$K\omega$ マニュアル

リリース *2.0.0*

2017 年 06 月 13 日

Contents

Chapter 1

概要

本資料は ISSP Math Library の内の、Krylov 部分空間法に基づくシフト線形方程式群ソルバーライブラリ $K\omega$ に関するマニュアルである. 本ライブラリは, (射影付き) シフト線形問題

$$G_{ij}(z) = \langle i|(z\hat{I} - \hat{H})^{-1}|j\rangle \equiv \boldsymbol{\varphi}_i^* \cdot (z\hat{I} - \hat{H})^{-1}\boldsymbol{\varphi}_j \tag{1.1}$$

を, Krylov 部分空間法を用いて解くためのルーチンを提供する. 言語は fortran を用いる. また, BLAS レベル 1 ルーチンを使用する.

Chapter 2

アルゴリズム

このライブラリは, $\hat{H}$ および z が複素数であるか実数であるかに応じて, 次の 4 種類の計算をサポートする ($\hat{H}$ は複素数の場合はエルミート行列, 実数の場合は実対称行列).

- $\hat{H}$ も z も両方複素数の場合 : Shifted Bi-Conjugate Gradient(BiCG) 法 *[1]*
- $\hat{H}$ が実数で z が複素数の場合 : Shifted Conjugate Orthogonal Conjugate Gradient(COCG) 法 *[2]*
- $\hat{H}$ が複素数で z が実数の場合 : Shifted Conjugate Gradient(CG) 法 (複素ベクトル)
- $\hat{H}$ も z も両方実数の場合 : Shifted Conjugate Gradient(CG) 法 (実ベクトル)

いずれの場合も Seed switching *[2]* を行う. 左ベクトルが N_L 個, 右ベクトルが N_R 個 (典型的には 1 個) あるとする. 以下, 各手法のアルゴリズムを記載する.

2.1 Seed switch 付き Shifted BiCG 法

$G_{ij}(z_k) = 0 (i = 1 \cdots N_L,\ j = 1 \cdots N_R,\ k = 1 \cdots N_z)$

do $j = 1 \cdots N_R$

$\boldsymbol{r} = \boldsymbol{\varphi}_j$,

$\tilde{\boldsymbol{r}} =$ 任意, $\boldsymbol{r}^{\mathrm{old}} = \tilde{\boldsymbol{r}}^{\mathrm{old}} = \boldsymbol{0}$

$p_{ik} = 0 (i = 1 \cdots N_L,\ k = 1 \cdots N_z),\ \pi_k = \pi_k^{\mathrm{old}} = 1 (k = 1 \cdots N_z)$

$\rho = \infty,\ \alpha = 1,\ z_{\mathrm{seed}} = 0$

do iteration

◦ シード方程式

$\rho^{\mathrm{old}} = \rho,\ \rho = \tilde{\boldsymbol{r}}^* \cdot \boldsymbol{r}$

$\beta = \rho / \rho^{\mathrm{old}}$

$\boldsymbol{q} = (z_{\mathrm{seed}} \hat{I} - \hat{H}) \boldsymbol{r}$

$\alpha^{\mathrm{old}} = \alpha,\ \alpha = \frac{\rho}{\tilde{\boldsymbol{r}}^* \cdot \boldsymbol{q} - \beta \rho / \alpha}$

◦ シフト方程式

do $k = 1 \cdots N_z$

$\pi_k^{\rm new} = [1+\alpha(z_k - z_{\rm seed})]\pi_k - \frac{\alpha\beta}{\alpha^{\rm old}}(\pi_k^{\rm old} - \pi_k)$

do $i = 1 \cdots N_L$

$p_{ik} = \frac{1}{\pi_k}\boldsymbol{\varphi}_i^* \cdot \boldsymbol{r} + \frac{\pi_k^{\rm old}\pi_k^{\rm old}}{\pi_k \pi_k}\beta p_{ik}$

$G_{ij}(z_k) = G_{ij}(z_k) + \frac{\pi_k}{\pi_k^{\rm new}}\alpha p_{ik}$

$\pi_k^{\rm old} = \pi_k,\ \pi_k = \pi_k^{\rm new}$

end do i

end do k

$\boldsymbol{q} = \left(1 + \frac{\alpha\beta}{\alpha^{\rm old}}\right)\boldsymbol{r} - \alpha\boldsymbol{q} - \frac{\alpha\beta}{\alpha^{\rm old}}\boldsymbol{r}^{\rm old},\ \boldsymbol{r}^{\rm old} = \boldsymbol{r},\ \boldsymbol{r} = \boldsymbol{q}$

$\boldsymbol{q} = (z_{\rm seed}^*\hat{I} - \hat{H})\tilde{\boldsymbol{r}},\ \boldsymbol{q} = \left(1 + \frac{\alpha^*\beta^*}{\alpha^{\rm old*}}\right)\tilde{\boldsymbol{r}} - \alpha^*\boldsymbol{q} - \frac{\alpha^*\beta^*}{\alpha^{\rm old*}}\tilde{\boldsymbol{r}}^{\rm old},\ \tilde{\boldsymbol{r}}^{\rm old} = \tilde{\boldsymbol{r}},\ \tilde{\boldsymbol{r}} = \boldsymbol{q}$

∘ Seed switch

$|\pi_k|$ が最も小さい k を探す. $\rightarrow z_{\rm seed},\ \pi_{\rm seed},\ \pi_{\rm seed}^{\rm old}$

$\boldsymbol{r} = \boldsymbol{r}/\pi_{\rm seed},\ \boldsymbol{r}^{\rm old} = \boldsymbol{r}^{\rm old}/\pi_{\rm seed}^{\rm old},\ \tilde{\boldsymbol{r}} = \tilde{\boldsymbol{r}}/\pi_{\rm seed}^*,\ \tilde{\boldsymbol{r}}^{\rm old} = \tilde{\boldsymbol{r}}^{\rm old}/\pi_{\rm seed}^{\rm old*}$

$\alpha = (\pi_{\rm seed}^{\rm old}/\pi_{\rm seed})\alpha,\ \rho = \rho/(\pi_{\rm seed}^{\rm old}\pi_{\rm seed}^{\rm old})$

$\{\pi_k = \pi_k/\pi_{\rm seed},\ \pi_k^{\rm old} = \pi_k^{\rm old}/\pi_{\rm seed}^{\rm old}\}$

if($|\boldsymbol{r}| <$ Threshold) exit

end do iteration

end do j

2.2 Seed switch 付き Shifted COCG 法

BiCG のアルゴリズムで, $\tilde{\boldsymbol{r}} = \boldsymbol{r}^*,\ \tilde{\boldsymbol{r}}^{\rm old} = \boldsymbol{r}^{\rm old*}$ とすると得られる.

