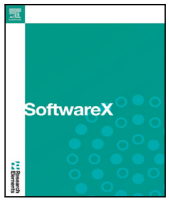
SoftwareX 25 (2024) 101613

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

PyPartMC: A Pythonic interface to a particle-resolved, Monte Carlo aerosol simulation framework

Zachary D'Aquino [a], Sylwester Arabas [b], Jeffrey H. Curtis [a], Akshunna Vaishnav [c], Nicole Riemer [a,*], Matthew West [d]

[a] Department of Atmospheric Sciences, University of Illinois Urbana-Champaign, Urbana, IL, USA
[b] Faculty of Physics and Applied Computer Science, AGH University of Krakow, Kraków, Poland
[c] Department of Electrical and Computer Engineering, University of Illinois Urbana-Champaign, Urbana, IL, USA
[d] Department of Mechanical Science and Engineering, University of Illinois Urbana-Champaign, Urbana, IL, USA

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.7662635

Keywords:
Python-to-Fortran interface
Particle-resolved aerosol simulation
Atmospheric modeling

ABSTRACT

PyPartMC is a Pythonic interface to PartMC, a stochastic, particle-resolved aerosol model implemented in Fortran. Both PyPartMC and PartMC are free, libre, and open-source. PyPartMC reduces the number of steps and mitigates the effort necessary to install and utilize the resources of PartMC. Without PyPartMC, setting up PartMC requires: working with UNIX shell, providing Fortran and C libraries, and performing standard Fortran and C source code configuration, compilation and linking. This can be challenging for those less experienced with computational research or those intending to use PartMC in environments where provision of UNIX tools is less straightforward (e.g., on Windows). PyPartMC offers a single-step installation/upgrade process of PartMC and all dependencies through the pip Python package manager on Linux, macOS, and Windows. This allows streamlined access to the unmodified and versioned Fortran internals of the PartMC codebase from both Python and other interoperable environments (e.g., Julia through PyCall). In particular, PyPartMC can be set up to handle the time-stepping loop for PartMC simulations making it possible to couple PartMC with other Python-interoperable packages, for either online diagnostics or additional simulation logic. Altogether, users of PyPartMC can setup, run, process and visualize output of PartMC simulations using a single general-purpose programming language.

Code metadata

Current code version	1.0.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00552
Code Ocean compute capsule	None
Legal Code License	GNU GPL v3.0
Code versioning system used	git
Software code languages, tools, and services used	C++, Fortran, C, Python, CMake, Jupyter
Compilation requirements, operating environments & dependencies	Dependencies: SuiteSparse, SUNDIALS, CAMP, PartMC, pybind11-JSON, pybind11, JSON-Fortran, nlohmann/json, netCDF OS: Linux, macOS, Windows
Link to developer documentation/manual	https://open-atmos.github.io/PyPartMC
Support email for questions	please use GitHub issue tracker

1. Motivation and significance

Open-source software fosters collaboration and accountability within the scientific community through joint development and maintenance ventures that accelerate scientific discovery and progress [1–4]. The interoperability and accessibility of proven computational tools produced through open-source projects can benefit from the development of bindings to popular high-level, general-purpose programming languages, such as Python. This helps in providing researchers with

* Corresponding author.
E-mail address: nriemer@illinois.edu (Nicole Riemer).

https://doi.org/10.1016/j.softx.2023.101613
Received 16 August 2023; Received in revised form 22 November 2023; Accepted 7 December 2023
Available online 23 December 2023

a community of contributors and users to support individual research pursuits. In this paper, we focus on the value of open-source software and the provision of a robust Python interface to an established Fortran codebase. The PyPartMC project documented in this paper has specific applicability in the field of aerosol science; however, the outlined design applies in general to the development of Pythonic interfaces to existing open-source models implemented in Fortran.

PyPartMC is a free, libre, and open-source Pythonic interface to the Fortran-implemented stochastic, particle-resolved aerosol model PartMC [5], currently at version 2.7.0. PartMC simulates the microphysical processes that aerosol particles undergo during their lifetimes in the atmosphere. These processes include new particle formation, emission from primary sources, Brownian diffusion, aqueous chemical processing, and removal by dry deposition and nucleation scavenging. PartMC categorically tracks the chemical composition of each computational particle, allowing the user to comprehensively simulate the mixing state and other physiochemical properties of an aerosol population. PartMC typically runs in conjunction with a solver that simulates the dynamic partitioning of semi-volatile organic and inorganic gases.

PartMC is a powerful open-source tool for simulating the dynamics of aerosol populations; however, running a PartMC simulation requires knowledge of shell and the use of a netCDF-Fortran library, which can present a significant hurdle for those with less experience in computation or those intending to use PartMC on platforms less compliant with the standard UNIX development environment. Current installation (or upgrade) of PartMC requires working command of Fortran compilers, CMake, netCDF, and awareness of the exact directories for a number of libraries on the user's machine. This process can be tedious and prone to errors due to the number of steps involved.

A primary motivating factor for developing PyPartMC is to remove limitations to installation and setup of PartMC dependent on the individual user's computational background. PyPartMC offers streamlined use of PartMC by providing a single-step installation process and flexibility in selecting the libraries and frameworks, for example, Jupyter and Dask [6] from the Python ecosystem, each individual user wishes to apply cooperatively during independent execution of simulations and routines based on PartMC. Prior to the development of PyPartMC, auxiliary programs written in Fortran were necessary to process the output from a PartMC simulation. In addition to providing a simpler installation experience, PyPartMC saves users of PartMC from having to code anything in Fortran. The interface capitalizes on many Python functionalities, such as garbage collection and native data types. Overall, one of the aims of the project is to enable researchers using PartMC to perform all tasks pertaining to setting up, running, post-processing and analyzing simulations using a single programming language.

