$K\omega$ Documentation

Release 0.2

January 27, 2017

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

License

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.

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 *[1]*
- ($\hat{H}$, z) = (real, complex): Shifted Conjugate Orthogonal Conjugate Gradient(COCG) method *[2]*
- ($\hat{H}$, 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 *[2]* is adopted. Hereafter, the number of the left (right) side vector is written as 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, $\boldsymbol{r}^{\mathrm{old}} = \tilde{\boldsymbol{r}}^{\mathrm{old}} = \boldsymbol{0}$

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

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

do iteration

◦ Seed equation

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

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

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

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

◦ 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 names of the routines is defined as follows.

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

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

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

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

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

  Release the allocated vectors in the library.

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

  Get the α, β, $z_{\rm seed}$, $\mathbf{r}^{\rm L}$ conserved at each iteration.

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

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

- `komega_bicg_restart`, `komega_cocg_restart`, `komega_cg_c_restart`, `CG_R_restart`

Note:

- 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 length of each solution vector is not always equal to N_H. In fact, the its length in the previous section is N_L. In this case, the length of the (bi-)conjugate gradient vector $\mathbf{p}_k(k = 1, \cdots N_z)$ also becomes N_L. We have to prepare a code for projecting N_H-dimensional vector onto $N_L$dimensional space.

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

- If the result converges (or a breakdown occurs), `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 `threshold`, this routine assumes the result is converged.
- We can obtain the history of $\alpha, \beta, \mathbf{r}^{\mathrm{L}}$ for restarting calculation. In this case, `itermax` must not be `0`.

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 α, β, 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 $|\pi_k|$. $\rightarrow z_{\rm seed},\ \pi_{\rm seed},\ \pi_{\rm seed}^{\rm old}$

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

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

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

`komega_bicg_update` finish

if(status(1) < 0 (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 $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\{\mathbf{x}_k\}$, $\mathbf{r}^{\rm L}$ $\boldsymbol{v}_2 = \boldsymbol{\varphi}_j$

`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 $\{z_k\}$

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

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

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

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

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

`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

◦ 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*| . $\rightarrow z_{\rm seed},\ \pi_{\rm seed},\ \pi_{\rm seed}^{\rm old}$

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

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

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

`komega_cocg_update` finish

if(status(1) < 0 (This indicates $|\boldsymbol{v}_2| <$ Threshold.)) 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 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*)
- Update results iteratively (**_update*)
- (Optional) Take the information for the restart (**_getcoef*, **_getvec*)
- Finalization (**_finalize*)

The restart calculation can be done by the following procedures.

- Initialization with the information of the previous calculation (**_restart*)
- Update results iteratively (**_update*)
- (Optional) Take the information for the further restart (**_getcoef*, **_getvec*)
- Finalization (**_finalize*)

Warning: Since $K\omega$ is **not** thread safe, these routine must be called from the outside of the OpenMP-parallel region.

For FORTRAN, the modules can be called by

```
USE komega_cg_r ! Conjugate-gradient method for real vectors
USE komega_cg_c ! Conjugate-gradient method for complex vectors
USE komega_cocg ! Conjugate-orthogonal conjugate-gradient mehod
USE komega_bicg ! Biconjugate-gradient method
```

To utilize routines of MPI / Hybrid parallelization version, the modules can be called as folows:

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

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

```
#include komega_cg_r.h
#include komega_cg_c.h
#include komega_cocg.h
#include komega_bicg.h
```

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

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

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

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

The dimension of solution vectors for the linearized equation. `ndim` for the dimension of variables in other routine is equal to this.

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

The dimension of projected solution vectors. `nl` for the dimension of variables in other routine is equal to this.

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

The number of shifted points. `nz` for the dimension of variables in other routine is equal to this.

```
REAL(8),INTENT(OUT) :: x(nl*nz) ! (for "CG_R_init", "CG_C_init")
COMPLEX(8),INTENT(OUT) :: x(nl*nz) ! (for other cases)
```

The solution vector. In this procedure, `0` vector is returned.

```
REAL(8),INTENT(IN) :: z(nz) ! (for "CG_R_init", "CG_C_init")
COMPLEX(8),INTENT(IN) :: z(nz) ! (for other cases)
```

Shifted points.

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

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.

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

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.

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

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

5.1.2 *_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

```
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) ! (for "CG_R_restart", "CG_C_restart")
COMPLEX(8),INTENT(IN) :: z(nz) ! (Other)
INTEGER,INTENT(IN) :: itermax
REAL(8),INTENT(IN) :: threshold
INTEGER,INTENT(IN) :: comm
```

The definition is same as **_init*. See the parameters in **_init*.

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

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 α diverges. `3` is returned if π_{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.

