Reviewer 1:

I quickly reviewed this paper and found that critical issues exist as follows:

Answer: We thank for the reviewer for their feedback.

1. Incorrect terminology even in abstract (e.g., model analysis)

   Answer: Thank you for bringing up this issue. However, the term "model analysis" can be found in at least a few publications, e.g.:

   - Liu, X., Yao, J., Wu, T., Zhang, S., Xu, F., Zhang, L., et al. (2021). Development of coupled data assimilation with the BCC climate system model: Highlighting the role of sea-ice assimilation for global analysis. Journal of Advances in Modeling Earth Systems, 13, e2020MS002368. https://doi.org/10.1029/2020MS002368
   - Rodell, M., and Coauthors, 2004: The Global Land Data Assimilation System. Bull. Amer. Meteor. Soc., 85, 381–394, https://doi.org/10.1175/BAMS-85-3-381.

   This terminology can also be found in the website of operational centres. For example, the term is used in the caption of Fig. 2.5.1 of in `https://confluence.ecmwf.int/display/FUG/Section+2.5+Model+Data+Assimilation%2C+4D-Var` (last accessed 24 July 2024) of the European Centre for Medium-range Weather Forecasts (ECMWF). To this end, we are confident that the term is correct. We also checked the manuscript and didn't find other potentially incorrect terminology.

2. No explanation of math symbols even in Eq. (1) (e.g., p_i)

   Answer: We kindly remind the reviewer that we defined p_i after Eq. (2). Based on the suggestion from the second reviewer, we have decided to remove this section. We have checked other mathematical symbols in the manuscript and are confident that everything is defined.

3. Larger forecast/analysis RMSEs than the prescribed observation errors in Fig. 4 (i.e., filter divergence occurs)

   Answer: We apologise for the confusion. As described in the caption, Fig. 4 is the free run ensemble without any assimilation being applied. We modified the figure to include both the freerun and SCDA experiment. This is consistent with the conclusion by Tondeur et al. (2020).

Therefore, there is a high possibility that this paper does not satisfy the standards of international journals.

Answer: We regret to hear the reviewer's opinion. However, we believe that by providing a new Python interface to PDAF and presenting a robust set of experiments to demonstrate its performance, this manuscript is a useful contribution to the international community.

Reviewer 2:

Summary:

The authors introduced a new Python interface for the PDAF software. The idea of implementing a Python wrapper for a sophisticated Fortran library is much welcomed, since coding in Python is much easier and it will allow more rapid development especially for new models written in Python. However, I found the authors missed the opportunity to convince the readers in this paper that pyPDAF will provide them with tools to quickly implement DA in their Python models. Instead, a lot of focus is put on the details such as second order exact sampling (in SEIK) and comparing weakly and strongly coupled DA. As a user, I only see some hints of what I need to do to use pyPDAF to build a DA system, but I don't know exactly how after reading the paper. With the benchmarking results the authors seem to prefer the Fortran-based PDAF anyway. The test case dimension is very small comparing to typical application, so there is a question whether the Python-Fortran type conversion overhead is significant if pyPDAF is applied to large-dimensional real models. Overall, I feel that the authors didn't promote the new software with convincing enough results.

Answer: We thank the reviewer's constructive advice and we revised the manuscript carefully. In our revised manuscript, we show that pyPDAF is usable for high-dimensional systems with limited overhead compared to the Fortran implementation. We reduce the description of second order exact sampling. We added one section that describes the design and implementation of a DA system using pyPDAF. We also revised the subsection that compares the computational efficiency of pyPDAF and PDAF using different dimensions of the state vector up to a large grid size of 2049×2049 grid points.

Major issues:

1. While Figs 1 and 2 provide the overview of software architectures, the readers will not understand exactly what's needed from their side to build a typical DA system fully based on Python. Can you provide an example with a typical cycling DA experiment setup (initial ensemble generation, ensemble forecasts, compute observation priors, collect/distribute state, assimilate algorithm, etc), and highlight the functions where users can either code their own version, or use something readily available from PDAF

core? The OMI is also quite confusing, what functions does it provide? I feel that you have omitted these details because they are available either in PDAF documentation or in previous papers, but listing the details here can convince readers more effectively that it is ”easy” to build a system using pyPDAF.

Answer: To address the issue of lack of example, we add a new section (Section 3.3): “Construction of data assimilation systems using pyPDAF” where we introduce the necessary user-supplied functions and their intentions for the local ensemble transform Kalman filter as an example. In this section, we also introduce some OMI functionalities and describe the need for additional user-supplied functions without OMI.

