Reply to referee #1

I thank anonymous referee #1 for his/her comments which have improved the manuscript. New manuscript text is italicized in the replies.

Comment: The abstract and first paragraph end with ellipses (!), there are abundant grammatical errors and typos, oftentimes the language used is casual (e.g., “I will just list rapidly the most important ones”) and numerous statements are strikingly vague (e.g., “the tool can be used in 0D and 1D mode, with schemes coming from different models and with different time-advance methods to produce different kind of plots”; “it led to problems maybe specific to our environment or source code”).

Response: The manuscript has been corrected by a native writer of English.

Comment: The content of the paper encompasses atmospheric modelling techniques and technicalities of Python-Fortran interfacing. None of these two subjects are covered in sufficient detail in my opinion (perhaps focusing on just one of those would be a path forward)? Noteworthy, already the title of the paper implies description of improvements to a particular microphysical scheme. These improvements, described in section 3.1, are presented as purely textual description with vague statements, e.g. „the graupel growth mode choice was updated. It is now more continuous and, hence, less time-step dependent”. Such approach does not match GMD’s standards aimed at clarity with respect to model formulation and versioning. The author does explicitly state that the “purpose of the paper is not to give an extended review of all the modifications”, yet in my opinion the way the model development is documented in the paper goes against GMD policies.

Response: The goal of the paper is to present the 0D tool. The ICE microphysical scheme and the modifications that have been applied on it are only there to illustrate the PPPY behaviours. To make it more clear, I suppressed the name of the scheme from the title, the abstract is somewhat rewritten and the improvements to the ICE scheme have been moved to an appendix to suppress them from the manuscript body.

Moreover, some details have been added in the section 2.1.2, in addition to the ctypesForFortran details in section 2.1.1 (in response to your comment below), to improve the OD presentation.

Comment: The Python-Fortran interfacing subject, covered in section 2, is presented with similar level of vagueness. The key components of the presented software included in the 11k LOC ctypesForFortran.py file are not discussed at all. Overall, I expect that independent use of the presented PPPY package, would not be easier than obtaining analogous functionality “from scratch” using a general-purpose Python package providing abstractions for interfacing compiled code (e.g., CFFI which has numerous documented examples depicting its usage with Fortran code and NumPy arrays).

Response: I do not consider that ctypesForFortran is the key component of the software because PPPY users can use ctypesForFortran, f2py, directly ctypes or another tool such as CFFI. However, I included a short description of the main features ctypesForFortran includes:

> *The PPPY user is free to use whichever Python-Fortran interfacing method he chooses (among the two aforementioned or other ones). The ctypesForFortran way intends to help the interfacing of Fortran functions and subroutines on a Linux system. It handles memory allocations and array memory order. Internally ctypesForFortran uses the Python ctypes module (which normally handles the C shared libraries) to interact with the library without*

adding a C or Fortran layer. It deals with Boolean, strings, integers and floats (32- and 64-bits) but does not support structures. The array and string arguments must be explicitly defined (no ``:``, ``.." or ``" are allowed in the interfaces) and no argument can be optional. If this is not the case, a wrapper must be written in Fortran meeting these requirements and calling for the original subroutine.*

In addition, to be more concrete, I included an annexe to give an example of PPPY usage.

When we encountered problems with f2py, we found easier to bypass these problems by writing the ctypesForFortran module that we can control. Maybe it exists a universal interfacing tool which is suitable in all circumstances but a rapid test shows that CFFI also brings problems concerning boolean scalars with some compilers (the binary representation of a Boolean scalar with Fortran is different depending on the compiler, eg. intel vs gfortran). I expect that this problem can be solved with a Fortran 2003 compatible compiler using “bind(c)” but this was not an option when we wrote ctypesForFortran.

Comment: Although, in principle, I would be reluctant to call something “too basic”, reading the manuscript I felt puzzled with regard to the intended audience of the paper. I feel confident that GMD readers do not require repeated verbose explanations on what numerical diffusion is and why it vanishes for integer Courant numbers. The same concerns such statements as: „Python was chosen because it allows to make plots ...”, “the computational time can be large when very small time steps are used” or “One process must take into account that a given specie can be consumed or produced, in the same time, by another process”.

Response:

- For the verbose explanations on numerical diffusion: Sect. 3.2 have been rewritten
- For the python choice: I’m sure many readers know that Python can make plots but what is important is that a **single** language can produce a plot **and** interact with a compiled code. In the revised manuscript, two sentences have been merge to be more concise and to not appear to be too basic:
  The tool consists of a Python package which drives the simulations and performs the comparison: initialization, the calling of the Fortran routines (using the original source code of the parametrization), the saving of the results (in HDF5 files using the h5py module) and the plotting of the results (through the matplotlib module).
- The remark about the computational time that can be large despite of being in a 0D mode is suppressed.
- For the interaction between processes, this can appear to be too basic but it is important to mention it because 1) this interaction is not taken into account in a number of microphysical schemes (except by preventing negative values for the hydrometeors) and this induces an uncertainty on the results, and 2) this is the reason why the splitting was introduced in the ICE scheme. I slightly reordered the sentences to exhibit more the relation between the interactions and the splitting:
  The modifications listed above aim at suppressing the time-step dependency present inside each of the microphysical processes. These modifications were sufficient to suppress or, at least, limit the dependency until time steps around 10 s (not shown). For greater time steps, each process must take into account that a given species can be consumed or produced, at the same time, by another process and that, therefore, this affects its efficiency. To address this issue, some kind of splitting was needed to reduce the effective time step used in the microphysical scheme.

Comment: Below, I am listing some more specific comments that perhaps can be helpful for the author, and that support my opinion outlined above:

- avoid frequent use of the word "tool" (over 40 occurrences including all but one sentences

of the abstract)

- avoid ellipses
- do not use programming notation such as “1.E-5” in the text
- time step vs. time-step, etc - please be consistent
- ensure the use of the words “statistical” and “physical” is justified for all its occurrences
- please do not call something “classical” without reason
- following phrases have certainly better alternatives: “home made”, “some behaviors of a scheme”, “object made from a class”, “intensity of the 0D simulations”, “more the content is important, more the fall is rapid”, “weak content”, “leads to do approximately the same computation”, “content is artificially put higher”
- capitalise Python
- use vector graphics for figures
- ensure consistency in bibliographic entries: abbreviated (with dots or without) and non-abbreviated journal names

Response: I did my best to take into account your remarks. Some of them need a specific reply:

- 1.E-5 was an error, this is corrected in the text
- The manuscript has been corrected by an English native writer
- The graphics are outputs from PPPY, they are png files

Reply to referee #2

I thank Andrew Barrett for his comments which have improved the manuscript. New manuscript text is italicized in the replies.

Comment 1: As a potential future user of this tool, I would appreciate a greater and clearer introduction of how the tool works. The language is still quite technical in places and certain terms (e.g. “objects”, “libraries” and “decorators”) are likely not well understood by future readers. While I acknowledge that there is extra information in the documentation of the tool, I suggest adding some more description to the paper. In particular I would like to see some more advice regarding the interfacing of the fortran code to the python tool. The current description sounds rather ad-hoc and I’m not clear how I would replicate this method.

Response: I modified Sec. 2.1 to be more didactic:

> *Two kinds of objects exist: those which represent a parametrization, and those representing the comparison. A standard object (an abstract class) is provided in order to define a parametrization (the PPPY box in Fig. 4). This abstract class already contains everything needed to perform the time advance and the saving of results but must be complemented (by inheritance) to incorporate the actual call to the different parametrization codes (Param1 and Param2 boxes of the figure). Finally, each parametrization can be used with different configurations. To achieve this, different instances (Param1.1, Param2.1 and Param2.2 boxes) are created, one for each of the configurations (e.g. time-step length, options specific to the parametrization).*
>
> *For the comparison, the provided class (PPPYComp in the figure) can be used directly or can be complemented (by inheritance, UserComp box in the figure) to add new diagnostics (e.g. new plot kind, computation of a derived variable to plot). An instance of the class is created for each comparison to perform (Comp box). A comparison is defined by the list of the parametrizations to use, the simulation length and the initial state. This comparison instance drives the parametrization instances to carry out the simulations and to plot the result.*

Moreover, an appendix is added to describe with more details the test example which is provided with PPPY. I think it is a good entry point to understand how the tool works before going through the other examples which are more interesting but also more complicated.

Comment 2: There is no description in the paper as to how this tool was used to identify and fix the causes of the time step dependence. The findings themselves are listed on page 9. Some additional description of how this tool enabled these model parameterization errors to be found and fixed is needed. This is the main benefit of this new tool, so it would be useful to see how it should be used.

Response: Please see comment #7

Comment 3: This paper is not the place to discuss in detail the merits of different (and in this case rather simple) sedimentation schemes, therefore I suggest shortening this section to allow for the above expansions.

Response: I reduced the sedimentation section.

Comment 4: The figures in this paper (which, I think, are produced by the PPPY tool itself) are missing units on all axes. Additionally, Figure 3 does not make clear which parameterization

scheme is shown in which panel. There two failings need to be fixed. Ideally, not just in the paper, but also in the code of the tool itself (for future users benefit).

Response: The unit on the time and altitude axis were indeed missing, I added them. And, I moved the unit of the plotted variable from the title to the y-axis (the PPPY user can already define the title and the x/y labels). In addition, I changed the title of the different panels of Figure 3.

Comment 5: There seems to be inconsistent model forcing used for the different schemes (or inconsistent physics regarding condensation in the different schemes). Comparing figures 1, 2 and 3 – the water vapor content (black line) for the shortest time step (1s) converges on a value of around 8 g/kg after 180 seconds, whereas for the other schemes in figure 2 and 3, the water vapor converges on 6 g/kg. What is the reason for this difference? Can it be corrected? I understand that the microphysics schemes will give different results for the hydrometeor concentrations, but the simple balance between temperature, water vapor and condensed water should be more or less the same for all schemes. At the moment it is difficult to compare results between the different figures/ microphysics schemes.

Response: In the ICE scheme, condensation is apart from the other microphysical processes. I first checked the time-step dependency in the condensation part. If condensation is activated (with one call to the subroutine by time step), it tends to hide somewhat the time-step dependency by making the different simulations to converge towards an equilibrium point. In particular with the ICE scheme with the same setup as used in the manuscript, the adjustment would suppress the cloud ice, and thus the time-step dependency of all the processes involving this specie will not be seen. Several setups would then be necessary to explore all the microphysical processes. But, indeed, with the adjustment, the vapour content of ICE with the same setup converges towards 6g/kg.

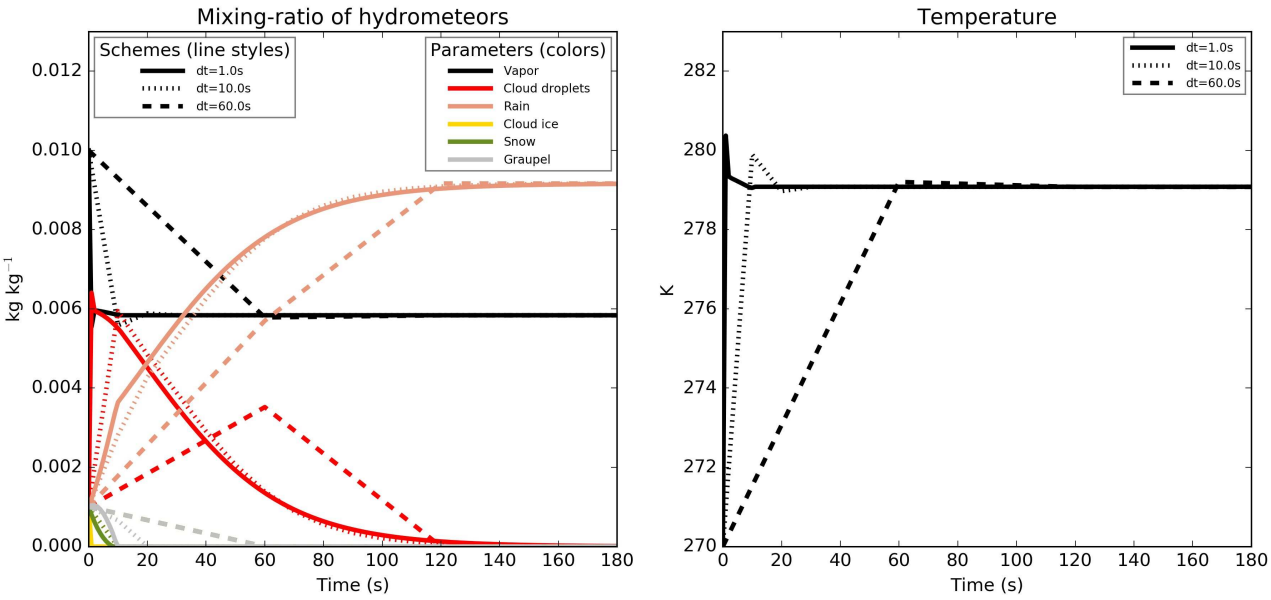

Illustration 1: Same as Fig. 1 of the manuscript but with the adjustment activated

For the LIMA scheme, adjustment is active, it would be easy to deactivate it. But, for the WRF schemes, the condensation process is embedded inside the microphysical parametrizations.

For the current study, I must deactivate the adjustment in ICE to fully explore all the microphysical processes. Because the goal is not to compare the results between schemes but to compare, for each scheme, the simulations for different time steps, I think it is not very important if the setup and/or the active processes differ. In a next study I hope to be able to compare the different schemes; in this next study it will be necessary to pay attention to the setup and to the active processes.

I made several modifications in the text to exhibit more this difference between the ICE scheme and the others:

- *(excluding the saturation adjustment and the sedimentation)*
- *In the simulations performed with this scheme (Fig. 2), the setup is the same as for the ICE scheme but the saturation adjustment is active.*
- *The WRF simulations are performed using the saturation adjustment included inside each scheme.*

Comment 6: page 5, figure 3. It is impossible to tell from figure 3 and the caption, which of the subpanels relate to which microphysics scheme. This should at minimum be added to the caption, and preferable to the figure panels too. The label "Schemes (line styles)" in the top left of each figure could easily be replaced by (e.g.) the scheme name/abbreviation for each panel. The comparison of different microphysics schemes in figure 3 is initiated with a rather unrealistic setup (approximate relative humidity is 165%). In full (3D/4D) model simulations, such supersaturation would never occur, and the microphysics schemes should not be expected to treat such situations fully realistically. Nevertheless, I find the differences between the schemes very interesting – I would be particularly interested to see how these same schemes performed in more realistic setups (e.g. with significantly reduced supersaturation at t=0 and/or when a constant cooling rate is applied)

Response: See comment #4 for the labels.

