Methods & Tools for Modelling of Atmospheric Cloud μ-Physics

habilitation in "Earth and related environmental sciences" discipline submitted to Jagiellonian Univ.

Sylwester Arabas

21 November 2025

AGH Faculty of Physics and Appl. CS Seminar

ENVIRONMENTAL PHYSICS GROUP

Sylwester Arabas: µ-CV

(PhD in Physics, University of Warsaw, 2013)

Institute of Geophysics, Faculty of Physics,
University of Warsaw
("adiunkt" 2014-2015)

(2015–2018: three-year non-academic employment period)

Institute of Computer Science and Computational Mathematics,
Faculty of Mathematics and Computer Science, Jagiellonian University
("adiunkt" 2018–2021 & 2022–2023)

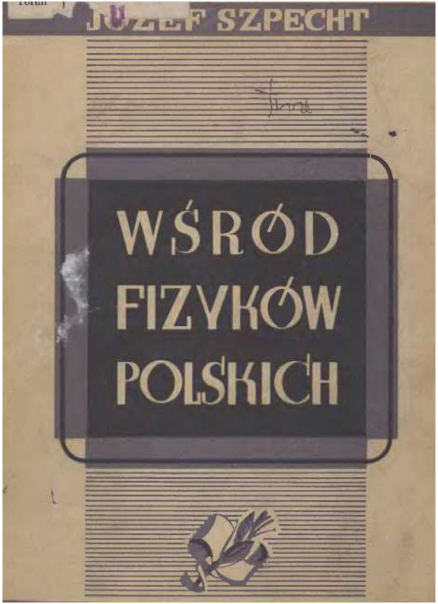
Department of Atmospheric Sciences,
University of Illinois Urbana-Champaign, USA
("postdoc" 2021–2022)

Environmental Physics Group, Faculty of Physics and Applied Computer Science,
AGH University of Krakow
(„adiunkt" since 2023)

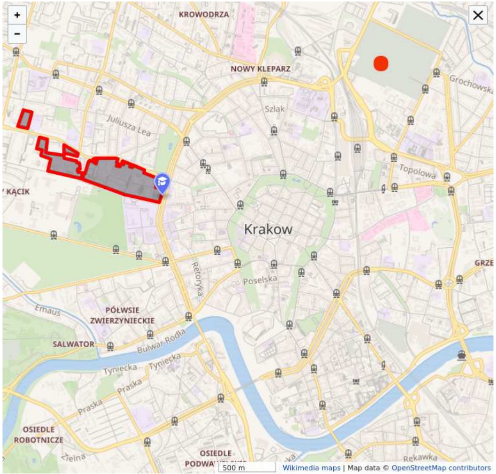
atmospheric cloud μ-physics models

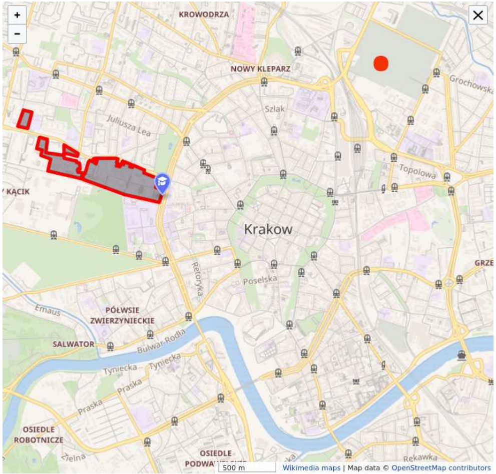
cloud := colloid of water droplets/crystals in air; colloidal instability$\rightsquigarrow$precipitation

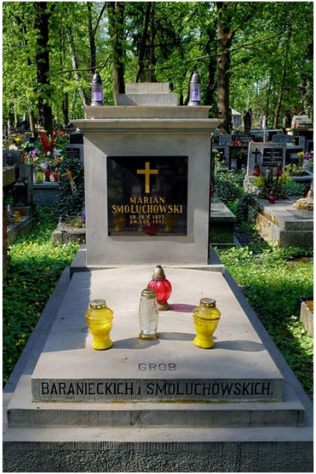
"Cloud and ship. Ukraine, Crimea, Black sea, view from Ai-Petri mountain"
(photo: Yevgen Timashov / National Geographic)

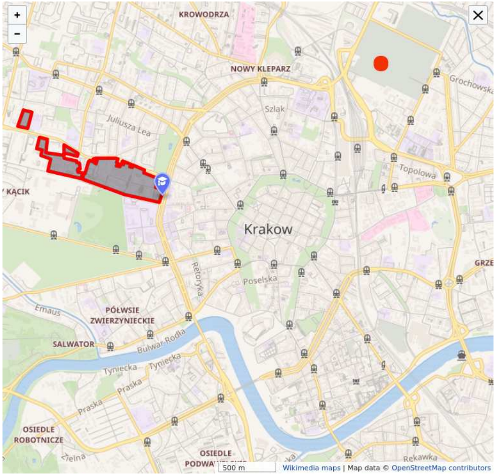


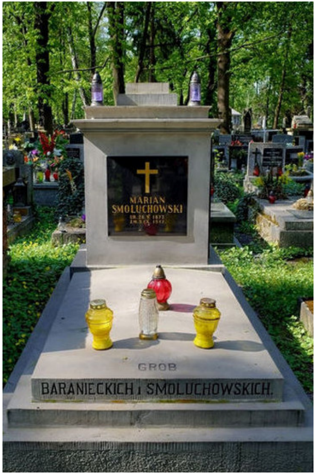
Szpecht 1939, Lwów, https://kpbc.umk.pl/dlibra/publication/75254/edition/86027/content


KROWODRZA
NOWY KLEPARZ
Grochowska
Szlak
Juliusza Lea
Topolowa
KĄCIK
Krakow
GRZE
Retoryka
Poselska
Emaus
PÓŁWSIE
ZWIERZYNIECKIE
SALWATOR
Bulwar Rodła
Tyniecka
Tyniecka
Praska
Praska
OSIEDLE
ROBOTNICZE
Zielna
OSIEDLE
Rękawka
500 m
Wikimedia maps | Map data © OpenStreetMap contributors


KROWODRZA
NOWY KLEPARZ
Szlak
Juliusza Lea
Grochowska
Topolowa
KĄCIK
Krakow
GRZE
Retoryka
Poselska
PÓŁWSIE
ZWIERZYNIECKIE
Emaus
SALWATOR
Bulwar Rodła
Tyniecka
Praska
OSIEDLE
ROBOTNICZE
OSIEDLE
Rękawka
500 m
Wikimedia maps | Map data © OpenStreetMap contributors


MARIAN
SMOLUCHOWSKI
GRÓB
BARANIECKICH i SMOLUCHOWSKICH.


KROWODRZA
NOWY KLEPARZ
Grochowska
Szlak
Juliusza Lea
Topolowa
Krakow
Poselska
Retoryka
Emaus
PÓŁWSIE
ZWIERZYNIECKIE
SALWATOR
Bulwar Rodła
Tyniecka
Praska
OSIEDLE
ROBOTNICZE
OSIEDLE
Rękawka
500 m
Wikimedia maps | Map data © OpenStreetMap contributors


MARIAN
SMOLUCHOWSKI
GROB
BARANIECKICH i SMOLUCHOWSKICH.


Dr MAURYCY PIUS
RUDZKI
HERBU PRAWDZIC
PROFESOR
UNIW. JAGIELLOŃSKIEGO
PRZEŻYŁ LAT 54
ZM 20 LIPCA 1916

Smoluchowski & Rudzki

Versuch einer mathematischen Theorie der Koagulationskinetik kolloider Lösungen.

Von

M. v. Smoluchowski.

(Mit 3 Figuren im Text.)

(Eingegangen am 8. 9. 16.)

I. Einleitung.

So sehr auch bis heute die Literatur über Koagulation kolloider Lösungen angewachsen ist, sind doch unsere Kenntnisse betreffs des quantitativen Verlaufs, sowie betreffs des Mechanismus des Koagulationsprozesses äusserst mangelhaft. Die meisten Forscher begnügen sich mit qualitativen Beobachtungen oder stellen ihre Messungsreihen in Tabellen oder Kurvenform[1]) dar, da die mathematische Wiedergabe derselben auf aussergewöhnliche Schwierigkeiten stösst.

In den interessanten Arbeiten[2]) von S. Miyazawa, N. Ishizaka, H. Freundlich, J. A. Gann wird allerdings eine formelmässige Zusammenfassung des empirischen Versuchsmaterials, sowie eine Aufklärung desselben nach Analogie mit den Gesetzen der chemischen Kinetik angestrebt. Aber klare Gesetzmässigkeiten haben sich bisher auf diese Weise nicht ergeben, und wurden sogar gewisse, anfangs aufgestellte Gesetzformeln (Paine, Freundlich und Ishizaka) bei exakterer Prüfung (Freundlich und Gann) als unhaltbar zurückgenommen[3]).

Die Erfolglosigkeit der bisherigen Versuche, auf dem empirisch-induktiven Wege zu einem Verständnis der hier geltenden Gesetze zu gelangen, kann man nun als einen Grund auffassen, einmal den um-

[1]) Vgl. z. B.: A. Galecki, Zeitschr. f. anorg. Chemie **74**, 174 (1912); Kolloid-Zeitschr. **10**, 169 (1912); A. Lottermoser, Kolloid-Zeitschr. **15**, 145 (1914); H. H Paine, Kolloidchem. Beihefte **4**, 24 (1912); Kolloid-Zeitschr. **11**, 115 (1912).

[2]) S. Miyazawa, Journ. Chem. Soc. Tokio **33**, 1179, 1210 (1912); N. Ishizaka, Zeitschr. f. physik. Chemie **83**, 97 (1913); H. Freundlich u. N. Ishizaka, ebendort **85**, 398 (1913); Kolloid-Zeitschr. **12**, 230 (1913); J. Gann, Kolloidchem. Beihefte **8**, 64 (1916).

[3]) Siehe Abschnitt VI.

Zeitschrift f. physik. Chemie. XCII. 9

1911. 2782.

PHYSIK DER ERDE

VON DR. M. P. RUDZKI

O. PROFESSOR AN DER UNIVERSITÄT KRAKAU

MIT SECHZIG ABBILDUNGEN IM TEXT UND FÜNF TAFELN

LEIPZIG 1911 · CHR. HERM. TAUCHNITZ

Smoluchowski 1918, DOI:10.1515/zpch-1918-9209

Maurycy Pius Rudzki

From Wikipedia, the free encyclopedia

Maurycy Pius Rudzki (b. 1862, d. 1916) was the first person to call himself a professor of geophysics. He held the Chair of Geophysics at the Jagiellonian University in Kraków, and established the Institute of Geophysics there in 1895. His research specialty was elastic anisotropy, as applied to wave propagation in the earth, and he established many of the fundamental results in that arena. [1]

Maurycy Pius Rudzki

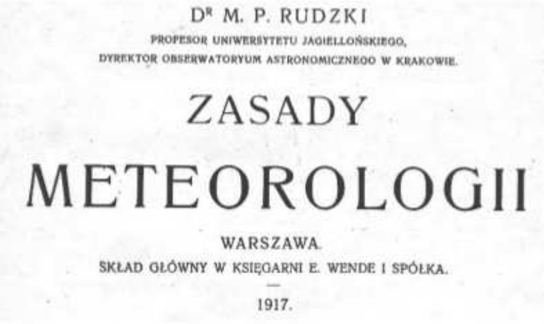
Maurycy Pius Rudzki

From Wikipedia, the free encyclopedia

Maurycy Pius Rudzki (b. 1862, d. 1916) was the first person to call himself a professor of geophysics. He held the Chair of Geophysics at the Jagiellonian University in Kraków, and established the Institute of Geophysics there in 1895. His research specialty was elastic anisotropy, as applied to wave propagation in the earth, and he established many of the fundamental results in that arena. [1]

Maurycy Pius Rudzki

"Principles of Meteorology" book (1917)



D^{R} M. P. RUDZKI

PROFESOR UNIWERSYTETU JAGIELLOŃSKIEGO,

DYREKTOR OBSERWATORYUM ASTRONOMICZNEGO W KRAKOWIE.

ZASADY

METEOROLOGII

WARSZAWA.

SKŁAD GŁÓWNY W KSIĘGARNI E. WENDE I SPÓŁKA.

—

1917.

Rudzki 1917: Principles of Meteorology

1-X-1921

Anno – Année XV. Vol. XXIX, N. CV-1 Serie II.

"SCIENTIA,,

International Review of Scientific Synthesis.

DIRETTORE - DIRECTEUR - EDITOR
EUGENIO RIGNANO

BOLOGNA
NICOLA ZANICHELLI
LONDON WILLIAMS & NORGATE
PARIS FÉLIX ALCAN

TECHNICAL BOOK REVIEW INDEX

ISSUED BY THE TECHNOLOGY DEPARTMENT OF THE CARNEGIE LIBRARY OF PITTSBURGH

Vol. 5 SEPTEMBER 1921 No. 3

Rudzki, M. P.
Zasady meteorologii. 160 p. 1917. Wende, Warsaw.
Scientia, v.29, 1921, no.5, p.389. 1¼ p.

REVUE SEMESTRIELLE

DES

PUBLICATIONS MATHÉMATIQUES

RÉDIGÉE SOUS LES AUSPICES DE LA SOCIÉTÉ MATHÉMATIQUE D'AMSTERDAM.

AMSTERDAM
DELSMAN EN NOLTHENIUS
1922

U 7. M. P. Rudzki. Zasady meteorologii (Principes de météorologie). Un vol. 8, p. 160. Varsovie, E. Wende, 1917, *Scientia*, XXIX, 1921 (p. 389—390).

---

http://pbc.gda.pl/dlibra/docmetadata?id=18434 **(translated)**

Rudzki 1917: Principles of Meteorology

NE 104

Anno – Année – Year XV. Vol. XXIX, N. CV-1 Serie – Série – Series II.

"SCIENTIA"

International Review of Scientific Synthesis.

DIRETTORE - DIRECTEUR - EDITOR
EUGENIO RIGNANO

BOLOGNA
NICOLA ZANICHELLI
LONDON
WILLIAMS & NORGATE
PARIS
FÉLIX ALCAN

TECHNICAL BOOK REVIEW INDEX

ISSUED BY THE TECHNOLOGY DEPARTMENT OF THE CARNEGIE LIBRARY OF PITTSBURGH

VOL. 5 SEPTEMBER 1921 No. 3

Rudzki, M. P.
Zasady meteorologii. 160 p. 1917. Wende, Warsaw.
Scientia, v.29, 1921, no.5, p.389. 1¼ p.

REVUE SEMESTRIELLE DES PUBLICATIONS MATHÉMATIQUES

RÉDIGÉE SOUS LES AUSPICES DE LA SOCIÉTÉ MATHÉMATIQUE D'AMSTERDAM.

AMSTERDAM
DELSMAN EN NOLTHENIUS
1922

U 7. M. P. RUDZKI. Zasady meteorologii (Principes de météorologie). Un vol. 8, p. 160. Varsovie, E. Wende, 1917. *Scientia*, XXIX, 1921 (p. 389—390).

http://pbc.gda.pl/dlibra/docmetadata?id=18434 **(translated)**

... in the atmosphere, nuclei are needed for condensation

Rudzki 1917: Principles of Meteorology

"SCIENTIA"

International Review of Scientific Synthesis.

DIRETTORE - DIRECTEUR - EDITOR

EUGENIO RIGNANO

BOLOGNA

NICOLA ZANICHELLI

LONDON

WILLIAMS & NORGATE

PARIS

FÉLIX ALCAN

TECHNICAL BOOK REVIEW INDEX

ISSUED BY THE TECHNOLOGY DEPARTMENT OF THE CARNEGIE LIBRARY OF PITTSBURGH

VOL. 5 SEPTEMBER 1921 No. 3

Rudzki, M. P.

Zasady meteorologii. 160 p. 1917. Wende, Warsaw.

Scientia, v.29, 1921, no.5, p.389. 1¼ p.

REVUE SEMESTRIELLE

DES

PUBLICATIONS MATHÉMATIQUES

RÉDIGÉE SOUS LES AUSPICES DE LA SOCIÉTÉ MATHÉMATIQUE D'AMSTERDAM.

AMSTERDAM

DELSMAN EN NOLTHENIUS

1922

U 7. M. P. Rudzki. Zasady meteorologii (Principes de météorologie). Un vol. 8, p. 160. Varsovie, E. Wende, 1917, *Scientia*, XXIX, 1921 (p. 389—390).

`http://pbc.gda.pl/dlibra/docmetadata?id=18434` **(translated)**

... in the atmosphere, nuclei are needed for condensation ... the air contains a lot of smoke, molecules of acids e.t.c.

Rudzki 1917: Principles of Meteorology

“SCIENTIA,,

International Review of Scientific Synthesis.

DIRETTORE - DIRECTEUR - EDITOR

EUGENIO RIGNANO

BOLOGNA

NICOLA ZANICHELLI

LONDON — WILLIAMS & NORGATE

PARIS — FÉLIX ALCAN

TECHNICAL BOOK REVIEW INDEX

ISSUED BY THE TECHNOLOGY DEPARTMENT OF THE CARNEGIE LIBRARY OF PITTSBURGH

VOL. 5 SEPTEMBER 1921 No. 3

Rudzki, M. P.
Zasady meteorologii. 160 p. 1917. Wende, Warsaw.
Scientia, v.29, 1921, no.5, p.389. 1¼ p.

REVUE SEMESTRIELLE

DES

PUBLICATIONS MATHÉMATIQUES

RÉDIGÉE SOUS LES AUSPICES DE LA SOCIÉTÉ MATHÉMATIQUE D'AMSTERDAM.

AMSTERDAM

DELSMAN EN NOLTHENIUS

1922

U 7. M. P. RUDZKI. Zasady meteorologii (Principes de météorologie). Un vol. 8, p. 160. Varsovie, E. Wende, 1917, *Scientia*, XXIX, 1921 (p. 389—390).

`http://pbc.gda.pl/dlibra/docmetadata?id=18434` **(translated)**

... in the atmosphere, nuclei are needed for condensation ... the air contains a lot of smoke, molecules of acids e.t.c. ... all these are hygroscopic bodies that attract vapour even when the air is not saturated yet

Rudzki 1917: Principles of Meteorology

Anno – Année – Year XV. Vol. XXIX, N. CV-1 Series II.

"SCIENTIA"

International Review of Scientific Synthesis.

DIRETTORE - DIRECTEUR - EDITOR
EUGENIO RIGNANO

BOLOGNA
NICOLA ZANICHELLI

LONDON WILLIAMS & NORGATE — PARIS FÉLIX ALCAN

TECHNICAL BOOK REVIEW INDEX

ISSUED BY THE TECHNOLOGY DEPARTMENT OF THE CARNEGIE LIBRARY OF PITTSBURGH

Vol. 5 — SEPTEMBER 1921 — No. 3

Rudzki, M. P.
Zasady meteorologii. 160 p. 1917. Wende, Warsaw.
Scientia, v.29, 1921, no.5, p.389. 1¼ p.

REVUE SEMESTRIELLE DES PUBLICATIONS MATHÉMATIQUES

RÉDIGÉE SOUS LES AUSPICES DE LA SOCIÉTÉ MATHÉMATIQUE D'AMSTERDAM.

AMSTERDAM
DELSMAN EN NOLTHENIUS
1922

U 7. M. P. Rudzki. Zasady meteorologii (Principes de météorologie). Un vol. 8, p. 160. Varsovie, E. Wende, 1917, *Scientia*, XXIX, 1921 (p. 389—390).

`http://pbc.gda.pl/dlibra/docmetadata?id=18434` **(translated)**

... in the atmosphere, nuclei are needed for condensation ... the air contains a lot of smoke, molecules of acids e.t.c. ... all these are hygroscopic bodies that attract vapour even when the air is not saturated yet ...
as rightly pointed out by Smoluchowski, usually it is not a single drop that falls but a whole plenty

Rudzki 1917: Principles of Meteorology

1-X-1921

Anno – Année – Year XV. Vol. XXIX, N. CV-5 Serie II.

NE 104

“SCIENTIA,,

International Review of Scientific Synthesis.

DIRETTORE - DIRECTEUR - EDITOR
EUGENIO RIGNANO

BOLOGNA
NICOLA ZANICHELLI

LONDON
WILLIAMS & NORGATE

PARIS
FÉLIX ALCAN

TECHNICAL BOOK REVIEW INDEX

ISSUED BY THE TECHNOLOGY DEPARTMENT OF THE CARNEGIE LIBRARY OF PITTSBURGH

VOL. 5 SEPTEMBER 1921 No. 3

Rudzki, M. P.
Zasady meteorologii. 160 p. 1917. Wende, Warsaw.
Scientia, v.29, 1921, no.5, p.389. 1¼ p.

REVUE SEMESTRIELLE
DES
PUBLICATIONS MATHÉMATIQUES

RÉDIGÉE SOUS LES AUSPICES DE LA SOCIÉTÉ MATHÉMATIQUE D'AMSTERDAM.

AMSTERDAM
DELSMAN EN NOLTHENIUS
1922

U 7. M. P. RUDZKI. Zasady meteorologii (Principes de météorologie). Un vol. 8, p. 160. Varsovie, E. Wende, 1917, *Scientia*, XXIX, 1921 (p. 389—390).

`http://pbc.gda.pl/dlibra/docmetadata?id=18434` **(translated)**

... in the atmosphere, nuclei are needed for condensation ... the air contains a lot of smoke, molecules of acids e.t.c. ... all these are hygroscopic bodies that attract vapour even when the air is not saturated yet ... as rightly pointed out by Smoluchowski, usually it is not a single drop that falls but a whole plenty ... contrast between the sizes of drops, of which clouds are made up, and the size of raindrops, is so great that the latter, of course, can not come straight from the condensation, only from the merging of many small droplets ...

Rudzki 1917: Principles of Meteorology

"SCIENTIA,,

International Review of Scientific Synthesis

DIRETTORE - DIRECTEUR - EDITOR
EUGENIO RIGNANO

BOLOGNA
NICOLA ZANICHELLI

LONDON
WILLIAMS & NORGATE

PARIS
FÉLIX ALCAN

TECHNICAL BOOK REVIEW INDEX

ISSUED BY THE TECHNOLOGY DEPARTMENT OF THE CARNEGIE LIBRARY OF PITTSBURGH

VOL. 5 SEPTEMBER 1921 No. 3

Rudzki, M. P.
Zasady meteorologii. 160 p. 1917. Wende, Warsaw.
Scientia, v.29, 1921, no.5, p.389. 1¼ p.

REVUE SEMESTRIELLE DES PUBLICATIONS MATHÉMATIQUES

RÉDIGÉE SOUS LES AUSPICES DE LA SOCIÉTÉ MATHÉMATIQUE D'AMSTERDAM.

