Response to the reviews of "SuperflexPy 1.2.1: An open source Python framework for building, testing and improving conceptual hydrological models."

We thank the two reviewers and the Editor for their careful reading of the manuscript. Their additional insightful feedback and suggestions have helped us further improve the manuscript and address remaining issues.

In the remainder of this document, the original comments by the reviewers are in typeset in *blue and italics font* and our replies are typeset in black font. The two reviewers are referred to as PK (Dr. Philipp Kraft) and AR2 (Anonymous Referee #2).

Response to comments by the Editor (Dr. Andrew Wickert)

Dear authors,

Thank you for your diligent revisions. After carefully reviewing both referee reports, as well as your response to the referees, I agree with them about some significant additional improvements that the manuscript will require prior to consideration for publication in GMD. Referee 1 has several substantial concerns, and Referee 2 suggests minor revisions but does have one major concern regarding the focus on transport vs. its apparent non-implementation in the current version of your code base.

I look forward to seeing a revised manuscript following your work to address these comments.

We thank the Editor and the reviewers for acknowledging our effort in improving the manuscript addressing most of their earlier comments, which in particular helped us clarify the contribution of SuperflexPy and place it in a better context in relation to other hydrological models.

We appreciate the additional reviewers' comments, which we address in this document.

Response to comments by PK (Dr. Philip Kraft)

The manuscript has been improved in several areas, but especially the mathematical description needs improvement.

We thank the reviewer for a very careful review of the manuscript and for multiple detailed comments, all of them with clear technical merit. Motivated by the reviewer's comments on numerical aspects, we have made several enhancements to increase the functionality of SuperflexPy and better demonstrate its capabilities. In particular, we have implemented: (i) a new numerical approximator implementing the Runge-Kutta 4 method (within the constraints of the "constant-within-timestep" approximation of fluxes explained in point PK.2), (ii) a new root finder implementing a Newton-bisection method that uses the analytical derivatives of the fluxes.

We appreciate the interest of the reviewer in numerical aspects of hydrological modelling and, consequently, of SuperflexPy. As authors, we of course also share these interests, and have historically pursued some of them in previous publications. We have responded to all technical questions below, explaining the organization of SuperflexPy numerical implementation, how it is intended to operate and have provided a clearer description of its assumptions and limitations. This is certainly an important improvement to the presentation.

With that in mind, we now also note, both here and in the manuscript itself, that SuperflexPy is "primarily" intended for experimentation with the conceptual model structure, with "secondary" options to experiment with the numerical implementation. This choice is in line with the target audience of the paper (general hydrological/environmental modelers), as explained in point PK.11. For these reasons, in order to keep the manuscript focused, we have opted to preserve the overall balance of the presentation which focuses on these conceptual modelling aspects and their software implementation, and have avoided adding a large amount of numerical detail, which as we note below generally follows the recommendations from previous publications.

Further, some questions raised by the reviewer have prompted us to reflect more deeply on the assumptions made in "practical" numerical approximations. That is a large research topic in its own right – and while related to it is nonetheless distinct from the current paper, which focuses on the software implementation of a flexible structure model. As such, several of these questions deserve a separate study where they can be investigated – and reviewed – in appropriate depth. It would be an injustice to these questions to be tacked in somewhere in this paper, and presenting them without a suitably detailed (and therefore length) context could cause confusion to readers without the technical background of the reviewer. We hope the reviewer appreciates this perspective. In any case we once again thank the reviewer for eliciting these clarifications and reflection, which we agree and hope will reduce the potential for reader confusion. We also hope they can lead to follow up studies focused more specifically on numerical implementations.

We should also add here that we agree with the reviewer on the need for a better description and illustration (in Figure 12, now 13 in the re-submitted manuscript) of how SuperflexPy is integrated into the ecosystem of modern online software management tools, and on the need for the UML diagram to be included in the main text (Figure 12 in the re-submitted manuscript). Our responses to all these issues are provided below.

MP1 and MP2:

***PK.1:** The discussion issue (MP1, PK1.1) has been solved in the revision and the manuscript has been much improved, both by toning down the introduction and by expanding section 5. The same applies to MP2.*

We thank the reviewer for acknowledging our effort in improving the paper with respect to these points.

***PK.2:** MP3: This issue is not solved sufficiently and needs improvement prior to publication.*

We agree that numerical aspects are very important in a hydrological modelling software. For this reason, SuperflexPy is designed to provide a balance of efficiency and flexibility in the selection of numerical solvers within the constraints imposed by the assumed model architecture (DAG). We agree with the reviewer that some of these ideas were not so clear in the previous submission - therefore we take the opportunity to clarify them here and in the revised manuscript.

Before moving to the specific comments, we describe in a consolidated way the numerics of SuperflexPy and their relationship to the DAG assumption as well as additional approximations.

