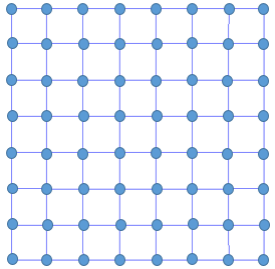
数値流体計算の基礎：簡単なイメージ

1. 空間をメッシュ化する

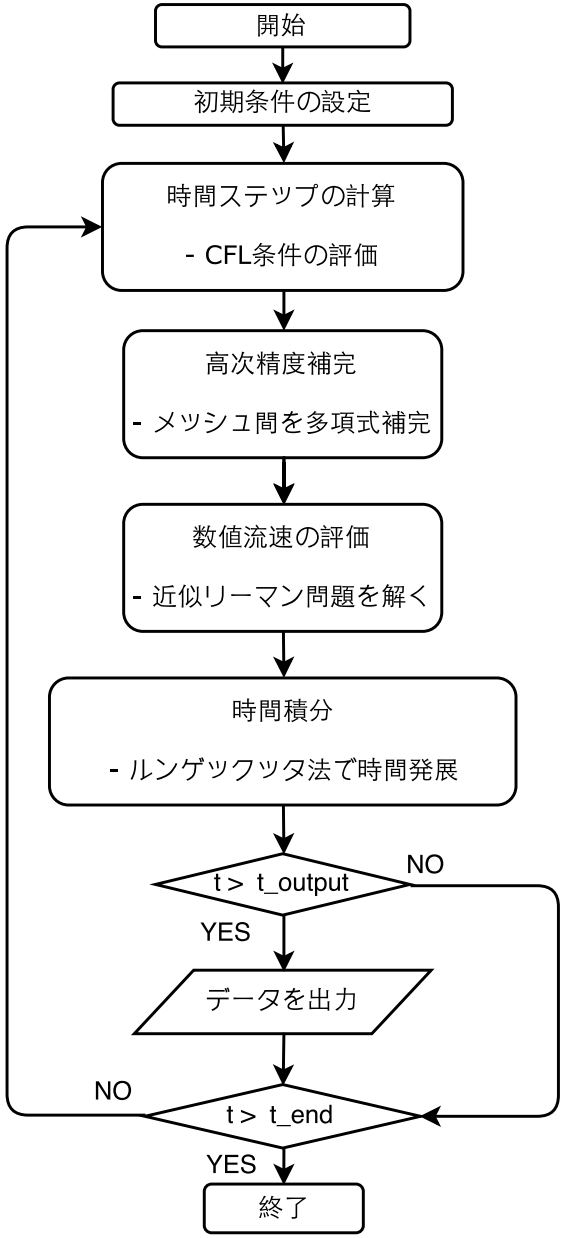


2. 流体の方程式(偏微分方程式)を指定し、
初期条件、境界条件を与えて数値的に解く。

基本的にはこれだけ。何が難しく、あるいは複雑にさせているのか？

1) スパコンを使っても、メッシュ数をものすごく大きくはとれない。メッシュ数を増やさなくても、精度がよくなる方法が必要。
2) 計算の時間ステップを大きくとれない。一つのメッシュを音速が伝わる時間が最大(クーラン条件)の時間幅となる。
3) 理学の場合、スピードよりも、正しさが求められる。

数値流体計算の基礎：簡単な流れ



← 空間をメッシュ化する

← 時間ステップ < メッシュサイズ ➗ 典型的な速度　クーラン条件

← メッシュを”滑らかに”して、境界の値を計算　空間高次精度化

← 境界での不連続な値の時間発展を近似的に解く　リーマン問題

← 時間発展を計算(e.g., ルンゲクッタ法)　時間高次精度化

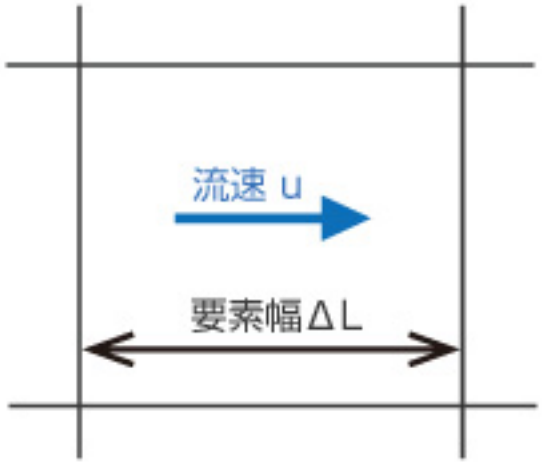
← データを保存する or スキップする

３つ含めて“近似リーマン解法”

