Response to review #2

Summary

Thank you for this helpful review! Original comments are in blue, our responses in black.

In this manuscript, the authors introduce a new land-ice modeling software package known as icepack. icepack is written in Python on top of the Firedrake library, which uses the domain-specific Unified Form Language (UFL) and provides a high-level symbolic description of the problem to facilitate the add of new physics and/or equations. The intended user base of icepack is the glaciological community, in particular, glaciologists who may not have an extensive background/training in computational science. The idea is to provide a code that would be easy to use/develop by this class of prospective users. Following a description of the code, some numerical examples are presented to demonstrate the method's capabilities and accuracy. The manuscript in question is well-written and interesting to read. Addressing usability is noteworthy and something that not enough authors in glaciology/climate science address. I personally have some qualms with some of the philosophy described by the authors, namely I worry about folks who are not familiar with numerical methods developing an application code in which a lot of what is under the code is hidden from them in some sense. In an ideal world, one would have glaciologists working with computational scientists to help them pick the right solvers, discretizations, etc., for their problem. The authors are correct that some solver options are for optimizing performance, which is secondary to getting the code/model running; but there are also solver/algorithm choices that depend very much on the physics (e.g., CG is only valid for symmetric problems) - is icepack designed so as to prevent the naive user from inadvertently using the incorrect default setting for their problem? I will assume it is to the extent it can be, and that the authors' argument is that the default hidden settings are likely to do less damage than some arbitrary settings a user might put in his/her input file without knowing what they are doing. Also, I realize that the reality is that many glaciologists do not have strong ties to computational scientists, and still wish to make progress in numerical modeling of land-ice; therefore, I will not focus too much on much "philosophical" perspective described above. Another qualm I have about the paper has to do with performance - I am skeptical whether icepack can really be performant if one tries to run it on continental scale problems, and it is not clear to me if the code is even parallel. I think the intention may be to use icepack as more of a sandbox for prototyping small problems (similar to FeniCS), in which case, this is not a huge deal. Despite the above concerns, I like this paper and see it published in GMD. I like in particular the idea of describing all the equations using the variational principle/action functional and having everything else propagate from there - not enough people do this. I do, however, feel that there is a lot of missing information in various parts of the paper, which should be filled in in the revision prior to the paper being suitable for publication. Please see below my enumerated list of questions/comments to address in the revision.

Specific comments

1. The authors suggest that C++ makes it inherently difficult to add new physics/PDEs (e.g., on p. 10 and p. 20), which I somewhat disagree with. One advancement that can make a C++-based code easy to add to is Automatic Differentiation (AD) - with AD, one can effectively code the weak form of the residual within a C++ code and AD will handle the rest, making it very easy to add new physics. An example is the Albany/Land Ice model (previously known as Albany/FELIX) of Tezaur et al. (https://doi.org/10.5194/gmd-8-1197-2015). I think it would be worth mentioning that there have been efforts like Albany/Land Ice out there to make C++-based codes more accessible to users of varied backgrounds. I agree that even an "easy-to-use" C++ code will be more difficult and more intimidating than a Python code, so I am not trying to minimize the authors' efforts at all.

   I hope this paragraph made the point more that we had specific goals that could better be achieved in Python and not as some blanket condemnation of C++ – I don't like being dogmatic about language choice. I've added a citation to Tezaur et al. and to another paper from the ISSM group about AD. I agree with you about AD tools: they definitely relieve the burden of having to rewrite the code to calculate variational derivatives of functionals of the solutions of the model with respect to input parameters upon changing the model or parameters. The experiences I had rolling my own adjoints are what pushed me to tools that would either have AD or a more symbolic approach like what FEniCS, Firedrake, and Devito offer. In idiomatic Python it's nonetheless possible to be much more flexible about function signatures than in C++ by virtue of being able to throw arbitrary data into `kwargs`. Now of course you pay for this in that all input validation is done at runtime rather than at compile time, but it's a tradeoff we had to make.

2. Can the authors comment on the overhead of the symbolic descriptions/manipulations done by their framework? This sounds potentially like it would be very expensive. How does the cost compare to automatic differentiation, for example? A broader question is: is computational performance/cost a concern for users of icepack, or is it intended to be a "sandbox" in which performance is secondary to being able to code up something "quick-and-dirty" for initial prototyping?

   See Rathgeber et al. 2016 for more information about the architecture of Firedrake and for performance benchmarks. We are constrained only by what

Firedrake can do, so we refer to this paper for performance with no change to the text.

