
$K\omega$ マニュアル

リリース 0.2

2017 年 01 月 12 日

Contents

1	概要	2
2	著作権	3
3	アルゴリズム	4
3.1	Seed switch 付き Shifted BiCG 法	4
3.2	Seed switch 付き Shifted COCG 法	5
3.3	Seed switch 付き Shifted CG 法	6
4	プログラム内でのライブラリの動作イメージ	8
4.1	Shifted BiCG ライブラリの動作イメージ	9
4.2	Shifted COCG ライブラリの動作イメージ	10
4.3	Shifted CG ライブラリの動作イメージ	11
5	使用方法	12
5.1	各ルーチンの説明	13
5.2	Shifted BiCG ライブラリを使用したソースコードの例	21
6	Contact	25
	Bibliography	27

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

$$G_{ij}(z) = \langle i | (z\hat{I} - \hat{H})^{-1} | j \rangle \equiv \varphi_i^* \cdot (z\hat{I} - \hat{H})^{-1} \varphi_j \quad (1.1)$$

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

著作権

This software is developed under the support of
“*Project for advancement of software usability in materials science*” by The
Institute for Solid State Physics, The University of Tokyo.

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more details, See ‘COPYING.LESSER’ in the root directory of this library.

Chapter 3

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 [Flommer2003]
- \hat{H} が実数で z が複素数の場合 : Shifted Conjugate Orthogonal Conjugate Gradient(COCG) 法 [Yamamoto2008]
- \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 [Yamamoto2008] 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.

3.1 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 $j = 1 \cdots N_R$

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

$$\tilde{\mathbf{r}} = \text{任意}, \mathbf{r}^{\text{old}} = \tilde{\mathbf{r}}^{\text{old}} = \mathbf{0}$$

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

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

do iteration

◦ シード方程式

$$\rho^{\text{old}} = \rho, \rho = \tilde{\mathbf{r}}^* \cdot \mathbf{r}$$

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

$$\mathbf{q} = (z_{\text{seed}} \hat{I} - \hat{H}) \mathbf{r}$$

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

◦ シフト方程式

do $k = 1 \cdots N_z$

```

 $\pi_k^{\text{new}} = [1 + \alpha(z_k - z_{\text{seed}})]\pi_k - \frac{\alpha\beta}{\alpha^{\text{old}}}(\pi_k^{\text{old}} - \pi_k)$ 
do  $i = 1 \cdots N_L$ 
   $p_{ik} = \frac{1}{\pi_k} \boldsymbol{\varphi}_i^* \cdot \mathbf{r} + \frac{\pi_k^{\text{old}} \pi_k^{\text{old}}}{\pi_k \pi_k} \beta p_{ik}$ 
   $G_{ij}(z_k) = G_{ij}(z_k) + \frac{\pi_k}{\pi_k^{\text{new}}} \alpha p_{ik}$ 
   $\pi_k^{\text{old}} = \pi_k, \pi_k = \pi_k^{\text{new}}$ 
end do  $i$ 
end do  $k$ 
 $\mathbf{q} = \left(1 + \frac{\alpha\beta}{\alpha^{\text{old}}}\right) \mathbf{r} - \alpha \mathbf{q} - \frac{\alpha\beta}{\alpha^{\text{old}}} \mathbf{r}^{\text{old}}, \mathbf{r}^{\text{old}} = \mathbf{r}, \mathbf{r} = \mathbf{q}$ 
 $\mathbf{q} = (z_{\text{seed}}^* \hat{I} - \hat{H}) \tilde{\mathbf{r}}, \mathbf{q} = \left(1 + \frac{\alpha^* \beta^*}{\alpha^{\text{old}*}}\right) \tilde{\mathbf{r}} - \alpha^* \mathbf{q} - \frac{\alpha^* \beta^*}{\alpha^{\text{old}*}} \tilde{\mathbf{r}}^{\text{old}}, \tilde{\mathbf{r}}^{\text{old}} = \tilde{\mathbf{r}}, \tilde{\mathbf{r}} = \mathbf{q}$ 
○ Seed switch
 $|\pi_k|$  が最も小さい  $k$  を探す.  $\rightarrow z_{\text{seed}}, \pi_{\text{seed}}, \pi_{\text{seed}}^{\text{old}}$ 
 $\mathbf{r} = \mathbf{r}/\pi_{\text{seed}}, \mathbf{r}^{\text{old}} = \mathbf{r}^{\text{old}}/\pi_{\text{seed}}^{\text{old}}, \tilde{\mathbf{r}} = \tilde{\mathbf{r}}/\pi_{\text{seed}}^*, \tilde{\mathbf{r}}^{\text{old}} = \tilde{\mathbf{r}}^{\text{old}}/\pi_{\text{seed}}^{\text{old}*}$ 
 $\alpha = (\pi_{\text{seed}}^{\text{old}}/\pi_{\text{seed}})\alpha, \rho = \rho/(\pi_{\text{seed}}^{\text{old}}\pi_{\text{seed}}^{\text{old}})$ 
 $\{\pi_k = \pi_k/\pi_{\text{seed}}, \pi_k^{\text{old}} = \pi_k^{\text{old}}/\pi_{\text{seed}}^{\text{old}}\}$ 
if(  $|\mathbf{r}| < \text{Threshold}$ ) exit
end do iteration
end do  $j$ 

