Post-Processing IBAMR Simulation Data in VisIt on a Computing Cluster

A Tutorial

Gabrielle Hobson
Scripps Institution for Oceanography at UCSD

A contribution to the IBAMR Tutorials Research Group

9/23/2020

Contents

- Installing VisIt on the cluster
- Creating a Python script to visualize data
- Running your script on the cluster
- IBFE meshes and displacement operators
- Resampling velocity fields in 3D
- Accessing a .VTK database
- Using .VTK databases – averaging velocity
- Restarting a job
- Things to remember
- Useful resources

Using VisIt on the cluster: advantages

- Speed! The post-processing time is much faster
- Your simulation data is already on the cluster, so no time spent downloading it
- Fewer human error issues when everything is automated
- Scripts make it easier to document and share your analysis

Installing VisIt on the cluster

Log on to the cluster and create a directory for the install. I used my home directory.

```
cd ~
mkdir visit
cd visit
```

Go to the VisIt website:

https://wci.llnl.gov/simulation/computer-codes/visit/executables

Download the last option in the list, which looks like:

Linux - x86_64 64 bit Redhat Enterprise Linux 7.5, 4.18.9-1.el7.elrepo.x86_64 #1 SMP, gcc 4.8.5 *(Includes Mesa support for rendering without a display. Only use on servers without a display.)*	Download

Installing VisIt on the cluster (continued)

Upload the file to the directory you created (using the command line or WinSCP, Filezilla, etc.)

```
scp visit3_1_2.linux-x86_64-rhel7-wmesa.tar.gz gmhobson@dogwood.unc.edu:~/visit/
```

Extract the contents:

```
tar -xvf visit3_1_2.linux-x86_64-rhel7-wmesa.tar.gz
```

This will create a directory called

```
visit3_1_2.linux-x86_64
```

The syntax to use VisIt to run a Python script called 'script.py' is as follows.

You'll need to change the line so that it contains the path to your visit directory.

```
/nas/longleaf/home/gmhobson/visit/visit3_1_2.linux-x86_64/bin/visit -cli -nowin -s
script.py
```

Creating Python scripts to automate analysis

- This part does involve using the GUI on your computer
- Have a whitespace – sensitive text editor at the ready (Notepad ++, Visual Studio Code, etc.)
- We create the Python scripts by recording the commands we are using in VisIt while we use them

Recording commands

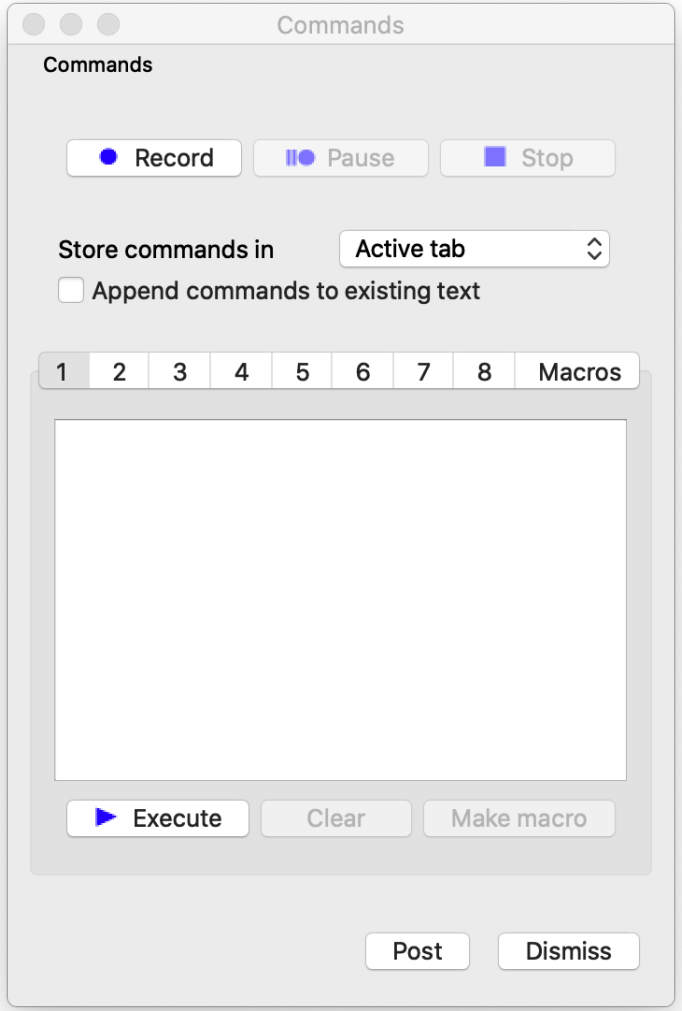
Click Controls → Command, or use the keyboard shortcut Control→ C (on Windows) or Command → Shift → C (on Mac).

