Interactive comment on “SICOPOLIS-AD v1: an open-source adjoint modeling framework for ice sheet simulation enabled by the algorithmic differentiation tool OpenAD” by Liz C. Logan et al.
Reviewer 1: Laurent Hascoet (Referee)

We would like to thank the reviewer for his careful read of the manuscript and constructive comments. In the following, we respond comment-by-comment (our replies are in bold-face).

General comments:
This article describes a new development in the SICOPOLIS glaciology simulation code, to introduce sensitivity/adjoint/gradient computations. The article describes why adjoint capability is a significant improvement to a simulation code, allowing for sensitivity studies and solution to inverse problems and parameter estimation. The article discusses these new possibilities specifically for glaciology. This new adjoint capability was introduced through the use of an Algorithmic Differentiation tool: OpenAD. The article describes the amount of work that this AD tool required, and the amount of work that it saved. The article also points to a few difficulties where AD tools still require the help of the end-user. Global performance of the adjoint-enabled code is described shortly. The article gives an in-depth discussion and interpretation of the obtained gradients for the glaciology and climate specialist, and points at further exploitation of this adjoint capability as further work. The article also provides some discussion about some observed deviation in the computed gradients.

Please note that I am not able to comment on the glaciology-specific parts of the text, although their general music seems completely reasonable.
The article is well structured and well written. It is easy to read, although some parts are obviously directed at true specialists of glaciology.

Like I write in the specific comments, I am slighty worried by the deviation observed for some gradient values, between adjoint and Divided Differences. I am only partly convinced by the explanation about numerical noise. The text also evokes the cases of non-smoothness of the implemented function. Could this be part of the explanation? I think this part of the discussion might be developed a bit, as some readers may really take it as an argument against AD.

I recommend publication of this article. It describes a solid work on an important code, it is useful as a clear example of what AD adjoints can do, it promotes AD towards the glaciology community, and it seeds for further work.

Specific comments:
P3, L12 : True, the adjoint propagation runs backward in time. More generally it runs backward the original simulation order (which happens here to be forward in time). Maybe it would be useful to stress that?

This is an excellent point and we have clarified this in the revision.

P7, L1 : Rather than "can be conceived as ...", I would advocate writing that "as soon as a numerical model is implemented as a code, it is in fact translated as ..."
We agree and rewrote this.

P9, L15 : Are the preprocessor options used (to exclude or include arts) at "compilation time" or at "differentiation time" ? I take it that you mean "differentiation time". Does this imply then that there will be one particular adjoint model of SICOPOLIS for each model configuration. If so, your text presents this as an advantage but you understand some people might consider this as a drawback, not having a unique adjoint SICOPOLIS source at hand. (Here I'm playing the devil's advocate, as I think any source-transformation AD tool will face the same drawback)

We appreciate the reviewer's concern, and the notion that some readers will regard this as a drawback. In the revised manuscript we have substantially extended the discussion to argue that computational requirements (especially with regard to memory footprint) are very different for forward versus adjoint models, and that for this reason (mainly, but not exclusively) the preferred option is to disable code at differentiation time. The argument is repeated here, and goes as follows:

> *Like many complex, time-evolving geophysical models, SICOPOLIS comes with a range of choices of model configuration, in particular numerical schemes, which the user may choose from. As a matter of convenience, the preferred implementation is to make all of these choices (or options) available at runtime, such as to minimize the need for recompiling the model. The same convenience is25available, in principle, to the AD-generated adjoint model. The control flow analysis of the AD tool identifies all possible flows of forward model execution and produces corresponding adjoint flow paths. However, close to two decades of experience with the application of AD to complex, time-evolving geophysical models, all of which have a range of numerical schemes that users may choose from (Heimbach et al., 2002, 2005; Forget et al., 2015), has shown that for the specific application of adjoint modeling, it is preferable to remove code that will not be executed in a given application from adjoint code generation (and subsequent compilation). The two main reasons for proceeding in this manner are:*
>
> *(i) Exclusion of forward model code that the user knows will not be executed may significantly simplify the AD tool's dependency and flow control analysis, avoid spurious dependencies that the AD tool may detect, and lead to more streamlined source code for the adjoint;*
>
> *(ii) Because of the reverse mode and requirement to store required variables in time-reversed order (e.g., those used for evaluating state-dependent conditions and nonlinear expressions), adjoint models will have a substantially larger memory footprint than their parent forward model (Heimbach et al., 2005). Memory requirements may be significantly increased if the adjoint model is required to keep track of a large range of conditional branches for execution.*
>
> *For these practical considerations, removing non-used forward model code at the time of adjoint code generation and subsequent compilation has proven to be highly preferable (although not strictly required). It is implemented here via C preprocessor (CPP) options*