```

3.2 Seed switch 付き Shifted COCG 法

BiCG のアルゴリズムで, $\tilde{\mathbf{r}} = \mathbf{r}^*, \tilde{\mathbf{r}}^{\text{old}} = \mathbf{r}^{\text{old}*}$ 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 $j = 1 \cdots N_R$

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

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

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

do iteration

○ シード方程式

$\rho^{\text{old}} = \rho, \rho = \mathbf{r} \cdot \mathbf{r}$

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

$\mathbf{q} = (z_{\text{seed}} \hat{I} - \hat{H}) \mathbf{r}$

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

○ シフト方程式

do $k = 1 \cdots N_z$

```

 $\pi_k^{\text{new}} = [1 + \alpha(z_k - z_{\text{seed}})]\pi_k - \frac{\alpha\beta}{\alpha^{\text{old}}}(\pi_k^{\text{old}} - \pi_k)$ 
do  $i = 1 \cdots N_L$ 
   $p_{ik} = \frac{1}{\pi_k} \boldsymbol{\varphi}_i^* \cdot \mathbf{r} + \frac{\pi_k^{\text{old}} \pi_k^{\text{old}}}{\pi_k \pi_k} \beta p_{ik}$ 
   $G_{ij}(z_k) = G_{ij}(z_k) + \frac{\pi_k}{\pi_k^{\text{new}}} \alpha p_{ik}$ 
   $\pi_k^{\text{old}} = \pi_k, \pi_k = \pi_k^{\text{new}}$ 
end do  $i$ 
end do  $k$ 
 $\mathbf{q} = \left(1 + \frac{\alpha\beta}{\alpha^{\text{old}}}\right) \mathbf{r} - \alpha \mathbf{q} - \frac{\alpha\beta}{\alpha^{\text{old}}} \mathbf{r}^{\text{old}}, \mathbf{r}^{\text{old}} = \mathbf{r}, \mathbf{r} = \mathbf{q}$ 
◦ Seed switch
 $|\pi_k|$  が最も小さい  $k$  を探す.  $\rightarrow z_{\text{seed}}, \pi_{\text{seed}}, \pi_{\text{seed}}^{\text{old}}$ 
 $\mathbf{r} = \mathbf{r} / \pi_{\text{seed}}, \mathbf{r}^{\text{old}} = \mathbf{r}^{\text{old}} / \pi_{\text{seed}}^{\text{old}}$ 
 $\alpha = (\pi_{\text{seed}}^{\text{old}} / \pi_{\text{seed}}) \alpha, \rho = \rho / (\pi_{\text{seed}}^{\text{old}} \pi_{\text{seed}}^{\text{old}})$ 
 $\{\pi_k = \pi_k / \pi_{\text{seed}}, \pi_k^{\text{old}} = \pi_k^{\text{old}} / \pi_{\text{seed}}^{\text{old}}\}$ 
if(  $|\mathbf{r}| < \text{Threshold}$ ) exit
end do iteration
end do  $j$ 