The setup used in the paper was chosen to allow for a maximum of microphysical processes to be active during a single simulation even if it is not fully realistic. Changing the setup by reducing the vapour content at t=0 (to use 4g/kg instead of 10g/kg) reduces the number of existing species for the ICE scheme (and hence the number of active processes) but does not suppress the time-step dependency. I think the figures with this new setup are less illustrative, I prefer keeping the old ones in the manuscript but you will find the new ones below:

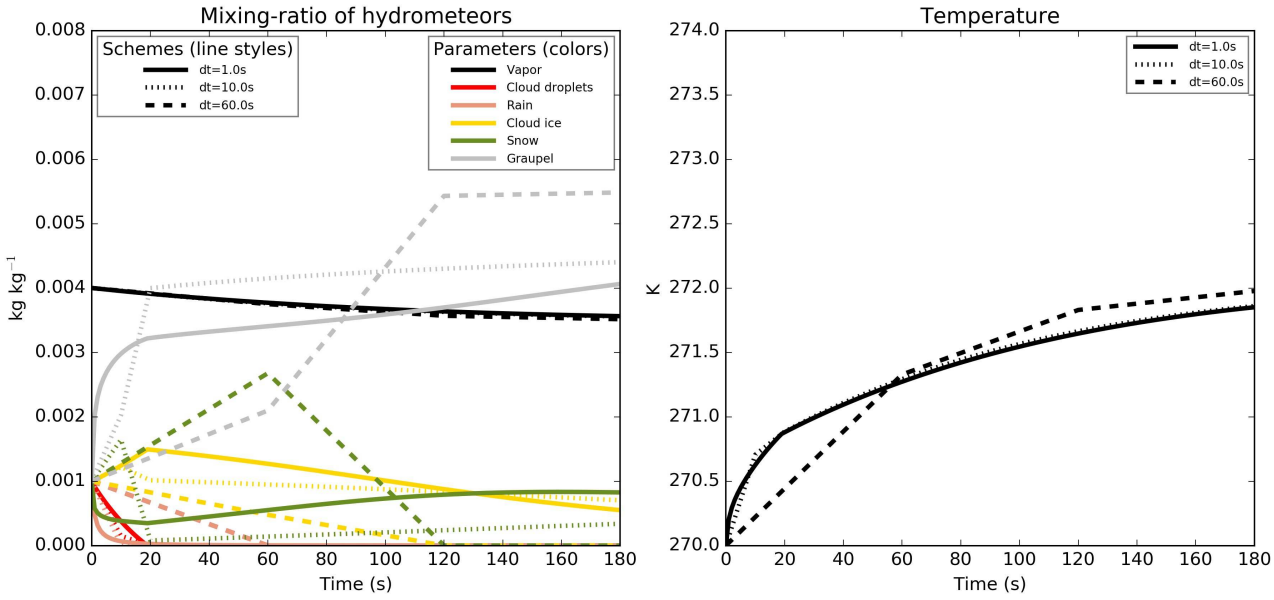

Illustration 2: Same as Fig. 1 of the manuscript but with a drier setup

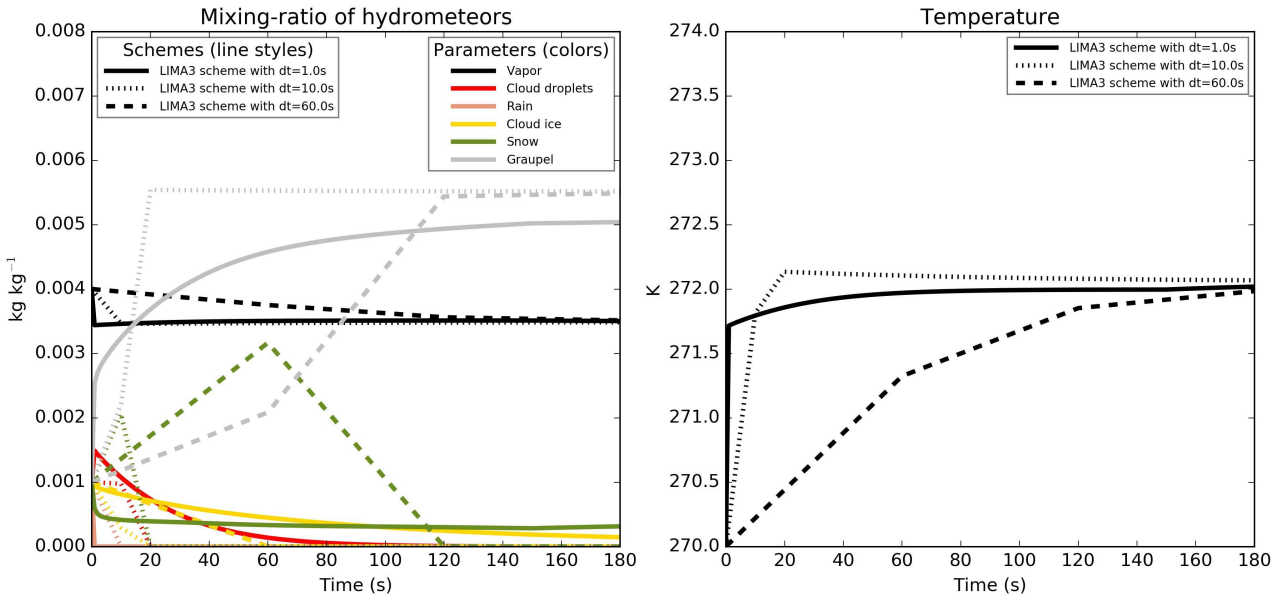

Illustration 3: Same as Fig. 2 of the manuscript but with a drier setup

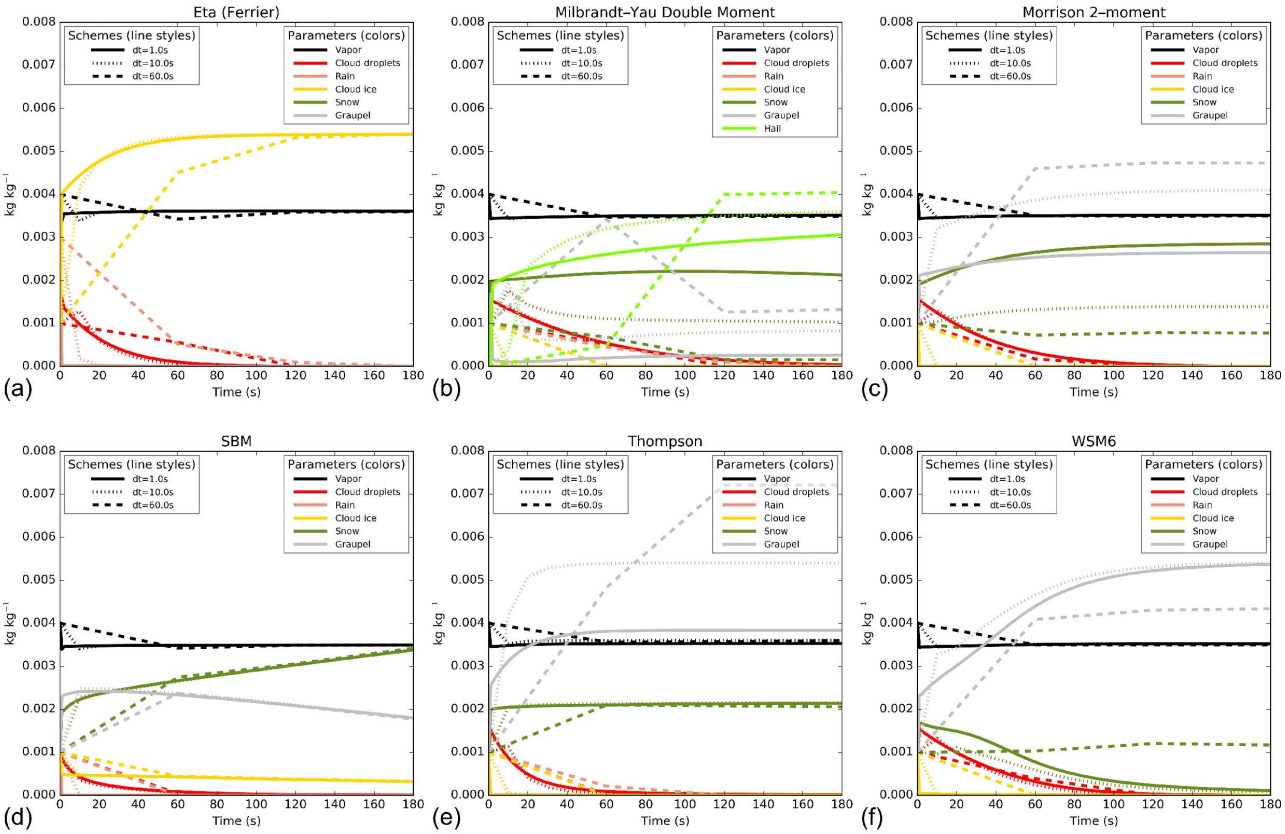

Illustration 4: Same as Fig. 3 of the manuscript but with a drier setup

I added a sentence in the manuscript to inform about the unrealistic setup:

> *The setup is not fully realistic (with an important supersaturation) but allows simulations to involve all the species and, hence, virtually all the microphysical processes. It was checked (not shown) that the time-step dependency still exists when using more realistic initial values.*

Comment 7: page 8, bullet points lines 9-28. These are all very interesting findings, but how did the PPPY tool help you to discover these factors as being important. It would be good to show the benefits of the tool you developed in achieving these findings. Was it simply a trial-and-error

process, or is there some aspect of PPPY that enables these errors to be determined more quickly?

Response: Unfortunately, I didn't find a better way than using a trial and error process (by enabling or not the different microphysical processes).

> *The simulations have been carried out several times activating and deactivating the different microphysical processes. To do this, the processes have been called individually by the PPPY software (when they are written in separate subroutines) or activated through switches or, at worst, (un-)commented in the source code. This trial-and-error process makes it possible to identify the processes that led to the oscillations and to the time-step dependency, and allowed the checking of each correction individually from the others.*

Comment 8: page 8, line 27-29 please clarify what you mean by "the conversion rate of graupel into hail is now computed from the wet growth rate of graupel and not from the total content of graupel". How large is this difference and why does it make such a difference?

Response: In the scheme the graupel is produced by the snow collecting liquid water, then the graupel growths by collecting other species (and with vapour deposition). If the graupel collects liquid species (rain or cloud), then there are two possibilities: the graupel is able to freeze the liquid content collected (this is called the dry growth mode) or not and a thin liquid film appears at the surface graupel (this is called the wet growth mode). The choice is made with the help of a heat budget.

> *the graupel growths mainly by collecting other species. When this collection implies liquid species (rain and/or cloud), there are two possibilities (called growth mode) depending on a heat balance: the graupel is able to freeze the entire collected liquid collected (dry mode) or a thin liquid film appears at the graupel surface (wet mode). In the original version of the ICE scheme there was confusion between the maximum content of liquid water than can be frozen (which must be used in the heat balance) and the content of liquid and ice water that can be collected in wet growth mode (which must be used to compute the graupel tendency). The correction made the mode choice more continuous. And because, ultimately, the graupel growth mode has an impact on the collection efficiency of icy species (snow and cloud ice) with the graupel, this choice can lead to significant differences in the collection rates. Hence, the scheme including the correction is less time-step dependent;*

If hail is activated, the wet growth mode contributes to the formation of hail. A conversion rate from graupel to hail is computed based on the collection rates and on the heat budget used to choose the growth mode. Then there are two ways of using this conversion rate:

- the old one: the conversion rate is applied to the entire graupel content. A given percentage of the graupel is then transformed into hail. For a same simulation length, depending on the number of time steps used, this conversion is made a different number of times. To simplify the reasoning, if the conversion rate is constant and equal to 0.5, the percentage of graupel converted into hail after 10s is 50% with one time step of 10s and 75% with two time steps of 5s.
- the proposed one: the conversion rate is only applied to the tendency of the graupel. The graupel already present at the beginning of the time step remains graupel and only the collected mass (in wet mode) can be converted into hail. This modification reduces the amount of hail produced and suppresses the time-step dependency.

The proposed version must still be validated but it is better than the old one because the time-step dependency is suppressed and because the old version had a tendency to produce small amount of hail under nearly all precipitating cold clouds (and this is no longer the case with the new version).

> *several modifications have been carried out on the processes involving the hail category as a prognostic field: the processes dealing with hail are now completely symmetric with those dealing with the graupel category (to ensure consistency even if this did not produce time-*

step dependency). A conversion fraction is computed from the heat balance used to choose the graupel growth mode. In the original version of the scheme, this fraction was applied on the total content of graupel; this induced a conversion tendency directly linked to the number of times the rate is applied (hence to the time step for a given simulation length). On the contrary, in the new version, the conversion fraction is applied on the wet growth rate, this way, no time-step dependency is produced. This was the main reason for the time-step dependency on the hail category.

Comment 9: Figures 11, 12 & 13. please make clear that the x-axis is timestep length (dt) – it could also be interpreted as timestep number (i.e. as a time-height plot)

Response: Thank you for the suggestion, it is done.

Comment: Technical/language corrections:

- page 2, line 9-10. please provide mode details about what differences were seen in the Meso-NH model when the time step was changed?

- page 3, line 9. what is a “tool package”. Where can the reader find it?

- page 3, line 13. “makes possible” → “makes it possible”

- page 4, line 1 correct to “consists of a python package written to ...”

- page 4, line 5 delete “which is the required”

- page 6, line 3 “needed to use the parameterization” → “needed by the parameterization”

- page 6, line 6 “that not exist” → “that do not exist”

-page 7, line 4 “to plug other” → “to plug in other”

- page 7, line 5 “one have to define” → “one has to define”

- page 8, line 2 “dependency on the simulation” → “dependency of the simulation”

- page 8, line 11 please quantify what you mean by “small time steps”- page 8, line 18 it is not clear to me what you mean by “graupel growth mode”, please give more details

- page 8, line 23 “has melt into rain” → “has melted into rain”

- page 8, line 25 “insure” → “ensure”

-page 9, line 10 & 11. please use standard scientific notation (e.g. 1.0x10 -5)

- page 9, line 11. please quantify the “substantial additional cost”

- page 9, line 16 “unique” → “single”

- repeated grammatical error (e.g. page 12, line 9; page 12, line 31; page 13, line 1, page 13, line 7): “Longer is the time step, more this part is important” → “The longer the time step is, the more important this part is”

- page 12, line 31. What is the “it” in “it reaches around 11%”?