Click Record to begin recording your commands.

Once you've completed the steps you want, click Stop.

Copy → Paste the code that appears in the box into your text editor.

You can follow these steps to create a script for the entire visualization process.



An Example

Throughout the next steps I will use the 2D rubber band example that is available on Laura's GitHub:

https://github.com/fairyflies9/IBAMR-Tutorials/tree/master/Examples/3-Example_2Drubber_band

The start of your script: opening databases

Opening a database that contains your data is the first step.

Here, I open the dumps.visit data from the rubber band simulation:

```
OpenDatabase("localhost:/nas/longleaf/home/gmhobson/code/rubber_band/viz_IB2d/dumps.visit")
```

However, it's a good idea to define the locations as strings at the start of your script.

```
string_dumps = "/21dayscratch/scr/g/m/gmhobson/flow_fields/rubber_band/dumps.visit"

string_mesh = "localhost/nas/longleaf/home/gmhobson/code/rubber_band/viz_IB2d/lag_data.visit”
```

Now I can easily refer to the locations of my dumps.visit data and my moving mesh data.

```
OpenDatabase(string_dumps)
OpenDatabase(string_mesh)
```

You can also specify a string for where you want to store the output (and refer to it later).

```
string_output = /21dayscratch/scr/g/m/gmhobson/flow_fields/rubber_band"
```

So far, the script looks like:

Adding in a database correlation and hiding the grid that shows up gives us this script.

```
# Example Script for IBAMR Tutorials

string_dumps =
"localhost:/nas/longleaf/home/gmhobson/code/rubber_band/viz_IB2d/dumps.visit"
string_mesh =
"localhost:/nas/longleaf/home/gmhobson/code/rubber_band/viz_IB2d/lag_data.visit"
string_output = "/21dayscratch/scr/g/m/gmhobson/flow_fields/rubber_band"

OpenDatabase(string_dumps)
OpenDatabase(string_mesh)

CreateDatabaseCorrelation("Correlation02",(string_dumps, string_mesh), 0)

SetActivePlots(0)
HideActivePlots()
```

Now for the fun part: visualizing data

As an example, here I plot the mesh as a first step.

```
AddPlot("Mesh", "rubber_band_512_mesh", 1, 0)
DrawPlots()
SetActivePlots(1)
MeshAtts = MeshAttributes()
MeshAtts.legendFlag = 0 # no legend
MeshAtts.lineWidth = 3 # thicker line
MeshAtts.meshColor = (0, 0, 128, 255) # nice dark blue
MeshAtts.meshColorSource = MeshAtts.MeshCustom
MeshAtts.pointSize = 0.05
MeshAtts.pointType = MeshAtts.Point
MeshAtts.pointSizePixels = 6 # larger points
MeshAtts.opacity = 1
SetPlotOptions(MeshAtts)
```

This isn't every line that would show up when recording – just the ones I'm interested in

Visualizing data, continued

```
ActivateDatabase(string_dumps)
AddPlot("Pseudocolor", "U_magnitude", 1, 0)
PseudocolorAtts = PseudocolorAttributes()
PseudocolorAtts.minFlag = 1
PseudocolorAtts.min = 0
PseudocolorAtts.maxFlag = 1
PseudocolorAtts.max = 0.05
PseudocolorAtts.centering = PseudocolorAtts.Nodal
PseudocolorAtts.colorTableName = "plasma"
PseudocolorAtts.invertColorTable = 0
PseudocolorAtts.opacityType = PseudocolorAtts.FullyOpaque
PseudocolorAtts.legendFlag = 1
PseudocolorAtts.lightingFlag = 1
SetPlotOptions(PseudocolorAtts)
DrawPlots()
```

Now for some color! I'm going to make a velocity pseudocolor plot. In 3d, this would be done on a slice.

