PyPartMC

aerosol dynamics package:

Sylwester Arabas[1], Zach D'Aquino[2], Jeff Curtis[2], Nicole Riemer[2], Matt West[3]
& [Py]PartMC contributors

Jan 26th 2024
Columbia University, New York

[1]Physics & Applied CS, AGH University of Krakow, Poland (agh.edu.pl)
[2]Atmospheric Sciences, University of Illinois at Urbana-Champaign (atmos.illinois.edu)
[3]Mechanical Science & Engineering, University of Illinois at Urbana-Champaign (mechse.illinois.edu)

- long story short:

- long story short:
  - PhD (physics) @ University of Warsaw, PL (+ JAMSTEC, JP)
    - ⇝ Arabas & Shima 2013 (JAS): LES with probabilistic particle-based μ-physics
    - ⇝ libcloudph++: particle-based μ-physics on GPU (Arabas et al. 2015, GMD)

- long story short:
  - PhD (physics) @ University of Warsaw, PL (+ JAMSTEC, JP)
    - ⇝ Arabas & Shima 2013 (JAS): LES with probabilistic particle-based μ-physics
    - ⇝ libcloudph++: particle-based μ-physics on GPU (Arabas et al. 2015, GMD)
  - "fintech" break:
    - ⇝ Arabas & Farhat 2020 (JCAM): derivative pricing as a transport problem

- long story short:
  - PhD (physics) @ University of Warsaw, PL (+ JAMSTEC, JP)
    - ⇝ Arabas & Shima 2013 (JAS): LES with probabilistic particle-based μ-physics
    - ⇝ libcloudph++: particle-based μ-physics on GPU (Arabas et al. 2015, GMD)
  - "fintech" break:
    - ⇝ Arabas & Farhat 2020 (JCAM): derivative pricing as a transport problem
  - postdoc @ Jagiellonian University, Kraków, PL
    - ⇝ PySDM & PyMPDATA

- long story short:
  - PhD (physics) @ University of Warsaw, PL (+ JAMSTEC, JP)
    - ⇝ Arabas & Shima 2013 (JAS): LES with probabilistic particle-based μ-physics
    - ⇝ libcloudph++: particle-based μ-physics on GPU (Arabas et al. 2015, GMD)
  - "fintech" break:
    - ⇝ Arabas & Farhat 2020 (JCAM): derivative pricing as a transport problem
  - postdoc @ Jagiellonian University, Kraków, PL
    - ⇝ PySDM & PyMPDATA
  - postdoc (Nicole Riemer's group) @ University of Illinois, Urbana-Champaign, USA
    - ⇝ Monte-Carlo immersion freezing in particle-based μ-physics & PyPartMC

- long story short:
  - PhD (physics) @ University of Warsaw, PL (+ JAMSTEC, JP)
    - ⇝ Arabas & Shima 2013 (JAS): LES with probabilistic particle-based μ-physics
    - ⇝ libcloudph++: particle-based μ-physics on GPU (Arabas et al. 2015, GMD)
  - "fintech" break:
    - ⇝ Arabas & Farhat 2020 (JCAM): derivative pricing as a transport problem
  - postdoc @ Jagiellonian University, Kraków, PL
    - ⇝ PySDM & PyMPDATA
  - postdoc (Nicole Riemer's group) @ University of Illinois, Urbana-Champaign, USA
    - ⇝ Monte-Carlo immersion freezing in particle-based μ-physics & PyPartMC
  - since mid '23: @ AGH University of Krakow, PL
    - ⇝ isotopic composition of water in particle-based μ-physics

- long story short:
  - PhD (physics) @ University of Warsaw, PL (+ JAMSTEC, JP)
    - ⇝ Arabas & Shima 2013 (JAS): LES with probabilistic particle-based μ-physics
    - ⇝ libcloudph++: particle-based μ-physics on GPU (Arabas et al. 2015, GMD)
  - "fintech" break:
    - ⇝ Arabas & Farhat 2020 (JCAM): derivative pricing as a transport problem
  - postdoc @ Jagiellonian University, Kraków, PL
    - ⇝ PySDM & PyMPDATA
  - postdoc (Nicole Riemer's group) @ University of Illinois, Urbana-Champaign, USA
    - ⇝ Monte-Carlo immersion freezing in particle-based μ-physics & PyPartMC
  - since mid '23: @ AGH University of Krakow, PL
    - ⇝ isotopic composition of water in particle-based μ-physics
- maintainer & developer:
  - github.com/numba-mpi
  - github.com/open-atmos/{PySDM,PyMPDATA,PyPartMC}