- page 13, line 5 “mean content is weaker” → “mean content is less”

- page 13, line 20 “by consequences” → “as a consequence”

- page 13, line 24 “this induces” → “this means”

- page 13, line 25 “larger to one” → “larger than one”

- page 14, line 3 “None of both schemes” → “Neither of these two schemes”

- page 14, line 9 “weaker” → “less”

- page 14, line 28 “whatever is the time step” → “whatever the time step is”

- page 14, line 32 “hypothesis done” → “hypothesis”

-page 15, line 2 “certainly reduce” → “certainly reduced”

- page 15, line 10 “This scheme allows to make fall the bigger drops quicker” → “This scheme allows the bigger drops to fall more quickly”

Response: Thank you for the numerous corrections you have suggested. In addition, the paper have been reviewed by a native writer of English. Some of your remarks need a reply, they are below:

- page 2, line 9-10. please provide mode details about what differences were seen in the Meso-NH model when the time step was changed?
    - The test simulations have been done several years ago and are no more available. A rerun of those simulations would be necessary to give extended details on the differences.
- page 3, line 9. what is a “tool package”. Where can the reader find it?
    - *A documentation is provided with the software (see the code availability section).*
- page 8, line 11 please quantify what you mean by “small time steps”- page 8, line 18 it is not clear to me what you mean by “graupel growth mode”, please give more details
    - small time steps: replaced by *simulations using the small time steps (shown in Fig. 7)*
    - graupel mode: please see comment #8
- page 9, line 10 & 11. please use standard scientific notation (e.g. 1.0x10 -5)
    - It was an error, the values are 0.01 and 0.05
- page 9, line 11. please quantify the “substantial additional cost”
    - *The iterations needed for the 0.05g kg^{-1} threshold induce an cost increase of about 5%, and for the 0.01g kg^{-1} threshold an additional 20% can be expected (this last figure is an estimation because no sensitivity test have been performed on the whole domain).*
- page 12, line 31. What is the “it” in “it reaches around 11%”?
    - sentence reformulated: *For the 60s time step, 11% of the total water reached the ground.*

Development of "Physical Parametrizations with PYthon" (PPPY, version 1.1), and its usage to reduce the time-step dependency in ~~the ICE~~ a microphysical scheme

Sébastien Riette[1]

[1]CNRM, Université de Toulouse, Météo-France, CNRS, Toulouse, France

Correspondence: Sébastien RIETTE (sebastien.riette@meteo.fr)

Abstract.

To help develop and compare physical parametrizations such as those found in a numerical weather or climate model, a new tool was developed. ~~The~~ This tool provides a framework with which to plug external parametrizations, run them in an offline mode (using one of the two time-advance methods available), save the results and plot diagnostics. ~~With the help of this tool,~~ The software can be used in an 0D and a 1D mode with schemes originating from various models. As for now, microphysical schemes from the Meso-NH model, the AROME (Applications of Research to Operations at Meso-scale) model and the Weather Research and Forecasting model have been successfully plugged. As an application, PPPY is used in this paper to suppress the origin of the time-step dependency of the microphysical scheme used in the Météo-France small scale operational numerical weather model~~was identified. The sources of dependency lied in some process formulations and in the algorithm used to allow the competition between the different processes. Some corrections have been introduced and their efficiency was checked with the tool. This usage illustrates how the tool can be used in 0D or 1D mode, with schemes coming from different models and with different time-advance methods to produce different kinds of plots.~~ . The tool helped to identify the origin of the dependency and to check the efficiency of the introduced corrections.

1 Introduction

A weather or climate numerical model contains several parametrizations (e.g. turbulence, convection . . .) ~~that interact together and~~ which interact not only together but with the dynamical core. When a parametrization is being developed or debugged, these interactions can hide and/or amplify a tested modification. When the goal is to compare two parametrizations hosted by different models, ~~theses~~ these interactions distort the comparison as the other model components can be very different (dynamical core, discretization and other parametrizations). To circumvent these effects, one can reduce the interactions by unplugging other parametrizations (ideal cases, aqua-planet experiments) ~~,~~ or by reducing the problem size (2D vertical simulations, single column model)~~. . .~~.

The choice of the comparison strategy depends on the intended goal: a full 3D-model is able to represent all the interactions whereas simplified models represent only a subset of these interactions. ~~However, even~~ Even the single column model however (one of the simplest ~~configuration~~configurations) is not always sufficient ~~to separate~~ for separating the impact of the different parametrizations (see, for example, point 8 of **?** conclusion). ~~Hence, a~~ A simpler framework could therefore be useful; this can be ~~toy models~~ a toy model in which only one parametrization is plugged.

An example of a toy model used to compare microphysical schemes is given by **?** ~~with~~ using the Kinematic Driver (KiD) model. This is also the approach taken here to develop ~~the PPPY tool in which we~~ PPPY in which one can plug existing parametrizations from different models, deal with the simulations, compare the outputs and plot the results. The tool described here has some common points with the KiD model but is able to deal with any parametrization (not only microphysics), integrates the graphical part and is very flexible ~~by~~ through the use of the ~~python~~ Python language to control the execution flow (for example running and comparing hundreds of different configurations is not an issue ~~, extending the tool by incorporating diagnostics should be simple, …~~ and incorporating new diagnostics is simple). The KiD model, for its part, allows advection and, hence, lies between ~~the tool described here~~ PPPY and a Single Column Model.

~~A difference was observed~~ Development motivation came from the observation of a difference on 3D simulations with the Meso-NH model (**?**) when the time step was changed. ~~Because the~~ The impact of the time step ~~was the most important on~~ being greatest for rain accumulation and ~~also~~ when prognostic hail was activated, the microphysical scheme (**?**, hereafter referenced as ICE) was suspected to be the ~~first responsible~~prime reason. To assess this dependency, simulations in a 0D mode, using only the microphysical core processes (excluding the saturation adjustment and the sedimentation), are performed using ~~the PPPYtool~~PPPY. To test all the processes, an initial state involving all the hydrometeors was chosen. ~~Initial~~ The initial mixing ratios were quite ~~important~~ large (10 $\mathrm{g\,kg^{-1}}$ for vapor, no hail and 1 $\mathrm{g\,kg^{-1}}$ for the other hydrometeors) and hail was activated (even if the illustrations here were made with simulations without hail to simplify the plots); the initial temperature was set to 270 K. The setup is not fully realistic (with an important supersaturation) but allows simulations to involve all the species and, hence, virtually all the microphysical processes. It was checked (not shown) that the time-step dependency still exists when using more realistic initial values.

When several hydrometeors are mixed in a model cell without exchange with the exterior, the microphysical processes tend ~~to an equilibrium state . And, this~~ towards a state of equilibrium. This final state must not depend on the time step used. ~~Moreover~~In addition, when two (or more) simulations ~~run~~ running with different time steps are compared, they should have the same results for common output times. In Fig. 1, we can see that the final state depends on the time step used (between 1 s and 60 s) for water vapor, rain and temperature. ~~Furthermore, the~~ The chaotic appearance of this plot is furthermore, a signature of the time-step dependency. Without this drawback, all curves of a same color ~~normally would~~ would normally follow the same time evolution. For example, after 60 s of simulation (~~which corresponds~~ corresponding to the order of magnitude of the time-step length used in the Météo-France small scale operational numerical weather model, AROME (Application of Research to Operations at Mesoscale, **?**), which share the same physical package with the Meso-NH model), ~~a~~ great uncertainty exists on the hydrometeors presence; depending on the time step used, rain, graupel and snow ~~can exist~~ may (with very significant content) or ~~not~~may not exist.

In the COnsortium for Small-scale MOdeling (COSMO) model, **?** also observed a time-step dependency on rain and hail accumulations ~~that they~~ traced back mainly to the interaction between the dynamics and the physics of the model, and, to a lesser extent, to some microphysical processes. The example shown in this paper demonstrates that a significant part of the time-step dependency can also be explained by the microphysical scheme itself.

The time-step dependency of the microphysical scheme is not specific to the ICE parametrization. The dependency is also observed ~~(Fig. 2)~~ with the Liquid Ice Multiple Aerosols (LIMA) scheme (**?**) (a quasi two-moment microphysical scheme in development in the Meso-NH and AROME models). ~~And, such~~ In the simulations performed with this scheme (Fig. 2), the setup is the same as for the ICE scheme but the saturation adjustment is active. Such a dependency was also observed in the Integrated Forecasting System (IFS) model (**?**), and ~~were~~ related to the formulation of the warm-rain processes. Moreover, some microphysical schemes of the Weather Research and Forecasting (WRF) Model (version 3.9.1.1) have been plugged ~~in the tool~~ and also exhibit time-step dependency, as shown in Fig. 3 for the Eta (Ferrier) scheme (**?**, panel a), the Milbrandt–Yau Double Moment scheme (**??**, panel b), the Morrison 2–moment scheme (**?**, panel c), the Hebrew University of Jerusalem Spectral Bin Microphysical (HUJI SBM) scheme (**?**, panel d), the Thompson scheme (**?**, panel e) and the WRF Single–moment 6–class (WSM6) scheme (**?**, panel f). ~~These example~~ The WRF simulations are performed using the saturation adjustment included inside each scheme. These examples also show how ~~the PPPY tool~~ PPPY can be useful ~~to exhibit some~~ for the exhibiting of some of the behaviors of a scheme (such as time-step dependency, oscillations ~~, water conservation, . . .~~ and water conservation) independently of the other model components.

Section 2 describes the technical choices and provides an overview of what can be done with the ~~tool~~software. Some examples of usage are given in Sect. 3 before the conclusion.

2 Functionalities and technical aspect

~~A documentation is available in the tool package~~Documentation is provided with the software (see the code availability section). To complement this documentation, this section gives some details on how to add a parametrization~~and describes the~~ , how the software works as well as describing the functionalities related to the parametrizations and ~~to~~ the comparisons.

2.1 Technical aspects

~~Python was chosen because it allows to make plots (through the matplotlib module for this tool), deal with common file formats and interact with compiled code (this feature makes possible to use the original source code of the parametrization).~~

The tool consists ~~in a python package is written to drive the simulations (initialization,~~ of a Python package which drives the simulations and performs the comparison: initialization, the calling of the Fortran routines ~~, saving of the results, plotting results).~~

~~Even with simulations limited to only one point, the computational time can be large when very small time steps are used; to avoid recomputing simulations already run, the script saves the results of the different simulations~~ (using the original source

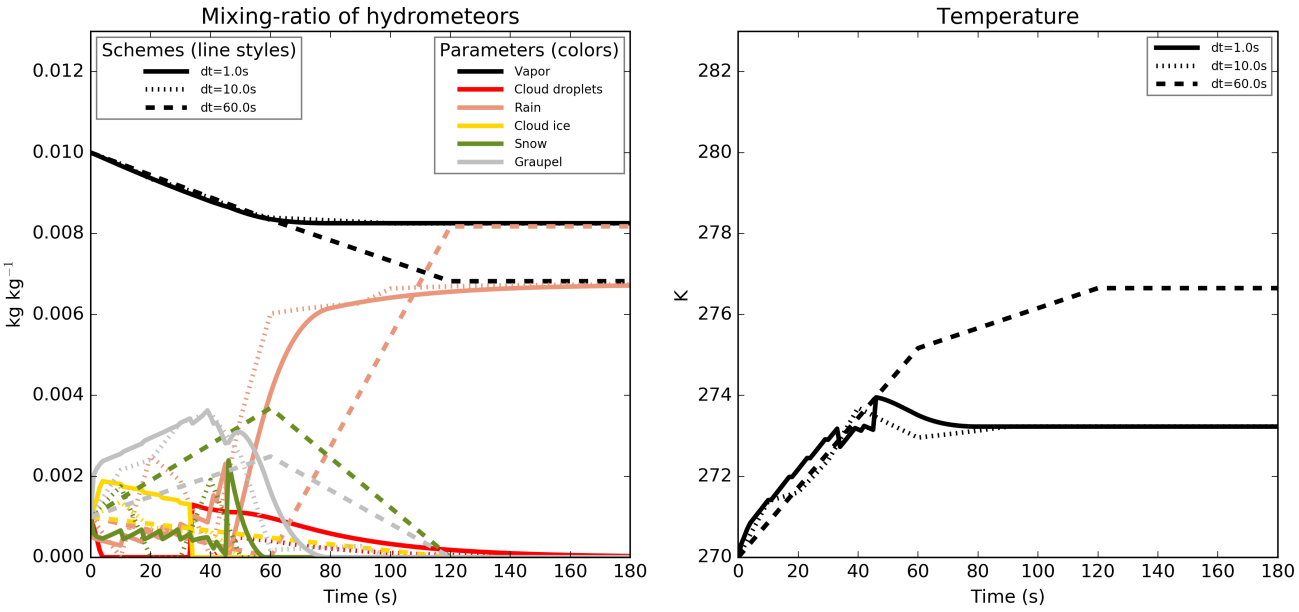

Figure 1. Time evolution of the mixing ratio of the different hydrometeors (in kg/kg, left panel) and of the temperature (in K, right panel). The simulations were performed using time steps between 1 s and 60 s.

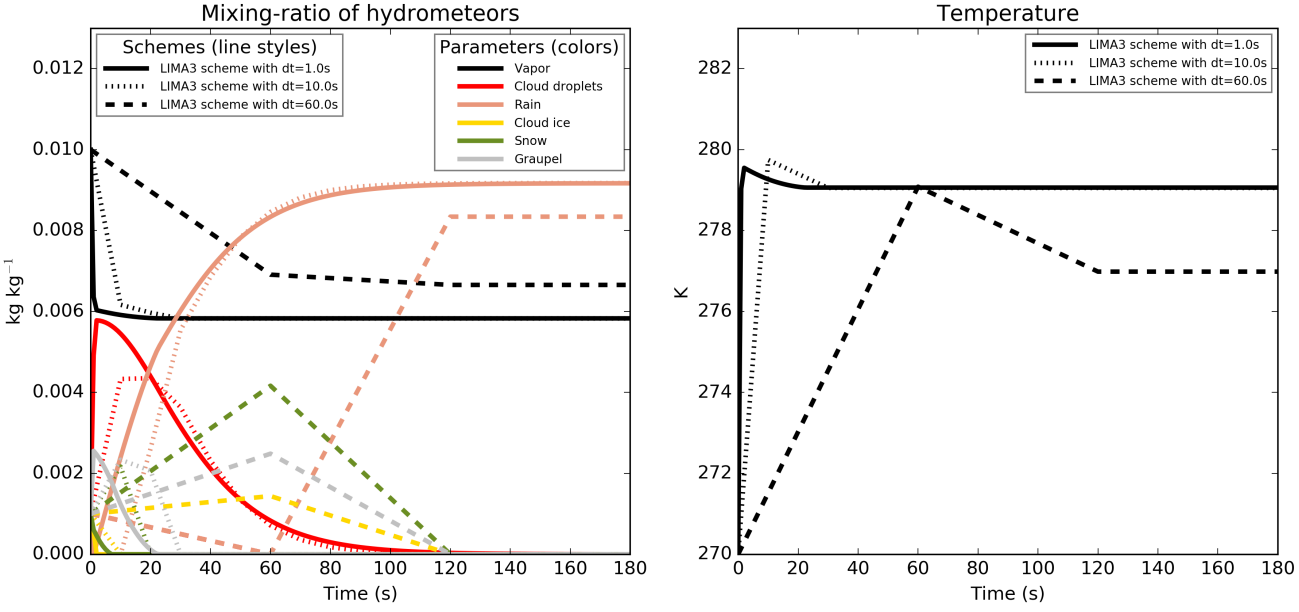

Figure 2. Same as Fig. 1 but using the LIMA scheme.

