Joint-Simulator チュートリアル（2019 VL 講習会資料）

2019/08/26 端野典平 hashino.tempei@kochi-tech.ac.jp

1. はじめに

Joint-Simulator は、Joint Simulator for Satellite Sensors の略で、衛星搭載センサーの観測量を再現することを目的とした、様々な放射モデルの利用を可能とするソフトウェアになります。人工衛星には様々なセンサーが搭載されています。みなさんにとって、最も身近な衛星画像はおそらく、日々の天気予報に登場する「ひまわり８号」の画像ではないでしょうか。AHI というセンサーが搭載されており、可視、近赤外、赤外の波長で、大気の観測をしています。他の例として、日本が開発して世界に貢献している降雨レーダというのがあります。これはマイクロ波長帯になります。熱帯降雨観測衛星 TRMM に搭載されて 17 年間以上も運用されていました。現在は全球降水観測計画 GPM に降水レーダ DPR が搭載され、高緯度の降雪も観測しています。一方、気象予報では、数値モデルによって大気の流れや雲降水の予測が日々、実施されています。もし仮に、この数値モデルで再現される大気を、人工衛星が観測したら、どんな画像ができるだろう。その答えを提供してくれるのが、Joint-Simulator になります。

Joint-Simulator が便利な理由は、一度、入力データと設定ファイルを用意すると、様々な衛星観測が再現できることです。この機会にぜひ、使ってみてはいかがでしょうか。一般には、以下の手順で、疑似衛星画像・データを作ります。

1. 入力データの準備
2. Joint-Simulator のコンパイル
3. Joint-Simulator の設定
4. Joint-Simulator の実行
5. 描画・解析

以下ではこの手順に沿って、NICAM の出力から衛星観測データを作ってみます。

2. 入力データの準備

Joint-Simulator の読み込むファイルは、netcdf データ形式で書かれたものになります。NICAM チュートリアルでは、NICAM の計算結果を緯度経度の単純バイナリデータにして出力しました。これらを、grds2ncd というプログラムで netcdf 形式に変換します。まず、NICAM チュートリアルの１５章で挙げた変数があることを確認しましょう。ここでは、NICAM_outputs というディレクトリを作り、そこにバイナリデータをコピーします。データがディレクトリ A にあるとします（ただし A はフルパスです）。

```
1. $ cd /work/gt32/(アカウント)
2. $ mkdir NICAM_outputs
3. $ cp -p A/*ctl NICAM_outputs/
4. $ cp -p A/*grd NICAM_outputs/
```

最初に Joint-Simulator 関係のソース・ファイルを置くディレクトリを作成し、移動します。

```
5. $ mkdir J-SIM
6. $ cd J-SIM
```

次に grds2ncd のパッケージ、D_grds2ncd.tar.gz をコピーし、解凍します。そして移動します。

```
7. $ cp -p /home/i55077/Source/D_grds2ncd.tar.gz .
8. $ tar zxf D_grds2ncd.tar.gz
9. $ cd D_grds2ncd
```

grds2ncd を作成するには、make を使います。ソース・ファイルがコンパイルされるときに必要なソフトウェアについて、library と include の場所を、Mkinclude 内を編集することで指定します。特に、netcdf と hdf5 を利用しますが、その場所は利用する計算機に依存するので、調べておく必要があります。

Oakforest-PACS では、module を load することで各ソフトウェアの library と include が自動的に設定されますので、Mkinclude 内の変数は空白のままで良いです。次を実行しましょう。

```
10.   $ module load intel
11.   $ module load hdf5_szip
12.   $ module load hdf5
13.   $ module load netcdf
14.   $ module load netcdf-fortran
```

もしくは、ホームディレクトリにある、.bashrc に追加しておくと、ログイン時に自動的に反映されます。

次にコンパイルして実行ファイル grds2ncd が同じディレクトリ内にできていることを確認します。

```
15.   $ make
16.   $ ls
```

これで、grds2ncd ができました。あとは grds2ncd が実行時に読み込む、設定ファイル grds2ncd.cnf を編集する作業に移ります。まずは、netcdf ファイルを作るディレクトリを作成しましょう。ここでは NCD という名前にします。

```
17.   $ cd /work/gt32/(アカウント)
18.   $ mkdir NCD
19.   $ cd NCD
```