AMSTERDAM
DELSMAN EN NOLTHENIUS
1922

U 7. M. P. RUDZKI. Zasady meteorologii (Principes de météorologie). Un vol. 8, p. 160. Varsovie, E. Wende, 1917. *Scientia*, XXIX, 1921 (p. 389—390).

`http://pbc.gda.pl/dlibra/docmetadata?id=18434` **(translated)**

... in the atmosphere, nuclei are needed for condensation ... the air contains a lot of smoke, molecules of acids e.t.c. ... all these are hygroscopic bodies that attract vapour even when the air is not saturated yet ... as rightly pointed out by Smoluchowski, usually it is not a single drop that falls but a whole plenty ... contrast between the sizes of drops, of which clouds are made up, and the size of raindrops, is so great that the latter, of course, can not come straight from the condensation, only from the merging of many small droplets ...

Rudzki 1917: Principles of Meteorology

"SCIENTIA"

International Review of Scientific Synthesis.

BOLOGNA
NICOLA ZANICHELLI

LONDON
WILLIAMS & NORGATE

PARIS
FÉLIX ALCAN

TECHNICAL BOOK REVIEW INDEX

ISSUED BY THE TECHNOLOGY DEPARTMENT OF THE CARNEGIE LIBRARY OF PITTSBURGH

VOL. 5 SEPTEMBER 1921 No. 3

Rudzki, M. P.
Zasady meteorologii. 160 p. 1917. Wende, Warsaw.
Scientia, v.29, 1921, no.5, p.389. 1¼ p.

REVUE SEMESTRIELLE DES PUBLICATIONS MATHÉMATIQUES

RÉDIGÉE SOUS LES AUSPICES DE LA SOCIÉTÉ MATHÉMATIQUE D'AMSTERDAM.

AMSTERDAM
DELSMAN EN NOLTHENIUS
1922

U 7. M. P. RUDZKI. Zasady meteorologii (Principes de météorologie). Un vol. 8, p. 160. Varsovie, E. Wende, 1917. *Scientia*, XXIX, 1921 (p. 389—390).

http://pbc.gda.pl/dlibra/docmetadata?id=18434 **(translated)**

... in the atmosphere, nuclei are needed for condensation ... the air contains a lot of smoke, molecules of acids e.t.c. ... all these are hygroscopic bodies that attract vapour even when the air is not saturated yet ... as rightly pointed out by Smoluchowski, usually it is not a single drop that falls but a whole plenty ... contrast between the sizes of drops, of which clouds are made up, and the size of raindrops, is so great that the latter, of course, can not come straight from the condensation, only from the merging of many small droplets ...

modelling coagulation: SCE & SDM

Smoluchowski's coagulation equation (SCE)

droplet concentration (continuous): $c(x, t) : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$

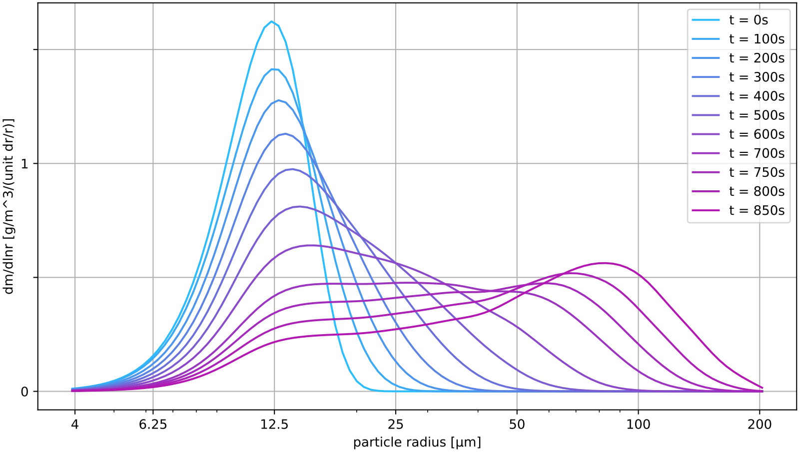
collision kernel: $a(x_1, x_2) : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$

$$\dot{c}(x) = \frac{1}{2} \int_0^x a(y, x-y) c(y) c(x-y) dy - \int_0^\infty a(y, x) c(y) c(x) dy$$

droplet concentration (discrete): $c_i = c(x_i)$

$$\dot{c}_i = \frac{1}{2} \sum_{k=1}^{i-1} a(x_k, x_{i-k}) c_k c_{i-k} - \sum_{k=1}^{\infty} a(x_k, x_i) c_k c_i$$

cloud droplet collisional growth

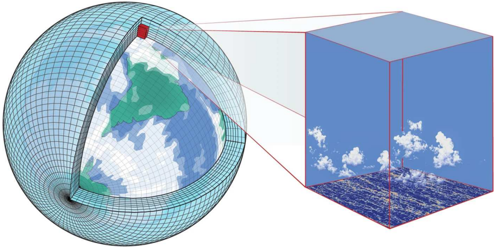


“Cloud and ship. Ukraine, Crimea, Black sea, view from Ai-Petri mountain”

(photo: Yevgen Timashov / National Geographic)

"Cloud and ship. Ukraine, Crimea, Black sea, view from Ai-Petri mountain"

(photo: Yevgen Timashov / National Geographic)



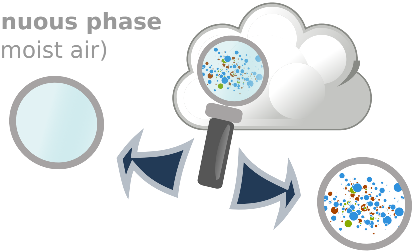
"Grid cells in a global climate model and a large-eddy simulation of shallow cumulus clouds at 5 m resolution"

(fig. from Schneider et al. 2017)

Eulerian vs. Lagrangian microphysics

Eulerian vs. Lagrangian microphysics

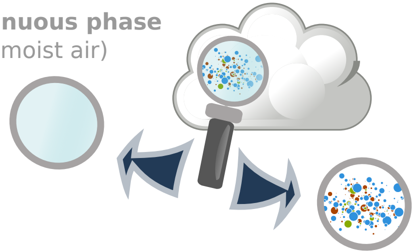
continuous phase
(moist air)



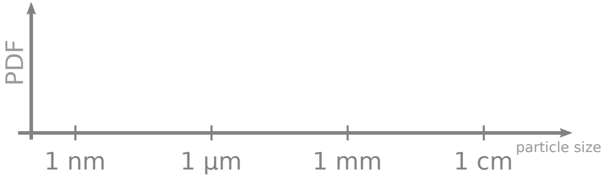
dispersed phase
(aerosol particles, cloud droplets, drizzle, rain, snow, ...)

Eulerian vs. Lagrangian microphysics

continuous phase
(moist air)

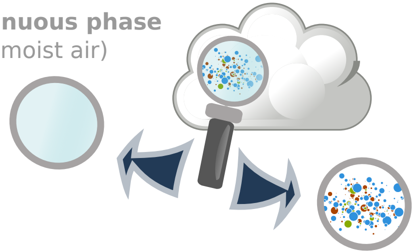


dispersed phase
(aerosol particles, cloud droplets, drizzle, rain, snow, ...)

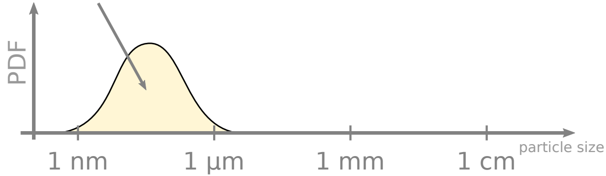


Eulerian vs. Lagrangian microphysics

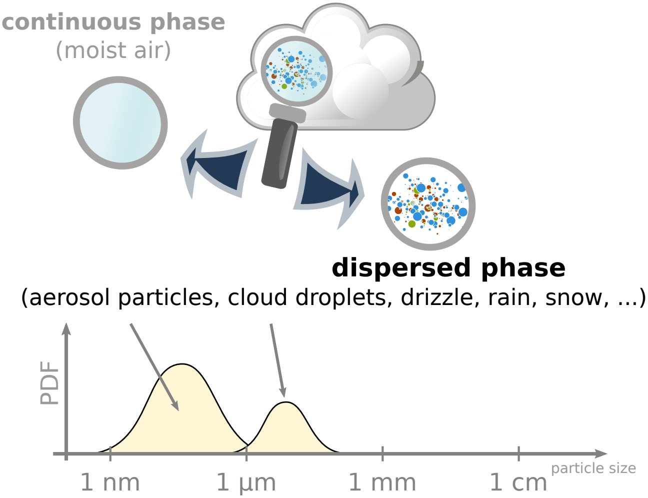
continuous phase
(moist air)



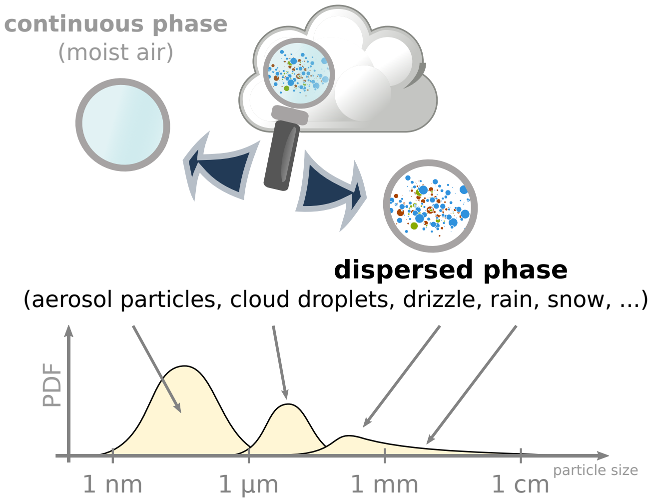
dispersed phase
(aerosol particles, cloud droplets, drizzle, rain, snow, ...)



Eulerian vs. Lagrangian microphysics

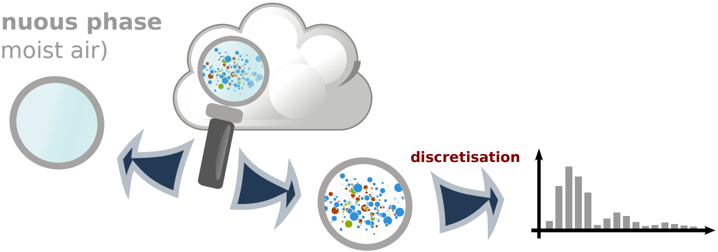


Eulerian vs. Lagrangian microphysics

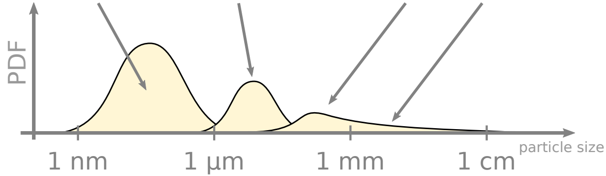


Eulerian vs. Lagrangian microphysics

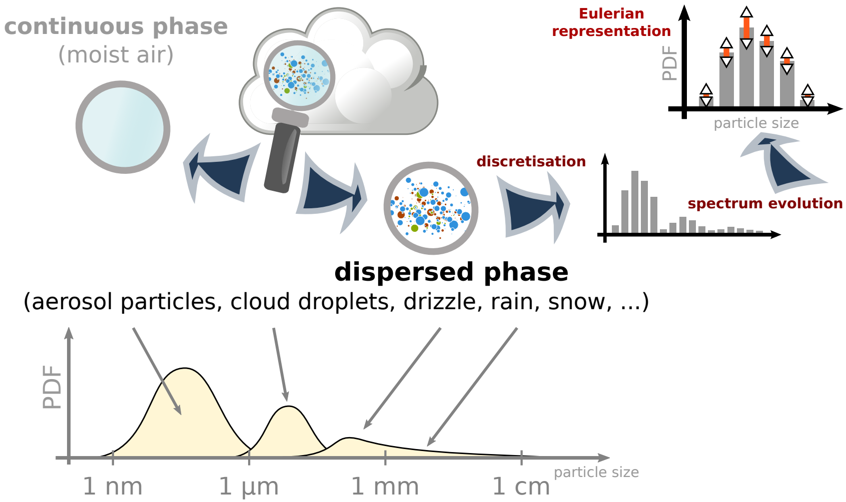
continuous phase
(moist air)



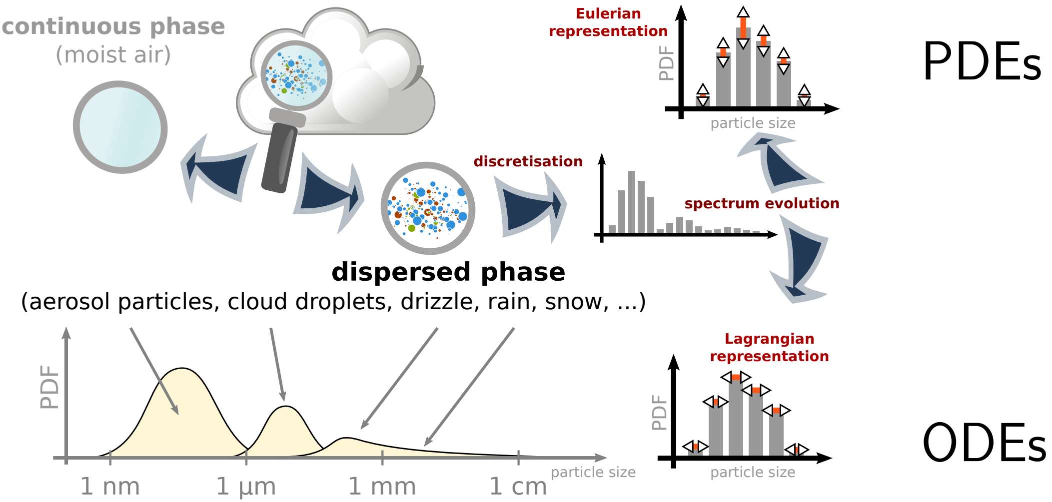
dispersed phase
(aerosol particles, cloud droplets, drizzle, rain, snow, ...)



Eulerian vs. Lagrangian microphysics



Eulerian vs. Lagrangian microphysics



Lagrangian microphysics: early works (0D)

JOURNAL OF METEOROLOGY

THE GROWTH OF CLOUD DROPS IN UNIFORMLY COOLED AIR

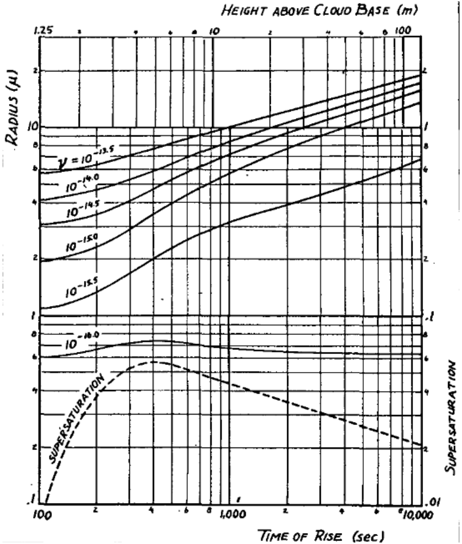
By Wallace E. Howell[1]

Blue Hill Meteorological Observatory, Harvard University[2]

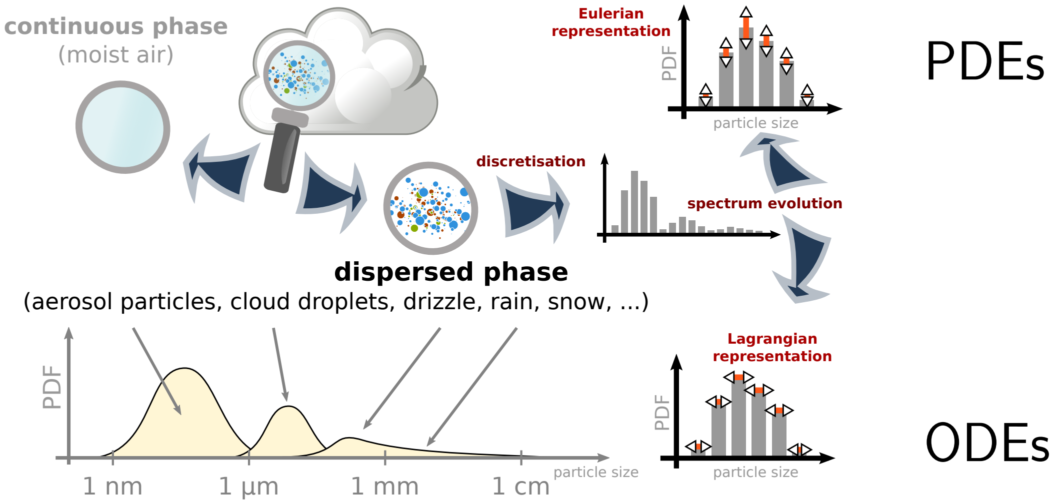
(Manuscript received 10 June 1948)

ABSTRACT

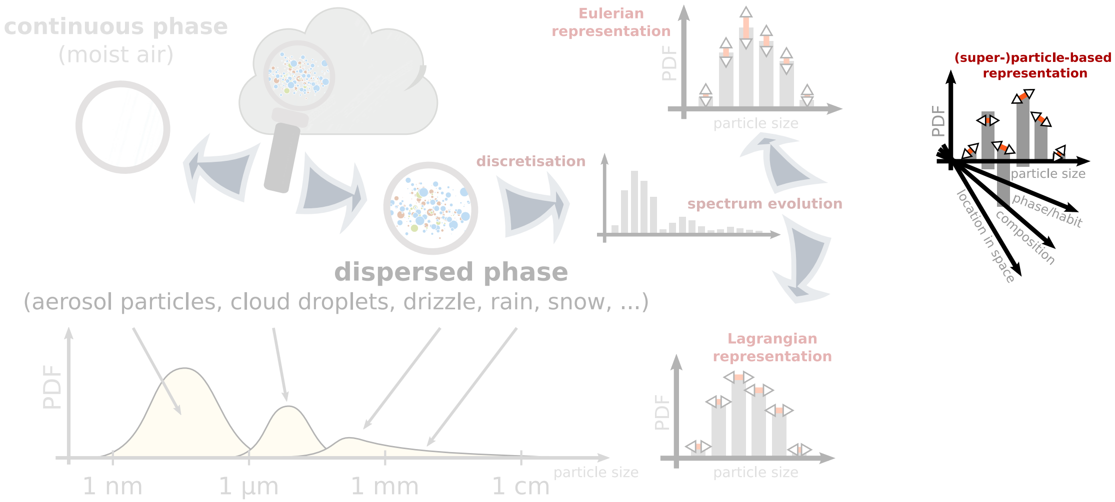
Recent studies of precipitation, aircraft icing, and visibility through fog have focussed attention on the physical constitution of clouds, a subject to which knowledge of the drop-size spectrum and its origin would be an important contribution. The drop-size spectrum resulting when air containing condensation nuclei is uniformly cooled may be computed, leading to a differential equation for the growth of a cloud drop which cannot be integrated analytically. A numerical method of integration is therefore employed.



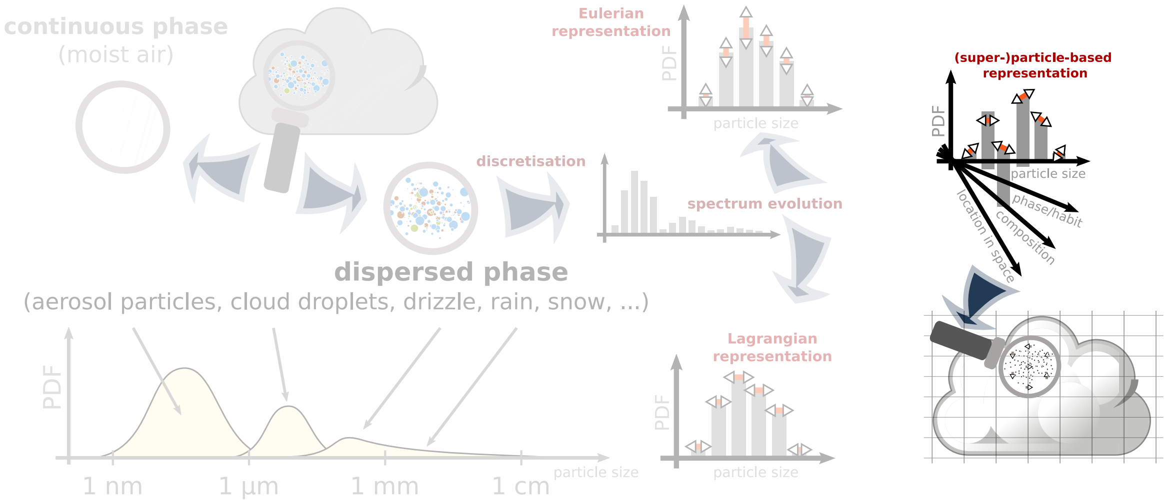
Eulerian vs. Lagrangian microphysics



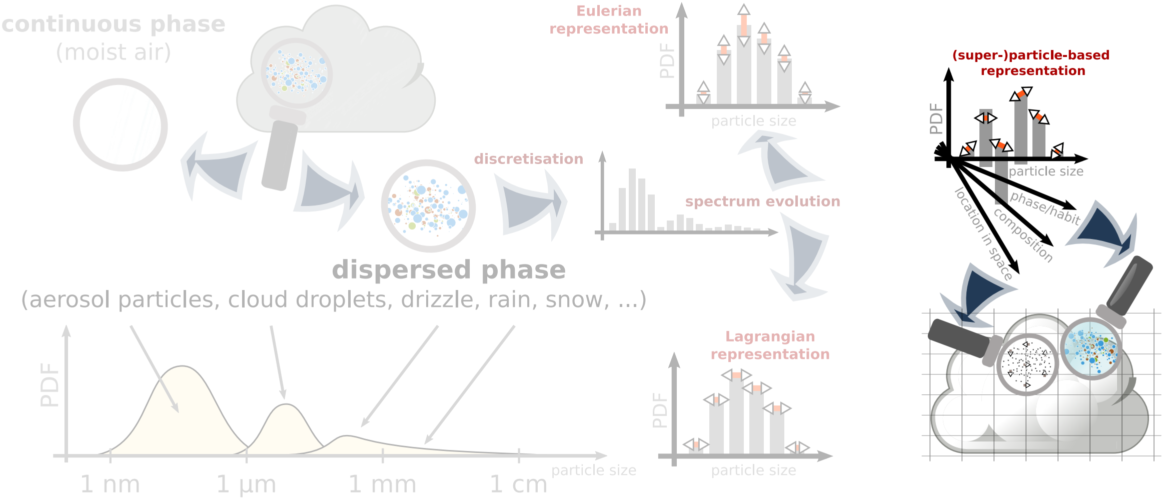
Eulerian vs. Lagrangian microphysics



Eulerian vs. Lagrangian microphysics



Eulerian vs. Lagrangian microphysics



Lagrangian microphysics: early works (3D)

JOURNAL OF THE ATMOSPHERIC SCIENCES

A Numerical Experiment on Stochastic Condensation Theory

TERRY L. CLARK AND W. D. HALL

National Center for Atmospheric Research,[1] *Boulder, CO 80307*

(Manuscript received 30 August 1978, in final form 20 November 1978)

ABSTRACT

A three-dimensional numerical model is used to study the effect of small-scale supersaturation fluctuations on the evolving droplet distribution in the first 150 m above cloud base. The primary purpose of this research is to determine whether the irreversible coupling between the thermodynamics and dynamics due to finite phase relaxation time scales τ_S is sufficient to produce significant small-scale horizontal variations in supersaturation. Thus, the paper is concerned only with this internal source for thermodynamic variability. All other source terms, such as the downgradient flux of the variance of thermodynamic fields, have purposely been neglected.

