This manuscript presents BinMod1d, a package for atmospheric coagulation and breakup using spectral methods. The paper is clearly written and the figures are intuitive. Overall, this is a useful software contribution that is appropriate for the target journal, filling an important niche for an accessible, open-source spectral-bin microphysics model implemented in a modern computing language. The manuscript provides relevant use cases, validation experiments, and demonstrations of the code's capabilities, and the software should be of interest to a broad community of researchers working in cloud microphysics and precipitation modeling.

The instructions for setting up and running the simulations were clear, and configuring the computational environment was straightforward. The figures were reproduced without difficulty and were consistent with those presented in the paper.

I recommend publication after the following suggestions are addressed.

Thank you very much for your comments and suggestions. I also want to thank you for taking the time to install and verify BinMod1D results in your computing environment. I have addressed your comments below in blue text.

Specific Comments

The parameter sbin appears throughout the manuscript but is not clearly defined upon first use. Readers unfamiliar with the author's notation must infer its meaning from the equation in line 197 and the subsequent discussion. Please provide an explicit definition (e.g., "number of bins per mass doubling") when first introduced.

I have rewritten this section by explicitly making the mass-doubling grid formula a separate equation with an explicit description of sbin as the number of bins per mass doubling. Note, that I have replaced all mentions of “s” with “s_{bin}” for consistency.

L183: More detail on the parallelization strategy would be helpful. Since multiple collision pairs may access the same source and destination bins, it is not immediately clear how race conditions are avoided. Are collision rates evaluated from a frozen particle distribution at the beginning of the timestep, or are bin populations updated during the collision loop? A brief description of the strategy would help readers assess the scalability of the implementation.

To answer your question: it’s the first approach for coalescence and the second approach for breakup. BinMod1D calculates the changes in bin mass/number for each bin-pair interaction as *flattened* arrays for all bins (and heights). For coalescence, the transfer is performed via *scatter adding* using the numpy add.at method and the recorded bin pair index for the bin-pair. For breakup, however, an additional Numba loop is used where the gains are calculated for *each thread* such that arrays are shaped (n_threads x heights x bins). Another loop then is used to do the transfer after to reduce the (n_threads x heights x bins) to (heights x bins) for each category. This is because the total breakup gain terms are

calculated from the loss terms that's calculated in the coalescence section. From trial and error, I found that this strategy was faster than the other methods I tried. I have incorporated this information in the manuscript as follows. Note that I reorganized this section to first describe the Wang et al. numerical integration methodology and then to describe the parallelization strategy at the end of section 2.1:

"To simplify the calculations, each I_k^r term is computed *separately* and then recombined for each k bin for each distribution. This is because each i-j interaction will generate gains in multiple, scattered bins. To perform these calculations, BinMod1D first flattens all arrays needed for each bin-pair interaction such that each bin-pair interaction has a *unique* set of integration values and destination bin indices. Then, BinMod1D efficiently utilizes "just-in-time" (JIT) compilation using the Numba Python package to perform the interaction integration calculations in *parallel for all bin-pair interactions at all heights at once*. This leads to BinMod1D effectively using machine code to loop through each possible bin-pair interaction which, in turn, leads to fast calculations with very low memory usage even when utilizing multiple distributions in the full 1D column model mode. For coalescence, the loss and gain transfer calculations are performed for each bin-pair interaction first using Numba-compiled loops and are stored in arrays with the length of the total number of bin pairs. **The transfer of the calculation from the i-j source space to the k and k+1 target space (Figure 2) is performed through scattered addition using the Numpy add.at method after all integration calculations are performed and stored. Collisional breakup, on the other hand, uses additional Numba-compiled loops after the breakup gain amounts are calculated to transfer the breakup fraction of the total i-j bin-pair source loss to each k target bin (see Figure 3) in the breakup distribution. Multidimensional temporary arrays are used to prevent race conditions during this transfer process where the Numba thread number, heights, and bins represent a three dimensional buffer array from which the transfer sequentially adds to each bin and height in the fragment distribution index for Mk (and Nk if 2 moments are used).** Several core BinMod1D components, including the parallelizable interaction kernel computations, were algorithmically streamlined using Google Gemini Pro as a code-refinement tool to ensure maximum computational speed in standard Python environments."

L316–324: The manuscript describes the use of multiple particle categories and destination-category routing through "cc_dest" and "br_dest". However, I think the paper would benefit from expanding on this, as well as an earlier description of how the kernels or efficiencies may change between classes. Please clarify whether kernels (or probably more importantly the associated efficiencies) are evaluated specifically for each category pair, and if all category pairs are allowed to interact. The flexibility is evident, but I think some recommendations for specifying these interactions would also be helpful, or at least a discussion of the default behavior when category-pair-specific information is not provided.

