11.4 Activity 1

In Lesson 11.3, you set up, compiled and ran a 1D (1-dimensional) test problem with the PLUTO MPI code. In this Lesson 11.4 you will solve the Rayleigh-Taylor Instability test problem in 2D. The initial condition consists of an interface separating two fluids with different densities in hydrostatic equilibrium. As the fluids fall into each other, the instability ensues.

At this point, it may be useful to browse the documentation for the PLUTO Test Problems:
http://plutocode.ph.unito.it/Doxygen/Test_Problems/files.html
http://plutocode.ph.unito.it/Doxygen/Test_Problems/_m_h_d_2_rayleigh___taylor_2init_8c.html

As before,you run these simulations on any computer on which you can compile C MPI code. In this activity we provide instructions for the Blue Waters supercomputer at NCSA, and any of the supercomputers in the NSF XSEDE network, including Stampede2 at TACC.

11.4.1 MPI on Blue Waters

On Blue Waters, there is a general purpose wrapper code to compile serial and parallel code using a variety of compilers: Cray, GNU, PGI and Intel. If you are using Blue Waters, then complete this section. First we need to tell PLUTO how to compile MPI code.

```
username@h2ologin2: vi $PLUTO_DIR/Config/Linux.mpicc.defs
```

Verify that the `CC` flag looks like this:

```
#######################################################################
#
#
#     Configuration file for mpicc (parallel)
#
#######################################################################
###
CC        = cc
CFLAGS    = -c -O3
LDFLAGS   = -lm

PARALLEL  = TRUE
USE_HDF5  = FALSE
USE_PNG   = FALSE
```

11.4.2 MPI on XSEDE

Many clusters like Stampede2 use mpicc to compile MPI code.

```
login4.stampede2(1020)$ which mpicc
```

```
/opt/apps/intel18/impi/18.0.2/bin/mpicc
```

However, if your installation returns:

```
/usr/bin/which: no mpicc in …
```

Then you will need to load an MPI compiler. This is often done using the module command.

```
login4.stampede2(1020)$ module avail mpi
login4.stampede2(1020)$ module load impi
login4.stampede2(1020)$ which mpicc

username@h2ologin2: vi $PLUTO_DIR/Config/Linux.mpicc.defs
```

Verify that the `CC` flag looks like this:

```
######################################################################
#
#
#     Configuration file for mpicc (parallel)
#
######################################################################
###
CC        = mpicc
CFLAGS    = -c -O3
LDFLAGS   = -lm

PARALLEL  = TRUE
USE_HDF5  = FALSE
USE_PNG   = FALSE
```

11.4.3 The Rayleigh-Taylor Test Problem

```
username@h2ologin2: cd $PLUTO_DIR/Test_Problems/MHD/Rayleigh_Taylor
username@h2ologin2: cp pluto_01.ini pluto.ini
username@h2ologin2: cp definitions_01.h definitions.h
username@h2ologin2: vi pluto.ini

[Time]

CFL              0.8
CFL_max_var      1.1
tstop            20.0
first_dt         1.e-3

[Static Grid Output]
```

```
uservar    0
dbl        -1.0  -1   single_file
dbl.h5     -1.0  -1
flt        -1.0  -1   single_file
flt.h5     -1.0  -1
vtk         1.0  -1   single_file
tab        -1.0  -1
ppm        -1.0  -1
png        -1.0  -1
log         10
analysis   -1.0  -1
```

```
username@h2ologin2: python $PLUTO_DIR/setup.py
```

Select: `Change makefile`
Select: `Linux.mpicc.defs`
Select: `Auto-update`

```
username@h2ologin2: make
username@h2ologin2: make clean
```

11.4.1 Submit a job with the Portable Batch System (PBS) (Blue Waters)

The command for submitting a batch job on Blue Waters is `qsub`. The command for running the job on the compute nodes on Blue Waters is called `aprun`. On other systems, the command is called `mpirun`. Check the user guide on your system. You will create a simple `qsub` script to setup and run your job. Find out how many physical cores exist on each node on your cluster. On Blue Waters, each node contains two sockets (CPUs) and each socket has 16 physical cores. So `ppn=32`. The minimum `walltime` on Blue Waters is 5 minutes. Edit the script accordingly:

```
username@h2ologin2: vi pluto.pbs
```

```
#!/bin/bash
#PBS -l nodes=1:ppn=32:xe
#PBS -l walltime=00:05:00
#PBS -N pluto

cd $PBS_O_WORKDIR

aprun -n 32 ./pluto
```

Now submit the job with `qsub`. Copy and paste the job name. Check the status of the job using `qstat`. The Status column can read Q (queued), R (running), and C (complete).

```
username@h2ologin2: qsub pluto.pbs
username@h2ologin2: qstat <job>.bw
```

11.4.2 Submit a job with Slurm (XSEDE)

The XSEDE supercomputers use SLURM to manage jobs. You may edit and compile your code on the login node, then submit that job from the login node. SLURM will queue the job, and run the job when your compute nodes become available. Depending on the number of nodes you request, the queue time could be a few minutes, or a few days. The SLURM command for submitting a job is `sbatch`. The command for running the job on the compute nodes on Stampede2 is called `ibrun`. On other systems, the command could be called `mpirun`. Check the user guide for your cluster. You will create a simple `sbatch` script to setup and run your job. Find out how many physical cores exist on each node on your cluster. On the SKX nodes on Stampede2 (like the `skx-normal` and `skx-dev` queues), each node has 2 sockets (CPUs), and each socket has 24 processors (cores). So, `-n 48` and `-N 1`.

On the KNL nodes (like the `normal` and `development` queues on Stampede2), each node can run up to 64 MPI tasks per node. Start with 32, so `-n 32`. Edit the script accordingly:

```
username@h2ologin2: vi pluto.slurm
```

```
#!/bin/bash
#SBATCH -p normal    #chooses the queue type normal
#SBATCH -n 32        #number of cores
#SBATCH -N 1         #number of nodes
#SBATCH -t 00:05:00  #max time to run

ibrun ./pluto
```

Now submit the job with `sbatch`. Check the status of all your jobs using `squeue` or `showq`.

```
username@h2ologin2: sbatch pluto.slurm
username@h2ologin2: showq -u
```

11.4.3 Check your output

When your job completes, MPI should produce a log file for each processor (32) and 15 vtk files obtained at 1-second intervals during the simulation. We will visualize this output later.
For now, check one of the .log files, and one of the pluto.o<job> files.

1. How long did the simulation take?

2. What was the resolution of the grid X-Y grid?
3. The simulation stopped after tstop=15.0 (these are code time units). How long (walltime) would the computation have taken for tstop=20?
4. Imagine you were to double the number of grid points in X and Y. How long (walltime) would the computation have taken for tstop=20?

11.4.4 Modify your inputs

We will now modify our problem, and rerun the simulation. Before we do that. Let's save our results from before in a directory called run01.

```
username@h2ologin2: mkdir run01
username@h2ologin2: mv *.out *.log *.vtk pluto.ini definitions.h
run01
```

We want to spatially refine our simulation by a factor of 2 in X and Y. We want to add a magnetic field, increase the density of the upper fluid, and change the left and right boundary conditions to reflective.