```

3.3 Seed switch 付き Shifted CG 法

BiCG のアルゴリズムで, $\tilde{\mathbf{r}} = \mathbf{r}, \tilde{\mathbf{r}}^{\text{old}} = \mathbf{r}^{\text{old}}$ 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 $j = 1 \cdots N_R$

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

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

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

do iteration

◦ シード方程式

$\rho^{\text{old}} = \rho, \rho = \mathbf{r}^* \cdot \mathbf{r}$

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

$\mathbf{q} = (z_{\text{seed}} \hat{I} - \hat{H}) \mathbf{r}$

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

◦ シフト方程式

do $k = 1 \cdots N_z$

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

do $i = 1 \cdots N_L$

```


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


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


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

    end do  $i$ 

    end do  $k$ 

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

    ◦ Seed switch
     $|\pi_k|$  が最も小さい  $k$  を探す.  $\rightarrow z_{\text{seed}}, \pi_{\text{seed}}, \pi_{\text{seed}}^{\text{old}}$ 
 $\mathbf{r} = \mathbf{r} / \pi_{\text{seed}}, \mathbf{r}^{\text{old}} = \mathbf{r}^{\text{old}} / \pi_{\text{seed}}^{\text{old}}$ 
 $\alpha = (\pi_{\text{seed}}^{\text{old}} / \pi_{\text{seed}}) \alpha, \rho = \rho / \pi_{\text{seed}}^{\text{old}^2}$ 
 $\{ \pi_k = \pi_k / \pi_{\text{seed}}, \pi_k^{\text{old}} = \pi_k^{\text{old}} / \pi_{\text{seed}}^{\text{old}} \}$ 
    if(  $|\mathbf{r}| < \text{Threshold}$  ) exit

    end do iteration

end do  $j$ 

```

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 $\alpha, \beta, z_{seed}, \mathbf{r}^L$ を取り出す.
- komega_bicg_getvec, komega_cocg_getvec, komega_cg_c_getvec, komega_cg_r_getvec
 $\mathbf{r}, \mathbf{r}^{\text{old}}, \tilde{\mathbf{r}}, \tilde{\mathbf{r}}^{\text{old}}$ 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. \mathbf{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.

$$\mathbf{r}^L = \hat{P}^\dagger \mathbf{r}, \quad \hat{P} \equiv (\varphi_1, \dots, \varphi_{N_L})$$

