THERMO-MORPHOLOGICAL MODEL MANUAL

VERSION 1.0

(Picture by Shaw Harisson, USGS)

Kevin de Bruijn
17/01/2025

Contents

Preface

I developed the thermo-morphological model as part of my MSc thesis with Deltares. The original idea came from ‘ARCTIC XBEACH’ (Ravens, Ulmgren, Wilber, Hailu, & Peng, 2017). Tom Ravens was also involved in this project as an advisory and member of the graduation committee.

A large part of this project consisted of the consolidation of the exiting work done with (and on) Arctic XBeach. In the context of my research, I also added/removed functionality, which I describe in my thesis (see the thermo-morphological model repository). As such, this model is substantially different from Arctic XBeach.

To facilitate further research using this model, I wrote this manual. This manual explains how to get the model code and required datasets, run simulations, and first steps towards post-processing.

Installation

All the model source code is publicly available on GitHub[1].

Configuration files and code used for post-processing are also available.

Cloning the repository

To use the model, one must first clone the source code from GitHub. This can be done using GIT software, or simply the GitHub desktop application.

GitHub Desktop

1. Open GitHub Desktop
2. Select File → Clone repository → URL
3. In your web browser, navigate to https://github.com/khdebruijn/thermo-morphological-model
    a. Select 'Code', and copy the URL
4. Enter the URL in GitHub Desktop, and choose a local file path

You now have the required code.

Install packages

The model runs through Python, which means you need a Python interpreter, and a list of packages.

I used Mambaforge[2] as a package manager.

I won't include guides on how to install Python and package managers here, as there are a bunch of resources online.

Once you have python installed, open the command prompt and navigate to the root folder of the thermos-morphological model directory.

Run the command 'pip install -r requirements.txt'. This will start an installation for all packages listed in the requirements file. If asked, type 'y' to confirm.

I tried to keep a list of packages and dependencies during my thesis. However, if any package is missing and the system throws and error, you can always install additional packages by typing 'pip install PACKAGE_NAME'.

[1] https://github.com/khdebruijn/thermo-morphological-model
[2] https://github.com/conda-forge/miniforge#mambaforge

Data

The model uses two types of data: initial conditions and a time series of forcing.

Initial conditions

Initial conditions are optional. They relate to the initial bed level. This bed level is only used if, in the configuration file, the parameter 'bathymetry.with_schematized_bathymetry' is set to 'False'.

In that case, you can define file names ('xfile' and 'depfile') in the same folder as your configuration, which are just 1D .txt files with a grid and a bed level. An example is given in the folder 'thermo- morphological model/runs/run_template/'.

If 'bathymetry.with_schematized_bathymetry' is set to 'True', the model with generate a schematized bed level upon initialization.

Forcing

The model always requires two types of forcing (filepaths can be defined in the configuration file):

- data.storm_data_path
- data.forcing_data_path

Both of these filepaths should lead to a .csv file, which contains a timeseries of required forcing data for at least the daterange that is being simulated, with intervals of 1 hour. For example, if a simulation should span all of 2019, these files should contain values for 01-01-2019 00:00:00, 01-01-2019 01:00:00, ..., 31-12-2019 22:00:00, 31-12-2019 23:00:00.

Each of these files should have a row for each hour in the daterange.

Storm Data

Storm data relates to hydrodynamic boundary conditions, and should be provided in the form of a .csv file that contains the following columns:

time	Hs(m),	Tp(s)	Dp(deg)	WL(m)

With:

- time: range of datetime values
- Hs(m): offshore significant wave height
- Dp(deg): peak wave direction[3]
- WL(m): water level

For each timestep, all columns should contain a value to force the model for that timestep.

[3] Value not actually used on current implementation due to 1D horizontal grid, so can be set to zero

Forcing Data

Forcing data relates to all other boundary conditions. We obtained these data from ERA5[4]. Descriptions are also available there.

The .csv file should include at least the following columns:

- time
- 10m_u_component_of_wind
- 10m_v_component_of_wind
- 2m_temperature
- mean_surface_latent_heat_flux
- mean_surface_net_long_wave_radiation_flux
- mean_surface_net_short_wave_radiation_flux
- sea_ice_cover
- sea_surface_temperature
- soil_temperature_level_1
- soil_temperature_level_2
- soil_temperature_level_3
- soil_temperature_level_4
- *soil_temperature_level_1_offs*
- *soil_temperature_level_2_offs*
- *soil_temperature_level_3_offs*
- *soil_temperature_level_4_offs*

These variables are each described on the ERA5 website, except for the *latter four*, which relate to the soil_temperature_level_X at the available four levels but at an offshore point.

Again, these forcing data should be converted to a timeseries in .csv format with a value for each column for each hour in a simulation.

[4] https://cds.climate.copernicus.eu/datasets/reanalysis-era5-single-levels?tab=overview

Running a simulation

Once the code is cloned, the Python environment set up, and the data obtained, you are ready to start simulating.

Create a subfolder in the ‘thermo-morphological model/runs/’ folder, and give it a custom name.

Then, copy the config.yaml file from the ‘thermo- morphological model/runs/run_template/’ folder into your run folder.