Visualizing data, continued

Here I remove the axes, and the user and time data. Then I zoom in to the desired view.
Note: commands to do with "spontaneous states", like when you click the zoom icon in the GUI, do not work on the cluster.

```
AnnotationAtts = AnnotationAttributes()
AnnotationAtts.axes2D.visible = 0
AnnotationAtts.userInfoFlag = 0
AnnotationAtts.databaseInfoFlag = 0
AnnotationAtts.timeInfoFlag = 0
SetAnnotationAttributes(AnnotationAtts)

View2DAtts = View2DAttributes()
View2DAtts.windowCoords = (-0.185258, 0.188338, -0.15681, 0.150774)
View2DAtts.viewportCoords = (0.2, 0.95, 0.15, 0.95)
View2DAtts.windowValid = 1
SetView2D(View2DAtts)
```

Visualizing data, continued

Now to save each time step.

```
# saving images
SetTimeSliderState(0)
k=0 #Starting number: if left as 0, the first timestep to be saved will be 1
N=29 #Ending number: set this to the last timestep you wish to save, minus 1
while k<N+1:
    TimeSliderNextState()
    SaveWindowAtts = SaveWindowAttributes()
    SaveWindowAtts.outputToCurrentDirectory = 0
    SaveWindowAtts.outputDirectory = string_output
    SaveWindowAtts.fileName = "rubber_band"
    SaveWindowAtts.family = 1
    SaveWindowAtts.format = SaveWindowAtts.PNG
    SaveWindowAtts.width = 1024
    SaveWindowAtts.height = 1024
    SetSaveWindowAttributes(SaveWindowAtts)
    SaveWindow()
    k+=1
```

Running the script

When you're ready, you can upload your script to the cluster and get ready to run it.

The syntax to run the script is in the form of:

```
path_to_VisIt_bin –cli –nowin –s name_of_script.py
```

In reality, if you're executing this in the folder that contains your script, this looks like:

```
/nas/longleaf/home/gmhobson/visit/visit3_1_2.linux-x86_64/bin/visit -cli
-nowin -s example_script.py
```

Once you've set your script running, you can check your output folder and make sure everything is coming in nicely.

