとの非ゼロ成分の個数 $num\_nonzero\_cols = [2, 1, 2, 1]$
- 非ゼロ成分に対する列添字 $nonzero\_cols = [1, 2, 4, 1, 4, 3]$
- 非ゼロ行列成分 $values = [7.1, 5.2, 6.4, 0.2, 4.3, 0.5]$

MPI 並列化は行単位

rokko::distributed_crs_matrix クラステンプレート

```
namespace rokko {
class distributed_crs_matrix {
public:
  template<typename SOLVER>
  distributed_crs_matrix(int row_dim, int col_dim, SOLVER& solver_in)
  void insert(int row, std::vector<int> const& cols, std::vector<double> const& values);
  void insert(int row, int col_size, int* cols, double* const values);
  void complete();
  int get_dim() const;
  int num_local_rows() const;
  int start_row() const;
  int end_row() const;
  void print() const;
};
}
```

rokko::distributed_crs_matrix クラス（使用例）

sample/sparse/distributed_crs_matrix.cpp

```
rokko::parallel_sparse_solver solver("anasazi");
int dim = 4;
rokko::distributed_crs_matrix mat(dim, dim, solver);

int num_nonzero_cols[] = {2, 1, 2, 1};
int nonzero_cols[] = {0, 1, 3, 0, 3, 2};
double values[] = {7.1, 5.2, 6.4, 0.2, 4.3, 0.5};

int current = 0;
for (int row = 0; row < dim; ++row) {
  mat.insert(row, num_nonzero_cols[row], &nonzero_cols[current], &values[current]);
  current += num_nonzero_cols[row];
}
mat.complete();
mat.print();
```

サンプルの実行

- Anasazi の場合

```
mpirun -np 4 ./exmple/sparse/cxx/distributed_crs_matrix anasazi
```

- SLEPc の場合

```
mpirun -np 4 ./example/sparse/cxx/distributed_crs_matrix slepc
```

Matrix free 方式

たいていの疎行列向け固有値ソルバでは、行列そのものではなく、固有値分解を行うべき行列とベクトル積を行うルーチンのみが必要である。

作り方

- rokko::distributed_mfree を継承したクラスを用意する。
- そのクラスの中で、
  - void multiply(const double* x, double* y) 関数
  - void diagonal(double* x) 関数 （SLEPc の場合）

  というメンバ関数を用意する。

Matrix free 方式（例）

```
class heisenberg_op : public rokko::distributed_mfree {
public:
  heisenberg_op(int L, const std::vector<std::pair<int, int> >& lattice) : L_(L), lattice_(
      lattice) {
    comm_ = MPI_COMM_WORLD;
    int nproc;
    MPI_Comm_size(comm_, &nproc);
    int n = nproc;
    int p = -1;
    do {
      n /= 2;
      ++p;
    } while (n > 0);
    local_N = 1 << (L-p);
    buffer_.assign(local_N, 0);
    dim_ = 1 << L;
  }
  void multiply(const double* x, double* y) const {
    rokko::heisenberg_hamiltonian::multiply(comm_, L_, lattice_, x, y, &(buffer_[0]));
  }
  int get_dim() const {
    return dim_;
  }
  int get_num_local_rows() const {
    return local_N;
  }
```

Matrix free 方式（例の続き）

```
private:
  MPI_Comm comm_;
  mutable std::vector<double> buffer_;
  int L_;
  int local_N;
  std::vector<std::pair<int, int> > lattice_;
  int dim_;
};
```

rokko::parallel_sparse_solver クラス

- ソルバの初期化

```
rokko::parallel_sparse_solver solver(name);
solver.initialize(argc, argv);
```

- ソルバの終了

```
solver.finalize();
```

- 対角化（CRS 行列の場合）

```
rokko::distributed_crs_matrix mat(dim, dim, solver);
...
パラメータparams の設定
solver.diagonalize(mat, params);
```

- 対角化（Matrix Free の場合）

```
heisenberg_op  mat(L, lattice);
パラメータparams の設定
solver.diagonalize(mat, params);
```

rokko::parallel_sparse_solver クラス（続）

- パラメータの設定法

```
rokko::parameters params;
params.set("Block Size", block_size);   // 必須
params.set("Maximum Iterations", max_iters);
params.set("Convergence Tolerance", tol);  // 必須
params.set("num_eigenvalues", nev);  // 必須
// 上記以外のパラメータ名は、ソルバライブラリ（Anasazi/SLEPc)と同じ。
params.set("Which", "LM");
params.set("routine", "lanczos");
```

- 固有値・固有ベクトルの取り出し

```
int i;
solver.eigenvalue(i);
std::vector<double> eigvec;
solver.eigenvector(i, eigvec);
```

テスト行列：ラプラシアン行列（Frank 行列の逆行列）

定義

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

テスト行列：ラプラシアン行列（Frank 行列の逆行列

理論固有値

$$\lambda_k = 2\left(1 - \cos\frac{2k+1}{2n+1}\pi\right) \quad (k = 0, \ldots, n-1)$$

Frank matrix has an eigenvalue 1
$\iff \frac{2k+1}{2n+1} = \frac{1}{3}$
$\iff n - 1$ is a multiple of 3.

