Geosci. Model Dev. Discuss.,
https://doi.org/10.5194/gmd-2020-409-RC1, 2021

Geoscientific Model Development Discussions
Open Access
EGU

Interactive comment on "SuperflexPy 1.2.0: an open source Python framework for building, testing and improving conceptual hydrological models" *by* Marco Dal Molin et al.

Philipp Kraft (Referee)

philipp.kraft@umwelt.uni-giessen.de

Received and published: 18 January 2021

The manuscript with the title "SuperflexPy 1.2.0: an open source Python framework for building, testing and improving conceptual hydrological models" by Marco Dal Molin et al, describes the structure and use cases of a toolbox to build conceptual hydrological models and test hypotheses about catchment behavior. The toolbox is based on an earlier approach SuperFLEX, but presents its translation into the programming language Python together with some new features.

The manuscript is well written , and I would like to leave language review to native speakers. The structure is mostly sound, but I have these major issues with the

Printer-friendly version

Discussion paper

manuscript and would like to have them addressed in a revised version:

1. The manuscript misses a discussion, where the reader can understand what kind of problems this new system can solve, that are not possible to tackle with the tools already available. What is the improvement of the status quo?
2. The mathematical framework, especially the integration of elements is not sufficiently explained
3. The user interface (in this case the API) of the model system does not comply with the norms and standards of the Python programming language.

Minor issues are some misunderstandings in the introduction and shortcomings in the explanation of the model element description.

1 General Comments

1.1 Missing discussion

The introduction explains that some of the existing frameworks are only suitable for lumped models, like MARRMOT, FUSE and PERSiST and stresses the importance of distributed conceptual models. However, model frameworks with that ability exist and are cited in the introduction: CMF, ECHSE, SUMMA and RAVEN. What problems exist with these frameworks, and how does SuperFlexPy solve these problems? The authors state in l. 210 as an aim for the ms: "Provide a broad discussion of how the SuperflexPy contributes to the toolkits available to the hydrological community, including existing flexible frameworks, in terms of intended scope of application, advantages, and limitations", but mention the capbilities only very briefly in the introduction. This should

be expanded and moved to the discussion to include SuperFlexPy in the comparison. The current sections 5.1 and 5.2 seems to be a good place (see specific comments). As author and maintainer of CMF I can give my view on the differences between CMF and SuperflexPy (references for information only, not intended as suggestions for the reference list of the manuscript) and would be very interested about differences and similarities with other frameworks.

Similarities:

- Model is composed as a Python script
- Well defined coupling with optimization / rejection framework (in both cases SpotPy)
- Object oriented design
- Interactive exploration of model behavior (via Python prompt)
- Unrestricted possibilities for spatial granularity
- Fine process granularity
- Open source, open access
- Available via PyPI
- User defined time stepping scheme

Features that SuperFlexPy has, but not CMF:

- Lag functions

Printer-friendly version

Discussion paper

- Toolbox extendible by users without compiling (only experimental and unpublished feature in CMF)

Features of CMF not present in SuperFlexPy:

- Calculation methods for ETpot (eg. Jehn et al., 2017)
- Implicit and explicit single and multistep solvers (eg. Kraft et al., 2011)
- Multiprocessor support for large model systems (eg. Wlotzka et al., 2017)
- Complex topology (cyclic bidirectional graph) (eg. Maier et al., 2017)
- Energy potential based flow descriptions (eg. Richards- and St. Venant equation etc.) (Maier et al., 2017; Windhorst et al., 2014)
- Tested solute and isotope transport (eg. Haas et al., 2013; Windhorst et al., 2014)
- Explicit run time loop for simple model coupling (eg. Kellner et al., 2017; Kraft et al., 2010)

1.2 Missing mathematical explanations

- The term numerical approximator is not well defined.
- What happens if the root finding procedure does not converge? Flexible time stepping or does the implementation stop with an exception? Typically happens with fast snowmelt or power law equations with a large exponent.
- The standard implicit euler method implementation with the Pegasus root finding algorithm should be explained briefly

- How do the solvers deal with discontinuous or not continuously differentiable flux equations? The problem is described by Knoben et al 2019's MARRMoT Paper, Ch. 2.4(https://doi.org/10.5194/gmd-12-2463-2019) - it is the reason why I gave up mimicking exisiting models with CMF.
- The system can use for the solution of single elements implicit solvers - the need for that was very well explained by the co-authors in their "ancient daemons paper". How the solutions of the elements are combined to a the response of the entire model is not explained. I guess, that some kind of operator split is employed, but how do they deal with non linear behavior between timesteps? Is the numerical error of the operator split somehow controlled? Is there some lag of fluxes that are routed through lower nodes in a network?

1.3 Programming interface

The programming interface has a number of quirks and behaviour outside the norms of the Python language. I guess it is unusual to request changes to the programing interface of the software presented, but both manuscript and software would be improved. Not knowing and following the Python standards was one of the biggest mistakes I did with CMF, and it is very difficult to change the programming interface later.