← 指定した時間まで計算が終わったら終了する。

数値流体計算の基礎：クーラン条件

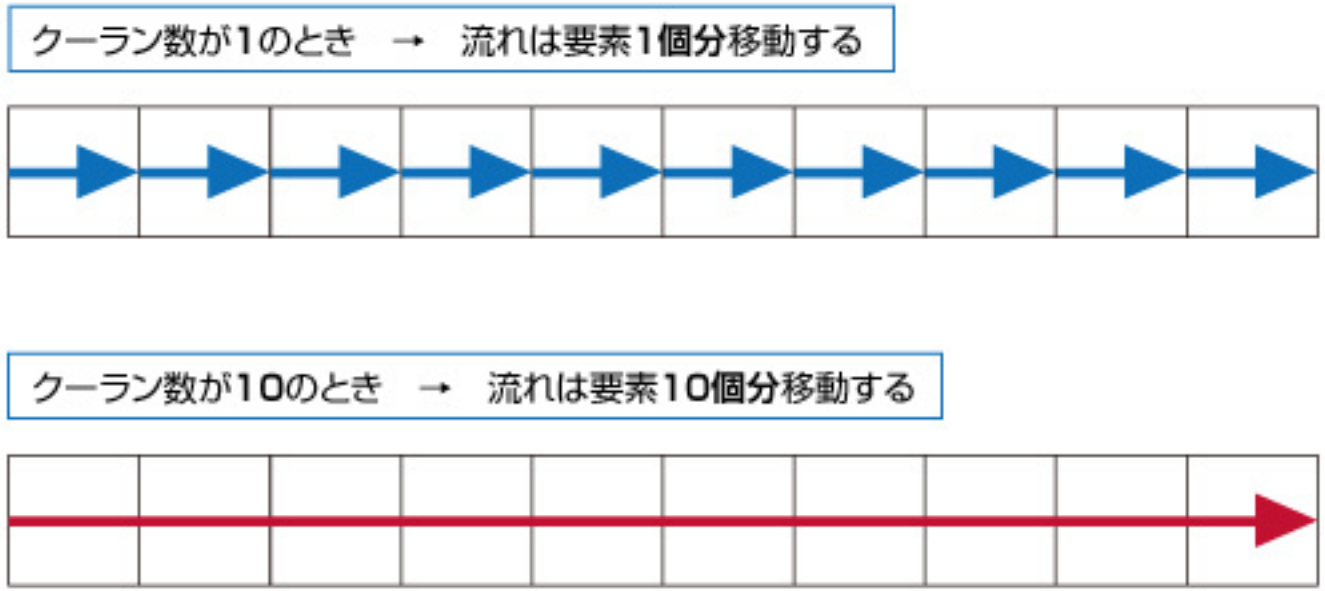
空間メッシュサイズ ΔL を 典型的な流体の速度 u で
割った時間 $\Delta L / u$ が、大きくできる時間ステップの最大値。
それより時間幅が大きいと、情報がセルを飛び越えてしまう。



