# Avalanche simulation software based on foam-extend

**Updated versions of code and manual can be found at https://bitbucket.org/matti2/fasavagehutterfoam.**

**The OpenFOAM user guide can be found on https://www.openfoam.com/documentation/user-guide/**

**Author: matthias.rauter@uibk.ac.at**

## Copyright

### Source code

All code is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with OpenFOAM. If not, see http://www.gnu.org/licenses/.

### User manual

This work is distributed under the Creative Commons Attribution 3.0 License (CC-BY).

### Raw simulation data

The digital elevation model for tutorial **wolfsgrube** is provided by AMT DER TIROLER LANDESREGIERUNG (AdTLR) Abteilung Geoinformation

Licence: Creative Commons Attribution 3.0 License (CC-BY).

## Solver Theory

The solver is based on a depth-integrated flow model, similar to the Savage-Hutter model (Savage and Hutter; 1989, 1991).

The theory of this solver is described by Rauter and Tukovic (2018). The respective preprint is availabel on arxiv.org.

The application to natural terrain is described by Rauter, Kofler, Huber and Fellin (submitted to GMD).

The **governing equations** can be expressed in terms of surface partial differential equations as

$$\frac{\partial h}{\partial t} + \boldsymbol{\nabla} \cdot \left( h \, \overline{\mathbf{u}} \right) = S_m - S_d, \quad (1)$$

$$\frac{\partial \left( h \, \overline{\mathbf{u}} \right)}{\partial t} + \boldsymbol{\nabla}_\mathrm{s} \cdot \left( h \, \overline{\mathbf{u}} \, \overline{\mathbf{u}} \right) = -\frac{1}{\rho} \boldsymbol{\tau}_\mathrm{b} + h \, \mathbf{g}_\mathrm{s} - \frac{1}{2 \, \rho} \boldsymbol{\nabla}_\mathrm{s} \left( h \, p_\mathrm{b} \right), \quad (2)$$

$$\boldsymbol{\nabla}_{\mathrm{n}} \cdot \left( h \, \overline{\overline{\mathbf{u}}} \, \overline{\overline{\mathbf{u}}} \right) = h \, \mathbf{g}_{\mathrm{n}} - \frac{1}{2 \, \rho} \, \boldsymbol{\nabla}_{\mathrm{n}} \, \left( h \, p_{\mathrm{b}} \right) - \frac{1}{\rho} \, \mathbf{n}_{\mathrm{b}} \, p_{\mathrm{b}} \, , \quad (3)$$

with **unknown flow fields**

- depth-averaged flow velocity $\overline{\overline{\mathbf{u}}}$,
- flow thickness $h$ and
- basal pressure $p_{\mathrm{b}}$.

**Constant parameters** are

- the density $\rho$ and
- the gravitational acceleration $\mathbf{g} = \mathbf{g}_{\mathrm{s}} + \mathbf{g}_{\mathrm{n}}$.

**User-selectable models** have to be provided for

- the basal friction $\boldsymbol{\tau}_{\mathrm{b}}$,
- the entrainment rate $S_m = \frac{\dot{q}}{\rho}$ and
- the deposition rate $S_d$.

**Initial conditions** have to be provided for

- the flow thickness $h$,
- the depth-integrated flow velocty $\overline{\overline{\mathbf{u}}}$ and
- the mountain snow cover thickness $h_{\mathrm{msc}}$ (only if entrainment/deposition is active).

**The classic pressure equation** can be applied by

- deactivating the second term on the right hand side of Equation (3) (switch pressureFeedback in tansportProperties).

**Spatial differential operators**

$$\boldsymbol{\nabla}_{\mathrm{n}} = \left( \mathbf{n}_{\mathrm{b}} \, \mathbf{n}_{\mathrm{b}} \right) \cdot \boldsymbol{\nabla}$$

and

$$\boldsymbol{\nabla}_{\mathrm{s}} = \left( \mathbf{I} - \mathbf{n}_{\mathrm{b}} \, \mathbf{n}_{\mathrm{b}} \right) \cdot \boldsymbol{\nabla}$$

are described in detail by Rauter and Tukovic (2018).

The numerical solution is based on the **Finite Area Library**.