First, edit the pluto.ini file. We will need to modify the Grid block, the Time block, the Boundary block, and the Static Grid Output block.

```
username@h2ologin2: cp pluto_05.ini pluto.ini
username@h2ologin2: cp definitions_05.h definitions.h
username@h2ologin2: vi pluto.ini
```

```
[Grid]

X1-grid   1    -1.0   1024   u     1.0
X2-grid   1    -1.0   1024   u     1.0
X3-grid   1    -0.5      1   u     0.5

[Time]

CFL                 0.8
CFL_max_var         1.1
tstop               20.0
first_dt            1.e-3

[Boundary]

X1-beg     reflective
X1-end     reflective
```

```
X2-beg    reflective
X2-end    reflective
X3-beg    periodic
X3-end    periodic

[Static Grid Output]

uservar     0
dbl        -1.0   -1   single_file
dbl.h5     -1.0   -1
flt        -1.0   -1   single_file
flt.h5     -1.0   -1
vtk         1.0   -1   single_file
tab        -1.0   -1
ppm        -1.0   -1
png        -1.0   -1
log         10
analysis   -1.0   -1

[Parameters]

ETA                                 2.5
GRAV                               -0.1
CHI                                 0.2
```

```
username@h2ologin2: python $PLUTO_DIR/setup.py
```

Select `Auto-update.`

```
username@h2ologin2: make
username@h2ologin2: make clean
```

11.4.5 Submit your job with PBS (Blue Waters)

How long (walltime) do you think this computation will take? Make sure that your walltime parameter in `pluto.pbs` is longer than you estimate. Make sure that your walltime parameter in `pluto.pbs` is at least 60 minutes. Try using 2 nodes, 32 cores per node, 64 tasks.

```
username@h2ologin2: vi pluto.pbs
```

```
#!/bin/bash
#PBS -l nodes=2:ppn=32:xe
#PBS -l walltime=01:00:00
#PBS -N pluto
```

```
cd $PBS_O_WORKDIR

aprun -n 64 ./pluto

username@h2ologin2: qsub pluto.pbs
username@h2ologin2: qstat -u <username>
```

11.4.6 Submit your job with slurm (XSEDE)

How long (walltime) do you think this computation will take? Make sure that your walltime parameter in `pluto.slurm` is longer than you estimate, at least 60 minutes. Try using 2 nodes, 64 cores total.

```
login4.stampede2(1020)$ vi pluto.slurm
```

```
#!/bin/bash
#SBATCH -p normal     #chooses the queue type normal
#SBATCH -n 64         #number of cores
#SBATCH -N 2          #number of nodes
#SBATCH -t 01:00:00   #max time to run
ibrun ./pluto
```

```
login4.stampede2(1020)$ sbatch pluto.slurm
login4.stampede2(1020)$ showq -u
```

11.4.7 Check your output

When your job completes, MPI should produce a log file for each processor (32) and 20 vtk files obtained at 1-second intervals during the simulation. Now, check one of the .log files, and one of the pluto.o<job> files. How long did the simulation take? How good was your estimate?

Let’s save our results in a directory called run05.

```
login4.stampede2(1020)$ mkdir run05
login4.stampede2(1020)$ mv *.out *.log *.vtk pluto.ini definitions.h
run05
```

In the next activity we will visualize our results from run01 and run05.

11.4 Activity 2

In this activity we will apply what we learned in Lesson 11.2 and Lesson 11.4 to visualize the simulation data we generated in 11.4 Activity 1.

Although you can run VisIt in client/server mode (option b. below), you may want to download the .vtk files to your local machine for this activity.

11.4.1. Open the run01 dataset

1. Start VisIt on your workstation.
2. Click on the Open icon to bring up the File open window.
   a. If your data resides on your local machine, select “localhost” from the Host menu.
   b. If your data resides on a remote machine, then select that host from the Host menu.
3. Navigate your file system to the folder run01.
4. Highlight all the .vtk files, then click OK.

11.4.1.1. Modify settings

To speed up animations, we will adjust the default animation controls.

1. Go to *Controls>Animation*
2. Click *Cache Animation for faster playback* and *Animation playback: looping*
3. *Click Apply* and *Dismiss*
4. Go to *Options>Save settings*

11.4.1.2. Examining scalar fields

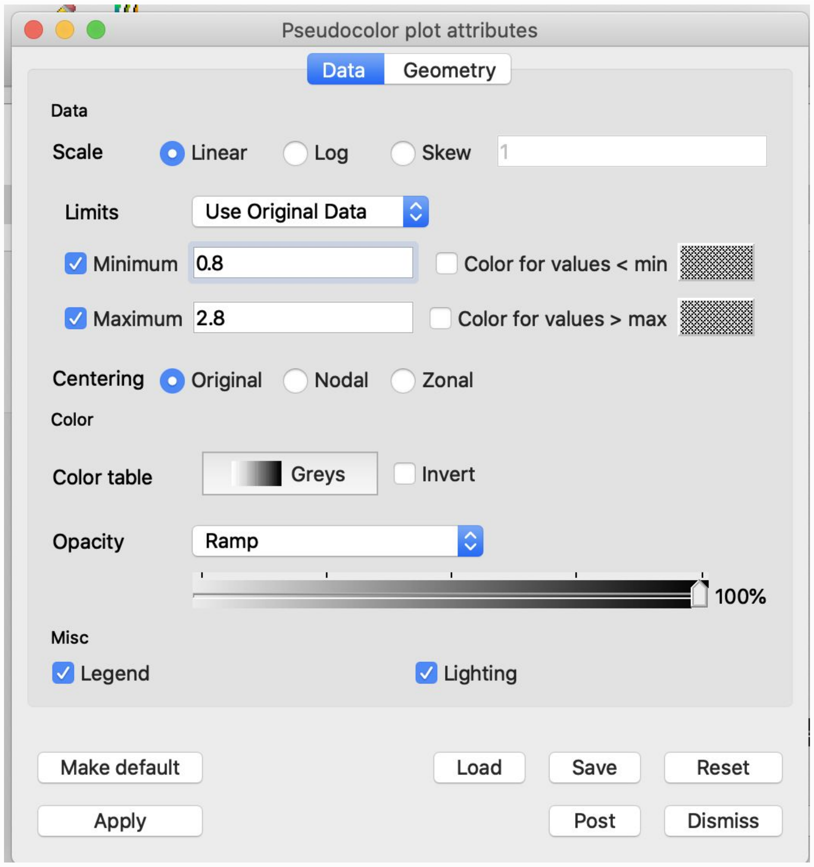
In addition to the mesh topology, this vtk dataset provides mesh fields:

- A scalar density field “rho”
- A scalar field “pressure”
- A vector field “velocity” Vx and Vy

- A vector field “magnetic field” Bx and By

We will use Pseudocolor plots to examine “rho”, the scalar density.

1. Go to *Add->Pseudocolor->rho*.
2. Click *Draw*.
3. Double click on the Pseudocolor plot to bring up the Pseudocolor plot attributes window.
4. Change the Data tab settings as follows. Note: on some displays, especially at low resolution, the Ramp setting may not show up. Increase the resolution.
5. Click *Apply*.
6. Click *Draw*.
7. Click *Play* in the *Time* animation controls.


Pseudocolor plot attributes

Data | Geometry

Data

Scale [✓ Linear] Log Skew 1

Limits: Use Original Data

[✓] Minimum 0.8 — [ ] Color for values < min

[✓] Maximum 2.8 — [ ] Color for values > max

Centering [✓ Original] Nodal Zonal

Color

Color table: Greys — [ ] Invert

Opacity: Ramp — 100%

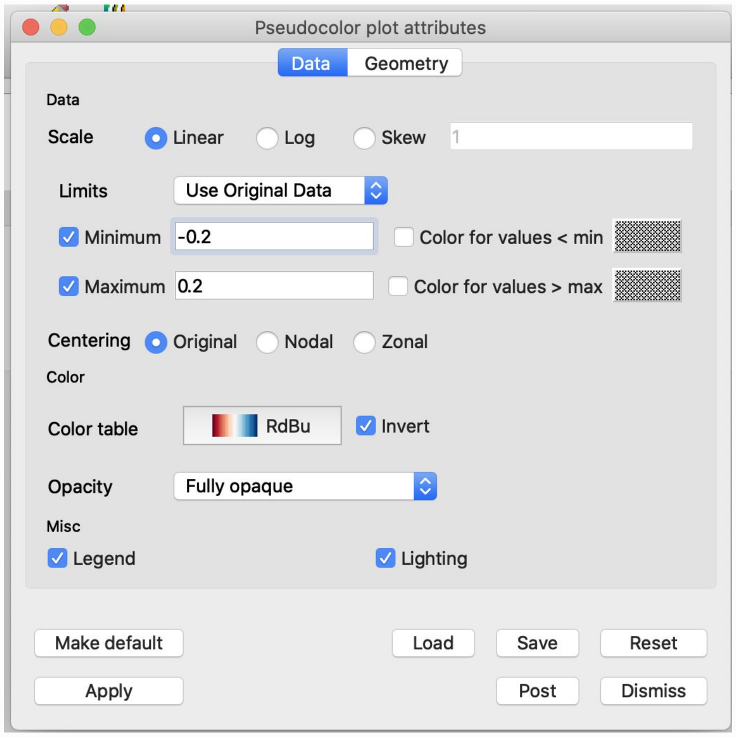
Misc

[✓] Legend — [✓] Lighting

You will see the density field animate in Grayscale. We will overplot the velocity in color. Hit the Stop button. Roll back to the first frame 0000. We will use Pseudocolor plots to examine “vx1”, the x-component

1. Go to *Add->Pseudocolor->vx1*
2. Double click on the Pseudocolor plot to bring up the Pseudocolor plot attributes window.
3. Change the Data tab settings as follows.
4. Click *Apply*.
5. Click *Draw*.