Our design of SuperflexPy is oriented towards facilitating experimentation with the conceptual model structure, including the number and connectivity of storage elements, the shape and parameterization of constitutive functions, spatial discretization, and so forth. For pragmatic reasons, we have made some assumptions in the numerical approximation that, while robust in their own right, do limit to some extent the numerical flexibility of the framework and have some implications when techniques such as adaptive time stepping are used. We thank the reviewer for identifying several of these limitations. Note also that these choices are based on our previous research publications and general experience with flexible modelling frameworks, which included detailed testing of multiple numerical algorithms (notably Clark & Kavetski, 2010; Kavetski & Clark, 2010). These earlier studies have indicated that the implicit Euler scheme with fixed time step is a robust choice for general hydrological modelling, which is the application that SuperflexPy is designed for. Nevertheless, SuperflexPy does offer some flexibility in the choice of numerical solvers, within the restrictions explained below.

SuperflexPy requires the conceptual model architecture to be defined as a directional acyclic graph (DAG), which implies some restrictions in the coupling of equations. If the model structure is not a DAG, then the model structure must be transformed in a DAG, e.g., by encapsulating the part of the structure that contains feedbacks into a self-contained new element. This is already discussed in section 5.2 of the paper and section 5.2 of the documentation.

Model structures without feedbacks (i.e., DAGs) offer several practical advantages, as already elaborated in Section 5.1.1 of the paper. From the numerical perspective, such models lend themselves to the simple "one-element-at-a-time" numerical solution approach, which reduces the solution of an ODE system to the solution of a sequence of multiple scalar ODEs. Note that, if the model structure is a DAG, the "one-element-at-a-time" approach per se does *not* introduce additional numerical errors.

However, within the "one-element-at-a-time" approach, we make the (additional) numerical approximation that the input fluxes into each element are constant within the model time step Δt. This approximation is consistent with the typical format of hydrological data, such as rainfall, PET, etc, which are tabulated in discrete steps (e.g., daily, hourly, etc). However, in our case we also apply this approximation to internal fluxes. This pragmatic approximation enables a further simplification of the solution procedure, because the output flux from each element becomes a scalar value - however it comes at the cost of introducing additional first-order discretization error, because the variation of internal fluxes within the time step Δt is ignored.

These first order approximations do not impact on time stepping schemes that are first order anyway (e.g., explicit/implicit Euler) and, which, at a given time step, use a single value of input fluxes to estimate a single value of output fluxes. The lack of impact on first order schemes is an appealing practical point because these time stepping schemes methods are commonly used in hydrological models and indeed are recommended for their general robustness (e.g., Kavetski & Clark, 2010). However, second and higher order time stepping schemes, as well as (adaptive) substepping schemes are impacted – because these approaches require input flux values at intermediate points within the time step. The impact of additional errors will reduce the overall accuracy back to first order (with respect to the full exact solution). However, they would not introduce any instabilities and indeed would still permit the advantages of adaptive time stepping in terms of facilitating convergence of the nonlinear root finder (connecting to reviewer's specific comment in point **PK.6**).

For completeness, we should also note that the "constant-within-timestep" approximation of fluxes is not per se a direct requirement for the "one-element-at-a-time" strategy (nor of the DAG assumption). Potentially, each element could output fluxes that vary within the time step Δt, allowing for a reduction (or even elimination) of these additional flux averaging errors. A more general implementation of SuperflexPy could adopt a different format for the fluxes – for example, using (instead of a single number) a look-up array of values, a function, or another data structure that allows for "time queries", etc. This approach would (potentially) re-enable higher order schemes and adaptive time stepping schemes to reach their formal asymptotic order of accuracy. However, we have not pursued these options in the current version of SuperflexPy, because the asymptotic order of accuracy (i.e., the order of accuracy as $\Delta t \to 0$) is far from the main concern when hydrological models are applied with input data resolution as course as daily. Moreover, for say a Runge Kutta 4 solver to achieve genuine 4th order accuracy would require a 4th order approximation of the rainfall and PET time series, which as such as impossible with practical data. For these and other reasons we favour the current numerical implementation based on the fixed step implicit Euler scheme, - indeed this implementation was used in all previous SUPERFLEX-F90 case studies.

In summary, the numerical implementation within SuperflexPy has the following characteristics:

- If each element is solved using a non-adaptive first order method (e.g., implicit Euler without substepping), then no additional approximation error is introduced; an explanation is given in point **PK.7**.
- If an element is solved using a higher order method and/or an adaptive-step method, then its outputs are averaged over the time step before they are used as the inputs to a downstream element, which does introduce additional numerical approximation error. As noted by the reviewer, this error will not be "seen" by the adaptive time stepping. This limitation does not affect stability but impacts on the overall accuracy (truncation error).