時間刻みの指標として、クーラン数を、 $C = \frac{u\Delta t}{\Delta L}$
と定義する。

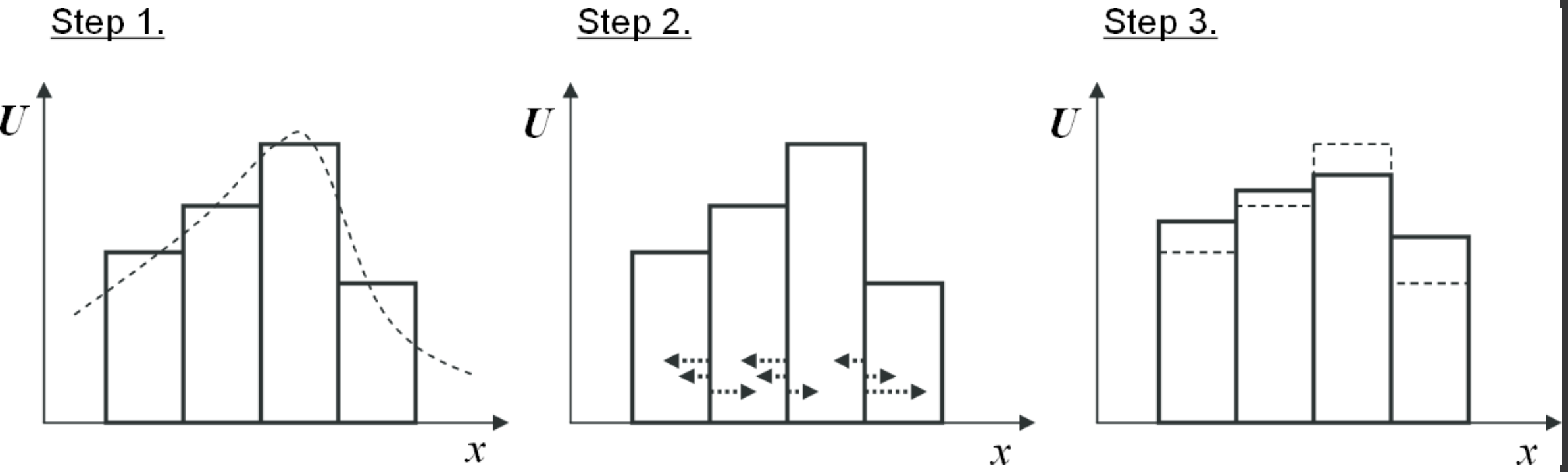
通常、クーラン数は、1 よりも
小さい数に設定する。

音速、流速が早い場合は、時間刻み
が小さくなり、シミュレーションを
ゆっくりとしか進められない。



情報がセルを超えて伝わってしまう。

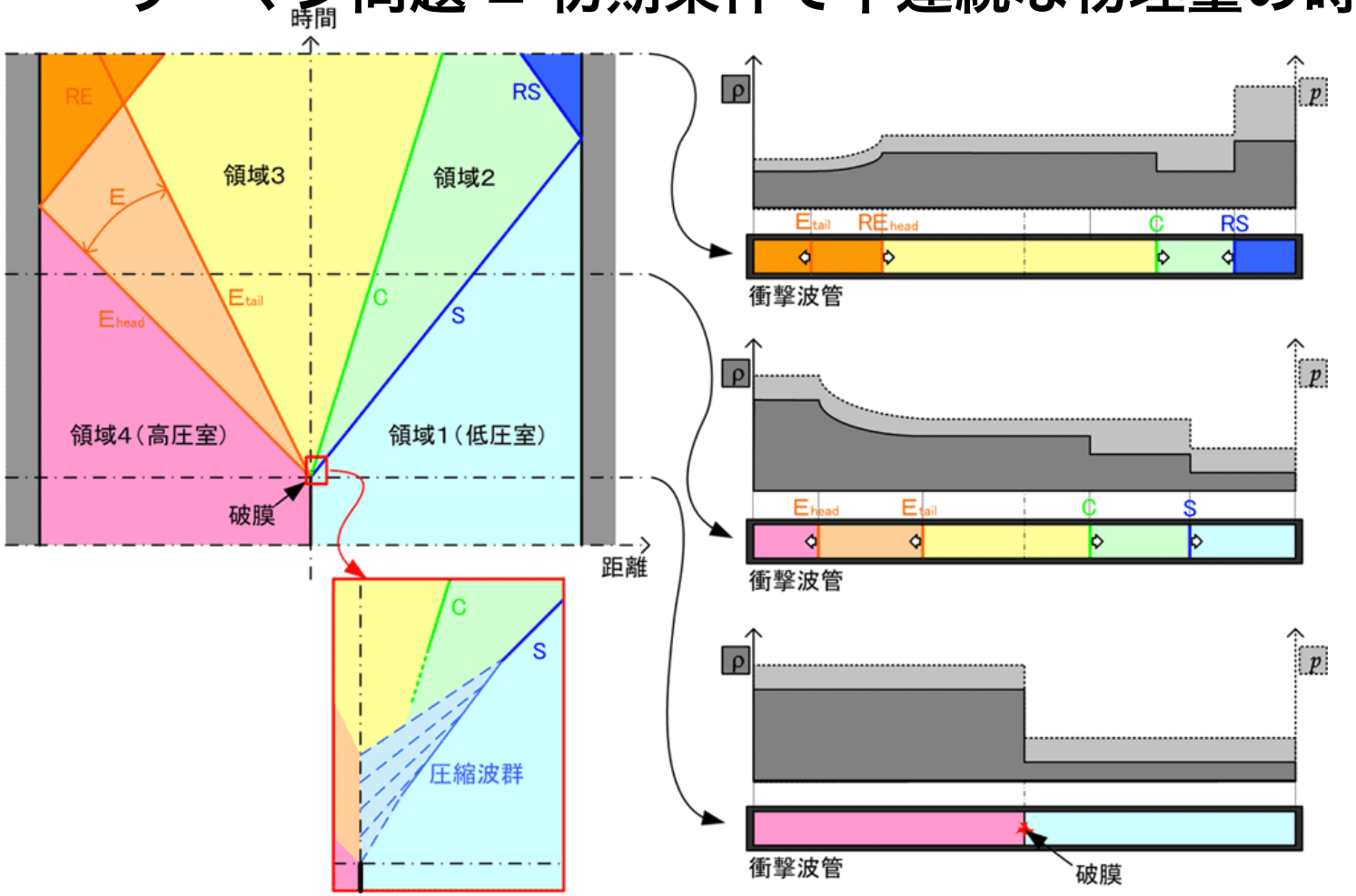
数値流体計算の基礎：近似リーマン解法



Step 1. 物理量を区分的定数関数で近似する。
Step 2. 各セル境界でリーマン問題を近似的に解く。
Step 3. 各セル内でリーマン問題の解を積分する。

数値流体計算の基礎：リーマン問題

リーマン問題 = 初期条件で不連続な物理量の時間発展をどう解くか？



衝撃波管問題
- シンプルなリーマン問題の例

数値シミュレーションは、物理量を離散化して解くこと。
すなわち、時々刻々とセル境界で発生するリーマン問題をどう解くか、という問題でもある。

数値流体計算の基礎：時間高次精度化

ルンゲクッタ法などで、時間発展を解く。

2.1.7 TIME_STEPPING

PLUTO has several time-marching algorithms which can be used in either a spatially split or unsplit fashion. If $\Delta t^n = t^{n+1} - t^n$ is the time increment between two consecutive steps and $\boldsymbol{\mathcal{L}}$ denotes the discretized spatial operator on the right hand side of Eq. (1.1), the possible options are:

- *EULER*: 1st (explicit) Euler algorithm is used to evolve from $\boldsymbol{U}^n$ to $\boldsymbol{U}^{n+1}$:

$$\boldsymbol{U}^{n+1} = \boldsymbol{U}^n + \Delta t^n \boldsymbol{\mathcal{L}}^n$$