PyPartMC

aerosol dynamics package:

Sylwester Arabas[1], Zach D'Aquino[2], Jeff Curtis[2], Nicole Riemer[2], Matt West[3]
& [Py]PartMC contributors

Jan 26th 2024
Columbia University, New York

[1]Physics & Applied CS, AGH University of Krakow, Poland (agh.edu.pl)
[2]Atmospheric Sciences, University of Illinois at Urbana-Champaign (atmos.illinois.edu)
[3]Mechanical Science & Engineering, University of Illinois at Urbana-Champaign (mechse.illinois.edu)

PyPartMC

aerosol dynamics package:

engineering Python-to-Fortran bindings
in C++, for use in Julia and Matlab

Sylwester Arabas[1], Zach D'Aquino[2], Jeff Curtis[2], Nicole Riemer[2], Matt West[3]
& [Py]PartMC contributors

Jan 26th 2024
Columbia University, New York

[1]Physics & Applied CS, AGH University of Krakow, Poland (agh.edu.pl)
[2]Atmospheric Sciences, University of Illinois at Urbana-Champaign (atmos.illinois.edu)
[3]Mechanical Science & Engineering, University of Illinois at Urbana-Champaign (mechse.illinois.edu)

plan of the talk

PyPartMC: context / statement of need

PyPartMC: goals and status

PyPartMC: design & implementation outline

PyPartMC: demo

PyPartMC: summary

PartMC

https://lagrange.mechse.illinois.edu/partmc/

https://lagrange.mechse.illinois.edu/partmc/

- Monte-Carlo aerosol dynamics simulation package

https://lagrange.mechse.illinois.edu/partmc/

- Monte-Carlo aerosol dynamics simulation package
- open source, v1.0.0 back in 2007

https://lagrange.mechse.illinois.edu/partmc/

- Monte-Carlo aerosol dynamics simulation package
- open source, v1.0.0 back in 2007
- developed by Riemer, West, Curtis, et al.

https://lagrange.mechse.illinois.edu/partmc/

- Monte-Carlo aerosol dynamics simulation package
- open source, v1.0.0 back in 2007
- developed by Riemer, West, Curtis, et al.
- box-model framework with a coupler to, e.g., WRF

https://lagrange.mechse.illinois.edu/partmc/

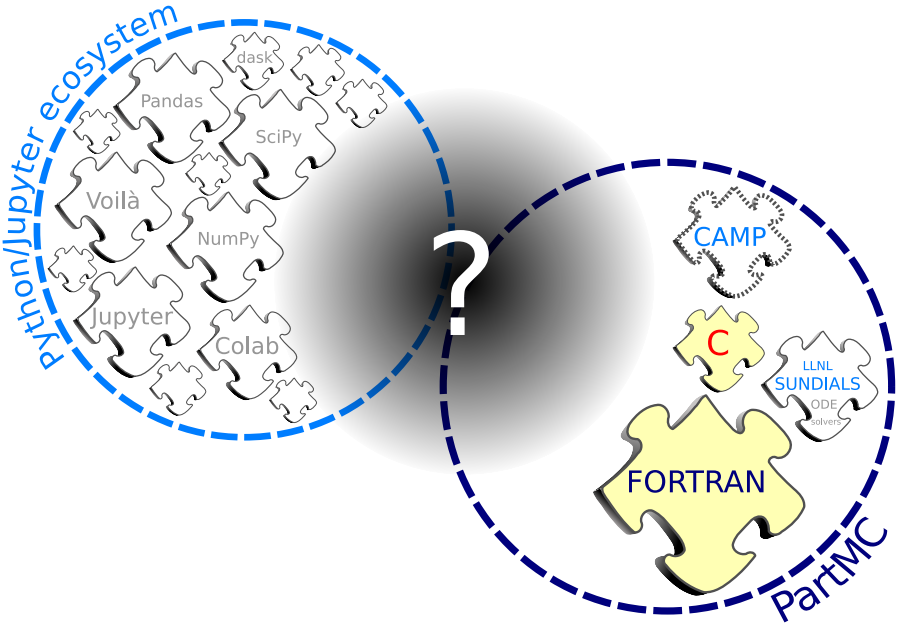
- Monte-Carlo aerosol dynamics simulation package
- open source, v1.0.0 back in 2007
- developed by Riemer, West, Curtis, et al.
- box-model framework with a coupler to, e.g., WRF
- coagulation, condensation, gas- and particle-phase chemistries (MOSAIC/CAMP)