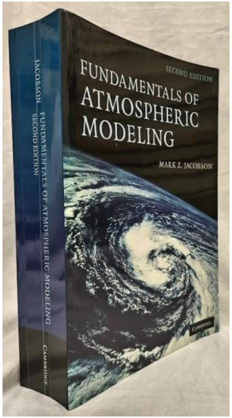
Lagrangian particle experiments were run in parallel with the basic Eulerian model. The purpose of these experiments is to relax some of the microphysical parameterization assumptions with respect to assumed distribution shape and as a result add credibility to the results of distribution broadening.

Eulerian vs. Lagrangian microphysics: a (probabilistic) breakthrough

pre-2009:

„advantage of the full-moving size structure is that core particle material is preserved during growth ... second advantage ... it eliminates numerical diffusion ... [but] nucleation, coagulation ... cause problems ... the full-moving structure is not used in three-dimensional models"[a]

„the use of a fixed grid allows for an easy implementation of collision processes, which is not possible for a moving grid (Lagrangian) approach"[b]



[a] Jacobson 2005: Fundamentals of Atmospheric Modeling

[b] Simmel & Wurzler 2006: Condensation and activation in sectional cloud microphysical models

Eulerian vs. Lagrangian microphysics: a (probabilistic) breakthrough

pre-2009:

„advantage of the full-moving size structure is that core particle material is preserved during growth … second advantage … it eliminates numerical diffusion … [but] nucleation, coagulation … cause problems … the full-moving structure is not used in three-dimensional models"[a]

„the use of a fixed grid allows for an easy implementation of collision processes, which is not possible for a moving grid (Lagrangian) approach"[b]

Shima 2009: Monte-Carlo particle-based collision algorithm for cloud simulations

[a] Jacobson 2005: Fundamentals of Atmospheric Modeling

[b] Simmel & Wurzler 2006: Condensation and activation in sectional cloud microphysical models

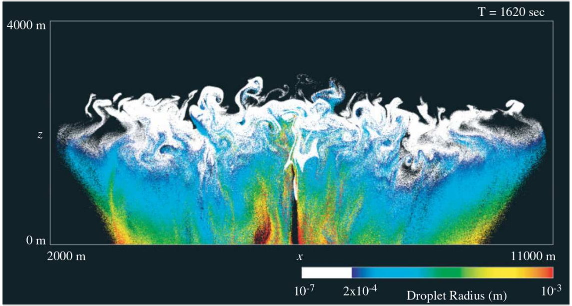
Eulerian vs. Lagrangian microphysics: a (probabilistic) breakthrough

pre-2009:

„advantage of the full-moving size structure is that core particle material is preserved during growth ... second advantage ... it eliminates numerical diffusion ... [but] nucleation, coagulation ... cause problems ... the full-moving structure is not used in three-dimensional models"[a]

„the use of a fixed grid allows for an easy implementation of collision processes, which is not possible for a moving grid (Lagrangian) approach"[b]

Shima 2009: Monte-Carlo particle-based collision algorithm for cloud simulations



Super-droplet simulation of a shallow convective cloud (figure: Shima et al. 2009, QJRMS)

[a] Jacobson 2005: Fundamentals of Atmospheric Modeling

[b] Simmel & Wurzler 2006: Condensation and activation in sectional cloud microphysical models

SCE (naïve impl)	SDM
method type	
mean-field, deterministic	Monte-Carlo, stochastic

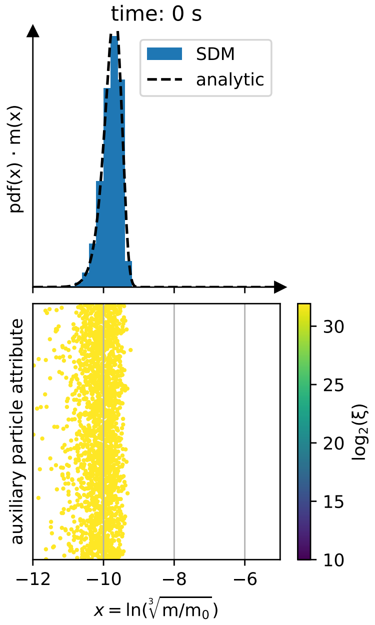
SCE (naïve impl)	SDM
method type	
mean-field, deterministic	Monte-Carlo, stochastic
considered pairs	
all (i,j) pairs	random set of $n_{sd}/2$ non-overlapping pairs, probability up-scaled by $(n_{sd}^2 - n_{sd})/2$ to $n_{sd}/2$ ratio

SCE (naïve impl)	**SDM**
method type	
mean-field, deterministic	Monte-Carlo, stochastic
considered pairs	
all (i,j) pairs	random set of $n_{sd}/2$ non-overlapping pairs, probability up-scaled by $(n_{sd}^2 - n_{sd})/2$ to $n_{sd}/2$ ratio
computation complexity	
$\mathcal{O}(n_{sd}^2)$	$\mathcal{O}(n_{sd})$ (and embarrassingly parallel)

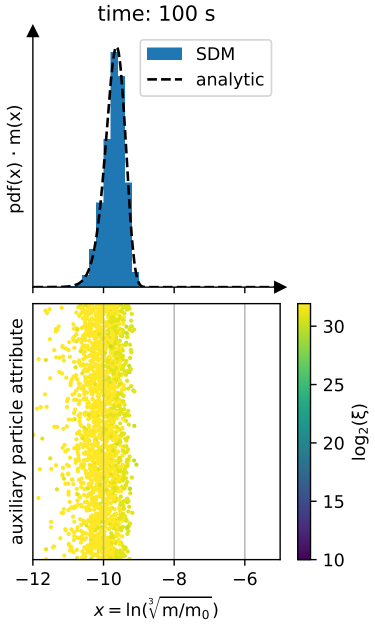
SCE (naïve impl)	SDM
method type	
mean-field, deterministic	Monte-Carlo, stochastic
considered pairs	
all (i,j) pairs	random set of $n_{sd}/2$ non-overlapping pairs, probability up-scaled by $(n_{sd}^2 - n_{sd})/2$ to $n_{sd}/2$ ratio
computation complexity	
$\mathcal{O}(n_{sd}^2)$	$\mathcal{O}(n_{sd})$ (and embarrassingly parallel)
collisions triggered	
every time step	by comparing probability with a random number

SCE (naïve impl)	SDM
method type	
mean-field, deterministic	Monte-Carlo, stochastic
considered pairs	
all (i,j) pairs	random set of $n_{sd}/2$ non-overlapping pairs, probability up-scaled by $(n_{sd}^2 - n_{sd})/2$ to $n_{sd}/2$ ratio
computation complexity	
$\mathcal{O}(n_{sd}^2)$	$\mathcal{O}(n_{sd})$ (and embarrassingly parallel)
collisions triggered	
every time step	by comparing probability with a random number
collisions	
colliding a fraction of $\xi_{[i]}$, $\xi_{[j]}$	collide all of min$\{\xi_{[i]}$, $\xi_{[j]}\}$ ("all or nothing")

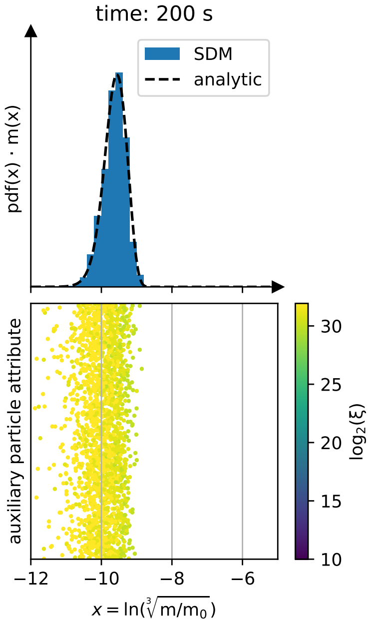
SCE (naïve impl)	SDM
method type	
mean-field, deterministic	Monte-Carlo, stochastic
considered pairs	
all (i,j) pairs	random set of $n_{sd}/2$ non-overlapping pairs, probability up-scaled by $(n_{sd}^2 - n_{sd})/2$ to $n_{sd}/2$ ratio
computation complexity	
$\mathcal{O}(n_{sd}^2)$	$\mathcal{O}(n_{sd})$ (and embarrassingly parallel)
collisions triggered	
every time step	by comparing probability with a random number
collisions	
colliding a fraction of $\xi_{[i]}$, $\xi_{[j]}$	collide all of min$\{\xi_{[i]}$, $\xi_{[j]}\}$ ("all or nothing")
interpretation	
concentration "c_i" in size bin "i"	besides c_i, each "particle" i carries other physicochemical attributes, e.g. position (x_i, y_i, z_i)



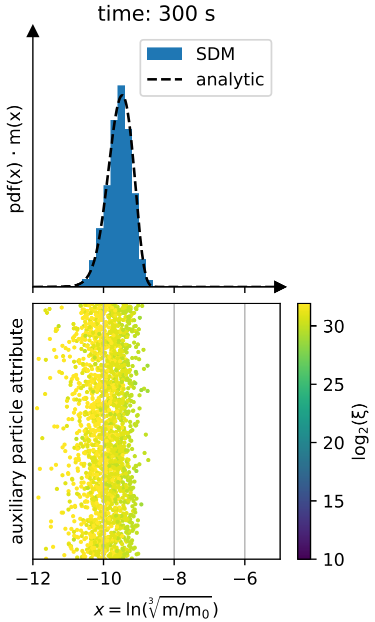

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



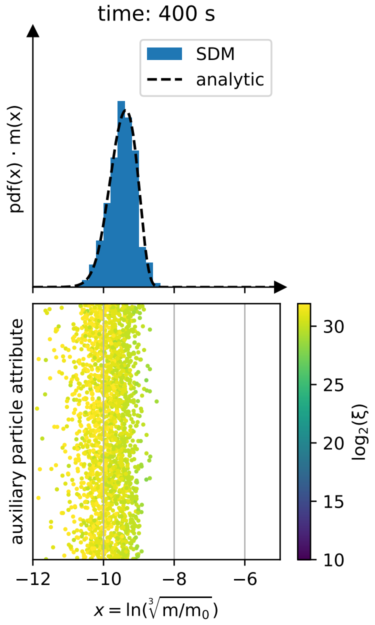

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



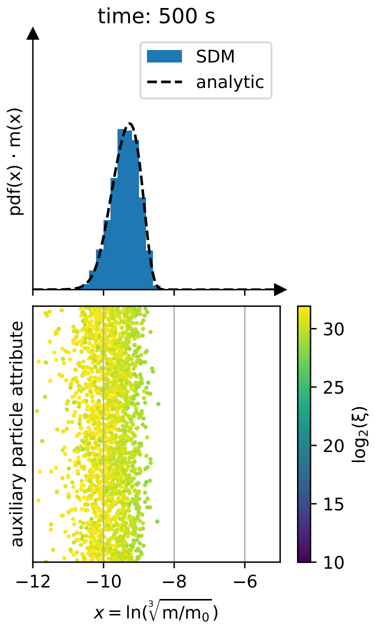

```python
def sdm(*,
        rng: np.random.Generator,
        ξ: abc.MutableSequence[int],
        m: abc.MutableSequence[float],
        kern: abc.Callable[[float, float], float],
        Δt: float,
        Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



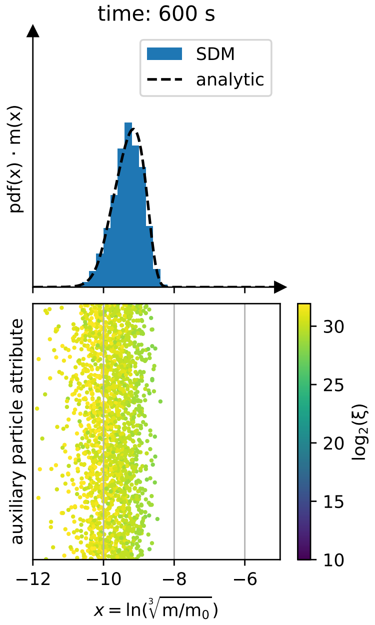

```
def sdm(*,
        rng: np.random.Generator,
        ξ: abc.MutableSequence[int],
        m: abc.MutableSequence[float],
        kern: abc.Callable[[float, float], float],
        Δt: float,
        Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



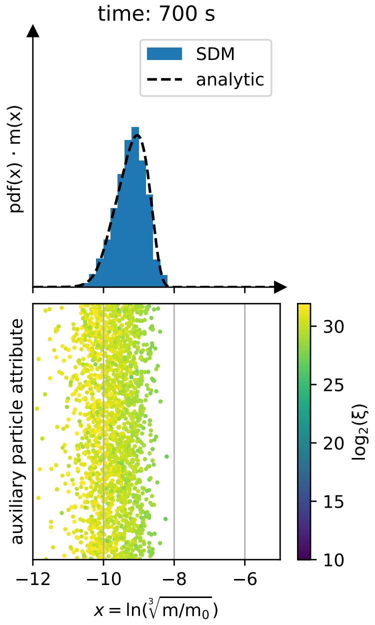

```python
def sdm(*,
        rng: np.random.Generator,
        ξ: abc.MutableSequence[int],
        m: abc.MutableSequence[float],
        kern: abc.Callable[[float, float], float],
        Δt: float,
        Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



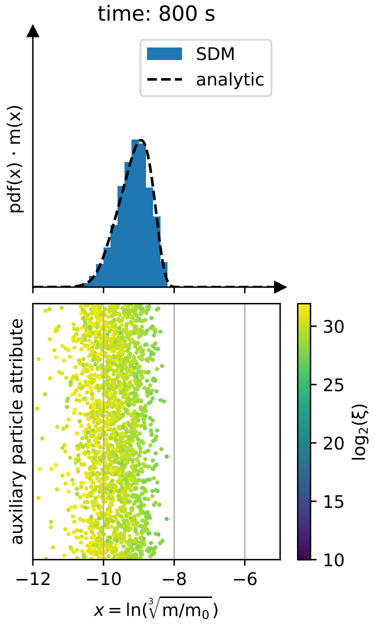

```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



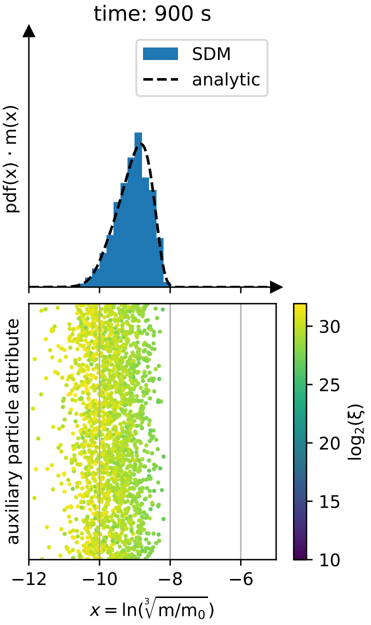

```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



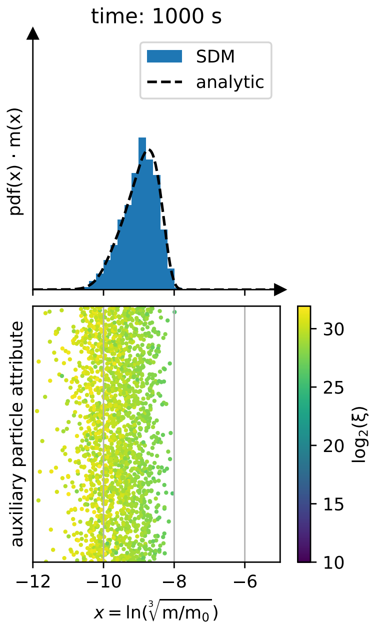

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



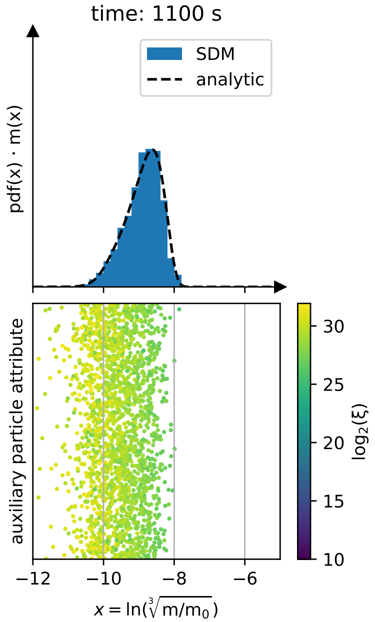

```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



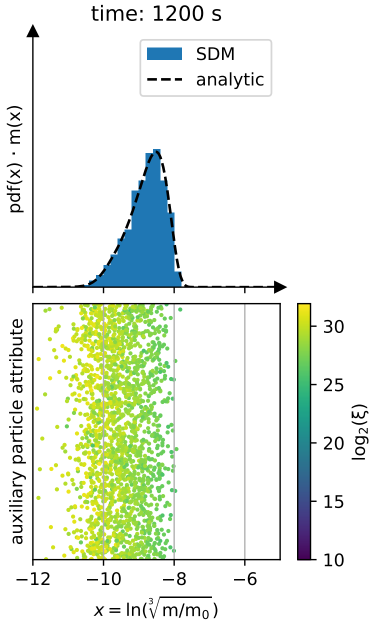

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



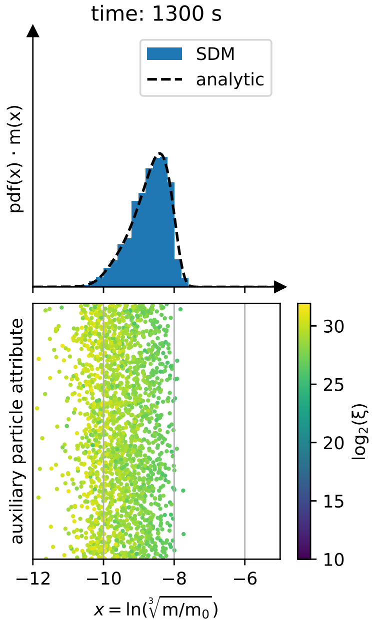

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



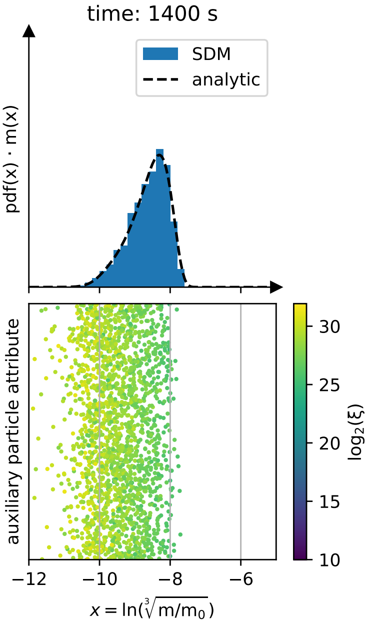

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



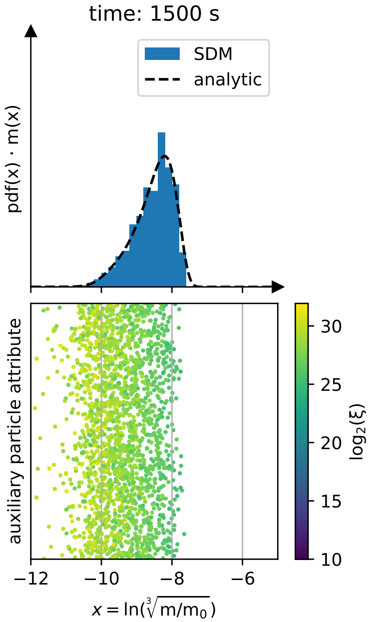

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



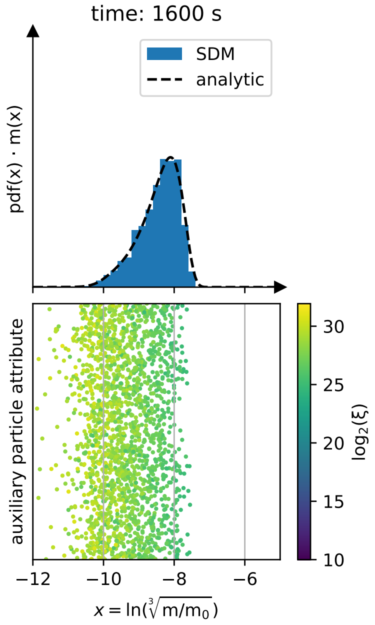

```python
def sdm(*,
        rng: np.random.Generator,
        ξ: abc.MutableSequence[int],
        m: abc.MutableSequence[float],
        kern: abc.Callable[[float, float], float],
        Δt: float,
        Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



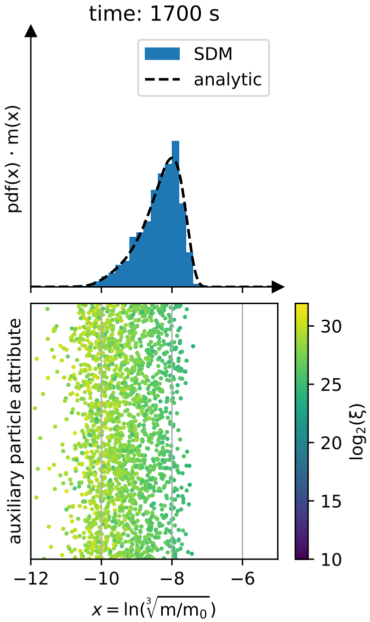

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    φ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > φ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



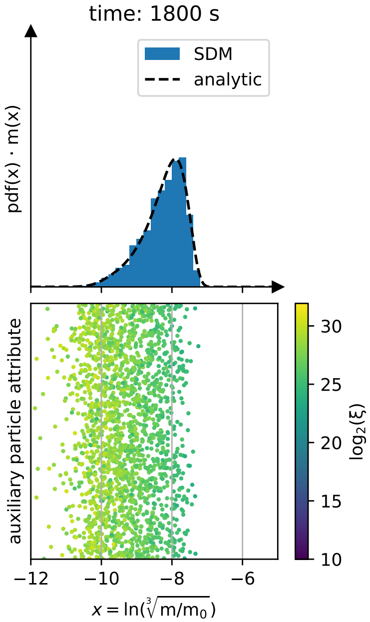

```python
def sdm(*,
        rng: np.random.Generator,
        ξ: abc.MutableSequence[int],
        m: abc.MutableSequence[float],
        kern: abc.Callable[[float, float], float],
        Δt: float,
        Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



