Automatic testing of solvers using PyFoam

If your results can't be right they should at least be consistent

Bernhard F.W. Gschaider

Innovative Computational Engineering

Darmstadt

25. June 2012

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Outline

Outline

Welcome

- The topic of this presentation is almost as "popular" (and as important) as documentation:

Testing

- This presentation will give an introduction to a testing framework that comes with PyFoam
- Tools we'll use are
  - PyFoam
  - swak4Foam
  - ctest

The situation

- **Ignaz Gartengschirrl** is a CFD-engineer specialized on applications of the damBreak-case
  - You may have met him in other presentations
  - he works with OpenFOAM 1.6-ext and 2.1
- Lately the damBreak-business was a bit slow so he extended his operations to incompressible fluids
  - But he still breaks dams whenever he can
- Ignaz' boss is not convinced that the incompressible solvers are top-quality
  - Asks Ignaz to do some systematic testing
- This is where this presentation starts

The importance of software-testing

From Wikipedia: Software testing can be stated as the process of validating and verifying that a software program/application/product:

1. meets the requirements that guided its design and development
2. works as expected
3. can be implemented with the same characteristics
4. satisfies the needs of stakeholders

Scopes of testing

Unit testing Testing single components

- for OpenFOAM this would involve writing tests for classes like `List` to `fvPatchField` ...
  - hard to do after the main development has happened but a valuable tool during development

Integration testing Testing the correct interaction of the components

- This is the level we'll be working on: seeing how it all comes together in a single solver

System testing Test the integrated system to see that it meets the requirements

- In our case the difference to integration testing is not completely clear

System integration testing Test interaction with third-party software in the deployment environment

- This has to do by every organization individually

Purpose of testing in CFD

The main reasons why someone would want to test a CFD-solver are:

correctness "Is the solution the solver provides the physical solution for the posed problem?"

consistency "Did the solution of the solver change in a new version?"

- or: "can I compare the results to results from the old solver"

changed performance "Does the solver take longer to get a solution?"

Warning

- This presentation contains **one** willingly implanted false information
  - If you spot it please don't tell anyone
- Several involuntary errors
  - If you spot them please tell me
- What is the difference between false information and an error?
  - You'll notice it when you see it

Conventions in the examples

- Shell input and output will look like this:

```
> w
16:15  up 10 days,  6:09, 1 user, load averages: 2.50 2.73 2.89
USER     TTY      FROM              LOGIN@  IDLE WHAT
bgschaid console  -                30May12 10days -
```

- Long lines will be annotated with <brk> and <cont>:

```
print "This is a very long line. In fact it is too long to fit into <brk>
   <cont>one line on the slide.\nMemo to self: write shorter lines"
```

- Three points in the output signify that some output has been omitted:

```
> ls -l /tmp/
drwx------  3 bgschaid    wheel    102 May 30 10:09 launch-0OFjoW/
...
-rw-r--r--  1 bgschaid    wheel  10272 Jun  9 16:57 top.txt
```

Used OpenFOAM-version

- In principle the examples work with both versions on the disc
  - 1.6-ext
  - 2.1
- Requirements are
  - PyFoam
  - swak4Foam
    - separate version for each OpenFOAM-version
- In the final incarnation swak4Foam for 1.6-ext got broken
  - It used to work
- **So we'll have to use OpenFOAM 2.1**

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Preparing for the examples

To get a directory with the examples do one of the following in your working directory (both should be equivalent)

- untar the archive

```
> tar xzf /home/caelinux/OpenFOAM-Workshop/OpenFOAM-7th-<brk>
  <cont>Workshop-2012/Training/BernhardGschaider/<brk>
  <cont>gschaiderTestingWithPythonData.tgz
```

- clone the `git` repository

```
1 > git clone -n /home/caelinux/OpenFOAM-Workshop/OpenFOAM-7<brk>
    <cont>th-Workshop-2012/Training/BernhardGschaider/<brk>
    <cont>gschaiderTestingWithPythonData
```

- got to the directory

```
1 > cd gschaiderTestingWithPythonData
```

- Now you should be in an empty directory

Switching the examples to a different state

- We're going to cover a lot of ground during this lecture
- If you get lost: don't despair
  - Throughout the presentations there will be commands like

```
1 > git checkout 01IntitialCase -b initial
```

- This means:
  - Got to the tag `0000_OriginalTutorial` (on the `zsh` entering `000` then pressing TAB will complete the tag name for you. Can you understand why Ignaz likes the `zsh`?)
  - Open a new branch `original` (that name you can choose for yourself)
- If you did changes that you want to keep for a later review (on the flight home) enter something like (the text of the message is yours)

```
1 > git commit -a -m "This works. Cool. But we're moving to <brk>
    <cont>the next stage"
```

- and later you can recall it with

```
1 > git checkout initial
```

Advertisement: zsh

- Ignaz likes the zsh
- With a good configuration file (.zshrc) it kicks the bash'es ... whatever
  - It is more colorful
  - It has a nice tab-completion
  - It knows about SVN, git or mercurial
    - For instance pressing the TAB-key after git checkout 01<TAB> will complete the 01 to 01InitialCase
    - The shell-prompt displays the current git-branch
- The zsh on the USB-key has all this defined

Outline

Previous presentations

If you're sitting here it's too late, but it won't hurt if you have seen these previous adventures of Ignaz:

- PyFoam - Happy foaming with Python: Held at the OpenFOAM-Workshop in Montreal (2009) Introduction of PyFoam - mostly about the utilities
- Automatization with pyFoam - How Python helps us to avoid contact with OpenFOAM: Held at the Workshop in Gothenburg (2010)
  Mainly about writing scripts with PyFoam
- No C++, please. We're users!: Held at the 2011 Workshop (PennState)
  This presentation is about swak4Foam but mentions some concepts of PyFoam (especially customRegexp)
- pyFoam - The dark, unknown corners: Held at the 2012 PFAU (Austrian User Meeting).
  About the tools for quantitative analysis

But you'll survive without having seen these

The mighty web

Further information about PyFoam are found

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

Outline

Outline

What is PyFoam

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

Outline

Case setup

- Cloning an existing case

```
> pyFoamCloneCase.py $FOAM_TUTORIALS/incompressible/simpleFoam/<brk>
  <cont>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.<brk>
  <cont>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

  pickledData A special file that stores information about the general state of the run

  PyFoamHistory Log with all the PyFoam commands used on that case

Plotting

- Any logfile can be analyzed and plotted

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

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

Outline

The language

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

Three things to know about Python

If you've programmed in a procedural language before then reading Python-programs should be pretty straightforward except for

1. Indentation does the same thing as { and } for C++
2. [] usually is about lists (creation or element access)
3. {} creates a dictionary (lookup table with general keys)
   - whose elements can be accessed with `d[key]`
4. `self` is the same as `this` in C++ (the object itself)

PyFoam as a library

- Below the surface PyFoam is a library that knows how to write OpenFOAM-files
- The "workhorse" is the class `ParsedParameterFile`
  - Needs a filename as a parameter
    - Reads a (OpenFOAM)-dictionary-file
  - Can be used like a regular Python-dictionary
    - Accessing components with the `[]`-operator for reading and writing
    - Dictionaries and lists are mapped to the OpenFOAM dictionaries and lists
  - Manipulated dictionary can be written back with `writeFile()`

ParsedParameterFile

- This example
  - reads the old deltaT and prints it
  - resets deltaT to a thousandths of the endTime in the controlDict
  - writes the changed controlDict back to file

```
1 from PyFoam.RunDictionary.ParsedParameterFile <brk>
    <cont>import ParsedParameterFile

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

Outline

Outline

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

The history/ancestors

swak4Foam is the merge of three previous projects:

funkySetFields a program to set fields with an arbitrary expression

groovyBC using an expression to set a boundary condition

simpleFunctionObjects a collection of function-objects to do simple (but useful) stuff

and has since grown far beyond them

Outline

groovyBC

Something like

```
inlet {
    type groovyBC;
    value uniform 0;
    valueExpression "-normal()*(1+sin(time()))<brk>
        <cont>*((pos().y-min(pos().y))*(max(pos<brk>
        <cont>().y)-pos().y))";
}
```

sets a pulsating, parabolic inlet condition

funkySetFields

- A command like

```
> funkySetFields -create -field COx -<brk>
  <cont>latestTime -expression "CO+CO2"
```

creates an additional field for post-processing.

- Existing fields can be manipulated. For instance

```
> funkySetFields -field alpha -keepPatches -<brk>
  <cont>time 0 -expression "(mag(pos())<1 && <brk>
  <cont>pos().z>0) ? 1 : 0"
```

sets a half-sphere

swakExpression

- In system/controlDict a function-object like

```
carbonOxide {
    type swakExpression;
    valueType patch;
    patchName outlet;
    expression "(CO+CO2)*phi";
    accumulations ( sum );
    verbose true;
}
```

records the amount going out of the system

Outline

Utilities

funkyDoCalc do calculations that are alike to `swakExpression` for post-processing

funkySetBoundaryField statically set boundary fields with expressions

replayTransientBC utility for testing transient boundary conditions without actually running the case

funkySetAreaField setting `FAM`-fields in `1.6-ext`

Function objects

Other function-objects (in various libraries that are part of `swak`) are:

expressionField create a field from an expression (like `funkySetFields`)

manipulateField change the values in parts of a field

patchAverage calculate the average on a patch

patchExpression calculate a general expression on a patch

volumeMinMax calculate the minimum and the maximum of a field

createSampledSet/Surface create a sampled set or surface that is then used for calculations

And many more

Esoteric function objects

Two interesting groups of function objects are

conditionals these are function objects that "own" a list of other function objects which is only executed if a condition is true. For instance:

- **executeIfExecutableFits** checks the name of the executable and only executes if it fits a pattern
- **executeIfObjectExists** only executes if a specific object (usually a field) exists in memory

PDE function objects Solves a PDE for field. Uses swak-expressions for the coefficients. The implemented PDEs are

- transport equation
- laplacian equation

In 2.x there is also a `swakCoded` function object that works like the `coded` FO but can read and write global `swak` variables

Python integration

- To allow general programming there are also function objects that integrate a Python-interpreter

  pythonIntegration executes a Python-script and reads and writes global variables. Applications are
  - reading stuff from data files (physical parameters) to use in swak-expressions
  - manipulation case data (for instance the controlDict or fvSolution) during a run
  - other ....

  executeIfPython only execute other function objects if the Python-program returns a non-zero value
  - Possible application: only dump data if certain conditions are satisfied
- Similar interfaces are possible for other scripting languages (Perl, Ruby, ...)
  - but someone else has to write them

Outline

Outline

"The only valid test"

- Ignaz boss used to be a theoretical physicist
  - So he distrusts experimenters
  - He is suspicious of simulation programs not written in `Fortran`
  - He **looooves** analytic solutions
- For all these reasons Ignaz thinks it is best to start with a test that involves an analytic solution
  - After some thinking he decides to use Taylor-Green vortex

The Taylor-Green vortex (TGV)

- It is a rarely known fact that the french named a train after this problem
- The TGV is a bit like the driven cavity-case. Except:
  - there are no walls
  - there is no driving
- In fact the TGV is a periodic assembly of 2D-vortices
  - initialized with **very** specific velocity and pressure distributions
  - the vortexes get dampened by the viscosity of the fluid
- Did I mention that there is an analytic solution for this?

