Code

File structure

```
├── classes
│   ├── channel_hydrology.py
│   ├── channel_network.py
│   ├── parameterizations.py
│   ├── peat_hydro_params.py
│   ├── peatland_hydrology.py
│   ├── peatland.py
├── CNM_domain_creation
│   ├── canal_network_graph.p
│   ├── cwl_preprocess_data.py
│   ├── file_pointers.xlsx
│   ├── step1_create_channel_network_right_directions.py
│   └── step4_generate_networkx_graph.py
├── cwl_utilities.py
├── data
│   ├── canal_mesh_cells.gpkg
│   ├── dipwell_data_from_first_rainfall_record.csv
│   ├── extrapolated_DTM.tif
│   ├── interp_padded_peat_depth.tif
│   ├── invented_blocks_layer.shp
│   ├── mesh
│   │   └── mesh_0.03.msh
│   ├── reprojected_dipwells.gpkg
│   ├── sourcesink.xlsx
│   └── weather_station_coords.xlsx
├── ET
│   ├── estimate_daily_et.py
│   └── evapotranspiration_fao.py
├── initial_condition
│   ├── best_initial_zeta.p
│   ├── initial_day_dipwell_coords_and_measurements.csv
│   └── initial_day_dipwell_coords_and_measurements_far_from_canals.csv
├── mesh
│   └── geo_file_generation.py
├── output
│   ├── ET.png
│   ├── plot.py
│   └── precipitation.png
├── environment.yml
├── file_pointers.xlsx
├── find_initial_condition.py
├── hydro_masters.py
├── main.py
├── math_diff_wave.py
├── math_preissmann.py
├── parameters.xlsx
├── preprocess_data.py
├── calibration.py
├── README.md
├── read_preprocess_data.py
└── utilities.py
```

Main folder

- Python files
  - main.py -- This is the script that serves as the interface to running the simulations
  - hydro_masters.py -- As important as main.py, this is a file to hide away from main.py the high-level hydrological functions.
  - find_initial_condition.py -- A script similar to main.py which was used to compute the initial WTD raster. It compares several modelled WTD rasters with the first day dipwell measurements, and it selects the best fit (MSE). See the manuscript for more details.
  - math_diff_wave.py and math_preissmann.py -- Programs containing the lowest level canal water level hydrology functions. (These should arguably be on a separate folder of their own, but I had some problems with python imports and I gave up.)
  - calibration.py -- Post-processing. Script used to inspect and plot the results of the calibration process. I didn't remove it because it contains some functions that could be useful in the post-processing of the results.
  - utilities.py and cwl_utilities.py -- They contain some helper functions for the rest of the code that are too mundane to appear in the other scripts.
  - preprocess_data.py -- Contains helper functions for reading and doing some preprocessing on the data. This is called by some of the hydro scripts inside classes.
  - read_preprocess_data.py -- Probably just dead code that I was too afraid to simply delete.
- Other files
  - environment.yml -- Used to install the proper python environment with conda. See README.md
  - file_pointers.xlsx -- Interface that provides pointers to the model input data files and the output folder for model results.
  - parameters.xlsx -- Specifies parameters for the model. The peat hydraulic properties, a.k.a. 'PHP', take more than one set of values (in the example given here, 1 ... 8. This is because we often do not know the site's peat hyraulic properties, and are interested in testing several different ones. This is done during the calibration process. The main.py script then allows to run the simulations either with a single parameter (as one would do once the php are known), or with several parameters in serial or parallel computation. (Note that the parallel computation has not been used lately, and it may need some polishing).

Classes folder

It contains the code doing most of the hydrological heavy lifting. 2 files correspond to the channel hydrology and 4 to the peatland hydrology. The purpose was to logically separate the 'set-up' of the model (parameters, data, etc.) from the model itself. (This aim was more successfull in the peatland part than in the channel part). Then the hydrology part receives the necessary information from the 'set-up' part. The best way to understand the inheritance relationships is to follow the instantiation of the classes from the main.py script, and from the constructot functions `set_up_peatland_hydrology()` and `set_up_channel_hydrology()`.

In the case of the peatland, the 'set-up' part consists of:

- peat_hydro_params.py -- Simple class holding the parameters needed for the peat hydrology, most read from the parameters.xlsx file.
- parameterizations.py -- Definition of different parmeterization functions for the peat hyraulic properties, with all the necessary conversions between variables. It also contains some useful plotting functions.
- peatland.py -- Handles the reading and processing of datasets for the peatland hydrology.

