

PySDM: particle-based cloud μ -physics modelling with python™

PySDM Team (presented by Sylwester Arabas)

23 Jan 2026

PySDM mini-workshop @UW.edu



PySDM

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



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

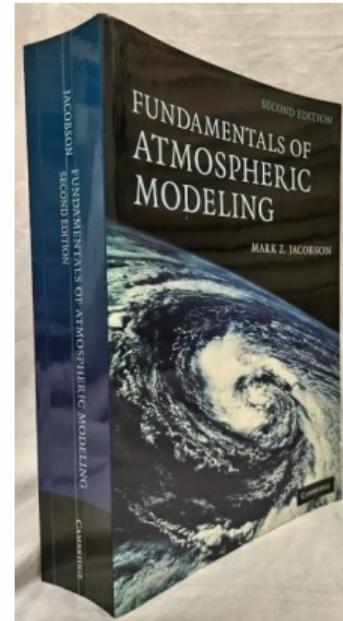
(photo: Yevgen Timashov / National Geographic)

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

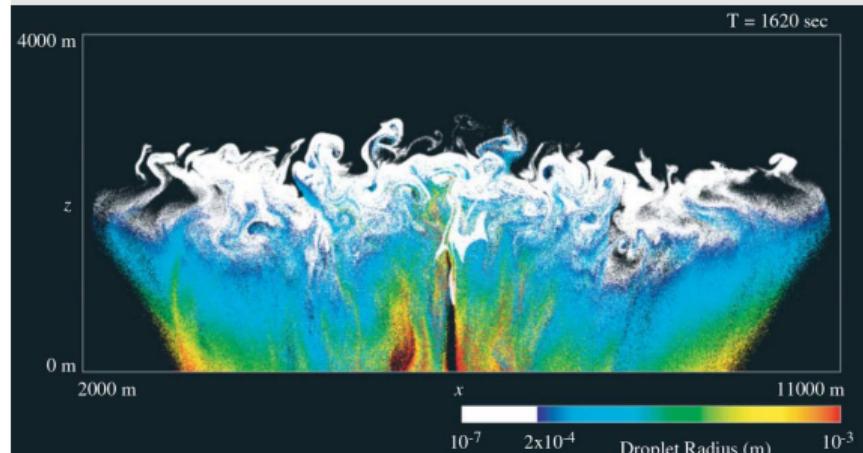
Eulerian vs. Lagrangian microphysics: a (probabilistic) breakthrough

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

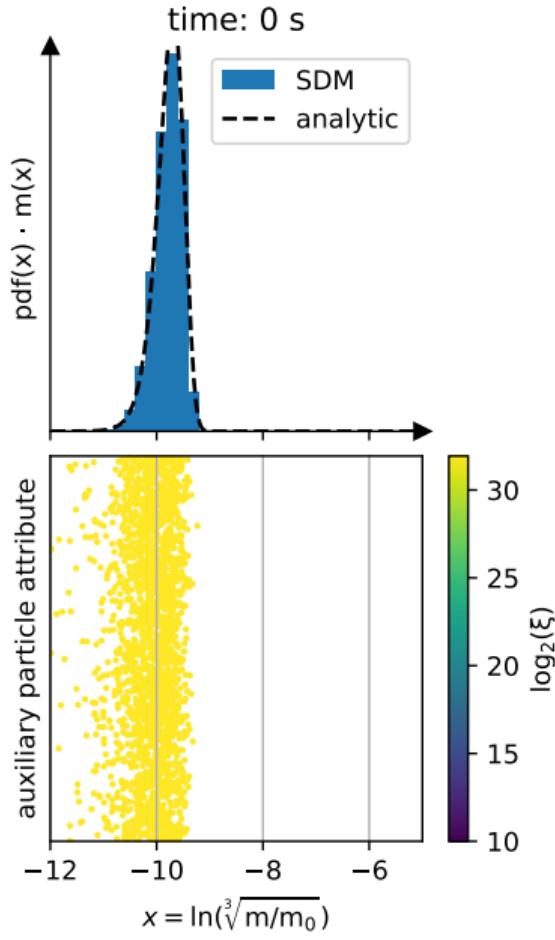
Shima 2009: Monte-Carlo particle-based collision algorithm for cloud simulations



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

^a Jacobson 2005: Fundamentals of Atmospheric Modeling

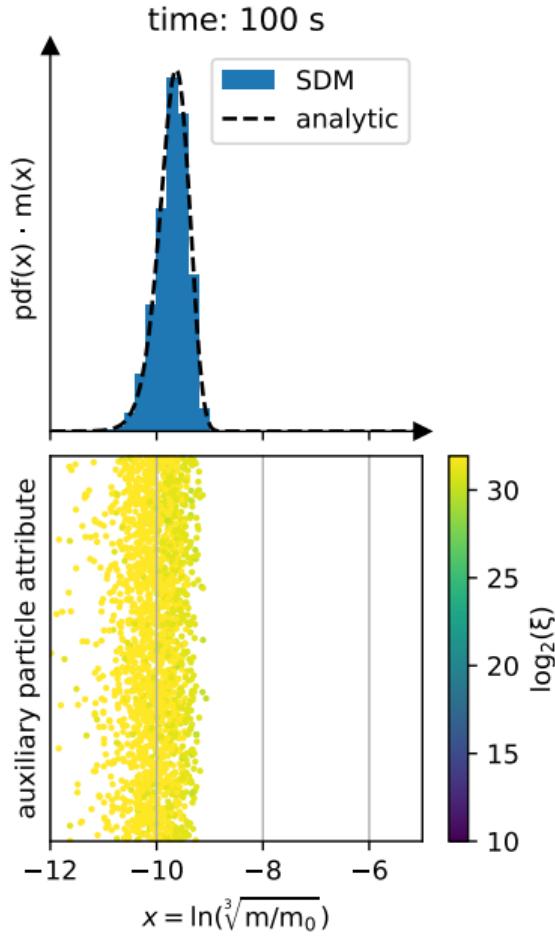
^b Simmel & Wurzler 2006: Condensation and activation in sectional cloud microphysical models



