OGGM Documentation

Release 0.1.b

OGGM Developers

March 29, 2016

OGGM

Open Global Glacier Model

The model builds upon Marzeion et al., (2012) and intends to become a global scale, modular, and open source model for glacier dynamics. The model accounts for glacier geometry (including contributory branches) and includes a simple (yet explicit) ice dynamics module. It can simulate past and future mass-balance, volume and geometry of any glacier in a fully automated workflow. We rely exclusively on publicly available data for calibration and validation.

The project is currently in intense development. Get in touch with us if you want to contribute.

CHAPTER 1

Documentation

1.1 What's New

1.1.1 v0.1 (29 March 2016)

Initial release, used to prepare the data submitted to ITMIX (see *ITMIX Experiment 2016*).

This release is the result of several months of development (outside of GitHub for a large part). Several people have contributed to this release:

- **Michael Adamer** (intern, UIBK), participated to the development of the centerline determination algorithm (2014)
- **Kévin Fourteau** (intern, UIBK, ENS Cachan), participated to the development of the inversion and the flowline modelling algorithms (2014-2015)
- **Alexander H. Jarosch** (Associate Professor, University of Iceland), developed the MUSCL-SuperBee model (PR23)
- **Johannes Landmann** (intern, UIBK), participated to the links between databases project (2015)
- **Ben Marzeion** (project leader, University of Bremen), scientific advisor
- **Fabien Maussion** (project leader, UIBK), core developer
- **Felix Oesterle** (Post-Doc, UIBK), develops OGGR and provided the AWS deployment script (PR25)
- **Timo Rothenpieler** (programmer, University of Bremen), participated to the OGGM deployment script (e.g. PR34, PR48), and developed OGGM installation tools
- **Christian Wild** (master student, UIBK), participated to the development of the centerline determination algorithm (2014)

1.2 Installing OGGM

OGGM itself is a pure python package, but it has several dependencies wich are not always trivial to install. The instructions below are self-explanatory and should work on any platform. However, complete beginners should get familiar with Python and its packaging ecosystem before trying to install and run OGGM.

OGGM is tested with the Python versions 2.7, 3.4 and 3.5.

For most users we recommend to use the conda package manager to install the dependencies With conda (all platforms). Linux users and people with experience with pip can follow the specific instructions With virtualenv (linux/debian).

1.2.1 Dependencies

Standard SciPy track:

- numpy
- scipy
- scikit-image
- pillow
- matplotlib
- pandas
- xarray
- joblib

Python 2 support:

- six

Configuration file parsing tool:

- configobj

I/O:

- netcdf4

GIS tools:

- gdal
- shapely
- pyproj
- rasterio
- geopandas

Testing:

- nose

Other libraries:

- salem
- cleo
- motionless (py3)

1.2.2 With conda (all platforms)

Prerequisites

You should have a recent version of git and of conda (either by installing miniconda or anaconda).

Linux users should install a couple of packages (not all of them are required but just to be sure):

```
$ sudo apt-get install build-essential liblapack-dev gfortran libproj-dev gdal-bin libgdal-dev netcd
```

Conda environment

We recommend to create a specific environment for OGGM. In a terminal window, type:

```
conda create --name oggm python=3.4
```

You can of course use any other name for your environment. Don't forget to activate it:

```
source activate oggm
```

(on windows: *activate oggm*)

Packages

Install the packages from the ioos channel:

```
conda install -c ioos geopandas matplotlib Pillow joblib netCDF4 scikit-image configobj nose pyproj r
```

After success, install the following packages from Fabien's github:

```
pip install git+https://github.com/fmaussion/motionless.git
pip install git+https://github.com/fmaussion/salem.git
pip install git+https://github.com/fmaussion/cleo.git
```

OGGM

We recommend to clone the git repository (or a fork if you want to participate to the development):

```
git clone git@github.com:OGGM/oggm.git
```

Then go to the project root directory:

```
cd oggm
```

And install OGGM in development mode:

```
pip install -e .
```

Testing

From the oggm root directory, type:

```
nosetests .
```

1.2.3 With virtualenv (linux/debian)

For Debian / Ubuntu / Mint users only!

Linux packages

For building stuffs:

```
$ sudo apt-get install build-essential python-pip liblapack-dev gfortran libproj-dev
```

For matplolib to work on **Python 2**:

```
$ sudo apt-get install python-gtk2-dev
```

And on **Python 3**:

```
$ sudo apt-get install tk-dev python3-tk python3-dev
```

For GDAL:

```
$ sudo apt-get install gdal-bin libgdal-dev python-gdal
```

For NETCDF:

```
$ sudo apt-get install netcdf-bin ncview python-netcdf
```

Virtual environment

Install:

```
$ sudo pip install virtualenvwrapper
```

Create the directory where the virtual environments will be created:

```
$ mkdir ~/.pyvirtualenvs
```

Add these three lines to the files: ~/.profile and ~/.bashrc:

```
# Virtual environment options
export WORKON_HOME=$HOME/.pyvirtualenvs
source /usr/local/bin/virtualenvwrapper_lazy.sh
```

Reset your profile:

```
$ . ~/.profile
```

Make a new environment with **Python 2**:

```
$ mkvirtualenv oggm_env -p /usr/bin/python
```

Or **Python 3**:

```
$ mkvirtualenv oggm_env -p /usr/bin/python3
```

(Details: http://simononsoftware.com/virtualenv-tutorial-part-2/)

Python Packages

Be sure to be on the working environment:

```
$ workon oggm_env
```

Install one by one the easy stuff:

```
$ pip install numpy scipy pandas shapely
```

For Matplotlib and **Python 2** we need to link the libs in the virtual env:

```
$ ln -sf /usr/lib/python2.7/dist-packages/{glib,gobject,cairo,gtk-2.0,pygtk.py,pygtk.pth} $VIRTUAL_EN
$ pip install matplotlib
```

(Details: http://www.stevenmaude.co.uk/2013/09/installing-matplotlib-in-virtualenv.html)

For Matplotlib and **Python 3** it doesn't seem to be necessary:

```
$ pip install matplotlib
```

Check if plotting works by running these three lines in python:

```
>>> import matplotlib.pyplot as plt
>>> plt.plot([1,2,3])
>>> plt.show()
```

If nothing shows-up, something got wrong.

