
The Community Firm Model

Release 1.1.0

Jun 26, 2023

Contents:

1	Requirements	3
2	Contributing	33
3	Indices and tables	35
	Python Module Index	37
	Index	39

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

The Community Firn Model (CFM) is an open-source, 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. 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. $\rho_{new} = \rho_{old} + d\rho/dt * dt$). Heat and other (e.g. water isotope, firn air, enthalpy) diffusion is solved using a fully-implicit finite-volume method (Patankar, 1980).

CHAPTER 1

Requirements

Python 3.6+, [numpy](#), [scipy](#), [h5py](#), [pandas](#)

The CFM should run on Windows, Linux, and OSX platforms. 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. Explicit instructions on a scientific Python installation are beyond the scope of this document, but tutorials can be readily found online.

1.1 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. Here, we generically use `example.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.

1.1.1 Running the CFM

NOTE: For a more up-to-date list of changes to the CFM, please check the changelog which is in the CFM's root folder.

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. Here, we generically use example.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.

1.1.2 Running the example

The CFM includes an example .json file (cleverly called example.json). To test that the CFM is running on your system properly, you need to make sure that you have a directory called CFMinput with the appropriate file inside of it (MERRA2_CLIM_df_72.5_-38.75.pkl).

Then run:

```
>>> python main.py example.json
```

It takes ~60 seconds to run on my laptop. You can compare the outputs to the results provided in the ‘example_results_compare’ directory.

1.1.3 The .json-formatted configuration file

The CFM uses a .json-formatted file to configure individual model runs. 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.

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 return a message that that name/value pair is missing and will use a default instead. For some name/value pairs the model run will fail. 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. ρ_s is the surface.

.json keys

InputFileFolder

Directory where the input csv files are located (usually a subdirectory of the directory that contains main.py, but user can specify an absolute paths as well.) Use “” if the input files are in the same directory as main.py.

type string

example inputdata

InputFileNameXXXX

The names of the input files for temperature, accumulation/smb, water isotopes, surface density, and melt. See 'Inputs for the CFM' section for more details.

type string

Example example_XXXX.csv

resultsFolder

Folder in which results are stored.

type string

Example example_results

initfirnFile

File containing initial conditions if you are using firn measurements/data (e.g. temperature, density) to begin the model run. See 'Inputs for the CFM' section for more details.

type string

Example example_firndata.csv

initprofile

Whether or not the CFM should use the initfirnFile to generate an initial condition.

type boolean

default False

input_type

(New in version 1.1.0) Specify what type of inputs you want to use - .csv (historic behavior) or pandas dataframe that is stored in a pickle.

type string

default csv

options csv, "dataframe"

DFresample

(New in version 1.1.0) Specify the resolution you want for your model run, which will be the re-sample interval for the dataframe (this only has functionality when input_type is dataframe) See <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Timedelta.html>

type pandas Timedelta (string)

Example 1D

DFfile

The filename of the pickle containing the climate dataframe.

type string

example example.pkl

physRho

The firn-densification physics to use for the model run.

type string

Options HLDynamic, HLSigfus, Li2011, Helsen2008, Arthern2010S, Arthern2010T, Li2015, Goujon2003, Barnola1991, Morris2014, KuipersMunneke2015, Crocus, Ligtenberg2011

MELT

Whether or not to include meltwater percolation physics in the model run.

type boolean

default False

ReehCorrectedT

If melt is enabled, (NEED TO FILL IN WHAT THIS DOES)

type boolean

default False

FirnAir

Whether or not to run the firn air module with the model run.

type boolean

default false

AirConfigName

Name of the .json configuration files that contains the parameters for the firn air module.

type string

default AirConfig.json

TWriteInt

How often to write the results to file, relative to the time-step size, i.e. 1 will write at every time step, 10 will write at every 10th time step, etc.

type int

default 1

TWriteStart

The time at which to start saving model results. The time is model time, so must correspond to the time in the input forcing files.

type float

Example If your model run is from 1958 to 2018, but you only want outputs from 2000 onwards, 'TWriteStart' should be 2000.

int_type

How to interpolate from the input file times to the model time. Use linear e.g. if you have sparse ice core climate data. Use nearest e.g. if you have monthly climate data and want to take monthly time steps (the model time may not be exactly the same as the input time).

type string

options nearest, linear

SeasonalTCycle

Whether or not to add a seasonal temperature cycle (on top of the forcing data). Use this only if you are using sub-annual time steps and your forcing data does not have a seasonal cycle already. Usually this would be if your forcing data is annual (or coarser resolution).

type boolean

default false

SeasonalThemi

If 'SeasonalTCycle' is True, specify which hemisphere you are modeling to get the summer/winter timing correct.

type string

options north, south

coreless

If 'SeasonalTCycle' is True, add the coreless winter. ADD MORE INFO HERE

type boolean

default false

Tamp

If 'SeasonalTCycle' is True, specify the amplitude of the cycle.

type float

default 10

units K

physGrain

Whether or not to track grain size evolution. Must be True for Arthern2010S physics.

type boolean

default false

calcGrainSize

True uses a parameterization to get a surface grain-size at each time step, and False uses a set grain size at the surface.

type boolean

default false

GrGrowPhysics

Which equation to use to calculate grain size evolution.

type string

options Arthern, "Katsushima"

heatDiff

Whether or not to include heat diffusion.

type boolean

default True

conductivity

Which parameterization for heat conductivity to use.

type string

options Schwander, "Yen_fixed", "Yen_var", "Anderson", "Yen_b", "Sturm", "VanDusen", "Schwerdtfeger", "Riche

variable_srho

Whether to vary the surface density through time. False uses a constant density.

type boolean

default False

srho_type

If variable_srho is true, how to vary the surface density through time. 'userinput' uses a csv file with surface density through time (must be specified with InputFileName_rho); 'param' uses a parametrization; 'noise' adds noise at each time step to the value specified by **rhos0**.

type string

options userinput, "param", "noise"

rhos0

Surface density at each time step if using a constant surface density, or the mean value if **variable_srho** is true and **srho_type** is 'noise'.

type float

default 350.0

units kg m⁻³

r2s0

Surface grain size at each time step if **calcGrainSize** is false.

type float

default 1e-8

units mm²

AutoSpinUpTime

Calculate the spin up time automatically based on the input accumulation rate and specified model domain depth; should be long enough to refresh the entire firn column during spin up.

type boolean

default false

yearSpin

How many years to spin up for. COULD EXPAND ON THIS

type float

stpsPerYearSpin

DEPRECATED How many time steps per year to take during spin up. Previously the CFM gave the option to have different values for spin up and main run; now spin up uses **stpsPerYear**.

stpsPerYear

How many time steps per year to take. E.g. 12 will make the model take monthly time steps, 1 will give annual time stepping. Take care to coordinate this value with your input files and the 'int_type'.

type float

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.

type float

default 3000

units m

HbaseSpin

The elevation of the bottom of the model domain above the bed. 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 depth 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.

type float

units m

D_surf

The CFM features a generic layer tracker called *D_con*; it can be used for a number of things. This is the value to assign a new layer at the surface at each time step.

type float

default 1

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 and will use the stress directly).

type string

default mean

options mean, "instant", "stress"

grid_outputs

Whether or not to put the outputs on a regular grid (i.e. evenly spaced vs. the internal variable grid)

type boolean

default True

grid_output_res

If grid_output is True, this is the spacing of the grid nodes in meters.