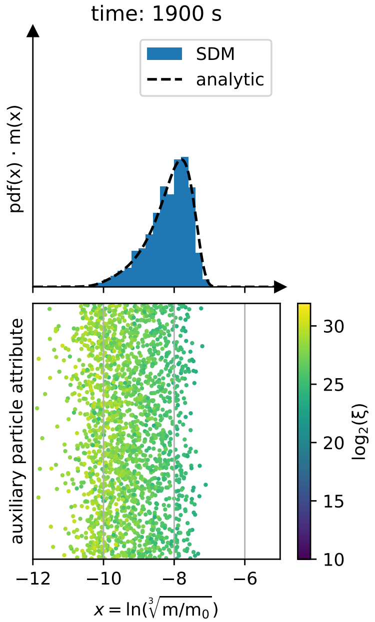

```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



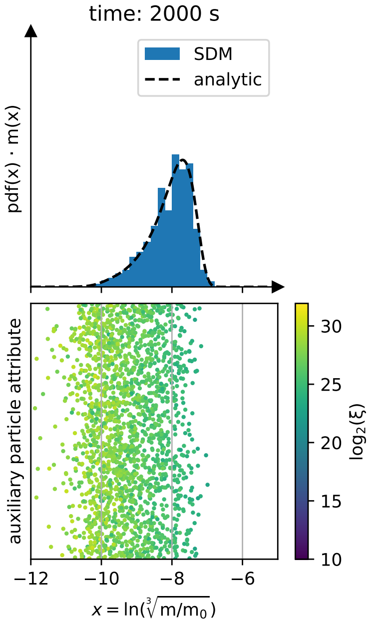

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



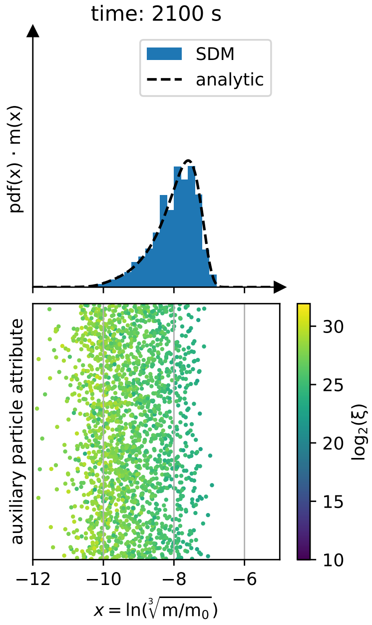

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



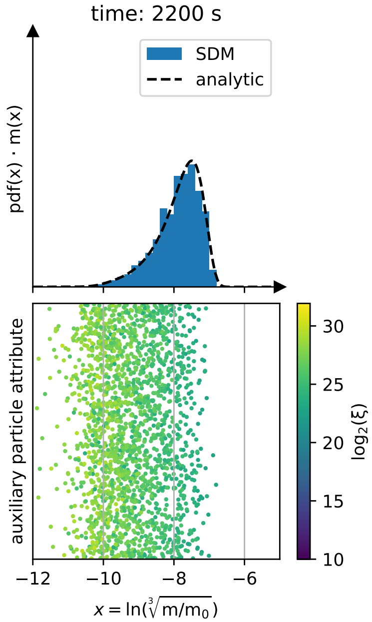

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



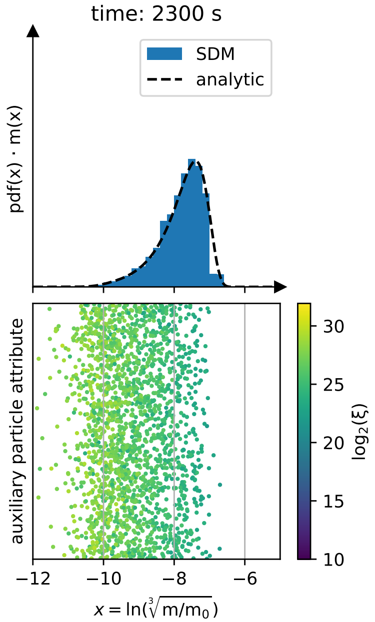

```
def sdm(*,
        rng: np.random.Generator,
        ξ: abc.MutableSequence[int],
        m: abc.MutableSequence[float],
        kern: abc.Callable[[float, float], float],
        Δt: float,
        Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



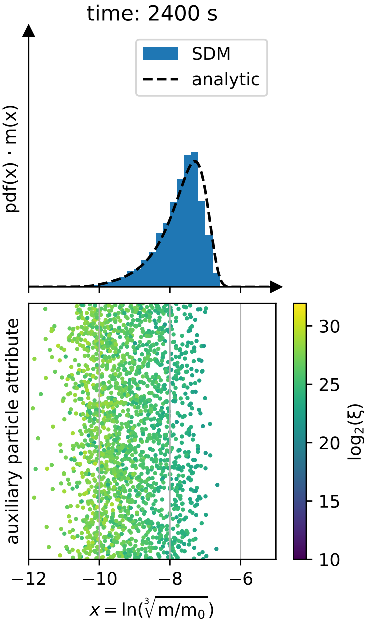

```python
def sdm(*,
        rng: np.random.Generator,
        ξ: abc.MutableSequence[int],
        m: abc.MutableSequence[float],
        kern: abc.Callable[[float, float], float],
        Δt: float,
        Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



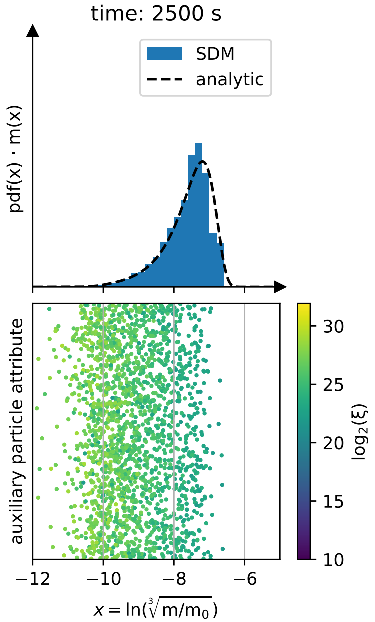

```python
def sdm(*,
        rng: np.random.Generator,
        ξ: abc.MutableSequence[int],
        m: abc.MutableSequence[float],
        kern: abc.Callable[[float, float], float],
        Δt: float,
        Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



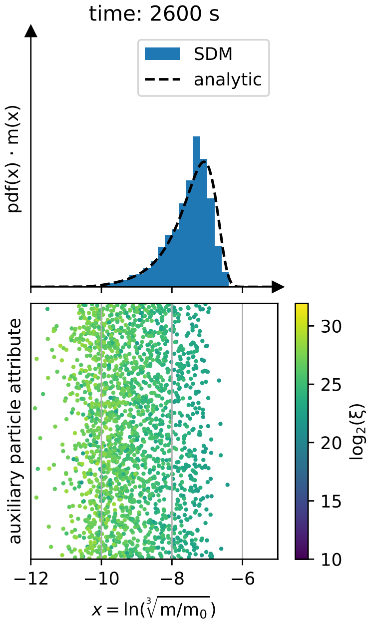

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



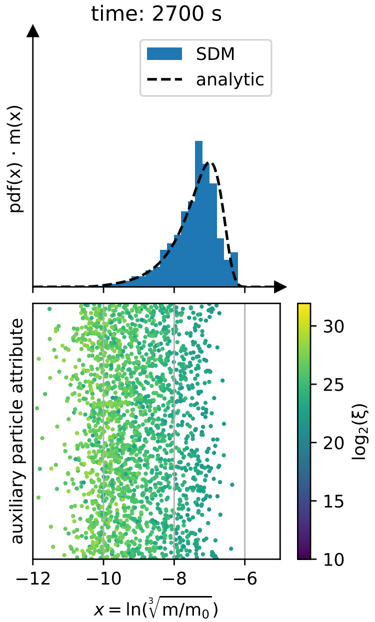

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



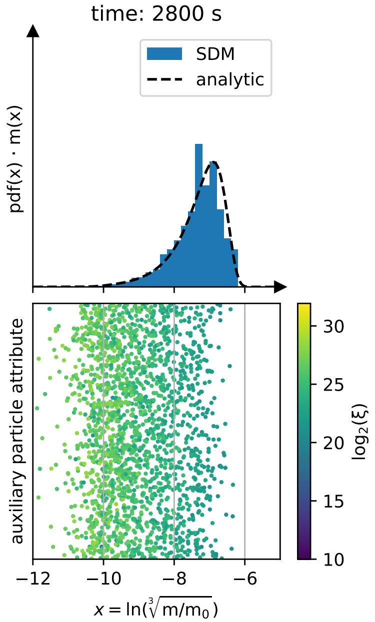

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



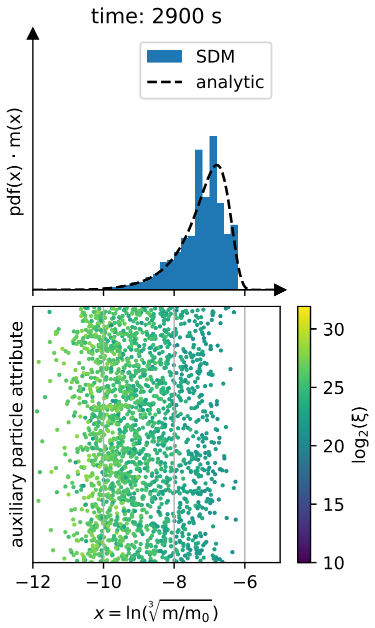

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



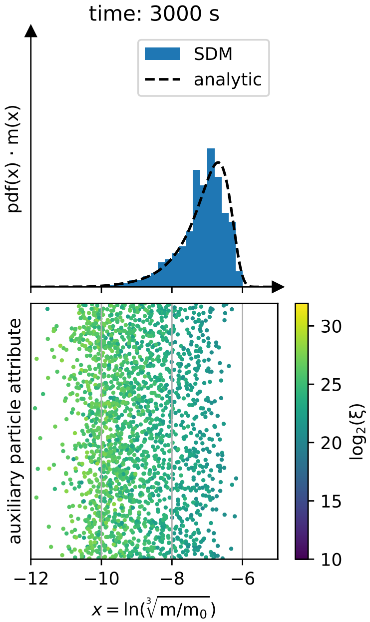

```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



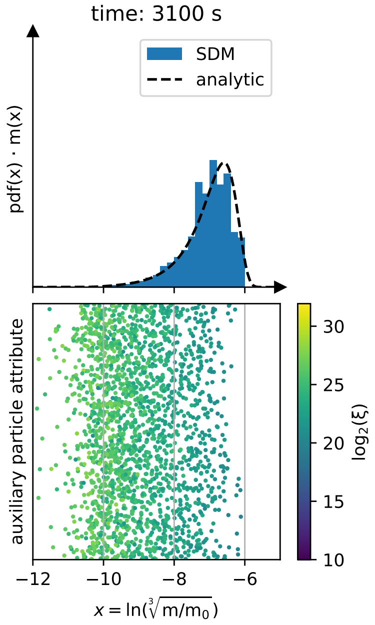

```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



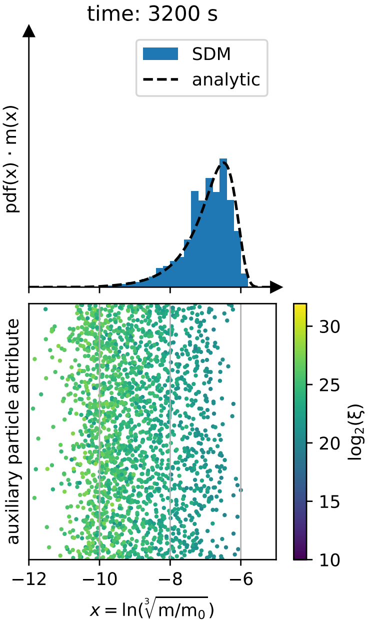

```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



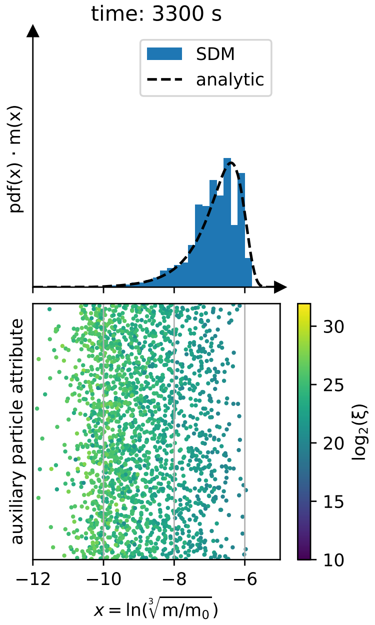

```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



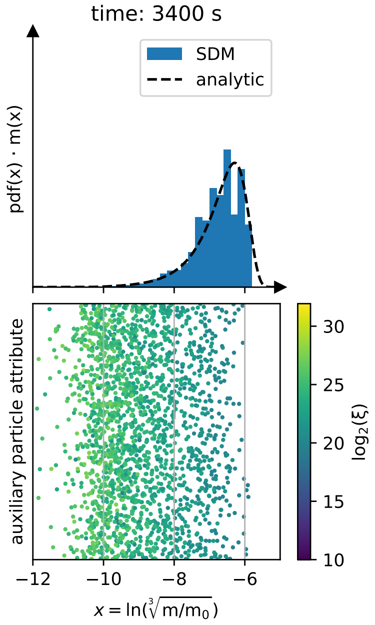

```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



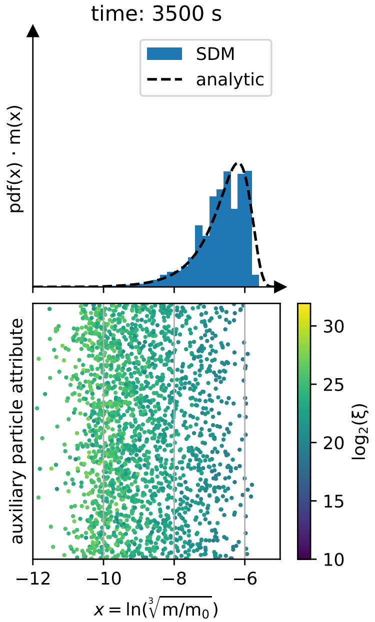

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```



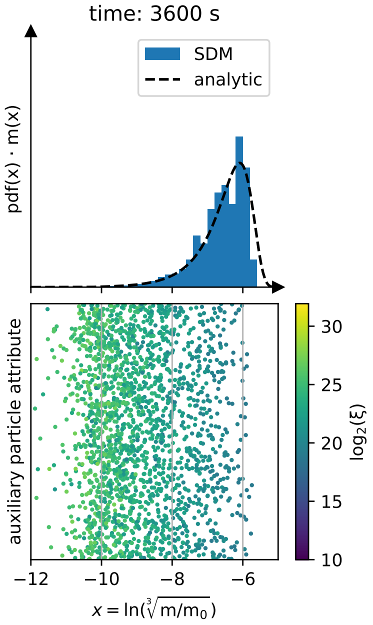

```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```




```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```




```
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```




```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = rng.permutation(n_s)[: 2 * n_pair]
    ϕ = rng.uniform(0, 1, n_pair)
    p_ratio = n_s * (n_s - 1) / 2 / n_pair
    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
        p_jk = kern(m[j], m[k]) * Δt / Δv
        if ξ[j] < ξ[k]:
            j, k = k, j
        p_α = ξ[j] * p_ratio * p_jk
        γ = p_α // 1 + (p_α - p_α // 1) > ϕ[α]
        if γ != 0:
            γ = min(γ, (ξ[j] / ξ[k]) // 1)
            if ξ[j] - γ * ξ[k] > 0:
                ξ[j] -= γ * ξ[k]
                m[k] += γ * m[j]
            else:
                ξ[j] = ξ[k] // 2
                ξ[k] -= ξ[j]
                m[k] += γ * m[j]
                m[j] = m[k]
```

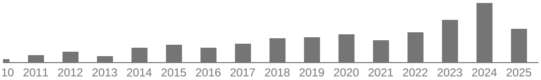
Shima et al. 2009 algorithm (BTW, Shima's PhD advisor was Yoshiki Kuramoto)

Shin-ichiro Shima

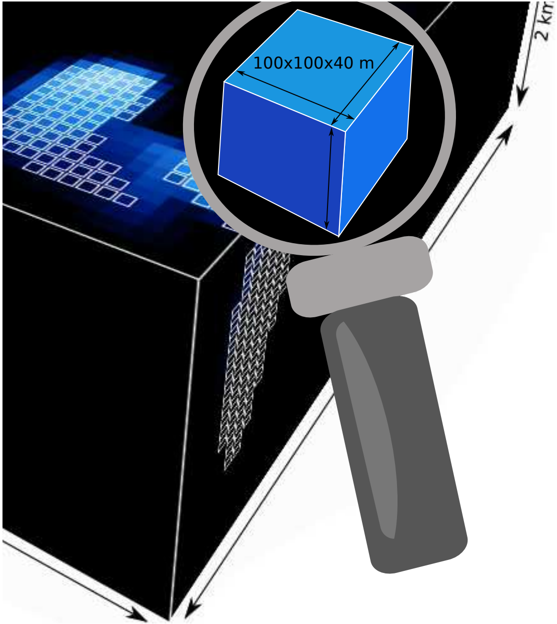
The super-droplet method for the numerical simulation of clouds and precipitation: a particle-based and probabilistic microphysics model coupled with a non-hydrostatic model

[PDF] from arxiv.org
Get this item at AGH

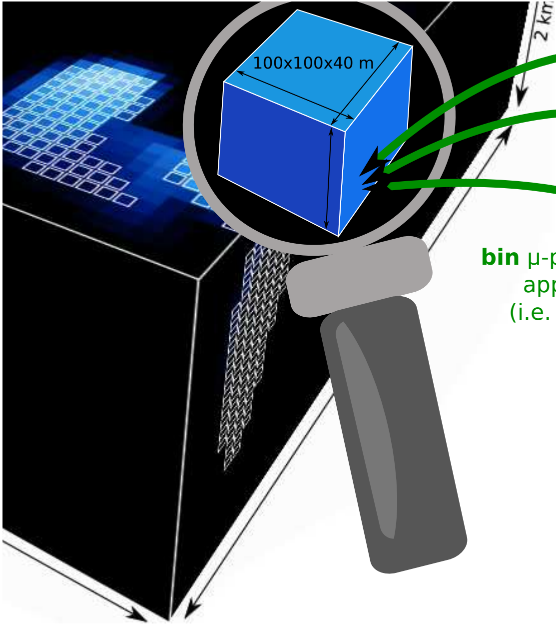
Authors	S Shima, Kanya Kusano, Akio Kawano, Tooru Sugiyama, Shintaro Kawahara
Publication date	2009/7/1
Journal	Quarterly Journal of the Royal Meteorological Society
Volume	135
Issue	642
Pages	1307-1320
Total citations	Cited by 309



Arabas & Shima 2013: SDM in LES


100x100x40 m
2 km

inside each grid box:

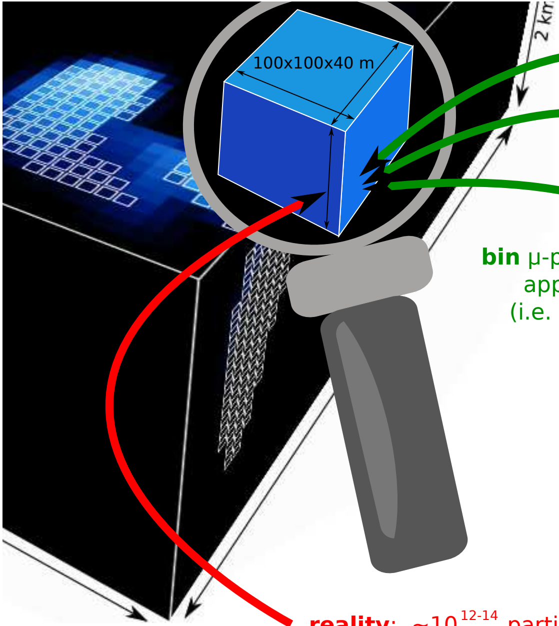


bulk μ-physics: q_c, q_r

2-moment μ-physics:
q_c, N_c, q_r, N_r

bin μ-physics: concentration density N(r)
approximated with a histogram
(i.e. defined for discrete drop radii r)

inside each grid box:

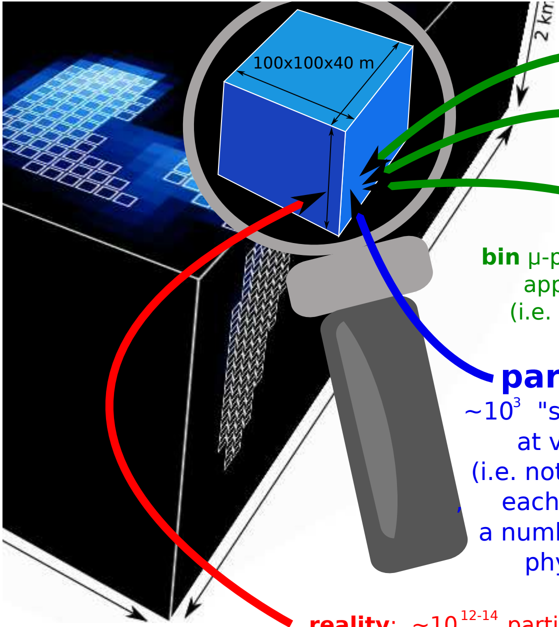


bulk μ-physics: q_c , q_r

2-moment μ-physics:
q_c, N_c, q_r, N_r

bin μ-physics: concentration density N(r)
approximated with a histogram
(i.e. defined for discrete drop radii r)

reality: ~$10^{12\text{-}14}$ particles of different sizes
(aerosol, cloud, drizzle, rain particles)



inside each grid box:

bulk μ-physics: q_c , q_r

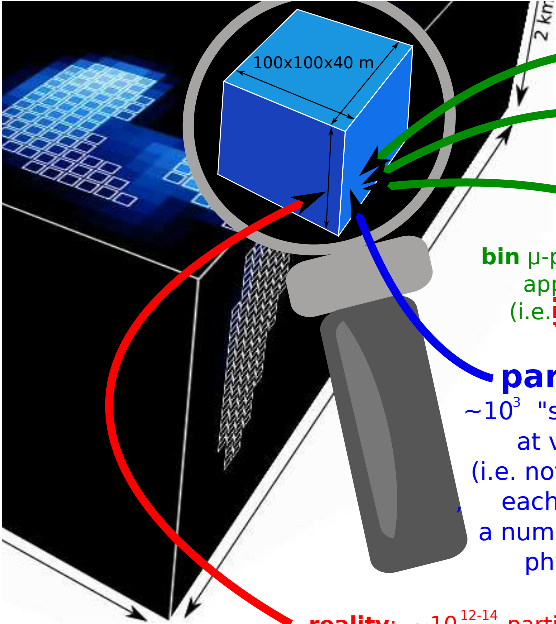
2-moment μ-physics:
q_c, N_c, q_r, N_r

bin μ-physics: concentration density N(r)
approximated with a histogram
(i.e. defined for discrete drop radii r)

particle-based μ-physics:
~10^3 "super-droplets" of variable radii,
at variable coordinates (x,y,z)
(i.e. not assigned to a given grid box)
each "super-droplet" representing
a number of real particles of the same
physico-chemical properties

reality: ~$10^{12\text{-}14}$ particles of different sizes
(aerosol, cloud, drizzle, rain particles)

inside each grid box:



bulk μ-physics: q_c , q_r

2-moment μ-physics:
q_c, N_c, q_r, N_r

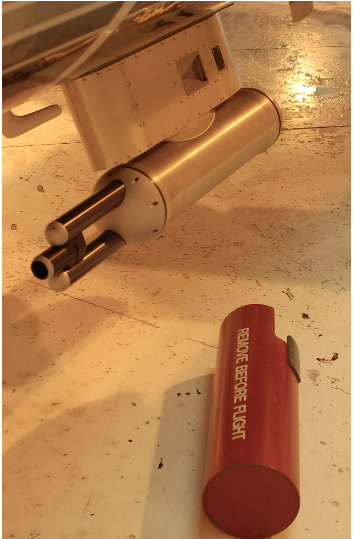
bin μ-physics: concentration density N(r)
approximated with a histogram
(i.e. defined for discrete drop radii r)

particle-based μ-physics:
~10^3 "super-droplets" of variable radii,
at variable coordinates (x,y,z)
(i.e. not assigned to a given grid box)
each "super-droplet" representing
a number of real particles of the same
physico-chemical properties

reality: ~$10^{12\text{-}14}$ particles of different sizes
(aerosol, cloud, drizzle, rain particles)

model validation: Fast-FSSP (Forward Scattering Spectrometer Probe)