$G_{ij}(z_k) = 0 (i = 1 \cdots N_L,\ j = 1 \cdots N_R,\ k = 1 \cdots N_z)$

do $j = 1 \cdots N_R$

$\boldsymbol{r} = \boldsymbol{\varphi}_j, \boldsymbol{r}^{\rm old} = \boldsymbol{0}$

$p_{ik} = 0 (i = 1 \cdots N_L,\ k = 1 \cdots N_z),\ \pi_k = \pi_k^{\rm old} = 1 (k = 1 \cdots N_z)$

$\rho = \infty,\ \alpha = 1,\ z_{\rm seed} = 0$

do iteration

∘ シード方程式

$\rho^{\rm old} = \rho,\ \rho = \boldsymbol{r} \cdot \boldsymbol{r}$

$\beta = \rho/\rho^{\rm old}$

$\boldsymbol{q} = (z_{\rm seed}\hat{I} - \hat{H})\boldsymbol{r}$

$\alpha^{\rm old} = \alpha,\ \alpha = \frac{\rho}{\boldsymbol{r}\cdot\boldsymbol{q} - \beta\rho/\alpha}$

∘ シフト方程式

do $k = 1 \cdots N_z$

$\pi_k^{\rm new} = [1+\alpha(z_k - z_{\rm seed})]\pi_k - \frac{\alpha\beta}{\alpha^{\rm old}}(\pi_k^{\rm old} - \pi_k)$

do $i = 1 \cdots N_L$

$p_{ik} = \frac{1}{\pi_k}\boldsymbol{\varphi}_i^* \cdot \boldsymbol{r} + \frac{\pi_k^{\rm old}\pi_k^{\rm old}}{\pi_k \pi_k}\beta p_{ik}$

$G_{ij}(z_k) = G_{ij}(z_k) + \frac{\pi_k}{\pi_k^{\rm new}}\alpha p_{ik}$

$\pi_k^{\rm old} = \pi_k$, $\pi_k = \pi_k^{\rm new}$

end do i

end do k

$\boldsymbol{q} = \left(1 + \frac{\alpha\beta}{\alpha^{\rm old}}\right)\boldsymbol{r} - \alpha\boldsymbol{q} - \frac{\alpha\beta}{\alpha^{\rm old}}\boldsymbol{r}^{\rm old},\ \boldsymbol{r}^{\rm old} = \boldsymbol{r},\ \boldsymbol{r} = \boldsymbol{q}$

∘ Seed switch

$|\pi_k|$ が最も小さい k を探す. → $z_{\rm seed},\ \pi_{\rm seed},\ \pi_{\rm seed}^{\rm old}$

$\boldsymbol{r} = \boldsymbol{r}/\pi_{\rm seed},\ \boldsymbol{r}^{\rm old} = \boldsymbol{r}^{\rm old}/\pi_{\rm seed}^{\rm old}$

$\alpha = (\pi_{\rm seed}^{\rm old}/\pi_{\rm seed})\alpha$, $\rho = \rho/(\pi_{\rm seed}^{\rm old}\pi_{\rm seed}^{\rm old})$

$\{\pi_k = \pi_k/\pi_{\rm seed},\ \pi_k^{\rm old} = \pi_k^{\rm old}/\pi_{\rm seed}^{\rm old}\}$

if($|\boldsymbol{r}| <$ Threshold) exit

end do iteration

end do j

2.3 Seed switch 付き Shifted CG 法

BiCG のアルゴリズムで, $\tilde{\boldsymbol{r}} = \boldsymbol{r},\ \tilde{\boldsymbol{r}}^{\rm old} = \boldsymbol{r}^{\rm old}$ とすると得られる.

$G_{ij}(z_k) = 0 (i = 1 \cdots N_L,\ j = 1 \cdots N_R,\ k = 1 \cdots N_z)$

do $j = 1 \cdots N_R$

$\boldsymbol{r} = \boldsymbol{\varphi}_j$, $\boldsymbol{r}^{\rm old} = \boldsymbol{0}$

$p_{ik} = 0 (i = 1 \cdots N_L,\ k = 1 \cdots N_z),\ \pi_k = \pi_k^{\rm old} = 1 (k = 1 \cdots N_z)$

$\rho = \infty,\ \alpha = 1,\ z_{\rm seed} = 0$

do iteration

∘ シード方程式

$\rho^{\rm old} = \rho,\ \rho = \boldsymbol{r}^* \cdot \boldsymbol{r}$

$\beta = \rho/\rho^{\rm old}$

$\boldsymbol{q} = (z_{\rm seed}\hat{I} - \hat{H})\boldsymbol{r}$

$\alpha^{\rm old} = \alpha,\ \alpha = \frac{\rho}{\boldsymbol{r}^* \cdot \boldsymbol{q} - \beta\rho/\alpha}$

∘ シフト方程式

do $k = 1 \cdots N_z$

$\pi_k^{\rm new} = [1+\alpha(z_k - z_{\rm seed})]\pi_k - \frac{\alpha\beta}{\alpha^{\rm old}}(\pi_k^{\rm old} - \pi_k)$

do $i = 1 \cdots N_L$

$p_{ik} = \frac{1}{\pi_k}\boldsymbol{\varphi}_i^* \cdot \boldsymbol{r} + \left(\frac{\pi_k^{\rm old}}{\pi_k}\right)^2 \beta p_{ik}$

$G_{ij}(z_k) = G_{ij}(z_k) + \frac{\pi_k}{\pi_k^{\rm new}}\alpha p_{ik}$

$\pi_k^{\rm old} = \pi_k$, $\pi_k = \pi_k^{\rm new}$

end do i

end do k

$\boldsymbol{q} = \left(1 + \frac{\alpha\beta}{\alpha^{\rm old}}\right)\boldsymbol{r} - \alpha\boldsymbol{q} - \frac{\alpha\beta}{\alpha^{\rm old}}\boldsymbol{r}^{\rm old},\ \boldsymbol{r}^{\rm old} = \boldsymbol{r},\ \boldsymbol{r} = \boldsymbol{q}$

○ Seed switch

$|\pi_k|$ が最も小さい k を探す. $\rightarrow z_{\rm seed},\ \pi_{\rm seed},\ \pi_{\rm seed}^{\rm old}$

$\boldsymbol{r} = \boldsymbol{r}/\pi_{\rm seed},\ \boldsymbol{r}^{\rm old} = \boldsymbol{r}^{\rm old}/\pi_{\rm seed}^{\rm old}$

$\alpha = (\pi_{\rm seed}^{\rm old}/\pi_{\rm seed})\alpha$, $\rho = \rho/{\pi_{\rm seed}^{\rm old}}^2$

$\{\pi_k = \pi_k/\pi_{\rm seed},\ \pi_k^{\rm old} = \pi_k^{\rm old}/\pi_{\rm seed}^{\rm old}\}$

if($|\boldsymbol{r}| <$ Threshold) exit

end do iteration

end do j

Chapter 3

インストール方法

3.1 大まかな手順

最もシンプルには次のとおりである.

```
$ ./configure --prefix=install_dir
```

これにより, ビルドに必要なコンパイラやライブラリ等の環境のチェックが行われ, Makefile 等が作成される. ただし install_dir はインストール先のディレクトリの絶対パスとする (以後各自のディレクトリ名で読み替えること). なにも指定しないと /use/local/ が設定され, 後述の make install で/usr/local/lib 内にライブラリが置かれる (したがって, 管理者権限がない場合には install_dir を別の場所に指定しなければならない). configure にはこの他にも様々なオプションがあり, 必要に応じて用途や環境に合わせてそれらを使用する. 詳しくは *configure* **のオプション** を参照.

configure の実行が正常に行われ, Makefile が生成された後は

```
$ make
```

とタイプしてライブラリ等のビルドを行う. これが成功したのちに

```
$ make install
```

とすると, ライブラリが install_dir/lib に, ミニアプリが install_dir/bin に置かれる. make install をしなくても, ビルドをしたディレクトリ内にあるライブラリやミニアプリを使うことは可能であるが, 使い勝手がやや異なる.

