swak4Foam - History and new features

Beyond funkySetFields and groovyBC

Bernhard F.W. Gschaider

MU Leoben

3. October 2011

Outline

Outline

Outline

What it's about

.... it's about swak4Foam (the SWiss Army-Knife for openFOAM)
The presentation

- gives an overview of the capabilities of swak4Foam
- shows some new (yet unreleased) features
- contains not a single picture or movie (hope you all got enough coffee)
  - But the relevant pictures can be easily produced from the examples that come with swak4Foam

Outline

Other information about swak4Foam

. . . because this presentation is short

- On the Wiki openfoamwiki.net
  - page about swak4Foam
  - pages about its predecessors
    - funkySetFields
    - groovyBC
- A "short" presentation "No C++, please. We're users!" (http://bit.ly/qMDlPc) from the OpenFOAM Workshop this year (BTW: you should have been there)
- The README-file that comes with the sources

Outline

Outline

funkySetFields

- Intended as a `setFields` on steroids
  - Set fields using arbitrary expressions
- First release in the March of 2006
- Uses the compiler generator tools `bison` and `flex` to parse the expressions
- Still in search of a better name

simpleFunctionObjects

- Collects a number of function objects that perform non-application-specific tasks
  - Evaluating the field and patch data
  - other stuff
- First release in April 2008
- No need for `bison` and `flex`
- The name is .... simple

groovyBC

- Extension of the idea of `funkySetFields` to boundary conditions
  - Arbitrary boundary conditions using expressions
- First release in the February of 2009
- Again built on the power of `flex` and `bison`
- The name sounds a bit "hippyish"
  - Any better proposals?

Outline

swak4Foam

- Fusion of groovyBC and funkySetFields
  - parsers collected in a common library
- First release in the September of 2010
- Won't work without bison and flex
- The name is a bit better
  - The original name was Project MacGyver
- The basic notion behind swak was that C++-code that was written for the use in **just one case** is evil
  - ... or stupid
  - ... or both

Outline

Outline

The original groovyBC

- In the simplest incarnation sets the value according to an expression

```
valueExpression "(1+sin(time()))*pos().y";
```

- Also allows setting the gradient

```
gradientExpression "1/(internalField(CO)+1e-10)";
```

- Whether the boundary is a Dirichlet or a Neumann is determined by the value-Fraction

```
fractionExpression "phi>0 ? 0 : 1";
```

Variables

- Complex expressions can be broken up into variables

```
variables ( "minX=min(pts().x);"
        "maxX=max(pts().x);"
        "sizeX=maxX-minX;"
);
```

- Variables can also be stored
  - they remember their value between time-steps
  - Are also written to disk

Talking to outsiders

- One feature that was suggested by a user for old-school groovyBC was to access the values on other boundaries
  - This is done via external variables
  - For technical reasons only a single value (min, max, average) is stored
- This has been extended for other entities (fields, sets, ...)

```
variables (
    "pOut{outlet}=sum(p*area())/sum(area());"
    "pFilter{cellZone'filter}=sum(vol()*p)/sum<brk>
        <cont>(vol());"
);
```

Using the machinery for other boundary conditions

- All the machinery of groovyBC can be used for boundary conditions that are more complex than mixed
  - swak currently has two examples
    - a variant of flowRateInletVelocity that takes the flow-rate from an expression
    - what groovyTotalPressure does is left as an exercise to the reader
- To write your own groovified boundary conditions take these two as examples. Basically
  - Inherit from the original boundary condition
  - Add a Driver-object
- If you groovified one of the original boundary conditions they are welcome as contributions

Outline

funkySetFields

- The most-used utility from swak
- Allows setting the a field using an expression
  - From the command line

```
funkySetFields -expression "mag(pos()-vector(0,0,1))<0.5 ?
```

- Or using a dictionary
  - In this mode all the stuff with variables is available
- Also allows manipulating certain cells using the condition-expression
- More examples on the Wiki

funkySetBoundaryFields

- Allows setting fields on boundary patches in a funkySetFields-like fashion
  - Main application is setting static boundary conditions. Either
    1. because you mistrust groovyBC
    2. the type of boundary condition is not supported by groovyBC and the BC is static
  - Only works with a dictionary
  - Any field on the boundary can be set
    - Even if it is not supposed to be a field. This will lead to run-time errors

funkyDoCalc

- Write your own summary of the case
  - The utility
    1. reads a number of expressions (on all entities supported by swak from a dictionary
    2. Evaluates them for specified time-steps
    3. data will be read
    4. Writes the results to the screen
- Useful if you don't want to evaluate functionObjects during the run