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

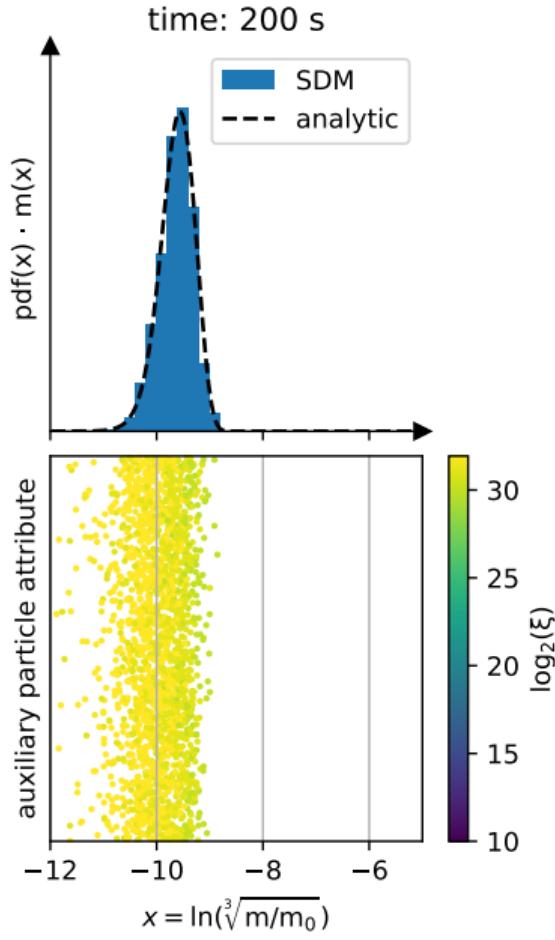
```



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

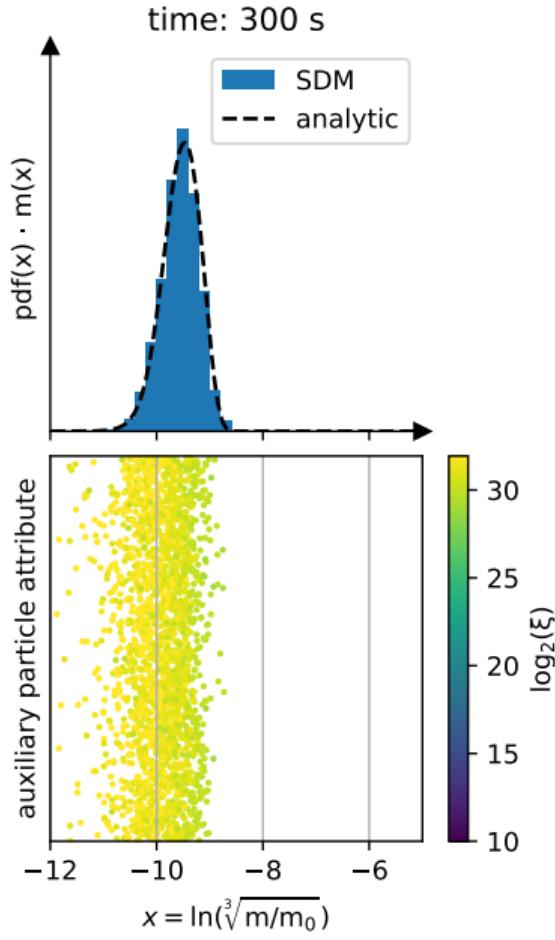
```



```

1  def sdm(*,
2      rng: np.random.Generator,
3      ξ: abc.MutableSequence[int],
4      m: abc.MutableSequence[float],
5      kern: abc.Callable[[float, float], float],
6      Δt: float,
7      Δv: float,
8      ):
9          """ SDM step assuming non-zero multiplicities """
10         n_s = len(ξ)
11         n_pair = n_s // 2
12         pairs = rng.permutation(n_s)[: 2 * n_pair]
13         φ = rng.uniform(0, 1, n_pair)
14         p_ratio = n_s * (n_s - 1) / 2 / n_pair
15         for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16             p_jk = kern(m[j], m[k]) * Δt / Δv
17             if ξ[j] < ξ[k]:
18                 j, k = k, j
19             p_α = ξ[j] * p_ratio * p_jk
20             γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21             if γ != 0:
22                 γ = min(γ, (ξ[j] / ξ[k]) // 1)
23                 if ξ[j] - γ * ξ[k] > 0:
24                     ξ[j] -= γ * ξ[k]
25                     m[k] += γ * m[j]
26                 else:
27                     ξ[j] = ξ[k] // 2
28                     ξ[k] -= ξ[j]
29                     m[k] += γ * m[j]
30                     m[j] = m[k]

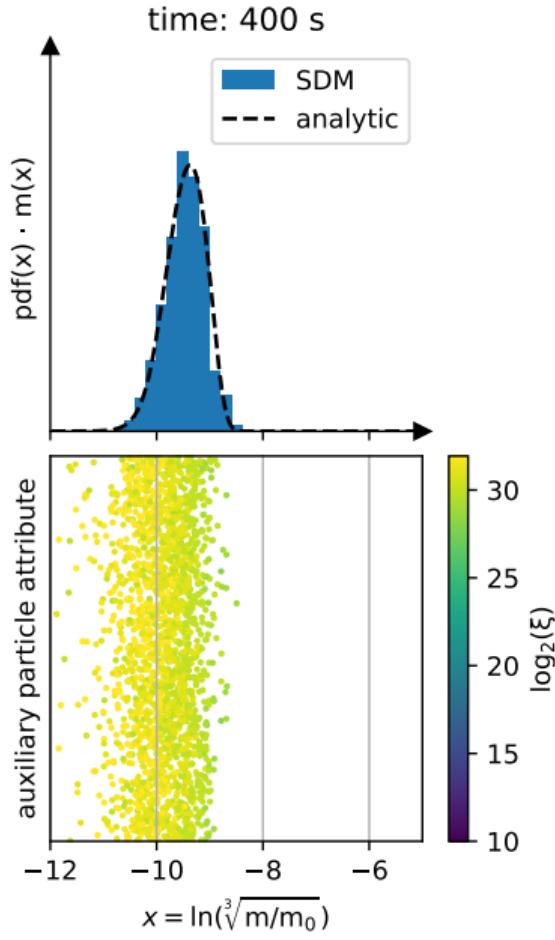
```