共有リンクを行ったプログラムの実行時にライブラリを探しにいけるよう, 環境変数 LD_LIBRARY_PATH に $K\omega$ をインストールしたディレクトリを追加する.

```
$ export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:install_dir/lib
```

3.2 configure のオプション

configure には多数のオプションと変数があり, それらを組み合わせて指定する. 指定しない場合にはデフォルト値が使われる.

```
$ ./configure --prefix=/home/komega/ --with-mpi=yes FC=mpif90
```

おもなものを次に挙げる.

`---prefix`

デフォルト: `---prefix=/usr/local/`. ライブラリ等のインストールを行うディレクトリツリーを指定する.

`--with-mpi`

デフォルト: `--with-mpi=no` (MPI を用いない). MPI を用いるか (`--with-mpi=yes`), 否かを指定する.

`--with-openmp`

デフォルト: `--with-openmp=yes` (OpenMP を用いる). OpenMP を用いるか否か (`--with-openmp=no`) を指定する.

`--enable-shared`

デフォルト: `--enable-shared`. 共有ライブラリを作成するか否か

`--enable-static`

デフォルト: `--enable-static`. 静的ライブラリを作成するか否か.

`--disable-zdot`

デフォルト: `--enable-zdot`. MacOSX の標準の BLAS 等では, ZDOTC および ZDOTU 関数が正常に動作しないため, このオプションでこれらの関数を使わないようにする.

`--enable-threadsafe`

デフォルト: `--disable-threadsafe`. もしも OpenMP のパラレルリージョンの内側で $K\omega$ を呼び出したい (それぞれのスレッドで異なる問題を解きたい) 場合には, このオプションを用いる (**試験的**).

`FC`

デフォルト: システムにインストールされている fortran コンパイラをスキャンして, 自動的に設定する. `--with-mpiP` を指定した時にはそれに応じたコマンド (`mpif90` 等) を自動で探し出して設定する. `configure` の最後に出力される `FC` が望んだものでは無かった場合には, `./configure FC=gfortran` のように手で指定する.

`--help`

このオプションを指定した時には, ビルドの環境設定は行われず, 上記を含めたすべてのオプションを表示する.

Chapter 4

プログラム内でのライブラリの動作イメージ

以下では N_R のループは省略する (各右辺ベクトルごとに同じ事をすればいいので). また $G_{ij}(z_k)$ の代わりに N_z 個の N_L 次元の解ベクトル $\mathbf{x}_k$ を求める.

ライブラリの各ルーチンの名前は次の通りである.

- `komega_bicg_init`, `komega_cocg_init`, `komega_cg_c_init`, `komega_cg_r_init`

  ライブラリ内部で使う (ユーザーの目に触れない) 変数の Allocate や初期値設定を行う.

- `komega_bicg_update`, `komega_cocg_update`, `komega_cg_c_update`, `komega_cg_r_update`

  Iteration の中で呼び出される. 解ベクトル群の更新等を行う.

- `komega_bicg_finalize`, `komega_cocg_finalize`, `komega_cg_c_finalize`, `komega_cg_r_finalize`

  Allocate したライブラリ内部ベクトルを開放する.

- `komega_bicg_getcoef`, `komega_cocg_getcoef`, `komega_cg_c_getcoef`, `komega_cg_r_getcoef`

  各 iteration で保存しておいた α, β, z_{seed}, $\mathbf{r}^{\mathrm{L}}$ を取り出す.

- `komega_bicg_getvec`, `komega_cocg_getvec`, `komega_cg_c_getvec`, `komega_cg_r_getvec`

  $\boldsymbol{r}$, $\boldsymbol{r}^{\mathrm{old}}$, $\tilde{\boldsymbol{r}}$, $\tilde{\boldsymbol{r}}^{\mathrm{old}}$ を取り出す.

- `komega_bicg_restart`, `komega_cocg_restart`, `komega_cg_c_restart`, `komega_cg_r_restart`

  保存しておいた α 等を用いて, 新規の z での計算を行う. $\boldsymbol{r}$ 等も有る場合には `komega_bicg_init`, `komega_cocg_init`, `komega_cg_c_init`, `komega_cg_r_init` の代わりに用いてリスタートすることもできる.

ノート:

- `komega_*_init` を呼び出す前にサイズ N_H のベクトルを 2 本 (BiCG の時には 4 本)Allocate しておく.
- ハミルトニアン-ベクトル積を行う部分はあらかじめ作成しておく.

- 解ベクトルを Allocate しておく. ただし, 解ベクトルの長さは必ずしも N_H である必要はない. 実際前節の場合は N_L である. この時 (双) 共役勾配ベクトル $\mathbf{p}_k$ も N_z 本の N_L 次元のベクトルである. ユーザーは N_H 次元の残差ベクトルを N_L 次元へ変換するルーチン/関数をあらかじめ作っておかなければならない.

$$\mathbf{r}^{\mathrm{L}} = \hat{P}^{\dagger}\boldsymbol{r}, \qquad \hat{P} \equiv (\boldsymbol{\varphi}_1, \cdots, \boldsymbol{\varphi}_{N_L})$$

- `komega_*_update` の出力 `status` の第一成分が負の値になった場合には, 解が収束した, もしくは破たんしたことを表す. したがって `status(1) < 0` でループを抜けるようにしておく.
- `komega_*_update` 内での収束判定には, シード点での残差ベクトルの 2-ノルムが使われる. すなわち, すべてのシフト点での残差ベクトルの 2-ノルムが `threshold` を下回った時に収束したと見做される.
- 各反復での $\alpha, \beta, \mathbf{r}^{\mathrm{L}}$ を保存しておき, あとで再利用する場合には最大反復回数 `itermax` を `0` 以外の値に設定する.

4.1 Shifted BiCG ライブラリの動作イメージ

Allocate $\boldsymbol{v}_{12}$, $\boldsymbol{v}_{13}$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$, $\{\mathbf{x}_k\}$, $\mathbf{r}^{\mathrm{L}}$ $\boldsymbol{v}_2 = \boldsymbol{\varphi}_j$

`komega_bicg_init(N_H, N_L, N_z, x, z, itermax, threshold)` start

Allocate $\boldsymbol{v}_3$, $\boldsymbol{v}_5$, $\{\pi_k\}$, $\{\pi_k^{\mathrm{old}}\}$, $\{\mathbf{p}_k\}$

Copy $\{z_k\}$

`itermax` $\neq 0$ ならば α, β, $\mathbf{r}^{\mathrm{L}}$ を保存する配列を確保する.

$\boldsymbol{v}_4 = \boldsymbol{v}_2^*$ (任意), $\boldsymbol{v}_3 = \boldsymbol{v}_5 = \mathbf{0}$,

$\mathbf{p}_k = \mathbf{x}_k = \mathbf{0}(k = 1 \cdots N_z)$, $\pi_k = \pi_k^{\mathrm{old}} = 1(k = 1 \cdots N_z)$

$\rho = \infty$, $\alpha = 1$, $z_{\mathrm{seed}} = 0$

($\boldsymbol{v}_2 \equiv \boldsymbol{r}$, $\boldsymbol{v}_3 \equiv \boldsymbol{r}^{\mathrm{old}}$, $\boldsymbol{v}_4 \equiv \tilde{\boldsymbol{r}}$, $\boldsymbol{v}_5 \equiv \tilde{\boldsymbol{r}}^{\mathrm{old}}$.)