Outline

functionObjects

- Function objects are a very useful facility in OpenFOAM
  - Basically plug-ins that are executed at every time-steps
  - Allow the addition of functionality to a case
- OpenFOAM comes with a number of useful functionObjects
  - objects for sampling (surfaces and sets)
  - writing additional fields
  - calculating forces
  - ...
- A very nice thing about functionObjects is that you can add functionality to a case without having to write a specialized solver/utility

simpleFunctionObjects

- Main functionality is calculating single values on patches and fields
  - Data is output to screen (optional) and to files
    - Maximums and minimums
    - Mass flow
    - Averages (also weighted by the mass flow on patches)
- Also other functionality
  - Some "obsoleted" because similar functionObjects were introduced to OpenFOAM
- simpleFunctionObjects now has been integrated into swak because some functionality depends on it
  - But still can be installed and used independently

swakExpression

- Allows evaluations of arbitrary expressions during a run
  - Builds on simpleFunctionObjects

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

- Has a little brother patchExpression that only works on patches

Other functionObjects in swak

expressionField Add a field that is calculated from an expression

The field stays in memory

clearExpressionField Remove a field that was created by `expressionField` from memory

manipulateField change values for an already existing fields (also only parts of the field)

Outline

Zones and Sets

- Sets and Zones are subsets of the total domain
  - Exist for cells and faces
  - Are either produced during mesh-creation or by the appropriate OpenFOAM-utilities
- swak offers the possibility to do calculations on them
  - Possible applications:
    - Calculate flows on control-surfaces
    - Averages in certain regions of the mesh
  - Cell values are interpolated to the faces for faceSets and faceZones

Sampled sets and surfaces

- There is a number of ways to specify sampled sets and sources in OpenFOAM
  - See the example that comes with the `sample`-utility
- Surprisingly `swak` can calculate on these too
  - In order to use them they have to be defined by special `functionObjects`
  - Possible applications
    - Flows through control-surfaces that are not aligned with the mesh
    - Calculations on the interface of a VOF-solver
    - Getting values at "measurement points" and using them in boundary conditions

Source terms as expressions

- This requires modifying the solver
- Two classes to modify the solved equations

expressionSource adds a source term that is calculated according to an expression

forceEquation Manipulate the equations so that it is fixed to a certain value in a part of the mesh

Outline

Outline

Communication between swak entities

- Until now each swak-entity lived on its own: no values were shared
  - Therefor certain evaluations had to be done twice
  - Certain parameters had to adapted in more than one place, making it hard to keep the case consistent
- Now swak offers global scopes
  - These are collections of variables
  - The scopes themselves have names too
- The possibility to have more than one global scope
  - Helps avoiding name-clashes
  - Adds a (crude) possibility for modularization

Defining and using global variables

- There are other ways (see below), but the first way to define a global variable is the addGlobalVariable-functionObject:

```
defineGridThickness {
    type addGlobalVariable;
    outputControl timeStep; // required by OpenFOAM
    outputInterval 1; // required by OpenFOAM
    globalName gridThick;
    globalScope fillDam;
    valueType scalar;
    value 0.0146;
};
```

- A list of global scopes can be added to most swak-dictionaries

```
globalScopes ( "fillDam" );
```

- They are searched in that order for unknown names

Outline

Disclaimer

- As said before: the purpose of swak4Foam is to avoid programming
  - But there are situations when it can't be avoided
- The main purpose of the following functionObjects is
  - supplying the program with swak global variables
  - getting values out of the snipplet to be used by swak

Making the codedFunktionObject play with swak

- Since 2.0 there is a function object codedFunctionObject
  - The user supplies a C++ code
    - The code automatically is compiled and executed
  - This allows accessing virtually **everything** in the solver and enables the functionObject to do **anything**
- The function object swakCoded extends this so that it can communicate with swak via global variables
  - The values are transformed appropriately between swak and C++ (to and from Fields)
- The additional parameters are

  swakToCodedNamespaces global namespaces whose variables should be available to the C++-program
  codedToSwakNamespace a global namespace that the program should write values to
  codedToSwakVariables Which C++-variables should be added to that namespace
- Depending on these variables C++-code is generated to enable the exchange of data
- To work properly a patch to OpenFOAM is required
  - Request for inclusion into the sources is pending

