Automatic case setup with pyFoamPrepareCase

The "A tale of two engines" edition

Bernhard F.W. Gschaider

HFD Research GesmbH

Genoa, Italy
11. July 2023

Outline I

1. Introduction
   - About this presentation
   - Who is this?
   - Technicalities
   - Case setup in `OpenFOAM`
   - `PyFoam` overview
   - Our case
2. Using templates
   - Stage 0: `pyFoamPrepareCase.py` without modifications
   - Stage 1: Adding information
   - Stage 2: Calculations and Variables
   - Stage 3: Control structures
   - Stage 4: Loops
3. Scripting

Outline II

- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Purpose of this presentation

- Give an overview of `pyFoamPrepareCase.py`
  - A utility to help set up cases
    - Part of `pyFoam` for over 3 years
  - Not a GUI but something better: a flexible way to script cases

Intended audience

- People who have experience with OpenFOAM
  - Editing the dictionaries should not be a problem
- Worked a little with `pyFoam`
- Are lazy: Don't enjoy doing the same work over and over again
- Are not afraid of a little programming
  - But only a very little bit

History of the presentation

- This is the third version of this presentation
  - It was first held 2015 at the Workshop in Ann Arbor (with cats)
  - A second version (without cats) was held 2018 in Shanghai
- Surprisingly little had to be changed
  - All changes were about changes in OpenFOAM
  - None of them because of changes in `pyFoamPrepareCase.py`
- This is good
  - New versions of `PyFoam` don't break existing cases
    - But new OpenFOAM-versions might
- The new thing is the inclusion of the Jinja2 templating engine

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Bernhard Gschaider

- Working with OpenFOAM™ since it was released
  - Still have to look up things in `Doxygen`
- I am not a core developer
  - But I don't consider myself to be an *Enthusiast*
- My involvement in the OpenFOAM™-community
  - Janitor of the `openfoamwiki.net`
  - Author of two additions for OpenFOAM™

    swak4foam Toolbox to avoid the need for C++-programming

    PyFoam Python-library to manipulate OpenFOAM™ cases and assist in executing them
  - Organizing committee for the OpenFOAM™ *Workshop*
- The community-activies are not my main work but *collateral damage* from my real work at ...

Heinemann Fluid Dynamics Research GmbH

The company

- Subsidary company of *Heinemann Oil*
  - Reservoir Engineering
  - Reservoir management

Description

- Located in Leoben, Austria
- Works on
  - Fluid simulations
    - OpenFOAM™ and Closed Source
  - Software development for CFD
    - mainly OpenFOAM™
- Industries we worked for
  - Automotive
  - Processing
  - ...

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

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 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"
    - Alternatively there are 3 dots

```
> 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
```

- In strings the symbol that looks like "a small 'U' with corners" means "just a white-space" character

```
Third line "with␣a␣string" to demonstrate spaces
```

How to follow this presentations

- All information to reproduce the results is on the slides
  - Assumes that you have an *OpenFOAM v2306+* installation
    - But we're not strict about this. Older versions work as well
    - Later we need a *foam-extend 5* installation
- There is a file with the different stages on the Wiki
  - You can download and untar it in a convenient location
    - More later
  - A docker image with the software and materials is available

Docker image with pre-installed PyFoam and swak4Foam

- Docker is a technology to run pre-packed containers based on Linux
  - Can be run on Linux, Windoze and Mac OS X
  - Saves the work of installing requirements and compiling software
    - Only docker is needed (see `https://www.docker.com/`)
    - Image downloads may be rather big
- There is an image prepared for this training
  - Found at `https://hub.docker.com/r/bgschaid/openfoam_by_ansible`
  - Based on Ubuntu 22.04 LTS
  - OpenFOAM v2306
  - Most recent release of PyFoam
  - Most current development version of swak4Foam
  - has no ParaView.
  - but it has the examples of this presentation
- The image was prepared with `https://openfoamwiki.net/index.php/Installation/Ansible`

Pulling the Docker-Image

Problems here:

- The image is over 2 Gig.
  - Depending on your network this might take some time
- You have to have `docker` installed on your machine

Pulling the container

This will download the container the first time around

```
> docker pull bgschaid/openfoam_by_ansible:training_pyfoam_prepare_ofw18
```

Getting the script

```
> wget https://bit.ly/ofw16docker -O runFoamContainer.sh
> chmod a+x runFoamContainer.sh
```

The actual URL for the script is `http://hg.code.sf.net/p/openfoam-extend/ansibleFoamInstallation/raw-file/226d8688cbaa/scripts/runFoamContainer.sh`

Starting the container

```
> ./runFoamContainer.sh bgschaid/openfoam_by_ansible:training_pyfoam_prepare_ofw18
```

After that you're on a shell inside the container

What `runFoamContainer.sh` does

The purpose of this script is to make using the Docker container as painless as possible

- Without an argument the script lists the locally available containers compatible with the script
- With an image name it starts the image in a new container
- mounts the working directory on the host machine to `/foamdata` on the container
  - data written to that directory is written to the host machine
    - and can be read during the next start of the machine
- Sets the user id of the user in the container to the id of the user on the host machine
  - Can read and write the same files as the host user

Starting the container

This demonstrates how data written inside the container is written to the host machine

```
❯ mkdir /tmp/work
❯ cd /tmp/work
❯ wget https://bit.ly/ofw16docker --quiet -O runFoamContainer.sh
❯ chmod a+x runFoamContainer.sh
❯ ./runFoamContainer.sh bgschaid/openfoam_by_ansible:training_pyfoam_prepare_ofw18
User dockeruser with UID: 2000 and GID: 2000
(OF:v2306-Opt) dockeruser@3646089fb8bf:/foamdata$ ls -l
total 4
-rwxrwxr-x 1 dockeruser dockeruser 693 Jul  7 10:47 runFoamContainer.sh
(OF:v2306-Opt) dockeruser@3646089fb8bf:/foamdata$ ls /Examples/
stage0_originalCase         stage2_templateParameters  stage5_coarseMesh     stage8_extendCoupled
stage10_parameterVariation  stage3_rhoSimpleFoam       stage6_meshCreate     stage9_parallel
stage1_addedCustomRegexp    stage4_plotlines           stage7_initPotential
(OF:v2306-Opt) dockeruser@3646089fb8bf:/foamdata$ cp -r /Examples/stage0_originalCase/ .
(OF:v2306-Opt) dockeruser@3646089fb8bf:/foamdata$ exit
exit
❯ ls -l
total 8
-rwxrwxr-x 1 bgschaid bgschaid  693 Jul  7 12:47 runFoamContainer.sh
drwxr-xr-x 5 bgschaid bgschaid 4096 Jul  7 12:48 stage0_originalCase

 /tmp/work                                                    23%  12:48:52
Emacs ✓  (OF:None) ❯
```

Figure: Docker container started and data written to local machine (version numbers differ)

How the containers were built

Further reading - unrelated

- The containers were built with Ansible
  - The roles for this are a side-project
- Basic explanation at `https://bit.ly/ans4Foam`
  - There is also a PDF-presentation there

Getting the Material

With docker

The docker image has the 10 stages in the directory /Examples

```
> ./runFoamContainer.sh bgschaid/openfoam_by_ansible:training_pyfoam_prepare_ofw18
User dockeruser with UID: 2000 and GID: 2000
> (OF:v2306-Opt) dockeruser@28437674bec7:/foamdata$ ls /Examples/
stage0_originalCase          stage2_templateParameters  stage5_coarseMesh      <brk>
      <cont>stage8_extendCoupled
stage10_parameterVariation   stage3_rhoSimpleFoam       stage6_meshCreate      <brk>
      <cont>stage9_parallel
stage1_addedCustomRegexp     stage4_plotlines           stage7_initPotential
```

Non-docker

The 4 main stages of the presentation are archived in a tar

- But it should be possible to reproduce everything from the slides

```
> wget https://bit.ly/pyFprepare23 -O  PyFoamPrepareCase_Genua2023_Material.tar.gz
> tar xvzf PyFoamPrepareCase_Genua2023_Material.tar.gz
stage0_originalCase.tar.gz
stage10_parameterVariation.tar.gz
stage1_addedCustomRegexp.tar.gz
...
```

Alternate URL: https://openfoamwiki.net/images/9/93/PyFoamPrepareCase_Genua2023_Material.tar.gz

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Format of the cases

- OpenFOAM-cases are directories with files
  - "The filesystem is the database"
- Basic structure are key / value dictionaries
- Quite old
  - But the current plain-text formats in use (`XML`, `JSON`, `YAML`) have no functional advantage
    - Whether they are easier to read or edit is subjective (we can have an emotional discussion about this during the social part of the Workshop)
    - "The OpenFOAM file format was XML before XML was mainstream"
- Advantages of the format:
  - Human-readable (self-documenting)
  - Order is not important
    - Unused information is ignored
- Disadvantages:
  - Huge number of files in a case (problem of the sysadmin)
  - Special cases (`boundary`-file) and not always unambiguous

Enhancements of the format

During the years the format got a number of enhancements that

- increased readability
- decreased the amount of redundancy

Those were:

regular expressions Allows specifying similar keys only once
- Common use: boundary conditions

#include pull in another file
- Common use: Setting "standard" conditions

macro substitution with $ insert dictionary entries in different places
- Common use: `$internalField` in boundary conditions

Further enhancements depend on the OpenFOAM-fork

Limitations of the data format

What the format can not do

- Conditions
  - "If we're using this solver use this setting. Otherwise the other one"
    - Newer versions of some forks can do that
- Iterations
  - "Insert 20 STL-files into this `snappyHexMeshDict`"

But that is OK. It's a data format. Not a programming language

Current case setup (non-GUI)

- Manually calling the utilities
  - Tedious
  - Can lead to errors
- Scripts (for instance `Allrun`)
  - Parameter files are modified by `sed` or `awk` (if they are)
    - Not very flexible
    - .... or a lot of boilerplate code
    - Only textual replacements in the files

`pyFoamPrepareCase.py` tries to make this more flexible and robust

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

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

How to get it

- The easiest way is to install it with `pip`

  ```
  pip install pyfoam
  ```

- or if you want to install it only for the current user

  ```
  pip install --user pyfoam
  ```

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

What else can PyFoam do for me?

- Write and read dictionaries from the command line
- Display the `blockMeshDict`
- Generate plots of Surfaces and sample lines
- Interact with paraView
- Control OpenFOAM-runs over the net

A proper introduction

A proper introduction to PyFoam (and swak4Foam) is `https://bit.ly/ofw17swakPyfoam` from the last workshop (full link `https://openfoamwiki.net/images/1/1a/BernhardGschaider-OFW17_swakPyFoamBasicTraining.pdf`)

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

The pitzDaily-tutorial

- One of the most popular tutorial cases in OpenFOAM
  - Used for a number of different solvers
- Backward-facing step
  - Described in a paper by *Pitz and Daily* with measurement data

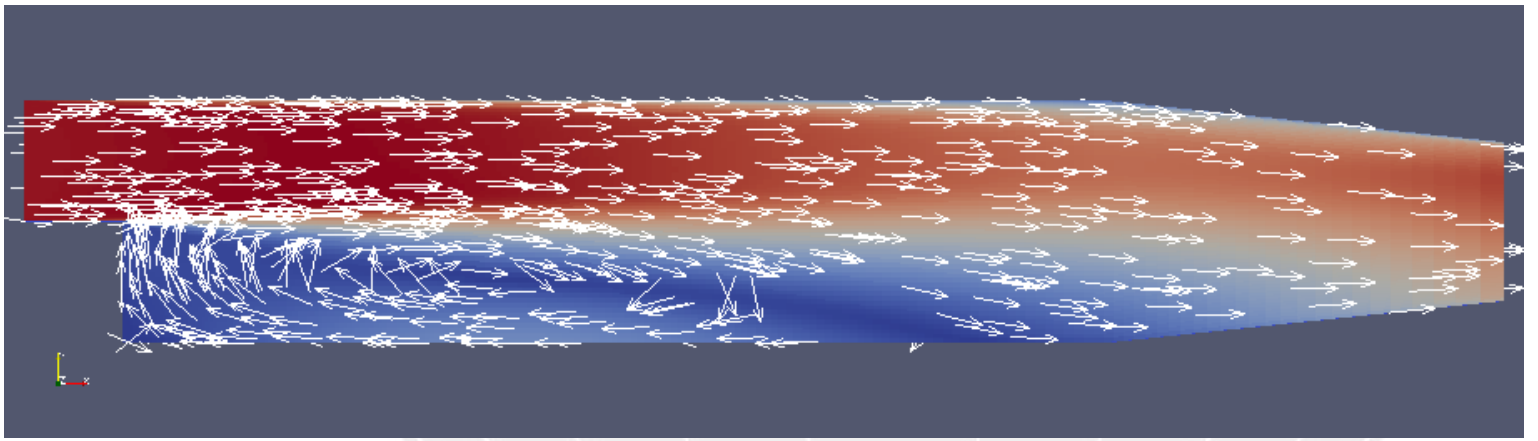


Figure: The pitzDaily-case

Measurement data on the case

From the original paper

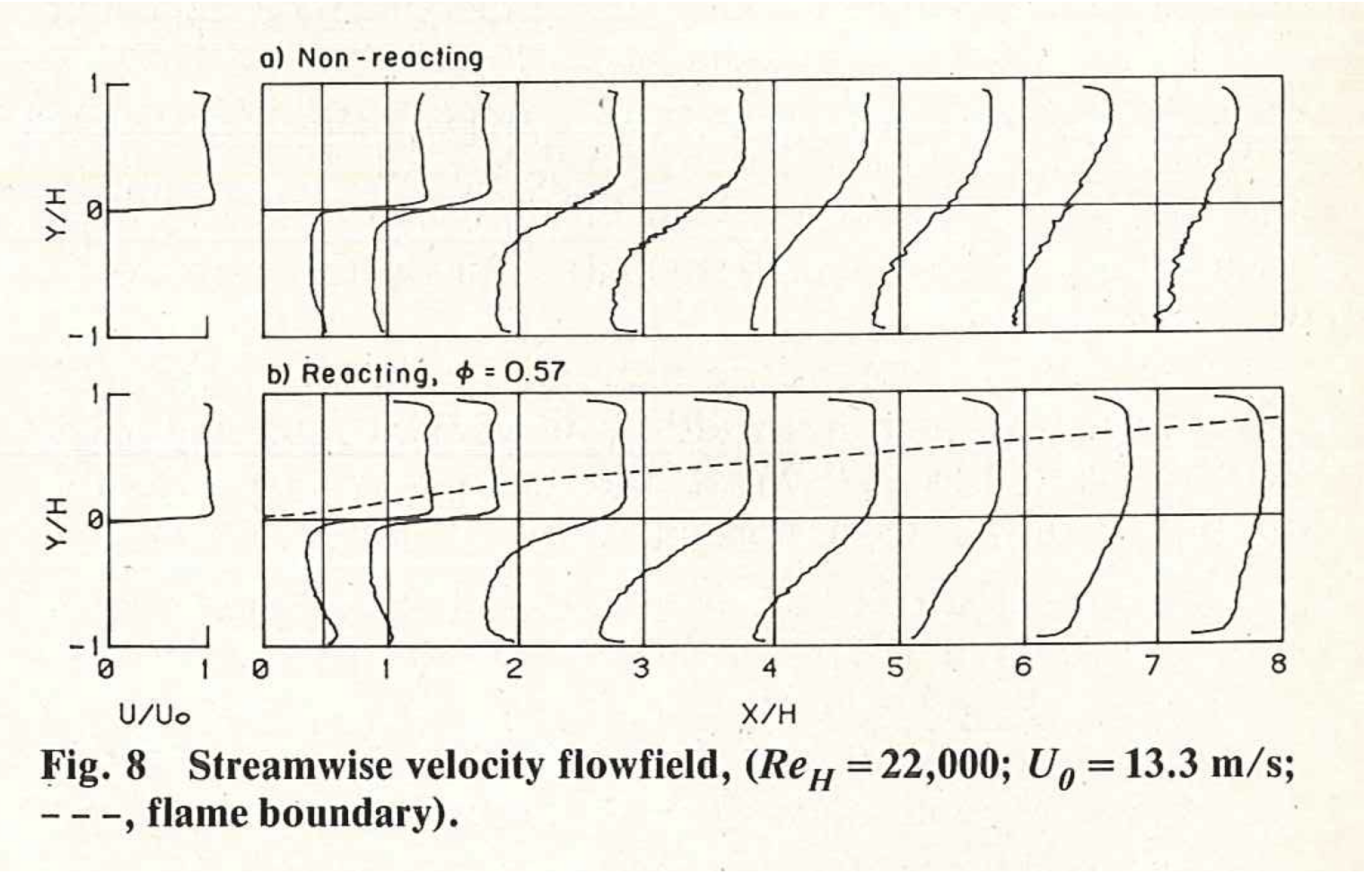


Fig. 8 Streamwise velocity flowfield, ($Re_H = 22{,}000$; $U_0 = 13.3$ m/s; – – –, flame boundary).