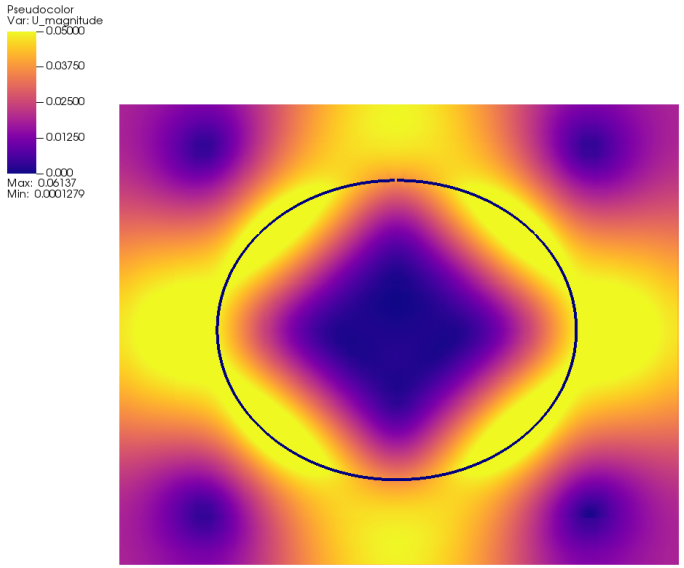
Running the script

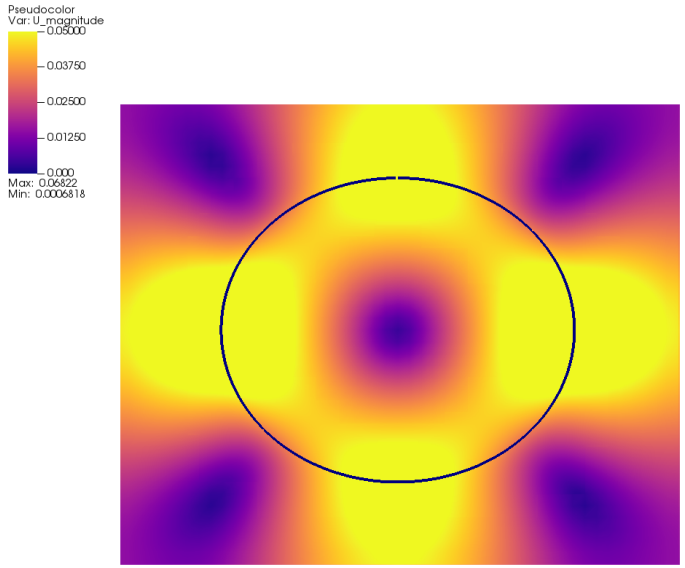
The output in your terminal will look something like:

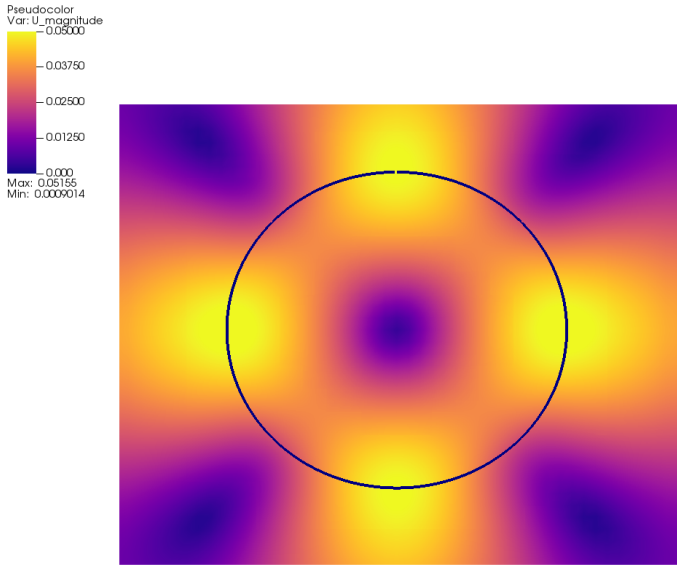
```
>>> [gmhobson@dogwood-login2 Scripts]$ Running: cli3.1.2 -nowin -s example_script.py
Running: viewer3.1.2 -nowin -noint -host 127.0.0.1 -port 5600
Running: mdserver3.1.2 -host 127.0.0.1 -port 5600
Error opening plugin file: /nas/longleaf/home/gmhobson/visit/visit3_1_2.linux-x86_64/3.1.2/linux-x86_64/plugins/d
atabases/libMADIOS2Database.so (libfabric.so.1: cannot open shared object file: No such file or directory)
Running: engine_ser3.1.2 -host 127.0.0.1 -port 5600
VisIt: Message - Rendering window 1...
VisIt: Message - Saving window 1...
VisIt: Message - Saved /21dayscratch/scr/g/m/gmhobson/flow_fields/rubber_band/rubber_band0000.png
VisIt: Message - Rendering window 1...
VisIt: Message - Saving window 1...
VisIt: Message - Saved /21dayscratch/scr/g/m/gmhobson/flow_fields/rubber_band/rubber_band0001.png
VisIt: Message - Rendering window 1...
VisIt: Message - Saving window 1...
VisIt: Message - Saved /21dayscratch/scr/g/m/gmhobson/flow_fields/rubber_band/rubber_band0002.png
VisIt: Message - Rendering window 1...
VisIt: Message - Saving window 1...
VisIt: Message - Saved /21dayscratch/scr/g/m/gmhobson/flow_fields/rubber_band/rubber_band0003.png
VisIt: Message - Rendering window 1...
VisIt: Message - Saving window 1...
```

Running the script

The result in your output folder should be .png files that look like:







IBFE Meshes and Displacement Operators

IBFE, unlike what we were doing in IBAMR, has meshes that will require a displacement operator to get them to move.

