The Community Firn Model

***** *This documentation is still a work in progress. Please inform me if you find glaring omissions!*

The Community Firn Model (CFM) is a modular firn-evolution model framework. Its most basic function is to predict the depth/density and depth/age profiles of firn (i.e. functioning as a firn-densification model), but it also includes modules to simulate heat transfer, meltwater percolation and refreezing, water isotope diffusion, firn-air diffusion and advection, and grain growth.

The modular nature of the CFM means that the user can easily choose to run the model using a firn-densification equation from any of a suite of published firn-densification equations (Table XX). The user can also choose which of the optional physics modules (e.g. grain growth) to include with the model run. The model is designed so that different firn-densification equations or modules simulating different physics can be easily integrated into the model.

The CFM is one dimensional and uses a Lagrangian (material-following) grid. The evolution of the density is calculated explicitly (e.g. $\rho_{new} = \rho_{old} + \frac{d\rho}{dt} * dt$). Heat and other (e.g. water isotope, firn air, enthalpy) diffusion is solved using a fully-implicit finite-volume method (Patankar, 1980).

The CFM is coded in Python 3 and can be run on Windows, Linux, and OSX platforms. In addition to the packages included in the Python standard library, the model requires the numpy, scipy, and h5py Python packages to be installed. Additionally, plotting CFM results using Python requires the matplotlib package. Installing Python 3 and the necessary packages is most easily done using a packaged Python distribution such as Anaconda or Enthought Canopy. Explicit instructions on a scientific Python installation are beyond the scope of this document, but tutorials can be readily found online.

1 Running the CFM

The CFM is mostly easily run from the command line, though it can alternatively be run from an integrated development environment (IDE) such as Spyder (included with the Anaconda Python distribution). To run the model, the user must set up a .json-formatted configuration file that specifies the settings for a particular model run (section 5.XX). Here, we generically use *config.json* as the name of that configuration file.

The model is run by calling *main.py* and specifying the name of the configuration file:

>>> python main.py config.json -n

If the results folder (specified in config.json) already exists and contains the results of a spin-up run (the spin-up file name is typically *CFMspin.hdf5*), the model will not run the spin-up routine again. If the user wishes to include the spin-up run, he/she should add the –n at the end of the above command to force the spin-up to run; if he/she wished to omit the spin up run, the –n should be omitted.

2 Model Inputs

The CFM is forced by surface-temperature and accumulation-rate boundary conditions. Additionally, the user can specify the surface-melt, surface-density and water-isotope values. These files are .csv formatted. The first row of these files is time (decimal date, i.e. 2015.3487) and the second row is the corresponding temperature/accumulation rate/boundary condition value at that time. Time must be going forward, i.e. the first column is a date some time ago and the last column is the most recent. (If the model is being forced with ice-core data, the user must be careful to ensure this is the case as ice-core data are often presented as years before present.) The times in the various input files do not need to be the same; they are interpolated onto a common axis. The units for temperature can be K or C. The CFM uses K, but it will change the temperature to K if

you use C. The units for accumulation rate/surface mass balance are m ice equivalent per year (see note in section 5.3)

3 The .json configuration file

JSON (JavaScript Object Notation) is a data-interchange file format. It consists of a number of names, each associated with a value. Values can be strings, Booleans, integers, floats, or arrays. Comments are not allowed, but can be added by considering the comment as a name/value pair. For the CFM, it provides a file format that is both easy to read and easy to alter in order to specify parameters for a particular model run. The configuration file is passed to the CFM, and the name/value pairs are read by the model and incorporated into the model run. The file format is editable in any text editor, and the name/value pairs are given by *name: value*, and different name/value pairs are separated by commas:

```
{
“Name1”: “string”,
“Name2”: true,
“Name3”: 3,
“Name4”: 6.7,
“Name5”: [“one”, “two”, “three”],
“Comment”: “Name5 is an array of three strings”,
}
```

The specific names that are in the configuration .json file for the CFM are as follows. If any of the name/value pairs are missing, the model will generally give an error and the model run will not complete. Note that in the .json file true/false are lowercase, but in the .py files they are True/False (first letter capitalized). The model automatically converts this.

"InputFileFolder": *string.* directory where the input csv files are located (usually a subdirectory of the directory that contains main.py. If not, the user must change code in firn_density_spin.py). Use "" if the input files are in the same directory as main.py.

"InputFileNameTemp": *string.* Name of the temperature input file, which is a .csv file (see section 5.2).

"InputFileNamebdot": Name of the accumulation-rate input file. It is a .csv. Units are m ice equivalent *per year*, not for that particular time step. So, if you are using monthly time steps, perhaps it snows 0.05 m (ice) in January 2015. That column of the csv would be:

2015.0
0.6

Because annual rate would be $0.05 \times 12 = 0.6$ m i.e. a^{-1}.

