swak4Foam and PyFoam

Introducing them as a pair.
Version 1.1

Bernhard F.W. Gschaider

Innovative Computational Engineering

Ann Arbor, USA
30. June 2015

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Outline I

Outline II

Outline III

Exporting data

6 Conclusions

Outline

Outline

What it's about

- Two pieces of software
  - swak4Foam
  - pyFoam
- . . . and how they can ease your life with OpenFOAM

Intended audience and aim

- Intended audience for this presentation:
  - people who already worked a bit with OpenFOAM

    worked a bit means: been through the tutorials and set up a case on their own
  - have heard that `PyFoam` and `swak4Foam` exist
- Aim of the presentation
  - Enable user to start using PyFoam and swak4Foam
  - No programming
- The presentation is designed so that all steps can be reproduced using the information on the slides
  - No training files are provided

Format of the presentation

- This is a hands-on tutorial
- We will use a standard tutorial case
- Modify it till it doesn't look like the original
- No additional files are needed
  - Everything you have to enter will be spelled out on the slides

Limitation

- In 2 hours we can only give superficial overview of the two packages
  - It is not sure whether we'll even be able to complete it
- For a complete reference of the swak-expressions have a look at the *Incomplete reference guide* that comes with `swak`
  - Expressions are completely described
  - Almost everything else is missing

Outline

What is PyFoam

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

What is swak4Foam

From
http://openfoamwiki.net/index.php/Contrib/swak4Foam

swak4Foam stands for SWiss Army Knife for Foam. Like that knife it rarely is the best tool for any given task, but sometimes it is more convenient to get it out of your pocket than going to the tool-shed to get the chain-saw.

- It is the result of the merge of
  - funkySetFields
  - groovyBC
  - simpleFunctionObjects

  and has grown since
- The goal of swak4Foam is to make the use of C++ unnecessary
  - Even for complex boundary conditions etc

The core of swak4Foam

- At its heart swak4Foam is a collection of parsers (subroutines that read a string and interpret it) for expressions on OpenFOAM-types
  - fields
  - boundary fields
  - other (faceSet, cellZone etc)
- ... and a bunch of utilities, function-objects and boundary conditions that are built on it
- swak4foam tries to reduce the need for throwaway C++ programs for case setup and postprocessing

Outline

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
```

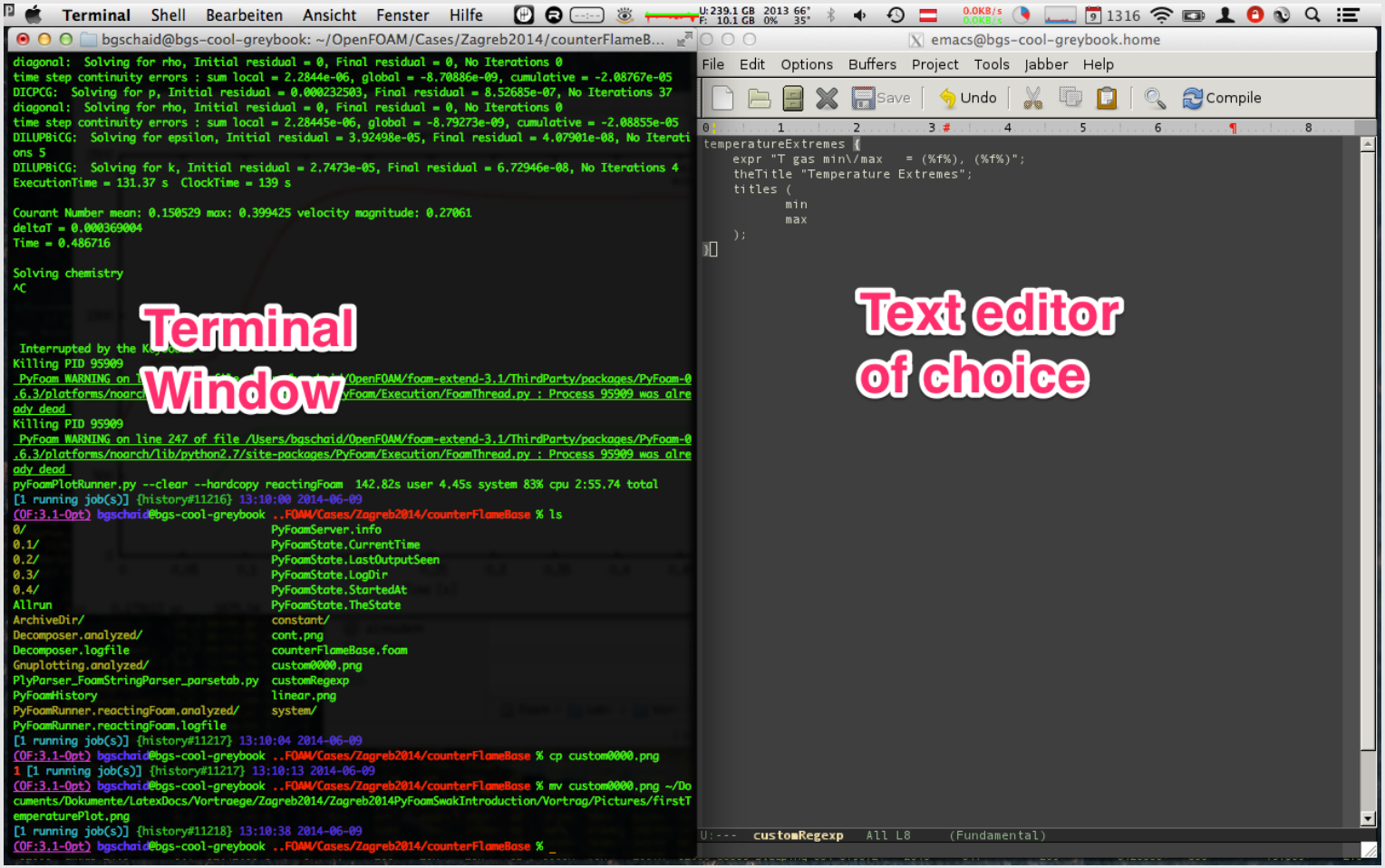
- Long examples will be a white box
  - Input will be prefixed with a > and blue
  - Long lines will be broken up
    - A pair of <brk> and <cont> indicates that this is still the same line in the input/output
  - «snip» in the middle means: "There is more. But it is boring"

```
> this is an example for a very long command line that does not fit onto one line of <brk>
      <cont>the slide but we 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 <brk>
      <cont>its glory
```

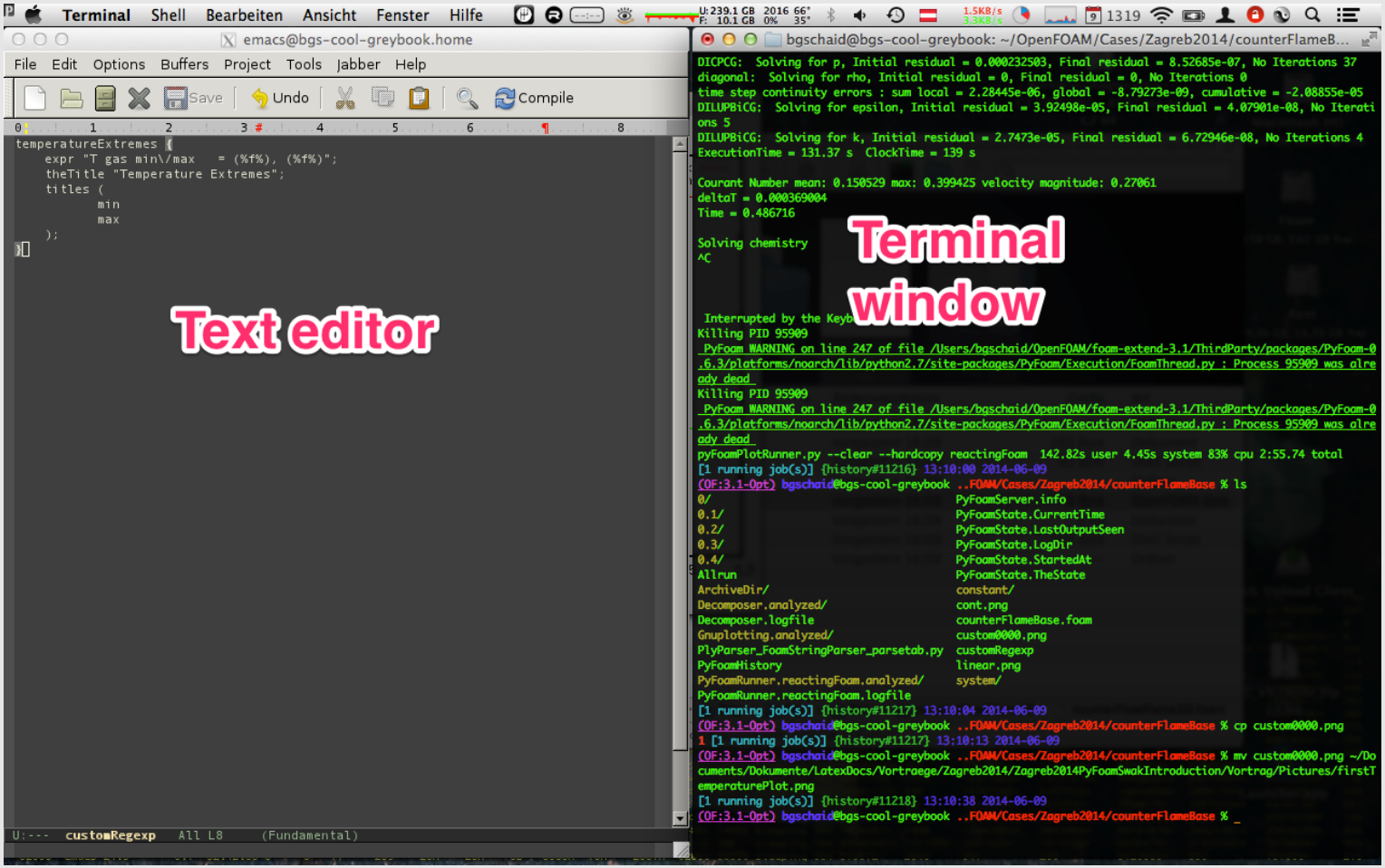
Work environment

- You will use two programs
  - A terminal
  - A text-editor
- For the text-editor you have the choice (these should be installed):
  - Emacs (king of text-editors)
  - VI
  - Kate with KDE
  - Gedit with Gnome
  - `nano`
  - `jedit`
  - `...`

Recommended screen layout



Setup for non-conformists



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Getting onto the same page

- During the remaining presentation we assume that
  - the `zsh` is used (optional. `bash` works too)
  - we use `foam-extend 3.1` (required)
- Switch to `zsh`

```
> zsh
```

- You should see a more colorful prompt with `(OF:-)` on the left
  - Only with correct environment set (probably only on the stick)
- Switch on Foam-Extend-3.1

```
> foamExtend
```

- Now the prompt should show `(OF:3.1-Opt)`
- Create a working directory and go there

```
> mkdir PyFoamAndSwak; cd PyFoamAndSwak
```

Make sure PyFoam is working

- There is a utility that helps make sure that PyFoam is working
  - and gives valuable information for support

```
Machine info: Darwin | bgs-cool-greybook | 14.3.0 | Darwin Kernel Version 14.3.0: Mon Mar 23 11:59:05 PDT <brk>
     <cont>2015; root:xnu-2782.20.48~5/RELEASE_X86_64 | x86_64 | i386

Python version: 3.4.3 (default, May 25 2015, 18:48:21)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)]

Python executable: /opt/local/bin/python

Python 3 is supported with PyFoam
PYTHONPATH: /Users/bgschaid/private_python:

Location of this utility: /Users/bgschaid/Development/OpenFOAM/Python/PyFoam/bin/pyFoamVersion.py

Version () Fork openfoam of the installed 16 versions:
                  extend-3.0 : /Users/bgschaid/foam/foam-extend-3.0
                  extend-3.1 : /Users/bgschaid/foam/foam-extend-3.1
      extend-3.1-checkScript : /Users/bgschaid/foam/foam-extend-3.1-checkScript
                  extend-3.2 : /Users/bgschaid/foam/foam-extend-3.2
            openfoam-1.6-ext : /Users/bgschaid/OpenFOAM/OpenFOAM-1.6-ext
openfoam-1.6-ext-nextRelease : /Users/bgschaid/OpenFOAM/OpenFOAM-1.6-ext-nextRelease
             openfoam-1.6.x : /Users/bgschaid/OpenFOAM/OpenFOAM-1.6.x
             openfoam-1.7.x : /Users/bgschaid/OpenFOAM/OpenFOAM-1.7.x
       openfoam-1.7.x-clean : /Users/bgschaid/OpenFOAM/OpenFOAM-1.7.x-clean
     openfoam-1.7.x-vectorN : /Users/bgschaid/OpenFOAM/OpenFOAM-1.7.x-vectorN
             openfoam-2.0.x : /Users/bgschaid/OpenFOAM/OpenFOAM-2.0.x
             openfoam-2.1.x : /Users/bgschaid/OpenFOAM/OpenFOAM-2.1.x
             openfoam-2.2.x : /Users/bgschaid/OpenFOAM/OpenFOAM-2.2.x
     openfoam-2.2.x-vectorN : /Users/bgschaid/OpenFOAM/OpenFOAM-2.2.x-vectorN
             openfoam-2.3.x : /Users/bgschaid/OpenFOAM/OpenFOAM-2.3.x
             openfoam-2.4.x : /Users/bgschaid/OpenFOAM/OpenFOAM-2.4.x
  This version of OpenFOAM uses the old calling convention

pyFoam-Version: 0.6.5

Path where PyFoam was found (PyFoam.__path__) is ['/Users/bgschaid/private_python/PyFoam']

Configuration search path: [('file', '/etc/pyFoam/pyfoamrc'), ('directory', '/etc/pyFoam/pyfoamrc.d'), ('file<brk>
     <cont>', '/Users/bgschaid/.pyFoam/pyfoamrc'), ('directory', '/Users/bgschaid/.pyFoam/pyfoamrc.d')]
Configuration files (used): ['/Users/bgschaid/.pyFoam/pyfoamrc', '/Users/bgschaid/.pyFoam/pyfoamrc.d/testit.<brk>
     <cont>cfg']
```

pyFoamVersion.py

- Information the utility gives
  - Machine
  - Used python
  - PYTHONPATH (where additional libraries are searched)
  - Information about the used PyFoam
    - Where configuration files are sought
  - Installed libraries relevant for PyFoam
    - With version if possible
- This information helps diagnosing problems
  - Copy this output when reporting problems that might be associated with the installation

Make sure swak4Foam is installed

- Call the most popular utility of swak4Foam
  - swakVersion reported below the usual header

```
> funkySetFields
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     3.2                                |
|   \\  /    A nd           | Web:         http://www.extend-project.de       |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build     : 3.2
Exec      : funkySetFields
Date      : Jun 05 2015
Time      : 17:19:31
Host      : bgs-cool-greybook
PID       : 25473
CtrlDict  : "/Volumes/Foam/Workshop2015/counterFlowFlame2D/system/controlDict"
Case      : /Volumes/Foam/Workshop2015/counterFlowFlame2D
nProcs    : 1
SigFpe    : Enabling floating point exception trapping (FOAM_SIGFPE).

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
swakVersion: 0.3.2 (Release date: 2015-05-31)
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


--> FOAM FATAL ERROR:
funkySetFields: time/latestTime option is required


    From function main()
    in file funkySetFields.C at line 723.

FOAM exiting
```

Outline

Outline

The case

- We're going to use a plain tutorial case
  - Add stuff to it until the original author won't recognize it anymore
- The case is `counterFlowFlame2D` for the `reactingFoam` solver
  - Simple combustion case
  - Plain `blockMesh`
    - On one side 100% mixture of CH_4 comes in
    - On the other side 23% of O_2
    - Burns in the middle
    - Products leave on top and bottom

$$CH_4 + 2O_2 \rightleftharpoons CO_2 + 2H_2O$$

counterFlowFlame2D overview

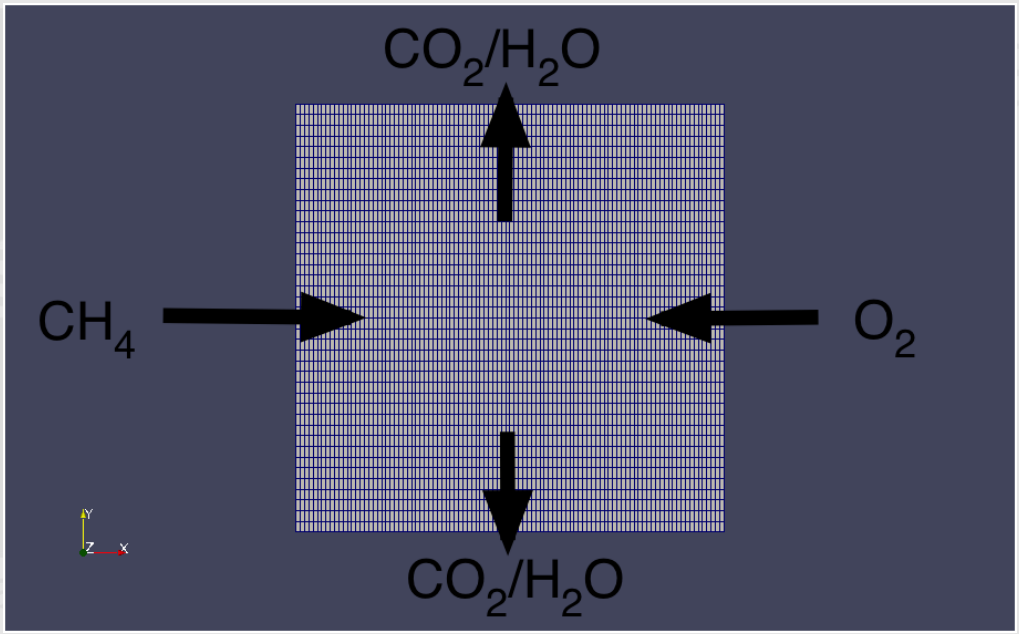


Figure: Species flowing in and out

Cloning

- First we get us the case
  - But only the things that are important
- We use the first PyFoam-utility for it
  - And afterwards check the results

```
> pyFoamCloneCase.py $FOAM_TUTORIALS/combustion/reactingFoam/ras/<brk>
    <cont>counterFlowFlame2D counterFlameBase
 PyFoam WARNING on line 117 of file /Users/bgschaid/OpenFOAM/foam-extend<brk>
      <cont>-3.1/ThirdParty/packages/PyFoam-0.6.3/platforms/noarch/lib/<brk>
      <cont>python2.7/site-packages/PyFoam/Applications/CloneCase.py : <brk>
      <cont>Directory  does not exist. Creating
> ls counterFlameBase
0/                      PyFoamHistory           counterFlameBase.foam
Allrun                  constant/               system/
```

What is cloned

- Files essential for the case
  - Initial directory `0` (but not other times)
  - `system`
  - `constant`
  - Files like `Allrun`
- Some files are created

  PyFoamHistory PyFoam-commands log their activity here
  counterFlameBase.foam A stub-file for the native Paraview-reader
- Some PyFoam-specific files are added here

What else can pyFoamCloneCase.py do for me

All PyFoam-utilities have a --help-option:

```
pyFoamCloneCase.py --help
Usage
=====
  pyFoamCloneCase.py <source> <destination>

Clones a case by copying the system, constant and 0-directories  If the <brk>
     <cont>case
is under VCS then the cloning mechanism of the VCS is used

Options
=======
--version                 show program's version number and exit
--help, -h                show this help message and exit

Default
-------
Options common to all PyFoam-applications

--psyco-accelerated       Accelerate the script using the psyco-library
                          (EXPERIMENTAL and requires a separatly installed
                          psyco)
--profile-python          Profile the python-script (not the OpenFOAM-program<brk>
     <cont>) -
                          mostly of use for developers
--profile-cpython         Profile the python-script (not the OpenFOAM-program<brk>
```

What we find in `help`

- Short description of the utility
- Options organized in sections
  - Options common to (most) PyFoam-utlities
    - For instance options that help with debugging
  - Options specific to the utility
    - For instance `--add-item` allows adding files/directories to be added to the clone
    - Possible application "Add the directory 10 as well"

Entering the case

- Next we enter the case directory

```
> cd counterFlameBase
```

- Prepare the mesh

```
> blockMesh
```

- Check that everything is alright

```
> checkMesh
```

Outline

Running the case

The probably most-used PyFoam-utility

```
> pyFoamPlotRunner.py reactingFoam
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     3.1                                |
|   \\  /    A nd           | Web:         http://www.extend-project.de       |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build    : 3.1
Exec     : reactingFoam
Date     : Jun 08 2014
Time     : 16:11:38
Host     : bgs-cool-greybook
PID      : 83164
CtrlDict : /Users/bgschaid/OpenFOAM/foam-extend-3.1/etc/controlDict
Case     : /Volumes/Foam/Cases/counterFlameBase
nProcs   : 1
SigFpe   : Enabling floating point exception trapping (FOAM_SIGFPE).

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Create mesh for time = 0

Reading chemistry properties
```

Window Nr 1 popping up

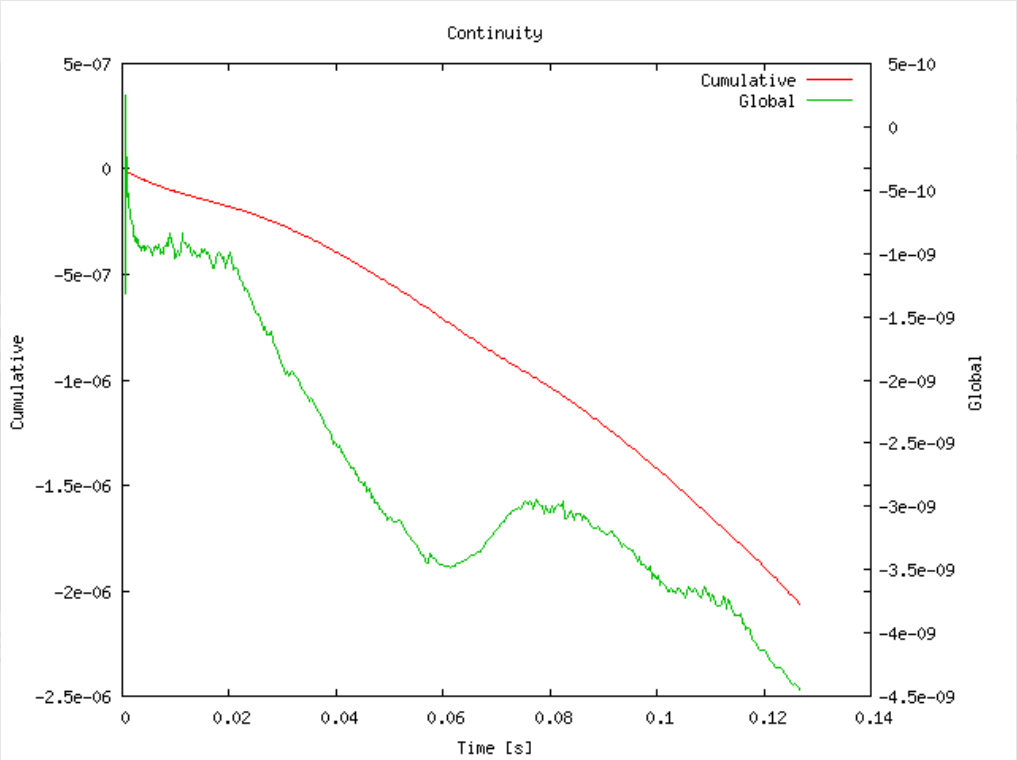


Figure: Continuity graph (ongoing)

The other window

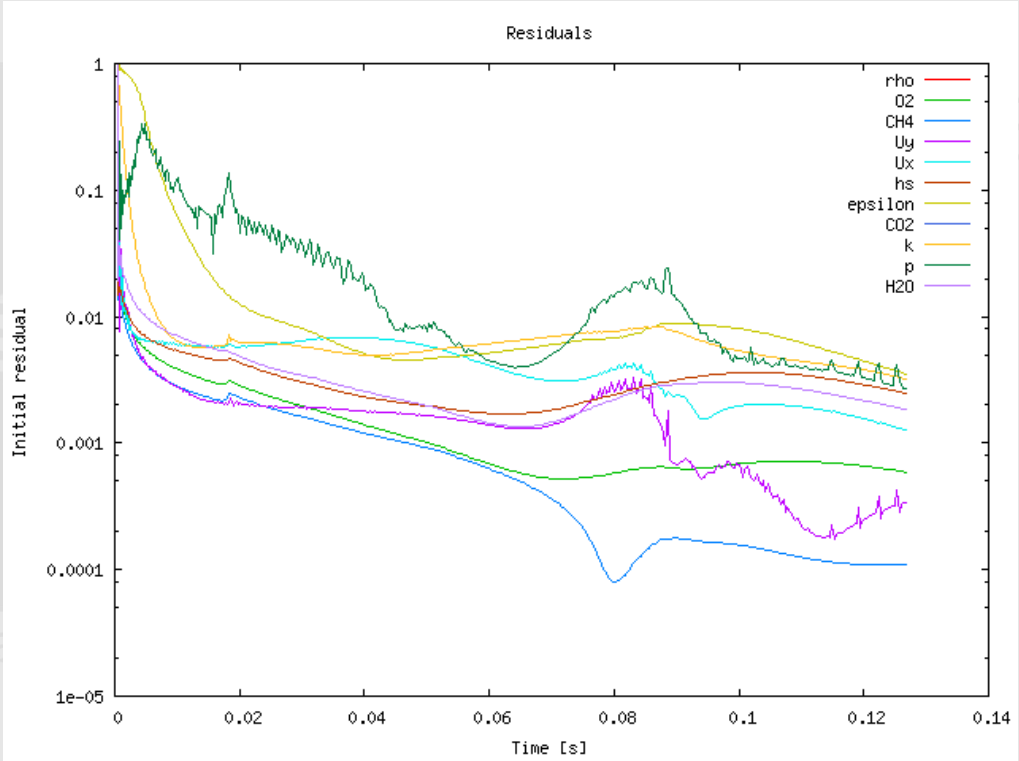


Figure: Residuals (ongoing)

How the directory looks afterwards

- Apart from the added time-directories there are other things that were not there before:

```
> ls
0/
0.1/
Allrun
Gnuplotting.analyzed/
PyFoamHistory
PyFoamRunner.reactingFoam.logfile
PyFoamServer.info
PyFoamState.CurrentTime
PyFoamState.LastOutputSeen
PyFoamState.StartedAt
PyFoamState.TheState
constant/
counterFlameBase.foam
system/
```

Added files

- PyFoam adds a number of files to the directory:

  PyFoamRunner.<solvername>.logfile A complete copy of what was written to the terminal

  PyFoamServer.info If you're using the network component of PyFoam this might help you

  PyFoamState.* Updated during the run and used by `pyFoamListCases.py` (another nice utility that you've got to find out about yourself)

  ***.analyzed** A directory with the results of the analysis. Contents usually are

  pickledPlots Information to recreate the plots

  pickledData Data about the run that can be read and processed by Python (of interest for scripters. Also see `pyFoamEchoPickledApplicationData.py`)

  pickledStartData, pickledUnfinishedData Versions of the above that are written during the run

  Log-files if the user chooses to write them

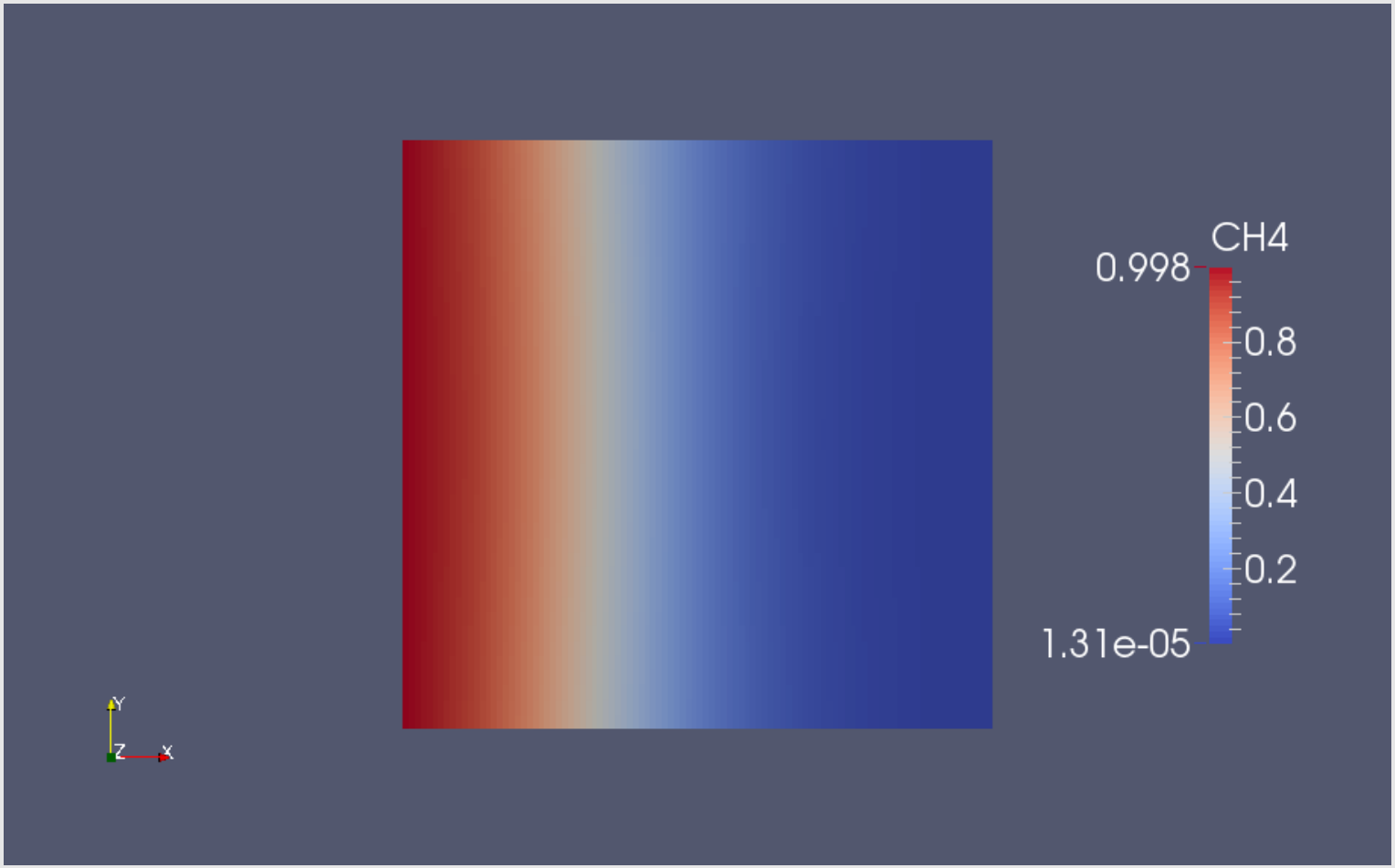
What did we actually simulate?

In case you forgot:

- Simple combustion case:
  - CH_4 coming in from the left
  - O_2 coming in from the right
  - Ignition in the middle
  - Outflow on top and bottom

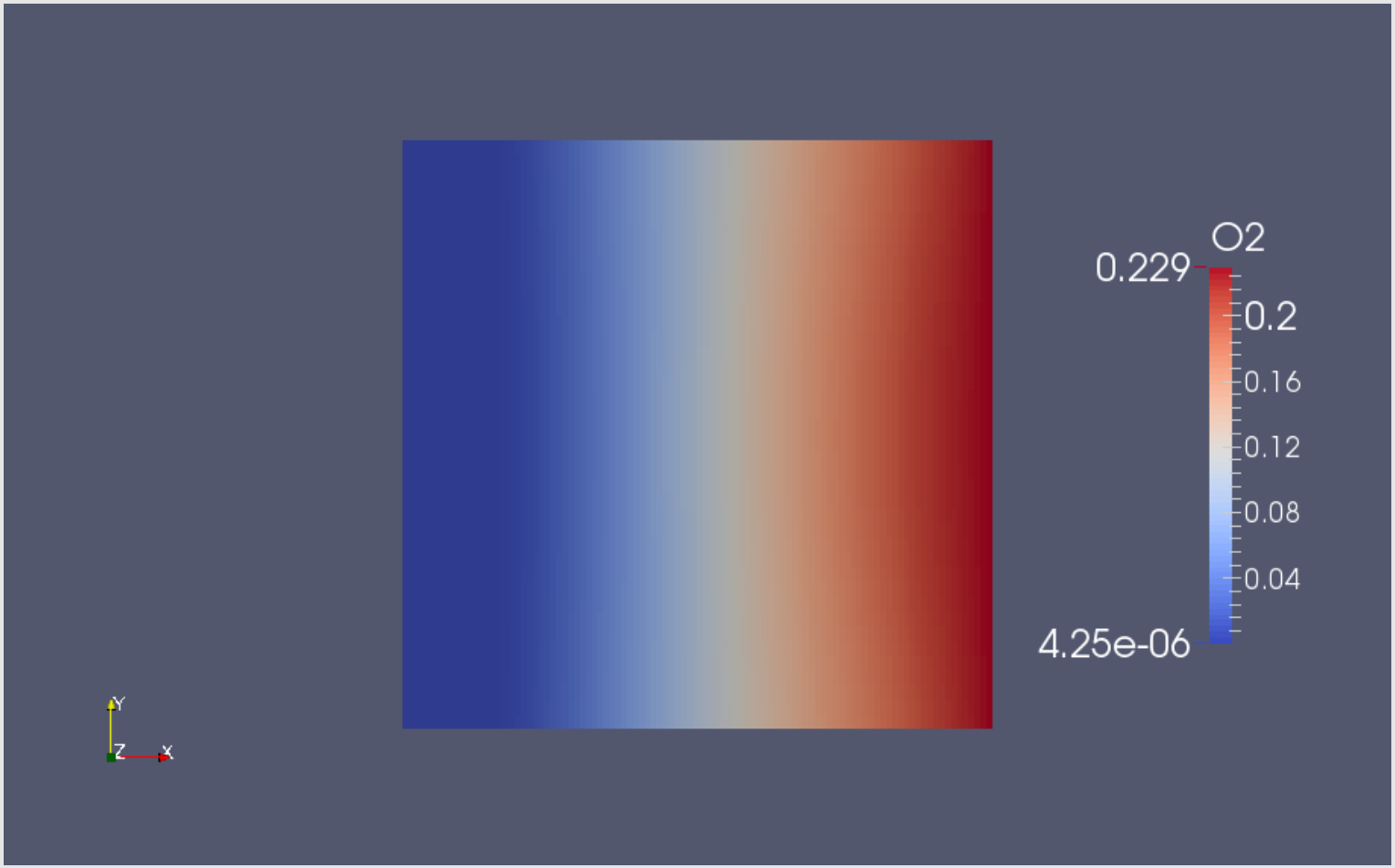
But let's look at the results:

Methan



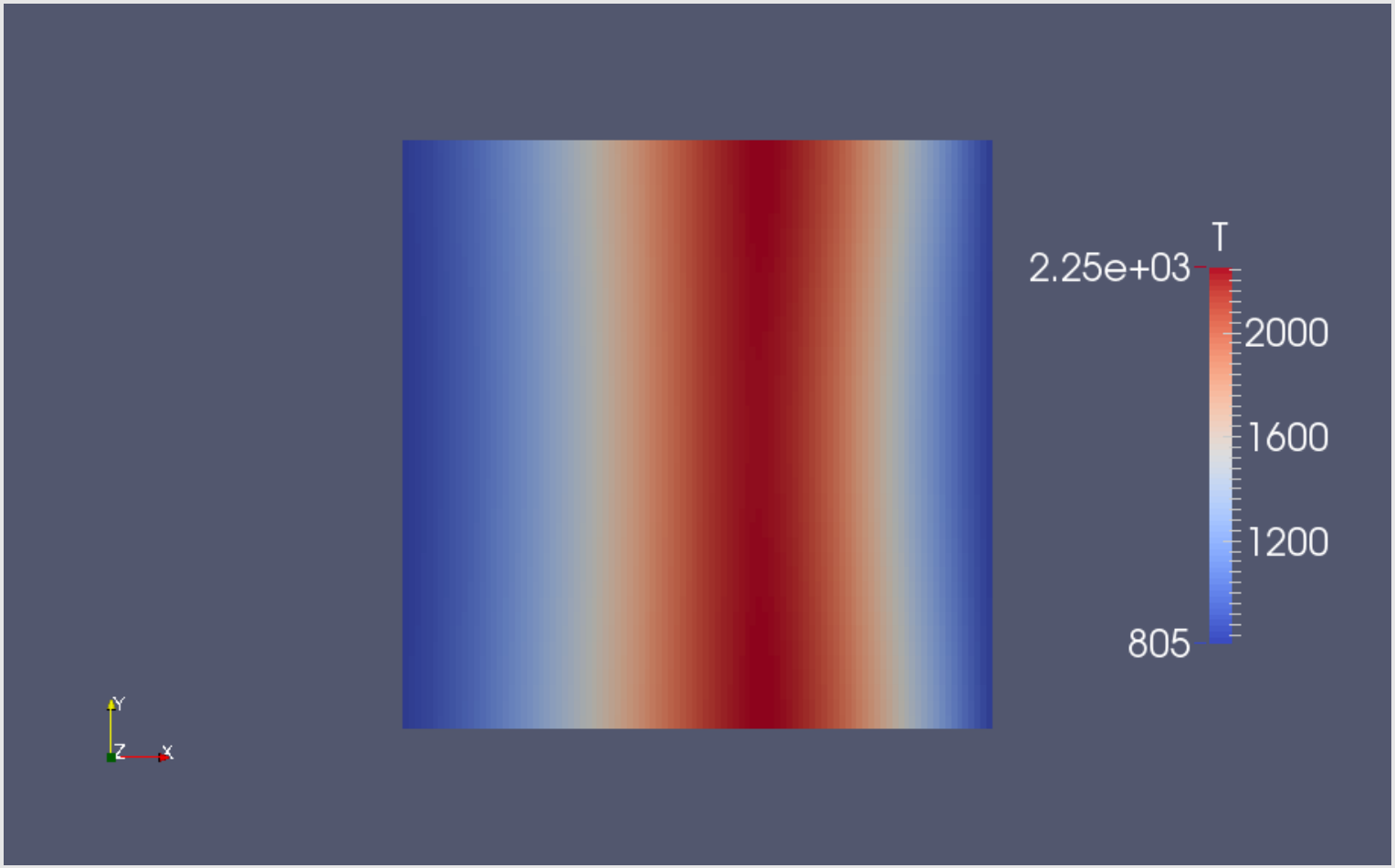
Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Oxygen



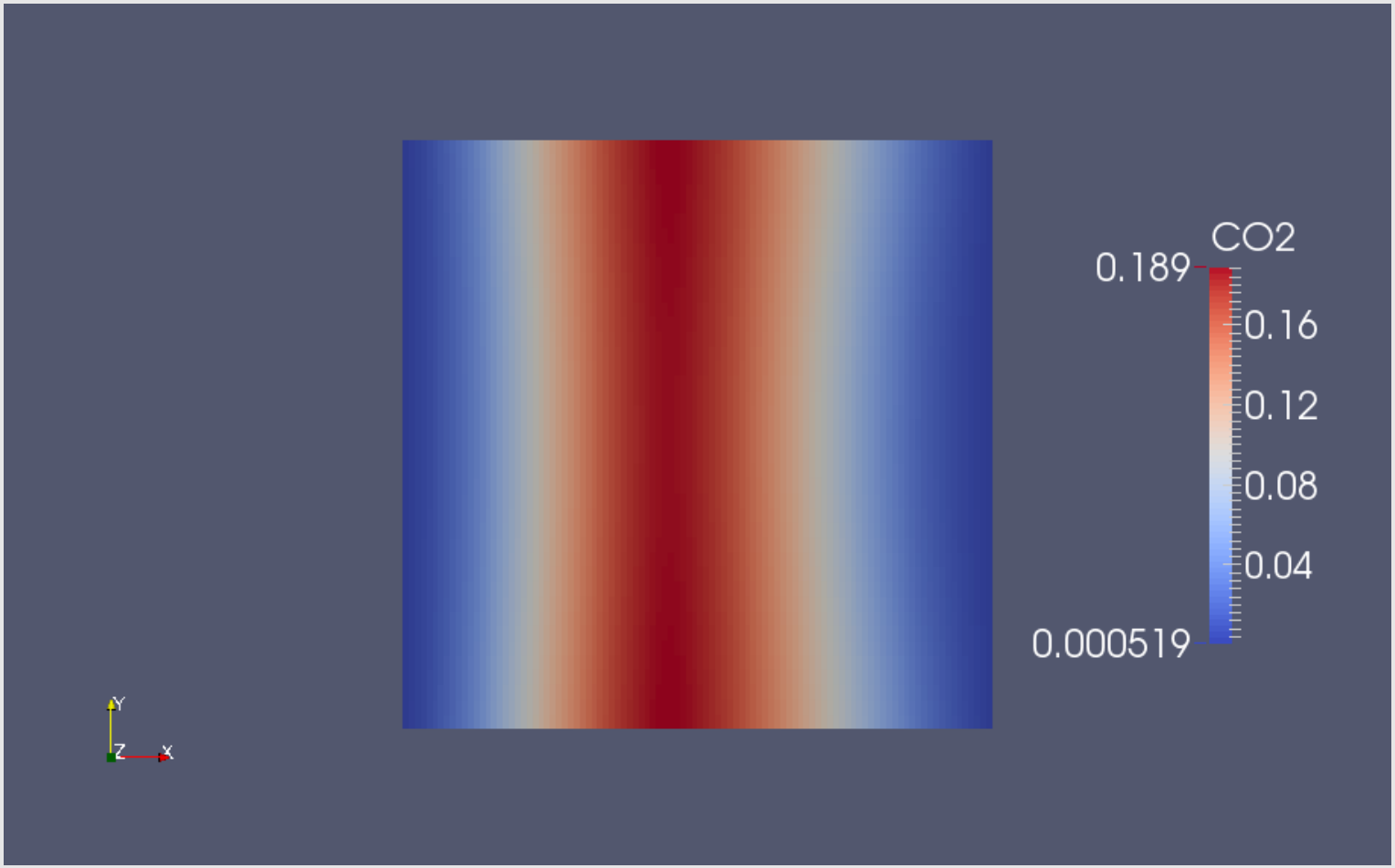
Strömungsforschung GmbH
ICE
Innovative Computational Engineering

The flame



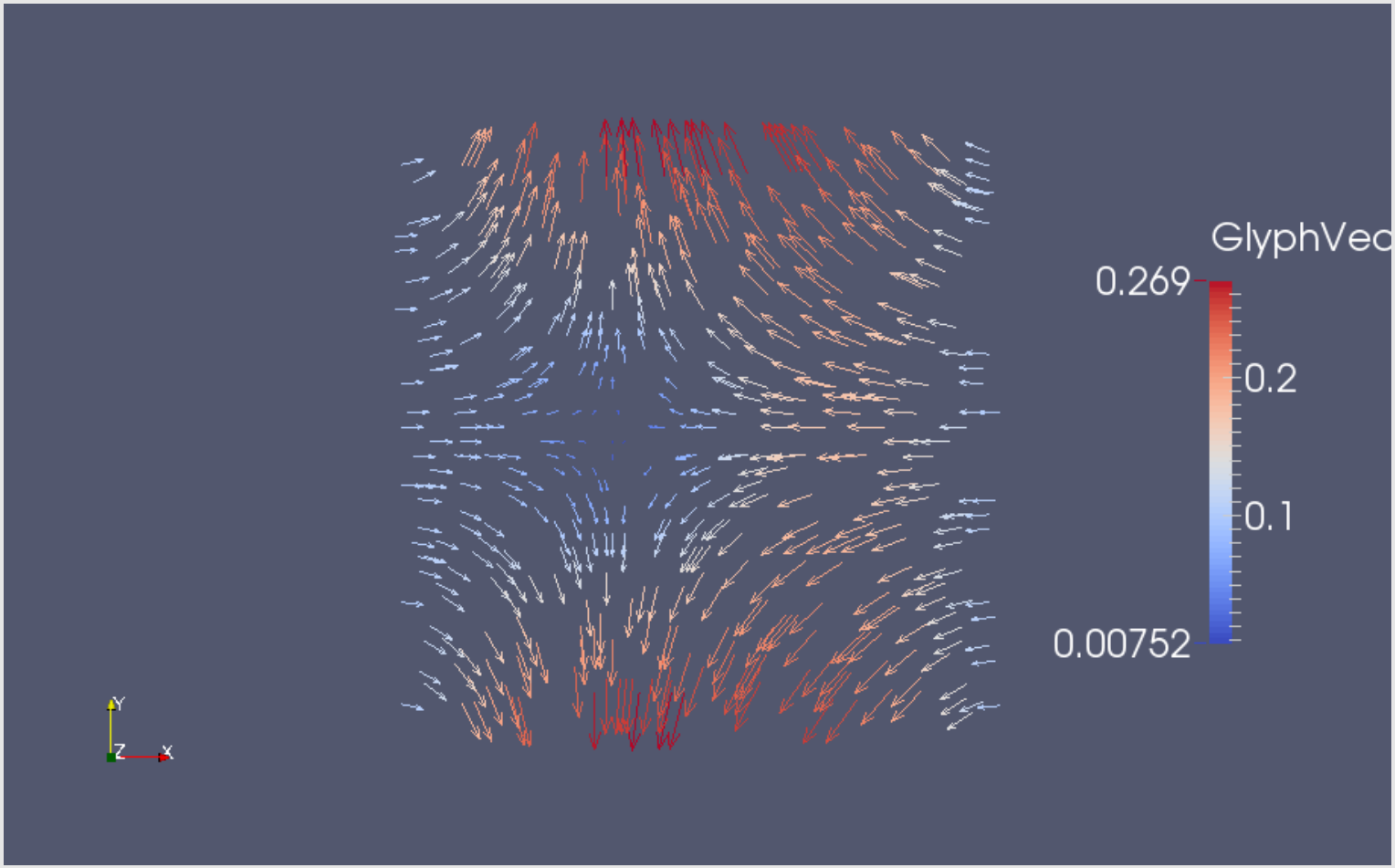
Strömungsforschung GmbH
ICE
Innovative Computational Engineering

The products



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

The flow



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Conventions for utilities

- Utilities that have a OpenFOAM-application (solver, utility) as an argument (`pyFoamPlotRunner.py` for instance):
  - First PyFoam-options (start with `--`)
    - No arguments ("non-options")
  - Then the OpenFOAM-applications
  - Options and arguments after that belong to OpenFOAM
- Other Utilities (`pyFoamClearCase.py` for instance)
  - Arguments and options can be mixed
  - Most utilities have the case they should work on as an argument
    - At least `.` for *current directory* is required

Getting a 1-page overview

A utility that prints information about a case in an easy-to-read form

```
> pyFoamCaseReport.py . --short-bc

Table of boundary conditions for t = 0
======================================

========== ============ ============ ============ ============
..         air          frontAndBack fuel         outlet
---------- ------------ ------------ ------------ ------------
Patch Type patch        empty        patch        patch
Length     40           8000         40           200
========== ============ ============ ============ ============
CH4        fixedValue   empty        fixedValue   inletOutlet
N2         fixedValue   empty        fixedValue   inletOutlet
O2         fixedValue   empty        fixedValue   inletOutlet
T          fixedValue   empty        fixedValue   inletOutlet
U          fixedValue   empty        fixedValue   zeroGradient
Ydefault   fixedValue   empty        fixedValue   inletOutlet
alphat     fixedValue   empty        fixedValue   zeroGradient
epsilon    fixedValue   empty        fixedValue   zeroGradient
k          fixedValue   empty        fixedValue   zeroGradient
mut        fixedValue   empty        fixedValue   zeroGradient
p          zeroGradient empty        zeroGradient fixedValue
========== ============ ============ ============ ============
```

Getting printed info

- Output of `pyFoamCaseReport.py` is *ReST* (*ReStructured Text*)
  - Can be converted with a number of utilities

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

Table of boundary conditions for t = 0

	air	frontAndBack	fuel	outlet
Patch Type	patch	empty	patch	patch
Length	40	8000	40	200
CH4	fixedValue	empty	fixedValue	inletOutlet
N2	groovyBC	empty	fixedValue	inletOutlet
O2	groovyBC	empty	fixedValue	inletOutlet
T	fixedValue	empty	fixedValue	inletOutlet
U	groovyBC	empty	groovyBC	zeroGradient
Ydefault	fixedValue	empty	fixedValue	inletOutlet
alphat	fixedValue	empty	fixedValue	zeroGradient
epsilon	fixedValue	empty	fixedValue	zeroGradient
k	fixedValue	empty	fixedValue	zeroGradient
mut	fixedValue	empty	fixedValue	zeroGradient
p	zeroGradient	empty	zeroGradient	fixedValue
sumSpec	zeroGradient	empty	zeroGradient	zeroGradient

Exercise: finding information

- What else can `pyFoamCaseReport.py` tell us?

Clearing the results

- We can clear the results with one command
  - All timesteps but the first
  - `system` and `constant` untouched

```
> pyFoamClearCase.py .
> ls
0/
Allrun
Gnuplotting.analyzed/
PyFoamHistory
PyFoamRunner.reactingFoam.logfile
PyFoamServer.info
PyFoamState.CurrentTime
PyFoamState.LastOutputSeen
PyFoamState.StartedAt
PyFoamState.TheState
constant/
counterFlameBase.foam
system/
```

Clear the PyFoam-stuff as well

- *PyFoam* leaves most of its own stuff untouched
- Additional data to clear can be specified