2. There are excessive details on the second-order exact sampling which I found not relevant for the topic of this paper. You can use SEIK to demonstrate that the software works, but there are many other options in PDAF that also available. You should definitely save the page limit for something more important. The same thing applies to the comparison between the weakly and strongly coupled DA. The main theme of the paper should be 1. providing the details of how pyPDAF is designed, 2. what’s its advantage over other Python DA prototypes, 3. how to use it to build a Python DA system, then 4. some results from a test case. Currently number 2 and 3 are quite missing.

   Answer: The second order exact sampling is used to generate the initial ensemble for experiments in this manuscript. As it is not the central of this manuscript, we reduce the content of the second order exact sampling in the revised manuscript. The weakly and strongly coupled DA is taken as an example to demonstrate the use of pyPDAF. However, we recognise that the original Sect 6.1 and 6.2 were too verbose and merged these sections into one and removed some content. We also removed the original Section 4 that introduce coupled DA. To address the point 2, in introduction, we added a discussion on the pyPDAF and other Python-based DA packages in terms of its applicability, efficiency and limitations:

   “For the application of DA in Python, DAPPER provides a variety of DA algorithms for twin experiments using low-dimensional Python models. The Ensemble and Assimilation Tool, EAT (Bruggeman et al., 2024) was proposed to set up a 1D ocean-biogeochemical DA system, which is a wrapper to a Fortran data assimilation system based on PDAF including the 1D ocean-biogeochemical model, GOTM-FABM. There are also Python packages designed mainly for pedagogical purposes in low-dimensional systems such as openDA (Ahmed et al., 2020) and filterpy (filterpy PyPI, last access: 2024-08-29). For high-dimensional applications, there are efficient implementations of DA packages such as HIPPYlib by Villa et al. (2021) and ADAO (SALOME The Open Source Integration Platform for Numerical Simulation, last access: 2024-08-29), but HIPPYlib does not have a focus on ensemble data assimilation approaches whereas ADAO provides various ensemble DA methodologies but it has no support for the localisation used in weather and climate applications. More recently, NEDAS (Ying, 2024) was recently introduced for offline ensemble DA in high-dimensional climate applications but it currently only supports limited DA algorithms.”

   Point 3 is addressed in the new Section 3.3 of “Construction of data assimilation systems using pyPDAF”.

3. Efficiency benchmark in Figure 8 is performed for a really small model ($129 \times 129 \times 3$) and ensemble size (16), the number of observations also really small (17×17?) What the figure is conveying is that coding in Python introduced additional overhead and the resulting software is much slower, therefore using the Fortran code is better? I would argue otherwise: for the test case chosen here, if it only take 0.02 seconds per cycle, I would much prefer coding in Python since it will save me days of work compared to coding in Fortran. It doesn’t matter if its 0.02 seconds or 0.005 seconds, they are trivial compare to the ”human overhead”.

   You miss the opportunity here to demonstrate that pyPDAF is a good alternative to PDAF: it should be close to the PDAF efficiency despite the Python-Fortran conversion overhead, but much favorably reducing human overhead in coding. We all know how the ETKF scales with problem sizes, it is roughly $O(Ne^3 + Ne^2 Nobs)$ per state variable. Arguably if the problem size is much larger, and computation spent on the DA problems increase, the overhead of doing the Python-Fortran object conversion can become trivial in propotion to the whole cost. Isn’t that a better story to tell?

Answer: The original intention of the efficiency benchmark is purely to investigate the potential efficiency loss in Python and Fortran implementation. We added tests with increasing spatial resolutions up to 2049×2049 number of grid points for global filter. The results show encouraging results for using pyPDAF with a global filter. The pyPDAF performance is similar to PDAF even when the size of the state vector is at a magnitude of 16 million where the overhead becomes less significant compared to the intensive numerical computation.

We also added one test with domain localisation. In this case, the computational cost increases significantly. The domain localisation also introduces increased overhead for some user-supplied functions as they’re executed repetitively for each local domain. To address this issue, we introduced a new “PDAFlocal” module in PDAF such that the overhead in the domain localisation is reduced and pyPDAF can achieve similar efficiency per analysis step compared to the PDAF implementation.

Minor issues:

1. Line 51, “The Python tool only has a Python interface to a few PDAF routines”, do you refer to EAT? and what’s the difference between this “Python interface” and pyPDAF?

   Answer: We apologise for the clarity issue. The tool EAT is not designed as a PDAF interface. Instead, it is a Python tool which is a wrapper of a Fortran DA system based on PDAF. We rephrased the sentence as ’... which is a wrapper to a Fortran data assimilation system based on PDAF including the 1D ocean-biogeochemical model.’

2. Line 54, "...can facilitate code development thanks to ...", maybe you mean "easier code development", because it is still possible to write Fortran code only a bit more time consuming.

   Answer: We add 'easier' in the sentence.