6. Click *Play*


Pseudocolor plot attributes

Data | Geometry

Data

Scale: [✓ Linear] Log Skew 1

Limits: Use Original Data

✓ Minimum -0.2 ☐ Color for values < min

✓ Maximum 0.2 ☐ Color for values > max

Centering: [✓ Original] Nodal Zonal

Color

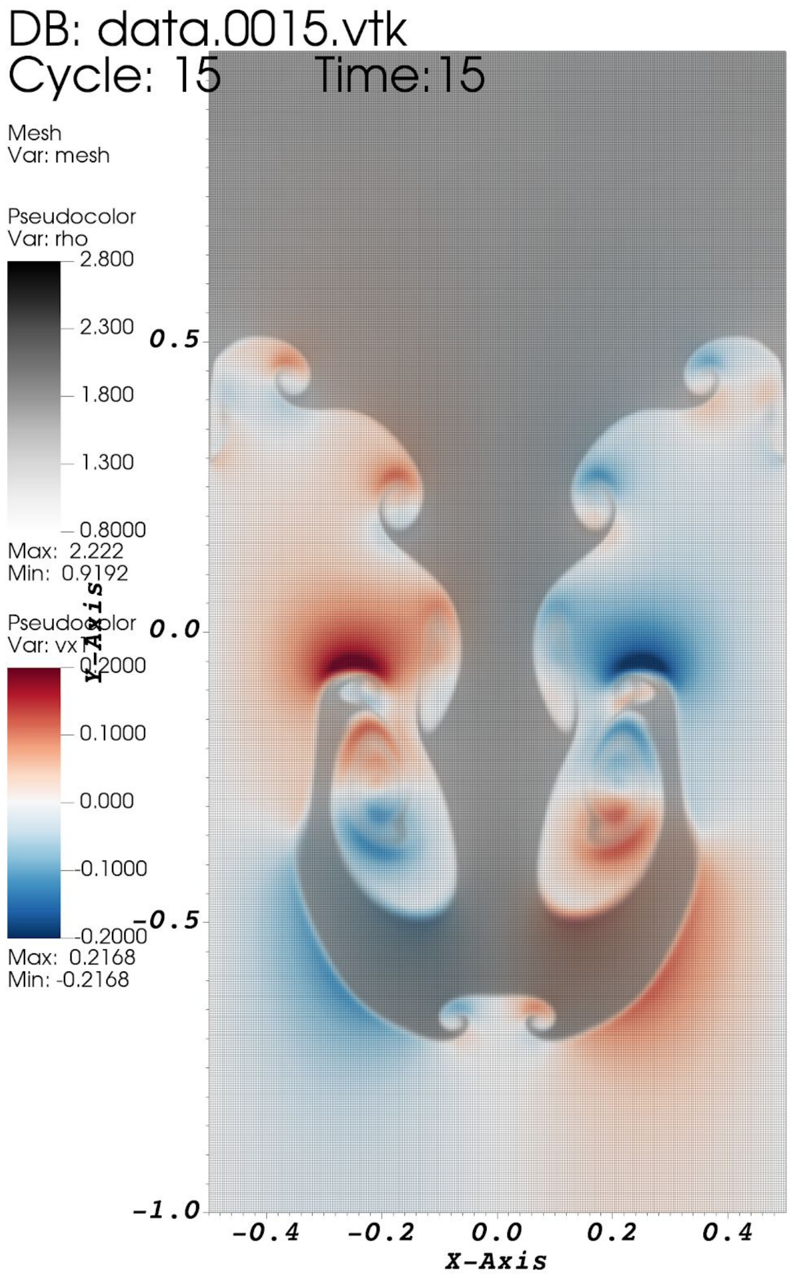
Color table: RdBu ✓ Invert

Opacity: Fully opaque

Misc

✓ Legend ✓ Lighting

The inverted red-blue color table is a good choice for velocity in astronomy: positive velocity is red, negative velocity is blue. Note that this is the x-component of the velocity in the horizontal direction. The horizontal velocity is very small at first and grows as a result of the instability.

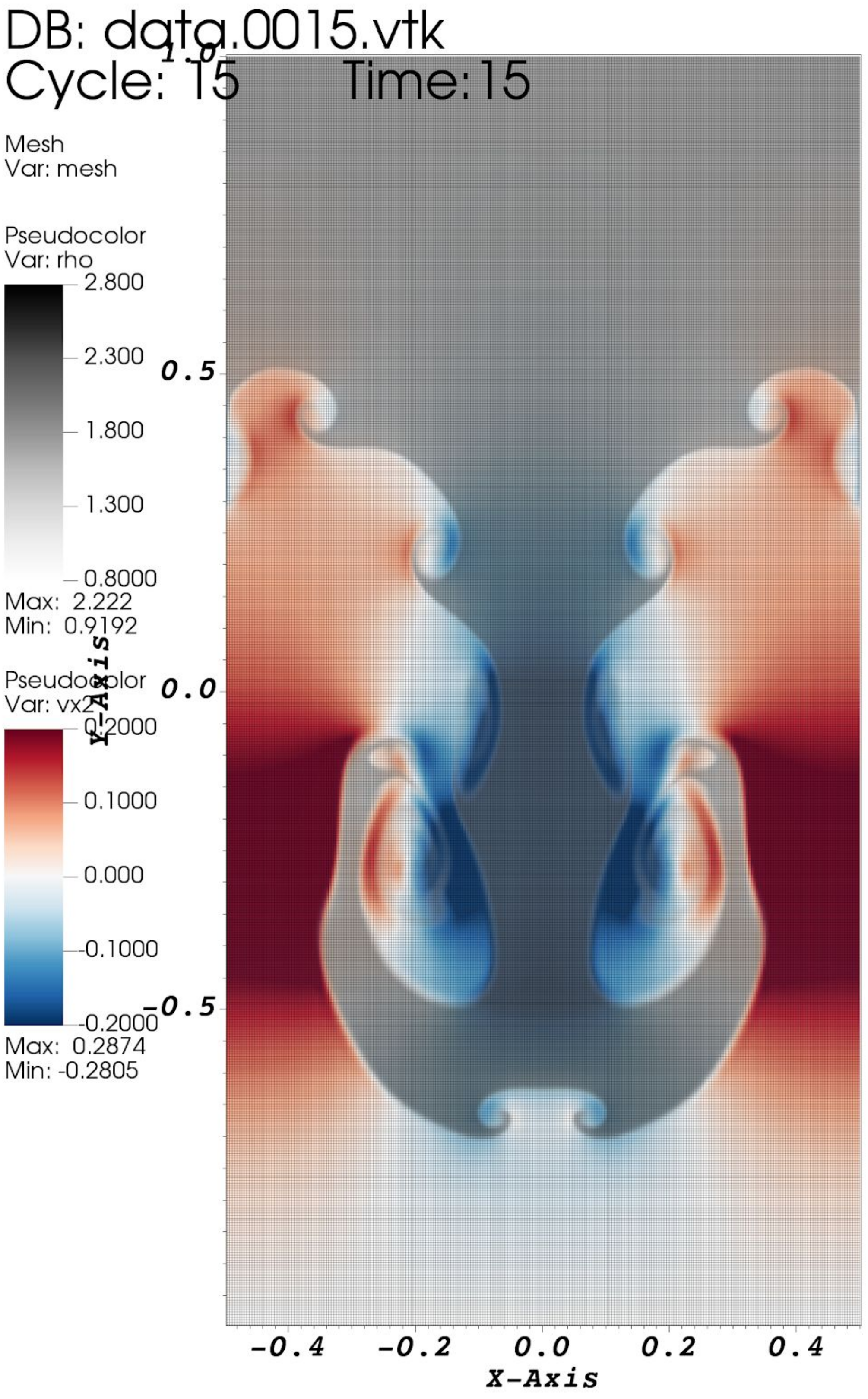


We will now overplot vx2, the y-component velocity in the same color. Hit the Stop button. Roll back to the first frame 0000. Click on Pseudocolor - vx1, then click Hide/Show. This will hide the vx1 plot.

