
R2D2

リリース 1.2

Hideyuki Hotta

2020 年 06 月 02 日

目次

第 1 章 R2D2 を使い始めるには	2
1.1 ディレクトリ構造の準備	2
1.2 コンパイル	5
1.3 基本的なパラメータ	5
1.4 初期条件	6
1.5 追加条件	6
1.6 スーパーコンピュータでのシェルスクリプト	7
1.7 初期条件データを受け取った場合	8
第 2 章 R2D2 を使うための環境設定	9
2.1 Fortran コードの環境設定	9
2.2 Python コードの環境設定	10
2.3 IDL コードの環境設定	16
第 3 章 典型的計算例	17
3.1 小規模局所光球計算	17
3.2 中規模黒点計算	20
3.3 全対流層計算	23
3.4 深い部分のみの計算	23
第 4 章 方程式	24
4.1 磁気流体力学	24
4.2 輻射輸送	25
第 5 章 コード	26
5.1 コーディングルール	26
5.2 コード構造	26
第 6 章 数値スキーム	27
6.1 MHD スキーム	27
第 7 章 パラメータ	28
第 8 章 座標生成	29

8.1	一様グリッド	29
8.2	非一様グリッド	30
第 9 章	境界条件	32
9.1	上部境界	33
9.2	下部境界	33
第 10 章	人工粘性	34
第 11 章	出力と読み込み	35
11.1	出力	35
11.2	読み込み	36
11.3	バージョン履歴	39
第 12 章	Paraview を用いた 3D データ表示	40
12.1	データ準備	40
12.2	Paraview を用いて 3 次元表示	42
第 13 章	R2D2 python でのキーワードの説明	46
13.1	self.p [dictionary]	46
13.2	self.qs [dictionary]	50
13.3	self.qq [dictionary]	51
13.4	self.qt [dictionary]	51
13.5	self.vc [dictionary]	51
第 14 章	Sphinx 使用の覚書	52
14.1	はじめに	52
14.2	インストール	52
14.3	HTML ファイルの生成	52
14.4	VS code の利用	53
14.5	環境設定	53
14.6	記法	54
第 15 章	ライセンス	57
第 16 章	出版論文	58
第 17 章	索引と検索ページ	59
第 18 章	TODO リスト	60
Python モジュール索引		62
索引		63

このページは太陽のための輻射磁気流体コード R2D2(RSST and Radiation for Deep Dynamics) のマニュアルである。

PDF 版はこちら



第1章

R2D2 を使い始めるには

1.1 ディレクトリ構造の準備

R2D2 は現在公開していないので、R2D2 の zip ファイルを堀田から受け取ったと仮定する。コードをしっかり読めば従う必要はないが、基本的には以下のようなディレクトリ構造で計算することを想定している。

```
project_name/
    └── run/
        ├── d001/
        ├── d002/
        ├── d003/
        ├── ...
        └── ...
    └── py/
    └── idl/
```

py には R2D2_py をクローンしてきたもの、idl には R2D2_idl をクローンしてきたものを配置する。ここは名前が変わっても問題ない。python(py) と idl のどちらかを使えば解析は可能である(両方ダウンロードする必要はない)。

堀田から受け取った R2D2.zip ファイルをそれぞれ run ディレクトリの d001 などと名前を変えて配置することで色々な計算ケースを実行するのが良いだろう。

R2D2 ディレクトリの中は以下のようディレクトリ構造になっている。

```
R2D2/
    ├── F90_deps.py
    ├── Makefile
    ├── README.md
    ├── gen_time.py
    ├── data/
        └── param/
            ├── nd.dac
            └── back.dac
```

(次のページに続く)

(前のページからの続き)

```
|      |          ├─ params.dac
|      |          └─ xyz.dac
|
|      |
|      ├─ qq/
|      |          ├─ qq.dac.e
|      |          ├─ qq.dac.o
|      |          ├─ qq.dac.00000000
|      |          ├─ qq.dac.00000001
|      |          ├─ ...
|      |
|      |
|      ├─ tau/
|      |          ├─ qq.dac.00000000
|      |          ├─ qq.dac.00000001
|      |          ├─ ...
|      |
|      |
|      ├─ remap/
|      |          ├─ remap_info.dac
|      |          ├─ qq/
|      |          |          ├─ qq.dac.00000000
|      |          |          ├─ qq.dac.00000001
|      |          |          ├─ ...
|      |          |
|      |          └─ v1/
|      |              ├─ c.dac
|      |              ├─ vla.dac.00000000
|      |              ├─ vla.dac.00000001
|      |              ├─ vla.dac.00000002
|      |              ├─ ...
|      |
|      |
|      └─ time/
|          ├─ mhd/
|          |          ├─ t.dac.00000000
|          |          ├─ t.dac.00000001
|          |          ├─ ...
|          |
|          └─ tau/
|              ├─ t.dac.00000000
|              ├─ t.dac.00000001
|              ├─ ...
|
|      └─ input_data/
|      └─ make/
|      └─ retired_src/
|      └─ sh/
|      └─ src/
|          ├─ all/
|          └─ include/
```

それぞれのファイル、ディレクトリの簡単な説明は以下である。

- `F90_deps.py` make/Makefile 生成のための python スクリプト。fortran コードの依存性を調べて、make/R2D2deps に出力する。新しいプログラムを作成したときは

```
python F90_deps.py
```

として make/R2D2deps を更新する。

- `Makefile` make をするときに make ディクレクトリに移動する為のファイル。編集する必要はない。編集すべき Makefile は make/Makefile に配置してある。
- `README.md` GitHub に表示する為の説明ファイル。情報が古くなっている可能性があるので、`README.md`を見るよりは、このウェブページの情報を参照されたい。
- `gen_time.py` 他のモデルから計算結果をもらったときに data ディレクトリや時間のファイルを生成する為の python スクリプト
- `copy_caseid.py` 続きの計算を行うためのスクリプト
- `data/` fortran プログラムを実行した後に、データが保存されるディレクトリ。fortran プログラムを実行すると自動的に生成される。
 - `data/param/` 基本的な計算設定パラメタを出力する為のディレクトリ
 - `data/qq/` チェックポイントのための 3 次元データを出力するためのディレクトリ
 - `data/remap/` 解析のためのリマッピングをしたあとのデータを格納するディレクトリ
 - * `data/remap/qq/` 計算データをリマッピングして扱いやすくした三次元データ。単精度で出力。解析には主にこのデータを使う。
 - * `data/remap/vl/` 計算実行中の解析データ格納
 - `data/time/` 出力した時間を記録したファイルを格納するディレクトリ。3 次元データの出力の時間と記録する mhd と光学的厚さ一定の場所の出力の時間を記録する tau のディレクトリがある。
 - * `data/time/mhd/` MHD 量のアウトプットの時間データ
 - * `data/time/tau/` 光学的厚さ一定の面アウトプットの時間データ

1.2 コンパイル

コンパイルは基本的に R2D2 のディクレトリで

```
make
```

とするのみである。使う計算機によって設定が違うので `make/Makfile` を編集する必要がある。このファイルの 1 行目に

```
SERVER=OFP
```

などと書いてある部分がある。計算機に応じてこの部分を書き換える。それぞれ以下のような対応になっている。
すでに使用できない計算機については説明しない。

- XC: CfCA XC50
- OFP: Oakforest-PACS
- FX: 名大 FX100

以下、堀田の個人環境なので、使用は推奨されない。どうしても個人の環境で使いたい時は堀田まで相談されたい。

- LOCAL: Ubuntu の GCC
- LOXAL_ifort: Ubuntu の ifort
- MAC: Mac の GCC

1.3 基本的なパラメータ

主に変更するパラメタは、領域サイズと格子点数であろう。

これらは、`src/all/geometry_def.F90` を編集することで変更できる。

まずは領域サイズ

```
real(KIND(0.d0)), parameter :: xmin = rsun - 23.876d8
real(KIND(0.d0)), parameter :: xmax = rsun + 0.7d8
real(KIND(0.d0)), parameter :: ymin = 0.d0
real(KIND(0.d0)), parameter :: ymax = 6.144d8*16.d0
real(KIND(0.d0)), parameter :: zmin = 0.d0
real(KIND(0.d0)), parameter :: zmax = 6.144d8*16.d0
```

と書いてある箇所で領域サイズを決定している。`*min, *max` はそれぞれ、`*` 方向の領域の最小値、最大値である。
`x` 方向については、太陽中心からの距離で定義してあるので、`rsun` を使うのが推奨される。

次に格子点数

```
integer, parameter, private :: nx0 = 128, ny0 = 64, nz0 = 64
```

```
integer, parameter :: ix0 = 4
integer, parameter :: jx0 = 16
integer, parameter :: kx0 = 16
```

などと書いてある箇所がある。nx0, ny0, nz0 はそれぞれ一つの MPI プロセスでの x, y, z 方向の格子点の数を定義している。一方、ix0, jx0, kx0 はそれぞれ x, y, z 方向の MPI プロセスの数を定義している。全 MPI プロセス数は ix0*jx0*kx0 となり、各方向の全体の格子点数はそれぞれ nx*ix0, ny*jx0, nz*kx0 となる。

1.4 初期条件

初期条件は、src/all/model.F90 で設定している。基本的な光球計算などは、鉛直速度にランダムな値を入れて計算を始めている。

1.5 追加条件

ある程度計算を行った後に、続きの計算として少し設定を変えたい場合の手続きを示す。例えば、磁場なしの熱対流計算を行った後に、磁場を加える場合などに有効である。

run/d001 での計算を:code:run/d002 に移す場合について説明する。run/d001 の下に copy_caseid.py というスクリプトがあるのでそれを実行する(なければ堀田からもらう)

```
python copy_caseid.py
```

実行すると

```
Q1. Input destination caseid for copy, like d001
```

と質問されるので、データを移す先の caseid を d002 などと入力する。

次に

```
Q2. Input time step for copy, like 10 or end
```

と質問されるので、移動したいデータの時間ステップを 10 などと入力する。チェックポイントのデータはデフォルトでは、10 回に一回しか出力していないので注意すること。

また、現在行った計算の最後の時間ステップのデータを移動したい時は 10 などの代わりに end と入力する。するとプログラム・データのコピーが始まる。すでに移動先(今回場合は d002 に data ディレクトリがある場合は、コピーが始まらないので、削除してからコピーすること)。

また、コピーが終わると移動先の `data/cont_log.txt` に元データの情報が記載してある。

データをコピーした後に、磁場などを付け加えたい時は `src/all/model_add.F90` を編集する。

```
do k = 1,nzg
do j = 1,nyg
do i = 1,nxg
    qq(5,i,j,k) = qq(5,i,j,k) + 200.d0
    qq(6,i,j,k) = qq(6,i,j,k)
    qq(7,i,j,k) = qq(7,i,j,k)
enddo
enddo
enddo
```

などと書いてある。この例では鉛直磁場に 200 G が足されている。次に `src/all/io.F90` を編集する。中ほどに

```
! add something
time00 = 0.d0
if(ns == 0 .and. nd == 0) then
!if(ns == 0) then
    !call model_sunspot
    !call model_fe_pff
    !call model_add
endif
```

と書かれている部分があるので、`call model_add` の部分のコメントアウトを外す。`ns` は現在の計算のステップ数(続きの計算では引き継がれない)、`nd` は計算全体のデータアウトプット回数(続きの計算で引き継がれる)。この二つの変数が 0 の時は、計算の一番はじめもしくは、データを引き継いだ時のみなので、その時のみ追加条件が発動する。