- 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 $\text{status}(1) < 0$ 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 $\alpha, \beta, \mathbf{r}^L$ 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 itermx 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  $\mathbf{v}_{12}, \mathbf{v}_{13}, \mathbf{v}_2, \mathbf{v}_3, \{\mathbf{x}_k\}, \mathbf{r}^L, \mathbf{v}_2 = \varphi_j$ 
komega_bicg_init(N_H, N_L, N_z, x, z, itermx, threshold) start
    Allocate  $\mathbf{v}_3, \mathbf{v}_5, \{\pi_k\}, \{\pi_k^{\text{old}}\}, \{\mathbf{p}_k\}$ 
    Copy  $\{z_k\}$ 
    itermx  $\neq 0$  ならば  $\alpha, \beta, \mathbf{r}^L$  を保存する配列を確保する.
     $\mathbf{v}_4 = \mathbf{v}_2^*$  (任意),  $\mathbf{v}_3 = \mathbf{v}_5 = \mathbf{0}$ ,
     $\mathbf{p}_k = \mathbf{x}_k = \mathbf{0} (k = 1 \dots N_z), \pi_k = \pi_k^{\text{old}} = 1 (k = 1 \dots N_z)$ 
     $\rho = \infty, \alpha = 1, z_{\text{seed}} = 0$ 
    ( $\mathbf{v}_2 \equiv \mathbf{r}, \mathbf{v}_3 \equiv \mathbf{r}^{\text{old}}, \mathbf{v}_4 \equiv \tilde{\mathbf{r}}, \mathbf{v}_5 \equiv \tilde{\mathbf{r}}^{\text{old}}.$ )
komega_bicg_init finish
do iteration
     $\mathbf{r}^L = \hat{P}^\dagger \mathbf{v}_2$ 
     $\mathbf{v}_{12} = \hat{H} \mathbf{v}_2, \mathbf{v}_{14} = \hat{H} \mathbf{v}_4$  [ $(\mathbf{v}_{12}, \mathbf{v}_{14}) = \hat{H}(\mathbf{v}_2, \mathbf{v}_4)$  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
         $\rho^{\text{old}} = \rho, \rho = \mathbf{v}_4^* \cdot \mathbf{v}_2$ 
         $\beta = \rho / \rho^{\text{old}}$ 
         $\mathbf{v}_{12} = z_{\text{seed}} \mathbf{v}_2 - \mathbf{v}_{12}, \mathbf{v}_{14} = z_{\text{seed}}^* \mathbf{v}_4 - \mathbf{v}_{14}$ 
         $\alpha^{\text{old}} = \alpha, \alpha = \frac{\rho}{\mathbf{v}_3^* \cdot \mathbf{v}_{12} - \beta \rho / \alpha}$ 
        ◦ シフト方程式
        do  $k = 1 \dots N_z$ 
             $\pi_k^{\text{new}} = [1 + \alpha(z_k - z_{\text{seed}})]\pi_k - \frac{\alpha\beta}{\alpha^{\text{old}}}(\pi_k^{\text{old}} - \pi_k)$ 
             $\mathbf{p}_k = \frac{1}{\pi_k} \mathbf{r}^L + \frac{\pi_k^{\text{old}} \pi_k^{\text{old}}}{\pi_k \pi_k} \beta \mathbf{p}_k$ 

```

```


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


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

end do k

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


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

o Seed switch
 $|\pi_k|$  が最も小さい  $k$  を探す.  $\rightarrow z_{\text{seed}}, \pi_{\text{seed}}, \pi_{\text{seed}}^{\text{old}}$ 
 $\mathbf{v}_2 = \mathbf{v}_2 / \pi_{\text{seed}}, \mathbf{v}_3 = \mathbf{v}_3 / \pi_{\text{seed}}^{\text{old}}, \mathbf{v}_4 = \mathbf{v}_4 / \pi_{\text{seed}}^*, \mathbf{v}_5 = \mathbf{v}_5 / \pi_{\text{seed}}^{\text{old}*}$ 
 $\alpha = (\pi_{\text{seed}}^{\text{old}} / \pi_{\text{seed}}) \alpha, \rho = \rho / (\pi_{\text{seed}}^{\text{old}} \pi_{\text{seed}}^{\text{old}})$ 
 $\{\pi_k = \pi_k / \pi_{\text{seed}}, \pi_k^{\text{old}} = \pi_k^{\text{old}} / \pi_{\text{seed}}^{\text{old}}\}$ 
komega_bicg_update finish
if(status(1) < 0 (これは  $|\mathbf{v}_2| < \text{Threshold}$  となった事を意味する)) exit
end do iteration
komega_bicg_finalize start
Dealocate  $\mathbf{v}_4, \mathbf{v}_5, \{\pi_k\}, \{\pi_k^{\text{old}}\}, \{\mathbf{p}_k\}$ 
komega_bicg_finalize finish

```

4.2 Shifted COCG ライブラリの動作イメージ