- measures laser light scattered by cloud droplets



(FSSP under the SPEC Learjet fuselage)

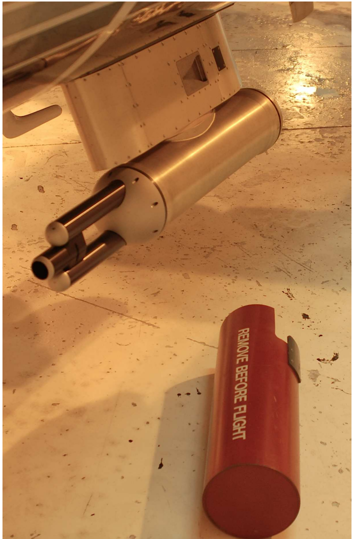
model validation: Fast-FSSP (Forward Scattering Spectrometer Probe)

- measures laser light scattered by cloud droplets
- single-particle counter

(FSSP under the SPEC Learjet fuselage)

model validation: Fast-FSSP (Forward Scattering Spectrometer Probe)

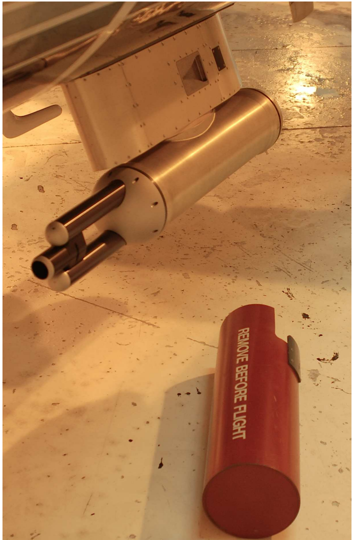
- measures laser light scattered by cloud droplets
- single-particle counter
- range: $2 - 50\,\mu m$ in diameter



(FSSP under the SPEC Learjet fuselage)

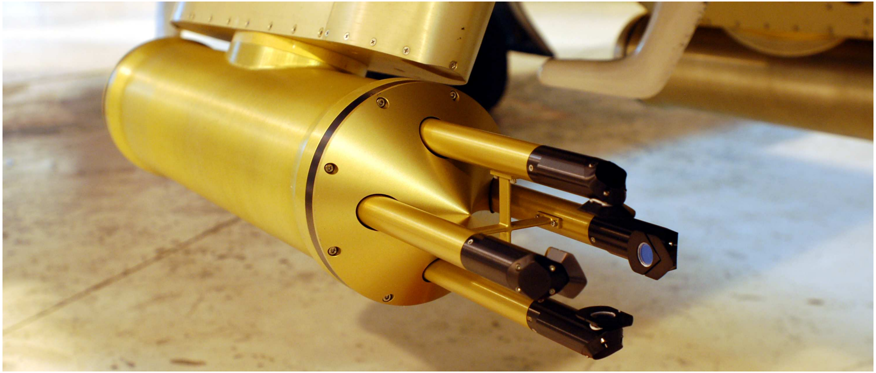
model validation: Fast-FSSP (Forward Scattering Spectrometer Probe)

- measures laser light scattered by cloud droplets
- single-particle counter
- range: $2 - 50\,\mu m$ in diameter
- developed by Météo-France



(FSSP under the SPEC Learjet fuselage)

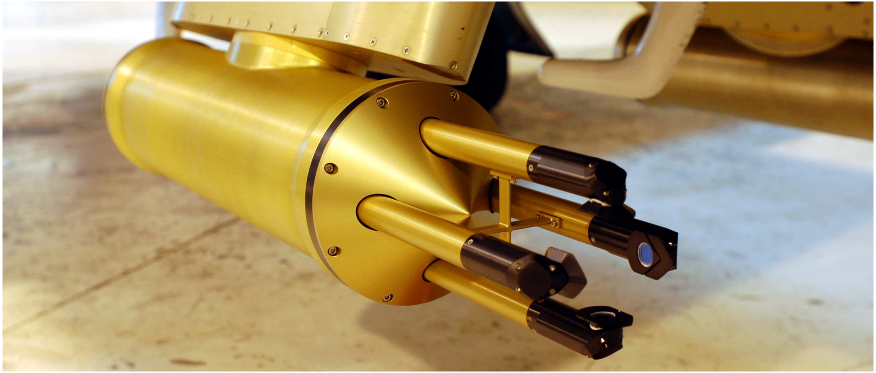
model validation: OAP-2DS (2-dimensional "stereo" optical array probe)



(OAP-2DS under the SPEC Learjet fuselage)

- registers shadows of particles on two photodiode arrays

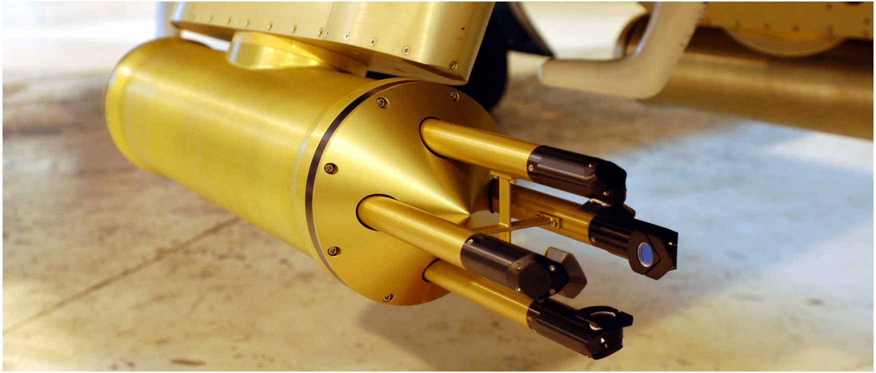
model validation: OAP-2DS (2-dimensional "stereo" optical array probe)



(OAP-2DS under the SPEC Learjet fuselage)

- registers shadows of particles on two photodiode arrays
- multiple droplets at a time, particle spectra via image analysis

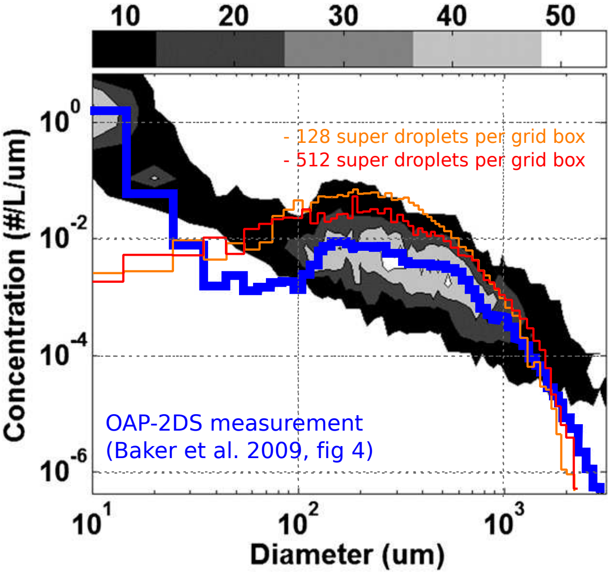
model validation: OAP-2DS (2-dimensional "stereo" optical array probe)



(OAP-2DS under the SPEC Learjet fuselage)

- registers shadows of particles on two photodiode arrays
- multiple droplets at a time, particle spectra via image analysis
- sizes cloud, drizzle and rain particles (5–3000 μm diam.)

model validation: OAP-2DS spectra vs. RICO SDM simulations



data in the plot based on:

- measurements (Baker et al.)

  ⇝ ca. 10h of research flight
- simulation (Arabas & Shima)

  ⇝ca. 10h on ES2 (∼100 TFLOPS)

PySDM: JIT-compiled SDM (open-source Python package)

100% python™ open-source code:

/ OPENATMOS / PySDM

PySDM goals:

- implementation of SDM + particle-based/Monte-Carlo models of other cloud processes

PySDM goals:

- implementation of SDM + particle-based/Monte-Carlo models of other cloud processes
- applicable in research on aerosol-cloud-interactions (and beyond)

  KPI: reproduction of results from classic literature, use in new research

PySDM goals:

- implementation of SDM + particle-based/Monte-Carlo models of other cloud processes
- applicable in research on aerosol-cloud-interactions (and beyond)
  KPI: reproduction of results from classic literature, use in new research
- **easy to reuse**: code (100% Python), documentation, examples/tutorials (Jupyter), extensibility (modular, high test coverage), interoperability (other languages, i/o), leveraging modern hardware (GPUs, multi-core CPUs)
  KPI: user feedback & multi-institutional contributions

PySDM goals:

- implementation of SDM + particle-based/Monte-Carlo models of other cloud processes
- applicable in research on aerosol-cloud-interactions (and beyond)
  KPI: reproduction of results from classic literature, use in new research
- **easy to reuse**: code (100% Python), documentation, examples/tutorials (Jupyter), extensibility (modular, high test coverage), interoperability (other languages, i/o), leveraging modern hardware (GPUs, multi-core CPUs)
  KPI: user feedback & multi-institutional contributions
- **accessibility**: seamless Linux/macOS/Windows Intel/ARM installation (pip)
  KPI: continuous integration on all targeted platforms

PySDM goals:

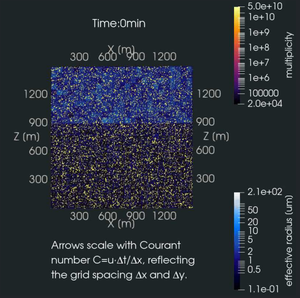
- implementation of SDM + particle-based/Monte-Carlo models of other cloud processes
- applicable in research on aerosol-cloud-interactions (and beyond)
  KPI: reproduction of results from classic literature, use in new research
- **easy to reuse**: code (100% Python), documentation, examples/tutorials (Jupyter), extensibility (modular, high test coverage), interoperability (other languages, i/o), leveraging modern hardware (GPUs, multi-core CPUs)
  KPI: user feedback & multi-institutional contributions
- **accessibility**: seamless Linux/macOS/Windows Intel/ARM installation (pip)
  KPI: continuous integration on all targeted platforms
- **curation**: open licensing (GPL), public versioned development (Github), archival (Zenodo)
  KPI: instant and anonymous execution of arbitrary version in commodity environments

PySDM

Documentation

What is PySDM?

PySDM is a package for simulating the dynamics of population of particles undergoing diffusional and collisional growth (and breakage). The package features a Pythonic high-performance (multi-threaded CPU & CUDA GPU) implementation of the Super-Droplet Method (SDM) Monte-Carlo algorithm for representing collisional growth (Shima et al. 2009), hence the name. It is intended to serve as a building block for simulation systems modelling fluid flows involving a dispersed phase, with PySDM being responsible for representation of the dispersed phase. Currently, the development is focused on atmospheric cloud physics applications, in particular on modelling the dynamics of particles immersed in moist air using the particle-based (a.k.a. super-droplet) approach to represent aerosol/cloud/rain microphysics. The key goal of PySDM is to enable rapid development and independent reproducibility of simulations in cloud microphysics while being free from the two-language barrier commonly separating prototype and high-performance research code. PySDM ships with a set of examples reproducing results from literature and serving as tutorials. The animation shown here depicts a flow-coupled simulation in which the flow is resolved using PySDM's sibling project: PyMPDATA. The examples include also single-column setups (with PyMPDATA used for advection) as well as adiabatic cloud parcel model setups (with PySDM alone sufficient to constitute a microphysics-resolving cloud parcel model in Python).



Jupyter notebook setting up and running the above PySDM simulation and generating the visualisation using Paraview

collisional breakup for (Py)SDM:
de Jong et al. 2023

Monte-Carlo collisional breakup (constant super-droplet number formulation)

Geosci. Model Dev., 16, 4193–4211, 2023
https://doi.org/10.5194/gmd-16-4193-2023

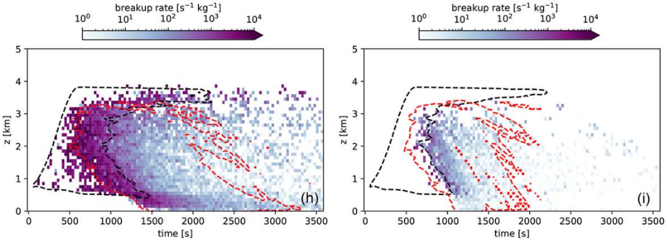
Geoscientific Model Development
Open Access
EGU

Development and technical paper

Breakups are complicated: an efficient representation of collisional breakup in the superdroplet method

Emily de Jong[1], John Ben Mackay[2,a], Oleksii Bulenok[3], Anna Jaruga[4], and Sylwester Arabas[5,b,c]

[1]Department of Mechanical and Civil Engineering, California Institute of Technology, Pasadena, CA, USA
[2]Scripps Institution of Oceanography, San Diego, CA, USA
[3]Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
[4]Department of Environmental Science and Engineering, California Institute of Technology, Pasadena, CA, USA
[5]Faculty of Physics and Applied Computer Science, AGH University of Krakow, Kraków, Poland
[a]formerly at: Department of Environmental Science and Engineering, California Institute of Technology, Pasadena, CA, USA
[b]formerly at: Department of Atmospheric Sciences, University of Illinois Urbana-Champaign, Urbana, IL, USA
[c]formerly at: Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

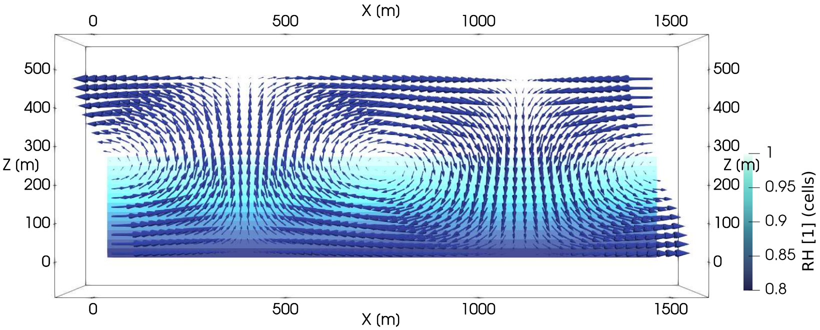


Monte-Carlo immersion freezing in (Py)SDM

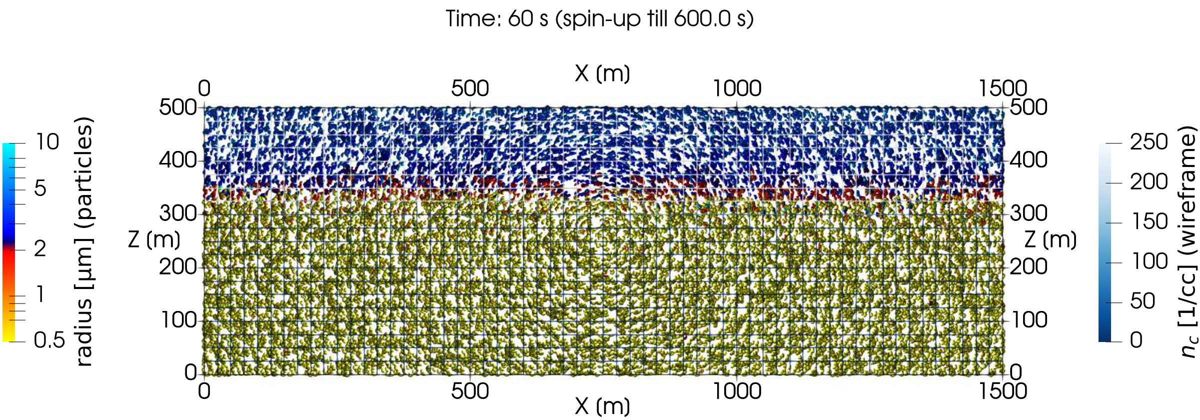
immersion freezing: bacteria and the Olympics

https://www.reuters.com/markets/commodities/making-snow-stick-wind-challenges-winter-games-slope-makers-2021-11-29/

immersion freezing: singular vs. time-dependent in flow-coupled simulation



immersion freezing: singular vs. time-dependent in flow-coupled simulation

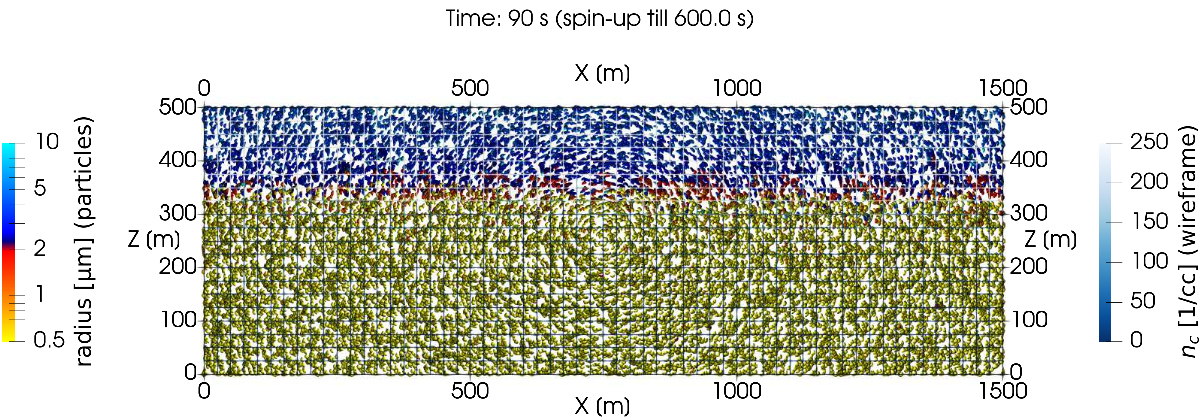


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $\quad N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

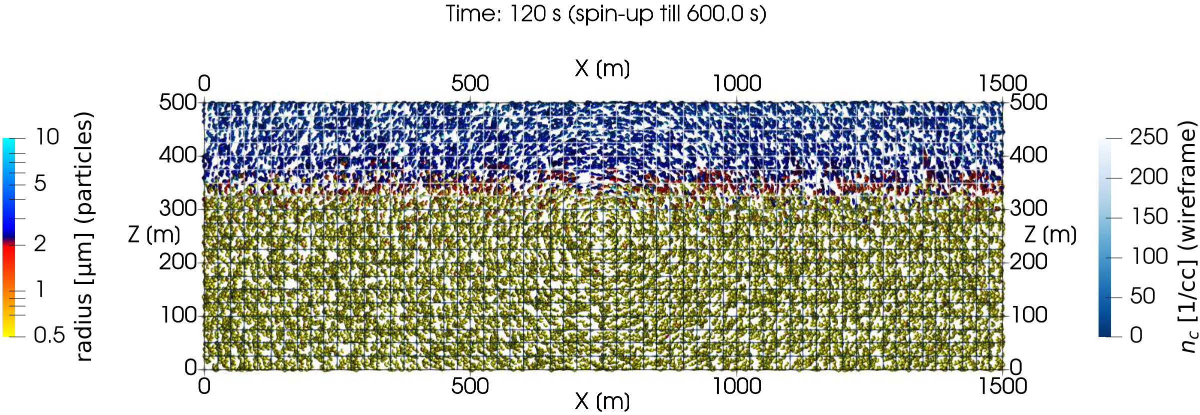


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

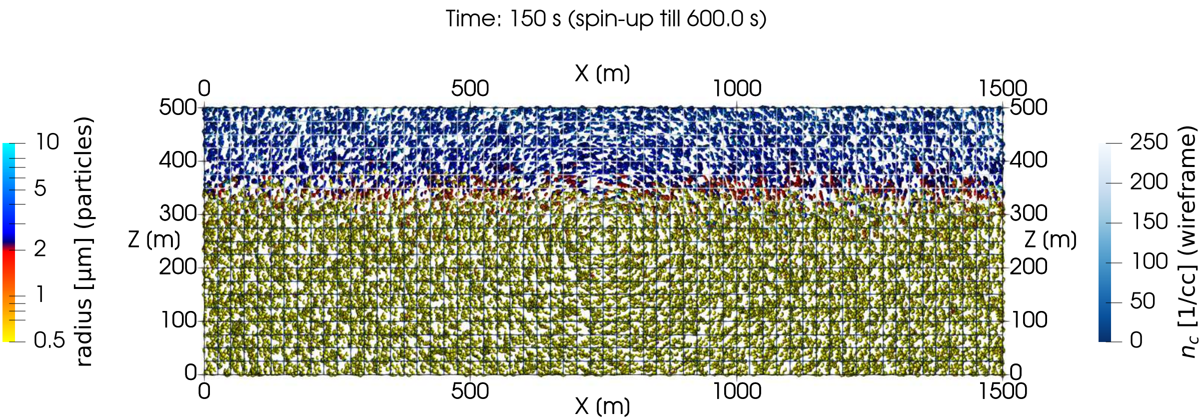


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

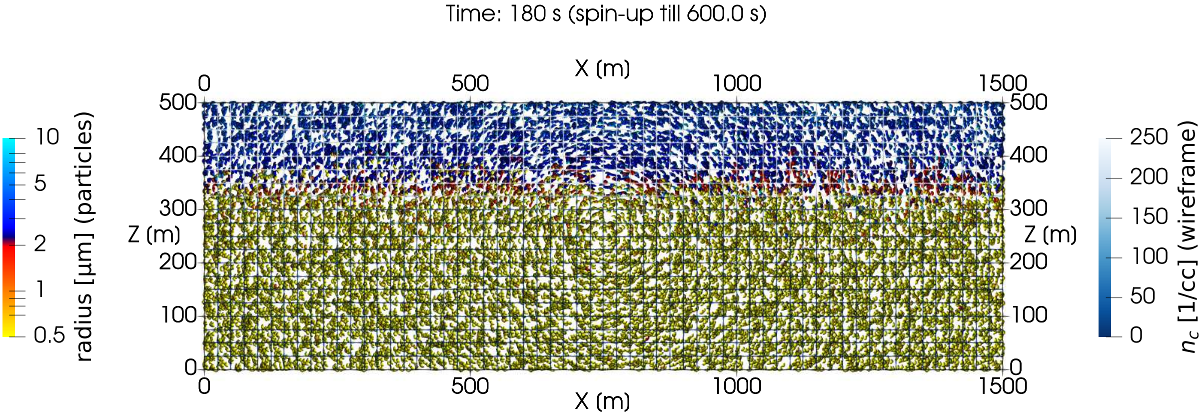


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

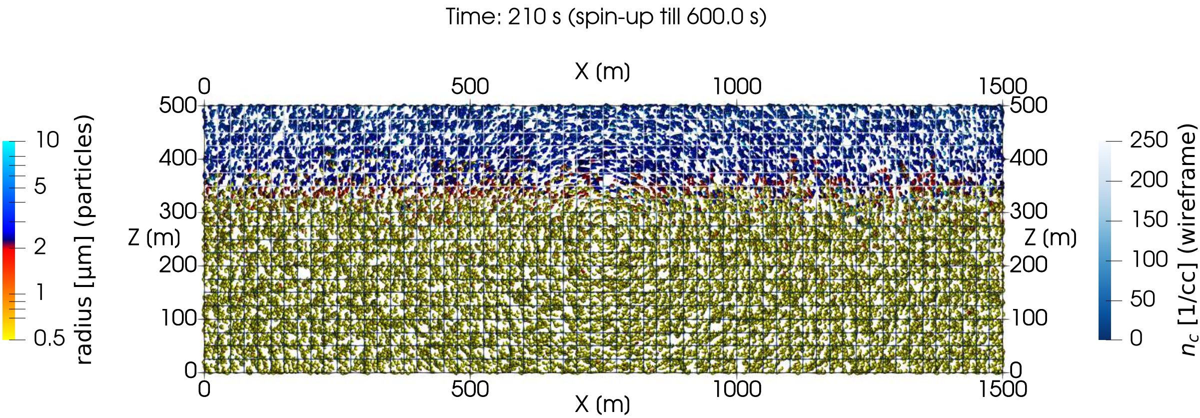


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $\quad N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