1.6 スーパーコンピュータでのシェルスクリプト

いくつかのスーパーコンピュータでジョブを投入するためのシェルスクリプトも `sh` ディレクトリに用意している。使用コア数などを変えたい時は、それぞれのスーパーコンピュータの使用説明書などを参照すること。今後使うことのできるものだけをあげる。

- `fx.sh`: 名大 FX100
- `ofp.sh`: Oakforest-PACS
- `xc.sh`: CfCA XC50

1.7 初期条件データを受け取った場合

熱対流が統計的定常に達するまでは非常に時間がかかるために、この計算が非常に困難になる。そのため、堀田がデータを提供することがある。堀田は data ディレクトリを丸ごと提供する。

このディレクトリに cont_log.txt というファイルがあるので、そこに示されている計算設定の情報を見て同じようになるように計算を設定する。

このディレクトリを書く実行のディレクトリの配下におき、実行すると続きの計算が始まる。

最終更新日：2020 年 05 月 21 日

第2章

R2D2 を使うための環境設定

R2D2 コードを使って計算するだけならば任意の fortran コンパイラ, FFTW, MPI のみがあれば良い。Python コードを使って解析する場合は、いくつかのモジュールが必要なので、そのインストールの方法もここで説明する。

2.1 Fortran コードの環境設定

2.1.1 Mac

Homebrew を用いて、必要なコンパイラ・ライブラリをインストールすることを推奨している。コンパイラと FFTW のインクルードファイルとライブラリの位置だけ指定すれば良いので、任意の方法でインストールして構わない。Homebrew 以外を用いる場合は、便宜 make/Makefile を編集すること。

Homebrew のインストール

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/`master/install`)"
```

gfortran のインストール

```
brew install gcc
```

OpenMPI のインストール

```
brew install openmpi
```

FFTW のインストール

```
brew install fftw
```

2.1.2 Linux (Ubuntu 18.04)

ここでは、Ubuntu 18.04 の場合のみを説明する。

gfortran のインストール

```
sudo apt-get install gfortran
```

OpenMPI のインストール

```
sudo apt-get install openmpi-doc openmpi-bin libopenmpi-dev
```

FFTW のインストール

```
sudo apt-get install libfftw3-dev
```

2.2 Python コードの環境設定

Anaconda をインストールし、以下に示すモジュール群をインストールする。Mac と Linux で共通する部分が多いのでまとめて説明を記す。

Anaconda のウェブサイト から対応するインストーラーをダウンロードする。

- Mac dmg ファイルをダウンロードして、インストール。インストールされる PATH が変わることが多いが、探して PATH を通す。/anaconda/bin や ~/opt/anaconda/bin など
- Linux ダウンロードしてきたシェルスクリプトファイルのあるディレクトリで.. code:

```
bash ~/Anaconda***.sh
```

インストールするディレクトリは /ホームディレクトリ/anaconda3 とする。/ホームディレクトリ/anaconda3 に PATH を通す。スペコンのログインノードなどでもインストール方法は共通である。

2.2.1 ipython の初期設定

以下は必須ではないが、ipython を使う時の初期設定ファイルである。~/.ipython/profile_default/startup/00_first.py というファイルを作りそこに以下のように記す。

```
import sys, os
import matplotlib.pyplot as plt
import scipy as sp
import numpy as np
from matplotlib.pyplot import pcolormesh, plot, clf, close
```

(次のページに続く)

(前のページからの続き)

```

from numpy import sin,cos,tan,arcsin,arccos,arctan,exp,log,log2,log10,mod,sqrt,
    ↪absolute,sinh,cosh,tanh,pi,arange
plt.ion()
from IPython.core.magic import register_line_magic
@register_line_magic
def r(line):
    get_ipython().run_line_magic('run', ' -i ' + line)
del r

```

最後に記した設定によって、

```
r (Python スクリプト名)
```

とするだけで、スクリプトを実行できるようになる。

2.2.2 Google スプレッドシート利用

計算設定などを Google スプレッドシートにまとめておくと便利である。R2D2 では、Python から直接 Google スプレッドシートに送信する方法を提供しているので、利用したい方は検討されたい。

手順については、[こちら](#) を参考にしたが、少し手順が違うのでこのページでも解説する。

まずは関連するモジュールのインストール。

```

pip install gspread
pip install oauth2client

```

プロキシなどの影響で pip が使えない時は以下のようにする

gspread のインストール

```

git clone git@github.com:burnash/gspread.git
cd gspread
ipython setup.py install

```

oauth2client のインストール

```

git clone git@github.com:googleapis/oauth2client.git
cd oauth2client
ipython setup.py install

```

プロジェクト作成

ウェブブラウザで <https://console.developers.google.com/cloud-resource-manager?pli=1> にアクセス。

「プロジェクトを作成」として、プロジェクトを作成

新しいプロジェクト

割り当て内の残りのプロジェクト数は 11 projects 件です。プロジェクトの増加をリクエストするか、プロジェクトを削除してください。[詳細](#)

[MANAGE QUOTAS](#)

プロジェクト名 * R2D2

プロジェクト ID: r2d2-266700。後で変更することはできません。 [編集](#)

場所 * 組織なし [参照](#)

親組織またはフォルダ

[作成](#) [キャンセル](#)

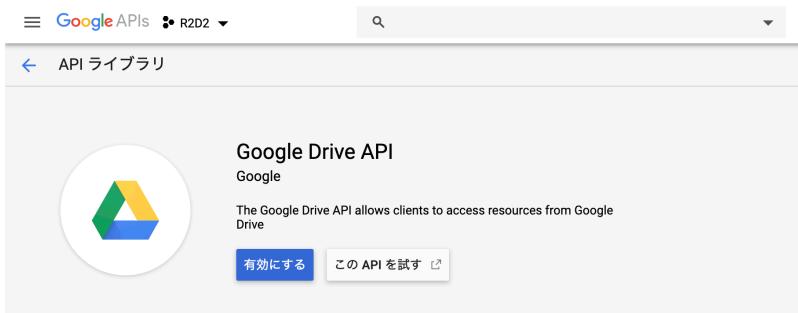
プロジェクト名は R2D2、場所は組織なしとする。

API 有効化

名前	ID	ステータス	課金額	ラベル	操作
組織なし	0				⋮
R2D2	r2d2-264402				⋮

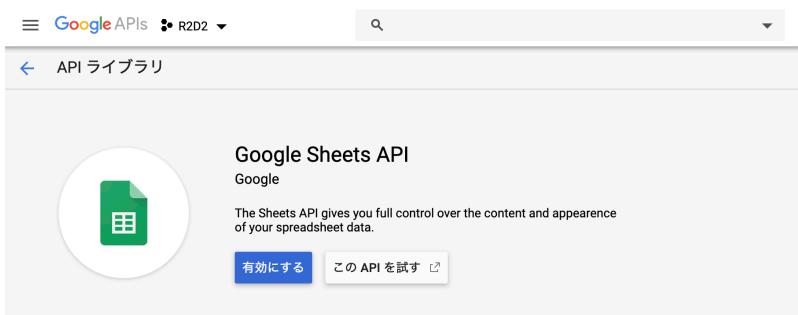
1 個のリソースが削除保留中です

次に検索窓に Google Drive と打ち込んで、Google Drive の API を検索



Google Drive API を有効にする。

同様に Google Sheets と検索



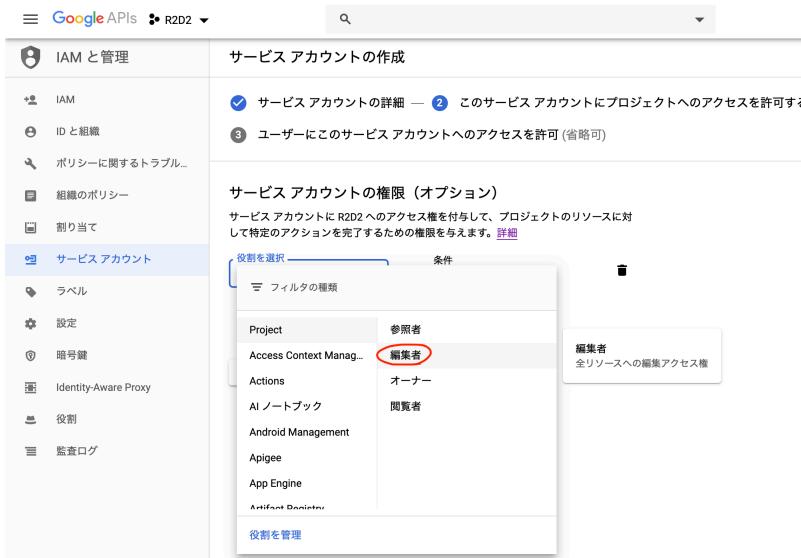
Google Sheets API を有効化

サービスアカウント作成

Google API ロゴ → 認証情報 → サービスアカウントとたどる。



サービスアカウント名は R2D2 とする



役割は編集者を選択

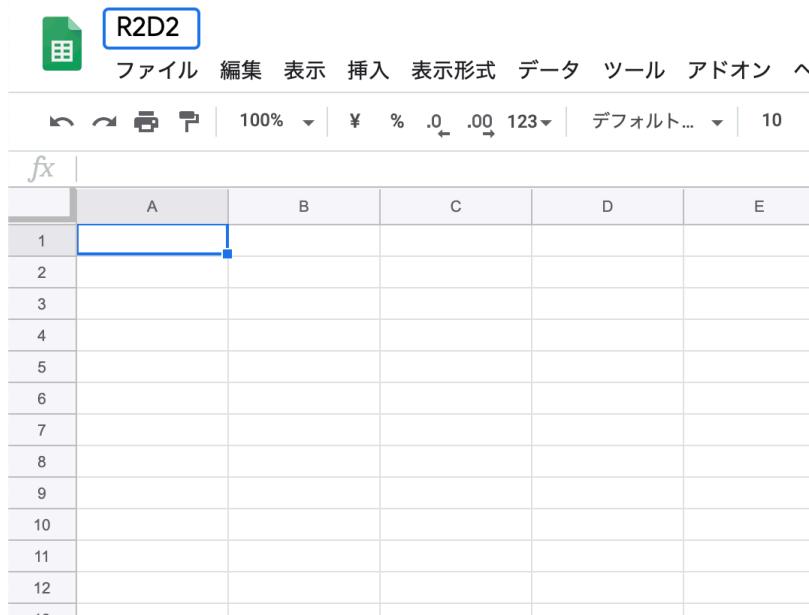


キーの生成では JSON を選択し、キーを生成する。ダウンロードしたファイルは、使用する計算機のホームディレクトリに json というディレクトリを作成し、その下に配置する。そのディレクトリには、この json ファイル以外には何も置かないこと。

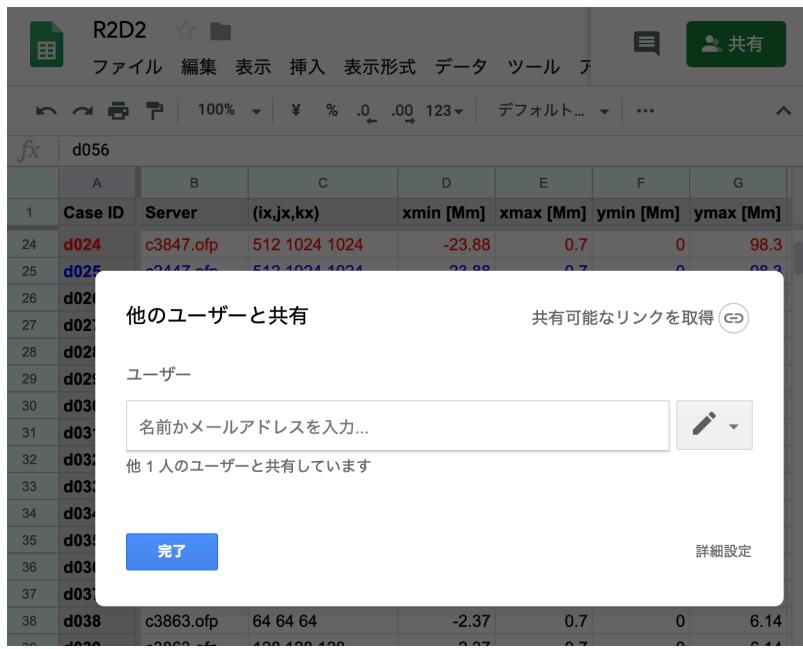
スプレッドシート作成

以下のウェブサイトから Google スプレッドシートを作成 <https://docs.google.com/spreadsheets/create>

名前はプロジェクト名とする。R2D2 では、py ディレクトリの上のディレクトリ名を読みそれをスプレッドシートの名前として情報を送るので、ディレクトリと同じ名前にする。



講習会では R2D2 としておく。



共有をクリックし、ダウンロードした json ファイルの中の client_email 行の E メールアドレスをコピーして、貼り付け。ここまでで、R2D2 から Google スプレッドシートにアクセスできるようになる。

2.3 IDL コードの環境設定

システムに IDL をインストールすれば、それのみで使える。ここでは説明しない。

最終更新日：2020 年 05 月 10 日

第3章

典型的計算例

ここでは、典型的計算設定について紹介する。

3.1 小規模局所光球計算

Vögler et al., 2005 などで行われている小規模局所光球計算の設定を説明する。

- **Makefile のオプション** 光球のみ一様グリッドで計算するので、`PPC := -Ddeep` や `PPC := -Dununiform` を設定していないかチェック。
- **計算領域・解像度** 計算領域は鉛直方向 (R2D2 では x 方向) に `rsun` から 700 km 上から 5.444 Mm 下までをとる。水平方向 (R2D2 では y と z 方向) には、6.144 Mm ずつとる。

太陽光球を計算するときは、鉛直方向には少なくとも 48 km、水平方向には少なくとも 192 km の格子間隔が必要である。ここでは各方向に 48 km の格子点間隔をとることにして、各方向に 128 グリッドづつ格子点を用意する。ここでは MPI プロセスを $2 \times 2 \times 2$ プロセス使う例を示す。これらの設定は `src/all/geometry_def.F90` を編集。