For **GDAL**, it's also not straight forward. First, check which version of GDAL is installed:

```
$ dpkg -s libgdal-dev
```

The version (10, 11, ...) should match that of the python package. Install using the system binaries:

```
$ pip install gdal==1.10.0 --install-option="build_ext" --install-option="--include-dirs=/usr/include
$ pip install fiona --install-option="build_ext" --install-option="--include-dirs=/usr/include/gdal"
```

(Details: http://tylerickson.blogspot.co.at/2011/09/installing-gdal-in-python-virtual.html)

Install further stuffs:

```
$ pip install pyproj rasterio Pillow geopandas netcdf4 scikit-image configobj joblib xarray
```

And the external libraries:

```
$ pip install git+https://github.com/fmaussion/motionless.git
$ pip install git+https://github.com/fmaussion/salem.git
$ pip install git+https://github.com/fmaussion/cleo.git
```

OGGM and tests

Refer to OGGM above.

1.3 Getting started

The best way to get you started with OGGM is to run the example case study in the Öztal region. You will find a jupyter notebook here or in the `oggm/docs/notebooks` directory.

1.4 Centerlines

1.5 Mass-balance

1.6 Ice dynamics

The glaciers in OGGM are represented by a depth integrated flowline model. The equations for the isothermal shallow ice are solved along the glacier centerline, computed to represent best the flow of ice along the glacier (see for example antarcticglaciers.org for a general introduction about the various type of glacier models).

Here we present the basic physics and numerics of the two models implemented currently in OGGM, the `FluxBasedModel` (homegrown model with a rather simple numerical solver) and the `MUSCLSuperBeeModel` (mass-conserving numerical scheme, see [Jarosch_etal_2013]).

1.6.1 Basics

Let S be the area of a cross-section perpendicular to the flowline. It has a width w and a thickness h and, in this example, a parabolic bed shape.

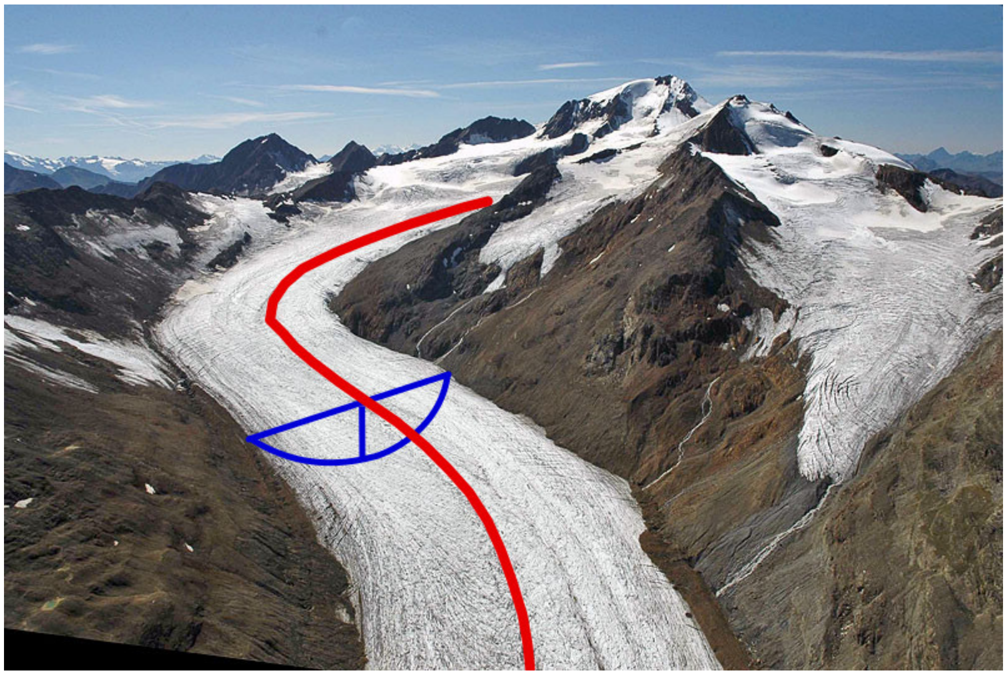


Figure 1.1: Example of a cross-section along the glacier flowline. Background image from http://www.swisseduc.ch/glaciers/alps/hintereisferner/index-de.html

Volume conservation for this discrete element implies:

$$\frac{\partial S}{\partial t} = w\,\dot{m} - \nabla \cdot q$$

where $\dot{m}$ is the mass-balance, $q = uS$ the flux of ice, and u the depth-integrated ice velocity ([Cuffey_Paterson_2010], p 310). This velocity can be computed from Glen's flow law as a function of the basal shear stress τ:

$$u = u_d + u_s = f_d h \tau^n + f_s \frac{\tau^n}{h}$$

The second term is to account for basal sliding, see e.g. [Oerlemans_1997] or [Golledge_Levy_2011]. It introduces an additional free parameter f_s and will therefore be ignored in a first approach. The deformation parameter f_d is better constrained and relates to Glen's temperaturedependent creep parameter A:

$$f_d = \frac{2A}{n+2}$$

The basal shear stress τ depends e.g. on the geometry of the bed [Cuffey_Paterson_2010]. Currently it is assumed to be equal to the driving stress τ_d:

$$\tau_d = \alpha \rho g h$$

where α is the slope of the flowline and ρ the density of ice. Both the `FluxBasedModel` and the `MUSCLSuperBeeModel` solve for these equations, but with different numerical schemes.

1.6.2 Bed thickness inversion

To compute the initial ice thikness h_0, OGGM follows a methodology largely inspired from [Farinotti_etal_2009] but using a different apparent mass-balance (see also: *Mass-balance*) and another calibration algorithm.

The principle is simple. Let's assume for now that we know the ice velocity u along the flowline of our present-time glacier. Then the above equations can be used to compute the section area S out of u and the other ice-flow parameters. Since we know the present-time width w with accuracy, h_0 can be obtained by assuming a certain geometrical shape for the bed.