そして、RUN_grds2ncd.sh というシェルスクリプトを D_grds2ncd からコピーします。中身をエディターで覗いてみましょう。

```
20.   $ cp /work/gt32/(アカウント)/J-SIM/D_grds2ncd/RUN_grds2ncd.sh .
21.   $ vim RUN_grds2ncd.sh
```

WORKDIR、XDIR、BINDIR、INDIR、INDIR_BND を各自のディレクトリ名に変更しましょう。このスクリプトの中に、grds2ncd.cnf が埋め込まれていて、&grds2ncd_param から/までの範囲が grds2ncd が読み込むの変数のリストになります。vname_grd で、NICAM の grads 用データを指定

して、vname_in_ncd は netcdf 形式にしたときの変数名になります。時間に依存しない変数は、vname_bnd_grd で指定します。今回のデータは、３次元データは５分ごと、２次元データも５分ごとに出力されています。この出力の頻度を時間(hour)の単位にして、frqhour3d と frqhour2d に与えます。３次元データを基準にすると７２時点の出力があるので mxntimes = 72 となり、１０分目のデータから(strtime = 2)、14 個飛ばし(timeinc = 14)でデータを netcdf にしていきます。あとの変数の意味は、prg_grds2ncd.f90 から解読してみてください。

さてそれでは、実際に grds2ncd を実行してみましょう。

```
22.   $ pjsub RUN_grds2ncd.sh
```

ジョブの進行具合は

```
23.   $ pjstat
```

で確認できます。ジョブが終了すると nicam_2018-07-06_18h00m00.nc が作成されます。ファイルの中身を覗いてみましょう。netcdf 形式のファイルには格納される変数名や次元、その他の情報が一緒に入っています。

```
24.   $ ncdump -h nicam_2018-07-06_18h00m00.nc
```

netcdf 形式では、次元の順番が、fortran で定義するものと逆になっていることに注意してください。変数の値をチェックしてみましょう。出力の時間は time に入っています。-v の後ろに time と入力します。

```
25.   $ ncdump -vtime -bf nicam_2018-07-06_18h00m00.nc

data:

 time = 0.1666667, 1.333333, 2.5, 3.666667, 4.833333, 6 ;
```

以上で、Joint-Simulator に入力するデータがすべて一つのファイルに入りました。

最後に、Joint-Simulator が入力するファイルのリストを作成します。ここではリストの名前を inpfile とします。

```
26.   $ ls -1 *.nc > inpfile
```

3. Joint-Simulator をコンパイル

Joint-Simulator をコンパイルに必要なソフトウェアは、

- C-preprocessor
- Fortran compiler
- netcdf もしくは netcdf-4/HDF5
- LAPACK (Linear Algebra PACKage)と BLAS(Basic Linear Algebra Subprograms)

になります。Oakforest-PACS ではすべて必要なものは揃っています。上記２章で module load コマンドのセットを実行していれば、十分です。

まず、Joint-Simulator のパッケージを上記で作成した J-SIM にコピーし、解凍しましょう。

```
1. $ cd /work/gt32/(アカウント)/J-SIM
2. $ cp -p /home/i55077/Source/Joint-Simulator_ver20180614_b.tar.gz .
3. $ tar zxf Joint-Simulator_ver20180614_b.tar.gz
```

```
4. $ cd Joint-Simulator_ver20180614_b
```

中にどういったファイルがあるか調べるため、以下のコマンドを打ちましょう。

```
5. $ ls -latr
```

- QUICK_START_SDSU.txt：手早く Joint-Simulator を実行する手順を説明するテキストファイルです。
- JSIMv1_8_UsersGuide.pdf：詳しいユーザーズマニュアルになります。
- CITATIONS：Joint-Simulator を構成するセンサーシミュレーターの論文などが置いてあります。
- DATAFILES：シミュレーションに必要なデータファイルがあります。
- SRC：Joint-Simulator のソース・ファイルがあります。
- SSLUT：Joint-Simulator が実行時に、高速化のために作り出すテーブルがあります。