```
#ifdef deep
    ...
    ... (ignore this)
#elif ideal
    ...
    ... (ignore this)
#else
    real(KIND(0.d0)), parameter :: xmax = rsun + 0.7d8
    real(KIND(0.d0)), parameter :: xmin = rsun - 5.444d8
#endif

#ifndef ideal
    real(KIND(0.d0)), parameter :: ymin = 0.d0
    real(KIND(0.d0)), parameter :: ymax = 6.144d0
    real(KIND(0.d0)), parameter :: ymax = 0.d0
    real(KIND(0.d0)), parameter :: ymax = 6.144d0
```

(次のページに続く)

(前のページからの続き)

```
#endif

...
integer, parameter, private :: nx0 = 64, ny0 = 64, nz0 = 64

...
integer, parameter :: ix0 = 2, jx0 = 2, kx0 = 2
```

と設定する。念のために

```
integer, parameter :: xdcheck = 2
integer, parameter :: ydcheck = 2
integer, parameter :: zdcheck = 2

...
integer, parameter :: iper = 0, jper = 1, kper = 1
```

となっているかチェックする。全て xdcheck などは 2 に設定してあると 3 次元計算となる。iper はそれぞれの方向に周期境界条件を使うかのフラグである。

- 音速抑制法 光球のみ計算の時は、音速抑制法を使う必要は無いので、音速抑制率を常に 1 に設定する。
src/all/background_init.F90 を編集。

```
#ifdef ideal
  ... (ignore this)
#else

  ...

do i = 1,ix00
  !xi00(i) = max(1.d0,160.d0*(ro00(i)/rob))**((1.d0/3.d0)*sqrt(dprdro(i)/
  ↪dprdro))
  xi00(i) = 1.d0

  ...
end do

  ...
#endif
```

と設定する。

- 境界条件 水平方向は周期境界を用いるので、鉛直方向の境界条件のみを設定すれば良い。磁場の上部境界条件はポテンシャル磁場の境界条件を用いる。他の設定は Vögler の論文を参照。src/all/bc_all.F90 で設定している。

```
#ifdef ideal
... (ignore this)
#else

#endif
#endif
#endif
```

とする。bcx(qq) が実行されるようにする。また、念のため

```
call bc_potential(qq)
```

がコメントアウトされていないかチェックする。

- 輻射輸送 輻射輸送は複数本の光線を解くのが良い。rte_def.F90 を編集する。推奨される設定は

```
logical, parameter :: rte_multiray_flag = .true.
logical, parameter :: rte_linear_flag = .false.
integer, parameter :: mhd_rte_ratio = 1
```

とするのが良い。

- 初期条件 初期条件は、鉛直方向速度(vx)にランダムな微小速度を与えていた。プラージュ領域を計算したい場合は model_init.F90 で

```
bx = 100.d0
```

とすれば良い。

- 出力アウトプット ここは本当は完全に自由だが、データの出力の設定である。粒状斑の動きを詳しく見ようと思ったら 30 秒ほどの時間ケーデンスで出力するのが良い。main.F90 を編集。

```
dtout = 30.d0
ifac = 1.d0
```

- その他注意事項 model_def.F90 で remap_calc.F90 で出力するスライスの位置を決定している。

```
integer, parameter :: jc = ny*jx0/2
integer, parameter :: kc = nz*kx0/2
```

と領域の真ん中を出力することにしているが、状況によって違う場所が出力されている場合がある。もし変なことが起こったらここをチェックしてみると良い。

また、io.F90 の中程に計算の途中に磁場などを追加する設定がある。ここに何か書いてあると初期

条件に足してしまうので、add somethingで検索して call model_* (*は任意) のところはコメントアウトするように。

3.2 中規模黒点計算

Rempel, 2012 で行われている中規模光球計算の設定を説明する。

- **Makefile のオプション** 光球のみ一様グリッドで計算するので、`PPC := -Ddeep` や `PPC := -Dununiform` を設定していないかチェック。
- **計算領域・解像度** 計算領域は鉛直方向 (R2D2 では x 方向) に `rsun` から 700 km 上から 5.444 Mm 下までをとる。水平方向 (R2D2 では y と z 方向) には、49.152 Mm づつとる。

太陽光球を計算するときは、鉛直方向には少なくとも 48 km、水平方向には少なくとも 192 km の格子間隔が必要である。ここでは鉛直方向に 48 km、水平方向に 96 km の格子点間隔をとることにして、鉛直方向に 128 グリッド、水平方向に 512 グリッドづつ格子点を用意する。ここでは MPI プロセスを $2 \times 4 \times 4$ プロセス使う例を示す。これらの設定は `src/all/geometry_def.F90` を編集。

```
#ifdef deep
    ...
    (ignore this)
#elif ideal
    ...
    (ignore this)
#else
    real(KIND(0.d0)), parameter :: xmax = rsun + 0.7d8
    real(KIND(0.d0)), parameter :: xmin = rsun - 5.444d8
#endif

#ifndef ideal
    real(KIND(0.d0)), parameter :: ymin = 0.d0
    real(KIND(0.d0)), parameter :: ymax = 49.152d8
    real(KIND(0.d0)), parameter :: ymax = 0.d0
    real(KIND(0.d0)), parameter :: ymax = 49.152d8
#endif

...
integer, parameter, private :: nx0 = 64, ny0 = 128, nz0 = 128

...
integer, parameter :: ix0 = 2, jx0 = 4, kx0 = 4
```

と設定する。念のために

```
integer, parameter :: xdcheck = 2
integer, parameter :: ydcheck = 2
integer, parameter :: zdcheck = 2
```

(次のページに続く)

(前のページからの続き)

...

integer, parameter :: iper = 0, jper = 1, kper = 1

となっているかチェックする。全て xdcheck などは 2 に設定してあると 3 次元計算となる。iper はそれぞれの方向に周期境界条件を使うかのフラグである。

- 音速抑制法 光球のみ計算の時は、音速抑制法を使う必要は無いので、音速抑制率を常に 1 に設定する。
src/all/background_init.F90 を編集。

```
#ifdef ideal
  ...
  (ignore this)
#else

  ...

do i = 1,ix00
  !xi00(i) = max(1.d0,160.d0*(ro00(i)/rob))**((1.d0/3.d0)*sqrt(dprdro(i)/
  →dprdrob)
  xi00(i) = 1.d0

  ...
end do

  ...

#endif
```

と設定する。

- 境界条件 水平方向は周期境界を用いるので、鉛直方向の境界条件のみを設定すれば良い。磁場の上部境界条件はポテンシャル磁場の境界条件を用いる。他の設定は Rempel の論文を参照。src/all/bc_all.F90 で設定している。

```
#ifdef ideal
  ...
  (ignore this)
#else

#ifndef deep
  ...
  (ignore this)
#else
  call bcx_sunspot(qq)
  !call bcx(qq)
  !call bcx_whole(qq)
#endif
```

とする。bcx_sunspot(qq) が実行されるようにする。また、念のため

```
call bc_potential(qq)
```

がコメントアウトされていないかチェックする。

ポテンシャル磁場から少しずらしたい時は、mhd_def.F90 の中で Rempel, 2012 で定義されている α が定義されている。

```
real(KIND(0.d0)), parameter :: potential_alpha = 2.5d0
```

などとする。

- 輻射輸送 輻射輸送は複数本の光線を解くのが良い。rte_def.F90 を編集する。推奨される設定は

```
logical, parameter :: rte_multiray_flag = .true.
logical, parameter :: rte_linear_flag = .false.
integer, parameter :: mhd_rte_ratio = 1
```

とするのが良い。

- 初期条件 初期条件は、鉛直方向速度(vx)にランダムな微小速度を与えていた。黒点以外は磁場をゼロにしておく。model_init.F90 で

```
bx = 0.d0
by = 0.d0
bz = 0.d0
```

とすれば良い。

黒点設置のために model_sunspot.F90 を編集。いくつかパラメタがあるが、変えたい時はプログラムのコメントを参照すること。io.F90 を編集し、

```
!add something
time00 = 0.d0
if(ns == 0 .and. nd == 0) then
  call model_sunspot
endif
```

とする。デフォルトでは、call model_sunspot がコメントアウトされていることが多いと思われる。

- 出力アウトプット ここは本当は完全に自由だが、データの出力の設定である。粒状斑の動きを詳しく見ようと思ったら 30 秒ほどの時間ケーデンスで出力するのが良い。main.F90 を編集。

