# Komega Documentation

***Release 0.1***

**kawamura**

January 10, 2017

Contents

Contents:

CHAPTER 1

# Overview

This document is a manual for $K(\omega)$ which is the library to solve the shifted linear equation within the Krylov subspace. This library provides routines to solve the following shifted linear equation (with the projection),

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

The source codes of $K(\omega)$ is written in FORTRAN and requires the BLAS Level 1 routines.

CHAPTER 2

# Copyright

CHAPTER 3

# Algorithm

This library provides the four kinds of numerical solvers. The kind of solvers is selected under the condition whether the Hamiltonian $\hat{H}$ and/or the frequency $z$ are complex or real number. It is noted that $\hat{H}$ must be Hermitian (symmetric) for complex (real) number.

- ($\hat{H}$, $z$ ) = (complex, complex): Shifted Bi-Conjugate Gradient(BiCG) method [Flommer2003]
- ($\hat{H}$, $z$ ) = (real, complex): Shifted Conjugate Orthogonal Conjugate Gradient(COCG) method [Yamamoto2008]
- ($\hat{H}$, $z$ ) = (complex, real): Shifted Conjugate Gradient(CG) method (using complex vector)
- ($\hat{H}$, $z$ ) = (real, real): Shifted Conjugate Gradient(CG) method (using real vector)

For above methods, seed switching [Yamamoto2008] is adopted. Hereafter, the number of the left (right) side vector, :math:‘ {boldsymbol varphi}_{i}‘ (:math:‘ {boldsymbol varphi}_{j}‘), in Eq.(1) is indicated by $N_L$ ($N_R$). The details of each algorithm are written as follows.

## 3.1 Shifted BiCG method with seed switching technique

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

do $j = 1 \cdots N_R$

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

$\tilde{\boldsymbol{r}}$ = an arbitrary vector, $\boldsymbol{r}^{\rm old} = \tilde{\boldsymbol{r}}^{\rm old} = \boldsymbol{0}$

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

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

do iteration

◦ Seed equation

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

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

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

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

◦ Shifted equation

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

Search $k$ which gives the smallest $|\pi_k|$ . $\rightarrow z_{\rm seed},\ \pi_{\rm seed},\ \pi_{\rm seed}^{\rm old}$

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

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

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

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

end do iteration

end do $j$

## 3.2 Shifted COCG method with seed switching technique

This method is obtained by $\tilde{\boldsymbol{r}} = \boldsymbol{r}^*,\ \tilde{\boldsymbol{r}}^{\rm old} = \boldsymbol{r}^{\rm old*}$ in the BiCG method.

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

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

∘ Seed equation

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

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

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

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

∘ Shifted equation

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

Search $k$ which gives the smallest $|\pi_k|$ . $\rightarrow z_{\rm seed},\ \pi_{\rm seed},\ \pi_{\rm seed}^{\rm old}$

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

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

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

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

end do iteration

end do $j$

## 3.3 Shifted CG method with seed switching technique

This method is obtained by $\tilde{\boldsymbol{r}} = \boldsymbol{r},\ \tilde{\boldsymbol{r}}^{\rm old} = \boldsymbol{r}^{\rm old}$ in the BiCG method.

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

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

∘ Seed equation

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

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

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

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

∘ Shifted equation

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

Search $k$ which gives the minimum value of $|\pi_k|$ . $\rightarrow z_{\rm seed},\ \pi_{\rm seed},\ \pi_{\rm seed}^{\rm old}$

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

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

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

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

end do iteration

end do $j$

CHAPTER 4

# Schematic workflow of this library

In the following description, the loop for $N_R$ is omitted for simplicity and instead of $G_{ij}(z_k)$, the $N_L$-dimensional vector $\mathbf{x}_k$ is obtained by using the library.

The essential procedures to use the library.

