Computing flowing avalanches with AVAC

Christophe Ancey

EPFL, Lausanne, Switzerland

March 2025

1 Installation

1.1 Requirements

AVAC 4 needs Clawpack version 5.11. See the related webpage. It may work with older Clawpack versions, but some functions may fail. It also needs a number of standard python scripts (numpy, matplotlib, IPython, etc.) whose installation is not a problem and a jupyter environnement (provided by an Integrated development environment such as Anaconda or Visual Studio Code).

1.2 Installation and first run

AVAC 4 is made up of a set of files:

- the jupyter notebook `AVAC.ipynb`,
- the python module `module_avac.py`,
- the archive `files.tar.gz`,
- the configuration file `AVAC_parameters.yaml`, and
- the optional jupyter notebook `yaml_export.ipynb`.

The jupyter notebook AVAC assists the user in the installation of all required files. The function

```
install_avac()
```

installs all the python and fortran files required by AVAC. The AVAC parameters are stored in a yaml configuration that can be edited using any text editor; the parameters can also be defined in a python dictionary and exported to the yaml configuration file (e.g., by using the `yaml_export` notebook). The parameters are loaded using the function

```
avac_parameters = import_configuration_files('AVAC_parameters.yaml')
```

The import function checks if the parameters are consistent and load them in the form of a dictionary. The digital elevation model is import using the function:

```
topo_file = reading_raster_file(avac_parameters['topography']['dem'])
```

It is recommended to first check file consistency by using the function:

```
check_raster(avac_parameters['topography']['dem'])
```

The starting areas are imported in the form of a shapefile:

```
starting_polygons = gp.read_file(avac_parameters['topography']['
    starting_areas'])
```

Once all the data have been loaded, AVAC creates a configuration dictionary, which is then exported in the form of a yaml configuration file called `AVAC_configuration.yaml`. The topographic and initial-condition files must be exported to claw-compatible files

```
export_claw_dem('topography.asc',xmin,xmax,ymin,ymax,nbx,nby,altitude)
export_claw_initiation_file(topo_file,zi)
```

The digital elevation model is a transformed into a type-3 claw raster file called `topography.asc`, while the shapefile is converted into a file `init.xyz`.

The user can then run CLAW from the jupyter notebook

```
make_output(avac_parameters,verbosity=False)
```

or directly from a command window:

```
make_output(avac_parameters,verbosity=False)
```

The jupyter notebooks proposes different tools for post-processing data and making plots and animations (see below).

1.3 Configuration file

The configuration file `AVAC_parameters.yaml` has several sections, and inside them all the variables must be filled. Consistency is check when imported in AVAC by the command `import_configuration_files('AVAC_parameters.yaml')`, but not all errors can be pinpointed by the function.

- release:
  - `d0` [float]: assumed release depth d_0 (m) on a flat terrain at the elevation z_{ref}.
  - `correction_slope` [boolean]: if True, then d_0 is corrected depending on local slope. If false, no correction is done.

  - `correction_elevation` [boolean]: if True, then d_0 is corrected depending on local elevation. If false, no correction is done.
  - `z_ref` [float]: reference elevation z_ref (m) at which d_0 has been estimated
  - `gradient_hypso` [float]: hypsometric gradient (m of additional snow per 100 m of elevation). This quantity represents the amount of snow that must be added at a point of elevation $z > z_{ref}$ to account for elevation difference between the reference point and any point. By default, this correction is always zero or positive. The gradient is estimated as the amount of additional snow for any 100-m elevation range.
  - `theta_cr` [float]: value of the critical slope angle θ_{cr} (deg) used in Quervain's model.
  - `nu` [float]: parameter ν used in Quervain's model.
- rheology
  - `model` [string]: the only available model is 'Voellmy' for now (the Coulomb model can be derived by setting $\xi = 0$).
  - `rho` [float]: snow mass density ϱ (kg·m^{-3}). Usually ϱ ranges from 100 kg·m^{-3} to 500 kg·m^{-3}.
  - `mu` [float]: Voellmy's friction coefficient μ.
  - `xi` [float]: Voellmy's friction coefficient ξ (m·s^{-2}).
  - `u_cr` [float]: critical velocity to force avalanche to stop u_{cr} (m·s^{-1}) where locally, its velocity $\bar{u}$ satisfies $\bar{u} < u_{cr}$ and terrain slope θ satisfies $|\theta| < \beta \arctan \mu$.
  - `beta` [float]: slope factor for the stopping criterion.
- topography
  - `dem` [string]: name of file providing the digital elevation model. The file must be a raster file `*.asc`. Three formats are accepted: `qgis`, `claw`, and `grass`. The differences between these formats mainly concern the contents of the header and the grid type. Please see the webpage devoted to this issue.
  - `starting_areas` [string]: name of the shapefile *.shp. The meta-data file *.shx must also be provided. Other meta-data files (e.g., *.prj) are optional
- computation
  - `t_max` [float]: maximum time of computation t_{max} (s).
  - `nb_simul` [float]: number of solutions that must be recorded as `fortq*` files.
  - `cfl_target` [float]: target value of the Courant–Friedrichs–Lewy number (CFL).
  - `cfl_max` [float]: target value of the Courant–Friedrichs–Lewy number (CFL).
  - `refinement` [float]: refinement level for the Adaptive Mesh Refinement algo-

rithm. By default, this value is zero. See the related Clawpack webpage

  - `domain_cell` [float]: number of cells in x- and y-direction for the computational domain.
  - `max_iter` [float]: maximum number of iterations in the claw algorithm.
  - `boundary` [string]: type of boundary conditions. Three possibilities: 'extrap' (extrapolation, default value), 'wall', and 'periodic'. Note that no user-customized condition has been implemented so far.
  - `output_directory` [string]: by default, we use the '_output' directory.