さて、それではコンパイルの作業に入りましょう。ソース・ファイルのディレクトリに移動します。

```
6. $ cd SRC
```

Joint-Simulator では、一つの CPU で計算する（シリアル）か、複数の CPU で計算するか、選択できます。SRC ディレクトリ内にある、define.h を編集して、Joint-Simulator の計算モードを指定します。複数の CPU を利用する場合は、処理するファイルを分割するか、計算領域を分割する方法があります。選択の指針としては

- ファイル分割：計算領域が小さくて一つのコアで計算でき、処理するファイルがたくさんある場合に使う。
- 領域分割：計算領域が大きい場合、もしくは 1 ファイルの計算に時間がかかる場合に使う。

今の設定では、# define MPI 2 と記述されていますので、領域分割となります。シリアル計算の場合は、# define MPI 0 に、ファイル分割の場合は、# define MPI 1 とします。

ここでも make を使ってコンパイルするので、ディレクトリ内にある makefile を編集します。grds2ncd のときと同様に、module load のコマンドを実行済みであれば、library や include の場所に関する変数は何も指定せずに、空欄のままで大丈夫です。計算機の環境が変わると、編集する必要があります。

それでは、コンパイルしてみましょう。

```
7. $ make
```

注意点としては、*.F ファイルがもともとのソース・ファイルであるということです。pre-processor により# define MPI というマクロ変数を代入して、*.f90 というファイルを作ります。その後に*.f90 を実際にコンパイルしています。ソース・ファイルの変更は、*.F に行ってください。*.f90 を変更しても上書きされるだけで、コンパイルには反映されません。

さて、SRC ディレクトリ内に、実行ファイル SDSU.x はできましたか？できていない場合はコンパイル時のエラーかリンク時のエラーだと思います。エラーメッセージを上から読んで、調査してみましょう。

4. Joint-Simulator の設定ファイル

本チュートリアルでは、全球降水観測計画 GPM 主衛星に搭載されている二周波降水レーダ DPR の観測量を NICAM の出力から再現します。そのための Joint-Simulator の動作設定をしましょ

う。まず、実行するためのディレクトリを作成し、そこに SRC ディレクトリから、Configure_SDSU.F をコピーしましょう。ここでは JSOUT というディレクトリを作ります。

```
1. $ cd /work/gt32/(アカウント)/
2. $ mkdir JSOUT
3. $ cd JSOUT
4. $ cp /work/gt32/(アカウント)/J-SIM/Joint-
   Simulator_ver20180614_b/SRC/Configure_SDSU.F .
```

Configure_SDSU.F は、NICAM や grds2ncd の設定ファイルと同様、複数の namelist から成ります。以下では実行に必要な主な変数について説明します。変数の説明について、適宜、JSIMv1_8_UsersGuide.pdf を参照してください。それではエディターで Configure_SDSU.F を開きましょう。

```
5. $ vim Configure_SDSU.F
```

- simulator_switch 内で、実行するセンサーシミュレータを選びます。ここで例として radar を選択します。radar = .true.になっていることを確認しましょう。他のセンサーシミュレータも、.true.にすれば一回の実行で出力が得られます。

```
! ########################################################################
! ####################  Configure Simulator Switch  ######################
! ########################################################################

$simulator_switch
  micro  = .false.     ! microwave simulator switch; on when .true. (logical)
  radar  = .true.     ! radar simulator switch; on  when .true.  (logical)
  visir  = .false.     ! visble/IR simulator switch; on when .true. (logical)
  lidar  = .false.     ! Lidar simulator switch; on when .true. (logical)
  isccp  = .false.     ! ISCCP-like simulator switch; on when .true. (logical)
  broad  = .false.     ! Lidar simulator switch; on when .true. (logical)
  GV     = .false.      ! GV simulator switch; on when .true. (logical)
  ease   = .false.     ! Earthcare Active-Sensor (EASE) simulator switch; on when .true. (logical)
$end
```