https://lagrange.mechse.illinois.edu/partmc/

- Monte-Carlo aerosol dynamics simulation package
- open source, v1.0.0 back in 2007
- developed by Riemer, West, Curtis, et al.
- box-model framework with a coupler to, e.g., WRF
- coagulation, condensation, gas- and particle-phase chemistries (MOSAIC/CAMP)
- highlight: aerosol mixing state evolution

https://lagrange.mechse.illinois.edu/partmc/

- Monte-Carlo aerosol dynamics simulation package
- open source, v1.0.0 back in 2007
- developed by Riemer, West, Curtis, et al.
- box-model framework with a coupler to, e.g., WRF
- coagulation, condensation, gas- and particle-phase chemistries (MOSAIC/CAMP)
- highlight: aerosol mixing state evolution
- object-oriented architecture, F90, extensive automated test suite


Python/Jupyter ecosystem
dask
Pandas
SciPy
Voilà
NumPy
Jupyter
Colab
?
CAMP
C
LLNL
SUNDIALS
ODE
solvers
FORTRAN
PartMC

plan of the talk

PyPartMC: context / statement of need

PyPartMC: goals and status

PyPartMC: design & implementation outline

PyPartMC: demo

PyPartMC: summary

project goals

- **lower the entry threshold for installing and setting up of PartMC**
  down to pip install PyPartMC, i.e., no manual dependency installation, no compilation, user doesn't even need to know FORTRAN is involved

project goals

- **lower the entry threshold for installing and setting up of PartMC** down to pip install PyPartMC, i.e., no manual dependency installation, no compilation, user doesn't even need to know FORTRAN is involved
- ensure the same experience on Linux, macOS & Windows

project goals

- **lower the entry threshold for installing and setting up of PartMC**
  down to pip install PyPartMC, i.e., no manual dependency installation, no compilation, user doesn't even need to know FORTRAN is involved
- ensure the same experience on Linux, macOS & Windows
- **lower the entry threshold for usage with Jupyter-based example notebooks**

project goals

- **lower the entry threshold for installing and setting up of PartMC**
  down to pip install PyPartMC, i.e., no manual dependency installation, no compilation, user doesn't even need to know FORTRAN is involved
- ensure the same experience on Linux, macOS & Windows
- **lower the entry threshold for usage with Jupyter-based example notebooks**
- streamline the dissemination of paper-result reproducers (peer review)

status of the project: v1.0 in Dec 2023 (started 2021)

SoftwareX

ORIGINAL SOFTWARE PUBLICATION

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

Zachary D'Aquino • Sylwester Arabas • Jeffrey H. Curtis • Akshunna Vaishnav • Nicole Riemer • Matthew West

Open Access • Published: December 23, 2023 • DOI: https://doi.org/10.1016/j.softx.2023.101613

Help Sponsors Log in Register

PyPartMC 1.0.1

```
pip install PyPartMC
```

Latest version

Released: Dec 16, 2023

plan of the talk

PyPartMC: context / statement of need

PyPartMC: goals and status

PyPartMC: design & implementation outline

PyPartMC: demo

PyPartMC: summary

pybind11

master | 27 Branches | 58 Tags | Go to file | Go to file | Code

README | License | Security

pybind11

pybind11 — Seamless operability between C++11 and Python

docs passing | docs stable | chat on gitter | Discussions Ask | CI passing | build passing

latest packaged version 2.11.1 | pypi v2.11.1 | conda-forge v2.11.0 | python 3.6 | 3.7 | 3.8 | 3.9 | 3.10 | 3.11 | 3.12

Setuptools example • Scikit-build example • CMake example

pybind11 is a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code. Its goals and syntax are similar to the excellent Boost.Python library by David Abrahams: to minimize boilerplate code in traditional extension modules by inferring type information using compile-time introspection.