# Getting started (for Linux)

**The majority of the here provided code is available in OpenFOAM-v1712 and provided in the standard installation: https://develop.openfoam.com/Community/avalanche**

## 1 Install foam-extend-4.0

You need to install foam-extend from source to get the newest version. The next official release will contain all required features, making installation more confortable.

- Get the source code via git:

```
cd ~
mkdir foam
cd foam
git clone git://git.code.sf.net/p/foam-extend/foam-extend-4.0 foam-extend-4.0
```

- Install dependencies (newest Ubuntu, may be different for other OS):

```
sudo apt-get update
sudo apt-get install git-core build-essential binutils-dev cmake flex \
zlib1g-dev qt4-dev-tools libqt4-dev libncurses5-dev libiberty-dev \
libxt-dev rpm mercurial graphviz python python-dev
```

- Change to build directory

```
cd ~/foam/foam-extend-4.0
```

- Edit preference file and bashrc

```
echo export WM_THIRD_PARTY_USE_BISON_27=1  >> etc/prefs.sh
echo "alias fe40='source \$HOME/foam/foam-extend-4.0/etc/bashrc'" >> $HOME/.bashrc
source etc/bashrc
```

- Set path to Qt (for paraview), may be different for your enviroment

```
export QT_BIN_DIR=/usr/bin/qmake
echo "export QT_BIN_DIR=$QT_BIN_DIR" >> etc/prefs.sh
```

- Start compiling

```
./Allwmake.firstInstall
```

More information and help: OpenFOAM Wiki

## 2 Build solver

- Clone code repositroy

```
cd ~/foam
git clone https://bitbucket.org/matti2/fasavagehutterfoam.git
```

- Change to build directory, load foam-extend enviroment

```
cd ~/foam/fasavagehutterfoam
fe40
```

- Start compiling

```
./Allwmake
```

## 3 Run test simulations

- Change to tutorials:

```
cd ~/foam/faSavagehutterfoam/tutorials/finiteArea/faSavageHutterFoam
```

- Choose tutorial (see below for details) and change to the respective folder:
    - centrifugalforce
    - dresslersdambreak
    - simpleslope
    - simpleslope_exp

    - rittersdambreak
    - stokersdambreak
    - wolfsgrube

- Load foam-extend enviroment and run example:

```
fe40
./Allrun
```

# Solver

## Implicit solver

The implicit solver can be found in `faSavagehutterfoam/implicictSolver`. It can be called with `faSavageHutterFoam`. Details about this solver can be found in Rauter and Tukovic (2018).

## Explicit solver

The explicit solver can be found in `faSavagehutterfoam/explicitSolver`. It can be called with `faSavageHutterFoamExp`. It is similar to the implicit solver but applies an explicit time integration scheme. Spatial terms are identical to the implicit solver. The explicit solver is faster than the implicit one but not very stable (yet). However, the simple slope example is running and results are similar to the implicit solver (except for some diffuison/creeping in the deposition zone.

# Utilities

## slopeMesh

`slopeMesh`

This simple utility can be used to create simple slopes as used in many of the tutorials.

For an example see `tutorials/simpleslope/constant/slopeMeshDict`.

The source code can be found in `faSavagehutterfoam/slopeMesh`.

## releaseAreaMapping

`releaseAreaMapping`

This utility can be used to set the initial condition (e.g. the initial release zone). This utility reads input from the file `constant/releaseArea`.
Usually the initial condition is set on `0/h` and `0/hentrain`. The initial velocity `0/Us` is usually set to `constant (0,0,0)`.

Three forms of relase areas can be created:

### Sphere

`sphere`

**Spherical release area**, as often used for small scale experiments.
For an example see `tutorials/simpleslope/constant/releaseArea` .
Setting a spherical mass on a slope:

```
fields
(
    h                                   //name of the field
    {
        default 0;                      //default value outside the release area
        regions                         //regions can contain many release areas
        (
            releaseArea2                //name of the release area
            {
                type       sphere;      //type sphere for spherical release
                center     (2.0 0.0 3.5); //center of the sphere
                r          2.0;         //radius of the sphere
                scale      1.0;         //scale up/down the resulting field
            }
        );
    }
);
```

