PyFoam 4 the Lazy

Post-simulation depression: "I actually have to **look** at all that data?"

Bernhard F.W. Gschaider

HFD Research GesmbH

Guimaraes, Portugal
28. June 2016

Outline I

Outline II

Outline

Outline

Initial response

- When the presentation was announced one of the intial responses was by `@dancombest`:
  - "@bgschaid For the "PyFoam for the lazy" training session, will this be scheduled late enough for the target audience?"
- It is not uncommon to confuse *lazy* with *sleepy*
- I mean lazy in the sense of this classic definintion
  - "A programmer is a lazy person who works hard to avoid further work"
- The lazy CFD-engineer stands by the coffee machine:
  - "The computer is currently producing the pictures you want. I can't help him. I set him up for that"
- The goal of this presentation to help you to be that engineer
- When talking to people outside of this room: replace lazy with efficient

Disclaimer

- This presentation will have no movies and fancy visualizations
  - That would be work and I'm lazy
- This presentation has less than 200 slides
  - Because I'm lazy

If you think that this presentation is against your work ethics:

- There are good training sessions in the other rooms where they make you work hard. You can go there
- On the other hand: I will challenge your laziness by making you do things

Used software

The presentation uses

- PyFoam 0.6.6 - to be released
- swak4Foam 0.4.0 - to be released
- OpenFOAM 3.0 (Foundation release)
- several Python-libraries (`pandas` etc)

All of these are on the stick. So you'd better use that

Command line examples

- In the following presentation we will enter things on the command line. Short examples will be a single line (without output but a ">" to indicate *input*)

```
> ls $HOME
```

- Long examples will be a grey/white box
  - Input will be prefixed with a > and blue
  - Long lines will be broken up
    - A pair of <brk> and <cont> indicates that this is still the same line in the input/output
  - «snip» in the middle means: "There is more. But it is boring"

Long example

```
> this is an example for a very long command line that does not fit onto one line of the slide but we <brk>
      <cont>have to write it anyway
first line of output (short)
Second line of output which is too long for this slide but we got to read it in all its glory. The end<brk>
      <cont> of the line is not yet her. But now it is and we're on a new line
```

Outline

After the CPUs cool down

There are three phases

1. Pre-processing
   - That is really hard work
   - PyFoam can help a bit here
     - See "Automatic case setup with `pyFoamPrepareCase`" from the Ann Arbor Workshop last year
2. Calculation
   - That runs pretty automatic. Not much work here
     - Except for the CPU
3. Post-processing
   - All the data on the disk has to be made useful
     - Interpreted
     - Prepared for presentation
   - That is work
     - Especially as it involves a lot of *mouse-pushing*

What we hear while doing our post-processing

- "Make pictures in Paraview"
  - "Make the same ones for every case"
  - "And make sure that the view angle is always the same"
  - "And I want the same pink to grey colorbar for the temperatures"
- "What is the average value of . . . "
  - "For all the cases, please"
  - "Are you sure that is the value for this case"
- "How does this evolve over time?=
  - "Could I have the data in Excel? I want to send it to my accountant"
  - "Accountant is confused by 'standard deviation'. Please don't put it into the spreadsheet"

Don't you wish you had one button to make these questions go away?

- In our case the button would be a script

Outline

What is PyFoam

- PyFoam is a library for
  - Manipulating OpenFOAM-cases
  - Controlling OpenFOAM-runs
- It is written in Python
- Based upon that library there is a number of utilities
  - For case manipulation
  - Running simulations
  - Looking at the results
- All utilities start with pyFoam (so TAB-completion gives you an overview)
  - Each utility has an online help that is shown when using the -help-option
  - Additional information can be found
    - on openfoamwiki.net
    - in the two presentations mentioned above

Case setup

- Cloning an existing case

```
> pyFoamCloneCase.py $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily test
```

- Decomposing the case

```
> blockMesh -case test
> pyFoamDecompose.py test 2
```

- Getting info about the case

```
> pyFoamCaseReport.py test --short-bc --decomposition | rst2pdf >test.pdf
```

- Clearing non-essential data

```
> pyFoamClearCase.py test --processors
```

- Pack the case into an archive (including the last time-step)

```
> pyFoamPackCase.py test --last
```

- List all the OpenFOAM-cases in a directory (with additional information)

```
> pyFoamListCases.py .
```

Running

- Straight running of a solver

```
> pyFoamRunner.py interFoam
```

- Clear the case beforehand and only show the time

```
> pyFoamRunner.py --clear --progress interFoam
```

- Show plots while simulating

```
> pyFoamPlotRunner.py --clear --progress interFoam
```

- Change `controlDict` to write all time-steps (afterwards change it back)

```
> pyFoamRunner.py --write-all interFoam
```

- Run a different OpenFOAM-Version than the default-one

```
> pyFoamRunner.py --foam=1.9-beta interFoam
```

- Run the debug-version of the current version

```
> pyFoamRunner.py --current --force-debug interFoam
```

Plotting

- Any logfile can be analyzed and plotted

```
> pyFoamPlotWatcher.py --progress someOldLogfile
```

- A number of things can be plotted
  - Residuals
  - Continuity error
  - Courant number
  - Time-step
- User-defined plots can be specified
  - Specified in a file `customRegexp`
  - Data is analyzed using regular expressions
  - We will see examples for this later
- The option `--hardcopy` generates pictures of the plots

What else can PyFoam do for me?

- Write and read dictionaries from the command line
- Display the `blockMeshDict`
- Interact with paraView
- Control OpenFOAM-runs over the net
- ...

Outline

400 cells

- Our work will be based on the `icoFoam`-tutorial `cavity`
  - Because small is good
- But we'll use a modified version
  - Works with `icoFoam`, `pimpleFoam`, `pisoFoam` and `rhoPimpleFoam`
    - To compare solvers
    - Should all give the same results
  - Added evaluations
    - Done mainly with `swak4Foam`
    - But also with "regular" function objects
- Additional choices
  - A simple and a graded mesh
    - To demonstrate comparing different resolutions
  - Adaptive or fixed time-step
    - If the solver supports that
  - Different turbulence models

pyFoamPrepareCase.py

- pyFoamPrepareCase.py is a utility for lazy people to set up OpenFOAM-cases
- Cases are "programmed" using *templates*
  - Proper dictionaries are create by filling in values
    - Helps setting cases up consistently
  - Allows conditional set-up ("if compresseible solver then set up differently")
- Sets the case up with just one command
  - User is encouraged to write scripts for mesh creation and case setup
    - Which can be templates as well
- Flexible way of specifying parameters to use during set up
- The example case was prepared to be set up with this utility
  - Switching between meshes etc

Getting onto the same page

- I assume you're on the stick
  - For other environments you're on your own
- Switch on OpenFOAM-3.0.x

```
> of30x
```

- Now the prompt should show `(OF:3.0.x)`

Getting the case files

- The example case can be found with the training materials
  - Works with OpenFOAM 3.0
- If you're on the stick do this
  - Lines starting with > are meant to be typed in
    - Everything else is output

```
> mkdir myWork
> cd myWork
> tar xvzf ~/Training/BernhardGschaider/CavityPyFoam4TheLazy.tgz
> cd CavityPyFoam4TheLazy
```

- If you're not on the stick: find the tar-file and untar it
  - But this presentation assumes that you have a fairly recent version of `swak4Foam`
    - Which is on the stick: best to work from the stick

Setting up the case and running it

- Setting up for the compressible solver and the graded mesh

```
> pyFoamPrepareCase.py . --values="{'solver':'rhoPimple','blockMesh':'graded'}"
```

- Takes care of everything
  - Including making sure that `mu` matches the `nu` from the incompressible solvers
- Also creates a "run script"

```
> ./runMe.sh
Clearing out old timesteps ....
Adding automatic plots: Custom11_cloudnumbermass
 t =       8.25  pos=(0.05690099084,0.05)
```

- Runs the solver and saves results to a unique directory

```
> ls post_rhoPimple_turb_kOmega_graded
Analyzed                              swakExpression_minStreamFunction
Logfile                               swakExpression_minStreamFunctionPos
lagrangian                            swakExpression_velocityStatistics
sample
```

Doing it all at once

. . . because we're lazy

- Set up all data at once
  - Runs various solvers with different combinations
  - `runMe.sh` stores in `post_*`-directories

```
./runAll.sh
```

- Additional directory `postRef` with the `icoFoam`-results as a reference

Outline

Things OpenFOAM always reports

This section gives an overview of the data that is generated by executing the solver
"Standard" output of OpenFOAM is:

- Initial residuals of the linear solvers
- Continuity
- Courant numbers

Statistics about the velocity

- To check for convergence we'll look for the evolution of the absolute velocity
  - Maximum
  - Weighted average
    - Weighted with the cell size
  - The 10% and the 90% quantile
    - Volume-weighted
    - Meaning: "10% of the volume has a lower velocity"
    - *99% quantile* is more stable than the maximum
- Calculated at every time
  - We'll use a swak4Foam function object for this

swak4Foam generates

- The `swakExpression` function object from `swak4Foam` can evaluate an expression and print statistics about it

constrolDict

```
velocityStatistics {
    type swakExpression;
    verbose true;
    outputControlMode timeStep;
    outputInterval 1;
    valueType internalField;
    expression "mag(U)";
    accumulations (
        weightedQuantile0.1
        weightedAverage
        weightedQuantile0.9
        max
    );
}
```

PyFoam reads and plots

- pyFoamRunner.py scans the output for date
  - Specification in a file customRegexp

Output and scan

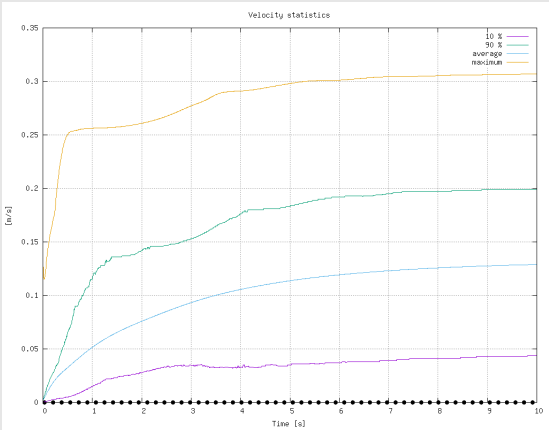
This output

```
Expression velocityStatistics :  weightedQuantile0.1=0.044 <brk>
      <cont>weightedAverage=0.1289522533 weightedQuantile0<brk>
      <cont>.9=0.199 max=0.3069643195
```

is caught by

```
velocity {
    theTitle "Velocity statistics";
    expr "Expression velocityStatistics : : weightedQuantile0<brk>
          <cont>.1=(.+) weightedAverage=(.+) <brk>
          <cont>weightedQuantile0.9=(.+) max=(.+)";
    titles (
        "10 %"
        average
        "90 %"
        maximum
    );
    ylabel "[m/s]";
}
```

Different velocities



The stream function

- From `https://www.wikiwand.com/en/Stream_function`
  - "The stream function is defined for incompressible (divergence-free) flows in two dimensions"
  - "The stream function can be used to plot streamlines"
- In our case the location of the minimum value of the *stream function* is the center of the vortex
- OpenFOAM has a utility `streamFunction` to calculate the *stream function* for a given flow field
  - `swak4Foam` has a function object that does the same thing
    - Code was "borrowed" from the utility
    - We're using that to calculate the location of the vortex center

`controlDict`

```
minStreamFunctionPos {
    type swakExpression;
    verbose true;
    outputControlMode timeStep;
    outputInterval 1;
    expression "streamF";
    valueType internalField;
    accumulations (
        min
    );
    expression "minPosition(streamF)";
}
```

Added particles

- The `swak` function object `evolveKinematicCloud` adds a particle cloud `coldParticleCloud` to the solver
  - For the *incompressible* solvers artificial ρ and μ fields have to be added (particle clouds need them)
    - With the `expressionField` function objects
    - This is switched py `pyFoamPrepareCase.py`
  - Most of the settings happen in the "regular" dictionary file with the settings
    - Particle injection: a cone injector
    - Particle properties and submodels (drag)
    - Particles are eliminated after 5 seconds by a `eliminateBySwakExpression` cloud function object

Automatic expressions in PyFoam

- Many solvers produce similar output
  - For instance: volume fractions on VoF (Volume of Fluid) solvers
  - Number of particles in solvers that have Lagrangian Particles
- All this output is of interest but
  - Writing it for every solver is repetitive
  - Making PyFoam scan for it always makes the scanning slower
- Alternative: Copying from the `customRegexp` of another case to the `customRegexp` of this case
  - This is not lazy. Nothing for us
- New Version of PyFoam can automatically scan for such information for the right solvers
  - If the name of the current solver fits a list of patterns called solvers the output is scanned for the plotinfo
- But how does PyFoam know about this?
  - Through the configuration system

The configuration system

- For things that may differ on systems PyFoam allows to configure them
  - For instance "how to properly call `mpirun`-program"
  - Configurations are organized is sections (for instance `[MPI]`)
    - There can be version specific sections (special treatment for `mpirun` in OpenFOAM 7.8 could be found in `[MPI-7.8]`)
  - In the sections there are keys (for instance `options_openmpi_pre` for additional parameters for `mpirun`)
  - Values for the options can be numbers, strings or Python lists or dictionaries (depends)
- Locations where configurations are found are (also listed by `pyFoamVersion.py`)
  1. Hardcoded in the PyFoam-sources
  2. System-wide in `/etc/pyFoam/`
  3. User-specific in `$HOME/.pyFoam`
  4. Per-case in a file `LocalConfigPyFoam` in the case directory
- Highest number wins

Listing the configuration

What are the currently used Settings

```
> pyFoamDumpConfiguration.py
[Autoplots]
cloudnumbermass: { 'plotinfo': { 'alternateAxis': ['.+_mass'],
                 'expr': 'Cloud: (.+)\\n +Current number of parcels += '
                         '(.+)\\n +Current mass in system += (.+)',
                 'idNr': 1,
                 'theTitle': 'Particle number and mass',
                 'titles': ['nr', 'mass'],
                 'type': 'dynamic',
                 'y2label': 'Mass in system',
                 'ylabel': 'Particle number'},
  'solvers': ['coal.+Foam', '.+Parcel.*Foam', 'sprayFoam']}
aspectratiodynamicmesh: { 'plotinfo': { 'expr': 'Max aspect ratio = (\\S+)',
                 'master': 'nonorhogonalitydynamicmesh',
                 'titles': ['max aspect'],
                 'type': 'slave'},
  'solvers': ['move.*Mesh', '.*DyM.*']}
cellvolumesdynamicmesh: { 'plotinfo': { 'alternateAxis': ['Total'],
                 'expr': 'Min volume = (\\S+)\\. Max volume = (\\S+)\\.  '
                         'Total volume = (\\S+)\\.',
                 'logscale': True,
                 'theTitle': 'Cell volumes',
                 'titles': ['Minimum', 'Maximum', 'Total'],
                 'ylabel': '[m^3]'},
  'solvers': ['move.*Mesh', '.*DyM.*']}
chtfluidcourant: { 'plotinfo': { 'alternateAxis': ['.+Diffusion.+'],
                 'expr': 'Region: (.+) Number mean: (.+) max: (.+)',
                 'idNr': 1,
                 'theTitle': 'Courant and diffusion number',
                 'titles': ['mean', 'max'],
                 'type': 'dynamic',
                 'y2label': 'Diffusion',
```

Configuration settings for the case

- Our solver is not "officially" a "cloud"-solver
  - But we want to scan the particle stuff
- `pyFoamPrepareCase.py` should be allowed to execute more complicated Python-expressions
  - This is a potential security hole

`LocalConfigPyFoam`

```
[Template]
allowexecution: True

[Plotting]
autoplots: cloudnumbermass
```

Outline

Outline

OpenFOAM tells us things

Two main mechanisms for OpenFOAM to tell the world about things

1. Writing files
   1. "Classic" field files (`U`, `T`, etc)
   2. Special files written to `postProcessing` by function objects
2. Printing stuff to the terminal

PyFoam is good at handling 1.2 and 2

- Not so good for 1.1: use `paraview`

Logging with PyFoam

- `pyFoamRunner.py` and `pyFoamPlotRunner.py` automatically capture the terminal output
  - Write it to a file `PyFoamRunner.icoFoam.logfile`
    - Instead of `icoFoam` the current solver/utility is used
  - Prints it back to the terminal
    - Unless being told not to (`-progress`)
- Basically:

```
> pyFoamRunner.py icoFoam
```

does

```
> icoFoam 2>&1 | tee PyFoamRunner.icoFoam.logfile
```

- But there is additional functionality:
  - Data analysis
  - Possibility to compress the logfiles
  - other

Processing log files with the watcher

- `pyFoamPlotWatcher.py` can process log files the same way the `Runner`-utilities do
  - Writes the same data
  - Assumes that the file is still being appended to
- Doesn't care where the logfiles come from
  1. PyFoam
  2. Normal file written by redirecting the output with >
  3. Output from you batch-queuing system
- Can't take its input from a pipe
  - But that's what the runners are for

Generating plots from the logfiles

All these utilities produce nice plots while running but "I need something to print out and hang on the wall to impress whoever comes to the office"

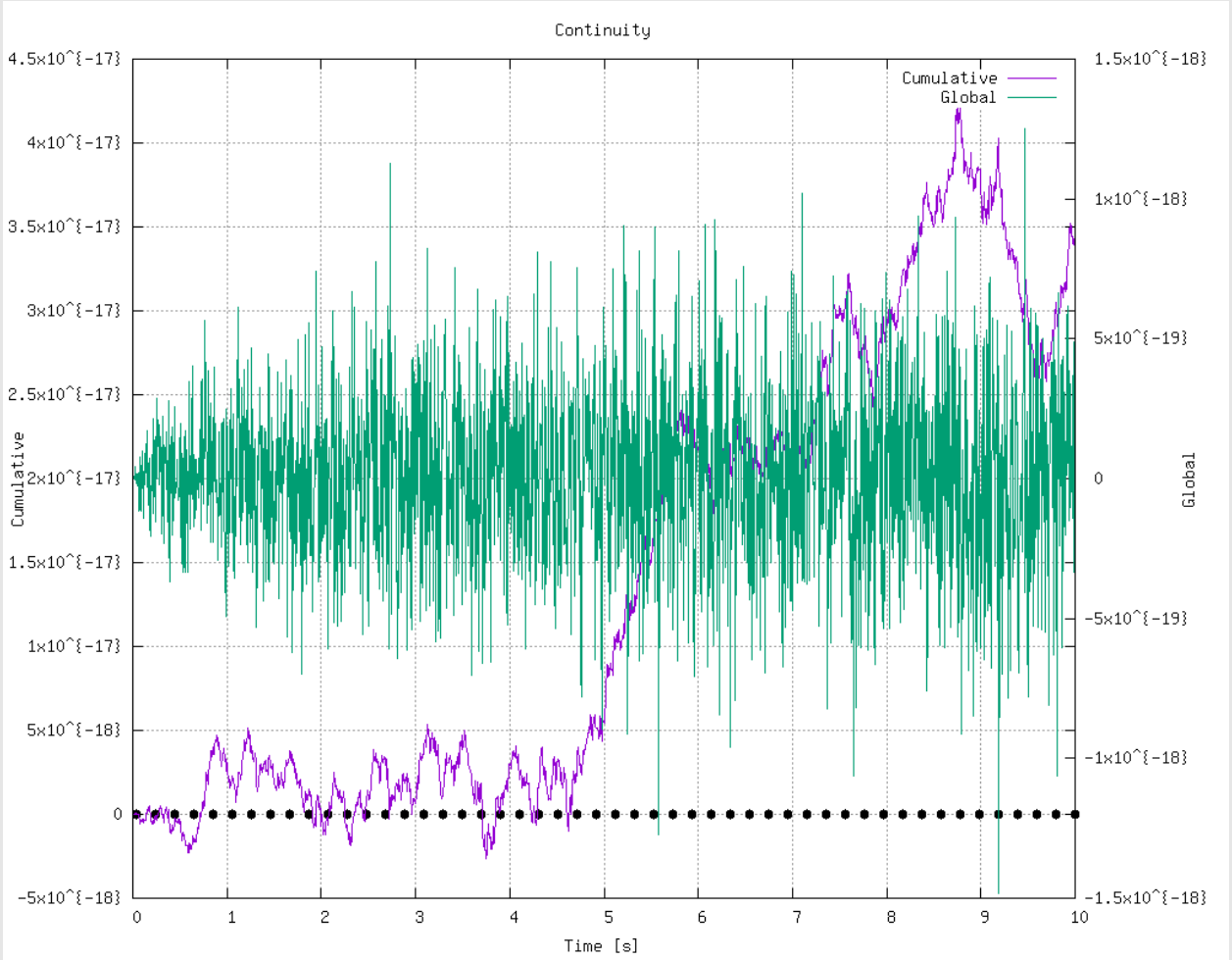
- `pyFoamPlotWatcher.py` and `pyFoamRunner.py` have an option `--hardcopy` that writes copies of the plots
  - Image format can be selected with `--format-of-hardcopy`
    - Plots are generated by `GnuPlot`. Some people don't like the look of those plots
  - Prefix can be added to the file names with `--prefix-hardcopy`
    - To distinguish plots from different runs

Simple hardcopy

```
> pyFoamPlotWatcher.py PyFoamRunner.icoFoam.logfile --hardcopy --prefix-hardcopy=ico --<brk>
      <cont>solver-not-running-anymore
 Reading regular expressions from customRegexp
Adding automatic plots: Custom07_cloudnumbermass
<<snip>>
End
> ls *.png
ico.cont.png         ico.custom0000_.png ico.linear.png
ico.custom0000.png   ico.custom0002.png
```