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

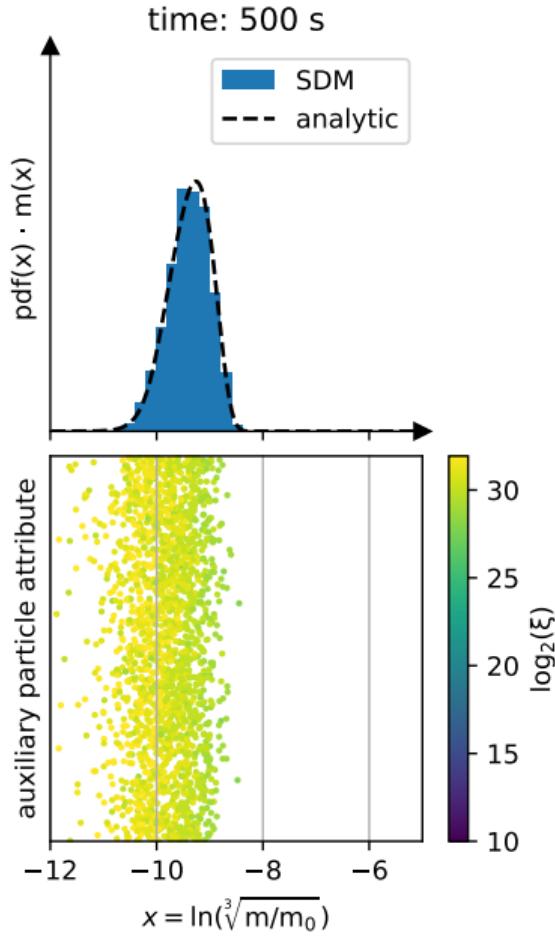
```



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

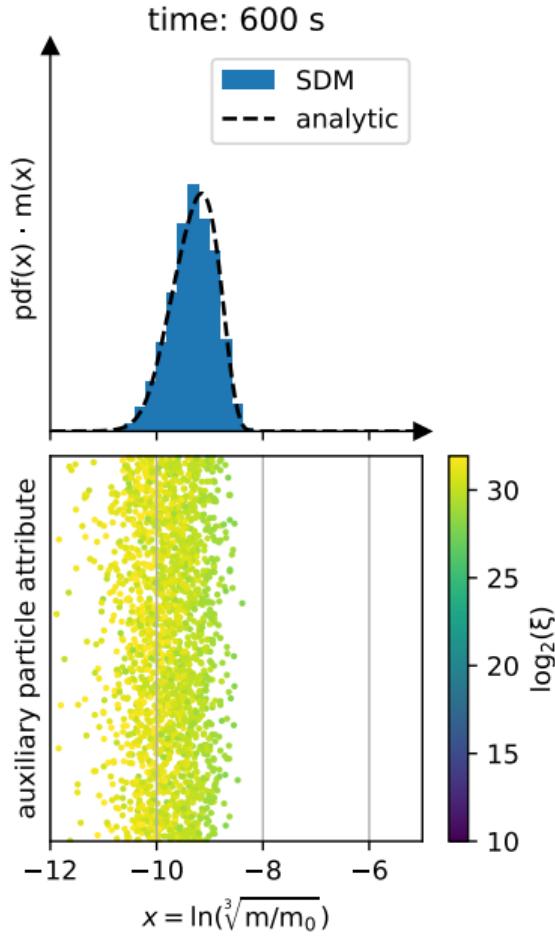
```