## Polygon

`polygon`

**Classic slab relase**.
For an example see `tutorials/wolfsgrube/constant/releaseArea` .
Setting a slab release on a slope:

```
fields
(
    h                                   //name of the field
    {
        default 0;                      //default value outside the release area
        regions                         //regions can contain many release areas
        (
            releaseArea1                //name of the release area
            {
                type       polygon;     //type polygon for a simple slab
                offset     (0 0 0);     //offset polygon by this value
                vertices                //list of points defining the area
                (
                    ( 4.0  -2.0   0.0)
                    ( 4.0   2.0   0.0)
                    ( 2.0   2.0   0.0)
                    ( 2.0  -2.0   0.0)
                );
                value      0.5;         //target value within the polygonal area
            }
        );
    }
);
```

### Polygon with linear function

`polygonlinear`

Same as polygon but with a linear function for the field within the polygon. Useful to generate complex initial conditions, e.g., the mountain snow cover thickness in Rauter et al. (2016).

```
fields
(
    h                                       //name of the field
    {
        default 0;                          //default value outside the release area
        regions                             //regions can contain many release areas
        (
            releaseArea1                    //name of the release area
            {
                type       polygonlinear;   //type polygon for a simple slab
                offset     (0 0 0);         //offset polygon by this value
                vertices                    //list of points defining the area
                (
                    ( 4.0  -2.0   0.0)
                    ( 4.0   2.0   0.0)
                    ( 2.0   2.0   0.0)
                    ( 2.0  -2.0   0.0)
                );
                valueAtZero 1.6;            //value at (x0,y0,z0)
                x0 0;                       //x0 (default value 0)
                dfdx 0;                    //growth with x (default value 0)
                y0 0;                       //y0 (default value 0)
                dfdy 0;                    //growth with y (default value 0)
                z0 1289;                   //z0 (default value 0)
                dfdz 0.0008;               //growth with z (default value 0)
                projectToNormal yes;       //translate vertical height to surface normal thickn
ess
            }
        );
    }
);
```

The source code can be found in `faSavagehutterfoam/releaseAreaMapping`.

## writeFlatness

`writeFlatness`

Write the flatness of finite area faces into the field `time/Flatness`. The flatness is defined as the ratio between face area and the area of individual triangles of a face (e.g. butterfly angle). A value of 1 indicates perfect flatness of the surface mesh.

The source code can be found in `faSavageHutterFoam/postProcessing/writeFlatness`.

## fafieldsToAscii

`fafieldsToAscii`

This tool reads all finite area fields from all time steps and writes result into a csv-file. The csv-file can be further used to create shape-files (see script `meshtxt2shape.py`).

Set parameters in `system/fafieldsToAsciiDict`, e.g:

```
offset (0.0 0.0 0.0);     //offset position vector

fields                    //list of fields to export.
(
    "Us"                  //here we export velocity
    "h"                   //and flow thickness
);
```

See also **foam2shape.py** for a direct export.

## faMeshToAscii

`faMeshToAscii`

This tool reads the finite area mesh and writes the respective geometry into a csv-file. The csv-file can be further used to create shape-files (see script `meshtxt2shape.py`).

Set parameters in `system/faMeshToAsciiDict`, e.g:

```
offset (0.0 0.0 0.0);       // offset position vector
```

See also **foam2shape.py** for a direct export.

# Scripts

Some useful python(2.7) scripts can be found in the folder `scripts`. Dependencies are `numpy`, `scipy` and `shapefile`.

### txt2mesh.py

This script can be used to generate an STL-file from a digital terrain model (ascii file).
Moreover, the script can create an OpenFOAM mesh. See also tutorial `wolfsgrube` (script `runMesh.sh` or `runPMesh.sh`).

```
usage: txt2mesh.py [-h] [-i I] [-o O] [-xres XRES] [-yres YRES] [-p1 P1]
                   [-p2 P2] [-p3 P3] [-p4 P4] [-plot] [-walls] [-mesh]
                   [-fillup] [-quad] [-trionly] [-exactcopy]
                   [-butterflyangle BUTTERFLYANGLE] [-offsetx OFFSETX]
                   [-offsety OFFSETY]

Taking a Raster-File and creating a STL-File

optional arguments:
  -h, --help            show this help message and exit
  -i I                  input filename
  -o O                  output filename
  -xres XRES            x resolution
```

```
  -yres YRES            y resolution
  -p1 P1                lower left edgepoint
  -p2 P2                lower right edgepoint
  -p3 P3                upper right edgepoint
  -p4 P4                upper left edgepoint
  -plot                 show a plot of dem and boundaries
  -walls                add walls to the stl (e.g. for cfmesh)
  -mesh                 create mesh instead of stl
  -fillup               fill up nan values with neatrest neighbor
  -quad                 create mesh with quads
  -trionly              create mesh with triangles only
  -exactcopy            create a triangulation with the resolution of the dem
  -butterflyangle BUTTERFLYANGLE
                        maximum allowed butterfly angle
  -offsetx OFFSETX      offset mesh by x m
  -offsety OFFSETY      offset mesh by y m
```

