富士通株式会社

2021年11月

作業報告書

GKVリスタート用データの補間・再領域分割のポスト処理プログラム開発

改版履歴

|  |  |  |
| --- | --- | --- |
| リリース | 版数 | 備考 |
| 2021/11/26 | 1.0 | 初版 |

目次

[1. はじめに 4](#_Toc87893556)

[2. 対象とするシステム 5](#_Toc87893557)

[2.1. 対象環境 5](#_Toc87893558)

[2.2. 対象ソフトウエア 5](#_Toc87893559)

[3. 作業実施内容 6](#_Toc87893560)

[3.1. 概要 6](#_Toc87893561)

[3.2. diag\_chgres\_cntモジュールの追加 7](#_Toc87893562)

[3.3. diag\_interpモジュールの追加 12](#_Toc87893563)

[3.4. diag\_cache\_cntモジュールの追加 16](#_Toc87893564)

[3.5. コンパイル方法 19](#_Toc87893565)

[4. 動作検証結果 21](#_Toc87893566)

[4.1. 検証方法 21](#_Toc87893567)

[4.2. 解像度の変更 22](#_Toc87893568)

[4.3. 領域分割数の変更 23](#_Toc87893569)

[4.4. 結論 23](#_Toc87893570)

# はじめに

本書は、名古屋大学様向け「GKVリスタート用データの補間・再領域分割のポスト処理プログラム開発」作業に関する作業報告書です。

GKVは、6次元格子配列および3次元格子配列の時間発展を計算するMPI並列コードであり、各MPIプロセスごとに7種類の配列データをFortranバイナリ形式またはNetCDF形式のファイルに出力します。

本作業では、GKVが出力したFortranバイナリ形式およびNetCDF形式のファイルの内、リスタート用データ(cnt)について、各次元の解像度および領域分割数を変更できるように、後処理プログラムdiagへの機能追加作業を実施します。

本書では、本作業において実施したdiagのプログラム変更について説明しています。

# 対象とするシステム

# 対象環境

* Linux x86\_64

# 対象ソフトウエア

* diag\_f0.61\_02

# 作業実施内容

# 概要

GKVは、6次元格子配列および3次元格子配列の時間発展を計算するMPI並列コードです。GKVは、各MPIプロセスごとに7種類の配列データ(phi, Al, mom, trn, tri, fxv, cnt)をFortranバイナリ形式またはNetCDF形式のファイルに出力します。

本作業では、GKVが出力したリスタート用データ(cnt)について、各次元の解像度および領域分割数を変更できるようにする機能を後処理プログラムdiagに実装するため、以下の作業を実施するものです。

* GKVのリスタート用データcnt(Fortranバイナリ形式またはNetCDF形式)について、各次元の解像度および領域分割数を変更し、Fortranバイナリ形式でファイル出力を行うサブルーチンchgres\_cnt\_fortranを作成する。
* 同様に、cntについて各次元の解像度および領域分割数を変更し、NetCDF形式でファイル出力を行うサブルーチンchgres\_cnt\_netcdfを作成する。
* 上記のサブルーチン群をFortranモジュールdiag\_chgres\_cntにまとめて実装し、これらの実行に必要な機能群をサブルーチンとして実装する。

- ファイル出力先ディレクトリの削除および作成(renew\_dir)

- 指定された解像度および領域分割数の整合性検証(check\_params)

- 複素数の線形補間(diag\_interp)

解像度変更については、第１、第２および第６次元については、解像度が増えた場合は0埋め、減った場合は切り捨てとし、その他の次元については線形補間を行います。

尚、動作検証作業については、Fortranバイナリ形式およびNetCDF形式での出力と、オリジナルのデータとの連続性を確認することで行います。詳細は4章を参照してください。

# diag\_chgres\_cntモジュールの追加

* + 1. モジュールソースファイル

src/diag\_chgres\_cnt.f90

* + 1. 使用モジュール

・diag\_header

・diag\_rb

・diag\_geom

・diag\_interp(新規作成)

・netcdf

尚、コンパイルおよびプログラム実行に際しては、事前にnetcdfモジュールの有効化が必要です。

netCDFがenvironment-moduleで提供されている環境では、以下の様にmoduleコマンドを実行します(モジュール名・モジュール構成は環境により異なります)。

$ module load netcdf-c

$ module load netcdf-fortran

netCDFがSpackでインストールされている環境では、以下の様にspackコマンドを実行します。

$ spack load netcdf-c

$ spack load netcdf-fortran

いずれの場合も、load後にnc-configコマンドが実行できることを確認してください。

$ nc-config –version

netCDF 4.4.1.1

* + 1. モジュール変数

・integer,parameter :: cntfos (= 900000000)  
 cntのFortranファイル入出力用ファイル機番

* + 1. 実装サブルーチン

① chgres\_cnt\_fortran

■ SYNOPSIS

　SUBROUTINE chgres\_cnt\_fortran( stpn, nnx, ngy, ngz, ngv, ngm, &

　　　　nnpw, nnpz, nnpv, nnpm, nnps, outdir )

