pyFoam - The dark, unknown corners

Quantative analysis, case control and more

Bernhard F.W. Gschaider

TU Wien

12. March 2012

Outline

Outline

Outline

What this presentation is about

- ... is about PyFoam
- But not the parts that are popular
  - ... or maybe a little bit
- It is about aspects of PyFoam that seem to be less popular
  - Helping with quantitative analysis
  - Checking the evolution of a case with a Version Control System (VCS)
  - Controlling an OpenFOAM-run over the net
    - Setting up a Meta-Server

Introducing Ignaz

- Ignaz Gartengschirrl is a CFD-engineer
  - Specialist on the damBreak-case
    - Avoided broadening his field of expertise
  - Is one of the people to use PyFoam the longest
    - Likes it, because it saves him time
- We will join him on a typical day of dam-breaking

Outline

Previous presentations

There are two previous presentations about the adventures of Ignaz:

- PyFoam - Happy foaming with Python: Held at the OpenFOAM-Workshop in Montreal (2009)
  Introduction of PyFoam - mostly about the utilities
- Automatization with pyFoam - How Python helps us to avoid contact with OpenFOAM: Held at the Workshop in Gothenburg (2010)
  Mainly about writing scripts with PyFoam

One presentation mentions PyFoam

- No C++, please. We're users!: Held at the 2011 Workshop (PennState)
  This presentation is about `swak4Foam` but mentions some concepts of PyFoam (especially `customRegexp`)

The mighty web

Further information about PyFoam are found

- on the openfoamwiki.net:
  - Overview of the utilities (always updated to the latest release)
  - Some examples on scripting
  - Release notes (what's new)
- the MessageBoard
  - Usually announcements of new releases (if I don't forget)

Outline

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

Outline

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
```

Generated files

- Typically PyFoam generates several files during a run (the names of some of those depend on the case-name)

  case.foam Stub-file to open the case in ParaView

  PyFoamRunner.solver.logfile File with all the output that usually goes to the standard-output

  PyFoamRunner.solver..analyzed Directory with the results of the output analysis

  pickledPlots A special file that stores all the results of the analysis

  PyFoamHistory Log with all the PyFoam commands used on that case

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

Outline

The language

- Python is a scripting language
  - Object oriented
    - But also supports other paradigms like functional programming and aspect oriented programming
  - A big library that comes with it
  - Has a very simple syntax
- Built as the scripting-glue into a number of CAE-software
  - ParaView, VisIt, Salome, ...
- There is a number of interesting libraries for technical mathematical uses
  - `matplotlib`, `numpy`, `scipy`
  - A lot of them are glued together in the Mathematica-like program Sage

PyFoam as a library

- Below the surface PyFoam is a library that knows how to write OpenFOAM-files

```
from PyFoam.RunDictionary.ParsedParameterFile import ParsedParameterFile

control=ParsedParameterFile("system/controlDict")
print "Timestep was",control["deltaT"]
controlDict["deltaT"]=controlDict["endTime"]/1000
print "Timestep now is",control["deltaT"]
control.writeFile()
```

Outline

Outline

Pictures vs numbers

Randy: "That looks good."
Earl: "Randy, don't eat that. It's poisonous!"
Randy: "How poisonous?"

- Most of the time we're happy if the simulation runs an "looks right"
  - But sometimes somebody comes along and asks "How right"?
- Usually then it is better to confidently say "Within 3% of the measured data" than "It's the same shade of blue in Paraview"
- OpenFOAM has utility to extract such data
  - PyFoam tries to ease the handling of this

Where OpenFOAM stores quantitative data

- OpenFOAM stores quantitative data the way it stores everything:
  - Organized in directories and files
  - Usually on the case-level there is a directory named after the utility that created the data and/or the name of the data set in the relevant dictionary
- Inside the files data is stored on a column-based basis
  - One line for each timestep (or location if it is sample-data)
  - Data columns separated by spaces
    - This allows to immediately plot it with GnuPlot
  - The first line may contain the names of the columns
    - For `sample` the names of the columns can be extracted from the file-names

The history of these specific PyFoam-utilities

- Started as little helper scripts to help with generating Gnuplot-graphs
  - Therefor the names
- Over time learned a lot more tricks
  - Calculating metrics of the data
  - Comparing data with other data
  - Writing data as CSV-files

Outline

Does the mesh influence the solution?

- Ignaz has been working with `interFoam/damBreak` for a long time
- ... but he is not sure: How much influence does the mesh have on the solution?
  - Just looking at the colorful pictures in Paraview doesn't convince him
- So he decided to compare the solution on a sample line

Where we're sampling

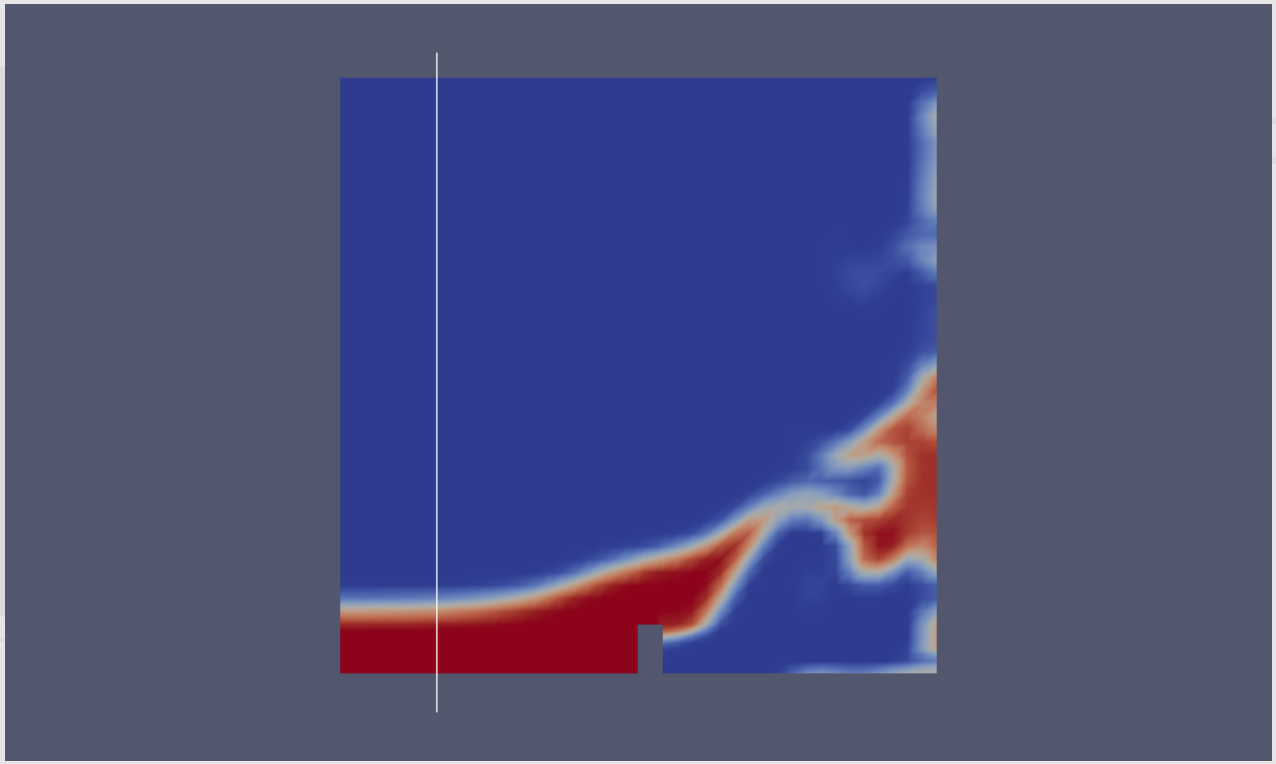


Figure: Location of the sample line

Setting up the sample in sampleDict (excerpt)