Thank you for this suggestion. While I wrote BinMod1D v.1.0.10 to only handle a *single kernel type for all categories*, the methodology itself actually utilizes kernel-based arrays with shape (pnum x Hlen x bins) where the zeroth index corresponds to the category pair with a total combination number of pnum = (dnum * (dnum + 1)) / 2, and dnum is the number of separate categories/distributions. Therefore, incorporating multiple kernels for each *combination pair* should not be particularly difficult as these kernels can simply be appropriately stacked along the zeroth index. However, I will need to think about how to best implement this change to ensure that users can easily and flexibly use multiple kernels. This change will be important for BinMod1D v.2.0.0 which will have mixed-phase particles and single-particle processes such as condensation, evaporation, sublimation, and melting.

I have included this clarification in the manuscript in section 2.1.

“As can be seen from Equation 4 and Equation 5, all integrals use the same collection kernel K(x,y|i,j) where BinMod1D v1.0.10 considers only a single type of kernel (e.g., hydrodynamic) between each combination of distribution categories.”

Additionally, I’ve expanded upon the cc_dest and br_dest descriptions in section 2.2 (see bold):

“Users can specify the number and type of distributions used in spectral_1d() by adding subsequent distributions to the list in the habit_params input parameter (e.g., habit_params=["snow","fragments"]). The length of habit_params determines the number of separate distributions that are solved where the names refer to the particle microphysical parameters in habits.py that are used for each distribution. **The spectral_1d() input parameter kernel determines the type of interaction kernel (i.e., K(x,y)) with the kernel input (e.g., kernel="Hydro", for the hydrodynamic or gravitational kernel). Users can explicitly set the cc_dest and br_dest input parameters when more than one distribution is specified. Here, cc_dest and br_dest determine the distribution destination of coalesced and breakup bin-pair interactions, respectively where integer values are given from 1 to len(habit_params). For example, a two-category BinMod1D simulation where the first distribution represents a primary category such as ice crystals and the second category represents snow aggregates would be initialized with cc_dest=2 and habit_params=[“snow”,“snow”]. Similarly, a two-category BinMod1D simulation with snow (and its aggregates) as one category and breakup fragments as the second category can be specified with cc_dest=1, br_dest=2, and habit_params = [“snow”, “fragments”]. Only one distribution/category is used when cc_dest= br_dest=1 (the default behavior).** Collision (Ecol), coalescence (Es) and breakup (Eb) efficiencies are treated as constants that can be passed into spectral_1d() directly (e.g., spectral_1d(Ecol=1.0,Es=0.5,Eb=0.5)). Here, the total collision-coalescence efficiency is given by E_CC = Eol Es whereas the total breakup efficiency E_BR is determined by: E_BR = E_col (1-Es) Eb. In this way, Eb represents the fraction of collisions that occur that do not coalesce but do break up.

Therefore, Eb=1.0 indicates the impossibility of rebounding whereas 0 < Eb < 1 indicates a combination of breakup and rebounding."

Eq. 18 Specify if Dx needs to be the larger size.

In the revised version, I've specified under equation 18 (now equation 19) that Dx>= Dy. While technically, the Long kernel *does* make the distinction between the larger and smaller sizes, this restriction is not currently incorporated in the code. However, this distinction doesn't matter for self-collection. Therefore, the results presented in the manuscript (i.e., Section 4, Figures 11 and 12, and Table 1) are still valid. Future updates to BinMod1D will ensure that the Long kernel has this distinction for scientific consistency with the Long kernel.

L106: Since the manuscript highlights the advantages of a Python implementation and this paragraph details the recent trend towards accessible languages, I suggest including citations of other Python-based microphysics frameworks that have coagulation, including Particula (https://github.com/Gorkowski/particula) and PySDM (https://github.com/open-atmos/PySDM). While these models use a Lagrangian rather than spectral-bin approach, they provide useful context for readers interested in open-source Python tools for cloud microphysics. It is also worth noting the language of LCM1d in L110.

I have included the Particula and PySDM references in this section as well. I have also specified the programming language as Python for each package/model as well.

While the repository includes tests for initialization, conservation, and basic diagnostics, the automated test suite appears to contain relatively few tests tied directly to the scientific validation presented in the manuscript. Given the emphasis on analytical benchmark solutions, breakup parameterizations, radar diagnostics, and multi-category interactions, I encourage the authors to consider incorporating more tests based on these published validation cases.

Thank you for this suggestion. The tests/ directory is meant to be strictly unit-based tests to ensure proper installation of BinMod1D and not scientific-oriented tests. The Jupyter notebook examples, on the other hand, are intended to provide these sorts of tests and are also meant to introduce users to the capabilities and use-cases of BinMod1D. I will definitely incorporate more scientific validation type tests in the second version of BinMod1D which will have single particle processes and other features not currently present in version 1.0.10.