PyPartMC is available as source and pre-compiled packages on pypi.org and can be installed at the command line via `pip install PyPartMC`. In case pre-compiled binaries for a given hardware or software configuration are unavailable, the `pip install` command orchestrates compilation of the package. The continuous integration (CI) setup maintained on the GitHub platform verifies a consistent installation experience across Linux, macOS and Windows platforms and is used for building pre-compiled packages. For Linux builds, standarized 'manylinux' environments are used to ensure compatibility with a wide range of systems. The source package ships with all the dependencies included (maintained using git submodules what streamlines updates and ensures version traceability). The binary packages have all dependencies statically linked, further minimizing installation environment constraints. The three-language (Fortran, C, C++) build automation is handled through CMake and wired to the Python package installation, making the process invisible to the user. Test automation is achieved with pytest [7].

Python's stature as a high-level, truly general-purpose programming language with its outstanding readability and ease of learning appeals to a much larger group of programmers that exceeds the size of the more specialized Fortran community. Exposing the PartMC codebase to the Python ecosystem enables individuals to take advantage of the expansive community of Python developers and users that provide support through online documentation and forums. These resources are not as prolific in the smaller Fortran community. In general, a lower-level language, like Fortran, requires more lines of code to accomplish the same task in a higher-level language. A programmer's productivity is constrained by the number of lines of code needed to accomplish a given task, and it has been shown that programmers write roughly the same number of line of code per unit time independent of the language used [8]. Consequently, tasks can be accomplished in shorter time with less code using higher-level programming languages, like Python. PyPartMC assists with comprehensibility and allows users to build code to their personal specifications at the rate of a high-level language with the performance boost of a low-level language.

All dependencies, such as SuiteSparse [9], SUNDIALS [10], Chemistry Across Multiple Phases (CAMP) [11], netCDF, and nlohmann/json [12], are free, open-source, and maintained with versioned releases, constituting a practical tool set for reproducible computational research. Simplifying installation and setup of PartMC through a Pythonic interface allows PyPartMC to be used to support the development of robust aerosol model benchmarking architectures. The ability to leverage resources from the PartMC simulation suite while taking advantage of existing libraries and frameworks that are ubiquitous in the scientific community provides the necessary infrastructure for PyPartMC to participate in the movement to improve STEM education through active learning applications [13]. One of the technologies that facilitates active learning by streamlining dissemination of interactive computational worksheets is Jupyter notebooks, which PyPartMC provides for usage examples (see Section 4 below). Noteworthy, the employment of Jupyter notebooks for active learning applications has already been demonstrated in the very field of aerosol science [14].

2. Software description

2.1. Particle-resolved modeling with PartMC

PartMC (Particle Monte Carlo) is a stochastic, particle-resolved aerosol box model, which resolves the composition of many individual aerosol particles within a well-mixed volume of air [5,15–17]. The documentation for PartMC can be found at http://lagrange.mechse.illinois.edu/partmc/partmc-2.6.1/doc/html. Aerosol processes such as coagulation, nucleation, emission, and deposition are implemented in a Monte Carlo fashion, i.e., by sampling the population with appropriate probabilities. Our particle-resolved approach uses a large number of discrete computational particles (10^4 to 10^6) to represent the particle population of interest. Each particle is represented by a "composition vector", which stores the mass of each constituent species within each particle and evolves over the course of a simulation according to various chemical or physical processes. The "weighted flow algorithm" [15,17] improves the model efficiency and reduces ensemble variance by assigning each computational particle a number concentration that this particle corresponds to. In contrast to other particle-based methods used for example in cloud physics, PartMC does not track the position of the computational particle in physical space. To account for multi-phase chemical processes, PartMC needs to be coupled to a chemistry code, e.g., the Model for Simulating Aerosol Interactions and Chemistry (MOSAIC) [18] or CAMP [11]. PartMC has been used in many applications as a box model, but has also been coupled to the Weather Research and Forecast model (WRF) to simulate aerosol processes and transport in a 3D model domain, with WRF as the host model supplying the flow field. Since the particles' locations in 3D space are not stored, we use a stochastic sampling approach to move particles between grid boxes [19].