- output
  - `delta_t` [float]: time δt (s) between records of the `fgmax` grid.
  - `output_format` [string]: 'binary32'. Other possibilities: 'ascii' or 'binary64'
  - `verbosity` [float]: maximum time of computation t_{max} (s).

- animation
  - `n_out` [float]: number of frames that must be recorded for creating an animation.
  - `variable` [string]: for the moment, the variable is either 'depth', 'pressure' or 'velocity'. The function `make_output` deals only with one variable:

    ```
    make_output(avac_parameters,verbosity=False)
    ```

    It is possible to create an animation for another variable by setting the dictionary value to another value. For instance, if we want to create showing the velocity evolution, then we can do:

    ```
    avac_parameters['animation']['variable'] = 'velocity'
    make_output(avac_parameters,verbosity=False)
    ```

Once loaded, the configuration file produced a nested dictionnaire. For instance, `avac_parameters['topography']` is a dictionary that provides the values related to the keys 'dem' and 'starting_areas'.

The configuration file can be edited with any text editor or word processor. It can also be created from the jupyter notebook `yaml_export.ipynb`.

1.4 Topographic data used by AVAC

1.4.1 Digital elevation model

AVAC needs to import a raster file in the form of an ASCII file. The file header contains the spatial extent, the cell size(s), and the code related to the "no data" case. The header can be structured differently depending on the provider and the type of grid. Rasters involves structured grids made of square cells, sometimes rectangular cells. There are two ways of representing cells (see Fig. 1):

- by its edges. For instance, in the x-direction, the computational domain starts at x_{min} and extends up to x_{max}. Cell i are identified by its left edge

$$x_{i-1} = x_{min} + (i-1)\delta x$$

and its right edge

$$x_i = x_{min} + i\delta x.$$

We have assumed that there are n in the x-direction and we define the cell size as

$$\delta x = \frac{x_{max} - x_{min}}{n}.$$

By definition, we have $x_0 = x_{min}$ and $x_n = x_{max}$.

- by the cell center positions. For instance, in the x-direction, the cell center (called *node*) of cell i is

$$x_{i-1/2} = x_{min} + (i - 1/2)\delta x.$$

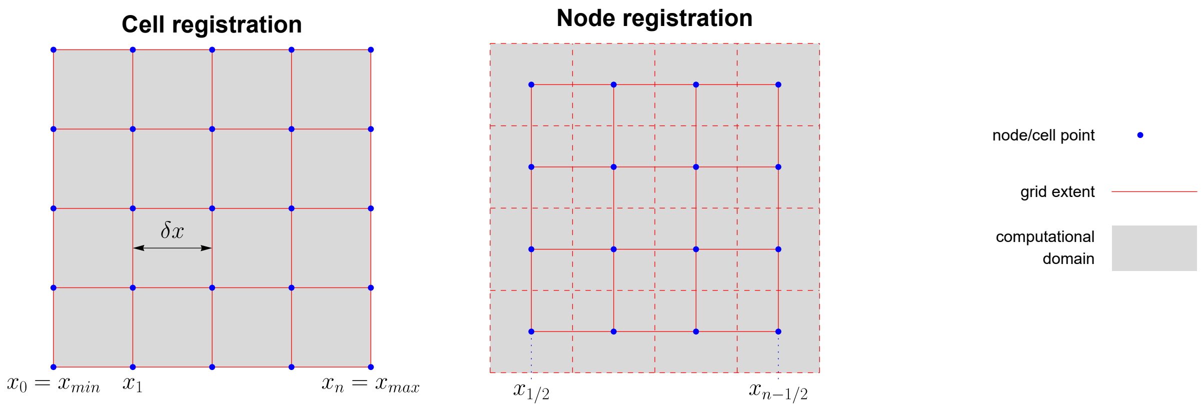

Figure 1: Raster grid registration. We distinguish between cells and nodes. Source : adapted from NOAA.

If in the header, AVAC finds variables such as `xlower` or `xllcorner`, it will deduce that the grid uses cells. On the opposite, if it finds `xllcenter`, then it considers that the grid uses nodes.

If created by Grass, raster files use cell registration, and the computational domain extent is described using variables `north`, `south`, `east`, and `west`. The numbers of cells in the x- and y-direction are given by `cols` and `rows`, respectively.

ESRI raster files provides the coordinates of the lower left corner `xllcorner` and `yllcorner`, respectively. Either the numbers of cells in the x- and y- directions (`ncols` and `nrows`, respectively), either the cell size are indicated. When the cell size is informed, it is usually a scalar value (implying that the cell is square). Occasionally, it may be an array of two numbers (implying that the cell is rectangular). Note that in that case, AVAC takes the first value as the cell size and will consider that the cells are square. The function issues `check_raster(file)` a warning message "more than one value for cellsize" if this case is met.

According to ESRI, rasters provide cell-averaged elevations. By contrast, GeoClaw assumes that elevations are pointwise values, and these values are used to compute cell-averaged elevations (see www.clawpack.org/grid_registration.html). In practice, we believe that most people use rasters as an array of pointwise elevations, and the procedure used by GeoClaw, and thus AVAC, is not a problem.

Recall that finite-volume methods compute cell-averaged values (for depth, momentum, and terrain elevation). Thus, the variables are computed at nodes $(x_{i-1/2}, y_{j-1/2})$. Depending on the type of grid registration, a difference of $(\delta x/2, \delta y/2)$. Functions used by AVAC 4 take this difference into account, but the user must be aware of the potential problem if he uses customized functions.