- *RK2*, *RK3*: 2nd or 3rd-order TVD Runge Kutta is used to advance the solution to the next time level:

RK2	RK3
$\boldsymbol{U}^* = \boldsymbol{U}^n + \Delta t^n \boldsymbol{\mathcal{L}}^n$	$\boldsymbol{U}^* = \boldsymbol{U}^n + \Delta t^n \boldsymbol{\mathcal{L}}^n$
–	$\boldsymbol{U}^{**} = \frac{1}{4}\left(3\boldsymbol{U}^n + \boldsymbol{U}^* + \Delta t^n \boldsymbol{\mathcal{L}}^*\right)$
$\boldsymbol{U}^{n+1} = \frac{1}{2}\left(\boldsymbol{U}^n + \boldsymbol{U}^* + \Delta t^n \boldsymbol{\mathcal{L}}^*\right)$	$\boldsymbol{U}^{n+1} = \frac{1}{3}\left(\boldsymbol{U}^n + 2\boldsymbol{U}^{**} + 2\Delta t^n \boldsymbol{\mathcal{L}}^{**}\right)$

(2.1)

When DIMENSIONAL_SPLITTING = *YES*, the operator $\boldsymbol{\mathcal{L}}$ in Eq. (2.1) is one-dimensional. Setting DIMENSIONAL_SPLITTING to *NO* makes the scheme dimensionally unsplit and the right hand side include contributions from all directions simultaneously. Unsplit implementation of the Runge-Kutta algorithms usually requires a more restrictive CFL condition, see Table 2.1.