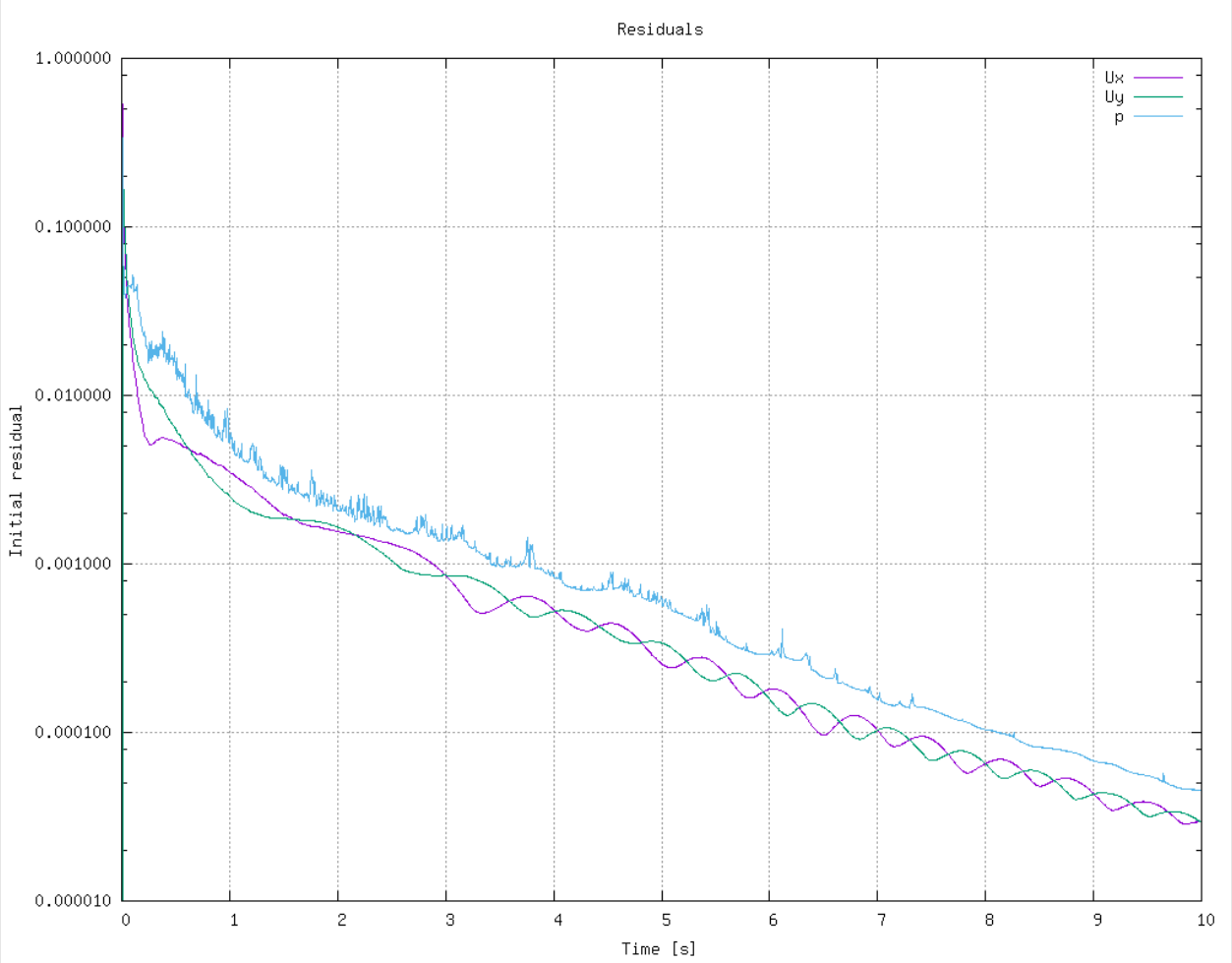
ico.cont.png

Continuity



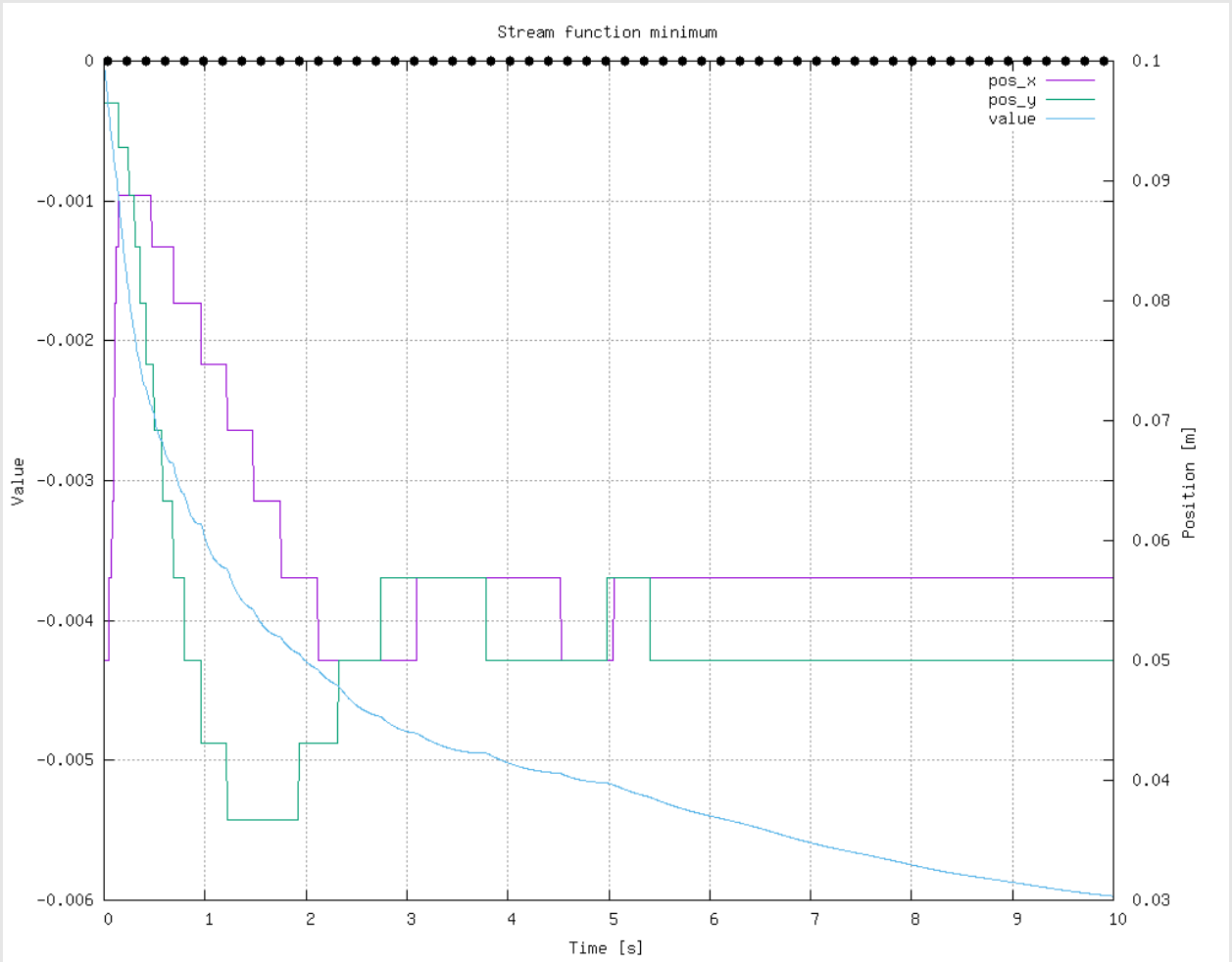
ico.linear.png

Residuals of the linear solvers



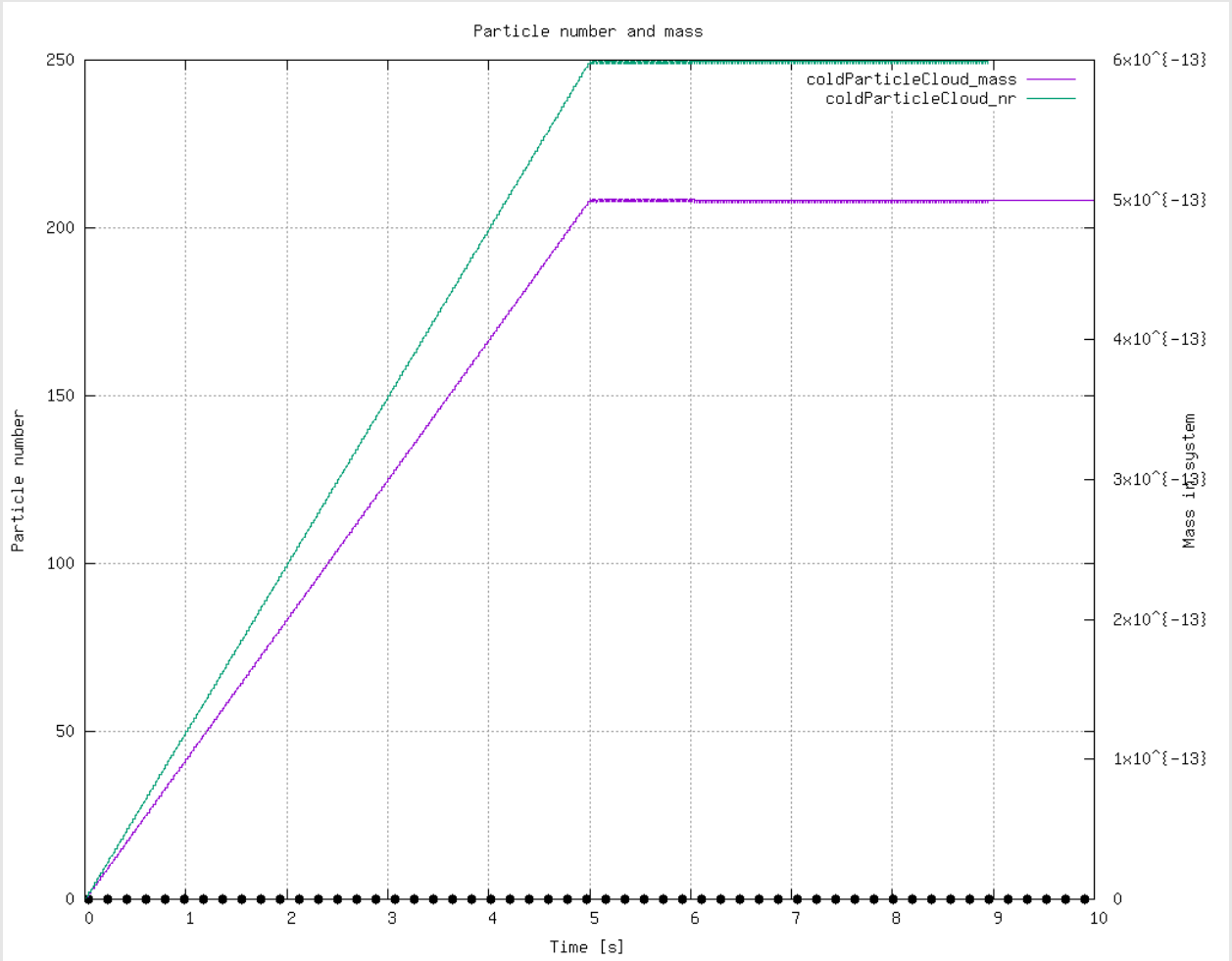
ico.custom0000.png

Stream function - location and value



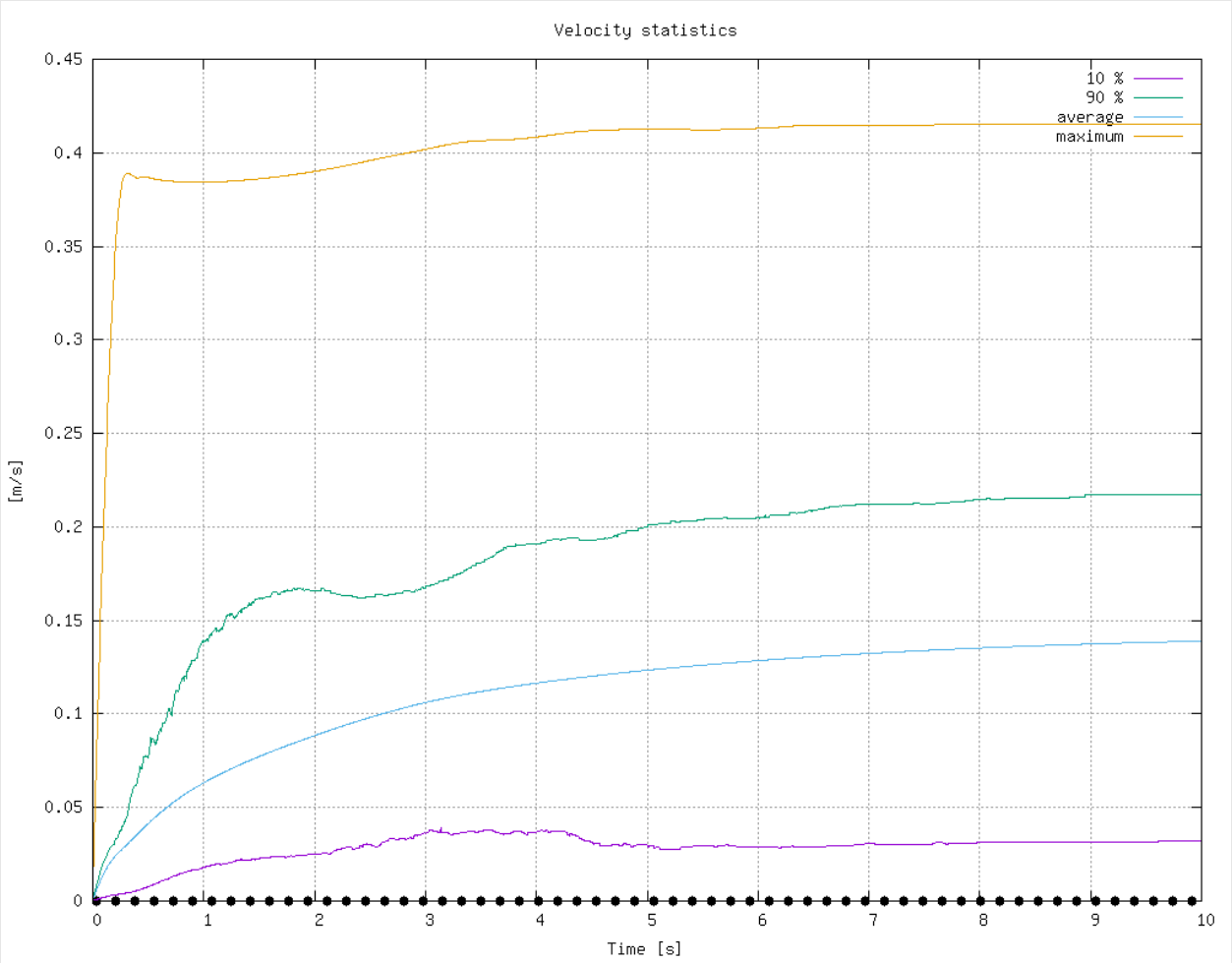
ico.custom0000_.png

Particle statistics



ico.custom0002.png

Velocity statistics



Outline

Analyzed data

- Analyzing large log files (several Megabytes) can take some time
  - Several minutes to hours
- Why wait so long for the results?
- Solution in PyFoam: when processing the log files save the already extracted data
  - Automatically written every second
    - But interval is raised if it takes too long
- Data is written to a file `PyFoamRunner.pimpleFoam.analyzed/pickledPlots`

What is *pickled* ?

- Text files are good
  - Easy to read
  - Work everywhere
- Text files are bad
  - Not very compact
  - Data has to processed again when reading
- Python standard library `pickle` writes Python-data-structures in a binary format an reads it again
  - Very compact
  - No processing needed
  - Works with every Python

Redoing the plots

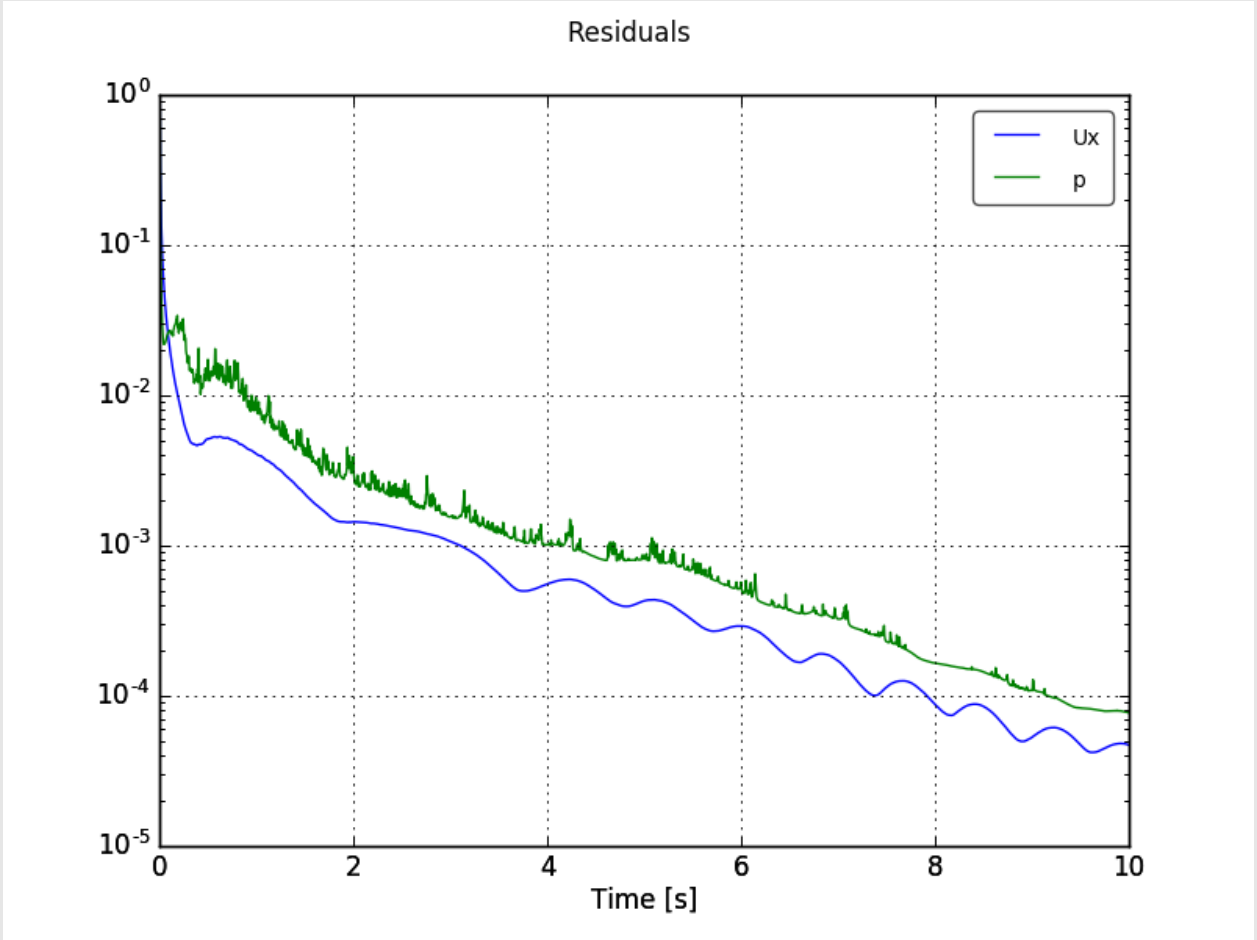
- The utility `pyFoamRedoCase.py`
  - Reads the pickled data from the plots
  - Generates images
- Uses `matplotlib` instead of `gnuplot`
  - Some say this is prettier
- Option `--pickle-file` says "read from file"
  - There is also a network-mode

Redoing is faster

```
> pyFoamRedoPlot.py post_ico_normal/Analyzed/pickledPlots --pickle-file  --picture-prefix=<brk>
      <cont>baseline_
Found 10 plots and 11 data sets
Adding line 8
Adding line 10
Adding line 4
Adding line 6
Adding line 2
Adding line 5
Adding line 11
Adding line 7
Adding line 9
Adding line 3
Adding line 1
Plotting 8 : streamFunctionMinimum
Plotting 4 : iterations
Plotting 5 : courant
Plotting 6 : timestep No data - skipping
Plotting 2 : continuity
```