```
dtout = 30.d0
ifac = 1.d0
```

- その他注意事項 model_def.F90 で remap_calc.F90 で出力するスライスの位置を決定している。

と領域の真ん中を出力することにしているが、状況によって違う場所が出力されている場合がある。もし変なことが起こったらここをチェックしてみると良い。

3.3 全対流層計算

課題: 全対流層計算の設定例

3.4 深い部分のみの計算

課題: 深い部分のみの計算の設定例

最終更新日：2020 年 05 月 10 日

第 4 章

方程式

R2D2 で解く方程式は以下である。現状では、デカルト座標 (x, y, z) のみを提供している。数値計算コードの中では、 x を重力方向（鉛直方向）に取っているが、論文を書く際は各自適切に判断されたい。

4.1 磁気流体力学

磁気流体力学の方程式は以下を解いている。

$$\begin{aligned}\frac{\partial \rho_1}{\partial t} &= -\frac{1}{\xi^2} \nabla \cdot (\rho \mathbf{v}) \\ \frac{\partial}{\partial t} (\rho \mathbf{v}) &= -\nabla \cdot (\rho \mathbf{v} \mathbf{v}) - \nabla p_1 - \rho_1 g \mathbf{e}_x + \frac{1}{4\pi} (\nabla \times \mathbf{B}) \times \mathbf{B} \\ \frac{\partial \mathbf{B}}{\partial t} &= \nabla \times (\mathbf{v} \times \mathbf{B}) \\ \rho T \frac{\partial s_1}{\partial t} &= -\rho T (\mathbf{v} \cdot \nabla) s + Q_{\text{rad}} \\ p_1 &= p_1(\rho_1, s_1, x)\end{aligned}$$

ここで ρ は密度、 \mathbf{v} は流体速度、 \mathbf{B} は磁場、 s はエントロピー、 p は圧力、 T は温度、 g は重力加速度、 Q_{rad} は輻射による加熱率である。

R2D2 では熱力学量を以下のように時間的に一定で x 方向の依存性のみを持つ 0 次の量とそこから擾乱の 1 次の量に分けている。

$$\begin{aligned}\rho &= \rho_0 + \rho_1 \\ p &= p_0 + p_1 \\ s &= s_0 + s_1 \\ T &= T_0 + T_1\end{aligned}$$

太陽内部では、 $\rho_1 \ll \rho_0$ などが成り立っているが、太陽表面では熱対流による擾乱と背景場は同程度となるので、R2D2 の中では $\rho_1 \ll \rho_0$ などは仮定しない。0 次の量は Model S を参考にして計算をしている。詳細は出版予定の論文 Hotta & Iijima, in prep (2020?) を参照されたい。

4.2 輻射輸送

課題：輻射輸送の方程式

最終更新日：2020 年 05 月 10 日

第 5 章

コード

5.1 コーディングルール

課題：コーディングルール

5.2 コード構造

課題：コード構造

最終更新日：2020 年 05 月 10 日

第 6 章

数値スキーム

6.1 MHD スキーム

6.1.1 空間微分

R2D2 では、4 次の中央差分を用いている。格子間隔が一様な場合には中央差分では微分は

$$\left(\frac{\partial q}{\partial x} \right)_i = \frac{-q_{i+2} + 8q_{i+1} - 8q_{i-1} + q_{i-2}}{12\Delta x_i}$$

となる。R2D2 では、非一様な格子間隔にも対応しており、

6.1.2 時間積分

R2D2 では、

課題： 数値スキーム (時間積分)

6.1.3 輻射輸送

課題： 数値スキーム (輻射輸送)

最終更新日：2020 年 05 月 10 日

第 7 章

パラメータ

課題：パラメータ。どのグローバル変数がどのモジュールで定義されているかを整理する。

最終更新日：2020 年 05 月 10 日

第 8 章

座標生成

R2D2 では中央差分法を用いているが、そのほとんどは数値フラックスを用いて書き直すことでき、提供される x , y , z などはセル中心で定義される。よって計算領域内の最初のグリッドは、計算境界から半グリッド進んだところにある。

また、R2D2 では一様グリッドと非一様グリッドどちらでも計算できるようにしている。

8.1 一様グリッド

一様グリッドを用いるときは

- 格子間隔を計算する

$$\Delta x = \frac{x_{\max} - x_{\min}}{N_x}$$

ここで、コードでは、配列の要素数には margin も含むので N_x を計算するには margin の部分を引く必要があることに注意。

- x_1 を設定。margin の分も考慮して計算する。
- do loop で順次足していく

コードは以下のようになる

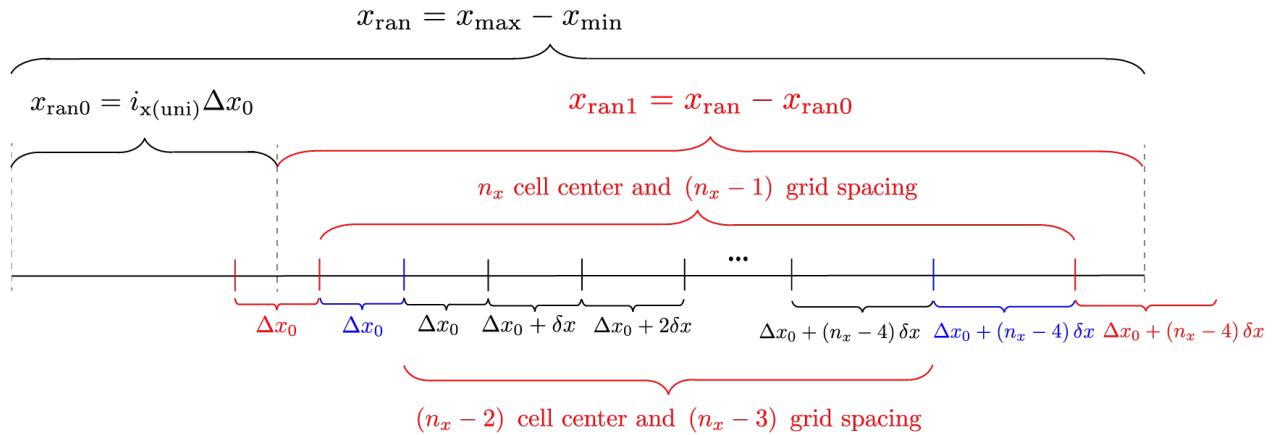
```
dx_unif = (xmax-xmin)/real(ix00-2*marginx)
x00(1) = xmin + (0.5d0-dble(marginx))*dx_unif
if(xdcheck == 2) then
  do i = 1+il,ix00
    x00(i) = x00(i-il) + dx_unif
  enddo
endif
```

8.2 非一様グリッド

非一様グリッドを用いるときは、太陽光球付近は、輻射輸送のために一様なグリッド、ある程度の深さから格子間隔が線形に増加する非一様グリッドを使うことにしている。光球近くは、光球をちゃんと解像するために一様グリッド、光球からある程度進むと、非一様グリッドを採用することにしている。実際の構造は以下のようにになっている。非一様グリッド領域の両端2つのグリッド間隔は一様グリッドをとるようにしている。

fortran のコードの中では

- $\Delta x_0 \rightarrow dx00$: 一様グリッドでの格子点間隔
- $i_{x(\text{uni})} \rightarrow ix_ununi$: 一様グリッドの格子点数
- $x_{\text{ran}} \rightarrow xrange$: 領域サイズ
- $x_{\text{ran}0} \rightarrow xrange0$: 一様グリッドの領域サイズ
- $x_{\text{ran}1} \rightarrow xrange1$: 非一様グリッドの領域サイズ
- $n_x \rightarrow nxx$: 非一様グリッドの格子点数



一様グリッドでのグリッド間隔は Δx_0 として、非一様グリッドでは δx ずつグリッド間隔が大きくなっていくと

する。

$$\begin{aligned}
 x_{\text{tran}} &= \frac{1}{2} \Delta x_0 + \Delta x_0 + \Delta x_0 + (\Delta x_0 + \delta x) + (\Delta x_0 + 2\delta x) + [...] \\
 &\quad + [\Delta x_0 + (n_x - 4) \delta x] + [\Delta x_0 + (n_x - 4) \delta x] + \frac{1}{2} [\Delta x_0 + (n_x - 4) \delta x] \\
 &= \Delta x_0 + \frac{1}{2} (n_x - 4) \delta x + 2\Delta x_0 + (n_x - 4) \delta x + \sum_{n=0}^{n_x-4} (\Delta x_0 + n\delta x) \\
 &= 3\Delta x_0 + \frac{3(n_x - 4) \delta x}{2} + \frac{[2\Delta x_0 + (n_x - 4) \delta x](n_x - 3)}{2} \\
 &= n_x \Delta x_0 + \frac{1}{2} n_x (n_x - 4) \delta x
 \end{aligned}$$

この関係式より、グリッド間隔の増分 δx を以下のように求めることができる。

$$\delta x = \frac{2(x_{\text{tran}} - n_x \Delta x_0)}{(n_x - 4) n_x}$$

最終更新日：2020 年 05 月 10 日

第9章

境界条件

論文を書くときは

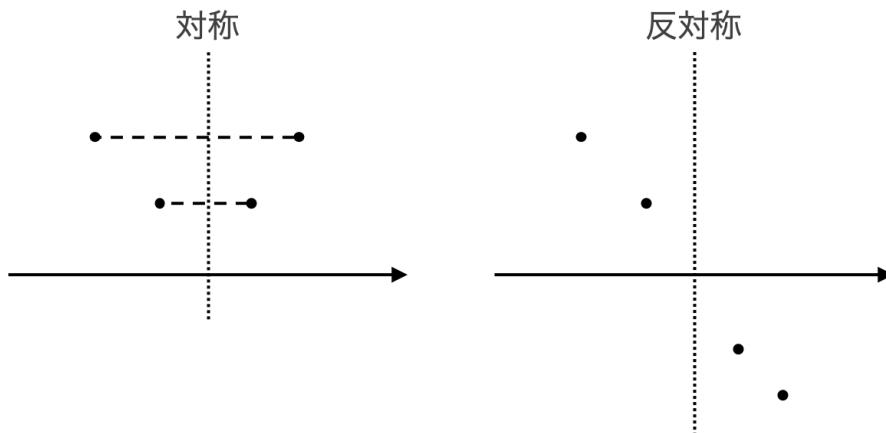
- x, y : 水平方向
- z : 鉛直方向

となっているが、R2D2 のコード内では

- x: 鉛直方向
- y, z: 水平方向

となっている。この取扱説明書では、コードに合わせた表記を用いる。

また、対称・反対称とは以下のような状況を表す。



9.1 上部境界

9.1.1 ポテンシャル磁場

磁場があるときは、上部ではポテンシャル磁場境界条件を使う。

9.2 下部境界

開く時どの質量フラックスも対称にする。 計算領域内の質量を一定に保つために、水平に平均した密度 $\langle \rho_1 \rangle$ は反対称。 そこからのずれ $\rho_1 - \langle \rho_1 \rangle$ は対称な境界条件をとる。

一方、エントロピー s_1 は上昇流で反対称、下降流で反対称な境界条件をとる。この心は、開く境界条件を取るときは計算をしている領域の結果は信用するが、外から入ってくる物理量は、計算領域に寄らないというものである。下降流は現在計算している領域内部での情報を持って計算領域の外に出ていくので、対称な境界条件を用いる。一方、上昇流は、計算している領域の外からの情報を持って計算領域に入ってくるので、反対称な境界条件を用いて擾乱をゼロにする。これは元々の Model S での量を上昇流のエントロピーに用いるということである。

最終更新日：2020 年 05 月 10 日

第 10 章

人工粘性

課題：人工粘性

最終更新日：2020 年 05 月 10 日

第 11 章

出力と読み込み

11.1 出力

11.1.1 Fortran コード

Slice データ

R2D2 では、スライスデータも高いケーデンスで出力できるようになっている。多くのデータを出力するのには効率の悪い方法になっているので、3-4 枚のスライスを出力するのに留めておくことが推奨される。 slice_def.F90 内でどの部分のスライスを出力するかを定義している。