- io_options 内で、1 次散乱参照テーブル用のディレクトリ(sdsu_dir_sslut)、データファイルのディレクトリ(sdsu_dir_data)を設定します。これらは上記の/work/gt32/(アカウント)/J-SIM/Joint-Simulator_ver20180614_b/ 以下にあります。sdsu_io_name で、上で作成した netcdf ファイル名をリストしたファイル、inpfile を指定します。output_suffix で、出力ファイルの形式を指定します。ここでは netcdf 形式としますが、単純バイナリも可能です。

```
! ########################################################################
! #################  Configure Input-Output Options ######################
! ########################################################################

$io_options
  sdsu_dir_sslut= '/work/gi55/i55077/J-SIM/Joint-Simulator_ver20180614_b/SSLUT/'  ! directory for th
  sdsu_dir_data = '/work/gi55/i55077/J-SIM/Joint-Simulator_ver20180614_b/DATAFILES/'       ! directo
ulator (character)
  sdsu_io_name  = 'inpfile'     ! name of model-input-list file  (character)
  verbose_SDSU  = .false.             ! if true, print out more comments during run. (logical)
  write_surface = .true.            ! if true, write out surface single scattering properties (logi
  write_opt     = .true.            ! if true, write out single scattering properties (logical)
  write_CRM3D   = .true.            ! if true, write out CRM 3D input file in GrADS format (logical
  write_CRM2D   = .true.            ! if true, write out CRM 2D input file in GrADS format (logical
  output_suffix = '.nc'             ! suffix of output name (character)  '.nc' for netcdf, '.bin' f
$end
```

- crm_options 内で、入力ファイルのあるディレクトリ(sdsu_dir_input)と、計算結果を出力するディレクトリ(sds u_dir_output)を設定します。入力するモデルの格子点数を ncdump -h を利用して確認しましょう。経度方向のグリッド数を mxgridx に、緯度方向を mxgridy に指定します。鉛直の層数は mxlyr で指定します。モデルの解像度は gridsize で設定できますが、beamconvolution を利用するときにのみ、使われます。そして計算の対象とする時間のインデックス(sdsu_itime_start, sdsu_ntimes, sdsu_itime_skip)を確認してください。ここでは時間節約のため、入力データのうち、６番目の時間のデータ(07-07 00 UTC)のみ、計算します。

```
! ########################################################################
! ###################  Configure Input Model options ####################
! ########################################################################

$crm_options


  sim_case  = 'GENERAL'  ! Weatether Research & Forecasting Model (character*10)
  sdsu_dir_input = '/work/gi55/i55077/NCD/'   ! input directory
  sdsu_dir_output = '/work/gi55//i55077/JSOUT/'   ! output directory (character*200)
  mxgridx   = 288    ! max grid # in horizontal x direction  (integer)
  mxgridy   =  73     ! max grid # in horizontal y direction (integer)
  mxlyr     =  40    ! max grid # in vertical direction  (integer) Toshi- check this lator
  gridsize  = 3.5e0   ! horizontal grid spacings [km] (real)

  sdsu_itime_start = 6,  ! time index to start
  sdsu_ntimes = 6,       ! max time index to finish
  sdsu_itime_skip = 1,   ! increment of time index
```

- iodiag_options 内で、熱力学変数を診断する方法や、netcdf ファイルにある変数名を指定します。NICAM の出力として、今回は気温、気圧、そして水蒸気混合比から熱力学変数と湿度を決定しますので、iread_thermo_opt = 1 とします。netcdf ファイルの変数名は、grds2ncd を実行する段階で default の名前にしてあるので、ここでは変更の必要はありません。他の気象モデル(WRF 等)を利用する場合には変更が必要で、すでにその例は Configure_SDSU.F に書かれています。

```
! ########################################################################
! ##############  Configure IO diagnoses Options ##########################
! ########################################################################
$iodiag_options
! NOTE this namelist is turned on by setting sim_case == GENERAL
!   take off '!c' at the beginning of a line to make the option effective.
! ---- beginning of NICAM setup ----
  iread_thermo_opt = 1,                !     1 (default) : input T, PRES, QV
                                       !     2 : input T, PRES, RH
                                       !     3 : input T, DEN, QV
                                       !     4 : input T, DEN, RH
                                       !     5 : input PB, T, QVAPOR for WRF
  ! idiag_z
  !     1 (default): read 1D layer heights [m]
  !     2          : read 3D base-state geopotential for WRF [m^2/s^2]

  ! idiag_psfc
  !     1 (default): read 2D surface pressure [Pa]
  !     2          : read 2D sea-level surface pressur
```

