ISSP Math Library
Shifted Krylov法ライブラリ $K(\omega)$ マニュアル
v 0.1.1

東京大学物性研究所 ソフトウェア高度化推進チーム

平成28年11月11日

目 次

1 概要

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

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

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.

2 著作権

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

いずれの場合も Seed switching 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 $j = 1 \cdots N_R$
　$\boldsymbol{r} = \boldsymbol{\varphi}_j$, $\tilde{\boldsymbol{r}} =$ 任意, $\boldsymbol{r}^{\rm old} = \tilde{\boldsymbol{r}}^{\rm old} = \boldsymbol{0}$
　$p_{ik} = 0(i = 1 \cdots N_L,\ k = 1 \cdots N_z),\ \pi_k = \pi_k^{\rm old} = 1(k = 1 \cdots N_z)$
　$\rho = \infty,\ \alpha = 1,\ z_{\rm seed} = 0$
　do iteration
　　∘ シード方程式
　　$\rho^{\rm old} = \rho,\ \rho = \tilde{\boldsymbol{r}}^* \cdot \boldsymbol{r}$
　　$\beta = \rho / \rho^{\rm old}$
　　$\boldsymbol{q} = (z_{\rm seed}\hat{I} - \hat{H})\boldsymbol{r}$
　　$\alpha^{\rm old} = \alpha,\ \alpha = \frac{\rho}{\tilde{\boldsymbol{r}}^* \cdot \boldsymbol{q} - \beta\rho/\alpha}$
　　∘ シフト方程式
　　do $k = 1 \cdots N_z$
　　　$\pi_k^{\rm new} = [1 + \alpha(z_k - z_{\rm seed})]\pi_k - \frac{\alpha\beta}{\alpha^{\rm old}}(\pi_k^{\rm old} - \pi_k)$
　　　do $i = 1 \cdots N_L$
　　　　$p_{ik} = \frac{1}{\pi_k}\boldsymbol{\varphi}_i^* \cdot \boldsymbol{r} + \frac{\pi_k^{\rm old}\pi_k^{\rm old}}{\pi_k \pi_k}\beta p_{ik}$

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

3.2 Seed switch 付き Shifted COCG 法

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

$G_{ij}(z_k) = 0(i = 1 \cdots N_L,\ j = 1 \cdots N_R,\ k = 1 \cdots N_z)$
do $j = 1 \cdots N_R$
$\boldsymbol{r} = \boldsymbol{\varphi}_j$, $\boldsymbol{r}^{\rm old} = \boldsymbol{0}$
$p_{ik} = 0(i = 1 \cdots N_L,\ k = 1 \cdots N_z),\ \pi_k = \pi_k^{\rm old} = 1(k = 1 \cdots N_z)$
$\rho = \infty,\ \alpha = 1,\ z_{\rm seed} = 0$
do iteration
○ シード方程式
$\rho^{\rm old} = \rho,\ \rho = \boldsymbol{r} \cdot \boldsymbol{r}$
$\beta = \rho/\rho^{\rm old}$
$\boldsymbol{q} = (z_{\rm seed}\hat{I} - \hat{H})\boldsymbol{r}$
$\alpha^{\rm old} = \alpha,\ \alpha = \frac{\rho}{\boldsymbol{r}\cdot\boldsymbol{q} - \beta\rho/\alpha}$
○ シフト方程式
do $k = 1 \cdots N_z$
$\pi_k^{\rm new} = [1 + \alpha(z_k - z_{\rm seed})]\pi_k - \frac{\alpha\beta}{\alpha^{\rm old}}(\pi_k^{\rm old} - \pi_k)$
do $i = 1 \cdots N_L$
$p_{ik} = \frac{1}{\pi_k}\boldsymbol{\varphi}_i^* \cdot \boldsymbol{r} + \frac{\pi_k^{\rm old}\pi_k^{\rm old}}{\pi_k\pi_k}\beta p_{ik}$
$G_{ij}(z_k) = G_{ij}(z_k) + \frac{\pi_k}{\pi_k^{\rm new}} \alpha p_{ik}$
$\pi_k^{\rm old} = \pi_k$, $\pi_k = \pi_k^{\rm new}$
end do i
end do k
$\boldsymbol{q} = \left(1 + \frac{\alpha\beta}{\alpha^{\rm old}}\right)\boldsymbol{r} - \alpha \boldsymbol{q} - \frac{\alpha\beta}{\alpha^{\rm old}}\boldsymbol{r}^{\rm old},\ \boldsymbol{r}^{\rm old} = \boldsymbol{r},\ \boldsymbol{r} = \boldsymbol{q}$
○ Seed switch
$|\pi_k|$ が最も小さい k を探す. $\rightarrow z_{\rm seed}$，$\pi_{\rm seed}$，$\pi_{\rm seed}^{\rm old}$
$\boldsymbol{r} = \boldsymbol{r}/\pi_{\rm seed}$，$\boldsymbol{r}^{\rm old} = \boldsymbol{r}^{\rm old}/\pi_{\rm seed}^{\rm old}$
$\alpha = (\pi_{\rm seed}^{\rm old}/\pi_{\rm seed})\alpha$, $\rho = \rho/(\pi_{\rm seed}^{\rm old}\pi_{\rm seed}^{\rm old})$
$\{\pi_k = \pi_k/\pi_{\rm seed}$，$\pi_k^{\rm old} = \pi_k^{\rm old}/\pi_{\rm seed}^{\rm old}\}$
if($|\boldsymbol{r}|$ <Threshold) exit
end do iteration