What we want to do with the case

- Modify the case so that we can compare
  - Different meshing strategies
  - Solvers
  - (Open)FOAM-versions
  - Influence of initial conditions
- In the end we want to be able to say "Set up the case with a coarse grid so that it runs with the coupled solver from `extend`. Go"
  - All from the same set of case-files

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Getting the base case

- Make sure we use the same version

```
> . ~/OpenFOAM/OpenFOAM-v2306/etc/bashrc
```

- Your version may differ
- Not necessary for the docker image
- Go to a common place and create our own workspace

```
> cd $HOME
> mkdir pitzDailyWithMod
> cd pitzDailyWithMod
```

- Get the files from the original case

```
> cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily/* .
```

Preparing it

- This is our first use of the utility
  - We didn't prepare the case for it

```
> pyFoamPrepareCase.py .
```

- A lot stuff is written to the terminal
  - Stuff about "looking for `template`-files"
    - Later we'll understand what this means
- `blockMesh` is automatically run
  - Because the utility found a `blockMeshDict`
    - The utility tries to be clever
    - ... but not too clever

Running

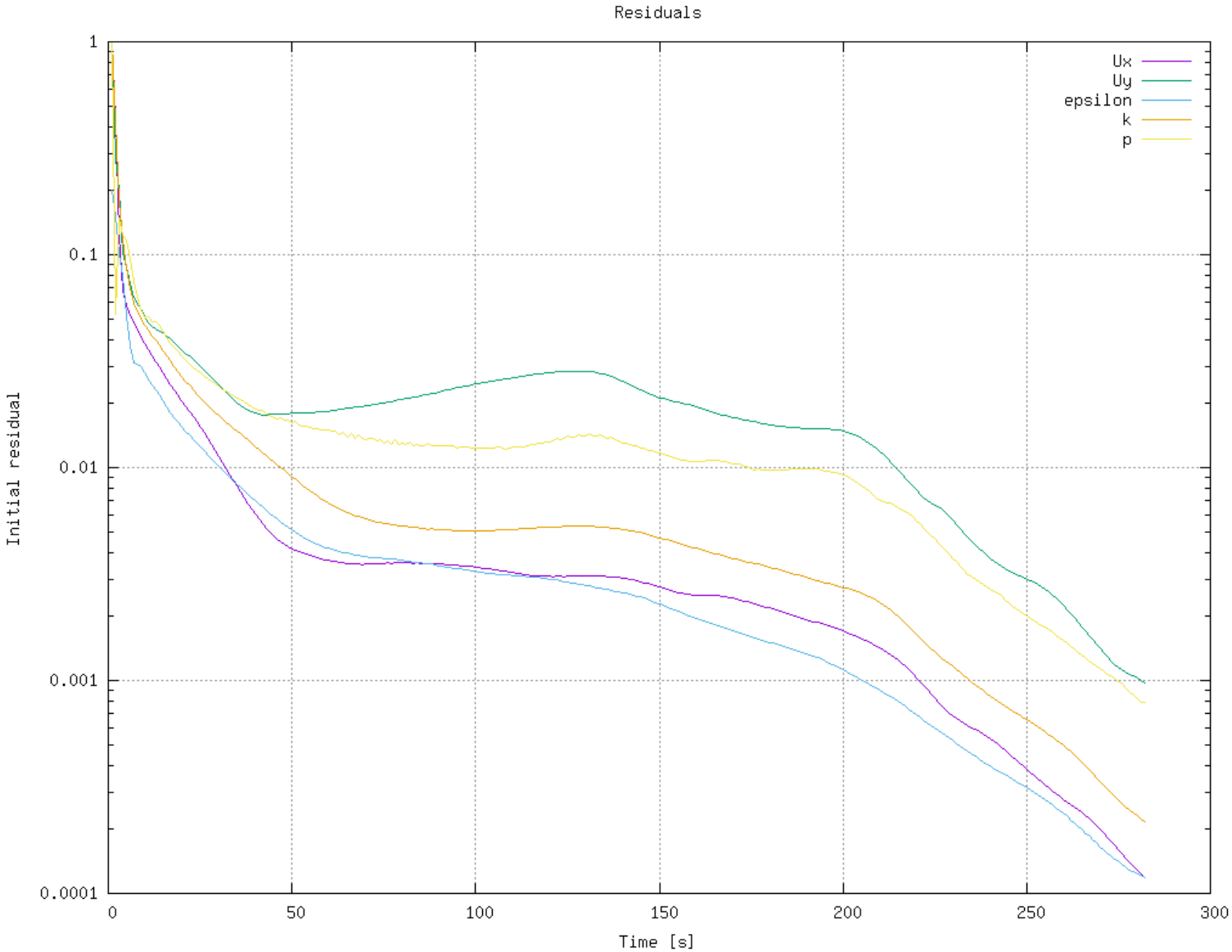
- The case is now ready to run
  - So we'll run it

```
> pyFoamPlotRunner.py --clear --progress --hardcopy --prefix=baseline auto
Clearing out old timesteps ....
Warning in /usr/local/bin/pyFoamPlotRunner.py : Replacing solver 'auto' with simpleFoam in <brk>
       <cont>arguments
 t =         282
```

- This pops up some windows with residuals etc
  - We generated hardcopies for later reference

Stage 0: `pyFoamPrepareCase.py` **without modifications**

Residuals



Spawn point

To get to the current state use `stage0_originalCase.tar.gz`

Spawn point

The point in a computer game that the player is "re-born" at if he "dies" during a level
Here this means "this is the way the files look after the past section and that the user will modify in the next section"

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

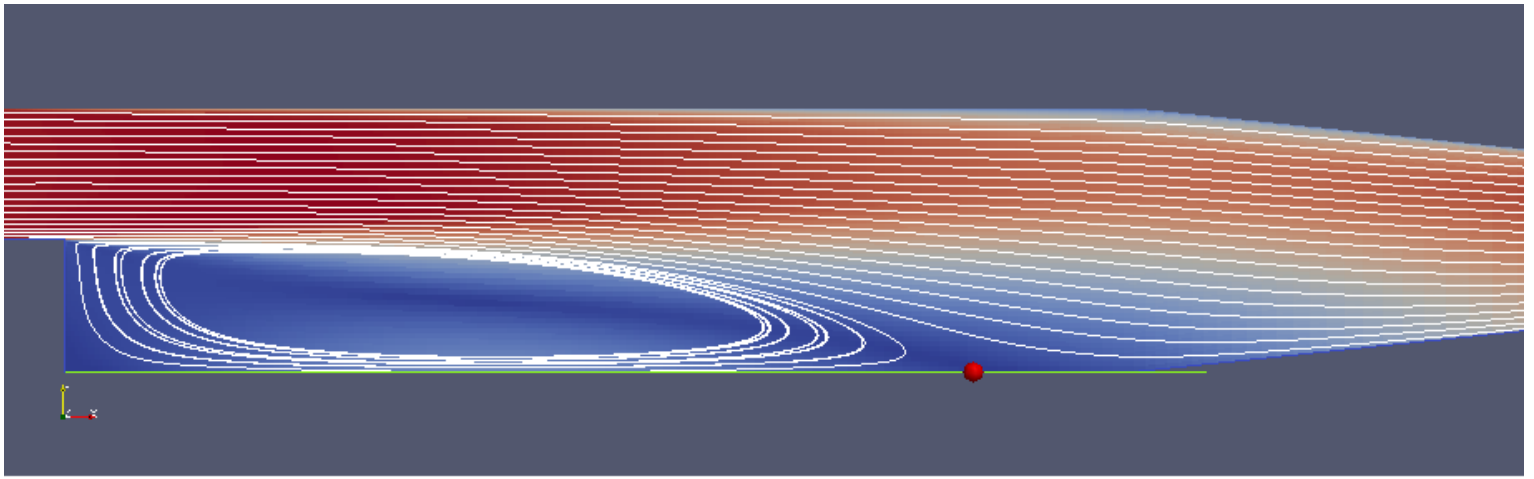
5 Conclusions

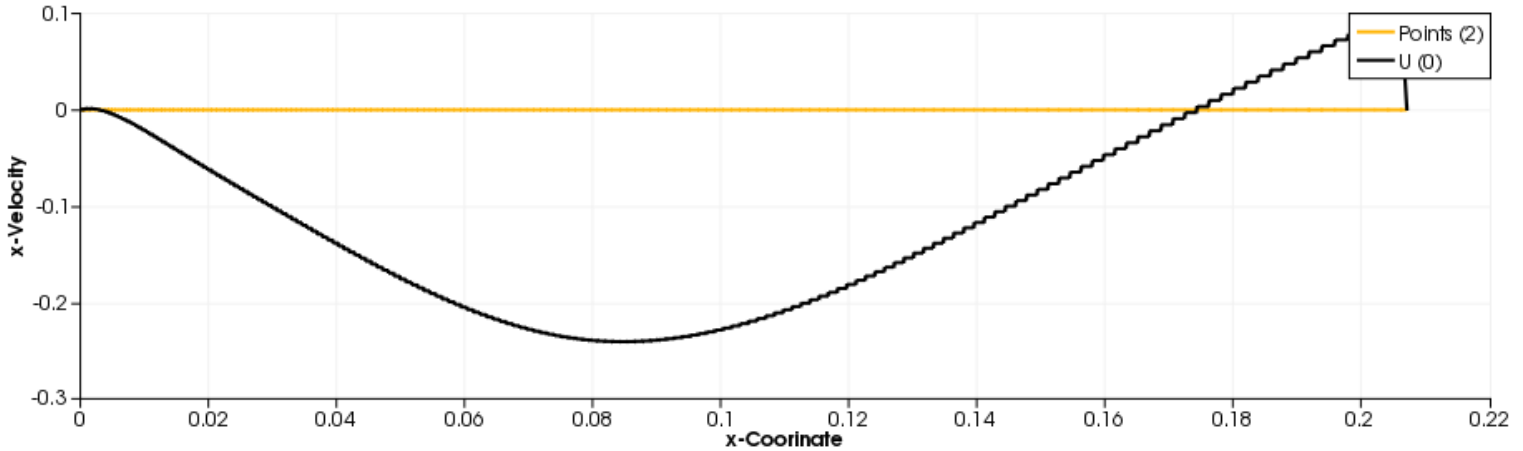
Adding a little

- Numerical convergence is not everything
  - Especially as the convergence criteria we use are quite low
    - But they were set that way in the tutorial
- To check how well converged the case is we add two evaluations
  1. Pressure difference between inlet and outlet
  2. The detachment point of the vortex behind the step
- We'll use `swak4Foam` to calculate these during the calculation

The detachment point

Definition: "Where the velocity over the lower wall becomes positive"





Adding swak-functionObjects

- Adding a few libraries to have the swak4Foam-functionality

controlDict

From now on a box (like this one) means "Add these files to the controlDict)

```
libs (
    "libsimpleSwakFunctionObjects.so"
    "libswakFunctionObjects.so"
    "libsampling.so"
);
```

Pressure difference

- Calculated by subtracting the average of the pressure on the two patches

controlDict in functions dictionary

```
pressureDiff {
    type patchExpression;
    patches (
        inlet
    );
    variables (
        "pOut{patch'outlet}=sum(p*area())/sum(area());"
    );
    accumulations (
        average
    );
    expression "p-pOut";
    verbose true;
}
```

Errata: this is the way it is done in the provided files. Probably using `weightedAverage` instead of `average` is more accurate

Find the detachment point

- On every face of the patch:
  - If the velocity is negative use the current position
  - Otherwise use a value "south" of the patch (-0.1)
- Get the maximum of all face-values (no pun intended)
  - This is the position of the detachment point
    - To be exact: half a face left of it ... but good enough

controlDict in functions

```
findPoint {
    type swakExpression;
    verbose true;
    valueType patch;
    patchName lowerWall;
    expression "internalField(U).x<0 ? pos().x : -0.1";
    accumulations (
        max
    );
}
```

Making PyFoam pick up the values

- swak4Foam writes the files to the console
  - We've got to tell PyFoam how to find them

customRegexp

```
deltaP {
        theTitle "Pressure difference";
        expr "Expression pressureDiff on inlet:  average=(%f%)";
        titles (avg);
        progress "dp: $0";
}
detatch {
        expr "Expression findPoint :  max=(%f%)";
        theTitle "Location of the detachment point";
        titles ( x );
        progress "detach: $0";
}
```

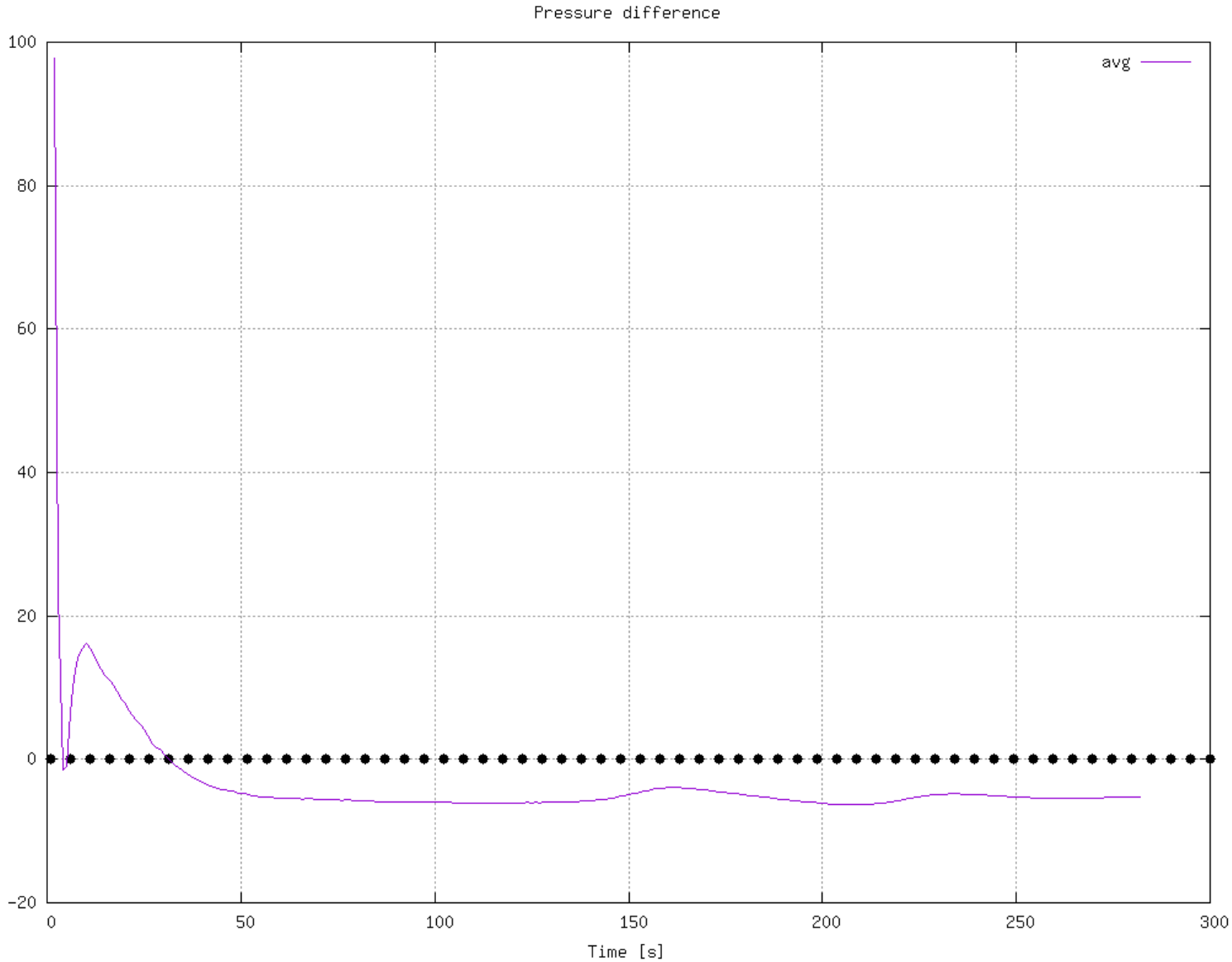
Re-running

- Now we re-run the case

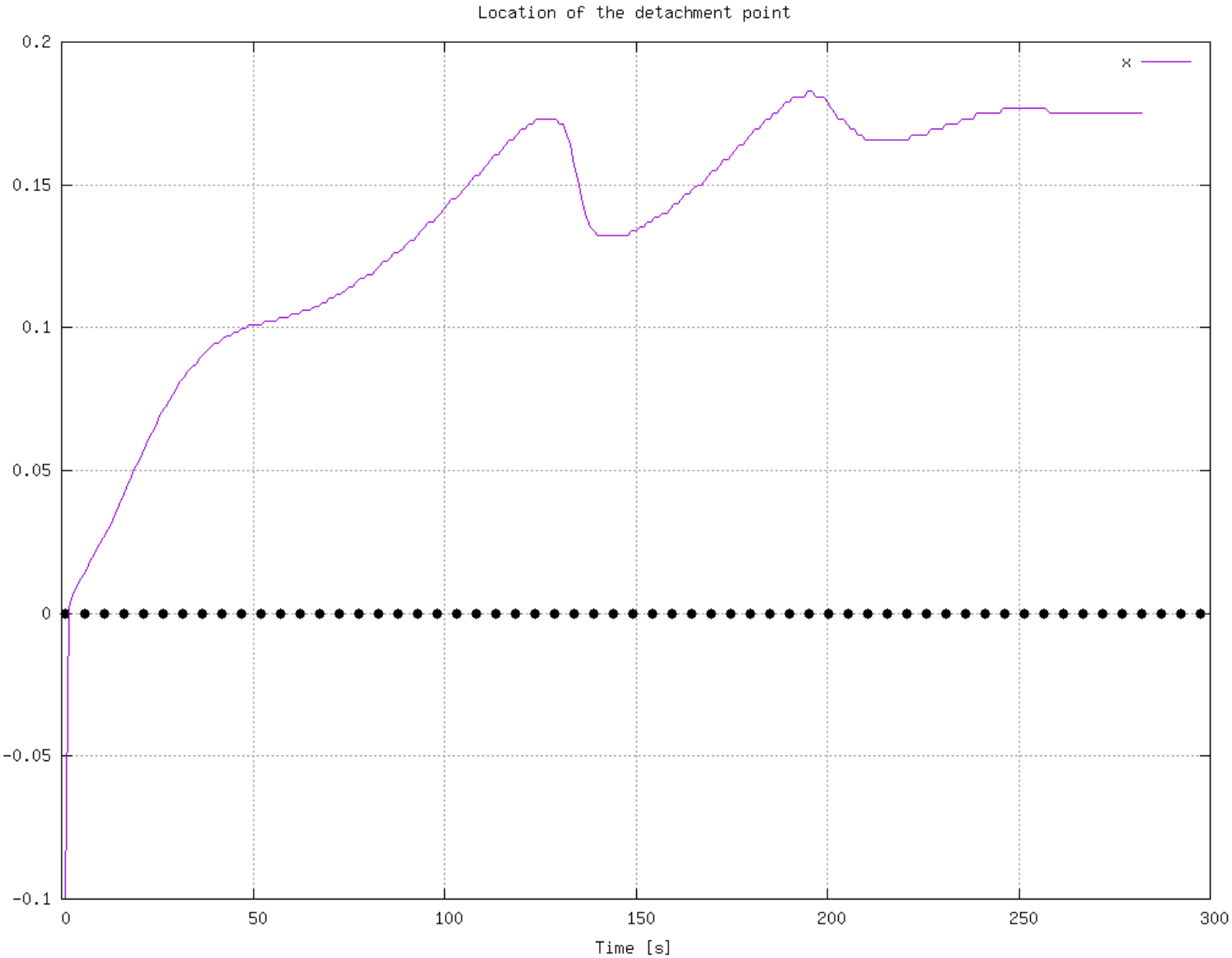
```
> pyFoamPlotRunner.py --clear --progress --hardcopy --prefix=baseline auto
 Reading regular expressions from /Volumes/Foam/Workshop2015/pitzDailyWorkshopStages/<brk>
       <cont>customRegexp
Clearing out old timesteps ....
Warning in /usr/local/bin/pyFoamPlotRunner.py : Replacing solver 'auto' with simpleFoam in <brk>
      <cont>arguments
Duration of pickling 0.021266698837280273 too long. Extending frequency from 1.0 to <brk>
      <cont>1.0633349418640137
 t =        282 dp: -5.19088 detach: 0.175012
```

- Needs the same amount of iterations
  - We didn't change anything about the setup
- Position and pressure difference are printed to the console
- We get nice plots of the evolution of these values

Pressure difference evolution



Evolution of the detachment point



Good enough

- Critique of the results
  - Pressure difference gets quite a kick in the beginning
    - This is typical for "unphysical" intial conditions
  - Detachment point still "searching for his destination" (this means: not converged)
  - Also: the numerical criteria (10^{-3}) are ... liberal
- But this is not the topic of this presentation
  - If you don't like the results: increase accuracy in `fvSolution` and rerun the examples

Spawn point

To get to the current state use `stage1_addedCustomRegexp.tar.gz`

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Phases of pyFoamPrepareCase.py

pyFoamPrepareCase does these things (in this order):

1. Clear old data from the case
2. Look for template-files and create real files from them
3. Create a mesh
4. Copy *original* files
5. Expand postTemplate-files
6. Execute a script for setting up initial conditions
7. Finally expand finalTemplate-files

We'll talk about all these things. But not in that order

- PyFoamPrepareCase also allows the execution of *decomposition scripts* after mesh generation and field setups
  - Because different mesh workflows need different decomposition strategies

Original files

- If `pyFoamPrepareCase.py` finds a file or directory with the extension `.org` it assumes that it should replace the extension-less file/directory with this
- We move the first time-step "out of the way"

```
> mv 0 0.org
```

- This makes sure that things we edit are not overwritten by OpenFOAM-utilities

Template files

- Files with the extension `.template` are handled by the Template engine built into PyFoam
  - Based on 3rd Party software (see below)
  - There now is an alternative to the built-in engine
- Definition of *Template engine* in Wikipedia:
  - "A template processor (also known as a template engine or template parser) is a piece of software or a software component that is designed to combine one or more templates with a data model to produce one or more result documents."
  - In our context:

    piece of software PyFoam (concrete: the class `TemplateFile`)

    templates the `.template` files

    result documents the case files
- For testing the template engine we can use the `pyFoamFromTemplate.py`-utility
  - Will be used on the next slides

First evaluation

- Strings beween |- and -| are evaluated by Python
  - The string representation of the result is inserted into the *result document*
- Available functions:
  - The regular `builtin`-functions of Python
  - All functions from the `math` module

testfile.template_

```
Sin(1) = |-sin(1)-|
```

Shell

```
> pyFoamFromTemplate.py --template-file=testfile.template_ --values-string="{}" --stdout
Sin(1) = 0.8414709848078965
```