CRS 行列の対角化

- C++ example/cxx/sparse/laplacian_crs_mpi.cpp

```
mpirun -np 4 ./example/cxx/sparse/laplacian_crs_mpi eigen_exa 5
```

- C 版 example/c/sparse/laplacian_crs_mpi.cpp

```
mpirun -np 4 ./example/c/sparse/laplacian_crs_mpi eigen_exa 5
```

- Fortran 版 example/fortran/sparse/laplacian_crs_mpi.cpp

```
mpirun -np 4 ./example/fortran/sparse/laplacian_crs_mpi eigen_exa 5
```

MatFree の対角化

- C++ example/cxx/sparse/laplacian_mfree_mpi.cpp

```
mpirun -np 4 ./example/cxx/sparse/laplacian_mfree_mpi eigen_exa 5
```

- C 版 example/c/sparse/laplacian_mfree_mpi.cpp

```
mpirun -np 4 ./example/c/sparse/laplacian_mfree_mpi eigen_exa 5
```

- Fortran 版 example/fortran/sparse/laplacian_mfree_mpi.cpp

```
mpirun -np 4 ./example/fortran/sparse/laplacian_mfree_mpi eigen_exa 5
```

量子 XYZ 模型

$$\mathcal{H} := \sum_{\langle i,j \rangle} \left(J_x^{\langle i,j \rangle} S_i^x \cdot S_j^x + J_y^{\langle i,j \rangle} S_i^y \cdot S_j^y + J_z^{\langle i,j \rangle} S_i^z \cdot S_j^z \right)$$

$\mathcal{H}$ は、局所ハミルトニアンの和で表せる:

$$\mathcal{H} = \sum_{\langle i,j \rangle} \mathcal{H}_{\langle i,j \rangle}$$

$$\text{where} \quad H_{\langle i,j \rangle} := \frac{1}{4} \begin{bmatrix} J_z & & & J_x - J_y \\ & -J_z & J_x + J_y & \\ & J_x + J_y & -J_z & \\ J_x - J_y & & & J_z \end{bmatrix}$$

量子スピン系の対角化（密行列、逐次・MPI 版）

- Heisenberg 模型（逐次版）
  example/cxx/sparse/heisenberg.cpp

```
./example/cxx/dense/heisenberg lapack
```

- Heisenberg 模型（MPI 版） example/cxx/dense/heisenberg_mpi.cpp

```
mpirun -np 4 ./example/cxx/dense/heisenberg_mpi eigen_exa
```

- XYZ 模型（逐次版） example/cxx/dense/xyz.cpp

```
./example/cxx/dense/xyz_dense lapack $HOME/rokko-0.3/test/input_data/xyz_1hexagon.ip
```

- XYZ 模型 (MPI 版) example/cxx/dense/xyz_mpi.cpp

```
mpirun -np 4 ./example/cxx/dense/xyz_mpi eigen_exa $HOME/rokko-0.3/test/input_data/
      xyz_1hexagon.ip
```

量子スピン系の対角化（疎行列、MPI 版、C++）

- Heisenberg 模型 (CRS) example/cxx/sparse/heisenberg_crs_mpi.cpp

```
mpirun -np 4 ./example/cxx/sparse/heisenberg_crs_mpi
```

- Heisenberg 模型 (MatFree) example/cxx/sparse/heisenberg_mfree_mpi.cpp

```
mpirun -np 4 ./example/cxx/sparse/heisenberg_mfree_mpi
```

- XYZ 模型 (CRS) example/cxx/sparse/xyz_crs_mpi.cpp

```
mpirun -np 4 ./example/cxx/sparse/xyz_crs_mpi
```

- XYZ 模型 (Matfree) example/cxx/sparse/xyz_mfree_mpi.cpp

```
mpirun -np 4 ./exmple/cxx/sparse/xyz_mfree_mpi
```

入力ファイル

test/input_data/heisenberg.ip

```
8 8    ←−−number of sites, number of bonds

0 1
1 2
2 3
3 4    ←−− bonds ⟨i, j⟩ (pairs of site numbers)
4 5
5 6
6 7
7 0

1.0 1.0 0.0
1.0 1.0 0.0
1.0 1.0 0.0
1.0 1.0 0.0
1.0 1.0 0.0    ←−− J_x^⟨i,j⟩, J_y^⟨i,j⟩, J_z^⟨i,j⟩ for each bond ⟨i, j⟩
1.0 1.0 0.0
1.0 1.0 0.0
1.0 1.0 0.0
```