- Give the vector size $N_H$ corresponding to the size of the Hilbert space and the number of the frequency $z$.
- Allocate the two vectors (in the case of BiCG method, four vectors) with the size of $N_H$.
- Give the function for the Hamiltonian-vector production.
- Allocate the solution vectors. It is noted that the number of the solution vectors is not always equal to $N_H \times N_z$. In fact, the number in the previous section is $N_L \times N_z$. In this case, the (bi-)conjugate gradient vector $\mathbf{p}_k$ is the vector with $N_L$-dimension and $N_z$ array. The transformation of the dimension of the residual vector from $N_H$ to $N_L$ must be done explicitly.

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

- If the result converges, `komega_????_update` return the first element of `status` as a negative integer. Therefore, please exit loop when `status(1) < 0` .
- The 2-norm is used for the convergence check in the routine `komega_????_update`. Therefore, if 2-norms of residual vectors at all shift points becomes smaller than K, this routine assumes the result is converged.
- Conserve $\alpha, \beta, \mathbf{r}^{\mathrm{L}}$ at each iteration and to use the above vectors later, set the maximum number of iteration `itermax`.

The names of the routines is defined as follows.

- `BiCG_init`, `COCG_init`, `CG_C_init`, `CG_R_init`

  Set the initial conditions such as the allocation of variables used in the library.

- `BiCG_update`, `COCG_update`, `CG_C_update`, `CG_R_update`

  These routines are called in the iteration to update the solution vectors.

- `BiCG_finalize`, `COCG_finalize`, `CG_C_finalize`, `CG_R_finalize`

  Release the allocated vectors in the library.

- `BiCG_getcoef`, `COCG_getcoef`, `CG_C_getcoef`, `CG_R_getcoef`

  Get the $\alpha$, $\beta$, $z_{\mathrm{seed}}$, $\mathbf{r}^{\mathrm{L}}$ conserved at each iteration.

- `BiCG_getvec`, `COCG_getvec`, `CG_C_getvec`, `CG_R_getvec`

  Get the vectors $\boldsymbol{r}$, $\boldsymbol{r}^{\mathrm{old}}$, $\tilde{\boldsymbol{r}}$, $\tilde{\boldsymbol{r}}^{\mathrm{old}}$.

- `BiCG_restart`, `COCG_restart`, `CG_C_restart`, `CG_R_restart`

## 4.1 The schematic workflow of shifted BiCG library

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 $\{z_k\}$

If `itermax` $\neq 0$ , allocate arrays to store $\alpha$, $\beta$, and:math:*{bf r}^{rm L}* at each iteration.

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

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

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

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

`komega_BiCG_init` finish

do iteration

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

$\boldsymbol{v}_{12} = \hat{H}\boldsymbol{v}_2$, $\boldsymbol{v}_{14} = \hat{H}\boldsymbol{v}_4$ [Or $(\boldsymbol{v}_{12}, \boldsymbol{v}_{14}) = \hat{H}(\boldsymbol{v}_2, \boldsymbol{v}_4)$ ]

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

∘ Seed equation

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

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

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

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

∘ Shifted equation

do $k = 1 \cdots N_z$

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

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

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

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

end do $k$

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

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

∘ Seed switch

Search $k$ which gives the smallest $|\pi_k|$ . $\rightarrow z_{\mathrm{seed}},\ \pi_{\mathrm{seed}},\ \pi_{\mathrm{seed}}^{\mathrm{old}}$

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

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

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

`komega_BiCG_update` finish

if(status(1) < 0 (This indicates $|\boldsymbol{v}_2| <$ Threshold)) 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 The schematic workflow of shifted COCG library

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

`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 $\{z_k\}$

If `itermax` $\neq$ `0` , allocate arrays to store $\alpha$, $\beta$, and $\mathbf{r}^{\rm L}$ .

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

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

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

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

`COCG_init` finish

do iteration

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

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

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

∘ Seed equationw

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

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

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

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

∘ Shifted equations

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

Search $k$ which gives the smallest |*pi_k*| . $\rightarrow z_{\rm seed},\ \pi_{\rm seed},\ \pi^{\rm old}_{\rm seed}$

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

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

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

`COCG_update` finish

if(status(1) < 0 (This indicates $|\boldsymbol{v}_2| <$ Threshold.)) exit

end do iteration

`COCG_finalize` start

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

`COCG_finalize` finish

## 4.3 The schematic workflow of shifted CG library

The workflow is the same as that of the shifted COCG library.