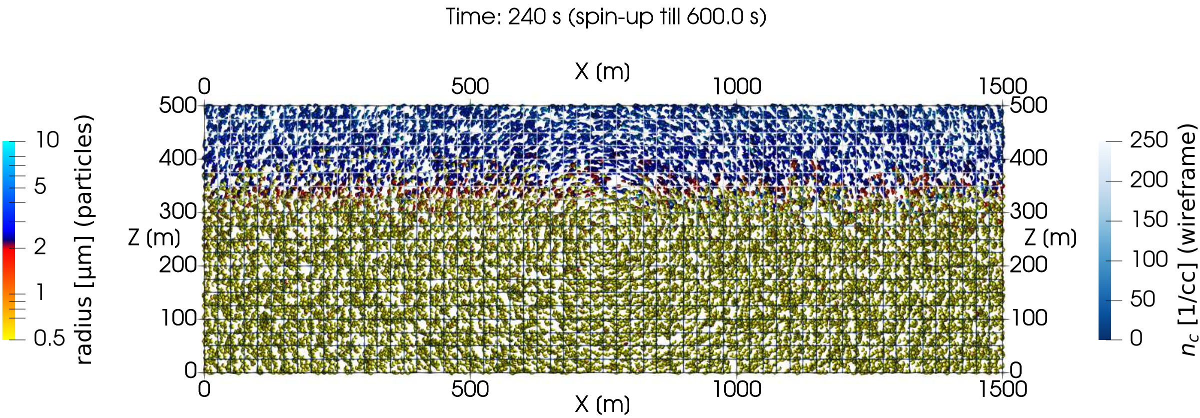


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation



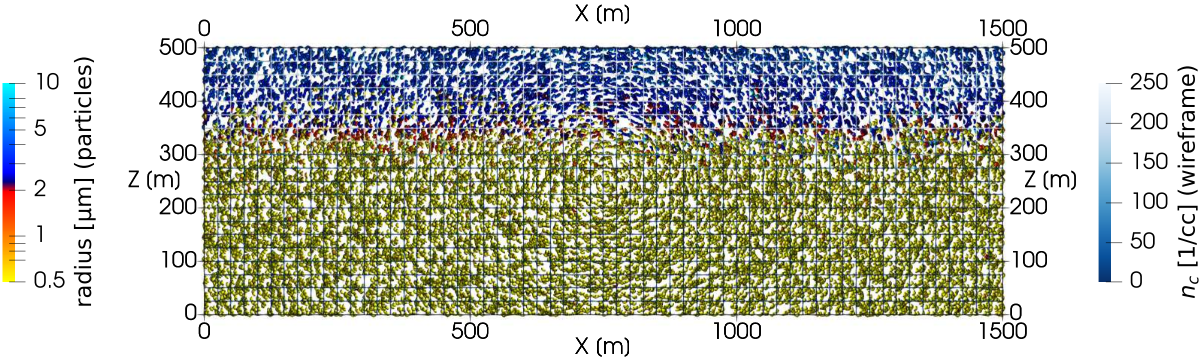
16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

Time: 270 s (spin-up till 600.0 s)

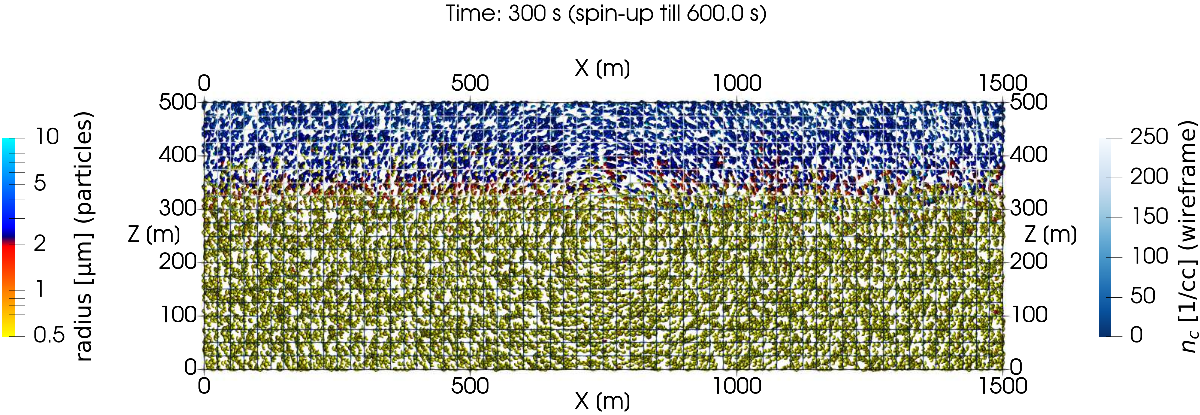


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

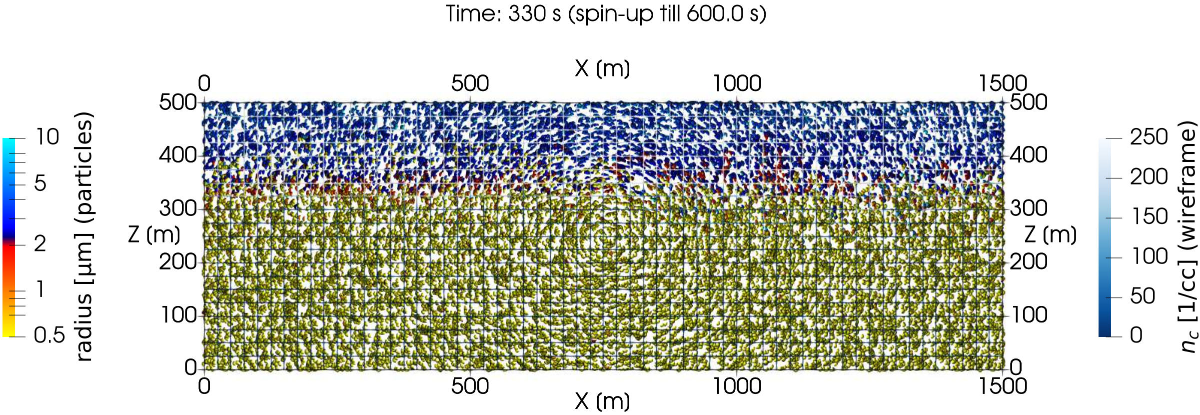


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

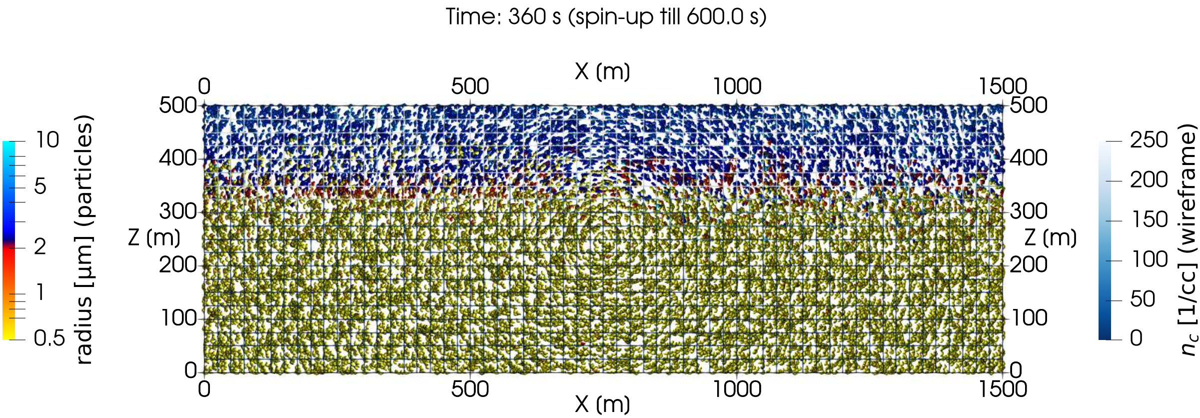


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

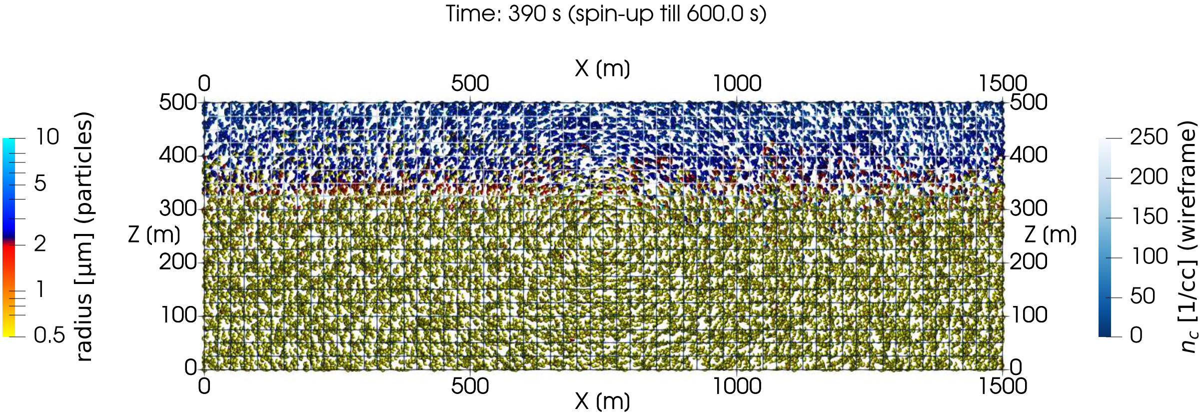


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

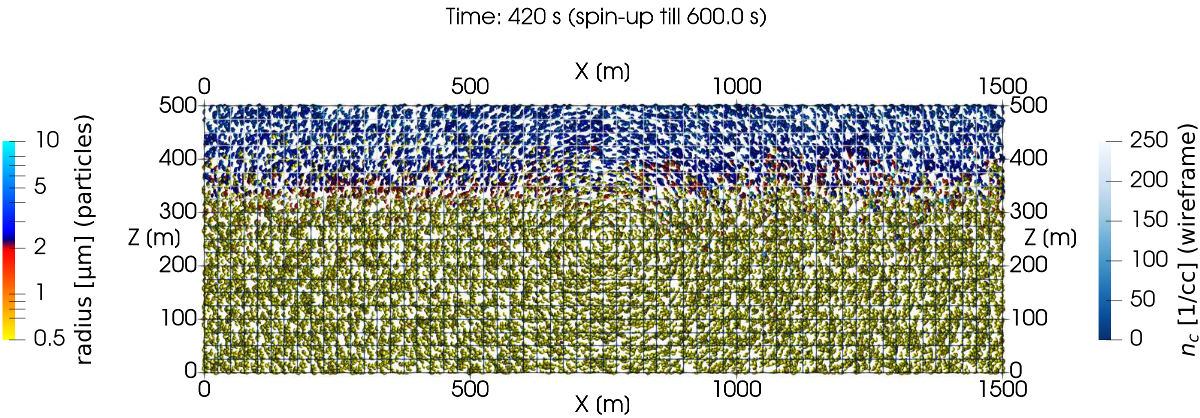


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

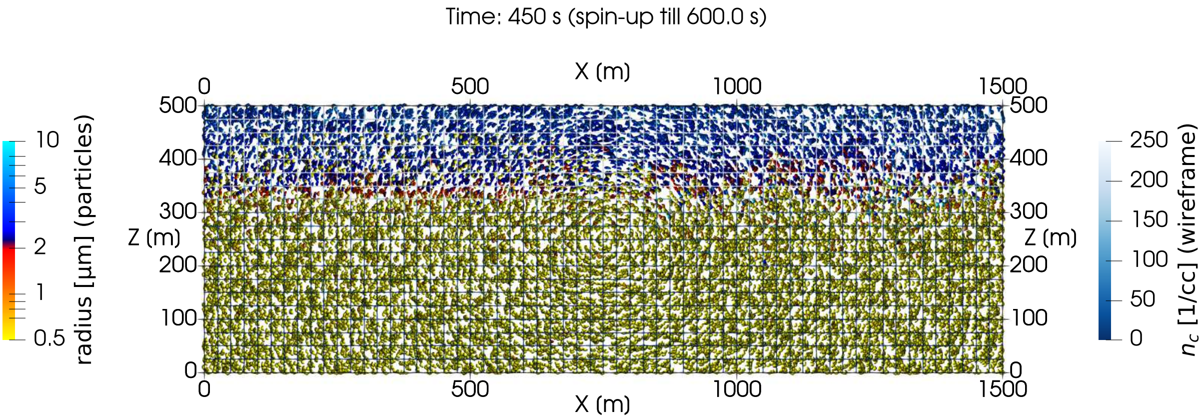


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

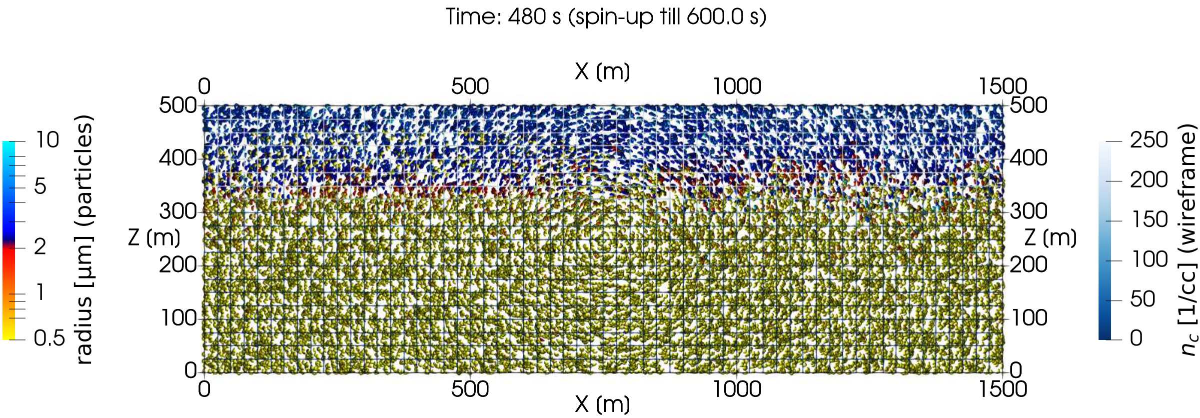


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

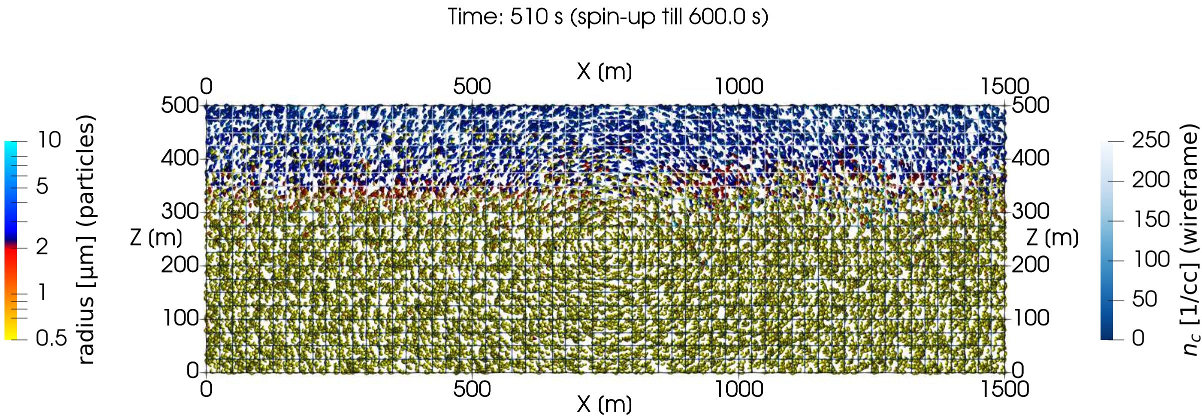


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

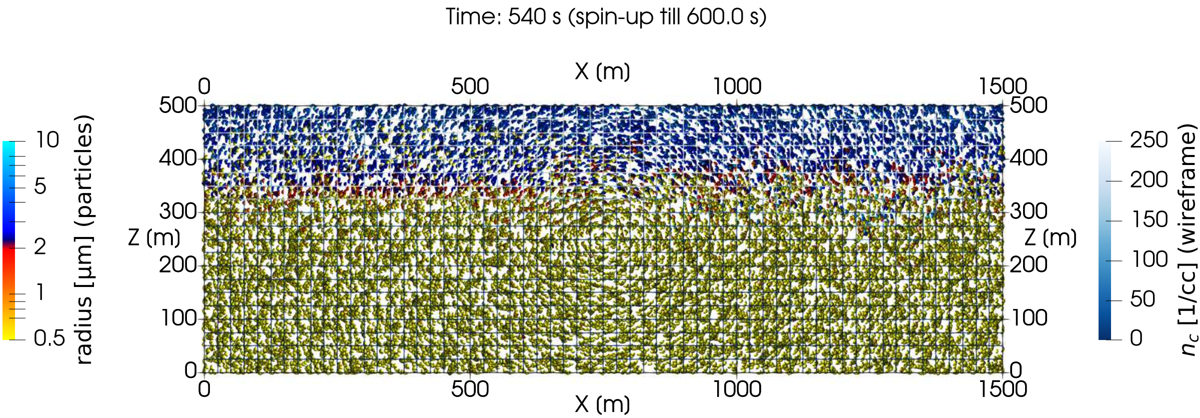


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $\quad N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

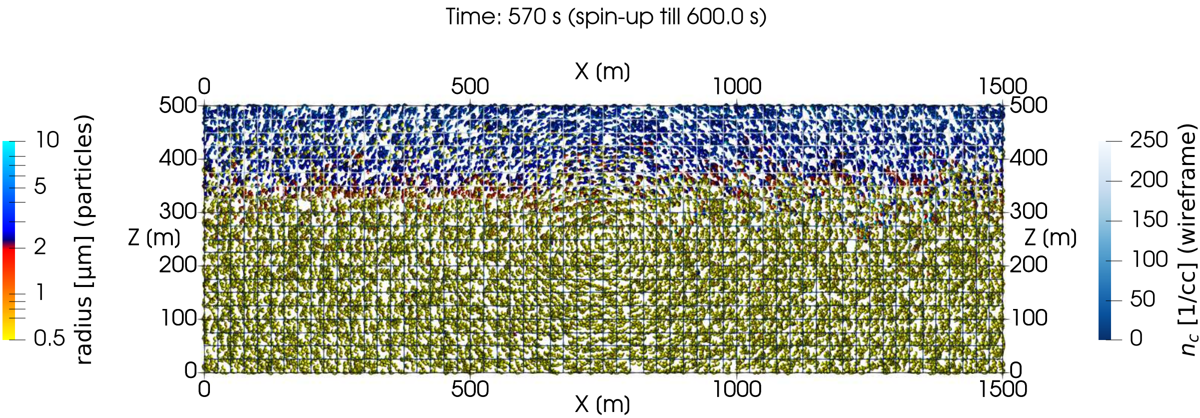


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

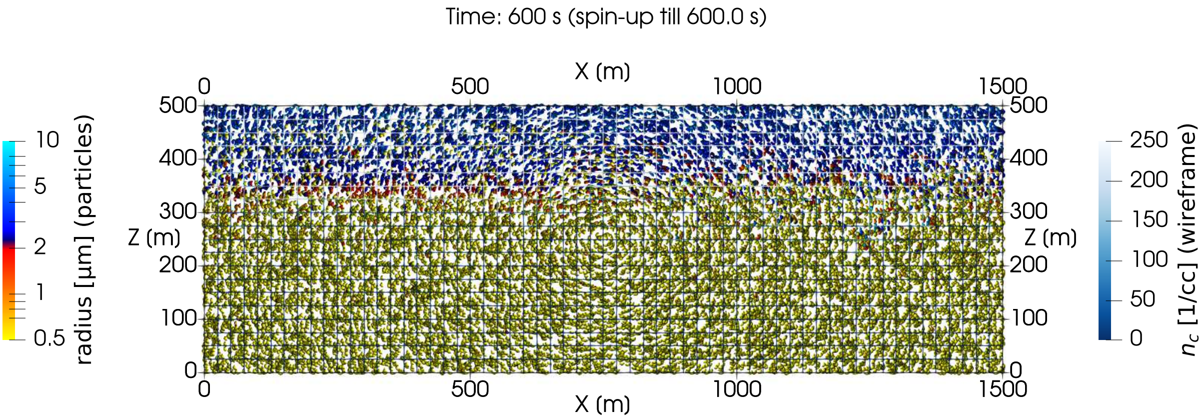


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

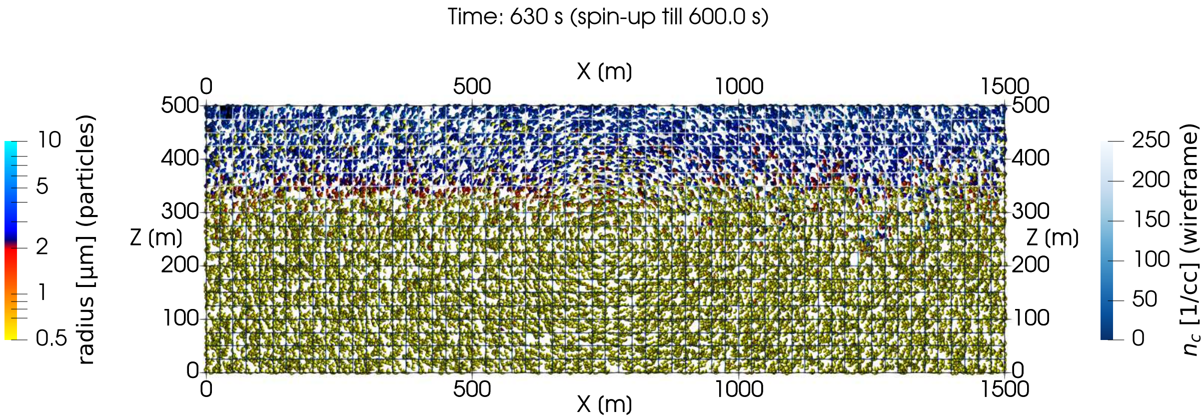


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\mathrm{aer}} = 300/cc$ (two-mode lognormal) $N_{\mathrm{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_{\mathrm{g}} = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