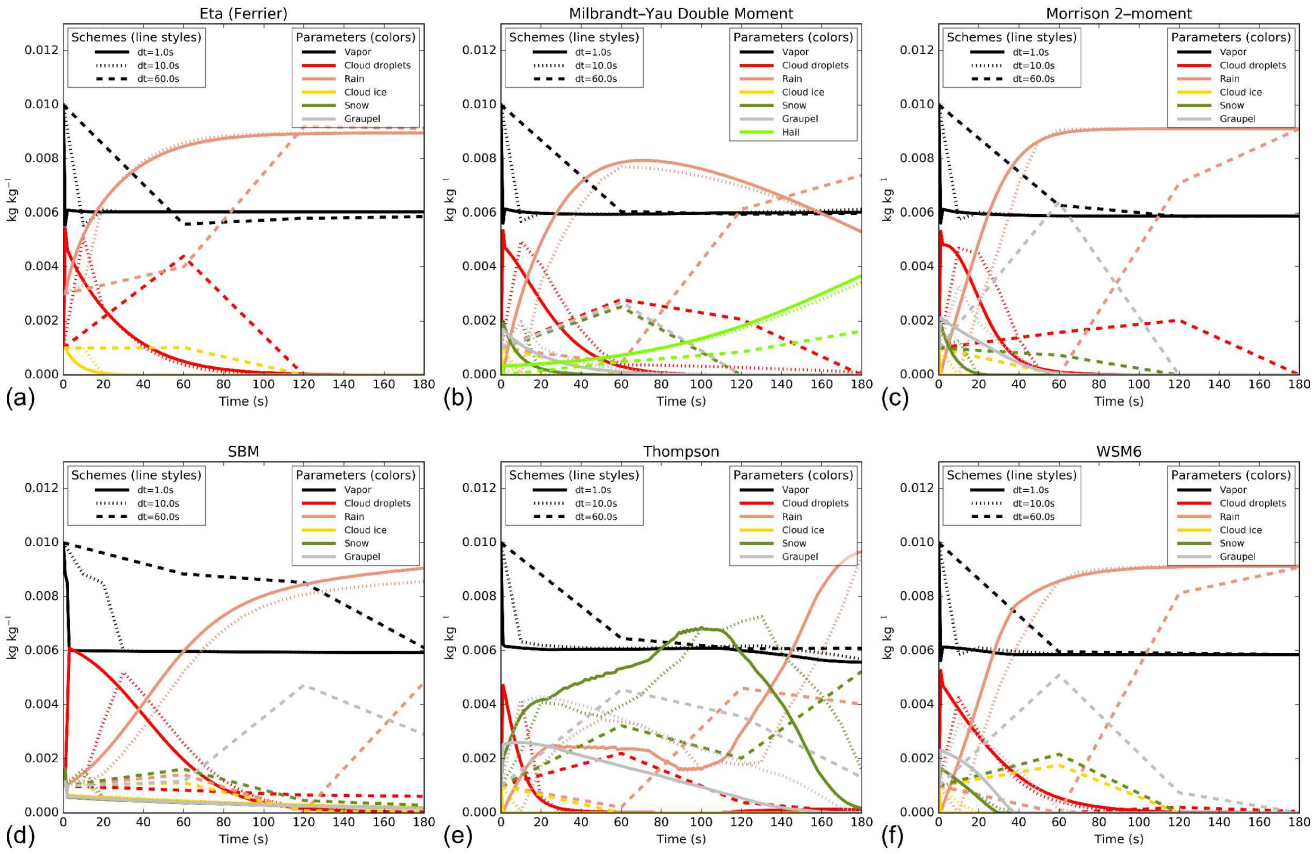

Figure 3. Same as left panel of Fig. 1 but using some schemes of the WRF model identified in the panel titles (see text for complete references).

code of the parametrization), the saving of the results (in HDF5 files ~~(~~using the h5py module~~which is the required~~) and the plotting of the results (through the matplotlib module).

~~The general design of the tool is as follow (see~~ Two kinds of objects exist: those which represent a parametrization, and those representing the comparison. A standard object (an abstract class) is provided in order to define a parametrization (the PPPY box in Fig. 4)~~: each parametrization code (built from FORTRAN source code for example) is associated to a python object (made from the provided PPPY class)~~. This abstract class already contains everything needed to perform the time advance and the saving of results but must be complemented (by inheritance) to incorporate the actual call to the different parametrization codes (Param1 and Param2 boxes of the figure). Finally, each parametrization can be used with different configurations. To achieve this, different instances (Param1.1, ~~and then, these different PPPY python objects are used by a PPPYComp object which performs the comparison.~~ Param2.1 and Param2.2 boxes) are created, one for each of the configurations (e.g. time-step length, options specific to the parametrization).

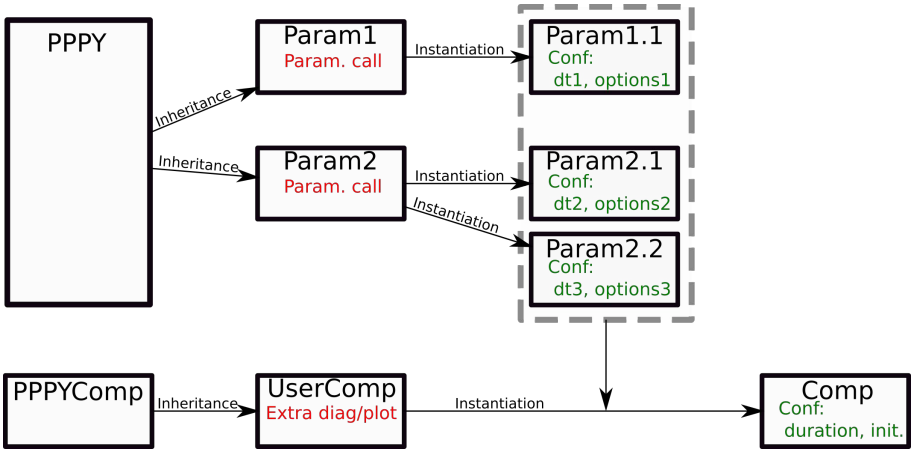

Figure 4. Tool organisation diagram. The 6 top boxes represent the parametrization objects while the 3 lower boxes are the comparison objects.

For the comparison, the provided class (PPPYComp in the figure) can be used directly or can be complemented (by inheritance, UserComp box in the figure) to add new diagnostics (e.g. new plot kind, computation of a derived variable to plot). An instance of the class is created for each comparison to perform (Comp box). A comparison is defined by the list of the parametrizations to use, the simulation length and the initial state. This comparison instance drives the parametrization instances to carry out the simulations and to plot the result.

These different objects are described in the following subsections. In addition, in the provided package, the `examples` directory contains, among other items, a test example which is commented on in Appendix A so as to illustrate the different steps described in the current section.

2.1.1 The low level part of the parametrization: source code and compilation

The ~~most tricky~~ trickiest part comes from the interfacing between ~~python~~ Python and the parametrization. This part is quite technical but ~~is~~ important as the main difficulty in using ~~this tool~~ PPPY with a new parametrization lies in this interfacing task.

If the parametrization was written using ~~python~~ Python (like the box-Lagrangian scheme used in Sect. 3.2) the interfacing would be straightforward but numerical weather and climate models often use Fortran and a ~~tool~~ module is needed to ~~do the interfacing~~ perform the interfacing with Python. There are several ways in which to accomplish this task; this paper ~~is not the place to do a review of the different ways buthere~~ does not set out to review these ways but, here, are listed the two ~~that were~~ used at some ~~stages of~~ point in the development process. ~~This two ways are still usable~~ These two ways can still be used even if the examples provided with the ~~tool~~ package use only the second one. Both methods aim at building a library (a collection of binary codes) which contain all the Fortran codes needed to call a given parametrization. The f2py utility can be used; it helps to build a shared library suitable to be imported and used from ~~python but it led to problems maybe specific to our environment or source code (dependency to the exact python version , argument order not preserved...~~ Python but it can be difficult to

use (in particular the built library depends on the exact Python version and the argument order is not always preserved). The second one is ~~home made; it aims at using the ctypes module (normally reserved for C codes) directly over~~ a module (named ctypesForFortran) provided with PPPY which acts directly on a shared library built from the Fortran source code. ~~For each Fortran subroutine (or function) to use, the interfacing consists in writing a python function with the same name as the Fortran subroutine and that returns the signature of it (that is~~ The interfacing consists of defining the signature (the interface of the subroutine written in a specific way) ~~. A decorator (a python object that modifies the behavior of a function) is provided to convert this signature function into a function that actually calls the Fortran subroutine (or function) and returns the result.~~ of each Fortran subroutine or function to employ.

The PPPY user is free to use whichever Python-Fortran interfacing method he chooses (among the two aforementioned or other ones). The ctypesForFortran way intends to help the interfacing of Fortran functions and subroutines on a Linux system. It handles memory allocations and array memory order. Internally ctypesForFortran uses the Python ctypes module (which normally handles the C shared libraries) to interact with the library without adding a C or Fortran layer. It deals with Boolean, strings, integers and floats (32- and 64-bits) but does not support structures. The array and string arguments must be explicitly defined (no ":", ".." or "*" are allowed in the interfaces) and no argument can be optional. If this is not the case, a wrapper must be written in Fortran meeting these requirements and calling for the original subroutine.

For the potential C-written parametrizations, ~~the interfacing must directly use~~ interfacing can directly employ the ctypes module.

~~Because the compilation can be~~ The compilation being a complex process (that can involve scripts that modify, on the fly, source codes), it could be difficult to isolate and compile, outside of the box, the source code needed for a given parametrization. To reduce ~~the~~ this difficulty, the ~~different provided examples~~ various examples provided with PPPY (in the example directory) follow this procedure:

- Modification of the model compilation script and/or Makefile file to include the option to build a position-independent code, suitable for dynamic linking ("-fPIC" option),
- Normal compilation of the model,
- Use of the ~~different~~ various object codes and/or static libraries built during the normal compilation step to build a shared library with the different entry points needed ~~to use~~ by the parametrization.

~~Because the shared object will be opened using the ctypes module (that normally handles C shared library), the Fortran subroutines and functions that will be used from the python script must not make use of some Fortran specific argument properties (e.g. assumed shape arrays, optional, assumed length character strings) that not exist in an identical way in C. If it is not the case, a wrapper must be written in Fortran that receives arguments declared with fixed lengths and calls the original subroutine.~~

At this stage the remaining difficulty is to identify the different routines that must be called upon to perform the parametrization initialization and execution.

2.1.2 The high level part of the parametrization: the PPPY ~~python~~ Python object

Once the compilation part is completed, a ~~python~~ Python object must be created in order to manipulate the compiled library. An abstract class (PPPY) is provided for this purpose and must be used, by inheritance, to build a class specific to the parametrization ~~to use.~~ employed.

The abstract class ~~already contains everything needed to perform the time advance and the saving of results and have~~ has placeholders for the requested standardized methods ~~that~~ and these must be implemented. ~~In these methods, new variables specific to the current scheme can be defined , conversions can be performed (for example to change the temperature variable from potential temperature to true temperature), modifications in memory representation can be done (all variables are 64 bit in the python script but can be converted into 32 bit for instance) and this is the place to make use of the shared library and run the actual parametrization code. All options concerning this parametrization can be made available through the initialization method. Then, this class is instantiated (to get the Param1 and Param2 classes of Fig. 4) by providing some options (e.g. time step , options specific to the parametrization)~~

Following the order of the execution flow, the first method to adapt is that of initialization. In this method, all the available options of the scheme are defined and consistency checks can be achieved. Among these options is the time step one (mandatory); the others being specific to the scheme.

The following method concerns the setup. Here, the computations that need to be done once by simulation are performed. For example, it could be constant definitions, pre-calculation of lookup tables or files fetching.

The initial state provided at the beginning of a simulation is common to all the parametrizations involved in the comparison. This state contains all the variables that must be monitored by the scheme although it is common for a given scheme to need additional prognostic variables. The build_init_state method is the place in which to define and initialize these variables and to add them to the initial state of the simulation. The output of this step is the first state saved in the output file.

The ~~tool was developed and tested to compare microphysical schemes but the code was made very general so it is possible to plug other parametrizations. As a consequence, the list of the physical quantities to monitor and compare is not hard written. Therefore, before trying to compare several parametrizations of a same process, one have to define the list of the variables to monitor with their units. For instance, temperature can be represented as potential temperature or true temperature in Celsius or Kelvin. The~~ execute method is in charge of calling the actual code of the parametrization, making use of the compiled library. It might be necessary to surround this call with conversions. Indeed, the same quantities must be monitored by ~~all the schemes~~ every scheme even if, internally, each scheme uses its own set of variables~~. Conversions to and from these quantities are done around the parametrization call~~; physical conversion (for example changing the temperature variable from potential temperature to true temperature) may therefore be needed. Moreover, modifications in memory representation may also be required (all variables are 64-bit in the Python script but can be converted into 32-bit for instance).

This class is then instantiated by providing the required options (time step, options specific to the parametrization).

2.1.3 Comparison ~~python~~ Python object

The parametrizations, which are instances created from the PPPY class (as stated above), are intended to be used by a ~~python object~~ Python object in order to perform a comparison. This comparison object (instantiated from the PPPYComp class, to get the Comp object of Fig. 4) is characterized by the list of parametrizations to use, the simulation length and the initial state of the simulations. ~~The object is responsible to run the~~ The comparison object is then responsible for the running of the different parametrizations (isolated from each other), ~~to compute diagnostics (that~~ the computing of diagnostics (which can be added by creating a custom comparison class by inheritance) and ~~to plot the results (plots~~ for the plotting of the results (plot methods can also be added).

2.2 Tool functionalities

~~The tool allows to compare~~ PPPY allows the comparison of several parametrizations. The different parametrizations can differ ~~by~~ from the underlying code or can differentiate themselves by the choice of the parameters controlling the scheme. The ~~tool also enables the~~ comparison between two identical parametrizations using different time steps or different time-advance methods is also possible. Two time-advance methods exist:

"step-by-step" like a true simulation, the output is computed from the output of the previous time step (Fig. 5).

"one-step" the output (at all output times) is computed by a direct integration from the initial state (Fig. 6).

The ~~tool was developed~~ development was conducted in such a way as to allow the comparison of any parametrizations, ~~and~~ not only microphysical schemes. The set of variables ~~followed by the tool~~ monitored is not limited to predefined ones; the user can add any variable of any dimensions. Moreover ~~the tool~~ PPPY is able to use schemes from different models (interfacing with AROME, Meso-NH and WRF has been done).