end do j

3.3 Seed switch 付き Shifted CG 法

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

$G_{ij}(z_k) = 0 (i = 1 \cdots N_L,\ j = 1 \cdots N_R,\ k = 1 \cdots N_z)$
do $j = 1 \cdots N_R$
  $\boldsymbol{r} = \boldsymbol{\varphi}_j$, $\boldsymbol{r}^{\rm old} = \boldsymbol{0}$
  $p_{ik} = 0 (i = 1 \cdots N_L,\ k = 1 \cdots N_z),\ \pi_k = \pi_k^{\rm old} = 1 (k = 1 \cdots N_z)$
  $\rho = \infty,\ \alpha = 1,\ z_{\rm seed} = 0$
  do iteration
    ○ シード方程式
    $\rho^{\rm old} = \rho,\ \rho = \boldsymbol{r}^* \cdot \boldsymbol{r}$
    $\beta = \rho / \rho^{\rm old}$
    $\boldsymbol{q} = (z_{\rm seed} \hat{I} - \hat{H}) \boldsymbol{r}$
    $\alpha^{\rm old} = \alpha,\ \alpha = \frac{\rho}{\boldsymbol{r}^* \cdot \boldsymbol{q} - \beta \rho / \alpha}$
    ○ シフト方程式
    do $k = 1 \cdots N_z$
      $\pi_k^{\rm new} = [1 + \alpha (z_k - z_{\rm seed})] \pi_k - \frac{\alpha \beta}{\alpha^{\rm old}} (\pi_k^{\rm old} - \pi_k)$
      do $i = 1 \cdots N_L$
        $p_{ik} = \frac{1}{\pi_k} \boldsymbol{\varphi}_i^* \cdot \boldsymbol{r} + \left(\frac{\pi_k^{\rm old}}{\pi_k} \right)^2 \beta p_{ik}$
        $G_{ij}(z_k) = G_{ij}(z_k) + \frac{\pi_k}{\pi_k^{\rm new}} \alpha p_{ik}$
        $\pi_k^{\rm old} = \pi_k$, $\pi_k = \pi_k^{\rm new}$
      end do i
    end do k
    $\boldsymbol{q} = \left(1 + \frac{\alpha \beta}{\alpha^{\rm old}}\right) \boldsymbol{r} - \alpha \boldsymbol{q} - \frac{\alpha \beta}{\alpha^{\rm old}} \boldsymbol{r}^{\rm old},\ \boldsymbol{r}^{\rm old} = \boldsymbol{r},\ \boldsymbol{r} = \boldsymbol{q}$
    ○ Seed switch
    $|\pi_k|$ が最も小さい k を探す. $\rightarrow z_{\rm seed}$, $\pi_{\rm seed}$, $\pi_{\rm seed}^{\rm old}$
    $\boldsymbol{r} = \boldsymbol{r} / \pi_{\rm seed}$, $\boldsymbol{r}^{\rm old} = \boldsymbol{r}^{\rm old} / \pi_{\rm seed}^{\rm old}$
    $\alpha = (\pi_{\rm seed}^{\rm old} / \pi_{\rm seed}) \alpha$, $\rho = \rho / {\pi_{\rm seed}^{\rm old}}^2$
    $\{\pi_k = \pi_k / \pi_{\rm seed}$, $\pi_k^{\rm old} = \pi_k^{\rm old} / \pi_{\rm seed}^{\rm old}\}$
    if($|\boldsymbol{r}|$ <Threshold) exit
  end do iteration
end do j