1.4.2 Initial conditions

AVAC needs a shapefile for the initial condition. This shapefile should only include polygons. If the shapefile is a set of polygons, then AVAC considers that these polygons are the avalanche's starting areas. A consequence is that all these areas will be released at the same initial time.

The release depth d_0 can be set to any constant value, which means that for all starting areas, the depth release is the same. It is possible to take slope and/or elevation dependence into account using Quervain's method, but apart from these corrections, it is not possible in the current version to set d_0 to values that differ from one starting area to another.

1.5 Python functions in AVAC 4

Clawpack provides a large number of python functions that be used to post-process numerical results. AVAC 4 provides its own functions.

1.5.1 Native GeoClaw functions

AVAC uses a number of python and fortran functionalities for the pre- and postprocessing data provided by the library GeoClaw from Clawpack:

- The file `setup.py` is the main script that contains all the data used by GeoClaw. AVAC 4 uses a slightly modified version of the original file to import the AVAC parameters defined in a yaml configuration file called `AVAC_configuration.yaml`.
- GeoClaw produces output files in the form of data files (named `fort.q*`), which tabulates position, depth, and momentum throughout the computational domain at a

given time. The problem with these files is that they involve a series grids, which can be split into series of patches if adaptive mesh refinement is used. ClawPack has python tools called PyClaw. PyClaw defines the `Solution` class in python, which makes it possible to import the `fort.q*` files (see the related webpage www.clawpack.org/pyclaw/solution.html). The function `grid_output_2d` from the library `gridtools` allows the user to extract a variable or a composition of variables and interpolated. The import and extraction processes are slow, and it is usually better to use the alternative below. In some cases, it may be useful to use `grid_output_2d` and the notebook `AVAC.ipynb` provides examples of application.

- In GeoClaw, it is possible to define fixed grids and to record either computed variables (depth, momentum, ground elevation, velocity magnitude) at a given time (`fgout` objects) or the maximum values reached these variables (`fgmax` objects). It is then possible to create objects in a python script and use the function `read_fgmax_grids_data` to import the files. The import processus is much faster than the procedure above. Note also that `fgout` objects can be used to create animations or to plot the solution at a given time.

1.5.2 Functions related to Clawpack and AVAC 4

- `check_claw`: Test if clawpack is installed. If so, it returns the CLAW path. Example:

  ```
  check_claw()
  ```

  produces

  ```
  '/home/ancey/clawpack_git/clawpack'
  ```

- `check_version`:
  - Input: CLAW path [string]
  - Output: array including CLAW version [integer], version identifier [integer]

  The information is extracted from the `setup.py` file in the main CLAW directory. Example:

  ```
  check_version(CLAW)
  ```

  produces

  ```
  [5, 11]
  ```

- `install_avac`. This function extracts the files required by AVAC. By default, it assumes that the files are to be installed in the working directory:
  - Optional argument as input:
    * `verbosity` [Boolean]. It is `False` by default. Execution is verbose. If `True`, further information is provided on the files extracted and their version

    * `path` [string]. By default, it is '.' by default (working directory). If a new path is specified and does not exist, it is created.
    * `archive` [string]. The archive name is 'files.tar.gz' by default. This archive contains:
        · fortran files: `b4step2.f90`, `qinit_module.f90`, `setprob.f90`, and `src2.f90`
        · `Makefile`
        · python scripts: `setrun.py`, `make_fgout_animation`
  - Output: None

The function uncompresses the files in the archive `files.tar.gz` and reads their version. If the uncompressed file does not exist in the working directory, then the function moves it there. If there is already a file with the same name in the working directory, then the function reads the version of the two versions. The version is indicated in a comment line in the header of each file. If the version of the compressed version is more recent than the existing file, then the function updates this file; otherwise, it does not anything. The function finally indicates the AVAC version. Example:

```
install_avac(verbosity = True)
```

produces

```
Installation of AVAC in the working directory: /home/ancey/test_AVAC4
Skipping qinit_module.f90, version 1.0 is up to date.
Skipping setprob.f90, version 1.0 is up to date.
Skipping setrun.py, version 1.0 is up to date.
Skipping src2.f90, version 1.0 is up to date.
Skipping b4step2.f90, version 1.0 is up to date.
Skipping make_fgout_animation.py, version 1.0 is up to date.
Skipping Makefile, version 4.0 is up to date.
=> You are using AVAC version 4.0.
```

- `import_configuration_files`. This function imports the configuration and checks configuration consistency.
  - Input: `file_name` [string]. By default, the file name should be `AVAC_parameters.yaml`
  - Output: a dictionary with AVAC parameters (see § 1.3)

The function can pinpoint a number of potential errors in the parameters but not all. When errors are detected, messages are displayed. Some errors may be fatal. The function provides the names of the raster file and the shapefile, and check their consistency. It then enumerates all the computation values and provides their values. Example:

```
avac_parameters = import_configuration_files('AVAC_parameters.yaml')
```

produces

```
Opening the configuration file AVAC_parameters.yaml...

- I found the DEM file new_raster2.asc.
   File import raises no issue.
- I found the shapefile ZA_1.shp containing the starting areas.
  It seems ok.

Everything looks fine so far...

Configuration file:
* animation_n_out = 45
* animation_variable = depth
* computation_boundary = extrap
* computation_cfl_max = 1
* computation_cfl_target = 0.5
* computation_domain_cell = 2
* computation_dry_limit = 0.01
* computation_max_iter = 100000
* computation_nb_simul = 10
* computation_output_directory = _output
* computation_refinement = 1
* computation_t_max = 90
* output_delta_t = 1
* output_output_format = binary32
* output_verbosity = 0
* release_correction_elevation = True
* release_correction_slope = True
* release_d0 = 1
* release_gradient_hypso = 0.03
* release_nu = 0.2
* release_theta_cr = 30
* release_z_ref = 1800
* rheology_beta = 1.1
* rheology_model = Voellmy
* rheology_mu = 0.22
* rheology_rho = 400
* rheology_u_cr = 0.1
* rheology_xi = 400
* topography_dem = new_raster2.asc
* topography_starting_areas = ZA_1.shp
```

Note that the displayed dictionary has been flattened.