In a previous lecture, Laura covered implementing this in the VisIt GUI, and I'm including the code here that will work in scripts on the cluster.

One thing to note is the line that sets the smoothing level: SubsetAtts.smoothingLevel = 2.
This sets the smoothing level to High and gives a better-looking result than None or Fast (0 or 1 in the script), at least for the 3d coral meshes.

IBFE Meshes and Displacement Operators, cont.

```
OpenDatabase(string_mesh)
AddPlot("Subset", "Mesh", 1, 0)
SetActivePlots(1)
AddOperator("Displace", 0)
DefineVectorExpression("dX", "{dX_0,dX_1,dX_2}")
DisplaceAtts = DisplaceAttributes()
DisplaceAtts.factor = 1
DisplaceAtts.variable = "dX"
SetOperatorOptions(DisplaceAtts, 0, 0)
SubsetAtts = SubsetAttributes()
SubsetAtts.colorType = SubsetAtts.ColorBySingleColor
SubsetAtts.singleColor = (255, 255, 255, 255)
SubsetAtts.subsetNames = ("Whole mesh (Mesh)")
SubsetAtts.smoothingLevel = 2
SubsetAtts.pointSize = 0.05
SubsetAtts.pointType = SubsetAtts.Point
SetPlotOptions(SubsetAtts)
DrawPlots()
```

Resampling velocity fields in 3D

Resampling velocity fields in 3D will give better results when plotting velocity vectors or when spatially averaging velocity components.

Resampling creates a .VTK database that you can then call in other scripts. However, on the cluster VisIt has trouble reading .VTK databases.

The workaround is to copy a dumps.visit file into the .VTK database directory.
Then edit it so that it contains a list of the names of all the .vtk files.

This dumps.visit file can be called in the OpenDatabase() command, and it opens the .VTK database.

```
OpenDatabase(string_dumps)
AddPlot("Vector", "U", 1, 0)
AddOperator("Resample", 0)
SetActivePlots(1)
SetActivePlots(1)
ResampleAtts = ResampleAttributes()
ResampleAtts.useExtents = 0
ResampleAtts.startX = -0.0375
ResampleAtts.endX = 0.0375
ResampleAtts.samplesX = 192
ResampleAtts.startY = -0.007
ResampleAtts.endY = 0.018
ResampleAtts.samplesY = 64
ResampleAtts.is3D = 1
ResampleAtts.startZ = -0.0075
ResampleAtts.endZ = 0.0075
ResampleAtts.samplesZ = 32
SetOperatorOptions(ResampleAtts, 0)
DrawPlots()
```

Resampling a 3D velocity field, cont.

It's a good idea to check that the extents of your domain in each direction divided by the number of samples gives you the same number (i.e. you have a uniform grid).

```
VectorAtts = VectorAttributes()
VectorAtts.glyphLocation = VectorAtts.UniformInSpace
VectorAtts.useStride = 0
VectorAtts.stride = 1
VectorAtts.nVectors = 393216
VectorAtts.lineStyle = VectorAtts.SOLID
VectorAtts.lineWidth = 0
VectorAtts.scale = 0.25
VectorAtts.scaleByMagnitude = 1
VectorAtts.autoScale = 1
VectorAtts.headSize = 0.25
VectorAtts.headOn = 1
VectorAtts.colorByMag = 1
VectorAtts.useLegend = 1
VectorAtts.vectorColor = (0, 0, 0, 255)
SetPlotOptions(VectorAtts)
TimeSliderNextState()
```

Resampling a 3D velocity field, cont.

The important thing here is to match the number of vectors (nVectors) to the product of the number of xSamples, ySamples, and zSamples.

```
ExportDBAtts = ExportDBAttributes()
ExportDBAtts.allTimes = 1
ExportDBAtts.db_type = "VTK"
ExportDBAtts.db_type_fullname = "VTK_1.0"
ExportDBAtts.filename = "visit_ex_db"
ExportDBAtts.dirname = string_output
ExportDBAtts.variables = ("U", "U_x", "U_y", "U_z")
ExportDBAtts.writeUsingGroups = 0
ExportDBAtts.groupSize = 48
ExportDBAtts.opts.types = (0, 0)
ExportDBAtts.opts.help = ""
ExportDatabase(ExportDBAtts)
```

Resampling a 3D velocity field, cont.