Therefore, the SuperflexPy user can still employ adaptive time stepping and higher order methods, albeit within the stated limitations. We agree these points are pertinent and were not sufficiently clear in the previous response RC1.8 and in section 5.2 (“Sequential solution of the elements”) of the documentation. We have now added a brief description of these issues in section 4.3 of the paper and section 5.2 of the documentation.

We now respond to the specific points raised.

PK.3: *RC1.4: Still unclear why the term numerical approximator and not integrator or solver is used (as in the math-lit), but an improvement is available.*

Our implementation of SuperflexPy proposes a specific architecture for the (numerical) solution of the ODEs, which considers two functionally distinct procedures, named the "numerical approximator" and the "root finder".

The “numerical approximator” routine, which is an instance of the abstract class `NumericalApproximator`, is responsible for creating a discrete **approximation** of the differential equation. For example, when using implicit Euler, the numerical approximator routine `ImplicitEulerPython` transforms the differential equation

$$\frac{\mathrm{d}S}{\mathrm{d}t} = P - kS^{\alpha}$$

into the algebraic function

$$\mathrm{f}(S_{t+1}) = \frac{S_{t+1} - S_t}{\Delta t} - P + kS_{t+1}^{\alpha}$$

where S_{t+1} and S_t is the state of the reservoir at the end and beginning of the time step, respectively; P is the precipitation over the time step and k and α are parameters.

The "root finder" routine, which is an instance of the class `RootFinder`, is then responsible for finding the value of S_{t+1} such as $f(S_{t+1}) = 0$. For example, SuperflexPy offers the root solver `PegasusPython`, which implements the Pegasus algorithm.

The separation of the overall ODE solution into these two components simplifies the implementation of new ODE solvers by allowing cleaner re-use of existing procedures. For example, a numerical approximator can be used with a different root finders with no changes to its code. In addition, the same root finder could be used with different numerical approximators.

Section 5.1.1 and 5.1.2 of the documentation indicate how to implement new numerical approximators and root finders by extending the abstract classes `NumericalApproximator` and `RootFinder`. When such architecture is adopted, the new code needed reduces to the definition of the algebraic approximation of the ODE (for the numerical approximator) and to the implementation of the algorithm for finding its solution (for the root finder) – i.e., avoiding the need to implement all the (considerable) auxiliary code that is needed to actually solve the ODE (e.g., looping in time, interfacing for the `ODEsElement`, etc.).

In terms of the choice of specific names "numerical approximator" and "root finder", we agree that many potential alternatives could be possible. However, note that the numerical approximator on its own does not actually **solve** or **integrate** the differential equation. For this reason we prefer to not use the terms "solver" or "integrator", which in our experience have a different meaning in the literature. Potentially, the name "integrator" could be assigned to the combined usage of "numerical approximator" and "root finder" to solve the differential equation, but in our opinion this is not really necessary and would just complicate the nomenclature.

As part of the manuscript revisions, we have enhanced section 4.3 of the paper and have restructured section 5.1 of the documentation. In particular the new section 5.1.3 explains how to implement a numerical solver for the ODEs from scratch, i.e. bypassing the numerical approximator and the root finder architecture and interfacing directly with the `ODEsElement`.

PK.4: *RC1.5: This was not meant as an implementation question: In cases of rapid, non-linear changes (eg Power-Law-Equation with an exponent > 4), implicit solvers often fail to converge for a specific time step – even A-stable solvers like the implicit Euler. Complex solvers (eg. RKF 45, CVODE and many others) use an adaptive time stepping scheme, which is, as the authors explain in their answer to RC1.8, not suitable for SuperFlexPy. This is important information and should be mentioned in the section 4.3*

For clarity, the original question (RC1.5) was:

> “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.”

And our reply was:

> “We agree this is an important point. In the Python implementation, we raise an exception; when using Numba, we return None because Numba does not support exceptions. In both cases user notices the problem (either the simulation crashes or the result is plenty of None values).”

We apologize for having mis-understood this question. The wording “does the implementation stop with an exception?” suggested to us it was a question about the behavior of the implementation.

We are aware that, in some situations, numerical solvers may fail to converge for a given time step size. As noted by the reviewer, adaptive time stepping in such cases would reduce the step size and attempt the step again.

In SuperflexPy, if the implemented fixed-step solvers (implicit or explicit Euler) fail to converge they do not fall back on other solvers (e.g., reducing the time step or changing the solver algorithm) but simply fail (i.e., raise an exception in the Python implementation or return `None` in the Numba implementation).

However we should add that the "one-element-at-a-time" strategy employed in SuperflexPy (see **PK.2**) enables the use of robust solvers that operate on a single ODE at a time. In such cases, the root finder also operates on a single algebraic equation at a time. Moreover, SuperflexPy proposes root finders that implement bracketing methods, which are guaranteed to converge (to a tolerance within the common constraints of floating point arithmetic) as long as the initial solution bounds are known. The bounds of the solution can be constructed from the reservoir equations and are provided by the flux methods. For example, the storage cannot be negative and cannot exceed the current storage plus all the input. In our experience with the earlier SUPERFLEX-F90, this setup achieves a robust numerical behavior.