~~Currently two~~ Two plot kinds are ~~available but other~~ currently available but others can be written by extending the tool. ~~The already existing plot methods~~ Plot methods already in existance can draw results for 0D and 1D simulations (the problem size can be reduced -slicing- for plotting ~~; this allows to run the~~ allowing the running of the simulations once on a variety of initial conditions and ~~plot the~~ the plotting of the results for only one point or profile). The y-axis is used to represent the ~~variable~~ intensity of the variables in the 0D simulations or the different points ~~of~~ in the 1D simulations. The two plot kinds differ in the x-axis ~~which is~~ used to represent the time (with different plots superimposed for the different schemes, ~~like~~ such as in the examples of Sect. 3.1) or the different schemes (with different plots superimposed for the different output times, ~~like in some examples~~ as in Fig. 11 to 13 of Sect. 3.2). In this context, the different schemes can be different parametrizations and/or ~~a same parametrization~~ the same parametrization but using different options (constants, configuration options or ~~time step~~ a time-step choice); this allows ~~to perform~~ the performance of sensitivity tests to one parameter. The figures ~~that~~ which illustrate the examples shown in this paper have been directly produced by the ~~tool~~software.

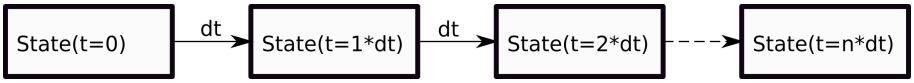

Figure 5. Time advance for a step-by-step simulation.

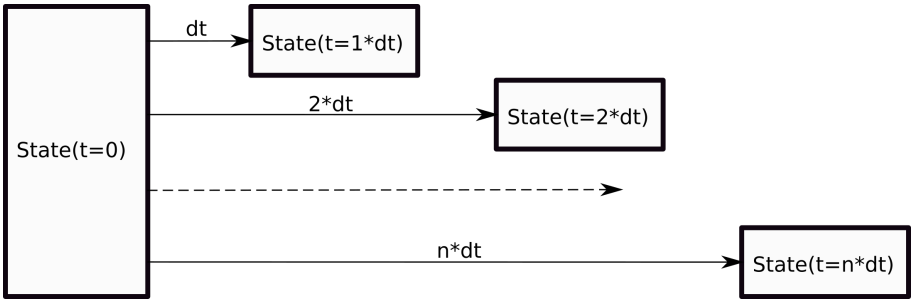

Figure 6. Time advance for a one-step simulation.

3 Application to microphysical parametrizations

The ICE microphysical scheme is divided ~~in~~ into three parts: a statistical adjustment to the saturation (to balance cloud water and ice with the vapor, according to the temperature), the core microphysical processes (collections, riming, vapor deposition, evaporation, ...) and the sedimentation. Each of these three parts ~~can~~ may contain sources of time-step dependency.

The adjustment to saturation modifies the temperature and hence modifies also the saturation point and then the cloud content. This feedback could be a source of time-step dependency but it was checked (not shown) that the saturation adjustment used reaches an equilibrium very quickly; the impact of a second iteration can ~~hardly be seen. However, the~~ barely be detected. The cloud ice fraction (~~which is~~ the ice content divided by the total -ice and liquid- content) ~~is function of~~ depends on the temperature (for ~~temperature~~ temperatures above 0 °C, the cloud is liquid, for ~~temperature~~ temperatures under -20 °C, the cloud is icy and the cloud ice fraction is linearly interpolated between these two points) and ~~, then,~~ a consumption of one of these two species in the core microphysical processes implies a consumption of the other ~~specie~~ species during the following saturation adjustment (to keep the cloud ice fraction consistent with the temperature). This mechanism leads to a time-step dependency.

In this section, two examples of the tool usage are shown. The first one deals with the time-step dependency due to the processes of the microphysical scheme (without adjustment and without sedimentation) ~~; and~~ in a 0D mode; the second one is a comparison of several sedimentation schemes, in a 1D mode.

3.1 Time-step dependency in the microphysical scheme

The final goal was to suppress the time-step dependency ~~on~~ of the simulations shown in Fig. 1. To achieve this result, a new set of simulations was performed using much smaller time steps (between 0.001 s and 1 s) ~~. The idea was~~ in order to look for a

convergence between the simulations when the time step decreases. In Fig. 7, the final values are ~~much~~ far more consistent but some oscillations are still visible on the time evolutions and values between 40 s and 60 s of integration are still very uncertain. It was necessary to use very small time steps in order to obtain numerical oscillations smaller than the physical variations.

The ~~purpose of the paper is not to give an extended review of all the modifications that were needed to suppress the time-step dependency. I will just list rapidly the most important ones: heat budgets must be computed when the feedback on temperature can stop the process. For example, when the temperature is positive, a specie cannot melt more than the quantity that implies a change in the temperature sign. The lack of heat budgets explain a large part of the time-step dependency observed with the small time steps; in the original version of ICE, the snow content rimed by cloud droplets (to produce graupel) was computed as a statistical adjustment: the process gave the mass of snow to convert into graupel, then this mass was divided by the time step. And, the mass of transformed snow did not take into account the quantity of cloud water involved. These two characteristics were at the origin of a time-step dependency. The process was modified using the **?** approach based on the comparison between the effective cloud droplets collection and the mass of water needed to transform low density snow into high density graupel; the graupel growth mode choice was updated. It is now more continuous and, hence, less time-step dependent. The graupel growth mode has an impact on the collection efficiency of icy species (snow and cloud ice)with the graupel and can lead to significant differences in the collection rates; the water shedding (cloud droplets becoming rain drops when collected and not frozen by the graupel) was activated only with negative temperature whereas the process must be active as long as the graupel exists (when the graupel has melt into rain, rain drops actually collect cloud droplets; the process must be continuous during the graupel melting); several modifications have been carried out on the processes involving the hail category as a prognostic field: the processes dealing with hail are now completely based on those dealing with the graupel category (to insure consistency even if this did not produce time-step dependency) and~~ simulations have been carried out several times activating and deactivating the different microphysical processes. To do this, the ~~conversion rate of graupel into hail is now computed from the wet growth rate of graupel and not from the total content of graupel (this was the main reason of the~~ processes have been called individually by the PPPY software (when they are written in separate subroutines) or activated through switches or, at worst, (un-)commented in the source code. This trial-and-error process makes it possible to identify the processes that led to the oscillations and to the time-step ~~dependency on the hail category).~~ dependency, and allowed the checking of each correction individually from the others.

~~These modifications try to suppress the time-step dependency present inside each of the microphysical processes. Another source of dependency lies in the interactions between these processes. One process must take into account that a given specie can be consumed or produced, in the same time, by another process. The modifications listed above were sufficient to suppress or, at least, limit~~ The purpose of this paper is not to detail the modifications that were needed to suppress the time-step dependency~~until time steps around 10 (not shown). To use time steps greater, some kind of splitting was needed to reduce the effective time step used in the microphysical scheme to allow interactions between processes~~, however the more important ones are listed in Appendix B.

~~The modified scheme allows two splitting methods: a classical time-splitting method that uses a fixed sub-time-step and a "mixing-ratio-splitting" method that computes, at each iteration, the sub-time-step to use in order to not have a single mixing~~

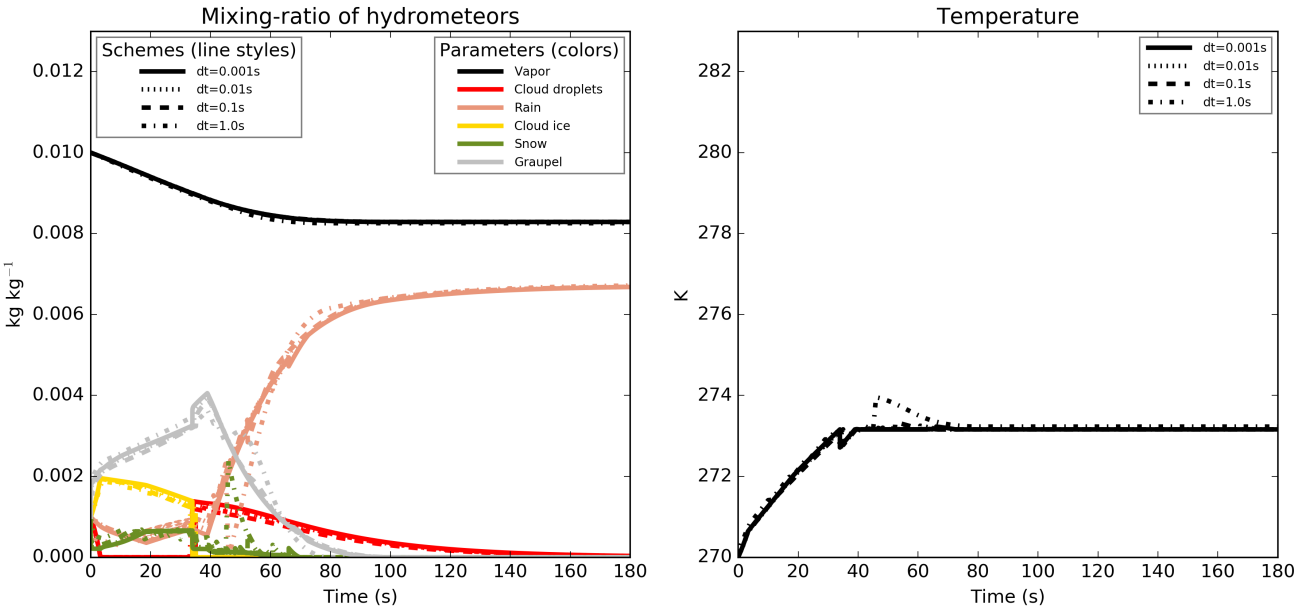

Figure 7. Same as Fig. 1 but using smaller time steps between 0.001 s and 1 s.

~~ratio change exceeding a given threshold. The second method has the advantage to adapt the number of iterations to the intensity of the microphysical processes. When little happens, only one iteration is performed; on the contrary, when the exchanges are intense, several iterations are performed.~~

~~With all these modifications, including a mixing-ratio-splitting, the~~

With the revised version of the microphysical scheme, the different simulations shown in Fig. 7 and Fig. 1 are now perfectly indistinguishable in Fig. 9 and Fig. 8 (the curves are superimposed at every common output ~~times~~time).

~~To produce these figures, the threshold used in the mixing-ratio-splitting (1.E-5) was small and could induce a substantial additional cost on a real simulation. This value was chosen to illustrate the scheme behavior but a value of 5.E-5 seems to be more acceptable (empirical value obtained in comparing cost and time-step dependency in a 2D simulation) for operations.~~

The 0D ~~tool was very useful to identify and correct~~ simulations were very useful for the identifying and correcting of the processes involved in the time-step dependency of the ICE microphysical scheme. It would ~~have been certainly~~ certainly have been possible to achieve the same result with another method ~~or tool but this method~~ but this one was convenient (0D simulations are very rapid and a ~~unique~~ single tool performs the simulations, compares the outputs and plots the comparison) and allowed ~~to completely isolate the~~ the complete isolation of the processes of interest from the other parts of the model (~~dynamic~~ dynamics with transport, other physical parametrizations, sedimentation and adjustment).

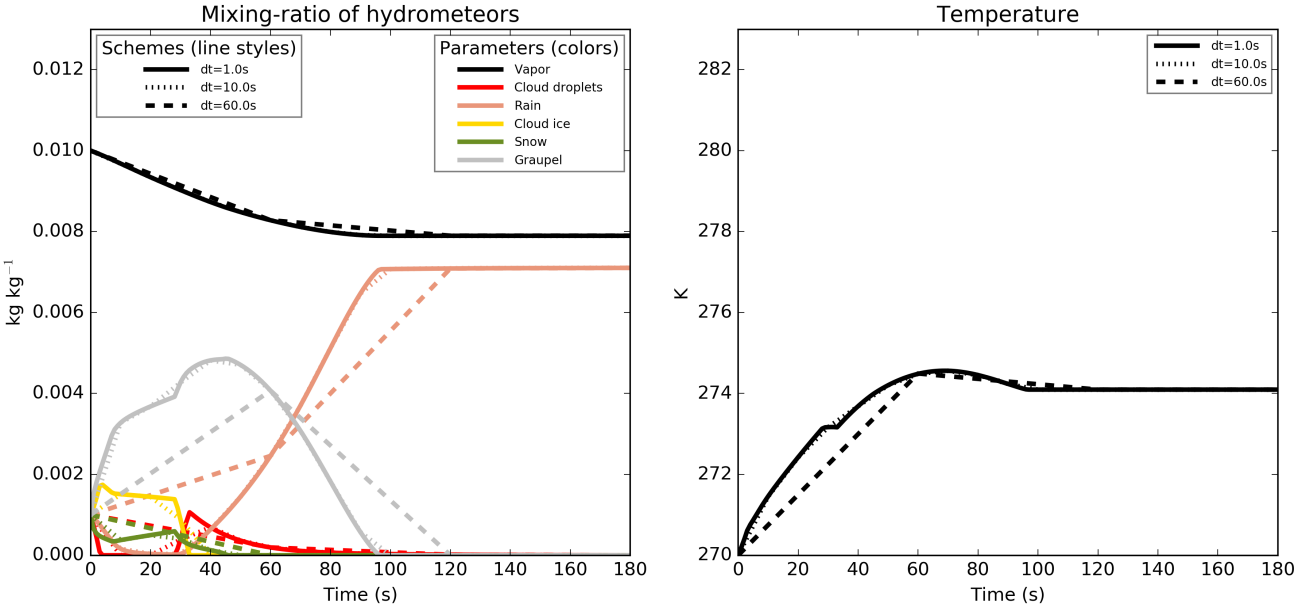

Figure 8. Same as Fig. 1 but using the new version of the microphysical scheme.