```
setFormat raw;
fields
(
    p
    U
    alpha1
);
sets
(
    lineX1
    {
        type          midPoint;
        axis          y;

        start         (0.10 -0.01 0.01);
        end           (0.10 0.6 0.01);
        nPoints       100;
    }
);
```

Sampling and examining the damage

- Ignaz runs the command to sample

```
1 > sample
```

- He examines the sampled data in the first directory

```
1 > ls sets/0
lineX1_U.xy          lineX1_alpha1.xy
```

- But the files in other directories look slightly different

```
: > ls sets/0.1
2 : lineX1_U.xy          lineX1_alpha1_p.xy
```

- So writing a simple shell-script would have to skip the data at the initial time

What is there

- Ignaz remembers `pyFoamSamplePlot.py`
- After consulting the output of

```
pyFoamSamplePlot.py --help
```

- he tries

```
> pyFoamSamplePlot.py . --dir=sets --info
Times :  ['0', '0.05', '0.1', '0.15', '0.2', <brk>
    <cont>'0.25', '0.3', '0.35', '0.4', '0.45',<brk>
    <cont> '0.5', '0.55', '0.6', '0.65', '0.7',<brk>
    <cont> '0.75', '0.8', '0.85', '0.9', <brk>
    <cont>'0.95', '1']
Lines :  ['lineX1']
Fields:  ['U', 'alpha1', 'p']
```

- so the utility found out what is there

Nomenclature

For the utility a dataset is specified by a triple:

time the time at which the sample was taken

line the sample line on which the data was taken (the name in the `sampleDict`)

field the name of the field. The utility distinguishes between scalar and vector-fields

If unspecified all values of these are used (if present in the data))

First Plot

- Ignaz tries a plot

```
> pyFoamSamplePlot.py . --dir=sets --field=alpha1 --time<brk>
    <cont>=0.2
set term png
set output "sets_lineX1_alpha1_0004.png"
set title "alpha1 at t=0.200000 on lineX1"
plot [][-1.10825e-61:1] "./sets/0.2/lineX1_alpha1_p.xy" <brk>
    <cont>using 1:2 notitle with lines
```

- Ignaz is not satisfied with this text and tries

```
> pyFoamSamplePlot.py . --dir=sets --field=alpha1 --time<brk>
    <cont>=0.2 | gnuplot
```

- And gets what he wants
  - **Please tilt your heads to the right for the next slide**

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

The actual plot

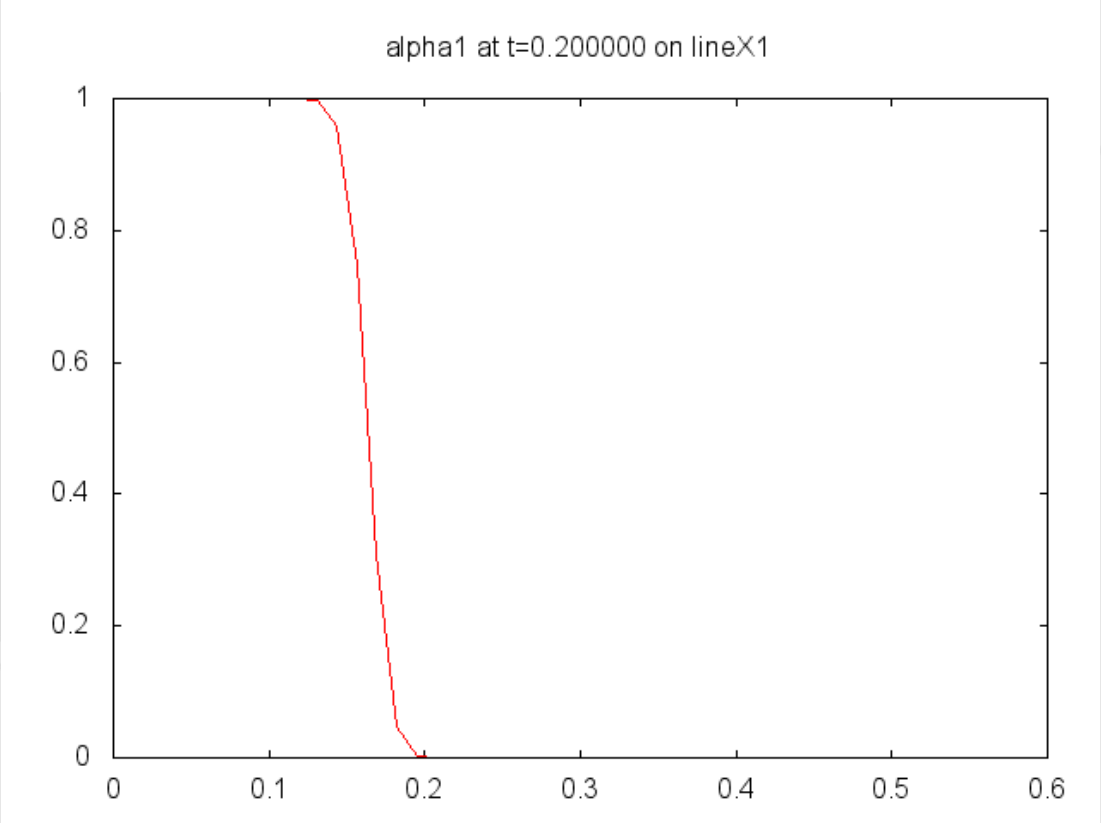


Figure: `alpha1` on the sample line

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

No time necessary

- Ignaz omits the --time-option and gets

```
pyFoamSamplePlot.py . --dir=sets --field=<brk>
    <cont>alpha1
set term png
set output "sets_lineX1_alpha1_0000.png"
set title "alpha1 at t=0.000000 on lineX1"
plot [][-6.60227e-56:1] "./sets/0/<brk>
    <cont>lineX1_alpha1.xy" using 1:2 notitle <brk>
    <cont>with lines
set output "sets_lineX1_alpha1_0001.png"
set title "alpha1 at t=0.050000 on lineX1"
plot [][-6.60227e-56:1] "./sets/0.05/<brk>
    <cont>lineX1_alpha1_p.xy" using 1:2 notitle<brk>
    <cont> with lines
set output "sets_lineX1_alpha1_0002.png"
```

Multiple plots

- If Ignaz would have piped it into `gnuplot` he would have gotten 21 pictures
  - The `[][-6.60227e-56:1]` means that PyFoam examined the data and scaled the plots to the minimum and maximum of **all** times
  - This allows easy animation of pictures
  - For instance looking at a slideshow with ImageMagick