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.

注意事項.

- `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, これは必ずしも $N_H \times N_z$ である必要はない. 実際前節の場合は $N_L \times N_z$ である. この時 (双) 共役勾配ベクトル $\mathbf{p}_k$ も N_z 個の N_L 次元のベクトルである. N_H 次元の残差ベクトルを N_L 次元へ変換するところはユーザーが行う.

$$\mathbf{r}^{\mathrm{L}} = \hat{P}^\dagger \boldsymbol{r}, \qquad \hat{P} \equiv (\boldsymbol{\varphi}_1, \cdots, \boldsymbol{\varphi}_{N_L}) \tag{2}$$

- 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, 若しくは破たんしたことを表す. したがって status(1) < 0 でループを抜けるようにしておく.

- komega_????_update 内での収束判定には, シード点での残差ベクトルの 2-ノルムが使われる. すなわち, すべてのシフト点での残差ベクトルの 2-ノルムが threshold を下回った時に収束したと見做される.

- 各 iteration での $\alpha, \beta, \mathbf{r}^{\mathrm{L}}$ を保存しておき, あとで利用する場合には最大反復回数 itermax を設定する.

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_finali
  Allocate したライブラリ内部ベクトルを開放する.

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

- komega_BiCG_getvec, komega_COCG_getvec, komega_CG_C_getvec, komega_CG_R_getvec
  $\boldsymbol{r}$, $\boldsymbol{r}^{\mathrm{old}}$, $\tilde{\boldsymbol{r}}$, $\tilde{\boldsymbol{r}}^{\mathrm{old}}$ を取り出す.

- komega_BiCG_restart, komega_COCG_restart, komega_CG_C_restart, komega_CG_R_restart
  保存しておいた α 等を用いて, 新規の z での計算を行う. $\boldsymbol{r}$ 等も有る場合には komega_BiCG_init, komega_COCG_init, komega_CG_C_init, komega_CG_R_init の代わりに用いてリスタートすることもできる.

4.1 Shifted BiCG ライブラリの動作イメージ

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

komega_BiCG_init(N_H, N_L, N_z, x, z, itermax, threshold) start

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

    Copy $\{z_k\}$

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

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

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

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

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

komega_BiCG_init finish

do iteration

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

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

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

○ シード方程式

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

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

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

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

○ シフト方程式

do $k = 1 \cdots N_z$

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

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

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

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

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

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

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

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

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

`komega_BiCG_update` finish

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

end do iteration

`komega_BiCG_finalize` start

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

`komega_BiCG_finalize` finish

4.2 Shifted COCG ライブラリの動作イメージ

Allocate $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\{\mathbf{x}_k\}, \mathbf{r}^{\mathrm{L}}$

$\boldsymbol{v}_2 = \boldsymbol{\varphi}_j$

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

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

Copy $\{z_k\}$

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

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

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

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

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

`COCG_init` finish

do iteration

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

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

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

○ シード方程式

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

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

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

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

○ シフト方程式

do $k = 1 \cdots N_z$

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

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

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

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

end do k

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

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

○ Seed switch

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

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

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

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

`COCG_update` finish

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

end do iteration

`COCG_finalize` start

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

`COCG_finalize` finish

4.3 Shifted CG ライブラリの動作イメージ

COCG と同様.