Executing Python-snipplets

- The pythonIntegration-functionObject allow the execution of Python-code
  - Python is the king of scripting languages
    - Has a wide set of available libraries
- The parameters are

  startFile/startCode execute the given code at the start of the simulation
  executeFile/Code execute code at every timestep
  endFile/Code execute in the end
  pythonToSwakNamespace global namespace data should be copied to
  pythonToSwakVariables the variables that should go to the namespace
  swakToPythonNamespaces namespaces that should be copied to the Python-world
  interactiveAfterExecute/Exception a nice way for debugging or trying out new things: instead of going on/failing the interactive Python-shell is opened and Python-commands can be tried
- Some variables get added to the Python-namespace:

  caseDir the current case
  parRun is this a parallel run?
  myProcNo the CPU-number of this process

Applications for the Python-Integration

- The Python-program only has access to a little bit of the case-data
  - But to all Python-libraries (no linking required)
  - One example would be the PyFoam-library
- Possible applications (in addition to the examples given below):
  - write results to a database
  - publish them to a webserver
  - generate graphs
  - ....

Example: using the gravitation direction in a BC

- In the controlDict

```
getGravity
{
    type pythonIntegration;
    startFile "$FOAM_CASE/readGravity.py";
    executeCode "";
    endCode "";
    pythonToSwakNamespace whichWayDown;
    pythonToSwakVariables (g up);
}
```

- the actual code

```
from PyFoam.RunDictionary.SolutionDirectory import SolutionDirectory
from PyFoam.RunDictionary.ParsedParameterFile import ParsedParameterFile
from os import path
from math import sqrt

sol=SolutionDirectory(caseDir)
gFile=ParsedParameterFile(path.join(sol.constantDir(),"g"))

g=gFile["value"]
gAbs=sqrt(reduce(lambda x,y:x+y*y,g,0))
up=g/(-gAbs)
```

- now every member of the swak-family knows where gravity points to

Example: changing the case during the run

- Sometimes it is necessary to start a (steady) simulation with small (cautious) relaxation factors
- This is just the start of functionObject that manipulates the `fvSolution`-file of the case

```
    adaptRelaxation
    {
        type pythonIntegration;
        startCode
            #{
from os import path
from PyFoam.RunDictionary.ParsedParameterFile import <brk>
    <cont>ParsedParameterFile
from math import log,exp,pow
control=ParsedParameterFile(path.join(caseDir,"system","controlDict"<brk>
    <cont>))

end=float(control["endTime"])

del control

scaleTill=int(end*0.5)
scaleFactor=10.

factor=exp(log(scaleFactor)/scaleTill)

print "Scale by actor:",factor
print

fvSol=ParsedParameterFile(path.join(caseDir,"system","fvSolution"),<brk>
    <cont>backup=True)
            #};
```

Integrating with pythonFlu

- PythonFlu is a library by Alexey Petrov that allows to use the libraries of OpenFOAM as Python-libraries
  - A way to write OpenFOAM-programs with the full speed of C++ but the joyfulness of Python
- The functionObject `pythonFluIntegration` allows using it from `swak`
  - Basically the same as `pythonIntegration`
  - Adds a variable `time` to the Python-namespace. This is an encapsulation of the `Time`-object from which every data-structure in OpenFOAM can be accessed
    - This makes it possible to do everything the `codedFunctionObject` can do
- The details how the `Time` variable comes to Python may change
  - Currently a slight addition to `pythonFlu` is needed

Outline

Function objects for PDEs

- Two recent additions are function objects to solve two basic PDEs:

  solveLaplacianPDE solves the equation

$$\frac{\partial \rho T}{\partial t} - \nabla \lambda \nabla T = S_{expl} + S_{impl} T \tag{1}$$

  solveTransportPDE solves the transport equation

$$\frac{\partial \rho T}{\partial t} + \div(\phi, T) - \nabla \lambda \nabla T = S_{expl} + S_{impl} T \tag{2}$$