- single_scattering_options 内で、参照テーブルの利用やテーブルの作り直しを指定します。radar, micro を実行する場合には lut_micro = .true., visir, lidar を実行する際には lut_visir = .true.と設定します。参照テーブルを利用することで精度は多少落ちますが高速に実行できるようになります。参照テーブルは一度作ると、次回からは作る必要はありません。雲微物理スキームの設定を psdinfo_options で変更したり、１次散乱モデルを変更した場合には、lut_replace = .true.にして作り直します。ここでは lut_replace = .false.としています。この設定でも、SSLUT ディレクトリに参照テーブルが存在しない場合には自動的に参照テーブルを生成します。実行前には SSLUT ディレクトリに*NSW*.dat というファイル名がないことを確認しましょう。

```
! ########################################################################
! ##############  Configure Single-Scattering LUTs Options ###############
! ########################################################################

$single_scatter_options


  lut_micro    = .true.     ! Particle single-scattering LUT options for micro/radar simulator (logical)
                            !.true.  : Use LUTs for microwave opt. Very Fast.
                            !.false. : Full solution of Mie routine. Slow, but accurate.
                            ! (This must be .false. for HUCM_SBM case.)

  lut_visir    = .true.     ! Particle single-scattering LUT option for visir simulator (logical)
                            !.true.  : Use LUTs for microwave opt. Very fast.
                            !.false. : Full solution of Mie routine Slow, but accurate.
                            ! (This must be .false. for HUCM_SBM case.

  lut_replace = .false.     ! Replace existing LUT, if you modify single-scattering routines (logical).
                            !.true.  : Replace single-scattering LUTs.
                            !.false. : Use existing Mie LUTs data.
```

- radar_options 内で、radar の設定を行います。他のセンサーシミュレータの場合は、それぞれの namelist の対応する箇所を変更します。DPR は二周波なので再現する周波数の数は mxfreq_radar = 2、周波数は freq_radar = 13.6, 35.5 と設定します。DPR の衛星観測量としてレーダー反射因子の減衰補正をしていないプロダクトと比較するので、attenuation = .false.にしています。あと反射因子の計算される鉛直グリッドを DPR に合わせて定義します。vint_radar = 1 とすることで有効になり、全総数 mxlyr_radar = 175, グリッド間の距離 dhgt_radar = 125.0, そして一番下の観測点 alt_start_radar = 125.0 と設定しています。vint_radar = 0 の場合は、NICAM の鉛直グリッドでの反射因子が出力されます。

```
! ########################################################################
! ########################  Configure Radar Sensor  ######################
! ########################################################################

$radar_options

  radar_sensor = 'DPR'  !sensor name (Dual-frequency Precipitation Radar)
  attenuation  = .false.      !true - attenuating radar reflectivity dBZ  false-non-attenuating
  ground_radar = .false.    !=.true. for ground-based sensor; =.false. for satellite-based sensor
  mxfreq_radar = 2              !The number of channels
  min_echo = 17.                    !minimal_detactable echo [dBZ]
  view_angle_radar = 0.           !viewing angle [deg]
  k2 = -999.,-999.  !dielectric constant |k^2| defaults (if not known -> -999.)
  freq_radar = 13.6, 35.5    !Channel frequencies [GHz]
  nch_radar = '13.6G  ', '35.5G  '  !lut character that is consistent to freq_radar
  fov_ct_radar =  5.0,5.0       ! Spatial resolution for cross-track FOV
  fov_dt_radar =  5.0,5.0       ! Spatial resolution for down-track FOV
  vint_radar = 1             ! vertical interpolation flag. 0: off, 1: on
  mxlyr_radar = 175,          ! number of grids for the interpolated grid
  dhgt_radar =  125.0,          ! height increment for the interpolated grid [m]
  alt_start_radar = 125.0     ! starting altitude for the interpolated grid [m]
```