Variables

- Variables are regular Python-variables
  - Primitive types: strings, number, booleans
  - Complex types: lists, dictionaries or objects
- Variables are usually set at the beginning
  - Can be used like regular global variables
- If a variable is used but has not been set the template evaluation fails
  - This can be changed: see online documentation for `--tolerant-expression-evaluation`
    - Usually not recommended

Adding a variable

- Now a variable `t` is used
- Variables can be passed to the utility (also `pyFoamPrepareCase.py`) with `--values-string`
  - This string is in Python-syntax
    - Not the OpenFOAM-syntax like everything else
    - But the `pyFoamPrepareCase.py` utility will accept OpenFOAM-syntax

testfile.template_

```
Sin(|-t-|) = |-sin(t)-|
```

Shell

```
> pyFoamFromTemplate.py --template-file=testfile.template_ --values-string="{'t':1}" --<brk>
      <cont>stdout
Sin(1) = 0.8414709848078965
```

Local variables

- These variables are local to a template file
- Allow breaking up of complex evaluations
  - Remove redundancy if an expression is needed more than once in the file
- Declaration of variable is in a line that starts with $$
  - This line will be completely removed from the *result document*
- The variable can be used from the line at which it was declared till the end of the template

Storing an intermediate Result

testfile.template_

```
$$ x=sin(t)
Sin(|-t-|) = |-x-| or |-sin(t)-|
```

Shell

```
> pyFoamFromTemplate.py --template-file=testfile.template_ --values-string="{'t':1}" --<brk>
      <cont>stdout
Sin(1) = 0.8414709848078965 or 0.8414709848078965
```

Assignments only

- In the usual syntax after `$$` only lines of the form `<value> = <expression>` are allowed
  - Other valid Python-lines (like `import os`) do not work
- A more general syntax can be switched on with `--allow-exec-instead-of-assignment`
  - This can also be switched on for a case in the `LocalConfigPyFoam`-file
- The reason why this is not switched on by default is for security concerns: Allowing general execution allows the template to do anything

Allowing execution

testfile.template

```
$$ import os
I am |-os.environ["USER"]-|

The machine: |-os.uname()-|
```

Shell

```
> pyFoamFromTemplate.py --template-file=testfile.template_ --values-string="{'t':1}" --stdout
 Error in /usr/local/bin/pyFoamFromTemplate.py : FatalError in PyFoam: 'PyFoam FATAL ERROR on line 129<brk>
       <cont> of file /usr/local/lib/python3.10/dist-packages/PyFoam/Basics/TemplateFile.py: Each <brk>
       <cont>definition must be of the form: <name>=<value> The string $$ import os is not. Try running<brk>
       <cont> the utility with the option --allow-exec-instead-of-assignment'
> pyFoamFromTemplate.py --template-file=testfile.template_{} --values-string="{'t':1}" --stdout --<brk>
      <cont>allow-exec
I am dockeruser

The machine: posix.uname_result(sysname='Linux', nodename='28437674bec7', release='6.2.0-10011-tuxedo<brk>
      <cont>', version='#14 SMP PREEMPT_DYNAMIC Wed Jun 28 18:29:09 UTC 2023', machine='x86_64')
```

What can be done between pipes

- Between |- and -| almost any regular Python-expression can be used
  - This includes *list comprehension* and *string formatting*
  - You can check on the Python-shell whether things work the way you think they should
- |- / -| can be inside a string of the template file
  - Will be replaced
  - No need to escape the " if the Python-expression needs a string
    - Hint: " and ' can be used interchangeably in Python. So there is almost never a need to escape strings (see above uses of `--values-string`)

Roles of " and '

- In

```
--values-string="{'t':1}"
```

the roles are

- `"` This is for the shell: "I'm only one argument. And don't try to interpret the {}"
- `'` This is for Python: "I am a string"

- How the string is interpreted can be always checked in Python:

Python-shell

```
> python
Python 3.10.6 (main, May 29 2023, 11:10:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> {'t':1}
{'t': 1}
>>>
```

Nice Python-tricks

testfile.template

```
$$ import os
The machine reformatted: |-",␣".join(os.uname())-|
Sum of squares: |-sum([i*i for i in range(100)])-|
Formatted: |-"%015.7E" % pi-|
$$ from os import path
Build path: |-path.join("theCase","system","controlDict")-|
```

Shell

```
> pyFoamFromTemplate.py --template-file=testfile.template --values-string="{'t':1}" --<brk>
      <cont>stdout --allow-exec
The machine reformatted: Linux, 28437674bec7, 6.2.0-10011-tuxedo, #14 SMP PREEMPT_DYNAMIC <brk>
      <cont>Wed Jun 28 18:29:09 UTC 2023, x86_64
Sum of squares: 328350
Formated: 003.1415927E+00
Build path: theCase/system/controlDict
```

The .template-files

This is what pyFoamPrepareCase.py does with them

- The case is searched recursively for .template-files
  - Each found file is evaluated as a template
  - The *result document* is saved under the same filename without the extension
    - If a file of that name was present it is overwritten
- The same set of variables is used for all templates

Setting the velocity

We now want to select the inlet velocity

1. Make the original velocity file a template

```
> mv 0.org/U 0.org/U.template
```

1. Replace 10 with a template expression

0.org/U.template

```
boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (|-UIn-| 0 0);
    }
```

The built-in engine: pyratemp

- The default template engine in PyFoam is `pyratemp`
  - `https://pypi.org/project/pyratemp/`
  - Very small source files
- The engine is included as a 3rd Party source
  - Makes deployment easier
    - Users can be sure "I have `PyFoam`. Then I have `pyratemp`"
- The syntax is a bit "special"

Alternative engine: Jinja2

- Jinja2 is a popular templating library
  - `https://palletsprojects.com/p/jinja/`
  - Used in popular python libraries like `Ansible`, `Flask`, ...
- Flexible syntax
  - And lots of information on the internet
- Easy to extend
- Newer versions of PyFoam include it as an alternate template engine
  - easier to use for people who already know it
  - easier to do certain things with it than with `pyratemp`
    - but some things are harder
- But
  - to big to be included into PyFoam
  - so the library has to be installed
    - for newer `PyFoam`-versions it is installed as a `requirement`
- Jinja does some things better and some things worse than `pyratemp`

Expressions in Jinja2

- The equivalent to |- and -| in pyratemp is {{ and }} in Jinja2
  - Expression between them is evaluated and the string is inserted in the result

testfile_jinja.template_

```
{{t}}/2 = {{t/2}}
```

Shell

Default is pyratemp. Jinja2 has to be explicitely specified

```
> pyFoamFromTemplate.py  --template-engine-default=jinja2 --template-file=testfile_jinja.<brk>
      <cont>template_ --values-string="{'t':3}" --stdout
3/2 = 1.5
```

0.org/U.template as a Jinja-template

```
inlet
{
    type            fixedValue;
    value           uniform ({{UIn}} 0 0);
}
```

Filters in Jinja2

- Something special to the Jinja2-syntax are filters
  - They are written with the *pipe* character

    `value | filter`
  - is equivalent to

    `filter ( value )`
  - but easier to read if more filters are applied

    `value | filter1 | filter2`

    vs

    `filter1(filter2(value))`
- many built-in filters

  `{{"llaw" | reverse | title}}`

  becomes

  `Wall`

Setting variables in Jinja2

- Special operations happen between {% and %}
- To set a variable `var` you write

  {% set var = <some expression> -%}
- The minus (-) near the end is optional
  - Means: Don't produce an empty line in the output

Importing Python libraries

- Jinja2 is not a programming language
  - so per default it doesn't allow programming stuff like "library importing"
    - but once the library is there it can be used
  - but we sometimes need libraries to do our stuff
- `PyFoam` adds a filter `import_module` that allows importing a Python module
  - has to be assigned to a variable to be used
    - a bit weird but the cleanest way to do it

Doing math

```
{% set math="math"|import_module -%}
{% set x=math.sin(t) %}
Sin({{t}}) = {{x}} or {{math.sin(t)}}
```

List comprehension and python execution in Jinja2

- Another "programming feature" of Python that is not supported by Jinja2 ist *list comprehension*
  - a compact way to write "simple loops"
- PyFoam adds the `python_eval`-filter to allow this (and other "complicated" constructs)
  - the program code is passed as a string to the filter
  - the filter executes it

Calculate a sum

```
{% set squared='[i*i for i in range(10)]'|python_eval -%}
Sum of squares: {{squared|sum}}
```

Handling OpenFOAM-data

PyFoam adds some filters to understand and generate the OpenFOAM file-format (using other parts of the PyFoam-library)

`parse_foam` parses string under the assumption that it is in OpenFOAM-format and returns the results as a Python data-structure

```
{% set original = "g (0 9.81 0);"|parse_foam %}
```

`parse_foam_file` takes a string and assumes that this is a filename. Reads and parses the file and returns Python-data

`format_foam` gets Python-data and produces a string in OpenFOAM file-format

Read and change relaxation

```
{% set relax=("system/fvSolution"|parse_foam_file).relaxationFactors -%}
{% set fields = relax.equations.update({"foo": 0.7}) -%}
Python: {{relax}}
Foam: {{relax|format_foam}}
```

Changing list and dictionary elements

- Another thing Jinja does not allow is the direct manipulation of list and dictionary elements
- for this PyFoam adds a filter: `python_exec`
  - works similarly to `python_eval`
- because this is potentially dangerous the `--allow-exec-instead-of-assignment` command-line option has to be used
- if you need this maybe `pyratemp` is the better choice

#+begin$_{src}$

Read and change relaxation

{% set original = "g (0 9.81 1);"|parse$_{foam}$ %} {% do "original['g'][2]=0"|python$_{exec}$ -%} Negative g: {{(-original.g)|format$_{foam}$}} #+end$_{src}$

Automatic selection of the used engine

- Both template engines can be used in the same project
  - but only one of them for a file
- The default is `pyratemp`. Unless
  1. other engine is set by the `--template-engine-default`
  2. a different engine is specified in `LocalConfigPyFoam`
- Before rendering a template `pyFoamPrepareCase` looks at the first lie of the file
  - if it finds the string `engine:` followed by an engine name there it uses that

`0.org/U.template` will be rendered by Jinja2

```
/*--------------------------------*- C++ -*----- engine:jinja2 --------------*\
| =========                      |                                            |
| \\      /  F ield              | OpenFOAM: The Open Source CFD Toolbox      |
|  \\    /   O peration          | Version:  v2212                            |
|   \\  /    A nd                 | Website:  www.openfoam.com                 |
|    \\/     M anipulation        |                                            |
\*---------------------------------------------------------------------------*/
```

Preparing the case

The output of `pyFoamPrepareCase.py` starts with list of the used variables

```
> pyFoamPrepareCase.py . --value="{'UIn':10}"
Using default values from /foamdata/stage2_templateParameters/default.parameters
Looking for template values .
Updating values {'UIn':10}
 Warning in /Users/bgschaid/Development/OpenFOAM/Python/PyFoam/bin/pyFoamPrepareCase.py : <brk>
       <cont>Values for which no default was specified: UIn

Used values

              Name - Value
-----------------------------------------
               UIn - 10
          caseName - "stage2_templateParameters"
          casePath - "/foamdata/stage2_templateParameters"
          foamFork - openfoamplus
       foamVersion - 2306
numberOfProcessors - 1

No script ./derivedParameters.py for derived values
Clearing .
Writing parameters to ./PyFoamPrepareCaseParameters
...
```

Python-format is "old school"

- Old versions of `pyFoamPrepareCase.py` only supported Python-format for `--value`

```
pyFoamPrepareCase.py . --value="{'UIn':10}"
```

- Current versions support a more "foamy" format
  - Looks like an excerpt from a dictionary
    - The semicolon is important

```
pyFoamPrepareCase.py . --value="UIn 10;"
```

- The utility will try both formats
  - Use the one that works

Pre-defined values

Some variables are automatically defined:

- **casePath** full file-system path to the current case
- **caseName** name of the case. Last component of `casePath`
- **foamVersion** version of the currently used (Open)FOAM installation
  - a Python-tuple
- **foamFork** which fork of FOAM this is (usually `openfoam`, `openfoamplus` or `extend`)
- **numberOfProcessors** On how many processors the case is going to be run
  - Can be set with the `--number-of-processors` option

Parameter files

- With many parameters using only `--values-string` would be tedious
- Variable values can be collected in parameter files
  - The syntax of these files is: regular OpenFOAM-dictionaries
    - No header necessary
- Multiple parameter files can be read
  - If there are multiple values for a variable: the last one wins
  - Values from `--values-string` overrule all

Default parameters

- If a file `default.parameters` is found in the case it is used first
  - Sets the default values for all parameters
- With the following file this is sufficient:

```
> pyFoamPrepareCase.py .
```

`default.parameters`

```
UIn 10;
```

Structured `default.parameters`

- `default.parameters` is special as it allows structuring the variables
  - This structuring is only for documentation purposes. It has no effect during replacement
- If a dictionary with the entries `description` and `values` is found it is assumed that this is a group of variables
  - Groups can be nested
- A dictionary with the entries `description` and `default` is assumed to be a verbose variable declaration
- Other dictionaries are assumed to be "real" dictionaries

Adding documentation

We now rewrite `default.parameters` with documentation on the variable

- Completely equivalent to the previous form
  - But more informative

`default.parameters`

```
boundary {
    description "Boundary conditions";
    values {
        UIn {
            description "Inlet velocity";
            default 10;
        }
    }
}
```

Slow inlet

pyFoamPrepareCase.py reports if a value was changed from the default value:

```
> pyFoamPrepareCase.py . --value="UIn 5;"
Using default values from /foamdata/stage2_templateParameters/default.parameters
Looking for template values .
Updating values UIn 5;

Used values

               Name - Value
-----------------------------------------
                UIn - 5
           caseName - "stage2_templateParameters"
           casePath - "/foamdata/stage2_templateParameters"
           foamFork - openfoamplus
         foamVersion - 2306
numberOfProcessors - 1

No script ./derivedParameters.py for derived values
Clearing .
Writing parameters to ./PyFoamPrepareCaseParameters
Writing report to ./PyFoamPrepareCaseParameters.rst
Found 0.org. Clearing 0
...
```

Reporting the used parameters

- The used variable values are reported in two files

  PyFoamPrepareCaseParameters a OpenFOAM-dictionary with all the values

  PyFoamPrepareCaseParameters.rst A *ReStructured Text*-file
  - This is a markup-language
- The *ReST*-file can be converted to more readable form with utilities like

```
> rst2pdf PyFoamPrepareCaseParameters.rst
```

(HTML-export also possible)

A printed report

Contents

1 Overwritten parameters 1
2 Parameters with defaults 1
2.1 Boundary conditions 1
2.2 Automatically defined values 1

1 Overwritten parameters

These parameters were set from the command line

UIn

5

2 Parameters with defaults

2.1 Boundary conditions

Short name: `boundary`

	default	description	Value
UIn	10	Inlet velocity	**5**

Figure: Part of the generated PDF

Regular parameter files

- Variables can be collected in parameter-files
  - These files are read with the `--parameter-file`-option

`slow.parameters`

```
UIn 5;
```

Using the file

```
> pyFoamPrepareCase.py . --parameter-file=slow.parameters
Using default values from /foamdata/stage2_templateParameters/default.parameters
Looking for template values .
Reading values from slow.parameters
```

Spawn point

- To get to the current state use `stage2_templateParameters.tar.gz`
- The template examples can be found in
  - `testfile.template_`
  - `testfile_jinja.template_`
  - `testfile_jinja_advanced.template_`

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Why control structures?

- Currently templates are only a "glorified" sed (the stream editor. Old people know it)
  - Values can be inserted, but ...
  - No decisions can be made
  - If we want to repeat something we've got to put it into the template multiple times
    - And we can't change the number
- Control structures enable programming languages to
  - Take decisions
  - Repeat things
- We need some of those

Changing the solver

- We want to select the incompressible and the compressible variant of `simpleFoam`
  - We add a parameter for this

`default.parameters`

```
solver {
    description "Used solver";
    default simpleFoam;
    options (
        simpleFoam
        rhoSimpleFoam
    );
}
```

But what is this `options`-entry for?

options for parameters

- options allows us to specify parameters that only have a finite list of choices
  - The value of options is a list with these choices
- If the user specifies a value for that parameter that is not in the list pyFoamPrepareCase.py fails with an error message
  - In our case the user can't specify interFoam as the solver

Variables derived from parameters

- Sometimes you want parameters that depend on the parameters you set
  - Here we need to distinguish between compressible and incompressible solvers
    - This is important if we're going to support more than 2 solvers
- If present the Python-file `derivedParameters.py` will be executed after all the parameters are read
- Python-variables that are created here will be available in all the templates

`derivedParameters.py`

```
if solver in ["rhoSimpleFoam"]:
    compressible=True
else:
    compressible=False
```

Testing the values

- Before really setting up the case we want to check if the values are set correctly
  - `--only-variables` reads the parameters and prints them
  - Then stops

```
> pyFoamPrepareCase.py . --only-variables
Using default values from /foamdata/stage3_rhoSimpleFoam/default.parameters
Looking for template values .

Used values

              Name - Value
-----------------------------------------
               UIn - 10
          caseName - "stage3_rhoSimpleFoam"
          casePath - "/foamdata/stage3_rhoSimpleFoam"
          foamFork - openfoamplus
       foamVersion - 2306
numberOfProcessors - 1
            solver - simpleFoam

Deriving variables in script ./derivedParameters.py
Added values: compressible
```

Kinematic vs Dynamic

- One problem people usually have with the incompressible FOAM-solvers is that the density has been "canceled out"
  - Also known as the "why is my pressure drop 40% wrong"-problem (answer: because your density is 1.4)
- This means for incompressible:
  1. We use the *kinematic viscosity* $\nu = \frac{\mu}{\rho}$ instead of the *dynamic* μ
  2. Pressure dimensions are different (divided by $\frac{kg}{m^3}$
- to adapt the files we use the `compressible`-parameter

Preparing for compressible solvers

- We copy some additional files from a compressible tutorial case
  - And make them templates
- We modify files that are affected by the different dimensions to be templates

```
> cp $FOAM_TUTORIALS/compressible/rhoPimpleFoam/les/pitzDaily/constant/<brk>
    <cont>thermophysicalProperties constant/thermophysicalProperties.template
> cp $FOAM_TUTORIALS/compressible/rhoPimpleFoam/les/pitzDaily/0/T 0.org/T.template
> mv 0.org/p 0.org/p.template
> mv 0.org/k 0.org/k.template
> mv 0.org/epsilon 0.org/epsilon.template
> mv constant/transportProperties constant/transportProperties.template
```

Properties of air

Adding material properties as parameters

default.parameters

```
air {
    description "Properties of air";
    values {
        dynVisc {
            description "Dynamic viscosity";
            default 1.8e-5;
        }
        molarWeight {
            description "Molar weight of air";
            default 28.9;
        }
        TAir {
            description "Temperature of the air";
            default 293;
        }
        pAir {
            description "Pressure of the air on the outlet";
            default 1e5;
        }
    }
}
```

Setting the new nu

- Instead of the previous value we now calculate the *kinematic viscosity*
  - From the dynamic viscosity (a parameter `dynVisc`)
  - From the density `rhoAir`
    - Which is calculated from other parameters (`TAir`, `molarWeight`) by the *perfect gas* equation

constant/transportProperties.template

```
$$ rhoAir=pAir/(TAir*8.1415e3/molarWeight)

// With rho=|-rhoAir-|
nu              nu [ 0 2 -1 0 0 0 0 ] |-dynVisc/rhoAir-|;
```

Slides vs example files

- The following examples on the slides will by in `pyratemp`
- Most of the files in the examples use Jinja2 because
- Converting the formats is "left to the reader as an exercise"
  - This is what textbooks usually say if the writer was to lazy to do updates

`LocalConfigPyFoam` sets Jinja2 as a default

```
[Template]
fallbackengine: jinja2
```