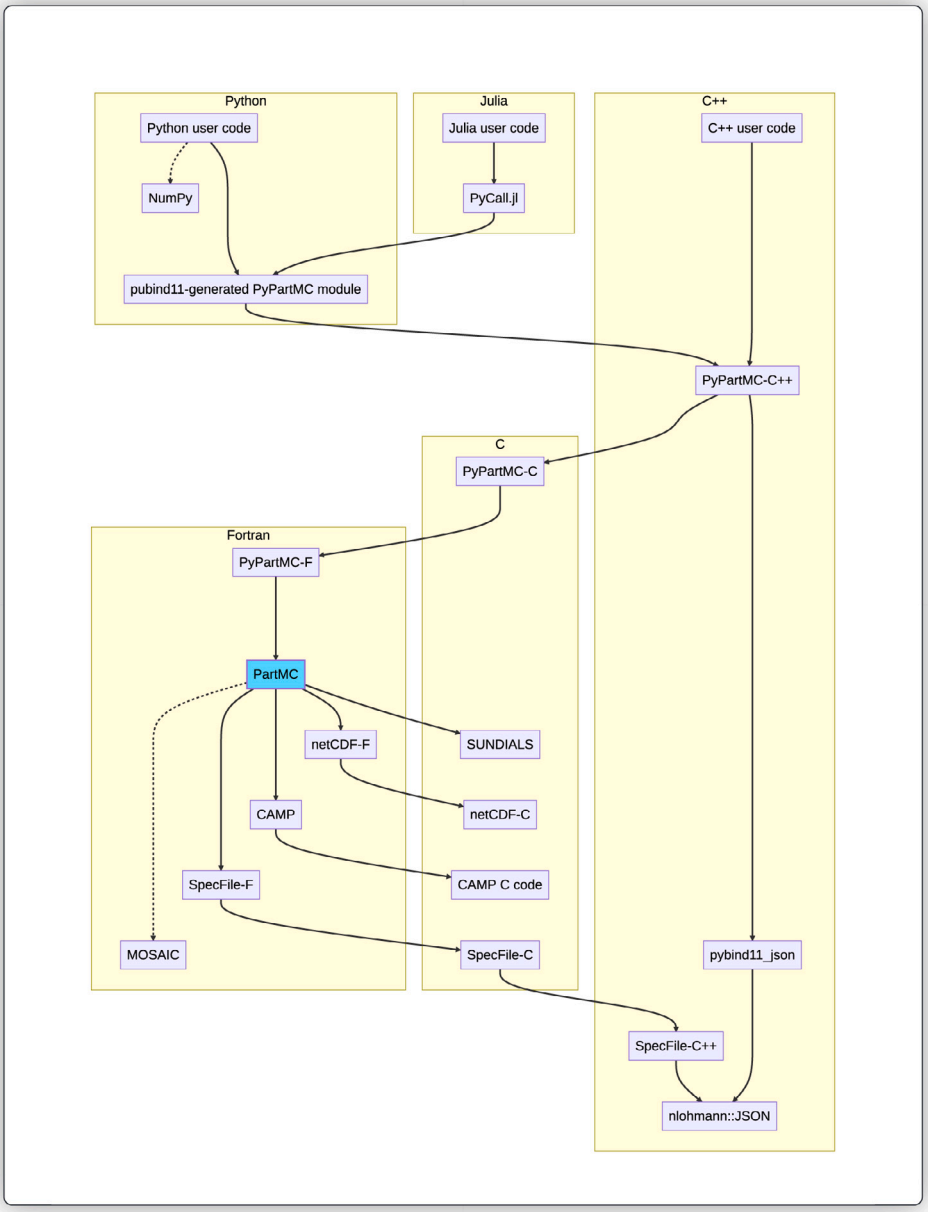


Fig. 1. Schematic of architecture of PyPartMC depicting the coupling between Fortran, C and C++ internals and the user code written in Python or Julia.

2.2. Software architecture

PyPartMC is written in Fortran, C, and C++. Fortran wrappers of the existing PartMC Fortran code are defined in the PyPartMC codebase using the standard Fortran `ISO_C_BINDING` functionalities. These are used in concert with a thin C-implemented wrapping layer to define C++ bindings to PartMC Fortran datatypes and routines. The pybind11 framework is then used to build the Python API from these C++ bindings (see Fig. 1 for an overview of the design). PartMC has been implemented in an object-oriented manner, leveraging Fortran derived types which are extensively used throughout the modular PartMC codebase. These derived types are exposed in the PyPartMC bindings as Python classes. Unlike the case of black-box Python bindings to Fortran code [e.g., as in 20], PyPartMC reveals the underlying internals of the Fortran modules to Python users.

PyPartMC is a Pythonic interface that will be natural to use for those familiar with the Python programming language but retains PartMC jargon. Python garbage collection deallocates Fortran objects instantiated through PyPartMC, ensuring user code is succinct and memory-safe. Instantiation and deserialization of exposed PartMC structures are handled using JSON-like data structures which provides an authentic Pythonic experience. Users leverage Python's built-in data types, syntax, and comprehension to construct and store simulation parameters which eliminates the need for any input text files used in typical PartMC simulation executions.

PyPartMC has continuous integration workflows for Linux, macOS, and Windows handled through GitHub Actions which verifies that any new feature added to the project is checked through automated testing and compiles successfully on all targeted platforms. Unit tests account for a significant portion of the codebase and certify that developments do not cause regressions by changing the behavior of previously working code. Test execution is orchestrated by pytest, and the PyPartMC unit test suite assures information and execution pass as expected between the wrapping code and PartMC internals. PartMC contains

its own testing suite that checks the underlying physical processes and their representations within the model.

Docstring-based PyPartMC API documentation is published on the web at https://open-atmos.github.io/PyPartMC upon each pull request merge. New versions of PyPartMC are easily disseminated thanks to automated uploads to pypi.org triggered by GitHub releases.

2.3. Software functionality

This software architecture preserves the performance advantages of a Fortran-implemented codebase and allows PyPartMC to simultaneously serve as a C++ API to the PartMC Fortran internals. This structure also circumvents the need for any temporary files and does not require any additional routines to be added to the PartMC codebase for interfacing.

The Python packaging infrastructure expedites dissemination of both source and binary packages ensuring code version traceability for PyPartMC bindings, PartMC itself, and all other dependencies. Noteworthy, this – together with the employed static linkage of all dependencies – effectively circumvents the notorious difficulties in disseminating binary version of multi-dependency Fortran software rooted in the lack of binary compatibility between compilers, and even between versions of the same Fortran compiler [21].

The design also capitalizes on Python's ability to integrate with other languages, so PyPartMC can be called from alternative environments. Similarly, PyPartMC can be used to couple PartMC with simulation or diagnostics components implemented in other software and hardware (e.g., GPU) solutions interoperable with Python. To this end, PyPartMC provides the option to perform time-stepping within PartMC simulations in Python, and this is one of the rationales for designing the package as an API to PartMC internals rather than merely wrapping PartMC solver control procedures. One of these environments includes the Julia programming language which has been adopted by many in the field of computational research, including aerosol scientists [14]. An example employing PyPartMC from Julia (using PyCall) is maintained in the PyPartMC README file on GitHub and shown in Section 3 below (also see diagram in Fig. 1).