- psdinfo_options 内で、エアロゾルモデルと雲微物理モデルのパラメターの指定を行います。この設定が一番大変かもしれません。基本的には使用された雲微物理モデルの論文等を調査して、粒径分布関数の仮定や落下速度、質量と粒径の関係式等、把握する必要があります。ここでは NSW6 という水物質の混合比のみを予測する BULK モデルを用いました。その設定はすでに Configure_SDSU.F に記述されています。液体粒子のカテゴリ ncat_liq_in = 2, 固体粒子のカテゴリ ncat_ice_in = 3 といった具合です。

```
! ########################################################################
! ####################  Configure PSD options ############################
! ########################################################################

$psdinfo_options
! NOTE this namelist is turned on by setting cloud_microphysics == GENERAL
!
! ---- beginning of NSW6 setup ----
  gnl_cmic_name = 'NSW',
  gnl_cmic_type = 'BULK',
  nbin_liq_in = 1,
  ncat_liq_in = 2,
  nbin_ice_in = 1,
  ncat_ice_in = 3,
```

例えば、雪カテゴリの場合は、名前 cname_ice_in(2) = 'snow'で、netcdf ファイルの雪の混合比は vname_m_ice(2) = 'QS'で与えられます。PSDINFO_ICE_IN には、粒径分布の種類(%DISNO)やそのパラメータ(%PSD_PAR)、質量と粒径の関係の係数(%MD_PAR)を指定します。粒径分布は一般ガンマ関数の一種の逆指数関数なので 1 とします。%PSD_PAR のうち、-999.0e0 としているのは、lambda(傾き)を予測された混合比から推定する、という意味です。NSW6 は one-moment scheme なので-999.0e0 は一つだけになります。落下速度と粒径の関係式は vtmpar_ice_in で設定します。注意として、これら係数はすべて CGS 単位で計算されるものです。

```
cname_ice_in(2) = 'snow',
vunit_ice(2) = 'Q',
vname_m_ice(2) = 'QS',
vname_n_ice(2) = 'CS',
dschm_ice_gnl(2) = 0,
PSDINFO_ICE_IN(2)%DISNO    =           1,
PSDINFO_ICE_IN(2)%PSD_PAR =   -999.0e0 ,   0.03e+0  ,    0.0e+0,   1.0e+0       ,
PSDINFO_ICE_IN(2)%MD_PAR  =   0.5235988e-1, 3.0e+0,
vtmpar_ice_in(:,2) = 153.054239,  0.25e+0,
```

エアロゾルモデルの設定も同様に行います。今回はエアロゾルの予測はしていないので、Joint-Simulator でも考慮しません。namelist、$crm_options の account_aerosol = .false. とすることで、エアロゾル粒子は無視されます。

5. Joint-Simulator の実行

いよいよ Joint-Simulator を実行します。スクリプト RUN_SDSU.sh を SRC ディレクトリからコピーしましょう。

```
1. $ cd /work/gt32/(アカウント)/JSOUT
2. $ cp /work/gt32/(アカウント)/J-SIM/Joint-
   Simulator_ver20180614_b/SRC/RUN_SDSU.sh .
```

RUN_SDSU.sh をエディターで開いてください。上でできた実行ファイル SDSU.x のディレクトリを、各自、指定する必要があります。変数は BINDIR になります。編集が終わるとあとは以下のようにジョブを投げるだけで、Joint-Simulator が実行されます。

```
3. $ pjsub RUN_SDSU.sh
```

上記のスクリプトで、#PJM –mpi proc=16 として、１６個の CPU を使います。それぞれの CPU に割り与えられたサブ領域で、レーダー反射因子の計算が行われて、出力のときに一つのファイルにまとめられます。mpi_sdsu*.log はそれぞれの CPU から出力されるログ・ファイルです。pjstat で計算の状態を確かめましょう。無事終了すると４つの netcdf ファイル(*.nc)が作られます。

それでは ncdump -h <ファイル名>を使って、中身を見てみましょう。それぞれの格納された変数について、単位(units)と簡単な説明(long_name)が書いてあります。

- nicam_2018-07-06_18h00m00.OPT.DPR.nc：光学パラメターの値を含む。
- nicam_2018-07-06_18h00m00.DPR.nc：NICAM から計算されたレーダー反射因子を含む。

はじめて計算する場合は、以下のファイルの出力値を確認しましょう。NICAM の出力を Joint-Simulator 内で整頓し、実際に使われた変数を含みますので、これらの変数がおかしい場合は、再現されたシグナル値は間違っています。