1. Go to *Add->Pseudocolor->vx2*
2. Modify the Pseudocolor plot attributes as before.

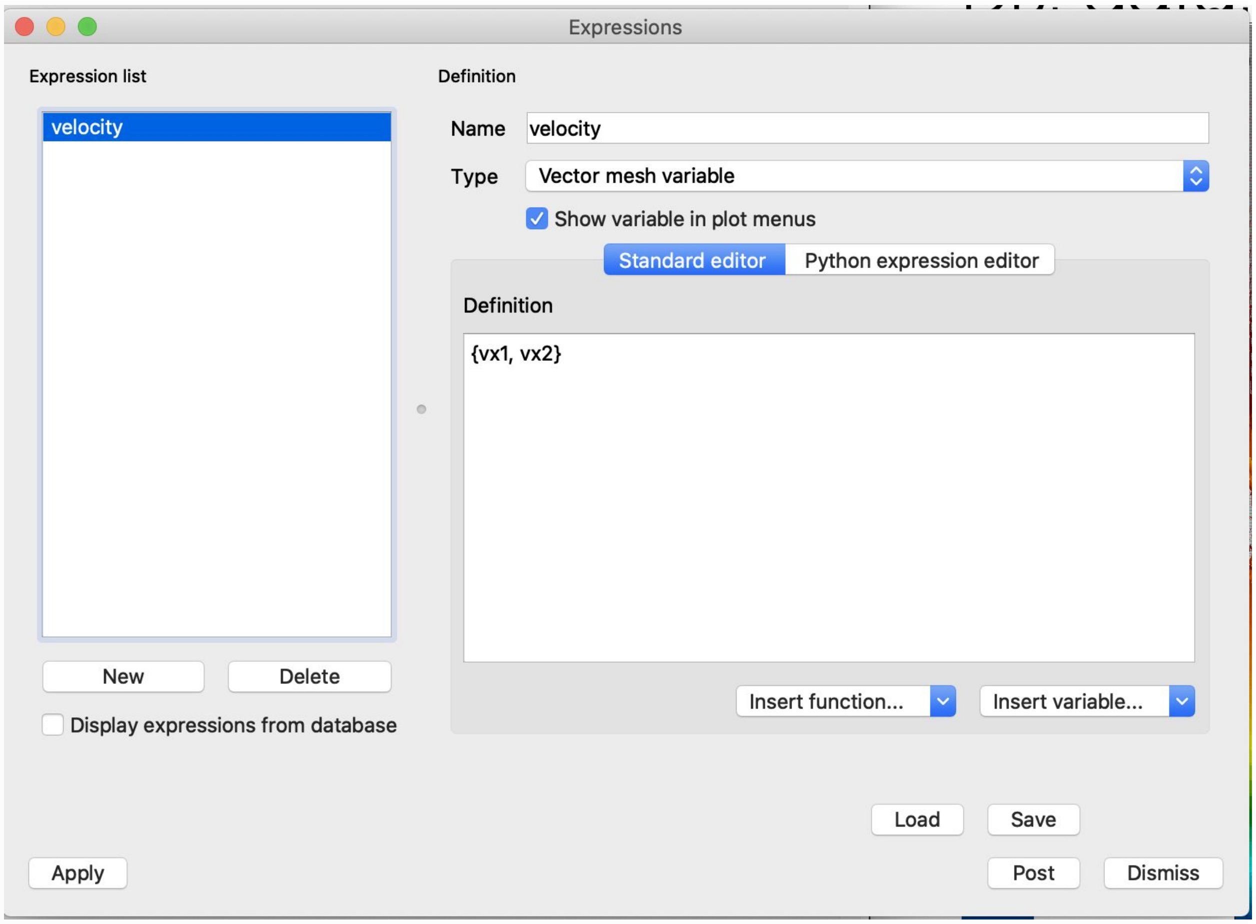
3. Click *Apply*.
4. Click *Draw*.
5. Click *Play*

Notice the vertical velocity shear (contrasting red and blue regions) where the Kelvin-Helmholtz instabilities arise.


DB: data.0015.vtk
Cycle: 15 Time:15
Mesh
Var: mesh
Pseudocolor
Var: rho
2.800
2.300
1.800
1.300
0.8000
Max: 2.222
Min: 0.9192
Pseudocolor
Var: vx2
0.2000
0.1000
0.000
-0.1000
-0.2000
Max: 0.2874
Min: -0.2805
1.0
0.5
0.0
-0.5
Y-Axis
-0.4
-0.2
0.0
0.2
0.4
X-Axis

The vtk files generated by PLUTO have two scalars vx1 and vx2, rather than a velocity vector. We want to exploit Visit's vector functionality, so we will create a vector.

1. Go to *Controls>Expressions*
2. Click *New*, change the name to velocity
3. In *Definition*, type: {vx1, vx2}
4. Click *Apply* and *Dismiss*



velocity

Name: velocity

Type: Vector mesh variable

☑ Show variable in plot menus

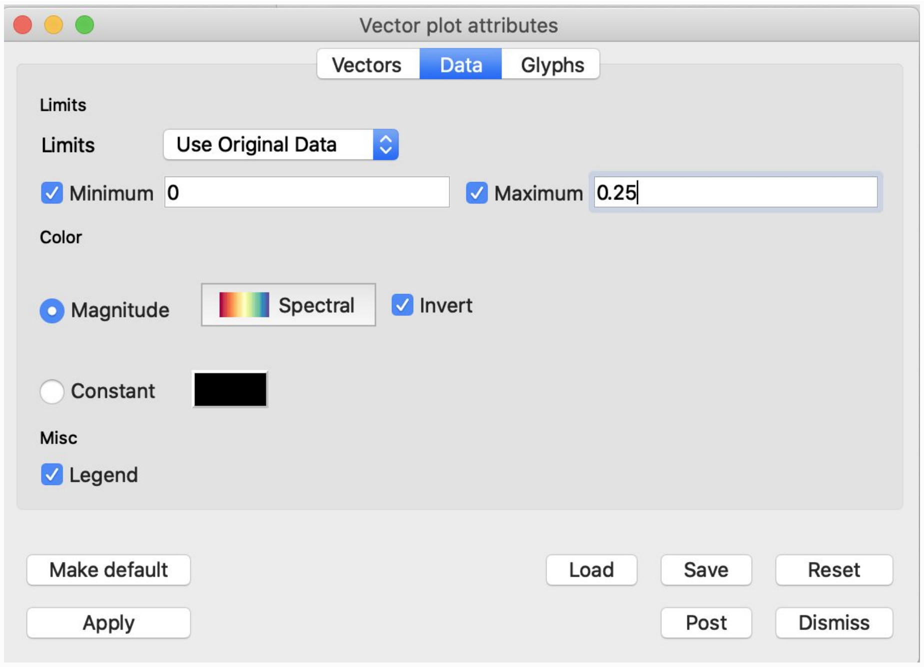
Definition

```
{vx1, vx2}
```

☐ Display expressions from database

1. Make sure your vx1 and vx2 Pseudocolor plots are hidden.

2. Go to *Add->Vector->velocity*.
3. Open the Vector plot attributes window.
4. Go to the *Data* tab
5. In the Limits section, check Minimum 0, check Maximum 0.25
6. In the *Color* section, change the *Magnitude* to *Spectral*, check the *Invert* option.
7. Go to the *Glyphs* tab.
8. In the *Scale* section, set the *Scale* to “0.5”.
9. In the *Style* section, set *Arrow body* to *Cylinder*.
10. In the *Rendering* section, set *Geometry Quality* to *High*.
11. Click *Apply* and *Dismiss*.
12. Click *Draw*.
13. Click *Play*.