In OGGM, a number of climate and glacier related parameters are fixed prior to the inversion, leaving only one free parameter for the calibration of the bed inversion procedure: the inversion factor f_{inv}. It is defined such as:

$$A = f_{inv}\, A_0$$

With A_0 the standard creep parameter (2.4e-24). Currently, f_{inv} is calibrated to minimize the volume RMSD of all glaciers with a volume estimation in the GlaThiDa database. It is therefore neither glacier nor temperature dependent and does not account for uncertainties in GlaThiDa's glacier-wide thickness estimations, two approximations which should be better handled in the future.

1.6.3 Flux based model

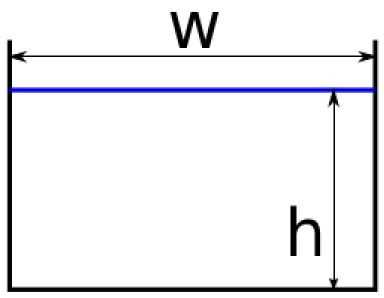
Most flowline models treat the volume conservation equation as a diffusion problem, taking advantage of the robust numerical solutions available for this type of equations. The problem with this approach is that it develops the $\partial S / \partial t$ term to solve for ice thikness h directly, thus implying different diffusion equations for different bed geometries (e.g. [Oerlemans_1997] with a trapezoidal bed).

The OGGM flux based model solves for the $\nabla \cdot q$ term on a staggered grid (hence the name). It has the advantage that the model numerics are the same for any bed shape, but it makes one important simplification: the stress $\tau = \alpha \rho g h$ is always the same, regardless of the bed shape. This doesn't mean that the shape has no influence on the glacier evolution, as explained below.

1.6.4 Glacier bed shapes

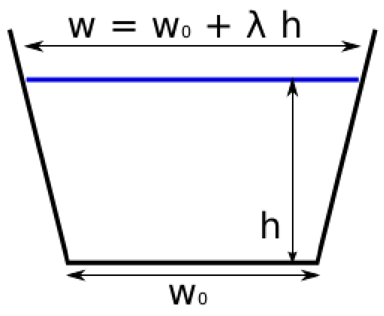
OGGM implements a number of possible bed-shapes. Currently the shape has no direct influence on ice dynamics, but it does influence how the width of the glacier changes with ice thickness and thus will influence the mass-balance $w\,\dot{m}$. It appears that the flowline model is quite sensitive to the bed shape.

VerticalWallFlowline



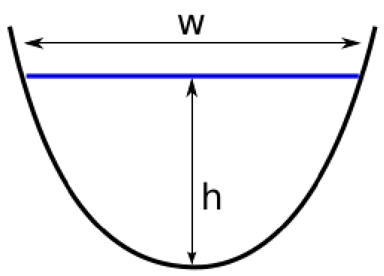
The simplest shape. The glacier width does not change with ice thickness.

TrapezoidalFlowline



Trapezoidal shape with two degrees of freedom. The width change with thickness depends on λ. [Golledge_Levy_2011] uses $\lambda = 1$ (a 45° wall angle).

ParabolicFlowline



Parabolic shape with one degree of freedom, which makes it particulary useful for the bed inversion: if S and w are known:

$$h = \frac{3}{2}\frac{S}{w}$$

The parabola is defined by the single bed-shape parameter $P_s = 4h/w^2$. Very small values of this parameter imply very *flat* shapes, unrealistically sensitive to changes in h. For this reason, the default in OGGM is to use the mixed flowline model.

MixedFlowline

A combination of trapezoidal and parabolic flowlines. If the bed shape parameter P_s is below a certain threshold, a trapezoidal shape is used instead.

1.6.5 MUSCLSuperBeeModel

A shallow ice model with improved numerics ensuring mass-conservation in very steep walls [Jarosch_etal_2013]. The model is currently in development to account for various bed shapes and tributaries and will likely become the default in OGGM.

1.6.6 Glacier tributaries

Glaciers in OGGM have a main centerline and, sometimes, one or more tributaries (which can themsleves also have tributaries, see *Centerlines*). The number of these tributaries depends on many factors, but most of the time the algorithm works well.

The main flowline and its tributaries are all handled the same way and are modelled individually. The difference is that tributaries can transport mass to the branch they are flowing to. Numerically, this mass transport is handled by adding an element at the end of the flowline with the same properties (with, thickness...) as the last grid point, with the difference that the slope α is computed with respect to the altitude of the point they are flowing to. The ice flux is then computed normally and transferred to the downlying branch.

The computation of the ice flux is always done first from the lowest order branches (without tributaries) to the highest ones, ensuring a correct mass-redistribution. The angle between tributary and main branch ensures that the former is not decoupled from the latter. If the angle is positive or if no ice is present at the end of the tributary, no mass exchange occurs.

1.6.7 References

1.7 Glacier working directories

See also: `GlacierDirectory`

The majority of OGGM tasks are so-called "entity tasks". They are standalone operations to be realized on one single glacier entity. These tasks are executed sequentially: they often need input generated by the previous task(s).

In order to avoid complicated chains of arguments, each task will read the input data from a glacier-specific directory and writes its output into the same directory, making the new data available for further computations.

These data files and their names are standardized and listed in the `oggm.cfg` module. If you want to implement your own task you'll have to add an entry to this file too.

1.7.1 Glacier divides

The glacier outlines provided by the RGI are the result of an automated (sometimes manually corrected) delineation procedure. In some cases, the RGI glacier is not the outline of a single glacier (in the dynamical sense) but is the outline of a "glacier complex". OGGM does not (yet) implement a method to divide these glacier complexes into individual glaciers [1], but it does have a framework to deal with user provided glacier divides.

An RGI entity can have one or more "divides", stored as polygons in a separate shapefile (`oggm.cfg.set_divides_db()`). Several OGGM tasks (such as `compute_centerlines()`) can work properly on single glaciers only, as shown on the example below (*left*: with divides, *right*: without).