■ 引数

・stpn integer, optional, intent(in)

　変換対象のステップ番号。[snum, enum]の範囲で指定。省略時はenum。

・nnx integer, optional, intent(in)

　新しい第１次元の解像度。省略時は変更なし。

・nny integer, optional, intent(in)

　新しい第2次元の解像度。省略時は変更なし。

・nnz integer, optional, intent(in)

　新しい第3次元の解像度。省略時は変更なし。

・nnv integer, optional, intent(in)

　新しい第4次元の解像度。省略時は変更なし。

・nnm integer, optional, intent(in)

　新しい第5次元の解像度。省略時は変更なし。

・nnpw integer, optional, intent(in)

　新しい第2次元方向の分割数。省略時は変更なし。

・nnpz integer, optional, intent(in)

　新しい第3次元方向の分割数。省略時は変更なし。

・nnpv integer, optional, intent(in)

　新しい第4次元方向の分割数。省略時は変更なし。

・nnpm integer, optional, intent(in)

　新しい第5次元方向の分割数。省略時は変更なし。

・nnps integer, optional, intent(in)

　新しい第6次元方向の分割数。省略時は変更なし。

・outdir character(len=\*), optional, intent(in)

　変換後のcntファイル群の出力先ディレクトリ。絶対パスか、カレント

　ディレクトリ(“.”)からの相対パスで指定。省略時は”./chgres\_cnt”。

■ 説明

chgres\_cnt\_fortranは、diagが入力したcntデータについて、引数で指定された解像度・領域分割数に変更した上で、Fortranバイナリ形式ファイルに出力します。処理の対象となるステップ番号は引数stpnで、またファイル出力先は引数outdirで指定したものとなります。

outdirで指定されたディレクトリが既に存在していた場合は、そのディレクトリは一旦削除され、新たにディレクトリが作成されて、そこにファイル出力されます。

出力ファイル名は、"gkvp.ランク番号(6桁).cnt.ステップ番号(3桁)" となります。

cntの解像度変更は、第１、第2および第３次元については、 解像度が増えた場合は0埋め、減った場合は切り捨てとし、その他の次元については線形補間を行います。

② chgres\_cnt\_netcdf

■ SYNOPSIS

　SUBROUTINE chgres\_cnt\_netcdf( stpn, nnx, ngy, ngz, ngv, ngm, &

　　　　nnpw, nnpz, nnpv, nnpm, nnps, outdir )

■ 引数

・stpn integer, optional, intent(in)

　変換対象のステップ番号。[snum, enum]の範囲で指定。省略時はenum。

・nnx integer, optional, intent(in)

　新しい第１次元の解像度。省略時は変更なし。

・nny integer, optional, intent(in)

　新しい第2次元の解像度。省略時は変更なし。

・nnz integer, optional, intent(in)

　新しい第3次元の解像度。省略時は変更なし。

・nnv integer, optional, intent(in)

　新しい第4次元の解像度。省略時は変更なし。

・nnm integer, optional, intent(in)

　新しい第5次元の解像度。省略時は変更なし。

・nnpw integer, optional, intent(in)

　新しい第2次元方向の分割数。省略時は変更なし。

・nnpz integer, optional, intent(in)

　新しい第3次元方向の分割数。省略時は変更なし。

・nnpv integer, optional, intent(in)

　新しい第4次元方向の分割数。省略時は変更なし。

・nnpm integer, optional, intent(in)

　新しい第5次元方向の分割数。省略時は変更なし。

・nnps integer, optional, intent(in)

　新しい第6次元方向の分割数。省略時は変更なし。

・outdir character(len=\*), optional, intent(in)

　変換後のcntファイル群の出力先ディレクトリ。絶対パスか、カレント

　ディレクトリ(“.”)からの相対パスで指定。省略時は”./chgres\_cnt”。

■ 説明

chgres\_cnt\_netcdfは、diagが入力したcntデータについて、引数で指定された解像度に変更した上で、netCDF形式ファイルに出力します。領域分割数指定は、解像度との整合性チェックに使用されます。処理の対象となるステップ番号は引数stpnで、またファイル出力先は引数outdirで指定したものとなります。

outdirで指定されたディレクトリが既に存在していた場合は、そのディレクトリは一旦削除され、新たにディレクトリが作成されて、そこにファイル出力されます。

出力ファイル名は、"gkvp.cnt.ステップ番号(3桁).nc" となります。

cntの解像度変更は、第１、第2および第３次元については、 解像度が増えた場合は0埋め、減った場合は切り捨てとし、その他の次元については線形補間を行います。

③ renew\_dir

■ SYNOPSIS

　SUBROUTINE renew\_dir( dir )

■ 引数

・dir character(len=\*), intent(in)

　再作成するディレクトリのパス名。絶対パスか、カレントディレクトリ(“.”)