Vector plot attributes

Limits

Limits: Use Original Data

[✓] Minimum 0   [✓] Maximum 0.25

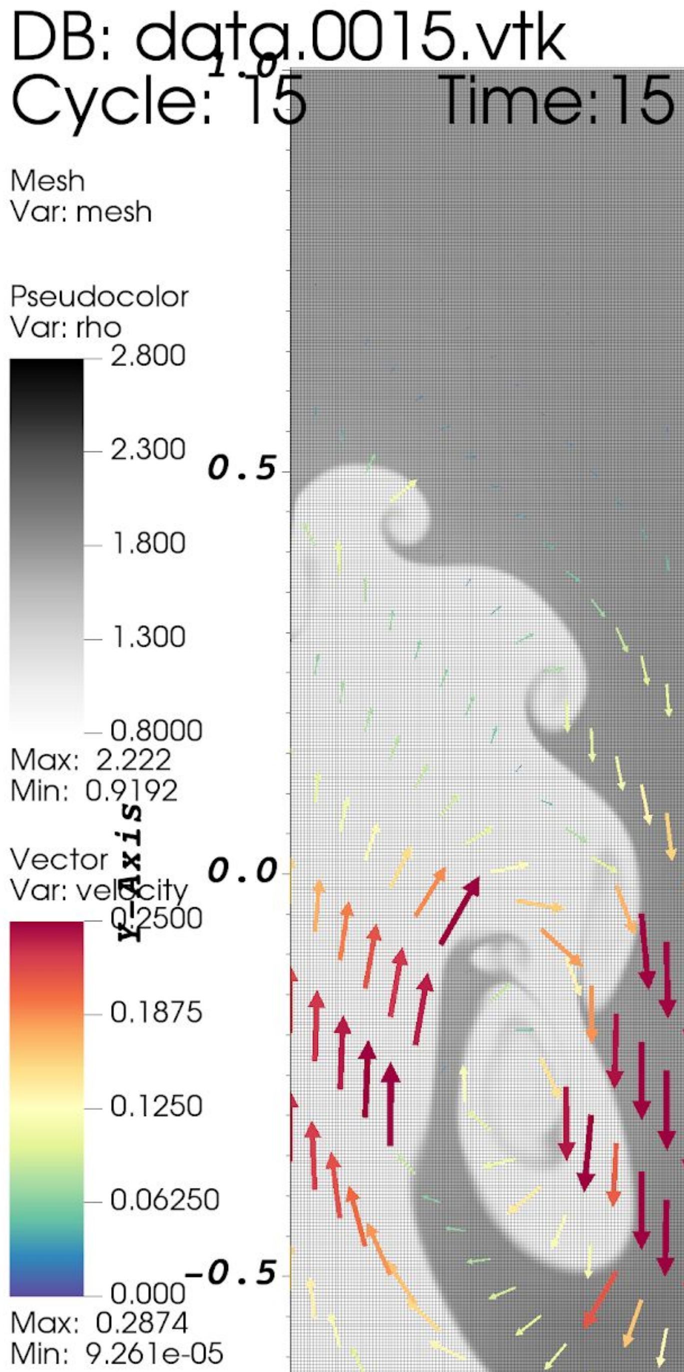
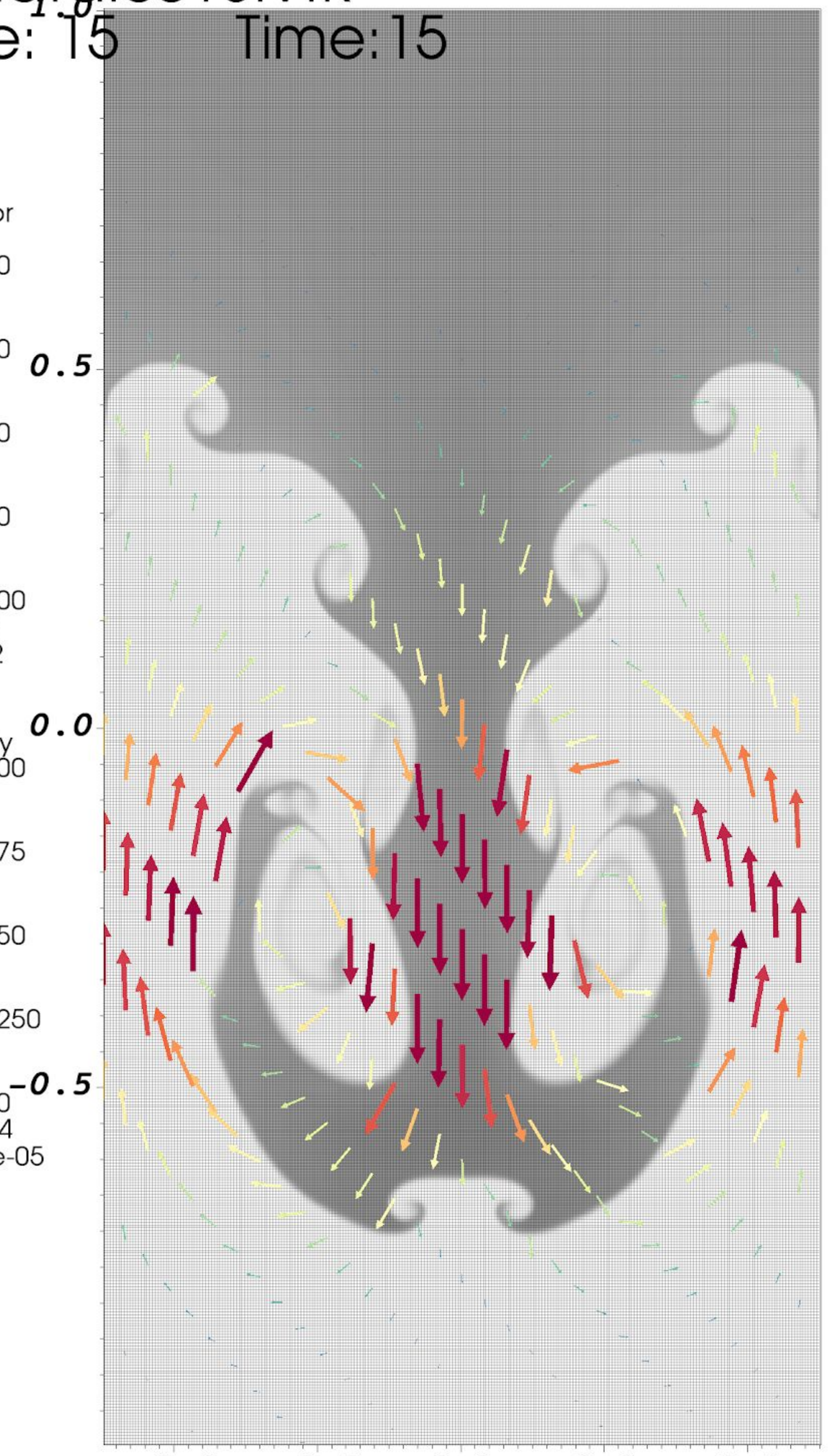
Color

[● Magnitude] Spectral [✓] Invert

Constant

Misc

[✓] Legend



DB: data.0015.vtk
Cycle: 15 Time:15
Mesh
Var: mesh
Pseudocolor
Var: rho
2.800
2.300
1.800
1.300
0.8000
Max: 2.222
Min: 0.9192
Vector
Var: velocity
0.2500
0.1875
0.1250
0.06250
0.000
Max: 0.2874
Min: 9.261e-05
1.0
0.5
0.0
-0.5
Y-Axis
-0.4
-0.2
0.0
0.2
0.4
X-Axis