type float

default 0.1

isoDiff

Whether or not to include water isotope diffusion in the model run.

type boolean

default False

iso

If isoDiff is true, which isotopes to model. 'NoDiffusion' will include the isotopes but does not diffuse them at each time step to allow analysis of the effects of advection and compaction alone (it uses the d18O forcing).

type list of strings

default ["18", "D", "NoDiffusion"]

options 18,D,NoDiffusion

spacewriteint

NOT WORKING CURRENTLY. Spatial resolution interval to save to results. 1 is every node; 2 is every other, etc.

type int

default 1

strain

Whether or not to include layer thinning due to horizontal strain from dynamic ice sheet/glacier flow.

type boolean

default False

du_dx

If strain is true, this is the horizontal strain rate. Future work will allow this to vary in time. NEED TO CHECK UNITS ARE CORRECT.

type float
default 1e-5
units m a⁻¹

outputs

Which outputs to save.

type list of strings
example ["density", "depth"]
options density, depth, temperature, age, dcon, bdot_mean, climate, compaction, grainsize, temp_Hx, isotopes, BCO, LIZ, DIP, LWC, gasses

resultsFileName

Name of the .hdf5 file that results are saved in.

type string
default CFMresults.hdf5

spinFileName

Name of the .hdf5 file that the spin up results are saved in.

type string
default CFMspin.hdf5

doublegrid

Whether or not to use the feature that keeps a high-resolution grid near the surface and a lower-resolution grid at greater depth.

type boolean
default false

nodestocombine

If **doublegrid** is True, this is how many nodes are combined into a single node at the high/low resolution boundary. So, if it is 50, at every 50th time steps 50 nodes will be combined into a single node.

type int
default 50

multnodestocombine

If **doublegrid** is True, this is how many nodes are combined into a single node at the boundary between the low and very low resolution grid. For example, if **nodestocombine** is 50, multnodes will combine ‘multnodestocombine’ of those 50-node thick layers into a single node.

type int
default 6

grid1bottom

If **doublegrid** is True, the depth (m) at which the high-resolution grid nodes are combined.

type float
default 10

grid2bottom

If **doublegrid** is True, the depth (m) at which the low-resolution grid nodes are combined to make the very-low resolution grid.

type float
default 20

spinup_climate_type

What climate to use for the spin up. ‘initial’ uses the very first value in the input .csv files and ‘mean’ uses the mean of the values in those files.

type string
options initial,mean

manual_climate

Manually specify the background climate (long-term means). This is useful if you are doing a very short model run, in which the input csv files may not be representative of the long-term climate.

type boolean
default false

deepT

If **manual_climate** is true, this is the long term site temperature (the temperature that would be measured at the bottom of a borehole).

type float
units K

bdot_long

If `manual_climate` is true, this is the long-term mean accumulation rate.

type float
units m ice eq. a⁻¹

manual_iceout

Allows the user to specify the ice that is effectively removed from the bottom of the firn due to ice sheet thinning from ice flow. In steady state, `iceout` is the same as the long-term ice equivalent accumulation rate (and that is what is used if `manual_iceout` is false).

type boolean
default false

iceout

If `manual_iceout` is True, this is the value.

type float
units m ice eq. a⁻¹

QMorris

The Morris and Wingham (2014) model allows for different activation energies; specify it here.

type float
default 110.0e3
units kJ mol⁻¹

timesetup

How to set up the time step size. ‘Exact’ uses the input files to find the times at which a time step occurs and the corresponding time-step size *dt*; ‘interp’ uses a uniform *dt* and interpolates the input data onto the timeline that the model generates with uniform time steps. ‘retmip’ is a specialty for the RETMIP experiment and may not be fully functional.

type string
options exact, interp, retmip

liquid

If **MELT** is true, which percolation scheme to use.

type string
options percolation_bucket, bucketVV, resingledomain, prefsnowpack

merging

If a model volume gets too thin, merge it with another. Needed for numerical stability with melt schemes.

type boolean

default false

merge_min

If merging is true, the thickness threshold at which merging should occur.

type float

default 1e-4

manualT

Option to use manual temperature measurements, e.g. from a thermistor string.

type boolean

default false

no_densification

Option to set densification to false (perhaps you are simulating temperature diffusion in a core in a lab)

type boolean

default false

rad_pen

Option to turn on radiation penetration module.

type boolean

default false

site_pressure

Set the pressure at the site, which can affect isotope diffusion.

type float

default 1013.25

output_bits

Set the bits for the outputs.

type string

default float32

spinUpdate

Specify if you want to update the spin file at some date.

type boolean

default false

spinUpdateDate

Specify the date at which to update the spin file. should correspond to the start of your reference climate interval.

type float

default 1980.0

DIPhorizon

Depth horizon at which to calculate DIP/FAC (because the bottom of the domain varies a bit).

type float

default 100.0

NewSpin

Whether or not to perform a new spin up (if the spin file exists already.)

type boolean

default false

1.1.4 Inputs for the CFM

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)

Test Linking Pages

1.1.5 CFM outputs

The CFM writes its outputs to a single.hdf5-format file. In older CFM versions, all nodes (aka model grid layers) were written to file. This is still an option, but there is also now an option to interpolate the results onto a regular grid (using “grid_outputs”: [true/false] in the .json file, setting the grid resolution using “grid_output_res”).

The outputs are 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. If “grid_outputs” = True, this will be a vector of depths. (The first value is the decimal date of the model initialization time, which you should ignore.)

MATRIX OUTPUTS:

density: (kg m⁻³) The density at the depths in ‘depth’

temperature: (K) Temperature at the depths in ‘depth’

age: (years) Age of the firn at the depths in ‘depth’

grainsize: (mm²) 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: (m³) volume of liquid present in that node, corresponds to ‘depth’

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.

dcon: (advanced use) 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 vector/matrix.

bdot_mean: (m a⁻¹ ice equivalent) the mean accumulation rate over the lifetime of each parcel of firn, corresponds with ‘depth’

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’.

VECTOR OUTPUTS:

forcing: This is the climate forcing inputs, i.e. it is just keeping a record of the data that is fed into the CFM. I included it to enable reproducibility; i.e., if you need to, you should be able to use these to create new forcing files to repeat a model run if you need to. 5 columns (as of this documentation update, though you can check writer.py, in the ‘if forcing_dict’ block, for any updates there.): decimal date, skin temperature, accumulation, snowmelt, rain.

Modelclimate: 3 columns: timestep, temperature (K), and accumulation rate (m a⁻¹ ice equivalent). This is useful if using interpolation to find determine the climate (e.g. long model runs for paleoclimate). This is probably not useful if you are doing runs with ‘timesetup’=‘exact’, as would be expected e.g. for runs investigating modern height and mass change.

DIP: the depth-integrated porosity and change in surface elevation. The columns in DIP are (with python’s 0-based indexing): (0) timestep, (1) Depth Integrated Porosity, aka Firn Air Content (m), to the bottom of the model domain, (2) surface elevation change since last time step (m), (3) cumulative elevation change since start of model run (m), (4) total compaction (m) of the firn column since last time step, (5) ‘corrected’ cumulative elevation change since start of model run (m), (6) ‘corrected’ cumulative elevation change since start of model run (m), (7) Firn Air Content above a specified horizon.

The ‘corrected’ fields (5 and 6) tries to account for any compaction that occurs between the bottom of the model domain and the depth where the column reaches ice density (917), in the case that you are not modeling to the ice density.