`komega_bicg_init` finish

do iteration

$\mathbf{r}^{\mathrm{L}} = \hat{P}^{\dagger}\boldsymbol{v}_2$

$\boldsymbol{v}_{12} = \hat{H}\boldsymbol{v}_2$, $\boldsymbol{v}_{14} = \hat{H}\boldsymbol{v}_4$ [$(\boldsymbol{v}_{12}, \boldsymbol{v}_{14}) = \hat{H}(\boldsymbol{v}_2, \boldsymbol{v}_4)$ とも書ける]

`komega_bicg_update(v_12, v_2, v_14, v_4, x, r_small, status)` start

◦ シード方程式

$\rho^{\mathrm{old}} = \rho$, $\rho = \boldsymbol{v}_4^* \cdot \boldsymbol{v}_2$

$\beta = \rho / \rho^{\mathrm{old}}$

$\boldsymbol{v}_{12} = z_{\mathrm{seed}}\boldsymbol{v}_2 - \boldsymbol{v}_{12}$, $\boldsymbol{v}_{14} = z_{\mathrm{seed}}^*\boldsymbol{v}_4 - \boldsymbol{v}_{14}$

$\alpha^{\mathrm{old}} = \alpha$, $\alpha = \frac{\rho}{\boldsymbol{v}_3^* \cdot \boldsymbol{v}_{12} - \beta\rho/\alpha}$

◦ シフト方程式

do $k = 1 \cdots N_z$

$\pi_k^{\mathrm{new}} = [1 + \alpha(z_k - z_{\mathrm{seed}})]\pi_k - \frac{\alpha\beta}{\alpha^{\mathrm{old}}}(\pi_k^{\mathrm{old}} - \pi_k)$

$\mathbf{p}_k = \frac{1}{\pi_k}\mathbf{r}^{\mathrm{L}} + \frac{\pi_k^{\mathrm{old}}\pi_k^{\mathrm{old}}}{\pi_k\pi_k}\beta\mathbf{p}_k$

$\mathbf{x}_k = \mathbf{x}_k + \frac{\pi_k}{\pi_k^{\rm new}}\alpha\mathbf{p}_k$

$\pi_k^{\rm old} = \pi_k$, $\pi_k = \pi_k^{\rm new}$

end do k

$\boldsymbol{v}_{12} = \left(1 + \frac{\alpha\beta}{\alpha^{\rm old}}\right)\boldsymbol{v}_2 - \alpha\boldsymbol{v}_{12} - \frac{\alpha\beta}{\alpha^{\rm old}}\boldsymbol{v}_3$, $\boldsymbol{v}_3 = \boldsymbol{v}_2,\ \boldsymbol{v}_2 = \boldsymbol{v}_{12}$

$\boldsymbol{v}_{14} = \left(1 + \frac{\alpha^*\beta^*}{\alpha^{\rm old*}}\right)\boldsymbol{v}_4 - \alpha^*\boldsymbol{v}_{14} - \frac{\alpha^*\beta^*}{\alpha^{\rm old*}}\boldsymbol{v}_5$, $\boldsymbol{v}_5 = \boldsymbol{v}_4,\ \boldsymbol{v}_4 = \boldsymbol{v}_{14}$

∘ Seed switch

$|\pi_k|$ が最も小さい k を探す. → $z_{\rm seed},\ \pi_{\rm seed},\ \pi_{\rm seed}^{\rm old}$

$\boldsymbol{v}_2 = \boldsymbol{v}_2/\pi_{\rm seed}$, $\boldsymbol{v}_3 = \boldsymbol{v}_3/\pi_{\rm seed}^{\rm old}$, $\boldsymbol{v}_4 = \boldsymbol{v}_4/\pi_{\rm seed}^{*}$, $\boldsymbol{v}_5 = \boldsymbol{v}_5/\pi_{\rm seed}^{\rm old*}$

$\alpha = (\pi_{\rm seed}^{\rm old}/\pi_{\rm seed})\alpha$, $\rho = \rho/(\pi_{\rm seed}^{\rm old}\pi_{\rm seed}^{\rm old})$

$\{\pi_k = \pi_k/\pi_{\rm seed},\ \pi_k^{\rm old} = \pi_k^{\rm old}/\pi_{\rm seed}^{\rm old}\}$

`komega_bicg_update` finish

if(status(1) < 0 (これは $|\boldsymbol{v}_2| <$ Threshold となった事を意味する)) exit

end do iteration

`komega_bicg_finalize` start

Deallocate $\boldsymbol{v}_4$, $\boldsymbol{v}_5$, $\{\pi_k\}$, $\{\pi_k^{\rm old}\}$, $\{\mathbf{p}_k\}$

`komega_bicg_finalize` finish

4.2 Shifted COCG ライブラリの動作イメージ

Allocate $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\{\mathbf{x}_k\}$, $\mathbf{r}^{\rm L}$ $\boldsymbol{v}_2 = \boldsymbol{\varphi}_j$

`komega_cocg_init(N_H, N_L, N_z, x, z, itermax, threshold)` start

Allocate $\boldsymbol{v}_3$, $\{\pi_k\}$, $\{\pi_k^{\rm old}\}$, $\{\mathbf{p}_k\}$

Copy $\{z_k\}$

`itermax` $\neq$ `0` ならば α, β, $\mathbf{r}^{\rm L}$ を保存する配列を確保する.

$\boldsymbol{v}_3 = \mathbf{0}$,

$\mathbf{p}_k = \mathbf{x}_k = \mathbf{0}(k = 1 \cdots N_z),\ \pi_k = \pi_k^{\rm old} = 1(k = 1 \cdots N_z)$

$\rho = \infty,\ \alpha = 1,\ \beta = 0,\ z_{\rm seed} = 0$

($\boldsymbol{v}_2 \equiv \boldsymbol{r}$, $\boldsymbol{v}_3 \equiv \boldsymbol{r}^{\rm old}$.)

`komega_cocg_init` finish

do iteration

$\mathbf{r}^{\rm L} = \hat{P}^{\dagger}\boldsymbol{v}_2$

$\boldsymbol{v}_1 = \hat{H}\boldsymbol{v}_2$

`komega_cocg_update(v_1, v_2, x, r_small, status)` start

∘ シード方程式

$\rho^{\rm old} = \rho,\ \rho = \boldsymbol{v}_2 \cdot \boldsymbol{v}_2$

$\beta = \rho/\rho^{\rm old}$

$\boldsymbol{v}_1 = z_{\rm seed}\boldsymbol{v}_2 - \boldsymbol{v}_1$

$\alpha^{\rm old} = \alpha,\ \alpha = \frac{\rho}{\boldsymbol{v}_2\cdot\boldsymbol{v}_1 - \beta\rho/\alpha}$

∘ シフト方程式

do $k = 1 \cdots N_z$

$\pi_k^{\rm new} = [1 + \alpha(z_k - z_{\rm seed})]\pi_k - \frac{\alpha\beta}{\alpha^{\rm old}}(\pi_k^{\rm old} - \pi_k)$

$\mathbf{p}_k = \frac{1}{\pi_k}\mathbf{r}^{\rm L} + \frac{\pi_k^{\rm old}\pi_k^{\rm old}}{\pi_k\pi_k}\beta\mathbf{p}_k$