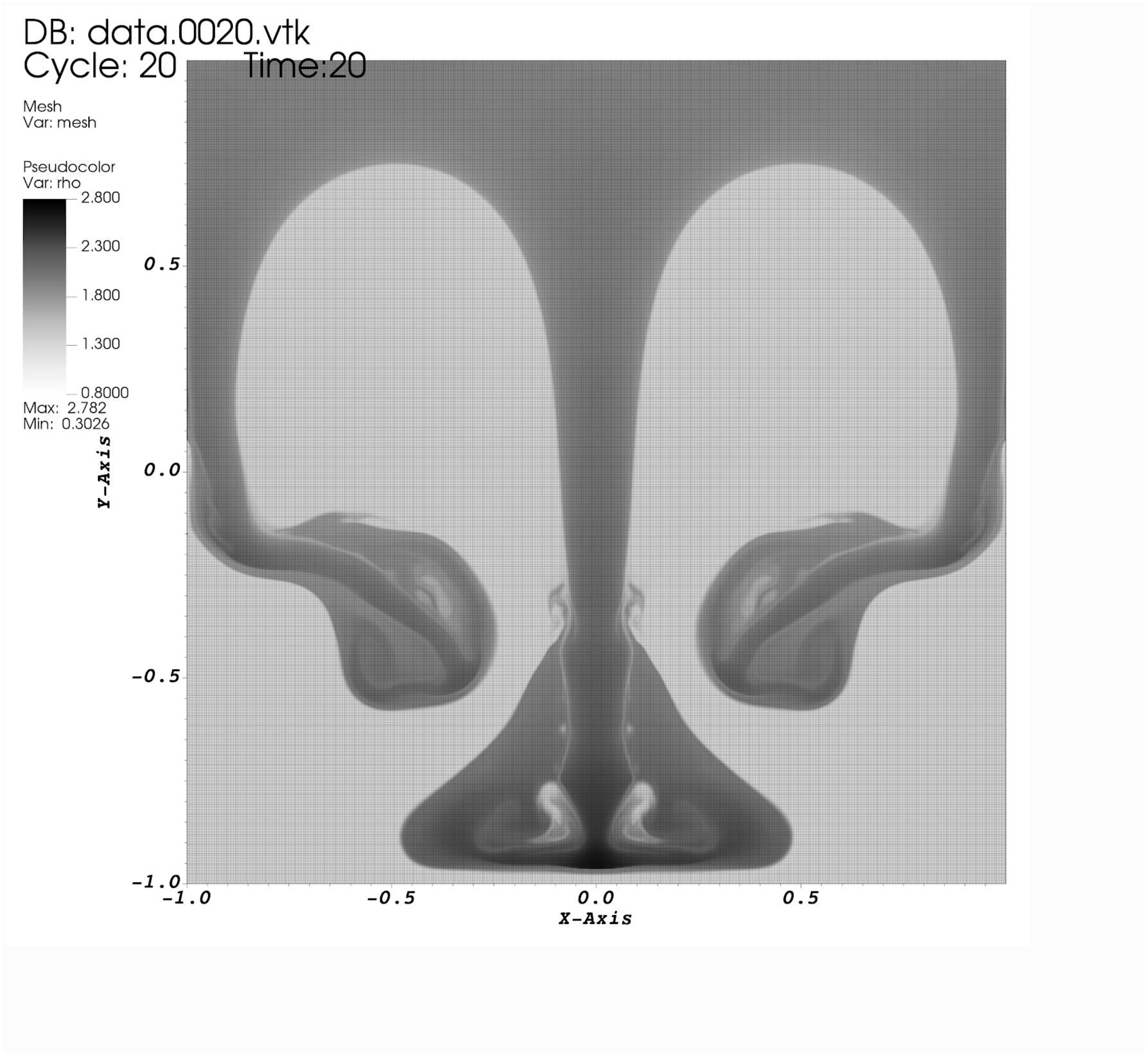
11.4.2 Visualize and analyze the run05 dataset

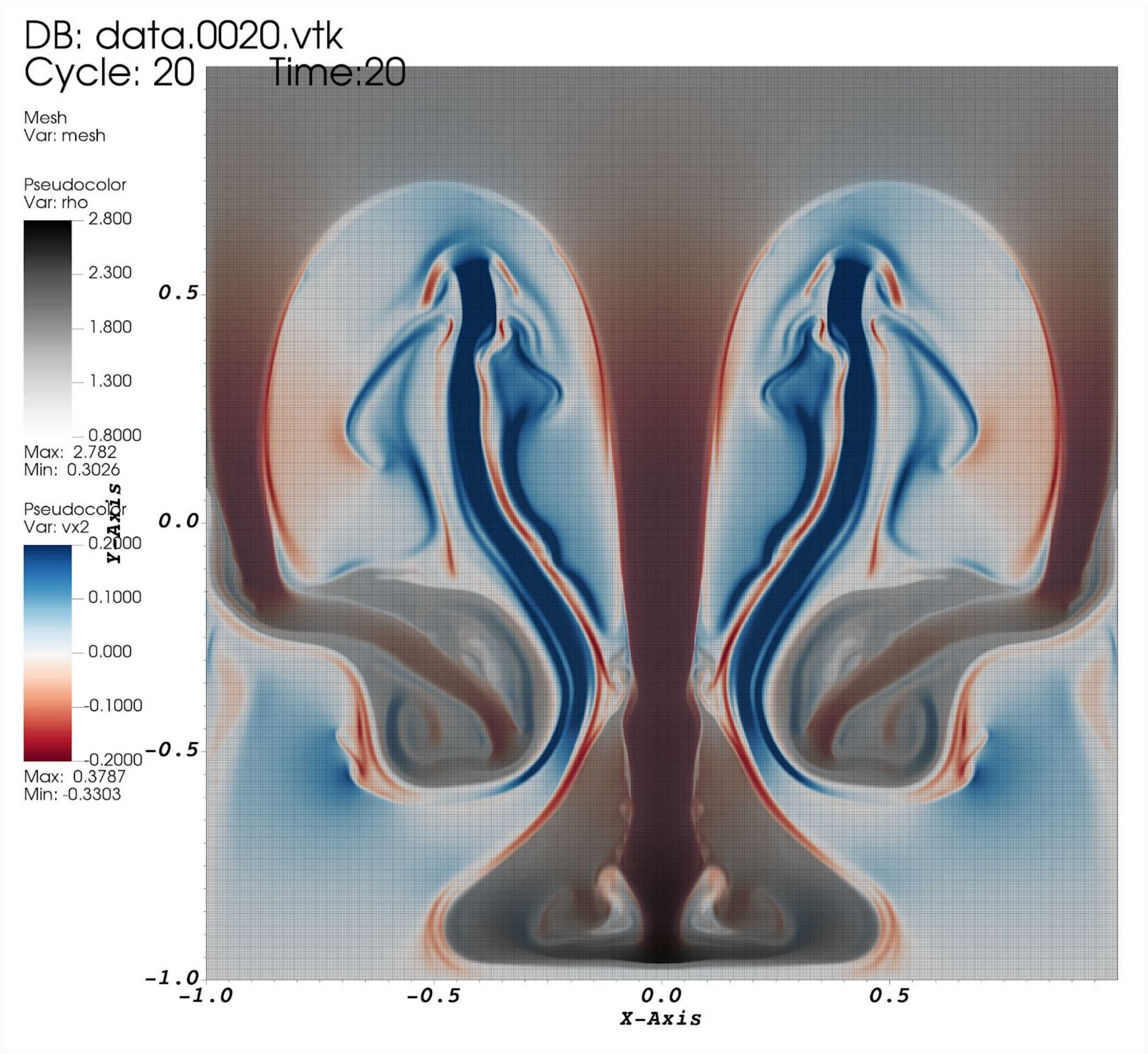
At this point, we are done with run01. You can save your animations in File>Save Movie… if you wish, but this is beyond the scope of this activity.