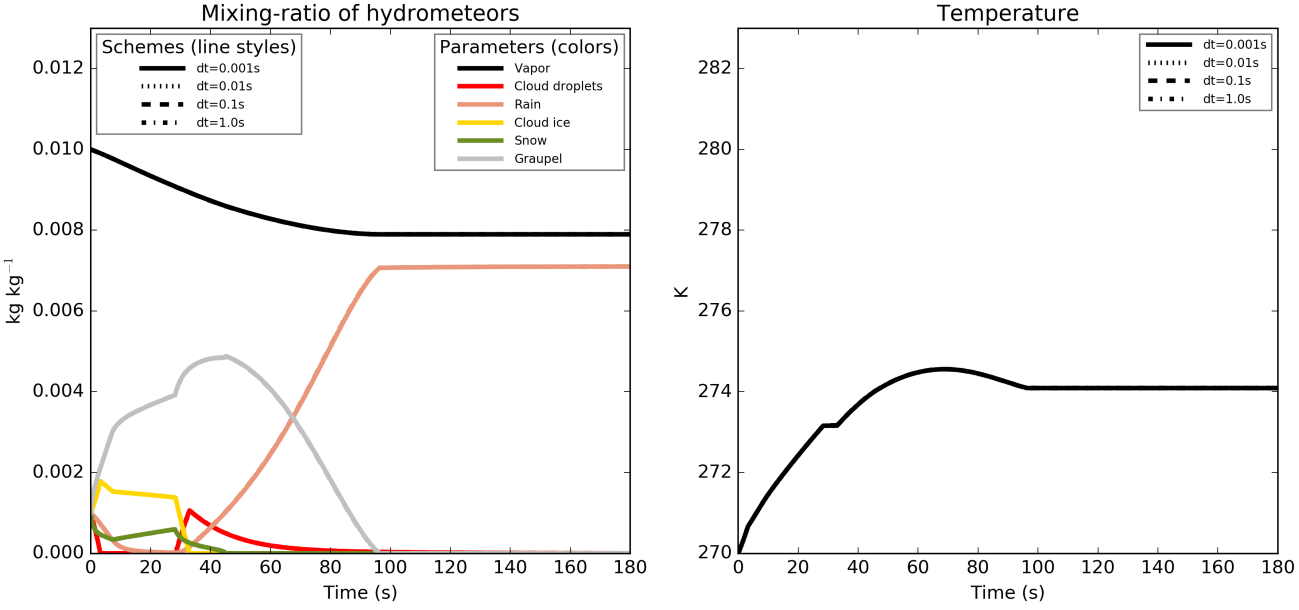

Figure 9. Same as Fig. 7 but using the new version of the microphysical scheme (all four curves are superimposed).

3.2 Sedimentation schemes

A similar time-step sensitivity test is done regarding the sedimentation scheme used in the model (all other parametrizations, including the microphysical processes, were turned off). Two schemes are available: the operational one (**?**) (BSB2010 hereafter), which is a statistical scheme and the other ~~one~~ which is an Eulerian scheme. In order to remain stable, the Eulerian scheme uses a time-splitting technique with an upstream differencing scheme. The internal time step[1] used for this computation is determined from the Courant-Friedrichs-Lewy (CFL) stability criterion based on a maximum fall velocity of 40 $\mathrm{m\,s^{-1}}$ if hail is allowed or 10 $\mathrm{m\,s^{-1}}$ otherwise (then, the same internal time step is used for all the species). ~~A part~~ Part of the difficulty for these schemes ~~to resolve accurately the sedimentation process~~ is the accurate resolution of the sedimentation process which comes from the hypothesis that the terminal fall speed is directly linked to the mean content (~~more the content is important, more the fall is rapid~~the more important the content, the more rapid the fall).

A vertical profile was initialized with a rain mixing-ratio of 0.1 $\mathrm{g\,kg^{-1}}$ in one cell at 1400 m above ground level; grid levels were 10 m thick. In order to build a reference solution independent from the time step, a box-Lagrangian scheme (based on **?**) is used with the one-step time-advance method (see Sect. 2.2 for a description of the time-advance method). Because the model schemes use a particle size distribution, the first reference simulation we can build is ~~by dividing~~ through the dividing of the total content into bins and ~~apply the~~ the application of the sedimentation on each bin (as for **?**). The reference time evolution is then shown in the upper panel of Fig. 10 (this reference is computed using only 25 bins to allow the reader to identify the trajectories of each bin; when more bins are used the time evolution gets smoother). The bigger drops reach the ground after around 200 s whereas the smaller ones ~~haven't~~ have yet to reach the ground after 1500 s. After 400 s of simulation (dashed vertical line on the plot), one third of the rain is expected to be on the ground and the remaining part spread in the column. ~~However, the model schemes~~ The model schemes however compute the mass-weighted bulk terminal fall velocity and apply this velocity to the entire content. With this hypothesis, the awaited time evolution is ~~given~~ provided by the lower panel of Fig. 10. In this case, all the rain content is expected to follow the same trajectory and to be near the ground after 400 s of simulation. The first reference simulation has a more ~~physical~~ realistic behavior and is able to reproduce the size-sorting effect but this result cannot be reached by the one-moment schemes used in this study. ~~Then, the~~ The bulk simulation is taken as the reference simulation with which to compare the model schemes over 400 s long simulations. According to the initial mixing-ratio, to the parameters used in the sedimentation scheme and to the hypothesis of a mass-weighted bulk terminal fall velocity, the rain is expected to fall with a 3.3 $\mathrm{m\,s^{-1}}$ speed, leading to a fall of 1320 m during a 400 s period. It should be noted that with a grid spacing of 10 m, the unit value for the CFL number is reached with a time step of around 3 s.

The top panel of Fig. 11 shows the resulting profile for different time steps (for time steps between 0.1 s to 60 s in steps of 0.1 s) using the BSB2010 scheme (this is ~~a~~ not a time evolution, all profiles are the result of a 400 s long integration). The time-step dependency is ~~obvious~~ evident and it must be noted that for time steps longer than 30 s a part of the water has reached the ground. ~~Longer is the time step, more this part is important. It reaches around 11% of the total water for the 60 time step.~~

[1] the term "internal time step" is reserved, in this paper, for the time step used internally in the scheme to perform the time splitting. It is different from the (external) time step used for the scheme integration

~~For time steps between 0.1 and 3 , longer is~~ The longer the time step ~~, more the bottom of the precipitation envelope gets closer to the ground. This behavior iscertainly common to all the schemes that do not diagnose or predict the vertical heterogeneity. Indeed, with a small time step, only a tiny fraction of the grid content is able to fall in the lower grid level during one integration step. At the next time step, it is hypothesized that the remaining water is uniformly distributed on the vertical in the cell; this implies 1) a computed fall speed slightly smaller because the mean content is weaker, and 2) a longer time needed to make fall the grid content because the part of this content which get closer to the bottom of the grid had been redistributed on the whole cell (upward diffusion). Smaller isthe CFL number, more the fall speed is reduced. A small CFL number must also lead to a diffusion downward because of the same redistribution mechanism applied to the weak content that just crossed the level boundary and is distributed over the whole grid at the next time step. But, because the content is weak, the fall speed is small and these cells are quite easily caught up by the surrounding cell with a greater content (thus falling with a a greater speed).~~

~~With a 3~~is, the greater this part is. For the 60 s time step, ~~the algorithm of the scheme induces a fall without diffusion which leads to the correct result (assuming that the fall speed is uniform for all the drops which is an hypothesis of the studied schemes because the mass-weighted terminal fall velocity is affected to all the drops). The P1 (defined in BSB2010 as the proportion of "rain which leaves the layer during the time step") and P2 ("Among all the drops that enter the layer under consideration (from above) during the time step, P2 is the proportion of those which also leave the layer (by the bottom) during the time step.") terms of BSB2010 evaluate to, respectively, one and zero; the rain crosses exactly one grid cell by time step.~~ 11% of the total water has reached the ground.

~~With slightly larger time steps, P1 is still one and P2 remains small. With P2 small, diffusion downward remains small, the peak content is still on the top level of the precipitation envelope. During the simulation, the level with the peak content falls of one level, catches up the content which has fallen quicker (during the preceding time step) and so, the diffusion stays weak during the rain fall. And, by consequences, P1 remains equal to one for the highest level with rain. But because the fall of the peak content occurs at a rate of one grid level by time step, the fall speed gets smaller when the time step increases because less iterations are needed to reach the 400 integration duration.~~

~~With much larger time steps, P1 is still one at the beginning and P2 is larger. With a greater value of P2, at the first time step, a significant rain content crosses several levels. This induces that the top level of the precipitation envelope may not have a sufficient content, at next time step, to insure a CFL number larger to one. This had two consequences: the top level of the envelope falls slower and the rain contained in the top level does not catch up the rain already fallen lower. This leads to an upward diffusion and to a reduction of the mean velocity at all levels because rain content is diluted on the vertical.~~

The BSB2010 scheme behaves differently regarding the CFL number ~~. The artifact~~ with varying upward and downward diffusions and a singularity point for the unit value. The artifacts seen in the figure ~~illustrates~~ illustrate the shortcoming of the scheme but ~~cannot be so important~~ can not be as large in a true model simulation. In a real case with advection, non-constant grid spacing and microphysical sources and sinks, ~~it is not possible to keep the CFL number constant and equal to one during the entire fall.~~ the CFL value is not constant and each column of the model is a mixture of these different behaviors.

For the Eulerian scheme (lower panel of Fig. 11), the time-splitting technique used with a very small internal time step of 0.25 s (value obtained considering a maximum fall speed of 40 ms^{-1}) leads to ~~do~~ approximately the same computation whatever ~~is~~ the time step of the simulation ~~,~~ greater than this value. ~~Therefore, there is~~ There is therefore no time-step dependency in the result. ~~However, the~~ The very small internal time step induces however, as for the BSB2010 scheme with small values of the CFL number, an upward diffusion and a reduced fall speed. ~~Moreover, the Eulerian scheme~~The Eulerian scheme, in addition, has an increased numerical cost due to the very small internal time step used.

~~None of both~~ Neither of these two schemes is correct; both diffuse the precipitation and have ~~a too small~~ too small a fall speed. ~~Moreover~~Additionally, the BSB2010 scheme exhibits a time-step dependency whereas the Eulerian scheme is costly. ~~The common problems to both schemes are the result of two mechanisms. Firstly, for small CFL values the schemes are penalized by the fact that the water content is supposed to be, at each time step, uniformly spread on the cell; if the time step is not long enough to put all the water in the level under, at the next time step, the remaining content is artificially put higher. Secondly, because we consider one-moment microphysics, when a content is split across two cells (by a first iteration of the algorithm), at the next time step the mean content over each of both cells is weaker and the mean fall speed is reduced. With a two-moment scheme, the content which had fallen in the lower level would be associated to bigger drops and would keep a greater fall speed than with the one-moment scheme.~~

~~Because none of the scheme can reproduce the right fall speed, a compromise could~~ A compromise can be found by optimizing the Eulerian scheme in order to obtain a scheme without time-step dependency and with a reasonable cost. ~~The idea is to not use a fixed value for the internal time step.~~ In the optimized version, at each iteration in the time splitting, the ~~maximum~~ time of integration is computed from the maximum CFL number on each column and for each ~~specie. The integration is done for one internal time step and then the following iteration is done the same way until the advance reaches the whole timestep.~~ species instead of using a constant (in space and time) value.

The result is shown in Fig. 12 for different values of maximum CFL number allowed. ~~The top panel is obtained with~~ With a maximum CFL number of one ~~. The~~ (top panel), the time-step dependency is quite large~~because, for integer CFL numbers, the fall occurs without diffusion. For CFL numbers slightly superior to an integer value, the first internal iterations are done with a CFL number of one (without diffusion) and the last internal iteration produces a great content surrounding a cell with a weak content that could be easily caught up during the next time step even if mean speed was reduced. For CFL numbers slightly inferior to an integer value , the last internal iteration produces a weak content surrounding a great content which lead to a diffusion and a reduced fall speed.~~

~~A maximum CFL value of one cannot be obtained everywhere and every time in a true simulation and, to reduce the risk of instability, a maximum CFL number allowed can be set. The~~ . With a maximum CFL number value of 0.1 (panel in the middle~~is obtained with a maximum CFL number value of 0.1. The~~), the resulting figure is almost identical to the ~~one obtained in~~ Fig. 11~~for which the small internal time step induced a small CFL number~~. The lower panel is for a larger maximum CFL number (0.8). For small time steps inducing a CFL number inferior to this maximum value, a time-step dependency can be seen ~~. For~~ but for large time-step values, the computation leads to the same results whatever ~~is~~ the time step ~~because, in this case, the internal time step used in the time-splitting algorithm isnot limited by the external time step.~~ is. In contrast with the original

version~~or the modified version with a small value for the maximum CFL number, the simulations are~~ , this last case is less diffusive and ~~are~~ is able to reproduce a peak value (in the bottom of the precipitation envelope). The resulting fall speed is still reduced (after the 400 s integration duration, all the water content should be near the ground according to the hypothesis~~done~~) but slightly better than those produced by the other versions of the Eulerian scheme.

In order to test further the impact of the algorithm on the sedimentation results, the box-Lagrangian scheme (used previously to build the reference results) is used in a simulation mode (using the step-by-step time-advance method). ~~Top~~ The top panel of Fig. 13 shows the resulting profiles after the 400 s long simulation using the bulk approach. The result is noisy, time-step dependent and there is still no rain on the ground. The noise could ~~be certainly reduce~~ certainly be reduced by imposing a speed continuity between the different layers of the model (following the idea of **?** for example)~~but this simulation demonstrates that among the different numerical schemes used in this paper (Eulerian, statistical and Lagrangian) none are able to reproduce the reference simulation. This is mainly due to the fact that after each time-step, if CFL is not one, the rain content is split over two grid cells and, then, the content is reduced at the next time step and this induces a diffusion on the vertical and a decreasing fall speed. We could try to reduce this feature by using an hybrid scheme (as used in ?) in conjunction with the~~ . The box-Lagrangian ~~scheme~~scheme can also be used in an hybrid mode (as used in **?**). At each time step, the content is divided ~~in bins, then,~~ into bins, each bin falls using the box-Lagrangian scheme ~~. At~~ and, at the end of the time step, the total content at each model layer is computed and used by the following iteration. The simulations are done using 500 bins (lower panel of Fig. 13). This scheme allows ~~to make fall~~ the bigger drops ~~quicker~~to fall more quickly. While this approach reduces the noise, some ~~time step~~ time-step dependency remains and the scheme is still not able to make ~~fall the precipitation quick~~ the precipitation fall quickly enough to reproduce the reference simulation (~~and this scheme~~ this scheme also does not fulfill the requirement of a mass-weighted bulk terminal fall velocity used to build the reference). ~~Nevertheless, it~~ It should be noted nevertheless that this result is quite similar to the result obtained with the new version of the Eulerian scheme (lower panel of Fig. 12). ~~It seems that no algorithm choice can reproduce the reference simulation due to the diffusion appearing during the fall that leads to a reduced speed. It seems difficult to find a solution to improve the simulations without prognosing an other moment of the distribution (this other moment must be advected and not only diagnosed for the sedimentation process). ? also came to the same conclusion and gave hints on the choice of the best moments to use and on the corrections that can be applied to a two-moment scheme to reduce the sedimentation error.~~

~~On the~~

On one hand, the algorithm choice ~~could~~ may well be important because ~~this choice~~ the selection modifies the effective fall speed and the relative position of precipitation on ground with respect to the cloud which generated it, through the horizontal advection. ~~But, on~~ On the other hand, none of the schemes tested in this study ~~is~~ are able to reproduce the reference simulation ~~. Moreover~~due to the diffusion appearing during the fall which leads in turn to a reduced speed. Additionally, with the mixing induced by the dynamic and the turbulence, and with the interaction with the other microphysical processes, this choice has little impact on the resulting simulation of a real 3D case (not shown). It is believed that no scheme could perform drastically better with the one-moment hypothesis.