The design choice of building an API for PartMC rather than using shell wrappers eliminates overhead and allows for more freedom in external coupling logic, providing the added benefit of supporting adoption of new features added to PartMC with compiler static analysis of the wrapping code. In cases where flags are read exclusively from PartMC input text files, no new API elements are needed. PyPartMC can simply be compiled against another version of PartMC to allow access to newly added flags in the PartMC codebase; however, this may not be possible if the interface were constructed in another manner. Additionally, prior installation of PartMC is not required, static linkage of PartMC and all dependencies ensures that the versions of the wrapper, library and dependencies match.

PyPartMC currently offers access to a subset of PartMC functions (approximately 20%). While the process of giving access to a larger subset is still ongoing, not all functions will need to be given access to. The choice of making functions available has been largely user-driven; starting from the most fundamental functions (returning particle masses), we added more recently functions that are related to specific post-processing functionalities (such as computing critical supersaturation and aerosol mixing state parameters).

Finally, regarding the performance of the implemented solution, it is worth highlighting that the overhead of starting the Python interpreter, loading the package and even (optionally) performing time-stepping loop in Python is only noticeable in simulations with impractically small state-vector sizes. Testing setups featuring 10^2 and 10^4 particles, this constant overhead was measured to amount to roughly 10% and 1% of the process CPU time, respectively.

3. Basic examples in Python and Julia

Fig. 2 depicts how the identical task of randomly sampling particles from an aerosol size distribution in PartMC can be done in three different programming languages. Listing (a) presents Python code employing PyPartMC. Listing (b) is an analogous code written in Julia where PyPartMC is accessed using PyCall to fully interoperate with Python. Listings (c-f) depict how this task is achieved using Fortran native to PartMC. In this case, the parameters of the particle size distribution are provided in external text files.

4. Illustrative examples (Jupyter notebooks)

PyPartMC ships with a collection of Jupyter notebook examples with Python kernel displaying how PyPartMC can be leveraged to access PartMC resources in a Python environment. In the README of the PyPartMC repository, users can navigate to a collection of badges corresponding to each example that will render the notebook with nbviewer, execute in the cloud via Binder or Google Colab [22], or create a standalone web application with Voilà. These example notebooks work out of the box and constitute experiential and manageable tools users can employ to explore the utility of the API and immediately begin building on the provided code. The notebooks illustrate how results obtained through PyPartMC based on PartMC simulations can be contained in a web-based notebook and easily distributed with minimal software and hardware dependencies. Example 1 showcases the comparison of two different software packages that both calculate the water uptake of aerosol particles. Example 2 illustrates the simulation of a typical PartMC scenario and post- processing of the raw data to obtain aggregate quantities that are common in aerosol science. Example 3 illustrates the simulation of PartMC to supply particle populations to an external Python package to compute aerosol optical properties.

Example 1: Comparison with PySDM

This example compares PyPartMC directly to PySDM [23], another package for simulating particle dynamics. Computational particles are instantiated from a dry aerosol size distribution prescribed by interactive widget sliders. PyPartMC bindings pass information regarding the environmental state, such as relative humidity and temperature, size distribution parameters, and the hygroscopic parameter, kappa, to PartMC internals to evaluate equilibrium state under the given conditions. Shown in Fig. 3(a), the initial dry distribution and the updated wet distribution predicted by PartMC are plotted alongside the result from PySDM. The two result curves from PartMC and PySDM look visually indistinguishable, however small numerical differences exist in calculating droplet size after the equilibriation with ambient relative humidity as shown in Fig. 3(b). This is to be expected given that two different code bases were used. The use of a Jupyter notebook with Python kernel allows figures to be easily produced and downloaded using matplotlib [24]. This particular example showcases the utility of employing PyPartMC to foster purposeful comparisons between different modeling frameworks, facilitated by both packages being exposed to Python and ready-to-use in a web-based notebook with interactive output. Additionally, PyPartMC has been incorporated into the PySDM v2 test suite to run automated checks against PartMC for this same equilibrium example [25].

Example 2: Urban plume simulation

In another example, we implemented a modified scenario based on [5] which considered an idealized urban plume. The scenario tracks the evolution of an air parcel which is advected over an urban area. During the advection process, the parcel experiences emissions of gases

a: Python code (with embedded data)

```python
import numpy as np

import PyPartMC as ppmc
from PyPartMC import si

aero_data = ppmc.AeroData((
    #      [density, ions in solution, molecular weight, kappa]
    {"OC": [1000 *si.kg/si.m**3, 0, 1e-3 *si.kg/si.mol, 0.001]},
    {"BC": [1800 *si.kg/si.m**3, 0, 1e-3 *si.kg/si.mol, 0]},
))

aero_dist = ppmc.AeroDist(
    aero_data,
    [{
        "cooking": {
            "mass_frac": [{"OC": [1]}],
            "diam_type": "geometric",
            "mode_type": "log_normal",
            "num_conc": 3200 / si.cm**3,
            "geom_mean_diam": 8.64 * si.nm,
            "log10_geom_std_dev": 0.28,
        }
    },
    {
        "diesel": {
            "mass_frac": [{"OC": [0.3]}, {"BC": [0.7]}],
            "diam_type": "geometric",
            "mode_type": "log_normal",
            "num_conc": 2900 / si.cm**3,
            "geom_mean_diam": 50 * si.nm,
            "log10_geom_std_dev": 0.24,
        }
    }],
)

n_part = 100
aero_state = ppmc.AeroState(aero_data, n_part, "nummass_source")
aero_state.dist_sample(aero_dist)
print(np.dot(aero_state.masses, aero_state.num_concs), "# kg/m3")
```