Formulas for the TGV

The Taylor-Green Vortex is specified for the domain $-\frac{\pi}{2} \leq x, y \leq \frac{\pi}{2}$ with the velocity

$$u = F(t) \sin x \cos y \tag{1}$$

$$v = -F(t) \cos x \sin y \tag{2}$$

$$\tag{3}$$

the damping function

$$F(t) = e^{-2\nu t} \tag{4}$$

and the pressure

$$p = \frac{\rho}{2}(\cos 2x + \sin 2y) F^2(t) \tag{5}$$

Outline

How to create a test

- The basic work-flow for creating a test is:
  1. Set up a template case
  2. Add measurements to the case
     - for instance with swak4Foam
  3. write a test-script
- We'll walk through this here

Preparing the case

- First lets get to the prepared state:

```
> git checkout 01IntitialCase -b intialState
```

- then we'll prepare the mesh

```
> cd taylorGreen
> blockMesh
...
> cp -r 0.orig 0
```

"Look, Ma: no constraints"

- Getting an overview of the boundary conditions

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

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

============ ============== ================
..           defaultFaces   rand
------------ -------------- ----------------
Patch Type   empty          symmetryPlane
Length       128            32
============ ============== ================
U            MISSING        symmetryPlane
p            MISSING        symmetryPlane
============ ============== ================
```

Setting up and running

- Setting the initial conditions according to the analytic solution ...
- ... and running it

```
> funkySetFields -time 0 -field U -expression "vector(-cos(<brk>
    <cont>pos().x)*sin(pos().y),sin(pos().x)*cos(pos().y),0)<brk>
    <cont>"
> funkySetFields -time 0 -field p -expression "1/4*(-cos(2*<brk>
    <cont>pos().x)-cos(2*pos().y))"
> pyFoamRunner.py --clear --progress pisoFoam
```

Initial condition for the Taylor-Green vortex

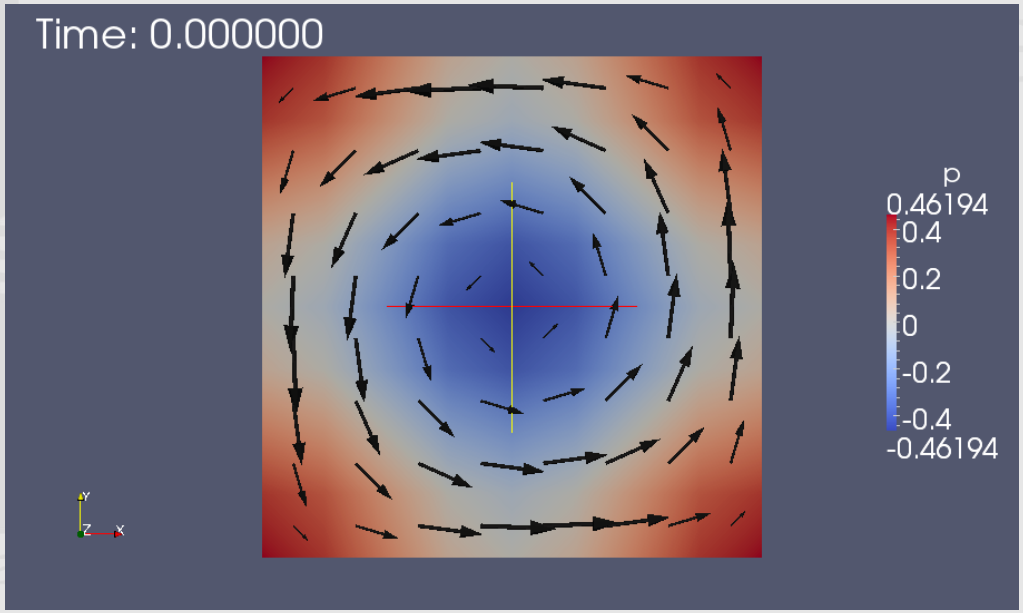


Figure: The vortex in the beginning

Results for the Taylor-Green vortex

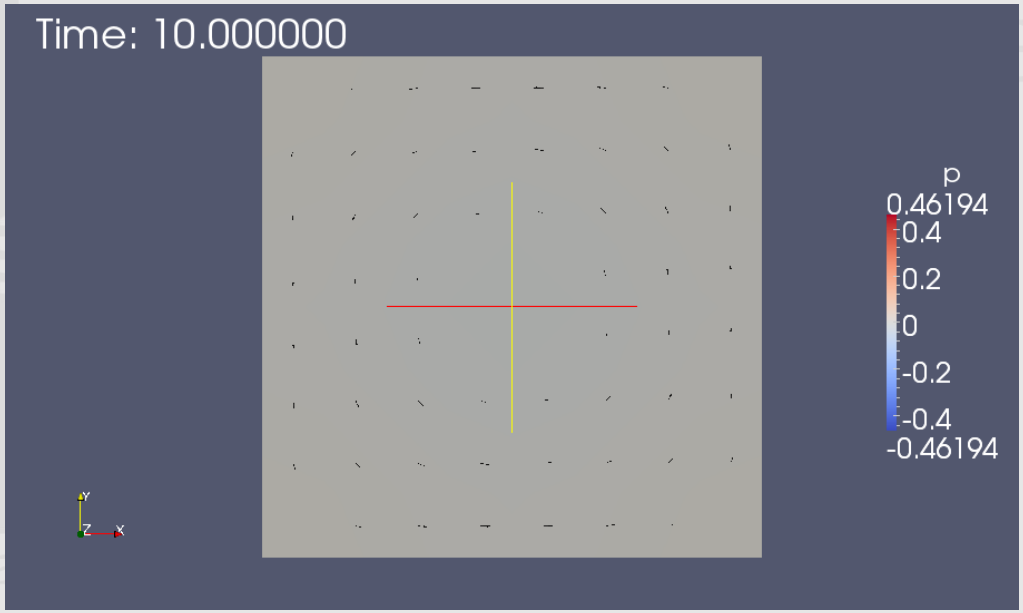


Figure: The vortex in the end

Going to the next stage

- Now that we're sure that the case runs we will
  - adapt it to check for the difference to the analytic solution
  - create a script to set it up
- First we'll set up the state:

```
1 > git checkout 02completeTaylorGreenCase
```

Function object for the analytic solution

- Now we create field with the analytic solution during the run

In controlDict:

```
#include "taylorGreenFormula.dict"

functions
{
    analyticalU
    {
        type expressionField;
        variables $taylorVariables;
        expression "taylorVelocity";
        outputControl timeStep;
        outputInterval 1;
        fieldName Uanalytical;
        autowrite true;
    }
...
```

Comparing simulaion with analytic

One of the evaluations in the controlDict

```
UError
{
    type swakExpression;
    valueType internalField;
    expression "mag(U-Uanalytical)";
    accumulations (
        max
        average
    );
    verbose true;
}
```

The actual implementation of the function

The file taylorGreenFormula.dict:

```
taylorVariables (
        "rhoEq=1;"
        "muEq=1e-1;"
        "nuEq=muEq/rhoEq;"
        "targetRe=1000;"
        "fZero=1;"
        "dampingFunction=fZero*exp(-2*nuEq*time());"
        "taylorVelocity=vector(-cos(pos().x)*sin(pos().y),<brk>
            <cont>sin(pos().x)*cos(pos().y),0)*<brk>
            <cont>dampingFunction;"
        "initialVelocity=vector(-cos(pos().x)*sin(pos().y),<brk>
            <cont>sin(pos().x)*cos(pos().y),0)*fZero;"
        "taylorPressure=-rhoEq/4*(cos(2*pos().x)+cos(2*pos<brk>
            <cont>().y))*dampingFunction*dampingFunction;"
);
```

Making PyFoam extract the numbers

Log output like

```
Expression UError :  max=0.00256771970419 average<brk>
    <cont>=0.00182106474621
```

is caught by this entry in the customRegexp-file:

```
uError {
        name uError;
        theTitle "Absolut error of the velocity";
        expr "Expression UError : max=(%f%) average=(%f%)"<brk>
            <cont>;
        titles (max average);
}
```

Wrapper script for running

Script runAndSummarize.py for printing some basic analytics:

```
#! /usr/bin/env python

from PyFoam.Applications.Runner import Runner as RunIt

run=RunIt([
    "--clear",
    "--progress",
#    "pimpleFoam"])
    "pisoFoam"])

data=run.getData()["analyzed"]
print
print
print "Maximum velocity",data["UMax"]["Simulation_max"],"Error:",data["<brk>
    <cont>uError"]["max"]
print "Maximum analytical velocity change",data["UMax"]["<brk>
    <cont>AnalyticalChange_max"],"Error:",data["uChangeError"]["max"]
print "Relative change (analytical):",data["UMax"]["AnalyticalChange_max"<brk>
    <cont>]/data["UMax"]["Initial_max"],
print "(real):",data["UMax"]["Change_max"]/data["UMax"]["Initial_max"]
print "Maximum pressure",data["pMax"]["Rel_max"],"Error:",data["pError"]["<brk>
    <cont>max"]
print "Relative Error velocity:",data["uError"]["max"]/data["UMax"]["<brk>
    <cont>Simulation_max"]
print "Relative Change Error velocity:",data["uChangeError"]["max"]/data["<brk>
    <cont>UMax"]["AnalyticalChange_max"]
```

Preparation in one script

The prepareCase.sh for setting up the case:

```
#! /bin/sh

pyFoamClearCase.py .

blockMesh

rm -rf 0
cp -r 0.orig 0

funkySetFields -time 0
```

Consistent input for funkySetFields

The funkySetFieldsDict uses the same formula as the check in the controlDict:

```
#include "taylorGreenFormula.dict";

expressions
(
        initVelocity
        {
                field U;
                variables $taylorVariables;
                expression "taylorVelocity";
                keepPatches true;
        }
        initP
        {
                field p;
                variables $taylorVariables;
                expression "taylorPressure";
                keepPatches true;
        }
);
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Running the updated case

```
> ./prepareCase.sh
> ./runAndSummarize.py
 Reading regular expressions from /Volumes/Foam/Subversion/<brk>
    <cont>Testingpreparation/testingWithPythonData/<brk>
    <cont>taylorGreen/customRegexp
Clearing out old timesteps ....
 t =          10

Maximum velocity 0.13308341611 Error: 0.000491629208963
Maximum analytical velocity change 0.8297742872 Error: <brk>
   <cont>0.000491629208963
Relative change (analytical): 0.861930762689  (real): <brk>
   <cont>0.861888744774
Maximum pressure 0.0102154386846 Error: 0.00140942870496
Relative Error velocity: 0.00369414329248
Relative Change Error velocity: 0.000592485470503
```

schung GmbH

ICE
Innovative Computational Engineering

Getting to the first script

- Now that the case is set up we'll get to the first script that
  1. prepares the case
  2. runs the solver
  3. does **no** tests
- First we go to the state:

```
> git checkout 03firstTestScript
```

First test script runTaylorGreenBasic.py

```
1  #! /usr/bin/env python

3  from PyFoam.Infrastructure.CTestRun import CTestRun

5  class TaylorGreenBasic(CTestRun):
       def init(self):
7          self.setParameters(solver="pisoFoam",
                              sizeClass="small",
9                             headLength=200,
                              originalCase="taylorGreen")
11
           self.addToClone("0.orig",
13                         "prepareCase.sh")

15     def meshPrepare(self):
           self.shell("./prepareCase.sh")
17
   if __name__=='__main__':
19     TaylorGreenBasic().run()
```

The basic design of CTestRun

- General goals of the design was
  - **simplicity** to make things no more complicated than writing a configuration file
    - but still have the flexibility of a full programming language
  - **robustness** if a part of the script fails the rest should go on
- The CTestRun-class does the following things:
  1. Set the parameters
  2. Prepare the case
     - decompose for parallel runs if necessary
  3. Run the solver
  4. Execute the tests
- Each of the phases is implemented in one or more methods
  - Subclasses may change the behavior by overriding these
    - Except for the running of the solver

The initialization method `init`

- Similar to the standard Python-constructor `__init__`
  - No calling of the constructor of the super-class is necessary
    - `CTestRun` takes care of this
    - One of the occasions where `CTestRun` hides `Python`-specifics from the user
    - **Never** use `__init__` with `CTestRun`
- Main work done in `init` is
  - Adding data to be cloned
  - Setting parameters

The concept of the 'template case'

- CTestRun never works on the original case
- Instead essential data is copied from a template case to the working case
  - Data that is usually needed to start the case:
    - system
    - constant
    - and 0
  - Data that is added with the addToClone-method
- This makes sure that the original data is not "damaged" by running the test
  - Also allows running more than one test in parallel
- The template case is specified with the originalCase-parameter

Setting parameters

- Parameters are set using `setParameters`
  - As named parameters of the form `<name>=<value>`
  - Settings in a subclass override values from the super-class
- There is a number of predefined parameters (some of the are required):

  solver Required. The name of the solver to be used

  sizeClass Defaults to `unknown`. Approximate size of the test

  originalCase Required. The template case

  originalCaseBasis Directory where the `originalCase` is sought. If unset the current directory is used

  parallel Defaults to `False`. Whether the test is to be run in parallel

  Not all parameters are listed here

The sizeClass

- Available sizeClasses are:

Name	Seconds	Time
unknown	60	60 seconds
tiny	60	60 seconds
small	300	5 minutes
medium	1800	half an hour
big	7200	two hours
huge	43200	12 hours
monster	172800	2 days
unlimited	2592000	30 days

- Try to set a fitting size-class as this will determine the timeout after which the case is terminated by ctest
  - CTestRun complains if the actual run-time is more than two classes "away" from the specified sizeClass

Preparing the mesh

- `prepareMesh` is a typical example for an overrideable method
  - Standard behavior is to run `blockMesh`
  - If the preparation is more complicated it has to be programmed
- The `shell` method runs a command as if it was run from the shell
  - Output is captured

Running the first "test"

```
> ./runTaylorGreenBasic.py
Creating test TaylorGreenBasic_pisoFoam_serial_small
Using solver /Users/bgschaid/OpenFOAM/OpenFOAM-1.7.x/<brk>
    <cont>applications/bin/darwinIntel64DPOpt/pisoFoam
Original case taylorGreen

 WARNING: No environment variable PYFOAM_CTESTRUN_WORKDIR <brk>
     <cont>defined. Using current directory

Running case in ./<brk>
    <cont>TaylorGreenBasic_pisoFoam_serial_small_runDir
No reference data specified

Wrapping method meshPrepare
Wrapping method casePrepare
Wrapping method parallelPrepare
Wrapping method postprocess
Wrapping method decompose
Wrapping method reconstruct
```

Output of CTestRun

- The script informs about its activities
  - After the initialization phase output of the script is prefixed with TEST <testname>
    - All other output is specified by the user in the tests
- Long output (for instance of the solver) is captured and only the first and last 50 lines (the actual number can be specified) are printed
  - Separator lines are added
- Errors and warnings are collected
  - Printed when they occur
  - and at the end (as a summary)

Parts of the output

```
...
TEST TaylorGreenBasic : Run solver
TEST TaylorGreenBasic : Executing /Users/bgschaid/OpenFOAM/OpenFOAM-1.7.x/applications/<brk>
    <cont>bin/darwinIntel64DPOpt/pisoFoam -case ./<brk>
    <cont>TaylorGreenBasic_pisoFoam_serial_small_runDir
 Reading regular expressions from TaylorGreenBasic_pisoFoam_serial_small_runDir/<brk>
     <cont>customRegexp
TEST TaylorGreenBasic : Execution ended
TEST TaylorGreenBasic : Execution was OK

TEST TaylorGreenBasic : Output of /Users/bgschaid/OpenFOAM/OpenFOAM-1.7.x/applications/<brk>
    <cont>bin/darwinIntel64DPOpt/pisoFoam -case ./<brk>
    <cont>TaylorGreenBasic_pisoFoam_serial_small_runDir :
TEST TaylorGreenBasic : The first 200 lines of the output. Of a total of 2536
TEST TaylorGreenBasic : /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.7.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
...
```

End of the output

```
...
TEST TaylorGreenBasic : <brk>
    <cont>/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
TEST TaylorGreenBasic : End of output

TEST TaylorGreenBasic : Solver ran until time 10.0
TEST TaylorGreenBasic : Running postprocessing tools
TEST TaylorGreenBasic : Running post-run tests
TEST TaylorGreenBasic : Looking for tests that fit the prefix <brk>
    <cont>postRunTest
TEST TaylorGreenBasic : No test fit the prefix postRunTest
TEST TaylorGreenBasic : Running serial post-run tests
TEST TaylorGreenBasic : Looking for tests that fit the prefix <brk>
    <cont>serialPostRunTest
TEST TaylorGreenBasic : No test fit the prefix serialPostRunTest
TEST TaylorGreenBasic : Total running time 10.8822350502 seconds

Test successful

Summary of warnings
WARNING: No environment variable PYFOAM_CTESTRUN_WORKDIR defined. Using<brk>
    <cont> current directory
```

Each test script a utility of its own

... and even has a help-text:

```
> ./runTaylorGreenBasic.py -h
Usage: runTaylorGreenBasic.py: [options]

Options:
  -h, --help            show this help message and exit

  Phase:
    Select which phases to run

    --no-clone          Skip cloning phase
    --no-preparation    Skip preparation phase
    --no-serial-pre-tests
...
```

Getting parameter values

Instead of running the script also informs about the parameter values that were set:

```
> ./runTaylorGreenBasic.py --parameter-value=solver
pisoFoam
> ./runTaylorGreenBasic.py --dump-parameters
autoDecompose       : True
doReconstruct       : True
headLength          : 200
nrCpus              : None
originalCase        : taylorGreen
originalCaseBasis   : None
parallel            : False
sizeClass           : small
solver              : pisoFoam
```

Detailed parameter info

Sometimes Ignaz asks himself "Where did I set **this** parameter?"

```
> ./runTaylorGreenBasic.py --verbose-dump-parameters
autoDecompose       : True
    Set by __new__ in /Users/bgschaid/private_python/PyFoam/Infrastructure/CTestRun.py <brk>
        <cont>line 47

doReconstruct       : True
    Set by __new__ in /Users/bgschaid/private_python/PyFoam/Infrastructure/CTestRun.py <brk>
        <cont>line 47

headLength          : 200
    Set by init in ./runTaylorGreenBasic.py line 10

nrCpus              : None
    Set by __new__ in /Users/bgschaid/private_python/PyFoam/Infrastructure/CTestRun.py <brk>
        <cont>line 47

originalCase        : taylorGreen
    Set by init in ./runTaylorGreenBasic.py line 10

originalCaseBasis   : None
    Set by __new__ in /Users/bgschaid/private_python/PyFoam/Infrastructure/CTestRun.py <brk>
        <cont>line 47

parallel            : False
    Set by __new__ in /Users/bgschaid/private_python/PyFoam/Infrastructure/CTestRun.py <brk>
        <cont>line 47

sizeClass           : small
    Set by init in ./runTaylorGreenBasic.py line 10

solver              : pisoFoam
    Set by init in ./runTaylorGreenBasic.py line 10
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Developing tests

- The actual tests are run after the solver finished
- Every method whose name starts with `postRunTest` is executed as a test
  - If during the test the method `fail()` is called the test is unsuccessful
  - The arguments to the method are printed immediately and at the end of the script
  - Otherwise the test is successful
- The option `--jump-to-tests` skips all phases before the tests
  - This is helpful for developing the tests

To get to the first tests:

```
1 > git checkout 04taylorGreenWithTests
```

The runInfo

- The solver is run using the Runner-class in PyFoam
- Data is collected from the output (the same that is usually plotted)
  - Predefined values like residuals, Iterations etc
  - User defined data in the customRegexp
  - The last values of all this data are accessible through the runInfo()-method
    - As a Python-dictionary
    - Contents can be printed using the --print-run-info-option
- The data is written to disc and can be read using the --read-run-info-option

Testing tests without running the solver

```
> ./runTaylorGreenBasic.py --jump-to-tests --read-run-info --print-run-info
....
TEST TaylorGreenBasic : Skipping cloning
TEST TaylorGreenBasic : Skipping case preparation
TEST TaylorGreenBasic : Skipping the serial pre-tests
TEST TaylorGreenBasic : Skipping the pre-tests
TEST TaylorGreenBasic : Skipping running of the simulation

runInfo used in further tests
{'OK': True,
 'time': 10.0,
 'analyzed': {'Continuity': {'Cumulative': -1.67748343251e-17,
                             'Global': 5.49262792193e-20},
              'UMax': {'AnalyticalChange_average': 0.584612101823,
                       'Simulation_average': 0.0936732688093,
                       'Simulation_max': 0.13308341611,
                       'AnalyticalChange_max': 0.8297742872,
                       'Analytical_average': 0.0936466715372,
                       'Analytical_max': 0.132918220272,
                       'Change_average': 0.58458573022,
                       'Change_max': 0.829733836868,
                       'Initial_average': 0.67825877336,
                       'Initial_max': 0.962692507472},
              'Execution': {'clock': 0.0, 'cpu': 0.010000000000000009},
              'Iterations': {'Ux': 1.0, 'Uy': 2.0, 'p': 12.0},
              'Linear': {'Ux': 0.0099072768056,
                         'Ux_final': 6.28399285481e-06,
                         'Ux_iterations': 1.0,
                         'Uy': 0.00990316129082,
                         'Uy_final': 2.25393477893e-08,
                         'Uy_iterations': 2.0,
                         'p': 0.0195989556445,
                         'p_final': 3.06334805676e-07,
                         'p_iterations': 12.0},
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Testing whether the run ended

- The first checks whether the time in the `runInfo` is equal to the `endTime` in the `controlDict`
  - If it is smaller then the run ended abnormally

```
def postRunTestReallyReachedEnd(self):
    if self.runInfo()["time"]<self.controlDict<brk>
        <cont>()["endTime"]:
        self.fail("Simulation did not reach <brk>
            <cont>endTime",
                  self.controlDict()["endTime"<brk>
                      <cont>],"only",
                  self.runInfo()["time"])
```

Additional helper methods

Some methods allow accessing the actual run-data

solution Returns a `SolutionDirectory`-object that encapsulates the solution. This is a PyFoam-class that has (amongst other things):

- **name** Full path to the location of the data
- **constant()** Returns the path to the `constant` directory
- **system()** Path to `system`
- **[ ]** Access the time directories

controlDict() A `ParsedParameterFile` of the `controlDict` of the case

Checking maximum velocity error

Check whether the largest error is more than 1% of the maximum velocity

```
def postRunTestEndVelocityError(self):
    data=self.runInfo()["analyzed"]
    error=data["uError"]["max"]/data["UMax"]["<brk>
        <cont>Simulation_max"]
    print "Relative velocity error:",error
    tolerance=0.01
    if error>tolerance:
        self.fail("Maximum error of velocity",
                  error,"is bigger than",<brk>
                      <cont>tolerance)
```

Output of the tests

```
...
TEST TaylorGreenBasic : Running post-run tests
TEST TaylorGreenBasic : Looking for tests that fit the prefix postRunTest
TEST TaylorGreenBasic : Running the test postRunTestEndChangeError
Relative error:of velocity change 0.000592485470503
TEST TaylorGreenBasic : Running the test postRunTestEndPressureError
Pressure error: 0.00140942870496
TEST TaylorGreenBasic : Running the test postRunTestEndVelocityError
Relative velocity error: 0.00369414329248
TEST TaylorGreenBasic : Running the test postRunTestReallyReachedEnd
TEST TaylorGreenBasic : 4 tests with prefix postRunTest run
TEST TaylorGreenBasic : Running serial post-run tests
TEST TaylorGreenBasic : Looking for tests that fit the prefix serialPostRunTest
TEST TaylorGreenBasic : No test fit the prefix serialPostRunTest
TEST TaylorGreenBasic : Total running time 0.390027046204 seconds

Test failed.

Summary of failures
WARNING: No environment variable PYFOAM_CTESTRUN_WORKDIR defined. Using current <brk>
    <cont>directory
FAILURE: ./TaylorGreenBasic_pisoFoam_serial_small_runDir already existing
```

Tolerance is too tight

A tolerance of 10^{-6} in `postRunTestEndPressureError` (not shown) produces this error message:

```
TEST TaylorGreenBasic : Running the test <brk>
    <cont>postRunTestEndPressureError
Pressure error: 0.00140942870496

Test failed: FAILURE: Maximum error of pressure <brk>
    <cont>0.00140942870496 is bigger than 1e-06

TEST TaylorGreenBasic : Running the test <brk>
    <cont>postRunTestEndVelocityError
```

And in the end the script prints:

```
Test failed.

Summary of failures
WARNING: No environment variable PYFOAM_CTESTRUN_WORKDIR <brk>
    <cont>defined. Using current directory
FAILURE: ./TaylorGreenBasic_pisoFoam_serial_small_runDir <brk>
    <cont>already existing
FAILURE: Maximum error of pressure 0.00140942870496 is <brk>
    <cont>bigger than 1e-06
```

The "=Already existing=" error is OK: this always happens with `--jump-to-tests`

Exception handling in CTestRun

- Wherever possible CTestRun wraps calls to methods with an exception handler
  - Prints the exception
  - Continues the execution
- This makes it possible to continue execution even if one method fails
  - This way the test only has to be executed once to get all failures
- Wrapped methods are
  - The ones listed at the start of the execution
  - All test-methods
- Wrapping doesn't catch basic syntax-errors
  - Those are reported before execution

Typo in the test

After rewriting a test Ignaz finds this in the summary:

```
FAILURE: Python problem during execution of <brk>
    <cont>postRunTestEndPressureError : global name 'toleance' is not<brk>
    <cont> defined
```

Looking further up in the log shows:

```
TEST TaylorGreenBasic : Running the test postRunTestEndPressureError
Pressure error: 0.00140942870496

Test failed: FAILURE: Python problem during execution of <brk>
    <cont>postRunTestEndPressureError : global name 'toleance' is not<brk>
    <cont> defined

Traceback (most recent call last):
  File "/Users/bgschaid/private_python/PyFoam/Infrastructure/<brk>
      <cont>CTestRun.py", line 910, in runAndCatchExceptions
    func(*args,**kwargs)
  File "./runTaylorGreenBasic.py", line 28, in <brk>
      <cont>postRunTestEndPressureError
    if error>toleance:
NameError: global name 'toleance' is not defined
TEST TaylorGreenBasic : Running the test postRunTestEndVelocityError
```

Strömungsforschung GmbH
CE
ve Computational Engineering

Outline

More of the same

- Ignaz is satisfied with the test he has
- But there are still things to test:
  - Is the solution grid-independent?
  - Does it work in parallel?
  - What about other solvers
- To go to this chapter type

```
> git checkout 05taylorGreenTestVariations
```

Inheritance in CTestRun

- Variations of a test are implemented by
  - writing a test-class that inherits from it
  - setting different parameters
    - if unset the values from the parent class will be kept
  - overriding callback methods
  - adding new tests
    - removing tests from the parent class can only be done by overriding them with dummy-implementations
- The init-method of the parent class **must not** be called from user code
  - But not doing that is easier than remembering to do so
    - This design decision should help avoiding some weird errors

Strömungsforschung GmbH
Innovative Computational Engineering

Change the solver

First Ignaz wants to know whether pimpleFoam produces the same results as pisoFoam:

```
1  #! /usr/bin/env python

3  from runTaylorGreenBasic import TaylorGreenBasic

5  class TaylorGreenPimple(TaylorGreenBasic):
       def init(self):
7          self.setParameters(solver="pimpleFoam")


9
   if __name__=='__main__':
11     TaylorGreenPimple().run()
```

Sometimes things **can** be that simple

Parallel processing

- By default all tests are being run as single processor runs
  - If run in parallel `CTestRun` takes care of all tings parallel (decomposing for instance)
- Parameters for parallel processing are:

  parallel Switch on parallelism

  nrCpus How many CPUs should we use

  doReconstruct Do the tests require that the data is reconstructed after the run? Default: `True`

  autoDecompose Default: `True`. If `False` a method `decompose()` that implements the decomposition has to be provided

More processors

Ignaz checks whether this **huge** simulation gives the same results on 2 CPUs

```
#! /usr/bin/env python

from runTaylorGreenBasic import TaylorGreenBasic

class TaylorGreen2Cpu(TaylorGreenBasic):
    def init(self):
        self.setParameters(parallel=True,
                           nrCpus=2,
                           doReconstruct=False)

if __name__=='__main__':
    TaylorGreen2Cpu().run()
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

The blockMesh

Ignaz parameterized the blockMesh when he started.
An excerpt looks like this:

```
nCells 8;
// nCells 16;

blocks
(
    hex (0 1 2 3 4 5 6 7) air ($nCells $nCells<brk>
        <cont> 1) simpleGrading (1 1 1)
);
```

This way changing nCells changes the mesh resolution

Changing the mesh

Ignaz writes a test that sets a different mesh resolution an calls the mesh preparation of the super-class:

```
#! /usr/bin/env python

from runTaylorGreenBasic import TaylorGreenBasic
from PyFoam.RunDictionary.ParsedParameterFile import <brk>
    <cont>ParsedParameterFile

from os import path

class TaylorGreenMediumGrid(TaylorGreenBasic):
    def init(self):
        self.setParameters(nCells=16)

    def meshPrepare(self):
        bm=ParsedParameterFile(path.join(self.solution().<brk>
            <cont>constantDir(),
                                         "polyMesh",
                                         "blockMeshDict"))
        bm["nCells"]=self["nCells"]
        bm.writeFile()
        super(TaylorGreenMediumGrid,self).meshPrepare()

if __name__=='__main__':
    TaylorGreenMediumGrid().run()
```

More on parameters

- In addition to the predefined parameters the user can define his own parameters
  - This allows developing more general cases:
    - Make the setup dependent on a parameter
    - Check for different values depending on the test
  - the variations of the test only have to set different values for the parameters
- Parameter values can be accessed as if the test object was a dictionary:

```
1 self["parameterName"]
```

- **Only** read access to the parameters is allowed
  - This should guard the user from introducing side-effects into the test

Make it even finer

An even finer mesh is only a question of changing one parameter:

```
1  #! /usr/bin/env python

3  from runTaylorGreenMediumGrid import TaylorGreenMediumGrid

5  class TaylorGreenFineGrid(TaylorGreenMediumGrid):
       def init(self):
7          self.setParameters(nCells=32,
                              unusedParameter="fooBar")
9

11 if __name__=='__main__':
       TaylorGreenFineGrid().run()
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Typos in parameters

- CTestRun keeps track of whether parameters are being used during the tests
- If they aren't then in the end it issues a warning

```
WARNING: Unused parameters (possible typo): ['<brk>
    <cont>unusedParameter']
```

- This helps spotting problems where for instance Ignaz would have written nCell instead of nCells and wondered "Why is it as fast as the coarse case?"

Comparisons

Checking the difference between different solver versions:

```
> pyFoamExecute.py --foam=1.7.x ./<brk>
    <cont>runTaylorGreenPimple.py --remove
```

gives different results:

Version	Solver	Relative Error Velocity
1.6-ext	pisoFoam	0.00369584769396
1.6-ext	pimpleFoam	0.00354091736403
2.1.x	pisoFoam	0.00369414329248
2.1.x	pimpleFoam	**0.0115969358359**

Ignaz wonders why one solver changed? (BTW: 1.7.x has the same problem)

Outline

Outline

Facing the real world

- Ignaz looks around for some measurement data
- He turns to "Combustion in a Turbulent Mixing Layer formed at a Rearward-Facing Step" by Robert W. Pitz and John W. Daily (1983)
  - Only considers the non-combustion parts
- Decides to do two tests:
  - One with the default tutorial-settings to make sure that consecutive OpenFOAM-versions are consistent
  - One with the conditions from the paper to make sure that the solver is physically correct
- But first lets set the state of our directory:

```
1 > git checkout 06pitzDailyBasic
```

The pitzDaily tutorial case

- Ignaz decides to base the tests on the pitzDaily-tutorial case that comes with OpenFOAM
  - and use the settings that are shipped with the current version

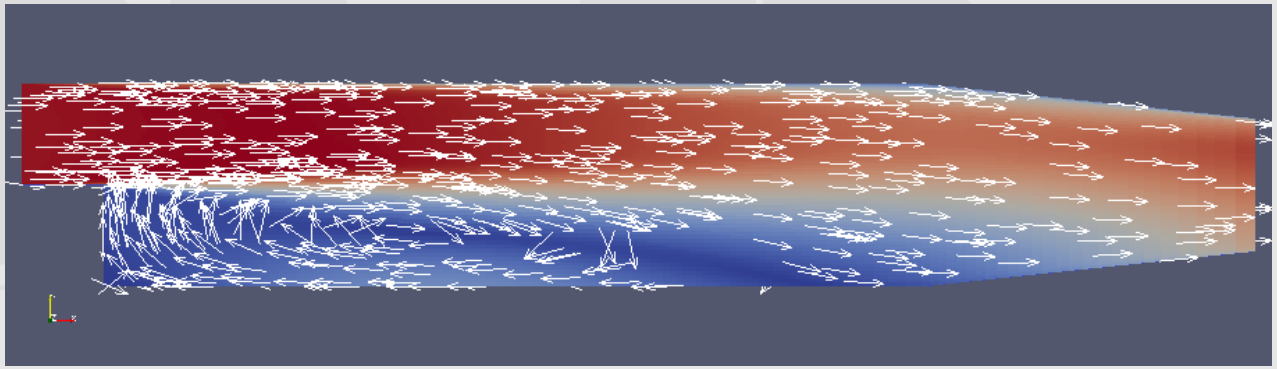


Figure: The pitzDaily-tutorial in its full glory

The base class in PitzDailyBasic.py

- This time the time should **not** reach the endTime
  - because the steady solver should stop before that

```
class PitzDailyRunBasic(CTestRun):
    def init(self):
        self.setParameters(solver="simpleFoam",
                           originalCase="$FOAM_TUTORIALS/<brk>
                               <cont>incompressible/<brk>
                               <cont>simpleFoam/pitzDaily",
                           sizeClass="small")

    def casePrepare(self):
        self.status("Modifying controlDict")
        self.controlDict()["endTime"]=2000
        self.controlDict().writeFile()

    def postRunTestCheckConverged(self):
        self.isNotEqual(
            value=self.runInfo()["time"],
            target=self.controlDict()["endTime"],
#            tolerance=1e-2,
            message="Reached endTime -> not converged")
```

Helper methods for comparing

- CTestRun has four helper functions for comparing numeric values:

  isEqual checks whether two values are the same to within a certain tolerance. The parameters are:

  - **value** the only required parameter. The value we're testing
  - **target** what value **should** be. Default: 0
  - **tolerance** Maximum difference between value and target to consider them equal. Default: 10^{-10}
  - **message** Additional information to be printed in case of failure (there is a default message)

  isNotEqual inverse of isEqual

  isBigger check whether value is bigger than threshold (which defaults to 0)

  isSmaller inverse of isBigger

Testing the plain tutorial

This script uses the inlet conditions from the tutorial (which are not the ones from the paper):

```
1  #! /usr/bin/env python

3  from PitzDailyBasic import PitzDailyRunBasic

5  class PitzDailyRunTutorial(PitzDailyRunBasic):
       def init(self):
7          pass

9  if __name__=='__main__':
       PitzDailyRunTutorial().run()
```

Modifying to replicate the paper

After the regular preparation reset the inlet velocity:

```
#! /usr/bin/env python

from PitzDailyBasic import PitzDailyRunBasic
from PyFoam.RunDictionary.ParsedParameterFile import ParsedParameterFile
from os import path

class PitzDailyRunPaper(PitzDailyRunBasic):
    def casePrepare(self):
        super(PitzDailyRunPaper,self).casePrepare()

        u=ParsedParameterFile(path.join(self.solution().name,
                                        "0",
                                        "U"))
        u["boundaryField"]["inlet"]["value"]="uniform (13.3 0 0)";
        u.writeFile();

if __name__=='__main__':
    PitzDailyRunPaper().run()
```

Not all versions converge

Of the versions on the disc 1.6-ext does not converge (2.1 does)

```
> ./runPitzDailyPaperCompare.py
...
TEST PitzDailyRunPaper : 1 tests with prefix postRunTest <brk>
    <cont>run
TEST PitzDailyRunPaper : Running serial post-run tests
TEST PitzDailyRunPaper : Looking for tests that fit the <brk>
    <cont>prefix serialPostRunTest
TEST PitzDailyRunPaper : No test fit the prefix <brk>
    <cont>serialPostRunTest
TEST PitzDailyRunPaper : Total running time 134.643706083 <brk>
    <cont>seconds

Test failed.

Summary of failures
WARNING: No environment variable PYFOAM_CTESTRUN_WORKDIR <brk>
    <cont>defined. Using current directory
WARNING: Removing old case ./<brk>
    <cont>PitzDailyRunPaper_simpleFoam_serial_small_runDir
FAILURE: Reached endTime -> not converged ( value 2000.0 <brk>
    <cont>within tolerance 1e-10 of target 2000 )
```

Outline

Additional tests

- Ignaz is not satisfied
  - The tests run
  - ... but the results are not compared to real data
- Two things have to be done
  1. Add data and other things to the templates (we don't have control over the tutorial as it is distributed with OpenFOAM)
  2. Compare the sampled data with reference data
- First we'll set the directory to the right state

```
> git checkout 07pitzDailySample
```

Sampling data

Checking the absolute error of the velocity:

```
def postRunTestVelocityProfilesAbsolute(self):
    comparison=self.compareSamples(
        data=self["sampleSets"],
        reference=self["referenceSet"],
        fields=self["sampleFields"])
    print
    print "Absolute error on sample lines"
    print comparison.compare()["max"]
    self.isEqual(
        value=comparison.compare()["max"].max(),
        tolerance=self["velocityAbsoluteTolerance"],
        message="Absolute error of fluid velocity")
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Methods for comparing data

Currently CTestRun has two methods for comparing simulation results to reference data:

compareSamples Compare the result on sample lines to reference data

- Built on the functionality of pyFoamSamplePlot.py. Play with this utility to understand it
- Returns a Data2DStatistics-object. See the reference of that for details

compareTimelines Compare data that evolves over time (usually probes or outputs from other function objects) with reference data

- Built on the functionality of pyFoamTimelinePlot.py

Sampling data 2

The relative error of the velocity:

```
def postRunTestVelocityProfiles(self):
    comparison=self.compareSamples(
        data=self["sampleSets"],
        reference=self["referenceSet"],
        fields=self["sampleFields"])
    print
    print "Relative error on sample lines"
    print comparison.relativeError()
    self.isEqual(
        value=comparison.relativeError().max(),
        tolerance=self["velocityRelativeTolerance"],
        message="Relative error of fluid velocity")
```

Setting data

The modified `init` specifies a location for the data and some additional parameters:

```
class PitzDailyRunPaper(PitzDailyRunBasic):
    def init(self):
        self.setParameters(velocityRelativeTolerance=1e-3,
                           velocityAbsoluteTolerance=1e-3,
                           referenceData="<brk>
                               <cont>PitzDailyPaperData",
                           sampleSets="experimentSample",
                           referenceSet="<brk>
                               <cont>experimentReference",
                           sampleFields=["magU"])
```

Adding data to the case

- If a parameter `referenceData` is found then a directory of that name is used to add data to that case
  - Allows adding stuff to a template case whose original can't be modified
- Contents of this directory:

  additionalFunctionObjects dictionary file. The contents of the entries `libs` and `functions` will be **added** to the respective entries in `controlDict`

  copyToCase a directory that is recursively searched. Each file found in this directory is copied to the same relative location in the case
    - for instance `copyToCase/system/sampleDict` is copied to `system/sampleDict` in the case

The reference data

- Reference data for `PitzDailyRunPaper` was measured from the paper:

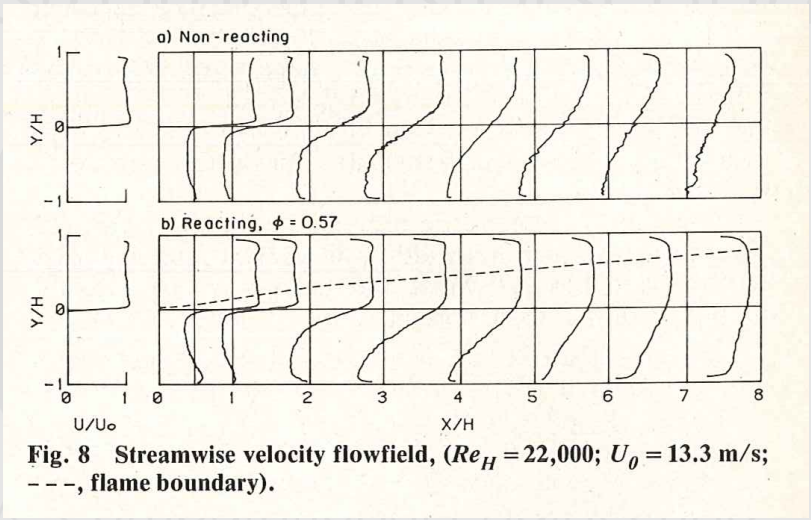


Figure: Measurements from the original paper

- For `PitzDailyRunTutorial` the data from the `1.7.x`-solver was used as reference

Getting the data to sample

Adding function-objects to sample data at the locations of the measurements

```
functions {
    magU {
        type expressionField;
        outputControl timeStep;
        outputInterval 1;
        autowrite true;
        expression "U.x";
        fieldName magU;
    }
  experimentSample
  {
      type sets;
      outputControl timeStep;
      outputInterval 1;
      fields (
          magU
      );
      interpolationScheme cell;
      sets (
          experiment1
          {
              type         midPoint;
              axis         y;
```

Setting data and postprocessing

Separate utility for sampling has to be run

```
class PitzDailyRunTutorial(PitzDailyRunBasic):
    def init(self):
        self.setParameters(velocityRelativeTolerance=1e-3,
                           velocityAbsoluteTolerance=1e-3,
                           referenceData="PitzDailyData",
                           sampleSets="sets",
                           referenceSet="referenceSet",
                           sampleFields=["U"],
                           vortexPos=0.175012)

    def postprocess(self):
        self.execute("sample")
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Running additional programs

- An additional callback method is `postprocess`:
  - run after the solver, but before the tests
- The method `execute()` runs an OpenFOAM-program
  - Additional arguments are passed to the program
  - If appropriate the program is run in parallel. Appropriate means
    - It is a parallel run
    - The case is already decomposed, but not yet reconstructed
  - Method returns a `Runner`-object
    - Can be used similarly to the `solverInfo()`
  - A named argument `regexps` allows specifying a list of regular expressions that the output is scanned for
    - Results are in the `Runner`-object

Data does not fit (and an unused parameter)

Running the test shows several warnings and failures:

```
Test failed.
2
Summary of failures
4 WARNING: No environment variable PYFOAM_CTESTRUN_WORKDIR <brk>
    <cont>defined. Using current directory
  WARNING: Removing old case ./<brk>
    <cont>PitzDailyRunTutorial_simpleFoam_serial_small_runDir<brk>
    <cont>
6 WARNING: No environment variable PYFOAM_CTESTRUN_DATADIR <brk>
    <cont>defined. Using current directory
  FAILURE: Reached endTime -> not converged ( value 2000.0 <brk>
    <cont>within tolerance 1e-10 of target 2000 )
8 FAILURE: Relative error of fluid velocity ( value <brk>
    <cont>0.124874833546  not within tolerance 0.001 of <brk>
    <cont>target 0 )
  FAILURE: Absolute error of fluid velocity ( value 0.10527  <brk>
    <cont>not within tolerance 0.001 of target 0 )
10 WARNING: Unused parameters (possible typo): ['vortexPos']
```

Comparing the velocity

Test	Original data	1.6-ext	2.1
Tutorial	1.7 simulation	not converged	**OK**
Paper	from experiments	not converged	no fit

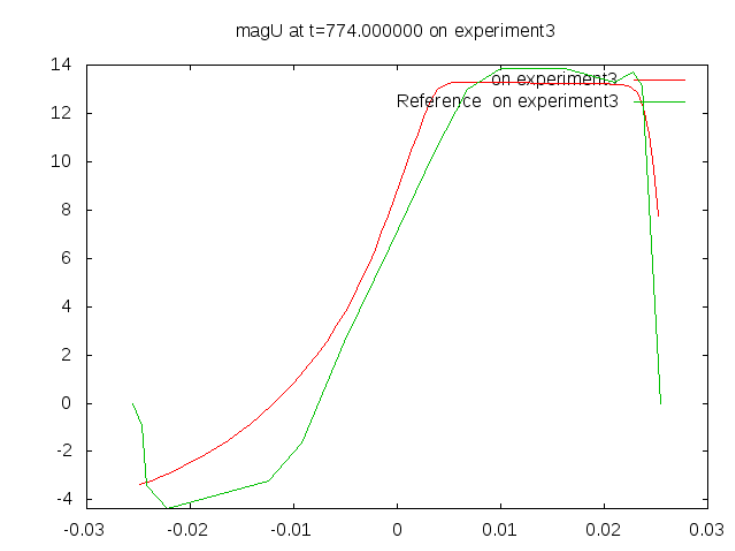


Figure: Simulation and experiment for 2.1

Environment variables

There are two environment variables that influence `CTestRun` (if they're unset the current directory is used):

`PYFOAM_CTESTRUN_WORKDIR` the directory where the test cases are generated and run

`PYFOAM_CTESTRUN_DATADIR` the directory where the test data is searched

For bigger test-suites it is recommended to not have these directories in the same place as the test-scripts

Outline

Ignaz final ambition

- As the profiling data didn't fit so well, Ignaz decides that he wants to add two tests for more obvious flow features:
  - pressure difference between inlet and outlet
  - the length of the vortex behind the step
- To get to the same state as Ignaz we'll use `git` one more time:

```
> git checkout 08additionalPitzTests
```

Pressure difference: using function objects for the test

This is one typical way of writing tests. There are 3 steps:

1. Make the solver output values to the terminal. Typically through a function object
   - I use `swak4foam`. But I'm biased
2. Make `PyFoam` analyze the data
   - Through the `customRegexp`-file
3. Add a testing method to your test-class
   - usually there is not much "programming" required

Calculating the pressure difference

First Ignaz adds a function object via PitzDailyData/additionalFunctionObjects :

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

Reading the difference from the output

Then he adds PitzDailyData/copyToCase/customRegexp so that PyFoam will analyze the data:

```
deltaP{
    expr "Expression pressureDiff on inlet <brk>
        <cont>:  average=(%f%)";
    name deltaP;
    titles (avg);
}
```

- %f% is shorthand for a regular expression that matches any floating-point number
- setting a name makes it easier to access the data later

Testing for the correct difference

Write a test-method to use the data:

```
def postRunTestPressureDifference(self):
    self.isEqual(
        value=self.runInfo()["analyzed"]["<brk><cont>deltaP"]["avg"],
        target=-5.17507, # according to the <brk><cont>2.1-solver
        tolerance=0.1,
        message="Pressure difference between <brk><cont>inlet and outlet")
```

The detachment point

- The point where the flow on the lower wall reverses its direction
  - approximate location is shown on the next picture by a red dot
- To find it Ignaz looks for the face on the lower wall which has
  1. a negative x-component of the velocity
  2. the largest x-coordinate
- On the next slide a graph of the x-component of the flow is shown
  - sampled on the green line in the picture above
  - the point Ignaz is looking for is where the black line crosses the beige line