CMake による Rokko ライブラリの取り込み方法

以下を参考にユーザプログラム用の CMakeLists.txt を書く。
その CMakeLists.txt に、Rokko のインストール先の share/rokko/UseRokko.cmake をインクルードする。

- C++
  - example/cxx/dense/CMakeLists.txt
  - example/cxx/sparse/CMakeLists.txt
- C
  - example/c/dense/CMakeLists.txt
  - example/c/sparse/CMakeLists.txt
- Fortran
  - example/fortran/dense/CMakeLists.txt
  - example/fortran/sparse/CMakeLists.txt

Makefile による Rokko ライブラリの取り込み方法

ユーザプログラムの Makefile に share/rokko/include.mk をインクルードする。

一般化固有値問題を標準固有値問題２個に分解して解く

一般化固有値問題 $Ax = \lambda Bx$　(where A: 対称行列，$B > 0$)
両辺に、$B^{-1/2}$ を左から掛ける。

$$B^{-1/2}A(B^{-1/2}B^{1/2})x = \lambda B^{1/2}x \tag{1}$$

$A' := B^{-1/2}AB^{-1/2}$, $x' := B^{1/2}x$ とおくと、

$$A'x' = \lambda x' \tag{2}$$

対称標準固有値問題に帰着する。
手順

1. B を固有値分解し、$B^{-1/2}$ を計算する。このとき、0 に近い固有値は逆数をとれないので、その逆数は 0 とする。
2. $A' := B^{-1/2}AB^{-1/2}$ を計算する。
3. 標準固有値問題 $A'x' = \lambda x'$ を解く。
4. 一般化固有値問題 $Ax = \lambda Bx$ の固有値は λ、固有ベクトルは $x = B^{-1/2}x'$ となる。

サンプルプログラムの実行

- C++

```
mpirun -np 4 ./example/cxx/dense/gev_fixedB_mpi eigen_exa
```

- C

```
mpirun -np 4 ./example/c/dense/gev_fixedB_mpi eigen_exa
```

ソースコード

example/cxx/dense/gev_fixedB_mpi.cpp

```
template<typename T, typename MATRIX_MAJOR>
void function_matrix(rokko::localized_vector<double> const& eigval_tmp, rokko::
      distributed_matrix<T, MATRIX_MAJOR> const& eigvec, rokko::distributed_matrix<T,
      MATRIX_MAJOR>& result, rokko::distributed_matrix<T, MATRIX_MAJOR>& tmp) {
  for (int local_j=0; local_j<eigvec.get_n_local(); ++local_j) {
    int global_j = eigvec.translate_l2g_col(local_j);
    double coeff = eigval_tmp(global_j);
    for (int local_i=0; local_i<eigvec.get_m_local(); ++local_i) {
      double value = eigvec.get_local(local_i, local_j);
      tmp.set_local(local_i, local_j, coeff * value);
    }
  }
  product(1, tmp, false, eigvec, true, 0, result);
}
```

ソースコード

```
template<typename T, typename MATRIX_MAJOR>
void diagonalize_fixedB(rokko::parallel_dense_solver& solver, rokko::distributed_matrix<T,
      MATRIX_MAJOR>& A, rokko::distributed_matrix<T, MATRIX_MAJOR>& B, rokko::localized_vector<
      double>& eigval, rokko::distributed_matrix<T, MATRIX_MAJOR>& eigvec, T tol = 0) {
  rokko::distributed_matrix<double, matrix_major> tmp(A.get_mapping()), Binvroot(A.get_mapping
      ()), mat(A.get_mapping());
  rokko::parameters params;
  int myrank = A.get_myrank();
  params.set("routine", "");
  solver.diagonalize(B, eigval, eigvec, params);
  // computation of B^{-1/2}
  for(int i=0; i<eigval.size(); ++i)
    eigval(i) = (eigval(i) > tol) ? sqrt(1/eigval(i)) : 0;
  function_matrix(eigval, eigvec, Binvroot, tmp);

  // computation of B^{-1/2} A B^{-1/2}
  product(1, Binvroot, false, A, false, 0, tmp);
  product(1, tmp, false, Binvroot, false, 0, mat);
  // diagonalization of B^{-1/2} A B^{-1/2}
  solver.diagonalize(mat, eigval, tmp, params);

  // computation of {eigvec of Ax=lambda Bx} = B^{-1/2} {eigvec of B^{-1/2} A B^{-1/2}}
  product(1, Binvroot, false, tmp, false, 0, eigvec);
}
```