1.5.3 Functions related to raster importation

- `check_raster`. The function checks whether the file `filename` is a raster file.
  - : `filename` [string]: the raster file must be placed in the working directory.

  - `Output`: Boolean. If the Boolean is True, then the file is a raster. The function returns False if import raises problems

The function displays the raster features. Example:

```
check_raster(avac_parameters['topography']['dem'])
```

produces

```
Raster file: new_raster2.asc

File new_raster2.asc exists in the wording directory.

No problem detected in the raster file

Raster features
* The raster format is: esri.
* The grid type is: cell.

----------------------
Feature        Value
----------------------
xmin      1007999.00
xmax      1010001.00
ymin      6469999.00
ymax      6472001.00
nbx             1001
nby             1001
cell size       2.00
```

- `reading_raster_file_features`. This function reads the header of a raster file to determine the spatial extent, the type of grid, and the cell size. The function `reading_raster_file` does some work to read and convert these data into a format compatible with Clawpack
  - Input: raster file [string]
  - Output: a list of 10 variables (`xmin` [float], `xmax` [float], `ymin` [float], `ymax` [float], `nbx` [float], `nby` [float], `cell_size` [float], `dictionnaire` [dictionary], `failure` [array], `remarks` [array])

As indicated in § 1.4, AVAC 4 uses only square cells ($\delta_x = \delta_y$. The first seven variables are used to define the computational domain extent. The function also provides a dictionary containing these seven variables. The variable `failure` is an array of Booleans that informs the user whether the function successfully finds the values related to the five variables x_{min}, y_{min}, n_x, n_y, and δ_x. The variable `remark` is an array of strings that informs the user about potential problems when extracting these values from the raster header. Example:

```
reading_raster_file_features(avac_parameters['topography']['dem'])
```

produces

```
(1007999.0,
 1010001.0,
 6469999.0,
 6472001.0,
 1001,
 1001,
 2.0,
 {'xmin': 1007999.0,
  'xmax': 1010001.0,
  'ymin': 6469999.0,
  'ymax': 6472001.0,
  'nbx': 1001,
  'nby': 1001,
  'cell_size': 2.0,
  'nodata_value': -9999},
 array(['True', 'True', 'True', 'True', 'True'], dtype='<U20'),
 array(['', '', '', '', ''], dtype='<U20'),
 'cell')
```

- `reading_raster_file`. This function reads raw data from the raster file. The file must be an ASCII-format raster. It then converts these data into a format compatible with Clawpack (digital elevation model as an object from the Topography class). See www.clawpack.org/topo.html
  - Input: raster file [string]
  - Output: a Topography object. For more information about the Topography class, see www.clawpack.org/topotools_module.html .

  As indicated in § 1.4, AVAC 4 uses only square cells ($\delta_x = \delta_y$. The first seven variables are used to define the computational domain extent. The function also provides a dictionary containing these seven variables. The variable `failure` is an array of Booleans that informs the user whether the function successfully finds the values related to the five variables x_{min}, y_{min}, n_x, n_y, and δ_x. The variable `remark` is an array of strings that informs the user about potential problems when extracting these values from the raster header. Example:

```
reading_raster_file(avac_parameters['topography']['dem'])
```

  produces

```
<clawpack.geoclaw.topotools.Topography at 0x7fdbb822ff50>
```

- `import_initial_condition`. This function imports the starting-area shapefile and counts the number of starting areas.

  - Input: shapefile path [string]
  - Output: set of polygons [geopandas frame], number of polygons [integer].

Example:

```
starting_polygons, nb_areas = import_initial_condition(
   avac_parameters['topography']['starting_areas'])
```

produces

```
There are 1 starting area(s) in the file ZA_1.shp.
Coordinate Reference System (CRS) of the shapefile: EPSG:2154
EPSG code of the shapefile: 2154
```

The coordinates of the polygon(s) are obtained from attribute `geometry`. Example:

```
starting_polygons.geometry
```

produces

```
0    POLYGON ((1009030.477 6471076.577, 1009048.836...
Name: geometry, dtype: geometry
```

1.5.4 Functions related to raster exportation

- `export_claw_initiation_file`. This function saves the initiation file as topotype-1 file.
  - Input: `topo_file` [Topography object], `zi` [array]
  - optional argument: `filename` [string]. By default, the file name is 'init.xyz'.

Example:

```
export_claw_initiation_file(topo_file,zi)
```

produces

```
Export of initial conditions to file init.xyz.
* maximum initial depth of starting zone  = 1.335519970703125 m
```

- `export_claw_dem`. This function saves the digital elevation model as topotype-3 file.
  - Input: `xmin` [float], `xmax` [float], `ymin` [float], `ymax` [float], `nbx` [float], `nby` [float], `alt` [array]
  - Optional argument: `name_file` [string]

name_file is ‘topography.asc’ by default. If you change it, think of changing in configuration_dictionary. AVAC does not necessarily use the raster file provided by the user, but instead, imports it and transforms into a topotype-3 raster file. The variables xmin, xmax, ymin, ymax, nbx, nby are the grid extent (x_{min}, x_{max}, y_{min}, y_{max}) and the number of columns (number of cells n_x in the x-directions) and rows (number of cells n_y in the y-directions). The variable alt is a two-dimensional array of dimensions $n_y \times n_x$ containing elevation data. Example:

```
export_claw_dem('topography.asc', xmin, xmax, ymin, ymax, nbx, nby,
   altitude)
```

produces

```
Export of DEM to file topography.asc.
```

1.5.5 Functions related to animation

- make_output. This function runs AVAC. Before execution, it also deletes any previous files that might have been produced by a previous run.
  - Input: avac_p [dictionary] : dictionary of configuration parameters
  - Optional argument: verbosity [Boolean]. If this variable is True, then the function displays messages during execution; otherwise, directs the messages to the file ‘avac.log’.