3. Line 56, "...written onto a disk", do you mean the model restart files?

   Answer: We rephrase the sentence as: "For offline DA systems, DA is performed utilising files written onto a disk, e.g., model restart files."

4. Line 65, "can improve dynamically balanced analysis", need a bit rephrasing, how about "can can improve dynamical balance in the analysis"

   Answer: This is rephrased.

5. Line 77, "This allows for an embarrassingly parallel... does not increase with the ensemble size", this statement is a bit too general. The embarrassingly parallel variant is the LETKF where each state variable can be analyzed separately. Some other EnKF variants is more tricky to parallelize. Also, in ETKF the computational cost does increase with ensemble size $O(Ne^3)$, I guess you mean at the end of the statement "does not increase with state dimension"

   Answer: We agree that this statement does not apply to all ensemble DA methods, so we rephrased the sentence to "The ensemble model forecast allows for an embarrassingly parallel implementation which means that, with sufficient computational resources, the wall clock computational time of the forecast does not increase with the ensemble size.". However, it is worth noting that the Monte Carlo methods are theoretically embarrassingly parallel. This is the case for stochastic ensemble Kalman filter or ensemble variational methods. We recognise that the PDAF parallelisation takes advantages of the domain localisation and this is mentioned in the section "In particular, the domain localisation is tailored for efficient parallel implementations that are commonly used in high-dimensional DA systems.".

6. Line 81, this sentence is repeat of the first sentence.

   Answer: We removed "under the Gaussian assumption" in the second sentence.

7. Line 86, "In practice computational resources limit the feasible ensemble size", are you referring to the fact that model forecasts cost a lot so that one cannot run huge ensemble forecasts?

   Answer: Yes. We added "... due to the cost of model forecasts" here.

8. Line 105, 108, you've defined PDAF earlier, so why repeating the full name over and over?

   Answer: We removed the full name.

9. Line 110, smoother, 3DVar and other non-linear filters, these are not introduced earlier, either mention these in the introduction, or adding some references here.

   Answer: We added relevant references.

10. Lines 113-120, why is the detail on second-order exact sampling provided here. PDAF not only provides SEIK but also a lot of other filter types, the SEIK-specific details should be moved to later maybe in the experiment design.

    Answer: The second-order exact sampling is the default method provided by PDAF for ensemble generation (see: `https://pdaf.awi.de/trac/wiki/EnsembleGeneration`), and we use this to generate initial ensemble. This sampling method is not limited to the SEIK filter, but was found to be very efficient in previous studies. As discussed in the major comments, we shortened the second-order exact sampling description. We keep the description in place as we consider it to be part of PDAF features instead of an experimental choice.

11. Line 131, the OMI is introduced here in one sentence, can you provide more details. How does the user utilize the functions provided in OMI? Do they import them through the pyPDAF interface or do they have to write their own Cython code to call their Fortran versions?

    Answer: The functionalities are added in place and all coding is in Python so that no Cython code has to be written by a user (see response in point 13 as well) .

    "The OMI routines handle the processing of observation vectors and error covariance matrix used by DA algorithms, provide support for the complex distance computation used by localisation. In the current version of PDAF V2.3, it also supports spatial interpolations on structured and unstructured grid, direct observation operator, and a diagonal or non-diagonal observation error covariance matrix. One can implement PDAF without OMI, but additional functions would be required.".

12. Line 134, "replaced by model restart files", this is not trivial to implement, how exactly can this be done? Since this paper only compared two online DA systems using PDAF and pyPDAF, I'm not sure implementing the offline DA is even relevant here.

    Answer: We agree that implementing I/O routines for the restart files can be a non-trivial task. We briefly mentioned that "...while the user-supplied collection and distribution routines manage the I/O operations of these restart files." Considering that offline DA can potentially be an important use-case of pyPDAF, we keep this sentence and add: “Offline DA implementation is a crucially supported feature of PDAF and a potentially important use-case for pyPDAF, but we will not discuss it in detail for the sake of brevity.”

13. Figure 2: There are several levels of “user supplied” functions, in both Python and the C interface. This is confusing. As a user using pyPDAF, do I need to code in Python and compile the package, or do I have to also write Cython code? Or, is it just two options?

    Answer: We add the following sentences: “pyPDAF is designed so that a DA system can be coded purely in Python including the user-supplied functions and function calls to algorithms implemented in PDAF. The interface to PDAF is provided through functions implemented using Cython, which provides the interface for calls from Python. Thus, the pyPDAF package itself is a mixed program of C, Fortran and Python.”, and added sentences in the caption of Fig. 2: “Here, only the Python component is exposed to pyPDAF users, and the Cython and Fortran implementations are internal implementations of pyPDAF”

14. Line 138, “Due to” → ”Thanks to”

    Answer: Changed.