The divides are handled as sub-directories in the glacier directory: each sub-directory stores the files specific to that divide. All the glaciers have at least one divide ("divide_01"), the divide ID 0 being reserved for the root directory which contains files concerning the entire RGI glacier. The tasks that work on divides only can be recognized by their `div_id=None` keyword argument (they also implement the `divide_task` decorator).

1.7.2 cfg.BASENAMES

This is a list of the files that can be found in the glacier directory or its divides:

apparent_mb.nc The apparent mass-balance data needed for the inversion.

catchment_indices.pkl A list of len n_centerlines, each element conaining a numpy array of the indices in the glacier grid which represent the centerline's catchment area.

centerlines.pkl A list of :py:class:oggm.Centerline instances, sorted by flow order.

climate_monthly.nc The monthly climate timeseries for this glacier, stored in a netCDF file.

dem.tif A geotiff file containing the DEM (reprojected into the local grid).

dem_source.pkl A string with the source of the topo file (ASTER, SRTM, ...).

downstream_line.pkl A shapely.LineString of the coordinates of the downstream line (flowing out of the glacier until the border of the domain) for each divide.

find_initial_glacier_params.pkl

geometries.pkl A `dict` containing the shapely.Polygons of a divide. The "polygon_hr" entry contains the geometry transformed to the local grid in (i, j) coordinates, while the "polygon_pix" entry contains the geometries transformed into the coarse grid (the i, j elements are integers). The "polygon_area" entry contains the area of the polygon as computed by Shapely (it is needed because the divides will have their own area which is not obtained from the RGI file).

glacier_grid.pkl A `salem.Grid` handling the georeferencing of the local grid.

gridded_data.nc A netcdf file containing several gridded data variables such as topography, the glacier masks and more (see the netCDF file metadata).

inversion_flowlines.pkl A "better" version of the Centerlines, now on a regular spacing i.e., not on the gridded (i, j) indices. The tails of the tributaries are cut out to make more realistic junctions. They are now "1.5D" i.e., with a width.

inversion_input.pkl List of dicts containing the data needed for the inversion.

inversion_output.pkl List of dicts containing the output data from the inversion.

inversion_params.pkl Dict of fs and fd as computed by the inversion optimisation.

local_mustar.csv A csv with three values: the local scalars mu*, t*, bias

[1] Kienholz, C., Hock, R., & Arendt, A. a. (2013). A new semi-automatic approach for dividing glacier complexes into individual glaciers. Journal of Glaciology, 59(217), 925–937.

major_divide.pkl A simple integer in the glacier root directory (divide 00) containing the ID of the "major divide", i.e. the one really flowing out of the glacier (the other downstream lines flowing into the main branch).

model_flowlines.pkl List of flowlines ready to be run by the model.

mu_candidates.pkl A pandas.Series with the (year, mu) data.

outlines.shp The glacier outlines in the local projection.

past_model.pkl

past_models.pkl

1.8 ITMIX Experiment 2016

Author: F. Maussion
Last updated: 28.03.2016

OGGM participates to the ITMIX experiment organised by the IACS Working Group on Glacier Ice Thickness Estimation.

The idea is to compare several models able to provide a distributed estimate of glacier ice thickness. The participants can submit their estimations for one (or more) of 18 selected glaciers. The experiment is supposed to be "blind", i.e. the perticipants are not aware of the actual ice thickness of the glaciers they are modelling.

The deadline for the experiment was February 29th. Definitely too early for OGGM, with which we had performed the inversion for the European Alps only. We still didn't want to miss this opportunity and started an intense development phase to make OGGM applicable globally. After quite a lot of work we are now able to provide an estimate for all glaciers except Starbuck in Antarctica. While this surely marks an important step in the development of OGGM, this project again raised many questions and digged a few issues out, since (as you will see below), nothing is easy when doing global scale distributed modelling.

1.8.1 ITMIX preprocessing

The glaciers are heterogeneous: valley glaciers, ice-caps, marine-terminating...

Blue: ITMIX outlines, Black: RGI outlines. White means that ITMIX didn't provide the topography.

The first step that we needed to do is to formalize all this data so that OGGM can deal with it. This turned out to be a bit complicated since the ITMIX data was not standardized:

- we updated the RGI outlines with the ITMIX ones where possible
- for ice-caps, we kept the RGI outlines because OGGM currently needs the "pieces of cake" to compute the centerlines - see the important implications below.
- we decided to to compute the inversion on the OGGM local grids, not on the IMIX maps. This is important because OGGM makes decisions about the grid spacing it uses. Furthermore, the entire workflow is depending on these standardized local maps. We introduced a new routine in the pipeline in order to update the SRTM/ASTER topography with the ITMIX one. This also was not trivial because some ITMIX topographies are stopping directly at the glacier boundary.

1.8.2 OGGM preprocessing

Topography

OGGM uses following DEM data:

- SRTM V4.1 for [60S, 60N] (http://srtm.csi.cgiar.org/)
- GIMP DEM for Greenland (https://bpcrc.osu.edu/gdg/data/gimpdem)
- Corrected DEMs for Svalbard, Iceland, and North Canada (http://viewfinderpanoramas.org/dem3.html)
- ASTER V2 elsewhere

The corrected DEMs where necessary because ASTER data has many issues over glaciers. Take for example the DEM for two glaciers in Iceland:

Note that the hypsometry provided in RGI V5 also contains these errors. While the problems with the right plot are obvious, the glacier on the left (*Dyngjujoekull*) is practically impossible to filter automatically. On the plot below, I show the hypsometry that OGGM computed and the one by Mathias Huss:

Up to a few discrepancies due to projection issues, we both have the problem of non-zero bins below 750 m a.s.l. Fortunately, thanks to the work by Jonathan de Ferranti, these problems are now resolved in OGGM:

There is potential for even better coverage of corrected DEM, but this would require a bit more work (J. de Ferranti's data is not always logically structured).

Calibration data

Another obstacle to global coverage was that the databases required for calibration (WGMS FoG and GlaThiDa) are not "linked" to RGI, i.e. there is no way to know which RGI entity corresponds to each database entry. Thanks to the work of Johannes Landmann at UIBK we now have comprehensive links with global coverage.