Example (create OpenFOAM Mesh of DEM `dem.asc` in folder `constant/polyMesh`):

```
mkdir constant/polyMesh

txt2mesh.py \
  -xres 100 -yres 200 \
  -p1 "-3609., 219911" -p2 "-4800., 221820."  -p3 "-4100., 222350." -p4 "-2880., 220200." \
  -fillup \
  -offsetx "2000" \
  -offsety "-221000" \
  -butterflyangle 5 \
  -i dem.asc \
  -mesh -o constant/polyMesh;
```

Example (create STL-file of DEM `dem.asc` for, e.g. cfMesh):

```
txt2mesh.py \
  -xres 400 -yres 800 \
  -p1 "-3609., 219911" -p2 "-4800., 221820."  -p3 "-4100., 222350." -p4 "-2880., 220200." \
  -fillup \
  -i dem.asc \
  -walls \
  -offsetx "2000" \
  -offsety "-221000" \
  -o surface.stl;
```

## meshtxt2shape.py

This script processes the CSV-files created by `fafieldsToAscii` and `faMeshToAscii` and produces Esri Shapefiles.

```
usage: meshtxt2shape.py [-h] [-meshFile MESHFILE] [-fieldsFile FIELDSFILE]
                        [-asPoints] [-o O]

optional arguments:
  -h, --help            show this help message and exit
  -meshFile MESHFILE    input meshfile (only for cells) (as csv, use
                        faMeshtoAscii)
```

```
  -fieldsFile FIELDSFILE
                        input fieldsfile (as csv, use faFieldstoAscii)
  -asPoints             export points in area centers instead of cells
  -o O                  output destination
```

See also **foam2shape.py** for a direct export.

## shape2txt.py

This script reads release areas from ESRI-shapefiles and creates the respective dictionary for releaseAreaMapping. See also tutorial `wolfsgrube` (script `runSim.sh` ).

```
usage: shape2txt.py [-h] [-i I] [-o O] [-f F] [-v V] [-x0 X0] [-y0 Y0]
                    [-z0 Z0] [-dfdx DFDX] [-dfdy DFDY] [-dfdz DFDZ]
                    [-normal NORMAL] [-d D] [-offsetx OFFSETX]
                    [-offsety OFFSETY]

optional arguments:
  -h, --help        show this help message and exit
  -i I              input shapefile
  -o O              write release file
  -f F              field name
  -v V              release area value
  -x0 X0            reference point x
  -y0 Y0            reference point y
  -z0 Z0            reference point z
  -dfdx DFDX        increase of f with x
  -dfdy DFDY        increase of f with y
  -dfdz DFDZ        increase of f with z
  -normal NORMAL    project to normal from vertical (yes/no)
  -d D              default value
  -offsetx OFFSETX  offset mesh by x m
  -offsety OFFSETY  offset mesh by y m
```

## foam2shape.py

This script directly reads OpenFOAM meshes and results and exports them as ESRI shapefiles. If this tool has problem reading OpenFOAM files, you should try faMeshToAscii and fafieldsToAscii in combination with meshtxt2shape.py.

```
usage: foam2shape.py [-h] [-case CASE] [-fields FIELDS] [-asPoints] [-o O]
                     [-offsetx OFFSETX] [-offsety OFFSETY]

optional arguments:
  -h, --help        show this help message and exit
  -case CASE        OpenFOAM case folder
  -fields FIELDS    input all fieldname (separated by space)
  -asPoints         export points in area centers instead of cells
  -o O              output destination
  -offsetx OFFSETX  offset mesh by x m
  -offsety OFFSETY  offset mesh by y m
```

## paraview_exportShapefile.py

This script is intended to use within ParaView in combination with the `Contour` filter. The script will export the

(currently selected) contour as ESRI-shapefile. Filename can be selected. Tested for paraview 5.0.1.