```
> pyFoamClearCase.py . --remove-analyzed --add="PyFoam*"
> ls
0/
Allrun
PyFoamHistory
constant/
counterFlameBase.foam
system/
```

- The History-file stays nevertheless

Friends of `pyFoamPlotRunner.py`

The functionality of `pyFoamPlotRunner.py` can be found in two other utilities:

pyFoamRunner.py Does all the `PlotRunner` does ... except plotting

- Applications:
  - running on text-only displays
  - long runs

pyFoamPlotWatcher.py Given a text file it parses it and plots the results

- Applications:
  - output of `pyFoamPlotRunner.py`
  - log files of OpenFOAM-runs (cluster for instance)

Options for `Runner` and/or `Plotter`

–clear-case Clears the case the way `pyFoamClearCase.py` does

–progress Swallow output of the solver and only print current time on terminal

–with-* Plot additional information like iteration, CPU-time etc

–hardcopy Generate PNG-files of the plots (that's how graphs for this presentation were made)

–write-files Write text files with the data from the plots

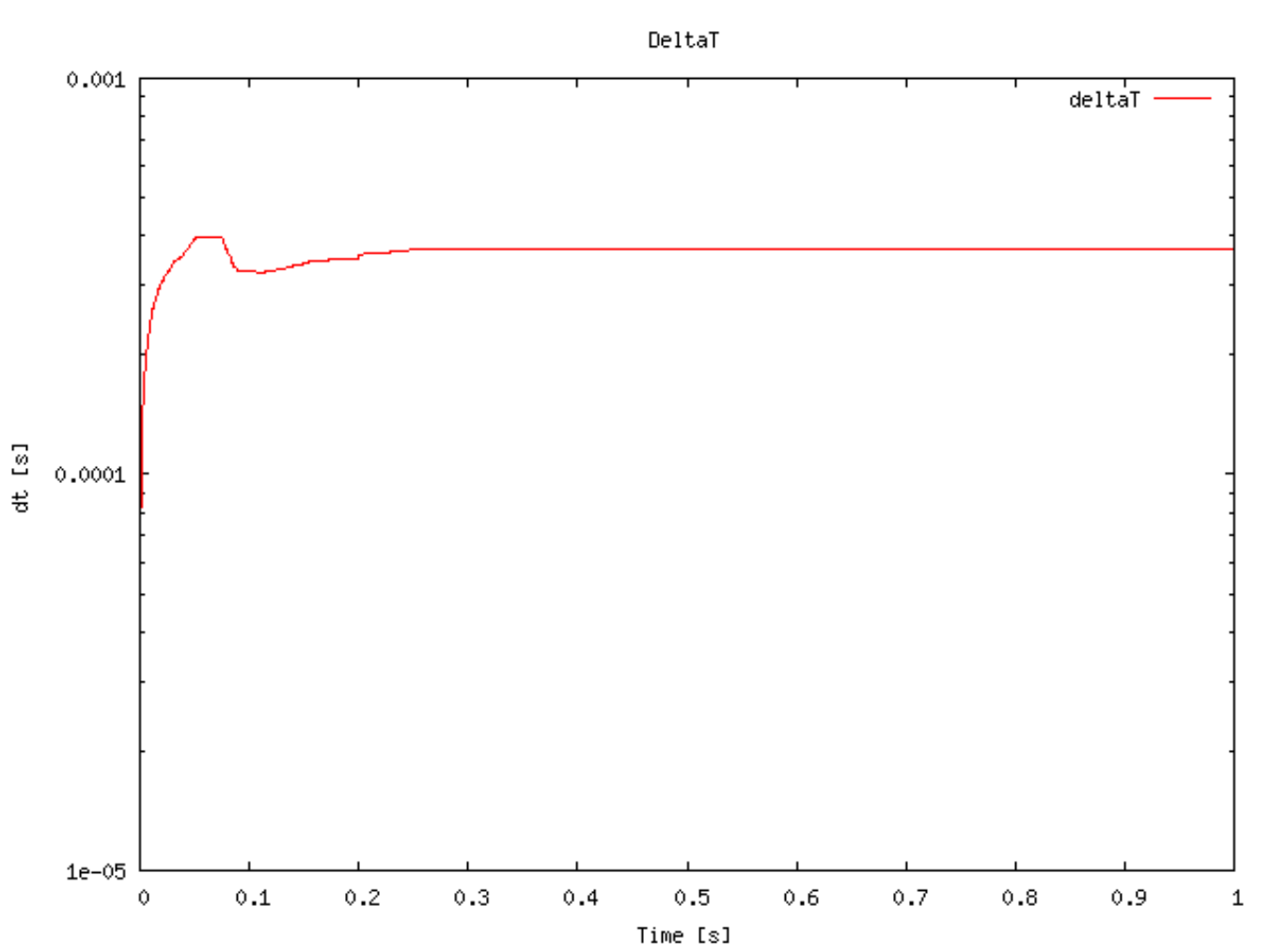
–write-all-timesteps modifies `controlDict` to write all data

- Only a good idea when you try to find problems

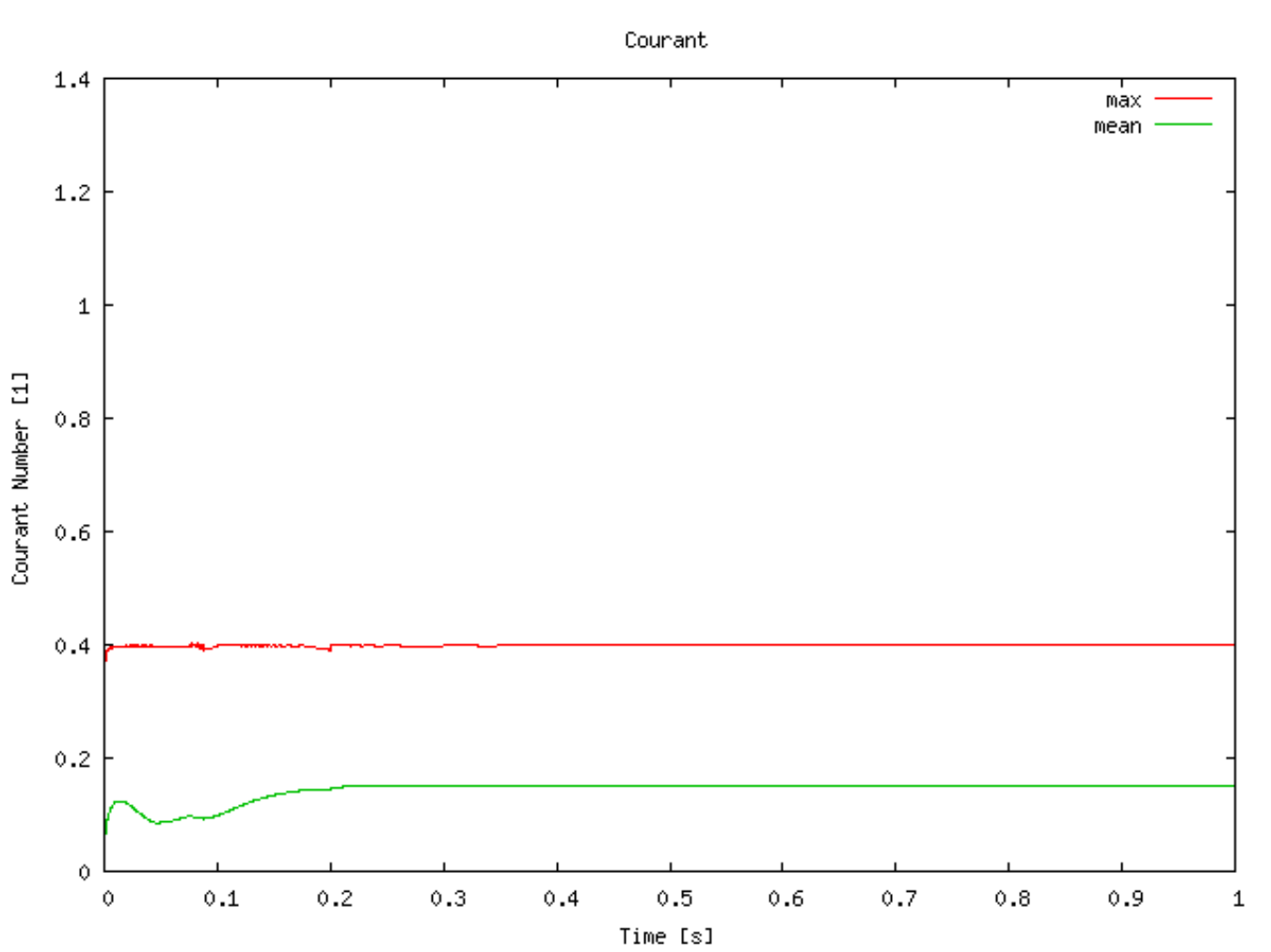
Exercise: Run. Then Plot

1. Start the simulation with the runner-utility (no plotting)
   - Make sure that data from the previous run is removed
   - only print the progress
2. Open another terminal window
   - Go to the directory
3. Use the plot watcher to plot data from the log file
   - In addition to the defaults plot at least the time-step

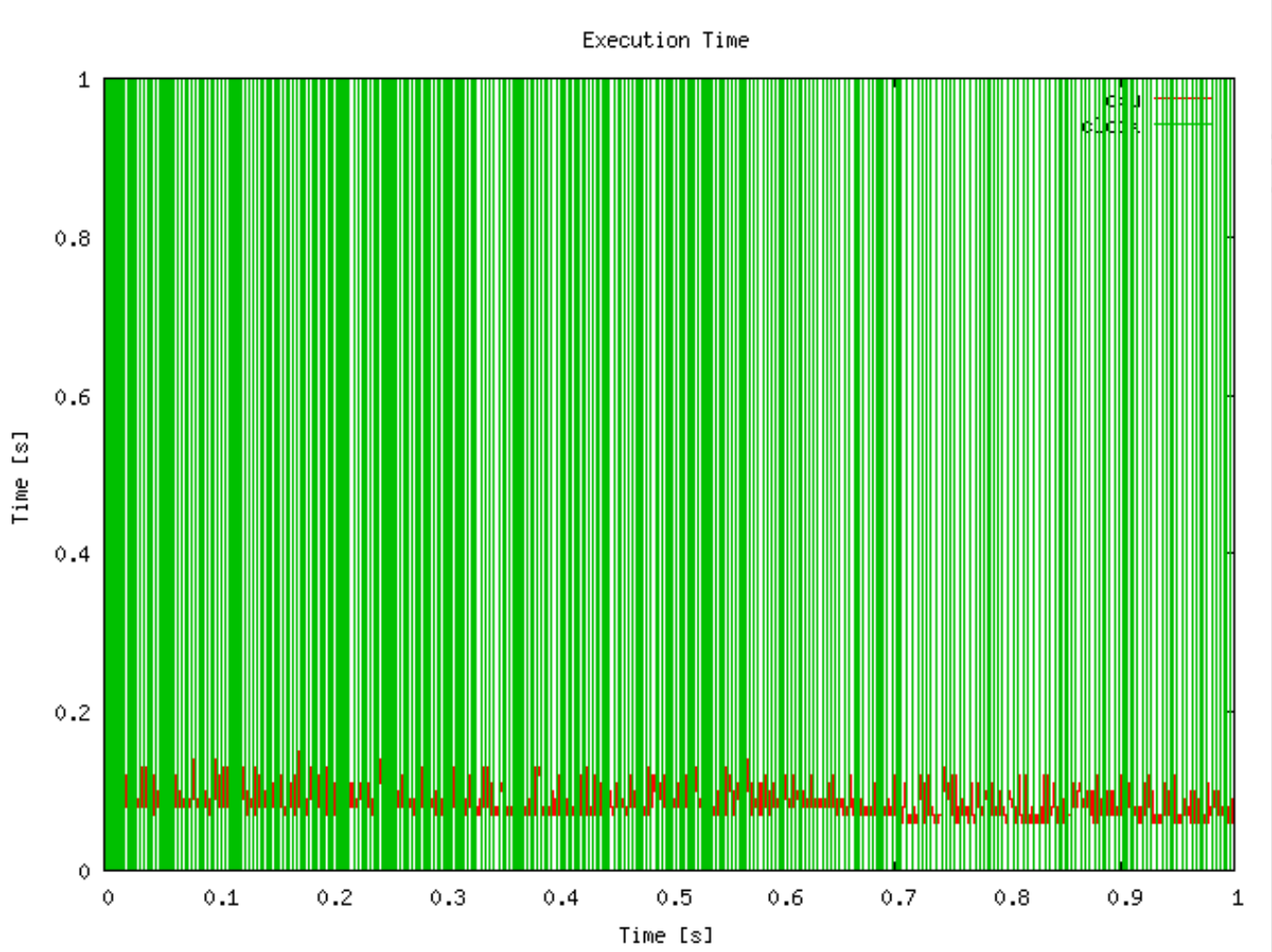
Timestep plot



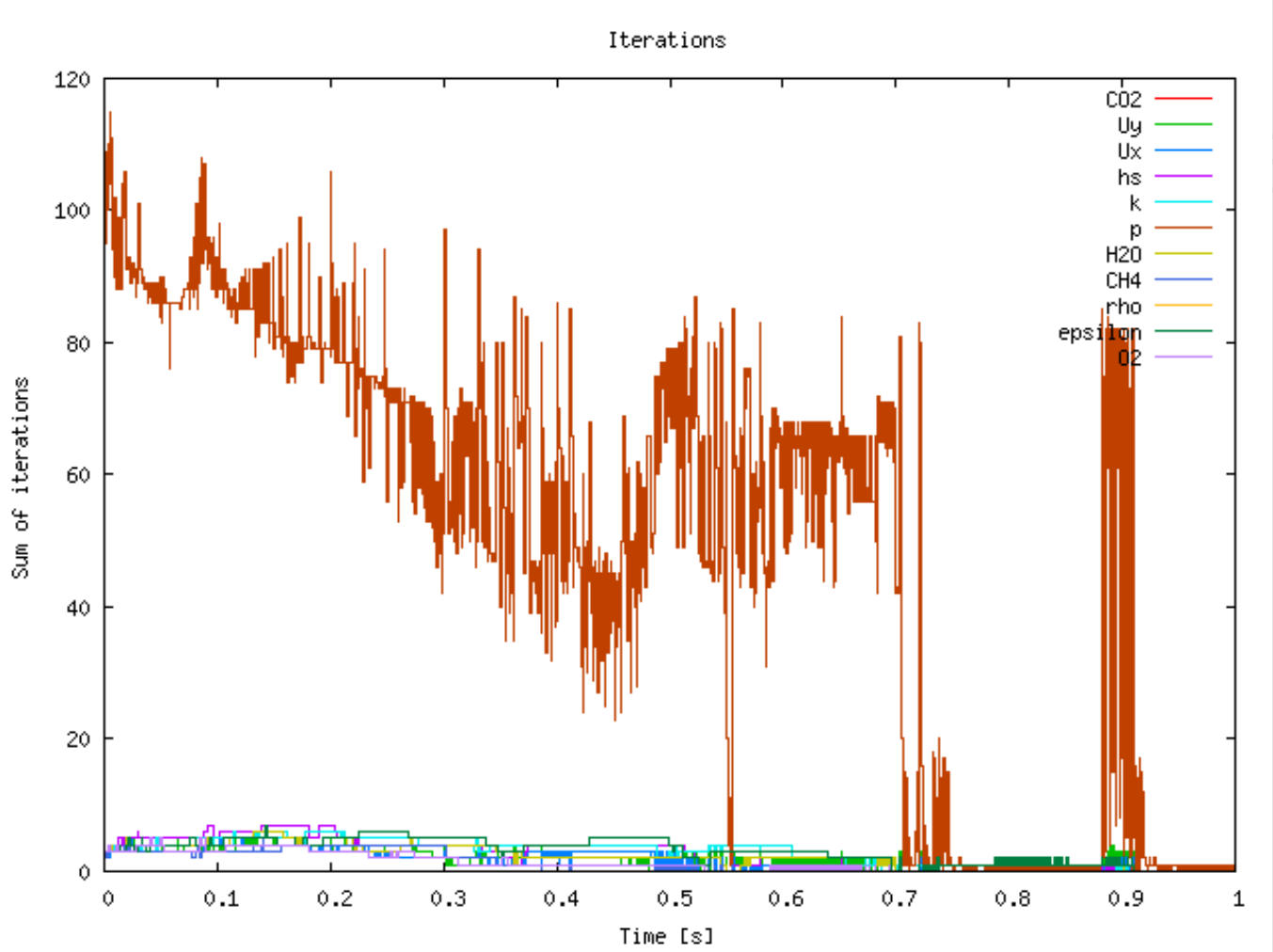
Courant number



Executiong time



Iterations


Iterations
Sum of iterations
Time [s]
0
20
40
60
80
100
120
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
CO2
Uy
Ux
hs
k
p
H2O
CH4
rho
epsilon
O2

Outline

Parallel running

- Decompose cases in one line
  - Create `decomposeParDict`
  - Run `decomposePar`

```
> pyFoamDecompose.py . 2
```

- Runner-utilities know how to handle parallel cases
  - Prepend `mpirun` (or different utilities if configured)
  - Automatically append `-parallel`
  - `--autosense-parallel` checks whether the case is decomposed or not and acts accordingly
    - Automatically gets the correct number of processors

```
> pyFoamRunner.py --auto reactingFoam
```

Instead of

```
> mpirun -n 2 reactingFoam -parallel
```

Fast switching of Foam-versions

- The switch `--foamVersion` allows fast selection of the used (Open)Foam-version. Just for the present command
  - Also possible to select Debug-version

Current case incompatible with OpenFOAM 2.3

```
pyFoamRunner.py --foamVersion=2.3.x --force-debug reactingFoam
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 2.3.x-081e976023a9
Exec   : reactingFoam

<<snip>>

--> FOAM FATAL IO ERROR:
cannot find file

file: /Volumes/Foam/Cases/counterFlameBase/constant/combustionProperties at line 0.

From function regIOobject::readStream()
in file db/regIOobject/regIOobjectRead.C at line 73.

FOAM exiting
```

"Simple" executing

- Sometimes things should be executed in a different OpenFOAM-environment
  - Without log-files
  - Without assuming the Foam calling convention
- `pyFoamExecute.py` does that
- Example: the current Foam-version is experimental and has no working Paraview
  - Fall back to the paraview of the other Version

```
> pyFoamExecute.py --foam=2.3.x paraview
```

Saving away cases

- Archiving cases with all the things needed to reproduce

```
> pyFoamPackCase.py . --tarname=/tmp/baseFlame.tgz --base-name=theFlame
> tar tzf /tmp/baseFlame.tgz
theFlame/constant/RASProperties
theFlame/constant/chemistryProperties
theFlame/constant/g
theFlame/constant/polyMesh/blockMeshDict
theFlame/constant/polyMesh/boundary
theFlame/constant/polyMesh/faces
theFlame/constant/polyMesh/neighbour
theFlame/constant/polyMesh/owner
theFlame/constant/polyMesh/points
theFlame/constant/reactions
theFlame/constant/thermo.compressibleGas
theFlame/constant/thermophysicalProperties
theFlame/constant/turbulenceProperties
theFlame/PyFoamHistory
```

Outline

Temperature extremes

- Hidden in the output of `reactingFoam` is information about the temperature range

```
DILUPBiCG:  Solving for hs, Initial residual = 0.00546778, Final residual =<brk>
     <cont> 8.5528e-08, No Iterations 4
T gas min/max   = 292.977, 983.056
DICPCG:  Solving for p, Initial residual = 0.11851, Final residual = <brk>
     <cont>8.22411e-07, No Iterations 49
```

- It would be cool to plot that as well
  - It is easy . . . but we've got to learn about something complicated . . .

Regular expressions

- Regular expressions are very popular for analyzing textual data (pattern matching)
  - For instance in OpenFOAM for flexible boundary conditions
  - Python comes with a library for analyzing them
  - There are slightly different dialects
    - For instance there are slight differences between the regular expressions of Python and OpenFOAM
    - But in 90% of all cases they behave the same
- The following slide gives a quick glance
  - Usually you won't need much more for PyFoam
- There is a number of cool "regular expression tester" (enter that in Google) applications on the web
  - One example: `http://regex101.com`

Regular expressions in 3 minutes

1. Most characters match only themself
   - For instance 'ab' matches only the string "ab"
2. The dot ('.') matches any character except a newline
   - Pattern 'a..a' matches (among others) "abba", "aBBa", "ax!a"
3. The plus '+' matches the character/pattern before it 1 or more times
   - 'a.+a' matches "aba", "abbbba" but not "aa"
4. '*' is like '+' but allows no match too
   - 'a.*a' matches "aba", "abbbba" and also "aa"
5. Parenthesis '()' group characters together. Patterns are numbered. They receive the number by the opening '('
   - 'a((b+)a)' would match "abba" with group 1 being "bba" and group 2 "bb"
6. To match a special character like '+-().|' prefix it with a '\'
   - To match "(aa)" you've got to write '\(aa\)'
   - Other special characters that occur frequently in OpenFOAM-output are '[]\{\}'

Matching the temperature

- The example string

```
T gas min/max   = 292.977, 983.056
```

- is matched by the regular expression

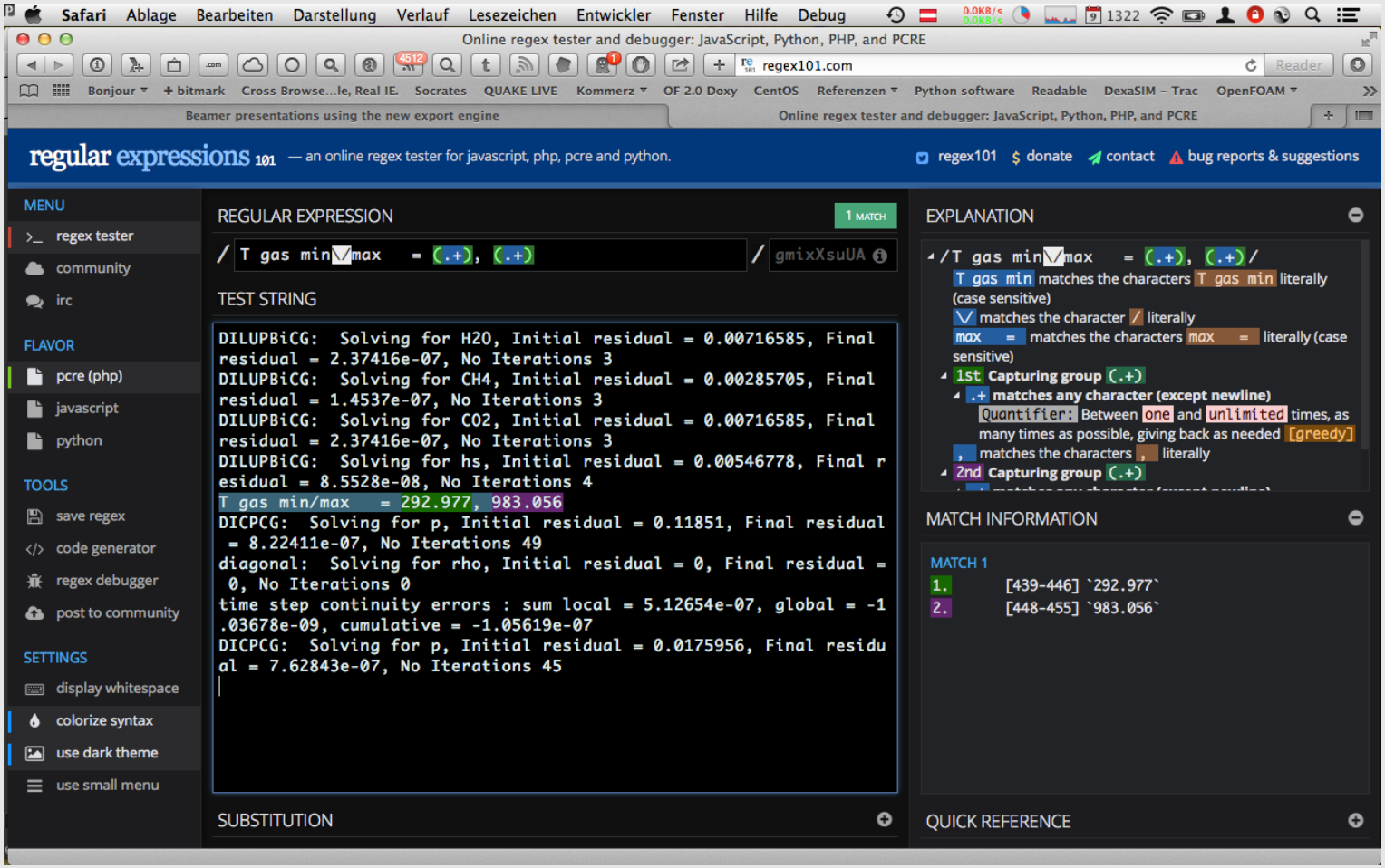
```
T gas min\/max   = (.+), (.+)
```

- with the groups
  1. 292.977
  2. 983.056
- Beware: The / has to be "escaped"
- Beware: Number of spaces has to be correct
- Beware: Simpler expression

```
T gas min\/max   = (.+)
```

- Matches with group 292.977, 983.056
  - Not 292.977 like one would have hoped (regular expressions are "greedy")

Testing the expression



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Matching floating point numbers

- The pattern to match all floating point numbers with regular expressions is quite complex:
  - Matching the sign
  - Exponential notation versus "normal"
- To make life easier PyFoam introduces a shorthand
  - If it finds the string '%f%' in a regular expression it replaces it with the correct regular expression
- This only works in PyFoam. Everywhere else this string will match "%f%"
- In our example:

```
T gas min\/max   = (%f%), (%f%)
```

The customRegexp-file

- If a file customRegexp is found in the case by a Plot-utility it is read
- It is in OpenFOAM-format:
  - a dictionary
  - all entries are dictionaries too
- The name of the entry is used to identify the data (for instance during writing)
- Most frequent entry in the dictionaries are:

expr	This is required. A regular expression that a line must match. All groups (enclosed by '()') are interpreted as data and plotted
theTitle	String with the title of the plot
titles	List of words/strings. The names that the data items will get in the legend

- customRegexp is important enough for PyFoam to be automatically cloned by pyFoamCloneCase.py

First customRegexp

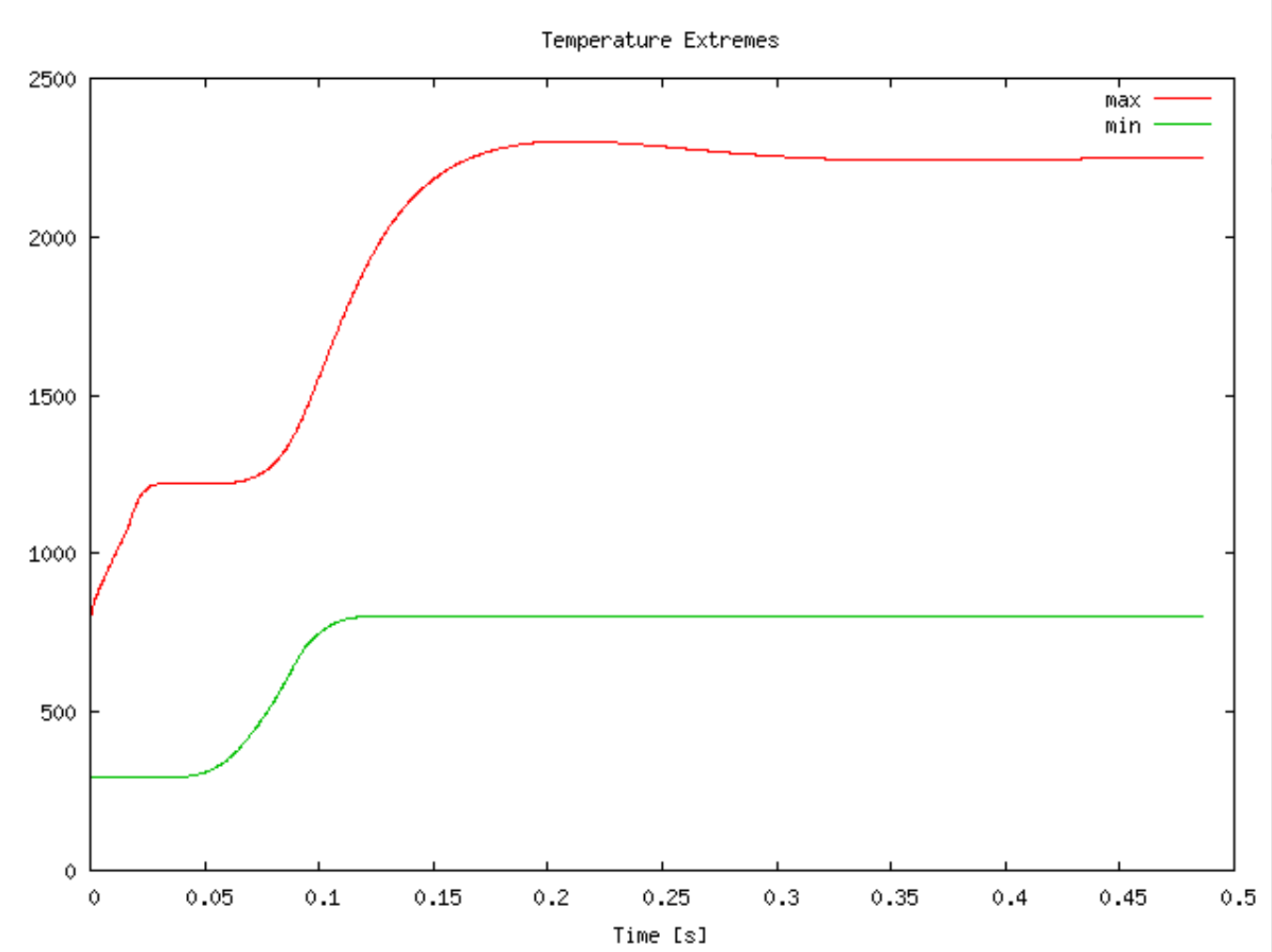
- In the case create with the text-editor of choice a customRegexp

customRegexp