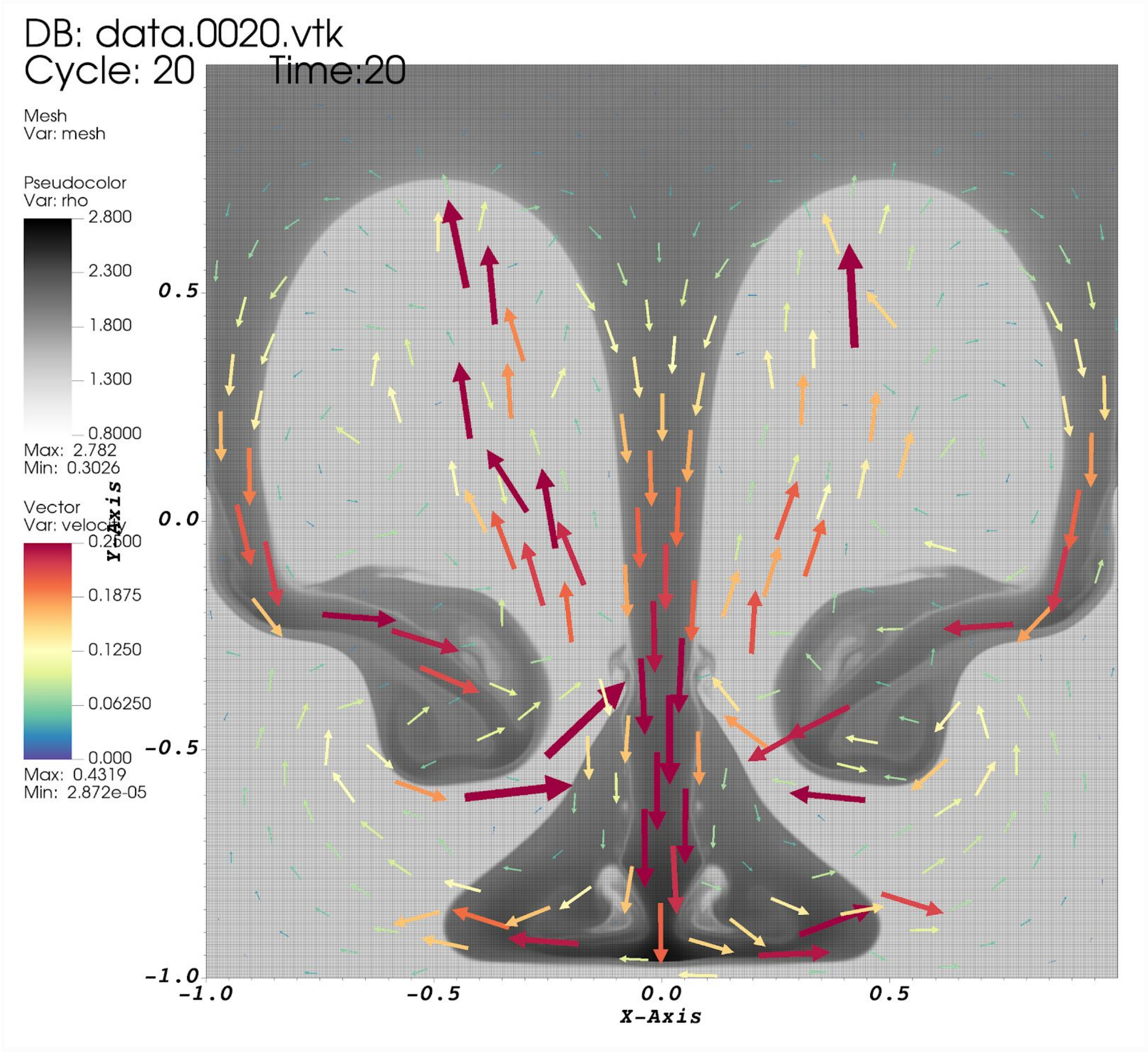
To save memory and speed up execution, we will close our plots, and close our run01 data. We will then open our run05 data.

1. Select the *Vector - velocity* plot, click *Delete*
2. Repeat for all the plots.
3. Near the top of the main window under Sources, click *Close*.
4. Under Sources, click *Open*.
5. Navigate your file browser to run05, highlight all the .vtk files, click OK.

You may now repeat the activity for the run05 data. Here's the output you should produce. Notice that (1) the x-axis now runs from -1 to 1, (2) the reflective boundary conditions at x=-1, x=1 and y=-1 act to contain the fluid, pushing it back into the volume. At the end of the activity we will define and plot the magnetic field vectors.


DB: data.0020.vtk
Cycle: 20 Time:20
Mesh
Var: mesh
Pseudocolor
Var: rho
2.800
2.300
1.800
1.300
0.8000
Max: 2.782
Min: 0.3026
0.5
0.0
-0.5
-1.0
Y-Axis
-1.0
-0.5
0.0
0.5
X-Axis


DB: data.0020.vtk
Cycle: 20 Time:20
Mesh
Var: mesh
Pseudocolor
Var: rho
2.800
2.300
1.800
1.300
0.8000
Max: 2.782
Min: 0.3026
Pseudocolor
Var: vx2
0.2000
0.1000
0.000
-0.1000
-0.2000
Max: 0.3787
Min: -0.3303
0.5
0.0
-0.5
-1.0
-1.0
-0.5
0.0
0.5
Y-Axis
X-Axis

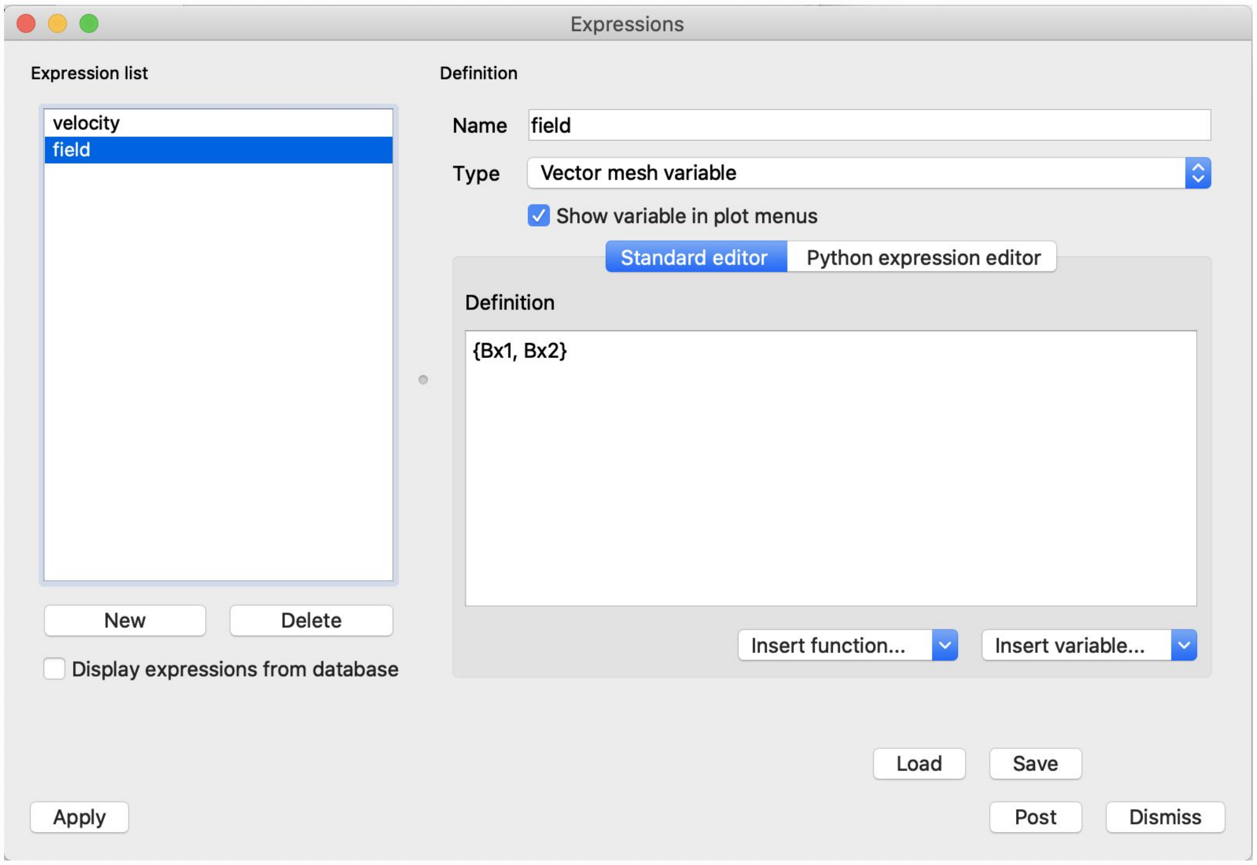


11.4.2.2 Examine the magnetic field

As we did earlier with the velocity vector, we want to create a magnetic field vector.