# User Selectable Models

## Friciton models

The friction model describes the friction between flowing mass (water body, avalanche) and the base.

It has to be set in the file `constant/transportProperties` (see below).

Friction models can be found in the folder `faSavagehutterfoam/frictionModels` .

To implement a new friction model, copy an exisitng one, rename it and modify it.

Currently there are the following friction models available:

### Darcy-Weisbach

`DarcyWeisbach`

Weisbach (1845) (wikipedia.org):

$$|\boldsymbol{\tau}| = C_f^2 \, \rho \, g \, |\mathbf{u}|^2$$

Parameters (see also tutorial `dresslersdambreak:` ):

```
frictionModel       DarcyWeisbach;
DarcyWeisbachCoeffs
{
    Cf              Cf [0 -1 2 0 0 0 0] 0.000625;
    g               g [0 1 -2 0 0 0 0] 9.81;
}
```

### Manning-Strickler

`ManningStrickler`

Manning (1890) (wikipedia.org):

$$|\boldsymbol{\tau}| = n^2 \, \rho \, g \, \frac{|\mathbf{u}|^2}{h^{1/3.}}$$

Parameters:

```
frictionModel       ManningStrickler;
ManningStricklerCoeffs
{
    n               n [0 0 0 0 0 0 0] 1.0;
    g               g [0 1 -2 0 0 0 0] 9.81;
}
```

### Voellmy

`Voellmy`

Friction model following Voellmy (1955):

$$|\boldsymbol{\tau}| = \mu\, p + \frac{\rho\, g}{\xi}\, |\mathbf{u}|^2$$

Parameters (see also tutorial `wolfsgrube`):

```
frictionModel       Voellmy;
VoellmyCoeffs                                       //parameters for Voellmy
{
    mu              mu [0 0 0 0 0 0 0] 0.25;        //dry friction coefficient
    xi              xi [0 1 -2 0 0 0 0] 10000;      //voellmy turbulence coefficient
}
```

### Kinetic Theory

`kt`

Simplified Kinetic Theory following Rauter et al. (2016).

$$|\boldsymbol{\tau}| = \mu\, p + \frac{\rho\, g}{\chi}\, \frac{|\mathbf{u}|^2}{h^2}$$

Parameters:

```
frictionModel       kt;
"ktCoeffs"                                          //parameters for simplified KT
{
    mu              mu [0 0 0 0 0 0 0] 0.25;        //dry friction coefficient
    chi             chi [0 -1 -2 0 0 0 0] 10000;    //turbulent friction coefficient
}
```

### mu(I)

`muI`

Popular mu(I) following Jop et al (2006) see also wikipedia.org

Shear rate at the base following Bagnold Profile:

$$\dot{\gamma} = \frac{5}{2}\, \frac{|\mathbf{u}|}{h}$$

Inertia number:

$$I = \frac{\dot{\gamma}\, d}{\sqrt{p/\rho_p}}$$

Friction coefficient depending on inertia number:

$$\mu = \mu_s + \frac{\mu_2 + \mu_s}{I_0/I + 1}$$

Basal friction:

$$|\boldsymbol{\tau}| = \mu(I)\, p$$

Parameters (see also tutorial `simpleslope`):

```
frictionModel       MuI;
MuICoeffs
{
    d               d [ 0 1 0 0 0 0 0] 0.005;        //particle diameter
    rho_p           rho_p [ 1 -3  0 0 0 0 0 ] 2500.; //particle density
    mu_s            mu_s [0 0 0 0 0 0 0 ] 0.38;      //friction coefficient (low limit)
    mu_2            mu_2 [0 0 0 0 0 0 0 ] 0.65;      //friction coefficient (high limit)
    I_0             I_0 [0 0 0 0 0 0 0 ] 0.30;       //reference inertia number
}
```

### Poliquen Forterre (2008)

`PoliquenForterre`

First mu(I)-rheology following Pouliquen & Forterre (2002). Similar to `muI`, however with the parametrisation of Pouliquen & Forterre (2002). See also Johnson and Gray, (2011).