This exports your resampled velocity field at each timestep.

Accessing a .VTK database

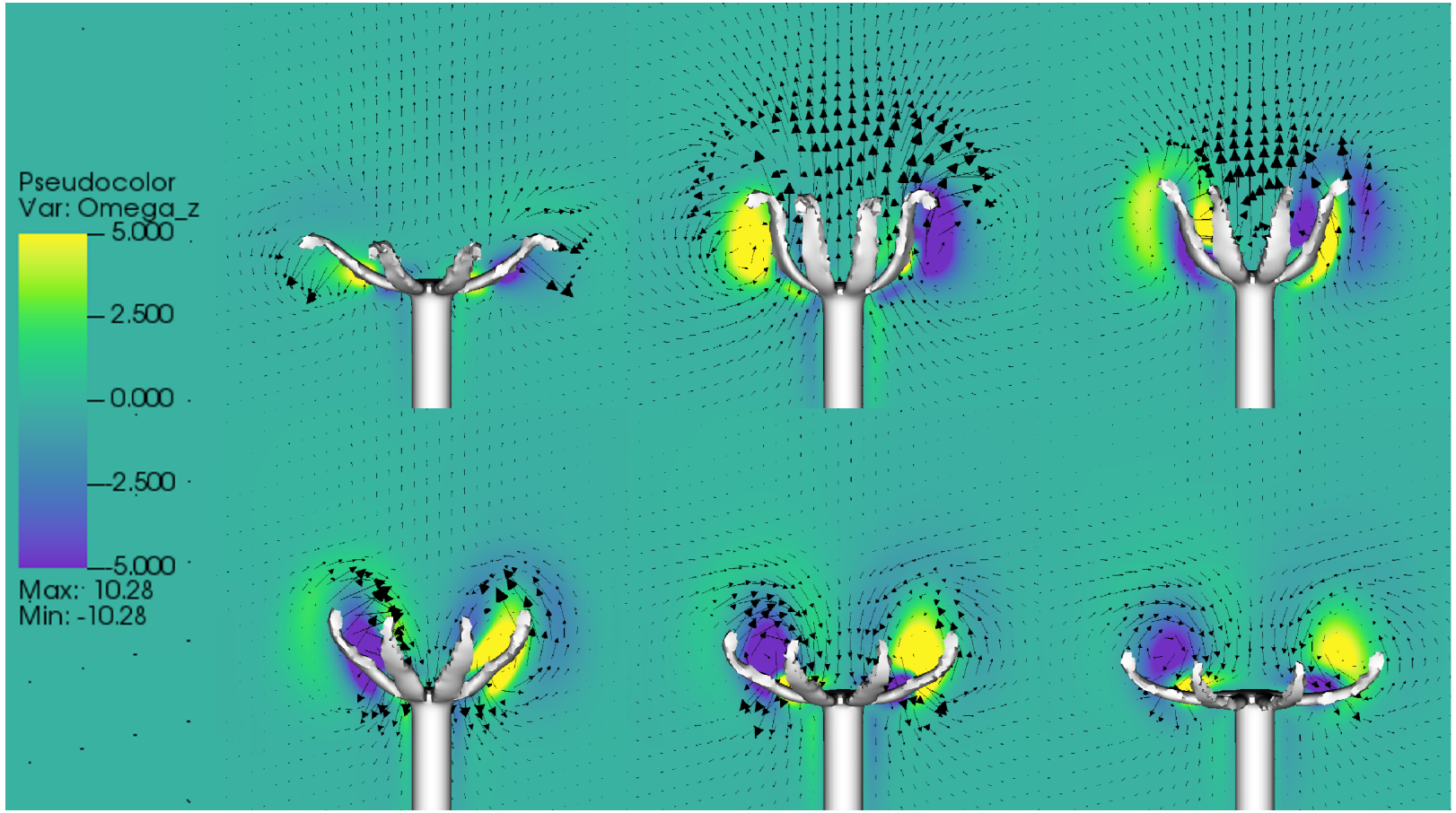
In a different script, you can open the .VTK database (once you've added the dumps.visit file containing the filenames).

```
string_vtk = "localhost:/path_to_vtk_database/dumps.visit"
OpenDatabase(string_vtk)
```

Resampling and exporting to create these databases is much, much faster on the cluster.