```
animate *.png
```

- But moving pictures make Ignaz dizzy so he does

```
pyFoamSamplePlot.py --mode=timesInOne . --dir <brk>
    <cont>=sets --field=alpha1
```

- and gets ...

A plot with a lot of lines

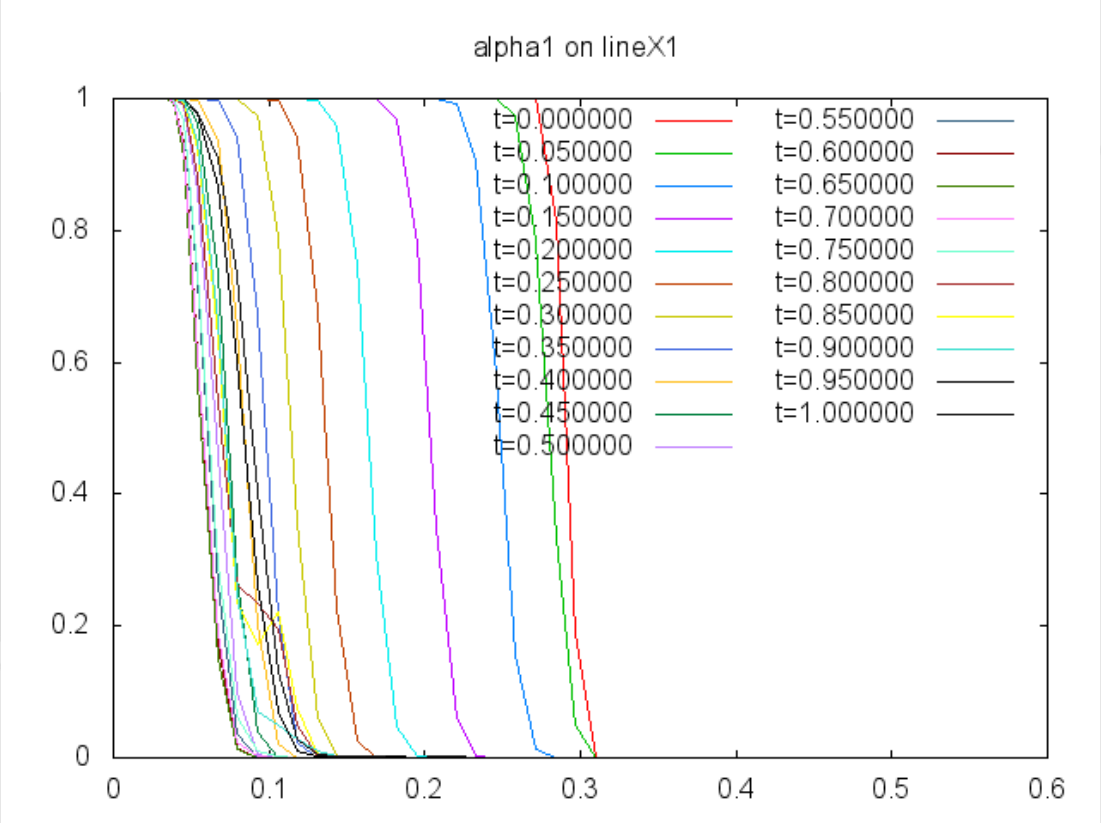


Figure: `alpha1` for **all** timesteps

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Different modes

- Multiple fields and times can be specified at once
- The utility offers four different modes how plots are collected

separate The default. Every value at every time gets a separate picture

timesInOne For each value the plots at all (specified) times gets collected into one picture

fieldsInOne For each time the plots for all (specified) values get collected into one picture. **Note:** this only makes sense if the values are of a similar scale

complete Everything is thrown into one huge plot

Vectors

- Ignaz wants to see a vector value:

```
> pyFoamSamplePlot.py . --dir=sets --field=U --time=0.1
set term png
set output "sets_lineX1_U_0002.png"
set title "U at t=0.100000 on lineX1"
plot [][0.197252:0.856962] "./sets/0.1/lineX1_U.xy" using 1:(sqrt($2**2+$3**2+$4**2)) notitle with lines
```

- But it is also possible to look at just one component

```
> pyFoamSamplePlot.py . --dir=sets --preferred --component=1 --field=U --time=0.1
set term png
set output "sets_lineX1_U_0002.png"
```

Velocity

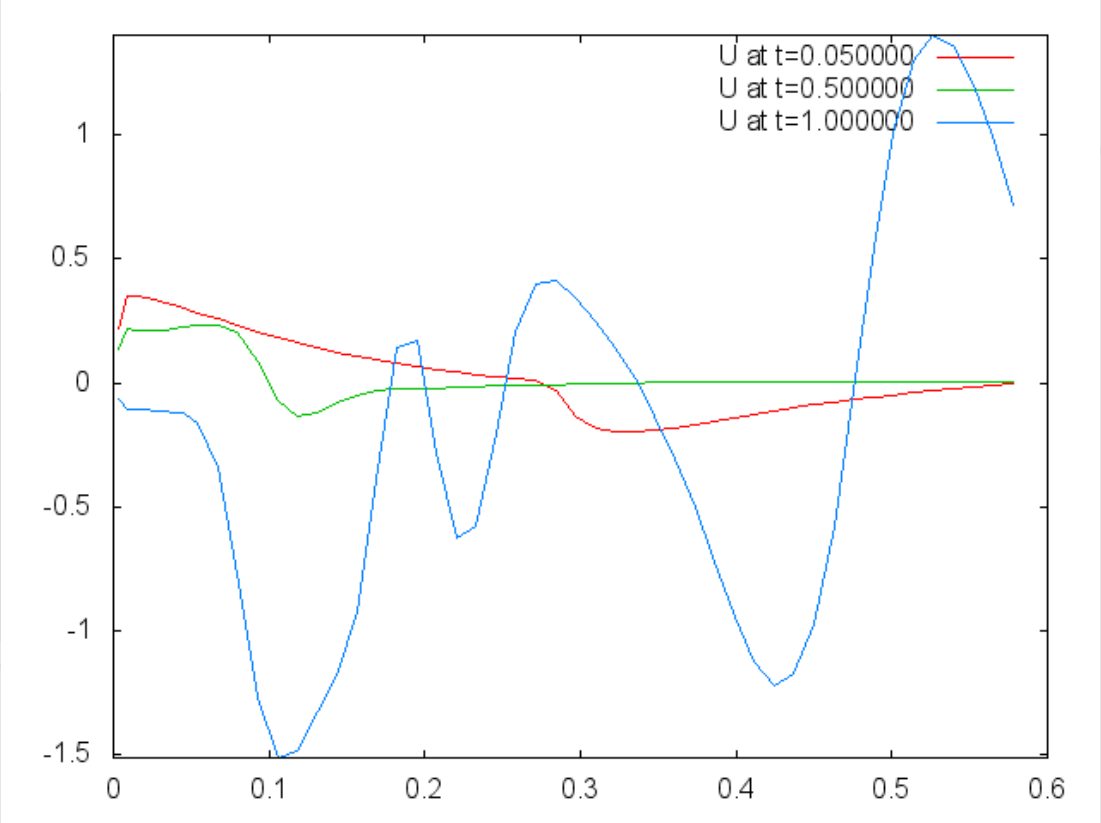


Figure: U for some timesteps

Getting finer

- Now Ignaz decides to make the mesh finer
- First he saves the old data for reference

```
> mv sets setsReference
```

- Then he edits `blockMeshDict`
  - Examining the difference with `hg diff` (see VCS-chapter):

```
 blocks
 (
-    hex (0 1 5 4 12 13 17 16) (23 8 1) simpleGrading (1 1 1)
-    hex (2 3 7 6 14 15 19 18) (19 8 1) simpleGrading (1 1 1)
-    hex (4 5 9 8 16 17 21 20) (23 42 1) simpleGrading (1 1 1)
-    hex (5 6 10 9 17 18 22 21) (4 42 1) simpleGrading (1 1 1)
-    hex (6 7 11 10 18 19 23 22) (19 42 1) simpleGrading (1 1 1)
+    hex (0 1 5 4 12 13 17 16) (35 12 1) simpleGrading (1 1 1)
+    hex (2 3 7 6 14 15 19 18) (29 12 1) simpleGrading (1 1 1)
+    hex (4 5 9 8 16 17 21 20) (35 63 1) simpleGrading (1 1 1)
+    hex (5 6 10 9 17 18 22 21) (6 63 1) simpleGrading (1 1 1)
+    hex (6 7 11 10 18 19 23 22) (29 63 1) simpleGrading (1 1 1)
 );
```

Running and looking at the data

- Ignaz does the usual: `blockMesh`, `setFields`, `interFoam` and `sample`
- Now adding the `--reference-directory`-option reveals an addition set of data:

```
> pyFoamSamplePlot.py . --dir=sets --reference-directory=<brk>
    <cont>setsReference --info
Times :  ['0', '0.05', '0.1', '0.15', '0.2', '0.25', '0.3',<brk>
    <cont> '0.35', '0.4', '0.45', '0.5', '0.55', '0.6', <brk>
    <cont>'0.65', '0.7', '0.75', '0.8', '0.85', '0.9', <brk>
    <cont>'0.95', '1']
Lines :  ['lineX1']
Fields:  ['U', 'alpha1', 'p']

Reference Data:
Times :  ['0', '0.05', '0.1', '0.15', '0.2', '0.25', '0.3',<brk>
    <cont> '0.35', '0.4', '0.45', '0.5', '0.55', '0.6', <brk>
    <cont>'0.65', '0.7', '0.75', '0.8', '0.85', '0.9', <brk>
    <cont>'0.95', '1']
Lines :  ['lineX1']
Fields:  ['U', 'alpha1', 'p']
```

Working with reference data

- If specified reference data is automatically used
  - If there is reference data that fits the selected data (time, line or field) it is added to the plots
  - The utility can be asked to be tolerant with the time (use the next best time-step)
- Ignaz redoes the last plot:

```
> pyFoamSamplePlot.py . --dir=sets --field=U <brk>
<cont>--time=0.05 --time=0.5 --time=1 --<brk>
<cont>mode=complete --prefer=0 --reference-<brk>
<cont>directory=setsReference --style=steps<brk>
<cont> | gnuplot
```

Comparing the x-velocity

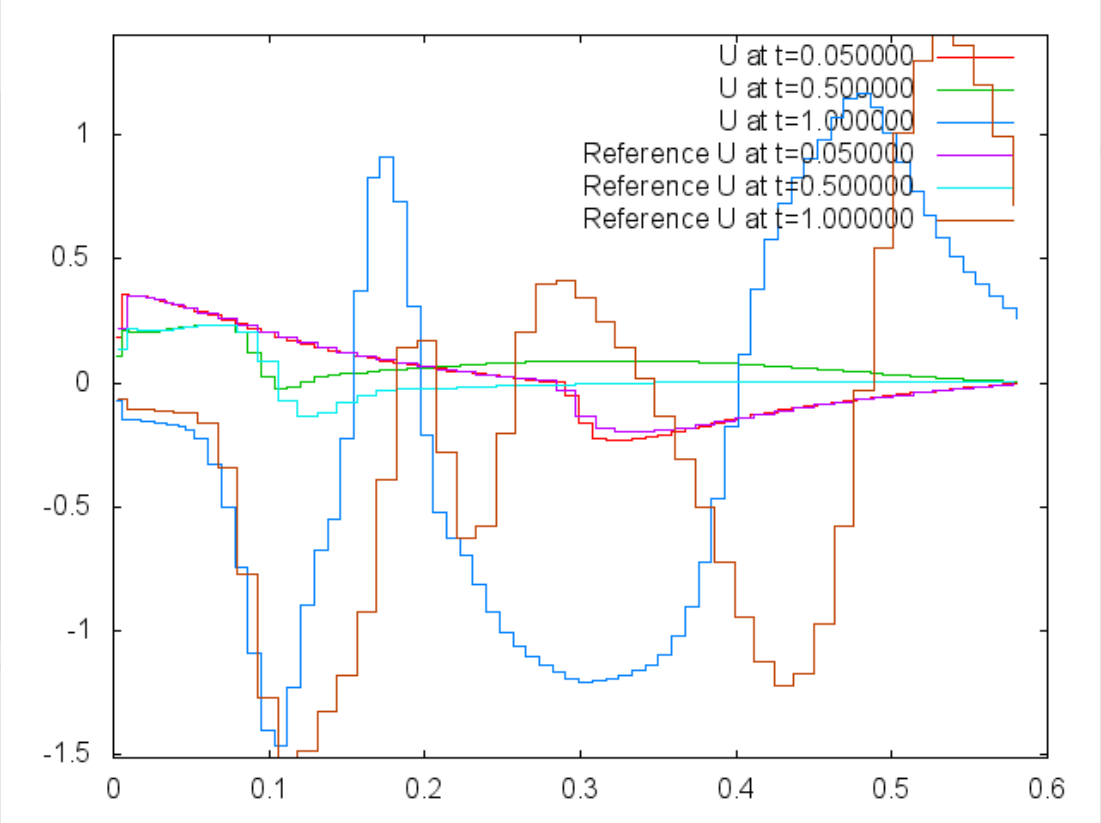


Figure: Similar U

How big is the pressure?

- Ignaz wants to know the values of the pressure at a given time
  - `--metric` gives him that information
    - Minimum and maximum
    - Average (purely algebraic per grid point or weighted by the grid-spacing)

```
> pyFoamSamplePlot.py . --dir=sets --field=p <brk>
    <cont>--time=0.05 --metrics
Metrics for p (Path: ./sets/0.05/<brk>
    <cont>lineX1_alpha1_p.xy )
  Min                    : -5.69937
  Max                    : 1016.9
  Average                : 205.842199107
  Weighted average       : 146.987573517
Data size: 75
  Time Range             : 0.00199996 0.579746
```

How different is the pressure?

- Ignaz wants to compare it to the reference solution
  - And does so with `--compare`
    - Use `--reference-time` or `--tolerant-reference-time` if there is trouble