　からの相対パスで指定。

■ 説明

renew\_dirは、引数dirで指定されたディレクトリについて、既に存在していた場合はそのディレクトリを一旦削除し、新たにディレクトリを作成します。

このサブルーチンはprivateであり、モジュール外からcallすることはできません。

④ check\_params

■ SYNOPSIS

　SUBROUTINE check\_params(nnx, ngy, ngz, ngv, ngm, &

　　　　nnpw, nnpz, nnpv, nnpm, nnps)

■ 引数

・nnx integer, intent(in)

　新しい第１次元の解像度。

・nny integer, intent(in)

　新しい第2次元の解像度。

・nnz integer, intent(in)

　新しい第3次元の解像度。

・nnv integer, intent(in)

　新しい第4次元の解像度。

・nnm integer, intent(in)

　新しい第5次元の解像度。

・nnpw integer, intent(in)

　新しい第2次元方向の分割数。

・nnpz integer, intent(in)

　新しい第3次元方向の分割数。

・nnpv integer, intent(in)

　新しい第4次元方向の分割数。

・nnpm integer, intent(in)

　新しい第5次元方向の分割数。

・nnps integer, intent(in)

　新しい第6次元方向の分割数。

■ 説明

check\_paramsは、引数で指定された解像度・領域分割数についての整合性チェックを行います。

このサブルーチンはprivateであり、モジュール外からcallすることはできません。

⑤ get\_org\_ivim

■ SYNOPSIS

　SUBROUTINE get\_org\_ivim( v, m, oiv, oim, vflag, mflag )

■ 引数

・v real(kind=DP), intent(in)

　探索する座標値のv成分。

・m real(kind=DP), intent(in)

　探索する座標値のm成分。

・oiv integer, dimension(2), intent(out)

　(v, m)で指定された座標を内包する、元の格子データにおけるv方向の

　インデックス範囲。

・oim integer, dimension(2), intent(out)

　(v, m)で指定された座標を内包する、元の格子データにおけるm方向の

　インデックス範囲。

・vflag integer, optional, intent(out)

　v方向についての結果フラグ。省略可能。

・mflag integer, optional, intent(out)

　m方向についての結果フラグ。省略可能。

■ 説明

get\_org\_ivimは、引数v, mで指定された座標値について、diagで読込み済みのcntデータにおける第４、第５次元方向の格子点座標と比較し、点(v, m)を内包する格子インデックス番号をoiv, oimに格納します。格子インデックス番号は1起点です。

引数vflagおよびmflagが指定されている場合、vおよびm方向について、それぞれ以下の結果フラグが格納されます。

　-1 : 探索する座標値が最小の格子点座標よりも小さい。インデックスには

　　　1および2を格納。

　 0 : 探索する座標値が格子点座標範囲内。

　 1 : 探索する座標値が最大の格子点座標よりも大きい。インデックスには

　　　格子数-1および格子数を格納。

このサブルーチンはprivateであり、モジュール外からcallすることはできません。。

# diag\_interpモジュールの追加

* + 1. モジュールソースファイル

src/diag\_chgres\_util.f90

* + 1. 使用モジュール

・diag\_header

* + 1. 実装クラス

① interp\_5d

■ メンバー変数

・f complex(kind=DP), dimension(:,:,:,:,:), pointer

　補間元のデータ値を格納する5次元複素数配列へのポインタ。

・x real(kind=DP), dimension(:), allocatable

　補間元データの第１次元の座標値を格納する配列。

・y real(kind=DP), dimension(:), allocatable

　補間元データの第2次元の座標値を格納する配列。

・z real(kind=DP), dimension(:), allocatable

　補間元データの第3次元の座標値を格納する配列。

・v real(kind=DP), dimension(:), allocatable

　補間元データの第4次元の座標値を格納する配列。

・m real(kind=DP), dimension(:), allocatable

　補間元データの第5次元の座標値を格納する配列。

・ilox integer

　第1次元のインデックス探索初期値。

・iloy integer

　第2次元のインデックス探索初期値。

・iloz integer

　第3次元のインデックス探索初期値。

・ilov integer

　第4次元のインデックス探索初期値。

・ilom integer

　第5次元のインデックス探索初期値。

・initialized logical

　初期化済みフラグ。

■ プロシージャ

・initialize => initialize\_interp\_5d

　初期化処理。 initialize\_interp\_5dサブルーチンを参照。

・interpolate => interpolate\_interp\_5d

　線形補間処理。 interpolate\_interp\_5dサブルーチンを参照。

・finalize => finalize\_interp\_5d

　終了処理。 finalize\_interp\_5dサブルーチンを参照。

■ 説明

interp\_5dは、5次元の複素数格子データに対する線形補間機能を実装するクラスです。

メンバー変数として、各次元方向の座標値列とデータ値を保持し、指定された座標値におけるデータ値を線形補間によって求めます。

各次元方向の座標値列は、 initializeプロシージャにおいてallocateされた配列に格納しますが、データ値については外部の配列への参照(pointer)のみを保持します。これらは、finalizeプロシージャにおいてdeallocateおよびnullifyされます。