```
integer, parameter :: nx_slice = 3 ! number of slice in x direction
integer, parameter :: ny_slice = 2 ! number of slice in y direction
integer, parameter :: nz_slice = 2 ! number of slice in z direction
```

この部分で、それぞれの方向に何枚のスライスを出力するかを定義。例えば nx_slice はスライスする y-z 平面の数となる。実際にどの部分を出力するかは続く部分で指定している。

```
real(KIND(0.d0)), dimension(nx_slice), save :: x_slice = (/xmin,rsun,xmax/)
real(KIND(0.d0)), dimension(ny_slice), save :: y_slice = (/ymin,0.5d0*(ymin + ymax) /)
real(KIND(0.d0)), dimension(nz_slice), save :: z_slice = (/zmin,0.5d0*(zmin + zmax) /)
```

nx_slice で定義した数と合うように個数を指定しなければならない。

11.2 読込

読み込みについては、Python コードと IDL コードを用意しているが、開発の頻度が高い Python コードの利用を推奨している。

11.2.1 Python コード

Python で R2D2 で定義された関数を使うには

```
import R2D2
```

として、モジュールを読み込む。R2D2 には `R2D2_data` クラスが定義しており、これをオブジェクト指向的に用いてデータを取り扱う。

以下にそれぞれの関数を示すが、docstring は記入してあるので

```
help(R2D2)
help(R2D2.R2D2_data)
```

などとすると実行環境で、モジュール全体や各関数の簡単な説明を見ることができる。

クラス

```
class R2D2.R2D2_data(datadir)
```

データの読み込みには R2D2 モジュールの中で定義されている `R2D2_data` クラスを使う必要がある。

```
import R2D2
datadir = '../run/d002/data'
d = R2D2.R2D2_data(datadir)
```

などとしてインスタンスを生成する。

Attribute

`R2D2_data.p`

基本的なパラメタ。格子点数 `ix` や領域サイズ `xmax` など。インスタンス生成時に同時に読み込まれる。

`R2D2_data.qs`

ある高さの 2 次元の ndarray が含まれる辞書型。`R2D2_data.read_qq_select()` で読み込んだ結果。

`R2D2_data.qq`

3 次元の numpy array。計算領域全体のデータ。`R2D2_data.read_qq()` で読み込んだ結果。

R2D2_data.qt

2 次元の numpy array。ある光学的厚さの面でのデータ。現在は光学的厚さ 1, 0.1, 0.01 でのデータを出力している。`R2D2_data.read_qq_tau()` で読み込んだ結果。

R2D2_data.vc

Fortran の計算の中で解析した結果。`R2D2_data.read_vc()` で読み込んだ結果。

R2D2_data.qc

3 次元の numpy array。計算領域全体のデータ。Fortran の計算でチェックポイントのために出力しているデータを読み込む。主に解像度をあげたいときのために使う `R2D2_data.read_qq_check()` で読み込んだ結果。

R2D2_data.q1

2 次元の numpy array。Fortran で定義したスライスデータ `R2D2_data.read_qq_slice()` で読み込んだ結果。実際にどの位置のスライスを読み込んでいるかは `R2D2.p['x_slice']`, `R2D2.p['y_slice']`, `R2D2.p['z_slice']` を参照すること。スライスの位置の配列が保存されている。

`R2D2_data.p` については、`init.py` などで

```
import R2D2
d = R2D2.R2D2_data(datadir)
for key in R2D2.p:
    exec(' %s = %s' % (key, 'R2D2.p["%s"]'))
```

などとしているために、辞書型の `key` を名前にする変数に値が代入されている。例えば、`R2D2_data.p['ix']` と `ix` には同じ値が入っている。

Method

メソッドで指定する `datadir` はデータの場所を示す変数。R2D2 の計算を実行すると `data` ディレクトリが生成されて、その中にデータが保存される。この場所を指定すれば良い。

R2D2_data.__init__(datadir)

インスタンス生成時に実行されるメソッド。計算設定などのパラメタが読み込まれる。`R2D2_data.p` にデータが保存される。

パラメータ `datadir(str)` -- データの場所

R2D2_data.read_qq_select(xs, n, silent)

ある高さのデータのスライスを読み込む。戻り値を返さない時も `R2D2_data.qs` にデータが保存される。

パラメータ

- `xs(float)` -- 読み込みたいデータの高さ
- `n(int)` -- 読み込みたい時間ステップ

R2D2_data.read_qq(n)

3 次元のデータを読み込む。戻り値を返さない時も *R2D2_data.qq* にデータが保存される。

パラメータ **n** (*int*) -- 読み込みたい時間ステップ

R2D2_data.read_qq_tau(n)

光学的厚さが一定の 2 次元のデータを読み込む。*R2D2_data.qt* にデータが保存される。

パラメータ **n** (*int*) -- 読み込みたい時間ステップ

R2D2_data.read_time(n)

時間を読み込む。

パラメータ **n** (*int*) -- 読み込みたい時間ステップ

戻り値 時間ステップでの時間

R2D2_data.read_vc(n)

Fortran コードの中で解析した計算結果を読み込む。戻り値を返さない時も *R2D2_data.vc* にデータが保存される。

パラメータ **n** (*int*) -- 読み込みたい時間ステップ

R2D2_data.read_qq_check(n)

チェックポイントのためのデータを読み込む *R2D2_data.qc* にデータが保存される。

パラメータ **n** (*int*) -- 読み込みたい時間ステップ

R2D2_data.read_qq_slice(n, n_slice, direc, silent)

slice_def.F90 で指定したスライスデータを読み込む。*R2D2_data ql* にデータが保存される。

パラメータ

- **n** (*int*) -- 読み込みたい時間ステップ
- **n_slice** (*int*) -- 何枚目のスライスを読み込むか
- **direc** (*str*) -- スライスの方向 'x', 'y', 'z' のどれか

11.2.2 IDL コード

GitHub の公開レポジトリ に簡単な説明あり

11.3 バージョン履歴

- ver. 1.0: バージョン制を導入
- ver. 1.1: 光学的厚さが 0.1, 0.01 の部分も出力することにした。qq_in, vc を config のグローバル変数として取扱うこととした。
- ver. 1.2: データ構造を変更。

最終更新日：2020 年 06 月 02 日

第 12 章

Paraview を用いた 3D データ表示

ここでは、Paraview を用いて R2D2 の計算結果を三次元表示する方法を説明する。

12.1 データ準備

すでに計算を実行していて、何らかのデータが準備できている状況を想定する。データを Paraview で扱うために VTK フォーマットに変換する。変換後に用意するべきデータは

- ある物理量の三次元データ
- ある物理量の $\tau = 1$ でのデータ

それぞれのデータのために R2D2 Python では以下の関数が用意してある。

`R2D2.vtk.write_3D(qq, x, y, z, file, name)`

3 次元のスカラー量を VTK フォーマットで出力するための関数

パラメータ

- `qq (float)` -- 出力したい物理量の 3 次元 numpy 配列
- `x (float)` -- x 方向の座標
- `y (float)` -- y 方向の座標
- `z (float)` -- z 方向の座標
- `file (str)` -- 出力先の VTK フォーマットの名前
- `name (str)` -- Paraview で管理するために用いる出力した物理量の名前

`R2D2.vtk.write_vtk.write_3D_vector(qx, qy, qz, x, y, z, file, name)`

3 次元のベクトル量を VTK フォーマットで出力するための関数

パラメータ

- **qx** (*float*) -- x 方向のベクトルの 3 次元 numpy 配列
- **qy** (*float*) -- y 方向のベクトルの 3 次元 numpy 配列
- **qz** (*float*) -- z 方向のベクトルの 3 次元 numpy 配列
- **x** (*float*) -- x 方向の座標
- **y** (*float*) -- y 方向の座標
- **z** (*float*) -- z 方向の座標
- **file** (*str*) -- 出力先の VTK フォーマットの名前
- **name** (*str*) -- Paraview で管理するために用いる出力した物理量の名前

`R2D2.vtk.write_optical_surface(qq, height, y, z, file, name)`

$\tau = 1$ での 2 次元データを出力するための関数

パラメータ

- **qq** (*float*) -- 出力したい物理量の 2 次元配列
- **height** (*float*) -- $\tau = 1$ の高さの情報。self.qt['he'] を出力することが想定されている。
- **y** (*float*) -- y 方向の座標
- **z** (*float*) -- z 方向の座標
- **file** (*str*) -- 出力先 VTK フォーマットの名前
- **name** (*str*) -- Paraview で管理するために用いる出力した物理量の名前

例えば、以下のようにして実行する

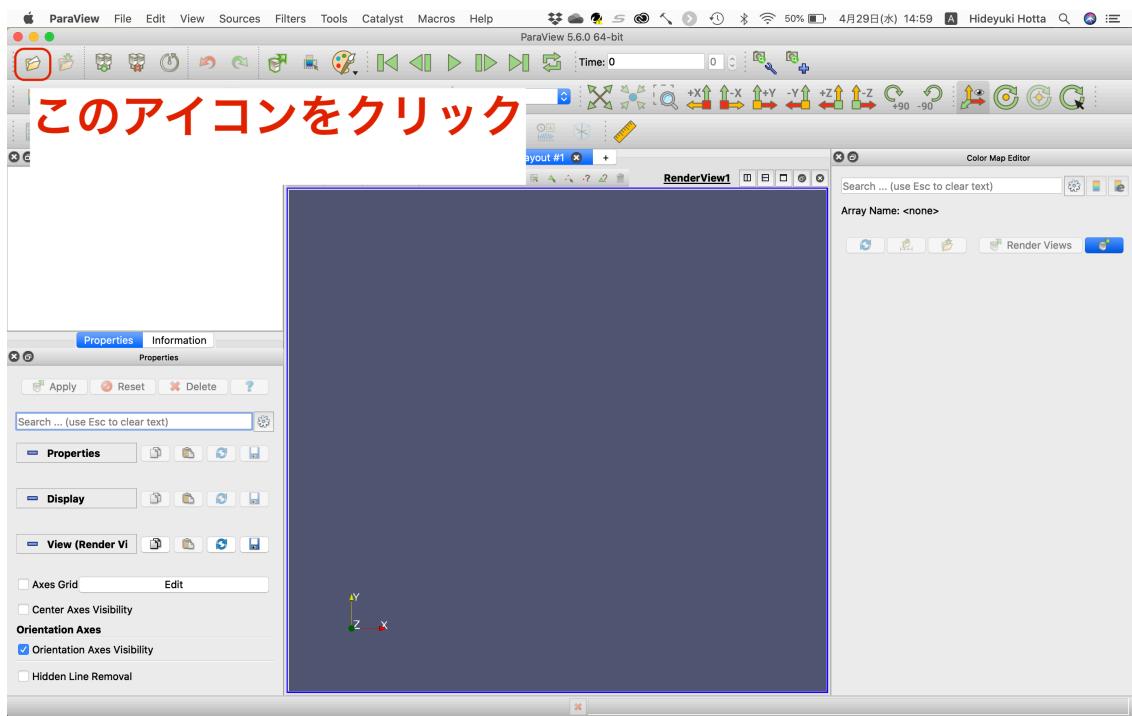
```
run init # 初期設定
d.read_qq(100) # 100 番目の 3 次元データを読み込む
bb = sqrt(d.qq['bx']**2 + d.qq['by']**2 + d.qq['bz']**2) # 磁場の強さを計算
d.read_tau(100) # 100 番目の tau=1 の 2 次元データを読み込む