$\mathbf{x}_k = \mathbf{x}_k + \frac{\pi_k}{\pi_k^{\rm new}}\alpha\mathbf{p}_k$

$\pi_k^{\rm old} = \pi_k,\ \pi_k = \pi_k^{\rm new}$

end do k

$\boldsymbol{v}_1 = \left(1 + \frac{\alpha\beta}{\alpha^{\rm old}}\right)\boldsymbol{v}_2 - \alpha\boldsymbol{v}_1 - \frac{\alpha\beta}{\alpha^{\rm old}}\boldsymbol{v}_3$

$\boldsymbol{v}_3 = \boldsymbol{v}_2$, $\boldsymbol{v}_2 = \boldsymbol{v}_1$

∘ Seed switch

$|\pi_k|$ が最も小さい k を探す. → $z_{\rm seed},\ \pi_{\rm seed},\ \pi_{\rm seed}^{\rm old}$

$\boldsymbol{v}_2 = \boldsymbol{v}_2/\pi_{\rm seed}$, $\boldsymbol{v}_3 = \boldsymbol{v}_3/\pi_{\rm seed}^{\rm old}$

$\alpha = (\pi_{\rm seed}^{\rm old}/\pi_{\rm seed})\alpha$, $\rho = \rho/(\pi_{\rm seed}^{\rm old}\pi_{\rm seed}^{\rm old})$

$\{\pi_k = \pi_k/\pi_{\rm seed},\ \pi_k^{\rm old} = \pi_k^{\rm old}/\pi_{\rm seed}^{\rm old}\}$

`komega_cocg_update` finish

if(status(1) < 0 (これは $|\boldsymbol{v}_2| <$ Threshold となった事を意味する)) exit

end do iteration

`komega_cocg_finalize` start

Deallocate $\boldsymbol{v}_3$, $\{\pi_k\}$, $\{\pi_k^{\rm old}\}$, $\{\mathbf{p}_k\}$

`komega_cocg_finalize` finish

4.3 Shifted CG ライブラリの動作イメージ

COCG と同様.

Chapter 5

使用方法

各ライブラリともユーザーはライブラリ名および型を指定し,

- 初期設定 (**_init*)
- アップデート (**_update*)
- (オプション) 再計算用の情報を取り出す. (**_getcoef*, **_getvec*)
- 終了関数 (**_finalize*)

の手順で関数を使用することで, 計算が実施される. なお, リスタートを行う場合には

- 前回の計算で残した再計算用の情報を用いた初期設定 (**_restart*)
- アップデート (**_update*)
- (オプション) 更なる再計算用の情報を取り出す. (**_getcoef*, **_getvec*)
- 終了関数 (**_finalize*)

の手順で実行する.

警告: $K\omega$ はスレッドセーフ **ではない** ので, これらのルーチンは必ず OpenMP のパラレルリージョンの外から呼ばなければならない. もしもパラレルリージョンの内側で $K\omega$ を呼び出したい (それぞれのスレッドで異なる問題を解きたい) 場合には, `configure` のオプション `--enable-threadsafe` を利用する (*configure* **のオプション** 参照). ただしこのモードは試験的なものである.

fortran から呼び出すときには

```
USE komega_cg_r ! 実ベクトルに対する共役勾配法
USE komega_cg_c ! 複素ベクトルに対する共役勾配法
USE komega_cocg ! 共線直交共役勾配法
USE komega_bicg ! 双共役勾配法
```

のようにモジュールを呼び出す (すべてのモジュールを呼び出す必要はなく, 行う計算の種類に対応するものだけでよい).

C/C++で書かれたプログラムから呼び出すときには、

```
#include komega.h
```

のようにヘッダーファイルを読み込む。また、スカラー引数はすべてポインタとして渡す。

また MPI/ハイブリッド並列のときにライブラリに渡すコミュニケーター変数を, 次のように C/C++のものから fortran のものに変換する。

```
comm_f = MPI_Comm_c2f(comm_c);
```

5.1 各ルーチンの説明

5.1.1 *_init

ライブラリ内部変数の割り付けおよび初期化を行う. シフト線形方程式を解く前に, 一番初めに実行する.

構文

Fortran

```
CALL komega_cg_r_init(ndim, nl, nz, x, z, itermax, threshold, comm)
CALL komega_cg_c_init(ndim, nl, nz, x, z, itermax, threshold, comm)
CALL komega_cocg_init(ndim, nl, nz, x, z, itermax, threshold, comm)
CALL komega_bicg_init(ndim, nl, nz, x, z, itermax, threshold, comm)
```

C/C++

```
komega_cg_r_init(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm);
komega_cg_c_init(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm);
komega_cocg_init(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm);
komega_bicg_init(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm);
```

パラメーター

```
INTEGER,INTENT(IN) :: ndim
```

線形方程式の次元. 以降のサブルーチンのパラメーターの次元で現れる ndim はこれと同じものになる.

```
INTEGER,INTENT(IN) :: nl
```

射影された解ベクトルの次元. 以降のサブルーチンのパラメーターの次元で現れる nl はこれと同じものになる.

```
INTEGER,INTENT(IN) :: nz
```

シフト点の数. 以降のサブルーチンのパラメーターの次元で現れる nz はこれと同じものになる.

```
REAL(8),INTENT(OUT) :: x(nl*nz) ! ("CG_R_init", "cg_c_init" の場合)
COMPLEX(8),INTENT(OUT) :: x(nl*nz) ! (それ以外)
```

解ベクトル. 0 ベクトルが返される.

```
REAL(8),INTENT(IN) :: z(nz) ! ("CG_R_init", "cg_c_init" の場合)
COMPLEX(8),INTENT(IN) :: z(nz) ! (それ以外)
```

シフト点.

```
INTEGER,INTENT(IN) :: itermax
```

リスタート用配列の割り付けのための最大反復回数. これを 0 にした場合にはリスタート用配列を割りつけない (したがって後述のリスタート用変数の出力を行えない)

```
REAL(8),INTENT(IN) :: threshold
```

収束判定用しきい値. シード方程式の残差ベクトルの 2-ノルムがこの値を下回った時に収束したと判定する.

```
INTEGER,INTENT(IN),OPTIONAL :: comm
```

オプショナル引数. MPI のコミニュケーター (`MPI_COMM_WORLD` など) を入れる. $K\omega$ を内部で MPI/Hybrid 並列するときのみ入力する. C 言語では使用しないときには `NULL` を入れる.

5.1.2 *_restart

リスタートを行う場合に *_init の代わりに用いる. ライブラリ内部変数の割り付けおよび初期化を行う. シフト線形方程式を解く前に, 一番初めに実行する.

