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 interface to run the simulations. It requires the flag `--ncpu x`, where `x` is the number of cpus that the user wants to run the simulation with. `x=1` is the default and will run the computations serially. When `x>1`, the simulations for several peat hydraulic properties will be run in parallel. It also accepts the flags `--yes-blocks` or `no--blocks` (default) to run the channel hydrology model with or without blocks.
  - 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

Here are stored the scripts necessary to produce the canal network from spatial data. See below under the Workflow section to learn how to use these.

data folder

Contains the data. Unsurprising.

ET folder

Usually we don't have any evapotranspiration measurements. I haven't had them in any customer project that I have processed. This folder contains the functions to model evapotranspiration from other (usually available) weather data with the Penman-Monteith equation. This was created by Samuli Launiainen.

In a separate document under the `/doc` folder I briefly explain how I calibrated it to tropical conditions.

initial_condition folder

Computing the initial conditions is one of the preparation steps for these simulations. In this folder I save all the data related to it.

In this case, the computations have been run from `find_initial_condition.py` (in the project folder). And the dipwell measurements to make comparisons to are stored in this folder, as well as the resulting `best_initial_zeta.p` in a python pickle format.

mesh folder

The code contained in this folder is experimental --but very well working!-- code in order to automate the gmsh mesh creation step. I have never used this in an actual project. Instead, I use the more 'manual' approach detailed in the Workflow section below: I create the mesh from QGIS.

Anyway, I leave this here just in case anyone wants to add this feature to future versions of this tool, in order to make it less human-dependent.

output folder

Collection of outputs generated by the model, as well as a script, `plot.py`, that I use to generate some of the figures. The name of the output folder can be changed from the `file_pointers.xlsx` file.

The WTD generated from the model are stored here, each in their own subfolder, see below.

UI. Input and output

I have made an effort so that *most* of the inputs and outputs are directed through the `file_pointers.xlsx` file in the project directory.

I say most but not all because some of the document and/or folder names are generated programmatically within the model. The output WTD rasters are the best example: depending on the parameter number, the blocked

status and possibly also other variables, the folder names in which they are stored change.

Workflow

Note: the version of QGIS that I am using is the latest stable as of December 2022: QGIS 3.24.

A typical customer project unfolds according to the following steps:

1. Get the data
2. Sanity check on the data. Are there NaNs? Are mean values of measured quantities reasonable? The usual stuff.
3. Prepare data and structures for the model.

- Compute ET.
- Prepare the mesh for the peat hydrology simulations.
- Prepare the channel network domain for the channel hydrology simulations.
- Compute WTD initial conditions.

4. Calibrate the peat hydraulic properties.
5. Run the model

3. Prepare the peat hydrology domain. Meshing.

Here we use `gmsh`, the standalone software package for creating meshes. `gmsh` should have been installed in the conda environment. To open it, activate the environment, i.e., run `conda activate hyrdotrope` in a terminal and run `gmsh`.

We also make use of the gmsh plugin for QGIS For more details, see this tutorial.

Step1: create a mesh from a raster (QGIS)

1. Vectorize raster area into a single polygon (hint: there are Qgis functions for this).
2. Create a mesh
    1. Go to Plugins>Gmsh>Generate a gmsh geometry file.
        1. If error about multiparts, use multipart to singleparts tool
    2. In mesh boundaries, select the single parts vector file just created from vectorization
    3. Leave None as the Mesh Size Layer
    4. Choose right projection
    5. Run all the processes suggested in the popup windows

Step2: resize mesh (optional)

Open gmsh. Once inside, open the .geo file generated by the gmsh plugin in QGIS in the previous step.

1. In the left panel, go to Mesh and click on 1D. Then on 2D.
2. To change the size of the mesh, 2 options: 1. Double right-click in the mesh>GLobal mesh size factor > choose number 2. OR go to Tools>Option>Mesh and under General tab modify Element size factor 3. After any of these, hit 1D and 2D as in step 1. The right mesh resolution depends on two factors: the area size and the computational power. Very small mesh cells result in very slow computations, but very large areas might require small enough cells so that the relevant water dynamics can be resolved. As a rule of thumb, mesh cells of the order of the DEM raster resolution (which is typically 100 m) are reasonable.

Note:

- With FiPy 3.4.2.1 and Gmsh 4.8.4, the default Gmsh exporting or saving options result in a Version 4 ASCII document, which fipy does not understand. Solution: Open the .msh in gmsh, File>Export>(choose .msh filename) > Save in Version 2 ASCII.
- Leave the options 'Save all elements' and ' parametric blablabla' unselected

If resize mesh not working

It might be because the original .geo file created by qgis has some too sharp edges. In that case:

- Smooth the (vector) area to be converted into .geo format in QGIS by using the Smooth algorithm. This will make edges of the surface smoother.
- Then, try meshing again in QGIS.

Prepare the channel network domain

Part 1. From open water shapefile to NetworkX graph

Four steps:

1. Infer water flow direction at each segment of the channel network based on the DEM at the two edges of the segment (QGIS + Python).
2. Segment the shapefile into dx-length lines, and store it in a single .gpkg file.
3. Snap blocks to the nodes of the network.
4. Take the output of Step 2 and create the NetworkX graph that will be used during the simulations. This includes the canal blocks.