CHAPTER 5

# Usage

The calculation is done to utilize functions by the following procedures.

- Initialization (init function)
- Update (update function)
- Output numerical results (call getcoef, getvec functions and output informations)
- Finalization (finalize function)

The restart calculation can be done by the following procedures.

- Initialization(restart function)
- Update (update function)
- Output numerical results (call getcoef, getvec functions and output informations)
- Finalization (finalize function)

For FORTRAN, the modules can be called by

```
USE komega_????
```

"????" is selected from the following words(methods) "CG_R", "CG_C", "COCG", "BiCG". To utilize routines of MPI / Hybrid parallelization version, the modules can be called as folows:

```
USE pkomega_????
```

When we call $K\omega$ from C/C++ codes, we should include the header file as

```
#include komega_????.h
```

Scaler arguments should be passed as pointers. For MPI/Hybrid parallelized routine, the above line becomes

```
#include pkomega_????.h
```

Also the communicator argument for the routine should be transformed from the C/C++'s one to the fortran's one as follows.

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

## 5.1 Details of each routines

### 5.1.1 ????_init, p????_init

Set and initialize internal variables in libraries. These routines should be called first before solving the shifted equation.

Syntax

Fortran (Serial/OpenMP)

```
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 parallel)

```
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++ Serial/OpenMP

```
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 parallel

```
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);
```

Parameters

- `ndim`

  `INTEGER`. Scalar. Input. The dimension of solution vectors for the linearized equation.

- `nl`

  `INTEGER`. Scalar. Input. The dimension of projected solution vectors.

- `nz`

  `INTEGER`. Scalar. Input. The number of shifted points.

- `x`

  `DOUBLE PRECISION` (for `CG_R_init`), `DOUBLE COMPLEX` (for other cases). The array with the length of `nl*nz`. Output. The solution vector. In this procedure, `0` vector is returned.

- `z`

  `DOUBLE PRECISION` (for `CG_R_init`, `CG_C_init`), `DOUBLE COMPLEX` (for other cases). The array with the length of `nz`. Input. Shifted points.

- `itermax`

  `INTEGER`. Scalar. Input. The maximum iteration number for allocating arrays for the restart calculation. When `itermax=0` , these arrays are not allocated, and the restart calculation described later becomes unavailable.

- `threshold`

  `DOUBLE PRECISION`. Scalar. Input. The threshold value for the convergence determination. When the 2-norm of the residual vector for the seed equation becomes smaller than this value, the calculation is finished.

- `comm`

  `INTEGER`. Scalar. Input. Only for MPI / Hybrid parallelization version. Communicators for MPI such as `MPI_COMM_WORLD`.

### 5.1.2 `komega_????_restart`, `pkomega_????_restart`

For the restart calculation, these routines are used instead of `?_init`. Set and initialize internal variables in libraries. These routines should be called first before solving the shifted equation.

Syntax

Fortran (Serial/OpenMP)

```
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/hybrid parallel)

```
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++ (Serial/OpenMP)

```
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/hybrid parallel)

```
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);
```

Parameters

- `ndim, nl, nz, x, z, itermax, threshold, comm`

  The definition is same as `?_init`. See the parameters in `?_init`.

- `status`

  `INTEGER`. The array with the length of `3`. Output. The error code is returned.

  **First component(`status(1)`)** If the solution is converged or a breakdown occurs, the current total number of iteration with the minus sign is returned. In other cases, this routine returns the current total number of iteration. The calculation is continuable only when `status(1)` is the positive value; otherwise the result is meaningless even if the calculation is continued.

  **Second component(`status(2)`)** `1` is returned if `itermax` is set as a finite value and the convergence condition is not satisfied at the `itermax`-th iteration. `2` is returned if $\alpha$ diverges. `3` is returned if $\pi_{\mathrm{seed}}$ becomes 0. In the case of `COCG_restart` or `BiCG_restart`, `4` is returned if the residual vector and the shadow residual vector are orthogonal. In other cases, `0` is returned.