```
> pyFoamSamplePlot.py . --dir=sets --field=p --time=0.05 --<brk>
    <cont>metrics --reference-directory=setsReference --<brk>
    <cont>compare --toler
Metrics for p (Path: ./sets/0.05/lineX1_alpha1_p.xy )
  Min                    : -5.69937
  Max                    : 1016.9
  Average                : 205.842199107
  Weighted average       : 146.987573517
  Time Range             : 0.00199996 0.579746
Comparing p with name lineX1_t=0.05 p (Path: ./<brk>
    <cont>setsReference/0.05/lineX1_alpha1_p.xy )
  Max difference         : 16.97023 (at 0.247936 )
  Average difference     : 3.01177972326
  Weighted average       : 2.87341242197
Data size: 75 Reference: 50
```

But is that a lot?

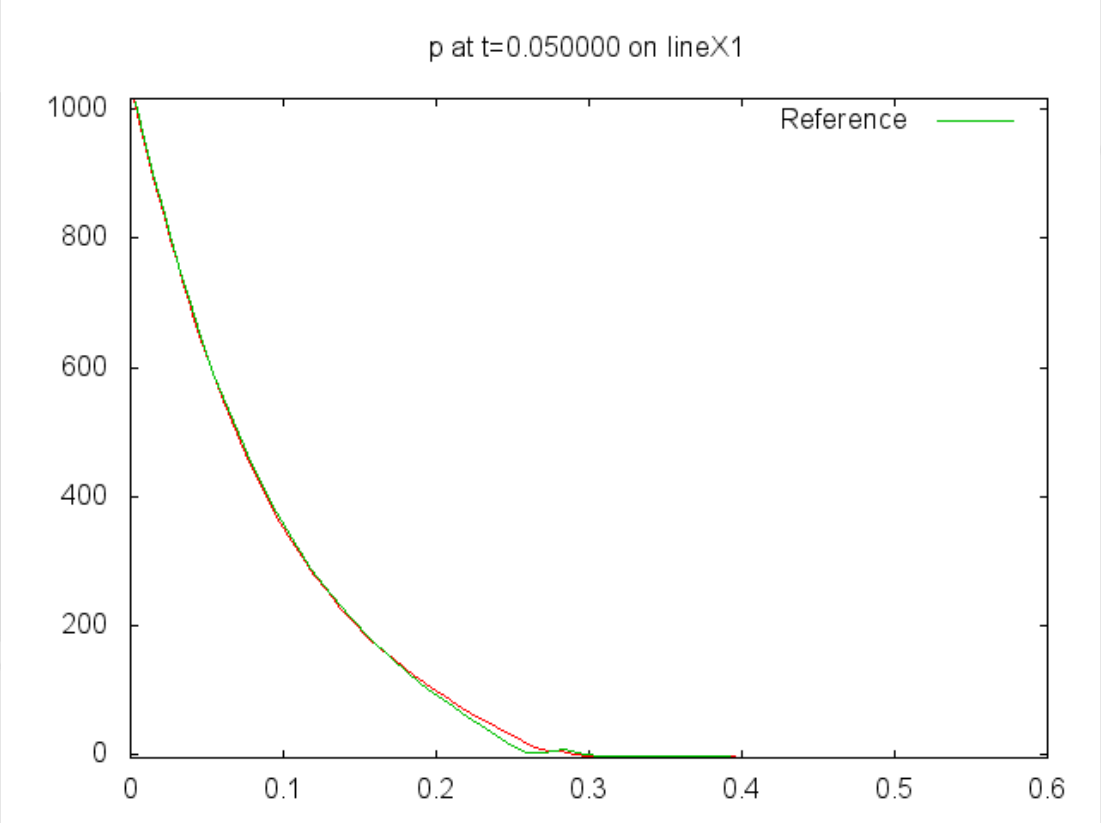


Figure: Comparing the p to the reference solution

Bringing the data somewhere else

- Gnuplot is not enough
  - Ignaz wants to do a more thorough analysis of the date
  - Ignaz's boss wants fancier graphs
- The "simplest" format for column-oriented data is CSV (Comma Separated Values)
  - Each line is a dataset
  - Columns are separated by commas
  - The first line may contain the column titles
- CSV can be read by Excel and almost any other program that handles data
  - OpenOffice, Paraview, ...

Writing the volume-fraction

- If the option `--csv-file` has a value then all datasets that would have been plotted are written in CSV-format
- Ignaz writes the volume fraction at all times to file:

```
> pyFoamSamplePlot.py . --dir=sets --field=<brk>
  <cont>alpha1 --csv=alpha1.csv
```

- On a GNOME-workstation he can open that file from the command line with whatever is the predefined application for CSV:

```
1 > gnome-open alpha1.csv
```

Reference data

- Ignaz wants to compare the fraction at a specific time

```
> pyFoamSamplePlot.py . --dir=sets --field=alpha1 --csv=alpha1.<brk>
    <cont>csv --time=0.5 --reference-directory=setsReference --<brk>
    <cont>tolerant
Warning in /Users/bgschaid/Development/OpenFOAM/Python/PyFoam/<brk>
    <cont>bin/pyFoamSamplePlot.py : Try the --resample-option
Error in /Users/bgschaid/Development/OpenFOAM/Python/PyFoam/bin<brk>
    <cont>/pyFoamSamplePlot.py : PyFoam FATAL ERROR on line 142 <brk>
    <cont>of file /Users/bgschaid/private_python/PyFoam/Basics/<brk>
    <cont>SpreadsheetData.py: Size of the arrays differs
```

- The reason for the error are the different grid resolutions
- So Ignaz tries it with the resampled data

```
> pyFoamSamplePlot.py . --dir=sets --field=alpha1 --csv=alpha1.<brk>
    <cont>csv --time=0.5 --reference-directory=setsReference --<brk>
    <cont>tolerant --resample
```

The joint CSV-data

Start of the file looks like this

```
col0,lineX1_t=0.5 alpha1,Reference lineX1_t=0.5 alpha1
1.9999599999999846e-03,1.0000000000000000e+00,nan
5.9998700000000030e-03,1.0000000000000000e<brk>
    <cont>+00,1.0000000000000000e+00
9.9997799999999780e-03,1.0000000000000000e<brk>
    <cont>+00,1.0000000000000000e+00
1.3999700000000034e-02,1.0000000000000000e<brk>
    <cont>+00,1.0000000000000000e+00
1.7999600000000102e-02,1.0000000000000000e<brk>
    <cont>+00,1.0000000000000000e+00
2.1999500000000169e-02,1.0000000000000000e<brk>
    <cont>+00,1.0000000000000000e+00
2.5999399999999890e-02,1.0000000000000000e<brk>
    <cont>+00,1.0000000000000000e+00
2.9999399999999898e-02,1.0000000000000000e<brk>
    <cont>+00,1.0000000000000000e+00
3.3999300000000313e-02,1.0000000000000000e<brk>
    <cont>+00,1.0000000000000000e+00
3.7999199999999687e-02,1.0000000000000000e<brk>
    <cont>+00,1.0000000000000000e+00
4.1999099999999755e-02,1.0000000000000000e<brk>
    <cont>+00,9.9482899999999633e-01
4.5998999999999822e-02,9.9909999999999881e<brk>
    <cont>-01,9.7575357718342592e-01
5.2252899999999796e-02,9.6576200000000093e<brk>
    <cont>-01,8.8879670733085314e-01
6.0760899999999963e-02,7.5487700000000202e<brk>
    <cont>-01,6.5877650000000981e-01
6.9268800000000537e-02,3.5511399999999852e<brk>
    <cont>-01,3.9838784861188375e-01
7.7776799999999316e-02,7.2275300000000066e<brk>
    <cont>-02,1.5860396972237689e-01
8.6284700000000584e-02,2.3881399999999968e<brk>
    <cont>-03,5.1543969180137712e-02
9.4792699999999363e-02,5.7976799999999690e<brk>
    <cont>-05,3.7038568341834967e-03
1.0330100000000039e-01,1.6909999999999988e<brk>
    <cont>-06,8.0238186581650357e-04
1.1180900000000056e-01,5.4423100000000245e<brk>
    <cont>-08,3.9453504999999964e-05
1.2031699999999934e-01,1.8227400000000056e<brk>
    <cont>-09,1.5625178166666684e-06
```

Notes about resampling

- Resampling is done by linearly interpolating between data points
- Always the reference data is resampled
  - This potentially "destroys" the "real" data
  - If this is a problem write the data separately
- `--extend-data` can be used to "stretch" the data if the datasets differ in size

Outline

When the levee breaks

- Ignazs boss decides that the important point is when the flood jumps over the dam
- Ignaz chooses three points which are important
  - Called
    - This Side
    - On The Dam
    - Other Side
  - But we don't call them that way
- Later he notices that the location of the points is not the best one

Locations of the measurments

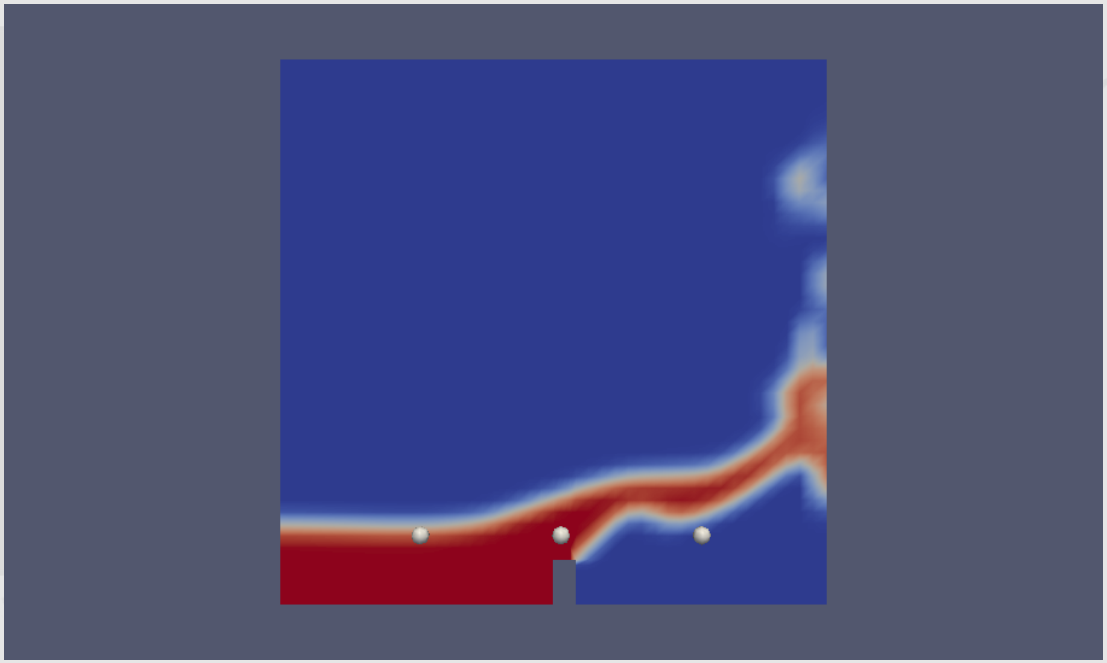


Figure: The three points

Addition to the controlDict