"InputFileNameIso": Same as input bdot and temp, except the time history of isotope values (per mil) at the surface.

"InputFileNamerho": Time series of the surface density. The units are kg m^{-3}.

"InputFileNamemelt": Time series of the melt. Units are m per year ice equivalent (same as accumulation rate)

"resultsFolder": folder that will be created and where results will be stored.

"physRho": The physics that will be used for the model run.

"_physRhoOptions": A list of all the physics that are available to run. This is a effectively a comment in the code. The list is currently (August 2018):

"HLdynamic","HLSigfus","Li2004","Li2011","Helsen2008","Arthern2010S","Arthern2010T","Spencer2001","Goujon2003","Barnola1991","Morris2013","KuipersMunneke2015","Crocus"

"MELT": true/false; whether or not to run melt physics.

"FirnAir": true/false; whether or not to run firn air physics (i.e. the gas model)

"AirConfigName": Name of the .json file that is the configuration file for the gas model, e.g. FirnAir.json. If 'FirnAir' is true, the CFM will open this file and use its values for the firn-air model run.

"TWriteInt": Time write interval – how often to save model output. 1 saves every time step; 2 saves every other time step, 3 saves every 3rd, etc. This is a clunky part of the model that will be improved in the future. The time steps at which writing occurs should be changed by changing the variable "TWrite" in the file firn_density_nospin.py.

"SeasonalTcycle": true/false; whether to implement the seasonal temperature cycle. The details of this feature should be checked in firn_density_nospin.py. It includes a formulation for a coreless winter (CITE).

"TAmp": Amplitude of the seasonal temperature cycle, if that option is used (Kelvin).

"physGrain": true/false; whether or not to track grain size.

"calcGrainSize": true/false (physGrain must be true as well for this to be true); true uses a parameterization to get a surface value and false uses a set grain size at the surface. It is not clear where the current parameterization (which is temperature-based) comes from (code was written by a previous research group member and no documentation exists); therefore it is currently not recommended to use this option. The default initial grain size is 0.1 mm if 'calcGrainSize' is false.

"**heatDiff":** true/false; whether to run heat diffusion.

"variable_srho": true/false; variable surface density (true) or not.

"srho_type": There are different options for the surface density if variable_srho is true. "userinput" loads the csv file specified by InputFileNamerho. "param" uses a parameterization for surface density from Kuipers Munneke and others (2015). "noise" creates a random time

series of surface density. Each value is drawn from a normal distribution with mean "rhos0" and standard deviation of 25 kg m^{-3}. This value can be changed in the file firn_density_nospin.py

"_srho_type_options": The options for "srho_type". See above.

"rhos0": surface density if variable_srho is false or if "srho_type" is 'noise'.

"r2s0": initial condition for grain size; used only in firn_density_spin.

"AutoSpinUpTime": true/false. If true, model will attempt to spin up the model for as long as it takes to refresh the entire firn column to the depth where density is 850 kg m^{-3}. Use caution if you need the spin up to refresh the firn column to the full depth.

"yearSpin": how many years to spin up for. Does not do anything if 'AutoSpinUpTime'

"stpsPerYearSpin": how many time steps per year during the spin up (e.g. 12 is monthly)

"H": Thickness of the ice sheet in meters. This is a bit confusing. Probably keep it at 3000 or so. That would mean the surface of the firn is 3000 m above the bed.

"HbaseSpin": The elevation of the bottom of the model domain above the bed (confusing bit part 2!). So, if you want to model to 250 m depth, and H is 3000, HbaseSpin will be 2750. Likewise, if you wanted to model just the top 50 m of firn, HbaseSpin will be 2950 (assuming H is 3000). This is an initial value at the start of the spin up. The base of the model domain will change due to the fact the model is Lagrangian with a fixed number of nodes; e.g. if the accumulation rate increases, each node will be thicker, and the base of the domain will be deeper.

"stpsPerYear": steps per year for the model run (it probably should always just be the same as stpsPerYearSpin).

"D_surf": The surface boundary value of the layer tracking routine (See section XX). Default is 0.

"bdot_type": The type of accumulation rate to use for the densification physics. 'Instant' is the instantaneous value (i.e. at that time step) of accumulation, 'mean' is the mean accumulation over the lifetime of a parcel of firn. ('Stress' is in progress, 11/17/18, and will use the stress directly).

"bdot_options": ["instant","mean","stress"] See above.

"isoDiff": true/false; whether to run water isotope diffusion.

"iso": string. Which water isotope you are modeling. (Different isotopes have different diffusivities).

"_isoOptions": ["18","D","NoDiffusion"]; no diffusion means that the isotopes are tracked but do not diffuse (it is something of a baseline case).