$$Fr = \frac{|\mathbf{u}|}{\sqrt{h\, g_n}}$$

$$h_{stop} = h\,\frac{\beta}{Fr}$$

$$\mu_{stop} = \tan(\zeta_1) + \frac{\tan(\zeta_2) - \tan(\zeta_1)}{1 + h_s/L}$$

$$\mu_{start} = \tan(\zeta_3) + (\tan(\zeta_2) - \tan(\zeta_1))\,\exp(-h_s/L)$$

$$\mu = \left(\frac{Fr}{\beta}\right)^{\gamma}(\mu_{stop} - \mu_{start}) + \mu_{start}$$

$$|\boldsymbol{\tau}| = \mu(I)\,p$$

Parameters:

```
frictionModel       PoliquenForterre;
PoliquenForterreCoeffs
{
    L               L [ 0 1 0 0 0 0 0 ] 0.010;
    zeta1           zeta1 [ 0 0 0 0 0 0 0 ] 21;
    zeta2           zeta2 [ 0 0 0 0 0 0 0 ] 30.7;
    zeta3           zeta3 [ 0 0 0 0 0 0 0 ] 22.2;
    beta            beta [ 0 0 0 0 0 0 0 ] 0.136;
    gamma           gamma [0 0 0 0 0 0 0] 1e-3;
}
```

## Entrainment models

The entrainment model has to be set in the file `constant/transportProperties` (see below).

Entrainment models can be found in the folder `faSavagehutterfoam/entrainmentModels`. Currently there are the following available:

### No entrainment

`entrainmentOff`

choose to turn off entrainment.

### Front

`Front`

Simple front entrainment. Entrainment of the total mountain snow cover within a cell is triggered when $h > h_{trigger}$.

Parameters:

```
entrainmentModel    Front;
FrontCoeffs
{
    htrigger        htrigger [ 0 1 0 0 0 0 0 ] 0.01;
}
```

### Erosionenergy

`Erosionenergy`

Entrainment model following SamosAT, see e.g., Rauter et al. (2016).

The entraiment rate $\dot{q}$ is calculated as

$$\dot{q} = \frac{\boldsymbol{\tau} \cdot \mathbf{u}}{e_b}$$

Parameters:

```
entrainmentModel    Erosionenergy;
ErosionenergyCoeffs
{
    eb              eb [0 2 -2 0 0 0 0] 11500;      //specific erosion energy
}
```

### Issler (2014)

`IsslerFC`

Entrainment model following frictional-collisional apporach of Isser (2014) (Apply carefully, not tested in depth).

### Medina (2008)

`Medina`

Entrainment model following the apporach of Medina (2008) (Apply carefully, not tested in depth).

## Deposition models

The deposition model has to be set in the file `constant/transportProperties` (see below).

Deposition models can be found in the folder `faSavagehutterfoam/entrainmentModels` (because of the similarity to entrainment). Currently there is one deposition model available:

### No deposition

`depositionOff`

Choose to turn off deposition.

### Stopingprofile

`Stopingprofile`

Deposition model presented at OpenFOAMWorkshop 12 in Exeter. Otherwise unpublished/unreleased. Based on a decelerating Bagnold profile (Apply carefully, tested but unreleased/unreviewed).

## Function Objects

Some function objects to extend the solvers cabilities are availabe. For general information on function objects in OpenFOAM, see OpenFOAM User Guide.

### Peak dynamic pressure

`dynamicpressure`

Calculates the dynamic pressure, defined as

$$p_{\mathrm{d}}(\mathbf{x}) = \max_{t} \left( \rho \left| \overline{\mathbf{u}}(\mathbf{x}, t) \right|^2 \right)$$

in every timestep and writes the peak dynamic pressure (maximum dynamic pressure over time) to harddisk. This is an important result for many practical applications, see, e.g. Rauter et al. (2016). To calcualate the peak dyanmic pressure, add the following code to `system/controlDict`

```
functions
(
    pd
    {
        type            "dynamicpressure";
        functionObjectLibs( "libfamfunctions.so" );

        outputControl   outputTime;                    //see OpenFOAM user guide
        rho             rho [ 1 -3  0 0 0 0 0 ] 200.;  //density
        hmin            hmin [ 1 0 0 0 0 0 0 ] 0.01;   //threshold for flow thickness

    }
);
```