  **Third component(`status(3)`)** The index of the seed point is returned.

- `iter_old`

  `INTEGER`. Scalar. Input. The number of iteration for the previous calculation.

- `v2`

  `DOUBLE PRECISION` (for `CG_R_restart`), `DOUBLE COMPLEX` (for other cases). The array with the length of `ndim`. Input. The residual vector at the last step for the previous calculation.

- `v12`

  `DOUBLE PRECISION` (for `CG_R_restart`), `DOUBLE COMPLEX` (for other cases). The array with the length of `ndim`. Input. The residual vector at the second from the last step for the previous calculation.

- `alpha_save`

  `DOUBLE PRECISION` (for `CG_R_restart`, `CG_C_restart`), `DOUBLE COMPLEX` (for other cases). The array with the length of `iter_old`. Input. The parameters $\alpha$ obtained by the previous calculation at each steps by (Bi)CG methods.

- `beta_save`

  `DOUBLE PRECISION` (for `CG_R_restart`, `CG_C_restart`), `DOUBLE COMPLEX` (for other cases). The array with the length of`iter_old`. Input. The parameters $\beta$ obtained by the previous calculation at each steps by (Bi)CG methods.

- `z_seed`

  `DOUBLE PRECISION` (for `CG_R_restart`, `CG_C_restart`), `DOUBLE COMPLEX` (for other cases). Scalar. Input. The value of the seed shift for the previous calculation.

- `r_l_save`

  `DOUBLE PRECISION` (for `CG_R_restart`), `DOUBLE COMPLEX` (for other cases). The array with the length of `nl*iter_old`. Input. The projected residual vector at each iteration for the previous calculation.

- v4

  Only used for BiCG_restart. DOUBLE COMPLEX. The array with the length of ndim. Input. The shadow residual vector at the last step for the previous calculation.

- v14

  Only used for BiCG_restart. DOUBLE COMPLEX. The array with the length of ndim. Input. The shadow residual vector at the second last step for the previous calculation.

### 5.1.3 komega_????_update, pkomega_????_update

It is called alternately with the matrix-vector product in the loop and updates the solution.

Syntax

Fortran (Serial/OpenMPI)

```
CALL komega_CG_R_update(v12, v2, x, r_1, status)
CALL komega_CG_C_update(v12, v2, x, r_1, status)
CALL komega_COCG_update(v12, v2, x, r_1, status)
CALL komega_BiCG_update(v12, v2, v14, v4, x, r_1, status)
```

Fortran (MPI/hybrid parallel)

```
CALL pkomega_CG_R_update(v12, v2, x, r_1, status)
CALL pkomega_CG_C_update(v12, v2, x, r_1, status)
CALL pkomega_COCG_update(v12, v2, x, r_1, status)
CALL pkomega_BiCG_update(v12, v2, v14, v4, x, r_1, status)
```

C/C++ (Serial/OpenMPI)

```
komega_CG_R_update(v12, v2, x, r_1, status);
komega_CG_C_update(v12, v2, x, r_1, status);
komega_COCG_update(v12, v2, x, r_1, status);
komega_BiCG_update(v12, v2, v14, v4, x, r_1, status);
```

C/C++ (MPI/hybrid parallel)

```
pkomega_CG_R_update(v12, v2, x, r_1, status);
pkomega_CG_C_update(v12, v2, x, r_1, status);
pkomega_COCG_update(v12, v2, x, r_1, status);
pkomega_BiCG_update(v12, v2, v14, v4, x, r_1, status);
```

Parameters

- v12

  DOUBLE PRECISION (for CG_R_update), DOUBLE COMPLEX (for other cases). The array with the length of ndim. In/Output. The product of the residual vector (v2) and the matrix. This routine returns the 2-norm of the updated residual vector as a first element of this array. This returned value is used, for examples, for printing the convergence profile.