Some of the ITMIX glaciers were in the database, I removed them manually for the sake of the experiment ("blind run"):

- WGMS: Kesselwandferner, Brewster, Devon, Elbrus, Freya, Hellstugubreen, Urumqi
- GlaThiDa: Kesselwandferner, Unteraar

These leaves us with 201861 WGMS glaciers with at least 5 years of mass-balance data available for calibration of the mass-balance:

And 133 GlaThiDa glaciers with glacier-wide average thickness estimates. The coverage of GlaThiDa is not very good, which is probably a problem:

1.8.3 Inversion procedure

Refer to the general documentation for details about the inversion procedure.

Here we go directly to the calibration results of the ice-thickness inversion. OGGM currently has only one free parameter to tune for the ice thickness inversion: Glen's creep parameter A. This is very similar to [Farinotti_etal_2009], with the difference that we are not calibrating A for each glacier individually: we tried to do that, but didn't manage (yet).

Land-terminating glaciers

Currently, A is varied until the glacier-wide thickness computed by OGGM minimizes the RMSD with GlaThiDa. We removed all ice-caps and marine-terminating glaciers from the dataset in order to avoid these specific cases (135 glaciers left). Here are the results of the calibration (left: volume-area scaling, right: OGGM):

We can see that OGGM has a slightly lower score than volume-area scaling (VAS). This is due to the presence of a couple of outliers. In particular, the thickest glacier in GlaThiDa and VAS is strinkingly thin in OGGM: this is the Black Rapids glacier in Canada, which is quite well mentioned in the literature because it is a surging glacier. Closer inspection in OGGM reveals that this glacier has one of the lowest mass-balance gradient. CRU precipitation is 849 mm yr ^{1} (after application of the 2.5 correction factor!), which I assume is too low.

We recall here that one central difference between our approach and that of [HussFarinotti_2012] is that we use real climate data to compute the apparent mass-balance, and thus have glacier specific mass-balance gradients. This is a strength but can also become a burden. The mass-balance gradient depends mostly on precipitation, but also on temperature and its seasonal cycle. Here I show the apparent mass-balance gradient in the ablation area of all GlaThiDa glaciers:

Is the MB gradient related to the error OGGM makes in comparison to GlaThiDa?

No not really. And this is similar with all other glacier characteristics I could look at until now. The error that OGGM makes is not easily attributable to specific causes... It it would, that would be great! Indeed, this would allow maybe to define a rule for the calibration factor A. If you have ideas at which parameter to look at, let me know!

VAS vs OGGM

I don't want the calibration to be too altered by the Black Rapids outlier, so I removed it from the calibration set. The plot now looks better, and after this little hack OGGM is even *slightly* better than VAS:

What is really interesting however is that OGGM and VAS are incredibly similar in their dissimilarity with GlaThiDa. So similar that if we plot the two approaches together on a scatter, one could argue that the thousands of lines of code of OGGM really aren't worth the effort ;). I think that I have to talk to David Bahr about this, he will surely be pleased.

A problem with large glaciers?

These results for the reference glaciers availble in GlaThiDa were somehow OK, but the issue with the Black Rapids glacier made me wonder a little bit and I decided to have a closer look. In the figure below I compare the OGGM inversion results with VAS for all glaciers I have at hand for the experiment (ITMIX + WGMS + GlaThiDa, land-terminating, no ice-caps).

After many hours (days?) of searching for a reasonable explanation for this behavior, I had to renounce. I will have to make a test under controlled conditions to see if this is actually a bug in OGGM or if it is something inherent to the methodology.

The problem with ice-caps

This one is easier: ice caps are cut into smaller pieces of cake, instead of being considered as a large glacier. Like VAS, OGGM will also fail since the volume of glaciers is expected to grow with a power law to its area. The flowline methodology followed by OGGM on ice caps is very likely to underestimate the total ice volume.

Marine-terminating glaciers

Using the apparent mass-balance method for calving glaciers will lead to overestimated thicknesses. In the future, OGGM will intend to parametrize mass-flux at calving fronts. For ITMIX, we will consider mass loss by calving for Columbia glacier only. From [Rasmussen_etal_2011], we assume two possible calving rates for Columbia: 4.3 and 8.0 km ^{3} ice eq. a $^{-1}$, to which we arbitrarily add a third case with 12 km ^{3} ice eq. a $^{-1}$. This mass loss at the calving front is distributed over the glacier and added to the apparent mass-balance, resulting in total ice volumes of 250, 273, and 291 km ^{3}, respectively.

1.8.4 Putting all this together

We find an optimised factor for A of 3.22 (i.e. our A is three times larger than the standard of 2.4e-24 [Cuffey_Paterson_2010]). This makes sense since we do not consider the sliding velocity in the inversion, which means that we need an ice which is less stiff to compensate.

The inversion procedure in OGGM is not designed to provide a distributed estimate of glacier thickness: in particular, the glacier widths in OGGM are not always geometrical as in Farinotti et al. (2009): they are also corrected so that the altitude-area distribution of the glacier is preserved.

We show a few examples of the normal inversion procedure in OGGM, which is meant to provide the intput to a flowline model:

Distributed ice thickness

In a final step, we had to somehow interpolate the flowline thicknesses to the ITMIX map. Here again we made the choice to keep as much logic as possible within the standard OGGM framework and defined a new task, `oggm.tasks.distribute_thickness()`. We compute the ice thickness on the OGGM local map and simply interpolate our results on the high resolution ITMIX grid only at the very end.

The distribution works as follows:

- for each pixel, the closest flowlines thicknesses within a 100m altitude range are interpolated using an inverse distance weight
- this value is then corrected with a factor depending on the distance to the glacier outlines
- finally, the thickness is corrected with a factor $1/\alpha^{\frac{N}{N+2}}$ as in Farinotti et al. (2009). Since I had no time to check if this correction works better, I submitted two versions of the interpolation (with or without local slope correction).

An example of the interpolation for Columbia glacier with (left) or without (right) slope correction:

For the ice caps, the interpolation methods lead to similar results. However, it is very likely that the flowline methodology used by OGGM is not working properly.