About

Seamless operability between C++11 and Python

pybind11.readthedocs.io/

#python #bindings

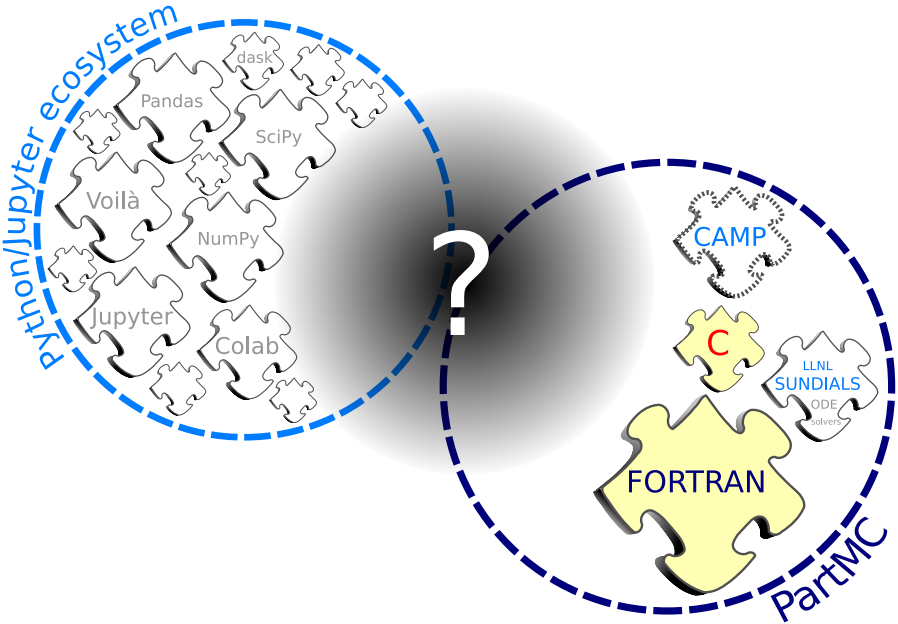
Readme
View license
Security policy
Activity
Custom properties
14.3k stars
250 watching
2.1k forks
Report repository