Furthermore, as now clarified in **PK.2**, SuperflexPy users can develop adaptive time stepping schemes for the single elements with the "constant-flux-within-a-timestep" limitations already discussed earlier in point **PK.2**. For this reason, it is also possible to overcome convergence problems by employing adaptive time stepping to the solution of the single equations.

We have reflected these points in the updated manuscript (section 5.1.5) and documentation (section 5.1).

PK.5: *RC1.6: Reference is provided now, but the properties of the algorithm should be stated in the supplemental material (limits and speed of convergance). The algorithm is not explained or just described as a mixture of regula falsi with the secant method.*

We have now added the following content to the Documentation (Section 5.1):

> The Pegasus algorithm is a bracket-based nonlinear solver similar to the well-known Regula Falsi algorithm. It employs a re-scaling of function values at the bracket endpoints to accelerate convergence for strongly curved functions. The authors of the paper (Dowell & Jarratt, 1972) claim that the algorithm exhibit superior asymptotic convergence properties to other modified linear methods.

The reference (Dowell & Jarratt, 1972) provides a complete algorithmic description of the Pegasus root finder. The algorithm is implemented exactly as described in the reference; hence, we prefer to avoid duplication of this content.

PK.6: *RC1.7: After careful reading of Supl-Section 5.1, I cannot find the information from this answer. The need for smoothing when using the implicit solver must be mentioned as a one-liner in the main text.*

For clarity, the original question (RC1.7) was:

> 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.

And our reply was:

> This is a pertinent point. Generally speaking the SuperflexPy philosophy is to use smooth flux functions. This may include applying smoothing to otherwise discontinuous formulations – please see previous publications such as Kavetski and Kuczera (2007).
>
> That said, if a user wanted to perform modelling experiments with discontinuous flux functions, the framework enables to do so. The EE solver can work with non-smooth RHS of the differential equations, whereas the IE solver requires smooth equations. Users could also integrate in SuperflexPy their own solvers with more specialized techniques for non-smooth problems.
>
> These points will be noted briefly in the revised paper and documentation.
>
> This aspect is now clarified in the supplementary material, section 5.1

The suggestion to use smooth methods is indeed mentioned in section 5.1

"SuperflexPy provides two built-in numerical approximators (implicit and explicit Euler) and a root finder (Pegasus method). **These methods are best suited when dealing with smooth flux functions. If a user wants to experiment with discontinuous flux functions, other ODE solution algorithms should be considered.**"

However, note that the use of non-smooth flux functions could cause convergence problems only if the root finder does not maintain brackets on the solution– e.g., in the classic Newton-Raphson root finder. Technically speaking non-smooth flux functions can also be used when the root finder is implemented using a bracketing algorithm such as bisection or Pegasus (e.g., Press et al., 1992). Indeed, this is another robustness benefit of the "one-element-at-a-time" strategy.

On the other hand, we still recommend smoothing the flux functions because jump discontinuities in these functions can cause mass balance discrepancies (essentially depending on which side of the jump discontinuity is used to calculate the fluxes). We have cited the work of Kavetski and Kuczera (2007) which provide a broader motivation for smoothing the constitutive functions.

We have now mentioned the preference for the usage of smooth flux functions also in the paper, section 4.3 and elaborated more in the documentation, section 5.1.

PK.7: *RC1.8: My concerns about numerical errors by the operator split are explained in the answer to the reviewers (RC1.8), but have not made it in the manuscript – neither in the main text nor in the supplemental material. In fact, both m/s and supplement are plainly wrong: m/s l. 514 suggest a free choice for the selected numerical solver and the supplement mat 5.1 suggests RK-solvers as an additional (not yet used) choice. However, RC1.8 explains me (but not the readers), that only single step Euler solvers are suitable to solve the system as other solvers would introduce the need for a formal integration of the fluxes over the (outer) timestep:*

"When fixed-step solvers are used, this "one-element-at-a-time" strategy is equivalent to applying the same (fixed-step) solver to the entire ODE system simultaneously (i.e., no additional approximation error is introduced). " (from answer to RC1.8)

This section needs to make it in the main text of the manuscript, together with a reference for the claim.

We agree that information on these numerical issues is pertinent, and have added it to the main text. The new content includes material from all points listed thus far.