R2D2.vtk.write_3D(bb, x, y, z, 'bb.vtk', 'bb')
# 変数名を bb としてファイル名 bb.vtk に 3 次元データを出力
R2D2.vtk.write_optical_surface(d.qt['in'], d.qt['he'], y, z, 'in.vtk', 'in')
# 変数名を in としてファイル名 in.vtk に 2 次元データを出力
```

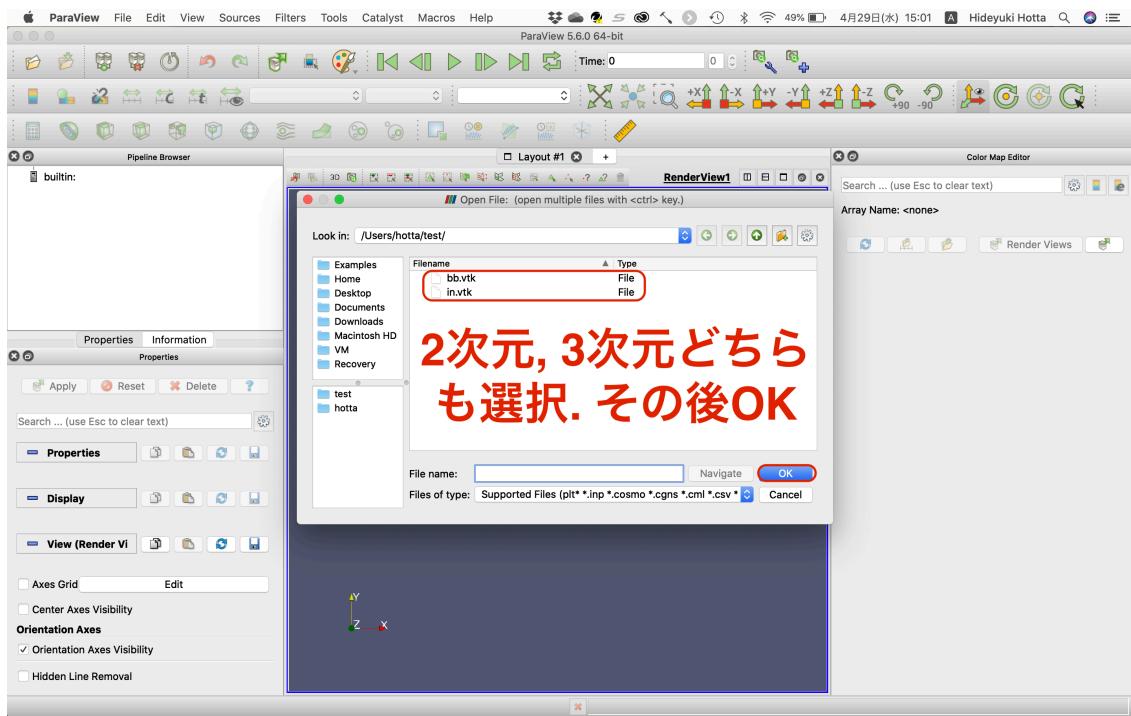
12.2 Paraview を用いて 3 次元表示

Paraview のサイト から Paraview をダウンロード。Windows, Linux, macOS のそれぞれのソフトウェアがあるのでインストール方法は各自確認すること。ここでは、macOS での利用方法を示すが、確認している限りは、Linux でもほとんど同じ。ここでは非常に簡単に Paraview の使い方を説明する。詳しくは Paraview の公式マニュアルなどを読むこと。

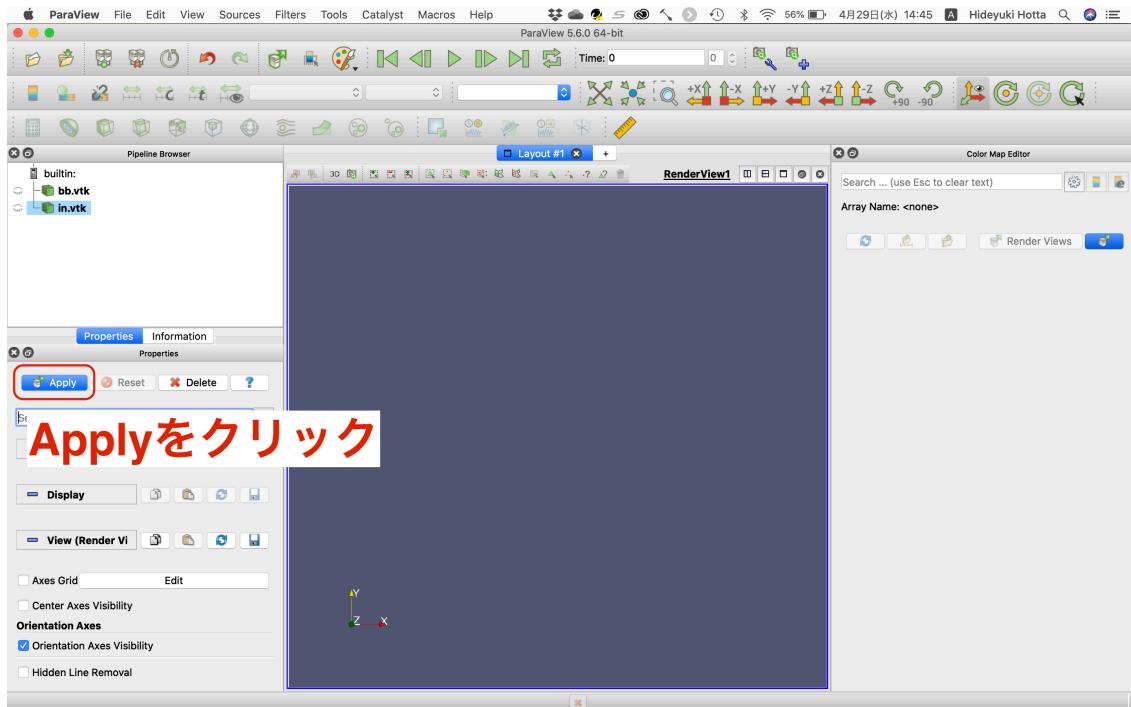
1. まず右上のファイルアイコンをクリック



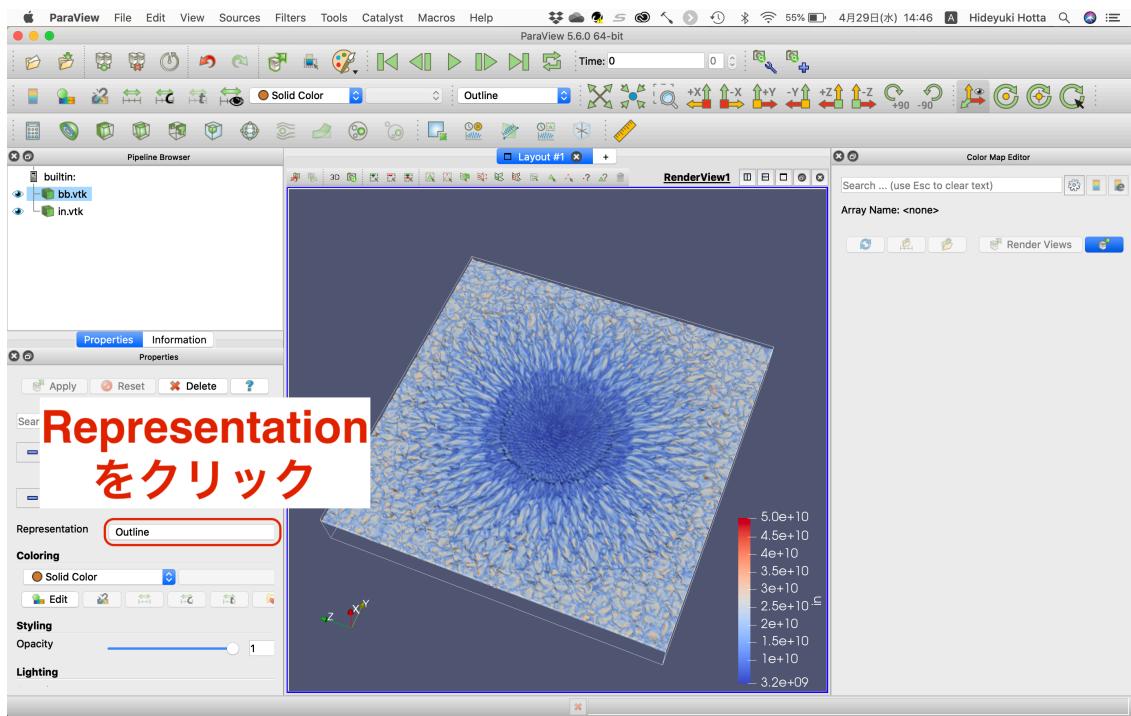
2. Python で生成したファイルを選択。2 次元, 3 次元ファイルどちらも選択する。一個一個選択しても良いし、一度に選択しても良い。時系列データの時は、すべてを一度に選択するとアニメーションを作りやすい。



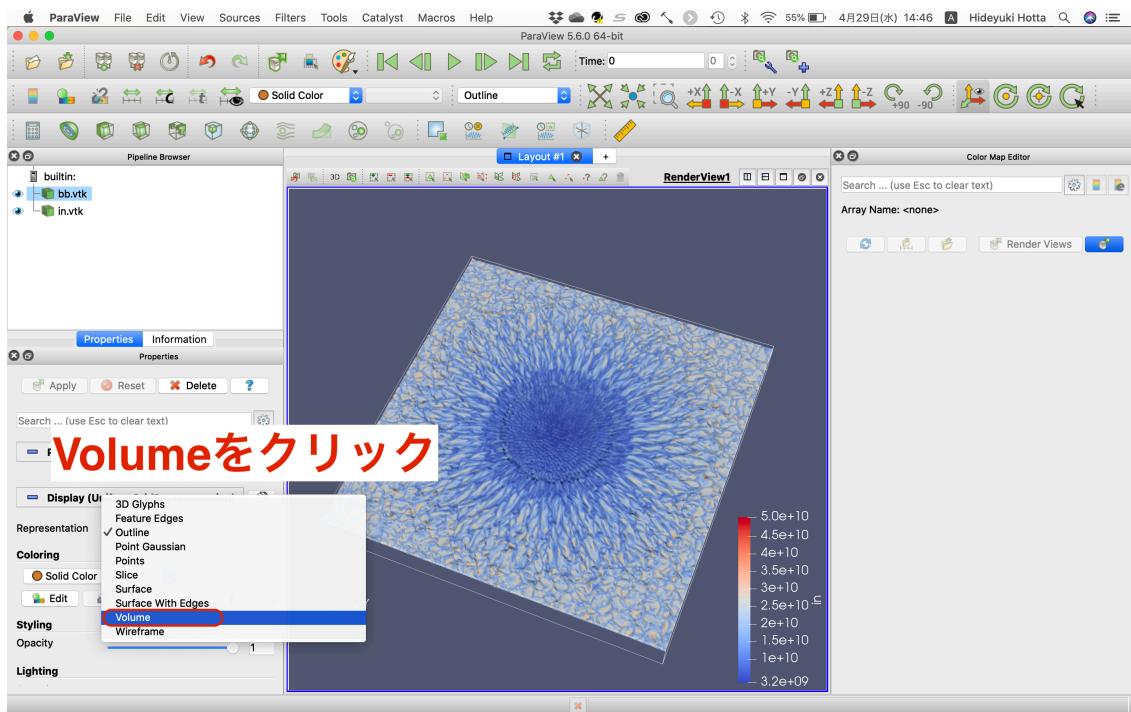
3. Apply をクリック。選択した二つのデータが表示される。



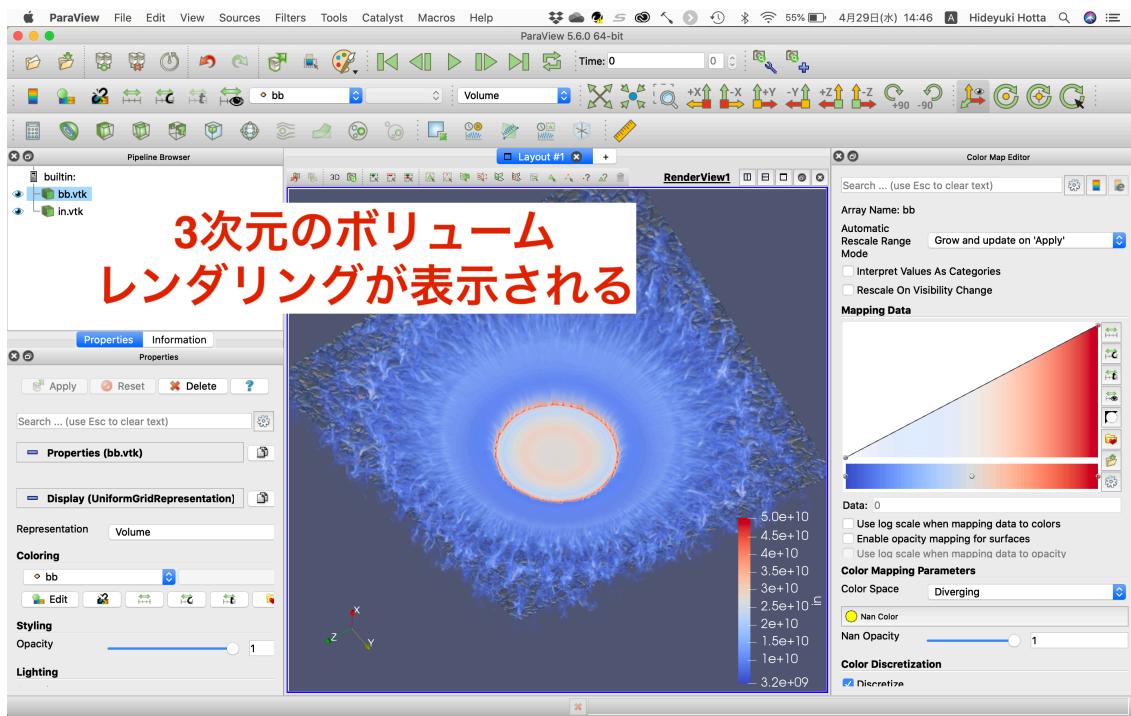
4. 2次元データの方は、すぐに面として表示されるが、三次元データは表示方法を選ぶ必要がある。



5. 三次元データのボリュームレンダリングが行いたいので、Volumeを選ぶ。



6. 三次元データのボリュームレンダリングが表示されるので、便宜描画を回転させて、解析する。



最終更新日：2020年05月10日

第 13 章

R2D2 python でのキーワードの説明

以下では、R2D2 python で使われている辞書型に含まれるキーの説明を行う

- キーの名前 (型) -- 説明 [単位]

というフォーマットを採用する。

R2D2 では、R2D2_data というクラスを用意している。

13.1 self.p [dictionary]