```

Allocate  $\mathbf{v}_1, \mathbf{v}_2, \{\mathbf{x}_k\}, \mathbf{r}^{\text{L}} \mathbf{v}_2 = \varphi_j$ 
komega_cocg_init(N_H, N_L, N_z, x, z, itermx, threshold) start
Allocate  $\mathbf{v}_3, \{\pi_k\}, \{\pi_k^{\text{old}}\}, \{\mathbf{p}_k\}$ 
Copy  $\{z_k\}$ 
itermx  $\neq 0$  ならば  $\alpha, \beta, \mathbf{r}^{\text{L}}$  を保存する配列を確保する.
 $\mathbf{v}_3 = \mathbf{0},$ 
 $\mathbf{p}_k = \mathbf{x}_k = \mathbf{0} (k = 1 \dots N_z), \pi_k = \pi_k^{\text{old}} = 1 (k = 1 \dots N_z)$ 
 $\rho = \infty, \alpha = 1, \beta = 0, z_{\text{seed}} = 0$ 
( $\mathbf{v}_2 \equiv \mathbf{r}, \mathbf{v}_3 \equiv \mathbf{r}^{\text{old}}.$ )
komega_cocg_init finish
do iteration
 $\mathbf{r}^{\text{L}} = \hat{P}^\dagger \mathbf{v}_2$ 
 $\mathbf{v}_1 = \hat{H} \mathbf{v}_2$ 
komega_cocg_update(v_1, v_2, x, r_small, status) start
o シード方程式
 $\rho^{\text{old}} = \rho, \rho = \mathbf{v}_2 \cdot \mathbf{v}_2$ 
 $\beta = \rho / \rho^{\text{old}}$ 

```

```

 $\mathbf{v}_1 = z_{\text{seed}} \mathbf{v}_2 - \mathbf{v}_1$ 
 $\alpha^{\text{old}} = \alpha, \alpha = \frac{\rho}{\mathbf{v}_2 \cdot \mathbf{v}_1 - \beta \rho / \alpha}$ 
◦ シフト方程式
do  $k = 1 \dots N_z$ 
     $\pi_k^{\text{new}} = [1 + \alpha(z_k - z_{\text{seed}})]\pi_k - \frac{\alpha\beta}{\alpha^{\text{old}}}(\pi_k^{\text{old}} - \pi_k)$ 
     $\mathbf{p}_k = \frac{1}{\pi_k} \mathbf{r}^L + \frac{\pi_k^{\text{old}} \pi_k^{\text{old}}}{\pi_k \pi_k} \beta \mathbf{p}_k$ 
     $\mathbf{x}_k = \mathbf{x}_k + \frac{\pi_k}{\pi_k^{\text{new}}} \alpha \mathbf{p}_k$ 
     $\pi_k^{\text{old}} = \pi_k, \pi_k = \pi_k^{\text{new}}$ 
end do  $k$ 
 $\mathbf{v}_1 = \left(1 + \frac{\alpha\beta}{\alpha^{\text{old}}}\right) \mathbf{v}_2 - \alpha \mathbf{v}_1 - \frac{\alpha\beta}{\alpha^{\text{old}}} \mathbf{v}_3$ 
 $\mathbf{v}_3 = \mathbf{v}_2, \mathbf{v}_2 = \mathbf{v}_1$ 
◦ Seed switch
 $|\pi_k|$  が最も小さい  $k$  を探す.  $\rightarrow z_{\text{seed}}, \pi_{\text{seed}}, \pi_{\text{seed}}^{\text{old}}$ 
 $\mathbf{v}_2 = \mathbf{v}_2 / \pi_{\text{seed}}, \mathbf{v}_3 = \mathbf{v}_3 / \pi_{\text{seed}}^{\text{old}}$ 
 $\alpha = (\pi_{\text{seed}}^{\text{old}} / \pi_{\text{seed}}) \alpha, \rho = \rho / (\pi_{\text{seed}}^{\text{old}} \pi_{\text{seed}}^{\text{old}})$ 
 $\{\pi_k = \pi_k / \pi_{\text{seed}}, \pi_k^{\text{old}} = \pi_k^{\text{old}} / \pi_{\text{seed}}^{\text{old}}\}$ 
komega_cocg_update finish
if(status(1) < 0 (これは  $|\mathbf{v}_2| < \text{Threshold}$  となった事を意味する)) exit
end do iteration
komega_cocg_finalize start
    Deallocate  $\mathbf{v}_3, \{\pi_k\}, \{\pi_k^{\text{old}}\}, \{\mathbf{p}_k\}$ 
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.