And the hydrology part consists of the peatland_hydrology.py file. It consists of an parent, abstract class with two child classes:

- `RectangularMeshHydro`, the hydrology model in a rectangular grid generated on the go by FiPy,
- `GmshMeshHydro`, the hydro model in an unstructured triangular grid that has to be inputted to the model beforehand. The triangular mesh is the default and the one that I have been using for the past year. The rectangular mesh model is legacy code that I haven't deleted because it may be useful in the future. However, many of the functions are not well-tested (for instance, boundary conditions are most likely wrong) and it is not advisable to use it.

In the case of the channels, the channel_network handles the data processing, and it contains the functions that define the topology of the channel network. The channel_hydrology.py contains the hydrology, minus the math_diff_wave.py and/or the math_preissmann.py files which actually contain the solutions to the differential equations, and live outside of this class structure.

CNM_domain_creation folder

data folder

ET folder

initial_condition folder

mesh folder

output folder

UI. Input and output

Workflow

Pre-processing of canal network dataset

- RectangularMeshHydro class is missing many necessary functions, and it is not usable as it is. However, I left it there because although it is dead code, it is also dead code that may help in future builds, if ready to do some archaeology.
- hydro_masters.py cntains hydro functions at the highest level of abstraction. These are called from main.py.

Notes

- 'DEM' and 'DTM' are used interchangeably. They stand for Digital Elevation/Terrain Model. And they really mean 'raster of the peat surface height from a reference datum'.

- In the canal water level code, I use the name “y” to describe the CWL. It is essentially the same quantity as ‘zeta’ in the peatland side of the code (i.e., water level as measured from the local peat height). The reason for using a different name was to not accidentaly confuse the two.

- There are two parameters that seem to define the depth of the channel bottom, `channel_bottom_below_DEM` and `porous_threshold`, but they do two different things.

    - `channel_bottom_below_DEM`: CWL can never go below this one, or else numerical errors occur. It should be thought as the impermeable layer at the bottom of the channels. Since we don't have data about the shape and slope of channel beds (which is necessary in open channel flow simulations), it is assumed to be parallel to the peatland surface. This is the parameter that gives the depth of that parallel surface.

    - `porous_threshold`: In reality, between the channel bed and the impermeable surface described above, there is always a layer of peat (or other water-conducting material). In this model we use a non-linear friction parameter, $n_{Manning}$, to describe the different dynamics of water flow in the channels and in the underlying peat. This parameter is used to describe $n_{Manning}$.

- No-flux Neumann are always applied to the upstream nodes of the channel network. The user can choose between two behaviours at the downstream nodes of the channel network: fixed CWL dirichlet BC or no-flux Neumann BC.
    - Fixed CWL Dirichlet BC: `downstream_diri_BC = True` in the `CWLHydroParameters` class reation, and the value given to the parameter `y_BC_below_DEM` in the parameters file.
    - No-flow Neumann BC: `downstram_diri_BC = False` in the `CWLHydroParameters` class creation. The `y_BC_below_DEM` parameter is ignored.

- There are 3 available modes for the solution of the open channel flow equations, with varying degrees of functionality. They are chosen using the `model_type` flag in the `set_up_channel_hyrology` function, which initializes the relevant classes.
    - `diff-wave-implicit` and `diff-wave-implicit-inexact` are two functioning available solvers for the diffusive wave version of the open channel flow equations. One is based on the basic Newton method and the other in the accelerated Newton method. The latter uses an inexact Jacobian to increase efficiency, see paper and references therein for more details. The inexact method is the most performant, the one that I have been using for the past year, and the one that should be used by default.
    - `preissmann` uses the Preissmann method for the full open channel flow equations. It is outdated and may lack many of the features of the diffusive wave solvers, but is left here because the finite-differences code (stored in math_preissmann.py) might be of future use. For a good introduction on this classical method, see Cunge 1980 and Haahti xxx.

- Use the unstructured triangular mesh version of the peat hydrological model. The rectangular one is not well mantained legacy code.

References

url, bibitiem:

- The paper is in the discussion stage at Biogeosciences.

- Cunge 1980

- Haahti xxx