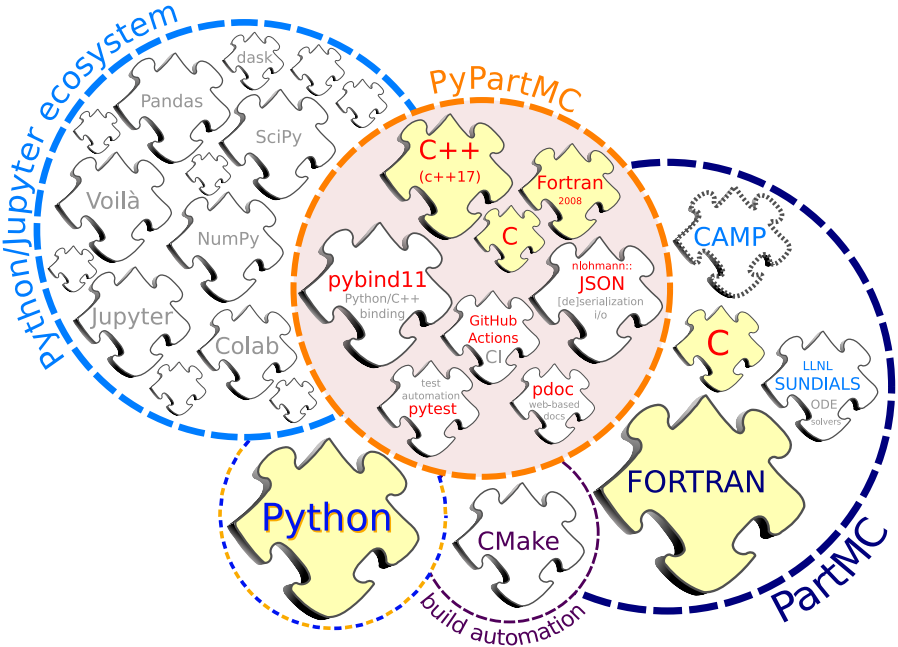
Releases 21

Version 2.11.1 Latest
on Jul 17, 2023

+ 20 releases

Contributors 337


Python/Jupyter ecosystem
dask
Pandas
SciPy
Voilà
NumPy
Jupyter
Colab
?
CAMP
C
LLNL
SUNDIALS
ODE
solvers
FORTRAN
PartMC


Python/Jupyter ecosystem
dask
Pandas
SciPy
Voilà
NumPy
Jupyter
Colab
PyPartMC
C++
(c++17)
Fortran
2008
C
pybind11
Python/C++
binding
GitHub
Actions
CI
nlohmann::
JSON
[de]serialization
I/o
test
automation
pytest
pdoc
web-based
docs
CAMP
C
LLNL
SUNDIALS
ODE
solvers
FORTRAN
PartMC
Python
CMake
build automation

solving the challenges

- written in C/Fortran/C++ as **C++ bindings** to PartMC intetrnals (derived types), Python bindings generated using **pybind11**

solving the challenges

- written in C/Fortran/C++ as **C++ bindings** to PartMC intetrnals (derived types), Python bindings generated using **pybind11**
- three-language build automation with CMake, test automation with pytest, CI workflows

solving the challenges

- written in C/Fortran/C++ as **C++ bindings** to PartMC intetrnals (derived types), Python bindings generated using **pybind11**
- three-language build automation with CMake, test automation with pytest, CI workflows
- JSON-based reimplementation of PartMC "spec-file" i/o module (unmodified code of PartMC uses original API)
  $\rightsquigarrow$ minimising effort to accommodate future additions to PartMC

solving the challenges

- written in C/Fortran/C++ as **C++ bindings** to PartMC intetrnals (derived types), Python bindings generated using **pybind11**
- three-language build automation with CMake, test automation with pytest, CI workflows
- JSON-based reimplementation of PartMC "spec-file" i/o module (unmodified code of PartMC uses original API)
  $\rightsquigarrow$ minimising effort to accommodate future additions to PartMC
- freeing of Python-allocated PartMC FORTRAN types through Python Garbage Collector

solving the challenges

- written in C/Fortran/C++ as **C++ bindings** to PartMC intetrnals (derived types), Python bindings generated using **pybind11**
- three-language build automation with CMake, test automation with pytest, CI workflows
- JSON-based reimplementation of PartMC "spec-file" i/o module (unmodified code of PartMC uses original API)
  $\rightsquigarrow$ minimising effort to accommodate future additions to PartMC
- freeing of Python-allocated PartMC FORTRAN types through Python Garbage Collector
- dependency version pinning with git submodules: PartMC (F), CAMP (C/F), json (C++), pybind11 (C++), json-fortran (F), netCDF (C/F), SUNDIALS (F/C), SuiteSparse (C), ...
  & backports of C++20 features to C++17 (multilinux!): span, string_view, optional

solving the challenges

- written in C/Fortran/C++ as **C++ bindings** to PartMC intetrnals (derived types), Python bindings generated using **pybind11**
- three-language build automation with CMake, test automation with pytest, CI workflows
- JSON-based reimplementation of PartMC "spec-file" i/o module (unmodified code of PartMC uses original API)
  $\rightsquigarrow$ minimising effort to accommodate future additions to PartMC
- freeing of Python-allocated PartMC FORTRAN types through Python Garbage Collector
- dependency version pinning with git submodules: PartMC (F), CAMP (C/F), json (C++), pybind11 (C++), json-fortran (F), netCDF (C/F), SUNDIALS (F/C), SuiteSparse (C), ...
  & backports of C++20 features to C++17 (multilinux!): span, string_view, optional
- all dependencies (incl. Fortran and C++ runtimes) statically linked (single-file install)

user perspective: Fortran (PartMC)