Loosely speaking, the path through the toolchain goes like this. A user creates a symbolic description of the weak form of the PDE they want to solve. Firedrake then computes a hash of this expression and looks to see if it has encountered this problem before. If not, it does a long, complicated, and expensive series of transformations to generate highly optimized C code that fills the relevant matrices and vectors. This C code is then compiled into a dynamic library for later reuse. If the user has solved this problem before, Firedrake simply looks up the dynamic library that it already generated. (Crucially, the only thing that matters is the symbolic shape of the problem, not the actual data that goes into it – you don't have to do codegen all over again just because you changed the boundary conditions or forcing.) In either case, Firedrake then calls into PETSc's scalable nonlinear equation solvers (SNES) to solve the resulting system of equations. As with any just-in-time compiled language, performance is slow the first time the code is run, and faster ever after. Most importantly, **the performance-critical parts are all written in C**. Firedrake has been shown to scale up to large problems on thousands of processors.

It is possible to ruin the performance of the application by accidentally hard-coding a floating point value into a symbolic expression of a PDE and then changing that value in a loop. For example you could easily make this mistake if you were doing adaptive timestepping on an evolutionary problem discretized via the method of lines. The remedy is to wrap this value in a `firedrake.Constant` object. This kind of performance regression is easily caught using `htop`. While this is more a result of programmer error, it's an easy mistake to make and the Firedrake team are working on ways to diagnose it and issue appropriate warnings.

For the problems that we have used icepack for so far, we have focused more on individual glaciers or catchments, and thus performance has been a secondary concern. We aim to move towards larger continental-scale problems in the future. The rate-limiting factor there is more our ability to find the right incantation of PETSc solver options and preconditioners than it is any inherent limitation in our tools.

3. I was a little bit confused about the reference of the FO Stokes-based model in this paper as a "hybrid model". I see that it is hybrid in the sense that you have a different discretization in the horizontal and vertical direction, but there are also hybrid ice models that use different PDEs in different domains, e.g., the ISCAL model of Ahlkrona et al. (https://doi.org/10.1016/j.quascirev.2016.01.032). Is the term "hybrid model" a common name for the approach in Section 2.2.3? Perhaps it is and I am not aware of it. Does the hybrid model described in the paper have the same applicability as say the First Order Stokes model? Can it be used for both Greenland and Antarctica at continental scales?

   The fundamental physics are the first-order model obtained by asymptotic expansion of the Stokes equations in the aspect ratio, also known as the Blatter-Pattyn equations. We have amended the text to make this clear. What I had imagined is that using only vertical basis functions up to degree 2 essentially defines its own semi-discrete physics model, similar to two- or three-layer ocean models. You can view these as very coarse discretizations of the primitive equations, or you can view them as simplified models in their own right. But I made this naming choice before there were many other collaborators on the project. The ensuing confusion has shown that this was a bad choice of terminology and we intend to change it in a future version. This model can be used for both regions at continental scales – it can capture both plug and shear flow.

4. Section 2.2.1: in my opinion, the authors do not provide sufficient justification for the penalty term, equation (7). They describe this as something that is added to smooth over artifacts - this would be needed based on the discretization, which there is little discussion of. The authors should state what order finite elements they are using - I presume they are linear, and that this is why the stabilization is needed? Why is stabilization needed only for the SIA? I think these things should be made clear.

   See comments by reviewer #3. The technical answer is that this makes the solution live in the Sobolev space $H^1(\Omega)$. A more heuristic answer is that the shallow ice approximation is usually assumed to hold only over distances greater than a few ice thicknesses. The penalty term is meant to filter out variability at length scales where the model doesn't even apply. No change to the text.

5. Section 2.2.2: there is some imprecision here in equation (13) - you have not defined anywhere that Γ is the boundary, and which boundary you are referring to. One can figure it out, but it is not precise. Ω is not defined either though one will assume invariably that this is an open bounded domain in 2D or 3D depending on which approximation one is looking at.

   These were not stated explicitly anywhere. We've added them to table 1.

6. The boundary conditions are not discussed very rigorously systematically - the authors seem to sprinkle in some boundary conditions here and there. I think the boundary conditions need to be given for each of the models at the time the models are presented - boundary conditions are needed to complete the definition of each models.