構文

Fortran

```
CALL komega_cg_r_restart(ndim, nl, nz, x, z, itermax, threshold, status, &
&                  iter_old, v2, v12, alpha_save, beta_save, z_seed, r_l_save, comm)
CALL komega_cg_c_restart(ndim, nl, nz, x, z, itermax, threshold, status, &
&                  iter_old, v2, v12, alpha_save, beta_save, z_seed, r_l_save, comm)
CALL komega_cocg_restart(ndim, nl, nz, x, z, itermax, threshold, status, &
&                  iter_old, v2, v12, alpha_save, beta_save, z_seed, r_l_save, comm)
CALL komega_bicg_restart(ndim, nl, nz, x, z, itermax, threshold, status, &
&                  iter_old, v2, v12, v4, v14, alpha_save, beta_save, &
&                  z_seed, r_l_save, comm)
```

C/C++

```
komega_cg_r_restart(&ndim, &nl, &nz, x, z, &itermax, &threshold, status, &
&                  &iter_old, v2, v12, alpha_save, beta_save, &z_seed, r_l_save, &comm);
komega_cg_c_restart(&ndim, &nl, &nz, x, z, &itermax, &threshold, status, &
&                  &iter_old, v2, v12, alpha_save, beta_save, &z_seed, r_l_save, &comm);
komega_cocg_restart(&ndim, &nl, &nz, x, z, &itermax, &threshold, status, &
&                  &iter_old, v2, v12, alpha_save, beta_save, &z_seed, r_l_save, &comm);
komega_bicg_restart(&ndim, &nl, &nz, x, z, &itermax, &threshold, status, &
&                  &iter_old, v2, v12, v4, v14, alpha_save, beta_save, &
&                  &z_seed, r_l_save, &comm);
```

パラメーター

```
INTEGER,INTENT(IN) :: ndim
INTEGER,INTENT(IN) :: nl
INTEGER,INTENT(IN) :: nz
REAL(8),INTENT(OUT) :: x(nl*nz)
REAL(8),INTENT(IN) :: z(nz) ! ("CG_R_restart", "cg_c_restart" の場合)
COMPLEX(8),INTENT(IN) :: z(nz) ! (それ以外)
INTEGER,INTENT(IN) :: itermax
REAL(8),INTENT(IN) :: threshold
INTEGER,INTENT(IN),OPTIONAL :: comm
```

*_init と同様.

```
INTEGER,INTENT(OUT) :: status(3)
```

エラーコードを返す.

第一成分 (`status(1)`)

解が収束した場合, もしくは計算が破綻した場合には現在の総反復回数にマイナスが付いた値が返される. それ以外の場合には現在の総反復回数 (マイナスが付かない) が返される. `status(1)` が正の値の時のみ反復を続行できる. それ以外の場合は反復を進めても有意な結果は得られない.

第二成分 (`status(2)`)

`itermax` を有限にして, かつ `itermax` 回の反復で収束に達しなかった場合には `1` が返される. α が発散した場合には `2` が返される. π_{seed} が 0 ににになった場合には `3` が返される. `COCG_restart` もしくは `BiCG_restart` で, 残差ベクトルと影の残差ベクトルが直交した場合には `4` が返される. それ以外の場合には `0` が返される.

第三成分 (`status(3)`)

シード点の index が返される.

```
INTEGER,INTENT(IN) :: iter_old
```

先行する計算での反復回数.

```
REAL(8),INTENT(IN) :: v2(ndim) ! ("CG_R_restart" の場合)
COMPLEX(8),INTENT(IN) :: v2(ndim) ! (それ以外)
```

先行する計算での最後の残差ベクトル.

```
REAL(8),INTENT(IN) :: v12(ndim) ! ("CG_R_restart" の場合)
COMPLEX(8),INTENT(IN) :: v12(ndim) ! (それ以外)
```

先行する計算での最後から 2 番目の残差ベクトル.

```
REAL(8),INTENT(IN) :: alpha_save(iter_old) ! ("CG_R_restart", "cg_c_restart"の場合)
COMPLEX(8),INTENT(IN) :: alpha_save(iter_old) ! (それ以外)
```

先行する計算での各反復での (Bi)CG 法のパラメーター α.

```
REAL(8),INTENT(IN) :: beta_save(iter_old) ! ("CG_R_restart", "cg_c_restart"の場合)
COMPLEX(8),INTENT(IN) :: beta_save(iter_old) ! (それ以外)
```

先行する計算での各反復での (Bi)CG 法のパラメーター β.

```
REAL(8),INTENT(IN) :: z_seed ! ("CG_R_restart", "cg_c_restart"の場合)
COMPLEX(8),INTENT(IN) :: z_seed ! (それ以外)
```

先行する計算でのシードシフト.

```
REAL(8),INTENT(IN) :: r_l_save(nl,iter_old) ! ("CG_R_restart"の場合)
COMPLEX(8),INTENT(IN) :: r_l_save(nl,iter_old) ! (それ以外)
```

先行する計算での各反復での射影された残差ベクトル.

```
REAL(8),INTENT(IN) :: v4(ndim) ! ("CG_R_restart" の場合)
COMPLEX(8),INTENT(IN) :: v4(ndim) ! (それ以外)
```

`BiCG_restart` の場合のみ使用. 先行する計算での最後の影の残差ベクトル.

```
REAL(8),INTENT(IN) :: v14(ndim) ! ("CG_R_restart" の場合)
COMPLEX(8),INTENT(IN) :: v14(ndim) ! (それ以外)
```

`BiCG_restart` の場合のみ使用. 先行する計算での最後から 2 番目の影の残差ベクトル.

5.1.3 *_update

ループ内で行列ベクトル積と交互に呼ばれて解を更新する.

構文

Fortran

```
CALL komega_cg_r_update(v12, v2, x, r_l, status)
CALL komega_cg_c_update(v12, v2, x, r_l, status)
CALL komega_cocg_update(v12, v2, x, r_l, status)
CALL komega_bicg_update(v12, v2, v14, v4, x, r_l, status)
```

C/C++

```
komega_cg_r_update(v12, v2, x, r_l, status);
komega_cg_c_update(v12, v2, x, r_l, status);
komega_cocg_update(v12, v2, x, r_l, status);
komega_bicg_update(v12, v2, v14, v4, x, r_l, status);
```

パラメーター

```
REAL(8),INTENT(INOUT) :: v12(ndim) ! ("CG_R_update" の場合)
COMPLEX(8),INTENT(INOUT) :: v12(ndim) ! (それ以外)
```

入力は残差ベクトル (v2) と行列の積. 出力は, 更新された残差ベクトルの 2-ノルムが, 先頭の要素に格納される (これは収束の具合を表示して調べる時などに用いる).

```
REAL(8),INTENT(INOUT) :: v2(ndim) ! ("CG_R_update" の場合)
COMPLEX(8),INTENT(INOUT) :: v2(ndim) ! (それ以外)
```

入力は残差ベクトル. 出力は更新された残差ベクトル.

```
REAL(8),INTENT(IN) :: v14(ndim) ! ("CG_R_update" の場合)
COMPLEX(8),INTENT(IN) :: v14(ndim) ! (それ以外)
```

影の残差ベクトル (v4) と行列の積.

```
REAL(8),INTENT(INOUT) :: v4(ndim) ! ("CG_R_update" の場合)
COMPLEX(8),INTENT(INOUT) :: v4(ndim) ! (それ以外)
```

入力は影の残差ベクトル. 出力は更新された影の残差ベクトル.

```
INTEGER,INTENT(OUT) :: status(3)
```

エラーコードを返す.

第一成分 (status(1))

解が収束した場合, もしくは計算が破綻した場合には現在の総反復回数にマイナスが付いた値が返される. それ以外の場合には現在の総反復回数 (マイナスが付かない) が返される. status(1) が正の値の時のみ反復を続行できる. それ以外の場合は反復を進めても有意な結果は得られない.