- Parameters required (depending on the PDE are):

  rho swak-expression and dimension for ρ
  lambda same for λ
  source/sourceImplicit S_{expl} and S_{impl}
  phi Name of the scalarSurfaceField that is ϕ (sorry. Currently no expression)
  fieldName name of the field T that the equation is solved for. There has to be a field-file with the correct boundary conditions
    - Also appropriate entries in fvSolution and fvSchemes

  solveAt When to solve. Possible are timestep, startup or write
  steady whether this is transient or steady

Example: steady solution of the heat conduction

- When this is added to the `flange`-case it calculates a steady solution in the beginning (to compare the transient solution with)

```
TSteady {
    type solveLaplacianPDE;
    outputControl    timeStep;
    outputInterval   1;

    solveAt startup;
    fieldName TSteady;
    steady true;
    //          rho "1" [0 -2 1 0 0 0 0];
    lambda "4e-5" [0 2 -1 0 0 0 0];
    source "0" [0 0 -1 1 0 0 0];
}
```

Example: adding transport

- Add this to the scalarTransportFoam pitzDaily-case to compare different discretization-schems (the actual schems have to be specified in fvSchemes

```
TLinear {
    type solveTransportPDE;
    outputControl   timeStep;
    outputInterval  1;

    solveAt timestep;
    fieldName TLinear;
    steady false;
    rho "1" [0 0 0 0 0 0 0];
    diffusion "0.01" [0 2 -1 0 0 0 0];
    source "0" [0 0 -1 1 0 0 0];
    phi phi;
}
TUpwind {
    $TLinear;

    fieldName TUpwind;
}
```

Outline

Outline

Integrating lagrangian data

- The last thing that is missing is the possibility to calculate with Lagrangian-data
- The problem is a technical one:
  - There is no general way to access the data of the particles like the `objectRegistry`-stuff for fields
- So a general way that doesn't require a recompilation of `swak4Foam` for every new particle class is needed

Improvements on the parsers

- Supporting surface-fields as results in the fields-parser
  - They are already supported as intermediates
- Face-interpolation for `cellSets` and `cellZones`
  - Nice to have but currently not essential
- Propagation of dimensions in expressions
  - as an option
- Allow non-uniform remote variables to be passed

Outline

Goodbye

Do you have

- Questions?
- Better names for `groovyBC` and `funkySetFields`?

Please:

- post bug-reports at the Mantis of `swak4Foam`
- if you think the stuff is underdocumented: consider adding examples to the Wiki

Outline

Outline

Supported OpenFOAM versions

- swak4Foam currently comes in two flavours

1.6/1.7 version

- Still the "standard" version.
- Won't work with older versions
  - But with 1.6-ext
  - Supports the finiteArea-stuff found there

2.0 version

- same features as the 1.x-version
- support for stuff that is only available in 2.0

- The two versions are almost identical in features and both maintained

bison and flex

- bison and flex are the GNU-alternatives to yacc and lex
- Are a parser and a lexer generator
- The programmer provides a grammar and actions. The generators produce valid C-code that parses the grammar and "does the right thing"
  - It would be possible to write the parser by hand ... but tedious and error-prone
- You only have to know that they have to be installed
- In not too old versions

Outline

Getting it

- Sources

1. Go to the Wiki-page
2. Decide whether you want

   stability use the version downloaded with `svn`

   action and tears get the mercurial-version

- Binaries
- There are binaries for some Linux-systems, but I don't maintain them
  - If somebody wants to do that ... be my guest
  - Of course I'll try to help you

Compiling

1. Go to the directory with the sources
2. Copy `swakConfiguration.example` to `swakConfiguration` and adapt it
   - if you want Python-support
3. Type `./Allwmake` and off you go

Deploying and Installing

- Usually libraries and utilities are installed to `$FOAM_USER_APPBIN` and `$FOAM_USER_LIBBIN`
- This means that only you can use them
- There is a script `copySwakFilesToSite.sh` that installs it to `$FOAM_SITE_LIBBIN` and `$FOAM_SITE_APPBIN`
- Only works if you have the appropriate rights
- This means that every user can use them
- Versions in `$FOAM_USER_APPBIN` and `$FOAM_USER_LIBBIN` take precedence
- This can lead to confusing situations