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 関数)
- アップデート (update 関数)
- 計算情報出力 (getcoef, getvec 関数などを呼び出力)
- 終了関数 (finalize 関数)

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

- 初期設定関数 (restart 関数)
- アップデート (update 関数)

- 計算情報出力 (getcoef, getvec 関数などを呼び出力)
- 終了関数 (finalize 関数)

の手順で実行する. fortran から呼び出すときには

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

のようにモジュールを呼び出す. `"????"`の部分には, `"CG_R"`, `"CG_C"`, `"COCG"`, `"BiCG"`が入る. MPI/Hybrid 並列版のルーチンを利用するときには,

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

のようにする.

5.1 各ルーチンの説明

5.1.1 `komega_????_init`, `pkomega_????_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)
```

パラメーター

- `ndim`

  `INTEGER`. スカラー. 入力. 線形方程式の次元.

- `nl`

  `INTEGER`. スカラー. 入力. 射影された解ベクトルの次元.

- `nz`

  `INTEGER`. スカラー. 入力. シフト点の数.

- `x`

  `DOUBLE PRECISION` (`CG_R_init` の場合), `DOUBLE COMPLEX` (それ以外). 長さ `nl*nz` の配列. 出力. 解ベクトル. `0` ベクトルが返される.

- z

  DOUBLE PRECISION (CG_R_init, CG_C_init の場合), DOUBLE COMPLEX (それ以外). 長さ nz の配列. 入力. シフト点.

- itermax

  INTEGER. スカラー. 入力. 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)

- threshold

  DOUBLE PRECISION. スカラー. 入力. 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.

- comm

  INTEGER. スカラー. 入力. MPI/Hybrid 並列版のみ. MPI のコミニュケーター (MPI_COMM_WORLD など) を入れる.

5.1.2 komega_????_restart, pkomega_????_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)
```

パラメーター

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

  ?_init と同様.

- status

  INTEGER. 長さ 3 の配列. 出力. エラーコードを返す.

  第一成分 (status(1)) 現在の総反復回数が返される. 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 0 が返される. status(1) が正の値の時のみ反復を続行できる. それ以外の場合は反復を進めても有意な結果は得られない.

  第二成分 (status(2)) itermax を有限にして, かつ itermax 回の反復で収束に達しなかった場合には 1 が返される. α が発散した場合には 2 が返される. π_{seed} が 0 にになった場合には 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.

- iter_old

  INTEGER. スカラー. 入力. 先行する計算での反復回数.

- v2

  DOUBLE PRECISION (CG_R_restart の場合), DOUBLE COMPLEX (それ以外). 長さ ndim の配列. 入力. 先行する計算での最後の残差ベクトル.

- v12

  DOUBLE PRECISION (CG_R_restart の場合), DOUBLE COMPLEX (それ以外). 長さ ndim の配列. 入力. 先行する計算での最後から 2 番目の残差ベクトル.

- alpha_save

  DOUBLE PRECISION (CG_R_restart, CG_C_restart の場合), DOUBLE COMPLEX (それ以外). 長さ iter_old の配列. 入力. 先行する計算での各反復での (Bi)CG 法のパラメーター α.

- beta_save

  DOUBLE PRECISION (CG_R_restart, CG_C_restart の場合), DOUBLE COMPLEX (それ以外). 長さ iter_old の配列. 入力. 先行する計算での各反復での (Bi)CG 法のパラメーター β.