`constant/transportProperties.template`

```
{% set rhoAir=pAir/(TAir*8.1415e3/molarWeight) -%}

// With rho={{rhoAir}}
nu              nu [ 0 2 -1 0 0 0 0 ] {{dynVisc/rhoAir}};
```

Setting compressible properties

- The material properties are directly inserted here

constant/thermophysicalProperties.template

```
mixture
{
    specie
    {
        nMoles          1;
        molWeight       |-molarWeight-|;
    }
    thermodynamics
    {
        Cv              712;
        Hf              0;
    }
    transport
    {
        mu              |-dynVisc-|;
        Pr              0.7;
    }
}
```

`if` in pyratemp templates

- All control structures in our templates are enclosed in `<!--(` and `)-->`
  - This syntax has its origins in HTML (comments)
- Apart from that the syntax (and behavior) is similar to the `if` in Python
  - First keyword `if`. Then a condtion
- The `if`-block is ended with a `<!--(end)-->`
  - The `<` has to be in the same column as the `<` of the starting `if`
- The text between `if` and `end` is only inserted if the condition is `True`
- It is also possible to have an `<!--(else)-->`-branch
  - Inserted if the condition is false
  - Or cascading with `<!--(elif ..)-->`

Trivial if-example

- Selecting text and setting a variable depending on an expression

testfile.template

```
<!--(if 1>2)-->
$$ val=True
Falsch
<!--(else)-->
$$ val=False
Richtig
<!--(end)-->
val: |-val-|
```

Shell

```
> pyFoamFromTemplate.py --template-file=testfile.template_ --stdout
Richtig
val: False
```

Branches in Jinja2

- They
  - start with {% if ... %}
  - end with {% endif %} (not just end)
  - have alternatives with {% else %}
  - additional conditions with {% elif ... %}

Branch in Jinja2

```
{% if 1>2 -%}
{% set val=True -%}
Richtig
{% else -%}
{% set val=False -%}
False
{% endif %}
val: {{val}}
```

Different pressure dimensions

- Distinguish between compressible and incompressible solvers

0.org/p.template

```
<!--(if compressible)-->
dimensions      [1 -1 -2 0 0 0 0];
internalField   uniform |-pAir-|;
<!--(else)-->
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform 0;
<!--(end)-->

boundaryField
{
    outlet
    {
        type            fixedValue;
        value           $internalField;
    }
```

Attention: the `$internalField` is important (used to be 0)

Single line `if`

- Wall functions have a prefix `compressible::` in their compressible form
- The "single line" `if` allows a very compact code
  - But like the C++ conditional `a ? b : c` it shouldn't be overused as it tends to make things unreadable

0.org/k.template

```
$$ prefix="compressible::" if compressible else ""

boundaryField
{
    upperWall
    {
        type            |-prefix-|epsilonWallFunction;
        value           uniform 14.855;
    }
    lowerWall
    {
        type            |-prefix-|epsilonWallFunction;
        value           uniform 14.855;
    }
```

Same Problem for k

0.org/k.template

#+begin_src c++ {% set prefix="compressible::" if compressible and foamVersion<(3,) else "" %}
boundaryField { upperWall { type {{prefix}}kqRWallFunction; value uniform 0.375; } lowerWall { type {{prefix}}kqRWallFunction; value uniform 0.375; } #+end_src

T means 'trivial'

- The only thing we do here is add the `TAir` we used to calculate the density

0.org/T.template

```
internalField   uniform |-TAir-|;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           $internalField;
    }

    outlet
    {
        type            inletOutlet;
        inletValue      $internalField;
        value           $internalField;
    }
```

Adding discretization schemes

- rhoSimpleFoam needs a few additional schemes

system/fvSchemes

```
divSchemes
{
    default          none;
    div(phi,U)       bounded Gauss upwind;
    div(phi,k)       bounded Gauss upwind;
    div(phi,epsilon) bounded Gauss upwind;
    div(phi,R)       bounded Gauss upwind;
    div(R)           Gauss linear;
    div(phi,nuTilda) bounded Gauss upwind;
    div((nuEff*dev(T(grad(U))))) Gauss linear;

    // Added for rhoSimpleFoam
    div((muEff*dev2(T(grad(U))))) Gauss linear;
    div(phi,e)       bounded Gauss upwind;
    div(phi,Ekp)       bounded Gauss upwind;
}
```

Adding support for energy equation e

system/fvSolution

```
    "(U|k|epsilon|R|nuTilda|e)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0.1;
    }
}
SIMPLE
{
    nNonOrthogonalCorrectors 0;
    rhoMin          rhoMin [ 1 -3 0 0 0 ] 0.5;
    rhoMax          rhoMax [ 1 -3 0 0 0 ] 1.5;

    residualControl
    {
        p               1e-2;
        U               1e-3;
        e               1e-3;
        "(k|epsilon|omega)" 1e-3;
    }
}
```

Running Compressible

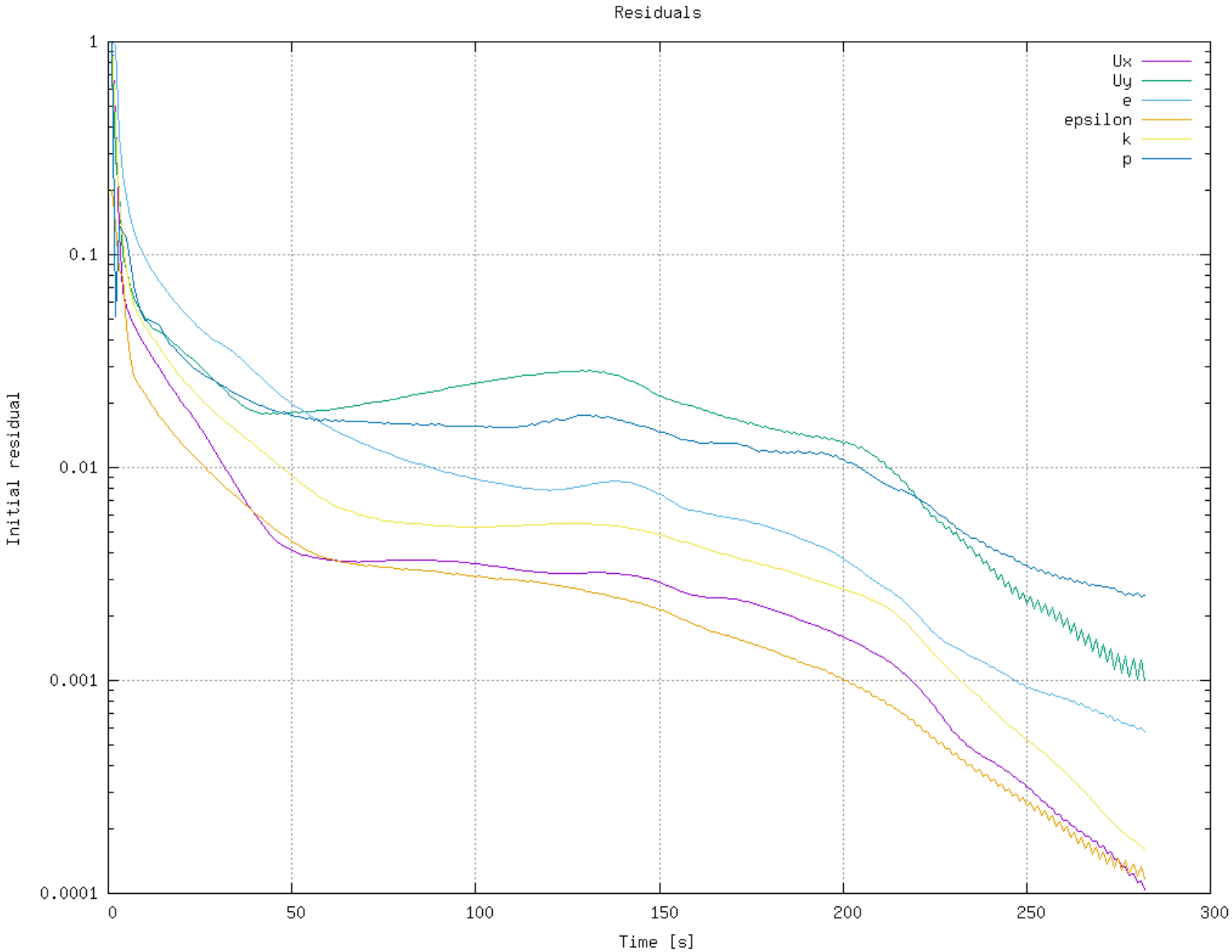
- We prepare and run for/with `rhoSimpleFoam`

```
> pyFoamPrepareCase.py . --values="solver rhoSimpleFoam;"
...
> pyFoamPlotRunner.py --clear --progress auto
 Reading regular expressions from /foamdata/stage3_rhoSimpleFoam/customRegexp
Clearing out old timesteps ....
 Warning in /usr/local/bin/pyFoamPlotRunner.py : Replacing solver 'auto' with rhoSimpleFoam<brk>
        <cont> in arguments
 t =         282 dp: -5.08118 detach: 0.173138
```

- The values for `simpleFoam` now have changed (because `nu` changed from 10^{-5} to $1.48 \cdot 10^{-5}$

```
> pyFoamPrepareCase.py . --values="solver simpleFoam;"
...
> pyFoamPlotRunner.py --clear --progress simpleFoam
 t =         278 dp: -4.38497 detach: 0.173138
```

Compressible residuals



Exercise: Pressure difference

- Pressure differences are different
  - Reason: Factor ρ (≈ 1.2) is missing in the incompressible case
- Your mission: make sure that in the incompressible case the "real" pressure is used
  1. Turn `controlDict` into a template
  2. Multiply the `expression` in `pressureDiff` with ρ
     - But only if not `compressible`
     - Use formula for ρ from `transportProperties.template`

Spawn point

To get to the current state use `stage3_rhoSimpleFoam.tar.gz`

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Repetitive tasks

- Computers are good at doing the same over and over again
  - Humans are not
    - And they make mistakes
- The `for`-loop is similar to the Python `for` (except for the `<!--( )-->` around it)
  1. Keyword `for`
  2. Name of the (running) variable
  3. Keyword `in`
  4. A list
     - Frequently: `range(5)` for numbers from 0 to 4

The for-loop

Printing square numbers and summing them

testfile.template

```
$$ total=0
<!--(for i in range(5))-->
$$ sq=i*i
$$ total+=sq
|-i-|*|-i-| = |-sq-|
<!--(end)-->
Total: |-total-|
```

Shell

```
0*0 = 0
1*1 = 1
2*2 = 4
3*3 = 9
4*4 = 16
Total: 30
```

Loops in Jinja2

- They start with {% for .... %}
- They end with {% endfor %}

loopstest.template

```
{% set ns = namespace(total=0) -%}
{% for i in range(5) -%}
{% set ns.total = ns.total + i*i -%}
{{i}} * {{i}} = {{i*i}}
{% endfor %}
Total: {{ns.total}}
```

The namespace is necessary because of the scoping rules of Jinja2 (see documentation)

Adding multiple sample lines

- We want to sample pressure and velocity on multiple lines in our geometry
- All the lines should be vertical
- Instead of specifying them by hand we make `controlDict` a template:

```
> mv system/controlDict system/controlDict.template
```

Specifying x-values to sample

- The locations of the lines are specified in a parameter
  - This way the number can be easily modified

`default.parameters`

```
postproc {
    description "Postprocessing stuff";
    values {
        xProfiles {
            description "List of locations of the profiles in mm";
            default ( -10 0 50 100 150 200 250 );
        }
    }
}
```

- In `controlDict` the sets are generated by a loop over `xProfiles`
  - Names of the sets are generated from the values

Adding the lines

sytem/controlDict.template

```
functions
{
    profiles {
        type sets;
        fields (
            U
        );
        outputControl   outputTime;
        interpolationScheme cell;
        setFormat raw;
        sets (
<!--(for x in xProfiles)-->
$$ xVal=1e-3*x
            x|-"%04.0f"%x-|
            {
                type midPoint;
                axis y;
                start ( |-xVal-|  -0.0255 0);
                end   ( |-xVal-|   0.0255 0);
            }
<!--(end)-->
        );
    }
```

Preparing and running

- This gives the same results as before
  - Except for the added profiles

```
> pyFoamPrepareCase.py . --values="solver simpleFoam;"
...
> pyFoamPlotRunner.py --clear --progress simpleFoam
 t =         278 dp: -4.38497 detach: 0.173138
> ls postProcessing/profiles/
100   200   278
```

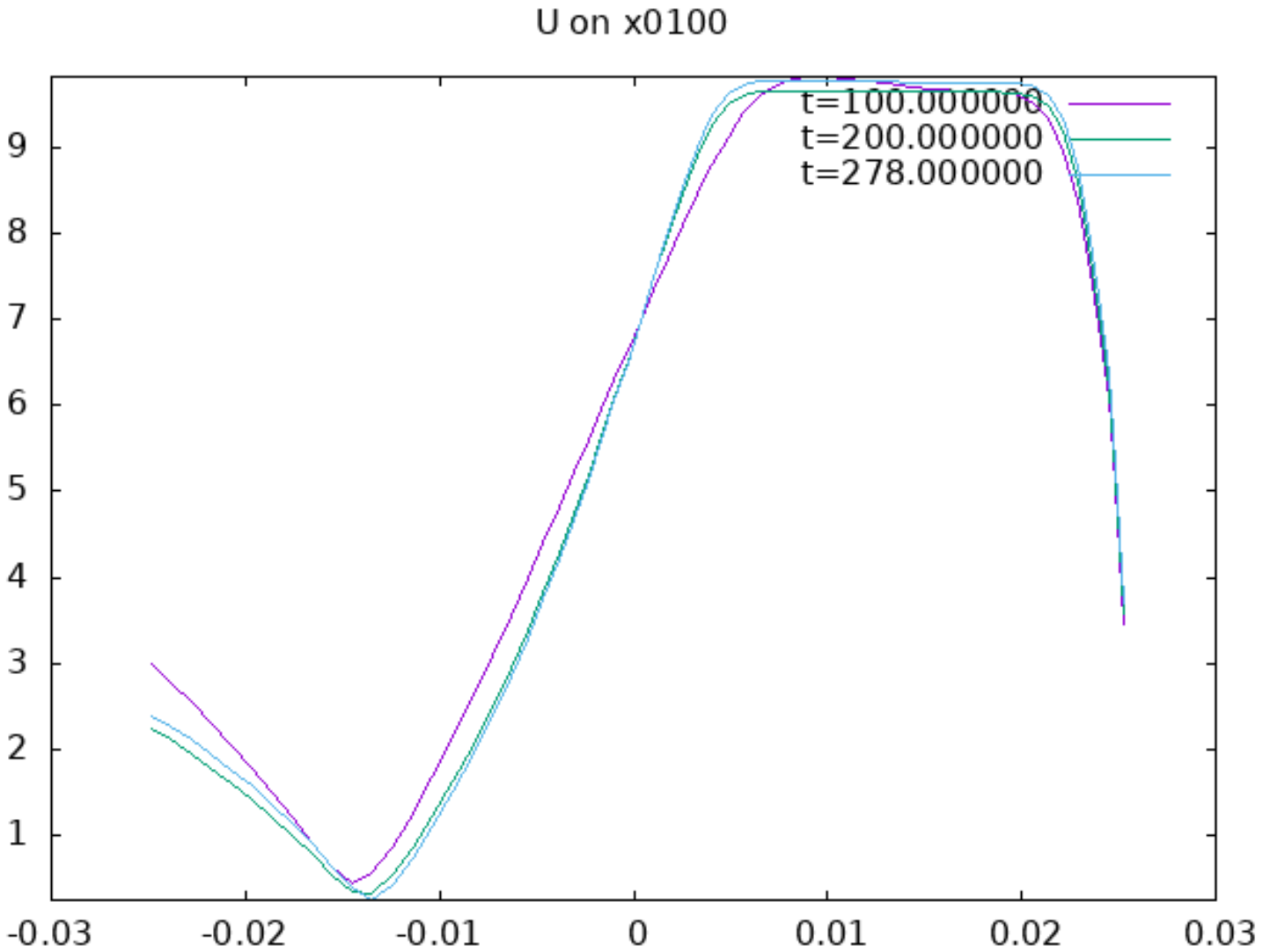
- Plotting 3 profiles can be quite . . . tedious
  - But PyFoam offers a utility for that

Plotting the sample data

- The `pyFoamSamplePlot.py` knows how to handle profile data
  - It can give information about the data present
  - Generate Gnuplot-commands to plot the data
    - Just pipe into GnuPlot
  - Collect the data and write into an Excel-file
- With all these things it is flexible about missing fields

```
> pyFoamSamplePlot.py . --dir=postProcessing/profiles --info
Times :  ['100', '200', '278']
Lines :  ['x-010', 'x0000', 'x0050', 'x0100', 'x0150', 'x0200', 'x0250']
Fields:  ['U']
> pyFoamSamplePlot.py . --dir=postProcessing/profiles --line=x0100 --mode=timesInOne | <brk>
      <cont>gnuplot
> ls *.png
postProcessing_profiles_x0100_U.png
> pyFoamSamplePlot.py . --dir=postProcessing/profiles --line=x0100 --mode=timesInOne --<brk>
      <cont>excel-file=simpleFoam_Line0100.xlsx
```

Velocity at different times



Further info on templates

- As the base of the template engine an independent library was "borrowed":
  `http://www.simple-is-better.org/template/pyratemp.html`
  - The reason for the <!---syntax is that this library was intended for web-development
- The library is included with the sources of PyFoam
  - Therefor there is no need to install an external dependency
    - This is also the main reasons why this and not a more popular templating engine like `Jinja2` is used

Further info on Jinja2

- The website of Jinja2 ist `https://jinja.palletsprojects.com/`
- The most important info for users is the *Template Designer Documentation*
  `https://jinja.palletsprojects.com/en/3.1.x/templates/`

Nesting of controls

- If control structures are nested then it is important that the inner structures are indented
  - <-- opening and closing the structure must be aligned
- In the following example there is branch inside a loop

testing.template

```
<!--(for p in ["in","out","topWall","wallBottom"])-->
  <!--(if p.lower().find("wall")>=0)-->
|-p-| is a wall
  <!--(end)-->
<!--(end)-->
```