Leading underscore:

In all code examples, where behavior of superflexpy components is extended / changed (polymorphism for object oriented programmers) the authors of the framework indicate with a leading underscore "something". A leading underscore of a class member has a clear and well defined meaning in the Python community:

"_single_leading_underscore: weak "internal use" indicator. E.g. from M

Printer-friendly version

Discussion paper

import * does not import objects whose names start with an underscore." (https://www.python.org/dev/peps/pep-0008/#public-and-internal-interfaces)

Internal use means, class members with a leading underscore should nearly never be assessed by a user of the framework, neither for reading, nor for writing. The authors seem to understand the leading underscore to indicate the concept of a "protected" member in java, C# or C++, a concept that does not exist in Python.

Use of literals and implicit relations

Usage of literals to access parameter and state names instead of keyword arguments or class properties makes usage and composition of the model components harder to write. If "magic" literals are avoided, modern Python IDE's (integrated development environments) can help with code completion. If the framework can allow one of the following two alternatives to create a component, the IDE can help with code completion, instead of all purpose dictionaries. For network creation see comment l.469

```
linear_reservoir = LinearReservoir(
    k=0.1, S0=10.0,
    approximation=num_app,id='LR'
)
```

```
linear_reservoir = LinearReservoir(approximation=num_app,id='LR')\
    .parameters(k=0.1)\
    .states(S0=10.0)
```

The Splitter interface relates to the position of certain input datasets in a list, that does not exist explicitly. This is quite hard to follow and to read. The definition of the topology (falsely called "topography") of the Network class is redundant and uses the id-string

literals instead of the node objects directly. A very easy to read variant for the creation of the Network is explicit setting of the downstream node:

```
stgallen.downstream = appenzell
```

Another option would be to define the tree as a nested list structure. Each list contains the node and the left and right upsteam nodes, if present.

```
thur_catchment = Network([
    andelfingen,
    [halden,
     [stgallen,
      [appenzell]
      ],
     [jonschwil,
      [mogelsberg, mosnang]],
     [herisau],
     [frauenfeld, [waengi]]
     ]
])
```

Most modern programming environments (IDE) can mark typos in the node names with both versions presented here, and even employ code completion to help the user with typing, but this does not work for meaningful literals.

2 Specific Comments

132: RAVEN is clearly a framework for conceptual models (no energy potential / head based flow equations) for internal transport but CMF was originally developed for phys-

ically based models.

171 - 180: Here is a description of what RAVEN and SUMMA do missing.

180: CMF can model substance / isotope advective transport with adsorption without additional software, and reactive fluxes in coupled model approaches

187: Same fine granularity as CMF, RAVEN and MARRMoT.

210: See general comments

213 - 218: Link sections with the numbered aims of the paper

225: "An element can represent an entire catchment. . . ": From my understanding this might be technically true, if the catchment can be represented by a single process. However, the meaning of "element" in SuperFlexPy is simpler to understand if 225 is changed to: "An element represents a specific process within the catchment." If needed this sentence can be extended by "In special cases, this specific process covers the entire catchment behavior and a single element is sufficient for the model."

Eq1: To ensure mass conservancy, how can g_s ever be different to $\frac{dS}{dt} = X(t) - g_y(S(t), X(t); ?)$?

230: This is a bit unclear: does SuperFLEXpy support substance transport? If yes, only theoretically, or has it been tested already?

234: How would a multistate reservoir look like (mathematically)?

242: Please give an example or more concrete description, what a connection is in terms of hydrological processes. As they are needed later, please explain the splitter, junction and transparent elements here.

260: Here only weight is mentioned (as the area fraction) while in 310 weight and area are different things. Please explain more consistent

274: The Level 3 concept predates the given references by a long time, please use

more classical example (eg. TOPMODEL, HBV (not light), etc.)

276: Same as above, please give a reference to classic "Level 4" model, eg. SWAT, SHETRAN, etc.

295: This is a structure problem: You need the numerical solution of the ODE for the construction of the model, but the numerical solution is not yet explained. Please consider moving Section 4.3 up as a new section 2.2. However, section 4.3 does not explain the terms "numerical approximator" and "root finder" and how these work together. Secondly, the "numerical approximator" is given as a parameter to each reservoir element. How many instances of numerical approximators exist? One global object to solve the entire ODE-system or is the numerical approximator copied and is a non-shared object of the reservoir? Or is the numerical approximator more a kind of function without any notion of specific state? If each element is solved on its own, how is the whole system integrated, i.e. how does the operator split work? As the "standard" solver is the own implementation of ImplicitEuler with Pegasus root finder a short explanation of the math behind it is missing. How does the composition of integration solution work?

305: The definition of the routing is quite obscure. Somehow the routing is derived from a List[List[Element]]. Since "explicit is better than implicit" (Zen of Python) an alternative (obvious) way to define routing would be preferable. Otherwise some more explanation, how the nested list translates into a tree structure is needed.

310: What unit is used for area?

315: Topography is not the right term here, do you mean topology? The dictionary with id's to define the topology is not helpful when using IDE's with code completion

392: "Input fluxes" is ambiguous here: Input can either mean the direction of the flux (input flux adds water to the element) or in the information sense: input is an externally defined time series that may add or take water from the element. This ambiguity needs

to be addressed in the whole manuscript and the documentation.