Step 1 (QGIS + Python)

Input: original water bodies file Output: shapefile with water flow directions

- Reproject the waterbodies (canal network) shapefile to the project CRS.
- If the shapefile is in the LinestringZ format, run `Drop M/Z values` in QGIS to drop the Z value and return a Linestring format shapefile.
- Run `Split with lines` native QGIS tool. Choose the reprojected canal network shapefile in both fields.
- As a result, we get the canal network shapefile segmented at each junction. Save it as a .gpkg file
- Add relevant filepaths for Step 1 to the file_pointers.xlsx file under the CNM_domain_creation folder. The output file should be a .shp ESRI Shapefile´.
- Run the python script `step1_create_channel_network_right_directions.py`.

Step 2 (QGIS)

Input: .shp file from Step 1 with the right water flow directions. Output: .gpkg file of the canal network segmented to dx meters lines.

- Load the .shp file generated by the python script in step 1 above into QGIS
- (Optional) check water flow direction changing the style of the layer and selecting arrow.
- Run GRASS tool v.split. Choose Maximum segment lenght = xxx m (this is the dx of the CNM, be sure to match that in the code!), and the option Force segment to be exactly of given length, except for the last one

Step 3 (QGIS)

The position of canal blocks must coincide exactly with that of the nodes of the canal network. The reason for this is that NetworkX treats blocks as node attributes. For this:

- Select the shapefile output of Step2 (the channel net split at dx uniform segments) and apply the GRASS v.to.points algorithm.
- Use SAGA's `Snap points to points` in order to snap the blocks to the nearest channel network node.

Step 4 (Python)

Run the `step4_generate_networkx_graph.py` with the appropriate file pointers. It can be run as a command line executable, where the only available option is to bypass the block location file and to generate a graph without any blocks. This is only recommended if there is no such block file in the project, because it is always possible to run the model with or without blocks in the simulation phase. The output is a `.p` pickle file that is the input for the main program.

Additional details

The details of what happens inside these functions are the following:

- Read the .gpkg file with `geopandas`. Read the interpolated 10×10DTM with `rasterio`
- Iterate through the lines in the geopackage and assign to each line's ends the height from the 10×10DTM

- Use the height difference between the ends of the lines to give a direction to water flow. The ends of the lines are now nodes, and the lines, edges.
- Read the edge list and the node list as a graph in networkx.
- Remove "complex" junctions that cannot be described by the junction equations in the Preissmann scheme of the SaintVenants. (By complex junctions I mean junctions where the down or up nodes are the down or up nodes of more than one junction). Fixing these junctions is done by removing problematic edges.
- Also, fix the cases in which a down node of a junction is also a downstream node overall. This would result in 2 unreconcilable equations for the y of the downstream node. This is also done by removing conflicting edges.

Part 2. Get mesh cells that correspond to canals

I use QGIS to compute the mesh centers, save them to a file, then read them with python.

1. Import desired gmsh mesh as a shapefile: Plugins>gmsh>Convert a Gmsh mesh file into shapefiles. The shapefile will be a collection of geometries (triangles)
2. Extract the centers of the triangles with `Centroids`. You may want to save this as a geopackage file. This makes the rest of the computations much faster when the data is large.
3. `Nearest neighbour analysis` to get the information about mean adjacent cell distance in the gmsh mesh. This will be used in the next step, along with a simple geometrical idea, to get the mesh centers that are part of the canal network.
4. Use `Extract within distance` to get those mesh cell centers that are *close* to the canal network. The files to use are the canal network nodes and the mesh centroids. But how close is "close"? Well, if we assume an equilateral triangles mesh, the maximum distance from its center to any other point inside the triangle (i.e., the distance from the center to any of its vertices) is equal to the distance between cell centers in the mesh. So, in the field `Where the features are within` we have to write the mean distance between cell centers from step 3.

For example, in the last Forest Carbon project, the expected mean distance for the 0.03 scale factor mesh was 48.6 meters.

Calibration of the peat hydraulic properties (php)

The peat hydraulic properties are what determines the dynamics of water flow in peat. There are few studies that have direct measurements of phps in tropical peatlands, and these measures show great intra- and inter-site variability. Our strategy to cope with this uncertainty has been to 'calibrate' the model to specific conditions.

This calibration process consists in the following:

1. Choose a time-period where high quality (i.e., high-frequency) weather and dipwell data are available.
2. Startig from a common initial condition (for instance, the one obtained by running the initial condition script as detailed above), run the model forward during the specified time window, for different values of php.
3. Compare the WTD results with the dipwell data and choose the best fitting parameters. This is done with the `calibration.py` script.

This is usually a computationally expensive process. The chosen time window should not be too large, or the calibration process will take forever. This is a good step to try to use the parallel version of the model.

Other 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'.

- 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 contains hydro functions at the highest level of abstraction. These are called from main.py.

- 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.

- The data appearing here was obtained from the last customer project with Forest Carbon. The model needs data about block positions, even if later the user decides to switch off the option and compute the water dynamics without any block. For that reason, I needed to create the `data/invented_blocks_layer.shp` file, where I put some random points for the blocks.

- Due to confidentiality, the DEM and the peat depth raster maps will and shall henceforth be distributed separately from the automatically tracking git repository.

References

url, bibitiem:

- The manuscript in which the present hydrological model is described is in the discussion stage at Biogeosciences. You may find it here.

- Cunge, J. A., Holly, F. M., & Verwey, A. (1980). Practical aspects of computational river hydraulics. Pitman Advanced Pub. Program.

- Haahti, K., Younis, B. A., Stenberg, L., & Koivusalo, H. (2014). Unsteady Flow Simulation and Erosion Assessment in a Ditch Network of a Drained Peatland Forest Catchment in Eastern Finland. Water Resources Management, 28(14), 5175–5197. https://doi.org/10.1007/s11269-014-0805-x