Detachment point of the eddy

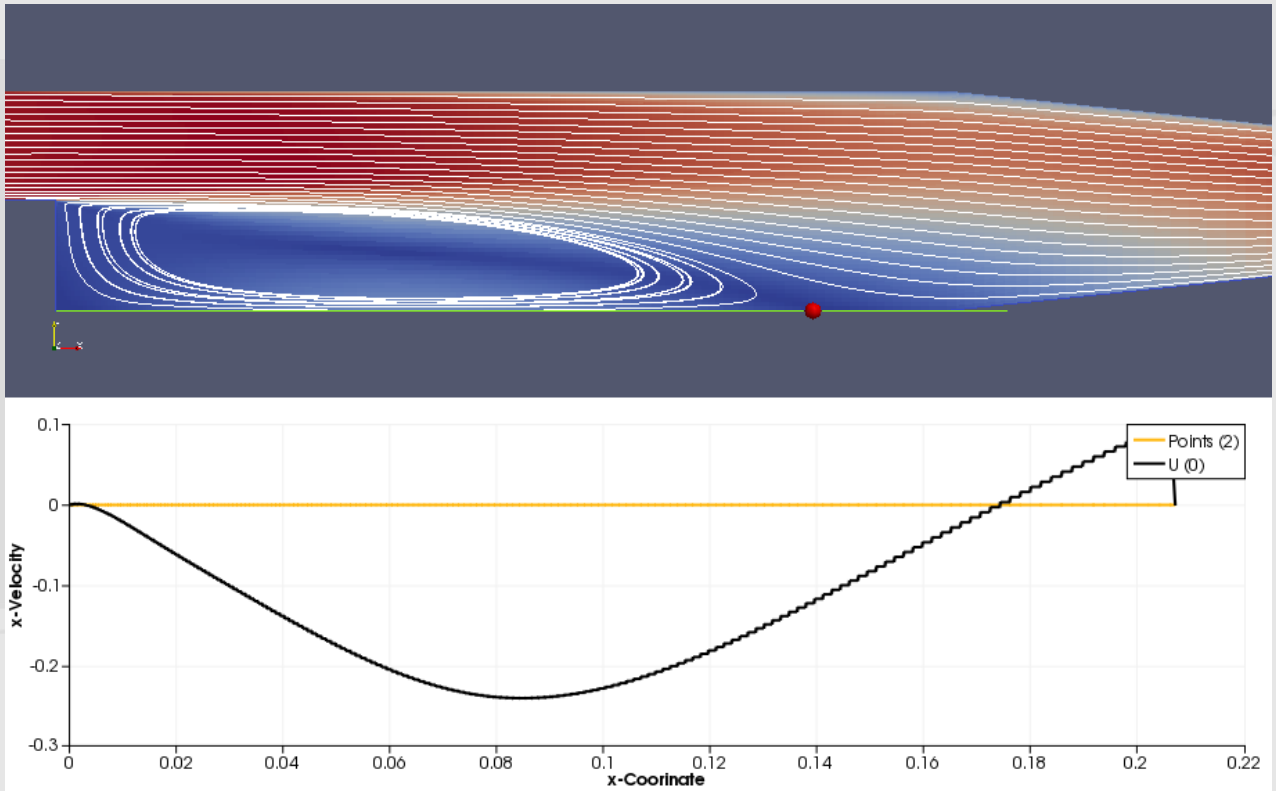


Figure: Location of the detachment point

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Writing tests that use a utility

This is similar to the other technique

1. In the test call a utility that writes the results to the terminal
   - for calling use the `execute()`-method
   - Instead of a `customRegexp`-file you've got to provide the regular expressions for scanning with the `regexp`-parameter
     - This parameter takes a list, so more than one value can be tested
2. Read the results from the dictionary and test them

Finding the detachment point

Ignaz adds a file `PitzDailyData/copyToCase/findDetachmentPoint` for `funkyDoCalc` (of course he tested it before on a finished case):

```
findPoint {
    valueType patch;
    patchName lowerWall;
    expression "internalField(U).x<0 ? pos().x<brk>
        <cont> : -1e10";
    accumulations (
        max
    );
}
```

Testing for the same point

The actual testing method:

```
def postRunTestVortexDetachment(self):
    fdc=self.execute("funkyDoCalc -time 1: %case%/<brk>
        <cont>findDetachmentPoint",
                     regexps=[r"findPoint : : max=(%f%)"])
    value=fdc["analyzed"]["Custom01"]["value 0"]
    print "Detachment of vortex",value
    self.isEqual(value=value,
                 target=self["vortexPos"],
                 tolerance=1e-3,
                 message="Detachment position of the vortex<brk>
                     <cont>")
```

Outline

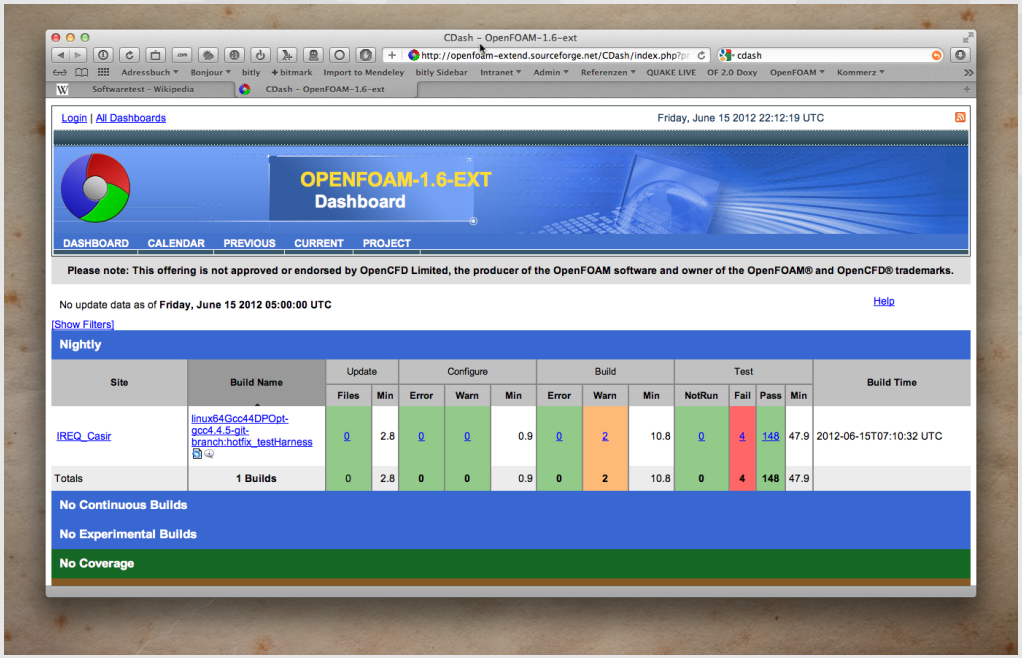
Outline

What is CTest

- ctest is part of the CMake-suite by KitWare (makers of VTK/ParaView)
  - Used to collect tests
  - If necessary compiles the sources
  - Execute them in an orderly fashion
    - Collect the output and
    - Check whether they were successful
  - Send the results to a server
- CDash is the name of the server software
  - Collects the results and
  - Displays them on a dashboard
    - Failed and successful tests
    - Which machines did the test
    - Since when are the tests failing
    - Log-files of the test

CDash-server at openfoam-extend

- Since 2010 there is a CDash on the openfoam-extend at SourceForge
  - Tests the tutorials for 1.6-ext
  - Only checks whether they compile and run. No results are checked



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Outline

Running all the tests

- Ignaz now has a suite of tests
  - some of them not successful
- He wants to
  1. collect them automatically
  2. Execute them
  3. Send the results to a server to keep up to date when they are finally fixed
- To see what he did we use `git` the last time:

```
1 > git checkout 09ctestReady
```

Information for ctest

When asked test-scripts report

- the name of the test
  - generated from the class-name
  - and size, solver and whether it is parallel
- the timeout (determined by the sizeClass of the test)
  - if it runs longer the driver will terminate the test

```
1 > ./runTaylorGreenBasic.py --print-test-name
  TaylorGreenBasic_pisoFoam_serial_small
3 > ./runTaylorGreenBasic.py --timeout
  300
```

Collecting the tests

- To be usable as tests the scripts
  - Must be executable
  - The name must fit a convention (otherwise we'll have to add them by hand). In our case the convention is
    - Extension is `.py`
    - Name starts with `run`
    - Other Python-files are only "support"
- The following script builds this test-suite and adds timeout information
  - Written in the `cmake`-language (which is a cross between `shell`-scripts and a Basic-like language)

Building a list of all tests in CMakeLists.txt

```
project(TestCTestRun)
cmake_minimum_required(VERSION 2.6)

enable_testing()
INCLUDE(CTest)

FILE(GLOB pythonTests run*.py)
LIST(SORT pythonTests)

FOREACH(aTest ${pythonTests})
  execute_process(
    COMMAND ${aTest} --print-test-name
    OUTPUT_VARIABLE testName
    RESULT_VARIABLE testResult
    OUTPUT_STRIP_TRAILING_WHITESPACE
  )

  execute_process(
    COMMAND ${aTest} --timeout
    OUTPUT_VARIABLE testTimeout
    RESULT_VARIABLE testResult2
    OUTPUT_STRIP_TRAILING_WHITESPACE
  )

  IF(testResult OR testResult2)
    MESSAGE("Running ${aTest} failed ... not added")
  ELSE()
    MESSAGE("Adding ${aTest} as ${testName} with timeout ${testTimeout}")
    add_test(${testName} ${aTest})
    set_tests_properties (${testName} PROPERTIES TIMEOUT ${testTimeout})
  ENDIF()
ENDFOREACH(aTest)
```

Executing cmake

```
> cmake .
-- The C compiler identification is GNU 4.2.1
-- The CXX compiler identification is Clang 3.1.0
-- Checking whether C compiler has -isysroot
-- Checking whether C compiler has -isysroot - yes
-- Checking whether C compiler supports OSX deployment target flag
-- Checking whether C compiler supports OSX deployment target flag - yes
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/gschaiderTestingWithPythonData/<brk>
    <cont>runPitzDailyPaperCompare.py as PitzDailyRunPaper_simpleFoam_serial_small with timeout <brk>
    <cont>300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/gschaiderTestingWithPythonData/<brk>
    <cont>runPitzDailyTutorial.py as PitzDailyRunTutorial_simpleFoam_serial_small with timeout 300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/gschaiderTestingWithPythonData/<brk>
    <cont>runTaylorGreen2Cpu.py as TaylorGreen2Cpu_pisoFoam_parallel_2Cpus_small with timeout 300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/gschaiderTestingWithPythonData/<brk>
    <cont>runTaylorGreenBasic.py as TaylorGreenBasic_pisoFoam_serial_small with timeout 300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/gschaiderTestingWithPythonData/<brk>
    <cont>runTaylorGreenFineGrid.py as TaylorGreenFineGrid_pisoFoam_serial_small with timeout 300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/gschaiderTestingWithPythonData/<brk>
    <cont>runTaylorGreenMediumGrid.py as TaylorGreenMediumGrid_pisoFoam_serial_small with timeout <brk>
    <cont>300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/gschaiderTestingWithPythonData/<brk>
    <cont>runTaylorGreenPimple.py as TaylorGreenPimple_pimpleFoam_serial_small with timeout 300
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/<brk>
    <cont>gschaiderTestingWithPythonData
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Running ctest

- ctest takes a number of options. The most important are
  - -S specifies the driver script
  - -V Verbose output from the tests
    - This is useful for our case as the tests take several minutes to run
  - -R only run tests whose names match a certain regular expression
    - for instance -R with the regular expression pimpleFoam.+small executes only the small pimpleFoam-cases
    - useful for redoing only failed tests

What the driver script does

Ignaz driver script does the following things:

- set the CTEST_SITE so that CDash knows who's reporting
- sets CTEST_BUILD_NAME to distinguish different builds or OpenFOAM-versions
- set the location of the sources and binaries
  - currently not used
- Increases the amount of output that the tests are allowed to generate
- Clears the case-directories generated by previous tests
- Runs the tests
  - Building sources and reporting the results to CDash is currently disabled

Driver script doCTest.txt