- v2

  DOUBLE PRECISION (for CG_R_update), DOUBLE COMPLEX (for other cases). The array with the length of ndim. In/Output. The residual vector is input and the updated residual vector is output.

- v14

  Only used for BiCG_update. DOUBLE COMPLEX. The array with the length of ndim. In/Output. The product of the shadow residual vector (v4) and the matrix is input.

- `v4`

  Only used for `BiCG_update`. `DOUBLE COMPLEX`. The array with the length of `ndim`. In/Output. The shadow residual vector is input and the updated vector is output.

- `status`

  `INTEGER`. The array with the length of `3`. Output. The error code is returned.

  **First component (`status(1)`)** If the solution is converged or a breakdown occurs, the current total number of iteration with the minus sign is returned. In other cases, this routine returns the current total number of iteration. The calculation is continuable only when `status(1)` is the positive value; otherwise the result is meaningless even if the calculation is continued.

  **Second component (`status(2)`)** `1` is returned if `itermax` is set as a finite value in the `?_init` routine and the convergence condition is not satisfied at the `itermax`-th iteration. `2` is returned if $\alpha$ diverges. `3` is returned if $\pi_{\mathrm{seed}}$ becomes 0. In the case of `COCG_restart` or `BiCG_restart`, `4` is returned if the residual vector and the shadow residual vector are orthogonal. In other cases, `0` is returned.

  **Third component (`status(3)`)** The index of the seed point is returned.

### 5.1.4 `komega_????_getcoef`, `pkomega_????_getcoef`

Get the coefficients used in the restart calculation. To call these routines, `itermax` in `?_init` routine must not be `0` .

Syntax

Fortran (Serial/OpenMP)

```
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/hybrid parallel)

```
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++ (Serial/OpenMP)

```
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/hybrid parallel)

```
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);
```

Parameters

- `alpha_save`

DOUBLE PRECISION (for CG_R_getoef, CG_C_getoef), DOUBLE COMPLEX (for other cases). The array with the length of the number of maximum iteration. Output. The parameters $\alpha$ of the (Bi)CG method at each iteration.

- beta_save

  DOUBLE PRECISION (for CG_R_getoef, CG_C_getoef), DOUBLE COMPLEX (for other cases). The array with the length of the number of maximum iteration. Output. The parameters $\beta$ of the (Bi)CG method at each iteration.

- z_seed

  DOUBLE PRECISION (for CG_R_getoef, CG_C_getoef), DOUBLE COMPLEX (for other cases). Scalar. Output. Seed shift.

- r_l_save

  DOUBLE PRECISION (for CG_R_getoef), DOUBLE COMPLEX (for other cases). The array with the length of the number of maximum iteration $\times$ nl. Output. The projected residual vectors at each iteration.

### 5.1.5 komega_????_getvec, pkomega_????_getvec

Get the residual vectors to use the restart calculation. To call these routines, itermax in the ?_init routine must not be 0.

Syntax

Fortran (Serial/OpenMP)

```
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/hybrid parallel)

```
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++ (Serial/OpenMP)

```
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/hybrid parallel)

```
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);
```

Parameters

- r_old

  DOUBLE PRECISION (for CG_R_getvec), DOUBLE COMPLEX (for other cases). The array with the length of ndim. Output. The residual vector at the second last step in the previous calculation.

- `r_tilde_old`

  Only used for `BiCG_getvec`. `DOUBLE COMPLEX`. The array with the length of `ndim`. Output. The shadow residual vector at the second last step in the previous calculation.

### 5.1.6 `komega_????_getresidual`, `pkomega_????_getresidual`

Get the values of 2-norm of the residual vector at each shift points. These routines can be called from anywhere between `komega_????_init` and `komega_????_finalize`. These routines do not affect the calculation results.