```
functions
{
    probes
    {
        type            probes;
        functionObjectLibs ("libsampling.so");
        enabled         true;
        outputControl   timeStep;
        outputInterval  1;

        fields
        ( p U alpha1 );

        probeLocations
        (
            (0.15 0.075 0.01)
            (0.30 0.075 0.01)
            (0.45 0.075 0.01)
        );
    }
}
```

How OpenFOAM writes the data

- Directory with the name of the functionObject
  - In it a directory named after the time the functionObject was created
- One file for every field
- Format of the fields is column-oriented
  - Vectors are written in the usual OpenFOAM-notation
    - With (and)
    - This confuses Gnuplot
  - Names of the positions either
    - in the first line
    - in the first 4 lines
  - `pyFoamTimelinePlot.py` can handle both formats

Finding out what is there

- Ignaz wants to find out whether `pyFoamTimelinePlot.py` recognizes the data

```
> pyFoamTimelinePlot.py . --dir=probes --info
Write Times      :  ['0']
Used Time        :  0
Fields           :  ['alpha1', 'p', 'U']  <brk>
    <cont>Vectors:  ['U']
Positions        :  ['(0.15 0.075 0.01)', '(0.3 <brk>
    <cont>0.075 0.01)', '(0.45 0.075 0.01)']
Time range       :  (0.0011904800000000001, 1.0)
```

Description of `--info`-output

Write Times Times at which probes were created (usually only one)

Used Time Which of the write times is used

Fields The written fields

Vectors Which of those is a vector value

Positions Names of the positions (as extracted from the files)

Time Range Over which range timeline data exists

The basic-mode

- There are two modes
  - One of them always has to be specified

    lines line-plot of the value over time

    bars bar chart of the values at a selected time
- The plots can be collected in two ways in a plot

  fields for each field one plot of all positions

  positions for each position all fields in one plot

Getting the fraction over time

- Ignaz wants to plot the change of `alpha1` over time

```
> pyFoamTimelinePlot.py . --dir=probes --field <brk>
  <cont>=alpha1 --basic-mode=lines | gnuplot
```

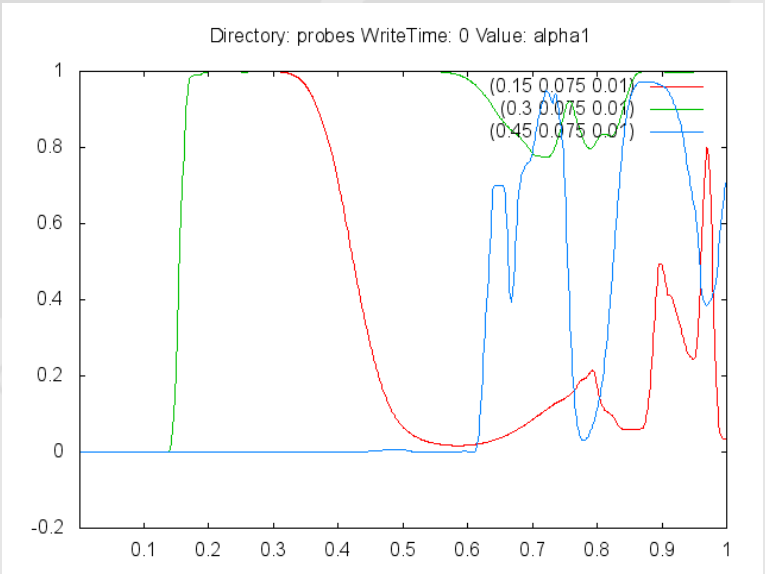


Figure: Time-line plot

Fractions at a specific time

- An he is interested in the exact values at $t = 0.5$

```
1 > pyFoamTimelinePlot.py . --dir=probes --field <brk>
    <cont>=alpha1 --basic-mode=bars --time=0.5 <brk>
    <cont>--collect=positions | gnuplot
```

Figure: Bar plot

Preparing for comparing

- Ignaz saves the current data as a reference

```
1 > mv probes probeReference
```

- And he refines the mesh again
- And reruns the case

Comparing the fraction

- Ignaz wants the value over the dam

```
1 > pyFoamTimelinePlot.py . --dir=probes --field <brk>
    <cont>=alpha1 --basic-mode=lines --<brk>
    <cont>reference-directory=probeReference --<brk>
    <cont>position="(0.3 0.075 0.01)" | gnuplot
```

Comparing the data

- Ignaz finds that because the points are very close over the water-line the results differ quite substantially:

```
> pyFoamTimelinePlot.py . --dir=probes --field --<brk>
  <cont>=alpha1 --basic-mode=lines --<brk>
  <cont>reference-directory=probeReference --<brk>
  <cont>position="(0.3 0.075 0.01)" --compare
Comparing alpha1 on (0.3 0.075 0.01) index 1 (<brk>
  <cont>path: ./probes/0/alpha1 )
  Max difference       : 0.98572490559
  Average difference   : 0.10763347239
  Weighted average     : 0.12292201846
Data size: 1603 Reference: 991
  Time Range           : 0.00119048 1.0
```

Other features

- Has other similar features like the `pyFoamSamplePlot.py` utility
  - CSV-files
  - Handling of vectors
- But some things are slightly inconsistent between the two
  - Especially the handling of the velocity-components

Outline

Different cases

- ... have different timesteps
- Ignaz nevertheless wants to combine them into one file