Shell

```
topWall is a wall
wallBottom is a wall
```

Nesting of controls in Jinja2

- Jinja2 doesn't care about indentation
  - Enough said

Spawn point

To get to the current state use `stage4_plotlines.tar.gz`

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

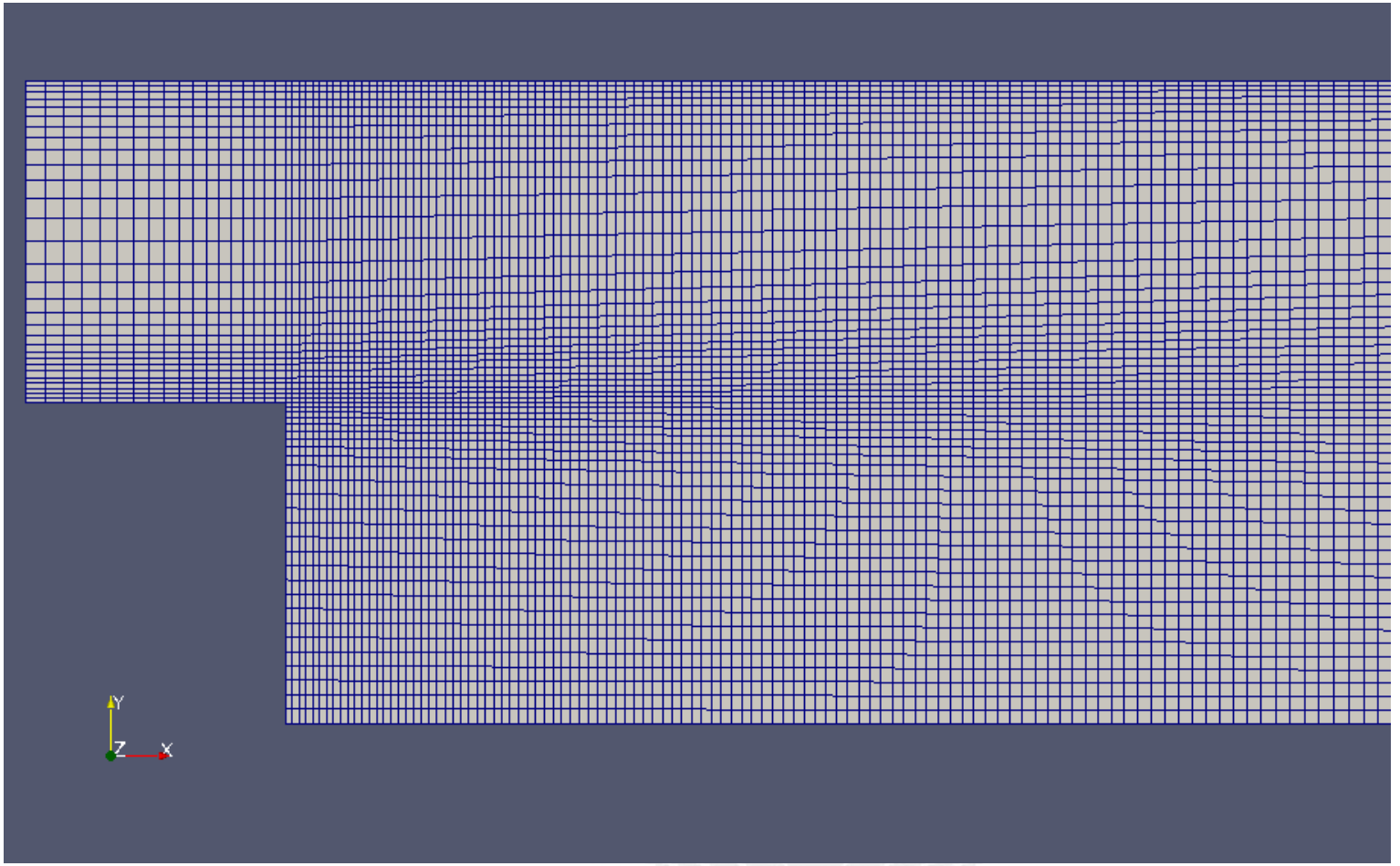
4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Automatic mesh creation

- Until now we didn't have to do anything to create a mesh
  - The mesh was just there
- Reason is that if `pyFoamPrepareCase.py` has no information about the mesh creation it assumes `blockMesh`
- *Information about mesh creation* means
  - A script `meshCreate.sh` exists!
- `meshCreate.sh` is executed instead of `blockMesh`
  - That is not a big improvement over `Allrun`
  - but
  - `meshCreate.sh` can be a `.template`
    - Those are expanded before the mesh creation phase
- We're now going to write a `meshCreate.sh.template`
  - But first we'll just modify `blockMeshDict`

The original mesh



Preparing blockmeshDict.template

- The original mesh is
  - Graded
  - Fine
- We want to check the influence of
  - Grading
  - The fineness

```
> mv system/blockMeshDict system/blockMeshDict.template
```

Adding parameters for blockMesh

- Adding two parameters
  - One to multiply the number of cells in each block with
  - One to switch off grading
- Defaults are to reproduce the original mesh

default.parameters

```
meshing {
    description "Meshing parameters";
    values {
        resFactor {
            description "Resolution compared to the original";
            default 1.;
        }
        graded {
            description "Whether mesh grading should be used";
            default true;
        }
    }
}
```

Calculating mesh number

- Add a few variables
  - One for each number of cells in the blocks
- Simply multiply with the factor
  - `ceil` rounds up (avoids zero cell blocks)
  - Result is a `float`. `int` 'casts' back to what Foam expects

`blockMeshDict.template`

```
$$ nrX1=int(ceil(18*resFactor))
$$ nrX2=int(ceil(180*resFactor))
$$ nrX3=int(ceil(25*resFactor))
$$ nrY1=int(ceil(30*resFactor))
$$ nrY2=int(ceil(27*resFactor))
```

Jinja

```
{% set nrX1=math.ceil(18*resFactor)|int -%}
{% set nrX2=math.ceil(180*resFactor)|int <brk>
    <cont> -%}
{% set nrX3=math.ceil(25*resFactor)|int -%}
{% set nrY1=math.ceil(30*resFactor)|int -%}
{% set nrY2=math.ceil(27*resFactor)|int -%}
```

Defining the blocks

- Use the mesh number variables from above
- Using an inline `if` to switch off grading

blockMeshDict.template

```
$$ posYRStr="$posYR"

blocks
(
    hex (0 3 4 1 11 14 15 12)
    (|-nrX1-| |-nrY1-| 1)
    simpleGrading |-"(0.5 "+posYStr+" 1)" if graded else "(1 1 1)"-|

    hex (2 5 6 3 13 16 17 14)
    (|-nrX2-| |-nrY2-| 1)
    |-"edgeGrading (4 4 4 4 "+negYStr+" 1 1 "+negYStr+" 1 1 1 1)" if graded else "simpleGrading (1 1 1)"-|

    hex (3 6 7 4 14 17 18 15)
    (|-nrX2-| |-nrY1-| 1)
    |-"edgeGrading (4 4 4 4 "+posYStr+" "+posYRStr+" "+posYRStr+" "+posYStr+" 1 1 1 1 1)" if graded else "simpleGrading (1 1 1)"-|

    hex (5 8 9 6 16 19 20 17)
    (|-nrX3-| |-nrY2-| 1)
    simpleGrading |-"(2.5 1 1)" if graded else "(1 1 1)"-|

    hex (6 9 10 7 17 20 21 18)
    (|-nrX3-| |-nrY1-| 1)
    simpleGrading |-"(2.5 "+posYRStr+" 1)" if graded else "(1 1 1)"-|
);
```

Setting and running a coarser mesh

- Time for our first serious parameter file
  - Parameter files can have any name
    - But PyFoam recognizes `.parameters` during cloning
  - Format of the file is: FOAM-dictionary without header
- Collect settings that belong together in one file
  - Use with `--parameter-file`
    - Can be specified more than once

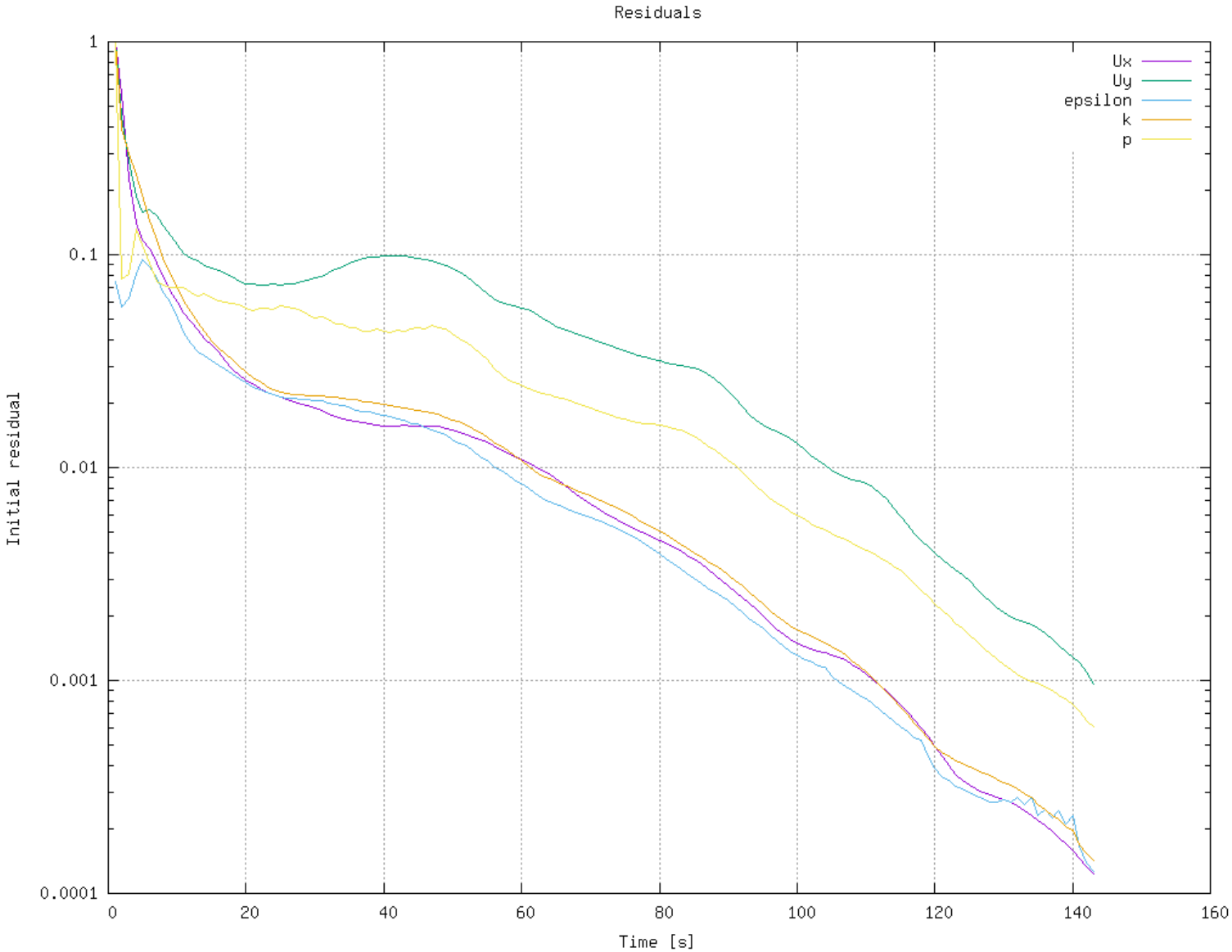
coarseMesh.parameters

```
resFactor 0.5;
graded no;
```

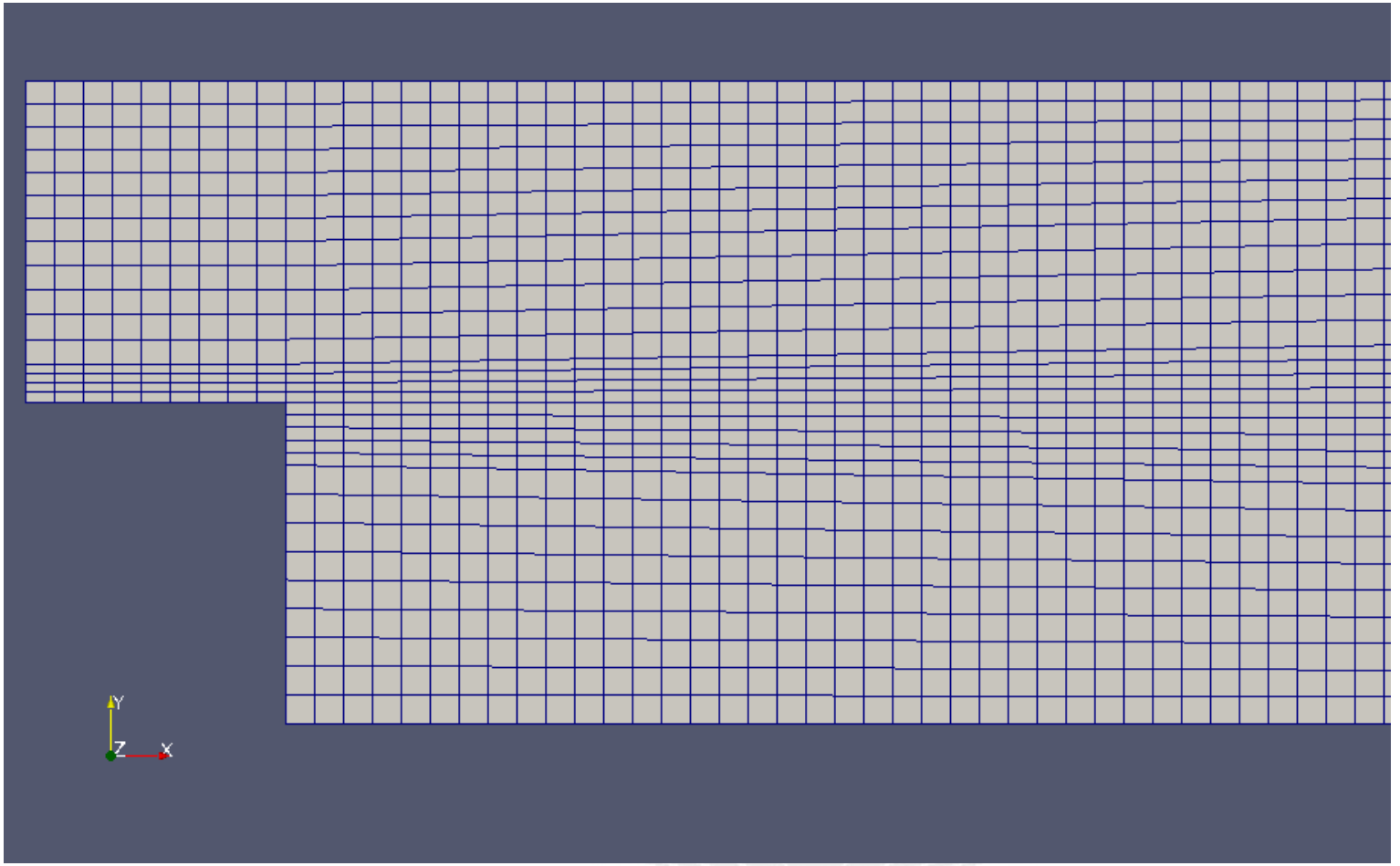
Running

```
> pyFoamPrepareCase.py . --parameter=coarseMesh.parameters
...
> pyFoamPlotRunner.py --clear --progress simpleFoam
 t =         143 dp: -6.15499 detach: 0.161367
```

Residuals for the coarse mesh



The coarse ungraded mesh



Spawn point

To get to the current state use `stage5_coarseMesh.tar.gz`

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Adding a boundary layer

- The question:
  - "Is an ungraded mesh with a boundary layer as good as a graded mesh?"
    - Will not be answered in this presentations
    - But we will implement the tools to answer it
- User can specify the number of boundary layers
  - Each boundary layer is created by splitting the boundary cells exactly in half
    - Modifying this ratio is left as an exercise to the reader
  - A number of 0 keeps the original mesh

```
mv meshCrate.sh meshCreate.sh.template
```

The parameter

- Adding one more parameter to the `meshing` group

defaultParameters

```
meshing {
    description "Meshing parameters";
    values {
        nrBoundaryLayers {
            description "Number of boundary layers";
            default 0;
        }
        resFactor {
            description "Resolution compared to the original";
            default 1.;
        }
        graded {
            description "Whether mesh grading should be used";
            default true;
        }
    }
}
```

Setting up the mesh

- Template generates a script to
  - First call `blockMesh`
  - Then call `refineWallLayer` for the wall-patches

meshCreate.sh.template

```
#! /bin/bash

blockMesh

<!--(for i in range(nrBoundaryLayers))-->
refineWallLayer -overwrite "(upperWall lowerWall)" 0.5
<!--(end)-->
```

It would have also been possible to have a loop in the bash-language. I just happen to not like the syntax of shell scripts

Reusing coarse parameters with #include

- Copy/Pasting parameter sets is not always the best idea
  - The meaning of 'coarse' changes and you'll have to change in all files
- Recommendation:
  - Collect settings for *a coarse mesh* in one file
  - Reuse it with `#include`

coarseMeshLayers.parameters

```
#include "coarseMesh.parameters"

nrBoundaryLayers 2;
```

Testing layers

- We run the layered mesh
- Observations:
  - Number of iterations decrease
    - .... until they don't
  - Both results (pressure and detachment) also change significantly
    - Whether this is to the better: no idea

Shell

```
> pyFoamPrepareCase.py . --parameter=coarseMeshLayers.parameters
...
> pyFoamPlotRunner.py --clear --progress simpleFoam
 t =       2000 dp: -3.9965 detach: 0.163655
```

Better convergence with layers?

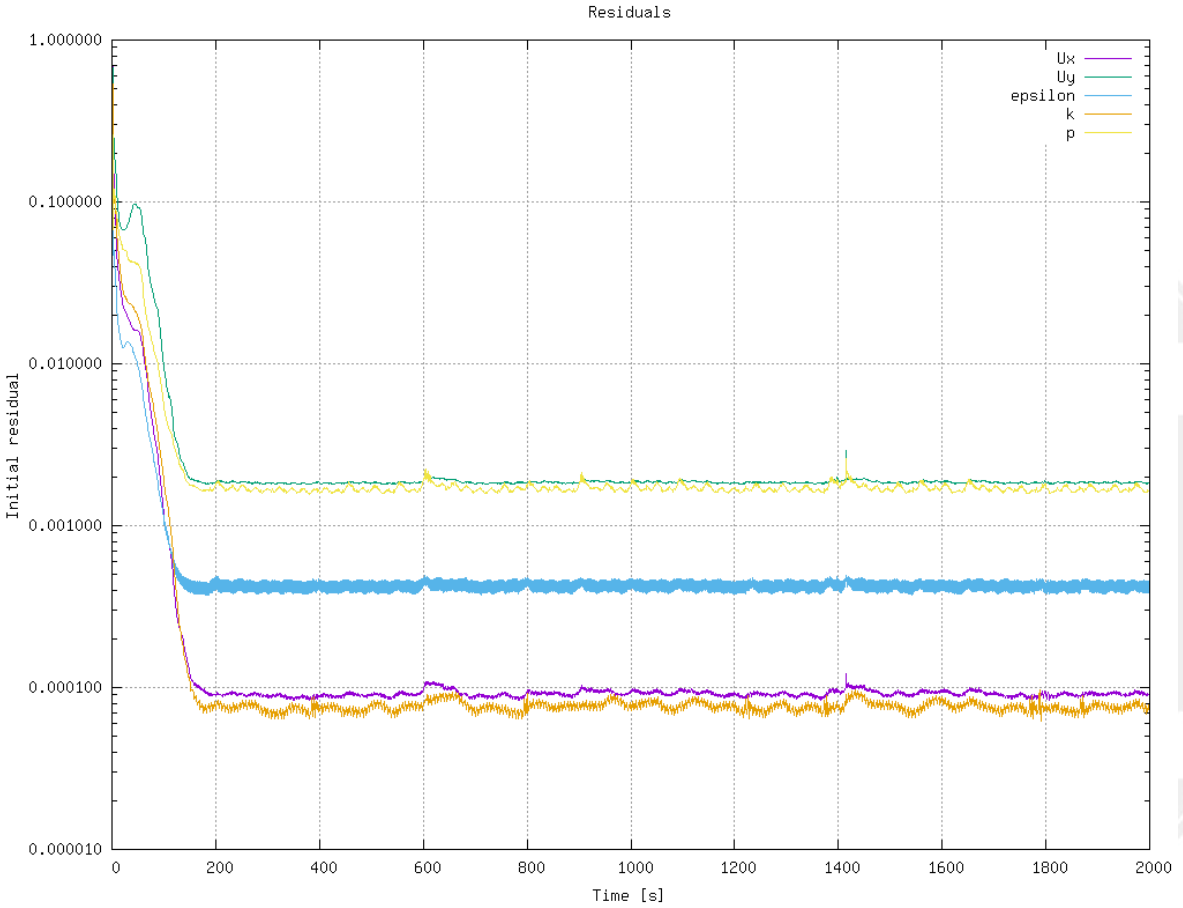
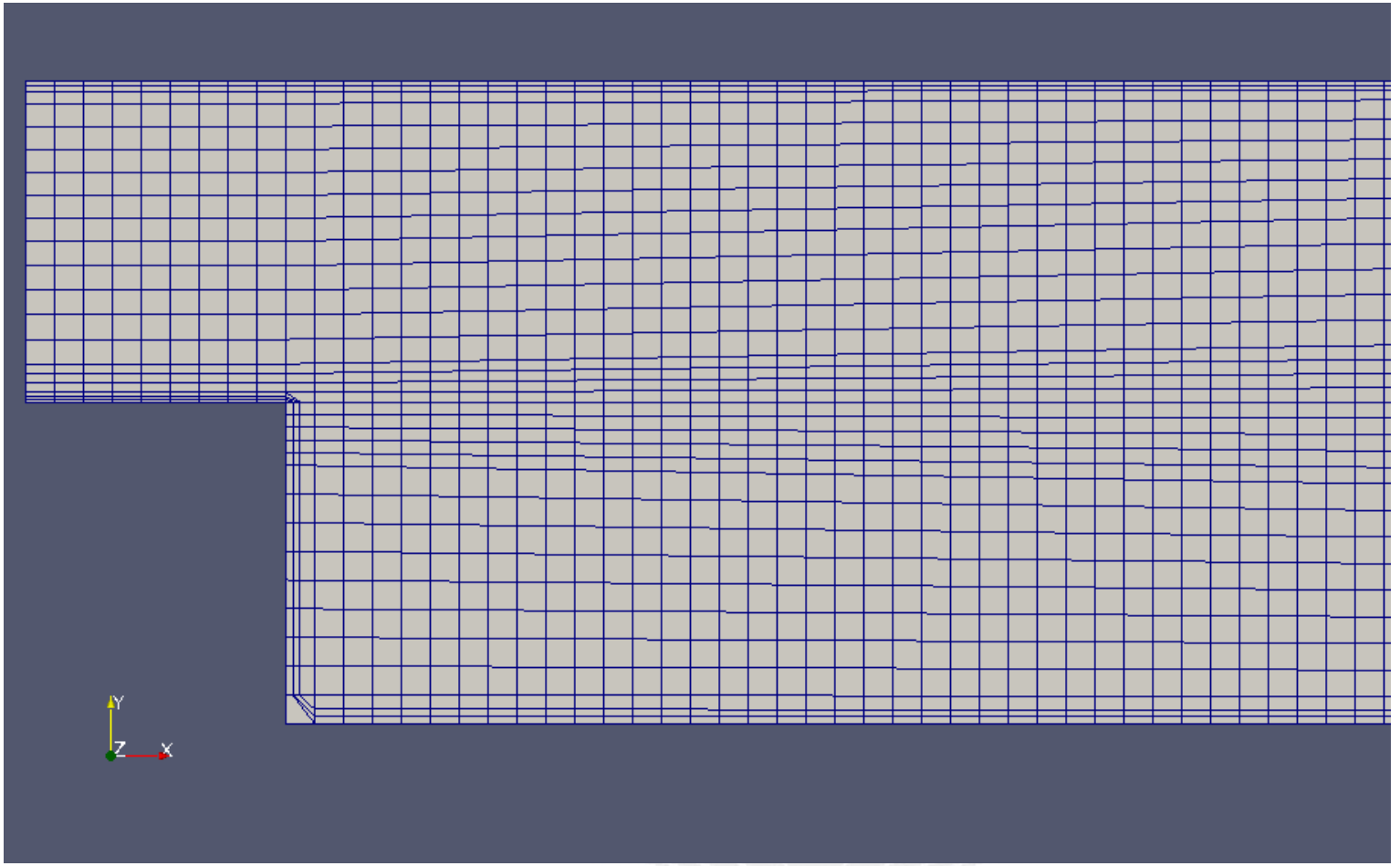