This function creates the output directory called ‘_output’ and place all the output files (fort.q*, fgmax*, etc.) there. It also creates a log file ‘avac.log’ in the working directory. Example:

```
make_output(avac_parameters,verbosity=False)
```

produces

```
I will make an AVAC computation.
Times: from t = 0 to t = 90 s with a time step dt = 9.0 s.
Makefile:72: avertissement : surchargement de la recette pour la
   cible « all »
/home/ancey/clawpack_git/clawpack/clawutil/src/Makefile.common:323:
   avertissement : ancienne recette ignorée pour la cible « all »
rm -f xgeoclaw                    b4step2.f90.html qinit_module.f90.
   html setprob.f90.html src2.f90.html  make_fgout_animation.py.html
   module_avac.py.html setrun.py.html adjoint.data.html amr.data.html
    claw.data.html dtopo.data.html fgmax_grids.data.html fgout_grids.
   data.html flagregions.data.html friction.data.html gauges.data.
   html geoclaw.data.html multilayer.data.html qinit.data.html
   refinement.data.html regions.data.html setprob.data.html surge.
   data.html topo.data.html   Makefile.html
```

```
rm -f .data .output .plots .htmls
Computation is successful.
```

- `export_raster`. The function export a table of results to a raster file (ESRI format).
  - Input: `fname` [string] name of the raster file, `tableau` [array] table of values to be exported, `xll` [float] value of x_{min} (lower corner abscissa), `yll` [float] value of y_{min} (lower corner ordinate), `cellsize` [float] value of δ_x (cell size).
  - Optional argument: `ndata` is the no-data value. It takes the default value −9999.

  Example: if we want to export the maximum avalanche depth, we first import the fixed-grid data and deduce the flow depth (h). We create a masked array to remove zero values and then export the resulting array:

```
# Read fgmax data:
fgno = 1
fg = fgmax_tools.FGmaxGrid()
fg.read_fgmax_grids_data(fgno)
fg.read_output(outdir=outdir)
depth    = ma.masked_where(fg.h<0.001, fg.h )
export_raster('depth_max.asc',depth.filled(),xmin,ymin,
   avac_parameters['computation']['domain_cell'],ndata=-9999)
```

- `make_animation`. This function creates an animation. The variable shown by the animation has been defined in the configuration dictionary by the variable `avac_parameters['animation']['variable']`.
  - Input: `avac_p` [dictionary] : dictionary of configuration parameters
  - Optional argument: `verbosity` [Boolean]. If this variable is True, then the function displays messages during execution; otherwise, directs the messages to the file ‘animation.log’.

  This function creates two output files: an mp4 animation and an html embedding the animation (plus some extra buttons). It also creates a log file ‘animation.log’ in the working directory. Example:

```
make_animation(avac_parameters,verbosity=False)
```

  produces

```
I will make an animation for the depth variable.
Times: from t = 0 to t = 90 s with a time step dt = 2.0 s.
```

1.5.6 Lambda functions related to data extraction

- `fn_eta`,

- `fn_sol`,
- `fn_h`

1.5.7 Functions related to Quervain's

- `correctingFactor1`
- `correctingFactor2`

1.6 History of development

- AVAC 1: developed in 1992 on the basis of Jean-Paul Vila's code (solver of the Saint-Venant equations based on an HLL algorithm). Written in Fortran 77. The model used a Bingham rheology.
- AVAC 2: creation of an interface using Mathematica functionalities in 1999 for pre- and postprocessing. Rheology considered: Voellmy or Coulomb. In 2002, a module (AERO) was created to compute powder-snow avalanches.
- AVAC 3: implementation of TsunamiClaw routines (developed by David George) in 2009. Since 2017, AVAC has used the GeoClaw fortran routines for solving the Saint-Venant equations on sloping terrain. The interface involved Mathematica or Grass functions.
- AVAC 4: AVAC now uses Clawpack and GeoClaw 5.11.

AVAC 4 is not compatible with AVAC 3 (or earlier versions). It has been substantially rewritten.

2 Equations solved by AVAC and numerical solutions

2.1 Governing equations

AVAC solves the two-dimensional shallow-flow equations (also called *Saint-Venant equations*) on an irregular topography:

$$\begin{aligned}
\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(hv) + \frac{\partial}{\partial y}(hv) &= 0, && (1)\\
\frac{\partial}{\partial t}(hu) + \frac{\partial}{\partial x}(hu^2) + \frac{\partial}{\partial y}(huv) + gh\frac{\partial h}{\partial x} &= -gh\frac{\partial z_b}{\partial x} - \frac{\tau_{b,x}}{\varrho}, && (2)\\
\frac{\partial}{\partial t}(hv) + \frac{\partial}{\partial y}(huv) + \frac{\partial}{\partial y}(hv^2) + gh\frac{\partial h}{\partial y} &= -gh\frac{\partial z_b}{\partial y} - \frac{\tau_{b,y}}{\varrho}, && (3)
\end{aligned}$$

where $\boldsymbol{u}(x,\, y,\, t) = (u,\, v)$ denotes the velocity field at point $(x,\, y)$ and at time t. Flow velocity is depth-averaged across the flow depth $h(x,\, y,\, t)$. We work in a Cartesian frame: x refers to the horizontal axis, while y is the axis normal to x. The vertical axis is denoted by z. The bed topography is a function $z_b(x,\, y)$. Avalanche density ϱ is assumed to be constant. The avalanche is subject to the gravitational forces (g gravitational acceleration) and bed friction $\boldsymbol{\tau} = (\tau_{b,x},\, \tau_{b,y})$ along z_b.