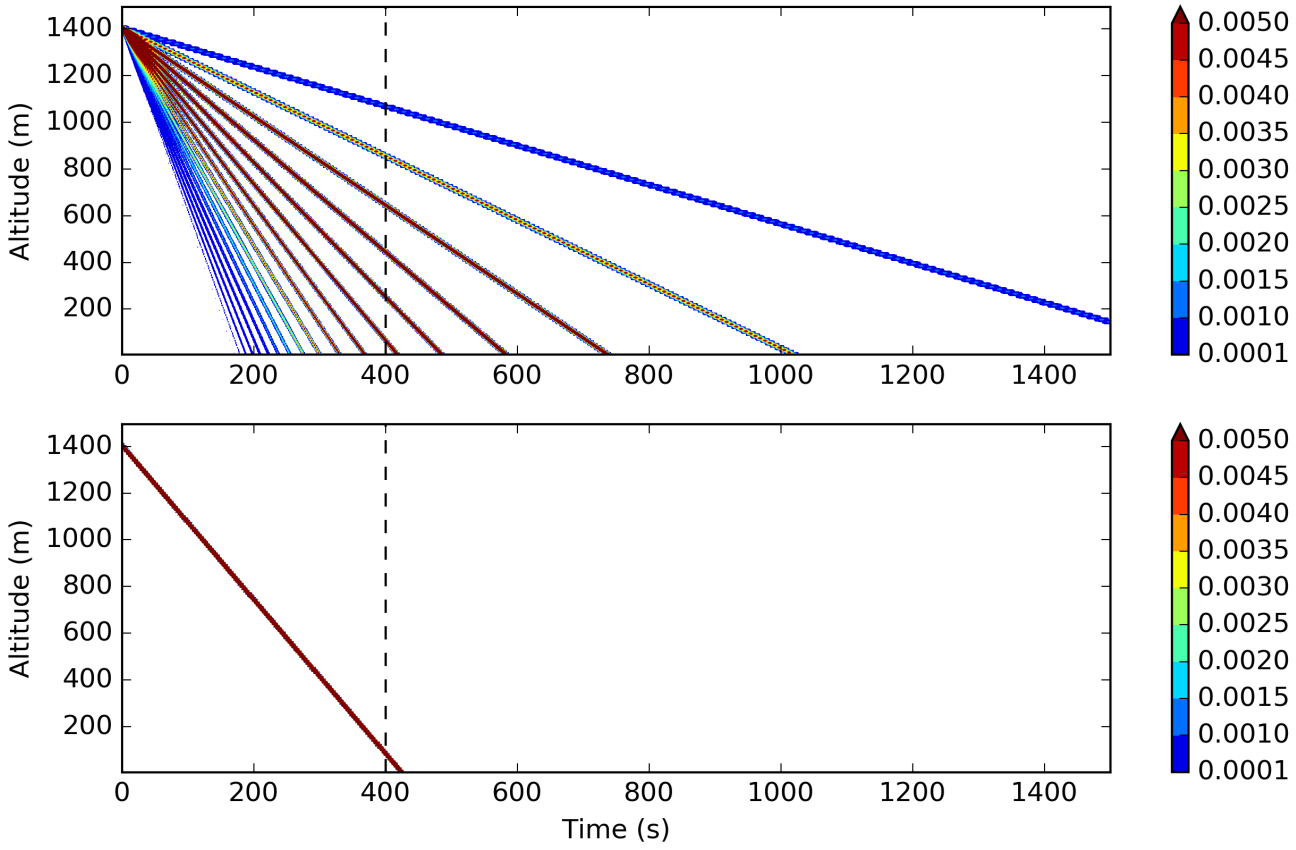

Figure 10. Time evolution of the vertical profile of rain mixing-ratio (the color scale represents the mixing-ratio in $\mathrm{g\,kg^{-1}}$) for a bin box-Lagrangian scheme (upper panel) and a bulk box-Lagrangian scheme (lower panel).

~~In the future, this~~ As from the conclusions of **?**, one can expect better behavior from two-moment schemes. This study could be extended in the future to the two-moment sedimentation scheme used by LIMA.

This section illustrated how PPPY can be used to compare and exhibit the main behaviors of different 1D sedimentation schemes written in Fortran and Python using different time-advance methods.

4 Conclusions

In this paper, a new ~~tool~~ software designed to allow the comparison of Physical Parametrizations with PYthon (PPPY) independently of all other model components was described technically and functionally. Its ability to use ~~FORTRAN-compiled~~ Fortran-compiled library from different models, as well as ~~python based parametrizations~~Python based parametrizations, has been shown and used through two examples. ~~The tool~~ It has been successfully used (in a 0D mode) to identify the sources of the time-step dependency which was present in the microphysical scheme in use in the AROME and Meso-NH models. Some

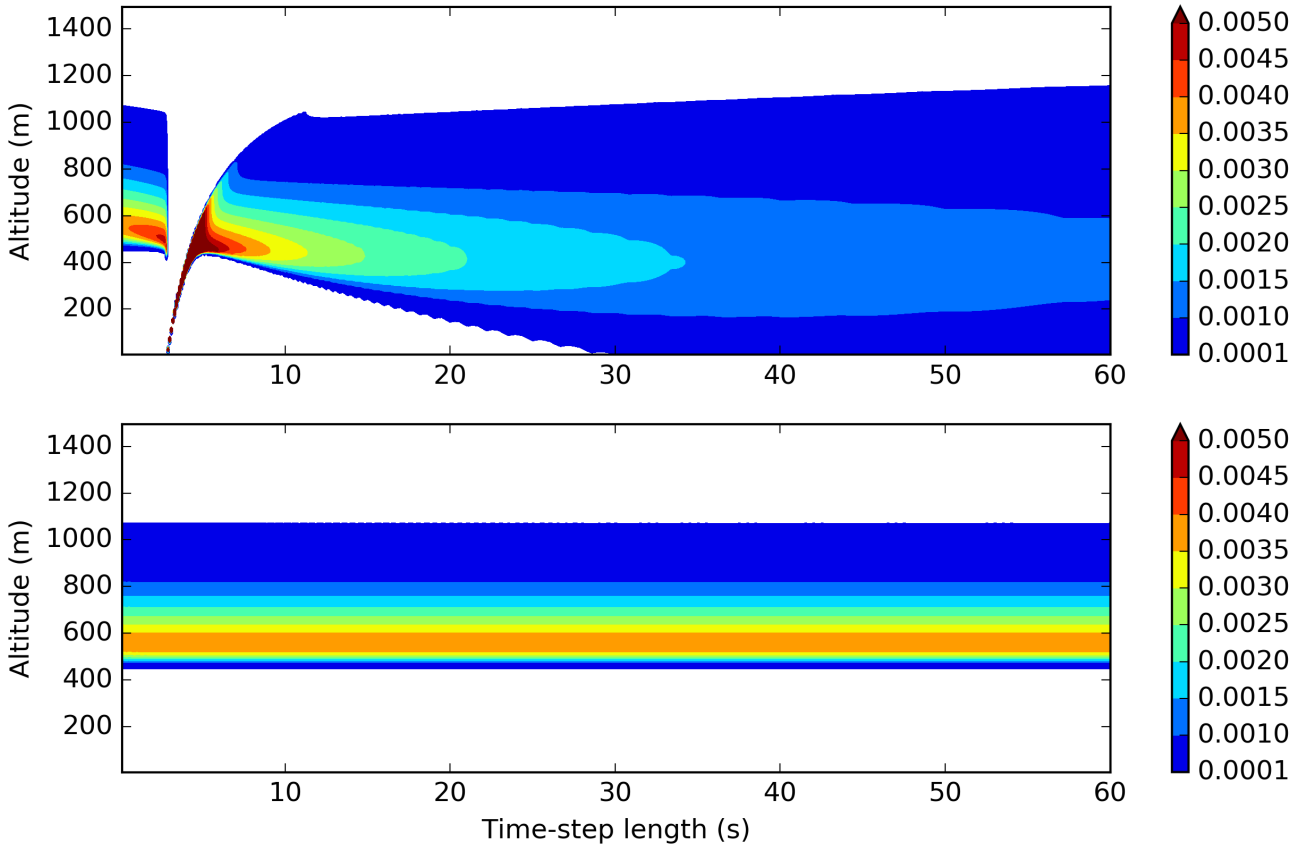

Figure 11. Vertical profile of the rain mixing-ratio (the color scale represents the mixing-ratio in $\mathrm{g\,kg^{-1}}$) after a 400 s long integration for different time steps (the time step varies between 0.1 s and 60 s with a 0.1 s step leading to 600 different simulations, abscissa) for the BSB2010 scheme (upper panel) and the Eulerian scheme as available in the operational source code (lower panel).

solutions have been proposed to correct the scheme and have been tested with ~~the tool. Then, the~~ it. The sedimentation schemes have then been plugged and compared (in a 1D mode) to a reference box-Lagrangian scheme. These two examples have shown that it would be beneficial to use this kind of tool systematically when developing a parametrization in order to perform simple tests providing a first validation step (mass conservation, time-step dependency, absence of oscillations), before ~~going through~~ undergoing more complex validation stages (1D model, full simulations).

In addition to the ICE scheme, the LIMA microphysical scheme and some of the WRF microphysical schemes have been plugged~~into the tool. Then, it~~ . It could now be used to compare microphysical schemes originally hosted by different models (AROME, Meso-NH and WRF). In order to compare ~~them~~these, it will be necessary to work on the initialization of the different schemes to ~~insure~~ ensure that the results can ~~actually~~ in fact be compared. ~~Moreover~~In addition, it will be necessary to define a suitable time step for each scheme ~~to compare~~ for comparisons because of the time-step dependency present in the different

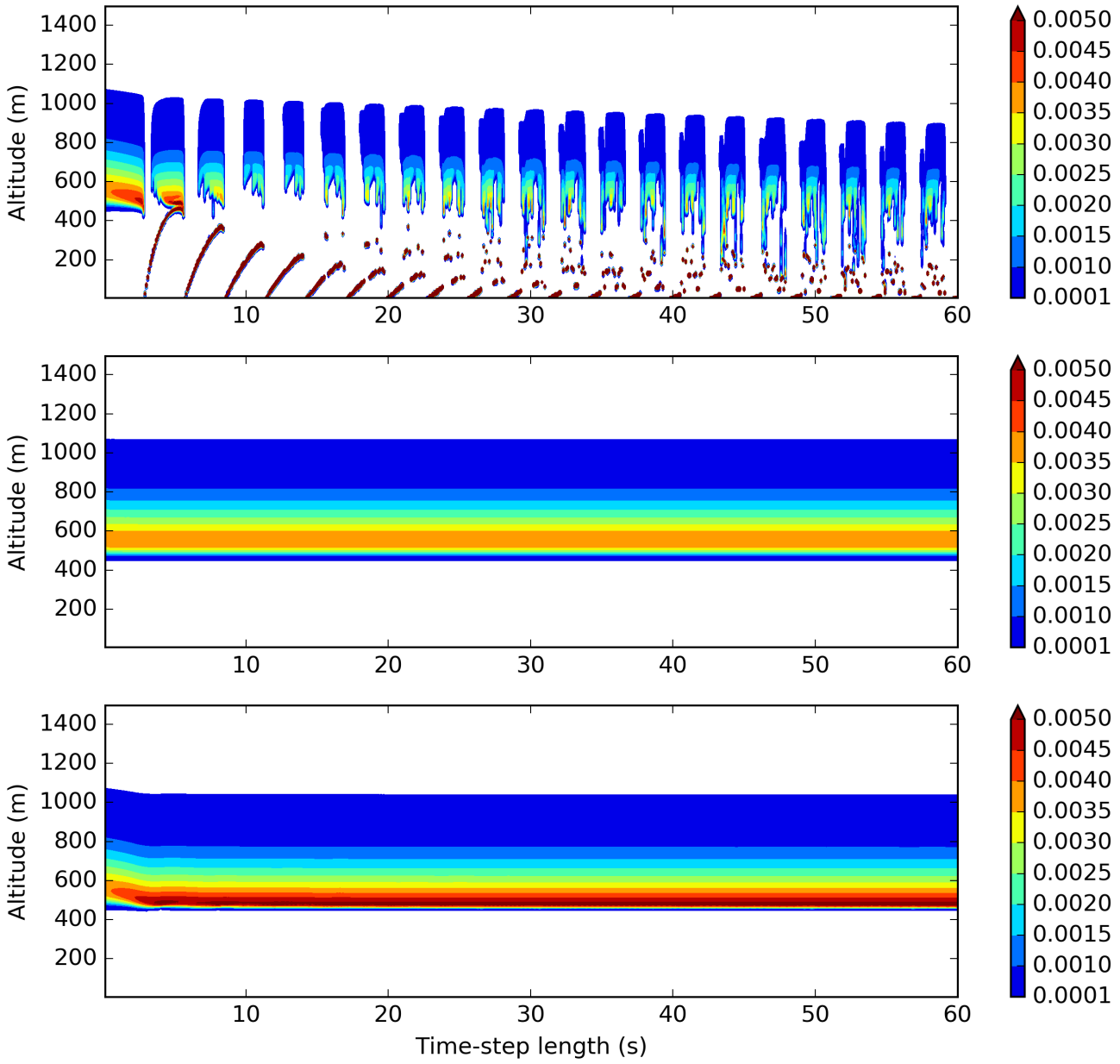

Figure 12. Same as Fig. 11 but for the modified version of the Eulerian scheme using different maximum values for the CFL number (1.0, 0.1 and 0.8 from top to bottom). The color scale represents the mixing-ratio in $\mathrm{g\,kg^{-1}}$.

schemes (a way to suppress it would be to select, for each scheme, the greater time step which allows the convergence towards the solution given by smaller time steps).