Figure: The relaxation of 0.9 does not let this converge (smaller values help)

Mesh with layers



Logfiles for the scripts

- The output of `meshCreate.sh` can be very long
  - for instance if one of the tools used in the script is `snappyHexMesh`
- ... and interesting
  - for instance `snappyHexMesh`
- Therefor the full output goes to a file `meshCreate.sh.log`
  - And to the terminal as well
- The `caseSetup.sh`-phase (see below) also has a logfile

Spawn point

To get to the current state use `stage6_meshCreate.tar.gz`

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Scripting initial conditions

- Sometimes life is more complicated than `uniform 0`
- Then you need to call some utilities to set the initial conditions
  - `setFields`
  - `funkySetFields`
- To do this `pyFoamPrepareCase.py` calls a script `caseSetup.sh`
  - This script can also be generated from a template
- In our case we want to initialize the velocity field with `potentialFoam`
  - Check if this leads to faster convergence

Different templates for different stages

- Some utilities need different versions of the same file
  - `controlDict` for instance
    - Some utilities choke on the `functions`
    - Some need different timesteps (`foamyHexFoam`)
- Sometimes a template needs results that are only available after the meshing phase
  - Names of the patches for instance
- So in addition to `.template` two other templates are available

  .postTemplate expanded after meshing but before case setup
  .finalTemplate expanded after case setup

Adaptions necessary for potentialFoam

To use potential foam some things have to be done

- Add a switch
- A setup-script
- Special pressure field
- Numerical settings
  - in `fvSchemes` and `fvSolution`

New parameter

Default is to not use potentialFoam

default.parameters

```
initial {
    description "Initial conditions";
    values {
        potentialFlow {
            description "Initialize with potentialFoam";
            default false;
        }
    }
}
```

The setup script

- Does two things:
  - Call `potentialFoam`
  - Remove `phi` as the dimensions don't fit the compressible solver

caseSetup.sh.postTemplate

```
#! /bin/sh

<!--(if potentialFlow)-->
potentialFoam

  <!--(if compressible)-->
rm 0/phi*
  <!--(end)-->
<!--(end)-->
```

This could also have been a normal `.template`

Special pressure file

- potentialFoam needs an 'incompressible' pressure
  - Also: an initial pressure of 10^5 causes numerical problems
- Copy so that our 'old' pressure template will be used for the flow solver

```
> cp 0.org/p.template 0.org/p.finalTemplate
```

- Edit the pressure for potentialFoam

0.org/p.template

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    outlet
    {
        type            fixedValue;
        value           $internalField;
    }
```

Numerical stuff

system/fvSolution

```
    Phi
    {
        $p;
    }
}
potentialFlow
{
    $SIMPLE;
    nNonOrthogonalCorrectors 3;
}
```

system/fvSchemes

```
fluxRequired
{
    default           no;
    p                 ;
    Phi               ;
}
```

This run has potential

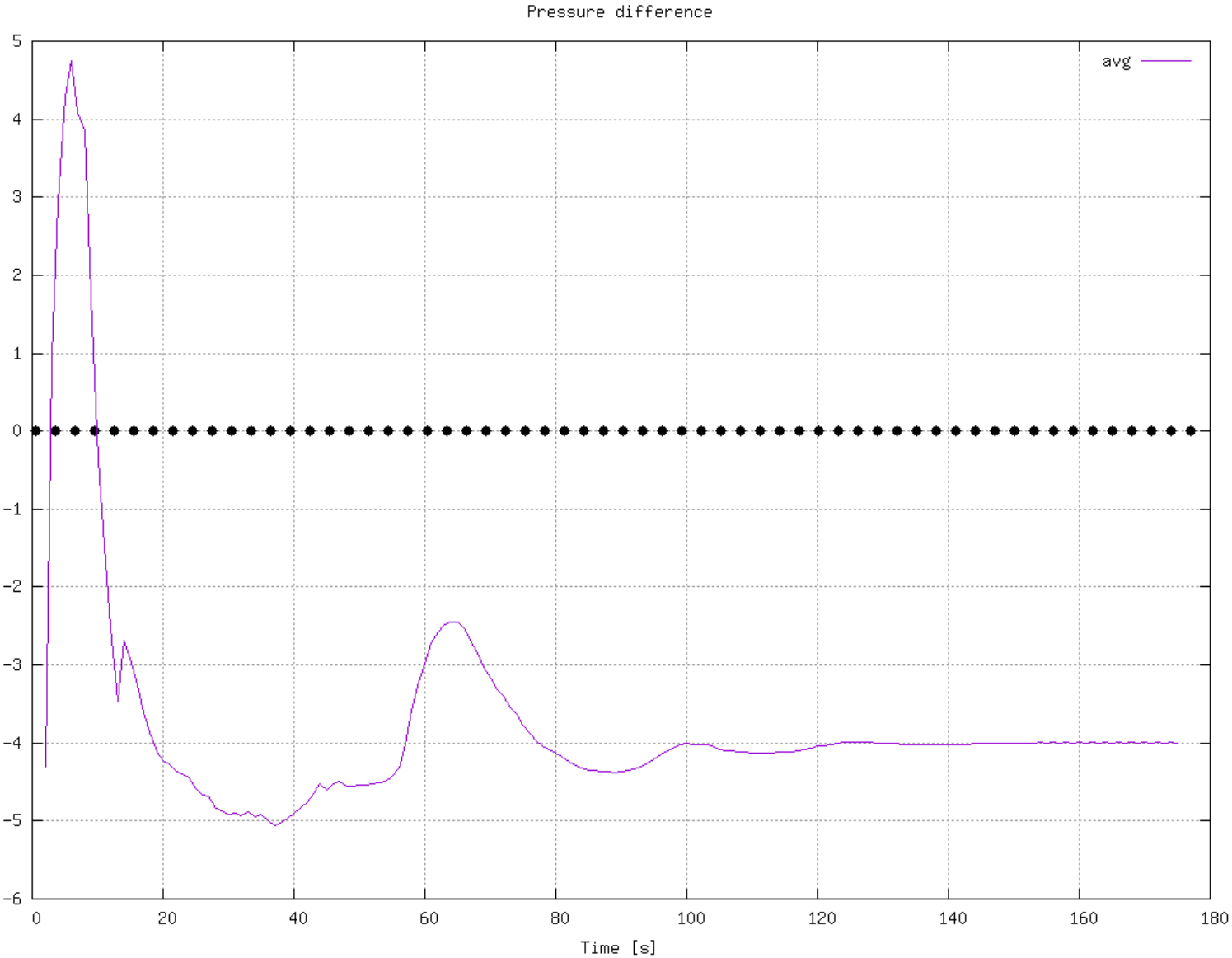
Again: our two utilities

Shell

```
> pyFoamPrepareCase.py . --parameter=coarseMeshLayers.parameters --value="potentialFlow yes<brk>
      <cont>;"
...
> pyFoamPlotRunner.py --clear --progress --hardcopy --prefix=potential simpleFoam
 t =        175 dp: -3.99923 detach: 0.163655
```

- More iterations than before, but improvement on the pressure
- For some reason the "coarse + layer" run now converges

Initial pressure spike: almost gone



How did we set this up?

- Many pyFoam-utilities automatically write their actions to a file PyFoamHistory
- This allows answering questions like "On Tuesday the run converged. How did I call pyFoamPrepareCase then?"

```
> cat PyFoamHistory
Fri Jul  7 21:21:09 2023 by dockeruser in 28437674bec7 :Application: pyFoamPrepareCase.py .<brk>
      <cont> --parameter=coarseMeshLayers.parameters --value="potentialFlow yes;"  | with <brk>
      <cont>cwd /foamdata/stage7_initPotential  |
Fri Jul  7 21:21:43 2023 by dockeruser in 28437674bec7 :Application: pyFoamPlotRunner.py --<brk>
      <cont>clear --progress --hardcopy --prefix=potential auto  | with cwd /foamdata/<brk>
      <cont>stage7_initPotential  |  Starting
Fri Jul  7 21:21:44 2023 by dockeruser in 28437674bec7 :Application: pyFoamPlotRunner.py --<brk>
      <cont>clear --progress --hardcopy --prefix=potential auto  | with cwd /foamdata/<brk>
      <cont>stage7_initPotential  |  Ending
```

Spawn point

To get to the current state use `stage7_initPotential.tar.gz`

Outline

1. Introduction
   - About this presentation
   - Who is this?
   - Technicalities
   - Case setup in OpenFOAM
   - PyFoam overview
   - Our case
2. Using templates
   - Stage 0: pyFoamPrepareCase.py without modifications
   - Stage 1: Adding information
   - Stage 2: Calculations and Variables
   - Stage 3: Control structures
   - Stage 4: Loops
3. Scripting
   - Stage 5: Improvements to the mesh
   - Stage 6: non-trivial mesh creation
   - Stage 7: Complex initial conditions
4. Advanced
   - Stage 8: Switching Foam-Versions
   - Stage 9: Parallelization
   - Stage 10: Parameter Variations
   - Other topics
5. Conclusions

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Caution: old content

- `foam-extend 5.0` has a bug
  - the `pitzDaily` tutorial blows up
    - other `simpleFoam` cases work
  - it seems to be a strange bug in the compiler (`gcc 11` on Ubuntu)
    - so if you're doing work on another machine it might work
- therefor in this section results from `foam-extend 4.1` were used
  - but everything should work once that bug is fixed
- But the main purpose of this section is showing how to distinguish different OpenFOAM-forks

foam-extend and the pUCoupledFoam-solver

- Until now we've been working with `OpenFOAM v2306+`
  - OF 10 should also work
- But there is also `foam-extend 5.0`
  - Which has slightly different formats for files
  - But also a very interesting additional solver: `pUCoupledFoam`
    - Steady state, incompressible
    - Pressure and momentum equation are solved simultaneously
- We'd like to compare the fully coupled solver with the others
  - And also whether `simpleFoam` between the two forks differs

Adding the solver

First we add a third solver to the list

default.parameters

```
solver {
    description "Used solver";
    default simpleFoam;
    options (
        simpleFoam
        rhoSimpleFoam
        pUCoupledFoam
    );
}
```

Different usage for old refineWallLayer

- In foam-extend it can only handle one patch at a time
  - This was the default behavior for OpenFOAM before 2.3.1
- Using the automatically defined foamFork and foamVersion parameters we can select the right behavior

meshCreate.sh.template

```
#! /bin/bash

blockMesh

<!--(for i in range(nrBoundaryLayers))-->
  <!--(if foamFork!="extend" and foamVersion>(2,3,1))-->
refineWallLayer -overwrite "(upperWall␣lowerWall)" 0.5
  <!--(else)-->
refineWallLayer -overwrite lowerWall 0.5
refineWallLayer -overwrite upperWall 0.5
  <!--(end)-->
<!--(end)-->
```

Missing function object in extend

- foam-extend has no streamLine function object
  - Not a big deal, because we didn't use it in our evaluations

system/controlDict.template

```
functions {

<snip>

<!--(if foamFork!="extend")-->
    streamLines
    {
        type            streamLine;

<snip>

    }
<!--(end)-->
}
```

No bounded in extend

- Problem: `foam-extend` has no `bounded` for `div`-schemes
- Make `fvSchemes` a template

```
> mv system/fvSchemes system/fvSchemes.template
```

- and only use `bounded` when available

fvSchemes.template

```
<!--(if foamFork!="extend")-->
$$ bounded="bounded"
<!--(else)-->
$$ bounded=""
<!--(end)-->

divSchemes
{
    default          none;
    div(phi,U)       |-bounded-| Gauss upwind;
    div(phi,k)       |-bounded-| Gauss upwind;
```

Proposal for an exercise

- Add a configuration parameter to switch off `bounded` in FOAM-versions that support it
  - See how it affects convergence

Additional schemes

- The coupled solver also needs an additional `div`-scheme

fvSchemes.template

```
...

    // differences in foam-extend
    div((muEff*dev2(grad(U).T()))) Gauss linear;
    div(U,p) Gauss upwind phi;
    div(phi,h)      |-bounded-| Gauss linear;
    div((nuEff*dev(grad(U).T()))) Gauss linear;

<!--(if solver=="pUCoupledFoam")-->
    div(U)          Gauss linear;
<!--(end)-->
}
```

Coupled solver

- The coupled solver needs a special linear solver
- Also: energy is a different field: `e` vs `h`

```
> mv system/fvSolution system/fvSolution.template
```

fvSolution.template

```
solvers
{
<!--(if solver=="pUCoupledFoam")-->
    Up
    {
        solver BiCGStab;
        preconditioner Cholesky;

        tolerance 1e-09;
        relTol  0.0;

        minIter 1;
        maxIter 500;
    }
<!--(end)-->
...
    "(U|k|epsilon|R|nuTilda|e|h)"
    {
        solver          smoothSolver;
...
```

Convergence criterion

- foam-extend sets one convergence criterion for all fields
  - Instead of the "per-field" crtiterion of OpenFOAM

fvSolution.template

```
SIMPLE
{
...
<!--(if foamFork=="extend")-->
    convergence 1e-3;
<!--(end)-->
}
```

Settings for pUCoupledFoam

- Coupled solver needs additional settings

fvSolution.template

```
<!--(if solver=="pUCoupledFoam")-->
blockSolver
{
    $SIMPLE;
}

fieldBounds
{
    U       500;
    p       -5e4 5e4;
}
<!--(end)-->
```

Different relaxation

- Also: relaxation is specified in a slightly different way
  - And the coupled solver allows higher relaxation

fvSolution.template

```
relaxationFactors
{
...
<!--(if foamFork=="extend")-->
  <!--(if solver=="pUCoupledFoam")-->
    U               0.99;
  <!--(else)-->
    U               0.7;
  <!--(end)-->
    p               0.3;
    rho             0.05;
    k               0.9;
    epsilon         0.9;
    h               0.9;
<!--(end)-->
}
```

Old thermophysics format

- The format of `thermoType` is still in the old-school 'Fortrany' style

thermophysicalProperties.template