### Radar simulation

`radar`

Simulates an avalanche radar similar to GEODAR (Köhler et al. 2016). To use this function object, add the following code to `system/controlDict`

```
functions
(
    radar_sum_velocity
    {
        type            "radar";
        functionObjectLibs( "libfamfunctions.so" );

        outputControl   outputTime;                    //see OpenFOAM user guide
```

```
        tableOutPut     "radar.csv";                    //name of output file
        intensity       "sum_velocity";                 //method of calculating intensity
        position        (595249 126797 1487);           //position of the radar
        hmin            0;                              //threshold for flow thickness
        xmin            0;                              //nearest rangegate
        xmax            2500;                           //furthest rangegate
        deltaX          10;                             //distance between rangegates
        log             on;                             //write Log to Stdout
    }
);
```

### Secondary slab release

`slabs`

This function obejct can be used to release scondary slabs at a specific point in time. To use this function object, add the following code to `system/controlDict`

```
functions
(
    slabs
    {
        type            "slabs";
        functionObjectLibs( "libfamfunctions.so" );

        slabs                                           //List of slabs
        (
            slab1                                       //first slab
            {
                vertices                                //polygon defining the slab area
                (
                    (593393.82   127694.27   0)       //list of points,
                    (593391.56   127690.3    0)       //defining a polygon
                    (593400.63   127682.64   0)       //for the slab
                );
                releasetime 7.0;                        //release time of the slab
            }

            slab2                                       //second slab
            {
                vertices
                (
                    (593578.35   127618.5    0)
                    (593600.54   127610.85   0)
                    (593639.56   127598.6    0)
                );
                releasetime 20;
            }
        );

        outputControl     outputTime;
    }
);
```

### Energy and mass conservation

`totalenergy`

Calculates the total energy in the system. To use this function object, add the following code to `system/controlDict`

```
functions
(
    totalEnergy
    {
        type            "totalenergy";
        functionObjectLibs( "libfamfunctions.so" );

        outputControl   outputTime;                    //see OpenFOAM user guide
        rho             rho [ 1 -3  0 0 0 0 0 ] 200.;  //density
        tableOutPut     "energy.csv";                  //name of output file
        log             on;                            //write Log to Stdout
    }
);
```

### Momentum conservation

`totalmomentum`

Calculates the total momentum in the system. To use this function object, add the following code to `system/controlDict`

```
functions
(
    totalMomentum
    {
        type            "totalenergy";
        functionObjectLibs( "libfamfunctions.so" );

        outputControl   outputTime;                    //see OpenFOAM user guide
        rho             rho [ 1 -3  0 0 0 0 0 ] 200.;  //density
        log             on;                            //write Log to Stdout
    }
);
```

# Solver Settings

## Simulation Time, Timesteps

Simulation time and time stepping controls can be found in the file `system/control`. See OpenFOAM User Manual for details. Most important settings:

- `endTime` : Time until the solver runs
- `writeInterval` : Intervalls at which results are saved
- `maxCo` : Maximum Courant-number. Recommended value is 1.

## Initial and Boundary Conditions

Initial and boundary conditions can be set in the files `0/h`, `0/Us` and `0/hentrain`, similar as in other OpenFOAM

solver. The tool `ReleaseAreaMapping` can be used to create appropriate initial conditions.

## Transport Properties

Most physical constants can be found in `constant/transportProperties`. Example:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}

pressureFeedback    off;                            //turn curvature and lateral pressure inte
raction on/off
explicitDryAreas    off;                            //eliminate dry cells in the linear system

rho                 rho [ 1 -3  0 0 0 0 0 ] 200.;   //density
xi                  xi [ 0 0 0 0 0 0 0] 1;          //shape factor is bagnold

hmin                hmin [ 0 1 0 0 0 0 0] 1e-6;     //binding the hight
u0                  u0 [ 0 1 -1 0 0 0 0] 1e-7;      //tolerance for velocity
tau0                tau0 [ 0 2 -1 0 0 0 0] 0;       //artifical viscosity
h0                  h0 [ 0 1 0 0 0 0 0] 1e-4;       //hight threshold for friction models

frictionModel       Voellmy;                        //chose your friction model
entrainmentModel    Erosionenergy;                  //chose your entrainment model
depositionModel     depositionOff;                  //chose your deposition model