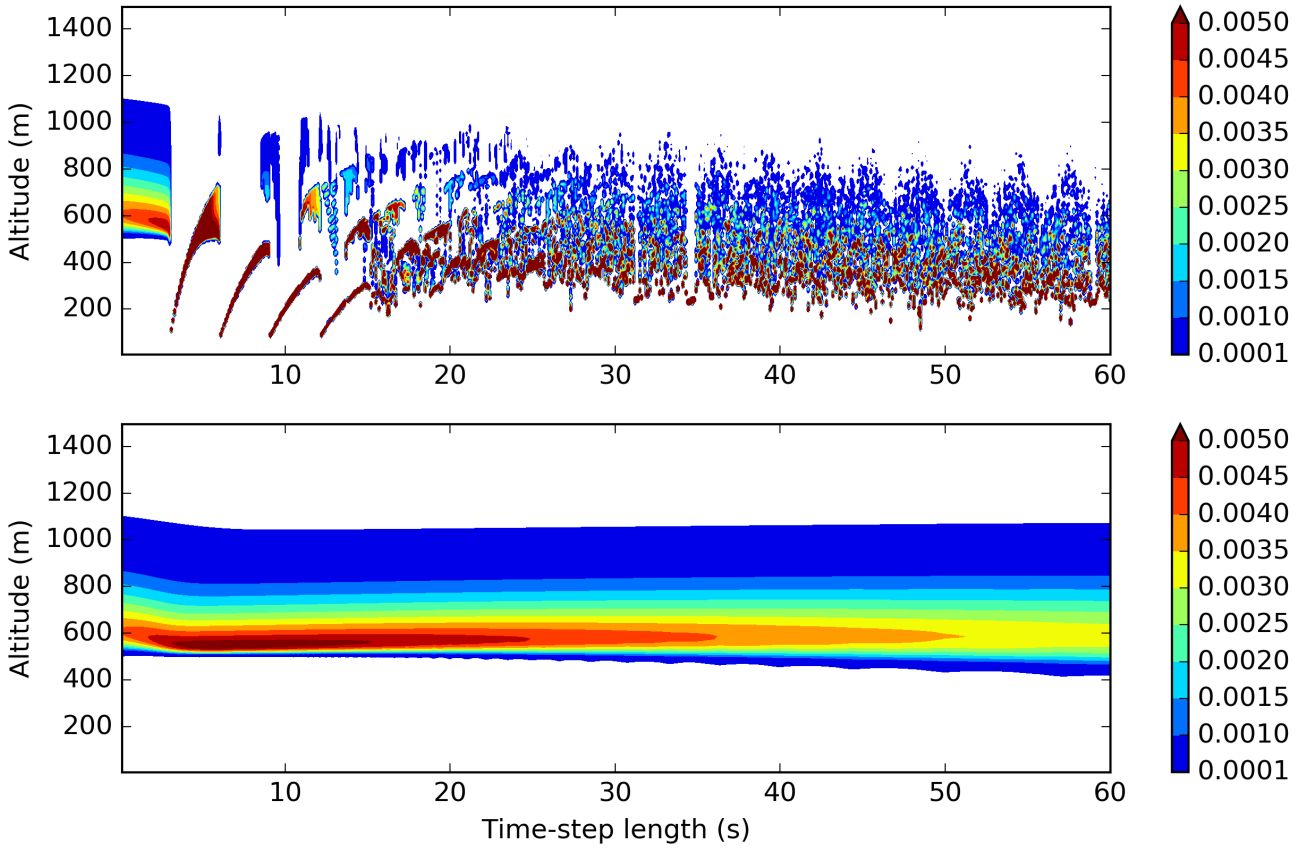

Figure 13. Same as Fig. 11 but for the box-Lagrangian scheme using the bulk approach (upper panel) and an hybrid approach (lower panel). The color scale represents the mixing-ratio in g kg^{-1}.

~~The tool~~ PPPY is not limited to microphysical schemes and, in the future, it could also be used to compare other parametrizations ~~like~~ such as mass flux or turbulence schemes.

Code availability. PPPY is freely available under CeCILL-C license agreement (a French equivalent to the L-GPL license; http://www.cecill.info/licences/C_V1-en.txt). PPPY v1.1 can be downloaded at https://doi.org/10.5281/zenodo.3490380.

Appendix A: Simple example

Several examples of PPPY usage are provided with the software. Among them, a special example is intended to show how the different Python objects interact with each other and with the Fortran code; this is the test example which can be found in the `examples/test` directory of PPPY. This example is used in this appendix to illustrate the different steps needed to perform a parametrization comparison.

A1 Compilation

Let's assume that the following code is put inside a file named `param.F90` and represents a model parametrization that we want to use with PPPY:

```
SUBROUTINE PARAM1(X, Y)
IMPLICIT NONE
REAL(KIND=8), INTENT(IN), DIMENSION(:, :) :: X
REAL(KIND=8), INTENT(OUT), DIMENSION(:, :) :: Y
Y=X+1
END SUBROUTINE PARAM1
```

It is suggested that the compilation procedure of the model be employed. One must therefore ensure that the normal compilation of the model builds a position-independent code, suitable for dynamic linking (`-fPIC` option). If not, the Makefile or the compilation script of the model must be updated to include such an option.

To use ctypesForFortran, a wrapper must be written to hide certain characteristics. The exposed dummy arguments:

- must not be of assumed shape or assumed rank (including string length) type;
- must not be optional;
- of Boolean type must be one-byte long.

In the test example, a wrapper (written in a file named `param_py.F90`) is needed to hide the assumed shape characteristic:

```
SUBROUTINE PARAM1_PY(X, Y, I1, I2)
IMPLICIT NONE
INTERFACE
   SUBROUTINE PARAM1(X, Y)
      REAL(KIND=8), DIMENSION(:,:), INTENT(IN) :: X
      REAL(KIND=8), DIMENSION(:,:), INTENT(OUT) :: Y
   END SUBROUTINE PARAM1
```

```
END INTERFACE
INTEGER(KIND=8), INTENT(IN) :: I1, I2
REAL(KIND=8), INTENT(IN), DIMENSION(I1, I2) :: X
REAL(KIND=8), INTENT(OUT), DIMENSION(I1, I2) :: Y
CALL PARAM1(X, Y)
END SUBROUTINE PARAM1_PY
```

When it is possible, it is suggested that this wrapper be included in the source code of the model in order to benefit from the Makefile or compilation script. If this proves impossible, the wrapper must be compiled outside the model environment but one must use the same compilation options for the wrapper as those used for the model.

Often, a parametrization must be initialized by calling a specific subroutine (in particular to set up constant values). In this example, this step is achieved by calling the following subroutine (which is also included in the `param_py.F90` file and does not need to be wrapped):

```
SUBROUTINE INIT(ICONF)
IMPLICIT NONE
INTEGER(KIND=8), INTENT(IN) :: ICONF
END SUBROUTINE INIT
```

In addition, the example includes a `PARAM2` subroutine (copy of `PARAM1` except that Y is $X + .9$) and an associated `PARAM2_PY` subroutine.

It is then supposed that a normal model compilation produces the compiled version of all these subroutines. In the example, the compilation is obtained (using gfortran) by the following command lines: `gfortran -c -fPIC param.F90` and `gfortran -c -fPIC param_py.F90`.

The last step of the compilation process is to build a shared library with `PARAM1_PY`, `PARAM2_PY` and `INIT` as entry points. This can be done through adding a compilation target in the model Makefile or compilation script, or by performing a manual build. In the example, the command line `gfortran -shared -g -o param.so param_py.o param.o` produces the `param.so` file.

A2 The parametrization from the Python point of view

To use the previously build shared library from Python using the ctypesForFortran module, the following code is needed:

```
import ctypesForFortran
IN = ctypesForFortran.IN
OUT = ctypesForFortran.OUT
INOUT = ctypesForFortran.INOUT
```

```
ctypesFF, handle = ctypesForFortran.ctypesForFortranFactory('./param.so')

@ctypesFF()
def init(ICONF): #Name of the function must be the name of the actual fortran function
    "init function"
    return ([ICONF],
            [(numpy.int64, None, IN), #INTEGER, INTENT (IN)
            ],
            None)

@ctypesFF()
def param1_py(x):
    "Function that actually call the parameterisation"
    return ([x, x.shape[0], x.shape[1]],
            [(numpy.float64, x.shape, IN),
             (numpy.float64, x.shape, OUT),
             (numpy.int64, None, IN),
             (numpy.int64, None, IN)
             ], None)
```

In this example, one of the Fortran subroutines is named `INIT` (case insensitive). By default, it is supposed that the compiled object's name is the Fortran subroutine name (lowercase) with a trailing underscore (`init_` in this example)). If this is not the case (because of a different compiler behaviour or a Fortran module use), a different prefix and/or suffix can be set in argument of the `ctypesFF` function of the example to obtain a decorator able to find and call the Fortran code.

The Python function must return three elements:

- the list of the values expected in input by the Fortran subroutine;
- the list of the dummy arguments of the Fortran subroutine;
- the type of the returned value for a Fortran function (None for a subroutine).

Each dummy argument is described by a tuple: type of the argument expressed as a numpy type, shape (or None for a scalar) of the argument and input/output status. More examples are available inside the ctypesForFortran module source code. It should be noted that this part can be replaced by the use of other interfacing method such as f2py.

To use the parametrization with PPPY, a Python class must be written (see the `pppy_param1.py` file of the example for a complete implementation) by inheritance: `class pppy_param1(pppy.PPPY):`. The class contains a `__call__` method which calls the different methods in this order: setup, build_init_state, execute (which is called in a loop) and lastly finalize.

The methods that can or must be implemented to represent a parametrization are described below (only the execute method is mandatory):

- __init__: This method can be implemented to deal with the possible parametrization options.
- setup: This method does the initialization part that cannot be done earlier (in the __init__ method) or needs to be done again before each of the simulations. In the various provided examples, this is the place where the shared library is opened, where signatures of Fortran routines are defined and where the initialization of the Fortran modules are performed.
- finalize: This method can be useful for cleaning the memory or disk after running a simulation.
- build_init_state: When performing a comparison, each of the parametrizations is called with the same initial state. This method is the place at which to add state or diagnostic variables specific to the parametrization.
- execute: This method calls the parametrization to perform a time advance.

More details about these methods are given in the PPPY documentation.

A3 Comparison

The comparison is performed by the `comp_test.py` file.

Firstly, the parametrizations must be defined by choosing the time step (dt argument), the time-advance method (method), the names (the name argument is used for the plot labels and the tag one for building file names) and the possible options (solib and iconf here). For the example described here, this is done by the following lines:

```
param_1 = pppy_param1(dt=60., #time step to use with this parametrization
                      method='step-by-step', #like a true simulation
                      name="Param #1", #name to use for plots
                      tag="param_1", #tag to use for file names
                      solib=solib, #1st pppy_param1 option: shared lib file name
                      iconf=iconf) #2nd pppy_param1 option: configuration
```

Several parametrizations can be defined by changing the time step, the time-advance method or the options, using the same source code or not.

The comparison is defined and the simulations are performed by:

```
comp = PPPYComp(schemes=[param_1, param_2], #List of parametrizations to compare
                output_dir=output_dir, #directory to store the results (hdf5 files)
                duration=180., #duration of simulation
                init_state=dict(x=numpy.array([0.])), #initial state
```

```
                name="First test", #name to use in plots
                tag="firstTest") #tag to use for file names
comp.run()
```

A simple plot is obtained with:

```
plot = ('evol', #time evolution
        dict(var_names=['x'])) #of variable x
fig, plots = comp.plot_multi((1, 1), #only one plot
                             [plot])
plt.show()
```

Appendix B: Main modifications needed to suppress the time-step dependency in the ICE scheme

The most important modifications that were needed to suppress the time-step dependency in the ICE scheme are listed above:

- heat budgets must be computed when the feedback on temperature can stop the process. For example, when the temperature is positive, a species cannot melt more than the quantity that implies a change in the temperature sign. The lack of heat budgets explain a large part of the time-step dependency observed with the simulations using the small time steps (shown in Fig. 7);
- in the original version of ICE, the snow content rimed by cloud droplets (to produce graupel) was computed as an adjustment: the process provided the mass of snow to convert into graupel and this mass was then divided by the time step. The mass of transformed snow did not take into account the quantity of cloud water involved. These two characteristics were at the origin of a time-step dependency. The process was modified using the **?** approach based on the comparison between the effective cloud droplets collection and the mass of water needed to transform low density snow into high density graupel;
- the graupel growths mainly by collecting other species. When this collection implies liquid species (rain and/or cloud), there are two possibilities (called growth mode) depending on a heat balance: the graupel is able to freeze the entire collected liquid (dry mode) or a thin liquid film appears at the graupel surface (wet mode). In the original version of the ICE scheme there was confusion between the maximum content of liquid water than can be frozen (which must be used in the heat balance) and the content of liquid and ice water that can be collected in wet growth mode (which must be used to compute the graupel tendency). The correction made the mode choice more continuous. And because, ultimately, the graupel growth mode has an impact on the collection efficiency of icy species (snow and cloud ice) with the graupel, this choice can lead to significant differences in the collection rates. Hence, the scheme including the correction is less time-step dependent;

- the water shedding (cloud droplets becoming rain drops when collected and not frozen by the graupel) was activated only with negative temperature whereas the process must be active as long as the graupel exists (when the graupel has melted into rain, rain drops actually collect cloud droplets; the process must be continuous during graupel melting);
- several modifications have been carried out on the processes involving the hail category as a prognostic field: the processes dealing with hail are now completely symmetric with those dealing with the graupel category (to ensure consistency even if this did not produce time-step dependency). A conversion fraction is computed from the heat balance used to choose the graupel growth mode. In the original version of the scheme, this fraction was applied on the total content of graupel; this induced a conversion tendency directly linked to the number of times the rate is applied (hence to the time step for a given simulation length). On the contrary, in the new version, the conversion fraction is applied on the wet growth rate, this way, no time-step dependency is produced. This was the main reason for the time-step dependency on the hail category.

The modifications listed above aim at suppressing the time-step dependency present inside each of the microphysical processes. These modifications were sufficient to suppress or, at least, limit the dependency until time steps around 10 s (not shown). For greater time steps, each process must take into account that a given species can be consumed or produced, at the same time, by another process and that, therefore, this affects its efficiency. To address this issue, some kind of splitting was needed to reduce the effective time step used in the microphysical scheme.

The modified scheme allows two splitting methods: a classical time-splitting method that uses a fixed sub-time-step and a "mixing-ratio-splitting" method that computes, at each iteration, the sub-time-step to use in order not to have a single mixing ratio change exceeding a given threshold. The second method has the advantage of adapting the number of iterations to the intensity of the microphysical processes. When little happens, only one iteration is performed; on the contrary, when the exchanges are intense, several iterations are performed.

To produce Fig. 9 and Fig. 8, the threshold used in the mixing-ratio-splitting (0.01 $\mathrm{g\,kg^{-1}}$) was small and would induce a substantial additional cost on a real simulation. This value was chosen to illustrate the scheme behavior but a value of 0.05 $\mathrm{g\,kg^{-1}}$ seems to be more acceptable (empirical value obtained in comparing cost and time-step dependency in a 2D simulation) for operations. The iterations needed for the 0.05 $\mathrm{g\,kg^{-1}}$ threshold induce a cost increase of about 5%, and for the 0.01 $\mathrm{g\,kg^{-1}}$ threshold an additional 20% can be expected (this last figure is an estimation because no sensitivity test, regarding this threshold, has been performed on the whole domain used for operations).

Competing interests. The author declares that he has no conflict of interest.

Acknowledgements. I ~~which~~ wish to thank B. Vié for having plugged the LIMA scheme into ~~the tool~~PPPY, and the trainee team which ~~work~~ worked on the WRF plugging: A. Riandet, L. Richecoeur and M.-L. Roussel. I also thank C. Lac and B. Vié for their helpful comments on the manuscript.