1.8.5 Conclusions

We were able to provide an estimate of ice thickness for all glaciers except Starbuck in Antarctica. I am less confident about the distributed maps than the total volume estimates. I am also much less confident about the ice-caps (they are probably totally crap) and also less confident about the larger glaciers than the smaller ones.

1.8.6 References

1.9 API reference

Here we will add the documentation for selected modules.

1.9.1 Entity tasks

`define_glacier_region`	Very first task: define the glacier's grid.
`glacier_masks`	Converts the glacier vector geometries to grids.
`compute_centerlines`	Compute the centerlines following Kienholz et al., (2014).
`compute_downstream_lines`	Compute the lines continuing the glacier (one per divide).
`catchment_area`	Compute the catchment areas of each tributary line.
`initialize_flowlines`	Transforms the geometrical Centerlines in the more "physical" InversionFlowlines.
`catchment_width_geom`	Compute geometrical catchment widths for each point of the flowlines.
`catchment_width_correction`	Corrects for NaNs and inconsistencies in the geometrical widths.
`mu_candidates`	Computes the mu candidates.
`prepare_for_inversion`	Prepares the data needed for the inversion.
`volume_inversion`	Computes the inversion the glacier.
`distribute_thickness`	Compute a thickness map of the glacier using the nearest centerlines.
`init_present_time_glacier`	First task after inversion.
`find_inital_glacier`	Search for the glacier in year y0

oggm.tasks.define_glacier_region

`oggm.tasks.`**`define_glacier_region`**(*gdir*, ***kwargs*)

Very first task: define the glacier's grid.

Defines the local glacier projection (Transverse Mercator), chooses a resolution for the grid, and transforms the DEM as well as the RGI shape into it.

Parameters **gdir** : oggm.GlacierDirectory

entity : geopandas entity

the glacier geometry to process

Returns Files writen to the glacier directory:

dem.tif A geotiff file containing the DEM (reprojected into the local grid).

glacier_grid.pkl A `salem.Grid` handling the georeferencing of the local grid.

outlines.shp The glacier outlines in the local projection.

oggm.tasks.glacier_masks

`oggm.tasks.`**`glacier_masks`**(*gdir*, ***kwargs*)

Converts the glacier vector geometries to grids.

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

geometries.pkl A `dict` containing the shapely.Polygons of a divide. The "polygon_hr" entry contains the geometry transformed to the local grid in (i, j) coordinates, while the "polygon_pix" entry contains the geometries transformed into the coarse grid (the i, j elements are integers). The "polygon_area" entry contains the area of the polygon as computed by Shapely (it is needed because the divides will have their own area which is not obtained from the RGI file).

gridded_data.nc A netcdf file containing several gridded data variables such as topography, the glacier masks and more (see the netCDF file metadata).

oggm.tasks.compute_centerlines

`oggm.tasks.`**`compute_centerlines`**(*gdir*, ***kwargs*)

Compute the centerlines following Kienholz et al., (2014).

They are then sorted according to the modified Strahler number: http://en.wikipedia.org/wiki/Strahler_number

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

centerlines.pkl A list of :py:class:oggm.Centerline instances, sorted by flow order.

gridded_data.nc A netcdf file containing several gridded data variables such as topography, the glacier masks and more (see the netCDF file metadata).

oggm.tasks.compute_downstream_lines

`oggm.tasks.`**`compute_downstream_lines`**(*gdir*, ***kwargs*)

Compute the lines continuing the glacier (one per divide).

The idea is simple: starting from the glacier tail, compute all the routes to all local minimas found at the domain edge. The cheapest is "the One".

The task also determines the so-called "major flowline" which is the only line flowing out of the domain. The other ones are flowing into the branch. The rest of the job (merging all centerlines + downstreams into one single glacier is realized by `init_present_time_glacier()`).

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

downstream_line.pkl A shapely.LineString of the coordinates of the downstream line (flowing out of the glacier until the border of the domain) for each divide.

major_divide.pkl A simple integer in the glacier root directory (divide 00) containing the ID of the "major divide", i.e. the one really flowing out of the glacier (the other downstream lines flowing into the main branch).

oggm.tasks.catchment_area

`oggm.tasks.`**`catchment_area`**(*gdir*, ***kwargs*)

Compute the catchment areas of each tributary line.

The idea is to compute the route of lowest cost for any point on the glacier to rejoin a centerline. These routes are then put together if they belong to the same centerline, thus creating "catchment areas" for each centerline.

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

catchment_indices.pkl A list of len n_centerlines, each element conaining a numpy array of the indices in the glacier grid which represent the centerline's catchment area.

oggm.tasks.initialize_flowlines

`oggm.tasks.`**`initialize_flowlines`**(*gdir*, ***kwargs*)

Transforms the geometrical Centerlines in the more "physical" InversionFlowlines.

This interpolates the centerlines on a regular spacing (i.e. not the grid's (i, j) indices. Cuts out the tail of the tributaries to make more realistic junctions.

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

inversion_flowlines.pkl A "better" version of the Centerlines, now on a regular spacing i.e., not on the gridded (i, j) indices. The tails of the tributaries are cut out to make more realistic junctions. They are now "1.5D" i.e., with a width.

oggm.tasks.catchment_width_geom

`oggm.tasks.`**`catchment_width_geom`**(*gdir*, ***kwargs*)

Compute geometrical catchment widths for each point of the flowlines.

Updates the 'inversion_flowlines' save file.

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

inversion_flowlines.pkl A "better" version of the Centerlines, now on a regular spacing i.e., not on the gridded (i, j) indices. The tails of the tributaries are cut out to make more realistic junctions. They are now "1.5D" i.e., with a width.

oggm.tasks.catchment_width_correction

`oggm.tasks.`**`catchment_width_correction`**(*gdir*, ***kwargs*)

Corrects for NaNs and inconsistencies in the geometrical widths.

Interpolates missing values, ensures consistency of the surface-area distribution AND with the geometrical area of the glacier polygon, avoiding errors due to gridded representation.

Updates the 'inversion_flowlines' save file.