Using .VTK databases

The .VTK database can be useful to plot uniform velocity vectors on a slice:



Using .VTK databases – Averaging Velocity

The .VTK database can also be used for spatial averaging. This script plots a pseudocolor of the vertical velocity in a box in the domain. Then it averages in that box over time.

```
OpenDatabase(string_vtk)
AddPlot("Pseudocolor", "U_y", 1, 0)
AddOperator("Box", 0)
SetActivePlots(0)
BoxAtts = BoxAttributes()
BoxAtts.amount = BoxAtts.Some  # Some, All
BoxAtts.minx = -0.0075
BoxAtts.maxx = 0.0075
BoxAtts.miny = 0.0043
BoxAtts.maxy = 0.0097
BoxAtts.minz = -0.0075
BoxAtts.maxz = 0.0075
BoxAtts.inverse = 0
SetOperatorOptions(BoxAtts, 0)
DrawPlots()
```

Using .VTK databases – Averaging Velocity, cont.

A query over time performs the temporal average.

You can set the end_time to a very high number and VisIt will automatically clamp to the number of timesteps you have. This way you don't accidentally cut your query short.

```
SetActivePlots(0)
QueryOverTimeAtts = GetQueryOverTimeAttributes()
QueryOverTimeAtts.timeType = QueryOverTimeAtts.DTime  # Cycle, DTime, Timestep
QueryOverTimeAtts.startTimeFlag = 0
QueryOverTimeAtts.startTime = 0
QueryOverTimeAtts.endTimeFlag = 0
QueryOverTimeAtts.endTime = 1
QueryOverTimeAtts.strideFlag = 0
QueryOverTimeAtts.stride = 1
QueryOverTimeAtts.createWindow = 0
QueryOverTimeAtts.windowId = 2
SetQueryOverTimeAttributes(QueryOverTimeAtts)
SetQueryFloatFormat("%g")
QueryOverTime("Average Value", end_time=1000, start_time=0, stride=1)
```

Using .VTK databases – Averaging Velocity, cont.