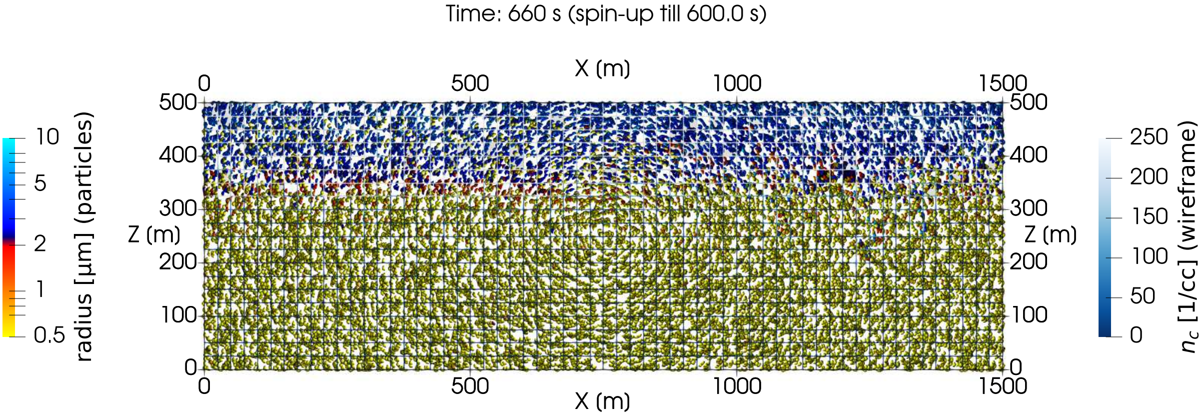


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

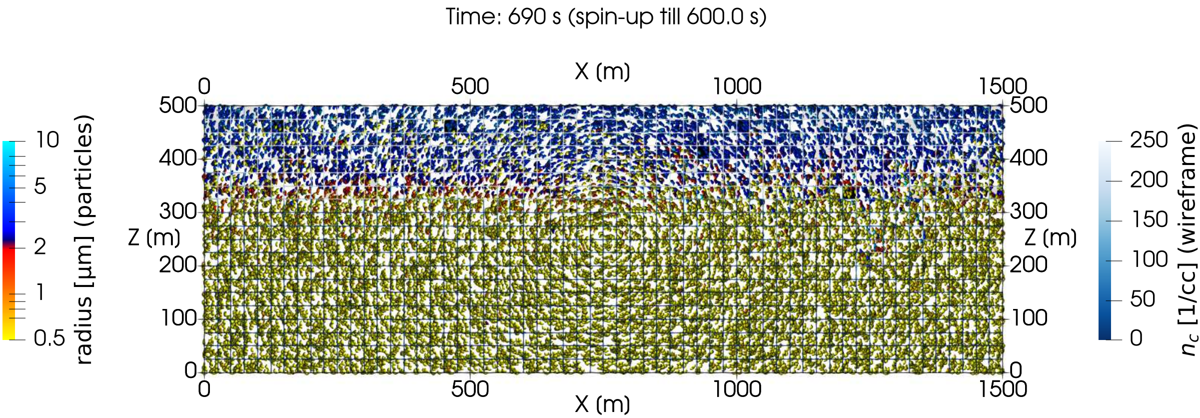


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_{\text{g}} = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

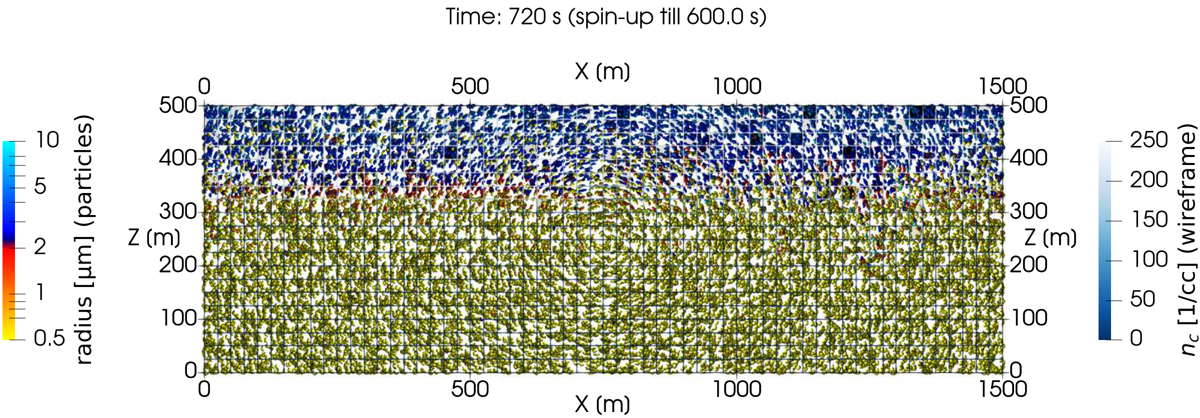


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

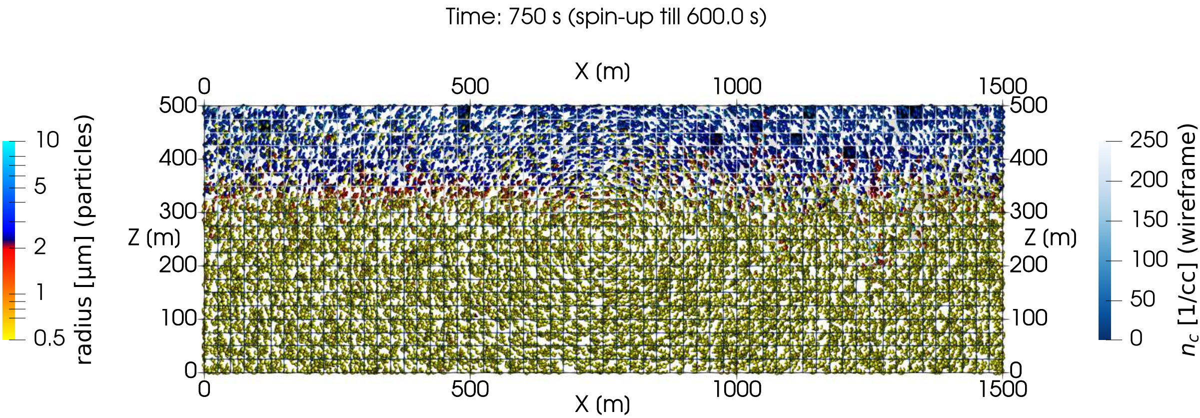


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

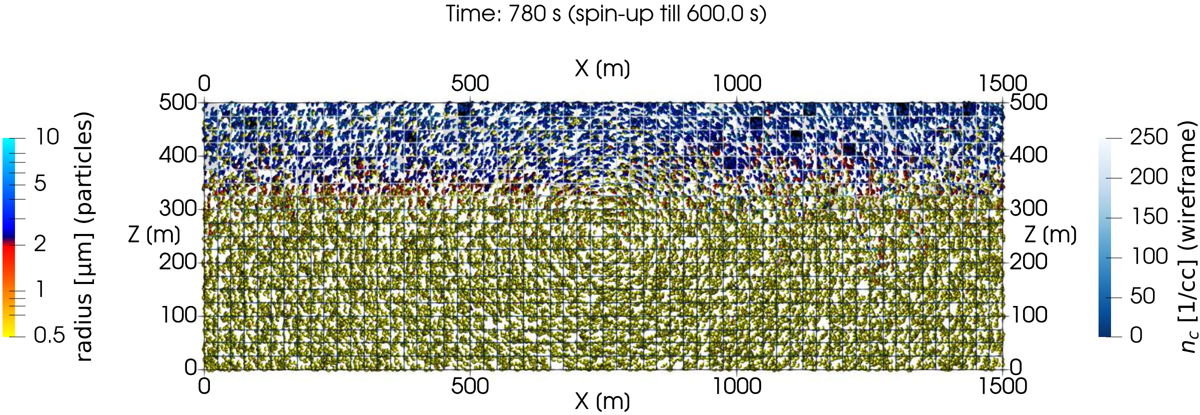


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

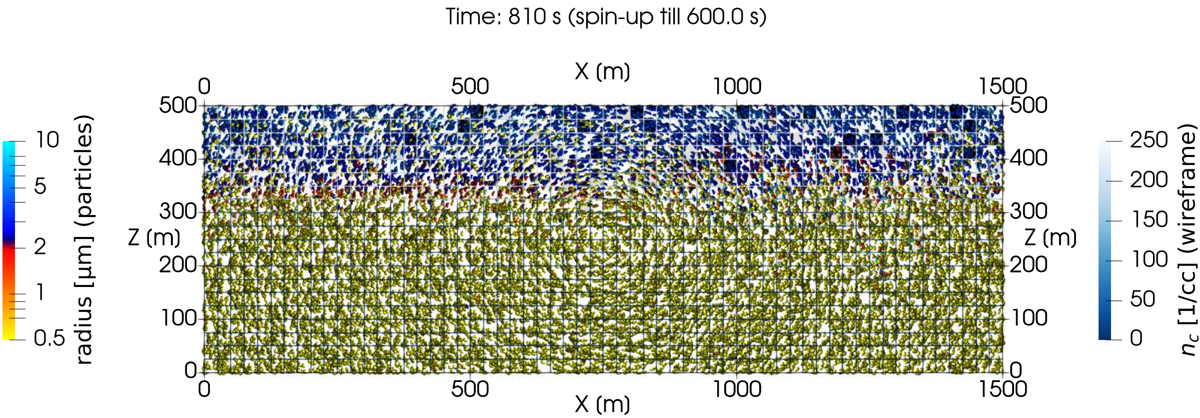


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_{\text{g}} = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

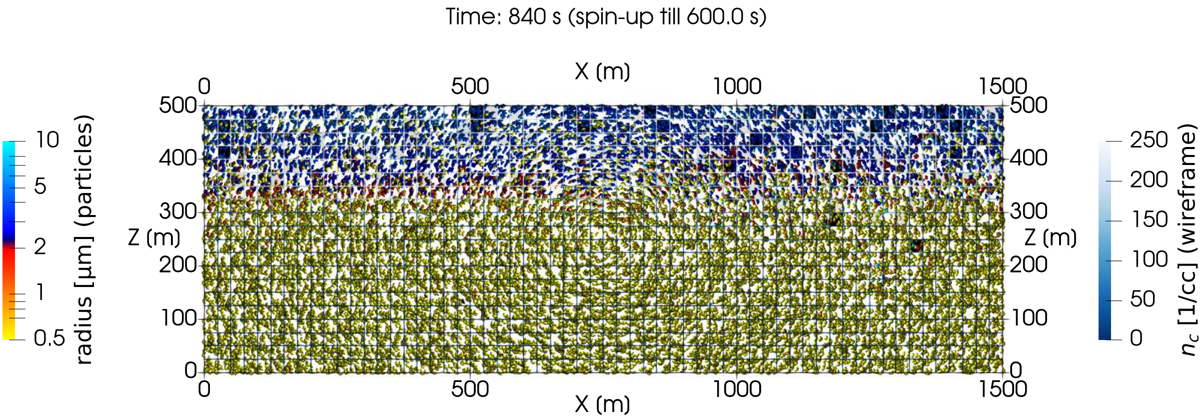


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

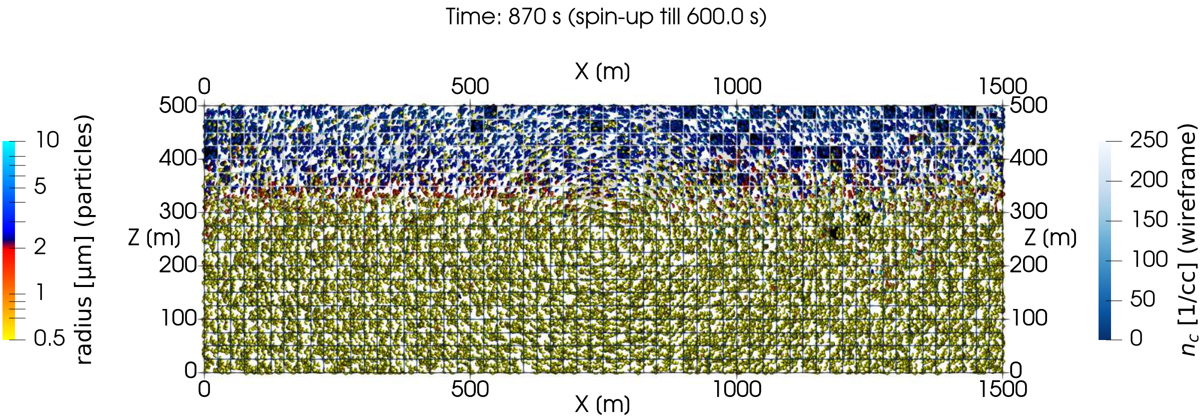


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

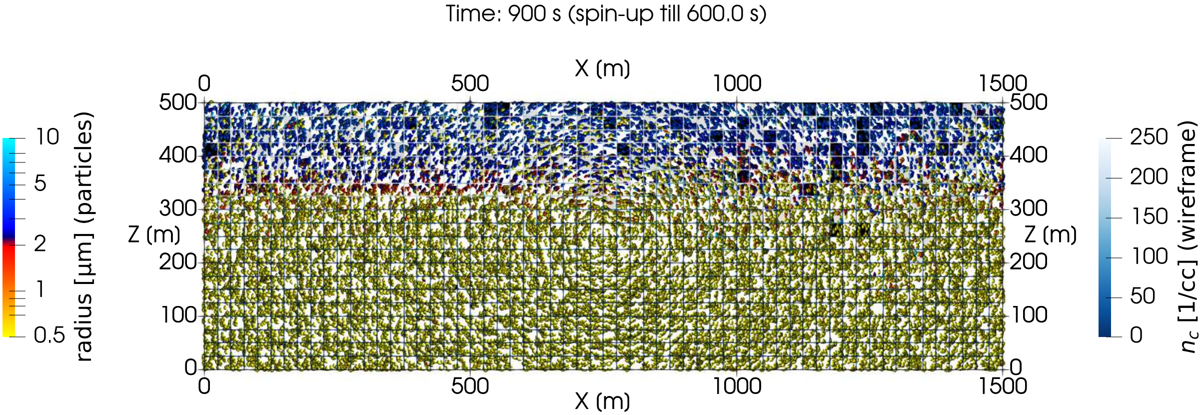


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

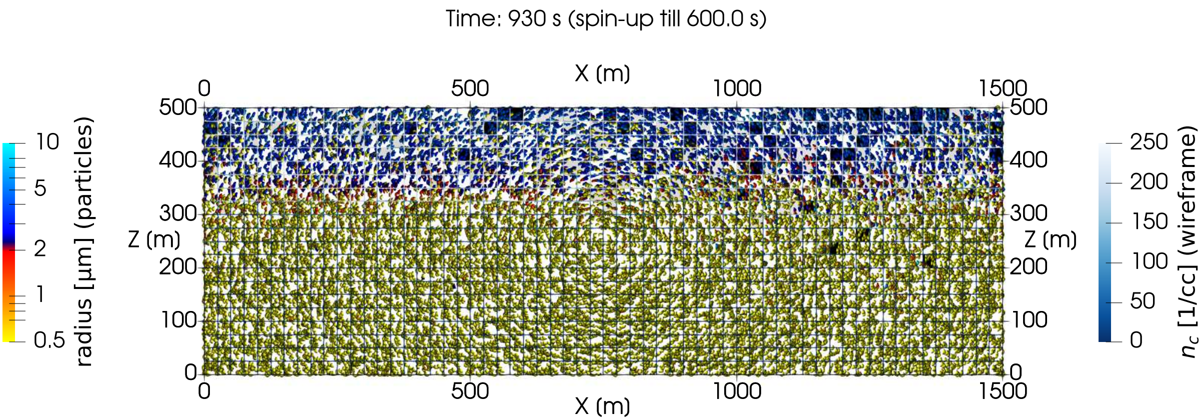


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

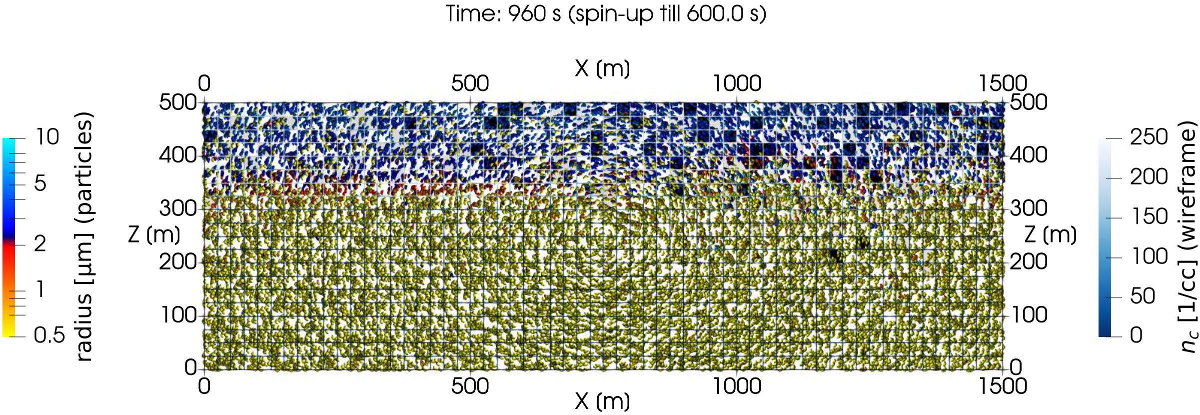


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation



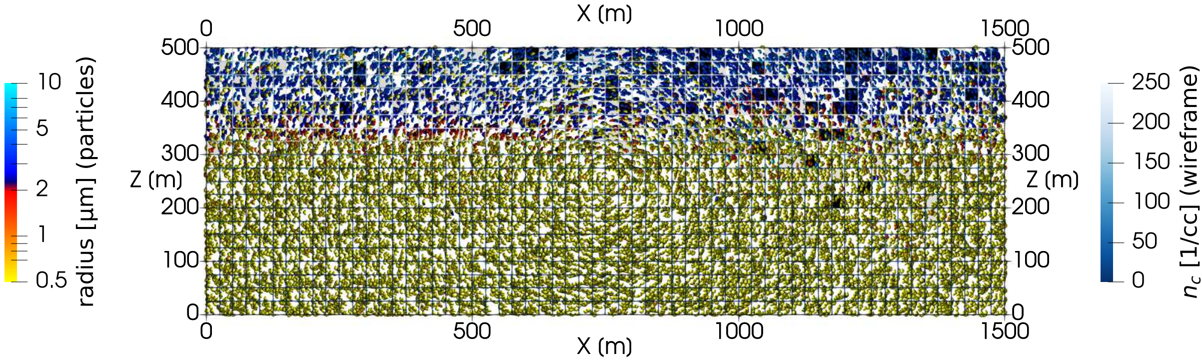
16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

Time: 990 s (spin-up till 600.0 s)

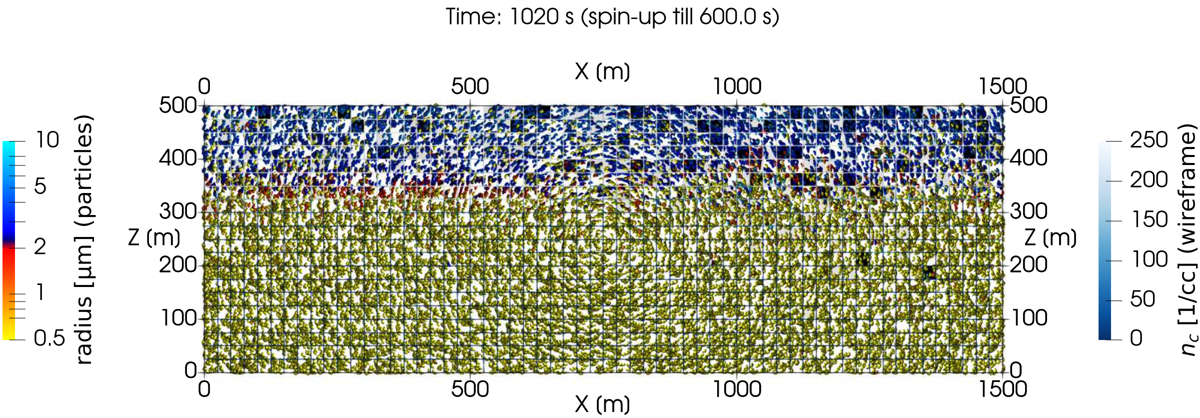


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $\quad N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