第二成分 (status(2))

*_init ルーチンで, itermax を有限にして, かつ itermax 回の反復で収束に達しなかった場合には 1 が返される. α が発散した場合には 2 が返される. π_{seed} が 0 にになった場合には 3 が返される. COCG_update もしくは BiCG_update で, 残差ベクトルと影の残差ベクトルが直交した場合には 4 が返される. それ以外の場合には 0 が返される.

第三成分 (`status(3)`)

シード点の index が返される.

5.1.4 *_getcoef

後でリスタートをするときに必要な係数を取得する. このルーチンを呼び出すためには, *_init ルーチンで `itermax` を `0` 以外の値にしておく必要がある.

また, このルーチンで使われる総反復回数 (`iter_old`) は *_update の出力 `status` を用いて次のように計算される.

```
iter_old = ABS(status(1))
```

構文

Fortran

```
CALL komega_cg_r_getcoef(alpha_save, beta_save, z_seed, r_l_save)
CALL komega_cg_c_getcoef(alpha_save, beta_save, z_seed, r_l_save)
CALL komega_cocg_getcoef(alpha_save, beta_save, z_seed, r_l_save)
CALL komega_bicg_getcoef(alpha_save, beta_save, z_seed, r_l_save)
```

C/C++

```
komega_cg_r_getcoef(alpha_save, beta_save, &z_seed, r_l_save);
komega_cg_c_getcoef(alpha_save, beta_save, &z_seed, r_l_save);
komega_cocg_getcoef(alpha_save, beta_save, &z_seed, r_l_save);
komega_bicg_getcoef(alpha_save, beta_save, &z_seed, r_l_save);
```

パラメーター

```
REAL(8),INTENT(OUT) :: alpha_save(iter_old) ! ("CG_R_restart", "cg_c_restart"の場合)
COMPLEX(8),INTENT(OUT) :: alpha_save(iter_old) ! (それ以外)
```

各反復での (Bi)CG 法のパラメーター α.

```
REAL(8),INTENT(OUT) :: beta_save(iter_old) ! ("CG_R_restart", "cg_c_restart"の場合)
COMPLEX(8),INTENT(OUT) :: beta_save(iter_old) ! (それ以外)
```

各反復での (Bi)CG 法のパラメーター β.

```
REAL(8),INTENT(OUT) :: z_seed ! ("CG_R_restart", "cg_c_restart"の場合)
COMPLEX(8),INTENT(OUT) :: z_seed ! (それ以外)
```

シードシフト.

```
REAL(8),INTENT(IN) :: r_l_save(nl,iter_old) ! ("CG_R_restart"の場合)
COMPLEX(8),INTENT(IN) :: r_l_save(nl,iter_old) ! (それ以外)
```

各反復での射影された残差ベクトル.

5.1.5 *_getvec

後でリスタートをするときに必要な残差ベクトルを取得する. このルーチンを呼び出すためには, *_init ルーチンで `itermax` を `0` 以外の値にしておく必要がある.

構文

Fortran

```
CALL komega_cg_r_getvec(r_old)
CALL komega_cg_c_getvec(r_old)
CALL komega_cocg_getvec(r_old)
CALL komega_bicg_getvec(r_old, r_tilde_old)
```

C/C++

```
komega_cg_r_getvec(r_old);
komega_cg_c_getvec(r_old);
komega_cocg_getvec(r_old);
komega_bicg_getvec(r_old, r_tilde_old);
```

パラメーター

```
REAL(8),INTENT(OUT) :: r_old(ndim) ! ("CG_R_getvec" の場合)
COMPLEX(8),INTENT(OUT) :: r_old(ndim) ! (それ以外)
```

先行する計算での最後から 2 番目の残差ベクトル.

```
COMPLEX(8),INTENT(OUT) :: r_tilde_old(ndim)
```

BiCG_getvec の場合のみ使用. 先行する計算での最後から 2 番目の影の残差ベクトル.

5.1.6 *_getresidual

各シフト点での残差ベクトルの 2-ノルムを取得する. このルーチンは *_init と *_finalize の間の任意の場所で呼び出すことが出来る. また, いつ何回呼び出しても最終的な計算結果には影響を与えない.

構文

Fortran

```
CALL komega_cg_r_getresidual(res)
CALL komega_cg_c_getresidual(res)
CALL komega_cocg_getresidual(res)
CALL komega_bicg_getresidual(res)
```

C/C++

```
komega_cg_r_getresidual(res);
komega_cg_c_getresidual(res);
komega_cocg_getresidual(res);
komega_bicg_getresidual(res);
```

パラメーター

```
COMPLEX(8),INTENT(OUT) :: res(nz)
```

各シフト点での残差ベクトルの 2-ノルム.

5.1.7 *_finalize

ライブラリ内部で割りつけた配列のメモリを解放する.

構文

Fortran

```
CALL komega_cg_r_finalize()
CALL komega_cg_c_finalize()
CALL komega_cocg_finalize()
CALL komega_bicg_finalize()
```

C/C++

```
komega_cg_r_finalize();
komega_cg_c_finalize();
komega_cocg_finalize();
komega_bicg_finalize();
```