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

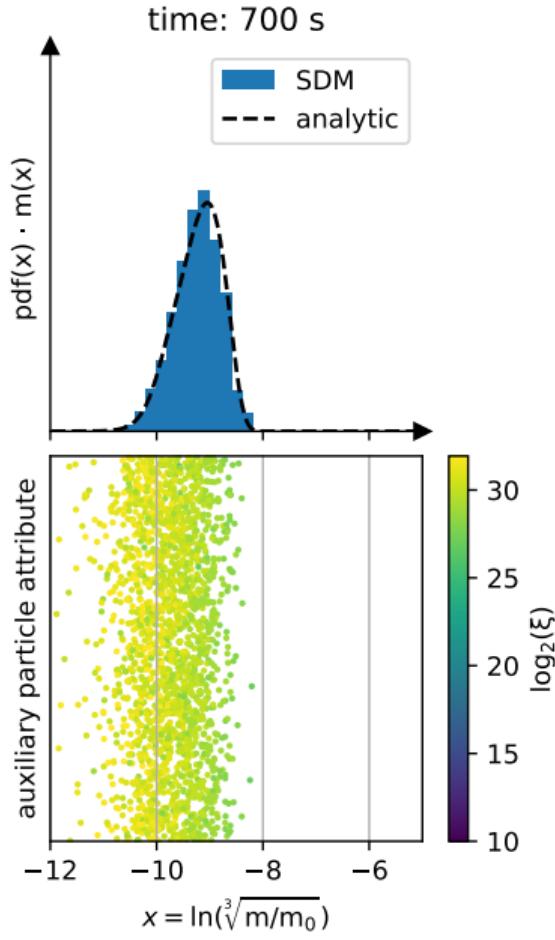
```



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

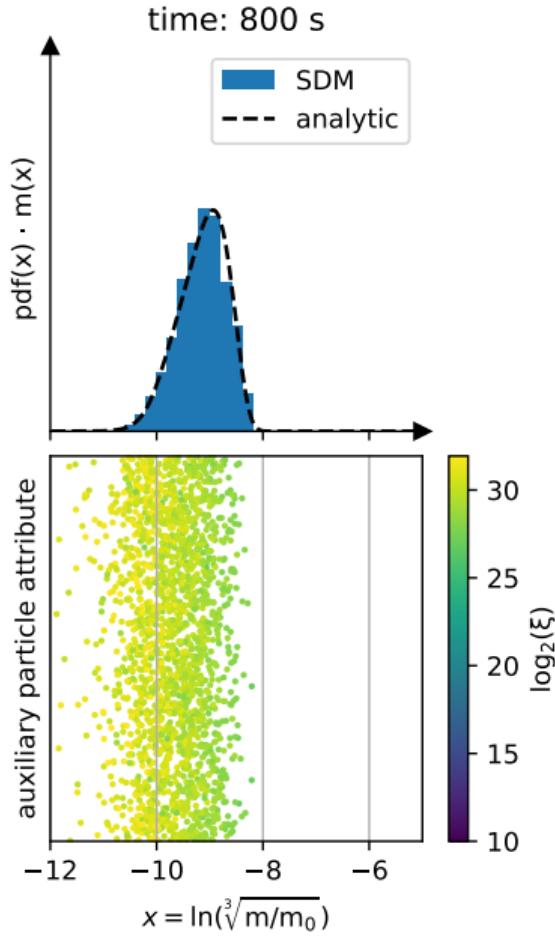
```



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

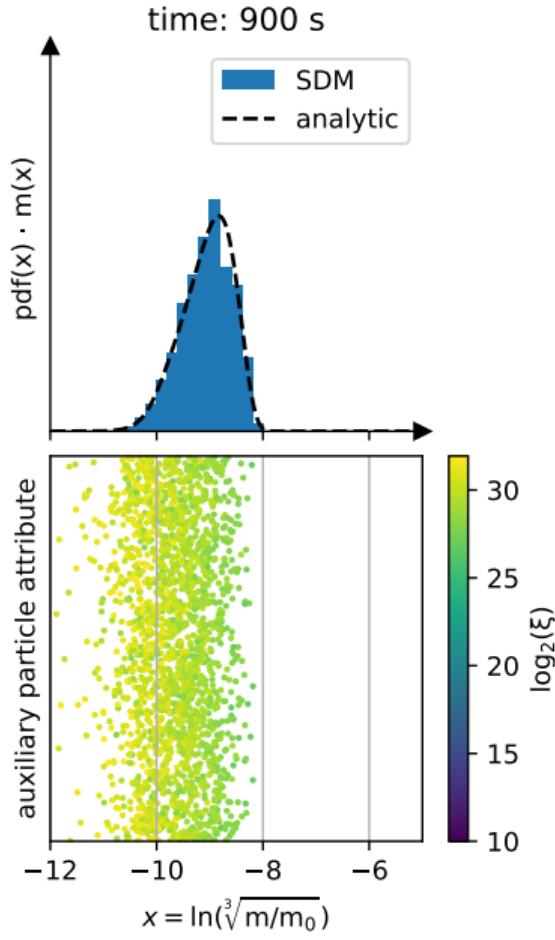
```



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