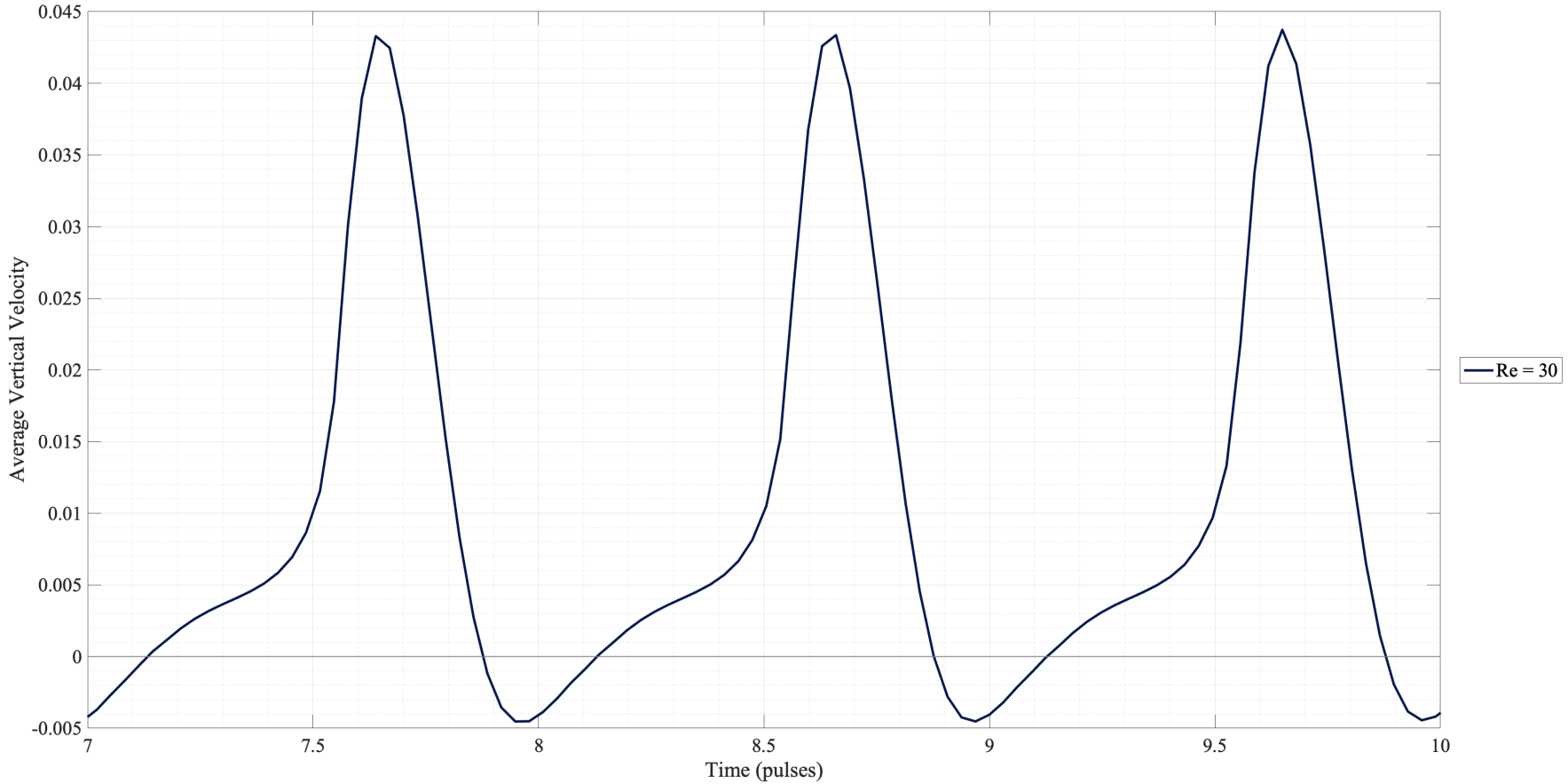
Finally, you can save your average as a .curve file. This can be opened and plotted in Matlab.

The .curve file will have a line at the very beginning with text preceded by a #. This will prevent it loading as a matrix in Matlab, but you can open the file and remove it manually.

```
SetActiveWindow(2)
SaveWindowAtts = SaveWindowAttributes()
SaveWindowAtts.outputToCurrentDirectory = 0
SaveWindowAtts.outputDirectory = "."
SaveWindowAtts.fileName = "avg_uy"
SaveWindowAtts.family = 1
SaveWindowAtts.format = SaveWindowAtts.CURVE
SetSaveWindowAttributes(SaveWindowAtts)
SaveWindow()
```

Using .VTK databases – Averaging Velocity, cont.

An example of a .curve file plotted in Matlab is:



Restarting a job

If you want to restart a job that has unexpectedly stopped, add the path to the restart_IB3d directory and the number of the last restart file to your submission script.

```
#!/bin/bash
#SBATCH --job-name=re48
#SBATCH --ntasks-per-node=44
#SBATCH -N 4
#SBATCH --time=72:00:00
#SBATCH --partition=528_queue
#SBATCH --output output

mpirun ./main3d input3d $PWD/restart_IB3d 995000
```

Things to remember

- These Python scripts are whitespace sensitive
- Errors that pop up in your command line should have the line number in them, this will help with debugging
- If you change the order in which you do things, you may need to change the number in the SetActivePlots() command.
- Having a good structure for your input and output file directories will save you time and confusion

Useful Resources

- Of course, all of Laura's tutorials and examples:
  https://github.com/fairyflies9/IBAMR-Tutorials
- The VisIt user manuals, for information about what different types of analysis do:
  https://visit-sphinx-github-user-manual.readthedocs.io/en/develop/index.html
- Shannon Jones' YouTube tutorials:
  https://www.youtube.com/channel/UCo4Govw4uKhLYiBbcgHqWrg