In AVAC, the shallow-flow equations are closed using the Voellmy law:

$$\boldsymbol{\tau} = \mu\varrho g h\frac{\boldsymbol{u}}{|\boldsymbol{u}|} + \varrho g\frac{\boldsymbol{u}|\boldsymbol{u}|}{\xi}, \tag{4}$$

where μ and ξ are friction coefficients reflecting Coulomb friction and turbulent dissipation, respectively (?). From a dimensional viewpoint, μ has no physical unit, whereas ξ is homogeneous to acceleration (thus m/s^2). The Coulomb model can also be used (??)

$$\boldsymbol{\tau} = \mu\varrho g h\frac{\boldsymbol{u}}{|\boldsymbol{u}|}. \tag{5}$$

If the Coulomb model can be seen as as asymptotic limit of the Voellmy model when $\xi \to 0$, it is a degenerate case from the mathematical standpoint (taking the limit $\xi \to 0$ causes trouble in Eq. (4)), and its treatment requires to modify the AVAC code.

The governing equations are supplemented with initial conditions. We assume that the avalanche's starting area is a surface $\mathcal{S}$ of the bed surface z_b for which:

$$\begin{aligned}
h(x,\, y,\, 0) &= h_0, && (6)\\
\boldsymbol{u}(x,\, y,\, 0) &= 0, && (7)
\end{aligned}$$

where h_0 is the initial flow depth. The model ignores any mass variations due to snow entrainment. The avalanche can see its initial volume decrease when the friction forces exceed the inertial and gravitational forces.

To solve the governing equations (1)–(7), we use the algorithm developed by **?**. For technical reasons (detailed below), we assume that the local velocity drops to zero where it is lower than a threshold and the local bed slope is lower than μ:

$$\boldsymbol{u} = 0 \text{ if } |\boldsymbol{u}| \leq u_* \text{ and } \tan\theta \leq \beta\mu, \tag{8}$$

where $\tan\theta = |-\nabla z_b|$ is the local slope, u_* is the velocity threshold, and β is a parameter.

2.2 Historical background

The first analytical avalanche-dynamics model was proposed in the early 1920s by Paul Mougin (**?**), who assumed that avalanches behaved like sliding blocks subject to Coulomb friction. An avalanche accelerates indefinitely (i.e., without reaching a steady state) as long as the slope of the natural terrain is greater than the friction coefficient μ. When it is no longer the case, the avalanche decelerates and comes to a halt.

In the 1950s, Adolf Voellmy added turbulent friction to Coulomb friction (**?**). In that case, if the slope is sufficiently long, the avalanche reaches a constant asymptotic velocity

$$\boldsymbol{u}_\infty|\boldsymbol{u}_\infty| = h\xi\left(\nabla z_b - \mu\frac{\boldsymbol{u}}{|\boldsymbol{u}|}\right). \tag{9}$$

The following form of this equation has been widely used by engineers:

$$u_\infty = \sqrt{\xi d\,(\sin\theta - \mu\cos\theta)}, \tag{10}$$

where $d = h\cos\theta$ is the flow thickness (normal to bed slope). This equation is structurally close to Equation (9), but it involves a slightly different measure of flow depth. Indeed, Voellmy implicity considered a curvilinear frame, for which depth is measured normal to the ground, whereas we use a Cartesian framework. Although the curvilinear frame can be considered more physical, it leads to governing equations that are more complex, and thus more difficult to solve numerically. As the Voellmy law is empirical and only partially supported by field evidence, using a Cartesian frame is sufficient. Note that field measurements show that the Coulomb law performs better than the Voellmy model at predicting avalanche velocities (**?**). As the empirical formulation of the Voellmy model has been widely used by engineers for decades (**???**), this is the model implemented in AVAC. The Coulomb model is obtained by taking $\xi \to \infty$.

The first avalanche-dynamics model based on Voellmy's law was suggested by Bruno Salm in the 1960s (**?**). In the 1970s, Soviet researchers further developed the idea, in particular by using the analogy between floods and avalanches to propose the Saint-Venant equations (1)–(3) to model avalanche motion (**??**).

As Equations (1)–(7) are nonlinear, there are few cases for which an analytical solution exists. It is therefore necessary to solve these equations numerically. Arguably the first numerical avalanche-dynamics model was the one proposed at the end of the 1970s by Gérard

Brugnot and Rémi Pochat, based on the finite difference method (?). This type of numerical scheme is not well suited to solving hyperbolic equations such as the Saint-Venant equations. Indeed, discontinuous solutions (*shocks*) can develop, and methods based on finite differences generally fail to capture the dynamics of these shocks. The computed solutions are then flawed, to a varying degree depending on the case considered. This difficulty was overcome with the development of finite-volume methods. Jean-Paul Vila proposed the first numerical code based on a finite volume scheme in the 1980s (???). The AVAC code was developed by C. Ancey in the early 1990s on the basis of Jean-Paul Vila's and Gilbert Martinet's works (???).

The first generation of finite volume models suffered from a number of shortcomings. This has led to the development of a multitude of methods inspired by Godunov's method (???). There is currently no universal method for solving the Saint-Venant equations. Each method has its advantages and disadvantages. For example, HLL schemes – commonly used in commercial codes such as RAMMS (?) – are simple to use, but they do not allow the front to be computed if the bed is dry ahead of the flow front.