```
temperatureExtremes {
    expr "T␣gas␣min\/max␣␣␣=␣(%f%),␣(%f%)";
    theTitle "Temperature␣Extremes";
    titles (
            min
            max
    );
}
```

- Test by running the watcher (pyFoamWatcher.py PyFoamRunner.reactingFoam.logfile)
  - customRegexp is automatically found and used

Temperature curve



Outline

Outline

Case preparation in OpenFOAM

- Usually done by executing a number of commands
  - `blockMesh`
  - other mesh utilities
  - `setFields` or similar to set up initial conditions
- Tedious if done by hand
  - but easy to automate with scripts
- Usually scripted with `Allrun`-scripts
  - Scripts do a lot of similar work
    - For instance copy `0.org` to `0` to get "clean" initial conditions
  - But are not very robust in terms of error handling
- `PyFoam` offers an alternative

Prepare case with PyFoam

- Create "clean" initial conditions
  - From now on we only edit the files in `0.org`

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

- Run the preparation utility

```
> pyFoamPrepareCase.py .
```

- This does
  1. clears old data from the case
  2. Copies `0.org` to `0`
  3. Runs `blockMesh`
- Could do a number of other things
  - evaluate templates
  - execute scripts
- Details on this in my other talk today

Outline

What are function-objects

- Function objects are "plugins"
- Loaded and initialized at the start of the simulation
- Executed at the end of each timestep
  - And at the end of the run
- OpenFOAM already has a number of functionObjects
  - swak4Foam adds a lot more

Adding function objects to a case

- Usually no function objects are available
- 2 entries in `controlDict`:

  libs List with additional libraries to load
  - function objects in the library are available from then on

  functions Dictionary with function object specification
  - names of the entries are used for output
  - values are sub-dictionaries
    - Mandatory entry `type` determines type of the function object
    - All other entries depend on the function object

Adding simpleFunctionObjects

- oldest part of swak4Foam
  - used to be an independent project

controlDict

```
libs (
    "libsimpleFunctionObjects.so"
);

functions {
    carbonDioxide {
        type banana;
    }
}
```

From now on if a box is for controlDict it means "add this entry to functions"

The old banana-trick

- Getting a full list of function objects is easy
  - Just use banana as type (kiwi would work too)

```
> reactingFoam
<<snip>>
deltaT = 0.000398406


--> FOAM FATAL ERROR:
Unknown function type banana

Valid functions are :

40
(
correctThermo
dynamicFunctionObjectListProxy
executeIfEnvironmentVariable
executeIfExecutableFits
executeIfFunctionObjectPresent
executeIfObjectExists
executeIfOpenFOAMVersionBiggerEqual
executeIfParallelSerial
executeIfStartTime
```

Range of carbon-dioxide

- `volumeMinMax` gets a list of `fields`
  - Calculates the minimum and maximum value of them and outputs it
    - To a file in special directory
    - To the terminal ... sometimes

controlDict

```
functions {
    carbonDioxide {
        type volumeMinMax;
        outputControlMode timeStep;
        outputInterval 1;
        verbose true;
        fields (
            CO2
        );
    }
}
```

Description of the entries

- Entries common to a lot of function objects:
  - **verbose** print results to the terminal.
    - Otherwise they are "only" written to a file (PyFoam can't process them)
  - **outputControlMode** When should output be made.
    - For a list of possible values use *banana-trick*
  - **outputInterval** specific for `timeStep`. How many timesteps between outputs
- Specific entry for this FO:
  - **fields** List of fields

Output of CO_2

This is the extra output we now see when running `reactingFoam`:

```
DILUPBiCG:  Solving for k, Initial residual = 0.005698, Final residual = <brk>
     <cont>1.54878e-07, No Iterations 3
ExecutionTime = 8.1 s  ClockTime = 11 s

Courant Number mean: 0.124187 max: 0.397025 velocity magnitude: 0.362538
deltaT = 0.000275536
 Range of CO2 [ 0 , 0.0501254 ] [0 0 0 0 0 0 0]
Time = 0.0143082

Solving chemistry
diagonal:  Solving for rho, Initial residual = 0, Final residual = 0, No <brk>
     <cont>Iterations 0
```

Exercise: Plot range of CO_2

- Add expressions to the `customRegexp` to plot the range
  - Caution: `[` and `]` have meaning in regular expressions and must be escaped with a `\`

Plotting more information about the species

- Just plotting the range of the species is pretty limited information
  - For all we know maximum could be only in one cell (all other cells are near the minimum)
- We'll extend this

It is time to go to the heart of `swak`

- Expressions are the core functionality of swak4Foam
  - Basically strings which are evaluated at run-time
    - Errors in the expression occur when the expression is evaluated. Not at start-up
- Expression syntax is modeled on the C++/C/Java-syntax for expressions
  - With some exceptions
  - Some additional OpenFOAM-specific things (like & for *dot vector product*)
  - Should be easy to understand even if you're not familiar with these programming languages
- Let me repeat myself: a complete documentation for them is in the *Incomplete Reference Guide*

Adding swak-functionObjects

- Library simpleSwakFunctionObjects combines
  - simpleFunctionObjects: collect data over time
  - with the expressions of swak

controlDict

```
libs (
    "libsimpleFunctionObjects.so"
    "libsimpleSwakFunctionObjects.so"
);
```

The simplest possible expression

- Just one field

controlDict

```
specieCH4 {
    type swakExpression;
    valueType internalField;
    outputControlMode timeStep;
    outputInterval 1;
    expression "CH4";
    accumulations (
        min
        weightedQuantile0.25
        weightedAverage
        weightedQuantile0.75
        max
    );
    verbose true;
}
```

What `swakExpression` does

- Reads an `expression` and evaluates it
- But where?
  - That is what `valueType` says
    - `internalField` means "on the field"
    - another example is `patch` (on a patch specified by `patchName`)
    - For more see the reference guide
- Boils it down to one or more single numbers specified in `accumulations`

Values for `accumulations`

min, max, average, sum these should be pretty self-explanatory

median The value for which 50% of the distribution are smaller than this. More robust alternative to `average`

quantile `quantile0.25` for instance is the value for which 25% of the distribution are smaller than it

range The difference of the quantile of $\frac{1+f}{2}$ and $\frac{1-f}{2}$. For instance `range0.9` gives the range in which 90% of the values are (from the quantile 5% to 95%)

smaller The fraction of the distribution that is smaller than a given value

bigger The inverse of `smaller`

`weighted` accumulations

- Take a *weight* of the values into account
  - For the `internalField` the weights are the cell volume
- Weighted values are usually physically more meaningful
  - Especially for mesh with large differences in cell sizes
  - For `average` a tiny cell would contribute as much as a huge cell
    - This is usally not what we mean with "average temperature" as it depends on the discretization
  - `weightedAverage` does

$$\frac{\sum_i T_i V_i}{\sum_i V_i}$$

  - or as we say in swak for a `internalField`

```
"sum(T*vol())/sum(vol())"
```

Expressions. Some general words

- Expressions are always between ""
- Syntax is oriented on C++/C/Java
  - Strange things like &&, || etc
- "Usual" precedence rules
  - * before + for instance
- For a complete reference see *The incomplete reference guide*

Fields

- When `swak` finds an unknown name in an expression then it assumes that it is a field name
  - Looks for a field of that name in memory
  - Post-processing utilities also look on the disc
- Inserts the value of the field into the equation
  - With correct type (scalar, vector, tensor)
  - and location (cell-center, face-center)
- Expressions may fail because types don't fit
  - "You can't add a scalar on a surface to a vector at the cell-center"
  - Of course the error messages aren't that clear
    - Usually something about "Unexpected XYZ"

Hint: Getting a list of the available fields

- Adding a function object (from `simpleFunctionObjects`)

```
whichFields {
    type listRegisteredObjects;
}
```

- Prints a list of available fields (and non-fields):

```
           Name                             Type Autowrite
============== ================================ =========
            CH4                volScalarField Yes
          CH4_0                       IOobject No
            CO2                       IOobject Yes
          CO2_0                       IOobject No
            H2O                       IOobject Yes
          H2O_0                       IOobject No
             N2                volScalarField Yes
             O2                volScalarField Yes
           O2_0                       IOobject No
 RASProperties                     dictionary No
              S                       IOobject No
              T                volScalarField Yes
```

The other species

- For the other species we would have to copy everything
  - but if `functions` is a dictionary we let `OpenFOAM` do the work

controlDict

```
specieO2 {
    $specieCH4;
    expression "O2";
}
specieH2O {
    $specieCH4;
    expression "H2O";
}
specieCO2 {
    $specieCH4;
    expression "CO2";
}
```

The new output

- Rerunning `reactingFoam` produces more output

```
ExecutionTime = 191.39 s  ClockTime = 194 s

Courant Number mean: 0.150489 max: 0.39903 velocity magnitude: 0.270343
deltaT = 0.000369004
Range of CO2 [ 0 , 0.188742 ] [0 0 0 0 0 0 0]
Expression specieCH4 :  min=1.31196e-05 weightedQuantile0.25=0.0226 <brk>
    <cont>weightedAverage=0.345197 weightedQuantile0.75=0.675 max=0.99792
Expression specieO2 :  min=4.24977e-06 weightedQuantile0.25=0.015 <brk>
    <cont>weightedAverage=0.0986023 weightedQuantile0.75=0.1736 max<brk>
    <cont>=0.229422
Expression specieH2O :  min=0.00042456 weightedQuantile0.25=0.024625 <brk>
    <cont>weightedAverage=0.0771507 weightedQuantile0.75=0.129923 max<brk>
    <cont>=0.154522
Expression specieCO2 :  min=0.000518584 weightedQuantile0.25=0.0299 <brk>
    <cont>weightedAverage=0.0942367 weightedQuantile0.75=0.1587 max<brk>
    <cont>=0.188742
Time = 0.779336
```

Plotting the species

- With our current knowledge we'd need four expressions in `customRegexp`
  - This would be tedious
  - We'd get four different plot windows
  - With more complicated chemical reactions the problem gets worse
- But as the outputs look quite similar the regular expressions offer a solution

Dynamic Plotting

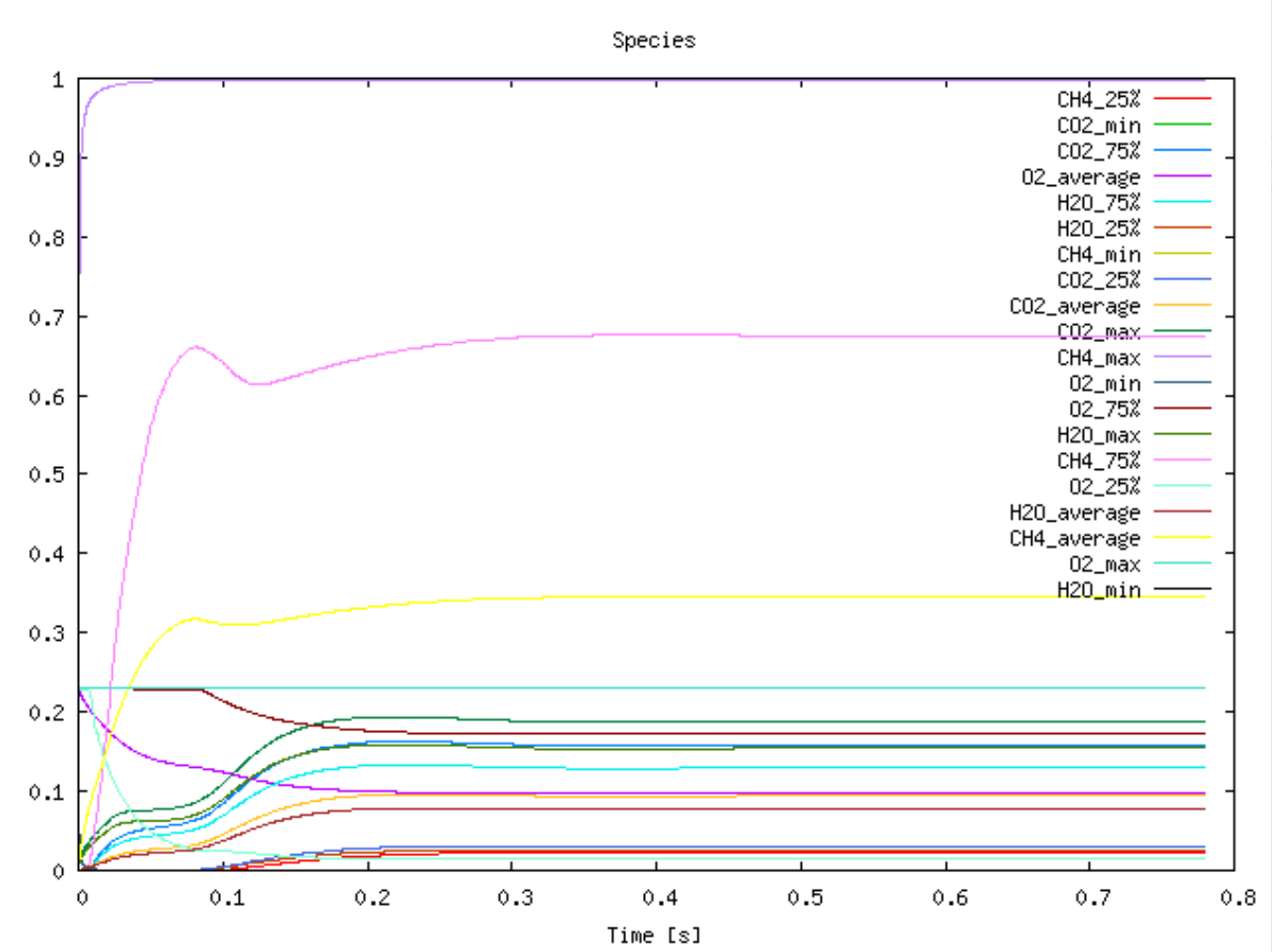
- Add to the dictionary in `customRegexp` an entry `type`
  - Value: `dynamic`
- Now PyFoam needs another entry: `idNr`
  - This is the index of the matching group that holds a name
    - Remember: groups are numbered by the occurrence of the (
    - Counting starts with 1
- For each name a different data-set is plotted
  - But all in the same graph
  - name is added to the `titles`

Entry for species-plotting

customRegexp

```
species {
        theTitle "Species";
        expr "Expression␣specie(.+)␣:␣␣min=(%f%)␣weightedQuantile0.25=(%f%)<brk>
             <cont>␣weightedAverage=(%f%)␣weightedQuantile0.75=(%f%)␣max=(%f<brk>
             <cont>%)";
        type dynamic;
        idNr 1;
        titles (
                min
                "25%"
                average
                "75%"
                max
        );
}
```

Too much information



Getting the location of the maximum

- `maxPosition(expr)` means: "find the position of the maximum value of `expr`"
  - Any `accumulation` is fine as this is a *uniform value*
    - Except `sum`

```
whereMaxT {
    $specieCH4;
    expression "maxPosition(T)";
        accumulations (
            average
        );
}
```

Produces this output

```
Expression whereMaxT :  average=(0.0121 -0.00975 -6.16298e-2
```

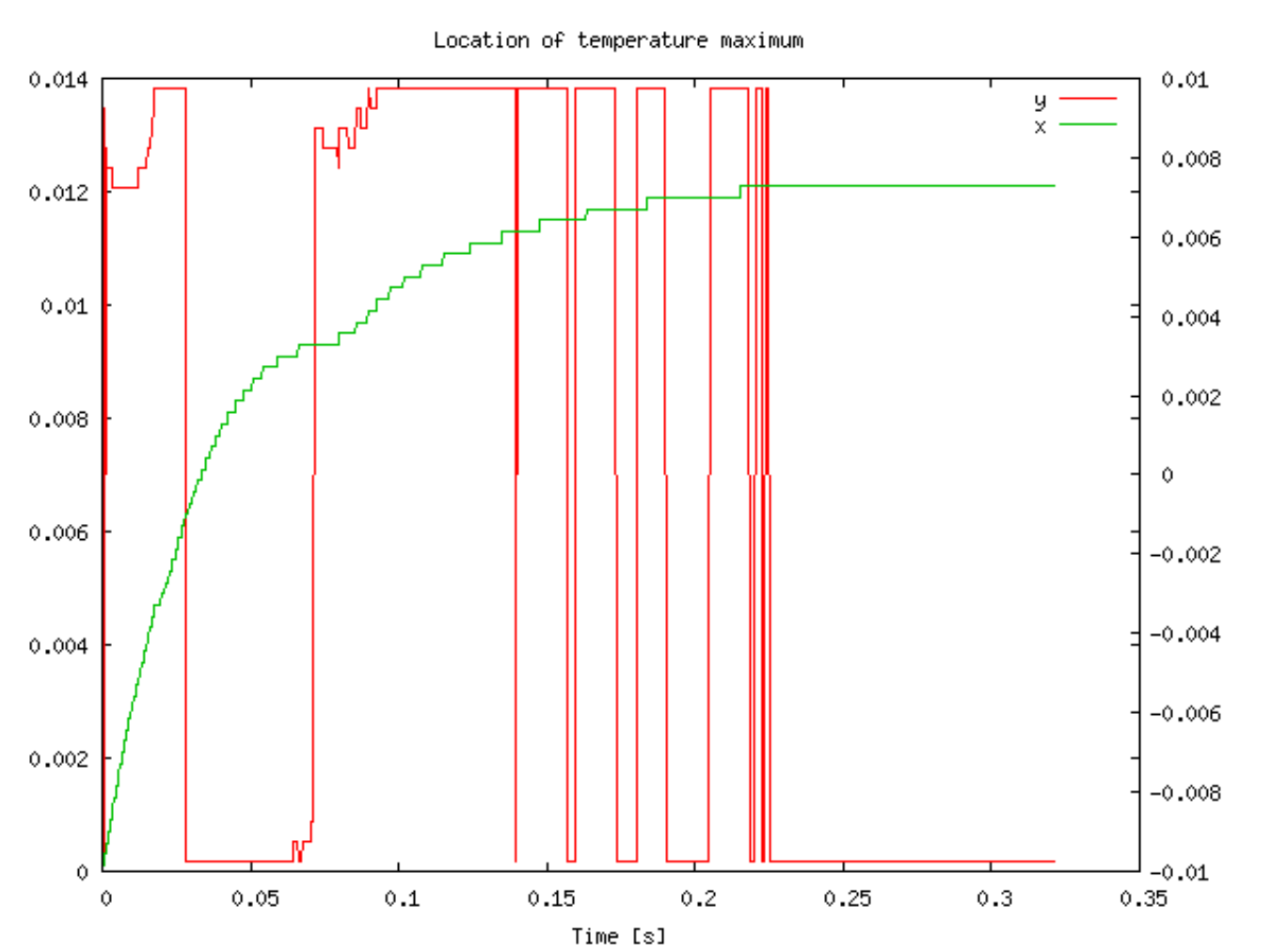
Plotting the location of the maximum temperature

customRegexp

```
maxTLocation {
             theTitle "Location␣of␣temperature␣maximum";
             expr "Expression␣whereMaxT␣:␣␣average=\((%f%)␣(%f%)␣%f%\)";
             titles (
                    x
                    y
             );
             alternateAxis ( y );
}
```

- `alternateAxis` means "put values in this list on the right axis"
  - Useful if values of different scale should be in one scale

The maximum moves to the right



Setting GnuPlot-options

- A number of options usually set with the set-keyword in Gnuplot can be set in the dicts (for details see the GnuPlot-documentation)

logscale	use a logarithmic scale for the y-axis
ylabel	label on the y-axis
y2label	same on the alternate axis
xlabel	Override the usual Time [s]
with	use something different than lines (points, steps)
gnuplotCommands	A list of strings with GnuPlot-commands that are executed during execution

Addition to customRegexp to get x-axis for y

```
ylabel "x";
y2label "y";
gnuplotCommands (
    "set x2zeroaxis linewidth 3"
);
```

Calculating the stoechiometric ratio

- The question is "Are there enough O_2-molecules in the cell to burn all CH_4"
  - Or are there too many
- The fractions that OpenFOAM uses are mass-fractions
  - We need mole-fractions

Variables

- Optional entry is `variables`
  - List of strings
  - Each string is of the form `<name> = <expression>;`
    - The semicolon is important
    - Means: Evaluate the `expression` and put it into a variable named `name`
  - The defined variables can be used in
    - subsequent variable-specifications
    - `expression`
  - Values are forgotten afterwards

expressionField

The library swakFunctionObjects adds function objects that "only" use expressions

- No timeline-recording etc

controlDict

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

- expressionField calculates expression and puts it into a field named fieldName

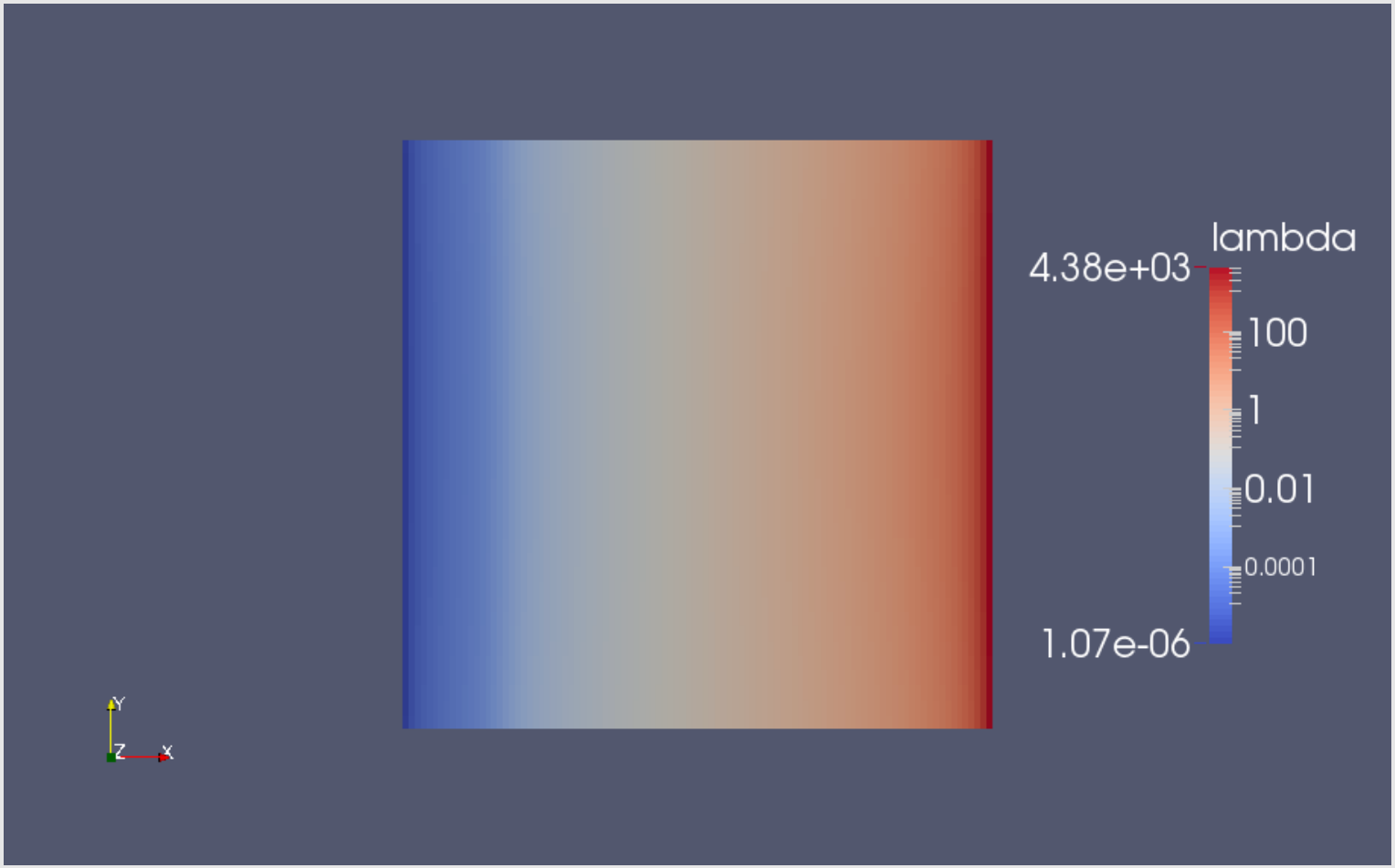
Calculating λ

controlDict

- The molar masses (12.0107 etc) are hard-coded here