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). MPI/Hybrid 並列版のルーチンを利用するときには、

```
USE pkomega_cg_r
USE pkomega_cg_c
USE pkomega_cocg
USE pkomega_bicg
```

のようにする.

C/C++で書かれたプログラムから呼び出すときには、


```
#include komega_cg_r.h
#include komega_cg_c.h
#include komega_cocg.h
#include komega_bicg.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。MPI/Hybrid 並列版のルーチンを利用するときには、

```
#include pkomega_cg_r.h
#include pkomega_cg_c.h
#include pkomega_cocg.h
#include pkomega_bicg.h
```

のようにする。また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 シリアル/OpenMP 版

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

Fortran MPI/Hybrid 並列版

```
CALL pkomega_cg_r_init(ndim, nl, nz, x, z, itermax, threshold, comm)
CALL pkomega_cg_c_init(ndim, nl, nz, x, z, itermax, threshold, comm)
CALL pkomega_cocg_init(ndim, nl, nz, x, z, itermax, threshold, comm)
CALL pkomega_bicg_init(ndim, nl, nz, x, z, itermax, threshold, comm)
```

C/C++ シリアル/OpenMP 版

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

C/C++ MPI/Hybrid 並列版

```
pkomega_cg_r_init(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm);
pkomega_cg_c_init(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm);
pkomega_cocg_init(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm);
pkomega_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) :: comm
```

MPI/Hybrid 並列版のみ. MPI のコミュニケーター (MPI_COMM_WORLD など) 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 (シリアル/OpenMP 版)

```
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)
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)
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)
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)
```

Fortran (MPI/ハイブリッド並列版)

```

CALL pkomega_cg_r_restart(ndim, nl, nz, x, z, itermax, threshold, comm, status, &
& iter_old, v2, v12, alpha_save, beta_save, z_seed, r_l_save)
CALL pkomega_cg_c_restart(ndim, nl, nz, x, z, itermax, threshold, comm, status, &
& iter_old, v2, v12, alpha_save, beta_save, z_seed, r_l_save)
CALL pkomega_cocg_restart(ndim, nl, nz, x, z, itermax, threshold, comm, status, &
& iter_old, v2, v12, alpha_save, beta_save, z_seed, r_l_save)
CALL pkomega_bicg_restart(ndim, nl, nz, x, z, itermax, threshold, comm, status, &
& iter_old, v2, v12, v4, v14, alpha_save, beta_save, &
& z_seed, r_l_save)

```

C/C++ (シリアル/OpenMP 版)

```

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);
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);
komega_cocg_restart(&ndim, &nl, &nz, x, z, &itermax, &threshold, status, &
& &iter_old, v2, v12, alpha_save, beta_save, &z_seed, r_l_save);
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);

```

C/C++ (MPI/ハイブリッド並列版)

```

pkomega_cg_r_restart(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm, status, &
& &iter_old, v2, v12, alpha_save, beta_save, &z_seed, r_l_save);
pkomega_cg_c_restart(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm, status, &
& &iter_old, v2, v12, alpha_save, beta_save, &z_seed, r_l_save);
pkomega_cocg_restart(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm, status, &
& &iter_old, v2, v12, alpha_save, beta_save, &z_seed, r_l_save);
pkomega_bicg_restart(&ndim, &nl, &nz, x, z, &itermax, &threshold, &comm, status, &
& &iter_old, v2, v12, v4, v14, alpha_save, beta_save, &
& &z_seed, r_l_save);

```

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) :: 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 (シリアル/OpenMPI 版)