```

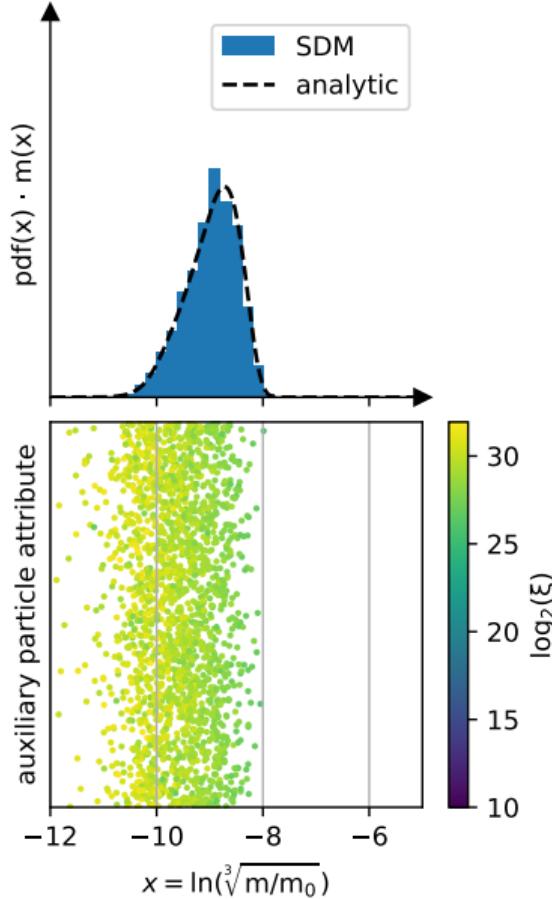


```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

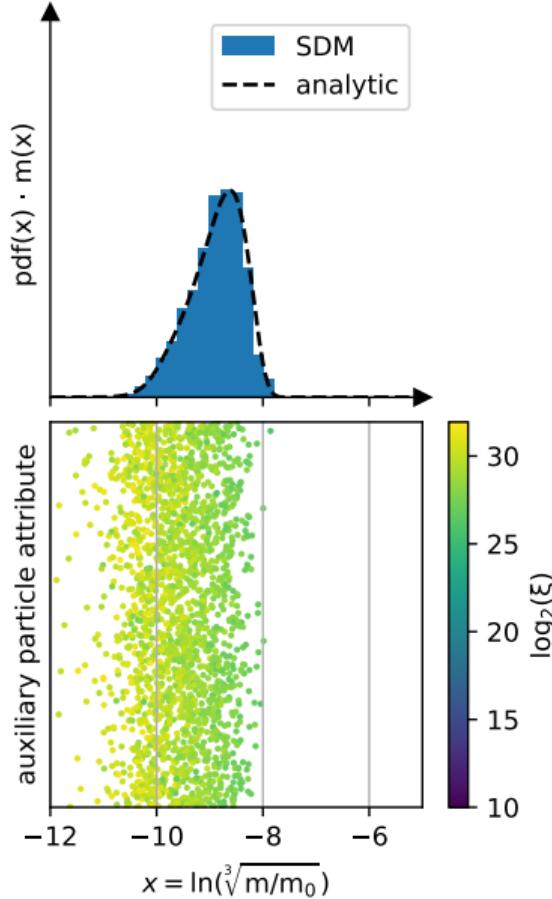
```

time: 1000 s



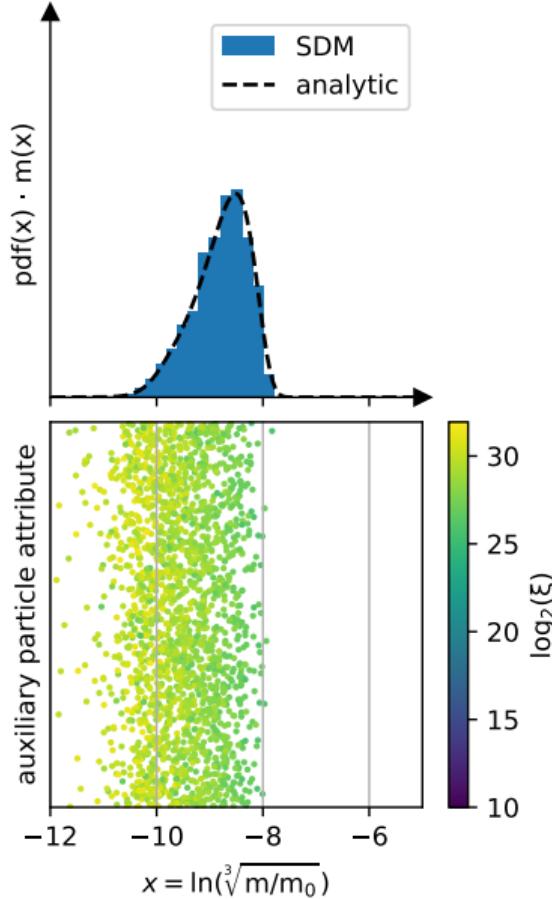
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 1100 s



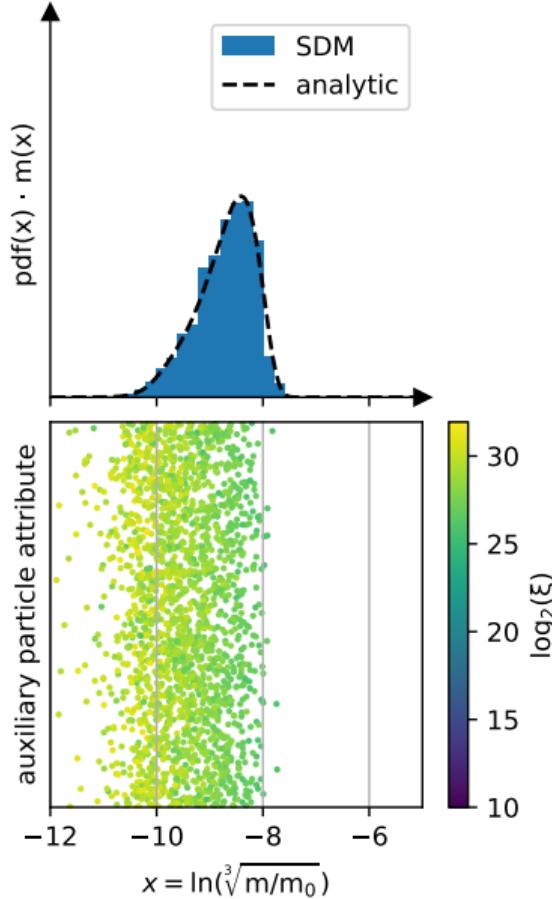
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 1200 s