```
1  cmake_minimum_required(VERSION 2.6)
2
3  EXECUTE_PROCESS(
4          COMMAND hostname -f
5          OUTPUT_VARIABLE CTEST_SITE
6  )
7
8  EXECUTE_PROCESS(
9          COMMAND pyFoamBuildHelper.py --name
10         OUTPUT_VARIABLE CTEST_BUILD_NAME
11 )
12
13 SET(CTEST_SOURCE_DIRECTORY ".")
14 SET(CTEST_BINARY_DIRECTORY ".")
15
16 SET(CTEST_CUSTOM_MAXIMUM_FAILED_TEST_OUTPUT_SIZE 1000000)
17 SET(CTEST_CUSTOM_MAXIMUM_PASSED_TEST_OUTPUT_SIZE 1000000)
18
19 FILE(GLOB runDirs $ENV{PYFOAM_CTESTRUN_WORKDIR}/*_runDir)
20
21 FOREACH(rd ${runDirs})
22   FILE(REMOVE_RECURSE ${rd})
23 ENDFOREACH(rd)
24
25 # ctest_start(Nightly)
26 # update and build here
27 ctest_test()
28 # ctest_submit()
```

Calling everything from one script

- Ignaz doesn't want to forget anything so he writes a single script `runAndSubmitTests.sh` to execute
  - Later this script can be called nightly from a `cron`-job to get constant reports about the state of the code

```
#!/bin/sh

export PYFOAM_CTESTRUN_WORKDIR=`pwd`/Testruns
export PYFOAM_CTESTRUN_DATADIR=`pwd`

cmake .

ctest -V -S doCTest.txt $*
```

Running the tests

Finally Ignaz decided he wants to run the tests:

```
> ./runAndSubmitTests.sh
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/<brk>
    <cont>testingWithPythonData/runPitzDailyPaperCompare.py as <brk>
    <cont>PitzDailyRunPaper_simpleFoam_serial_small with timeout 300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/<brk>
    <cont>testingWithPythonData/runPitzDailyTutorial.py as <brk>
    <cont>PitzDailyRunTutorial_simpleFoam_serial_small with timeout 300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/<brk>
    <cont>testingWithPythonData/runTaylorGreen2Cpu.py as <brk>
    <cont>TaylorGreen2Cpu_pisoFoam_parallel_2Cpus_small with timeout 300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/<brk>
    <cont>testingWithPythonData/runTaylorGreenBasic.py as <brk>
    <cont>TaylorGreenBasic_pisoFoam_serial_small with timeout 300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/<brk>
    <cont>testingWithPythonData/runTaylorGreenFineGrid.py as <brk>
    <cont>TaylorGreenFineGrid_pisoFoam_serial_small with timeout 300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/<brk>
    <cont>testingWithPythonData/runTaylorGreenMediumGrid.py as <brk>
    <cont>TaylorGreenMediumGrid_pisoFoam_serial_small with timeout 300
Adding /Users/bgschaid/OpenFOAM/Subversion/Testingpreparation/<brk>
    <cont>testingWithPythonData/runTaylorGreenPimple.py as <brk>
    <cont>TaylorGreenPimple_pimpleFoam_serial_small with timeout 300
-- Configuring done
-- Generating done
....
```

Strömungsforschung GmbH
CE
Computational Engineering

... and the results

```
    Start 1: PitzDailyRunPaper_simpleFoam_serial_small
1/7 Test #1: PitzDailyRunPaper_simpleFoam_serial_small .......***Failed   69.53 sec
    Start 2: PitzDailyRunTutorial_simpleFoam_serial_small
2/7 Test #2: PitzDailyRunTutorial_simpleFoam_serial_small ....***Failed   68.21 sec
    Start 3: TaylorGreen2Cpu_pisoFoam_parallel_2Cpus_small
3/7 Test #3: TaylorGreen2Cpu_pisoFoam_parallel_2Cpus_small ...   Passed    9.26 sec
    Start 4: TaylorGreenBasic_pisoFoam_serial_small
4/7 Test #4: TaylorGreenBasic_pisoFoam_serial_small ..........   Passed    2.31 sec
    Start 5: TaylorGreenFineGrid_pisoFoam_serial_small
5/7 Test #5: TaylorGreenFineGrid_pisoFoam_serial_small .......   Passed    3.01 sec
    Start 6: TaylorGreenMediumGrid_pisoFoam_serial_small
6/7 Test #6: TaylorGreenMediumGrid_pisoFoam_serial_small .....   Passed    4.66 sec
    Start 7: TaylorGreenPimple_pimpleFoam_serial_small
7/7 Test #7: TaylorGreenPimple_pimpleFoam_serial_small .......***Failed    3.49 sec

57% tests passed, 3 tests failed out of 7

Total Test time (real) = 160.48 sec

The following tests FAILED:
Cannot create directory /Testing/Temporary
Cannot create log file: LastTestsFailed.log
          1 - PitzDailyRunPaper_simpleFoam_serial_small (Failed)
          2 - PitzDailyRunTutorial_simpleFoam_serial_small (Failed)
          7 - TaylorGreenPimple_pimpleFoam_serial_small (Failed)
```

Running only one solver

Ignaz wants to know whether the latest updates to pisoFoam made any difference:

```
> ./runAndSubmitTests.sh -R pisoFoam
...
    Start 3: TaylorGreen2Cpu_pisoFoam_parallel_2Cpus_small
1/4 Test #3: TaylorGreen2Cpu_pisoFoam_parallel_2Cpus_small ...   Passed   20.39 <brk>
    <cont>sec
    Start 4: TaylorGreenBasic_pisoFoam_serial_small
2/4 Test #4: TaylorGreenBasic_pisoFoam_serial_small ..........   Passed    3.41 <brk>
    <cont>sec
    Start 5: TaylorGreenFineGrid_pisoFoam_serial_small
3/4 Test #5: TaylorGreenFineGrid_pisoFoam_serial_small .......   Passed    4.04 <brk>
    <cont>sec
    Start 6: TaylorGreenMediumGrid_pisoFoam_serial_small
4/4 Test #6: TaylorGreenMediumGrid_pisoFoam_serial_small .....   Passed    3.55 <brk>
    <cont>sec

100% tests passed, 0 tests failed out of 4

Total Test time (real) =  31.43 sec
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Outline

Talking with CDash

The presentation is already too long. For details see
http://public.kitware.com/Wiki/CDash

- Installing a CDash-server
  - You'll need a webserver with a database
- Creating a project on the user interface
- Make the server known to the script
  - The parameters can be downloaded through the user interface and are placed in the test directory
- Prepare the ctest-driver script to submit the results
  - In our example you've got to uncomment just the last line

Other testing-frameworks

In principle tests scripts based on `CTestRun` can be used for other automatic testing frameworks (there are a lot)
The requirements are:

- the framework must allow the execution of separate programs
- a test has "failed" when it returns a non-zero return code

But it hasn't be tried yet and the class may need some adaptions

Outline

Outline

The importance of writing tests

- Writing tests will increase your confidence in the solver you write
- Start writing tests as soon as you have the first version of the solver ready
  - Ideally even before. With values that yo know to be correct. From
    - literature
    - experiments
    - analytic evaluations
- Run tests after every major change to the solver
  - If they fail either adapt the solver or the test
- Size is important:

  as big as necessary the simulation should generate meaningful results

  as small as possible The bigger the testsuite the rarer are the testruns

Unsolved problems in this presentation

- Why do new `pimpleFoam`-versions give the wrong results for the Taylor-Green case?
  - It is still possible that this is a set-up problem because `fvSolution` and `fvSchemes` are from `pisoFoam`
- The not-so-good results for the `pitzDaily`-case
  - Everything has to be rechecked
    - Reading of the data
    - Case setup

Unmentioned features

- Additional callbacks

  casePrepare run after meshPrepare
  parallelPrepare preparation after the case is decomposed
  reconstruct custom reconstruction of parallel runs

- Additional tests (method names start with it):

  serialPreRunTest Tests run **before** the case is decomposed
  preRunTest Tests run **before** the solver is run
  serialPostRunTest Tests run after the case is reconstructed

  - All the tests in our examples were run in parallel

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

The future of CTestRun

- The class is (like the rest of PyFoam) under constant development
  - The general interface will "freeze" with the next release of PyFoam
    - No need to rewrite the tests when PyFoam is being upgraded
  - Mostly bugfixes and additional helper-functions
- Suggestions welcome
  - Where could the interface be easier?
  - What additional helpers are needed?
    - Without being solver-specific

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Outline

We're leaving Ignaz

- Ignaz has a framework for his tests
  - This doesn't mean that the tests all run **OK**
  - ... but once Ignaz knows that something is wrong he can try to fix it
- We'll leave him
  - And wish him good luck with fixing the remaining tests

Goodbye

Sorry for the shortness of the presentation

Questions?

My question: Anyone interested in starting a test-suite for the standard solvers? (For instance using the CDash and the repositories at openfoam-extend)