```
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)
```

Fortran (MPI/ハイブリッド並列版)

```
CALL pkomega_cg_r_update(v12, v2, x, r_l, status)
CALL pkomega_cg_c_update(v12, v2, x, r_l, status)
CALL pkomega_cocg_update(v12, v2, x, r_l, status)
CALL pkomega_bicg_update(v12, v2, v14, v4, x, r_l, status)
```

C/C++ (シリアル/OpenMPI 版)

```
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);
```

C/C++ (MPI/ハイブリッド並列版)

```
pkomega_cg_r_update(v12, v2, x, r_l, status);
pkomega_cg_c_update(v12, v2, x, r_l, status);
pkomega_cocg_update(v12, v2, x, r_l, status);
pkomega_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 (シリアル/OpenMP 版)

```
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)
```

Fortran (MPI/ハイブリッド並列版)

```
CALL pkomega_cg_r_getcoef(alpha_save, beta_save, z_seed, r_l_save)
CALL pkomega_cg_c_getcoef(alpha_save, beta_save, z_seed, r_l_save)
CALL pkomega_cocg_getcoef(alpha_save, beta_save, z_seed, r_l_save)
CALL pkomega_bicg_getcoef(alpha_save, beta_save, z_seed, r_l_save)
```

C/C++ (シリアル/OpenMP 版)

```
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);
```

C/C++ (MPI/ハイブリッド並列版)

```
pkomega_cg_r_getcoef(alpha_save, beta_save, &z_seed, r_l_save);
pkomega_cg_c_getcoef(alpha_save, beta_save, &z_seed, r_l_save);
pkomega_cocg_getcoef(alpha_save, beta_save, &z_seed, r_l_save);
pkomega_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 (シリアル/OpenMP 版)

```
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)
```

Fortran (MPI/ハイブリッド並列版)

```
CALL pkomega_cg_r_getvec(r_old)
CALL pkomega_cg_c_getvec(r_old)
CALL pkomega_cocg_getvec(r_old)
CALL pkomega_bicg_getvec(r_old, r_tilde_old)
```

C/C++ (シリアル/OpenMP 版)

```
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);
```

C/C++ (MPI/ハイブリッド並列版)

```
pkomega_cg_r_getvec(r_old);
pkomega_cg_c_getvec(r_old);
pkomega_cocg_getvec(r_old);
pkomega_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 (シリアル/OpenMP 版)

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

Fortran (MPI/ハイブリッド並列版)

```
CALL pkomega_cg_r_getresidual(res)
CALL pkomega_cg_c_getresidual(res)
CALL pkomega_cocg_getresidual(res)
CALL pkomega_bicg_getresidual(res)
```

C/C++ (シリアル/OpenMP 版)

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

C/C++ (MPI/ハイブリッド並列版)

```
pkomega_cg_r_getresidual(res);
pkomega_cg_c_getresidual(res);
pkomega_cocg_getresidual(res);
pkomega_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 (シリアル/OpenMP 版)

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

Fortran (MPI/ハイブリッド並列版)

```
CALL pkomega_cg_r_finalize()
CALL pkomega_cg_c_finalize()
CALL pkomega_cocg_finalize()
CALL pkomega_bicg_finalize()
```


C/C++ (シリアル/OpenMP 版)

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

C/C++ (MPI/ハイブリッド並列版)

```
pkomega_cg_r_finalize();
pkomega_cg_c_finalize();
pkomega_cocg_finalize();
pkomega_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
& itermx,   & ! Max. number of iteration
& iter_old  & ! Number of iteration 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_1(:), & ! (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., restart 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

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.

Bibliography

[Flommer2003] 1. Frommer, Computing **70**, 87 (2003).

[Yamamoto2008] 19. Yamamoto, T. Sogabe, T. Hoshi, S.-L. Zhang, and T. Fujiwara, J. Phys. Soc. Jpn. **77**, 114713 (2008).