15. Line 141, “...also allows for an efficient code development and modifications...” this sentence needs some work. I get the first half that pyPDAF allows you to build an online DA system for a Python model. But why does the second half relate to the first half?

    Answer: We changed the sentences to: “Hence, a Python interface to PDAF allows the design of an online DA system with such Python-based models. These range from low-dimensional toy dynamical systems to high-dimensional weather and climate systems. Compared to a Fortran-coded DA system, a Python DA system can be implemented efficiently and allows for easier modifications such that users can focus on scientific problems.”

16. Line 143, “...before performing an optimised implementation for high-dimensional Fortran-based models”, there are Python models that are high-dimensional with well optimised numerics, I don’t see the point here why using pyPDAF for a high-dimensional system is not possible now for a prototypical system.

    Answer: We agree with the reviewer that pyPDAF can be used with high-dimensional Python models. We changed the sentence mentioned in comment 15 to “Hence, a Python interface to PDAF allows for designing an online DA system with such Python-based models. These ranges from low-dimensional toy dynamical systems to high-dimensional weather and climate systems.”

17. Line 149, “pyPDAF fully supports the parallel features”, can you provide more details how the MPI featuers are utilized in the Python interface, in Fig. 2 is every function run with MPI, is the Python code run with mpi4py?

    Answer: Following the sentence, we added “The application of pyPDAF in high-dimensional models can also be shown by its support of the parallel features of PDAF, which use the Message Passing Interface (MPI, Message Passing Interface Forum, 2023). For this, a pyPDAF DA system relies on the “mpi4py” package for MPI support. The pyPDAF system can also support shared memory parallelisation in PDAF when built with OpenMP. ”

18. Figure 4: the error time series seems to be not reaching steady state, it keeps increasing and there is a sign of exponential growth towards the end. Is the filter actually stable in time?

    Answer: We apologise for the clarity of the figure. As suggested in its caption, Figure 4 was a time series of the RMSE and ensemble spread of free run that is not constrained by DA. To avoid confusion, we added RMSE and ensemble spread of the SCDA results as a comparison with revised figure caption as well.

19. Line 265, ”16 members...ETKF without spatial localization”, given the results in Fig. 4 maybe you want to add some localization to stablize the filter.

    Answer: As indicated by the results in the new Figure 4, the global filter leads to stable ensemble spread and RMSEs. This is consistent with the results by Tondeur et al. (2020).

20. Line 278, this is not true for the final 100 years, errors are larger than spread.

    Answer: We’d like to stress that the free run ensemble spread follows the trend of RMSEs and it is difficult to attribute the low spread to the initial ensemble considering the long model run. We added: “even though the spread is lower than the RMSE after 200 years”

21. Figure 8: you didn’t provide information on all the subroutines, what does “init dim obs” mean?

    Answer: We change the name of “init dim obs” to “OMI setup”, which hopefully is clearer than “init dim obs”. We add an explanation of “init dim obs” in Section 3.3:

    “The primary purpose of the function is to obtain the dimension of observation vector, *dim_obs* at the current time step as implied by its name. In this function, one has to provide further observation information to OMI.”

References

Ahmed, S. E., Pawar, S., and San, O.: PyDA: A Hands-On Introduction to Dynamical Data Assimilation with Python, Fluids, 5, https://doi.org/10.3390/fluids5040225, 2020.

Bruggeman, J., Bolding, K., Nerger, L., Teruzzi, A., Spada, S., Skákala, J., and Ciavatta, S.: EAT v1.0.0: a 1D test bed for physical–biogeochemical data assimilation in natural waters, Geoscientific Model Development, 17, 5619–5639, https://doi.org/10.5194/gmd-17-5619-2024, 2024.

filterpy PyPI: `https://pypi.org/project/filterpy/`, last access: 2024-08-29.

Message Passing Interface Forum: MPI: A Message-Passing Interface Standard Version 4.1, URL `https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf`, 2023.

SALOME The Open Source Integration Platform for Numerical Simulation: `http://www.salome-platform.org/`, last access: 2024-08-29.

Tondeur, M., Carrassi, A., Vannitsem, S., and Bocquet, M.: On temporal scale separation in coupled data assimilation with the ensemble kalman filter, Journal of Statistical Physics, 179, 1161–1185, https://doi.org/10.1007/s10955-020-02525-z, 2020.

Villa, U., Petra, N., and Ghattas, O.: HIPPYlib: An Extensible Software Framework for Large-Scale Inverse Problems Governed by PDEs: Part I: Deterministic Inversion and Linearized Bayesian Inference, ACM Trans. Math. Softw., 47, https://doi.org/10.1145/3428447, 2021.

Ying, Y. M.: nansencenter/NEDAS: v1.0-beta, Zenodo [code], https://doi.org/10.5281/zenodo.10525331, 2024.