399: User facing methods should not start with an underscore. There is no concept of "protected" in Python (see general issue 2)

411: ditto

427: Is there a better symbol for the snow reservoir than WR? What is W?

443: Here is an explanation of splitters and junction missing, because the "gaps" can only exists together with the multipath structure (or better: explain them in section 2).

448: What is the role of the upper_splitter here?

465: When a unit is composed from elements (why the term "layers"?), are those elements copied or referenced (must be copied, but is not mentioned in the manuscript)? If I count correctly, each unit has 4 states and 11 parameters. So each node has 8 states and 22 parameters?

466: Same question here: does assigning a unit to a node make a copy of the unit? If yes, I would understand "consolidated" a kind of template. As the code is given, the parameters for consolidated and unconsolidated are the same, how do you change their parameters, if the units are copied? If copied, the model has 80 states and technically 220 parameters, that could be tuned independently, correct? Of course, the parameters for each of the 10 consolidated units might be coupled, to be the same, by convention.

469: cf. general issue 2

490: An UML-class diagram of the main components would be helpful

506 - 519: See comments to line 295

540: In l. 163, "Interoperability with external software, for example for model calibration and uncertainty analysis" is claimed as one of the desired features of modeling code.

This is a good place to mention how SuperFlexPy can be interfaced with calibration / uncertainty packages like Ostrich, SpotPy, PEST etc.

546: This section would be much improved, if there would be a single working prototype in the documentation for it, otherwise it should be shortened substantially and its content moved into section 5.3 (future development).

559 - 584: The missing manuscript aim 3 should go here. Section 5.1 can be enhanced by citing articles dealing or struggling with the "right complexity" across framework boundaries

585 - 605: How do the limitations of SuperFLEXPy compare to the other hydrological frameworks, especially the similar ones: MARRMoT, RAVEN, CMF, ECHSE? (See general comment)

654: Missing link to the Github repository. Figure 12 should be deleted, as it conveys only little information. The role of binder for the framework is unclear.

658: Please write the DOI out here. BTW, the subtitle of the code release "**The** flexible language of hydrological modelling" (emphasis mine) is a bit bold, given that similar frameworks / domain languages are available since a decade.

Figures:

Fig 2: See general comments about literals

Fig. 5: Users should not access class fields starting with an underscore (see general comment 3). Please use comments to explain l.8 and l.9. Is self._fluxes and self._fluxes_python callable or are these values?

Fig 6: see general comment 3.

Fig 7: Catchment boundaries and HRU boundaries can be presented in a single map

Fig 8: Why does "W" denote snow?

Fig 11: Parametrization of the Splitters is unclear. Missing different parametrization of the two HRU templates (if they are templates in fact)

Fig 12: Not necessary, can be removed

3 References

(no need to include into manuscript)

Haas, E., Klatt, S., Fröhlich, A., Kraft, P., Werner, C., Kiese, R., Grote, R., Breuer, L. and Butterbach-Bahl, K.: LandscapeDNDC: a process model for simulation of biosphere-atmosphere-hydrosphere exchange processes at site and regional scale, Landscape ecology, 28(4), 615-636, 2013.

Jehn, F. U., Breuer, L., Houska, T., Bestian, K. and Kraft, P.: Incremental model breakdown to assess the multi-hypotheses problem, Hydrology and Earth System Sciences Discussions, 1-22, https://doi.org/10.5194/hess-2017-691, 2017.

Kellner, J., Multsch, S., Houska, T., Kraft, P., MĂźller, C. and Breuer, L.: A coupled hydrological-plant growth model for simulating the effect of elevated CO2 on a temperate grassland, Agricultural and Forest Meteorology, 246, 42-50, 2017.

Kraft, P., Multsch, S., Vaché, K., Frede, H. and Breuer, L.: Using Python as a coupling platform for integrated catchment models, Adv. Geosci, 27, 51-56, https://doi.org/10.5194/adgeo-27-51-2010, 2010.

Kraft, P., Vache, K. B., Frede, H.-G. and Breuer, L.: A hydrological programming language extension for integrated catchment models, Environmental Modelling and Software, 26, 828-830, https://doi.org/10.1016/j.envsoft.2010.12.009, 2011.

Maier, N., Breuer, L. and Kraft, P.: Prediction and uncertainty analysis of a parsimonious floodplain surface water-groundwater interaction model, Water Resources Research, 53(9), 7678-7695, 2017.

Windhorst, D., Kraft, P., Timbe, E., Frede, H.-G. and Breuer, L.: Stable water isotope tracing through hydrological models for disentangling runoff generation processes at the hillslope scale, Hydrology and Earth System Sciences, 18(10), 4113-4127, 2014.

Wlotzka, M., Heuveline, V., Klatt, S., Kraus, D., Haas, E., Kiese, R., Butterbach-Bahl, K., Kraft, P. and Breuer, L.: Parallel Multiphysics Simulations Using OpenPALM with Application to Hydro-Biogeochemistry Coupling, in Modeling, Simulation and Optimization of Complex Processes HPSC 2015, pp. 277-291, Springer, 2017.

Printer-friendly version

Discussion paper