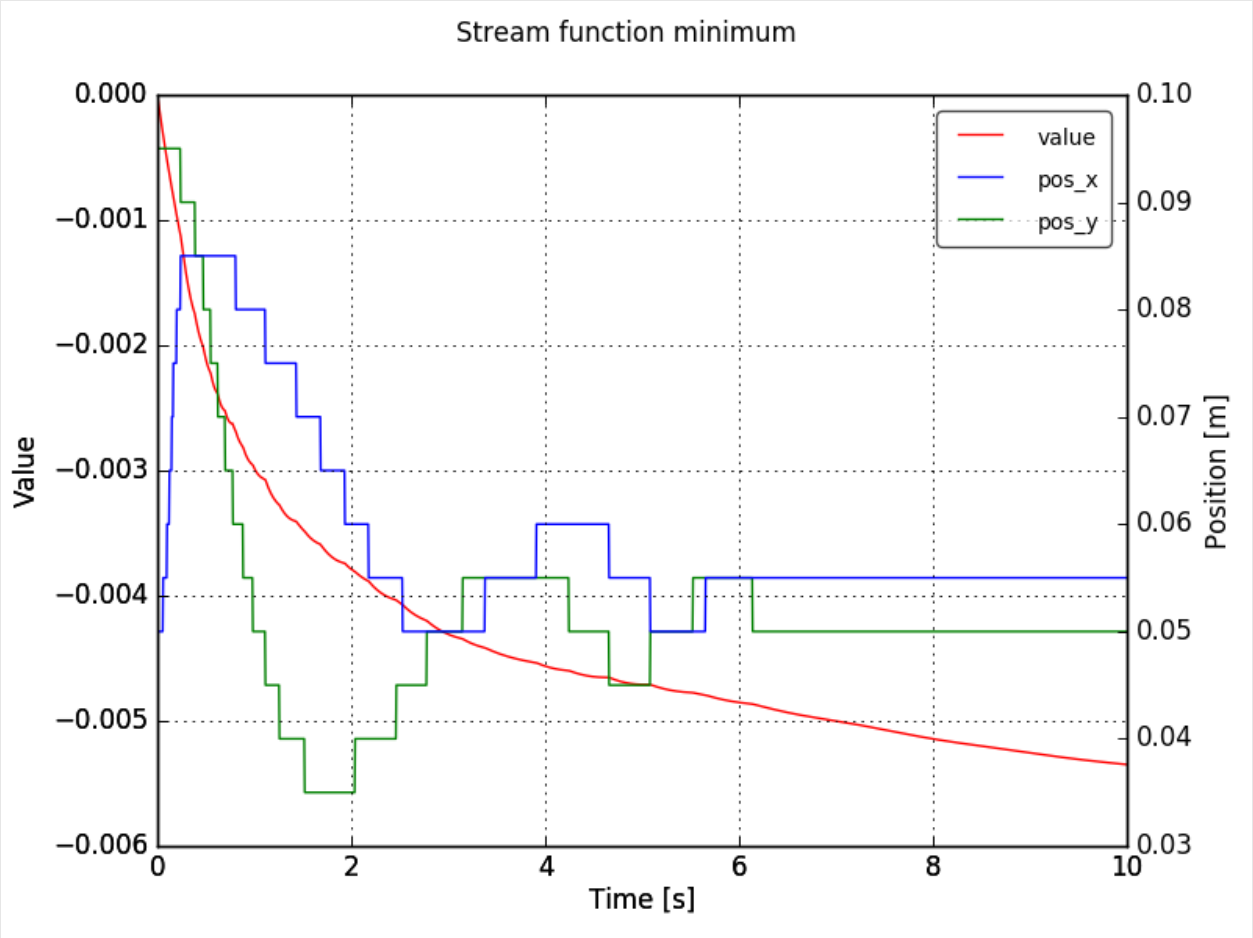
baseline_linear.png

Linear solver residuals



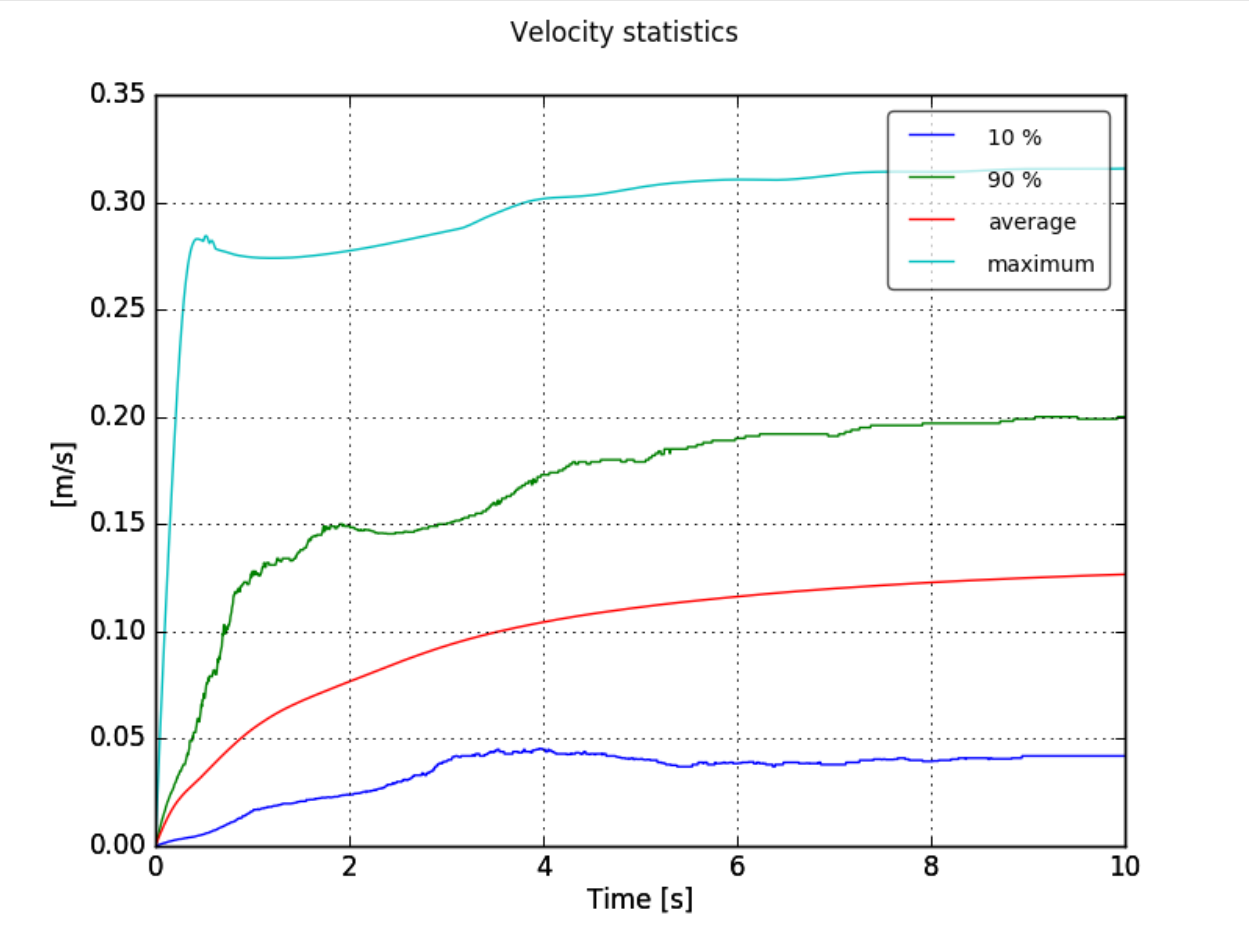
baseline_streamFunctionMinimum.png

Location of the stream function minimum



baseline_velocity.png

Velocity statistics



Making the plots look geeky

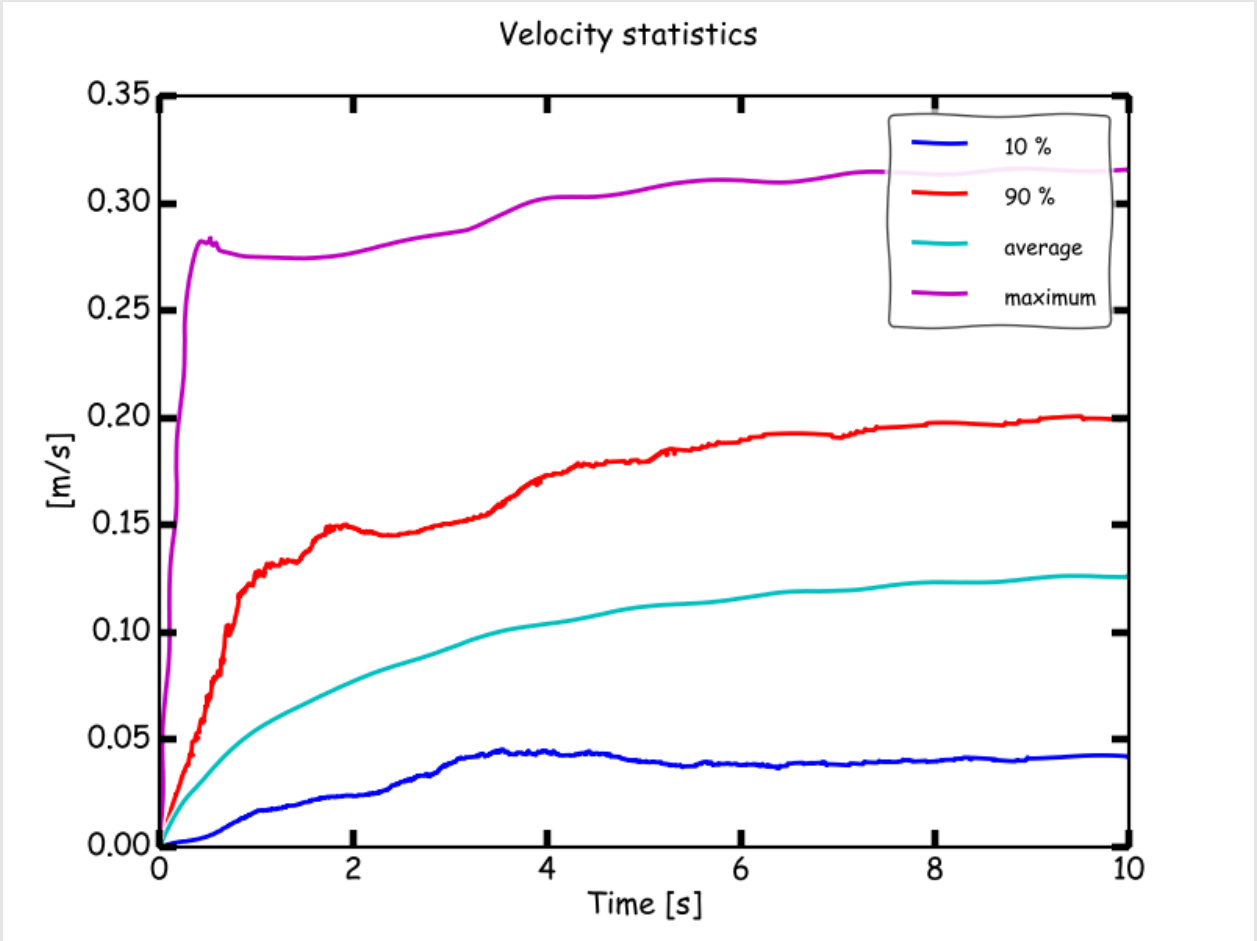
- Different implementations for doing the plots can be choosen
  - With the option `--implementation` (surprise!)
- One implementation is `gnuplot`
  - If you want the look of the `pyFoamPlotRunner.py`
- Another implementation is `xkcd`
  - Variation of `matplotlib`
  - Makes plots look like the web-comic `https://xkcd.com/`

Make it shaky

```
> pyFoamRedoPlot.py post_ico_normal/Analyzed/pickledPlots --pickle-file  --picture-prefix=<brk>
      <cont>xkcdBaseline_ --implementation=xkcd
Found 10 plots and 11 data sets
Adding line 8
Adding line 10
Adding line 11
```

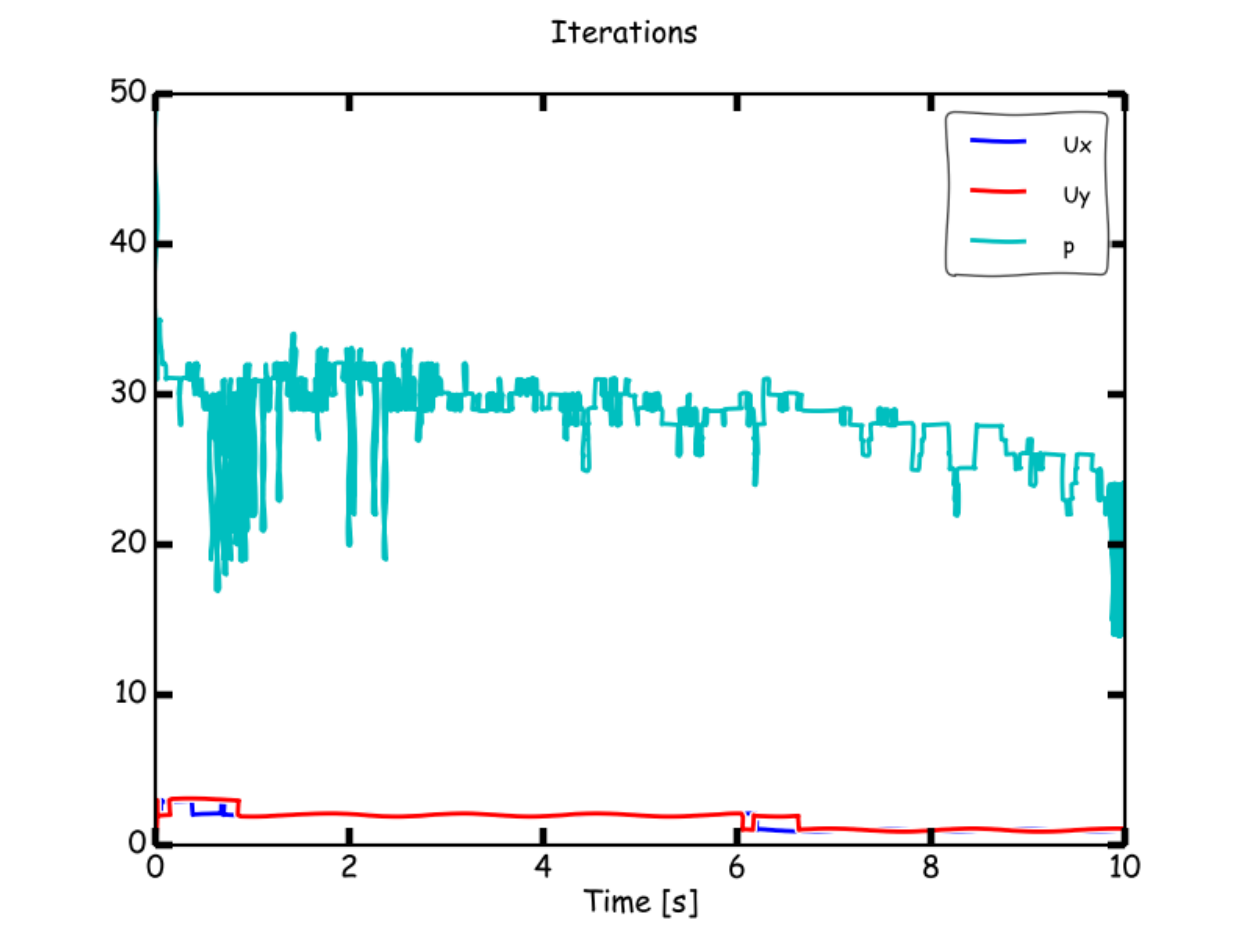
xkcdBaseline_velocity.png

Velocity diagram drawn by hand



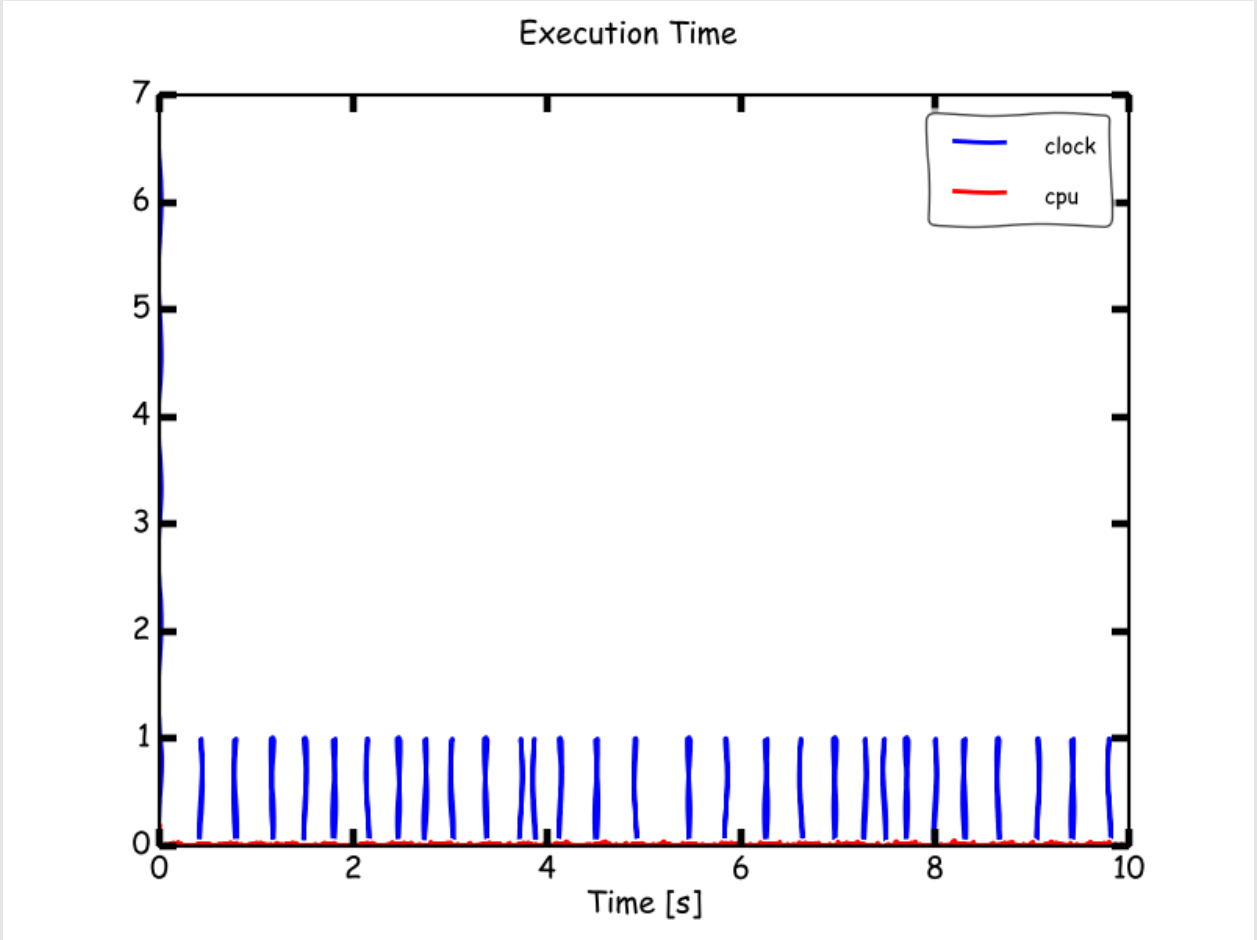
xkcdBaseline_iterations.png

Number of iterations



xkcdBaseline_execution.png

Execution per time-step



Writing for un-geeky programs

- But sometimes a XKCD-style plot is not wanted
  - Data should be visualized in a "serious" program
- `--csv-files` makes `pyFoamRedoPlot.py` write CSV-files
  - Nothing is plotted
- `--excel-files` writes files for *Micro$oft Excel*
  - Requires additional Python-libraries

Somebody wants to use Office in the office

```
> pyFoamRedoPlot.py post_ico_normal/Analyzed/pickledPlots --pickle-file  --excel-files --<brk>
      <cont>file-prefix=ms_
Found 10 plots and 11 data sets
<<snip>>
> ls *.xls
ms_cloudnumbermass.xls       ms_execution.xls             ms_streamFunctionMinimum.xls
ms_continuity.xls            ms_iterations.xls            ms_velocity.xls
ms_courant.xls               ms_linear.xls
```

Case parameters

- `pyFoamRunner.py` knows about a lot of things
  - For instance
    - Solver name
    - When the run was started
    - Warnings
    - Analyzed data (only the last value)
    - ...
- All this data is collected in a file `pickledData`
  - While the simulation is running an intermediate file `pickledUnfinishedData` is written
- All this data is available for your own scripts
  - Easiest: Python. As the pickled-format is Python-specific other languages have a hard time
- There is a utility that prints all the information

Printing case data

Only the start of the information

```
> pyFoamEchoPickledApplicationData.py --pickled-file=PyFoamRunner.pimpleFoam.analyzed/<brk>
      <cont>pickledData --print-data
{'OK': True,
 'analyzed': {'Continuity': {'Cumulative': 6.354004801e-19,
                             'Global': 3.101036487e-18},
              'Courant': {'max': 0.9404536707, 'mean': 0.4539791106},
              'Custom': {'cloudnumbermass': {'coldParticleCloud_mass': 4.984615385e-13,
                                             'coldParticleCloud_nr': 249.0},
                         'streamFunctionMinPos': {'pos_x': 0.055,
                                                  'pos_y': 0.05},
                         'streamFunctionMinimum': {'value': -0.005422092566},
                         'velocity': {'10 %': 0.043,
                                      '90 %': 0.195,
                                      'average': 0.1249998037,
                                      'maximum': 0.3057894994}},
              'Custom01_streamFunctionMinimum': {'value': -0.005422092566},
              'Custom02_streamFunctionMinPos': {'pos_x': 0.055,
                                                'pos_y': 0.05},
              'Custom03_velocity': {'10 %': 0.043,
                                    '90 %': 0.195,
                                    'average': 0.1249998037,
                                    'maximum': 0.3057894994},
              'Custom11_cloudnumbermass': {'coldParticleCloud_mass': 4.984615385e-13,
                                           'coldParticleCloud_nr': 249.0},
<<snip>>.
```

Outline

The pyFoamListCases.py utility

- `ls` for OpenFOAM-users
  - Only lists directories that are an OpenFOAM-case
  - Prints additional information:
    - Number of written timesteps
    - Time the case was "modified" (for instance: timestep was written): `mtime`
    - Still running?
  - Cases are listed with rising `mtime` ("newest" are at the bottom)
- Additional output can be switched on
  - Parallel timesteps
  - Disk usage
- Sort order can be changed

Only two cases

```
> pyFoamListCases.py .
                    mtime |           hostname | first - last (nrSteps) nowTime s       state<brk>
                          <cont> |     solver | name
-------------------------------------------------------------------------------------------------
      <cont>
Fri Jun 24 00:55:22 2016 | bgs-cool-greybook |     0 -   2.6 (     14)     2.5 s Interrupted<brk>
      <cont> |   pisoFoam | ./testCavity
Fri Jun 24 01:02:26 2016 | bgs-cool-greybook |     0 -    10 (     51)    10.0 s     Finished<brk>
      <cont> | pimpleFoam | ./CavityPyFoam4TheLazy
```

Adding custom output to listings

- Data from the `pickledData` can be added to the listing
  - For instance "show me the Courant numbers of all the cases"
- User has to specify
  - The name of the data column
  - How to find the value
    - Dictionary keys separated py `::`

Maximum Co

```
pyFoamListCases.py . --custom-data=maxCo=analyzed::Courant::max
 Warning in /Users/bgschaid/Development/OpenFOAM/Python/PyFoam/bin/pyFoamListCases.py : <brk>
       <cont>Parameter '--solver-name-for-custom-data' should be set if '--custom-data' is <brk>
       <cont>used
                    mtime |          hostname | first - last (nrSteps) nowTime s       state<brk>
                          <cont> |      solver |       maxCo_ | name
-------------------------------------------------------------------------------------------------
      <cont>
Fri Jun 24 00:55:22 2016 | bgs-cool-greybook |     0 -  2.6 (     14)     2.5 s Interrupted<brk>
      <cont> |   pisoFoam | 0.2691075704 | ./testCavity
Fri Jun 24 01:02:26 2016 | bgs-cool-greybook |     0 -   10 (     51)    10.0 s    Finished<brk>
      <cont> | pimpleFoam | 0.9404536707 | ./CavityPyFoam4TheLazy
```

Go three slides back to see why `analyzed::Courant::max` finds the Courant number

Outline

Outline

ParaView

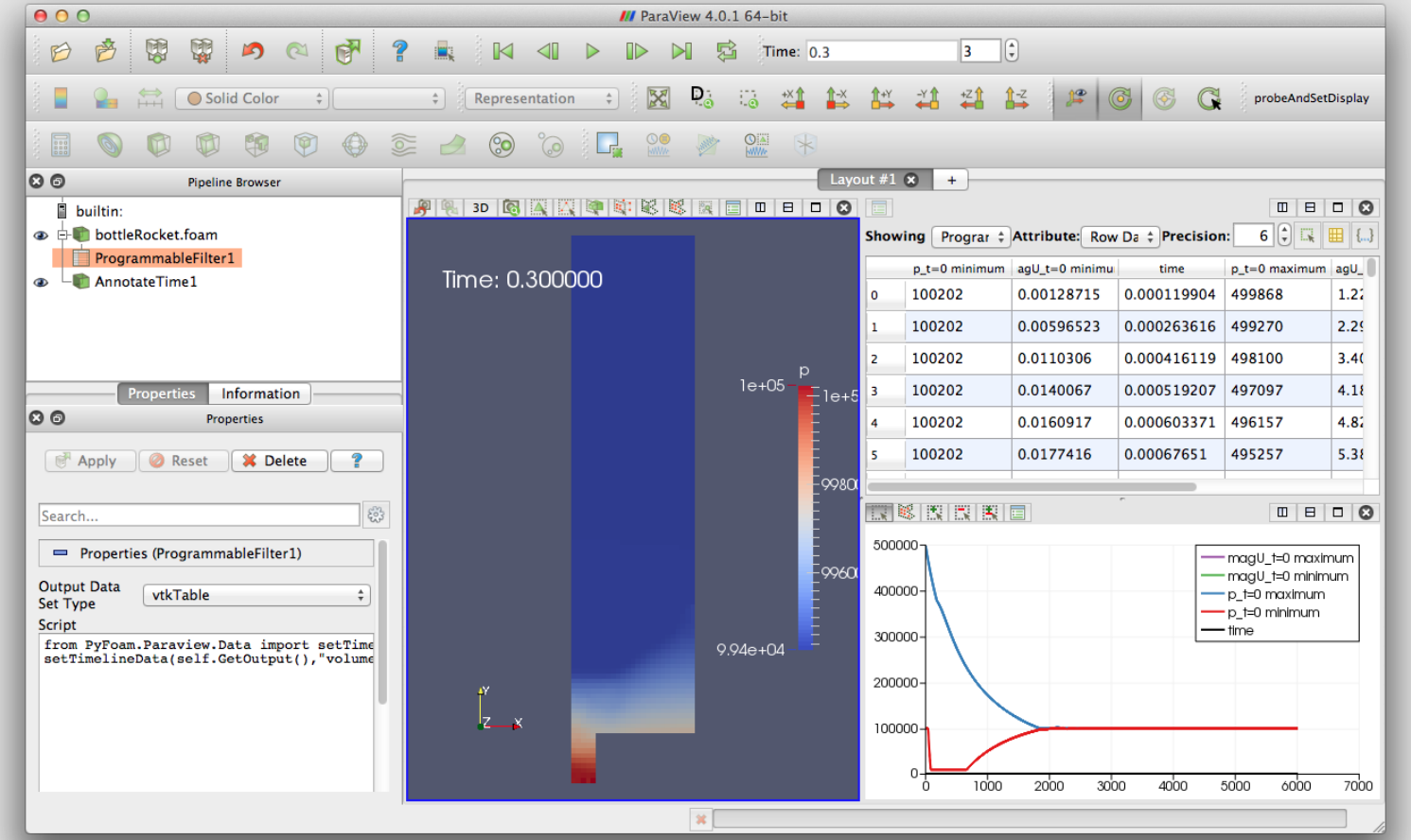
- Developed and published by Kitware
  - Based on the VTK-library
    - Also by KitWare
    - Basis for other Postprocessors
- Very flexible filter system
- Has built-in Python-bindings
  - Allows programming filters in Python
  - Built on tried and tested Python-libraries: `numpy` and friends
  - Hint: not enabled in older OpenFOAM-versions