Firn air to a horizon (7) is there because due to the model’s lagrangian format, the bottom of the domain varies in time, and if the bottom is at a density less than ice density it can be useful to consider the FAC to a specified depth,.

(I'd advise using caution if you use the modeled elevation change fields - we have made changes to the code to merge deeper nodes together to save computing time, but in doing so that can make the model-outputted dH have blips in it. If you want to work with this field let me know and I can do a bit of testing. I'd recommend structuring your analyses around the FAC variable.)

BCO: bubble close-off properties. 10 columns: time, Martinerie close-off age, Martinerie close-off depth, age of 830 kg m⁻³ density horizon, depth of 830 kg m⁻³ density horizon, Martinerie lock-in age, Martinerie lock-in depth, age of 815 kg m⁻³ density horizon, depth of 815 kg m⁻³ density horizon, depth of zero closed porosity. The Martinerie fields refer to parameterizations for close-off properties published by Patricia Martinerie in the early 1990s. See references below.

meltvol: two columns: decimal time, total melt volume at that time step [m w.e.]

refreeze: two columns: decimal time, total liquid water refreezing at that time step [m w.e.]

runoff: two columns: decimal time, total liquid water runoff at that time step [m w.e.]

References Goujon, C., Barnola, J.-M., and Ritz, C. (2003), *Modeling the densification of polar firn including heat diffusion: Application to close-off characteristics and gas isotopic fractionation for Antarctica and Greenland sites*, *J. Geophys. Res.*, 108, 4792, doi:10.1029/2002JD003319, D24.

Martinerie, P., Lipenkov, V. Y., Raynaud, D., Chappellaz, J., Barkov, N. I., and Lorius, C. (1994), *Air content paleo record in the Vostok ice core (Antarctica): A mixed record of climatic and glaciological parameters*, *J. Geophys. Res.*, 99(D5), 10565– 10576, doi:10.1029/93JD03223.

Martinerie, P., Raynaud, D., Etheridge, D. M., Barnola, J.-M., & Mazaudier, D. (1992, August). *Physical and climatic parameters which influence the air content in polar ice*. *Earth and Planetary Science Letters*. Elsevier BV. [https://doi.org/10.1016/0012-821x\(92\)90002-d](https://doi.org/10.1016/0012-821x(92)90002-d)

1.2 Files included with the CFM

Here is a list.

1.2.1 airconfig.json

File configuring parameters for a model run that includes firn-air diffusion.

1.2.2 constants.py

Constants used in the CFM. Units are generally mks.

1.2.3 diffusion

This module contains the code for the diffusion scheme models.

code to handle diffusion (heat, enthalpy) calls solver Draws from Numerical Heat Transfer and Heat Flow (Patankar, 1980) Enthalpy from Voller, Swaminathan, and Thomas (1990) Isotope diffusion now has its own class.

`diffusion.LWC_correct(self)`

* **TEST FUNCTION** * If there is LWC in a layer after temperature diffusion and the temperature is less than zero, one option is to just balance the energy to increase the temperature and lower the LWC. It isn't the best way to solve the problem but it is one way.

This should be vectorized but that is not a priority.

`diffusion.enthalpyDiff (self, iii)`
 enthalpy diffusion function, new 1/30/19 - method from Voller and Swaminathan LWC is in volume (m^3)
 thermal diffusivity: $\alpha = K_{\text{firn}} / (\rho \cdot c_{\text{firn}})$

`diffusion.firnConductivity (self, iii, K_ice)`
 Function to set the firn's thermal conductivity based on one of a number of parameterizations.
 Choose your favorite! Default is Calonne et al. 2019. References are provided at the end of this script.

`diffusion.heatDiff (self, iii)`
 Heat diffusion function

Parameters

- **z** –
- **dz** –
- **Ts** –
- **rho** –

Returns self.Tz

Returns self.T10m

thermal diffusivity: $\alpha = K_{\text{firn}} / (\rho \cdot c_{\text{firn}})$

`diffusion.heatDiff_LWCcorr (self, iii, iters, correct_therm_prop)`
 IN DEVELOPMENT

just run the heat diffusion as normal and then balance energy.

`diffusion.heatDiff_Teff (self, iii)`
 IN DEVELOPMENT artificially set the temperature of volumes with liquid water to be higher than T_{melt}

`diffusion.heatDiff_highC (self, iii)`
 IN DEVELOPMENT

One way of dealing with liquid water in the firn is to just set the heat capacity to be very high.

1.2.4 example.json

An example .json configuration file.

1.2.5 fcts_snowpackflow.py

This script contains all the functions required to make the preferential flow scheme of snowpack work <https://models.slf.ch/p/snowpack/source/tree/HEAD/branches/dev/snowpack/snowpackCore/ReSolver1d.cc>

`fcts_snowpackflow.Micedryer (dz, rho, Mtheta, Mtheta_sat, crtn_theta, rhoimp, totrunoff)`
 This works the same as satexcess but in order to make sure that layers at pore close-off density ($\rho > \rho_{\text{imp}}$) are dry in MFdom.

`fcts_snowpackflow.Mrefreezing (dz, zstep, rho, grain, Tz, Mthetar_old, Mlwc, lwc_min_fr, Ptheta, PeffSat, Plwc, h_e, bigF, mu, crtn_theta, rhoimp, totrefrozen_lwc, refrozenlay, totrunoff)`
 Proceed to refreezing according to cold content of every layer. Adjust porosity and hydraulic properties accordingly. Verify that we don't oversaturate PFdom. If so, we call for Psatexcess

`fcts_snowpackflow.Msatexcess` (*dz, rho, Mtheta, Mtheta_sat, crtn_theta, rhoimp, totrunoff*)

For MFdom In case the water content of some layers exceeds the water content at saturation, we move the water to the layers below (in priority) and in the layers above (if there is not enough pore space in all the layers below)

`fcts_snowpackflow.NPtrid` (*a, b, c, d*)

function to solve tridiagonal matrix -> find x matrix in $Ax=d$ equation

a,b,c,d are arrays, should be made of floats, not integer a and c are 1 index shorter than b and d - [b0 c0 0. 0. ...] - [a0 b1 c1 0. ...] $A = [\dots] - [\dots]$

`fcts_snowpackflow.PFleave` (*dz, rho, Tz, Mtheta, Mthetar, Mthetar_old, MeffSat, Mtheta_sat, Mlwc, Ptheta, PeffSat, Plwc, Ptheta_sat, crtn_theta, rhoimp, aquif, PSatlim*)

When the saturation in the PF domain reaches a threshold value (PSatlim), backflow towards MFdom occurs. First we transfer as much water as the cold content of the layer can accomodate Second, if PSatlim is still exceeded, we equalise saturations in both domains Note that at the end, saturation might not be the same if we equalise because Mthetar can change subseuntly

`fcts_snowpackflow.PFleaveheat` (*dz, rho, Tz, Mtheta, Mthetar, Mthetar_old, MeffSat, Mtheta_sat, Mlwc, Ptheta, PeffSat, Plwc, Ptheta_sat, crtn_theta, kth, bigF, bigN, aquif, rhoimp, deltatime*)

This mimics refreezing in the PF domain by transferring water back to the MF dom depending on how much heat would be lost by water in PFdom (see paragraph 2.3 of Wever 2016) It depends on the tuning parameter bigN which represents number of preferential flow paths per unit area Based on equations (6) and (7) of Wever 2016 but caution, misformulation in (7)

`fcts_snowpackflow.Picedryer` (*dz, rho, Ptheta, Ptheta_sat, crtn_theta, rhoPdr, totrunoff*)

This works the same as satexcess but in order to make sure that layers at pore close-off density ($\rho > \rho_{imp}$) are dry in MFdom.