```
> pyFoamTimelinePlot.py . --dir=probes --field <brk>
  <cont>=alpha1 --csv=fineProbe.csv
> pyFoamTimelinePlot.py . --dir=probeReference <brk>
  <cont> --field=alpha1 --basic-mode=lines --<brk>
  <cont>csv=coarseProbe.csv
```

- There are two different ways to join the file

```
> pyFoamJoinCSV.py coarseProbe.csv fineProbe.<brk>
  <cont>csv joinedCoarse.csv
> pyFoamJoinCSV.py fineProbe.csv coarseProbe.<brk>
  <cont>csv joinedFine.csv
```

Interpolation during joining

- Ignaz looks at the lines of the files

```
> wc -l *.csv
    992 coarseProbe.csv
   1604 fineProbe.csv
    992 joinedCoarse.csv
   1604 joinedFine.csv
```

- The first file determines the resolution of the joined CSV-file
  - there are options to modify this behavior

Outline

Writing CSV-files with Redo-Plot

- pyFoamPlotWatcher.py currently can't write CSV-files
  - But he produces a file pickledPlots in the *.analyzed-directory
  - The contents of this can be written to CSV:

```
> pyFoamRedoPlot.py --csv-files --pickle-file <brk>
<cont>PyFoamRunner.interFoam.analyzed/<brk>
<cont>pickledPlots
```

- Now for instance the probe information and the timestep can be joined into one file

```
> pyFoamJoinCSV.py fineProbe.csv timestep.csv <brk>
<cont>joinedTimestepAndProbes.csv
```

Writing sampledSurfaces

- sampleDict has the possibility to specify surface to sample
  - One possible format to write the surfaces is VTK
  - When using this the utility pyFoamSurfacePlot.py can work with these
- The usage of pyFoamSurfacePlot.py is quite similar to the two utilities discussed above
  - Produces pictures (no pipe into another program required)
  - Flexible in determining which data is really there
  - Automatically rescaling to have the same data-range over the whole animation
  - Tries to automatically determine the best position for the camera
- It is no replacement for Paraview, but it allows to quickly produce animations (without opening Paraview)
  - Especially nice for producing the animations in batch-jobs

Outline

Outline

Motivation

- Sometimes questions arise like
  - "When did I set this parameter?"
  - "What is the difference between these two cases?"
  - "The case worked three weeks ago. What did I mess up since then?"
- In software-development a decent VCS helps with these things
  - And because an OpenFOAM-case is only a collection of files it can help with OpenFOAM too

What is VCS

- According to Wikipedia:

  Revision control, also known as version control and source control (and an aspect of software configuration management), is the management of changes to documents, programs, large web sites and other information stored as computer files. It is most commonly used in software development, where a team of people may change the same files.

- It doesn't **have** to be program files

Support of VCS in PyFoam

- A special command for initializing a VCS on a case: `pyFoamInitVCS.py`
  - Knows which files are important (`system`, `constant`, etc) and initializes accordingly
- Certain commands support it
  - `pyFoamCloneCase.py` does a VCS-=clone= if the case is version-controlled
  - All the Runner-scripts can commit the latest changes if being started
- For other work the VCS-commands itself can be used
- Currently only Mercurial is fully supported
  - Other VCS are only partially supported
    - This will only change if I need them or somebody sends me patches
- There is a mercurial extension `hg foamdiff` that does the same as a regular `hg diff` but compares OpenFOAM-dictionaries on a syntactic level
  - Not just as text files like the regular diff would

What is Mercurial

- Mercurial is a so called Distributed VCS
  - It doesn't need a central server
    - But then also nobody can tell which server is the definite one
  - Another DVCS is `git`
    - Almost as old as mercurial
    - Popular because it is used for the Linux-kernel

Why Mercurial?

- It can do the same things as `git` but it is simpler and more consistent
- It is written in Python
  - So interaction with PyFoam is easy
- It is easily extendable
  - Writing a plugin like `foamdiff` for `git` is no fun
  - A mercurial client can talk to a `git`-server ... with an extension
- If the case is in a `git`-controlled directory, then the two do not collide
  - Which they would if PyFoam used `git` for that

Mercurial for `git` users

- The basic usage is similar: write `hg` instead of `git`
- most subcommands are similar
  - `commit`, `diff` ..
  - the most annoying difference is that `fetch` and `pull` have exactly the opposite meaning
  - there is no staging area
    - but of course there is an extension that can replicate that
  - the branching system is slightly different
    - there is no artificial distinction between remote and tracking branches

Be careful

- Adding unnecessary files bloats the repository
- Don't add files to the repository for which it doesn't make sense
  - The grid (only the files that were used to produce it)
  - The results
- What makes sense:
  - Reference data from probes and samples

Outline

Trying different stuff at the same time

- Sometimes Ignaz does different things with a case at the same time
  - Improve the numerics
  - Test different physical parameters
- Merging the results of these without forgetting something can be a nerve wrecking experience
  - And there is always the issue "Where did this come from"
- For this task PyFoam supports us with its Mercurial-support
  - Other VCS-systems are possible

Setting up two different cases

- Make sure the master is under version control

```
cd masterCase
pyFoamInitVCSCase . --commit-message="Initial commit"
```

- Create two working cases (pyFoamCloneCase.py knows about Mercurial)

```
cd ..
pyFoamCloneCase.py masterCase numericsTests
pyFoamCloneCase.py masterCase parameterTests
```

Working on the cases

cd numericsTests

```
hg branch numericsBranch
```

- Work on the numerics
- Review the changes (Mercurial-style)

```
hg diff
```

- Review the changes (pyFoam-style)

```
hg foamdiff .
```

- Commit the changes

```
hg commit -m "Now the numerics seem to work"
```

- Push to the master case

```
hg push --new-branch
```

- Do the same in the physics-case

Getting all the changes

- Got to the master

```
cd ../masterCase
```

- Merge in the numerics

```
hg merge numericsBranch
```

- Merge in the physics:

```
hg merge physicsBranch
```

- All the wisdom in one case

```
hg log
```

Outline

Outline

Every PyFoam-run has his server

- Every instance of the `Runner`-class starts a small network-server
  - Of course this can be turned off
- This server monitors the progress of the OpenFOAM-simulation
  - Can also modify the case on the fly
- The server looks for a so called Meta-Server and registers there
  - If there is no Meta-Server there are no consequences
  - The Meta-Server keeps track of all the OpenFOAM-runs in the local network
- The server dynamically finds a free port number (18000) and listens there for commands

Diagram

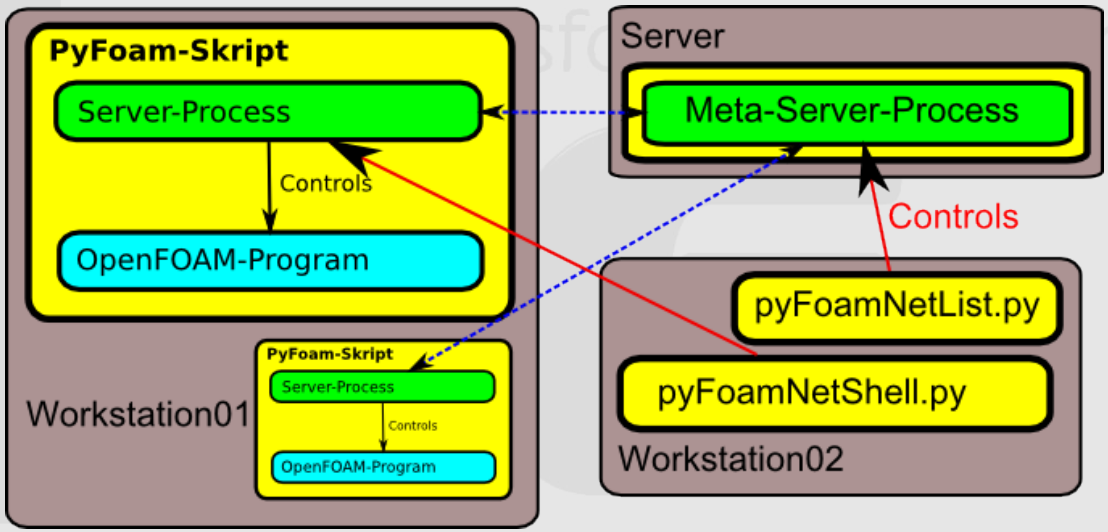


Figure: Server and Metaserver

Usage example

pyFoamNetShell.py Allows control of the server (for instance graceful stopping of runs) and therefor the program

pyFoamNetList.py Lists all runs in the network