b: Julia code (with embedded data)

```julia
using Pkg
Pkg.add("PyCall")

using PyCall
ppmc = pyimport("PyPartMC")
si = ppmc["si"]

aero_data = ppmc.AeroData((
  #       (density, ions in solution, molecular weight, kappa)
  Dict("OC"=>(1000 * si.kg/si.m^3, 0, 1e-3 * si.kg/si.mol, 0.001)),
  Dict("BC"=>(1800 * si.kg/si.m^3, 0, 1e-3 * si.kg/si.mol, 0))
))

aero_dist = ppmc.AeroDist(aero_data, (
  Dict(
    "cooking" => Dict(
      "mass_frac" => (Dict("OC" => (1,)),),
      "diam_type" => "geometric",
      "mode_type" => "log_normal",
      "num_conc" => 3200 / si.cm^3,
      "geom_mean_diam" => 8.64 * si.nm,
      "log10_geom_std_dev" => .28,
    )
  ),
  Dict(
    "diesel" => Dict(
      "mass_frac" => (Dict("OC" => (.3,)), Dict("BC" => (.7,))),
      "diam_type" => "geometric",
      "mode_type" => "log_normal",
      "num_conc" => 2900 / si.cm^3,
      "geom_mean_diam" => 50 * si.nm,
      "log10_geom_std_dev" => .24,
    )
  )
))

n_part = 100
aero_state = ppmc.AeroState(aero_data, n_part, "nummass_source")
aero_state.dist_sample(aero_dist)
print(aero_state.masses'aero_state.num_concs, "# kg/m3")
```

c: Fortran code

```fortran
program main
  use pmc_spec_file
  use pmc_aero_data
  use pmc_aero_mode
  use pmc_aero_dist
  use pmc_aero_state

  implicit none

  type(spec_file_t) :: f_aero_data, f_aero_dist
  type(aero_data_t) :: aero_data
  type(aero_dist_t) :: aero_dist
  type(aero_state_t) :: aero_state
  integer, parameter :: n_part = 100
  integer :: n_part_add
  real(kind=dp), dimension(n_part) :: num_concs, masses

  call spec_file_open("aero_data.dat", f_aero_data)
  call spec_file_read_aero_data(f_aero_data, aero_data)
  call spec_file_close(f_aero_data)

  call spec_file_open("aero_dist.dat", f_aero_dist)
  call spec_file_read_aero_dist(f_aero_dist, aero_data, aero_dist)
  call spec_file_close(f_aero_dist)

  call aero_state_zero(aero_state)
  call fractal_set_spherical(aero_data%fractal)
  call aero_state_set_weight(aero_state, aero_data, &
    AERO_STATE_WEIGHT_NUMMASS_SOURCE)
  call aero_state_set_n_part_ideal(aero_state, dble(n_part))
  call aero_state_add_aero_dist_sample(aero_state, aero_data, &
    aero_dist, 1d0, 0d0, .true., .true., n_part_add)

  num_concs = aero_state_num_concs(aero_state, aero_data)
  masses = aero_state_masses(aero_state, aero_data)
  print *, dot_product(num_concs, masses), "# kg/m3"
end
```

d: aero_dist.dat file (for Fortran code)

```
mode_name cooking
mass_frac cooking_comp.dat
diam_type geometric
mode_type log_normal
num_conc 3.2e9  # (#/m^3)
geom_mean_diam 8.64e-9  # (m)
log10_geom_std_dev 0.28

mode_name diesel
mass_frac diesel_comp.dat
diam_type geometric
mode_type log_normal
num_conc 2.9e9  # (#/m^3)
geom_mean_diam 5e-8
log10_geom_std_dev 0.24
```

e: cooking_comp.dat file (for Fortran code)

```
#        proportion
OC             1
```

f: diesel_comp.dat file (for Fortran code)

```
#        proportion
OC             0.3
BC             0.7
```

Fig. 2. Code listings depicting basic usage of PartMC data structures and algorithms in: (a) Python through PyPartMC; (b) Julia through PyCall and PyPartMC; (c–f) Fortran.

and aerosols for 12 h. After 12 h of simulation, the emissions are switched off and the parcel continues to evolve. The scenario presented here consists of gas and aerosol emissions, gas and aerosol dilution with the background, and particle coagulation. This differs from the case presented in [5] as gas phase chemistry, gas-aerosol partitioning and aerosol thermodynamics were not included. Normally, these components are incorporated by coupling the PartMC model with MOSAIC [18]. However, since MOSAIC code is only available upon request and not available as open source, we do not include it here.

In Fig. 4, we show the time evolution of the total aerosol number and mass concentration. This figure is constructed by calling PyPartMC functions that sum up the number concentrations and the total mass concentration of each particle.

Fig. 5(a) shows the evolution of the initial aerosol size distribution after 6, 12 and 24 h. This figure is constructed by calling functions to compute particle dry diameters and acquire arrays of number concentrations associated with each computational particle. With these two variables, a 1D histogram function is used to construct particle size distributions. Fig. 5(b) shows the two-dimensional distribution of black carbon mass fraction and particle dry diameter after 24 h of simulation time. Here the PyPartMC API is used to compute arrays of particle diameters and also acquire black carbon mass, total dry mass and number concentration of each computational particle. To analyze the data, a 2D histogram is created using arrays of dry diameters and black carbon mass fraction with each computational particle weighted by its respective number concentration.