The information provided in the reply to RC1.8 is already present in the documentation. For clarity, we report here our reply, highlighting in bold the parts that have already been copied in the documentation in section 5.2, titled "Sequential solution of the elements":

"**The SuperflexPy framework is built on a model representation that maps to a directional acyclic graph. Model elements are solved sequentially from upstream to**

> **downstream, with the output from each element being used as input to its downstream elements.**
>
> **When fixed-step solvers are used, this "one-element-at-a-time" strategy is equivalent to applying the same (fixed-step) solver to the entire ODE system simultaneously** (i.e., no additional approximation error is introduced). This is one of the pragmatic reasons we favor the fixed-step implicit Euler scheme.
>
> **When the solvers use internal substepping, then the "one-element-at-a-time" strategy does introduce additional approximation error. This additional approximation error is due to treating the fluxes as constant over the time step, whereas the exact solution would have varying fluxes within the time step. However, in most practical applications, this "uniform flux" approximation is already applied to the meteorological inputs (rainfall and PET), hence applying it to internal fluxes does not represent a large additional approximation.**
>
> The option to solve the system of equation jointly would avoid the "constant flux" approximation for the internal fluxes (but not for the meteorological one). However, the gain in accuracy is expected to be small and come at the expense of a considerable computational effort and additional code complexity.
>
> We agree these details are pertinent – they will be explained in the Documentation and a cross-reference will be added to the Paper.
>
> If individual elements have multiple outgoing fluxes (e.g., streamflow and evapotranspiration), these are calculated simultaneously by solvers such as IE, and there is no need to specify an order for how such outgoing fluxes are calculated (it is however necessary if EE is used)."

Next, we elaborate on the following statement:

> "When fixed-step solvers are used, this "one-element-at-a-time" strategy is equivalent to applying the same (fixed-step) solver to the entire ODE system simultaneously (i.e., no additional approximation error is introduced)"

In this case, the solution of upstream elements does not require the solution of the downstream elements. Technically speaking, the Jacobian matrix associated with the system of equations is lower triangular. Hence, the solution can proceed from upstream to downstream elements with no further approximation or iteration needed.

As a quick example, consider the system of ODEs for model M4, which has 2 reservoir elements, UR and FR (section 3.1 of the paper). When discretized using the implicit Euler (IE) scheme, the following system of nonlinear algebraic equations is obtained:

$$\begin{cases} \dfrac{S_{t+1}^{(\mathrm{UR})} - S_t^{(\mathrm{UR})}}{\Delta t} = P - E_{\mathrm{P}} \dfrac{\overline{S_{t+1}^{(\mathrm{UR})}}\left(1 + m^{(\mathrm{UR})}\right)}{\overline{S_{t+1}^{(\mathrm{UR})}} + m^{(\mathrm{UR})}} - P\left(\overline{S_{t+1}^{(\mathrm{UR})}}\right)^{\beta^{(\mathrm{UR})}} & \text{..... equation 1 (element 1, UR)} \\ \dfrac{S_{t+1}^{(\mathrm{FR})} - S_t^{(\mathrm{FR})}}{\Delta t} = P\left(\overline{S_{t+1}^{(\mathrm{UR})}}\right)^{\beta^{(\mathrm{UR})}} - k^{(\mathrm{FR})}\left(S_{t+1}^{(\mathrm{FR})}\right)^{\alpha^{(\mathrm{FR})}} & \text{..... equation 2 (element 2, FR)} \end{cases}$$

where the unknowns are $S_{t+1}^{(\mathrm{UR})}$ and $S_{t+1}^{(\mathrm{FR})}$, i.e., the storages in element 1 (UR) and element 2 (FR) respectively (note that $\overline{S_{t+1}^{(\mathrm{UR})}}$ is a function of $S_{t+1}^{(\mathrm{UR})}$).

Equation 1 contains unknown 1, and equation 2 contains both unknowns 1 and 2. Hence the system of equations (more precisely, its Jacobian matrix) is lower triangular, and can be solved using forward elimination: solve equation 1 for unknown 1, and then solve equation 2 for unknown 2 (keeping unknown 1 fixed).

These arguments generalize quite trivially when more than two reservoirs are present.

$$\begin{cases} \mathrm{f}_1(S_1) = 0 \\ \mathrm{f}_2(S_1, S_2) = 0 \\ \mathrm{f}_3(S_1, S_2, S_3) = 0 \\ \quad\vdots \\ \mathrm{f}_N(S_1, S_2, S_3, \ldots S_N) = 0 \end{cases}$$

It can be seen that no additional approximations are introduced when solving equations one at a time starting from unknown 1 and finishing with unknown *N*.

Note also that this analysis is distinct from the assumption that the fluxes are constant over the time step (see point **PK.2**).

Section 5.2 of the documentation is now titled "Sequential solution of the elements and numerical approximations". This aspect is also mentioned in section 4.3 of the paper.

PK.8: *RK-solvers of nth order use n-1 (or more) substeps to predict the final state at the output timestep by fitting an nth-order polynom into these substeps. Using the flux at Y(t, S(t)) or Y(t+1, S(t+1)) is not the correct number, as the solver calculates the ODE between these timesteps and introduces an uncaught numerical error into the system.*

We now provide a new numerical approximator that implements the Runge Kutta 4 (RK4) algorithm. Note that, however, due to the "constant-within-timestep" approximation (refer to **PK.2**), input fluxes to the element are treated as constant; output fluxes, on the other hand, can be calculated with intermediate states, when solving the differential equation of the element.