線形補間はinterpolateプロシージャで行います。ここでは、指定された座標値を内包する格子インデックス番号探索の効率化のために、過去の探索で見つかったインデックス番号をクラスのメンバー変数として格納しておき、次回の探索の際の初期値として使用します。

尚、座標値列およびデータ値の配列のインデックスは、全て1起点です。

* + 1. 実装サブルーチン

① initialize\_interp\_5d

■ SYNOPSIS

　SUBROUTINE initialize\_interp\_5d( me\_, nx, ny, nz, nv, nm )

■ 引数

・me\_ class(interp\_5d), intent(inout)

　このサブルーチンをプロシージャとするクラス。

・nx integer, intent(in)

　第1次元の格子数。2以上の値である必要がある。

・ny integer, intent(in)

　第2次元の格子数。2以上の値である必要がある。

・nz integer, intent(in)

　第3次元の格子数。2以上の値である必要がある。

・nv integer, intent(in)

　第4次元の格子数。2以上の値である必要がある。

・nm integer, intent(in)

　第5次元の格子数。2以上の値である必要がある。

■ 説明

このサブルーチンは、interp\_5dクラスのinitializeプロシージャとしてcallされます。引数で指定された各次元方向の格子数で座標値格納配列をallocateし、デフォルトの座標値(インデックス番号-1、ただし第１次元だけはインデックス番号-1-(nx-1)/2)を設定します。また、initializeを.true.に設定します。

② interpolate\_interp\_5d

■ SYNOPSIS

　SUBROUTINE interpolate\_interp\_5d( me\_, x, y, z, v, m, f, istat )

■ 引数

・me\_ class(interp\_5d), intent(inout)

　このサブルーチンをプロシージャとするクラス。

・x real(kind=DP), intent(in)

　線形補間する点の第1次元座標値。

・y real(kind=DP), intent(in)

　線形補間する点の第2次元座標値。

・z real(kind=DP), intent(in)

　線形補間する点の第3次元座標値。

・v real(kind=DP), intent(in)

　線形補間する点の第4次元座標値。

・m real(kind=DP), intent(in)

　線形補間する点の第5次元座標値。

・f complex(kind=DP), intent(out)

　線形補間したデータ値。

・istat integer, optional, intent(out)

　結果フラグ。

■ 説明

このサブルーチンは、interp\_5dクラスのinterpolateプロシージャとしてcallされます。引数で指定された座標値(x, y, z, v, m)において線形補間したデータ値をfに格納します。格子点範囲外の座標値が指定された場合、境界付近の２点を参照して外挿補間を行います。

引数istatが指定されている場合、成功なら1を、失敗(initializeされていない、またはme\_%fにデータ値の配列が設定されていない)なら-1を格納します。

③ finalize\_interp\_5d

■ SYNOPSIS

　SUBROUTINE finalize\_interp\_5d( me\_ )

■ 引数

・me\_ class(interp\_5d), intent(inout)

　このサブルーチンをプロシージャとするクラス。

■ 説明

このサブルーチンは、interp\_5dクラスのfinalize プロシージャとしてcallされます。me\_%fをnullifyし、座標値列格納配列をdeallocateし、インデックス探索初期値を全て1に設定し、finalizeを.false.に設定します。

④ dintrv

■ SYNOPSIS

　pure SUBROUTINE dintrv( xl, x, ilo, ileft, iright, mflag )

■ 引数

・xl real(kind=DP), dimension(:), intent(in)

　座標値列を格納した配列(1起点)。

・x real(kind=DP), intent(in)

　探索する座標値。

・ilo integer, intent(inout)

　座標値列インデックスの探索初期値。探索結果が格納される。

・ileft integer, intent(out)

　xを内包する座標値列インデックスの左側の値が格納される。

・iright integer, intent(out)

　xを内包する座標値列インデックスの右側の値が格納される。

・mflag integer, intent(out)

　探索結果フラグが格納される。

■ 説明

dintrvは、引数xlで指定された座標値列において、引数xで指定された座標を内包する座標値列インデックス番号をileftおよびirightに格納します。

ileftおよびirightに格納されるインデックス番号の意味は、引数mflagに格納された探索結果フラグの値によって以下のようになります。

-1 : x < xl(1)の場合。ileft=1, iright=2を返す。

0 : xl(i) <= x < xl(i+1)の場合。ileft=i, iright=i+1を返す。

　　x==xl(n)の場合はileft=n-1, iright=nを返す。

1 : xl(n) < x の場合。ileft=n-1, iright=nを返す。

# diag\_cache\_cntモジュールの追加

* + 1. モジュールソースファイル

src/diag\_chgres\_util.f90

* + 1. 使用モジュール

・diag\_header

・diag\_rb

* + 1. 実装クラス

① cntbuff

■ メンバー変数

・idx complex(kind=DP)