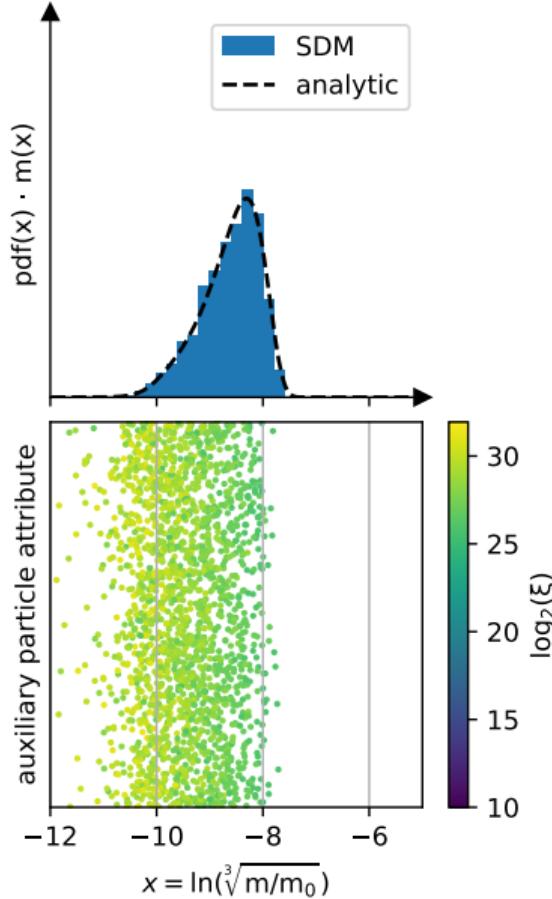
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 1300 s



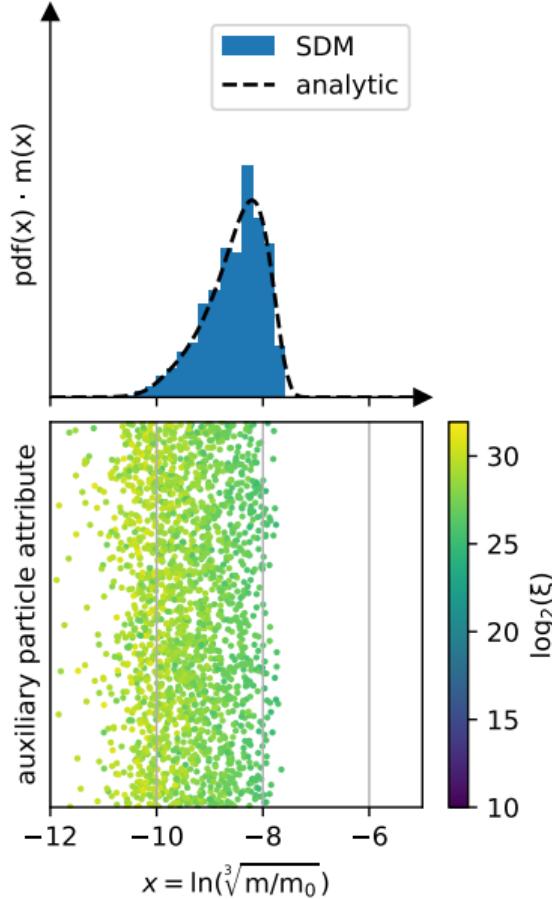
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 1400 s



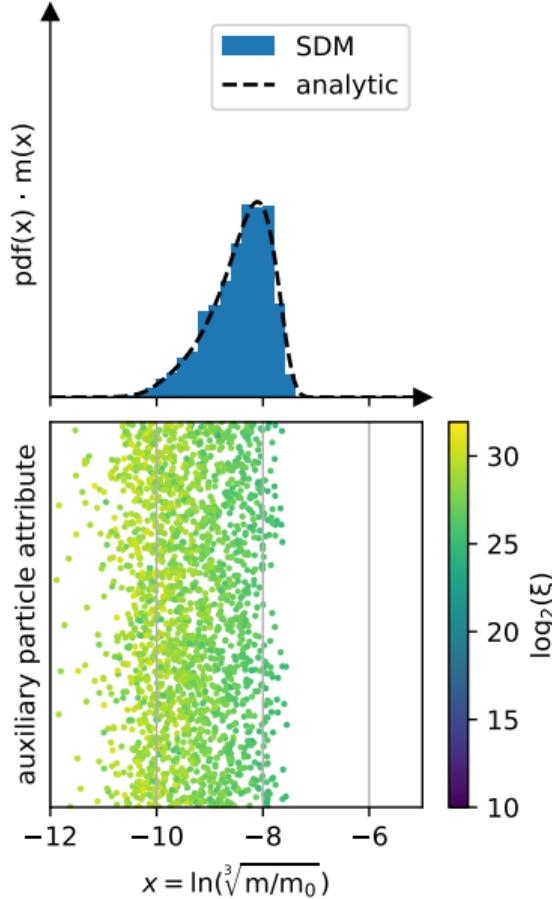
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 1500 s