`fcts_snowpackflow.Prefreezing` (*dz, rho, grain, Tz, Mthetar_old, Mtheta, Mlwc, lwc_min_fr, Ptheta, PeffSat, Plwc, bigF, h_e, mu, crtn_theta, dryfront, totrefrozen_lwc, refrozenlay, rhoimp, totrunoff*)

Not used in preferential flow scheme of Wever 2016 but might be a good idea to use. Proceed to refreezing of Plwc until dryfront according to cold content of every layer. Adjust porosity and hydraulic properties accordingly. Also needs the variables Mtheta and dryfront as input parameters (vs refreezing())

`fcts_snowpackflow.Psatexcess` (*dz, rho, Ptheta, Ptheta_sat, crtn_theta, rhoimp, totrunoff*)

This works the same as Msatexcess but for PFdom In case the water content of some layers exceeds the water content at saturation, we move the water to the layers below (in priority) and in the layers above (if there is not enough pore space in all the layers below)

`fcts_snowpackflow.TDMAsolver` (*a, b, c, d*)

Way faster than NPtrid!!

TDMA solver -> find x matrix in $Ax=d$ equation a,b,c,d are arrays, should be made of floats, not integer a and c are 1 index shorter than b and d - [b0 c0 0. 0. ...] - [a0 b1 c1 0. ...] $A = [\dots] - [\dots]$ A is nxn tridiagonal matrix with a - b - c as diagonals, x is nx1 matrix, d is nx1 matrix Sources: <https://gist.github.com/cbellei/8ab3ab8551b8dfc8b081c518ccd9ada9> https://en.wikibooks.org/wiki/Algorithm_Implementation/Linear_Algebra/Tridiagonal_matrix_algorithm#Python

`fcts_snowpackflow.combineCFM` (*split_list, rhoF, dzF, TzF, massF, lwcF, Plwc_memF, r2F, refrozenF*)

F for fine grid C for coarse grid Here, we need the vectors that are outputs of the flow routine: - combine them back to reintegrate these to the CFM - Note that we also need the split_list ! (nb of sublayers into which every CFM layer was split by the split function) - Normally, RE routine (including freezing) should not affect dz and r2 variables but only use them -> not necessary to combine them and to give them back to CFM (which can keep its own self.dz and self.r2)


```
fcts_snowpackflow.entrysuction(dz, Mtheta, Mthetar, Mthetar_old, MeffSat, Mtheta_sat,
                               Mlwc, Ptheta, PeffSat, Plwc, Ptheta_sat, crtn_theta, aquif,
                               MSat_westag)
```

When the pressure exceeds the water entry suction of layer below, water penetrates from MFdom layer to PFdom of layer below We convert water entry suction in more intuitive saturation value If after transfer, saturation of the layer below in PFdom is still inferior to saturation in layer above in MFdom, we equalise saturations

```
fcts_snowpackflow.layerequaliser_eq(dz, Mtheta, Mthetar, Mthetar_old, MeffSat, Mtheta_sat,
                                    Mlwc, Ptheta, PeffSat, Plwc, Ptheta_sat, crtn_theta,
                                    aquif)
```

Whenever the effective saturation in the MFdom exceeds the saturation in the PFdom within a same layer, we equalise saturations Not sure this is physically realistic but that is what is written in Wever 2016 <https://models.slf.ch/p/snowpack/source/tree/HEAD/branches/dev/snowpack/snowpackCore/ReSolver1d.cc> + confirmed by Nander Wever, email 27 Sept 2018

```
fcts_snowpackflow.lengthendom(self, rho_short, dz_short, Tz_short, mass_short, lwc_short,
                              Plwc_mem_short, r2_short)
```

After having solved the flow, we concatenate back new values with deep values that were not taken into account during the flow routine (because all their rho values was above rhoimp from a certain depth). !! This function has to be called before the flow routine but AFTER the melting !!

```
fcts_snowpackflow.restrictdom(self)
```

Limit the domain to reduce computational time of flow solving Spot the deepest layer below pore close-off density (rhoimp). The bottom of the domain will be the next layer. Thus, we do not take all the column where rho is constantly above pore close-off density (rhoimp). But we make sure not to exclude any layer that has liquid water

```
fcts_snowpackflow.runoff(dz, rho, Mhead, Mtheta, Mthetar, Mthetar_old, MeffSat, Mlwc,
                        Mtheta_sat, theta_min_fr, crtn_theta, slope, rhoimp, aquif, deltatime,
                        totrunoff)
```

This is the Greenland specific runoff function of Zuo and Oerlemans 1996 (21) We proceed to runoff only for layers of which water content is above theta_min_fr. CAUTION: this might be changed!! We don't apply runoff in the aquifer

```
fcts_snowpackflow.splitCFM(rhoC, dzC, TzC, massC, lwcC, Plwc_memC, r2C, vert_res)
```

F for fine grid C for coarse grid We split the layers of the CFM in layers of a certain maximal thickness. Maximal thickness must be specified in vert_res. With the implementation of upstream weighted mean for K at interfaces, we can take large vert_res value!

1.2.6 firn_air.py

code to deal with modeling firn-air diffusion

```
class firn_air.FirnAir(air_config, Gs, z, modeltime, Tz, rho, dz, gaschoice, bdot)
```

```
__init__(air_config, Gs, z, modeltime, Tz, rho, dz, gaschoice, bdot)
    Initialize Firn Air class
```

```
diffusivity()
    D_0 is CO2 in free air. gam_x is the diffusivity relative to CO2 D_x=D_0*gam_x is the free-air (or any)
    diffusivity for that species diffu_full has units m^2/s
```

```
firn_air_diffusion(AirParams, iii)
    Solve the diffusion equation. Calls solver.py
```

```
porosity()
    Calculate the total, open, and closed porosity. co is close-off, cl is closed
```

`firn_air.gasses` (*gaschoice, T, p_a, M_air*)
Function to set up specifics for different gasses.

1.2.7 firn_density_nospin

1.2.8 firn_density_spin

Class for spinning up the model.

This file spins up to a steady-state firn column using constant temperature, accumulation rate, and surface density. This works well for long model runs with big time steps (e.g. for ice-core related questions).