- nicam_2018-07-06_18h00m00.CRM3D_NSW.nc：３次元の変数を含む。
- nicam_2018-07-06_18h00m00.CRM2D_NSW.nc：２次元の変数を含む。

たとえば、nicam_2018-07-06_18h00m00.DPR.nc の１３GHz レーダー反射因子を表示するには、

```
4. $ ncdump –vdbz_13 –bf nicam_2018-07-06_18h00m00.DPR.nc
```

とタイプします。-v のあとに変数名をタイプします。

6. 観測との比較

最後に、観測との簡単な比較をしてみましょう。まず、Joint-Simulator で計算されたレーダー反射因子の図を作ってみます。grads を利用する場合は、コントロールファイルを以下からコピーしましょう。

```
1. $ cd /work/gt32/(アカウント)/JSOUT
2. $ cp -p /home/i55077/Source/fig_scripts/sim/*ctl .
3. $ cp -p /home/i55077/Source/fig_scripts/sim/cb*gs .
```

そして、grads を立ち上げます。上記の 4 つの netcdf ファイルに対応するコントロールファイルがあります。open コマンドではなくて、xdfopen コマンドで開きましょう。

ncl を利用する場合は、以下をコピーしましょう。

```
4. $ cd /work/gt32/(アカウント)/JSOUT
5. $ cp -p /home/i55077/Source/fig_scripts/sim/*ncl .
6. $ cp -p /home/i55077/Source/fig_scripts/sim/mkfigs_ncl.sh .
```

そして、以下を実行すると 2km でのレーダー反射因子が描画できます。

```
7. $ ./mkfigs_ncl.sh
```

radar_dbz13_2km.eps というファイルができていれば、成功です。eps を表示するには

```
8. $ evince radar_dbz13_2km.eps
```

です。mkfigs_ncl.sh を開いて、高度を変更してみましょう。高度とともに反射因子の値はどうなるでしょうか？また、westjp_waterpath.ncl の行からコメントを外すと鉛直積算の水の量をプロットすることもできます。どの粒子カテゴリの量が多いでしょうか？

観測のデータをコピーしましょう。

```
9. $ cd /work/gt32/(アカウント)
10.   $ cp -rp /home/i55077/Obs .
11.   $ cd Obs
```

この中には、再現実験の期間に本州を通過した GPM のデータが hdf5 形式であります。以下から grads 用のコントロールファイルと ncl 用のスクリプトをコピーしましょう。

```
12.   $ cp -p /home/i55077/Source/fig_scripts/obs/* .
```

grads のコントロールファイルは open コマンドで開くことができます。ncl を利用するには

```
13.   $ ./mkfigs_ncl.sh
```

と打ちます。radar_obs_track1.eps と radar_obs_swath_dbz13_2km_1.eps が出力されます。エディターで mkfigs_ncl.sh を開いて、反射因子の高度を変えてプロットしてみましょう。NICAM から計算した反射因子と比較してみましょう。

7. おわりに

さて、無事に衛星データを作ることができたでしょうか？今回は GPM/DPR の観測量を計算してみました。スナップショットを比較するだけでも、モデルの特性がある程度はつかめるのではないかと思います。余力のある人は、ひまわり 8 号の疑似観測データを再現してみて、雲の広がりなど、比べてみてはいかがでしょうか。

モデル検証という一連の作業の肝としては、気象・気候モデルで利用しているものや仮定を、Joint-Simulator でも使うこと、です。例えば、可視域の計算では、モデルで利用している地表面アルベドを Joint-Simulator に入力します。雲降水の検証に興味がある場合は、モデルで仮定される粒径分布の情報を設定します。AMSR 等の受動型マイクロ波の場合は、陸上だと土壌水分量が大切になる周波数もあります。

モデル検証のみならず、ある物理量の放射への感度を調べるというツールにもなります。また、気象・気候モデルが完璧だと仮定して、打ち上がっていない衛星の観測量を作り出すこともできます。ある現象がモデルで再現されたときに、リモートセンシングでは、どう観測されるか、調べることもできるはずです。どうぞご活用ください。