```
import R2D2
self = R2D2.R2D2_data(datadir)
```

とすると、初期設定が読み込まれる。self は R2D2_data のオブジェクトであり、名前は任意である。init.py や mov.py では、オブジェクト名は d としてある。

13.1.1 出力・時間に関する量

- datadir (str) -- データの保存場所
- nd (int) -- 現在までのアウトプット時間ステップ数 (3 次元データ)
- nd_tau (int) -- 現在までのアウトプット時間ステップ数 (光学的厚さ一定のデータ)
- dtout (float) -- 出力ケーデンス [s]
- dtout_tau (float) -- 光学的厚さ一定のデータの出力ケーデンス [s]
- ifac (int) -- dtout/dtout_tau
- tend (float) -- 計算終了時間。大きく取ってあるためにこの時間まで計算することはあまりない [s]

- swap (int) -- エンディアン指定。big endian は 1、little endian は 0。IDL の定義に従っている。
- endian (char) -- エンディアン指定。big endian は >、little endian は <。python の定義に従っている。
- m_in (int) -- 光学的厚さ一定のデータを出力する変数の数
- m_tu (int) -- 光学的厚さ一定のデータの層の数

13.1.2 座標に関する量

- xdcheck (int) -- x 軸方向に解いているか。解いていたら 2、解いていなかったら 1
- ydcheck (int) -- y 軸方向に解いているか。解いていたら 2、解いていなかったら 1
- zdcheck (int) -- z 軸方向に解いているか。解いていたら 2、解いていなかったら 1
- margin (int) -- マージン (ゴーストセル) の数
- nx (int) -- 1 MPI スレッドあたりの x 方向の格子点の数
- ny (int) -- 1 MPI スレッドあたりの y 方向の格子点の数
- nz (int) -- 1 MPI スレッドあたりの z 方向の格子点の数
- ix0 (int) -- x 方向の MPI 領域分割の数
- jx0 (int) -- y 方向の MPI 領域分割の数
- kx0 (int) -- z 方向の MPI 領域分割の数
- ix (int) -- x 方向の格子点数 $ix0 * nx$
- jx (int) -- y 方向の格子点数 $jx0 * ny$
- kx (int) -- z 方向の格子点数 $kx0 * nz$
- npe (int) -- 全 MPI スレッドの数 $npe = ix0 * jx0 * kx0$
- mtype (int) -- 変数の数
- xmax (float) -- x 方向境界の位置 (上限値) [cm]
- xmin (float) -- x 方向境界の位置 (下限値) [cm]
- ymax (float) -- y 方向境界の位置 (上限値) [cm]
- ymin (float) -- y 方向境界の位置 (下限値) [cm]
- zmax (float) -- z 方向境界の位置 (上限値) [cm]
- zmin (float) -- z 方向境界の位置 (下限値) [cm]

- x (float) [ix] -- x 方向の座標 [cm]
- y (float) [jx] -- y 方向の座標 [cm]
- z (float) [kx] -- z 方向の座標 [cm]
- xr (float) [ix] -- x/rsun
- xn (float) [ix] -- $(x - rsun) * 1.e-8$
- deep_top_flag (int) --
- ib_excluded_top (int) --
- rsun (float) [ix] -- 太陽半径 [cm]

13.1.3 背景場に関する量

- pr0 (float) [ix] -- 背景場の圧力 [dyn cm^{-2}]
- te0 (float) [ix] -- 背景場の温度 [K]
- ro0 (float) [ix] -- 背景場の密度 [g cm^{-3}]
- se0 (float) [ix] -- 背景場のエントロピー [$\text{erg g}^{-1} \text{K}^{-1}$]
- en0 (float) [ix] -- 背景場の内部エネルギー [erg cm^{-3}]
- op0 (float) [ix] -- 背景場のオパシティ [$\text{g}^{-1} \text{cm}^{-2}$]
- tu0 (float) [ix] -- 背景場の光学的厚さ
- dsedr0 (float) [ix] -- 背景場の鉛直エントロピー勾配 [$\text{erg g}^{-1} \text{K}^{-1} \text{cm}^{-1}$]
- dtedr0 (float) [ix] -- 背景場の鉛直温度勾配 [K cm⁻¹]
- dprdro (float) [ix] -- 背景場の $(\partial p / \partial \rho)_s$
- dprdse (float) [ix] -- 背景場の $(\partial p / \partial s)_\rho$
- dtedro (float) [ix] -- 背景場の $(\partial T / \partial \rho)_s$
- dtedse (float) [ix] -- 背景場の $(\partial T / \partial s)_\rho$
- dendro (float) [ix] -- 背景場の $(\partial e / \partial \rho)_s$
- dendse (float) [ix] -- 背景場の $(\partial e / \partial s)_\rho$
- gx (float) [ix] -- 重力加速度 [cm s⁻²]
- kp (float) [ix] -- 放射拡散係数 [$\text{cm}^2 \text{s}^{-1}$]

- cp (float) [ix] -- 定圧比熱 [erg g⁻¹ K⁻¹]
- fa (float) [ix] -- 対流層の底付近の輻射によるエネルギーflux。光球付近では輻射輸送を直に解くために含まれないが、上部境界が光球ではない場合は、上部境界付近の人工的なエネルギーflux (冷却が含まれる) [erg cm⁻²]
- sa (float) [ix] -- 上記 fa による加熱率 [erg cm⁻³]
- xi (float) [ix] -- 音速抑制率
- ix_e (int) -- 状態方程式の密度の格子点数
- jx_e (int) -- 状態方程式のエントロピーの格子点数

13.1.4 解析のためのデータ再配置 (remap) に関する量

- m2da (int) -- remap で出力した解析量の数
- cl (char) [m2da] -- remap で出力した解析量の名前
- jc (int) -- self.vc ['vxp'] などで出力するスライスの y 方向の位置
- kc (int) -- 浮上磁場の中心と思っている場所を出力 (あまり使わない)
- ixr (int) -- remap 後の x 方向分割の数
- jxr (int) -- remap 後の y 方向分割の数
- iss (int) [npe] -- remap 後配列のそれぞれの MPI プロセスの x 方向の初めの位置
- iee (int) [npe] -- remap 後配列のそれぞれの MPI プロセスの x 方向の終わりの位置
- jss (int) [npe] -- remap 後配列のそれぞれの MPI プロセスの y 方向の初めの位置
- jee (int) [npe] -- remap 後配列のそれぞれの MPI プロセスの y 方向の終わりの位置
- iixl (int) [npe] -- remap 後配列のそれぞれの MPI プロセスの x 方向の格子点数
- jjxl (int) [npe] -- remap 後配列のそれぞれの MPI プロセスの y 方向の格子点数
- np_ijr (int) [npe] -- x, y 方向の MPI プロセスの位置を入力すると MPI プロセス番号を返す配列
- ir (int) [npe] -- MPI プロセス番号を入れると x 方向の MPI プロセスの位置を返す配列
- jr (int) [npe] -- MPI プロセス番号を入れると y 方向の MPI プロセスの位置を返す配列
- i2ir (int) [ix] -- x 方向の格子点の位置を入れると x 方向の MPI プロセスの位置を返す配列
- j2jr (int) [jx] -- y 方向の格子点の位置を入れると y 方向の MPI プロセスの位置を返す配列

13.1.5 スライスデータに関する量

- nx_slice [int] -- x 一定面のスライスの数
- ny_slice [int] -- y 一定面のスライスの数
- nz_slice [int] -- z 一定面のスライスの数
- x_slice [float] -- x 一定面のスライスの位置 [cm]
- y_slice [float] -- y 一定面のスライスの位置 [cm]
- z_slice [float] -- z 一定面のスライスの位置 [cm]

13.2 self.qs [dictionary]