Configuration

The config.yaml file contains all information and customizable settings for your simulation.

This includes data paths to your initial/forcing data, start- and end dates, and much more. Values are set as Booleans (True/False), numeric, or strings (text in quotes). You can change values as much as you want, but keep the types consistent with the template.

A small explanation is given in the config.yaml file for each variable. Variables are categorized as:

- data
- model
- wrapper
- bathymetry
- xbeach
- thermal
- output
- sensitivity

Starting a simulation

There are several ways to run the simulation from any Python interpreter. I will explain my preferred option, which requires the least additional installations, below. However, feel free to use your preferred method.

1. Open your conda/miniforge command prompt
2. Navigate to the correct directory using (e.g., ‘cd ...../thermo-morphological model/’
3. Activate environment with packages (e.g., ‘mamba activate ENVIRONMENT_NAME’)
4. Start the simulation using the following command:
    a. ‘python main.py RUN_NAME’
        i. replace RUN_NAME with the name of the subfolder in the ‘thermo-morphological model/runs/’ containing your config.yaml file.
5. Your simulation will with initialization, and then loop through all timesteps, printing progress to the screen in the command prompt.
6. Note that simulations of each year of simulations will generate approximately 2GB of output by default.

Additional information

Any simulation starts with defining your configuration in config.yaml, and running the ‘main.py’ file while providing the run ID of your simulation to let it know which simulation to perform. The main.py file operates on the highest level and loops through all simulated timesteps. It calls specific code from other python files. These python files are located in the ‘utils’ subfolder.

The bulk of the developed code is located in the ‘model.py’ module. During initialization, the main.py file creates an instance of the **Simulation** class, which is completely defined in model.py. All functions of the **Simulation** class contain documentation. Other supporting functions are imported from ‘bathymetry.py’, ‘miscellaneous.py’, and ‘visualizaton.py’.

We used the main.py script to keep operations organized, while most of the executed code is located in ‘model.py’. If you would like to add your own code, we recommend that you create a new script to import code from, e.g., by creating a new **Simulation**-type class that inherits its properties from **Simulation**, and add this code in main.py. For question, you can contact me through email[5].

For post-processing, the ‘main_results.py’ script is ran. Again, this script loops through all (output) timesteps, and imports any required functions from another file (specifically, the results.py file).

[5] khdebruijn@tudelft.nl

Post-processing

Reading output

The location to which simulation output is written, is defined in the config.yaml file.

Output is generated in three ways:

1. For each output timestep (defined in the config.yaml), a NetCDF file is generated with the timestep number as filename.
    a. For example, '0000000000.nc' corresponds to the output generated right after initialization. NetCDF files can be opened using the 'xarray' package in Python, 'quickplot' (by Deltares), or other resources available online.
    b. This NetCDF file contains:
        i. a bunch of single parameters (not tied to any coordinate),
        ii. and arrays tied to the horizontal x-coordinate of the modelled transect.
2. For each timestep with XBeach active, the files output by XBeach are saved separately in a subfolder called 'xb_files', with subfolders for each timestep with XBeach active.
3. Upon initialization, a bunch of files are generated that relate to the simulation process. The following files are generated:
    a. text files containing all the timesteps with corresponding datetimes.
    b. Also, a csv files with timeseries of the ground temperature is generated.
    c. The solar flux calculator generates .txt file than contains a matrix, which maps a specific inclination and day-of-year to a solar-flux-factor[6].
    d. If specified in the config.yaml file, the output of the first few XBeach timesteps is also saved in NetCDF form, directly into the output folder.

Creating animations

I developed code to automatically loop through the output folder and generate a figure for each timestep, which shows the bathymetry over time, the 1D thermal models, the thaw depth, the surface heat fluxes, and the simulation progress.

The code required to generate these figures is present in the thermos-morphological model repository, and should thus already be cloned to your device. It can be ran in the following way:

1. Open your conda/miniforge command prompt
2. Navigate to the correct directory using (e.g., 'cd ...../thermo-morphological model/'
3. Activate environment with packages (e.g., 'mamba activate ENVIRONMENT_NAME')
4. Start the post-processing using the following command:
    a. 'python main_results.py RUN_NAME all XXX YYY'
        i. replace RUN_NAME with the name of the subfolder in the 'thermo-morphological model/runs/' containing your config.yaml file.
        ii. Replace XXX with the first x-coordinate, i.e., the left bound of the domain being plotted. We recommend a value of 1300 when using the default values for the schematized bathymetry.

[6] See thesis for further explanation.

iii. Replace YYY with the second x-coordinate, i.e., the right bound of the domain being plotted. We recommend a value of 1400 when using the default values for the schematized bathymetry.

Note on post-processing

I used a bunch of different jupyter notebooks (the .ipynb file extension) to read output, analyze results, and create figures. These are mostly specialized to my application and contain local file paths. The files are publicly accessible, but require some work to make use of. Feel free to contact me through email[5] for any questions.

Bibliography

Ravens, T., Ulmgren, M., Wilber, M., Hailu, G., & Peng, J. (2017). Arctic-capable coastal geomorphic change modeling with application to Barter Island, Alaska.