"spacewriteint": interger. interval of the spatial nodes to write; 1 is every node; 2 is every other, etc.

"strain": true/false; whether to consider longitudinal strain

"du_dx": float. strain rate if "strain" is true; a value around 1e-5 is a good guess if you want to play. Currently (1/23/17) it is a fixed value, but a future release will allow variance with depth.

"outputs": which outputs you want the model to save, e.g.: ["density", "depth", "compaction ","DIP","BCO","temperature","LWC"]

"output_options": all of the options for outputs. ["density", "depth", "temperature", "age", "dcon", "bdot_mean", "climate", "compaction_rate", "grainsize", "temp_Hx", "isotopes", "BCO", "LIZ", "DIP", "LWC", "gasses"]

"resultsFileName": name of the results file; default: "CFMresults.hdf5",

"spinFileName": name of the spin-up results file; default: "CFMspin.hdf5"

"doublegrid": true/false; whether to use the regrid feature, which allows for a grid with varying resolution. (See Section XX)

"nodestocombine": integer. How many nodes should be combined into a single node if using double grid.

"grid1bottom": Float. Depth (m) at which the model transitions from the high-resolution grid to the lower-resolution grid if doublegrid is used.

5.3 Model outputs

The CFM writes its outputs to a single.hdf5-format file. By default, all nodes are written to file. The output is only saved at the time steps specified by the user with the variable TWrite. Most of the outputs should be self-explanatory. Many of them are big 2D matrices; the first column is time, and the values throughout are the particular values at the depth found in the corresponding cell in the depth output. Set the outputs you want in the .json file. The available outputs are:

depth: (m) The depth of each model node.

density: (kg m^{-3}) The density at the depths in 'depth

temperature: (K) Temperature at the depths in 'depth'

age: (years) Firn Age at the depths in 'depth'

dcon: Dcon is a layer-tracking routine; to use it you need to dig into the code a bit and program it how you want, but for example you could set it up so that each model node that has liquid water gets a 1 and all others get a zero. Corresponds to depth.

bdot_mean: (m a^{-1} ice equivalent) the mean accumulation rate over the lifetime of each parcel of firn, corresponds with 'depth'

climate: The temperature (K) and accumulation rate (m a^{-1} ice equivalent) at each time step – useful if using interpolation to find determine the climate.

compaction: (m) Total compaction of each node since the previous time step; corresponds to 'depth'. To get compaction rate you need to divide by the time-step size. To get compaction over an interval you need to sum numerous boxes.

grainsize: (mm^2) the grain size of the firn, corresponds to 'depth'

temp_Hx: the temperature history of the firn (See Morris and Wingham, 2014)

isotopes: (per mil) water isotope values, corresponds to 'depth'

LWC: (m^3) volume of liquid present in that node, corresponds to 'depth'

DIP: the depth-integrated porosity and change in surface elevation. 4 columns: The first is time, second is DIP to the bottom of the model domain (m), third is change in domain thickness since last time step (m), fourth is change in domain thickness since start of model run (m).

DIP also saves a variable called DIPc, which is a matrix of the cumulative porosity to the depth in 'depth'

BCO: bubble close-off properties. 10 columns: time, Martinerie close-off age, Marinerie close-off depth, age of 830 kg m^{-3} density horizon, depth of 830 kg m^{-3} density horizon, Martinerie lock-in age, Marinerie lock-in depth, age of 815 kg m^{-3} density horizon, depth of 815 kg m^{-3} density horizon, depth of zero porosity.

FirnAir: only works if FirnAir is true in the config.json. Saves gas concentrations, diffusivity profile, gas age, and advection rates of air and firn, all corresponding to 'depth'.

4 Detailed description of model files, modules, and model features.

5.4.1 main.py

This is the file that is (generally) called if you are running the CFM. It calls firn_density_spin and firn_density_nospin.

5.4.2 firn_density_spin.py

This 'spins up' the model to a steady state. It only uses a single value for temperature and accumulation rate at every time step (good for ice cores when you are doing a 40,000-year model run; bad for altimetry runs when the daily/monthly variability makes a difference for spin up. This module may be removed in future versions and integrated into firn_density_spin. It is set up as a class.

5.4.3 firn_density_nospin.py (class)

This is the file in which everything happens. It sets up the model grid and the initial and boundary conditions. Then it time steps through the model run, calling the different modules throughout. Set up as a class.

5.4.4 constants.py

All of the universal/global constants in one handy file. Hopefully all of the units are in there…

5.4.5 diffusion.py

As the name suggests, this file handles all of the diffusive processes: heat, enthalpy, and isotopes. It sets up the finite volumes and diffusivity and calls solver. All of these are set up as methods.