To spin up using a climate with a bit of variability (like a reference climate interval) for e.g. altimetry or melt related runs, this script will essentially create an initial condition. In this case, set 'yearSpin' in your .json file to some small number (e.g. 1); otherwise you are wasting computing time.

```
class firn_density_spin.FirnDensitySpin(config, climateTS=None)
    Parameters used in the model, for the initialization as well as the time evolution:

    : gridLen: size of grid used in the model run (unit: number of boxes, type: int)

    : dx: vector of width of each box, used for stress calculations (unit: m, type: array of ints)

    : dz: vector of thickness of each box (unit: m, type: float)

    : z: vector of edge locations of each box (value is the top of the box) (unit: m, type: float)

    : dt: number of seconds per time step (unit: seconds, type: float)

    : t: number of years per time step (unit: years, type: float)

    : modeltime: linearly spaced time vector from indicated start year to indicated end year (unit: years,
        type: array of floats)

    : years: total number of years in the model run (unit: years, type: float)

    : stp: total number of steps in the model run (unit: number of steps, type: int)

    : T_mean: interpolated temperature vector based on the model time and the initial user temperature data
        (unit: ???, type: array of floats)

    : Ts: interpolated temperature vector based on the model time & the initial user temperature data may
        have a seasonal signal imposed depending on number of years per time step (< 1) (unit: ???, type: array
        of floats)

    : bdot: bdot is meters of ice equivalent/year. multiply by 0.917 for W.E. or 917.0 for kg/year (unit: ???,
        type: )

    : bdotSec: accumulation rate vector at each time step (unit: ???, type: array of floats)

    : rhos0: surface accumulate rate vector (unit: ???, type: array of floats)

    Returns D_surf diffusivity tracker (unit: ???, type: array of floats)

time_evolve()
    Evolve the spatial grid, time grid, accumulation rate, age, density, mass, stress, and temperature through
    time based on the user specified number of timesteps in the model run. Updates the firn density using a
    user specified
```

1.2.9 firnbatch_generic

1.2.10 hl_analytic

The Herron and Langway analytic model.

`hl_analytic.hl_analytic(rhos0, h, THL, AHL)`

Model steady-state firn density and age profiles and bubble close-off, uses m w.e. a^{-1}

Parameters

- **rhos0** – surface density
- **h** – depth
- **THL** –
- **AHL** –

Return age age vector of firn column with steady-state dynamics

Return rho density vector of firn column with steady state dynamics

1.2.11 isotopeDiffusion.py

Code for isotope diffusion.

`class isotopeDiffusion.isotopeDiffusion(spin, config, isotope, climateTS, stp, z, updated-StartDate=None, modeltime=None)`

Isotope diffusion class.

Note that presently isotope diffusion only works with .csv inputs.

`isoDiff(IsoParams, iii)`

Isotope diffusion function

Parameters

- **iter** –
- **z** –
- **dz** –
- **rho** –
- **iso** –

Returns self.phi_t

1.2.12 main

1.2.13 melt.py

`melt.bucket(self, iii)`

Percolation bucket scheme, with edits by Max Several parameters can be set by the user (see below `### USER CHOICES ###`)

Coded by Vincent Verjans

`melt.darcyscheme (self, iii)`

Threshold thickness for ice lenses to be impermeable Approach: modify density for calculation of Darcy variables (upper threshold set to 910) Liquid water input is distributed in upper nodes rather than using a flux boundary condition at the surface node (causes high runoff) Input is accomodated until impermeable node reached: remaining input runs off Melting and refreezing occur at each Darcy step

1.2.14 merge.py

Script that contains 3 functions. These are to be used if we want to proceed to merging of thin layers of the firn column. I suggest we specify in json input if merge is true/false and the thickness threshold ('merge_min'): "merging": true, "merge_min": 5e-3 `mergesurf()`: for layer[0] and layer[1], to be used in `time_evolve()` of `firn_density_nospin` `mergenotsurf()`: for layers[2:], to be used in `time_evolve()` of `firn_density_nospin` `mergeall()`: for all layers, to be used at the end of `firn_density_spin` CAUTION: - not used for all variables (e.g. `du_dx`) - nothing is done considering gas neither for isotopes @author: verjans

`merge.mergeall (self, thickmin, iii)`

We spot the layers that are under a certain thickness threshold and we merge these with the underlying layer This has to be launched at the end of the spinup, we use other functions in `firn_density_nospin`. Here we change `self.complexes` as we modify the number of layers above 80m depth.

`merge.mergenotsurf (self, thickmin, iii)`

This function is to call during `time_evolve` function of `firn_density_nospin`. We merge all the layers below a thickness threshold except the layers of indices 0 and 1. This allows layers that became too thin due to compaction to be merged with the layer below. Minimum thickness threshold must be specified as `thickmin` We don't do this for surface layer because that would lead to any newly accumulated layer to be merged if RCM forcing is on a short time scale. The surface layer has its own function `mergesurf()`.

`merge.mergesurf (self, thickmin, iii)`

This function is to call during `time_evolve` function of `firn_density_nospin`. We merge the surface layer[0] with the layer[1] below as long as layer[1] remains under a certain thickness threshold. By applying condition on layer[1] instead of layer[0], we avoid merging all newly accumulated layers in the case we use a RCM forcing on a short time scale. Thickness threshold must be specified and consistent with the one of `mergenotsurf()`.

1.2.15 physics.py

This module contains all of the firn-densification models.

physics.py

The Community Firn Model physics module.

equations for each of the densification models.

class `physics.FirnPhysics (PhysParams)`

class for the various firn densification schemes

The standard parameters that get passed are: `iii`, `steps`, `gridLen`, `bdotSec`, `bdot_mean`, `bdot_type`, `Tz`, `T10m`, `rho`, `sigma`, `dt`, `Ts`, `r2`, `physGrain`

if you want to add physics that require more parameters, you need to change the 'PhysParams' dictionary in both the `spin` and `nospin` classes.

Parameters

- **steps** – # of steps per year
- **gridLen** –

- **bdotSec** –
- **Tz** –
- **rho** –
- **sigma** –

Return drho_dt

Return viscosity

Arthern_2010S ()

This is the steady-state solution described in the main text of Arthern et al. (2010) Accumulation units are $\text{kg/m}^2/\text{year}$

Arthern_2010T ()

This is the transient solution described in the appendix of Arthern et al. (2010)

Uses stress rather than accumulation rate.

Barnola_1991 ()

uses m W.E. (zone 1) and stress (zone 2)

Breant2017 ()

Bréant, C., Martinerie, P., Orsi, A., Arnaud, L., and Landais, A.: Modelling firn thickness evolution during the last deglaciation: constraints on sensitivity to temperature and impurities, *Clim. Past*, 13, 833–853, <https://doi.org/10.5194/cp-13-833-2017>, 2017.

Brils_2022 ()

Units are mm W.E. per year b_{dot} is meant to be accumulation over a reference period (20 years for spin up, 1 year for regular?) (not mean over the lifetime of a parcel)

Crocus ()

Uses stress

GSFC2020 ()

Taken from Medley et al. (2020) Accumulation units are $\text{kg/m}^2/\text{year}$ Subscript here are changed to 1 and 2 (correspond to densification stage) (in Medley et al. 2020 they are 0 and 1)

Goujon_2003 ()

Uses stress

HL_Sigfus ()

Accumulation units are m W.E. per year (zone 1); uses stress for zone 2

HL_dynamic ()

Accumulation units are m W.E. per year viscosity doesn't work

Helsen_2008 ()

Accumulation units are m W.E. per year (?) Equation is from Arthern et al. 2010 (2) (Arthern implies units are m I.E.; not doing that here)

KuipersMunneke_2015 ()

Units are mm W.E. per year b_{dot} is meant to be accumulation over a reference period (20 years for spin up, 1 year for regular?) (not mean over the lifetime of a parcel)

Li_2004 ()

Accumulation units are m W.E. per year (?) Equation from Arthern, 2010 (eq. 2): not sure where Rob got that? (Arthern implies accumulation is m I.E./year for b_{dot} ; not doing that here.) Paper would lead me to believe that it is m W.E.

Needs to have the vapor flux coded in if we want to use these physics properly.