VoellmyCoeffs                                       //parameters for friction model
{                                                   //herein Voellmy fiction model
    mu              mu [0 0 0 0 0 0 0] 0.25;        //dry friction coefficient
    xi              xi [0 1 -2 0 0 0 0] 10000;      //voellmy turbulence coefficient
}

ErosionenergyCoeffs                                 //parameters for entrainment model
{
    eb              eb [0 2 -2 0 0 0 0] 10000;      //specific erosion energy
}
```

## Numerical Schemes

See `system/faSchemes` and the OpenFOAM User Manual. Note that this solver is based on the Finite Area Method. Therefore, the numerical schemes are found in the file **fa**Schemes (instead of **fv**Schemes).

## Numerical Solver

See `system/faSolution` and the OpenFOAM User Manual. Note that this solver is based on the Finite Area Method. Therefore, the numerical solution algorithms are found in the file **fa**Solution (instead of **fv**Solution).

# Tutorials

All tutorials contain a `Allrun` script which will conduct all steps required to run a simulation.

## stokersdambreak

Example from Rauter and Tukovic (2018), section 6.1.1. This example demonstrates results for Stokers dam break example.

## rittersdambreak

Example from Rauter and Tukovic (2018), section 6.1.2. This example demonstrates results for Ritters dam break example.

## dresslerdambreak

Example from Rauter and Tukovic (2018), section 6.1.3. This example demonstrates results for Dresslers dam break example.

## centrifugalforce

Example from Rauter and Tukovic (2018), section 6.2. This example demonstrates results for the basal pressure due to centrifugal forces. No friction.

## simpleslope

Example from Rauter and Tukovic (2018), section 6.3. This example demonstrates a popular shallow granular flow test case on a simply curved slope.

## (simpleslope_exp)

Same as simpleslope but set up to work with the (experimental) explicit solver ( `faSavageHutterFoamExp` ).

## wolfsgrube

Example from Rauter, Kofler and Fellin (submitted to GMD).

This tutorial shows the recalculation of a real scale avalanche in Tirol/Austria. Find out more about this avalanche in Fischer et al. (2015).
This tutorial applies the solver and some tools of this package. Moreover it is using some python scripts which can be found in the folder `scripts` .

### Running the example:

- **Preparation.**
  Change into the respective directory and load foam-extend enviroment:

```
cd ~/foam/faSavagehutterfoam/tutorials/finiteArea/faSavageHutterFoam/wolfsgrube
fe40
```

- **Create mesh (using CfMesh).**
  To create the mesh with CfMesh (pMesh), you can run the script:

  ```
  ./runpMesh.sh
  ```

- **Alternative: Create structured mesh directly with pyhton (not recommended)**
  To create the mesh (100 x 200 cells) you can run the script:

  ```
  ./runMesh.sh
  ```

  There are some other scripts ( `runMeshFine.sh` , `runMeshVeryFine.sh` ) to change the cell count. You can also edit one of the scripts to adapt it to your needs.

- **Run the simulation.**
  To run the simulation on a single core, you can run the script:

  ```
  ./runSim.sh
  ```

  To run it using MPI you can use the script:

  ```
  ./runSimMPI.sh
  ```

- **Export results to shape files**
  To export all results to shape files, run the script:

  ```
  ./runGISExport.sh
  ```

## Changing some parameters.

**Most relevant parameters can be set in `constant/transportProperties` . Here you can select friction model, entrainment model, and change parameters.**

## Changing topography, release area.

To change the topography, look into the script `runMesh.sh` and modify it.
To change the release area or the release height, look into the script `runSim.sh` / `runSimMPI.sh` and modify it. The release area is creaed in the second and third step. In here we read the release area from a shape file ( `rawdata/release.shp` ). You can also modify the dictionary `constant/releaseArea` manually.

# Trouble Shooting.

If you encounter stability issues or other problems try one of the following:

- Reduce relaxing factors in `system/faSolution` .
- Increase iterations and decrease tolerances in `system/faSolution` .
- Reduce CoMax in `system/controlDict` .
- Change friction model. We observed some problems in the runout zone with Voellmy. Kinetic theory turned out to be more stable due to the depenence on flow thickness.
- Switch to **first order interpolations**. First order interpolations are much more stable and run with high relaxing factors.
- Limit the flow thickness to small value (hmin) in `constant/transportProperties`

- Contact the author at matthias.rauter@uibk.ac.at