   We added this statement to section 2.1: "We implement two types of boundary conditions for the prognostic equation. Users can specify an inflow flux value and this value becomes a source of thickness at any point along the domain boundary where the ice velocity is pointing in to the domain. The flux at the inflow boundary can change in time. Second, we impose outflow boundary conditions on any part of the domain where the ice velocity is pointing outwards. Which segments of the boundary are inflow or outflow are diagnosed automatically by calculating the sign of the dot product between the velocity and the unit outward normal vector."

We also added an entirely new section which is now §2.3 in the text just on the boundary conditions for the different diagnostic models.

7. Certain terms in the equations I do not believe are defined anywhere, for instance, in equation (10), there is no expression given for the strains $\dot{\epsilon}(u)$. This is one of the things I cam across that need to be made more precise.

   This was stated in table 1 but we've added the definition as $\frac{1}{2}(\nabla u + \nabla u^{\top})$ and added a sentence to the text referring readers to table 1.

8. Section 2.2.3: the authors comment that higher degree polynomials can be used in the vertical layer in the hybrid approach. What order is typically used?

   Added the following text: "Going up to a degree-4 model is sufficient to capture the exact solution for the shallow ice approximation. In the tutorial notebooks for icepack, we use up to degrees 2 and 4, but the test suite checks up to degree 8."

9. Section 2.2.3: this might be a naive question, but does Glen's flow law come into the hybrid model? I was expecting to see it there, but maybe I'm missing something.

   This was a bad oversight on our part – the hybrid model does use Glen's flow law and we've added more detail at the end of this section describing the terms in the action functional (equations 22 through 26 in the revised text). The main difference is the term for viscous power dissipation.

10. P. 10: the discussion here about substituting model components suggests it may be possible to use different models in different regions and couple them (a la the ISCAL method). Is this possible, or something that the authors are thinking to add to their model/code?

    Implementing this idea will require some new developments to Firedrake, namely first-class support for subdomains, defining different PDEs on different subdomains, and defining matching conditions for the solutions at the interfaces between subdomains. The Firedrake developers are working on this feature right now as it's very much in demand. We are very much interested in, for example, using SIA in the interior of the ice sheet and SSA in the ice streams and margins, as this would give a much less computationally-intensive way to do some form of whole-ice sheet modeling than using, say, the first order model. No change to the text.

11. Section 2.4: This section is very incomplete. You need to give the enthalpy equation and given the Glen's law expression as well, since it is mentioned.

    We have added some text and equations describing the model we used, the boundary conditions, and the shear heating rate, the latter of which implicitly includes Glen's flow law.

12. In my opinion, there is not enough discussion of the thickness equation (Section 2.1) and how it is discretized. In typical ice sheet models, this equation is used to change the ice extent - one meshes up a region of "potential" ice, and then uses the thickness to dynamically determine a mask for ice-covered regions. Do you do something like this in your model? It should be discussed for completeness. I think you maybe start to do this in Section 4.1, but it is very confusing and hard to make the connection.

    We added this statement to section 2.1: "Icepack represents the thickness using continuous, piecewise polynomial basis functions in each cell of the mesh. In the examples we use up to degree 2 and the unit tests use up to degree 4. We have not yet implemented a formulation that works with discontinuous basis functions, but this extension is completely feasible within our framework." See also previous comment on boundary conditions for the thickness equation. We also added a longer description of our treatment of ice-free regions (which is very ad hoc for now) at the end of section 2.1. This is a weak point at present and we plan to improve this in future versions.

13. What sort of meshes do you use in your model (in the horizontal dimension, for the hybrid one)? Structured/unstructured? Hex/tet (quad/tri)?

    We use unstructured triangular meshes although Firedrake in principle can use unstructured quad meshes. We've added the word "unstructured" to clarify this.

14. Section 3: It is not clear from the description what the inverse problem you are describing is for. Is it to obtain parameters in the model like the basal friction using observational data of e.g. surface velocity? There really needs to be more discussion here, and I personally would like to see a mathematical statement of a representative inverse problem you are solving. It would be worth citing the work Perego et al. on optimization-based inversion, if what you are doing is similar: https://doi.org/10.1002/2014JF003181. BTW, the basal friction has not been defined, yet it is discussed - it needs to be defined earlier, when talking about boundary conditions (which needs to be added).

    We added a brief description of the mathematics of the inverse problem to be solved. We also added: "The state to be estimated can be any single input field to the diagnostic model – basal friction, rheology, or another field that the user has added by customizing the model."