```
xs = 0.99*rsun
ns = 10
self.read_qq_select(xs,ns)
```

として高さ xs での二次元スライスを読み込む

- ro (float) [jx,kx] -- 密度の擾乱 ρ_1 [g cm^{-3}]
- vx (float) [jx,kx] -- x 方向の速度 v_x [cm s^{-1}]
- vy (float) [jx,kx] -- y 方向の速度 v_y [cm s^{-1}]
- vz (float) [jx,kx] -- z 方向の速度 v_z [cm s^{-1}]
- bx (float) [jx,kx] -- x 方向の磁場 B_x [G]
- by (float) [jx,kx] -- y 方向の磁場 B_y [G]
- bz (float) [jx,kx] -- z 方向の磁場 B_z [G]
- se (float) [jx,kx] -- エントロピーの擾乱 s_1 [$\text{erg g}^{-1} \text{K}^{-1}$]
- pr (float) [jx,kx] -- 圧力の擾乱 p_1 [dyn cm^{-2}]
- te (float) [jx,kx] -- 温度の擾乱 T_1 [K]
- op (float) [jx,kx] -- 不透明度(オパシティ) κ [$\text{g}^{-1} \text{cm}^{-2}$]

13.3 self.qq [dictionary]

self.qs と同様

13.4 self.qt [dictionary]

ほぼ self.qs と同様だが、以下の追加量が保存してある。

13.5 self.vc [dictionary]

最終更新日：2020 年 05 月 10 日

第 14 章

Sphinx 使用の覚書

14.1 はじめに

Sphinx は、reStructuredText から HTML や Latex などの文章を生成するソフトウェアである。Sphinx の公式サイト 最近では Markdown でも記述できるが、結局最後のところは reStructuredText で記述することになるので、現状では、Markdown は使用していない。このウェブサイトも Sphinx で生成しているので、覚書をここに記す。

14.2 インストール

ここでは Anaconda がすでにインストールしてある Mac に Sphinx をインストールすることを考える。基本的には以下のコマンドを実行するのみである。

```
pip install sphinx
```

Markdown を使いたい時は以下のようにする。

```
pip install commonmark recommonmark
```

14.3 HTML ファイルの生成

適当なディレクトリを作成(ここでは test)とする。そこで、sphinx-quickstart コマンドにより Sphinx で作るドキュメントの初期設定を行う。

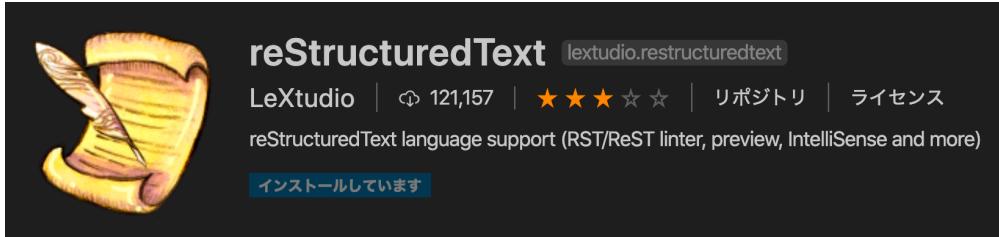
```
mkdir test # ディレクトリ作成
cd test    # ディレクトリに移動
sphinx-quickstart
```

いくつか質問をされる。基本的には読めばわかる質問であるが少し戸惑う質問を以下にあげる。

- プロジェクトのリリース: 1.0 などと version を答える。後に conf.py を編集すれば変更可能
- プロジェクトの言語: デフォルトは英語の en であるが、日本語を使いたい時は ja とする

14.4 VS code の利用

VS code を利用すると快適に reStructuredText を作成することができる。*.rst ファイルを VS code で開くと自動で確認されるが、以下のプラグインをインストールする。



Cmd+k Cmd+r で画面を分割してプレビューできる。正しい conf.py の場所を設定する必要がある。

14.5 環境設定

デフォルトの設定では、数式を書く時に Mathjax を使用するようで、数式の太字が意図するように表示されなかつたので svg で出力することにした。以下のように conf.py に追記する。

```
extensions += ['sphinx.ext.imgmath']
imgmath_image_format = 'svg'
imgmath_font_size = 14
pngmath_latex='platex'
```

また、ウェブサイトのテーマを変更することもできる。どのようなテーマがあるかは Sphinx のテーマを参照。好きなテーマを選んで conf.py に以下のように設定。

```
html_theme = 'bizstyle'
html_theme_options = {'maincolor' : "#696969"}
```

今後変更の余地あり。

14.6 記法

14.6.1 リンク

- 外部ウェブサイト

```
`Twitter <https://twitter.com>`_
```

などとすると

`Twitter`

とリンクが生成される

- 内部サイト

自分で作成しているドキュメントをリンクするには

```
:doc:`index`
```

などとすると

`R2D2 マニュアル`

とリンクが生成される。

14.6.2 コード

Sphinx では、コードを直接記載することができる。また、言語に合わせてハイライトも可能。コードの表記に選択できる言語は `Pygments` にまとめてある。

```
.. code:: fortran

    implicit none
    real(KIND=0.d0) :: a,b,c

    a = 1.d0
    b = 2.d0
    c = a + b
```

このようにすると、以下のように表示される

```
implicit none
real(KIND=0.d0) :: a,b,c

a = 1.d0
```

(次のページに続く)

(前のページからの続き)

```
b = 2.d0
c = a + b
```

14.6.3 画像

画像の挿入には `image` ディレクティブを使う。オプションで、画像サイズなどを調整できる。堀田はだいたい `width` で調整している。

```
.. image:: source/figs/R2D2_logo.png
    :width: 350 px
```

とすると下記のように画像が挿入される。



14.6.4 数式

Sphinx では Latex を用いて数式を記述することができる。1 行の独立した数式を取り扱うときは

```
.. math::

\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{v})
```

とすると以下のように表示される。

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{v})$$

インラインの数式では

```
ここで :math:`\rho_1=x^2` とする
```

とすると

ここで $\rho_1 = x^2$ とする

と表示される。

14.6.5 To DO

To Do を書いておきたい場所に

```
... todo:: 方程式を書く
```

と書くと、To Do が示される。トップページに

```
... todolist:
```

と書いてあるので、To Do のまとめが示されている。

最終更新日：2020 年 05 月 10 日

第 15 章

ライセンス

R2D2 は現状、公開ソフトウェアではなく再配布も禁じている。これは今後、変更される可能性はあるが、開発者(堀田)の利益を守るためにある。共同研究者のみが使って良いというルールになっており、R2D2 の使用には、以下の規約を守る必要がある。

- 再配布しない
- 改変は許されるが、その時の実行結果について堀田は責任を持たない
- R2D2 で実行する計算は、堀田と議論する必要がある。パラメタ変更などの細かい変更には相談する必要はないが、新しいプロジェクトを開始するときはその都度相談すること。堀田自身のプロジェクト、堀田の指導学生のプロジェクトとの重複を避けるためである。
- R2D2 を用いた論文を出版するときは Hotta et al., 2019, Hotta and Iijima, 2020 を引用すること。
- R2D2 を用いた研究を発表するときは、R2D2 のロゴの使用が推奨される(強制ではない)。

改訂履歴

2020 年 4 月 29 日 Hotta and Iijima, 2020 を加えた

第 16 章

出版論文

R2D2 を用いた研究で出版された論文は以下です。

- Hotta, Iijima, and Kusano, 2019, Science Advances, 5, eaau2307
- Toriumi and Hotta, 2019, ApJ, 886, L21
- Hotta and Iijima, 2020, MNRAS, 494, 2523

第 17 章

索引と検索ページ

- genindex
- search

第 18 章

TODO リスト

課題: 人工粘性

(元のエントリは、/Users/hotta/Dropbox/work/R2D2/R2D2-manual/artdif.rst の 4 行目です)

課題: コーディングルール

(元のエントリは、/Users/hotta/Dropbox/work/R2D2/R2D2-manual/code.rst の 6 行目です)

課題: コード構造

(元のエントリは、/Users/hotta/Dropbox/work/R2D2/R2D2-manual/code.rst の 10 行目です)

課題: 輻射輸送の方程式

(元のエントリは、/Users/hotta/Dropbox/work/R2D2/R2D2-manual/equation.rst の 43 行目です)

課題: パラメータ。どのグローバル変数がどのモジュールで定義されているかを整理する。

(元のエントリは、/Users/hotta/Dropbox/work/R2D2/R2D2-manual/parameter.rst の 4 行目です)

課題: 数値スキーム (時間積分)

(元のエントリ は、 /Users/hotta/Dropbox/work/R2D2/R2D2-manual/scheme.rst の 23 行目です)

課題： 数値スキーム（輻射輸送）

(元のエントリ は、 /Users/hotta/Dropbox/work/R2D2/R2D2-manual/scheme.rst の 28 行目です)

課題： 全対流層計算の設定例

(元のエントリ は、 /Users/hotta/Dropbox/work/R2D2/R2D2-manual/typical_case.rst の 327 行目です)

課題： 深い部分のみの計算の設定例

(元のエントリ は、 /Users/hotta/Dropbox/work/R2D2/R2D2-manual/typical_case.rst の 332 行目です)

最終更新日：2020 年 05 月 10 日

Python モジュール索引

r

R2D2, 36

索引

`__init__()` (*R2D2.R2D2_data* のメソッド), 37
`p` (*R2D2.R2D2_data* の属性), 36
`qc` (*R2D2.R2D2_data* の属性), 37
`q1` (*R2D2.R2D2_data* の属性), 37
`qq` (*R2D2.R2D2_data* の属性), 36
`qs` (*R2D2.R2D2_data* の属性), 36
`qt` (*R2D2.R2D2_data* の属性), 36

R2D2
 モジュール, 36
`R2D2.vtk.write_3D()`
 組み込み関数, 40
`R2D2.vtk.write_optical_surface()`
 組み込み関数, 41
`R2D2.vtk.write_vtk.write_3D_vector()`
 組み込み関数, 40
`R2D2_data` (*R2D2* のクラス), 36
`read_qq()` (*R2D2.R2D2_data* のメソッド), 37
`read_qq_check()` (*R2D2.R2D2_data* のメソッド), 38
`read_qq_select()` (*R2D2.R2D2_data* のメソッド), 38
`read_qq_slice()` (*R2D2.R2D2_data* のメソッド), 38
`read_qq_tau()` (*R2D2.R2D2_data* のメソッド), 38
`read_time()` (*R2D2.R2D2_data* のメソッド), 38
`read_vc()` (*R2D2.R2D2_data* のメソッド), 38

`vc` (*R2D2.R2D2_data* の属性), 37

モジュール
 R2D2, 36
組み込み関数
 `R2D2.vtk.write_3D()`, 40
 `R2D2.vtk.write_optical_surface()`, 41
 `R2D2.vtk.write_vtk.write_3D_vector()`, 40