PK.9: *While the user is free to use any Jacobian-free root finder, the choice of the ODE-solver is (obviously) limited to implicit and explicit Euler methods (PECE methods might also an alternative). There is no interface to calculate the Jacobian matrix of an Element, hence Newton-like root finding algorithms are not suitable.*

The lack of facility to communicate the Jacobian of an element was indeed a limitation of the previous version of SuperflexPy. As part of the revision, and motivated by the reviewer comment, we have generalized the implementation of the flux function methods to accommodate the analytical calculation of the derivatives of the fluxes with respect to the state. These values are then propagated by the numerical approximators and are provided to the root finder. In turn, this enables the root finder to use algorithms that employ analytical derivatives, such as the classic Newton-Raphson. We have provided a new root finder `NewtonPython` that uses derivatives unless the resulting root jumps out of the brackets, in which case a bisection step is employed (see Press et al., 1992 for the principles of this algorithm).

For generality, this new functionality is implemented as "optional": if the user implements a new flux function but does not wish to derive and implement analytical derivatives, they can specify `None` as the value and then use a derivative-free root finder such as Pegasus.

Note also that the derivatives can be calculated numerically by the root finder itself as part of its internal approximations – this option is trivially available to any root finder but can be computationally expensive and according to the numerical literature is seldom beneficial when solving scalar equations.

We have updated the documentation (chapters 5, 8 and 10) to reflect this enhancement in the SuperflexPy framework.

PK.10: *The freedom of the solver choice is quite limited by the use of the sequential solution of the DAG approach – this is of course valid, but should be made explicit.*

The new section 5.1.3 of the documentation shows how to implement new solvers “from scratch” within the limitations stated in **PK.2**.

MP4:

PK.11: *The classical structure of scientific writing is of course not directly fitting with a model description paper. However, I would see the choice of math, programming language and design principles rather as the methods of a model implementation and the resulting code and use examples as the results section. The new, frequent links to other sections, are a poor surrogate for a cleaner structure but are an improvement over the original manuscript.*

As noted in the previous round of reviews, the paper has been organized to cater to two distinct audiences:

- general hydrological/environmental modelers with interest in the capabilities and usage patterns of the software;
- specialist researchers with interest in technical implementation details.

Meeting the expectations of these two audiences requires some compromises. Our choice has been to progress from simple aspects accessible to the broader audience to more specialized aspects requiring a stronger technical background in numerical computation and software design.

We appreciate that a specialist reader may prefer a different presentation structure, but putting highly technical details first could easily confuse readers without a specialist background. With that in mind, we do appreciate the reviewer feedback that the revision has been an improvement over the original manuscript.

Additional issues:

Section 4.2:

PK.12: *The m/s mentions 8 times the object oriented design of the implementation and but does not feature the object oriented design choices at a prominent place. The UML-diagram is now hidden in the last section of the supplemental material. The UML-like diagram should be moved to the main text in section 4.2, as it is essential for the understanding of the object oriented design. Now section 4.2 lists, how the OO-design helps to accomplish certain goals, but we, as the readers, can only guess what that OO-design is.*

We fully agree with this comment. The UML diagram has been integrated in section 4.2.

PK.13 [the comment has been re-formatted to facilitate its reading]: *Fig 12: I am familiar with most services and software mentioned in Fig 12 (except binder), however, I had a hard time to understand it.*

1. *Mixing cloud services like github, binder, zenodo, and read the docs with a file format (Jupyter-Notebooks) on an equal level does not help to understand any of these services.*

2. *Having the developer and the user as the same person (symbol) complicates the understanding with the blue and black lines.*

3. *The authors state, that explaining the ecosystem around SuperflexPy is important – while I do not follow the premise, explaining the services is possibly better done with text. But if the ecosystem is important enough for a large figure in the main text (while omitting the UML-Diagram), then the importance should be highlighted throughout the paper, as the object oriented design is. I still recommend to delete this figure.*

4. *If the authors are absolutely sure, this figure is needed, they need to*

a. *redraw the figure using two persons and kicking out Jupyter-Notebooks (as they are not a web service themself) and test the figure with friends from the intended audience*

b. *explain the figure in much more detail and the role of every mentioned service therein, and*

c. *introduce throughout the paper the importance of a webservice ecosystem for modern model development (eg. mention in section 1.2, practical criteria).*

5. *However, as of now, this figure is hardly explained, and someone who is not familiar with these services will not profit from it. Even worse, the figure in its current state is prone to misunderstandings and does a disservice to the paper. Moving the figure as is to the supplement material does not solve the issues mentioned above.*

We believe that a brief description of the "ecosystem" of web services is important for general users in the hydrological community, who in our experience are often not up-to-date with many of these web services/tools. Figure 12 is intended to help readers navigate the way SuperflexPy is integrated into this broader ecosystem. That said, we agree with several points made by the reviewer regarding some technical inaccuracies/confusions in the way Figure 12 was presented, and have made the following changes to the figure:

- Removed Jupyter as it is indeed a file format not a web service
- Distinguished the "user" from the "developer"
- Distinguished automated steps (dashed lines) from “manual” steps (continuous lines)

Moreover, as suggested by the reviewer, we have enhanced the main text to motivate the importance of a deployment pipeline and of using web services in the introduction of the paper (lines 179-181).

