Latest Cryogrid Update

8th of December 2022

1 General

- contains mainly the changes of Thomas, Robin, Kristoffer from ITCH
- implementation into the spatial runs including clustering by Sebastian, same idea but slightly changed
- further testing needed, but it doesn't crash
- idea behind it: terrain class that processes any spatial information (altitude, slope, aspect, horizon angle)
- originally this was on the level of *tile*, now it has been put in *run_info* level (as soon as we run spatially distributed runs it needs to be on the *run_info* level anyway)
- it's an object called *spatial*, which creates and contains spatial information and overwrites the tile's parameter altitude/slope/aspect/.. in the parameter file through the *run_info*. therefor no structural changes at the *tile* level (thus old runs should still work)
- this is then passed to the forcing class, which modifies the forcing data according to terrain, and does any spatial interpolation
- run parameters are now given at the right level, i.e. spatial parameters (lat, lon, altitude, slope etc.) are given in the *spatial* class, rahter than in *forcing* or *tile*

There are two main ways of applying the model:
1) apply it to a point (what is mostly being done)
2) spatial simulations (where more classes are involved)

2 Point Simulations

- *run_1d_standard* becomes *run_1D_point*
- has tile class but also *point class*, which is the spatial for point simulations (if point class is left empty in the param, then this is the same setup as before the CryoGrid update)
- thus all spatial info (lat, lon, area) have been moved into spatial class
- if you want to give spatial information you have to define a point class
- 3 point classes are currently available: *point_simple* and *point_slope* and *point_DEM*
- *point_simple*: implementation like it was before the update incl. lat, lon, alt (has default values for all else, i.e. no slope, no shading etc.)
- *point_slope*: if you want terrain: horizon angles (points), it calculates the sky view factor; slope, aspect
- *point_DEM* class: you give a list of (lat, long, area) and a DEM (geotif format), and the class calculates the spatial parameters. Note that the DEM has to be one file for the whole area
- reproject2utm : regridding to utm with a huge DEM is a problem, and is not neccesairy if you only need altitude (as for global runs).
- main diff to spatial runs: single point uniquely defined by the coordinates vs. spanning a grid before; for single point runs this will be more accurate as the coordinates are nor projected and interpolated on a given grid, which is the reason this feature is being kept despite the fact that one could practically also run a single point in the spatial simulation setup

3 Spatial Simulations

- *run_1d_standard* becomes *run_spatial_spinup* (you need a spinup here anyway, but in the run itself nothing else changes)
- the spatial class here is the *coordinate_system* class, which provides a list of coordinates in space
- so far two *coordinate_system* classes available: *lat_lon* and *coordinates_from_file*
- *lat_lon*: delta lat is chosen much smaller (Svalbard example / polar region)
- *coordinates_from_file*: any other coord. system
- *data_class* populates the point list from *coordinate_system* class. DATA_DEM does the same for all as the POINT_DEM class

- *data_class* contains e.g. DEM; should be easy to also add a glacier mask; but also albedo and whatever else one would want to automatically be used
- data class gets a list of lat lon and then provides these data; if you need you can write your own data class; e.g. forest global; you write a data class that includes a list of lat lon and a land cover type
- *mask* classes: choose areas to simulate according to a mask; normal mask classes are in a geographic sense; but there can also be data mask classes, which need the data to be processed first (e.g. only lowland/mask with altitude); mask classes are also used in ESA_CCI where the spatial run only runs where there is borehole validation
- assign tile property class: overwrite everything that is written in the provider class; this checks if dataset glacier is 1 (is essentially the same as editing the excel file)
- update 1 to 1: updates properties in the list , can be added depending on which datasets one has, e.g. field albedo from a albedo data class, then this will be updated automatically
- tag_out_w_number gets the run number with the tag, so that it can be identified
- there is a class now that does clustering: *run_spatial_spinup_clustering*

4 TOPOSCALE

- two classes available: *forcing_slope_seb_toposcale* and *forcing_slope_seb_toposcale_slice*
- available now: download procedure that produces standard format files in standard format, produced by request scripts (python scripts); these will be provided to everyone with instructions
- *forcing_slope_seb_toposcale*: entire dataseries is produced and you can compare the entire timeseries to your stations
- *forcing_slope_seb_toposcale_slice*: same but directly works on the raw data, so you give this the folder raw, you tell it if monthly/yearly/.. data, pressure levels to use; it reads data in at runtime, this is very fast and works well but you never get the full timeseries to compare your models to (good for optimization)
- calling datasets during runtime instead of as a preprocessing step is especially helpful in terrain development applications
- modify get_ERA5.m to just extract an area (cut-out-area)

5 DEM

- fast download of a globally available DEM (Copernicus GLO-30 Digital Elevation Model) via `https://portal.opentopography.org/raster?opentopoID=OTSDEM.032021.4326.3`
- select area, download geotif