```
stoechiometric {
    type expressionField;
    fieldName lambda;
    autowrite true;
    outputControl timeStep;
    outputInterval 1;
    variables (
        "MCH4=12.0107+4*1.00794;"
        "MO2=2*15.9994;"
    );
    expression "(0.5*O2/MO2)/(max(CH4,1e-10)/MCH4)";
}
```

λ



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Exercises: Getting numbers

- Question: what could `autowrite` mean?
- Plot percentage of space where $\lambda > 1$
  - Add `swakExpression`
  - Extend `customRegexp`

What goes in must come out

- We want to check whether the mass-flows add up
- They should
  - After the initial phase
- Info: by convention in OpenFOAM `phi` is the mass flow on one face
  - Summing it up gives the total mass flow on a patch
- `patchExpression` calculates expression on patches
  - Patches specified in a list
    - Elements can be regular-expression

First the mass-flows on the patches

controlDict

```
massFlows {
    type patchExpression;
    valueType internalField;
    outputControlMode timeStep;
    outputInterval 1;
    patches (
        ".*"
    );
    expression "phi";
    accumulations (
        sum
    );
    verbose true;
}
```

Picking up the mass-flows

That should be easy by now

customRegexp

```
massFlows {
          theTitle "Mass flows";
          expr "Expression massFlows on (.+):  sum=(%f%)";
          type dynamic;
          idNr 1;
          titles (
                 sum
          );
}
```

Getting values from somewhere else

- `swak` allows variables calculated "somewhere else"
- General notation is

```
<name>{<type>'<ename>}=<expression>;
```

- Meaning: Calculate `expression` on entity `ename` of type `type` and put result into `name`
  - Limitation: the result of `expression` must be uniform
    - For instance a `sum`, `min`, `max`, ..
- If only `ename` is given, then it is assumed that `type` is `patch`
- There is an extension to the syntax for multi-region cases
  - Look it up in the reference

`#include` for variable lists

- Entry of the form

"`#<name>;`"

means "Get variable list from `name` and insert it here"

- This allows splitting and reusing variable lists

One patch sums up

controlDict

```
massFlowSum {
    type swakExpression;
    valueType patch;
    patchName outlet;
    outputControlMode timeStep;
    outputInterval 1;
    verbose true;
    patchMF (
        "fuelMF{fuel}=sum(phi);"
        "airMF{patch'air}=sum(phi);"
    );
    variables (
        "#patchMF;"
    );
    expression "sum(phi)+fuelMF+airMF";
    accumulations (
        average
    );
}
```

Adding data to another graph

- We'd like to get all data into one graph, but
  - `massFlowSum` prints to a separate line
  - Doesn't fit the `massFlows` in `customRegexp` (`average` instead of `sum`)
- Putting it into the other graph:
  - Set `type` to `slave`
  - An additional entry `master` is needed for the graph that does the actual plotting
- No additional graphs window opened
- More than one `slave` plot can be added to a `master`
  - The latest `PyFoam` has a type `dynamicslave` that allows mixing the capabilities of `dynamic` and `slave`

Send sum to the other graph

customRegexp

```
massFlowSum {
            type slave;
            master massFlows;
            expr "Expression␣massFlowSum␣:␣␣average=(%f%)";
            titles (
                   sum
            );
}
```

How big is the mass flow deficit?

- Compare to the amount of mass in the simulation

controlDict

```
relativeDeficit {
    $massFlowSum;
    variables (
        "#patchMF;"
        "sumMass{internalField'}=sum(vol()*rho);"
    );
    expression "(sum(phi)+fuelMF+airMF)/sumMass";
}
```

Append the deficit to the plot

customRegexp

```
relativeDeficit {
            type slave;
            master massFlows;
            expr "Expression relativeDeficit : average=(%f%)";
            titles (
                   deficit
            );
}
```

The scale of the relative deficit is quite different. So we want it on the `alternateAxis`

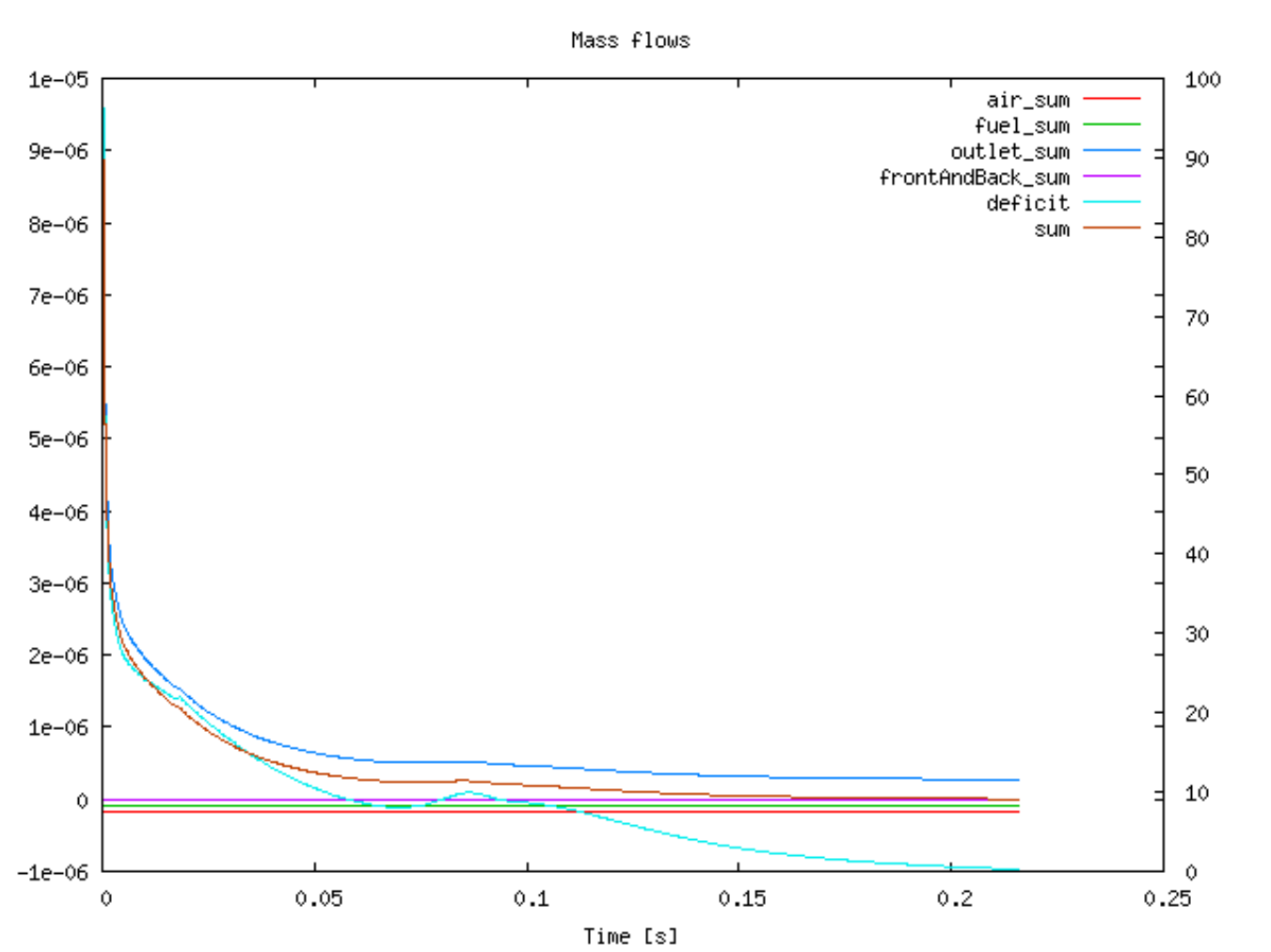
Changed entry (alternateAxis added)

customRegexp

```
massFlows {
          theTitle "Mass flows";
          expr "Expression massFlows on (.+):  sum=(%f%)";
          type dynamic;
          idNr 1;
          titles (
                  sum
          );
          alternateAxis (
                  deficit
          );
}
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Mass-flows plotted



Outline

Sometimes numbers speak louder than work

- Sometimes you don't need fancy graphics. Just basic statistics

```
> fieldReport -time 0: CO2 -csvName CO2Development
<<snip>>
Time = 1

 Reading Field CO2 of type volScalarField

Internal field:
Size | Weight Sum                  4000 |          8e-08
Range (min-max)           0.000518583 |       0.188743
Average | weighted          0.0942368 |      0.0942368
Sum | weighted                376.947 |    7.53895e-09
Median | weighted               0.0932 |          0.0932

End
```

- See `-h` option for more . . . options

Fancy numbers

- For more elaborate post-processing there is `funkyDoCalc`
  - Basically "Execute `swakExpressions` on data on disc"
- User specifies a file
  - Dictionary with sub-dictionaries
  - Format like `swakExpression` but without function-object-specific stuff (`type`, `output*`)
- Data is printed to screen
  - Like `fieldReport` there is the option to write a CSV-file

Outline

Why plugins

- There are functions that are of interest for some
  - But not for all
- Adding them to the regular grammar would
  - Complicate the grammar
  - Would bloat the basic swak library
  - be problematic because the solver may not have the required fields
    - Turbulence for instance
  - Would not solve the problem something very special is required
    - For instance a wrapper for a in-house chemistry solver

Function plugins in `swak`

- Function plugins are special libraries
  - Loaded via `libs`-list
- They register new functions in a table
- The functions can be used like built-in function
  - Possible argument types (depend on the function) are
    - Expressions (but a very specific type -for instance vector- is required)
    - Numbers
    - Words
- The first time the parser is called it prints a list of all available functions
  - With parameter descriptions

Getting the reaction rates

- To see how fast each species is converted we need the reaction rates of the chemistry model
  - There is a special plugin for information about the chemistry

controlDict

```
libs (
    "libsimpleFunctionObjects.so"
    "libsimpleSwakFunctionObjects.so"
    "libswakFunctionObjects.so"
    "libswakChemistryModelFunctionPlugin.so"
);
```

Listing of the added functions

- First time an expression is evaluated swak prints a list of available plugin-functions is printed
  - No need to search for non-existing documentation

```
"Loaded plugin functions for 'FieldValueExpressionDriver':"
  psiChem_RR:
    "volScalarField psiChem_RR(primitive/word speciesName)"
  psiChem_RRError:
    "volScalarField psiChem_RRError()"
  psiChem_RRSumPositive:
    "volScalarField psiChem_RRSumPositive()"
  psiChem_Sh:
    "volScalarField psiChem_Sh()"
  psiChem_dQ:
    "volScalarField psiChem_dQ()"
  psiChem_deltaTChem:
    "volScalarField psiChem_deltaTChem()"
  psiChem_tc:
    "volScalarField psiChem_tc()"
  psiChem_updateChemistry:
    "volScalarField psiChem_updateChemistry(primitive/scalar timestep)"
```

Description of plugin functions

- The description of the plugin-functions are of this format:

  First line name of the function (for instance `psiChem_RR`)

  Second line expected parameters and results. Format is similar to a C++/Java function-specification:

  1. Result type (for instance =volScalarField)
  2. Function name again (same as above)
  3. in "`()`" a list of parameters with type (for instance `primitive/word`) and a descriptive name (here `speciesName`)

- Argument types can be
  - Fields
  - `primitives`: things like single numbers, names

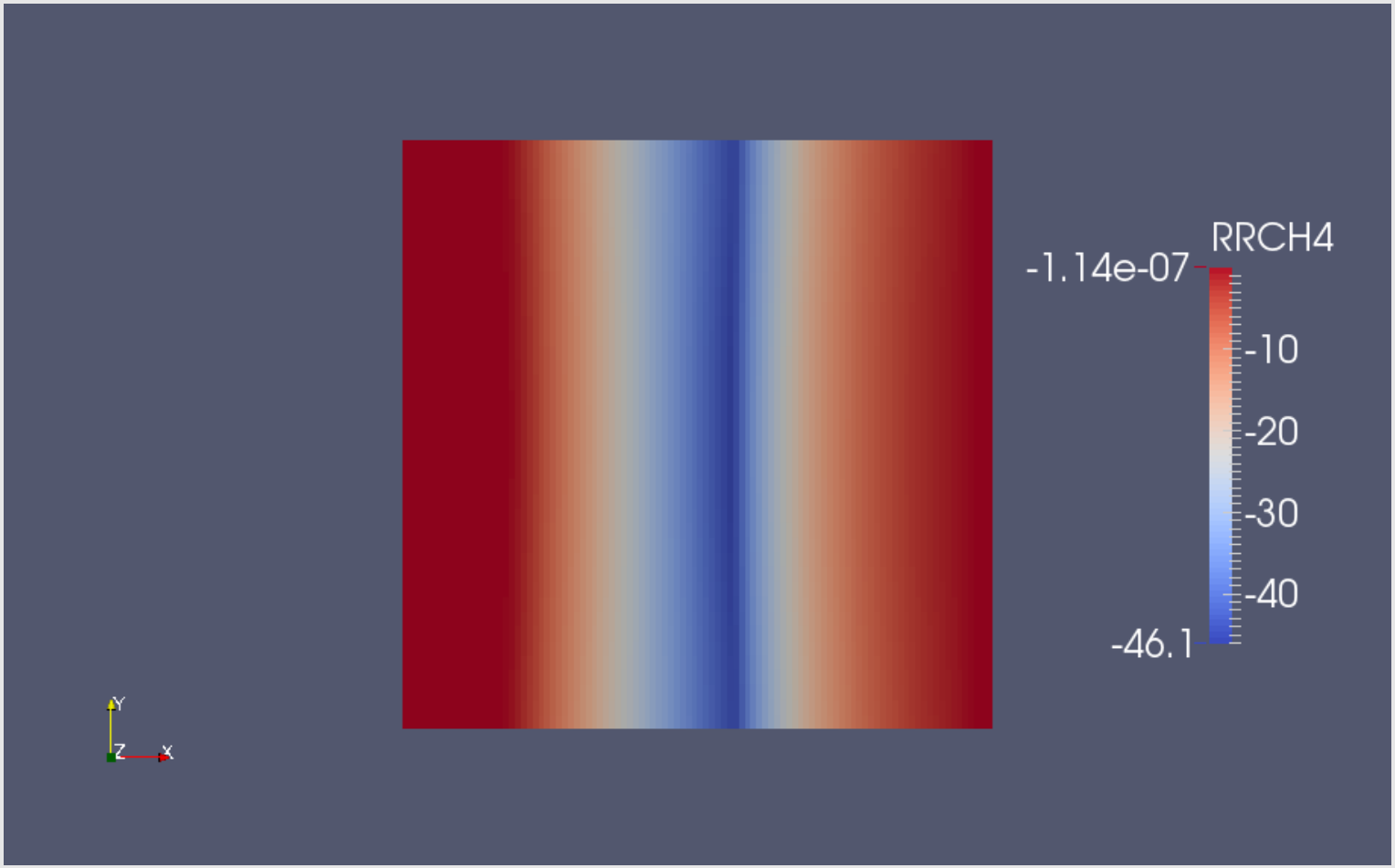
Reaction rate of CH_4

- Reaction rates are returned by `psiChem_RR`
  - Not calculated! The last values are used
  - Argument is a `word`: the species name

controlDict

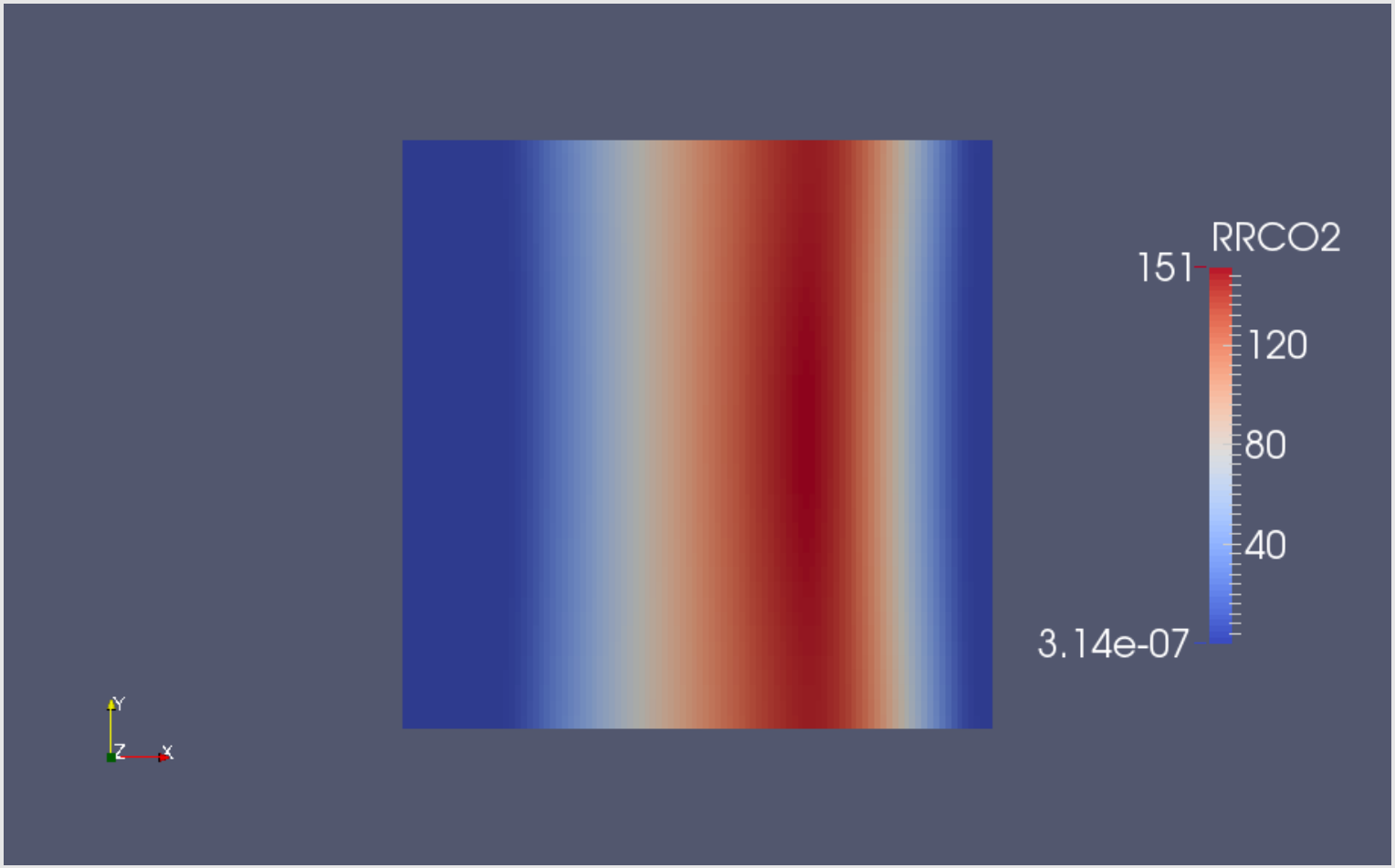
```
reactionRateCH4 {
    type expressionField;
    fieldName RRCH4;
    outputControl outputTime;
    autowrite true;
    expression "psiChem_RR(CH4)";
}
```

Reaction rate CH_4



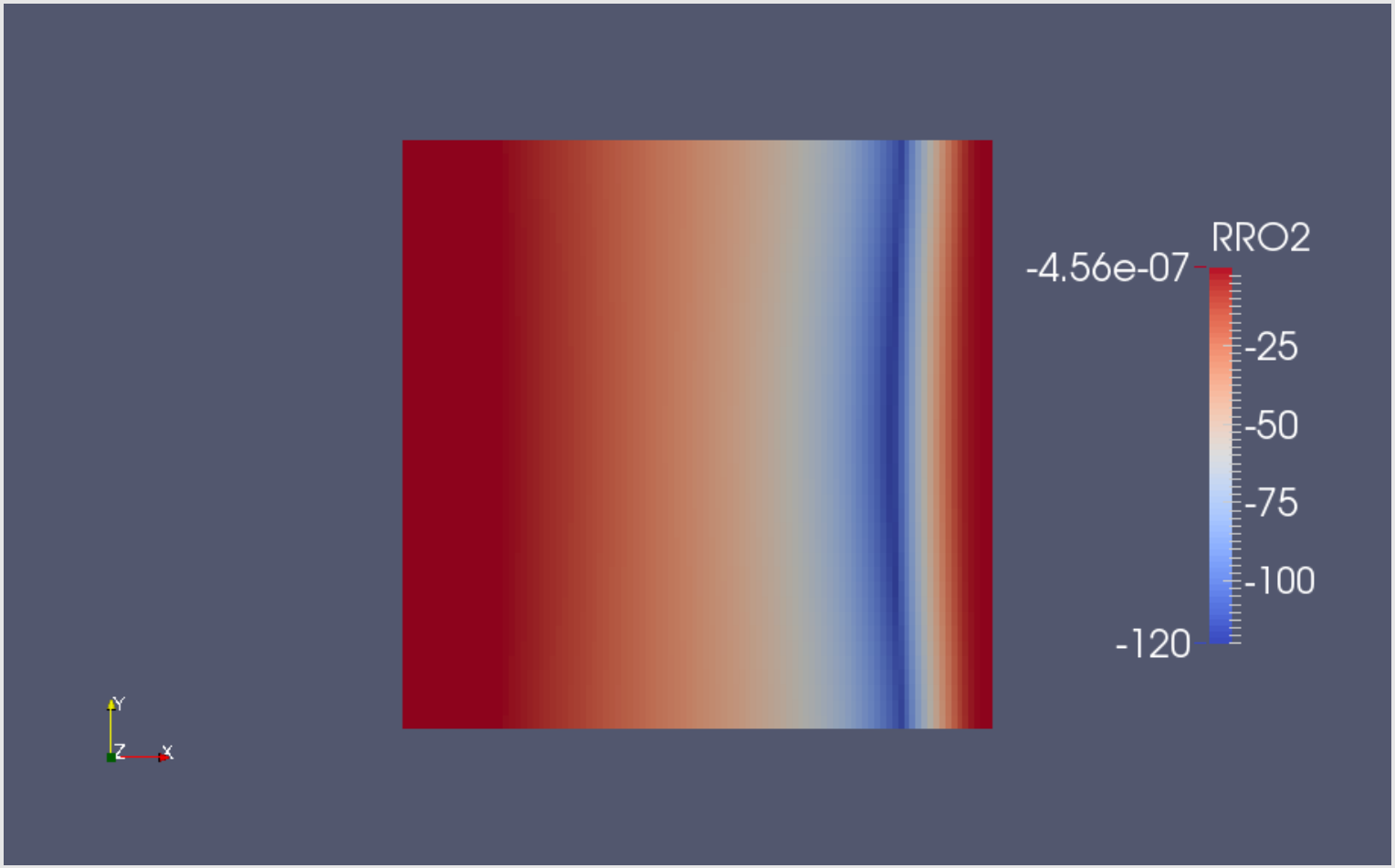
Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Reaction rate CO_2



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Reaction rate O_2



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Exercise

- Add expressions for other reaction rates
- Add up reaction rates for CH_4 and O_2

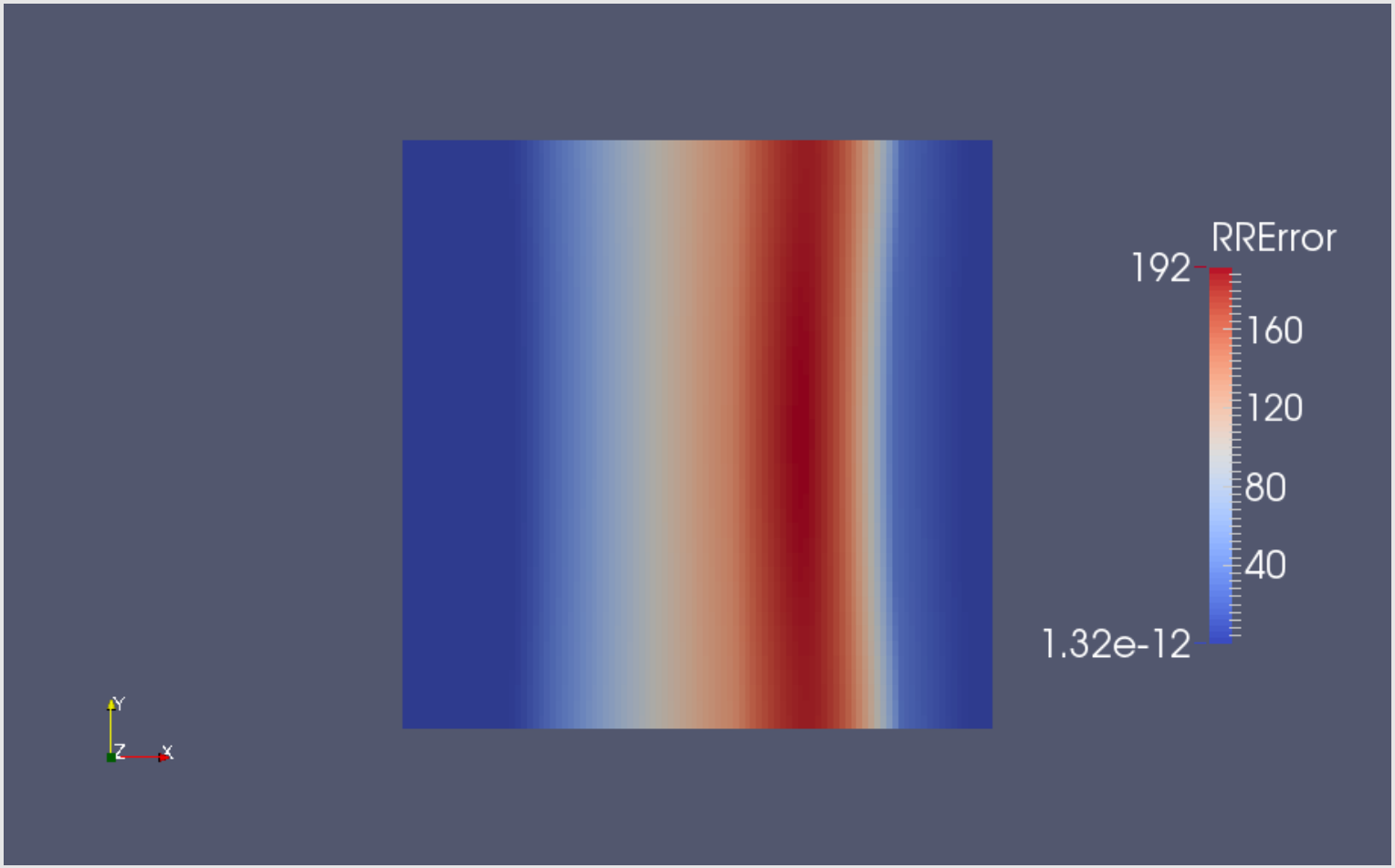
Additional information about the chemistry