5.2 Shifted BiCG ライブラリを使用したソースコードの例

以下, 代表的な例として Shifted BiCG ライブラリの場合の使用方法を記載する.

```
PROGRAM my_prog
  !
  USE komega_bicg, ONLY : komega_bicg_init, komega_bicg_restart, &
  &                       komega_bicg_update, komega_bicg_getcoef, &
  &                       komega_bicg_getvec, komega_bicg_finalize
  USE solve_cc_routines, ONLY : input_size, input_restart, &
  &                             projection, &
  &                             hamiltonian_prod, generate_system, &
  &                             output_restart, output_result
  !
  IMPLICIT NONE
  !
  INTEGER,SAVE :: &
  & ndim,     & ! Size of Hilvert space
  & nz,       & ! Number of frequencies
  & nl,       & ! Number of Left vector
  & itermax, & ! Max. number of iteraction
  & iter_old   ! Number of iteraction of previous run
  !
  REAL(8),SAVE :: &
  & threshold ! Convergence Threshold
  !
  COMPLEX(8),SAVE :: &
  & z_seed ! Seed frequency
  !
  COMPLEX(8),ALLOCATABLE,SAVE :: &
  & z(:)          ! (nz): Frequency
  !
  COMPLEX(8),ALLOCATABLE,SAVE :: &
  & ham(:,:), &
  & rhs(:), &
  & v12(:), v2(:), & ! (ndim): Working vector
  & v14(:), v4(:), & ! (ndim): Working vector
  & r_l(:), & ! (nl) : Projeccted residual vector
  & x(:,:) ! (nl,nz) : Projected result
  !
  ! Variables for Restart
  !
  COMPLEX(8),ALLOCATABLE,SAVE :: &
  & alpha(:), beta(:) ! (iter_old)
```

```
!
COMPLEX(8),ALLOCATABLE,SAVE :: &
& r_l_save(:,:) ! (nl,iter_old) Projected residual vectors
!
! Variables for Restart
!
INTEGER :: &
& iter,     & ! Counter for Iteration
& status(3)
!
LOGICAL :: &
& restart_in, & ! If .TRUE., sestart from the previous result
& restart_out   ! If .TRUE., save datas for the next run
!
! Input Size of vectors, numerical conditions
!
CALL input_size(ndim,nl,nz)
CALL input_condition(itermax,threshold,restart_in,restart_out)
!
ALLOCATE(v12(ndim), v2(ndim), v14(ndim), v4(ndim), r_l(nl), &
&         x(nl,nz), z(nz), ham(ndim,ndim), rhs(ndim))
!
CALL generate_system(ndim, ham, rhs, z)
!
WRITE(*,*)
WRITE(*,*) "#####  CG Initialization  #####"
WRITE(*,*)
!
IF(restart_in) THEN
  !
  CALL input_restart(iter_old, zseed, alpha, beta, r_l_save)
  !
  IF(restart_out) THEN
     CALL komega_bicg_restart( &
     &     ndim, nl, nz, x, z, itermax, threshold, &
     &     status, iter_old, v2, v12, v4, v14, alpha, &
     &     beta, z_seed, r_l_save)
  ELSE
     CALL komega_bicg_restart( &
     &     ndim, nl, nz, x, z, 0, threshold, &
     &     status, iter_old, v2, v12, v4, v14, alpha, &
     &     beta, z_seed, r_l_save)
  END IF
  !
  ! These vectors were saved in BiCG routine
  !
  DEALLOCATE(alpha, beta, r_l_save)
  !
  IF(status(1) /= 0) GOTO 10
  !
ELSE
   !
   ! Generate Right Hand Side Vector
   !
   v2(1:ndim) = rhs(1:ndim)
   v4(1:ndim) = CONJG(v2(1:ndim))
   !v4(1:ndim) = v2(1:ndim)
   !
```

```
   IF(restart_out) THEN
      CALL komega_bicg_init(ndim, nl, nz, x, z, termax, threshold)
   ELSE
      CALL komega_bicg_init(ndim, nl, nz, x, z, 0, threshold)
   END IF
   !
END IF
!
! BiCG Loop
!
WRITE(*,*)
WRITE(*,*) "#####  CG Iteration  #####"
WRITE(*,*)
!
DO iter = 1, itermax
   !
   ! Projection of Residual vector into the space
   ! spaned by left vectors
   !
   r_l(1:nl) = projection(v2(1:nl))
   !
   ! Matrix-vector product
   !
   CALL hamiltonian_prod(Ham, v2, v12)
   CALL hamiltonian_prod(Ham, v4, v14)
   !
   ! Update result x with BiCG
   !
   CALL komega_bicg_update(v12, v2, v14, v4, x, r_l, status)
   !
   WRITE(*,'(a,i,a,3i,a,e15.5)') "lopp : ", iter, &
   &                             ", status : ", status(1:3), &
   &                             ", Res. : ", DBLE(v12(1))
   IF(status(1) < 0) EXIT
   !
END DO
!
IF(status(2) == 0) THEN
   WRITE(*,*) "  Converged in iteration ", ABS(status(1))
ELSE IF(status(2) == 1) THEN
   WRITE(*,*) "  Not Converged in iteration ", ABS(status(1))
ELSE IF(status(2) == 2) THEN
   WRITE(*,*) "  Alpha becomes infinity", ABS(status(1))
ELSE IF(status(2) == 3) THEN
   WRITE(*,*) "  Pi_seed becomes zero", ABS(status(1))
ELSE IF(status(2) == 4) THEN
WRITE(*,*) "  Residual & Shadow residual are orthogonal", &
&          ABS(status(1))
END IF
!
! Total number of iteration
!
iter_old = ABS(status(1))
!
! Get these vectors for restart in the Next run
!
IF(restart_out) THEN
   !
```

```
      ALLOCATE(alpha(iter_old), beta(iter_old), r_l_save(nl,iter_old))
      !
      CALL komega_bicg_getcoef(alpha, beta, z_seed, r_l_save)
      CALL komega_bicg_getvec(v12,v14)
      !
      CALL output_restart(iter_old, z_seed, alpha, beta, &
      &                   r_l_save, v12, v14)
      !
      DEALLOCATE(alpha, beta, r_l_save)
      !
    END IF
    !
10  CONTINUE
    !
    ! Deallocate all intrinsic vectors
    !
    CALL komega_bicg_finalize()
    !
    ! Output to a file
    !
    CALL output_result(nl, nz, z, x, r_l)
    !
    DEALLOCATE(v12, v2, v14, v4, r_l, x, z)
    !
    WRITE(*,*)
    WRITE(*,*) "#####  Done  #####"
    WRITE(*,*)
    !
END PROGRAM my_prog
```

Chapter 6

プログラムの再配布

6.1 自分のプログラムに Komega を含める

$K\omega$ ライブラリは下記の *Lesser General Public License* (LGPL) に基づいて配布されている. これはかいつまんで言うと次のようなことである.

- 個人的なプログラムや, 研究室や共同研究者等のグループでソースコードをやりとりする時には, 自由にコピペしたり改変して良い.
- 公開したり売ったりするプログラムに関しては次のとおりである.
  - 配布するソースコードに $K\omega$ をそのまま, あるいは改変して含めるときには, そのプログラム本体を LGPL/GPL で配布する.
  - 配布するソースコードに含めず呼び出すだけならばライセンスによらず自由に配布できる.
  - ただしバイナリファイルを配布する場合に, そのバイナリに $K\omega$ が静的リンクされている場合には LGPL/GPL で配布する. 動的リンクされている (したがって $K\omega$ そのものはバイナリに含まれていない) 場合にはライセンスによらず自由に配布できる.

6.2 Autoconf を使わずに Komega をビルドする

このパッケージでは Autotools (Autoconf, Aitomake, Libtool) を使って $K\omega$ をビルドしている. もし再配布するソースコードに $K\omega$ を含めるときに, Autoconf の使用に支障がある場合には, 以下の簡易版の Makefile を使うと良い (タブに注意).

```
F90 = gfortran
FFLAGS = -fopenmp -g -O2 #-D__MPI -D__NO_ZDOT -D__KOMEGA_THREAD

.SUFFIXES :
.SUFFIXES : .o .F90

OBJS = \
komega_cg_c.o \
komega_cg_r.o \
komega_cocg.o \
komega_bicg.o \
komega_math.o \
komega_vals.o
```

```
all:libkomega.a

libkomega.a:$(OBJS)
     ar cr libkomega.a $(OBJS)

.F90.o:
     $(F90) -c $< $(FFLAGS)

clean:
     rm -f *.o *.a *.mod

komega_cg_c.o:komega_math.o
komega_cg_c.o:komega_vals.o
komega_cg_r.o:komega_math.o
komega_cg_r.o:komega_vals.o
komega_cocg.o:komega_math.o
komega_cocg.o:komega_vals.o
komega_bicg.o:komega_math.o
komega_bicg.o:komega_vals.o
komega_math.o:komega_vals.o
```

プリプロセッサマクロ `__MPI`, `__NO_ZDOT`, `__KOMEGA_THREAD` はそれぞれ `configure` のオプション `--with-mpi=yes`, `--disable-zdot`, `--enable-thread` に対応する.

6.3 Lesser General Public License

Chapter 7

Contact

このライブラリについてのご意見, ご質問, バグ報告等ありましたら下記までお問い合わせください。

河村光晶

mkawamura_at_issp.u-tokyo.ac.jp

_at_を@に変えてください.

Chapter 8

参考文献

[1] A. Frommer, Computing **70**, 87 (2003).

[2] S. Yamamoto, T. Sogabe, T. Hoshi, S.-L. Zhang, and T. Fujiwara, J. Phys. Soc. Jpn. **77**, 114713 (2008).