To take advantage of advances in the field of finite-volume methods, we decided to use the fortran library called CLAWPACK developed by Randall LeVeque and his collaborators (??). This library includes is a package called GEOCLAW originally developed by David George during his these (???). The Riemann solver included in this package is well-suited to water flows, especially those involving irregular topographies and wet/dry areas. The CLAWPACK library has other advantages such as parallel computing and self-adaptive meshing (AMRCLAW).

3 Numerical solutions

AVAC uses Clawpack/Geoclaw for solving the Saint-Venant equations (1)–(3) in a fixed Cartesian frame. These equations take the tensorial form

$$\frac{\partial}{\partial t}\boldsymbol{U} + \nabla \boldsymbol{F}(\boldsymbol{U}) = \boldsymbol{S}, \tag{11}$$

where $\boldsymbol{U} = (h, hu, hv, z_b)$ is the unknown, and $\boldsymbol{S}$ is the source term. The computation strategy involves first solving the homogenous problem (?) :

$$\frac{\partial}{\partial t}\boldsymbol{U} + \nabla \boldsymbol{F}(\boldsymbol{U}) = 0, \tag{12}$$

then correcting the solution by taking the effect of the source term on $\boldsymbol{q} = h\boldsymbol{u} = (hu, hv)$:

$$\varrho\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{q} = S(\boldsymbol{U}), \tag{13}$$

where $S(\boldsymbol{U})$ takes the following form for the Voellmy model:

$$\begin{aligned} S(\boldsymbol{U}) &= -\mu\varrho g h\frac{\boldsymbol{u}}{|\boldsymbol{u}|} - \varrho\frac{g}{\xi}|\boldsymbol{u}|\boldsymbol{u}, & (14)\\ &= -\mu\varrho g h\frac{\boldsymbol{q}}{|\boldsymbol{q}|} - \varrho\frac{g}{\xi h^2}|\boldsymbol{q}|\boldsymbol{q}. & (15)\\ & & (16) \end{aligned}$$

Let us assume that we have computed the solution $\boldsymbol{q}_*$ to the homogenous equation (12), and we are now seeking the solution at time $k+1$. Using a semi-implicit discretization of (13) leads to

$$\boldsymbol{q}^{k+1} = \boldsymbol{q}^* - \mu g h \mathrm{d}t \frac{\boldsymbol{q}^{k+1}}{|\boldsymbol{q}_*|} - \mathrm{d}t \frac{g}{\xi h^2} |\boldsymbol{q}^*| \boldsymbol{q}^{k+1}, \quad (17)$$

$$\boldsymbol{q}^* = \boldsymbol{q}^{k+1} \left(1 + \frac{\mu g h \mathrm{d}t}{|\boldsymbol{q}_*|} + \frac{g \mathrm{d}t}{\xi h^2} |\boldsymbol{q}^*| \right), \quad (18)$$

$$\boldsymbol{q}^{k+1} = \frac{\boldsymbol{q}^*}{1 + \mathrm{d}t \left(\frac{\mu g h}{|\boldsymbol{q}_*|} + \frac{g}{\xi h^2} |\boldsymbol{q}^*| \right)}. \quad (19)$$

This is the scheme used in `src2.f90` provided with the AVAC files.

4 Implementation

4.1 Prerequisite

The first step is to install the ClawPack library (AVAC is compatible with version 5). See the webpage www.clawpack.org/installing.html.

4.2 Installation

AVAC involves scripts and files based on the GeoClaw library. It thus inherits the GeoClaw architecture, its properties and tools. AVAC includes:

- The `Makefile` file required for compiling and linking the fortran files. If the ClawPack arborescence has been respected and the environment variable $CLAW properly defined, AVAC can be executed from the command window without any further work.
- The `topo.asc` file provides an example of topography file in the ESRI ASCII format. See the webpage //www.clawpack.org/topo.html devoted to this format. It is possible to change the file type and name. Do not forget to changer the `setup.py` file (line 384). By default, AVAC reads a type-2 file called `topo.asc`.
- The parameter file `voellmy.data`. One can change the following parameters: snow density ϱ, Voellmy parameters μ and ξ of Eq. (4), and the two empirical parameters: the velocity threshold u_* and slope parameter β that are used in Eq. (8). The values are read sequentially (do not change their order). If a Coulomb model is used, the variable `Coulomb` should be set at 1 (and used any value for ξ as it will not be used in the `src2.f90` file).

```
# voellmy/coulomb parameters
```

```

300 =: snow_density
800 =: Voellmy xi
0.2 =: Voellmy/Coulomb mu
0.3 =: velocity threshold
1.1 =: beta_slope
0 =: coulomb
```

- The `module_voellmy.f90` module reads the data contained in `voellmy.data` and passes them on the solver.
- The file `src2.f90` is the source term associated with Voellmy's friction (or Coulomb). The solver first solves the homogeneous equations associated with the Saint-Venant equations (1)–(3), then corrects this solution by using an implicit Euler scheme to take friction into account (see the annexe).
- The python script `setrun.py` does the preliminary job before executing the main program. Several lines have to be documented for each case stud (see § 5).
- The file called `initial.xyz` contains the initial flow depths the form of xyz file (x_i, y_i, $h_{0,i}$). How the data are sorted has to comply with the ClawPack rules for type-1 files. See www.clawpack.org/topo.html. The first line corresponds to the north-west corner (grid's upper left corner). **Warning**: the `qinit_module.f90` module (placed in $geoclaw/src/2d/shallow) was modified otherwise the initial depth would have been set at zero when the code was executed. The following lines (near line 129) was changed:

```
if (qinit_type < 4) then
    if (aux(1,i,j) <= sea_level) then
        q(qinit_type,i,j) = q(qinit_type,i,j) + dq
    endif
else if (qinit_type == 4) then
    q(1,i,j) = max(dq-aux(1,i,j),0.d0)
endif
```

  The lines force the code to ignore initial conditions where the elevation is above the see level. To these lines, we added

```
q(1,i,j)=dq
```

  The modified `qinit_module.f90` module is joined to AVAC and is called by the Makefile.