1. Go to *Controls>Expressions*
2. Click *New*, change the name to *field*
3. In *Definition*, type: {vx1, vx2}

4. Click *Apply* and *Dismiss*.



velocity
field

Name field

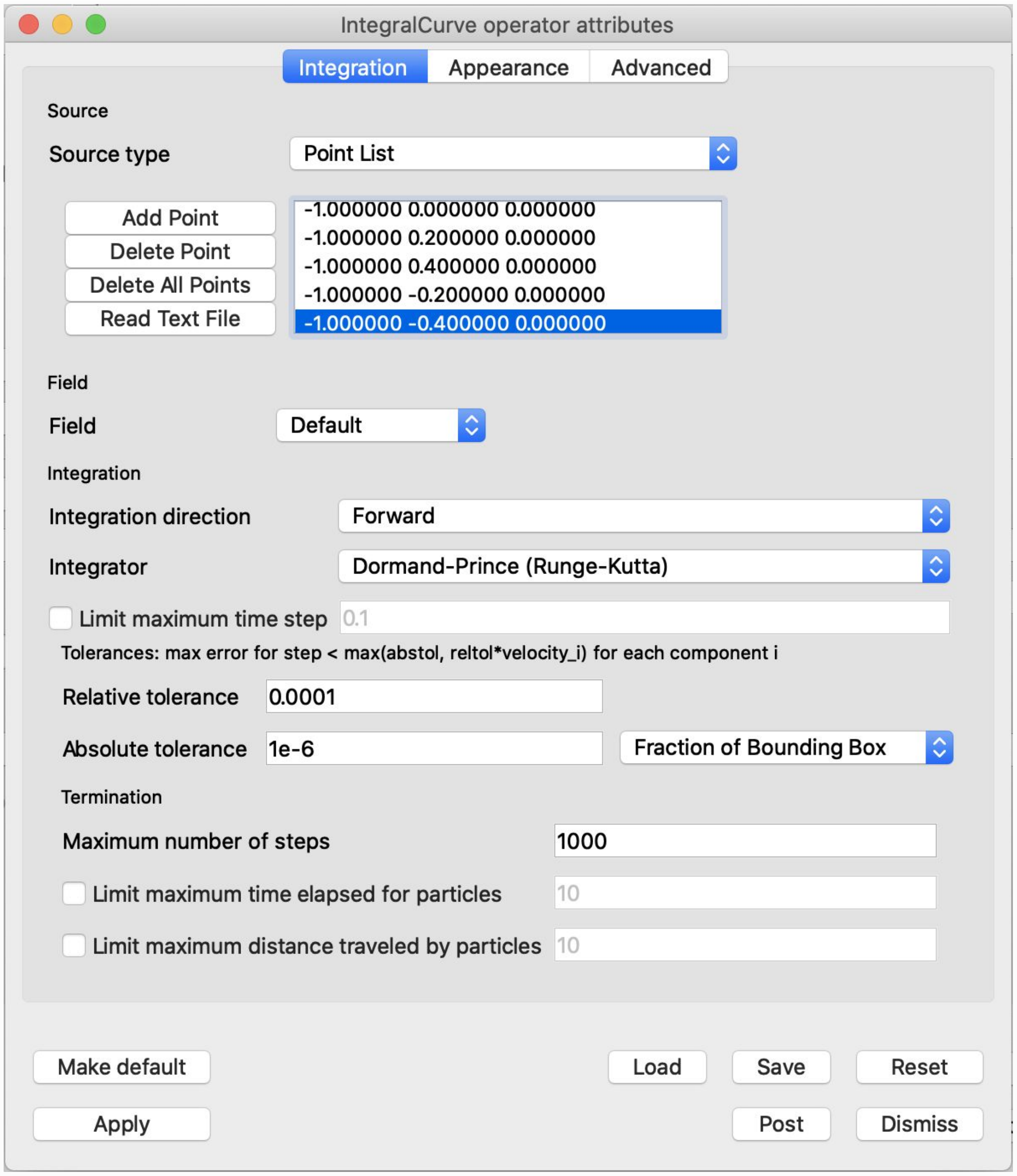
Type Vector mesh variable

✓ Show variable in plot menus

Definition

{Bx1, Bx2}

1. Go to *Add->Pseudocolor->operators->IntegralCurve->field*
2. Open the IntegralCurve operator attributes window.
3. In the *Integration* tab, under *Source*, select *Source type - Point List*
4. You will add points as needed. Each point is at x=-1 and spans a range of y: -0.4, -0.2, 0.0, 0.2 and 0.4 (see image below).
5. Click *Apply* and *Dismiss*.
6. Click *Play*. Advance to the last time frame (20).


IntegralCurve operator attributes

Source

Source type: Point List

```
-1.000000 0.000000 0.000000
-1.000000 0.200000 0.000000
-1.000000 0.400000 0.000000
-1.000000 -0.200000 0.000000
-1.000000 -0.400000 0.000000
```

Field

Field: Default

Integration

Integration direction: Forward

Integrator: Dormand-Prince (Runge-Kutta)

☐ Limit maximum time step: 0.1

Tolerances: max error for step < max(abstol, reltol*velocity_i) for each component i

Relative tolerance: 0.0001

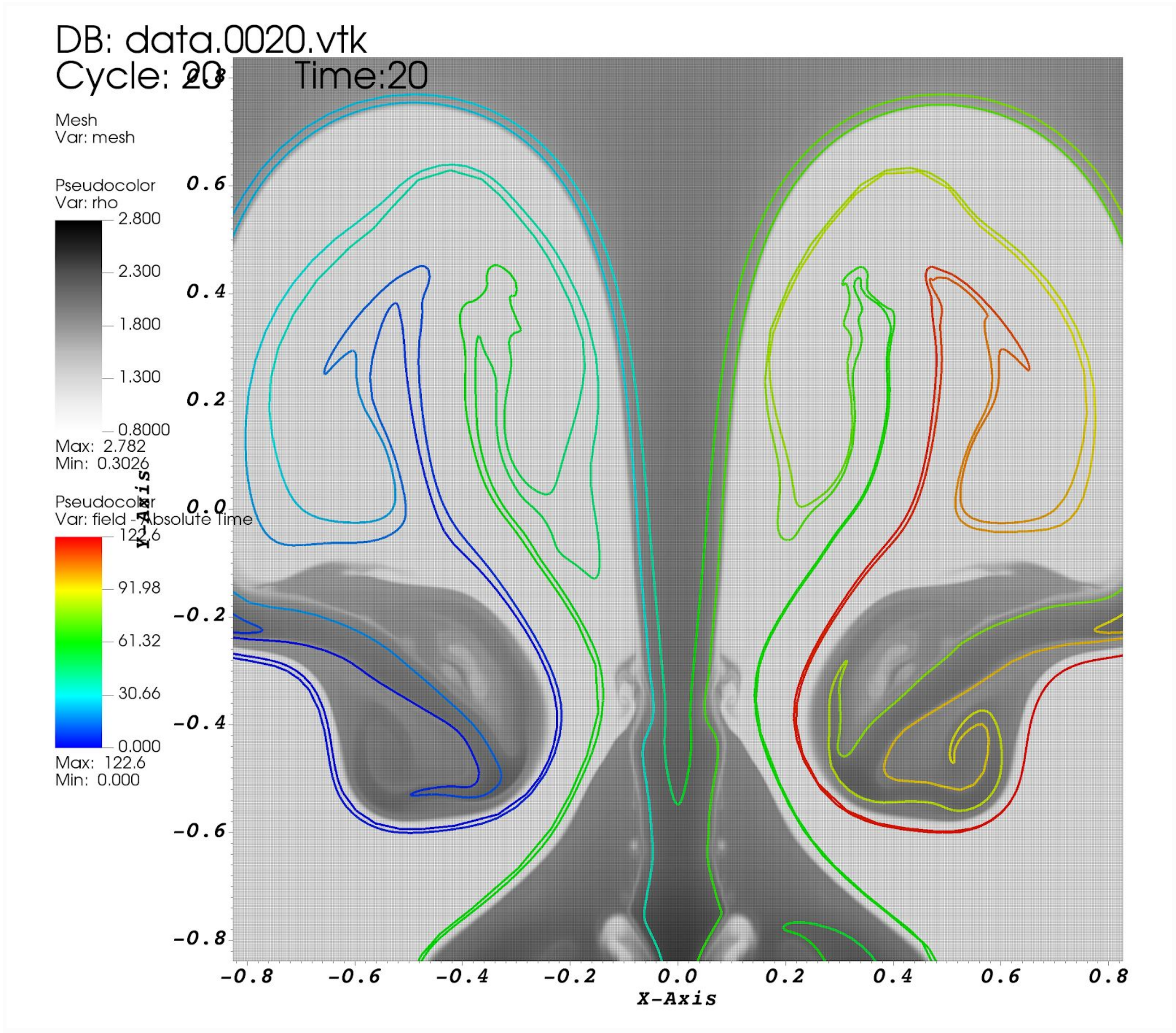
Absolute tolerance: 1e-6 Fraction of Bounding Box

Termination

Maximum number of steps: 1000

☐ Limit maximum time elapsed for particles: 10

☐ Limit maximum distance traveled by particles: 10



The field lines start out horizontally threading the fluid. As the simulation evolves, the field is pulled along by the fluid as they fall and rise. Buoyant lower density fluid (light gray) is pushed up, denser fluid (dark gray) falls, dragging the magnetic field lines.