- Error of the chemistry (sum of rates)
- t_c reported by the chemistry model

controlDict

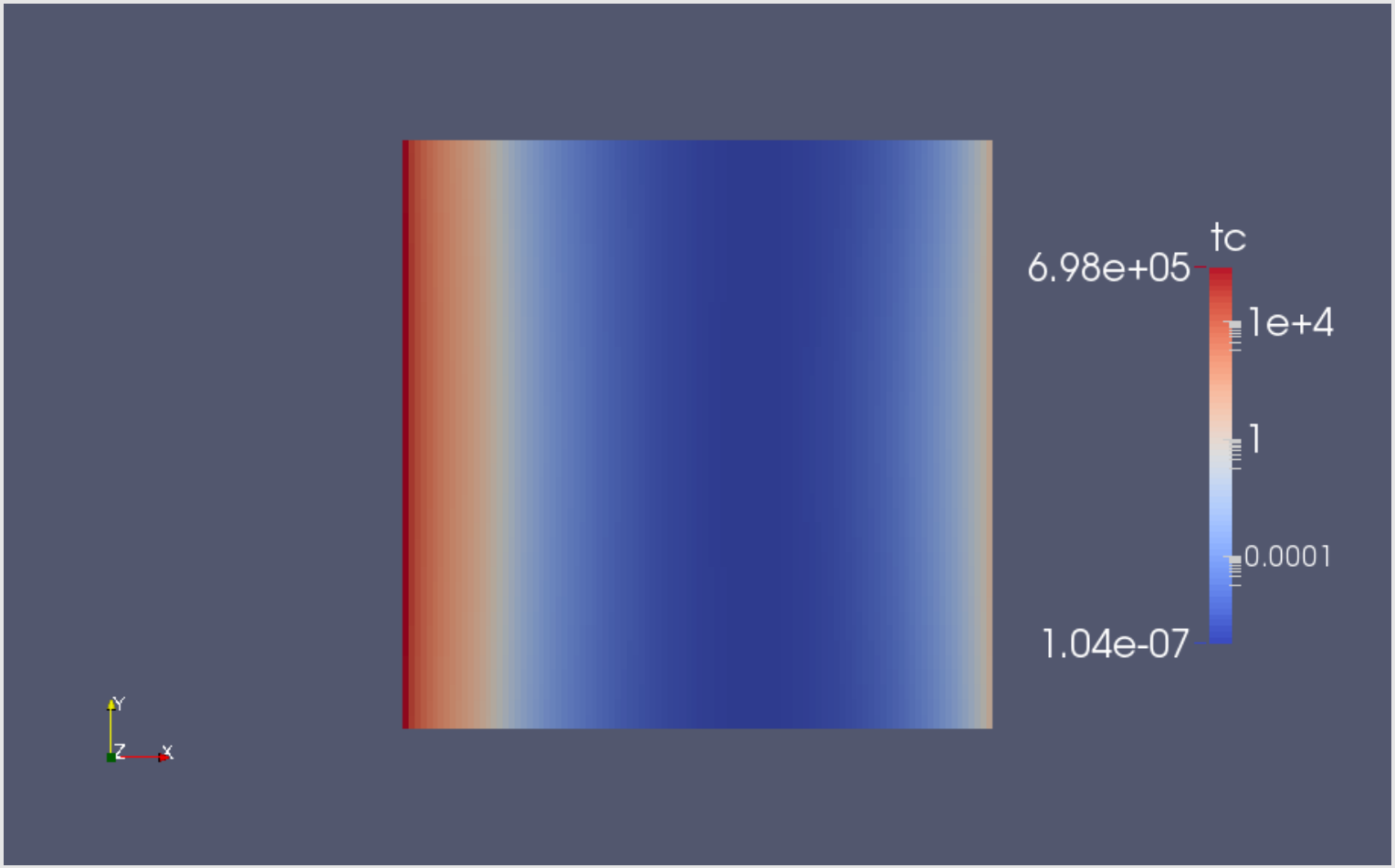
```
reactionRateError {
    $reactionRateCH4;
    fieldName RRError;
    expression "psiChem_RRError()";
}
reactionTime {
    $reactionRateCH4;
    fieldName tc;
    expression "psiChem_tc()";
}
```

Sum of reaction rates



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Time of chemistry



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Problem with the time-discretization

- Chemistry solver gets the current chemical composition Y_{old}
- Is asked to integrate for the Δt of the flow solver
  - For integration smaller time-steps are used
- Records the new composition Y_{new}
- Calculates reaction rate used in the flow solver as

$$RR = \frac{Y_{new} - Y_{old}}{\Delta t}$$

- This is "only" an average of the real reaction rates
  - May be missleading if reaction fast compared to Δt
- We want to find out: Is this a problem here?

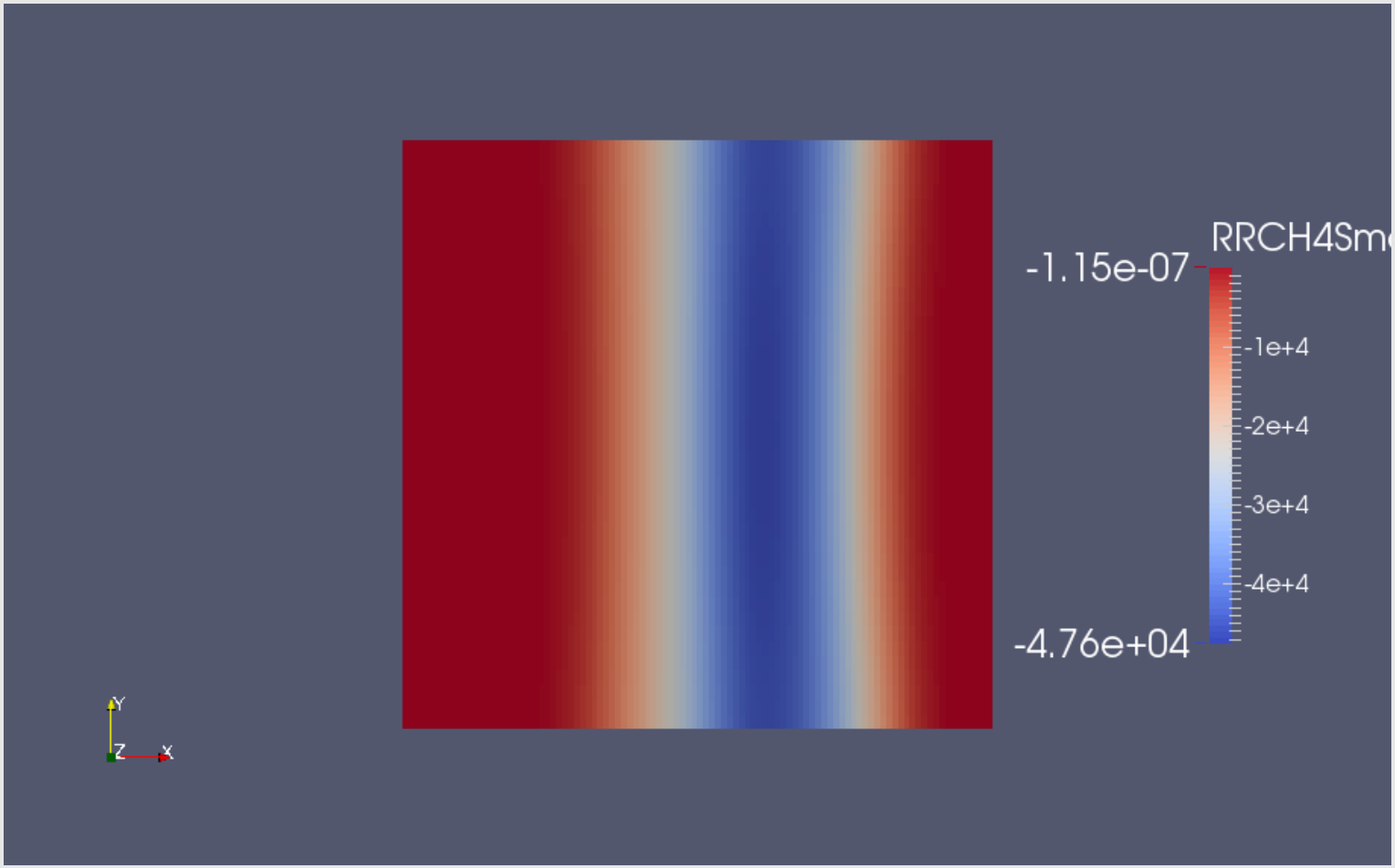
Calculating rate for smaller time-step

- Reaction rate for a smaller timestep is "nearer" to the real reaction rate
- Function `psiChem_updateChemistry` triggers a recalculation of the chemistry
  - Argument is Δt
  - Returns 0
  - Subsequent calls to `psiChe_` use the new reaction rates

controlDict

```
reactionRateCH4Small {
    $reactionRateCH4;
    fieldName RRCH4Small;
    expression "psiChem_updateChemistry(0.0000001)+psiChem_RR(CH4)";
}
```

Different rate for CH_4



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Exercise

- Write chemistry error for small timesteps
- Try reproducing the chemistry error by adding reaction rates

Outline

Changing the case

- Until now we did not change the results
  - But this will change

Outline

groovyBC

- This is probably the most popular part of swak
- It allows setting boundary conditions by writing swak-expressions

controlDict

```
libs (
    "libsimpleFunctionObjects.so"
    "libsimpleSwakFunctionObjects.so"
    "libswakFunctionObjects.so"
    "libswakChemistryModelFunctionPlugin.so"
    "libgroovyBC.so"
);
```

Using groovyBC

- Set the `type` of the boundary condition to `groovyBC`
- The three entries used are:

  valueExpression expression to be used as the boundary value
  gradientExpression the gradient (optional)
  fractionExpression If 1 then `valueExpression` is used. If 0 the `gradientExpression` (optional)
- These expressions are evaluated at every time-step
- Pro-tip: It is good practice to set `value` to a sensible value as the expressions are not evaluated at startup

Velocity distribution for fuel

0.org/U (not 0/U)

```
fuel
{
    type            groovyBC;
    value           $internalField;
    variables (
        "minY=min(pts().y);"
        "maxY=max(pts().y);"
        "middleY=0.5*(minY+maxY);"
        "widthY=maxY-minY;"
    );
    valueExpression "-(mag(pos().y-middleY)<0.25*widthY␣?␣0.15␣:␣0.05)*<brk>
        <cont>normal()";
}
air
{
    $fuel;
}
```

What is new here

min, max that should be self-explanatory

pos Cell centers on the patch

pts Places of the points on the patch

.y Get the *y*-component of a vector

normal Unit vector normal to the faces

? : A expression of the form "`a ? b : c`" means "if the logical expression 'a' is `true` use result 'b', otherwise result 'c' "

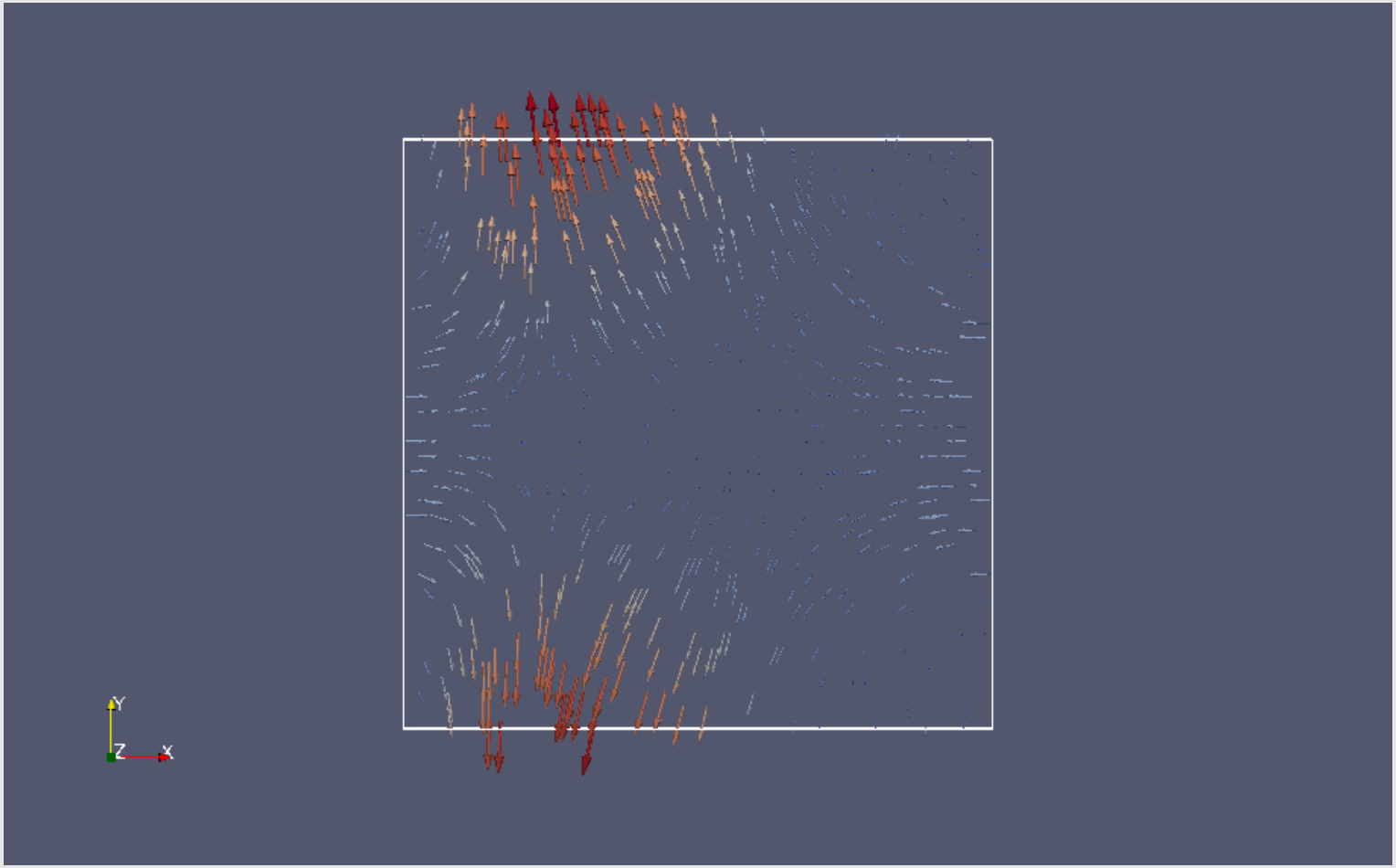
Running

- Important: First do

```
> pyFoamPrepareCase.py .
```

- This copies `0.org` to `0`
- Then run `reactingFoam`

Changed velocity



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Exercises

- `cos()` is the usual trigonometric function
  - Try building a "smoother" inlet-profile
- `time()` is the current simualtion time
  - Try building a pulsating inlet condition

Outline

Adding feedback

- We don't want the simulation to become "too hot"
  - One way to achieve this is letting less oxygen in
- Usually the mass fraction of oxygen at inlet `air` is 0.27
  - If the average temperature in the "middle" is $> 1800K$ the mass fraction is reduced to 0.1
- Definition of "middle":
  - The cells in a radius of $5mm$ around the center
  - We have to specify a `cellSet` with the name `testRange`

topoSources in swak

- Utilities like setSet or topoSet use sub-classes of topoSource
  - swak4Foam specifies such sub-classes
    - Allows using of expressions

controlDict

```
libs (
    "libsimpleFunctionObjects.so"
    "libsimpleSwakFunctionObjects.so"
    "libswakFunctionObjects.so"
    "libswakChemistryModelFunctionPlugin.so"
    "libgroovyBC.so"
    "libswakTopoSources.so"
);
```

Adding a script to create the mesh

- If `pyFoamPrepareCase.py` finds a script `meshCreate.sh` in the case directory it executes this instead of `blockMesh`
  - Used for non-standard meshes
  - Must include a call to `blockMesh` (if that is used)

meshCreate.sh

```
#! /usr/bin/env bash

rm -rf constant/polyMesh/sets
blockMesh
setSet -batch system/setSet.middleCircle
```

Creating the cellSet

- setSet creates cell, face and point-sets
  - In OpenFOAM 2.3 we use topoSet
    - That can also create zones

system/setSet.middleCircle

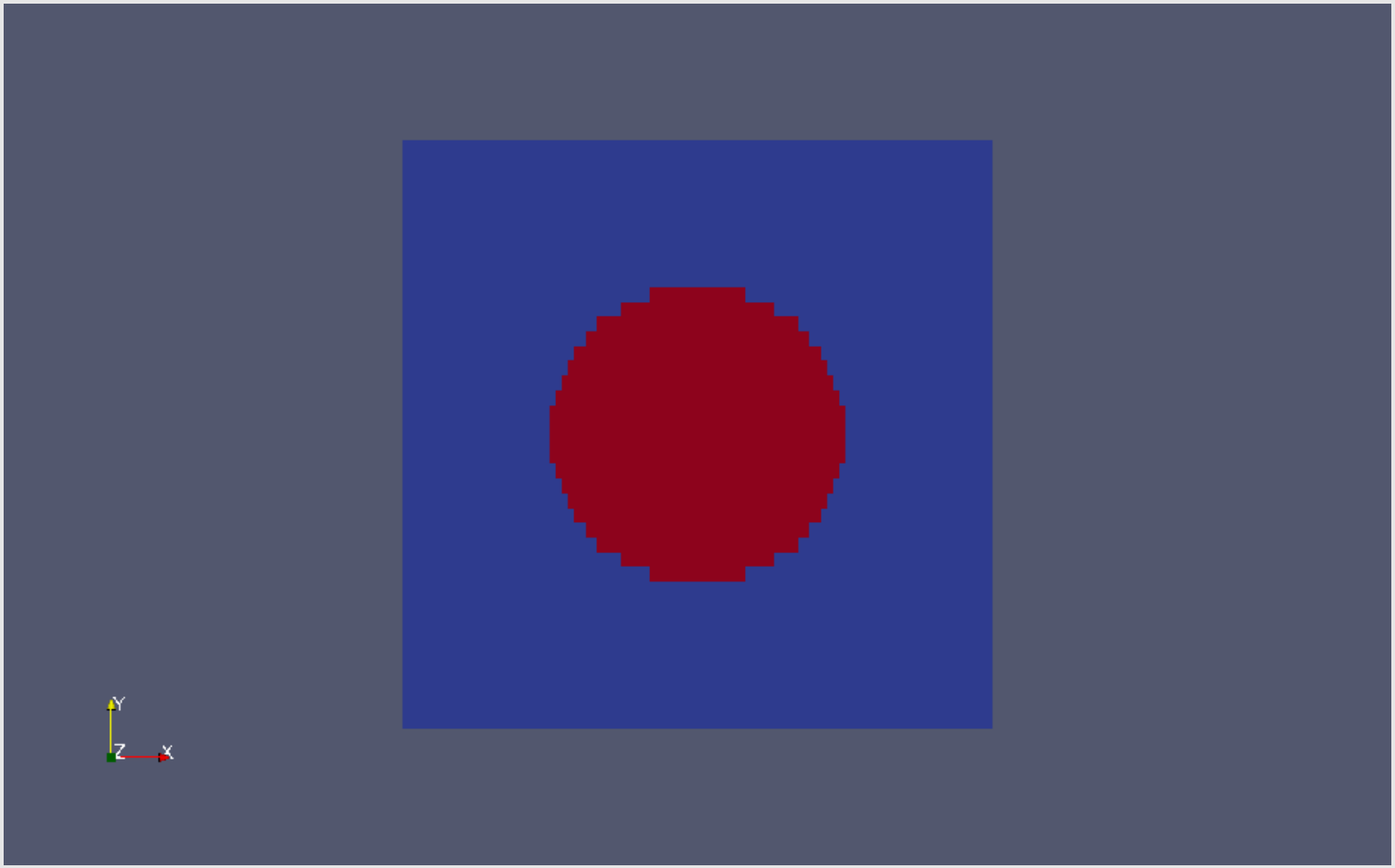
```
cellSet testRange new expressionToCell "mag(pos()-vector(0.01,0,0))<0.005"
```

Preparing the case

- Execute `pyFoamPrepareCase.sh` to create the cell-set

```
> pyFoamPrepareCase.py .
  <<snip>>
> ls -l constant/polyMesh/sets/testRange
-rw-r--r--  1 bgschaid  staff  5610 Jun 11 18:29 constant/polyMesh/sets/<brk>
     <cont>testRange
```

Red cells are part of `testRange`



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Boundary condition for "oxygen-sensor"

- Volume weighted average of `T` on `cellSet` with the name `testRange`

0.org/O2

```
air
{
    type            groovyBC;
    value           uniform 0.23;
    variables (
        "highVal=0.23;"
        "lowVal=0.1;"
        "threshold=1800;"
        "targetT{cellSet'testRange}=sum(T*vol())/sum(vol());"
    );
    valueExpression "targetT<threshold ? highVal : lowVal";
}
```

Nitrogren

- Make sure that the sum of fractions is 1

0.org/N2

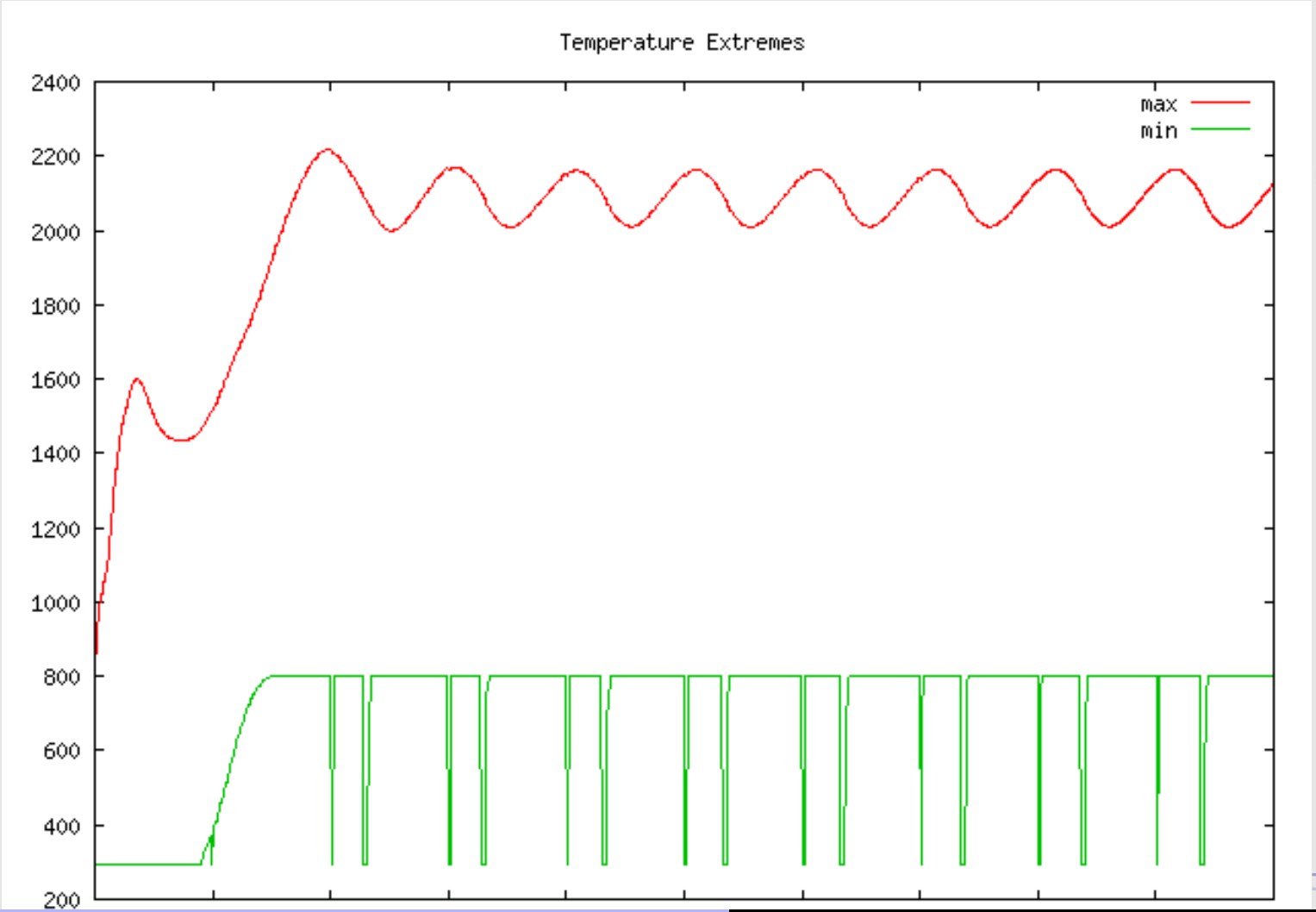
```
air
{
    type            groovyBC;
    value           uniform 0.77;
    valueExpression "1-O2";
}
```