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

The number of iteration for the previous calculation.

```
REAL(8),INTENT(IN) :: v2(ndim) ! (for "CG_R_restart")
COMPLEX(8),INTENT(IN) :: v2(ndim) ! (Other)
```

The residual vector at the last step for the previous calculation.

```
REAL(8),INTENT(IN) :: v12(ndim) ! (for "CG_R_restart")
COMPLEX(8),INTENT(IN) :: v12(ndim) ! (Other)
```

The residual vector at the second from the last step for the previous calculation.

```
REAL(8),INTENT(IN) :: alpha_save(iter_old) ! (for "CG_R_restart", "CG_C_restart")
COMPLEX(8),INTENT(IN) :: alpha_save(iter_old) ! (Other)
```

The parameters α obtained by the previous calculation at each steps by (Bi)CG methods.

```
REAL(8),INTENT(IN) :: beta_save(iter_old) ! (for "CG_R_restart", "CG_C_restart")
COMPLEX(8),INTENT(IN) :: beta_save(iter_old) ! (Other)
```

The parameters β obtained by the previous calculation at each steps by (Bi)CG methods.

```
REAL(8),INTENT(IN) :: z_seed ! (for "CG_R_restart", "CG_C_restart")
COMPLEX(8),INTENT(IN) :: z_seed ! (Other)
```

The value of the seed shift for the previous calculation.

```
REAL(8),INTENT(IN) :: r_l_save(nl,iter_old) ! (for "CG_R_restart")
COMPLEX(8),INTENT(IN) :: r_l_save(nl,iter_old) ! (Other)
```

The projected residual vector at each iteration for the previous calculation.

```
REAL(8),INTENT(IN) :: v4(ndim) ! (for "CG_R_restart")
COMPLEX(8),INTENT(IN) :: v4(ndim) ! (Other)
```

The shadow residual vector at the last step for the previous calculation.

```
REAL(8),INTENT(IN) :: v14(ndim) ! (for "CG_R_restart")
COMPLEX(8),INTENT(IN) :: v14(ndim) ! (Other)
```

The shadow residual vector at the second last step for the previous calculation.

5.1.3 *_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_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/hybrid parallel)

```
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++ (Serial/OpenMPI)

```
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/hybrid parallel)

```
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);
```

Parameters

```
REAL(8),INTENT(INOUT) :: v12(ndim) ! (for "CG_R_update")
COMPLEX(8),INTENT(INOUT) :: v12(ndim) ! (Other)
```

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.

```
REAL(8),INTENT(INOUT) :: v2(ndim) ! (for "CG_R_update")
COMPLEX(8),INTENT(INOUT) :: v2(ndim) ! (Other)
```

The residual vector is input and the updated residual vector is output.

```
REAL(8),INTENT(IN) :: v14(ndim) ! (for "CG_R_update")
COMPLEX(8),INTENT(IN) :: v14(ndim) ! (Other)
```

The product of the shadow residual vector (`v4`) and the matrix is input.

```
REAL(8),INTENT(INOUT) :: v4(ndim) ! (for "CG_R_update")
COMPLEX(8),INTENT(INOUT) :: v4(ndim) ! (Other)
```

The shadow residual vector is input and the updated vector is output.

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

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 α diverges. `3` is returned if π_{seed} becomes 0. In the case of `COCG_update` or

`BiCG_update`, `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 *_getcoef

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

The total number of iteration (`iter_old`) used in this routine is computed by using `status` which is an output of **_update* as follows:

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

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

```
REAL(8),INTENT(OUT) :: alpha_save(iter_old) ! (for "CG_R_restart", "CG_C_restart")
COMPLEX(8),INTENT(OUT) :: alpha_save(iter_old) ! (Other)
```

The parameters α of the (Bi)CG method at each iteration.

```
REAL(8),INTENT(OUT) :: beta_save(iter_old) ! (for "CG_R_restart", "CG_C_restart")
COMPLEX(8),INTENT(OUT) :: beta_save(iter_old) ! (Other)
```

The parameters β of the (Bi)CG method at each iteration.

```
REAL(8),INTENT(OUT) :: z_seed ! (for "CG_R_restart", "CG_C_restart")
COMPLEX(8),INTENT(OUT) :: z_seed ! (Other)
```

Seed shift.

```
REAL(8),INTENT(IN) :: r_l_save(nl,iter_old) ! ("CG_R_restart")
COMPLEX(8),INTENT(IN) :: r_l_save(nl,iter_old) ! (Other)
```

The projected residual vectors at each iteration.

5.1.5 *_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

```
REAL(8),INTENT(OUT) :: r_old(ndim) ! (for "CG_R_getvec")
COMPLEX(8),INTENT(OUT) :: r_old(ndim) ! (Other)
```

The residual vector at the second last step in the previous calculation.

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

The shadow residual vector at the second last step in the previous calculation.

5.1.6 *_getresidual

Get the values of 2-norm of the residual vector at each shift points. These routines can be called from anywhere between **_init* and **_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

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

The values of 2-norm of the residual vector at each shift points are returned.

5.1.7 *_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 :: &
  & 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

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.

CHAPTER 7

Reference

[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).