- z_seed

  DOUBLE PRECISION (CG_R_restart, CG_C_restart の場合), DOUBLE COMPLEX (それ以外). スカラー. 入力. 先行する計算でのシードシフト.

- r_l_save

  DOUBLE PRECISION (CG_R_restart の場合), DOUBLE COMPLEX (それ以外). 長さ nl*iter_old の配列. 入力. 先行する計算での各反復での射影された残差ベクトル.

- v4

  BiCG_restart の場合のみ使用. DOUBLE COMPLEX. 長さ ndim の配列. 入力. 先行する計算での最後の影の残差ベクトル.

- v14

  BiCG_restart の場合のみ使用. DOUBLE COMPLEX. 長さ ndim の配列. 入力. 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 komega_????_update, pkomega_????_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)
```

パラメーター

- v12

  DOUBLE PRECISION (CG_R_update の場合), DOUBLE COMPLEX (それ以外). 長さ ndim の配列. 入出力. 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).

- v2

  DOUBLE PRECISION (CG_R_update の場合), DOUBLE COMPLEX (それ以外). 長さ ndim の配列. 入出力. 入力は残差ベクトル. 出力は更新された残差ベクトル.

- v14

  BiCG_update の場合のみ使用. DOUBLE COMPLEX. 長さ ndim の配列. 入力. 影の残差ベクトル (v4) と行列の積.

- v4

  BiCG_update の場合のみ使用. DOUBLE COMPLEX. 長さ ndim の配列. 入出力. 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.

- status

  INTEGER. 長さ 3 の配列. 出力. エラーコードを返す.

  第一成分 (status(1)) 現在の総反復回数が返される. 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 0 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. $\pi_{\rm seed}$ が 0 にになった場合には 3 が返される. `COCG_update` もしくは `BiCG_update` で, 残差ベクトルと影の残差ベクトルが直交した場合には 4 が返される. それ以外の場合には 0 が返される.

第三成分 (`status(3)`) シード点の index が返される.

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

後でリスタートを刷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.
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)
```

パラメーター

- `alpha_save`

  `DOUBLE PRECISION` (`CG_R_getoef`, `CG_C_getoef` の場合), `DOUBLE COMPLEX` (それ以外). 総反復回数と同じ長さの配列. 出力. 各反復での (Bi)CG 法のパラメーター α.

- `beta_save`

  `DOUBLE PRECISION` (`CG_R_getoef`, `CG_C_getoef` の場合), `DOUBLE COMPLEX` (それ以外). 総反復回数と同じ長さの配列. 出力. 各反復での (Bi)CG 法のパラメーター β.

- `z_seed`

  `DOUBLE PRECISION` (`CG_R_getoef`, `CG_C_getoef` の場合), `DOUBLE COMPLEX` (それ以外). スカラー. 出力. シードシフト.

- `r_l_save`

  `DOUBLE PRECISION` (`CG_R_getoef` の場合), `DOUBLE COMPLEX` (それ以外). `nl`$\times$ 総反復回数の長さ配列. 出力. 各反復での射影された残差ベクトル.

5.1.5 komega_????_getvec, pkomega_????_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)
```

パラメーター

- r_old

  DOUBLE PRECISION (CG_R_getvec の場合), DOUBLE COMPLEX (それ以外)．長さ ndim の配列．出力．先行する計算での最後から 2 番目の残差ベクトル．

- r_tilde_old

  BiCG_getvec の場合のみ使用．DOUBLE COMPLEX. 長さ ndim の配列．出力．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 komega_????_getresidual, pkomega_????_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 komega_????_init と komega_????_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)
```

パラメーター

- res

  DOUBLE PRECISION. 長さ nz の配列．出力．各シフト点での残差ベクトルの 2-ノルム．

5.1.7 komega_????_finalize, pkomega_????_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()
```

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_
  &                       komega_BiCG_getcoef, komega_BiCG_getvec, komega_Bi
  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))
     !
```

```fortran
       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
```

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.

参考文献

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

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