A succinct explanation of the tools and their roles depicted in Figure 12 can be found in the main text Section 5.1.3

> Figure 12 shows the online software management tools that are used to develop and deploy SuperflexPy. The framework itself, including source code, documentation, examples, etc., is hosted on **GitHub**. Automated workflows are then used to create new releases (**PyPI**), get DOIs for the software releases (**Zenodo**), host the documentation (**ReadTheDocs**), and run the examples (Jupyter and **Binder**).

A more detailed explanation is provided in the documentation Chapter 2, which shows already the same picture and explains, with greater detail, the role of the services (Binder was missing and has been now added)

> The source code, documentation, and examples are part of the official repository of SuperflexPy hosted on **GitHub**. A user who wishes to read the source code and/or modify any aspect of SuperflexPy (source code, documentation, and examples) can do it using GitHub.
>
> New releases of the software are available from the official Python Package Index (**PyPI**), where SuperflexPy has a dedicated page. [link to the PyPI page]
>
> The documentation builds automatically from the source folder on GitHub and is published online in **Read the Docs**. [link to the documentation]
>
> Examples are available on GitHub as Jupyter notebooks. These examples can be visualized statically or run in a sandbox environment (see Examples for further details). [Link to a page in the documentation that lists the examples and links to GitHub and Binder]

We thank the reviewer for their feedback on this important usability aspect, which is now presented in a clearer and technically more sound way.

PK.14: *Jansen et al (2020) reference: This is unpublished work, and I as a reviewer am unable to check the content of this reference and review its role in the paper. The author's claim about its content might be wrong. As such, the reference needs to be removed. If a public preprint had been cited, this problem would not exist.*

The reference in question is the following one:

> Jansen, K. F., Teuling, A. J., Craig, J. R., Dal Molin, M., Knoben, W. J. M., Parajka, J., Vis, M., and Melsen, L. A.: Mimicry of a conceptual hydrological model (HBV): What's in a name?, Water Resources Research, n/a, e2020WR029143, https://doi.org/10.1029/2020WR029143, 2021.

That paper is not unpublished work - it was accepted before the previous revision of the SuperflexPy manuscript was re-submitted, and the reference given was (and still is) for the accepted paper.

We checked that the doi is working properly, so that the Reviewer can certainly access it if they wish. We understand that the "n/a" may have caused some confusion, and have corrected it.

Response to comments by AR2

AR2.1: *The authors have done a commendable job addressing the detailed comments of the reviews. While there was quite a bit of "pushback" with respect to reviewer suggestions for including more rigorous comparisons with existing frameworks and inclusion of more implementation details, I found their arguments for resisting these recommendations for the most part convincing, and they have done an effective job addressing the spirit of these comments without (for instance) over-duplicating the contents of the software documentation or getting pulled into the details of an exhaustive model intercomparison. As such, I recommend acceptance subject to (very) minor revision.*

We thank the reviewer for their recognition of our effort in improving the paper and for their appreciation of our reasoning against some earlier proposed changes (where those would have been impractical within the scope of the current paper).

Major comment:

AR2.2: *I still have a bit of concern with the undue apparent stress on transport simulation capabilities which are *not present in the existing model*. This is highlighted as a key "realm" in the application scope (line 193 of marked up revision), at line 203, and elsewhere.*

This concern could be mitigated by revising line 193 to "Extendibility for future applications, e.g., isotope or pesticide transport modelling". Any hydrological model can technically be extended to support transport, and it is by no means clear that SuperFlexPy is more extendible than others (without explicitly demonstrating it).

We appreciate the concern of the reviewer. As part of the revisions, to avoid an inadvertent "undue stress" on this concept, we have checked every mention of "transport simulation" in the context of SuperflexPy to ensure it clearly refers to extendibility for future applications (thus addressing the reviewer concerns) rather than a currently available feature. References to transport simulation in general hydrological modelling contexts (rather than in SuperflexPy-specific contexts) were kept as is, as they indeed provide the motivation to support future extendibility.