ParaView support in PyFoam

- Not demonstrated in this presentation
  - "Finding" the case location and reading files:
    - Example: read locations of `probes` from `controlDict` and add to to the visualization
    - Example: read `g` and show the vector pointing "down"
  - Fining the time-directory for the current time
  - Drawing primitives
  - Reading data files and adding as tables (using the tools shown elsewhere in this presentation)
    - Not tested for some time (might fail)
    - Example: read `probes` fail and display it as well

Reading data into Paraview

Adding a line plot automatically



Outline

State files in ParaView

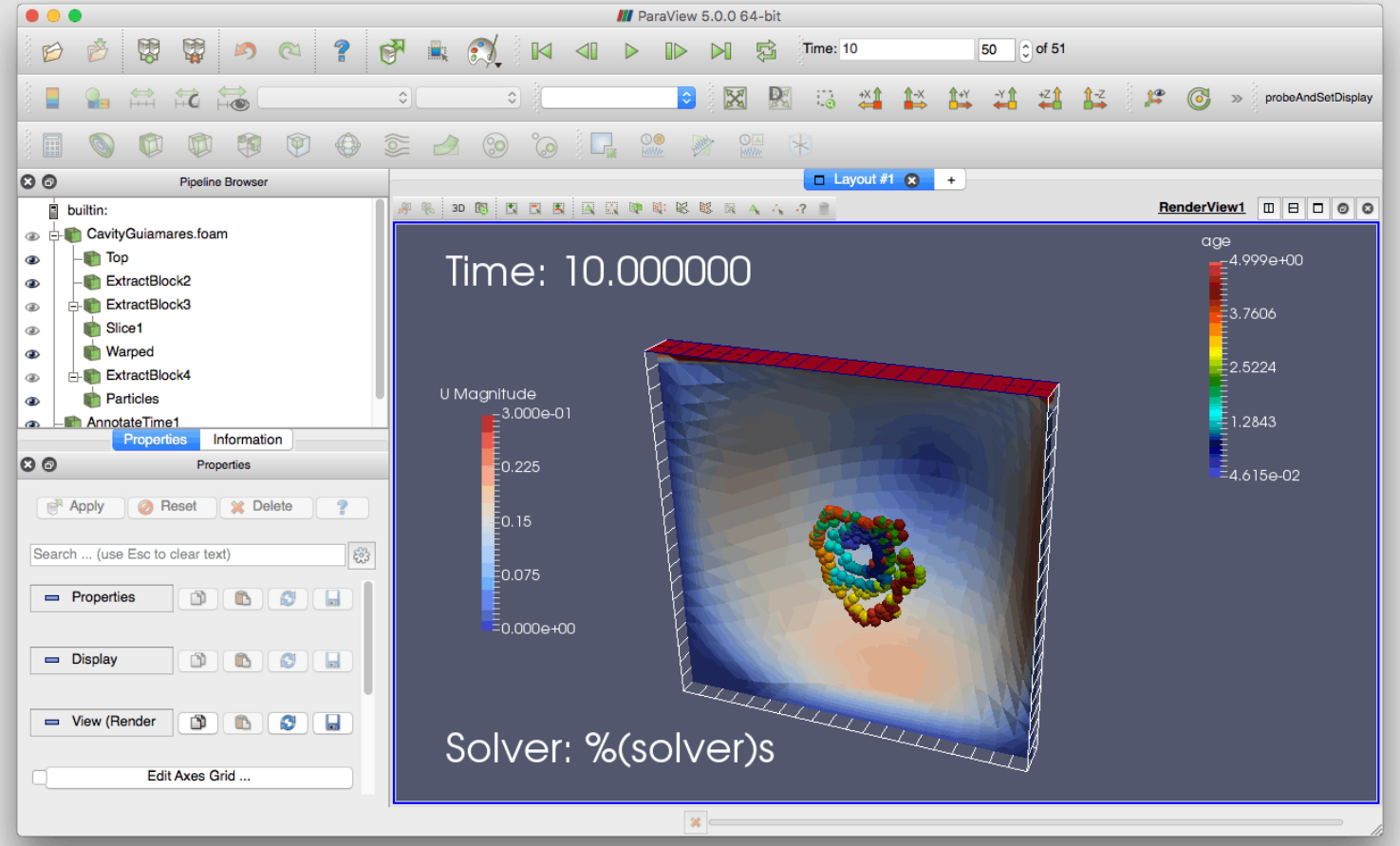
- Great time-saving feature of ParaView
  - Which now (== the last few years) works quite stable
- The way to work with it
  1. Do a complicated visualization
  2. Save it with `Save State`
  3. Close Paraview
  4. Copy state-file to another case
  5. Open Paraview
  6. Press `Load state` and select state-file
  7. Paraview is confused and asks for the case
  8. Do the same visualization with another case
- Saves a lot of time
  - But it can be even easier

pyFoamPVSnapshot.py

- Utility in pyFoam that needs three informations
  1. A state-file
  2. The case
  3. One or more times
- In return it does:
  1. Create a copy of the state-file
  2. Manipulate it to point to the case
  3. Load into a GUI-less version of Paraview (pvpython)
  4. Write pictures
- Can do a few other things
- This allows quickly creating reference pictures for similar cases
  - Which look exactly the same

Creating an "interesting" state

Warped plane with particles



Description of the visualization

- Plane-cut through the geometry
  - "Warped" with the value of the stream-function
  - Colored with the velocity
- Moving wall
  - Colored with the velocity
- Particles
  - "Blown up" to be spheres
  - Colored with they age

Nothing too fancy. But still needs 5 minutes of your life-time to click this together

String interpolations

- You'll have noticed `%(solver)s` in the text
  - This is Python string-interpolation
  - Means: "take value of variable `solver` and insert it as string"
- This is a nice way to label the pictures you generate
- Ways variables are defined
  - PyFoam automatically adds a variable `casename`
  - if the case was set up with `pyFoamPrepareCase.py` then the parameters used there will be available with `--add-prepare-case-parameters`
  - User can specify a dictionary with values with the `--replacements-parameter`
- `--list-replacements` lists all available values

Making a snapshot

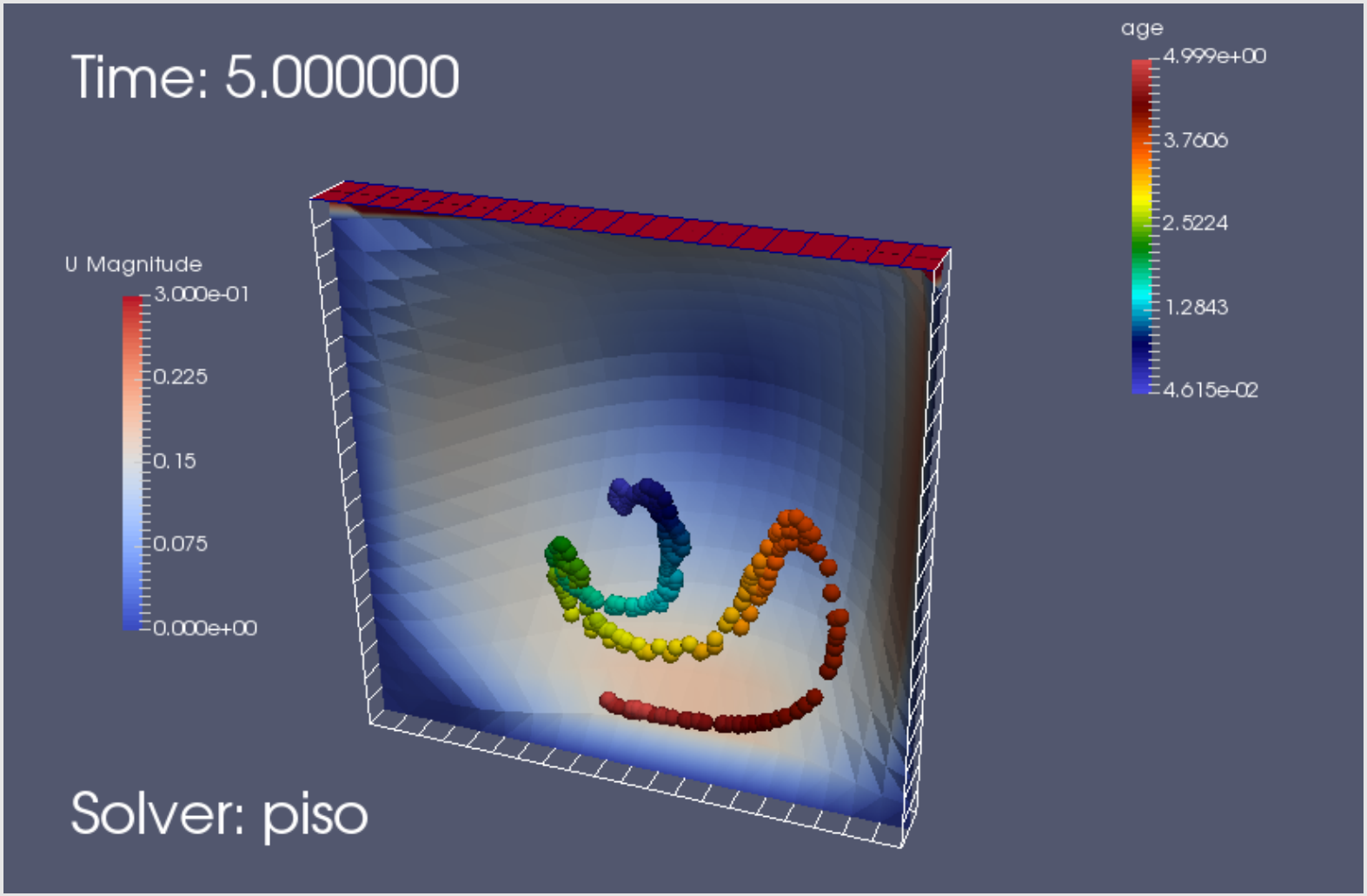
Making one picture

```
> pyFoamPVSnapshot.py --time=5 --state=warpedAndParcels.pvsm . --add-prepare --list-replace
```

- Starts the utility
- Does not open a window
  - If Paraview was compiled with Offscreen-rendering
- Prints some confusing messages
  - But also the list of the parameters
- Writes a snapshot

The snapshot

Plain snapshot



Recoloring stuff

- Every source in Paraview has a unique name
  - List of the sources can be printed with `--get-sources-list` if you don't want to go to the GUI
- The color for each source can be changed with the `--colors-for-filters`-option
  - Value is a Python dictionary with

    key the name of the source

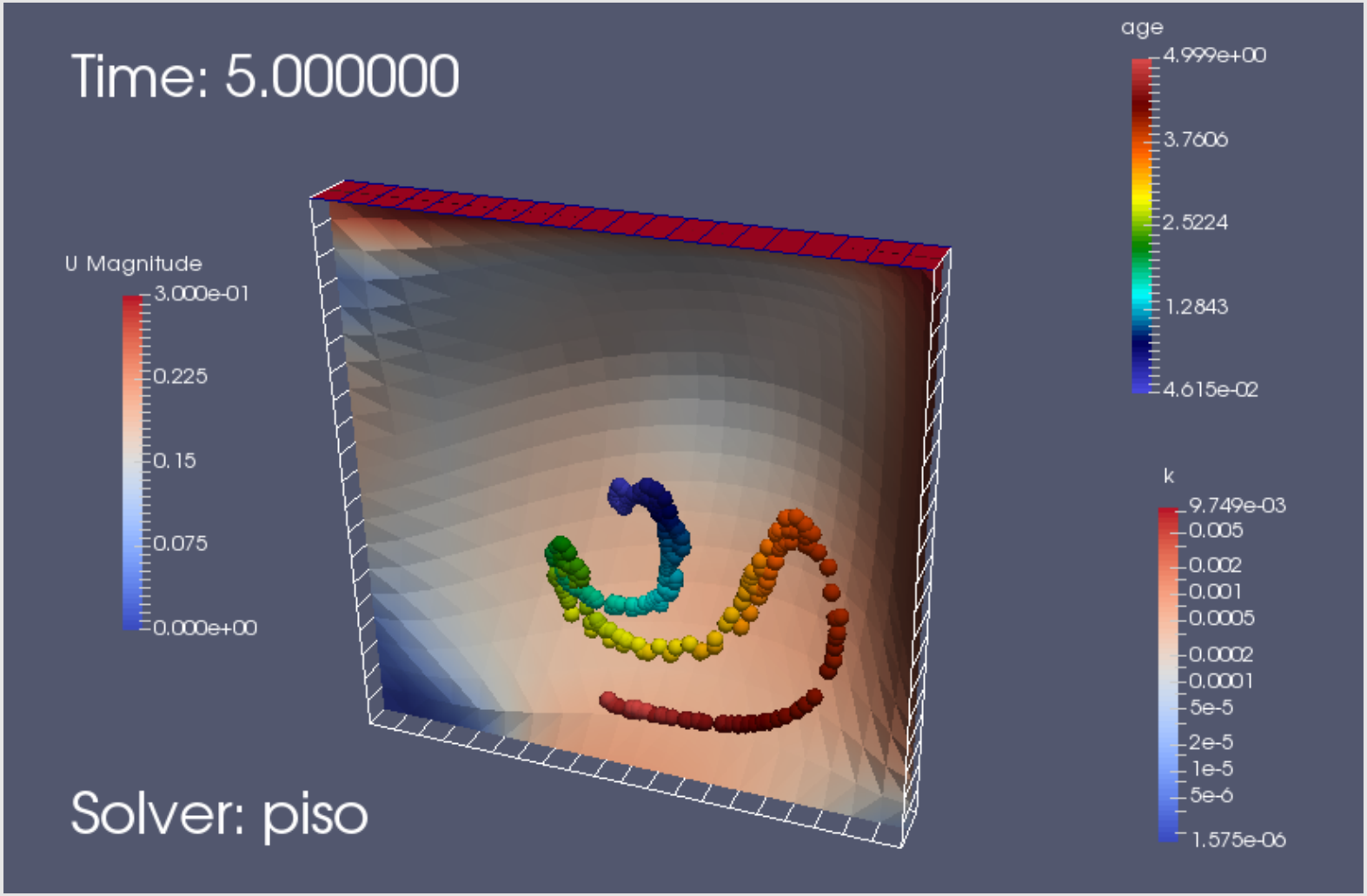
    value a tuple with the association (`POINTS`, `CELL`) and the field name

Recoloring the warped plane

```
> pyFoamPVSnapshot.py --time=5 --state=warpedAndParcels.pvsm . --add-prepare --list-replace<brk>
      <cont> --colors-for-filters="{'Warped':('POINTS','k')}"
```

Recolored plane

Plane with k



Beware

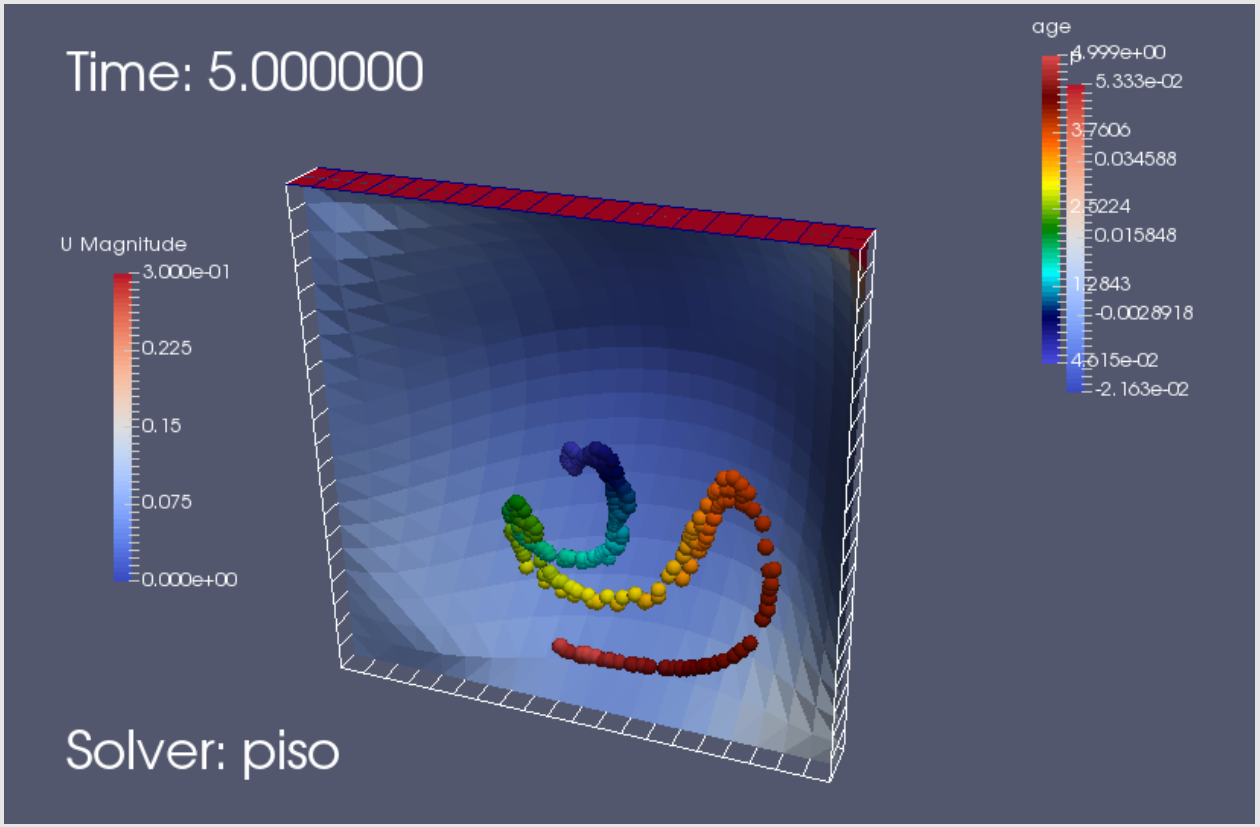
- The utility can only work with what is in the State-file
  - If you never set the position of a colorbar the colorbar will be somewhere
- The boundaries in the current case have to be similar to the boundaries in the original case
  - Number of boundaries and order are important
    - It is not sufficient for a boundary to be named `inlet`. It has to be the third boundary (just an example)
    - Not the utilities fault. This also fails for loading state files through the GUI

Wrong colorbar

```
> pyFoamPVSnapshot.py --time=5 --state=warpedAndParcels.pvsm . --add-prepare --list-replace<brk>
      <cont> --colors-for-filters="{'Warped':('CELLS','p')}"
```

Pressure bar missing

Recolored bad



Obviously we didn't place the colorbar for p in the state-files

Other Features

- Choose magnifications with `--magnification`
  - Make *huuuuge* pictures for billboards
- Looks at the case and switches between `Decomposed` and `Reconstructed` for the built-in Paraview reader
  - Basically uses the one that has more timesteps
  - But you can force it to use one of the two options
  - Independent of what was used to generate the state-file
- `--geometry-type` writes the geometry information in a number of formats
  - Useful if you want to pimp up the visualization in Blender or similar

Outline

Outline

What are timelines

- What are *timelines* in my definition:
  - A sequence of tuples of which one rises monotonically.
    - We'll call the rising one: time
- Usually we have these timelines in text files:
  - One tuple per line
  - The "time" first
- Timelines are written by:
  - Standard function objects and utilities like `probes`, `forces` etc
  - other programs like `swak4Foam`

Timelines are messy

- What is there?
  - Some function objects write a first line starting with # (the "header") with information about the tuples that are going to come
    - `probes` writes the coordinates over several lines. But that is different for versions of OpenFOAM (and of course foam-extend)
    - Sometimes there is no header at all
- Which type does the data have?
  - The "header" doesn't say if the value is a vector
    - And sometimes vectors are written as `"1 2 3"` and sometimes as `(1 2 3)`
- Times differ
  - Solvers might have different timesteps
    - Makes it har to compare $t = 23.51$ with another solver because that one only has $t = 23.5$ and $t = 23.52$
- PyFoam tries to help with these problems

The pyFoamTimelinePlot.py-utility

- The purpose of this utility is: plotting timelines
  - Also: comparing timelines from different cases
- It tries to handle some of the situations described above
- The utility always needs two pieces of information:
  1. The name of the case (as a parameter)
  2. The actual data directory in that as the option `--directory`
- The utility does not generate the plots. It only generates the Gnuplot-commands to generate the plots
  - If Gnuplot can do one thing then it is plotting

Gnuplot

- `http://www.gnuplot.info/` is a popular OpenSource plotting program
  - Quite flexible
    - Because it has no GUI but is controlled by text commands
    - For some *flexible* means *confusing*
  - Very fast
  - The plots have a certain *vintage charm*
- PyFoam uses it for live plotting
  - Because it is fast

Typical Gnuplot-session

```
> plot "datafile.txt" using 1:3
> set title "My data"
> set term png
> set output "thePicture.png"
> replot
```

Checking what is there

- We'll work with one of the directories that we prepared with `runAll.sh`
- First we'll check which data is actually there
- The `--info` tells us which things it finds in the directory

Write Times	times when the timelines started writing (usually 0 but may be more due to restarts)
Used Time	Which of the `Write Times` it is going to use
Fields	Values it found in the directory (files). If it identified vector fields it tells us so
Positions	Columns in the header
Time range	How "long" is the `Used Time`

Just checking

```
> pyFoamTimelinePlot.py . --info --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction
Write Times     :  ['0']
Used Time       :  0
Fields          :  ['minStreamFunction']
Positions       :  ['min']
Time range      :  (0.005, 10.0)
```