Syntax

Fortran (Serial/OpenMP)

```
CALL komega_CG_R_getresidual(res)
CALL komega_CG_C_getresidual(res)
CALL komega_COCG_getresidual(res)
CALL komega_BiCG_getresidual(res)
```

Fortran (MPI/hybrid parallel)

```
CALL pkomega_CG_R_getresidual(res)
CALL pkomega_CG_C_getresidual(res)
CALL pkomega_COCG_getresidual(res)
CALL pkomega_BiCG_getresidual(res)
```

C/C++ (Serial/OpenMP)

```
komega_CG_R_getresidual(res);
komega_CG_C_getresidual(res);
komega_COCG_getresidual(res);
komega_BiCG_getresidual(res);
```

C/C++ (MPI/hybrid parallel)

```
pkomega_CG_R_getresidual(res);
pkomega_CG_C_getresidual(res);
pkomega_COCG_getresidual(res);
pkomega_BiCG_getresidual(res);
```

Parameters

- `res`

  `DOUBLE PRECISION`. The array with the length of `nz`. Output. The values of 2-norm of the residual vector at each shift points are returned.

### 5.1.7 `komega_????_finalize`, `pkomega_????_finalize`

Release memories of the arrays stored in the library.

Syntax

Fortran (Serial/OpenMP)

```
CALL komega_CG_R_finalize()
CALL komega_CG_C_finalize()
CALL komega_COCG_finalize()
CALL komega_BiCG_finalize()
```

Fortran (MPI/hybrid parallel)

```
CALL pkomega_CG_R_finalize()
CALL pkomega_CG_C_finalize()
CALL pkomega_COCG_finalize()
CALL pkomega_BiCG_finalize()
```

C/C++ (Serial/OpenMP)

```
komega_CG_R_finalize();
komega_CG_C_finalize();
komega_COCG_finalize();
komega_BiCG_finalize();
```

C/C++ (MPI/hybrid parallel)

```
pkomega_CG_R_finalize();
pkomega_CG_C_finalize();
pkomega_COCG_finalize();
pkomega_BiCG_finalize();
```

## 5.2 Sample codes for using shifted BiCG library

As a typical example, the usage of shifted BiCG library is shown below.

```
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 :: &
  & rnd_seed, &
  & 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 :: &
& itermin, & ! First iteration in this run
& iter,    & ! Counter for Iteration
& status(3)
!
! Input Size of vectors
!
CALL input_size(ndim,nl,nz)
!
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)
!
! Check: Whether the restart file is exist.
!
CALL input_restart(iter_old, zseed, alpha, beta, r_l_save)
!
WRITE(*,*)
WRITE(*,*) "#####  CG Initialization  #####"
WRITE(*,*)
!
IF(iter_old > 0) THEN
  !
  ! When restarting, counter
  !
  itermin = iter_old + 1
  CALL komega_BiCG_restart(ndim, nl, nz, x, z, max(0,itermax), &
  &                        threshold, &
  &                 status, iter_old, v2, v12, v4, v14, alpha, &
  &                 beta, z_seed, r_l_save)
  !
  ! These vectors were saved in BiCG routine
  !
  DEALLOCATE(alpha, beta, r_l_save)
  !
  IF(status(1) /= 0) GOTO 10
  !
ELSE
   !
   itermin = 1
   !
   ! Generate Right Hand Side Vector
```

```
    !
    v2(1:ndim) = rhs(1:ndim)
    v4(1:ndim) = CONJG(v2(1:ndim))
    !v4(1:ndim) = v2(1:ndim)
    !
    CALL komega_BiCG_init(ndim, nl, nz, x, z, max(0,itermax), &
    &                     threshold)
    !
END IF
!
! BiCG Loop
!
WRITE(*,*)
WRITE(*,*) "#####  CG Iteration  #####"
WRITE(*,*)
!
DO iter = 1, abs(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
iter_old = ABS(status(1))
!
! Get these vectors for restart in the Next run
!
IF(itermax > 0) 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

If you have any comments, questions, bug reports etc. about this library, please contact to the main developer (Mitsuaki Kawamura) by sending the e-mail (the address is shown below).

mkawamura_at_issp.u-tokyo.ac.jp

Please change _at_ into @, when you will send the e-mail.

[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).