Case setup and running

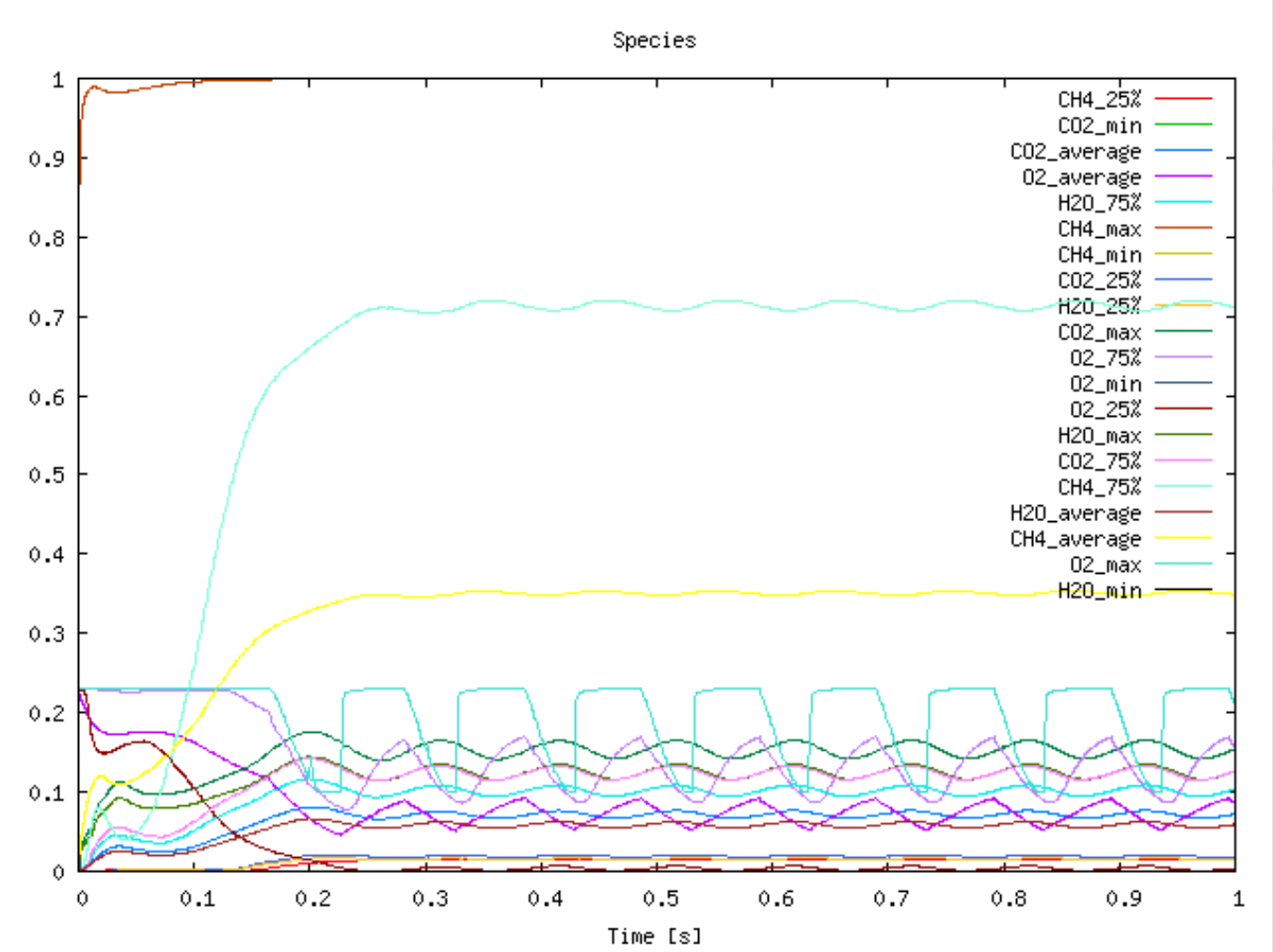
- We don't need to create the mesh again
  - The `cellSet` is already created
  - `pyFoamPrepareCase.py` has an option for that
    - Handy for big cases

```
> pyFoamPrepareCase.py . --no-mesh
<<snip>>
> pyFoamRunner.py  --clear --progress  reactingFoam
Clearing out old timesteps ....
 t =  0.0276251
```

Temperatures with feedback



Species with feedback



Exercises

- Add `swakExpression` to 'measure' temperatures in `cellSet`
  - Whichever `accumulations` seem suitable
  - Add as `slave`-plot to the other temperatures
- Change the trigger
  - $T_{max} > 1800$ in zone
  - Weighted with inverse difference to center (the nearer to the center the more influence a cell has)
- Make transition less sharp
  - Decrease O_2 linearly between $1800K$ and $1900K$

Outline

Ancient history

- The oldest part of `swak4Foam` is `funkySetFields`
  - It is where the idea of general expressions was first implemented
- Capabilities
  - Creating new fields
  - Manipulation existing ones
- Used for
  - Post-processing: "I need the temperature in Fahrenheit instead of Kelvin"
  - Pre-processing: "I need an `alpha1` initial condition in the form of a semi-sphere"

Command-line mode

- This mode allows rapid creation/manipulation of fields
  - But: No variables available
- The field for the slide "Red cells are part of `testRange`" was created this way:

```
> funkySetFields -create -field centerCells -time 0 -expression "set(<brk>
    <cont>testRange) ? 1 : 0"
```

- Meaning: "*Create* a *field* named *centerCells* at time *0* that is 1 in the `cellSet testRange` and 0 everywhere else"

Boundary conditions of the created fields

- The default for "regular" (not `cyclic` etc) patches is `zeroGradient`
- A list `valuePatches` can be specified
  - These are `fixedValue` and get their value from the next cell
- For existing fields the boundary conditions are erased
  - This is usually not desired
  - Use `keepPatches` to prevent this behavior

Dictionary mode for `funkySetFields`

- If no field is specified `funkySetFields` works in "dictionary mode"
  - Reads dictionary with a list `expression` in it
  - In the list sub-dictionaries
    - Will be "executed" in sequence
  - Format of the dictionaries is a mixture of `swakExpression` and a `funkySetFields`-call
    - Options from the command line are entries in the dictionary

Initializing chemistry

- Idea: let amount of CH_4 gradually rise from `fuel` to `air`
  - Other way round for O_2
- "Burning" would start sooner
- The presented solution is more general than necessary:
  1. Calculate center of `air`-patch
     - Same for `fuel`
  2. Get "direction" by calculating difference
  3. For every cell "project" center onto direction to get distance to `air/fuel`
  4. Linear interpolate according to distance

Calculate CH_4

system/funkySetFieldsDict.setInitialChemistry

```
expressions
(
        initMethan
        {
            field CH4;
            calcDistance (
                "centerFuel{fuel}=sum(pos()*area())/sum(area());"
                "centerAir{air}=sum(pos()*area())/sum(area());"
                "fromTo=(centerAir-centerFuel)/mag(centerAir-centerFuel);"
                "distance=(fromTo&(pos()-centerFuel))/mag(centerAir-<brk>
                    <cont>centerFuel);"
            );
            variables (
                "#calcDistance;"
                "valFuel{fuel}=sum(area()*$field)/sum(area());"
                "valAir{air}=sum(area()*$field)/sum(area());"
            );
            expression "valFuel+distance*(valAir-valFuel)";
            keepPatches true;
        }
```

Same for O_2

system/funkySetFieldsDict.setInitialChemistry

```
initOxygen
{
    field O2;
    calcDistance (
        "centerFuel{fuel}=sum(pos()*area())/sum(area());"
        "centerAir{air}=sum(pos()*area())/sum(area());"
        "fromTo=(centerAir-centerFuel)/mag(centerAir-centerFuel);"
        "distance=(fromTo & (pos()-centerFuel))/mag(centerAir-centerFuel);"
    );
    variables (
        "#calcDistance;"
        "valFuel{fuel}=sum(area()*$field)/sum(area());"
        "valAir{air}=sum(area()*$field)/sum(area());"
    );
    expression "valFuel+distance*(valAir-valFuel)";
    keepPatches true;
}
```

The $

- Using `$name` in expressions is relatively new in `swak`
  - Means "use dictionary entry `name`"
  - Supports relative references (see OpenFOAM Release-Notes)
  - Knows how to handle dimensioned date (see `swak` Release-Notes)

Make things add up to 1

system/funkySetFieldsDict.setInitialChemistry

```
        initRest
        {
            field N2;
            keepPatches true;
            expression "1-(CH4+O2)";
        }
        test
        {
            field sumSpec;
            create true;
            expression "CH4+O2+N2";
        }
);
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Let pyFoamPrepareCase.py do the work

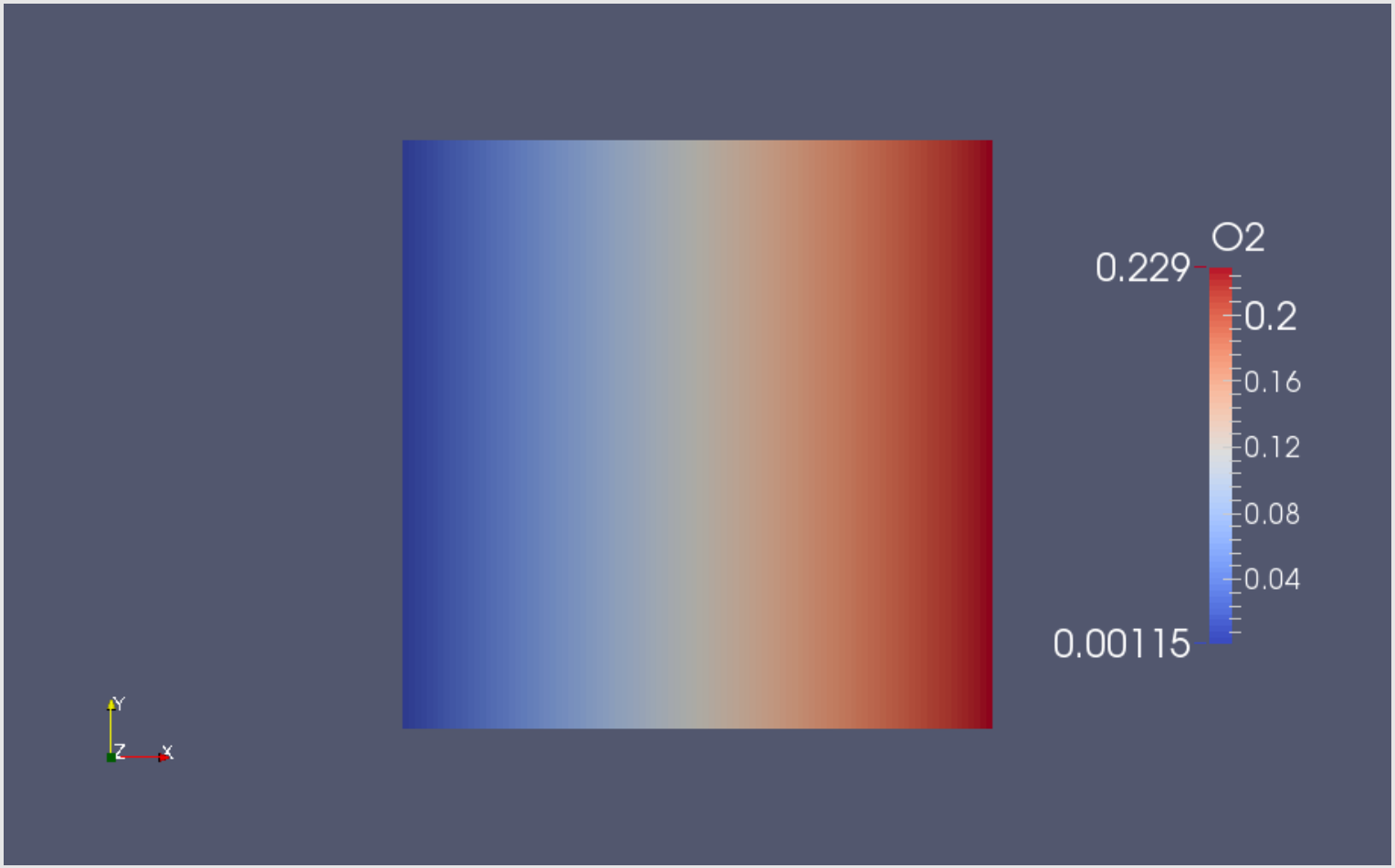
- If pyFoamPrepareCase.py finds a script file casePrepare.sh it executes this after the mesh creation
  - Main application: initial conditions
- Set up the case (including meshing):

```
> pyFoamPrepareCase.py .
```

casePrepare.sh

```
#! /usr/bin/env bash

funkySetFields -time 0 -dictExt setInitialChemistry -noCacheVariables
```

Initial Oxygen field



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Exercises

- Write temperature field `Timp` with temperature in Fahrenheit for post-processing
- Initialize `U` with a pattern that approximates the final solution

Outline

Being cruel (to the solver)

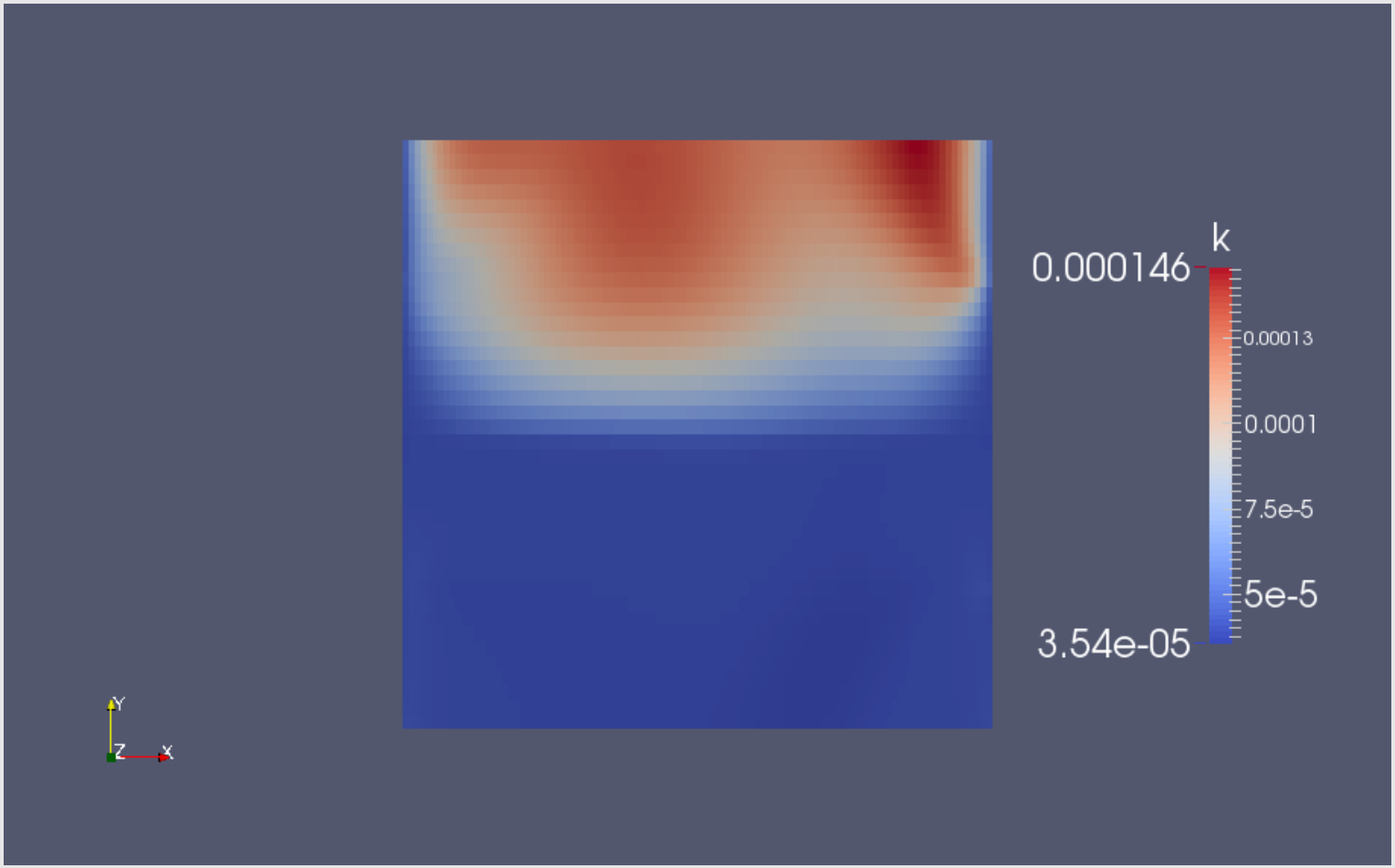
- Nice thing about simulations is that you can do things that are not feasible in real-life
  - Like "switching off" turbulence
- To demonstrate the influence of the turbulence `k` is limited in the lower half of the domain
  - To do this we use a function object `manipulateField`
  - The logical expression `mask` determines whether this cell will be changed

Setting k

controlDict

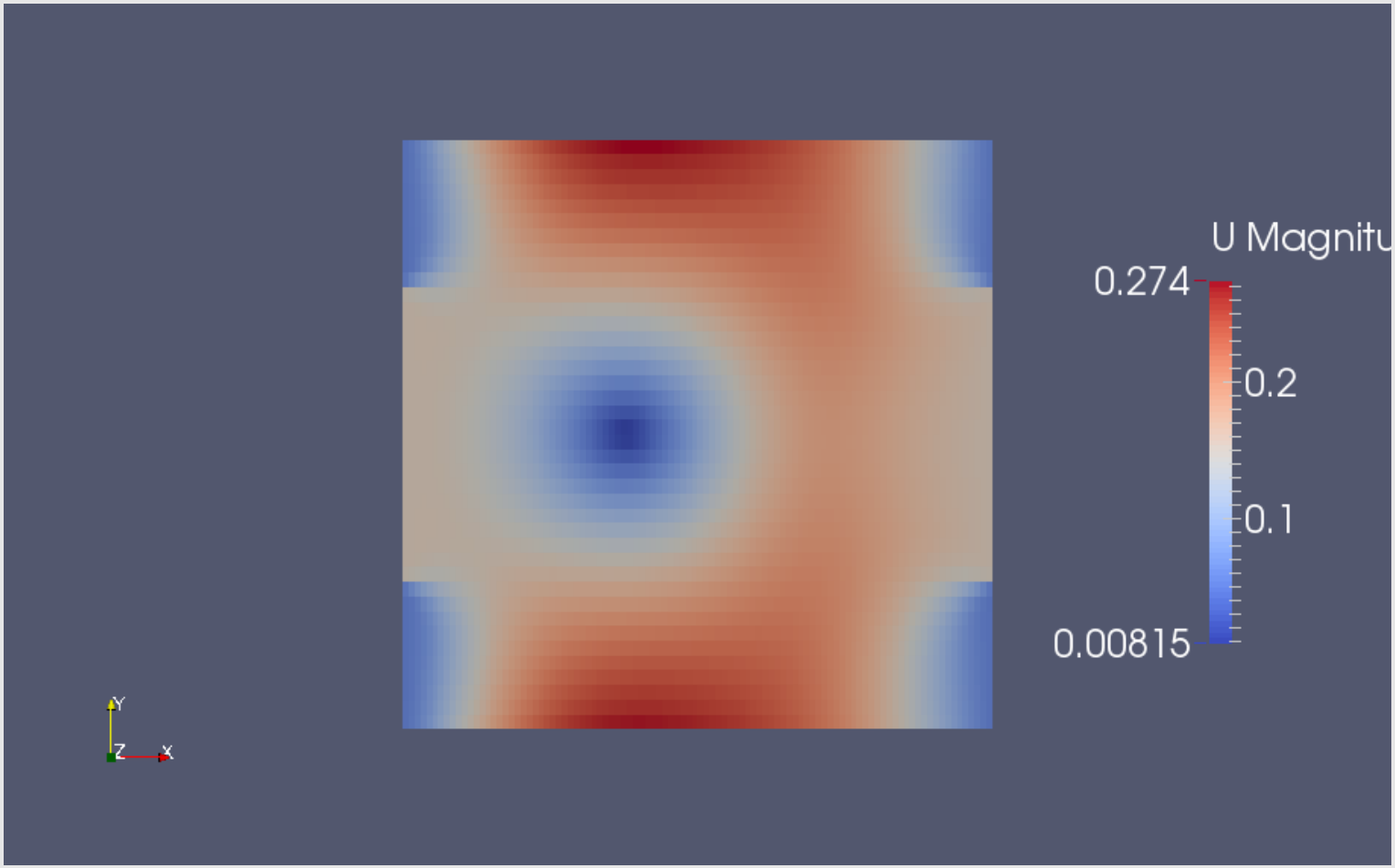
```
limitLowerK {
    type manipulateField;
    outputControl timeStep;
    outputInterval 1;
    fieldName k;
    variables (
        "inK{fuel}=sum(k*area())/sum(area());"
    );
    expression "inK";
    mask "pos().y<0 && k>inK";
}
```

The manipulated field `k`



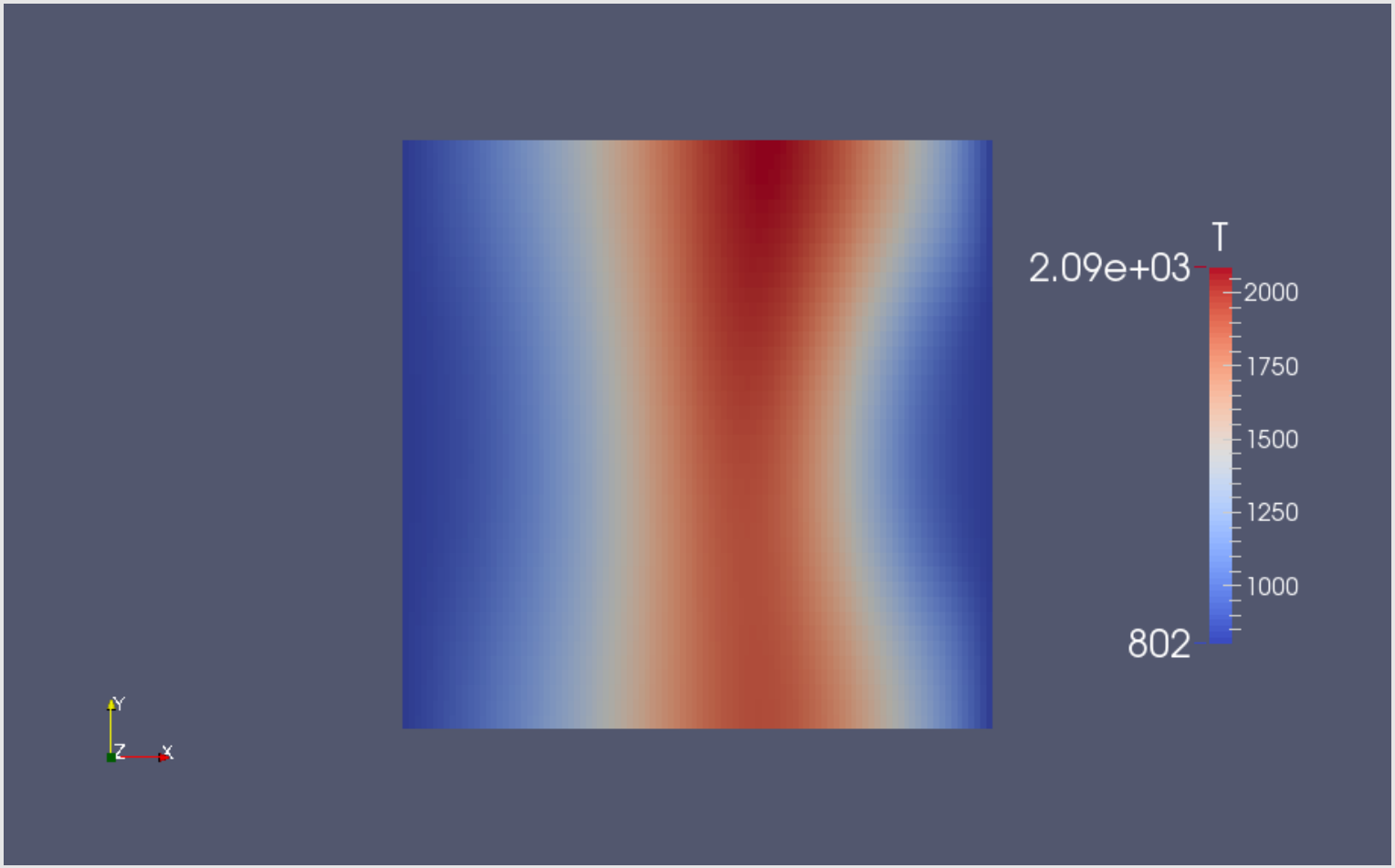
Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Velocity looks almost the same



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Temperature differs



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Outline

Why add particles?

- Because we can
- But sometimes there are sensible reasons:
  - Use particles for visualizing the flow
  - Solver doesn't support particles but we're interested in the way particles behave

Getting particle properties

- Setting up the particle properties would be a training in itself
  - We get settings from the tutorials
  - Adapt them slightly
    - Next slides show only the changed places in the file

Copying sensible settings

```
> cp $FOAM_TUTORIALS/lagrangian/icoLagrangianFoam/channelParticles/constant<brk>
    <cont>/kinematicCloudProperties constant
```

Change the injector

- Inject from the `fuel` patch

kinematicCloudProperties

```
// InjectionModel                  ConeInjection;
InjectionModel                     PatchInjection;

// This goes below $ConeInjectionCoeffs
PatchInjectionCoeffs {
    $ConeInjectionCoeffs;
    patchName fuel;
    SOI                    0.01;
    U0 (0.1 0 0);
}
```

Adapt patch names

kinematicCloudProperties

```
PatchPostProcessingCoeffs
{
    maxStoredParcels 10000;
    patches (
            // in
            // out
        outlet
    );
}
```

Add function objects for clouds

controlDict

```
libs (
    "libsimpleFunctionObjects.so"
    "libsimpleSwakFunctionObjects.so"
    "libswakFunctionObjects.so"
    "libswakChemistryModelFunctionPlugin.so"
    "libgroovyBC.so"
    "libswakTopoSources.so"
    "libsimpleLagrangianFunctionObjects.so"
);
```

The function object that moves the particles

- Specification of fields that particles uses as the continuous phase
  - Could as well be completely different fields (for instance an `expressionField`)

controlDict