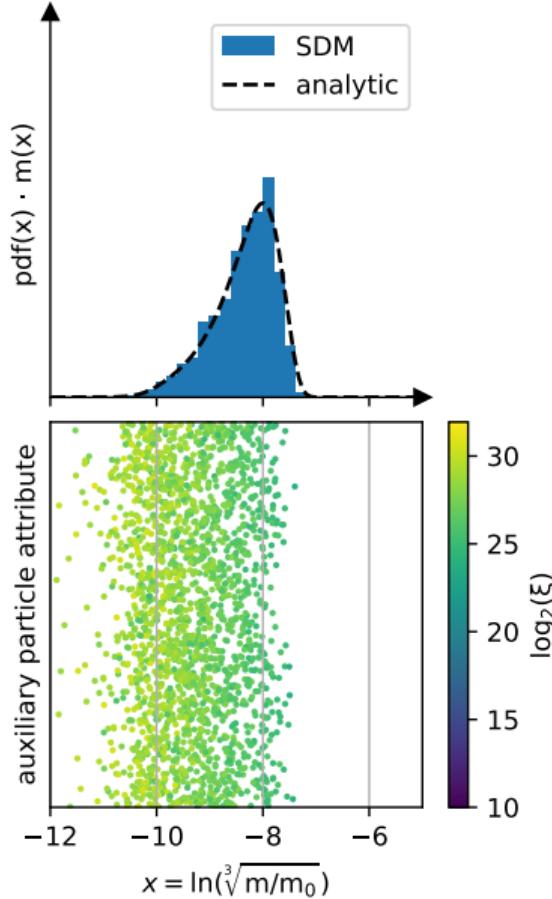
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 1600 s



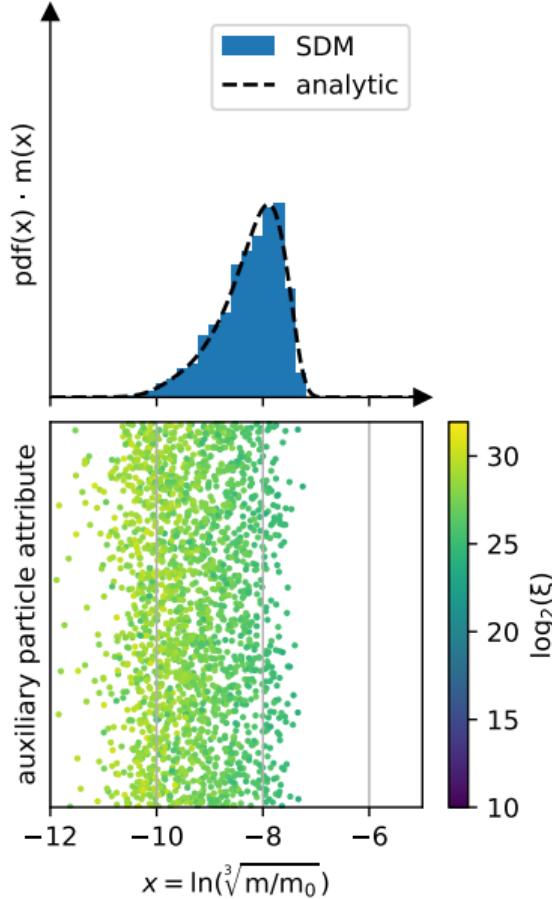
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 1700 s



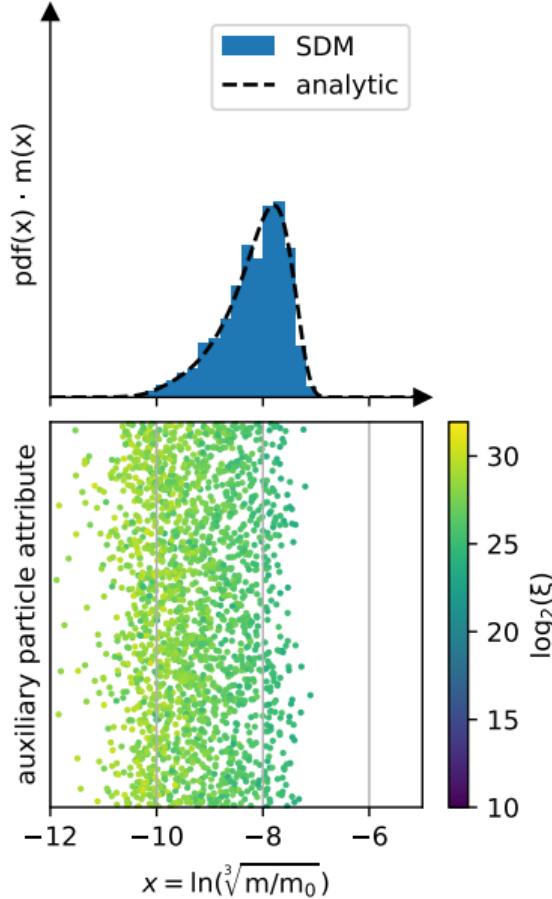
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 1800 s



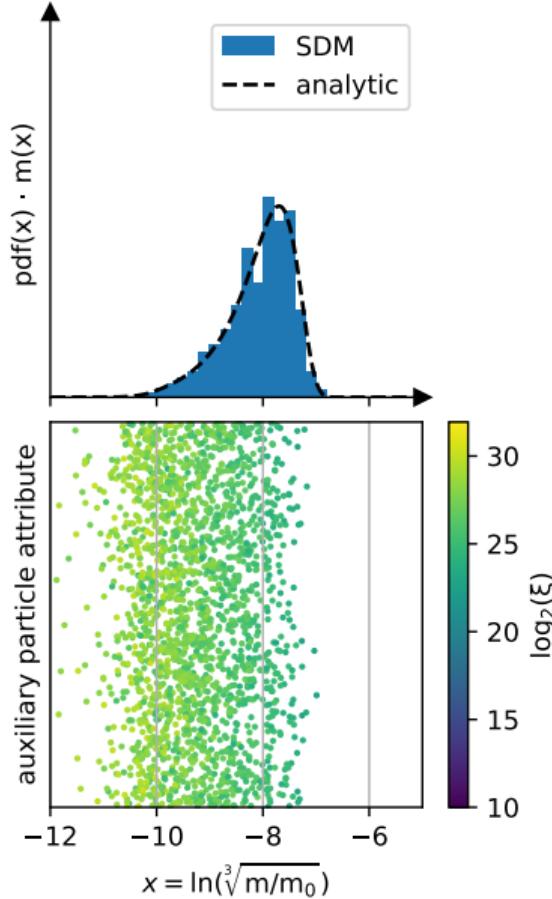
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 1900 s