Li_2011 ()

Accumulation units are m W.E. per year (email correspondence with J. Li, 12/3/13) Temperature in the equation for beta is in C. Temperature in the $8.36(273.15-T)^{-2.061}$ is in K. (implied in Arthern)

beta should be calculated with the long-term mean accumulation rate

Li_2015 ()

Accumulation units are m W.E. per year (email correspondence with J. Li, 12/3/13) Temperature in the equation for beta is in C. Temperature in the $8.36(273.15-T)^{-2.061}$ is in K. (implied in Arthern)

beta should be calculated with the long-term mean accumulation rate

Ligtenberg_2011 ()

Units are mm W.E. per year b_dot is meant to be accumulation over a reference period (20 years for spin up, 1 year for regular?) (not mean over the lifetime of a parcel)

Max2018 ()

Experimental: based on firn compaction data from FirnCover campaigns

Max2018b ()

Experimental: based on firn compaction data from FirnCover campaigns

Morris_HL_2014 ()

Uses stress instead of accumulation.

Need to choose physics for zone 2. Herron and Langway here.

Simonsen_2013 ()

Accumulation units are $\text{kg/m}^2/\text{year}$

Veldhuijsen_2023 ()

Units are mm W.E. per year b_dot is meant to be accumulation over a reference period (20 years for spin up, 1 year for regular?) (not mean over the lifetime of a parcel)

graincalc (iii)

Evolve the grain size

This is the same as graingrowth except that we do not calculate for the surface grain, which is done by surfacegrain() function

surfacegrain ()

function to calculate the surface grain size at each time step

1.2.16 **plotter.py**

A script to do basic plotting operations with the CFM outputs.

1.2.17 **prefflow_snowpack.py**

This is a code imitating the preferential flow scheme developed in Wever et al. (2016) Water flow in firn with dual domain approach: - Richards Equation in Matrix Flow domain and Preferential Flow domain

Few differences with Wever 2016 as: -Use of a constant bigF value (part of the pore space allocated to each domain) -Runoff function from Zuo and Oerlemans 1996 -Possible to apply PFfreezing if cold wave penetrates from the surface -Possible to build up aquifer at end of domain -Use of upstream weighted mean to determine hydraulic conductivity at interfaces avoids oscillations for large mesh size -We use a changing bottom boundary if saturated layers accumulate: don't solve RE for the saturated layers at end of the firn column - don't solve RE in dry part of the domain

Density of last layer of the domain should always be ≥ 830

1.2.18 RCMpkl_to_spin

2/24/2021

This script takes a pandas dataframe containing climate data for a particular site and generates climate histories to feed into CFM as forcing.

The script resamples the data to the specified time step (e.g. if you have hourly data and you want a daily run, it resamples to daily.)

At present, spin up is generated by just repeating the reference climate interval over and over again.

YOU MAY HAVE TO EDIT THIS SCRIPT A LOT TO MAKE IT WORK WITH YOUR FILE STRUCTURE AND WHAT CLIMATE FILES YOU HAVE.

And, for now there are little things you need to search out and change manually, like the reference climate interval. Sorry!

@author: maxstev

RCMpkl_to_spin.**effectiveT**(T)
The Arrhenius mean temperature.

RCMpkl_to_spin.**makeSpinFiles**(CLIM_name, timeres='1D', Tinterp='mean', spin_date_st=1980.0,
spin_date_end=1995.0, melt=False, desired_depth=None,
SEB=False, rho_bottom=916)

load a pandas dataframe, called df_CLIM, that will be resampled and then used to create a time series of climate variables for spin up. the index of must be datetimeindex for resampling. df_CLIM can have any number of columns: BDOT, TSKIN, SMELT, RAIN, SUBLIM (use capital letters. We use SMELT because melt is a pandas function) Hopefully this makes it easy to adapt for the different climate products.

UNITS FOR MASS FLUXES IN THE DATAFRAMES ARE kg/m² PER TIME STEP SIZE IN THE DATA FRAME. e.g. if you have hourly data in the dataframe, the units for accumulation are kg/m²/hour - the mass of precip that fell during that time interval.

CFM takes units of m ice eq./year, so this script returns units in that format.

Parameters

- **timeres** (*pandas Timedelta (string)*) – Resampling frequency, e.g. '1D' is 1 day; '1M' for 1 month.
- **melt** (*boolean*) – Whether or not the model run includes melt
- **Tinterp** ('mean', 'effective', or 'weighted') – how to resample the temperature; mean is regular mean, 'effective' is Arrhenius mean; 'weighted' is accumulation-weighted mean
spin_date_st: float
decimal date of the start of the reference climate interval (RCI)
- **spin_date_end: float** decimal date of the end of the RCI

Returns

- **CD** (*dictionary*) – Dictionary full of the inputs (time, SMB, temperature, etc.) that will force the CFM. Possible keys to have in the dictionary are: 'time', which is decimal date; 'TSKIN' (surface temperature), 'BDOT' (accumulation, m i.e.), 'SMELT' (snowmelt, m i.e.), and 'RAIN'.
- **StpsPerYr** (*float*) – number of steps per year (mean) for the timeres you selected.
- **depth_S1** (*float*) – depth of the 550 kg m⁻³ density horizon (or other density; you can pick) this is used for the regrid module

- **depth_S2** (*float*) – depth of the 750 kg m⁻³ density horizon (or other density; you can pick) this is used for the regrid module
- **desired_depth** (*float*) – this is the depth you should set to be the bottom of the domain if you want to model to 916 kg m⁻³.

`RCMpk1_to_spin.toYearFraction` (*date*)
convert datetime to decimal date

1.2.19 re_snowpack.py

This is a code applying water flow in firn with single domain approach: - Richards Equation in Matrix Flow domain, no Preferential Flow domain

Essentially a simplification of the dual domain flow scheme

1.2.20 reader

Functions to read model inputs.

`reader.read_init` (*folder, resultsFileName, varname*)

Read in data for initial depth, age, density, and temperature to run the model without spinup

Parameters **folder** – the folder containing the files holding depth, age, density, and temperature

`reader.read_input` (*filename, StartDate=None*)

Read in data from csv input files

Parameters **filename** – name of the file which holds the accumulation rate data

Return input_data vector of field of interest (e.g. temperature, accumulation rate from a specified csv file)

Return input_year corresponding time vector (in years)

1.2.21 regrid.py

Script to change the grid to have different resolutions at different depths.

`regrid.init_regrid` (*self*)

Used in `firm_density_spin` for the initial regridding.

`regrid.init_regrid22` (*self*)

Splits the column in 5 grids: grid1(high res)-grid2(low res)-grid22(v. low res)-grid23(low res)-grid3(high res)

Used in `firm_density_spin` for the initial regridding.

`regrid.regrid` (*self*)

Called in both `firm_density_spin` and `firm_density_nospin`

There are 3 subgrids in the regrid module. Grid 1 is the high resolution grid near the surface. Grid 2 is the lower resolution grid at greater depths; a user-defined number of nodes (`self.c['nodestocombine']`; refer as NTC here) are combined occasionally (every NTC time steps) to make one new node within grid 2. Grid 3 is at the bottom and has split up one grid 2 node back into a high-resolution grid (1 node into NTC nodes), which can be removed at each time step to keep the model Lagrangian.

the variable `gridtrack` keeps track of which subgrid each node is in.