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
```

user perspective: Python (PyPartMC)

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")
```

user perspective: Python (PyPartMC) & Julia (via PyCall.jl)

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")
```

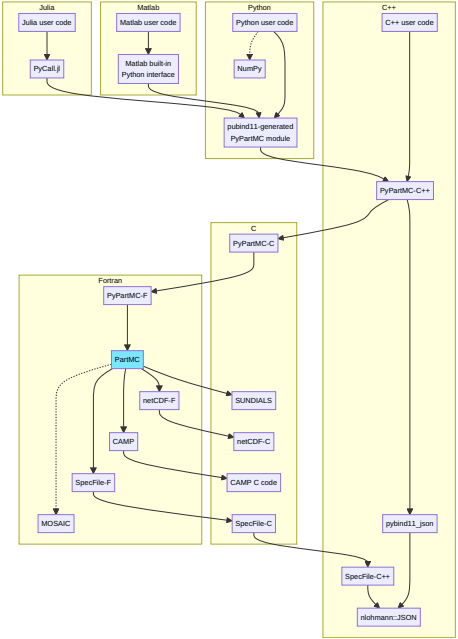
user perspective: Matlab (built-in Python bridge)

```
ppmc = py.importlib.import_module('PyPartMC');
si = py.importlib.import_module('PyPartMC').si;

aero_data = ppmc.AeroData(py.tuple({ ...
  py.dict(pyargs("OC", py.tuple({1000 * si.kg/si.m^3, 0, 1e-3 * si.kg/si.mol, 0.001}))), ...
  py.dict(pyargs("BC", py.tuple({1800 * si.kg/si.m^3, 0, 1e-3 * si.kg/si.mol, 0}))) ...
}));

aero_dist = ppmc.AeroDist(aero_data, py.tuple({ ...
  py.dict(pyargs( ...
    "cooking", py.dict(pyargs( ...
      "mass_frac", py.tuple({py.dict(pyargs("OC", py.tuple({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 ...
    )) ...
  )), ...
  py.dict(pyargs( ...
    "diesel", py.dict(pyargs( ...
      "mass_frac", py.tuple({ ...
        py.dict(pyargs("OC", py.tuple({.3}))), ...
        py.dict(pyargs("BC", py.tuple({.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);
masses = cell(aero_state.masses());
num_concs = cell(aero_state.num_concs);
fprintf('%g # kg/m3\n', dot([masses{:}], [num_concs{:}]))
```


Julia
Julia user code
PyCall.jl
Matlab
Matlab user code
Matlab built-in Python interface
Python
Python user code
NumPy
pubind11-generated PyPartMC module
C++
C++ user code
PyPartMC-C++
C
PyPartMC-C
Fortran
PyPartMC-F
PartMC
netCDF-F
SUNDIALS
CAMP
netCDF-C
SpecFile-F
CAMP C code
MOSAIC
SpecFile-C
pybind11_json
SpecFile-C++
nlohmann::JSON

PyPartMC API docs: https://open-atmos.github.io/PyPartMC/

PyPartMC API documentati

https://open-atmos.github.io/PyPartMC/

Index

Functions

condense_equilib_particles
pow2_above
run_part

Classes

AeroData
AeroState
EnvState
GasData
GasState
RunPartOpt
Scenario

Module **PyPartMC**

PyPartMC is a Python interface to PartMC.

Functions

```
def condense_equilib_particles(arg0: EnvState, arg1: AeroData,
                               arg2: AeroState)
```

Call condense_equilib_particle() on each particle in the aerosol to ensure that every particle has its water content in equilibrium.

```
def pow2_above(arg0: int) -> int
```

Return the least power-of-2 that is at least equal to n.