5.4.6 firn_air.py

Runs the firn-air module, including setting up boundary and intial conditions, handles the fir-air diffusion. This is a class.

5.4.7 hl_analytic.py

An implementation of the Herron and Langway analytic solution. Method.

5.4.8 melt.py

Functions defining the physics of the meltwater percolation, keeping track of how much liquid water there is and where it goes.

5.4.9 physics.py

Included all of the different densification physics (e.g. Herron and Langway, Ligtenberg, etc.) and also the grain-growth physics. This would also be an appropriate place to put other physics, such as evolution of microstructural properties. Set up as a class, though I am not sure that is the best way of doing this.

5.4.10 reader.py

Functions that read in the input files and the results from the spin up.

5.4.11 regrid.py

In order to run daily (or some high-resolution) time steps, it becomes prohibitively expensive due to the fact that the model domain gets a new box for each time step (that is, the grid spacing is dependent on the time step size). The regrid.py module solves this by re-gridding the model domain below a user-specified depth. **Use caution with this module! I think that it is more or less correct; however, it does not work with monthly (or larger) time steps. I have not yet sorted out at what time-step size it breaks down. Also: the way that liquid water is handled will remove any ice lenses that are present when nodes are combined (it conserves mass, so firn + ice = denser firn). So, ensure that the depth of grid 1 to grid 2 transition is adequately deep that you do not expect water to percolate that deep.**

The fields in the .json file need to be updated; there are three of them:

Regrid is true/false; it specifies that the regridding routine should be implemented. **grid1bottom** is the depth where you want to transition between the high and lower resolution grid. "Grid 1" is the upper, high-resolution grid, and "grid 2" is the deeper, lower resolution grid. **Nodestocombine** is the number of volumes that get combined into one single volume from the bottom of grid 1. Call this n for the rest of the document.

The regrid module retains the Lagrangian grid. Layers are tracked as being in grid 1, 2, or 3. Grid 3 is the very bottom, where the last volume will be split up into high-resoution layers (each keeping the properties of the previous, large volume that was split up).

During spin-up, the model initializes with a full high-resolution grid, but then takes all volumes below 'grid1bottom' and combines every 'nodestocombine' nodes into a single node. This is all using just z and dz – there is no need to change density and age at this point. Each node is assigned a value of 1, 2, or 3 based on which grid portion it is in, which is tracked in the variable 'gridtrack'.

When time stepping begins, the layers are advected downward as usual. A layer is added on top at each time step, and one is taken away from the bottom. So, for some number of time steps (n) the index of the first layer in grid 2 is increasing, and the number of layers in grid 3 is decreasing. Once all of the grid 3 layers are gone (n time steps have passed), it is time to regrid. The last n layers from grid 1 are combined into one single layer, which becomes the new top of grid 2. Since n layers have been consolidated into one (n to 1), the very last layer (which is the last layer of grid 2) is split into n layers (1 to n). The upshot is that the number of volumes stays constant. The updated grid is made by concatenating the remaining grid 1 layers, the new consolidated layer, the remaining grid 2 layers, and the new split-into-n layers.

The ‘grid1bottom’ does not actually end up being the depth bottom of grid 1, but it does reach a steady state that is deeper than the user-specified depth. The reason has to do with the thickness of the layers at the bottom that are being removed versus the thickness of the layers that are being added. Once the regridding is initialized in the spin up, the model does not consider the user-specified depth; it only considers the number of layers being consolidated. It does make sense that if you are adding a thick layer and removing a thin layer, the depth of the grid1-grid2 transition is going to go downward. So, a to-do is to perhaps figure out how to better predict where the 1-2 transition will occur once a steady state is reached. One thing to think about is how thick you want the lower-resolution layers to be. If you are taking daily time steps and your annual accumulation is ~0.2 m i.e., the high resolution layers will be ~5.5e-4 m i.e. thick. If you combine 30 layers, those layers will be something like 0.016 m i.e. thick.

5.4.12 solver.py

This script solves the diffusion equation. It is based on Patankar (1980).

5.4.13 writer.py

This script writes the model output to an output file in hdf5 format. It is formatted to write just the outputs that are specified in the configuration (.json) file. Writer.py also writes the results of the model spin up.

Model hosting/access

The CFM is on github: https://github.com/UWGlaciology/CommunityFirnModel

Parallelization

The CFM is not very parallelizable because processes happen in a time-stepping loop, and there are not many things to solve within the time-stepping loop. I have not worked on clocking the

model to see where the bottlenecks are, but I imagine that they are solving the diffusion equation and writing to disk. Having lots of RAM and an SSD will make you happy.

I have extensively used GNU Parallel to batch model runs; i.e. I set up a script that takes different arguments and calls the CFM. GNU parallel will leverage all of your cores to run those different model runs simultaneously.