Basic plotting - lines

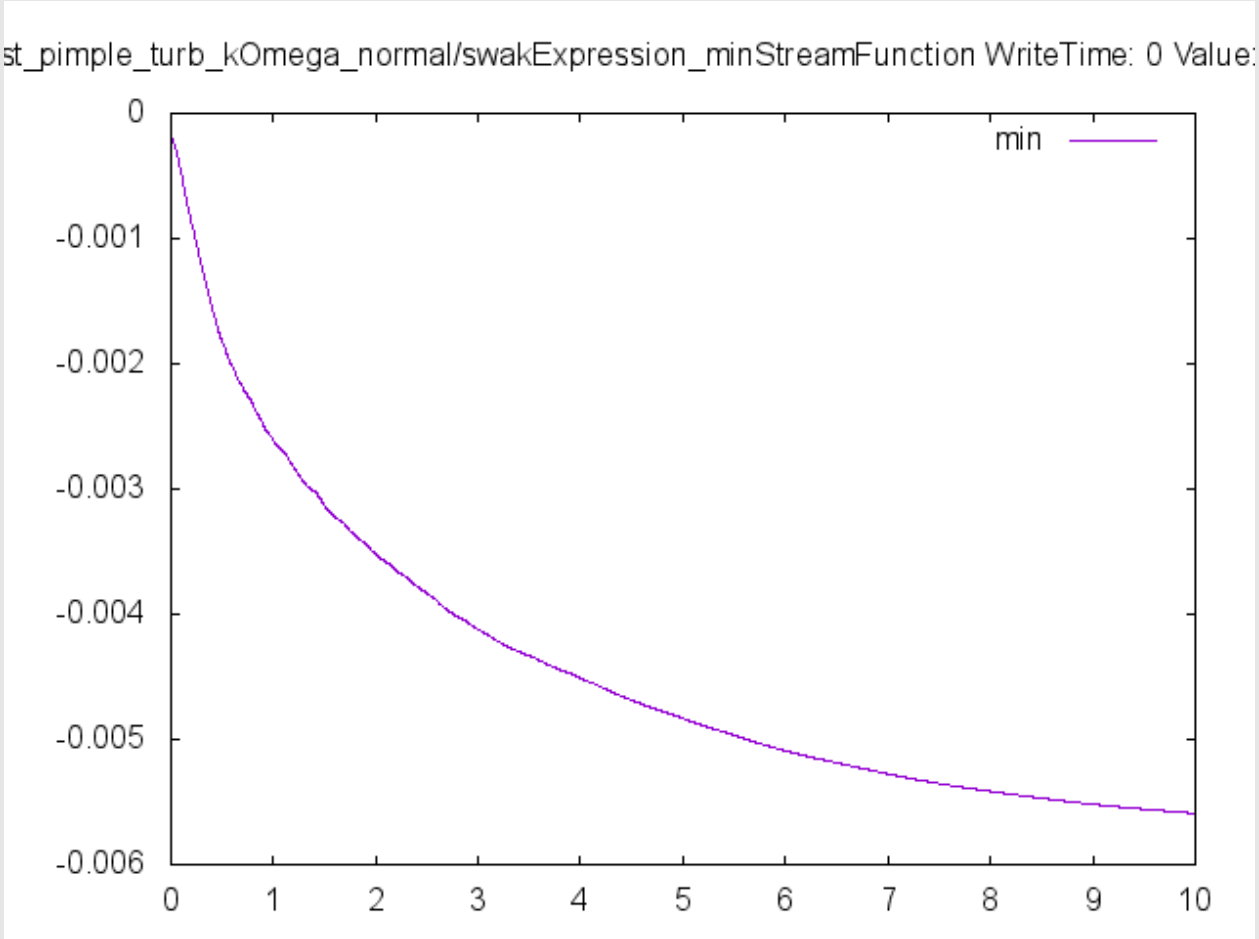
- To plot a `--basic-mode` has to be chosen. Usually `lines`
- We only have one `field`, `position` and `time` so we don't have to choose
  - Otherwise `all` will be used
- The utility prints the Gnuplot-commands to the terminal. The options are
  1. Copy/pasting and editing
  2. Piping to the command line

First plot

```
> pyFoamTimelinePlot.py . --basic-mode=lines --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction
set term png nocrop enhanced
set xrange [0.005:10]
set output "<brk>
      <cont>post_pimple_turb_kOmega_normal_swakExpression_minStreamFunction_writeTime_0_Value_minStreamFun
      <cont>.png"
set title "Directory: post\\_pimple\\_turb\\_kOmega\\_normal/swakExpression\\<brk>
      <cont>_minStreamFunction WriteTime: 0 Value: minStreamFunction"
plot  "./post_pimple_turb_kOmega_normal/swakExpression_minStreamFunction/0/<brk>
      <cont>minStreamFunction" using 1:2 title "min" with lines
> pyFoamTimelinePlot.py . --basic-mode=lines --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction | gnuplot
```

The result

Falling Stream-function



The generated output

- The output is designed to generate a very simple `PNG`
  - If you want something more elaborate:
    - hand-edit it
    - let a program like `sed` edit it for you
- Names tend to be very long
  - This makes sure that you don't have to guess
  - And with *Tab*-completion this shouldn't be a problem
- Only problem is that annotations are sometimes have the wrong proportions for the plot
  - But we're lazy. Not *art critics*

Selecting the time

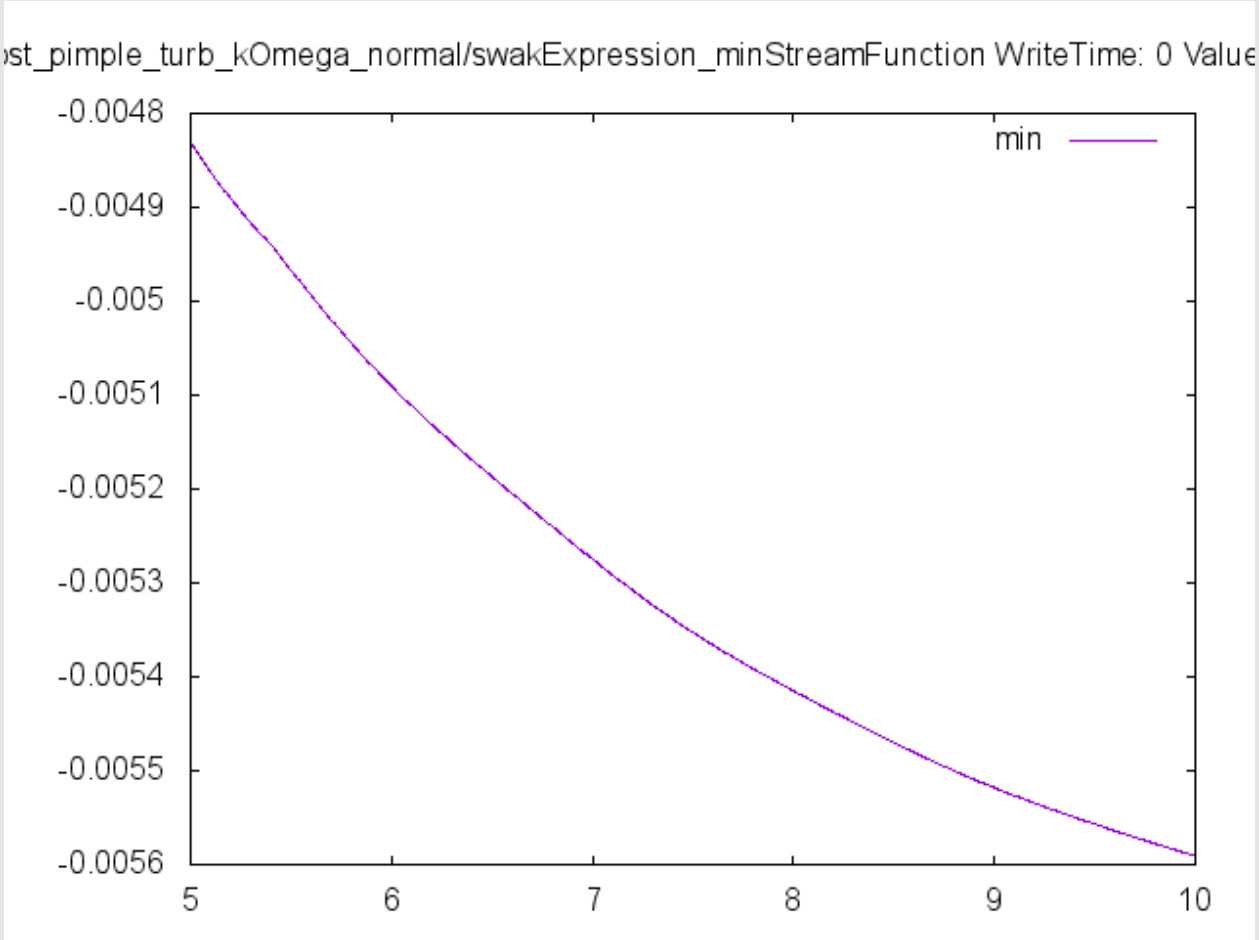
- Two options to select the time range:
  - min-time for the start of the plot
  - max-time for the end

Starting at half-time

```
> pyFoamTimelinePlot.py . --basic-mode=lines --min-time=5 --dir=<brk>
      <cont>post_pimple_turb_kOmega_normal/swakExpression_minStreamFunction
set term png nocrop enhanced
set xrange [5:10]
set output "<brk>
      <cont>post_pimple_turb_kOmega_normal_swakExpression_minStreamFunction_writeTime_0_Value_minStreamFun
      <cont>.png"
set title "Directory: post\\_pimple\\_turb\\_kOmega\\_normal/swakExpression\\<brk>
      <cont>_minStreamFunction WriteTime: 0 Value: minStreamFunction"
plot  "./post_pimple_turb_kOmega_normal/swakExpression_minStreamFunction/0/<brk>
      <cont>minStreamFunction" using 1:2 title "min" with lines
```

The result

Only the second half



Collect lines

- Our dataset doesn't have many fields and positions
  - For instance: `probe` values of `T` and `p`
    - Do we want a plot for each position of `p` and `T` together
    - Or one plot for `T` at all positions
- The `--collect-lines-by` option lets us choose

  fields All `T` in one plot. This is the default

  positions One plot for each position.
- Note on the `position` / `fields` nomenclature:
  - This makes sense for `probes`
    - It was developed for those
  - For other function objects this is a bit confusing
    - For instance: Temperature statistics on a number of patches
    - Patch names would be `fields`
    - `min` and `max` would be `positions`

Comparing two runs

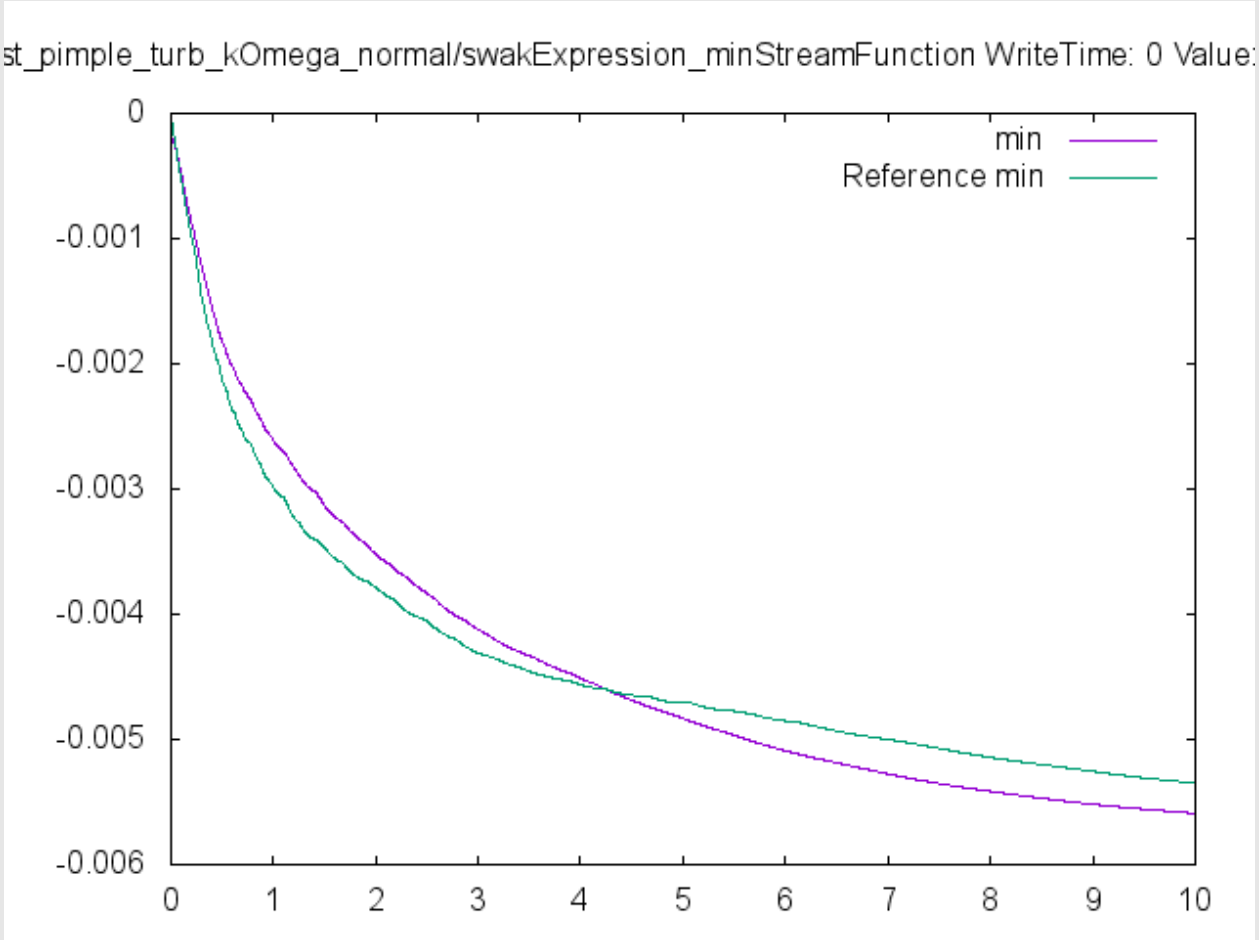
- Utility allows specification of either
  - reference-dir Different directory in the same case
  - reference-case Different case but same directory
- If the same data is present there it is plotted as well
  - Check with `--info` what is there

Reference plot

```
> pyFoamTimelinePlot.py . --basic-mode=lines --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction --field=minStreamFunction --reference-dir=<brk>
      <cont>post_ico_normal/swakExpression_minStreamFunction
set term png nocrop enhanced
set xrange [0.005:10]
set output "<brk>
      <cont>post_pimple_turb_kOmega_normal_swakExpression_minStreamFunction_writeTime_0_Value_minStreamFun
      <cont>.png"
set title "Directory: post\\_pimple\\_turb\\_kOmega\\_normal/swakExpression\\<brk>
      <cont>_minStreamFunction WriteTime: 0 Value: minStreamFunction"
plot  "./post_pimple_turb_kOmega_normal/swakExpression_minStreamFunction/0/<brk>
      <cont>minStreamFunction" using 1:2 title "min" with lines  ,  "./post_ico_normal/<brk>
      <cont>swakExpression_minStreamFunction/0/minStreamFunction" using 1:2 title "Reference<brk>
      <cont> min" with lines
```

Different results

Stream-function in two different cases



Checking the numbers

- These results look very different
  - Not surprising: different solvers. And: one has turbulence
- To compare we can use
  - –metrics statistics on the current line
    - Average and weighted average are the same for non-varying timestepping
  - –compare difference to the reference
    - Pro-tip: the reference data might be measurement data. If you believe in validation

Comparing the numbers

```
> pyFoamTimelinePlot.py . --basic-mode=lines --dir=post_pimple_turb_kOmega_normal/swakExpression_minStreamFunction <brk>
      <cont>--field=minStreamFunction --reference-dir=post_ico_normal/swakExpression_minStreamFunction --compare --<brk>
      <cont>metric
Metrics for minStreamFunction on min index 0 (Path: ./post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction/0/minStreamFunction )
  Min                 : -0.005590765609
  Max                 : -0.000147963669
  Average             : -0.0044114939011
  Weighted average    : -0.00441167542064
  Time Range          : 0.005 10.0
Comparing minStreamFunction on min index 0 (path: ./post_pimple_turb_kOmega_normal/swakExpression_minStreamFunction<brk>
      <cont>/0/minStreamFunction )on original data points
  Max difference      : 0.00039955739
  Average difference  : 0.000225141188701
  Weighted average    : 0.000225154809118
Data size: 2001 Reference: 2001
```

Similar solvers are better

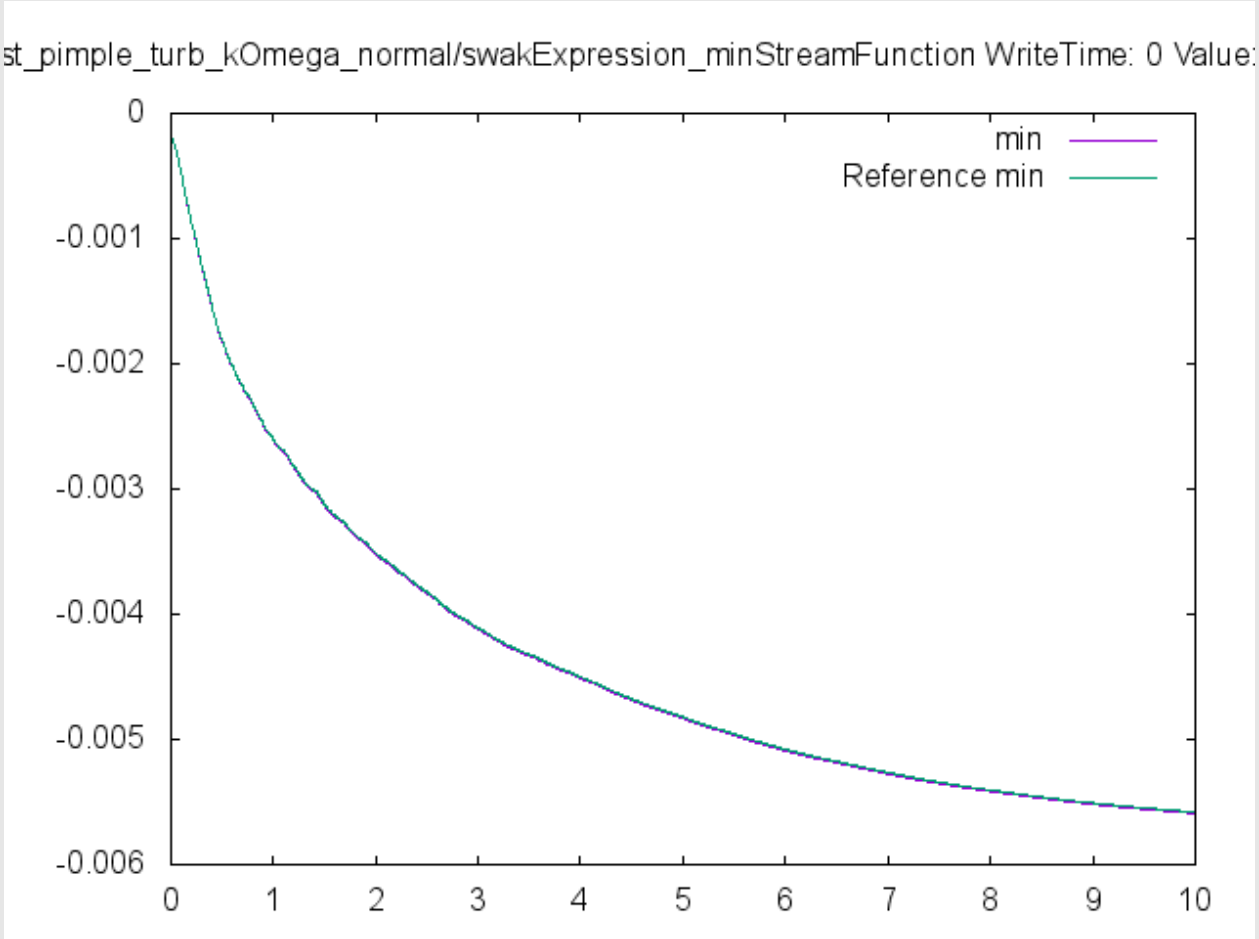
- I know: the command lines are very long
  - But the command line history of the shell is your friend
  - And it is easy to adapt things
- Lets look at the difference to the compressible solver
  - But same turbulence model

Smaller differences

```
> pyFoamTimelinePlot.py . --basic-mode=lines --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction --field=minStreamFunction --reference-dir=<brk>
      <cont>post_rhoPimple_turb_kOmega_normal/swakExpression_minStreamFunction --compare --<brk>
      <cont>metric
Metrics for minStreamFunction on min index 0 (Path: ./post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction/0/minStreamFunction )
  Min                 : -0.005590765609
  Max                 : -0.000147963669
  Average             : -0.0044114939011
  Weighted average    : -0.00441167542064
  Time Range          : 0.005 10.0
Comparing minStreamFunction on min index 0 (path: ./post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction/0/minStreamFunction )on original data points
  Max difference      : 1.3012487e-05
  Average difference  : 1.06563696608e-05
  Weighted average    : 1.06594625107e-05
Data size: 2001 Reference: 2001
```

Almost the same

Compressible gives almost the same result



Working with vectors

- Vector fields are handled differently
- `--info` lists them
- Usually they are recalculated to absolut values
- But you can select single components with `--vector-mode`

Checking for vectors

```
> pyFoamTimelinePlot.py .  --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunctionPos --info
Write Times     :  ['0']
Used Time       :  0
Fields          :  ['minStreamFunctionPos'] Vectors:  ['minStreamFunctionPos']
Positions       :  ['min']
Time range      :  (0.005, 10.0)
```