　データの(v, m)インデックス(複素数型で格納)

・pd complex(kind=DP), dimension(2\*nx+1, global\_ny+1, 2\*global\_nz)

　データを格納する複素数型配列

・age integer

　データを最後に使用してからの世代数

■ プロシージャ

・set => set\_idx\_pd

　rb\_cnt\_ivimisloopでデータブロックを格納する。set\_idx\_pdを参照。

■ 説明

diag\_cache\_cntは、サブルーチンrb\_cnt\_ivimisloopで取得したデータブロックをキャッシュする際の、1データブロックを保持するクラスです。

(v, m)のインデックスは複素数としてidxに格納します。また、最後にこのデータブロックを使用してからの世代数をageに格納し、LRUでのキャッシュ破棄に使用します。

* + 1. モジュールメンバー

•ips integer

　対象sインデックス番号

•stp integer

　対象タイムステップ番号

•num\_blks integer

　キャッシュするデータブロック数

•cnt\_lst type(cntbuff), dimension(:), allocatable

　キャッシュバッファ

* + 1. 実装サブルーチン

① set\_idx\_pd

■ SYNOPSIS

　SUBROUTINE set\_idx\_pd(this, v, m, s, t)

■ 引数

・this class(cntbuff), intent(inout)

　このサブルーチンをプロシージャとするクラス。

・v, m, s, t integer, intent(in)

　データブロックの(v, m, s, t)インデックス番号。

■ 説明

このサブルーチンは、引数(v, m, s, t)で指定されたデータブロックをサブルーチンrb\_cnt\_ivimisloopを使用して読み込み、this%pdに格納します。

また、this%ageを1に設定します。

② initialize\_cache

■ SYNOPSIS

　SUBROUTINE initialize\_cache( is, istp, nb, istatus )

■ 引数

・is integer, intent(in)

　キャッシュするデータのsインデックス番号

・istp integer, intent(in)

　キャッシュするデータのタイムステップ番号

・nb integer, optional, intent(in)

　キャッシュするデータブロック数（省略時はglobal\_nv\*2個）。

・istatus integer, optional, intent(out)

　allocation文のステータス値格納領域。

■ 説明

nbで指定されたサイズのデータキャッシュの初期化を行います。

cntbuff型の構造体をnb個allocateします。

処理の最初にfinalize\_cacheをcallするため、事前のfinalize\_cache呼び出しは必要ありません。

③ get\_blk

■ SYNOPSIS

　SUBROUTINE get\_blk( v, m, p )

■ 引数

・v, m integer, intent(in)

　取得するデータの(v, m)インデックス番号

・p complex(kind=DP), dimension(:,:,:), intent(out)

　取得するデータの格納領域

■ 説明

(v, m)で指定されたサイズのデータブロックを取得し、pにセットします。

指定されたデータブロックがキャッシュ上に存在する場合はそれを使用し、存在しない場合は空いているキャッシュ領域または最後に使用したキャッシュ領域に上書きで、rb\_cnt\_ivimisloopでデータブロックを読み込み、それを使用します。

④ finalize\_cache

■ SYNOPSIS

　SUBROUTINE finalize\_cache

■ 引数

・なし

■ 説明

キャッシュのクリアを行います。

# コンパイル方法

* + 1. Makefileの修正

本作業において、diagの構成ソースファイルが追加されているため、diagをコンパイルするためのMakefileには修正が必要になります。

本作業でコンパイルに用いた修正済みのMakefileを以下に示します。

・Linux x86\_64環境

diag/backup/Makefile\_ubuntu

上記のMakefile中では、以下のソースファイルを追加する変更を行なっています。

・diag/src/diag\_chgres\_util.f90

・diag/src/diag\_chgres\_cnt.f90

FILEIOにdiag\_rb\_netcdfを指定している、いないに関わらずコンパイルオプション（INC）に `pkg-config –cflags netcdf-fortran` および `pkg-config –cflags netcdf` が、リンケージオプション（LIB）に `pkg-config –libs netcdf-fortran` および `pkg-config –libs netcdf` が、それぞれ追加されるように記述しています。

そのため、コンパイルおよびプログラム実行時にnetcdfモジュールの有効化が必要となります。

netCDFがenvironment-moduleで提供されている環境では、以下の様にmoduleコマンドを実行します(モジュール名・モジュール構成は環境により異なります)。

$ module load netcdf-c

$ module load netcdf-fortran

netCDFがSpackでインストールされている環境では、以下の様にspackコマンドを実行します。

$ spack load netcdf-c

$ spack load netcdf-fortran

* + 1. サンプルメインルーチンFortranソースファイル

本作業において実装したサブルーチンを実行するための、diagプログラムのサンプルメインルーチンを記述したFortran90プログラムソースファイルを、以下のパスに格納しています。