We list below all the sentences in the paper where modelling of transport processes or chemistry is mentioned (only here, in order to maintain a correspondence between question and answer, line numbers refer to the marked up version that the reviewers refers to) to clarify our actions:

1. Line 89

   "However, their application extends to the simulation of other environmental variables such as groundwater levels (e.g., Seibert and McDonnell, 2002) and soil moisture (e.g., Matgen et al., 2012), as well as water chemistry (e.g., Bertuzzo et al., 2013; Ammann et al., 2020)."

This sentence is about the general areas of application of hydrological models, not about SuperflexPy. Therefore, the sentence was kept as is.

2. Line 189

> "In terms of application scope of a flexible framework for conceptual hydrological modeling, we focus on the following "realms": [...] Substance transport modelling, including water isotopes, pesticides, etc".

We have changed to "Support or extendibility for future applications, e.g. substance transport modelling, including water isotopes, pesticides, etc.", as proposed by the reviewer. Note that this is a general statement for flexible frameworks.

3. Line 195

> "In terms of software implementation, we consider the following practical criteria: […] 2. Ease of modification and extension. Even a comprehensive software implementation will eventually require extension. For example, a modeling framework intended to simulate streamflow may require extension to simulate water chemistry."

This sentence refers to the desired property that a flexible framework should be easy to modify and extend – and mentions the simulation of transport processes as an example of possible future extension. SuperflexPy is designed with this requirement in mind (i.e., of being easy to modify and extend). Therefore, no change has been made – indeed by definition an example of future extension should be something not implemented in the current code.

4. Line 231

> "The original Fortran implementation of SUPERFLEX, hereafter referred to as SUPERFLEX-F90, has been used in a series of case studies over the last decade, […] inclusion of pesticide/substance transport (e.g. Ammann et al., 2020)."

This sentence refers to past applications of SUPERFLEX-F90, which does in fact include a substance transport module. Note that SUPERFLEX-F90 is a different implementation that as such is unrelated to SuperflexPy. Therefore, the sentence is factually correct and was kept as is.

5. Line 722

> "The capability to simulate multiple fluxes and states is intended to support the extension of SuperflexPy to new modelling scenarios. Several such scenarios may be of interest, including the transport of chemical substances (e.g., Fenicia et al., 2010; Ammann et al., 2020) […]".

The sentence lists possible applications where "the capability to simulate multiple fluxes and states" may be useful. The general ability to simulate multiple fluxes and states does not imply that specific modelling contexts where such one of the applications is simulating transport processes does not implies that this is readily available.

We already have remarked this concept also in the following paragraph (line 730)

> "While the current examples in SuperflexPy do not include all the cases listed above, […]"

We have changed the sentence to "support the **future** extension" (i.e., adding the word "future") to put emphasis, on the fact that this feature is not yet implemented.

These changes address the remaining confusion regarding "what is" vs. "what is not" supported, and clearly state that transport simulation is currently not supported.

Minor comments: (line numbers refers to marked up manuscript)

AR2.3: *line 200- "modifications and extensions"-->"modification and extension"*

Thank you – change implemented.

AR2.4: *line 217- remove "or even impossible"*

Thank you – change implemented.

AR2.5: *line 244- "highlighted implementation choices" - such as? This is very vague. If you are going to note that SuperFlexPy will address these limitations, you have to state what they are.*

We agree this was vague. We have clarified on line 207 that this mainly refers to the use of a "master template" from which specific model structures are derived. Note that subsequent Tables 1 and 2 provide a detailed summary of differences, which are moreover discussed in the text in section 5.1.

AR2.6: *line 383- The value of the stand alone statement "All SuperFlexPy componets are..." is unclear (as is the connection to the previous paragraph). What does it mean to be "characterized by" a state or parameter?*

This sentence introduces that SuperflexPy components have states and p*arameters.* We have changed "characterized by" to "have" for clarity.

AR2.7: *line 409- "More specifically" ->"Specifically"*

Thank you – change implemented.

AR2.8: *references- some cleanup of the references is needed w.r.t. inconsistent capitalization, etc.*

Thank you for noticing this - we have now fixed all issues we could spot.

References

Clark, M. P., & Kavetski, D. (2010). Ancient numerical daemons of conceptual hydrological modeling: 1. Fidelity and efficiency of time stepping schemes. *Water Resources Research,* *46*(10). https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2009WR008894

Dowell, M., & Jarratt, P. (1972). The "Pegasus" method for computing the root of an equation. *BIT Numerical Mathematics, 12*(4), 503-508. https://doi.org/10.1007/BF01932959

Kavetski, D., & Clark, M. P. (2010). Ancient numerical daemons of conceptual hydrological modeling: 2. Impact of time stepping schemes on model analysis and prediction. *Water Resources Research,* *46*(10). https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2009WR008896

Press, W. H., Teukolsky, S. A., Flannery, B. P., & Vetterling, W. T. (1992). *Numerical recipes in Fortran 77: volume 1, volume 1 of Fortran numerical recipes: the art of scientific computing*: Cambridge university press.