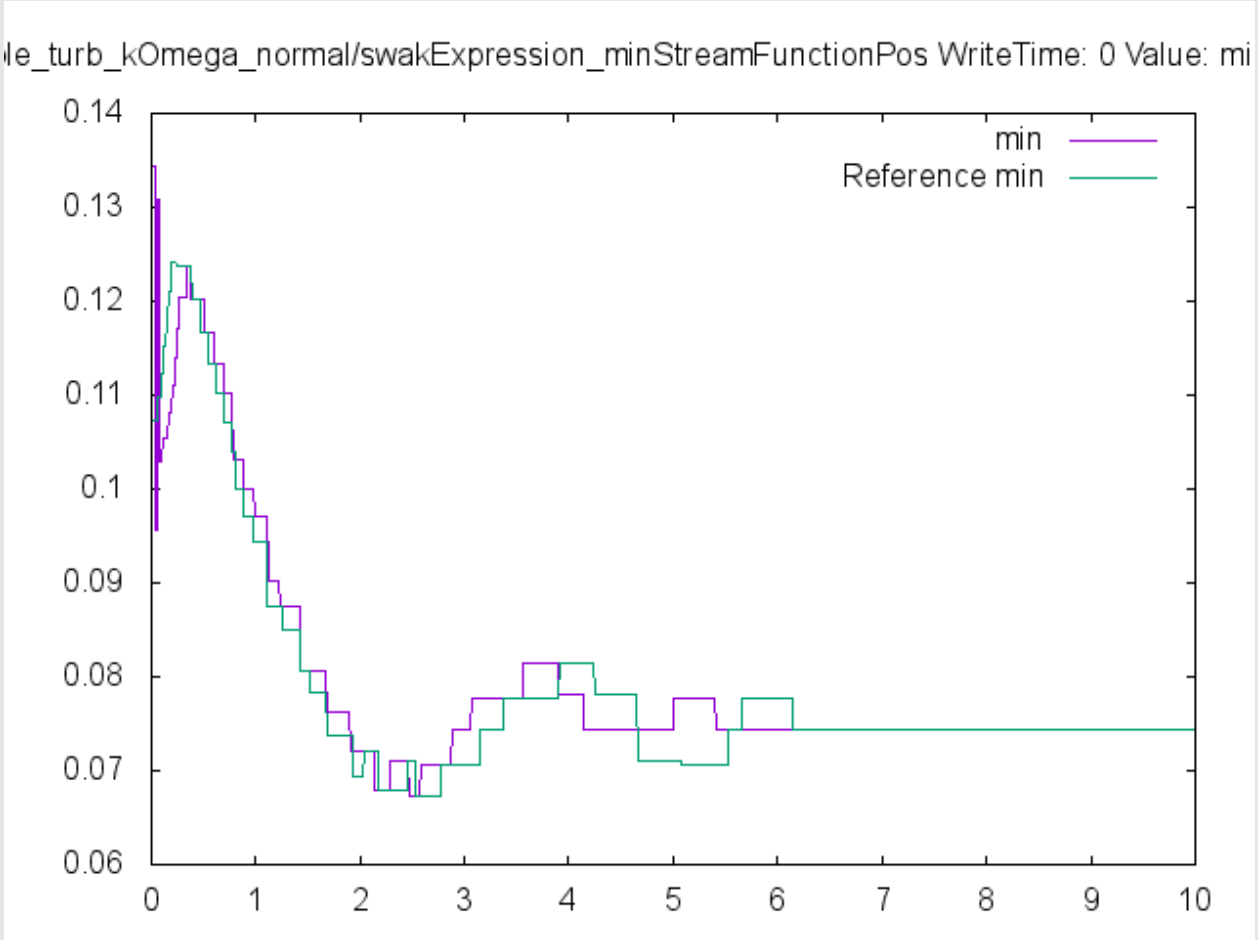
Plotting the positions

First absolute. Then only x

```
> pyFoamTimelinePlot.py . --dir=postProcessing/swakExpression_minStreamFunctionPos --field=<brk>
      <cont>minStreamFunctionPos --basic-mode=lines --reference-dir=post_ico_normal/<brk>
      <cont>swakExpression_minStreamFunctionPos
set term png nocrop enhanced
set xrange [0.00588235:10]
set output "<brk>
      <cont>postProcessing_swakExpression_minStreamFunctionPos_writeTime_0_Value_minStreamFunctionPos_mag<
      <cont>.png"
set title "Directory: postProcessing/swakExpression\\_minStreamFunctionPos WriteTime: 0 <brk>
      <cont>Value: minStreamFunctionPos\\\_mag"
plot  "< tr <./postProcessing/swakExpression_minStreamFunctionPos/0/minStreamFunctionPos -d<brk>
      <cont> '()'" using 1:(sqrt($2*$2+$3*$3+$4*$4)) title "min" with lines  ,  "< tr <./<brk>
      <cont>post_ico_normal/swakExpression_minStreamFunctionPos/0/minStreamFunctionPos -d <brk>
      <cont>'()'" using 1:(sqrt($2*$2+$3*$3+$4*$4)) title "Reference min" with lines
> pyFoamTimelinePlot.py . --vector-mode=x --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunctionPos --field=minStreamFunctionPos --basic-mode=<brk>
      <cont>lines --reference-dir=post_ico_normal/swakExpression_minStreamFunctionPos
set term png nocrop enhanced
set xrange [0.005:10]
set output "<brk>
      <cont>post_pimple_turb_kOmega_normal_swakExpression_minStreamFunctionPos_writeTime_0_Value_minStream
      <cont>.png"
set title "Directory: post\\_pimple\\_turb\\_kOmega\\_normal/swakExpression\\<brk>
      <cont>_minStreamFunctionPos WriteTime: 0 Value: minStreamFunctionPos\\\_x"
plot  "< tr <./post_pimple_turb_kOmega_normal/swakExpression_minStreamFunctionPos/0/<brk>
      <cont>minStreamFunctionPos -d '()'" using 1:2 title "min" with lines  ,  "< tr <./<brk>
      <cont>post_ico_normal/swakExpression_minStreamFunctionPos/0/minStreamFunctionPos -d <brk>
      <cont>'()'" using 1:2 title "Reference min" with lines
```

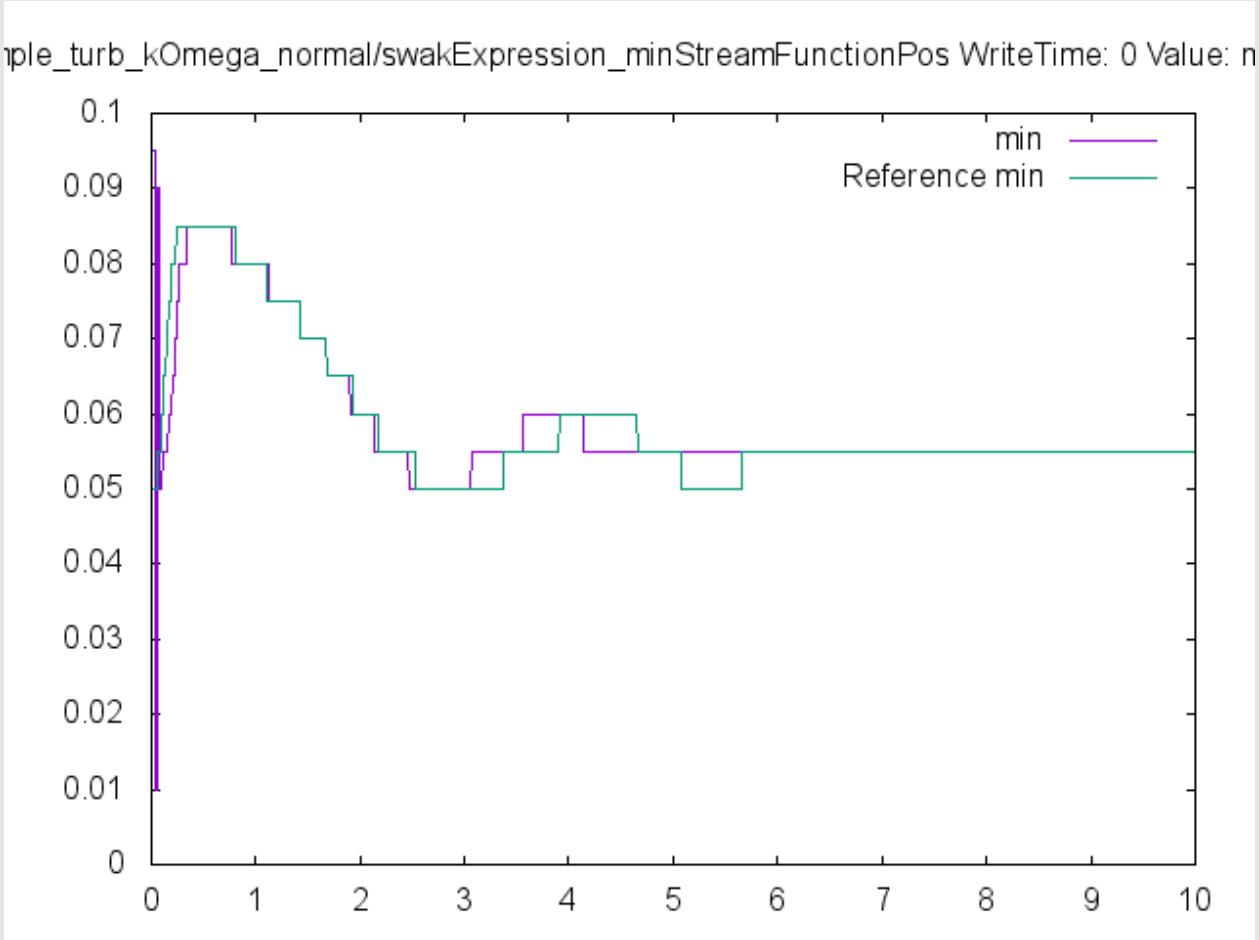
Absolute length of the vector

Distance from $(0, 0)$



x-component

x coordinate of the minimum



Different time resolutions

- Often cases with differing time-steps are compared
  - Plotting is not a problem: Gnuplot doesn't care
  - `-comppare` handles this by interpolating (linearly) between the data sets

Compare to the case with variable timestep $Co = 0.5$

```
> pyFoamTimelinePlot.py . --basic-mode=lines --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction --field=minStreamFunction --reference-dir=<brk>
      <cont>post_pimple_turb_kOmega_Co_0.500000_normal/swakExpression_minStreamFunction --<brk>
      <cont>compare --metric
Metrics for minStreamFunction on min index 0 (Path: ./post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction/0/minStreamFunction )
  Min                 : -0.005590765609
  Max                 : -0.000147963669
  Average             : -0.0044114939011
  Weighted average    : -0.00441167542064
  Time Range          : 0.005 10.0
Comparing minStreamFunction on min index 0 (path: ./post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction/0/minStreamFunction )on original data points
  Max difference      : 7.270348e-05
  Average difference  : 4.09671535934e-05
  Weighted average    : 4.09489994811e-05
Data size: 2001 Reference: 1175
```

Outline

sample output

- The sample utility and functionObject write field values as a function of space
  - One file per "line"
    - Separate files for scalars and vectors
  - One directory per time
- File names determine the values that are in it
  - If specified field values are missing then you can only know from the filename
    - If the field is not present in the first time step then it is only missing there
- The pyFoamSamplePlot.py utility tries to make sense of all of this
  - Also: different resolutions

Looking at the sample data

```
> post_ico_normal/sample/10
lineX_U.xy          lineX_streamCell.xy lineY_U.xy          lineY_streamCell.xy
> ls post_pimple_turb_kOmega_normal/sample/10/
lineX_U.xy                  lineY_U.xy
lineX_k_omega_streamCell.xy lineY_k_omega_streamCell.xy
```

What does pyFoamSamplePlot.py recognize?

- The utility needs the same things as pyFoamTimelinePlot.py:
  1. The case
  2. The --directory
- It also has a --info option

Getting info

```
> pyFoamSamplePlot.py . --dir=post_ico_normal/sample --info
Times :  ['0.2', '0.4', '0.6', '0.8', '1', '1.2', '1.4', '1.6', '1.8', '2', '2.2', '2.4', <brk>
      <cont>'2.6', '2.8', '3', '3.2', '3.4', '3.6', '3.8', '4', '4.2', '4.4', '4.6', '4.8', <brk>
      <cont>'5', '5.2', '5.4', '5.6', '5.8', '6', '6.2', '6.4', '6.6', '6.8', '7', '7.2', <brk>
      <cont>'7.4', '7.6', '7.8', '8', '8.2', '8.4', '8.6', '8.8', '9', '9.2', '9.4', '9.6', <brk>
      <cont>'9.8', '10']
Lines :  ['lineX', 'lineY']
Fields:  ['U', 'streamCell']
> pyFoamSamplePlot.py . --dir=post_pimple_turb_kOmega_normal/sample --info
Times :  ['0.2', '0.4', '0.6', '0.8', '1', '1.2', '1.4', '1.6', '1.8', '2', '2.2', '2.4', <brk>
      <cont>'2.6', '2.8', '3', '3.2', '3.4', '3.6', '3.8', '4', '4.2', '4.4', '4.6', '4.8', <brk>
      <cont>'5', '5.2', '5.4', '5.6', '5.8', '6', '6.2', '6.4', '6.6', '6.8', '7', '7.2', <brk>
      <cont>'7.4', '7.6', '7.8', '8', '8.2', '8.4', '8.6', '8.8', '9', '9.2', '9.4', '9.6', <brk>
      <cont>'9.8', '10']
Lines :  ['lineX', 'lineY']
Fields:  ['U', 'k', 'omega', 'streamCell']
```

Getting the sample plots

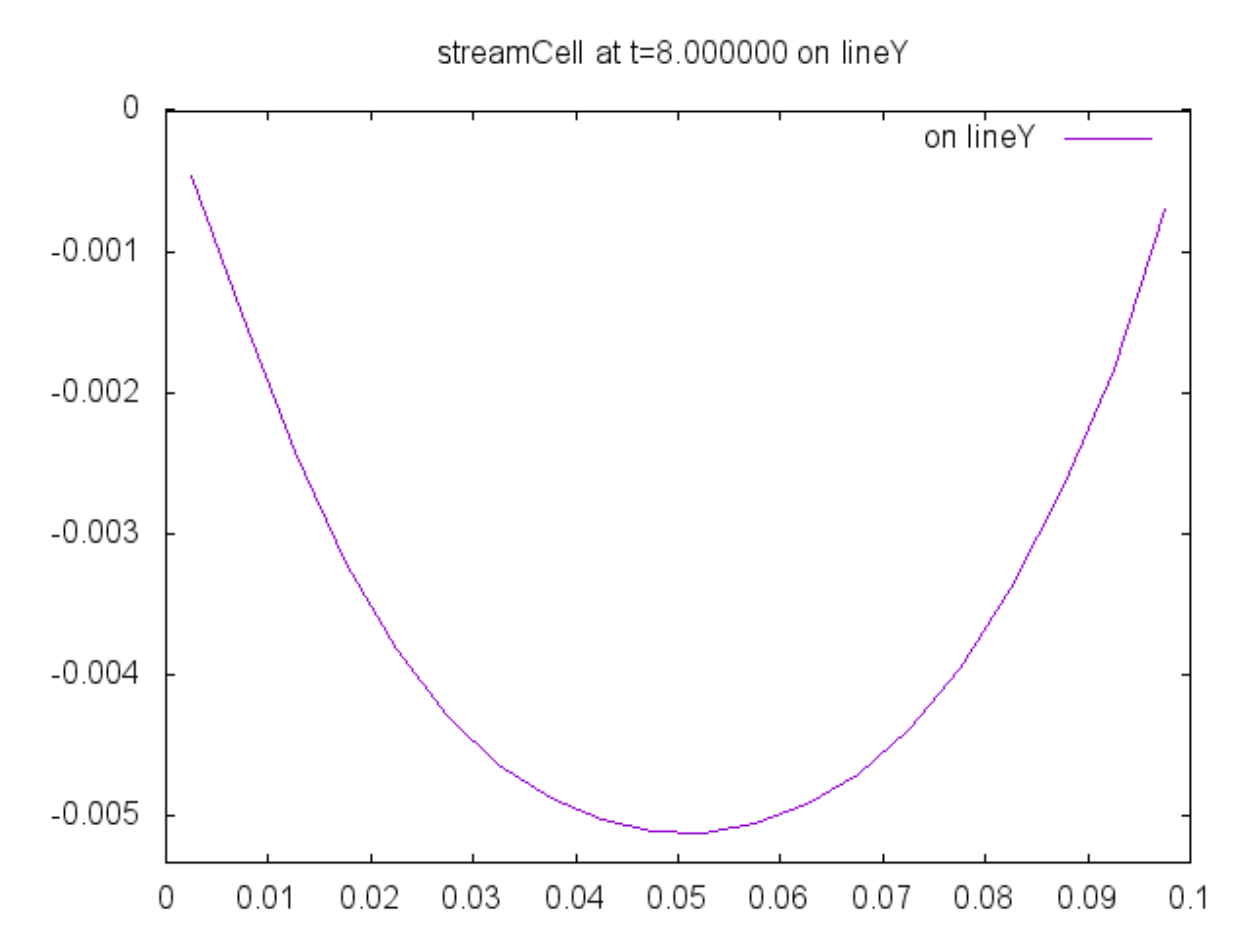
- To generate the plots the utility needs
  - –line the lines to plot
  - –time at which times to plot
  - –field the values to plot
- All can be specified more than once
  - If unspecified: all are used

Getting a lot of plots

```
> pyFoamSamplePlot.py . --dir=post_ico_normal/sample --field=streamCell
set term png
set output "post_ico_normal_sample_lineX_streamCell_t=0.200000.png"
set title "streamCell at t=0.200000 on lineX"
plot [][-0.00532797:-8.44374e-06] "./post_ico_normal/sample/0.2/lineX_streamCell.xy" using <brk>
      <cont>1:2 title " on lineX" with lines
set output "post_ico_normal_sample_lineY_streamCell_t=0.200000.png"
set title "streamCell at t=0.200000 on lineY"
plot [][-0.00532797:-8.44374e-06] "./post_ico_normal/sample/0.2/lineY_streamCell.xy" using <brk>
      <cont>1:2 title " on lineY" with lines
set output "post_ico_normal_sample_lineX_streamCell_t=0.400000.png"
set title "streamCell at t=0.400000 on lineX"
plot [][-0.00532797:-8.44374e-06] "./post_ico_normal/sample/0.4/lineX_streamCell.xy" using <brk>
      <cont>1:2 title " on lineX" with lines
set output "post_ico_normal_sample_lineY_streamCell_t=0.400000.png"
set title "streamCell at t=0.400000 on lineY"
plot [][-0.00532797:-8.44374e-06] "./post_ico_normal/sample/0.4/lineY_streamCell.xy" using <brk>
      <cont>1:2 title " on lineY" with lines
```

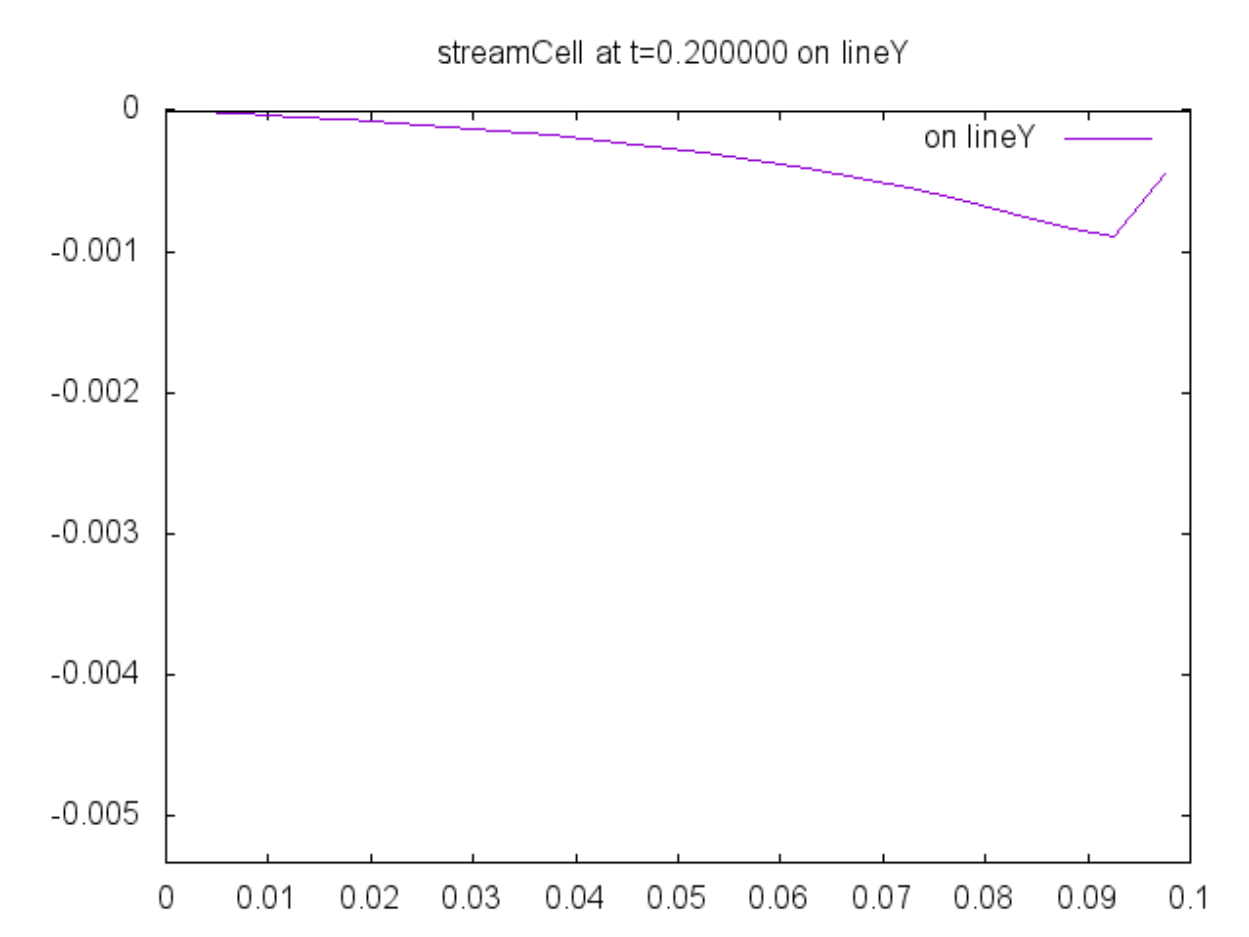
One plot (of 100)

Stream function almost converged



Another plot (of 100)

Stream function in the beginning



Scaling and animating

- You may have noticed that both plots have the same range
  - That's because PyFoam looked at the date before plotting
    - And produced the `[][-0.00532797:-8.44374e-06]` ranges
    - It's nice that PyFoam is not lazy
  - That allows using these plots for animations
    - To feed them into `ffmped` or `mencode` it is convenient to use `--index-instead-of-time` to have integer indices instead of times
- A number of options allows modifying the scaling

  –unscaled switches off scaling

  –scale-domain Scales all x-axes to the same length
    - Useful for animations and geometries with different widths

  There are other options (manually setting the ranges for instance)

Collecting lines

- Currently every time/field/line combination has a separate plot
- The --mode-parameter changes that. Possible values are:

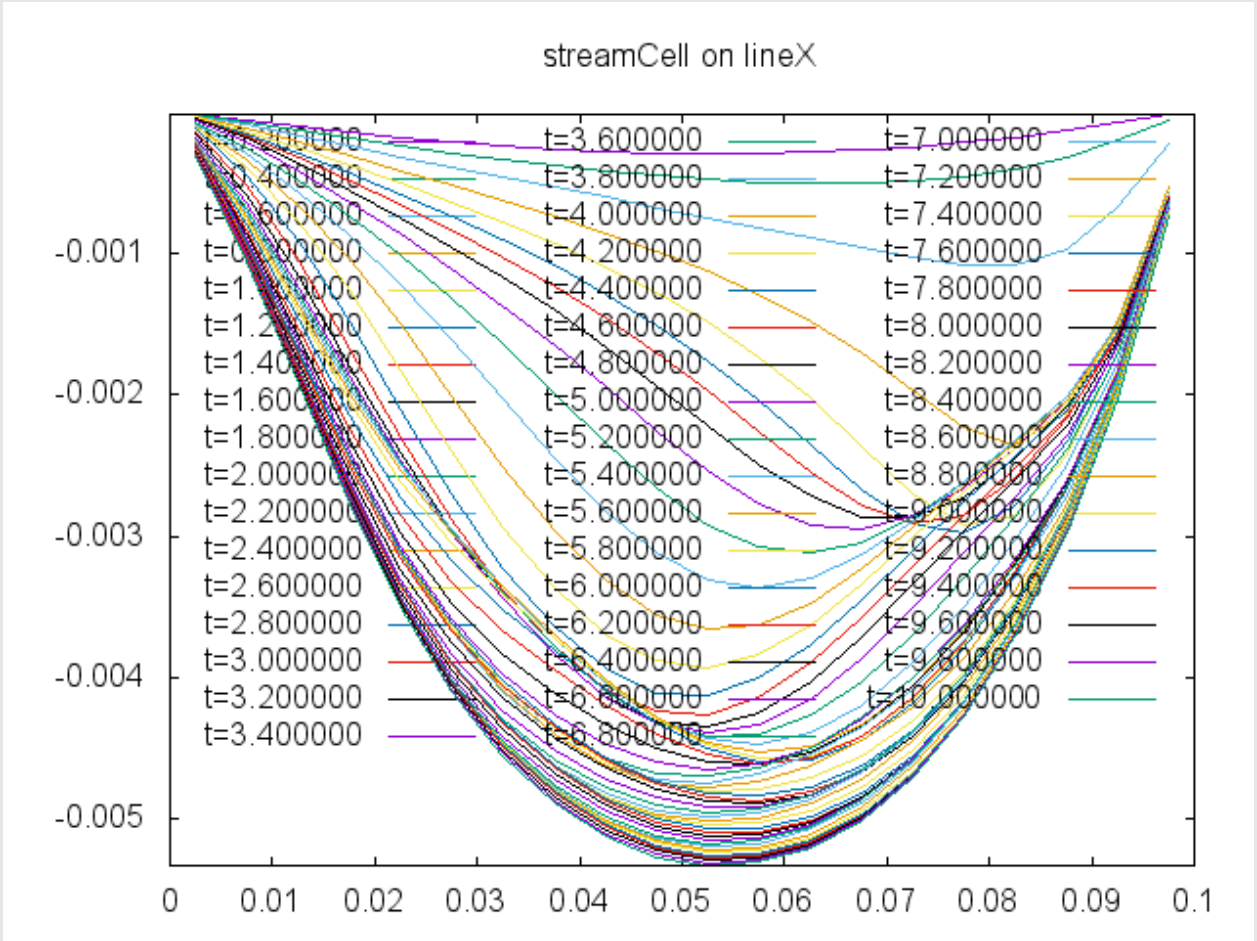
separate	this is the usual setting
timesInOne	all times are in one plot (but separate line and field)
fieldsInOne	all fields in one
linesInOne	all lines
complete	everything in one plot

Getting all times (lazy people wouldn't do that)

```
> pyFoamSamplePlot.py . --dir=post_ico_normal/sample --field=streamCell --line=lineX --mode<brk>
      <cont>=timesInOne
set term png
set output "post_ico_normal_sample_lineX_streamCell.png"
set title "streamCell on lineX"
plot [][-0.00532797:-1.99448e-05] "./post_ico_normal/sample/0.2/lineX_streamCell.xy" using <brk>
      <cont>1:2 title "t=0.200000" with lines , "./post_ico_normal/sample/0.4/<brk>
      <cont>lineX_streamCell.xy" using 1:2 title "t=0.400000" with lines , "./<brk>
      <cont>post_ico_normal/sample/0.6/lineX_streamCell.xy" using 1:2 title "t=0.600000" <brk>
      <cont>with lines , "./post_ico_normal/sample/0.8/lineX_streamCell.xy" using 1:2 title <brk>
      <cont>"t=0.800000" with lines , "./post_ico_normal/sample/1/lineX_streamCell.xy" using<brk>
      <cont> 1:2 title "t=1.000000" with lines , "./post_ico_normal/sample/1.2/<brk>
      <cont>lineX_streamCell.xy" using 1:2 title "t=1.200000" with lines , "./<brk>
      <cont>post_ico_normal/sample/1.4/lineX_streamCell.xy" using 1:2 title "t=1.400000" <brk>
```

All the times

Can you spot the correct colors?