```
<!--(if foamFork=="extend")-->
thermoType      hPsiThermo<pureMixture<constTransport<specieThermo<hConstThermo<perfectGas<brk>
      <cont>>>>>>;
mixture         air 1 |-molarWeight-| 1000 0 |-dynVisc-| 0.7;
<!--(else)-->
thermoType
{
    type            hePsiThermo;
...
}
<!--(end)-->
```

Running on foam-extend

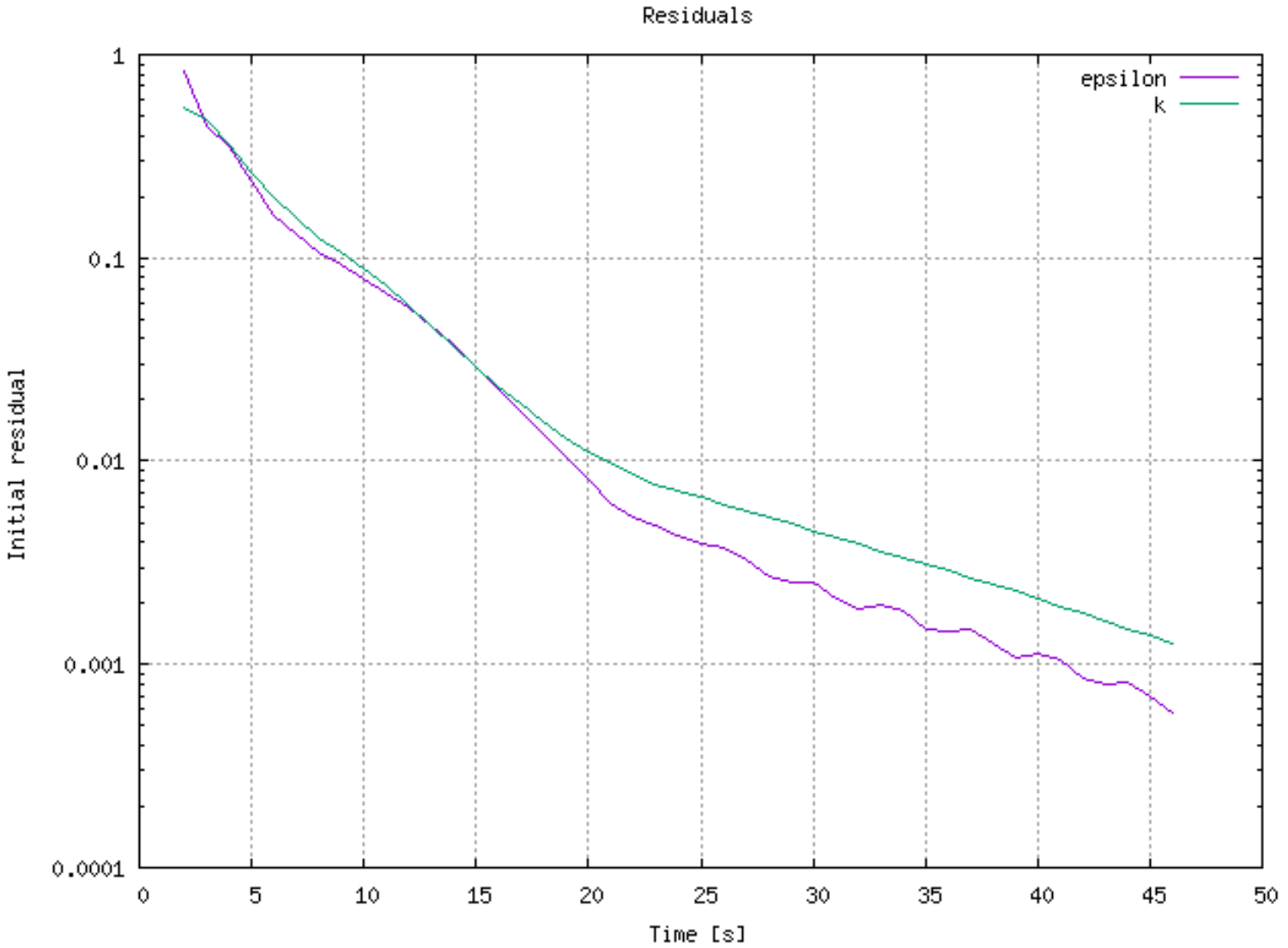
- Now we switch the shell to `foam-extend` and try different solvers

Shell

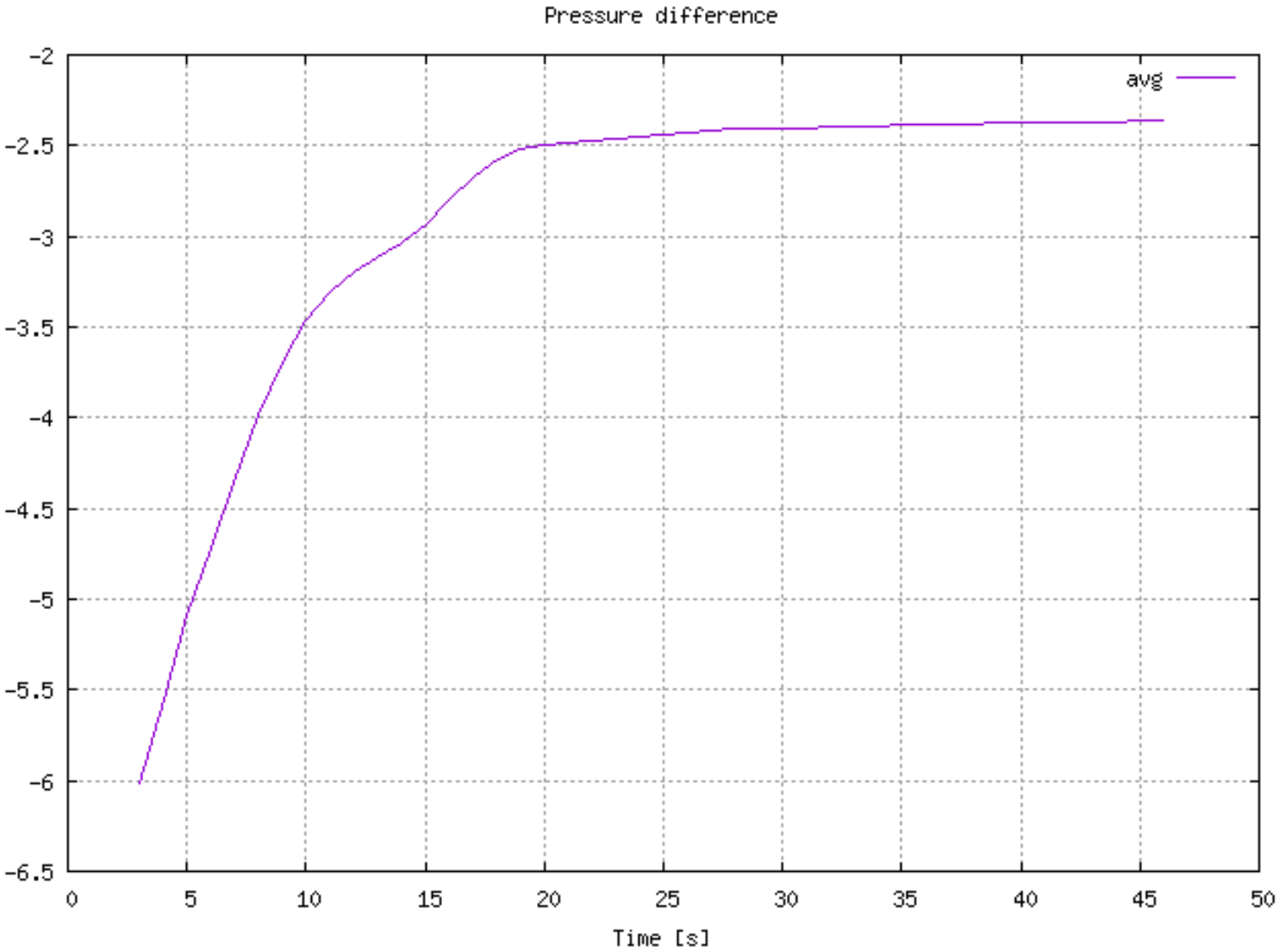
```
> . ~/foam/foam-extend-4.1/etc/zshrc
> pyFoamPrepareCase.py . --parameter=coarseMeshLayers.parameters --value="potentialFlow yes;"
...
> pyFoamPlotRunner.py --clear  --progress simpleFoam
 t =         267 dp: -2.78484 detach: 0.168234
> pyFoamPrepareCase.py . --parameter=coarseMeshLayers.parameters --value="potentialFlow yes; solver pUCoupledFoam;"
...
> pyFoamPlotRunner.py --clear  --progress pUCoupledFoam
 t =          43 dp: -2.35515 detach: 0.165944
> pyFoamPrepareCase.py . --parameter=coarseMeshLayers.parameters --value="solver pUCoupledFoam;"
...
> pyFoamPlotRunner.py --clear  --progress pUCoupledFoam
 t =          46 dp: -2.36399 detach: 0.168234
```

- Number of iterations for `simpleFoam` slightly different
- Coupled solver has drastically less iterations

Coupled solver residuals



Pressure difference with coupled



Spawn point

To get to the current state use `stage8_extendCoupled.tar.gz`

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Working around decomposePar

- Obviously we want our tests to run in parallel too
- But there are some problems:
  - `decomposePar` doesn't like `.template` files when it finds them in `0`
    - Thinks their format is incorrect and dies
- `pyFoamPrepareCase.py` does not automatically support parallelization
  - Because it differs too much between work-flows what can be done in parallel
    - There is no "one size fits all"
- But it offers some help
  - Command line option that sets the number of processors
  - Some "regular PyFoam" features

Decomposing

- Using `pyFoamDecompose.py` for decomposition
- Moving template files to where `decomposePar` won't find them
  - And move them back again

`meshCreate.sh.template`

```
<!--(if numberOfProcessors>1)-->
cp -r 0.org 0
rm 0/*.template

mkdir 0.tmp
mv 0/*.finalTemplate 0.tmp

pyFoamDecompose.py . |-numberOfProcessors-|

  <!--(for p in range(numberOfProcessors))-->
cp 0.tmp/* processor|-p-|/0/
  <!--(end)-->

rm -r 0.tmp
<!--(end)-->
```

--autosense-parallel

- pyFoamRunner.py has an option -autosense-parallel
  - This say:
    - If you find processor-directories then run the program in parallel
    - ... otherwise: single processor
- Prefixing utilities after the decomposition with this saves us from having a if parallel for every on of them

caseSetup.sh.postTemplate

```
#! /bin/sh

<!--(if potentialFlow)-->
pyFoamRunner.py --auto potentialFoam
  <!--(if compressible)-->
rm -f 0/phi* processor*/0/phi*
  <!--(end)-->
<!--(end)-->
```

Files have to be removed from the processor directories as well

Making sure it runs in parallel

- Because `.finalTemplate` has not been decomposed we have to make sure manually that it works
  - Add `procBoundary`-files to `p`

0.org/p.finalTemplate

```
boundaryField
{
...
    "procBoundary.*"
    {
        type            processor;
        value           $internalField;
    }
}
```

Running

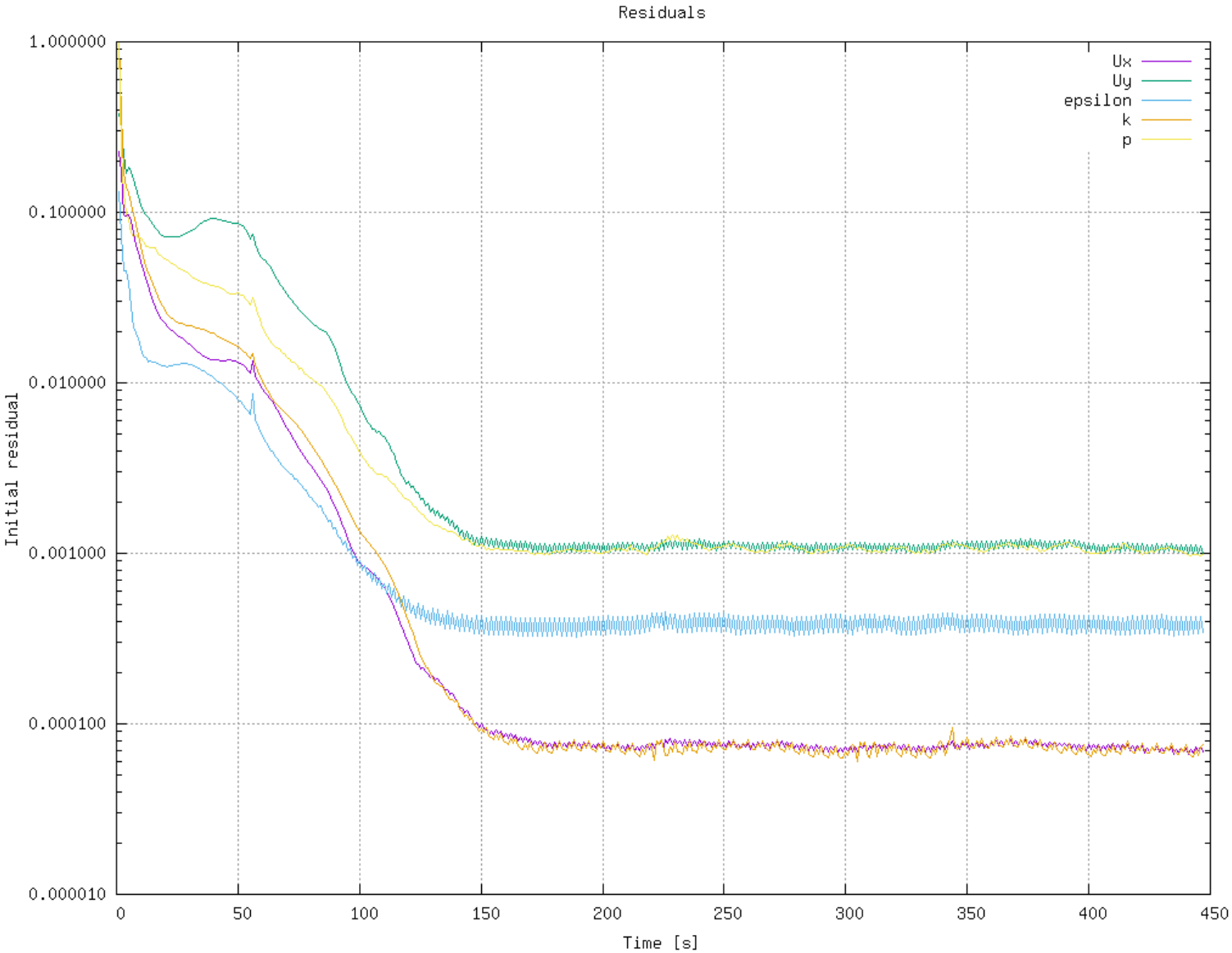
For running the solver we use `--autosense-parallel` for the `pyFoamRunner.py` as well

Shell

```
> pyFoamPrepareCase.py . --parameter=coarseMeshLayers.parameters --value="potentialFlow yes<brk>
      <cont>;" --number-of-processors=2
...
 Warning in /usr/local/bin/pyFoamPrepareCase.py : Case should be decomposed to 2 cpus but <brk>
       <cont>no decompose script ( decomposeMesh.sh decomposeFields.sh decomposeCase.sh ) <brk>
       <cont>found
Case setup finished
> pyFoamPlotRunner.py --clear  --progress --auto simpleFoam
 t =         447 dp: -3.99513 detach: 0.163655
```

That is a lot of timesteps/iterations

Eventually it converges



Other decomposition hooks

- `PyFoamPrepareCase` looks for scripts for decomposing in different phases

  decomposeMesh.sh runs after `meshCreate.sh`
  decomposeFields.sh runs after `0.org` has been copied to `0`
  decomposeCase.sh runs after `caseSetup.sh`
- Usually not all of them are used
  - Depends on the way the mesh was generated
    - and on certain utilities not working in parallel
  - Gives us flexibility

Spawn point

To get to the current state use `stage9_parallel.tar.gz`

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Parameter variations

- The utility `pyFoamRunParameterVariation.py` is based on the functionality of `pyFoamPrepareCase.py`
  - Idea: set up and run as usual.
    - Just change some parameters systematically
- Specification of the variation in a dictionary
- `values` is a dictionary with parameters and their variations
  - Either a list of simple values
  - Or a list of dictionaries
    - This is for cases where only specific sets of parameters make sense
    - Needs a `defaults`-entry if values are missing from the set

The variation file

- Compare solvers and mesh resolutions

meshResolution.variation

```
defaults {
    nrBoundaryLayers 0;
    graded true;
}

values {
    solver (
        simpleFoam
        rhoSimpleFoam
        pUCoupledFoam
    );
    resFactor (
        0.5 1 1.5 2
    );
    sets (
        {
            nrBoundaryLayers 1;
            graded false;
        }
        { }
    );
 }
```

Explanation

solver Compare three solvers
- This entry is always required (even if only one entry)

resFactor try 4 resolution factors

sets Compare grading/no layers with ungraded/layers
- but no graded/layers and ungraded/no layers

Total of $4 \times 3 \times 2 = 24$ variations

Preparing

- The utility has to be executed outside our case

```
cd ..
```

- Will create a clone of our case for every variation specified
  - This behavior can be changed
    - Do variations in the original case
    - Create only one clone and do all variations there
  - But we'll go for the extra copies
    - That way we can afterwards check the full results
    - Not only the summaries

Listing the variations

- Every variation has a number
  - We want to see which one sets which values

Shell

```
> pyFoamRunParameterVariation.py stage10_parameterVariation/ stage10_parameterVariation/meshResolution.variation --<brk>
      <cont>list-variations

==============================
24 variations with 3 parameters
==============================


==================
Listing variations
==================

Variation 0 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 0.5, 'solver': 'simpleFoam'}
Variation 1 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 0.5, 'solver': 'rhoSimpleFoam'}
Variation 2 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 0.5, 'solver': 'pUCoupledFoam'}
Variation 3 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 1, 'solver': 'simpleFoam'}
Variation 4 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 1, 'solver': 'rhoSimpleFoam'}
Variation 5 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 1, 'solver': 'pUCoupledFoam'}
Variation 6 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 1.5, 'solver': 'simpleFoam'}
Variation 7 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 1.5, 'solver': 'rhoSimpleFoam'}
Variation 8 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 1.5, 'solver': 'pUCoupledFoam'}
Variation 9 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 2, 'solver': 'simpleFoam'}
Variation 10 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 2, 'solver': 'rhoSimpleFoam'}
Variation 11 : {'nrBoundaryLayers': 1, 'graded': false, 'resFactor': 2, 'solver': 'pUCoupledFoam'}
Variation 12 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 0.5, 'solver': 'simpleFoam'}
Variation 13 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 0.5, 'solver': 'rhoSimpleFoam'}
Variation 14 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 0.5, 'solver': 'pUCoupledFoam'}
Variation 15 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 1, 'solver': 'simpleFoam'}
Variation 16 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 1, 'solver': 'rhoSimpleFoam'}
Variation 17 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 1, 'solver': 'pUCoupledFoam'}
Variation 18 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 1.5, 'solver': 'simpleFoam'}
Variation 19 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 1.5, 'solver': 'rhoSimpleFoam'}
Variation 20 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 1.5, 'solver': 'pUCoupledFoam'}
Variation 21 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 2, 'solver': 'simpleFoam'}
Variation 22 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 2, 'solver': 'rhoSimpleFoam'}
Variation 23 : {'nrBoundaryLayers': 0, 'graded': true, 'resFactor': 2, 'solver': 'pUCoupledFoam'}
```

Running the variation

- We run the actual variation
  - Specify the variation file
  - and the original case
- This takes some time
  - And creates 24 new directories

Shell

```
> mkdir variations
> pyFoamRunParameterVariation.py stage10_parameterVariation stage10_parameterVariation/<brk>
      <cont>meshResolution.variation  --every-variant-one-case-execution --auto-create-<brk>
      <cont>database --quiet --clone-to=variations
...
> ls variations
meshResolution.variation_00000_stage10_parameterVariation
meshResolution.variation_00001_stage10_parameterVariation
meshResolution.variation_00002_stage10_parameterVariation
meshResolution.variation_00003_stage10_parameterVariation
meshResolution.variation_00004_stage10_parameterVariation
meshResolution.variation_00005_stage10_parameterVariation
meshResolution.variation_00006_stage10_parameterVariation
...
```

List the cases

We want to get an overview of the created case directories

Shell