15. Section 4.1: I find this section confusing. I assume you are talking about discretizing the thickness equation here - that should be made clear. I disagree with several statements in this section as well. "The simplest explicit timestepping schemes are unstable with CG finite elements" - if you are talking about CFL stability, this is not true. You need to satisfy a CFL condition which could give rise to very small time-steps but you can get the scheme to be stable. I'm also confused about the notion of SUPG as a time-stepping scheme - I think of SUPG as a finite element approach to deal with advection-dominated flow problems, for example, that does not have anything to do with time- stepping.

Maybe you are referring to upwinding? In any case, I think SUPG has nothing to do with forward Euler, so the discussion about forward Euler requiring parameters is erroneous. Additionally, I don't understand the comment about implicit Euler smoothing out sharp discontinuities... I believe explicit and implicit Euler have effectively the same diffusion and dispersion properties, so there should not really be a difference between the schemes. Did the authors verify their time-stepper on a manufactured problem to ensure that it was implemented correctly?

This was sloppily written. What we should have said was that SUPG confers some of the benefits of upwind finite difference stencils when using continuous Galerkin basis functions. I was basing the statement about stability on the expression for the numerical amplification factor for the θ-scheme with piecewise linear finite elements from section 3.5.2 in Donea and Huerta, Finite Element Methods for Flow Problems. In any case, we've cut much of this section to focus more on the implementation in icepack rather than what other packages use. The text was also out of date; the actual default now is an implicit scheme with a Lax-Wendroff correction that gives higher order accuracy in time. Our statement that implicit Euler has predominantly diffusive errors was not to imply that explicit Euler doesn't share the same property. We were trying to draw a contrast between what types of errors are tolerable for the prognostic model as opposed to other problems like damage transport, which is described in the next paragraph in the text.

16. Section 4.2: there is an approach discussed in Tezaur et al. (https://doi.org/10.5194/gmd-8-1197-2015) for dealing with bad initial conditions in a Newton solver that relies on homotopy continuation that would be worth citing. It is an alternate to the approach you describe that lets you get away with not doing a line search for Newton. By the way, it should be no surprise that Newton is not converging without a line search - in general Newton is not guaranteed to converge from an arbitrary initial guess without the line search.

    Added a reference to the Albany paper as well as another one on trust region methods. We described a fairly rudimentary line search method in more detail than perhaps is necessary for a reader who's a seasoned modeler. It's a bit of a pet peeve of mine when papers just say "We used Newton!" without any attention to the globalization strategy, which can make a huge difference to the solver robustness.

17. Section 4.3, lines 411-412: there are actually ways to construct weighted norms to deal with the issue of DOFs having different orders of magnitude for the purpose of convergence.

    I think you can use a lumped mass matrix as the H_0 in BFGS too, but I find it to be far preferable to use something like Gauss-Newton which gives mesh-independent convergence and which achieves close to the second-order rate of full Newton on many problems. No change to the text.

18. Section 4.4: Again, it is not clear to me what is your inverse problem. You need to state this explicitly so it is clear.

    See correction to section 3. The point of the inverse solver class is that it is very general with respect to what field is being inferred and what diagnostic model is being used. Since the solver was designed to work for many different inverse problems

19. Section 4.5.1: you talk about problems defined on “extruded geometries” - do you ever use non- extruded geometries? It has been shown in various references that there can be numerical problems for land-ice solvers that do not use extruded geometries, e.g. Tezaur et al.

    We have restricted our implementation to use only extruded geometries. Having made this choice, we might not ever be able to solve really geometrically complex problems like what the Elmer/Ice crowd did with the drainage of the lake underneath Tête Rousse glacier. But the simplifications that this results in for the vast majority of glaciological applications are so advantageous that it’s a sacrifice we’re willing to accept. No change to the text.

20. Section 4.5.2: I think this section needs to be made earlier, and other BCs need to be added to that discussion.

    See response to previous comment and the additional section we added on boundary conditions. This section describes some extra care that we had to do in our implementation which was purely a consequence of our choice of basis functions and which does not appear in the idealized mathematical form of the model. In keeping with our overall goal of splitting the paper up into a section on what we’re solving and a different section on how we’re solving it, we’ve kept this section where it is.

21. Section 4.6: The authors mention running their code on 1 core. Is the code parallel - can it be run on multiple cores? Are there any hope for performance portability of the code to take advantage of emerging HPC architectures, e.g., GPUs?