```
> pyFoamNetList.py --time --user=igarten
Hostname            |   Port   | User          | Command Line
-------------------------------------------------------------------------------
compute-0-3.local   |   18002  |     igarten   | interFoam -case damBreak1.<brk>
    <cont>restart
  Time: 0.0888 Timerange: [ 0.0798 , 0.16 ]  Mesh created: 0.0798 -><brk>
     <cont> Progress: 11.22% (Total: 11.22%)
  Started: 2009-May-14 11:08   Walltime:  35630.4s  Estimated End: <brk>
     <cont>2009-May-18 03:20
-------------------------------------------------------------------------------
compute-0-2.local   |   18000  |     igarten   | interFoam -case damBreak3.<brk>
    <cont>restart
  Time: 0.09123 Timerange: [ 0.0798 , 0.16 ]  Mesh created: 0.0798 -<brk>
     <cont>> Progress: 14.25% (Total: 14.25%)
  Started: 2009-May-14 09:02   Walltime:  43211.8s  Estimated End: <brk>
     <cont>2009-May-17 21:15
-------------------------------------------------------------------------------
```

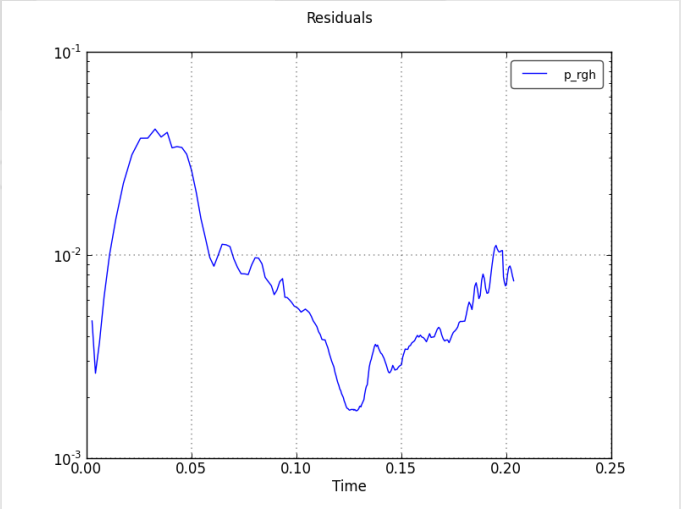
Example session with the NetShell

```
> pyFoamNetShell.py compute-0-3.local 18002
Connected to server compute-0-3.local on port 18002
42 available methods found
PFNET> help
For help on a method type 'help <method>'
Available methods are:
        actualCommandLine
        argv
...
        wallTime
        write
PFNET>  help stop
Method    :  stop
Signature :  signatures not supported
Stops the run gracefully (after writing the last time-step to disk)
PFNET> time()
0.08889
PFNET> stop()
PFNET>
Goodbye
```

Getting the plots over the wire

- pyFoamRedoPlot.py allows to get the plots from a remote machine

```
> pyFoamRedoPlot.py --server localhost 18000
Found 7 plots and 7 data sets
Adding line 1
...
```



Outline

Where should we look for runs?

- The Meta-Server has to know in which networks to look for runs
- This has to be done on the machine and as the user that is going to run the Meta-Server
  - Check with `pyFoamVersion.py` which configuration files are searched
  - In one of them add something like (this may vary depending on your network)

```
[Metaserver]
searchservers: <brk>
    <cont>127.0.0.1/32,192.168.1.0/24,192.168.0.0/24 <brk>
    <cont>
```

- Check with `pyFoamDumpConfiguration.py` that this configuration is really used

Running on a machine

- This is simple:
  - Make sure that for the current user PyFoam is in the `PYTHONPATH`
  - Run the script `pyFoamMetaServer.py` that is found in the `sbin`-directory of the distribution
- The script goes into daemon mode
  - It detaches itself from the shell (no `&` needed) and runs to infinity
- It listens on port 17999
  - Can be controlled via `pyFoamNetShell.py` on that port
    - That is also the proper way to kill it
- Later make sure that it is started on reboot

Making sure that processes find the Metaserver

- Every user that wants his runs to connect to the Meta-sErver should have something like this in her configuration

```
[Metaserver]
ip: 127.0.0.1
```

- Check with `pyFoamDumpConfiguration.py`
- Site-wide configuration is possible
  - See with `pyFoamVersion.py` which directories are searched

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Outline

Problems

I won't lie to you. There are some problems:

- The Meta-Server crashes occasionally (every 3 months or so)
  - This is a bit hard to debug
- There is no security
  - Everybody can kill every process
  - Only use it in environments where you can trust people

Currently I have no intention to fix these because

- Works for me
- Nobody complained

Outline

Outline

"Pipifying"" the commands

- PyFoam-utilities may pass data in pickled format to each other
- This allows nice applications like this:
  - "Print the relative error of the pressure and Ux"

```
> pyFoamTimelinePlot.py . --dir=probes --field=U --vector=x<brk>
  <cont> --field=p --basic-mode=lines --reference-<brk>
  <cont>directory=probeReference --compare --silent --<brk>
  <cont>metrics --pickle-application-data=stdout | <brk>
  <cont>pyFoamPrintData2DStatistics.py --relative-error
2
  Relative Error
4
  = ================== ================== ==================
6 .. (0.15 0.075 0.01) (0.45 0.075 0.01) (0.3 0.075 0.01)
  = ================== ================== ==================
8 p 0.311599749073     0.639612636893     0.604420673856
  U 0.407374218959     0.786966080813     0.298642111637
10 = ================== ================== ==================
```

Change supported Python-versions

- Currently PyFoam supports Python-versions from 2.2-2.7
  - By exceptions and module-substitution
- The currently most used version is 2.6
  - 2.7 the most recent
- Python 3 is coming (slowly but persistent)
  - Sources are incompatible with the 2.x-line
- The plan is to drop support for all versions older than 2.6
  - 2.6 has facilities for the forward-compatibility to Python 3
  - This may be hard for people who are stuck with older Python versions (mostly cluster installations)
    - Ask your admin for an additional `python26`-package (in parallel to the standard-Python)
    - or stay with an older PyFoam-version

Outline

Anybody awake?

- Thanks for listening
- Questions?
- What I'd like:
  - World peace
  - Feedback
    - Especially bug-reports
  - Other contributions

Last words

- One bug had to die during the preparation of this presentation
  - `pyFoamSamplePlot.py` did not correctly plot the x component of a vector
    - It is fixed and will be in the next release
    - One moment of silence, please
    - It will not be missed
- `swak4Foam` was never used during the making of this presentation
  - This required a great deal of self-restraint
  - But of course `swak4Foam` produces files that `pyFoamTimelinePlot.py` can handle