`regrid.regrid22` (*self*)

Called in both `firm_density_spin` and `firm_density_nospin` 5 grids:

grid1 -> high resolution determined by accumulation events grid2 -> low resolution by merging the batch of lowest nodestocombine layers of grid 1 grid22 -> very low resolution by merging the batch of lowest multnodestocombine layers of grid 2 grid23 -> low resolution by splitting the lowest layer of grid22 in multnodestocombine thinner layers New layer of grid23 is formed only when their stock is empty grid3 -> high resolution by splitting the lowest layer of grid23 in nodestocombine layers

gridtrack keeps track of which grid each layer is in

`regrid.regrid22_reciprocal(self)`

Reciprocal of regrid22: must be called if we accumulate too many grid3 nodes because of heavy melting of surface nodes (problematic in ablation area) -> merge k batches of n1 grid3 nodes into k grid23 nodes (k is maximum nb of batches of n1 grid3 nodes available) if nb of layers in grid2 is <k:

-> calculate the nb of supplementary grid2 layers required -> calculate xx: the number of grid22 layers that must be split to provide the supplementary grid2 layers -> merge xx batches of n2 grid23 layers into xx grid22 layer -> divide xx grid22 layers into xx*n2 grid2 layers

-> divide k grid2 layer into n1 grid1 layer

5 grids: grid1 -> high resolution determined by accumulation events grid2 -> low resolution grid22 -> very low resolution grid23 -> low resolution grid3 -> high resolution

gridtrack keeps track of which grid each layer is in

1.2.22 siteClimate_from_RCM

1.2.23 solver

Functions to solve the diffusion equation

`solver.A(P)`

Power-law scheme, Patankar eq. 5.34

`solver.F_upwind(F)`

Upwinding scheme

`solver.apparent_heat(z_edges, Z_P, nt, dt, Gamma_P, phi_0, nz_P, nz_fv, phi_s, mix_rho, c_vol, LWC, mass_sol, dz, ICT, rho_firn, iii=0)`

transient 1-d diffusion finite volume method

This is for standard heat (no liquid water), isotope, and air diffusion. If there is liquid water is should use the enthalpy solver.

Parameters

- **z_edges** –
- **Z_P** –
- **nt** –
- **dt** –
- **Gamma_P** –
- **phi_0** –
- **nz_P** –
- **nz_fv** –
- **phi_s** –

Return phi_t

`solver.solver(a_U, a_D, a_P, b)`
function for solving matrix problem

Parameters

- **a_U** –
- **a_D** –
- **a_P** –
- **b** –

Return phi_t

`solver.transient_solve_EN(z_edges, Z_P, nt, dt, Gamma_P, phi_0, nz_P, nz_fv, phi_s, mix_rho, c_vol, LWC, mass_sol, dz, ICT, rho_firn, iii=0)`
transient 1-d diffusion finite volume method for enthalpy

This is for heat diffusion when there is liquid water present. It uses a source term for the latent heat associated with the liquid water.

Parameters

- **z_edges** –
- **Z_P** –
- **nt** – number of iterations; deprecated (now uses while loop)
- **dt** – time step size
- **Gamma_P** –
- **phi_0** – temperature profile [C]
- **nz_P** –
- **nz_fv** –
- **phi_s** – surface temperature [C]

:param g_liq :param c_vol: [J/m3/K] 'volume-averaged specific heat of mixture', or rho * cp. (so really heat capacity)

Return phi_t

The source terms S_P and S_C come from the linearization described in Voller and Swaminathan, 1991, equations 31 and 32. and Voller, Swaminathan, and Thomas, 1990, equation 61

`solver.transient_solve_TR(z_edges, Z_P, nt, dt, Gamma_P, phi_0, nz_P, nz_fv, phi_s, tot_rho, c_vol, airdict=None)`
transient 1-d diffusion finite volume method

This is for standard heat (no liquid water), isotope, and air diffusion. If there is liquid water is should use the enthalpy solver.

Parameters

- **z_edges** –
- **Z_P** –
- **nt** –
- **dt** –

- **Gamma_P** –
- **phi_0** –
- **nz_P** –
- **nz_fv** –
- **phi_s** –

Return phi_t

`solver.w(airdict, z_edges, rho_edges, Z_P, dZ)`

Function for downward advection of air and also calculates total air content.

1.2.24 sublim.py

Script for sublimation.

`sublim.sublim(self, iii)`

Sublimation of the surface layers, partially based on melt.py We don't do anything energy-wise (no modification of Tz) Layers are sublimated in turn, starting with the surface layer Liquid water is sublimated before the ice matrix

1.2.25 writer

writer.py

Functions for writing model outputs.

`writer.SpinUpdate(self, mtime)`

Overwrite the variables in the spin file to whatever they are at time = mtime

Parameters `mtime` (*float*) – Time (model time) at which the

`writer.write_nospin_hdf5(self, Mout_dict, forcing_dict=None)`

Write the results from the main model run to hdf file.

Parameters `Mout_dict` (*dict*) – contains all of the model outputs; each key is the name of the output

`writer.write_spin_hdf5(self)`

Write the model outputs to hdf file at the end of spin up.

1.3 Extra information about running the CFM

Here is some extra information you might use when running the CFM.

1.3.1 Using the CFM's doublegrid functionality

The CFM's numerical scheme is Lagrangian, and each snowfall event gets a new layer (node) added on (and one is removed from the bottom). This creates a lot of nodes (slow!) when time steps are small. So, we developed a 'regrid' or 'doublegrid' function, which combines some specified number of nodes at a specified depth to reduce the number of nodes. Then, to improve this further, there is an additional node combining depth to make an even-lower resolution grid.

Here is Vincent Verjans' description of this function:

This new scheme is very similar to the original regrid scheme. The difference is that it uses a coarser resolution below grid2, in a grid that I called grid22. So now, you can have a transition in vertical resolution at two different depths. We still have grid1 at high resolution, then grid2 at low resolution and then grid22 at very low resolution. Below grid22, there is the grid23 which is again the same resolution as grid2. As before, the grid3 provides the stock of layers to be removed at each accumulation event. We have thus 5 different grids now: grid1: high resolution determined by accumulation events grid2: low resolution grid22: very low resolution grid23: low resolution grid3: high resolution

First, we proceed to an initial gridding in `firn_density_spin`. This splits the model domain into grid1 - grid2 - grid22 - grid23 - grid3. The transition depth and the number of nodes to merge from grid1 to grid2 are given by the json entries "grid1bottom" and "nodestocombine" as before. In addition, you can now include in the json the entries "grid2bottom" and "multnodestocombine". The former defines the transition depth from grid2 to grid22 and the latter determines how many nodes of grid2 are merged in a node of grid22. So, 1 node of grid22 is made of ("nodestocombine"x"multnodestocombine") nodes of the grid1. The grid23 nodes are formed of the same number of initial nodes merged together as the grid2 nodes. And the grid3 nodes are formed of a single initial node (as before). Note that by setting multnodestocombine to 0, the regrid scheme will work exactly as before and there won't be any grid22 and grid23.

Second, in the `firn_density_nospin` run, `regrid22` is called every time the stock of grid3 nodes is empty. At this point, we merge nodes of grid1 into a grid2 node and we split a grid23 node into grid3. But when the stock of grid23 layers is empty, this does not work anymore. So, if this is the case, we merge nodes of grid2 into a node of grid22 and we split a node of grid22 into nodes for grid23. After that, we can again split a grid23 node into nodes for grid3. There is a good reason we use the grid23 to form grid3 nodes and we don't split a grid22 node into grid3 nodes. The grid22 nodes are very thick. If we split a single grid22 node into "nodestocombine" nodes for grid3, the grid3 nodes would be thicker than the grid1 nodes. On the long term, that would cause a thinning of the total CFM domain because every time a grid1 node is accumulated, a grid3 node is removed. By using grid23, we effectively split a single grid22 node into ("nodestocombine"x"multnodestocombine") nodes for the grid3.