```
> pyFoamListCases.py variations/
                   mtime |     hostname | first - last (nrSteps) nowTime s            state | solver | name
-----------------------------------------------------------------------------------------------------------
Fri Jul  7 21:56:53 2023 | 28437674bec7 |     0 -  156 (      3)   156.0 s Finished - Ended |   None | meshResolution.variation_00000
Fri Jul  7 21:56:56 2023 | 28437674bec7 |     0 -  144 (      3)   144.0 s Finished - Ended |   None | meshResolution.variation_00001
Fri Jul  7 21:56:57 2023 | 28437674bec7 |     0 -    0 (      1)    None s          Finished |   None | meshResolution.variation_00002
Fri Jul  7 21:57:04 2023 | 28437674bec7 |     0 -  244 (      4)   244.0 s Finished - Ended |   None | meshResolution.variation_00003
Fri Jul  7 21:57:12 2023 | 28437674bec7 |     0 -  244 (      4)   244.0 s Finished - Ended |   None | meshResolution.variation_00004
Fri Jul  7 21:57:14 2023 | 28437674bec7 |     0 -    0 (      1)    None s          Finished |   None | meshResolution.variation_00005
Fri Jul  7 21:57:35 2023 | 28437674bec7 |     0 -  403 (      6)   403.0 s Finished - Ended |   None | meshResolution.variation_00006
Fri Jul  7 21:58:00 2023 | 28437674bec7 |     0 -  403 (      6)   403.0 s Finished - Ended |   None | meshResolution.variation_00007
Fri Jul  7 21:58:02 2023 | 28437674bec7 |     0 -    0 (      1)    None s          Finished |   None | meshResolution.variation_00008
Fri Jul  7 21:58:58 2023 | 28437674bec7 |     0 -  626 (      8)   626.0 s Finished - Ended |   None | meshResolution.variation_00009
Fri Jul  7 22:00:03 2023 | 28437674bec7 |     0 -  625 (      8)   625.0 s Finished - Ended |   None | meshResolution.variation_00010
Fri Jul  7 22:00:07 2023 | 28437674bec7 |     0 -    0 (      1)    None s          Finished |   None | meshResolution.variation_00011
Fri Jul  7 22:00:09 2023 | 28437674bec7 |     0 -  167 (      3)   167.0 s Finished - Ended |   None | meshResolution.variation_00012
Fri Jul  7 22:00:12 2023 | 28437674bec7 |     0 -  165 (      3)   165.0 s Finished - Ended |   None | meshResolution.variation_00013
Fri Jul  7 22:00:13 2023 | 28437674bec7 |     0 -    0 (      1)    None s          Finished |   None | meshResolution.variation_00014
Fri Jul  7 22:00:20 2023 | 28437674bec7 |     0 -  278 (      4)   278.0 s Finished - Ended |   None | meshResolution.variation_00015
Fri Jul  7 22:00:28 2023 | 28437674bec7 |     0 -  282 (      4)   282.0 s Finished - Ended |   None | meshResolution.variation_00016
Fri Jul  7 22:00:29 2023 | 28437674bec7 |     0 -    0 (      1)    None s          Finished |   None | meshResolution.variation_00017
Fri Jul  7 22:00:50 2023 | 28437674bec7 |     0 -  483 (      6)   483.0 s Finished - Ended |   None | meshResolution.variation_00018
Fri Jul  7 22:01:15 2023 | 28437674bec7 |     0 -  471 (      6)   471.0 s Finished - Ended |   None | meshResolution.variation_00019
Fri Jul  7 22:01:17 2023 | 28437674bec7 |     0 -    0 (      1)    None s          Finished |   None | meshResolution.variation_00020
Fri Jul  7 22:02:10 2023 | 28437674bec7 |     0 -  667 (      8)   667.0 s Finished - Ended |   None | meshResolution.variation_00021
Fri Jul  7 22:03:14 2023 | 28437674bec7 |     0 -  647 (      8)   647.0 s Finished - Ended |   None | meshResolution.variation_00022
Fri Jul  7 22:03:16 2023 | 28437674bec7 |     0 -    0 (      1)    None s          Finished |   None | meshResolution.variation_00023
```

Note: Directory namse have been clipped for better readability

Data of one run

- Data from each run is stored in a *pickled file*.
- Can be dumped with a utility
  - Read from any Python-script with the `pickled` module

Shell

```
> pyFoamEchoPickledApplicationData.py --print-data --pickled-file=variations/meshResolution.<brk>
      <cont>variation_00000_stage10_parameterVariation/meshResolution.variation_simpleFoam.analyzed/pickledData
{'OK': True,
 'analyzed': {'Bounding': {'epsilon_avg': 72.0684,
                           'epsilon_max': 2421.58,
                           'epsilon_min': -14.7865},
              'Continuity': {'Cumulative': -2.07774, 'Global': -0.000307881},
              'Custom': {'deltaP': {'avg': -5.28467},
                         'detatch': {'x': 0.165944}},
              'Custom01_deltaP': {'avg': -5.28467},
              'Custom02_detatch': {'x': 0.165944},
...
 'parameters': {'TAir': 293,
                'UIn': 10,
                'caseName': '"meshResolution.variation_00000_stage10_parameterVariation"',
                'casePath': '"/foamdata/variations/meshResolution.variation_00000_stage10_parameterVariation"',
                'compressible': 'False',
                'dynVisc': 1.8e-05,
                'foamFork': 'openfoamplus',
                'foamVersion': 2306,
                'graded': false,
                'molarWeight': 28.9,
                'nrBoundaryLayers': 1,
                'numberOfProcessors': 1,
                'pAir': 100000.0,
                'potentialFlow': 'false',
                'resFactor': 0.5,
                'solver': 'simpleFoam',
                'xProfiles': '"[-10, 0, 50, 100, 150, 200, 250]"'},
...
```

Reading the database information

- Data of the variations is stored in a SQlite-database
- There is also a utility to dump this to (for instance) Excel
  - But can also be processed in Python

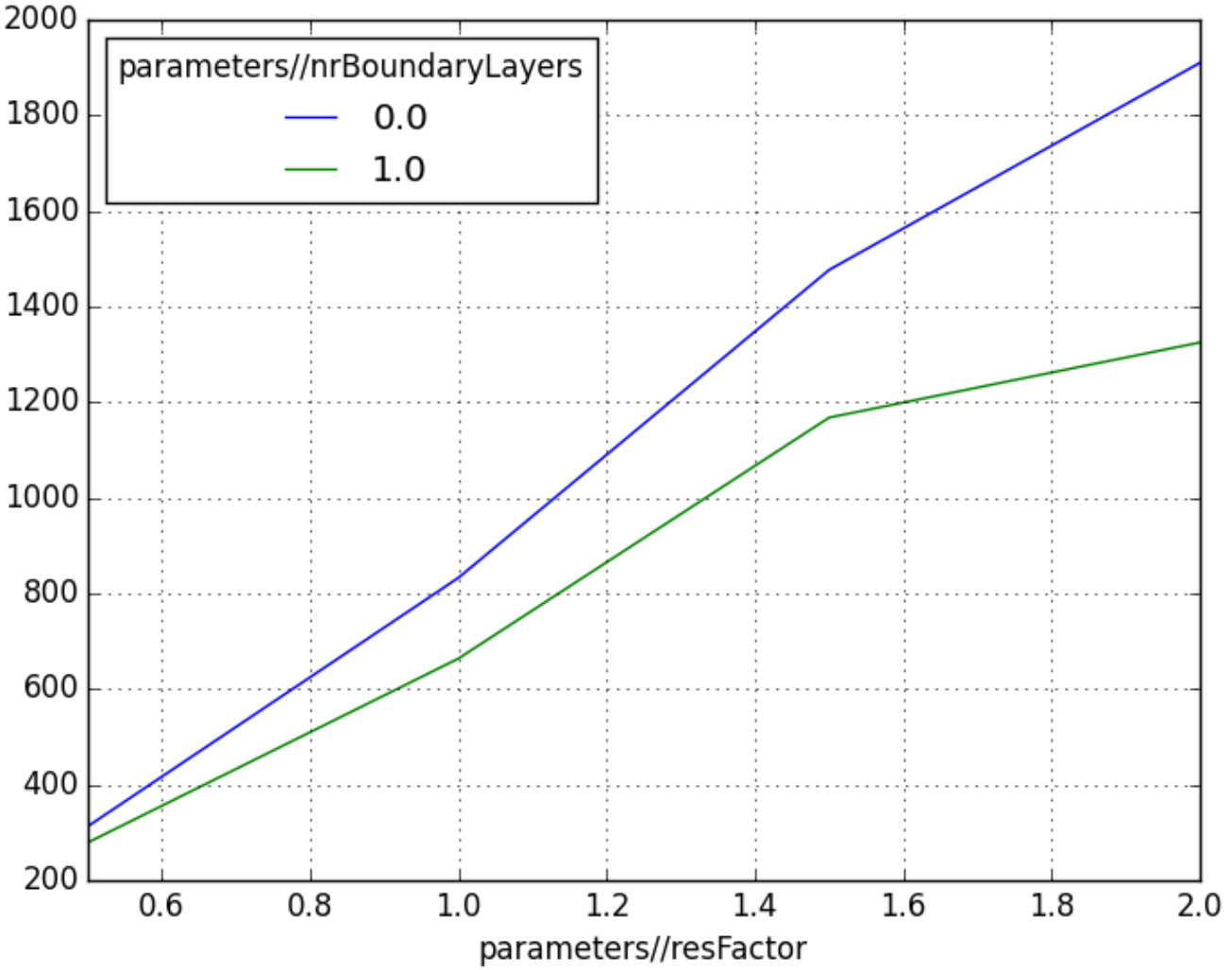
Shell

```
> pyFoamDumpRunDatabaseToCSV.py stage10_parameterVariation/meshResolution.variation.database pitzDaily.xlsx --interactive-after --<brk>
      <cont>excel

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

Python 3.10.6 (main, May 29 2023, 11:10:38) [GCC 11.3.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.31.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:  %matplotlib
Using matplotlib backend: TkAgg
In [2]: dump=self.getData()["dump"]
In [3]: dump.keys()
Out[3]:
Index(['runInfo//insertionTime', 'runInfo//lines', 'runInfo//uniqueid',
       'runInfo//logfile', 'runInfo//casefullname', 'runInfo//casename',
       'runInfo//solver', 'runInfo//solverFull', 'runInfo//commandLine',
       'runInfo//hostname', 'runInfo//remark', 'runInfo//starttime',
...
In [4]: pivot=dump[dump["parameters//solver"]=="simpleFoam"].pivot("parameters//resFactor","parameters//nrBoundaryLayers")
In [5]: pivot["runInfo//time"]
Out[5]:
parameters//nrBoundaryLayers    0.0    1.0
parameters//resFactor
0.5                            167.0  156.0
1.0                            278.0  244.0
1.5                            483.0  403.0
2.0                            667.0  626.0
In [6]: pivot["runInfo//time"].plot()
In [7]: pivot2=dump[dump["parameters//solver"]=="rhoSimpleFoam"].pivot("parameters//resFactor","parameters//nrBoundaryLayers")
In [8]: pivot["runInfo//time"]/pivot2["runInfo//time"]
Out[8]:
parameters//nrBoundaryLayers       0.0       1.0
parameters//resFactor
0.5                            1.012121  1.083333
1.0                            0.985816  1.000000
1.5                            1.025478  1.000000
2.0                            1.030912  1.001600
```

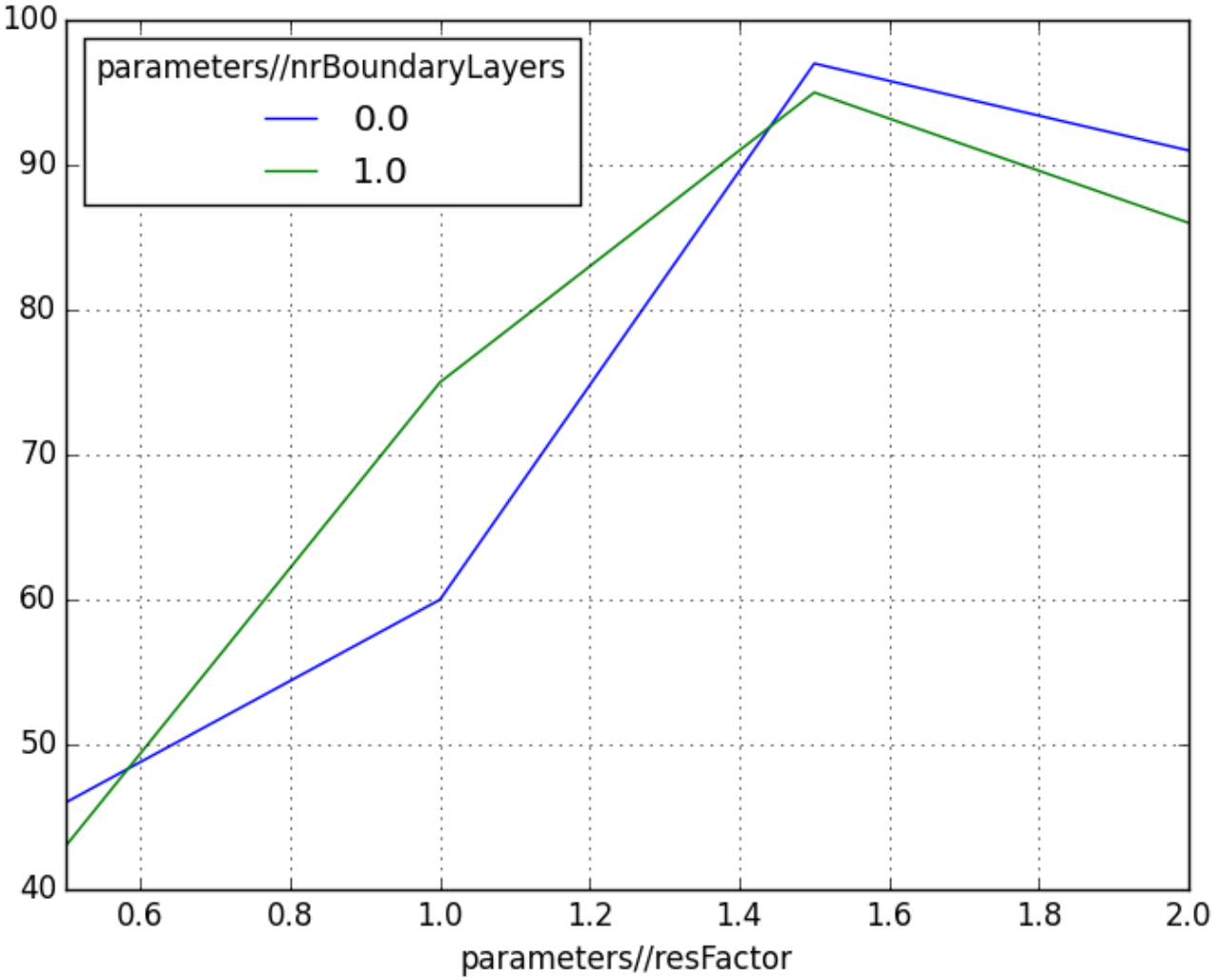
Analyzing the variation data

- This only gives a glimpse of what is possible
  - Wouldn't draw any conclusion on this concrete case
- Data is stored in formats that require no additional Python-modules
  - *Pickled* files
  - `SQlite` databases
- So it should be easy to write your own solution
- For analyzing use the toolset you're most comfortable with
  - `pandas`
  - Excel ...

Iterations of simpleFoam

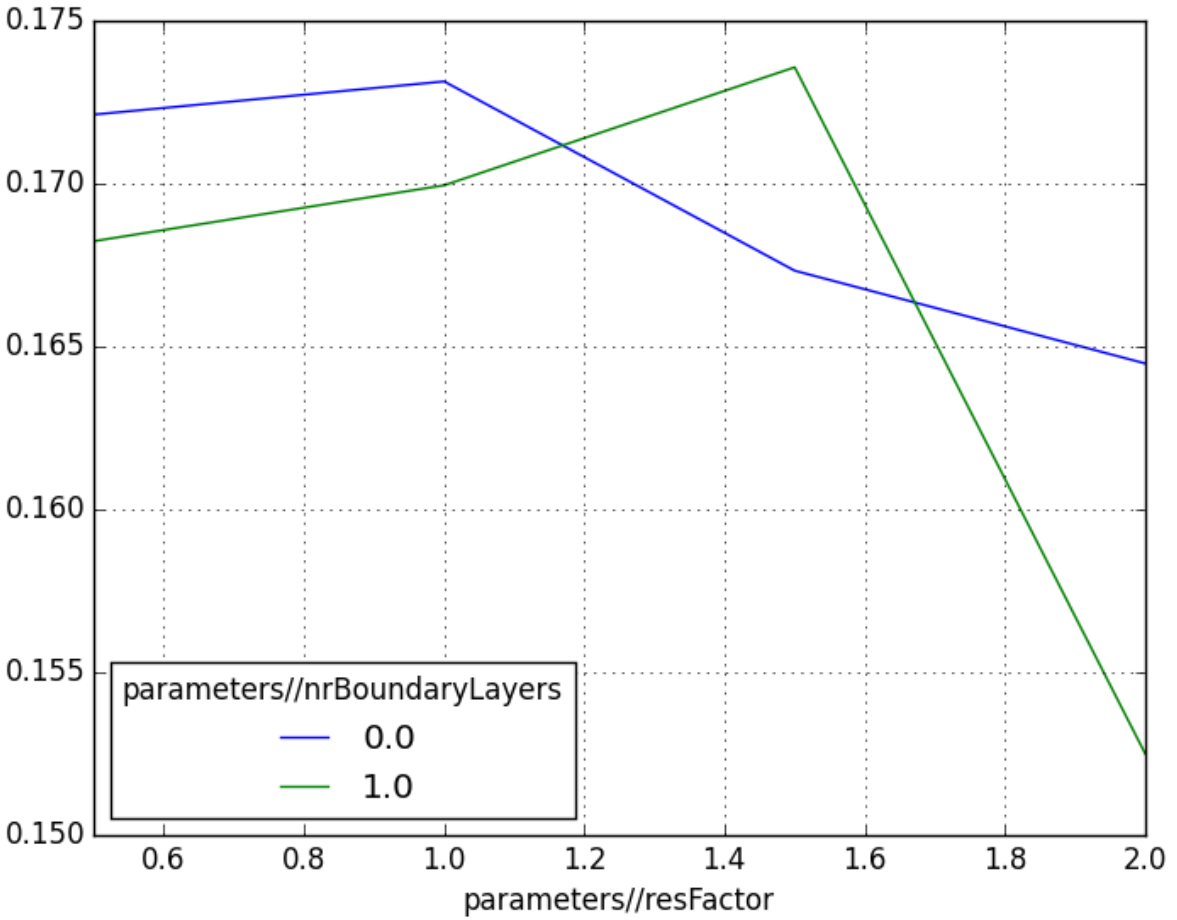


Iterations of pUCoupledFoam



Not yet converged

```
pivot["analyzed//Custom//detatch//x"].plot()
```



End of game

To get to the current state use `stage10_parameterVariation.tar.gz`

Outline

1 Introduction
- About this presentation
- Who is this?
- Technicalities
- Case setup in OpenFOAM
- PyFoam overview
- Our case

2 Using templates
- Stage 0: pyFoamPrepareCase.py without modifications
- Stage 1: Adding information
- Stage 2: Calculations and Variables
- Stage 3: Control structures
- Stage 4: Loops

3 Scripting
- Stage 5: Improvements to the mesh
- Stage 6: non-trivial mesh creation
- Stage 7: Complex initial conditions

4 Advanced
- Stage 8: Switching Foam-Versions
- Stage 9: Parallelization
- Stage 10: Parameter Variations
- Other topics

5 Conclusions

Switching off phases

- There is a number of switches that switches off certain phases
- Most used: `--no-meshing`
  - Exampled: Meshing is a big `snappyHexMesh`
    - You don't want to wait hours just because you set up new boundary conditions

Version control systems

- Setting up a case for `pyFoamPrepareCase.py` is a bit like programming
- When programming it is good to use a *version control system*
  - Mercurial
  - Git
  - ...
- Only put the files you actually edit under version control
- Benefits:
  - Documentation on "when was this set and why"
  - Branching allows trying things

Outline

1. Introduction
   - About this presentation
   - Who is this?
   - Technicalities
   - Case setup in OpenFOAM
   - PyFoam overview
   - Our case
2. Using templates
   - Stage 0: pyFoamPrepareCase.py without modifications
   - Stage 1: Adding information
   - Stage 2: Calculations and Variables
   - Stage 3: Control structures
   - Stage 4: Loops
3. Scripting
   - Stage 5: Improvements to the mesh
   - Stage 6: non-trivial mesh creation
   - Stage 7: Complex initial conditions
4. Advanced
   - Stage 8: Switching Foam-Versions
   - Stage 9: Parallelization
   - Stage 10: Parameter Variations
   - Other topics
5. Conclusions

What we learned

- "Programming" a case to be set up with `pyFoamPrepareCase.py`
  - Disadvantage:
    - Needs a bit of work and thought
  - Advantages:
    - Once done it is a two-step to set up a case and run it
    - Harder to make mistakes while setting it up
- We only showed a little example case, but this utility has been successfully used for big, complex cases
  - Automatically adding large numbers of STLs, snapping them. Calculating initial conditions depending on the conditions of the STLs

Further options

- Not every option of `pyFoamPrepareCase.py` has been covered
  - Like every PyFoam-utility it has a `--help` command line option
  - That lists every option

Recommended Execises

- Set up `fvSchemes` to try different discretization schemes
- Adapt so that transient solvers can be used as well
- Modify to use a `snappyHexMesh`-geometry
- Find the "real" detachment point

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