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

inversion_flowlines.pkl A "better" version of the Centerlines, now on a regular spacing i.e., not on the gridded (i, j) indices. The tails of the tributaries are cut out to make more realistic junctions. They are now "1.5D" i.e., with a width.

oggm.tasks.mu_candidates

`oggm.tasks.`**`mu_candidates`**`(`*gdir*, ***kwargs*`)`

Computes the mu candidates.

For each 31 year-period centered on the year of interest, mu is is the temperature sensitivity necessary for the glacier with its current shape to be in equilibrium with its climate.

For glaciers with MB data only!

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

mu_candidates.pkl A pandas.Series with the (year, mu) data.

oggm.tasks.prepare_for_inversion

`oggm.tasks.`**`prepare_for_inversion`**`(`*gdir*, ***kwargs*`)`

Prepares the data needed for the inversion.

Mostly the mass flux and slope angle, the rest (width, height) was already computed. It is then stored in a list of dicts in order to be faster.

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

inversion_input.pkl List of dicts containing the data needed for the inversion.

oggm.tasks.volume_inversion

`oggm.tasks.`**`volume_inversion`**`(`*gdir*, ***kwargs*`)`

Computes the inversion the glacier.

If fs and fd are not given, it will use the optimized params.

Parameters **gdir** : oggm.GlacierDirectory

use_cfg_params : bool, default=False

if True, use A and fs provided by the user in the config file and if False, use the results of the optimisation.

Returns Files writen to the glacier directory:

inversion_output.pkl List of dicts containing the output data from the inversion.

oggm.tasks.distribute_thickness

`oggm.tasks.`**`distribute_thickness`**`(`*gdir*, ***kwargs*`)`

Compute a thickness map of the glacier using the nearest centerlines.

This is a rather cosmetic task, not relevant for OGGM but for ITMIX. Here we take the nearest neighbors in a certain altitude range.

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

gridded_data.nc A netcdf file containing several gridded data variables such as topography, the glacier masks and more (see the netCDF file metadata).

oggm.tasks.init_present_time_glacier

`oggm.tasks.`**`init_present_time_glacier`**`(`*gdir*, ***kwargs*`)`

First task after inversion. Merges the data from the various preprocessing tasks into a stand-alone dataset ready for run.

In a first stage we assume that all divides CAN be merged as for HEF, so that the concept of divide is not necessary anymore.

This task is horribly coded and needs work

Parameters **gdir** : oggm.GlacierDirectory

Returns Files writen to the glacier directory:

model_flowlines.pkl List of flowlines ready to be run by the model.

oggm.tasks.find_inital_glacier

`oggm.tasks.`**`find_inital_glacier`**`(`*gdir*, ***kwargs*`)`

Search for the glacier in year y0

Parameters **gdir: GlacierDir object**

div_id: the divide ID to process (should be left to None)

Returns Files writen to the glacier directory:

past_model.pkl

1.9.2 Global tasks

`distribute_climate_data`	Reads the climate data and distributes to each glacier.
`compute_ref_t_stars`	Detects the best t* for the reference glaciers.
`distribute_t_stars`	After the computation of the reference tstars, apply the interpolation to each individual glacie
`optimize_inversion_params`	Optimizes fs and fd based on GlaThiDa thicknesses.

oggm.tasks.distribute_climate_data

`oggm.tasks.`**`distribute_climate_data`**`(`*gdirs*`)`

Reads the climate data and distributes to each glacier.

Generates a NetCDF file in the root glacier directory (climate_monthly.nc) It contains the timeseries of temperature, temperature gradient, and precipitation at the nearest grid point. The climate data reference height is provided as global attribute.

Not to be multi-processed.

Parameters **gdirs: list of oggm.GlacierDirectory objects**

oggm.tasks.compute_ref_t_stars

`oggm.tasks.`**`compute_ref_t_stars`**(*gdirs*)

Detects the best t* for the reference glaciers.

Parameters gdirs: list of oggm.GlacierDirectory objects

oggm.tasks.distribute_t_stars

`oggm.tasks.`**`distribute_t_stars`**(*gdirs*)

After the computation of the reference tstars, apply the interpolation to each individual glacier.

Parameters gdirs: list of oggm.GlacierDirectory objects

oggm.tasks.optimize_inversion_params

`oggm.tasks.`**`optimize_inversion_params`**(*gdirs*)

Optimizes fs and fd based on GlaThiDa thicknesses.

We use the glacier averaged thicknesses provided by GlaThiDa and correct them for differences in area with RGI, using a glacier specific volume-area scaling formula.

Parameters gdirs: list of oggm.GlacierDirectory objects

1.9.3 Classes

`GlacierDirectory`	Organizes read and write access to the glacier's files.

oggm.GlacierDirectory

class `oggm.`**`GlacierDirectory`**(*rgi_entity*, *base_dir=None*, *reset=False*)

Organizes read and write access to the glacier's files.

It handles a glacier directory created in a base directory (default is the "per_glacier" folder in the working directory). The role of a GlacierDirectory is to give access to file paths and to I/O operations in a transparent way. The user should not care about *where* the files are located, but should know their name (see *cfg.BASENAMES*).

A glacier entity has one or more divides. See *Glacier working directories* for more information.

`__init__`(*rgi_entity*, *base_dir=None*, *reset=False*)

Creates a new directory or opens an existing one.

Parameters rgi_entity: glacier entity read from the shapefile

base_dir: path to the directory where to open the directory

defaults to "conf.PATHPATHS['working_dir'] + /per_glacier/"

reset: emtpy the directory at construction (careful!)

Attributes

dir	(str) path to the directory
rgi_id	(str) The glacier's RGI identifier
glims_id	(str) The glacier's GLIMS identifier (when available)
rgi_area_km2	(float) The glacier's RGI area (km2)
cenlon	(float) The glacier's RGI central longitude
rgi_date	(datetime) The RGI's BGNDATE attribute if available. Otherwise, defaults to 2003-01-01
rgi_region	(str) The RGI region name
name	(str) The RGI glacier name (if Available)

Methods

`__init__`(rgi_entity[, base_dir, reset])	Creates a new directory or opens an existing one.
`create_gridded_ncdf_file`(fname[, div_id])	Makes a gridded netcdf file template.
`get_filepath`(filename[, div_id])	Absolute path to a specific file.
`has_file`(filename[, div_id])	
`log`(func[, err])	
`read_pickle`(filename[, div_id])	Reads a pickle located in the directory.
`write_monthly_climate_file`(time, prcp, temp, ...)	Creates a netCDF4 file with climate data.
`write_pickle`(var, filename[, div_id])	Writes a variable to a pickle on disk.