・src/diag\_main\_chgres\_fort.f90 Fortranバイナリファイル出力用

・src/diag\_main\_chgres\_netcdf.f90 netCDFファイル出力用

これらのファイルの記述を参考に、diagプログラムのメインルーチンに実装サブルーチンの呼び出しを記述し、コンパイルを行うことで、本機能を利用することができます。

# 動作検証結果

# 検証方法

今回の作業では、gkvp\_f0.57で”input\_vmec/metric\_boozer.bin.dat”を入力データとして第４ステップまで計算した結果に対し、diagに実装した解像度変更・分割数変更機能を適用した結果を評価することで、正当性の検証を行います。

尚、”input\_vmec/metric\_boozer.bin.dat”データでは、各次元のサイズは以下の値です。

nx = 6

global\_ny = 6

global\_nz = 8

global\_nv = 12

global\_nm = 7

また、領域分割数は以下の値で計算を実行しました。

nprocw = 2

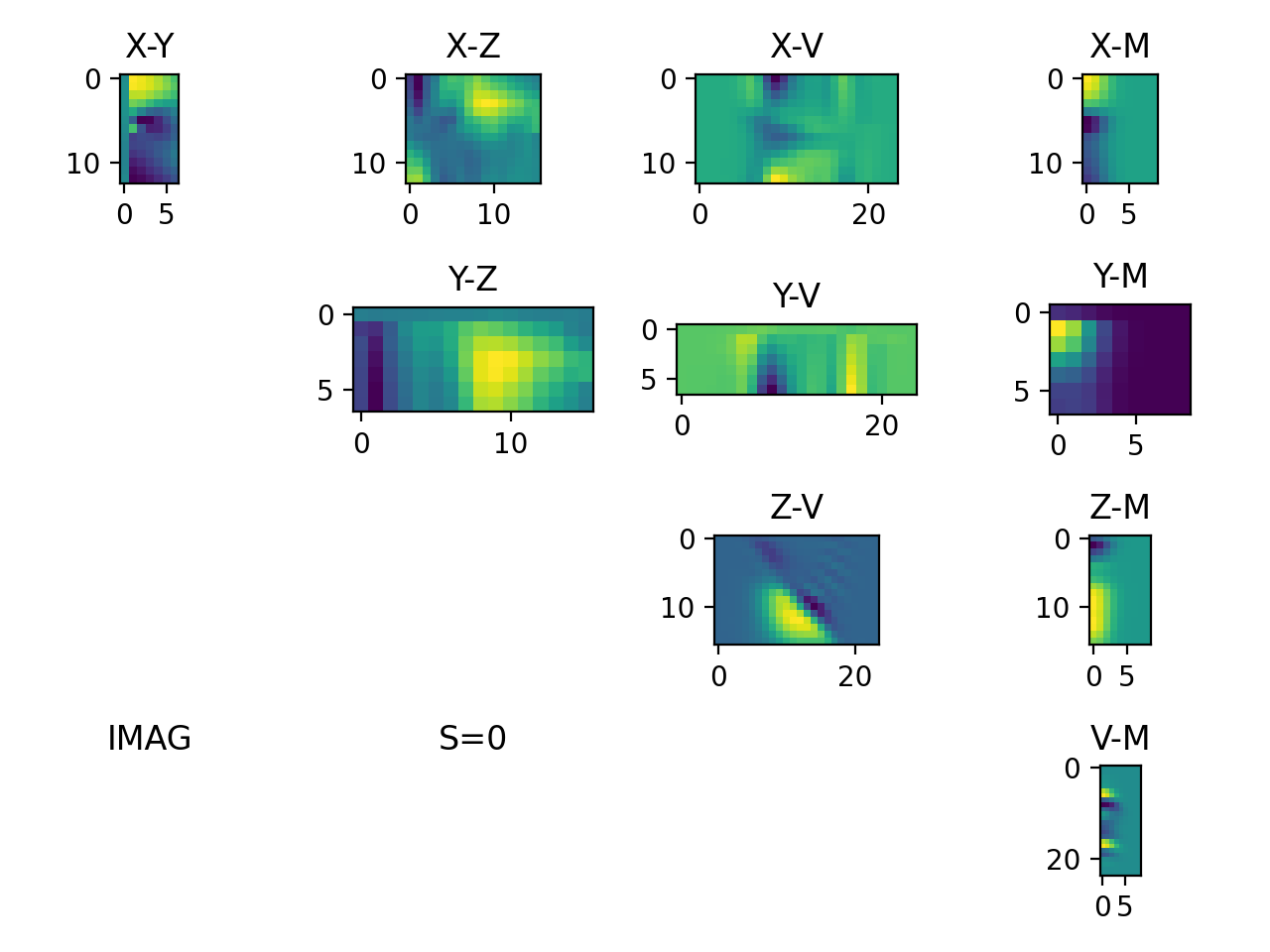
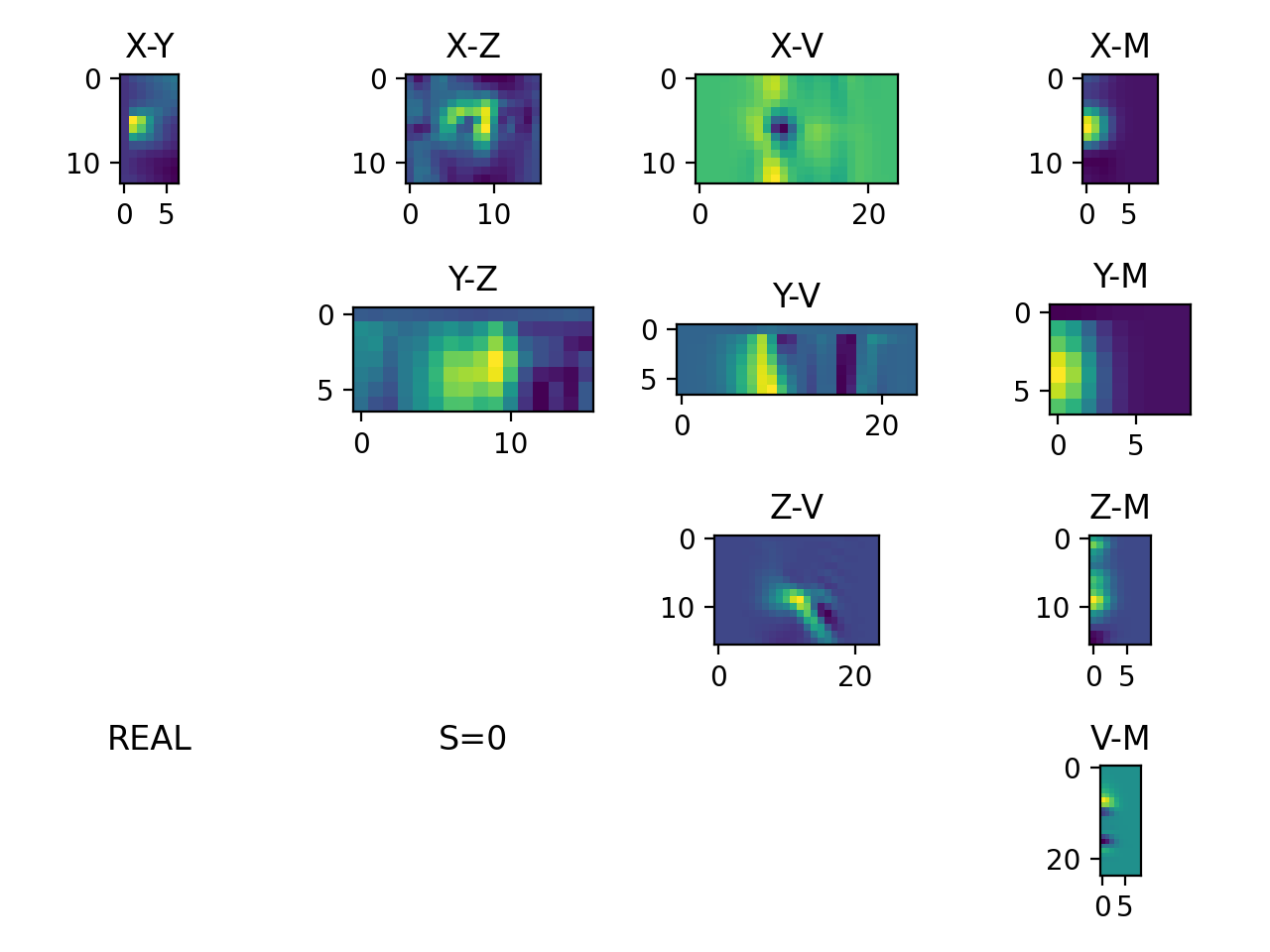
nprocz = 2

nprocv = 2

nprocm = 2

nprocs = 2

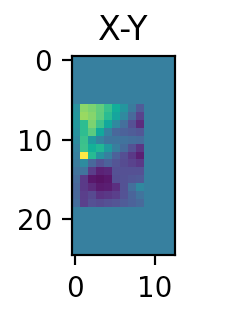
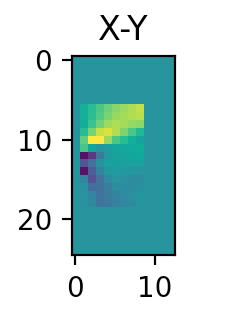
計算結果の最終(第４)ステップにおける、cntの各次元方向の中央位置での直交スライスのプロットを下図に示します。