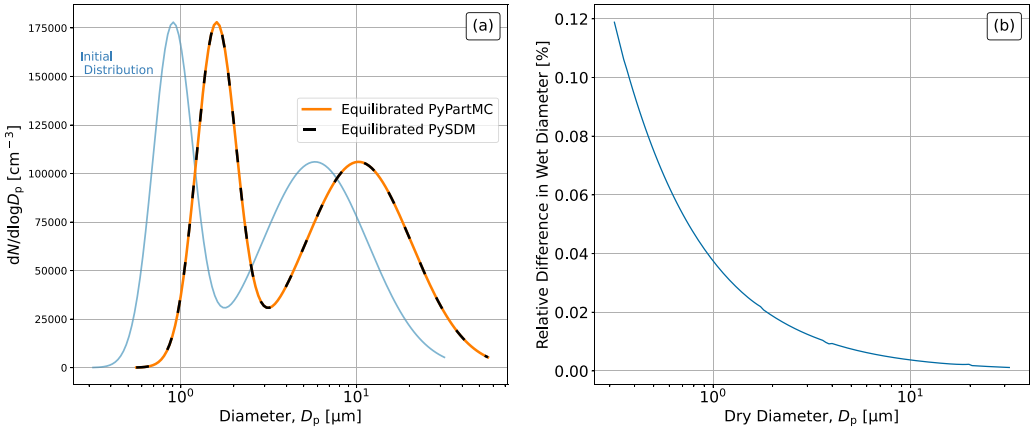


Fig. 3. (a) Initial dry size distribution (light blue) and distributions after water uptake from PyPartMC and PySDM (orange and black, respectively) evaluated at a temperature of 295 Kelvin, relative humidity of 82%, and hygroscopicity parameter kappa of 1.1. Note that the solutions from PyPartMC and PySDM are almost identical, and the orange and black curve are on top of each other. (b) Relative difference in wet diameter from PyPartMC and PySDM as a function of dry diameter. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

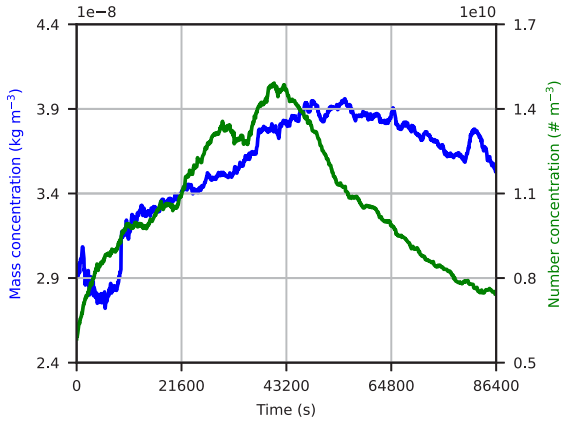


Fig. 4. Evolution of total number concentration and total mass concentration.

Example 3: Optical properties calculation using an external Python package

As a final example, we implemented a particle simulation where we incorporated calls to an external library (PyMieScatt) that computes the optical properties of particle scattering and absorption [26]. PyPartMC evolves the aerosol population and then computes the per-particle core diameters and total diameters. Here Python controls the timestepping using a "time block", which passes to Fortran a time interval consisting of many time steps to simulate. For this example, we chose a time block of 3600 s and an internal model time step of 60 s. Upon completion of each time block, a function to compute single-particle optics is called by passing the AeroState type, which is a collection of AeroParticles. Inside this function, arrays of particle core diameter and particle total diameter are constructed, and the entire list of particles is iterated over to compute the scattering and absorption efficiency of each particle. Additionally, the function computes bulk optical coefficients of scattering and absorption by computing each particle's cross sectional area multiplied by optical efficiencies and weighted by each computational particle's number concentration. Overall, the involved arrays can be quite large, since they have the dimension of $N_{\text{particles}} \times N_{\text{species}}$ (on the order of 10,000 × 20), and having an API interface facilitates the analysis since it avoids exporting to files using the wide range of data formats supported by Python libraries (including JSON, netCDF). Fig. 6 shows the results of computing absorption efficiencies of all particles in the population at a given time. This example shows how PyPartMC makes available a large quantity of data in the Python environment that can be easily passed to other Python libraries. Possibilities of external libraries include other optical models as well as gas/aerosol chemistry and machine learning codes, all of which may potentially feedback to the PyPartMC simulation as applicable.

5. Impact

Removing limitations to and streamlining the use of PartMC with PyPartMC will facilitate the dissemination of computational research results through independent execution of PartMC simulations, which could prove advantageous during the peer review process. Additionally, the ability to easily package examples, simple simulations, and results in a web-based notebook allows PyPartMC to support the efforts of many members of the scientific community, including researchers, instructors, and students, with nominal software and hardware requirements.

Researchers familiar with Python can take advantage of the PartMC simulation suite through the PyPartMC interface to tackle their own unique research questions and interests without ever needing to know about Fortran's involvement, while still gaining access to PartMC internals (Fortran derived types exposed as Python classes). This distinction makes the utility of the PartMC codebase more accessible and usable to those outside of the scientific modeling enclave.

Python interfacing provides the added benefit of full access to NumPy [27]. NumPy's position as the foundational array programming library for Python has supported research endeavors in numerous fields, and its memory model makes passing and manipulating arrays between Python, C, and Fortran straightforward [28]. It also facilitates the integration of PartMC with Python-based machine learning platforms such as PyTorch [29] as well as with other programming languages through prominent Python-bridging tools (e.g., PyCall for the Julia language, see Fig. 2).

More frequent comparisons between experiments, measurements, and simulations to include PyPartMC guide further development and collaborative improvements. Visible development contributes to transparency in research and enables easy tracking of model performance over time. The open-source nature of PartMC in conjunction with PyPartMC's flexibility in external coupling empowers users to modify the code to suit their own unique research goals and tackle complex scientific questions with any number of interoperable software libraries and frameworks at their disposal.