Attributes

`divide_dirs`	list of the glacier divides directories.
`divide_ids`	Iterator over the glacier divides ids.
`grid`	
`n_divides`	Number of glacier divides.
`rgi_area_m2`	

1.9.4 Mass-balance

Mass-balance models in the `oggm.core.models.massbalance` module

`TstarMassBalanceModel`	Constant mass balance: equilibrium MB at period t*.
`BackwardsMassBalanceModel`	Constant mass balance: MB for [1983, 2003] with temperature bias.
`TodayMassBalanceModel`	Constant mass-balance: MB during the last 30 yrs.
`HistalpMassBalanceModel`	Mass balance during the HISTALP period.

oggm.core.models.massbalance.TstarMassBalanceModel

class `oggm.core.models.massbalance.`**`TstarMassBalanceModel`**(*gdir*, *bias=0.0*)
Constant mass balance: equilibrium MB at period t*.

`__init__`(*gdir*, *bias=0.0*)
Instanciate.

Methods

`__init__`(gdir[, bias])	Instanciate.
`get_mb`(heights[, year])	Returns the mass-balance at given altitudes for a given moment in time.
`set_bias`([bias])	

oggm.core.models.massbalance.BackwardsMassBalanceModel

class `oggm.core.models.massbalance.`**`BackwardsMassBalanceModel`**(*gdir*, *use_tstar=False*, *bias=0.0*)

Constant mass balance: MB for [1983, 2003] with temperature bias.

This is useful for finding a possible past galcier state.

`__init__`(*gdir*, *use_tstar=False*, *bias=0.0*)

Instanciate.

Methods

`__init__`(gdir[, use_tstar, bias])	Instanciate.
`get_mb`(heights[, year])	Returns the mass-balance at given altitudes for a given moment in time.
`set_bias`([bias])	

oggm.core.models.massbalance.TodayMassBalanceModel

class `oggm.core.models.massbalance.`**`TodayMassBalanceModel`**(*gdir*, *bias=0.0*)

Constant mass-balance: MB during the last 30 yrs.

`__init__`(*gdir*, *bias=0.0*)

Instanciate.

Methods

`__init__`(gdir[, bias])	Instanciate.
`get_mb`(heights[, year])	Returns the mass-balance at given altitudes for a given moment in time.
`set_bias`([bias])	

oggm.core.models.massbalance.HistalpMassBalanceModel

class `oggm.core.models.massbalance.`**`HistalpMassBalanceModel`**(*gdir*)

Mass balance during the HISTALP period.

`__init__`(*gdir*)

Instanciate.

Methods

`__init__`(gdir)	Instanciate.

Continued on next page

Table 1.10 – continued from previous page

`get_mb`(heights[, year])	Returns the mass-balance at given altitudes for a given moment in time.
`set_bias`([bias])	

CHAPTER 2

Get in touch

- To ask questions or discuss OGGM, send us an e-mail.
- Report bugs, share your ideas or view the source code on GitHub.

CHAPTER 3

License

OGGM is available under the open source GNU GPLv3 license.

CHAPTER 4

About

Status Experimental - in development

License GNU GPLv3

Authors Fabien Maussion, Alexander H. Jarosch, Felix Oesterle, Timo Rothenpieler, Ben Marzeion

See whats-new for a list of all contributors.

Funding Austrian Research Foundation FWF, Projects P22443-N21 and P25362-N26

[Cuffey$_{P}aterson_{2}$010] Cuffey, K., and W. S. B. Paterson (2010). The Physics of Glaciers, ButterworthHeinemann, Oxford, U.K.

[Farinotti$_{e}tal_{2}$009] Farinotti, D., Huss, M., Bauder, A., Funk, M., & Truffer, M. (2009). A method to estimate the ice volume and ice-thickness distribution of alpine glaciers. Journal of Glaciology, 55 (191), 422–430.

[Golledge$_{L}evy_{2}$011] Golledge, N. R., and Levy, R. H. (2011). Geometry and dynamics of an East Antarctic Ice Sheet outlet glacier, under past and present climates. Journal of Geophysical Research: Earth Surface, 116(3), 1–13.

[Jarosch$_{e}tal_{2}$013] Jarosch, a. H., Schoof, C. G., & Anslow, F. S. (2013). Restoring mass conservation to shallow ice flow models over complex terrain. Cryosphere, 7(1), 229–240. http://doi.org/10.5194/tc-7-229-2013

[Oerlemans$_{1}$997] Oerlemans, J. (1997). A flowline model for Nigardsbreen, Norway: projection of future glacier length based on dynamic calibration with the historic record. Journal of Glaciology, 24, 382–389.

[Cuffey$_{P}aterson_{2}$010] Cuffey, K., and W. S. B. Paterson (2010). The Physics of Glaciers, ButterworthHeinemann, Oxford, U.K.

[Farinotti$_{e}tal_{2}$009] Farinotti, D., Huss, M., Bauder, A., Funk, M., & Truffer, M. (2009). A method to estimate the ice volume and ice-thickness distribution of alpine glaciers. Journal of Glaciology, 55 (191), 422–430.

[HussFarinotti$_{2}$012] Huss, M., & Farinotti, D. (2012). Distributed ice thickness and volume of all glaciers around the globe. Journal of Geophysical Research: Earth Surface, 117(4), F04010.

[Rasmussen$_{e}tal_{2}$011] Rasmussen, L. A., Conway, H., Krimmel, R. M., & Hock, R. (2011). Surface mass balance, thinning and iceberg production, Columbia Glacier, Alaska, 1948-2007. Journal of Glaciology, 57(203), 431–440.

Index