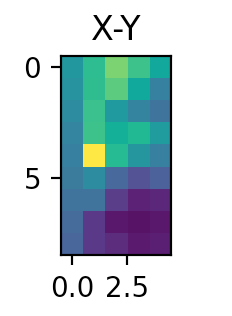
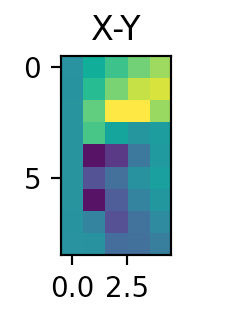
# 解像度の変更

①　X, Y次元の解像度

X次元およびY次元方向の解像度変更では、解像度が増加した場合は０で埋め、減少した場合は切り捨てです。



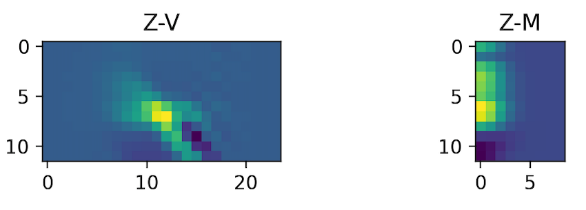
左図は、nxおよびglobal-nyを２倍し、 nx=12, global\_ny=12とした場合のXYスライス面のプロット(縦軸がX、右が実部、左が虚部)です。X次元方向では上下の増加部分に、Y次元方向では右の増加部分に、それぞれ０が埋められています。



左図は、nx=4, global\_ny=4とした場合のXYスライス面のプロット(縦軸がX、右が実部、左が虚部)です。X次元方向では上下に、Y次元方向では右方向に、それぞれ領域がトリミングされています。

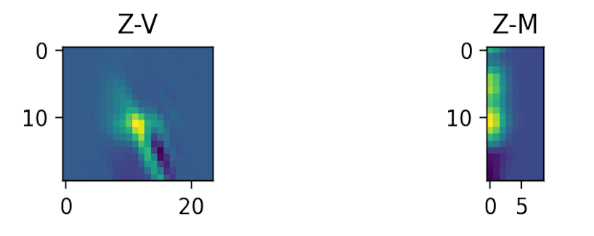
②　Z, V, M次元の解像度

Z次元、V次元、M次元方向の解像度変更では、各次元方向のシミュレーションボックスのサイズは変わらず、解像度の変更によって変化した格子点位置での値を線形補間により求めます(ただし、Z次元方向の上端はLz-δzであるので、解像度変更によって境界の座標も変化します)。



左図は、Z次元方向の解像度を変更した場合のZVおよびZMスライス面のプロット(縦軸がZ)です。上段はglobal\_nz=6の場合、下段はglobal\_nz=10の場合です。

いずれの場合でも、線形補間によって値の変化が滑らかになっています。



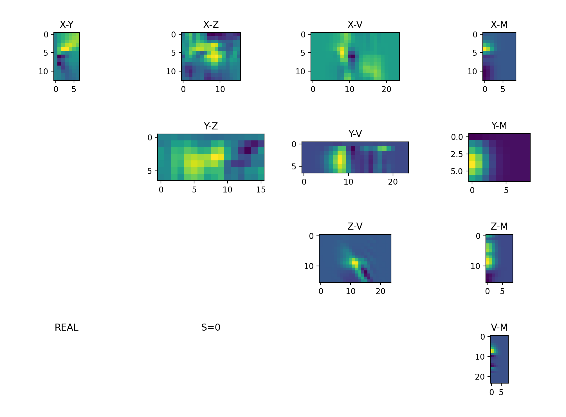
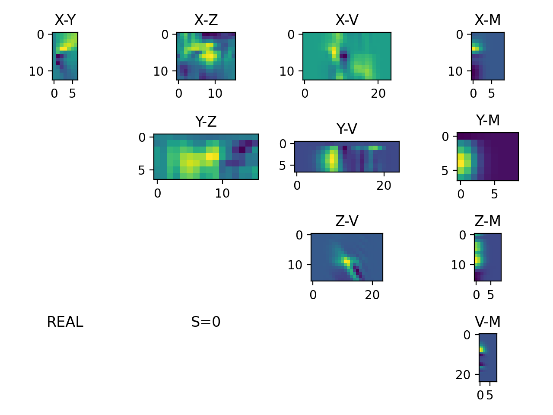
# 領域分割数の変更

領域分割数を変更した場合の検証は、以下の方法で行います。

・FORTRANバイナリ出力した際のファイル数

・出力されたファイルの値の連続性

・分割数を変更した出力ファイルを用いてGKVをリスタート実行

* 検証ケース①　nprocv：2 → 3 に変更  
  ・バイナリ出力ファイル数：32 → 48  
  ・連続性の確認  
    
  　分割数変更前(左)と変更後(右)の直交断面で変化なし。  
  ・GKVのリスタート実行  
  　元の分割数で第4ステップまで実行→分割数変更→第5ステップの計算が  
  　正常に終了した。

# 結論

前章までに示した結果をもって、本作業で実装したGKVリスタート用データの補間・再領域分割機能は正常に動作しているものと判断しました。

以上