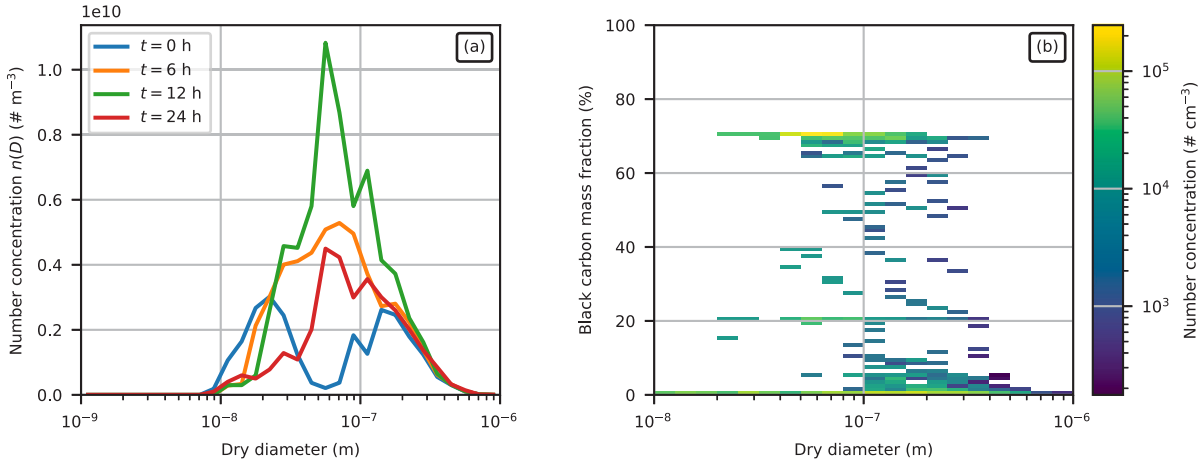


Fig. 5. (a) Number distributions $n(D)$ for the simulation with coagulation after 0, 6, 12 and 24 h. (b) Two-dimensional number distribution of black carbon dry mass fraction and particle dry diameter after 24 h of simulation.

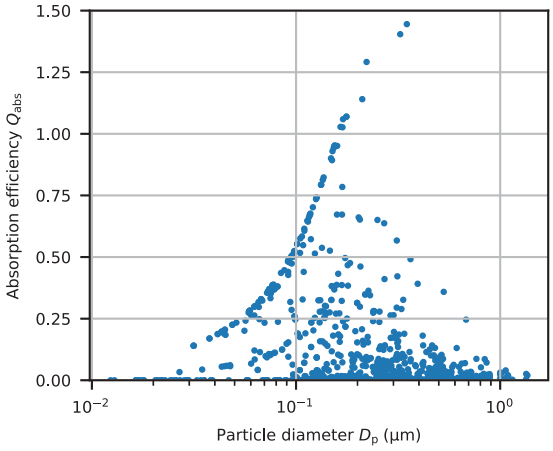


Fig. 6. Absorption efficiency of individual particles as a function particle diameter. Each point represents an individual particle.

6. Conclusions

Open-source scientific projects strive to make research code more accessible to the wider scientific community by mitigating the steps and effort needed to apply these resources while simultaneously providing the infrastructure necessary to couple with existing tools. Increasing accessibility and usability of scientific software introduces more diversity to the community as researchers and students from various backgrounds and institutions can implement and advance the same project.

In this paper, we highlight the role of a Pythonic interface in making an existing open-source aerosol model more accessible and usable to those with less computational experience. Installation (or upgrade) of PyPartMC and all dependencies is reduced to a single command, permitting access to the functionalities of the PartMC codebase on Linux, macOS, and Windows. Example Jupyter notebooks exhibit how PyPartMC may be used in a Python environment to assist in executing meaningful comparisons between PartMC and other modeling frameworks, paving the way for more robust model benchmarking exercises. This capability also demonstrates the ease of replicating and disseminating research results with PyPartMC.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

There is no associated dataset. Figures presented in the paper can be independently recreated using code packaged with PyPartMC. PyPartMC and all its dependencies are free/libre and open-source software. Each release of the package is persistently archived at Zenodo: https://doi.org/10.5281/zenodo.7662635.

Acknowledgments

This study was supported by the U.S. Department of Energy's Atmospheric System Research, an Office of Science Biological and Environmental Research program, under grants DE-SC0021034 and DE-SC0022130, and by the National Science Foundation, United States under grant NSF-AGS 19-41110. SA acknowledges support from the Polish National Science Centre (grant no. 2020/39/D/ST10/01220).

References

[1] Heistermann M, Collis S, Dixon M, Giangrande S, Helmus J, Kelley B, et al. The emergence of open-source software for the weather radar community. Bull Am Meteorol Soc 2015;96(1):117–28. http://dx.doi.org/10.1175/BAMS-D-13-00240.1.

[2] Bangerth W, Heister T. What makes computational open source software libraries successful? Comput Sci Discov 2013;6(1):015010. http://dx.doi.org/10.1088/1749-4699/6/1/015010.

[3] Piva E, Rentocchini F, Rossi-Lamastra C. Is open source software about innovation? Collaborations with the open source community and innovation performance of software entrepreneurial ventures. J Small Bus Manag 2012;50(2):340–64. http://dx.doi.org/10.1111/j.1540-627X.2012.00356.x.

[4] Kogut B, Metiu A. Open-source software development and distributed innovation. Oxford Rev Econ Policy 2001;17(2):248–64. http://dx.doi.org/10.1093/oxrep/17.2.248.