```
def run_part(arg0: Scenario, arg1: EnvState, arg2: AeroData, arg3: AeroState,
             arg4: GasData, arg5: GasState, arg6: RunPartOpt)
```

Do a particle-resolved Monte Carlo simulation.

Classes

```
class AeroData (...)
```

Aerosol material properties and associated data.

The data in this structure is constant, as it represents physical quantities that cannot

plan of the talk

PyPartMC: context / statement of need

PyPartMC: goals and status

PyPartMC: design & implementation outline

PyPartMC: demo

PyPartMC: summary

PyPartMC: demo

https://github.com/open-atmos/PyPartMC

plan of the talk

PyPartMC: context / statement of need

PyPartMC: goals and status

PyPartMC: design & implementation outline

PyPartMC: demo

PyPartMC: summary

PyPartMC firsts (?):

PyPartMC firsts (?):

- using PartMC on Windows

PyPartMC firsts (?):

- using PartMC on Windows
- using pybind11 for Fortran

PyPartMC firsts (?):

- using PartMC on Windows
- using pybind11 for Fortran
- using pybind11-generated packages from within Matlab

PyPartMC firsts (?):

- using PartMC on Windows
- using pybind11 for Fortran
- using pybind11-generated packages from within Matlab

PyPartMC [fun] facts:

- architecture entirely contingent on PartMC's modular/OOP design (and tests!)

PyPartMC firsts (?):

- using PartMC on Windows
- using pybind11 for Fortran
- using pybind11-generated packages from within Matlab

PyPartMC [fun] facts:

- architecture entirely contingent on PartMC's modular/OOP design (and tests!)
- 500+ lines of CMake code (compilation, static linkage of dependencies)

PyPartMC firsts (?):

- using PartMC on Windows
- using pybind11 for Fortran
- using pybind11-generated packages from within Matlab

PyPartMC [fun] facts:

- architecture entirely contingent on PartMC's modular/OOP design (and tests!)
- 500+ lines of CMake code (compilation, static linkage of dependencies)
- Conda packaging tricky due to static linkage

PyPartMC firsts (?):

- using PartMC on Windows
- using pybind11 for Fortran
- using pybind11-generated packages from within Matlab

PyPartMC [fun] facts:

- architecture entirely contingent on PartMC's modular/OOP design (and tests!)
- 500+ lines of CMake code (compilation, static linkage of dependencies)
- Conda packaging tricky due to static linkage
- no automatic dissemination of universal binaries for macOS yet (gfortran limitation)

PyPartMC firsts (?):

- using PartMC on Windows
- using pybind11 for Fortran
- using pybind11-generated packages from within Matlab

PyPartMC [fun] facts:

- architecture entirely contingent on PartMC's modular/OOP design (and tests!)
- 500+ lines of CMake code (compilation, static linkage of dependencies)
- Conda packaging tricky due to static linkage
- no automatic dissemination of universal binaries for macOS yet (gfortran limitation)
- Matlab bridge has issues, but Matlab Github Actions highly appreciated!

PyPartMC firsts (?):

- using PartMC on Windows
- using pybind11 for Fortran
- using pybind11-generated packages from within Matlab

PyPartMC [fun] facts:

- architecture entirely contingent on PartMC's modular/OOP design (and tests!)
- 500+ lines of CMake code (compilation, static linkage of dependencies)
- Conda packaging tricky due to static linkage
- no automatic dissemination of universal binaries for macOS yet (gfortran limitation)
- Matlab bridge has issues, but Matlab Github Actions highly appreciated!
- SoftwareX review: actually also concerned code/installation

PyPartMC firsts (?):

- using PartMC on Windows
- using pybind11 for Fortran
- using pybind11-generated packages from within Matlab

PyPartMC [fun] facts:

- architecture entirely contingent on PartMC's modular/OOP design (and tests!)
- 500+ lines of CMake code (compilation, static linkage of dependencies)
- Conda packaging tricky due to static linkage
- no automatic dissemination of universal binaries for macOS yet (gfortran limitation)
- Matlab bridge has issues, but Matlab Github Actions highly appreciated!
- SoftwareX review: actually also concerned code/installation
- exception propagation from C++ through Fortran to C++ compiler dependent

PyPartMC enables:

PyPartMC enables:

- single-command (pip) install on Windows, macOS & Linux

PyPartMC enables:

- single-command (pip) install on Windows, macOS & Linux
- using unmodified PartMC internals from Python, Julia, Matlab... and C++

PyPartMC enables:

- single-command (pip) install on Windows, macOS & Linux
- using unmodified PartMC internals from Python, Julia, Matlab... and C++
- using PartMC within test suites of other Python packages (as is the case of PySDM)

PyPartMC enables:

- single-command (pip) install on Windows, macOS & Linux
- using unmodified PartMC internals from Python, Julia, Matlab... and C++
- using PartMC within test suites of other Python packages (as is the case of PySDM)
- leveraging Python binary dissemination system for PartMC and dependencies (static linkage)

PyPartMC enables:

- single-command (pip) install on Windows, macOS & Linux
- using unmodified PartMC internals from Python, Julia, Matlab... and C++
- using PartMC within test suites of other Python packages (as is the case of PySDM)
- leveraging Python binary dissemination system for PartMC and dependencies (static linkage)
- encapsulating simulation setup/input within one single-language file (e.g., for paper review)

PyPartMC enables:

- single-command (pip) install on Windows, macOS & Linux
- using unmodified PartMC internals from Python, Julia, Matlab... and C++
- using PartMC within test suites of other Python packages (as is the case of PySDM)
- leveraging Python binary dissemination system for PartMC and dependencies (static linkage)
- encapsulating simulation setup/input within one single-language file (e.g., for paper review)
- extending PartMC simulation/diagnostics logic with Python code (optics with PyMieScatt)

PyPartMC enables:

- single-command (pip) install on Windows, macOS & Linux
- using unmodified PartMC internals from Python, Julia, Matlab... and C++
- using PartMC within test suites of other Python packages (as is the case of PySDM)
- leveraging Python binary dissemination system for PartMC and dependencies (static linkage)
- encapsulating simulation setup/input within one single-language file (e.g., for paper review)
- extending PartMC simulation/diagnostics logic with Python code (optics with PyMieScatt)
- streamlined workflows for generating simulation ensembles (no need for input text files!)

PyPartMC enables:

- single-command (pip) install on Windows, macOS & Linux
- using unmodified PartMC internals from Python, Julia, Matlab... and C++
- using PartMC within test suites of other Python packages (as is the case of PySDM)
- leveraging Python binary dissemination system for PartMC and dependencies (static linkage)
- encapsulating simulation setup/input within one single-language file (e.g., for paper review)
- extending PartMC simulation/diagnostics logic with Python code (optics with PyMieScatt)
- streamlined workflows for generating simulation ensembles (no need for input text files!)
- offering users (students) a single-language familiar environment (Colab, ARM JupyterHub)

PyPartMC: Removing barriers in aerosol modeling

Submitter

Riemer, Nicole — University of Illinois Urbana-Champaign
West, Matthew — University of Illinois at Urbana-Champaign

Area of research

Aerosol Processes

Journal Reference

D'Aquino Z, S Arabas, J Curtis, A Vaishnav, N Riemer, and M West. 2024. "PyPartMC: A Pythonic interface to a particle-resolved, Monte Carlo aerosol simulation framework." *SoftwareX*, 25, 101613, 10.1016/j.softx.2023.101613.

Science

PartMC is a powerful open-source tool for aerosol simulations. However, it requires knowledge of shell and CMake, C and Fortran compilers, and installation and configuration of several C and Fortran dependencies. This is a significant hurdle for those with little experience in computation. PyPartMC offers a single-step installation process of PartMC and all dependencies through the pip Python package manager on Linux, macOS, and Windows. It provides 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).

Impact

- Ability to run PartMC simulations in the cloud, including using the ARM Jupyter Hub.

acknowledgements

ASR
Atmospheric System Research

NSF

NATIONAL SCIENCE CENTRE
POLAND

acknowledgements

ASR Atmospheric System Research

NSF

NATIONAL SCIENCE CENTRE POLAND

Thank you for your attention!

pypi.org/p/PyPartMC
github.com/open-atmos/PyPartMC
doi:10.1016/j.softx.2023.101613