that are enabled or disabled prior to generating the adjoint code (and prior to compilation time. We note that the implementation keeps runtime parameters and flags in place, such that the forward model default to keep all code available at runtime is not compromised. By pairing SICOPOLIS with source-transformation tool OpenAD, the adjoint model of SICOPO-10LIS may be generated automatically, for a large variety of forward model configurations (including detailed choices of model domain, numerics, as well as control variables and QoI).

P10, L3 : Does this raise the question about why, in the adjoint, some time steps are more stable than others? In other words, why can't one take the same time step sizes than in the forward simulation. That can be an interesting question for a Numerical Analysis specialist (not me...)

This is a well-spotted error on our part: we take that time step because it ensures stability in the forward model, *not* the adjoint model. We re-wrote for clarity.

P10, L31: I'd replace "sufficently approximates" with "is sufficiently consistent with", because I tend to think that it is divided differences that is an approximation of the other.

Done.

P11, Table 1: I would swap rows 7 and 8 for consistency with rows 3 and 4.

Done.

Deviations on rows 5 to 8 seem surprisingly high. Are they discussed in the text ? I read in P14 L20 that the finite difference chosen is around 5%, which can explain the high deviation. And yes, the explanation in P15 L1 may be right. But does the deviation decrease when the divided difference is smaller e.g. 0.5% instead of 5% ? As it is, a deviation of 57% is still worrying.

We discuss in somewhat more detail the mismatch of these finite volume and adjoint calculations on P15 L14. In general, the temperature control variables display a greater mismatch between adjoint and finite difference than the other control variables. The two main issues are (1) numerical noise when sensitivities are very small compared to the QoI, which affects the finite difference; and (2) n SICOPOLIS, the thermal equations employ a greater number of non-differentiable terms. We note this in the revised version, and point also to the appendix for a discussion on non-differentiable code. As for the choice of 5% deviation: this value was not only selected in accordance with a previous adjoint model of SICOPOLIS (Heimbach and Bugnon, 2009), but also because deviations < 5% did not result in appreciable changes to the cost function at all.

P15 L3: The question that comes immediately is how do these times compare with the primal simulation ? It might also be appropriate to describe the checkpointing scheme used in this

experiment, on time-stepping: is it multi-level, or binomial, how many checkpoints are used, how many duplicated forward steps. Are these questions left for future work? Oh, I see it is in the appendix, P24 L24. Could you just, in the main text, point out that the appendix mentions that ?

Indeed, it's described in Appendix B. Reference to it is now added.

P17 L17: The question of best practices makes me think that you may want to cite the Utke-Hascoet paper on that "Programming language features, usage patterns,..."

Done (thanks!).

(OMS 2016)
Technical corrections:
Done: P2, L25 : "construed"
Done: P4, L13 : "more confident projections" -> "more faithful" ?
Done: P5, L30 : "substantive ... to" -> "substantial ... over" ?
Done: P6, L1: "ever" or "even" ?
Done (deleted): P8, L15: why comma after warmer?
Done: P9, L25: "appendix" is repeated
Done: P10, L13: "instantaneously" ? "instantly"
Done (well-spotted): P13, L3: Is it January? Figure 3 caption writes July. Or did I miss something?
Done: P16, L18: "introduces" or "-introduced" ?