[5] Riemer N, West M, Zaveri RA, Easter RC. Simulating the evolution of soot mixing state with a particle-resolved aerosol model. J Geophys Res: Atmos 114(D9). http://dx.doi.org/10.1029/2008JD011073.

[6] Rocklin M, et al. Dask: Parallel computation with blocked algorithms and task scheduling. In: Proceedings of the 14th Python in science conference, vol. 130. SciPy Austin, TX; 2015, p. 136, URL https://web.archive.org/web/20190429074549id_/http://conference.scipy.org/proceedings/scipy2015/pdfs/matthew_rocklin.pdf.

[7] Krekel H, et al. pytest (2023, version 7.4.0), URL https://github.com/pytest-dev/pytest.

[8] Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. Best practices for scientific computing. PLOS Biol. 2014;12(1):e1001745. http://dx.doi.org/10.1371/journal.pbio.1001745.

[9] Davis TA. Algorithm 1000: suitesparse:graphblas: graph algorithms in the language of sparse linear algebra. ACM Trans. Math. Softw. 2019;45(4). http://dx.doi.org/10.1145/3322125.

[10] Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, et al. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. ACM Trans Math Software 2005;31(3):363–96. http://dx.doi.org/10.1145/1089014.1089020.

[11] Dawson ML, Guzman C, Curtis JH, Acosta M, Zhu S, Dabdub D, et al. Chemistry across multiple phases (CAMP) version 1.0: an integrated multiphase chemistry model. Geosci Model Dev 2022;15(9):3663–89. http://dx.doi.org/10.5194/gmd-15-3663-2022.

[12] Lohmann N. JSON for Modern C++ (2022, version 3.11.2). URL https://github.com/nlohmann.

[13] Freeman S, Eddy SL, McDonough M, Smith MK, Okoroafor N, Jordt H, et al. Active learning increases student performance in science, engineering, and mathematics. Proc Natl Acad Sci USA 2014;111(23):8410–5. http://dx.doi.org/10.1073/pnas.1319030111.

[14] Petters M. Interactive worksheets for teaching atmospheric aerosols and cloud physics. Bull Am Meteorol Soc 2021;102(3):E672–80. http://dx.doi.org/10.1175/BAMS-D-20-0072.2.

[15] DeVille REL, Riemer N, West M. Weighted flow algorithms (WFA) for stochastic particle coagulation. J Comput Phys 2011;230(23):8427–51. http://dx.doi.org/10.1016/j.jcp.2011.07.027.

[16] Curtis J, Michelotti M, Riemer N, Heath M, West M. Accelerated simulation of stochastic particle removal processes in particle-resolved aerosol models. J Comput Phys 2016;322:21–32. http://dx.doi.org/10.1016/j.jcp.2016.06.029.

[17] DeVille L, Riemer N, West M. Convergence of a generalized weighted flow algorithm for stochastic particle coagulation. Comput Dyn 2019;1–18. http://dx.doi.org/10.3934/jcd.2019003.

[18] Zaveri RA, Easter RC, Fast JD, Peters LK. Model for simulating aerosol interactions and chemistry (MOSAIC). J Geophys Res: Atmos 113(D13). http://dx.doi.org/10.1029/2007JD008782.

[19] Curtis JH, Riemer N, West M. A single-column particle-resolved model for simulating the vertical distribution of aerosol mixing state: WRF-PartMC-MOSAIC-SCM v1.0. Geosci Model Dev 2017;10:4057. http://dx.doi.org/10.5194/gmd-10-4057-2017.

[20] van den Oord G, Jansson F, Pelupessy I, Chertova M, Grönqvist JH, Siebesma P, et al. A python interface to the dutch atmospheric large-eddy simulation. SoftwareX 2020;12:100608. http://dx.doi.org/10.1016/j.softx.2020.100608.

[21] Kedward LJ, Aradi B, Čertík O, Curcic M, Ehlert S, Engel P, et al. The state of fortran. Comput Sci Eng 2022;24(2):63–72. http://dx.doi.org/10.1109/MCSE.2022.3159862.

[22] Bisong E. Google colaboratory. In: Building machine learning and deep learning models on google cloud platform. Springer; 2019, p. 59–64. http://dx.doi.org/10.1007/978-1-4842-4470-8.

[23] Bartman P, Bulenok O, Górski K, Jaruga A, Łazarski G, Olesik MA, et al. PySDM v1: particle-based cloud modeling package for warm-rain microphysics and aqueous chemistry. J Open Source Softw 7(72). http://dx.doi.org/10.21105/joss.03219.

[24] Hunter JD. Matplotlib: A 2D graphics environment. Comput Sci Eng 2007;9(3):90–5. http://dx.doi.org/10.1109/MCSE.2007.55.

[25] de Jong EK, Singer CE, Azimi S, Bartman P, Bulenok O, Derlatka K, et al. New developments in PySDM and PySDM-examples v2: collisional breakup, immersion freezing, dry aerosol initialization, and adaptive time-stepping. J Open Source Softw http://dx.doi.org/10.21105/joss.04968.

[26] Sumlin BJ, Heinson WR, Chakrabarty RK. Retrieving the aerosol complex refractive index using pymiescatt: A mie computational package with visualization capabilities. J Quant Spectrosc Radiat Transfer 2018;205:127–34. http://dx.doi.org/10.1016/j.jqsrt.2017.10.012.

[27] Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.

[28] Harris CR, Millman KJ, Van Der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.

[29] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. Pytorch: An imperative style, high-performance deep learning library. In: Advances In Neural Information Processing Systems, vol. 32. Curran Associates, Inc.; 2019, p. 8024–35, URL http://web.archive.org/web/20200420235543/http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.