    Firedrake relies on the package loo.py for code generation, which can target C and OpenCL. There is ongoing work with the developers of loo.py to target GPUs and other accelerators by generating OpenCL instead. Firedrake is built on PETSc and thus can run on parallel machines. See again Rathgeber et al. 2016 for performance and scaling benchmarks; Firedrake has been run on problems with millions of degrees of freedom on supercomputers, for example the UK national supercomputer ARCHER. No change to the text.

22. Section 4.6: Can you please clarify what you mean by the following statement? “Large problems, such as continental-scale modeling, will require more sophisticated and possibly problem-specific approaches”. I’m wondering in particular about the problem-specific approach part. There are models like first order Stokes that can be used at the continental scale and they are not really problem specific.

We have clarified the text to state that "problem-specific" refers more to the strategies we use to solve the resulting nonlinear systems of equations rather than to what equations are being solved, e.g. Stokes vs first-order Stokes: "For the demonstrations presented below, nearly all simulations run in a matter of minutes to hours on a single core. We have used sparse LU factorization to solve linear systems for many problem instances in order to eliminate the linear solver as a possible failure mode. Defaulting to a robust solution method is especially important for onboarding novice users who may not be familiar with different iterative linear solvers and preconditioners. Larger problems, such as continental-scale modeling, will require solving the diagnostic equations using the conjugate gradient method with an appropriate preconditioner to achieve parallel scalability. The particular structure of the problems we solve may be useful in choosing a preconditioner. For example, a rudimentary preconditioner for the hybrid model system could use the degree-0 model as the coarse space in a multigrid-type approach. These optimizations will be the subject of future work."

Since we use a modal basis in the vertical to discretize solutions of the 3D model, we can devise a p-type multigrid scheme along this axis. This is an example of using problem-specific knowledge to choose a solution strategy. Just using LU or throwing a black-box algebraic multigrid preconditioner at it would be failing to use this special structure. That said, we did not want to speculate too much on approaches that we haven't implemented yet.

23. p. 20: I don't understand why you need to create an analytical expression of (32) using special functions. Is this something specific to your framework, which requires expressions in a certain form for the symbolic representation?

    We don't need an analytical expression of equation 32, but rather of the antiderivative of that function with respect to u. This problem is specific to icepack because we have made the choice to use action principles to describe all of the diagnostic models. A package that also made the choice to use action principles but which was built on a different finite element modeling library or coded in an entirely different language would also need the antiderivative of this function. We are hampered by the fact UFL does not include support for hypergeometric functions. If we were instead writing everything from scratch in C++, we could call into Boost or GSL to evaluate hypergeometric functions. We believe that the benefits outweight the costs but this is a definite drawback of our approach. No change to the text.

24. It's great that you have executable documentation in something easy-to-use such as Jupyter notebooks! (no need to address this comment)

25. p. 29: are you considering putting in the first-order Stokes/Blatter Pattyn model into your code framework?

    The thing that we mistakenly called the "hybrid" model is really the first order / Blatter-Pattyn model. We have rewritten some of the text to try and make this clearer.

26. The methods do not discuss their code development/testing stance on icepack. How do users contribute to the code - through pull requests? Is there regression/performance testing? Continuous integration testing? These are all really important, especially if you have non-experts contributing to the code!

    This information is on the icepack website (https://icepack.github.io/developers). Users contribute through pull requests which are automatically checked against a regression testing suite. We try to keep the test coverage at 95% or higher. There is at present no automated performance testing short of looking at the timings from our CI service. Our development practices have changed appreciably even during the process of writing this manuscript; several of the contributors are students who are learning more about version control in tandem with learning to implement new or modify existing models. We have added references in the code and data availability section about where to find this information.

Minor comments

- "UFL" is not defined in the abstract.

  Changed the sentence to: "Icepack is built on the finite element modeling library Firedrake, which uses the Unified Form Language (UFL), a domain-specific language embedded into Python for describing weak forms of partial differential equations."

- p. 4, line 90: A is also called the "flow factor". I would mention here that it is usually a function of the temperature, which comes from a different equation.
- p. 5, line 125: should be "checking", not "check". ✓
- p. 5, line 126: I think it should be "Bueler" not "Beuler", if I'm thinking of the right person. ✓
- p. 5, line 147: change "we verified the correctness of the ice shelf model" to "we verified the correctness of our implementation of the ice shelf model". ✓
- p. 10, line 270: change "we'll" to "we will". ✓