1.3.2 Morris and Wingham (2014) fix

The Morris and Wingham (2014) paper describes a firn-densification model derived from repeat high-vertical-resolution borehole density logs. There was an error in that paper, which Liz Morris alerted me to. She sent a corrected description of the model, which you can download [here](#).

- View [Morris_CorrectionstoDensificationPaper.pdf](#).

Reference: Morris, E. M., and Wingham, D. J. (2014), Densification of polar snow: Measurements, modeling, and implications for altimetry, *J. Geophys. Res. Earth Surf.*, 119, 349– 365, doi:10.1002/2013JF002898.

CHAPTER 2

Contributing

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

c

constants, [18](#)

d

diffusion, [18](#)

f

fcts_snowpackflow, [19](#)

firn_air, [21](#)

firn_density_spin, [22](#)

h

hl_analytic, [23](#)

i

isotopeDiffusion, [23](#)

m

melt, [23](#)

merge, [24](#)

p

physics, [24](#)

prefflow_snowpack, [26](#)

r

RCMpk1_to_spin, [27](#)

re_snowpack, [28](#)

reader, [28](#)

regrid, [28](#)

s

solver, [29](#)

sublim, [31](#)

w

writer, [31](#)

Symbols

`__init__()` (*firn_air.FirnAir method*), 21

A

`A()` (*in module solver*), 29

`apparent_heat()` (*in module solver*), 29

`Arthern_2010S()` (*physics.FirnPhysics method*), 25

`Arthern_2010T()` (*physics.FirnPhysics method*), 25

B

`Barnola_1991()` (*physics.FirnPhysics method*), 25

`Breant2017()` (*physics.FirnPhysics method*), 25

`Brils_2022()` (*physics.FirnPhysics method*), 25

`bucket()` (*in module melt*), 23

C

`combineCFM()` (*in module fcts_snowpackflow*), 20

`constants` (*module*), 18

`Crocus()` (*physics.FirnPhysics method*), 25

D

`darcyscheme()` (*in module melt*), 23

`diffusion` (*module*), 18

`diffusivity()` (*firn_air.FirnAir method*), 21

E

`effectiveT()` (*in module RCMpkl_to_spin*), 27

`enthalpyDiff()` (*in module diffusion*), 18

`entrysuction()` (*in module fcts_snowpackflow*), 20

F

`F_upwind()` (*in module solver*), 29

`fcts_snowpackflow` (*module*), 19

`firn_air` (*module*), 21

`firn_air_diffusion()` (*firn_air.FirnAir method*), 21

`firn_density_spin` (*module*), 22

`FirnAir` (*class in firn_air*), 21

`firnConductivity()` (*in module diffusion*), 19

`FirnDensitySpin` (*class in firn_density_spin*), 22

`FirnPhysics` (*class in physics*), 24

G

`gasses()` (*in module firn_air*), 21

`Goujon_2003()` (*physics.FirnPhysics method*), 25

`graincalc()` (*physics.FirnPhysics method*), 26

`GSFC2020()` (*physics.FirnPhysics method*), 25

H

`heatDiff()` (*in module diffusion*), 19

`heatDiff_highC()` (*in module diffusion*), 19

`heatDiff_LWCcorr()` (*in module diffusion*), 19

`heatDiff_Teff()` (*in module diffusion*), 19

`Helsen_2008()` (*physics.FirnPhysics method*), 25

`hl_analytic` (*module*), 23

`hl_analytic()` (*in module hl_analytic*), 23

`HL_dynamic()` (*physics.FirnPhysics method*), 25

`HL_Sigfus()` (*physics.FirnPhysics method*), 25

I

`init_regrid()` (*in module regrid*), 28

`init_regrid22()` (*in module regrid*), 28

`isoDiff()` (*isotopeDiffusion.isotopeDiffusion method*), 23

`isotopeDiffusion` (*class in isotopeDiffusion*), 23

`isotopeDiffusion` (*module*), 23

K

`KuipersMunneke_2015()` (*physics.FirnPhysics method*), 25

L

`layerequaliser_eq()` (*in module fcts_snowpackflow*), 21

`lengthendom()` (*in module fcts_snowpackflow*), 21

`Li_2004()` (*physics.FirnPhysics method*), 25

`Li_2011()` (*physics.FirnPhysics method*), 25

`Li_2015()` (*physics.FirnPhysics method*), 26

Ligtenberg_2011() (*physics.FirnPhysics method*), 26
LWC_correct() (*in module diffusion*), 18

M

makeSpinFiles() (*in module RCMpkl_to_spin*), 27
Max2018() (*physics.FirnPhysics method*), 26
Max2018b() (*physics.FirnPhysics method*), 26
melt (*module*), 23
merge (*module*), 24
mergeall() (*in module merge*), 24
mergenotsurf() (*in module merge*), 24
mergesurf() (*in module merge*), 24
Micedryer() (*in module fcts_snowpackflow*), 19
Morris_HL_2014() (*physics.FirnPhysics method*), 26
Mrefreezing() (*in module fcts_snowpackflow*), 19
Msatexcess() (*in module fcts_snowpackflow*), 19

N

NPtrid() (*in module fcts_snowpackflow*), 20

P

PFleave() (*in module fcts_snowpackflow*), 20
PFleaveheat() (*in module fcts_snowpackflow*), 20
physics (*module*), 24
Picedryer() (*in module fcts_snowpackflow*), 20
porosity() (*firm_air.FirnAir method*), 21
prefflow_snowpack (*module*), 26
Prefreezing() (*in module fcts_snowpackflow*), 20
Psatexcess() (*in module fcts_snowpackflow*), 20

R

RCMpkl_to_spin (*module*), 27
re_snowpack (*module*), 28
read_init() (*in module reader*), 28
read_input() (*in module reader*), 28
reader (*module*), 28
regrid (*module*), 28
regrid() (*in module regrid*), 28
regrid22() (*in module regrid*), 28
regrid22_reciprocal() (*in module regrid*), 29
restrictdom() (*in module fcts_snowpackflow*), 21
runoff() (*in module fcts_snowpackflow*), 21

S

Simonsen_2013() (*physics.FirnPhysics method*), 26
solver (*module*), 29
solver() (*in module solver*), 30
SpinUpdate() (*in module writer*), 31
splitCFM() (*in module fcts_snowpackflow*), 21
sublim (*module*), 31
sublim() (*in module sublim*), 31

surfacegrain() (*physics.FirnPhysics method*), 26

T

TDMA solver() (*in module fcts_snowpackflow*), 20
time_evolve() (*firm_density_spin.FirnDensitySpin method*), 22
toYearFraction() (*in module RCMpkl_to_spin*), 28
transient_solve_EN() (*in module solver*), 30
transient_solve_TR() (*in module solver*), 30

V

Veldhuijsen_2023() (*physics.FirnPhysics method*), 26

W

w() (*in module solver*), 31
write_nospin_hdf5() (*in module writer*), 31
write_spin_hdf5() (*in module writer*), 31
writer (*module*), 31