```
fuelParticles {
    type evolveKinematicCloud;
    cloudName kinematicCloud;
    rhoName rho;
    UName U;
    muName mu;
}
```

Running with particles

```
> pyFoamRunner.py  --clear --progress  reactingFoam
<<snip>>
Manipulated field k in 864 cells with the expression "inK"

--> Cloud: kinematicCloud
Added 3 new parcels

Cloud: kinematicCloud
Total number of parcels added    = 465
Total mass introduced            = 1.37407e-05
Current number of parcels        = 303
Current mass in system           = 8.95362e-06

Time = 0.0515972

Solving chemistry
```

Problems with the particles

- Run starts well
  - but fails before writing the first time
- Finding that kind of problem can be tedious
  - "I wish it had crashed after writing. Then I could have a look in Paraview"
- swak has a solution
  - Surprise: a function-object

Core-dumps for cases

- This saves the last three time-steps and in case of a crash writes them
  - Caution: use it only when needed as it will require a lot of memory
    - And a little CPU-time

controlDict

```
lastThreeTimesteps {
    type writeOldTimesOnSignal;
    numberOfTimestepsToStore 3;
    writeCurrent true;
}
```

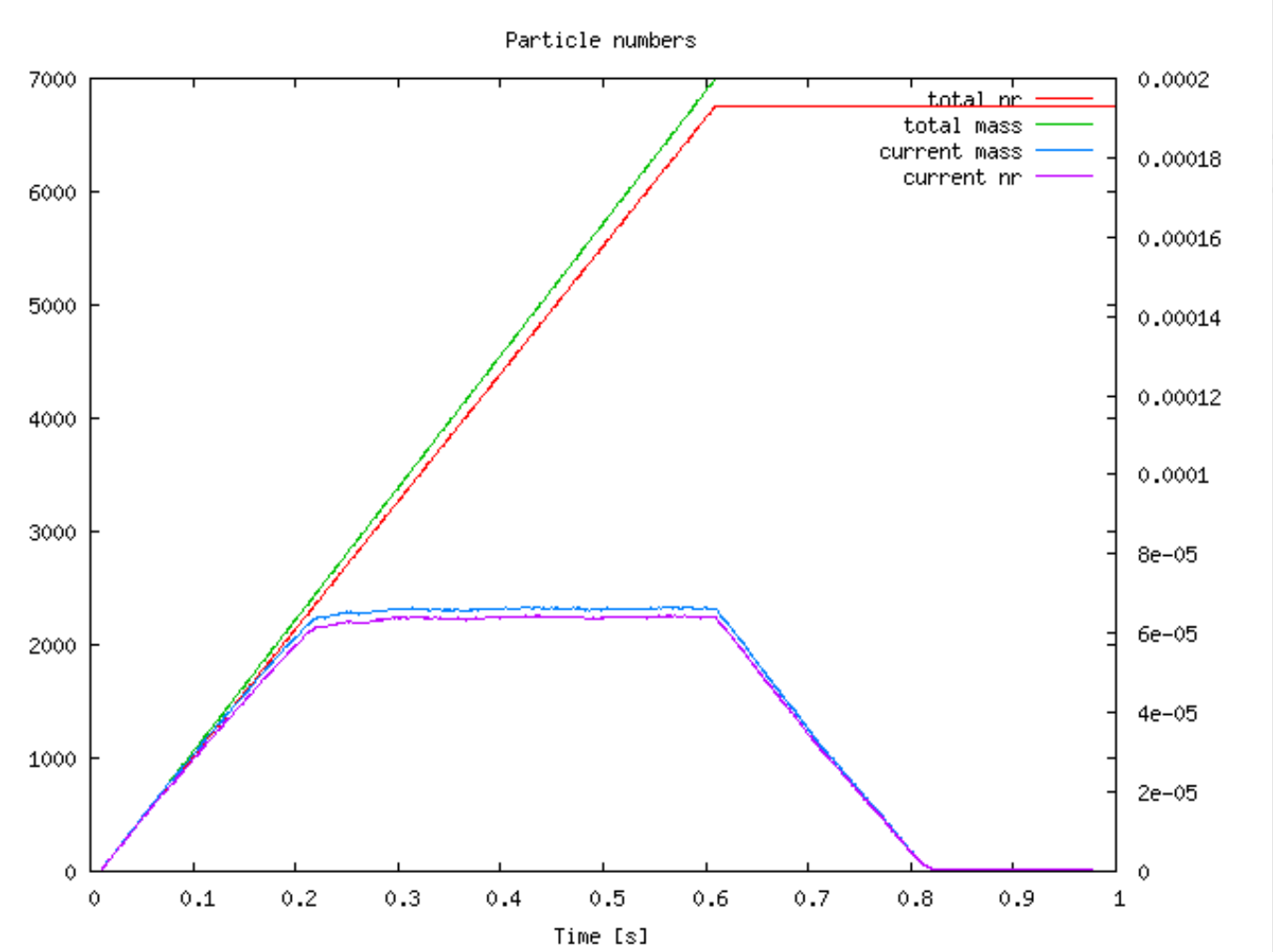
The last timesteps

After the crash:

```
> ls
0/          0.0507994/   0.0510653/
0.0513313/  0.0515972/
<<snip>>
> less 0.0515972/lagrangian/kinematicCloud/U
```

- Inspection shows that some particles have very high velocities
  - Reason can only be guessed
  - But if `limitLowerK` is disabled it runs well

Particles on the left, Mass on the right



Exercises

- Try to set up `customRegexp` to reproduce the previous graph
- Print statistics about the velocity of the particles relative to the gas phase
  - Adding `libswakLagrangianParser.so` to `libs` adds a `valueType cloud` for `swakExpression`
  - In a `cloud` expression `fluidPhase(U)` gives the gas velocity at the current particle position
    - `U` is the particle velocity
  - A sub-dictionary `interpolationSchemes` will be required
- You'll have to rely on the `banana`-trick

Outline

Outline

Why distributions

- Sometimes the single values from the `accumulations` are not enough information
  - but the full data-set is too much
- Information like "How many cells have a temperature between 1000 and 2000" can be found in distributions
- `swak4Foam` has 2 kinds of distributions
  - Distribution of a quantity
  - Average value of a quantity as a function of another
    - Easier to understand: "Average pressure as a function of the height"

Quick distribution primer

- `swakExpressionDistribution` is quite similar to `swakExpression`
  - Calculates `expression` and the computes how it is distributed
  - Expression `weight` determines how each value is weighted
    - For internal fields the cell volume is physically correct
    - But sometime something different is needed
  - Logical expression `mask` determines whether this value is actually used for the distribution
    - Allows things like "distribution of `T`, but only where `alpha1` smaller than 0.5"
- `distributionBinWidth` determines how coarse/fine the distribution is sampled
  - Value will be adapted if needed, but choose a sensible starting value

Distribution of the temperature

controlDict

```
distributionT {
    type swakExpressionDistribution;
    valueType internalField;
    outputControlMode deltaT;
    outputInterval 1;
    outputDeltaT 0.01;
    verbose true;
    expression "T";
    writeTimeline true;
    writeDistribution true;
    weight "vol()";
    mask "true";
    distributionBinWidth 20;
}
```

outputControlMode deltaT

- This is swak-specific
- Used in cases where
  - Output every timestep would be too much data
  - Only at output-times would not be enough
- Executes the function object every `outputDeltaT` seconds (simulation time)
- Does not manipulate the time-stepping
  - Therefor will not be exactly `outputDeltaT` apart
    - But it tries

Average T on the x-axis

- Expression abscissa is the axis on which the averages are taken

controlDict

```
distributionToverX {
    $distributionT;

    type swakExpressionAverageDistribution;
    abscissa "pos().x";
    binNumber 50;
    valueIfZero 0;
}
```

Plotting distribution data

- pyFoamSamplePlot.py assists in the plotting of data from sample
  - But it can do distributions too

Getting information about the available data

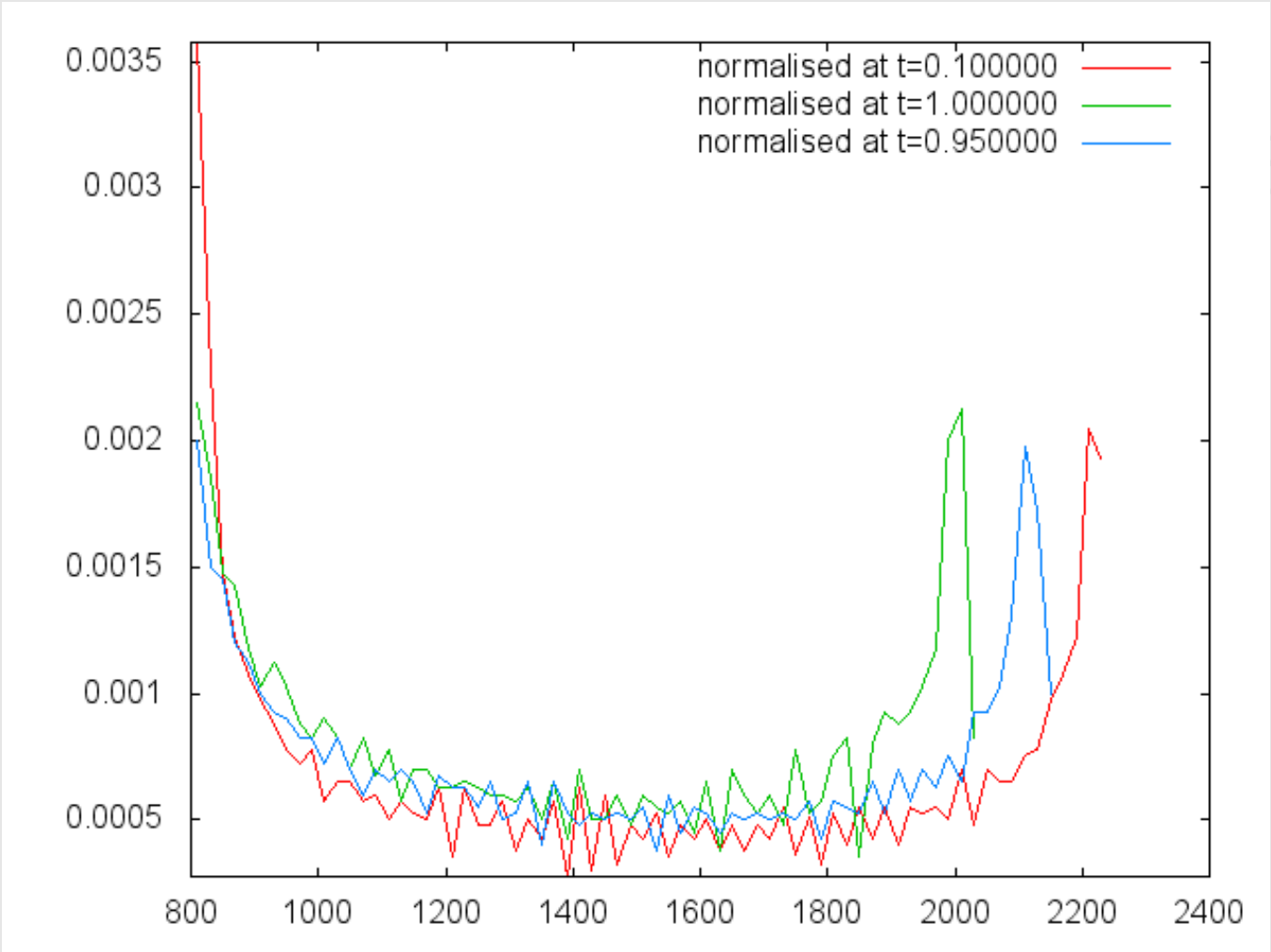
```
> pyFoamSamplePlot.py . --dir=postProcessing/<brk>
    <cont>swakExpressionDistribution_distributionT/distributions --is-<brk>
    <cont>distribution --info
Times :  ['0.00984479', '0.0199493', '0.0297894', '0.0398955',  <<snip>>  <brk>
    <cont>'0.95', '0.95984', '0.969947', '0.979787', '0.989894', '1']
Lines :  ['cumulative_x', 'x']
Fields:  ['normalised', 'raw']
```

Using gnuplot

- pyFoamSamplePlot.py (and pyFoamTimelinePlot) do not plot themselves
  - They only create commands for gnuplot

```
> pyFoamSamplePlot.py . --dir=postProcessing/<brk>
    <cont>swakExpressionDistribution_distributionT/distributions --is-<brk>
    <cont>distribution --line=x --field=normalised  --mode=complete --time<brk>
    <cont>=0.1 --time=1 --time=0.95
set term png
set output "<brk>
    <cont>postProcessing_swakExpressionDistribution_distributionT_distributions<brk>
    <cont>.png"
plot [][0.000275:0.003575] "./postProcessing/<brk>
    <cont>swakExpressionDistribution_distributionT/distributions/0.1/<brk>
    <cont>expression_distribution_x" using 1:2 title "normalised␣at␣t<brk>
    <cont>=0.100000" with lines , "./postProcessing/<brk>
    <cont>swakExpressionDistribution_distributionT/distributions/1/<brk>
    <cont>expression_distribution_x" using 1:2 title "normalised␣at␣t<brk>
    <cont>=1.000000" with lines , "./postProcessing/<brk>
    <cont>swakExpressionDistribution_distributionT/distributions/0.95/<brk>
    <cont>expression_distribution_x" using 1:2 title "normalised␣at␣t<brk>
    <cont>=0.950000" with lines
> pyFoamSamplePlot.py . --dir=postProcessing/<brk>
    <cont>swakExpressionDistribution_distributionT/distributions --is-<brk>
    <cont>distribution --line=x --field=normalised  --mode=complete --time<brk>
```

Distribution of the temperatures



Exercises

- Distribution of the velocity
- Distribution of CO_2 as a function of T
- Plot the other distribution

Outline

Writing data

- Instead of plotting `pyFoamSamplePlot.py` and `pyFoamTimelinePlot` can write data
  - For data-sets of different sizes the have to be `--resample`'d

Writing CSV and Excel

```
> pyFoamSamplePlot.py . --dir=postProcessing/<brk>
    <cont>swakExpressionDistribution_distributionT/distributions --is-<brk>
    <cont>distribution --line=x --field=normalised  --mode=complete --time<brk>
    <cont>=0.1 --time=1 --time=0.95 --resample --csv-file=distT.csv --excel<brk>
    <cont>-file=distT.xls
```

- Now use the spreadsheet software of your liking

"Replaying" long log files

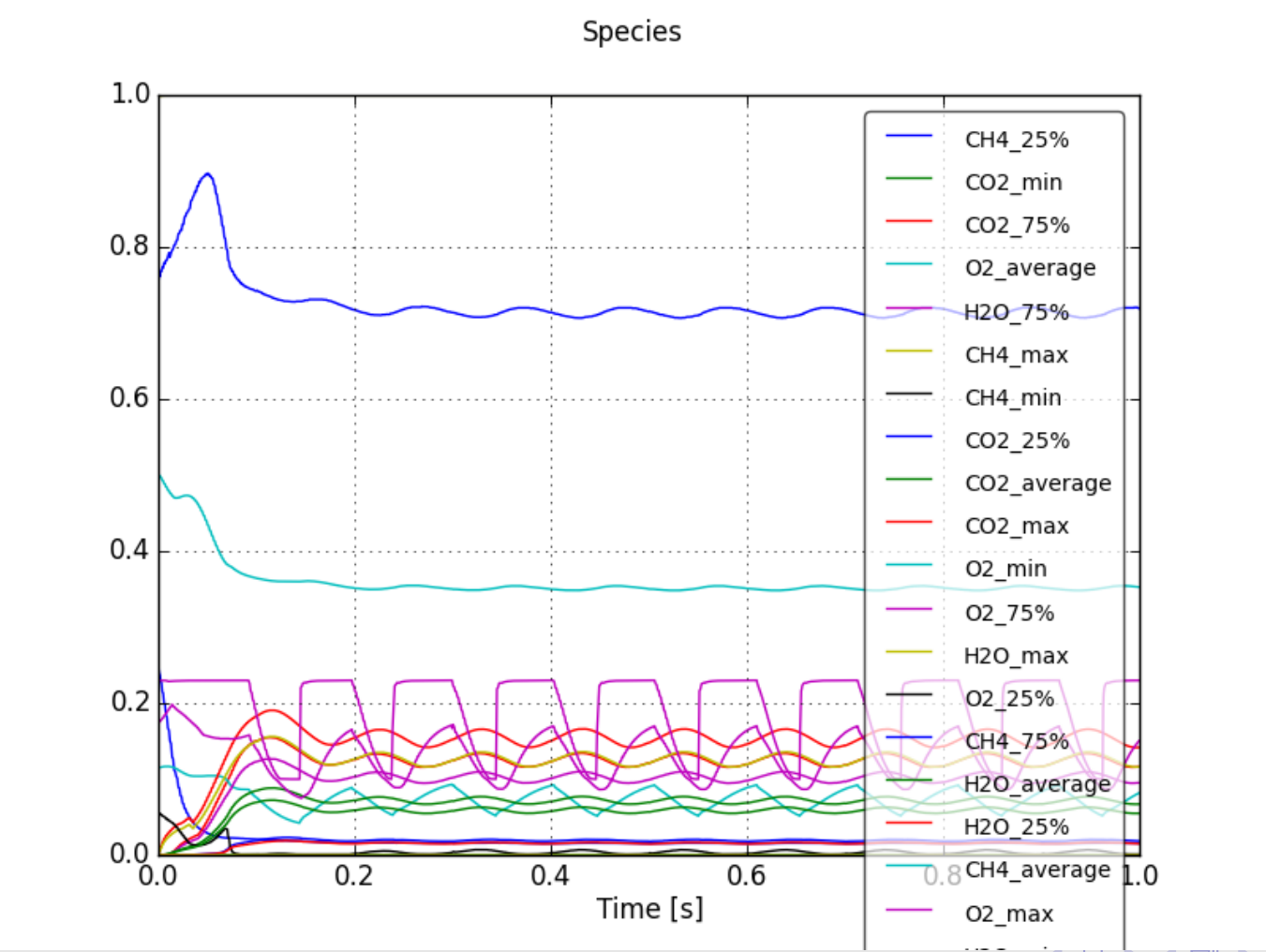
- Sometimes processing long log-files with `pyFoamLogWatcher.py` can take some time
  - Log-files with 1 GB have been seen in the wild
- Sometimes the log-files are gone
  - But the data directory is still there
- If the case was run with PyFoam there are *pickled* versions of the plot data
  - And also the run data
- This can be read and plotted by a special utility
  - Not with `gnuplot` but with `matplotlib`. A bit more aesthetic
    - `gnuplot`-stuff like `logscale` does not work

Redoing the plots

- This is pickled-mode
  - Network-mode is ... advanced

```
> pyFoamRedoPlot.py --pickle-file PyFoamRunner.reactingFoam.analyzed/<brk>
    <cont>pickledPlots
Found 12 plots and 17 data sets
Adding line 11
Adding line 10
Adding line 13
<<snip>>
Plotting 11 : massFlows
Plotting 10 : particles
Plotting 12 : maxTLocation
Plotting 1 : linear
Plotting 3 : bounding No data - skipping
Plotting 2 : continuity
Plotting 5 : courant
Plotting 4 : iterations
Plotting 7 : execution
Plotting 6 : timestep
Plotting 9 : species
Plotting 8 : temperatureExtremes
```

Redone species plot



Using data in `numpy`, `scipy`, `pandas`

- `numpy` and friends offer a great platform for processing data
  - Even better with `ipython` notebooks
- `pyFoamRedoPlot.py`, `pyFoamTimelinePlot.py` and `pyFoamSamplePlot.py` offer the possibility to directly export to this
- `--interactive-after-executiong` works for almost all PyFoam Utilities
  - Drops the user to a Python-shell
    - `ipython` if possible
  - The `self`-object holds data from the utility
    - Most of it with `self.getData()`
- Knowing `Python` is a plus

Getting the data from a run

On the shell

```
> pyFoamRedoPlot.py --pickle-file PyFoamRunner.reactingFoam.analyzed/<brk>
    <cont>pickledPlots --interactive-after-execution --pandas-data
Found 12 plots and 17 data sets
Adding line 11
<<snip>>
Plotting 9 : species
Plotting 8 : temperatureExtremes

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

Python 2.7.6 (default, Nov 19 2013, 19:15:05)
Type "copyright", "credits" or "license" for more information.

IPython 2.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: _
```

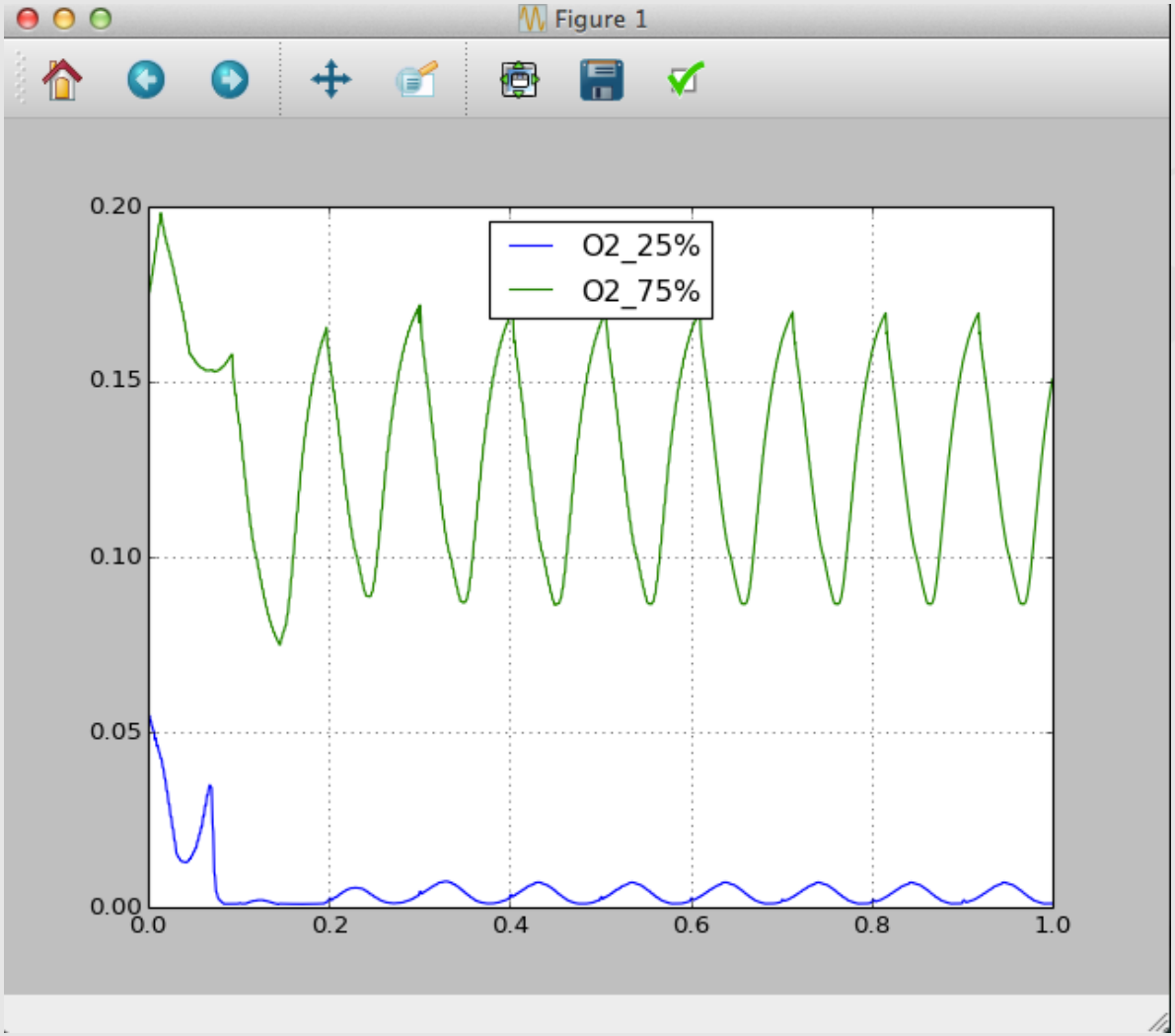
ng GmbH

Plotting only O_2

On the ipython-shell

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

In [2]: specData=self.getData()["plotData"]["species"]

In [3]: specData[["O2_25%","O2_75%"]].plot()
Out[3]: <matplotlib.axes.AxesSubplot at 0x10d669210>

In [4]: (specData["O2_75%"]-specData["O2_25%"]).describe()
Out[4]:
count    1683.000000
mean        0.122822
std         0.028402
min         0.073966
25%         0.094794
50%         0.123427
75%         0.149328
max         0.167547
dtype: float64

In [5]:
```

Plot from the shell



Outline

Further reading

- This presentation only covered parts of `PyFoam` and `swak4Foam`, but there is further information available:
  - On the OpenFOAM-wiki:
    - `http://openfoamwiki.net/index.php/Contrib/swak4Foam` in the section *Further Information* are links to previous presentations
    - `http://openfoamwiki.net/index.php/Contrib/PyFoam` in section *Other material*
  - The `Examples` directory of the `swak`-sources
  - Did I mention the *Incomplete reference guide* for `swak`?
  - The `--help`-option of the `PyFoam`-utilities

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Further presentations

- It was mentioned that the case runs only with `foam.extend-3.1` because the syntax of some files is different with OpenFOAM 2.3
- `pyFoamPrepareCase.py` can handle this kind of situations
  - With something called *templates*
- This is the topic of my other presentation today
  - Maybe I'll see you there?

Exercise

- How often was *The incomplete swak4Foam reference* mentioned today?

The exercises

- Most probably by the time we've reached this slide I said "Please stop typing, we're running out of time"
- Nevertheless you're encouraged to try the examples yourself
  - and do the exercises
- The complete case can be found at `https://bitbucket.org/bgschaid/counterflowflame2dswak4foampyfoam_ofw10/`
  - Feel free to fork it and add your own stuff

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