Comparing cases

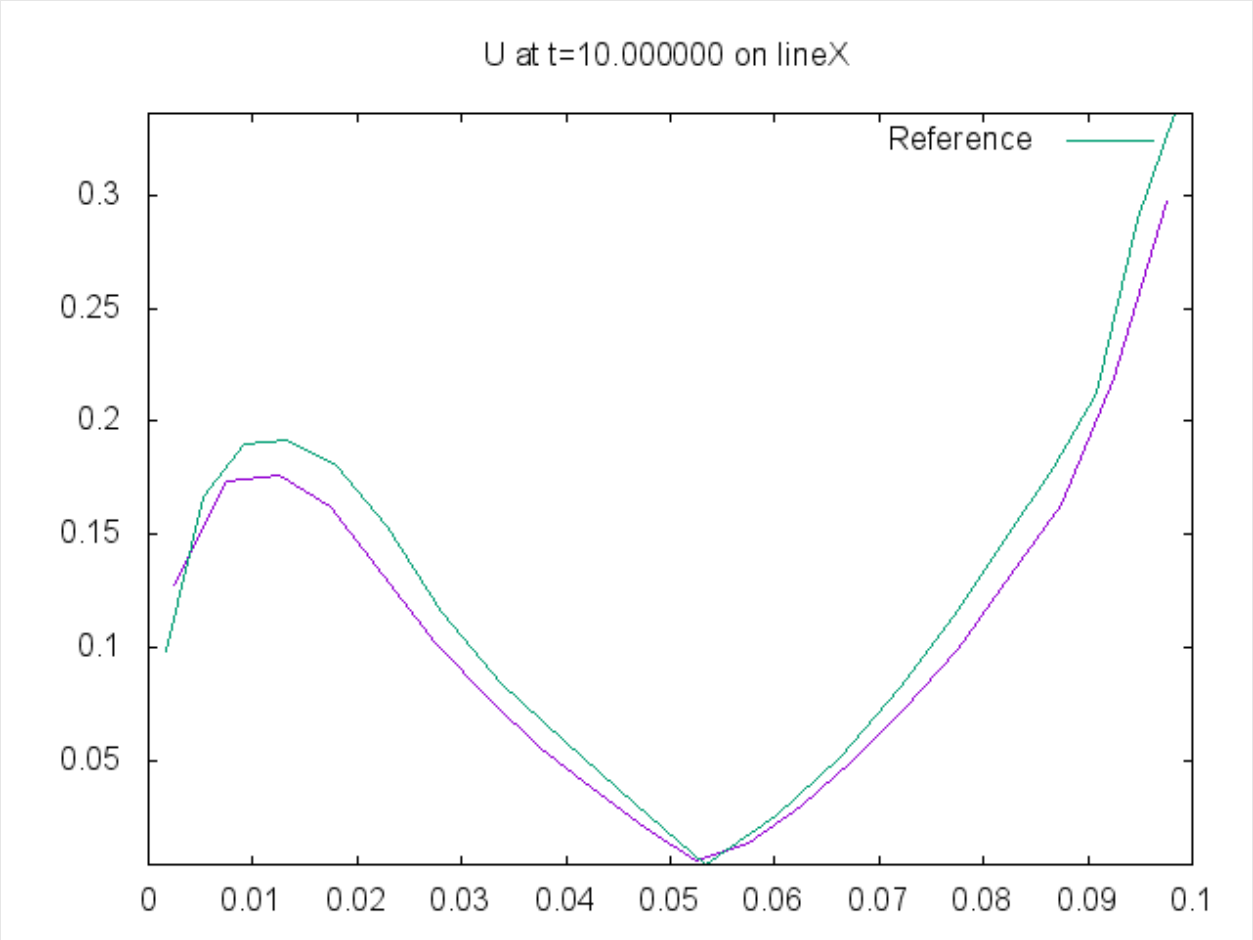
- Different cases can be compared like with `pyFoamTimelinePlot.py`

Comparing graded and normal mesh

```
> pyFoamSamplePlot.py . --dir=post_ico_normal/sample --line=lineX --field=U --time=10 --<brk>
      <cont>reference-dir=post_ico_graded/sample
set term png
set output "post_ico_normal_sample_lineX_U_t=10.000000.png"
set title "U at t=10.000000 on lineX"
plot [][0.00416369:0.336143] "./post_ico_normal/sample/10/lineX_U.xy" using 1:(sqrt($2**2+<brk>
      <cont>$3**2+$4**2)) notitle with lines , "./post_ico_graded/sample/10/lineX_U.xy" <brk>
      <cont>using 1:(sqrt($2**2+$3**2+$4**2)) title "Reference " with lines
> pyFoamSamplePlot.py . --dir=post_ico_normal/sample --line=lineX --field=U --time=10 --<brk>
      <cont>reference-dir=post_ico_graded/sample --metrics --compare
Metrics for U_x (Path: ./post_ico_normal/sample/10/lineX_U.xy )
  Min                 : -0.02213392102
  Max                 : 0.0108074608
  Average             : 0.00378394299825
  Weighted average    : 0.00351659158282
  Time Range          : 0.0025 0.0975
Comparing U_x with name lineX_t=10 U_x (Path: ./post_ico_graded/sample/10/lineX_U.xy )on <brk>
      <cont>original data points
  Max difference      : 0.0205089675181 (at 0.0875 )
  Average difference  : 0.00482816468906
  Weighted average    : 0.00479483370546
Data size: 20 Reference: 20

Metrics for U_y (Path: ./post_ico_normal/sample/10/lineX_U.xy )
```

Meshing does matter

Different results for different meshes



Outline

There is a world outside OpenSource (really)

- Gnuplot is fast and efficient
  - but some (not me) don't like the look of the plots
  - some (on some days even me) find the user-interface cryptic
- Quickly getting differences between curves is fine
  - But sometimes other calculations are needed (regression etc)
- To get the data to the outside world PyFoam has utilities to get data out
  - And sometimes cleaning it up beforehand

CSV and Excel

- There are two formats the data can be exported to
  - CSV *Comma Separated Values*
    - A very simple format: Numbers separated by commas (occasionally semicolons)
    - Can be imported into most data handling programs (but is not completely unproblematic)
    - Supported by the Python Standard library
  - Excel The native format of *Micro$oft Excel*
    - Not an open format
    - But supported by main competitor LibreOffice/OpenOffice
    - One advantag is that there can be more than one table per file
    - To write it Python needs a special library
- Most of the utilities we used so far can export to both
  - `pyFoamRedoPlot.py`
  - `pyFoamTimelinePlot.py`
  - `pyFoamSamplePlot.py`

Exporting the data

Writing CSV

```
> pyFoamRedoPlot.py post_ico_normal/Analyzed/pickledPlots --pickle-file --csv-files --file-<brk>
      <cont>prefix=baseline_
Found 10 plots and 11 data sets
Adding line 1
Adding line 3
<<snip>>
> ls *.csv                                                    :(
baseline_cloudnumbermass.csv       baseline_iterations.csv
baseline_continuity.csv            baseline_linear.csv
baseline_courant.csv               baseline_streamFunctionMinimum.csv
baseline_execution.csv             baseline_velocity.csv
> less baseline_streamFunctionMinimum.csv
time,value
5.00000000000000104e-03,-2.84517349700000028e-05
1.00000000000000021e-02,-5.66402601000000269e-05
1.49999999999999944e-02,-8.45680323299999924e-05
2.00000000000000042e-02,-1.12237314400000017e-04
2.50000000000000139e-02,-1.39649981799999869e-04
2.99999999999999889e-02,-1.66807518299999957e-04
3.50000000000000333e-02,-1.93711027899999941e-04
```

Two different timesteps

We're going to use these two CSV-files in the next examples

Same values, two cases, different times

```
> pyFoamTimelinePlot.py . --basic-mode=lines --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction --csv=fixedDeltaT.csv
> less fixedDeltaT.csv
time,minStreamFunction_t=0 min
5.000000000000000104e-03,-1.479636689999999893e-04
1.000000000000000021e-02,-1.873116001000000024e-04
1.499999999999999944e-02,-1.963009173000000111e-04
2.000000000000000042e-02,-2.095705089000000085e-04
<<snip>>.
> pyFoamTimelinePlot.py . --basic-mode=lines --dir=post_pimple_turb_kOmega_Co_1.000000<brk>
      <cont>_normal/swakExpression_minStreamFunction --csv=variableDeltaT.csv
> less variableDeltaT.csv
time,minStreamFunction_t=0 min
5.882352940999999689e-03,-1.663019844000000107e-04
1.281512605000000027e-02,-2.133572537000000106e-04
2.095359882999999965e-02,-2.290522763999999897e-04
3.037709363000000093e-02,-2.513809428000000120e-04
4.097852527999999717e-02,-2.742728834999999857e-04
5.321094641000000125e-02,-3.065476934000000201e-04
```

Resampling

- Sometimes one wants to join two data sets
  - If the times in two sets of lines differ then PyFoam complains
- The `--resample` option fixes this
  - Reference values are resampled to the current times
    - Linear interpolation between the original values
- Curves with differing time ranges have the problem "what to do with the 'outside'"
  - Usually the first curve 'wins'
    - If the other curve has 'outside' values they are thrown away
  - Option `--extend-data` fixes
    - Full range is used
    - Values 'outside' the original time range are set as 'nothing' (usually `NaN`)

Adding the reference time

One csv to check the difference

```
> pyFoamTimelinePlot.py . --basic-mode=lines --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction --field=minStreamFunction --reference-dir=<brk>
      <cont>post_pimple_turb_kOmega_Co_1.000000_normal/swakExpression_minStreamFunction --<brk>
      <cont>csv=resample.csv
 Warning in /Users/bgschaid/Development/OpenFOAM/Python/PyFoam/bin/pyFoamTimelinePlot.py : <brk>
       <cont>Try the --resample-option
 Error in /Users/bgschaid/Development/OpenFOAM/Python/PyFoam/bin/pyFoamTimelinePlot.py : <brk>
       <cont>FatalError in PyFoam: 'PyFoam FATAL ERROR on line 267 of file /Users/bgschaid/<brk>
       <cont>private_python/PyFoam/Basics/SpreadsheetData.py: Size of the arrays differs'
> pyFoamTimelinePlot.py . --basic-mode=lines --dir=post_pimple_turb_kOmega_normal/<brk>
      <cont>swakExpression_minStreamFunction --field=minStreamFunction --reference-dir=<brk>
      <cont>post_pimple_turb_kOmega_Co_1.000000_normal/swakExpression_minStreamFunction --<brk>
      <cont>csv=resample.csv --resample
> less resample.csv
time,minStreamFunction_t=0 min,Reference minStreamFunction_t=0 min
5.000000000000000104e-03,-1.479636689999999893e-04,nan
1.000000000000000021e-02,-1.873116001000000024e-04,-1.942499625318771627e-04
1.499999999999999944e-02,-1.963009173000000111e-04,-2.175707771913065291e-04
2.000000000000000042e-02,-2.095705089000000085e-04,-2.272132636407113481e-04
2.500000000000000139e-02,-2.221855639000000059e-04,-2.386400932092690847e-04
2.999999999999999889e-02,-2.345357240000000046e-04,-2.504874316298304313e-04
3.500000000000000333e-02,-2.459269740999999788e-04,-2.613633019735076934e-04
```

Outline

Why join

If we talk about *time* here we mean *the independent monotonic rising value*.
This also applies to `samples`

- Often we want to join data from different sources into one document
  - Timelines of different values
    - From different cases
- Of course this can be done in LibreOffice
  1. Load Dataset 1
  2. Load Dataset 2
  3. Copy data columns (not time) from 2 to 1
     - Make sure that the times are aligned
- But this is
  - tedious
  - error-prone
  - not easy to automatize
- For this task `pyFoamConvertToCSV.py` was created
  - It used to 'only' convert text files to `CSV` but took over the functionality of a utility `pyFoamJoinCSVFiles.py`
  - Can read and write
    - Plain text files (with spaces separating the values)
    - CSV files
    - Excel files

Basic usage

- Gets a list of N files ($N > 1$)
  - The first $N - 1$ files are the input
    - `--default-read-format` say which format they are
    - With `--automatic-format` it tries to automatically determine the format from the extension
  - The last file is the output
    - If it already exists the program fails
    - This should help you to not shoot you in the foot (accidentially overwrite 'good' data)
    - If you want to shoot: `--force` writes the data anyway
- `--print-columns` is nice as it shows the data columns that have been read
- If all input files have the same times nothing else has to be done

Joining data

- Default behaviour: only times from the first file are used
  - Other files are interpolated
    - This interpolation can be switched off with `--new-data-no-interpolate`
- `--add-times` adds the times from the other files
  - But for existing columns the names are not interpolated
    - This leaves "empty" cells
    - `--interpolate-new-times` interpolates these times
- `--extend-data` extends the time range
- All these options should cover most applications
  - "I want only the resolution of the first file"
  - "I want the data as fine as possible"
  - "I want only the original data"

Joining examples

```
> pyFoamConvertToCSV.py --automatic fixedDeltaT.csv variableDeltaT.csv joined.csv
> less joined.csv
time,fixed fixedDeltaT minstreamfunction_t0_min,variable variableDeltaT <brk>
      <cont>minstreamfunction_t0_min
5.000000000000000104e-03,-1.479636689999999893e-04,nan
1.000000000000000021e-02,-1.873116001000000024e-04,-1.942499625318771627e-04
1.499999999999999944e-02,-1.963009173000000111e-04,-2.175707771913065291e-04
2.000000000000000042e-02,-2.095705089000000085e-04,-2.272132636407113481e-04
2.500000000000000139e-02,-2.221855639000000059e-04,-2.386400932092690847e-04
> pyFoamConvertToCSV.py --automatic fixedDeltaT.csv variableDeltaT.csv onlyAdd.csv --add-<brk>
      <cont>times
> less onlyAdd.csv
 time,fixed fixedDeltaT minstreamfunction_t0_min,variable variableDeltaT <brk>
       <cont>minstreamfunction_t0_min
5.000000000000000104e-03,-1.479636689999999893e-04,nan
5.882352940999999689e-03,nan,-1.663019844000000107e-04
1.000000000000000021e-02,-1.873116001000000024e-04,-1.942499625318771627e-04
1.281512605000000027e-02,nan,-2.133572537000000106e-04
1.499999999999999944e-02,-1.963009173000000111e-04,-2.175707771913065291e-04
> pyFoamConvertToCSV.py --automatic fixedDeltaT.csv variableDeltaT.csv interpolateNew.csv <brk>
      <cont>--add-times --interpolate-new-times
> less interpolateNew.csv
time,fixed fixedDeltaT minstreamfunction_t0_min,variable variableDeltaT <brk>
      <cont>minstreamfunction_t0_min
5.000000000000000104e-03,-1.479636689999999893e-04,nan
5.882352940999999689e-03,-1.549074215456700606e-04,-1.663019844000000107e-04
1.000000000000000021e-02,-1.873116001000000024e-04,-1.942499625318771627e-04
1.281512605000000027e-02,-1.923728123042866135e-04,-2.133572537000000106e-04
1.499999999999999944e-02,-1.963009173000000111e-04,-2.175707771913065291e-04
2.000000000000000042e-02,-2.095705089000000085e-04,-2.272132636407113481e-04
```

Column names

- The data sets to be joined usually have very similar column names
  - PyFoam tries to make them unique by adding parts of the file names that are different
- Also has parameters to manipulate the column names
  - Choose a different column as time with `--time-name`
  - Use other column names with `--set-names`
- `--column-names` allows selecting the columns that will be copied to the output file
  - Can be switched to regular expressions
- Column names can be manipulated before writing
  - Replacing parts with `--column-name-replacements`
  - Applying Python one-lines to them with `--column-name-transformation`

Collecting all velocity maximums

Compare all cases

Using `--column-names` to select what we use

```
> pyFoamConvertToCSV.py --automatic post_*/swakExpression_velocityStatistics/0/<brk>
      <cont>velocityStatistics allVelocities.xls --write-excel --force  --add-times --<brk>
      <cont>interpolate-new --column-names="max" --print-columns
Columns in post_ico_graded/swakExpression_velocityStatistics/0/velocityStatistics :
   Time
   velocityStatistics max
Eliminated from post_ico_graded/swakExpression_velocityStatistics/0/velocityStatistics : <brk>
      <cont>weightedQuantile09, weightedAverage, weightedQuantile01
Columns in post_ico_normal/swakExpression_velocityStatistics/0/velocityStatistics :
   Time
   velocityStatistics max
Eliminated from post_ico_normal/swakExpression_velocityStatistics/0/velocityStatistics : <brk>
      <cont>weightedQuantile09, weightedAverage, weightedQuantile01
Columns in post_pimple_turb_kOmega_Co_0.500000_normal/swakExpression_velocityStatistics/0/<brk>
      <cont>velocityStatistics :

<<snip>>

Columns in written data :
   Time
   ico_graded velocityStatistics max
   ico_normal velocityStatistics max
   pimple_turb_kOmega_Co_0.500000_normal velocityStatistics max
   pimple_turb_kOmega_Co_1.000000_normal velocityStatistics max
   pimple_turb_kOmega_normal velocityStatistics max
   piso_turb_kOmega_normal velocityStatistics max
   rhoPimple_turb_kOmega_graded velocityStatistics max
   rhoPimple_turb_kOmega_normal velocityStatistics max
```

Dropping the redundant part

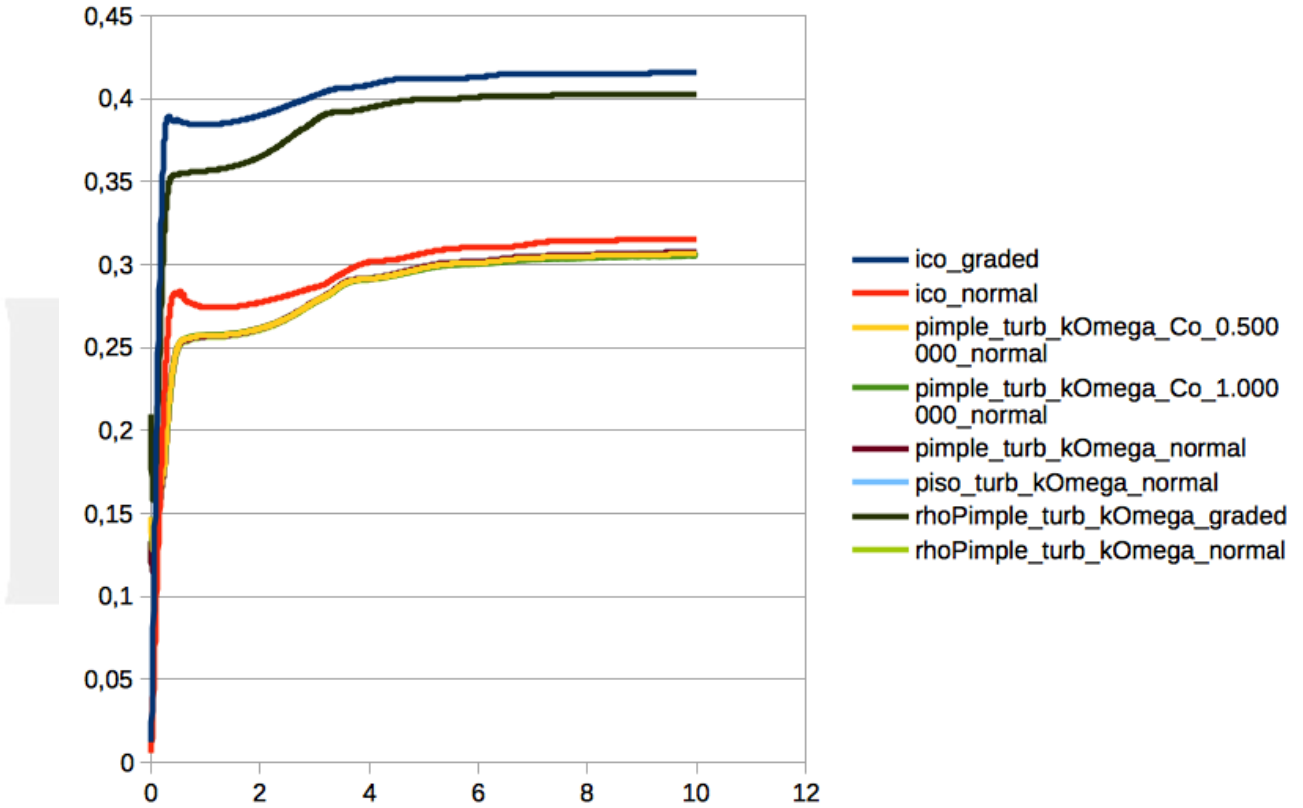
Adding a simple Python one-liner that drops everything after the first space . . . if there is a space

Only the first part

```
> pyFoamConvertToCSV.py --automatic post_*/swakExpression_velocityStatistics/0/<brk>
      <cont>velocityStatistics allVelocities.xls --write-excel --force  --add-times --<brk>
      <cont>interpolate-new --column-names="max" --print-columns --column-name-<brk>
      <cont>transformation="lambda c:c[:c.find(' ')] if c.find(' ')>0 else c"
Columns in post_ico_graded/swakExpression_velocityStatistics/0/velocityStatistics :
   Time
   velocityStatistics max
Eliminated from post_ico_graded/swakExpression_velocityStatistics/0/velocityStatistics : <brk>
      <cont>weightedQuantile09, weightedAverage, weightedQuantile01
Columns in post_ico_normal/swakExpression_velocityStatistics/0/velocityStatistics :

<<snip>>

Columns in written data :
   Time
   ico_graded
   ico_normal
   pimple_turb_kOmega_Co_0.500000_normal
   pimple_turb_kOmega_Co_1.000000_normal
   pimple_turb_kOmega_normal
   piso_turb_kOmega_normal
   rhoPimple_turb_kOmega_graded
   rhoPimple_turb_kOmega_normal
```

Plot of the velocities in LibreOffice

Why do the graded meshes have higher maximum velocities?