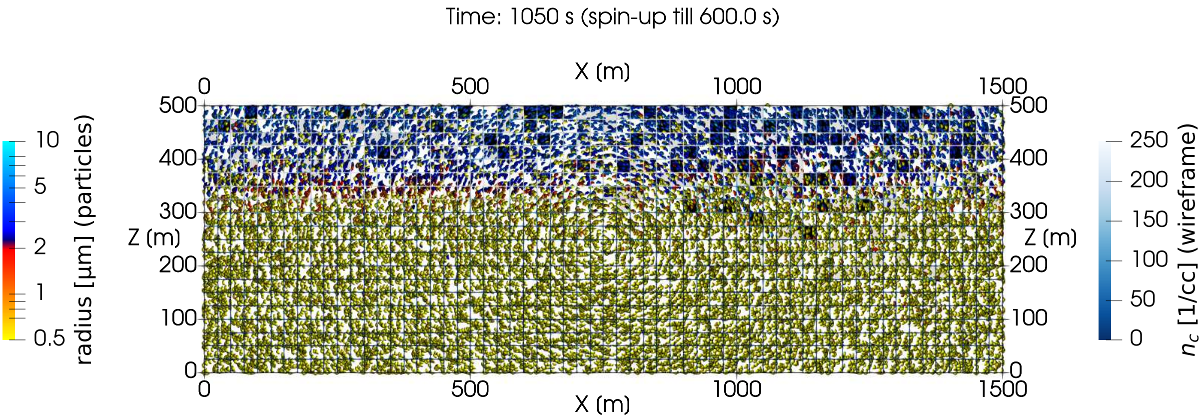


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

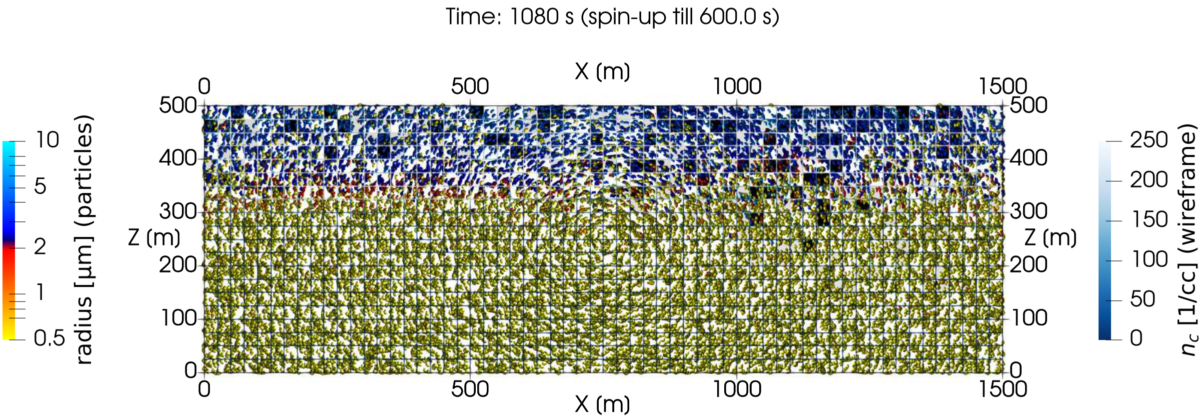


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $\quad N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

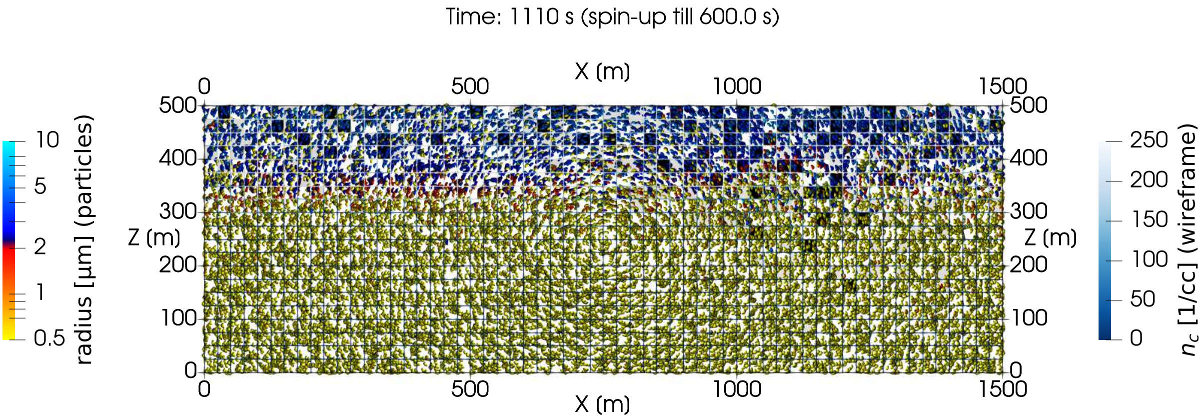


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

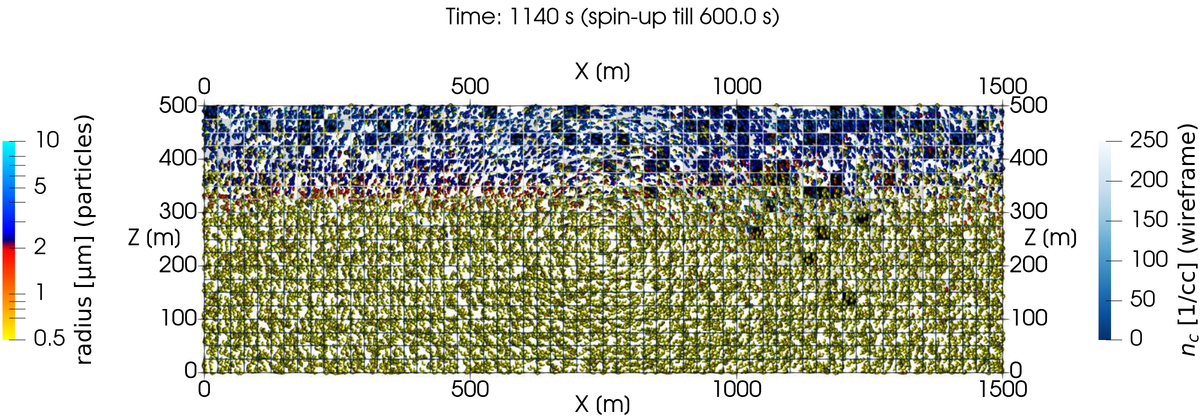


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_{\text{g}} = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

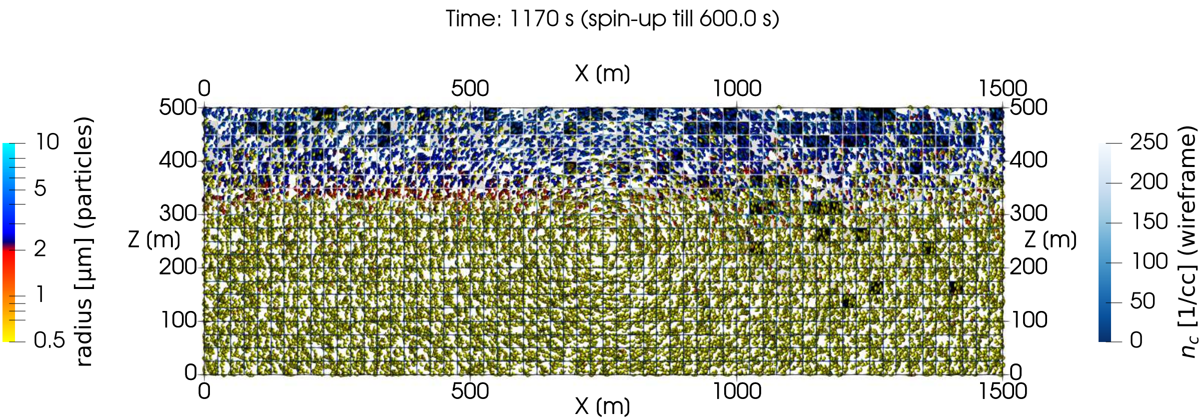


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation

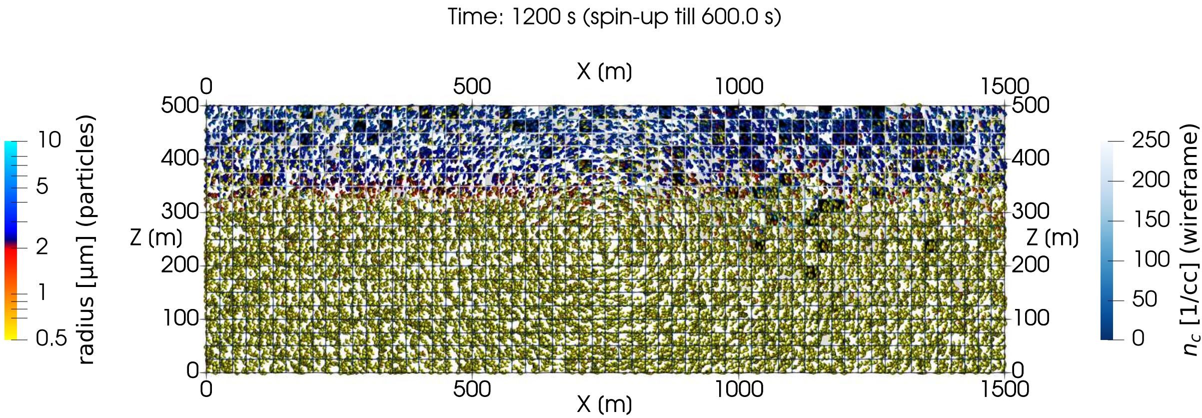


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $\quad N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

immersion freezing: singular vs. time-dependent in flow-coupled simulation



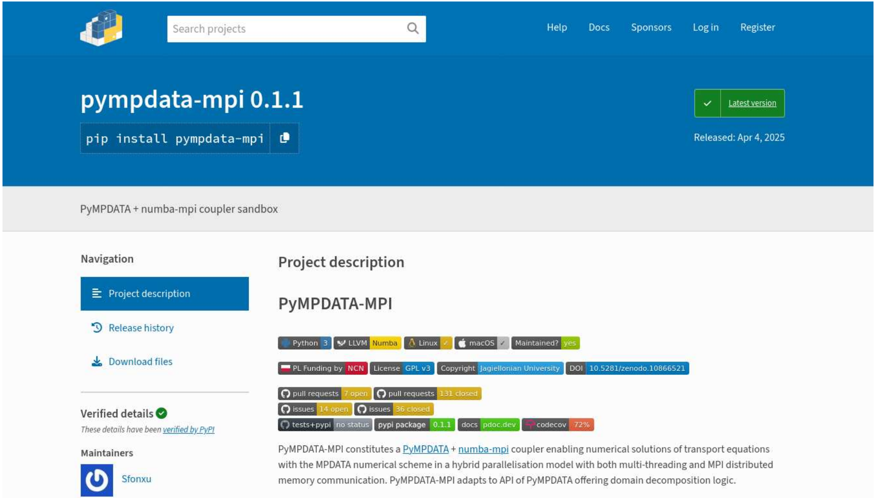
16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ µm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

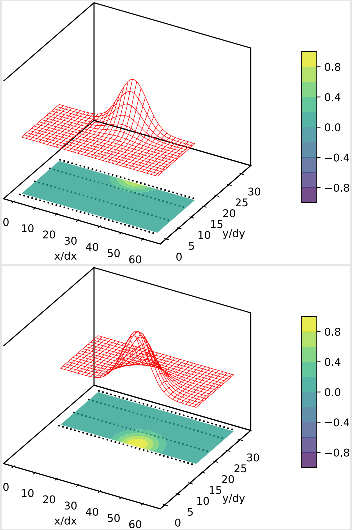
PySDM companion packages: PyMPDATA & Numba-MPI

PyMPDATA-MPI

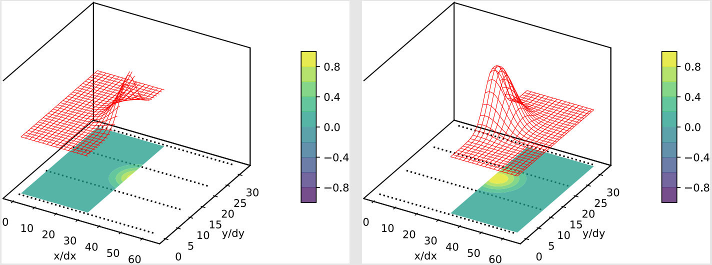


PyMPDATA-MPI: customisable hybrid threading + MPI parallelisation

threading dim = MPI dim



threading dimension ≠ MPI dimension



Derlatka et al. 2024 (SoftwareX, doi:10.1016/j.softx.2024.101897)

introducing Numba-MPI (now a dependency of py-pde)

ScienceDirect

SoftwareX

Volume 28, December 2024, 101897

Original software publication

Numba-MPI v1.0: Enabling MPI communication within Numba/LLVM JIT-compiled Python code

Kacper Derlatka [a 1], Maciej Manna [a 2], Oleksii Bulenok [a 3], David Zwicker [b], Sylwester Arabas [c]

[a] Faculty of Mathematics and Computer Science, Jagiellonian University in Kraków, Poland

[b] Max Planck Institute for Dynamics and Self-Organization, Göttingen, Germany

[c] Faculty of Physics and Applied Computer Science, AGH University of Krakow, Poland

https://doi.org/10.1016/j.softx.2024.101897

Abstract

The numba-mpi package offers access to the Message Passing Interface (MPI) routines from Python code that uses the Numba just-in-time (JIT) compiler. As a result, high-performance and multi-threaded Python code may utilize MPI communication facilities without leaving the JIT-compiled code blocks, which is not possible with the mpi4py package, a higher-level Python interface to MPI. For debugging or code-coverage analysis purposes, numba-mpi retains full functionality of the code even if the JIT compilation is disabled.

habilitation application

A: Methods and Tools for Modelling of Atmospheric Cloud Microphysics:

- **A1:** **Arabas**, Pawlowska 2011
  "*Adaptive method of lines for multi-component aerosol condensational growth and CCN activation*"
  Geosci. Model Dev. (EGU)
- **A2:** **Arabas**, Shima 2013
  "*Large-Eddy Simulations of Trade Wind Cumuli Using Particle-Based Microphysics with Monte Carlo Coalescence*"
  J. Atmos. Sci. (AMS)
- **A3:** **Arabas**, Jaruga, Pawlowska, Grabowski 2015
  "*libcloudph++ 1.0: a single-moment bulk, double-moment bulk, and particle-based warm-rain microphysics library in C++*"
  Geosci. Model Dev. (EGU)
- **A4:** **Arabas**, Shima 2017
  "*On the CCN (de)activation nonlinearities*"
  Nonlin. Proc. Geophys. (EGU)
- **A5:** Olesik, Banaśkiewicz, Bartman, Baumgartner, Unterstrasser, **Arabas** 2022
  "*On numerical broadening of particle-size spectra: A condensational growth study using PyMPDATA*"
  Geosci. Model Dev. (EGU)
- **A6:** de Jong, Mackay, Bulenok, Jaruga, **Arabas** 2023
  "*Breakups are complicated: An efficient representation of collisional breakup in the superdroplet method*"
  Geosci. Model Dev. (EGU)
- **A7:** **Arabas**, Curtis, Silber, Fridlind, Knopf, Riemer 2025
  "*Immersion Freezing in Particle-Based Aerosol-Cloud Microphysics: A Probabilistic Perspective on Singular and Time-Dependent Models*
  J. Adv. Model. Earth Syst. (AGU)

B: Implementation and Applications of Multidimensional Positive Definite Advection Transport Algorithm (MPDATA):

- **B1:** Jaruga, **Arabas**, Jarecka, Pawlowska, Smolarkiewicz, Waruszewski 2015
  "*libmpdata++ 1.0: a library of parallel MPDATA solvers for systems of generalised transport equations*"
  Geosci. Model Dev. (EGU)
- **B2:** **Arabas**, i Farhat 2020
  "*Derivative pricing as a transport problem: MPDATA solutions to Black-Scholes-type equations*"
  J. Comp. Appl. Math. (Elsevier)
- **B3:** Derlatka, Manna, Bulenok, Zwicker, **Arabas** 2024
  "*Numba-MPI v1.0: Enabling MPI communication within Numba/LLVM JIT-compiled Python code*"
  SoftwareX (Elsevier)

(underlined names indicate mentored students)

A: Methods and Tools for Modelling of Atmospheric Cloud Microphysics:

A1: **Arabas**, Pawlowska 2011
"*Adaptive method of lines for multi-component aerosol condensational growth and CCN activation*"
Geosci. Model Dev. (EGU)

A2: **Arabas**, Shima 2013
"*Large-Eddy Simulations of Trade Wind Cumuli Using Particle-Based Microphysics with Monte Carlo Coalescence*"
J. Atmos. Sci. (AMS)

A3: **Arabas**, Jaruga, Pawlowska, Grabowski 2015
"*libcloudph++ 1.0: a single-moment bulk, double-moment bulk, and particle-based warm-rain microphysics library in C++*"
Geosci. Model Dev. (EGU)

A4: **Arabas**, Shima 2017
"*On the CCN (de)activation nonlinearities*"
Nonlin. Proc. Geophys. (EGU)

A5: Olesik, Banaśkiewicz, Bartman, Baumgartner, Unterstrasser, **Arabas** 2022
"*On numerical broadening of particle-size spectra: A condensational growth study using PyMPDATA*"
Geosci. Model Dev. (EGU)

A6: de Jong, Mackay, Bulenok, Jaruga, **Arabas** 2023
"*Breakups are complicated: An efficient representation of collisional breakup in the superdroplet method*"
Geosci. Model Dev. (EGU)

A7: **Arabas**, Curtis, Silber, Fridlind, Knopf, Riemer 2025
"*Immersion Freezing in Particle-Based Aerosol-Cloud Microphysics: A Probabilistic Perspective on Singular and Time-Dependent Models*
J. Adv. Model. Earth Syst. (AGU)

B: Implementation and Applications of Multidimensional Positive Definite Advection Transport Algorithm (MPDATA):

B1: Jaruga, **Arabas**, Jarecka, Pawlowska, Smolarkiewicz, Waruszewski 2015
"*libmpdata++ 1.0: a library of parallel MPDATA solvers for systems of generalised transport equations*"
Geosci. Model Dev. (EGU)

B2: **Arabas**, i Farhat 2020
"*Derivative pricing as a transport problem: MPDATA solutions to Black-Scholes-type equations*"
J. Comp. Appl. Math. (Elsevier)

B3: Derlatka, Manna, Bulenok, Zwicker, **Arabas** 2024
"*Numba-MPI v1.0: Enabling MPI communication within Numba/LLVM JIT-compiled Python code*"
SoftwareX (Elsevier)

(underlined names indicate mentored students)

kudos team & contributors!

students in Kraków:

Graduate Students (@agh.edu.pl)

Agnieszka Żaba
Ongoing PhD project co-advised with Mirosław Zimnoch (prior: MSc in Mathematics, AGH, 2023, BEng in Applied Mathematics, Wrocław University of Science and Technology, 2018)

Konrad Bodzioch
Ongoing MSc project (major: Computer Science & Intelligent Systems; prior: BEng in Computer Science & Intelligent Systems, AGH, 2024)

Undergraduate Students (@fis.agh.edu.pl)

Gracjan Adamus
Ongoing internship (major: Applied Computer Science, AGH)

Daria Klimaszewska
Ongoing BEng project (major: Technical Physics, AGH; prior: BEng in Nanomaterials Engineering, AGH)

Aleksandra Strząbała
Ongoing BEng project (major: Micro- and Nanotechnologies in Biophysics, AGH)

Michał Wroński
Ongoing BEng project (major: Micro- and Nanotechnologies in Biophysics, AGH)

Alumni (@agh.edu.pl & @uj.edu.pl)

Emma Ware
Completed Erasmus+ traineeship in 2024/25 at AGH (from U. California Davis)

Paweł Magnuszewski
Completed MSc project in 2025 (major: Computer Science & Intelligent Systems, AGH)

Michał Kowalczyk
Completed BEng project in 2025 (major: Technical Physics, AGH)

Weronika Romaniec
Completed a (visual identity design) summer internship at AGH in 2024 (from Social Informatics, AGH)

Sanket Bhiogade
PhD student at AGH (2023-2024)

Agnieszka Makulska
Completed a summer internship at AGH in 2023 (from Physics, Univ. Warsaw)

Kacper Derlatka
Completed MSc project in 2023 (major: Computer Science, Jagiellonian Univ.)

Oleksii Bulenok
Completed MSc project in 2023 (major: Computer Science, Jagiellonian Univ.)

Piotr Bartman-Szwarc
Completed MSc project in 2020 (major: Computer Science, Jagiellonian Univ.)

Michael Olesik
Completed MSc project in 2020 (major: Physics, Jagiellonian Univ.)

PySDM, PyMPDATA and Numba-MPI contributors:

- caltech.edu: Sajjad Azimi, Ben Mackay & Anna Jaruga
- colorado.edu: Clare Singer
- columbia.edu: Jatan Buch
- ds.mpg.de: David Zwicker
- exeter.ac.uk: Daniel Partridge
- imgw.pl: Maciej Manna
- llnl.gov: Emily de Jong
- mpimet.mpg.de: Clara Bayley
- okayama-u.ac.jp: Grzegorz Łazarski
- uni-mainz.de: Tim Lüttmer
- uw.edu.pl: Agnieszka Makulska
- washington.edu: Jason Barr

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

OPENATMOS ENVIRONMENTAL PHYSICS GROUP AGH AGH UNIVERSITY OF KRAKOW

Team News Projects Journal Club Publications Contact

We are a team of students, researchers and developers at the **AGH University of Krakow** 🎓 in Poland 🇵🇱. Our activities concern free/libre and open-source software for **atmospheric sciences** 🌍, and **cloud physics** 🌨️ in particular. We develop example-rich tools to support reproducible **computational science**, while promoting best practices in **research software engineering**. Our team contributes to the open-source ecosystem we rely on. We explore modern, transferable technologies that are useful both within and beyond academia. Most of the projects we develop, maintain, and contribute to are hosted at **`github.com/open-atmos`** (while the **Kraków** airport IATA ✈ code is KRK, hence **`open-atmos-krk`** 😉). Before moving to the **AGH Environmental Physics Group** in 2023, we had been based at the Institute of Computer Science and Computational Mathematics, Jagiellonian University in Kraków.

FNP Foundation for Polish Science | National Science Centre Poland | Excellence Initiative Research University | Erasmus+ | PL-Grid

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

AGH University of Krakow

Team News Projects Journal Club Publications Contact

We are a team of students, researchers and developers at the **AGH University of Krakow** 🎓 in Poland 🇵🇱. Our activities concern free/libre and open-source software for **atmospheric sciences** 🌍, and **cloud physics** 🌨 in particular. We develop example-rich tools to support reproducible **computational science**, while promoting best practices in **research software engineering**. Our team contributes to the open-source ecosystem we rely on. We explore modern, transferable technologies that are useful both within and beyond academia. Most of the projects we develop, maintain, and contribute to are hosted at **`github.com/open-atmos`** (while the **Kraków** airport IATA ✈ code is KRK, hence **`open-atmos-krk`** 😉). Before moving to the **AGH Environmental Physics Group** in 2023, we had been based at the Institute of Computer Science and Computational Mathematics, Jagiellonian University in Kraków.

Thank you for your attention!

sylwester.arabas@agh.edu.pl