Pluto の userguide.pdf より

pluto：使用方法 -- 最も簡単な例

```
cd Test_Problems/HD/Sod
cd testrun
cp ../init.c .
cp ../pluto_01.ini pluto.ini
cp ../definitions_01.h definitions.h
python $PLUTO_DIR/setup.py → コンパイラを選択して抜けると、makefile が生成される。
make
./pluto
```

最低限に必要な３つのファイル

pluto.ini：起動時に読み込む初期設定ファイル
definitions.h：コンパイル時に読み込む初期設定
init.c：初期条件、境界条件が書かれたソースコード

その他に必要なファイルは、$PLUTO/Src 以下から呼ばれる。

pluto：使用方法 – init.c

初期条件を設定する。

```
void Init (double *v, double x1, double x2, double x3)
```

”セルの数”だけ呼び出される。
x1, x2, x3 がシミュレーション空間の離散化された座標
v はあるセルの物理量の配列。e.g., v[RHO] = 密度、v[PRS] = 圧力

境界条件を設定する。

```
void UserDefBoundary (const Data *d, RBox *box, int side, Grid *grid)
```

”境界の数”だけ呼び出される。
Data d = 全セル情報を保有する構造体。e.g.,
d->Vc[RHO][k][j][i] = (k, j, i)番目 のセルの密度
d->Vc[VX1][k][j][i] = (k, j, i)番目 のセルの１次元目の速度
d->Vc[VX2][k][j][i] = (k, j, i)番目 のセルの２次元目の速度
Rbox box = 境界のセルの配列。
int side = どの境界面かを示す整数。場合わけを簡単にするためのもの。
Grid *gird = シミュレーションの座標情報 e.g,
x1all = grid[IDIR].x; 一次元目の座標の配列を取得する

pluto：使用方法 – definition.h

コンパイル時に用いられる初期設定。

```
#define  PHYSICS                HD
#define  DIMENSIONS              1
…
#define  ROTATING_FRAME         NO
```

HD = 磁場を含まない流体計算

シミュレーションの次元

座標系の回転あり、なし。

注) このあとに、python $PLUTO/setup.py でインタラクティブに変更した場合は、
この definition.h も変更される。

pluto : 使用方法 – pluto.ini

実行時に読まれる初期設定。

[Grid]

	空間の分割数 (未使用)	下限値	次元数	一様 grid (Uniform)	上限値
X1-grid	1	0.0	400	u	1.0
X2-grid	1	.0	1	u	1.0
X3-grid	1	0.0	1	u	1.0

この場合、一次元のシミュレーションで x 軸方向を 0-1.0 まで 400 分割する。

[Static Grid Output]

output_dir run1 ← 任意。とすると、run の結果が run1 というディレクトリに丸ごと入るので debug しやすい。ただし、実行前に run1 ディレクトリを作成しておく。

pluto : 使用方法 – コードの変更する場合

作業ディレクトリの直下に、$PLUTO/Src 以下のファイルをコピーする。

そのファイルが本当に使われているのかを確認する。
QUIT_PLUTO(1);
を挿入して、そこで強制終了することをチェック。

pluto：使用方法 – pyPLUTO

インストール方法

```
cd $PLUTO_DIR/Tools/pyPLUTO
python setup.py install
```

```
[syamada] $ ipython

In [1]: import pyPLUTO as pp

In [2]: data1 = pp.pload(1) ← 一つのファイルごとに、一つの data オブジェクトを生成する。
Reading Data file : /Users/syamada/work/software/pluto/PLUTO/Test_Problems/HD/Sod/testrun/data.0001.dbl

In [3]: data1. ← タブを押すと、保存されているメンバー変数、配列がわかる。
data1.DataScan            data1.SimTime             data1.level               data1.wdir
data1.DataScanHDF5        data1.Slice               data1.n1                  data1.x1
data1.DataScanVTK         data1.datatype            data1.n1_tot              data1.x1r
data1.Dt                  data1.dx1                 data1.n2                  data1.x1range
data1.NStep               data1.dx2                 data1.n2_tot              data1.x2
data1.NStepStr            data1.dx3                 data1.n3                  data1.x2r
data1.ReadDataFile        data1.endianess           data1.n3_tot              data1.x2range
data1.ReadGridFile        data1.filetype            data1.nshp                data1.x3
data1.ReadMultipleFiles   data1.geometry            data1.prs                 data1.x3r
data1.ReadSingleFile      data1.irange              data1.rho                 data1.x3range
data1.ReadTimeInfo        data1.jrange              data1.vars
data1.ReadVarFile         data1.krange              data1.vx1

data1.rho
Out[3]:
array([ 1.        ,  1.        ,  1.        ,  1.        ,  1.        ,
        1.        ,  1.        ,  1.        ,  1.        ,  1.        ,
        1.        ,  1.        ,  1.        ,  1.        ,  1.        ,
        1.        ,  1.        ,  1.        ,  1.        ,  1.        ,
...
```

と表示される。あとは、matplotlib などでプロットする。