- The `valout.f90` file is a modification to the original file provided by GeoClaw. It enables exportation in the xyz format. Only the level-1 grid is exported. The first two columns are the positions x and y, the other columns are h, hu, hv and $|u| = \sqrt{u^2 + v^2}$.

5 Pre-processing

The *digital elevation model* (DEM) can be provided by a geographic information system such as QGis and Grass, which has the required routines to create and export the `topo.asc` and `initial.xyz` files. For the moment, we also provide the Mathematica notebook called (`TransformationClawpack.nb`), which does the job of exporting the topographic files in the proper format. See the test case "Recoin" and its associated files. In the near future, python scripts will be provided.

Once the topographic files have been placed in the working directory, the `setrun.py` script can be modified. We propose the `AddSetrun.py` file, which contains all variables commonly used (this file is then called by `setrun.py`).

One can also directly modify the default `setrun.py` file provided in Geoclaw. First, one starts with the computational domain's coordinates. Find the following lines (below line 55) :

```
    # ---------------
    # Spatial domain:
    # ---------------

    # Number of space dimensions:
    clawdata.num_dim = num_dim

    # Lower and upper edge of computational domain:
    clawdata.lower[0] = 925720.0
    clawdata.upper[0] = 926700.0

    clawdata.lower[1] = 6451140.
    clawdata.upper[1] = 6452155.



    # Number of grid cells: Coarsest grid
    clawdata.num_cells[0] = 100
    clawdata.num_cells[1] = 100
```

`clawdata.lower[0]` is the lower abscissa x_{min}, while `clawdata.upper[0]` is the upper bound x_{max} of the computational domain in the x-direction. The same holds for the y-direction. Then change

- the cell number in the x direction: `clawdata.num_cells[0]`.
- the cell number in the y direction: `clawdata.num_cells[1]`.

As different grid levels can be used, the information above concerns only the coarsest grid. For the other grids, if one want to add or change them, find the following lines (below line 255) and specify their numbers and refinement factors:

```
    # ---------------
    # AMR parameters:
    # ---------------
    amrdata = rundata.amrdata

    # max number of refinement levels:
    amrdata.amr_levels_max = 3

    # List of refinement ratios at each level (length at least mxnest-1)
    amrdata.refinement_ratios_x = [2,4,4]
    amrdata.refinement_ratios_y = [2,4,4]
    amrdata.refinement_ratios_t = [2,4,4]
```

For instance, three grid levels are here considered in addition to the coarsest grid:

- We pass from the coarsest grid to the first refined grid by taking a refinement factor of 2 for x, y, and t.
- We pass from the first to second grids by taking a refinement factor of 4 for x, y, and t.
- We pass from the second to third grids by taking a refinement factor of 4 for x, y, and t.

Grid refinement can be achieved in different areas, at different times and according to different criteria (see the ClawPack website to see all options). For example, in the following, an automatic mesh refinement is imposed to the entire domain:

```
    # == setregions.data values ==
    regions = rundata.regiondata.regions
    # to specify regions of refinement append lines of the form
    #  [minlevel,maxlevel,t1,t2,x1,x2,y1,y2]
    regions.append([1, 1, 0., 1.e10, 925720,927160., 6451140.,6452155.])
    regions.append([1, 2, 0., 1.e10, 925720,927160., 6451140.,6452155.])
```

The computational time is set below line 100:

```
    # -------------
    # Output times:
    #--------------

    # Specify at what times the results should be written to fort.q files.
    # Note that the time integration stops after the final output time.
    # The solution at initial time t0 is always written in addition.

    clawdata.output_style = 1

    if clawdata.output_style==1:
        # Output nout frames at equally spaced times up to tfinal:
        clawdata.num_output_times = 100
        clawdata.tfinal = 400.0
        clawdata.output_t0 = True  # output at initial (or restart) time?
```

The computational time is set by `clawdata.tfinal`. To specify how many output files have to be saved, the variable `clawdata.tfinal` has to be changed. Here, the computational time is 400 s, and every 4 s (physical time), numerical results are saved in the _output directory.

As of the Clawpack 5.7.0 version, it is possible to export h, hu et hv maxima in a given area. In AVAC, one considers a rectangular area (which generally focuses on the area of interest), and the `AddZoom.py` file has to completed:

```
zooming = False
# spatial domain
x_zoom_min = 1003721
x_zoom_max = 1005139
y_zoom_min = 6499941
y_zoom_max = 6501069
dx = 1.00071
dy = 1.00089
tmax = 90
tstart = 0
dt = 1
```

To activate export, the `zooming` has to be set at true. Then the area coordinates, the resolution (`dx` and `dy`), and times during which export is made, and the time step `dt` between two records.

6 Use

To clean the working directory, use the command

```
make clean
```

To compile the code and execute it, use the command

```
make .output
```

AVAC makes it possible to conduct a sensitivity analysis. The `launcher_random.py` python script draws random values of μ and ξ (together with other AVAC parameters) and stores the final rasters in a directory (called `runs`) for statistical analysis.

7 Post-process

The results are stored in files called `fort.qxxx` and `rasterxxx`, where xxx refers to the output number. Reading these output files requires the use of the python scripts provided in ClawPack or the Mathematica notebook `PostProcessClawPack.nb` provided with AVAC.

Refined (and automated) Mathematica notebooks are under development (contact C. Ancey for further information).