Don't be lazy. The answer is easy

Other features

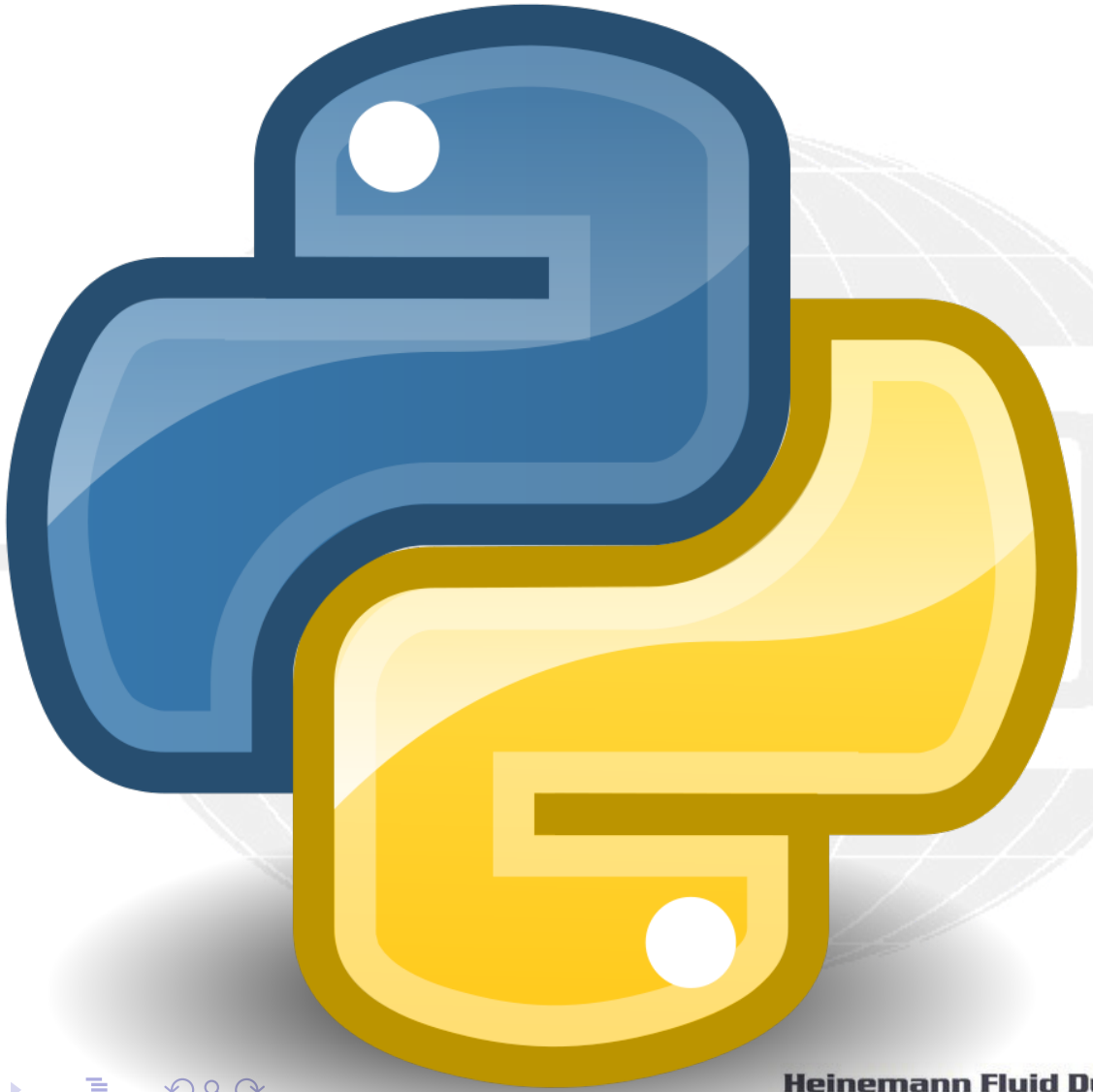
What wasn't shown

- Parameters to remove characters from the input
  - For instance some versions of the `forces` function objects add `()` and `numpy` can't handle that
- Doing calculations on columns and adding formulas to Excel files
  - Example: immediately recalculate Kelvin to Fahrenheit

Outline

Outline

This is what the official Python Logo looks like



This is what the logo should look like

The Python scientific stack

- Python is becoming one of the most popular environments for scientific computing

numpy	fast mastrix operations
pandas	tables (built on `numpy`)
scipy	Collection of numerical algorithms
matplotlib	Plotting library
ipython	Improved interactive shell
jupyter	Notebook environment built on `ipython`
sympy	Symbolic computation

- "MATLAB is sooo 90s"
  - Do yourself a favour: install these libraries and try them

Integration of PyFoam with the Python environment

- –interactive-after-execution : Instead of finishing the utility drops the user to the Python shell
  - Allows inspection of everything that is still in memory
  - Works for every PyFoam-utility
  - If `IPython` is present then this shell is used
- Data handling utilities have additional options to make the data available

  –pandas-data Wraps the data into a `pandas.DataFrame` and makes it available to the user
  - We'll use that to interactively "play" with the data
  - To reproduce only enter the stuff after `In[x]` :

Getting to the shell

Dropping to the shell

```
> pyFoamRedoPlot.py post_ico_normal/Analyzed/pickledPlots --pickle-file --pandas-data --<brk>
      <cont>interactive-after
Found 10 plots and 11 data sets
Adding line 9

<<snip>>

Plotting 5 : courant
Plotting 6 : timestep No data - skipping

Dropping to interactive shell ... found IPython ...up-to-date IPython

Python 3.4.4 (default, Mar  2 2016, 03:31:27)
Type "copyright", "credits" or "license" for more information.

IPython 4.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Preparing the environment

`In [i]:` is the prompt of `ipython`

matplotlib

```
In [1]: %matplotlib
Using matplotlib backend: Qt4Agg
```

This "magic" command enables interactive plotting windows

Getting the data

```
In [2]: data=self.getData()["plotData"]
```

- `self` is the utility itself
- `getData()` returns a dictionary with the utility specific data
  - Entry `plotData` is the data as seen in the plots

Getting the velocity data

DataFrame with the velocity data

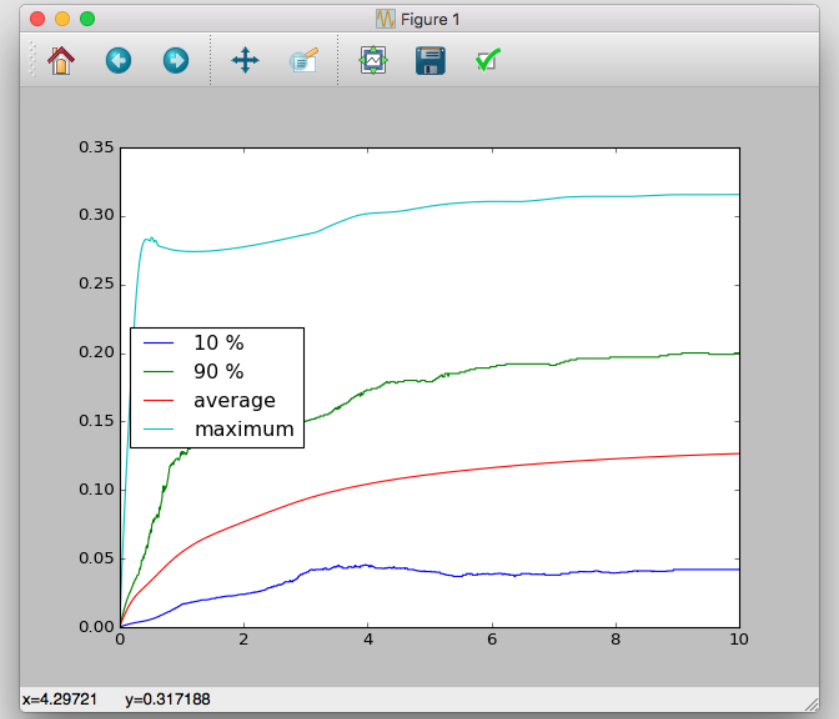
```
In [4]: vel=data["velocity"]

In [5]: vel.describe()
Out[5]:
                          10 %          90 %       average       maximum
count               2000.000000   2000.000000   2000.000000   2000.000000
mean                   0.034282      0.166222      0.099562      0.295723
std                    0.011498      0.040472      0.029071      0.033038
min                    0.000085      0.000910      0.000599      0.005686
25%                    0.030000      0.148000      0.085534      0.282549
50%                    0.039200      0.180000      0.111340      0.307001
75%                    0.042000      0.196000      0.121506      0.314090
max                    0.045500      0.200000      0.126599      0.315655
integral               0.342711      1.661722      0.995304      2.956423
valid length           9.995000      9.995000      9.995000      9.995000
weighted average       0.034288      0.166255      0.099580      0.295790
```

- `describe()` gives general statistics about the columns in a `DataFram`
  - `integral`, `valid length` and `weighted average` is something that PyFoam adds

Plotting the data

plot() means "Plot!"

```
In [6]: vel.plot()
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x111d9cba8>
```



Calculating the rate of change for the velocity

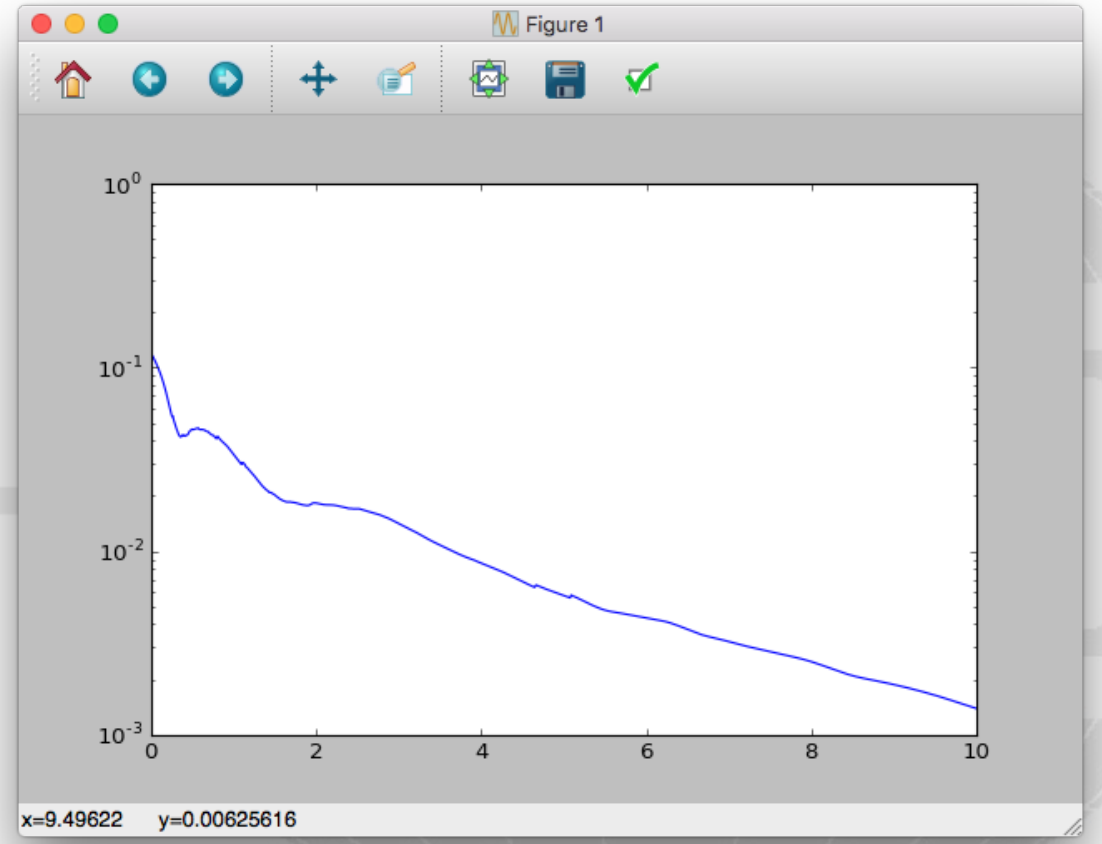
We'll say: "If the average velocity does not rise anymore then we're *converged*
Which is a bit ... naive.

More advanced pandas

```
In [7]: import pandas as pd

In [8]: vel["t"]=pd.Series(vel.index,index=vel.index)

In [9]: (vel.average.diff()/vel.t.diff()).plot(logy=True)
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x115296a20>
```

- We're adding a new field `t` to the frame for calculation
  - The time was already in the frame, but 'only' as the index
- Simple numerical differentiation gives the rate of change

Differentiated average velocity



What to do next with the data

- Interactive playing is nice. But to make it reproducible we could do two things:
  1. Write a proper script
     - Calling the utility can be replaced with the proper classes from `PyFoam.Applications`
  2. Write a proper documentation in a `jupyter` notebook
     - PyFoam has classes to inspect a case in a `jupyter/IPython` notebook

Outline

Getting particle data

- PyFoam has classes to read particle data from a case
  1. Inspects the `lagrangian` sub-directory
  2. Gets all the fields it "understands"
  3. Loads them into a `DataFrame`
- There currently is no utility to do that
  - Because I can't think of a general application for that. Only "special cases"
    - And for those 'hand-rolled' scripts are better
- Beware: for large particle numbers this may take some time
  - Reading. Calculations are fast thanks to `numpy`
- There are also classes to read particle data for the `patch` cloud-function objects
  - But we don't cover these here

What we're trying to do

- We had particles added to our case
- The cloud had a cloud function object `eliminateBySwakExpression`
  - Part of `swak4Foam`
  - Expression age>5 means that all particles older than $5s$ are eliminated
  - Properties of the particles are written to a new cloud `coldParticleCloudeliminateOldEliminatedPost`
- We want to find out where the particles are when they "die"

Starting

```
> ipython
```

Preparing the environment

Importing

```
In [1]: %matplotlib
Using matplotlib backend: Qt4Agg

In [2]: from PyFoam.RunDictionary.SolutionDirectory import SolutionDirectory

In [3]: from PyFoam.RunDictionary.LagrangianCloudData import LagrangianCloudData

In [4]: import pandas as pd

In [5]: from numpy import sqrt
```

- Importing the necessary PyFoam libraries
- And some external stuff

Loading particle data from one time-step

Just to see whether this works

Particles from $t = 10$

```
In [6]: lcd=LagrangianCloudData(".","coldParticleCloudeliminateOldEliminatedPost","10")

In [7]: lcd.data.Px
Out[7]:
0    0.052467
1    0.054251
2    0.052586
3    0.055570
4    0.052528
5    0.054092
6    0.053947
7    0.053156
8    0.052635
9    0.051244
Name: Px, dtype: float64
```

Only 10 particles died since the last writing

Putting all the data into one frame

SolutionDirectory knows all the timesteps

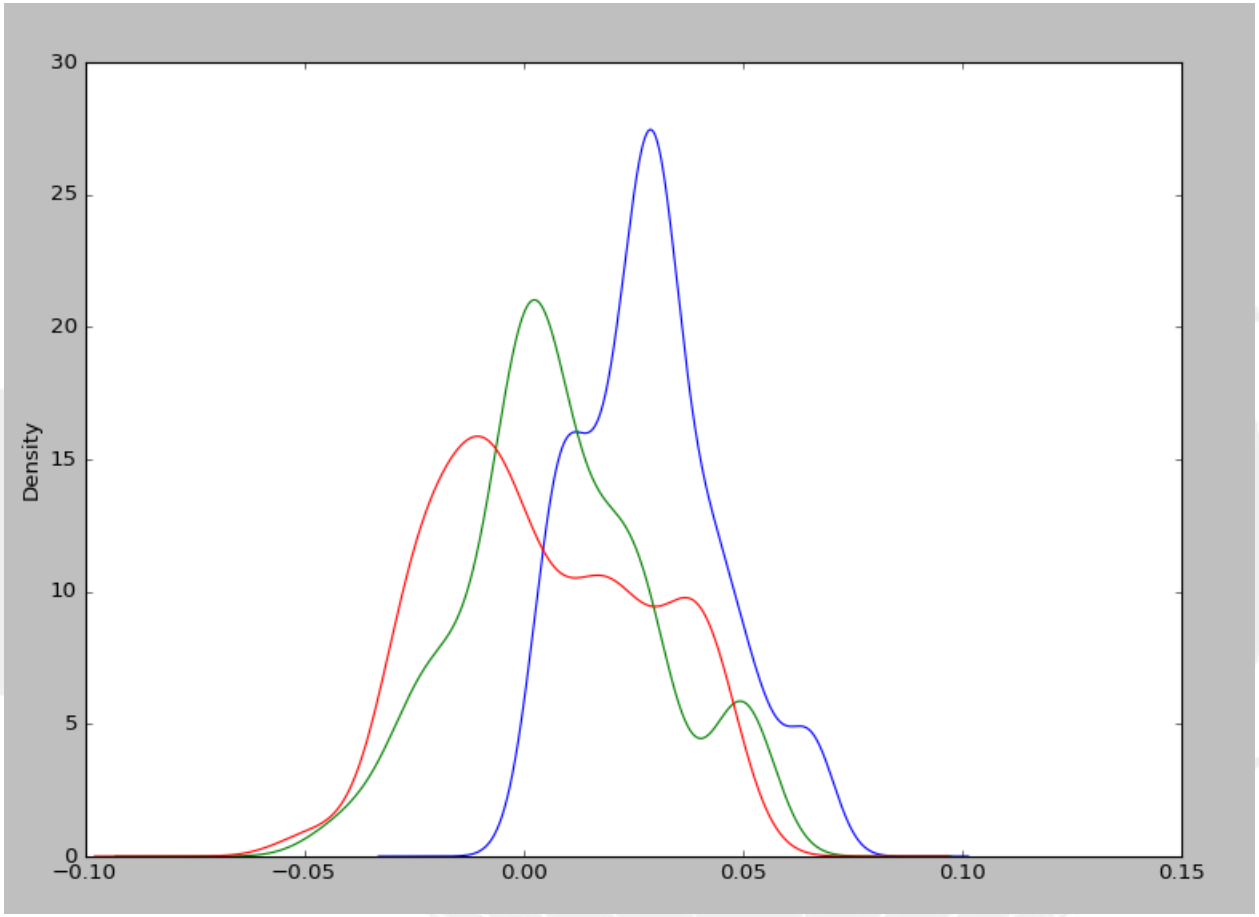
```
In [8]: sol=SolutionDirectory(".")

In [9]: data=pd.concat([LagrangianCloudData(".","<brk>
      <cont>coldParticleCloudeliminateOldEliminatedPost",t).data for t in sol.times if <brk>
      <cont>float(t)>5])

In [10]: data.keys()
Out[10]:
Index(['Px', 'Py', 'Pz', 'cellI', 'U_x', 'U_y', 'U_z', 'UTurb_x', 'UTurb_y',
       'UTurb_z', 'active', 'age', 'd', 'dTarget', 'nParticle', 'origId',
       'origProcId', 'rho', 'tTurb', 'typeId', 'globalId', 'nowCpu',
       'writeTime'],
      dtype='object')

In [11]: data.describe()
Out[11]:
                Px           Py             Pz       cellI          U_x  \
count  251.000000   251.000000   2.510000e+02  251.000000   251.000000
mean     0.058030     0.053991   5.000000e-03  217.039841    -0.001527
std      0.021822     0.023805   1.912008e-17   95.576097     0.084094
min      0.004630     0.000959   5.000000e-03    9.000000    -0.152329
25%      0.046125     0.035022   5.000000e-03  140.000000    -0.056106
50%      0.054484     0.049859   5.000000e-03  199.000000    -0.004958
75%      0.072186     0.072125   5.000000e-03  294.500000     0.044852
max      0.099999     0.098338   5.000000e-03  399.000000     0.698096

               U_y    U_z  UTurb_x  UTurb_y  UTurb_z      ...       dTarget  \
count  251.000000  251.0    251.0    251.0    251.0      ...         251.0
mean     0.002715    0.0      0.0      0.0      0.0      ...           0.0
std      0.055050    0.0      0.0      0.0      0.0      ...           0.0
min     -0.131327    0.0      0.0      0.0      0.0      ...           0.0
25%     -0.020167    0.0      0.0      0.0      0.0      ...           0.0
```

Doing the actual calculations

Adding new fields to the frame

```
In [12]: data["rPx"]=data.Px-0.05

In [13]: data["rPy"]=data.Py-0.05

In [14]: data["distCenter"]=sqrt(data.rPx*data.rPx+data.rPy*data.rPy)

In [15]: data.distCenter.describe()
Out[15]:
count    251.000000
mean       0.029366
std        0.016055
min        0.000630
25%        0.017590
50%        0.028603
75%        0.039393
max        0.067901
Name: distCenter, dtype: float64

In [16]: data.distCenter.plot(kind="density")
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x119fee5c0>

In [17]: data.rPx.plot(kind="density")
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x119fee5c0>

In [18]: data.rPy.plot(kind="density")
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x119fee5c0>
```

Distributions of the particle positions



The missing legend on that graph is proof of my laziness

Doing further things

- pandas offers lots of other possibilities
  - For instance: selecting particles that fit a certain criterion and analyzing them
- But you can always export the data for your un-hip Windows-colleague

To Excel

```
In [19]: data.to_excel("particleData.xlsx")
```

Trade secret: `to_excel` is used by the utilities for exporting data with =–excel-file

Outline

Being really lazy

- Must of the command lines here are very long
  - Which is a pain for lazy people
- What I like to do for a case-template I'm going to use often (> 2):
- Prepare a little shell script that
  1. Does some basic Paraview visualizations
     - That way I don't have to open Paraview to find out what went wrong
  2. Export data for the quantitative analysis
     - That way I'm sure I didn't copy the CO_2 concentration to the \$O$_2$\$-column by accident
  3. Generate plots of timelines etc
     - That way I can quickly check what might be of interest
     - Also I can inspect the effect of changes
- Always consider: what is the easiest job for the tool:
  - Native Foam program, PyFoam, shell-script

`--help` is your friend

- This presentation is not a canonical listing of all options there are
- To see them use the `--help`-option every PyFoam-utility has
  - Often something you think "should be there" already there
- If you think "this is bad English and/or cryptic" you're probably right
  - Let me know what would be better

Goodbye to you

Thanks for listening

Questions?

License of this presentation

Authors of this document are:

Bernhard F.W. Gschaider original author and responsible for the strange English grammar. Contact him for a copy of the sources if you want to extend/improve/use this presentation