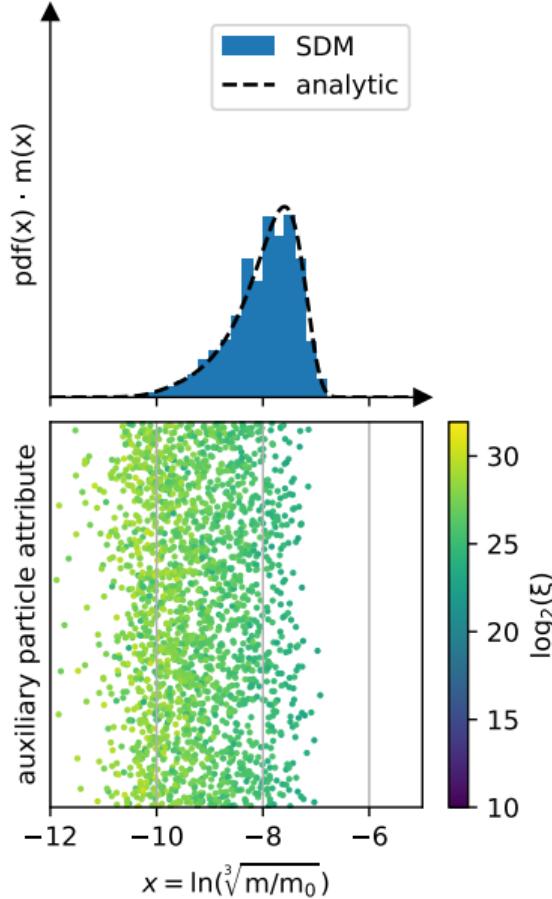
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 2000 s



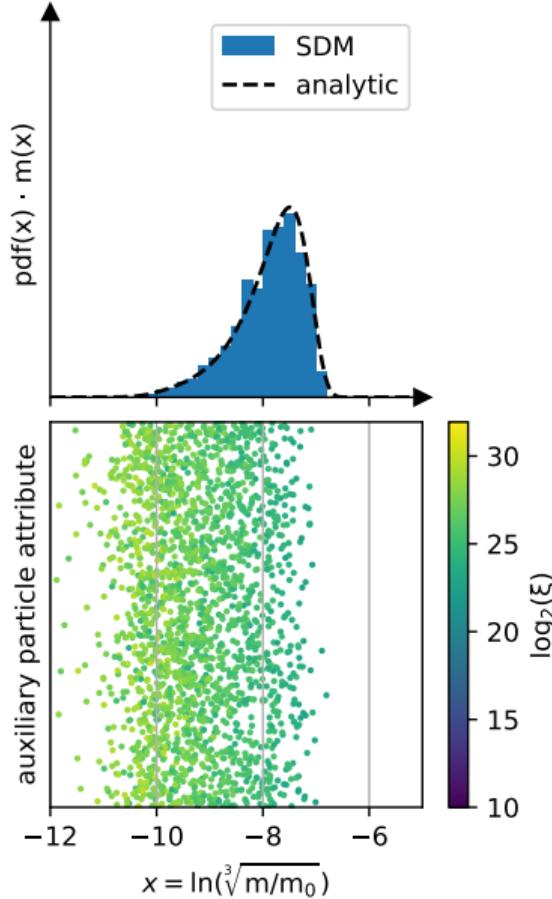
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 2100 s



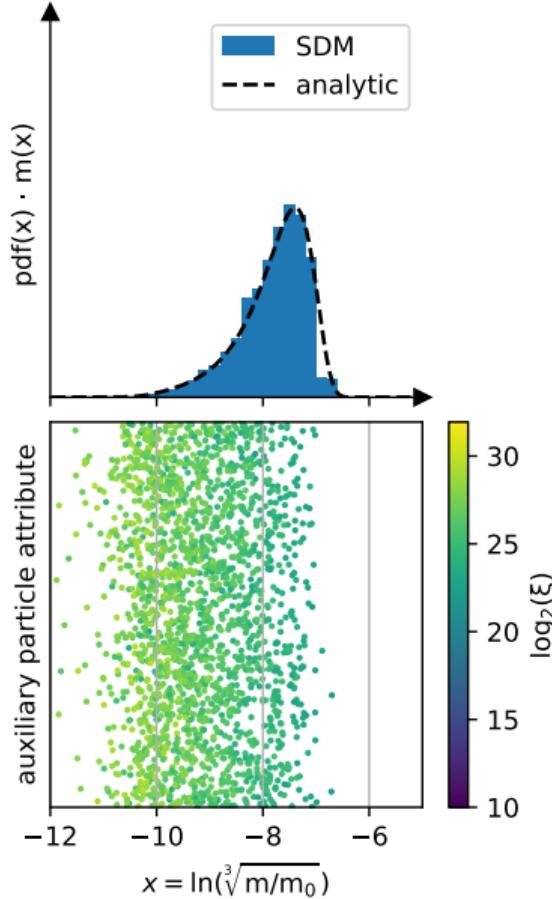
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 2200 s



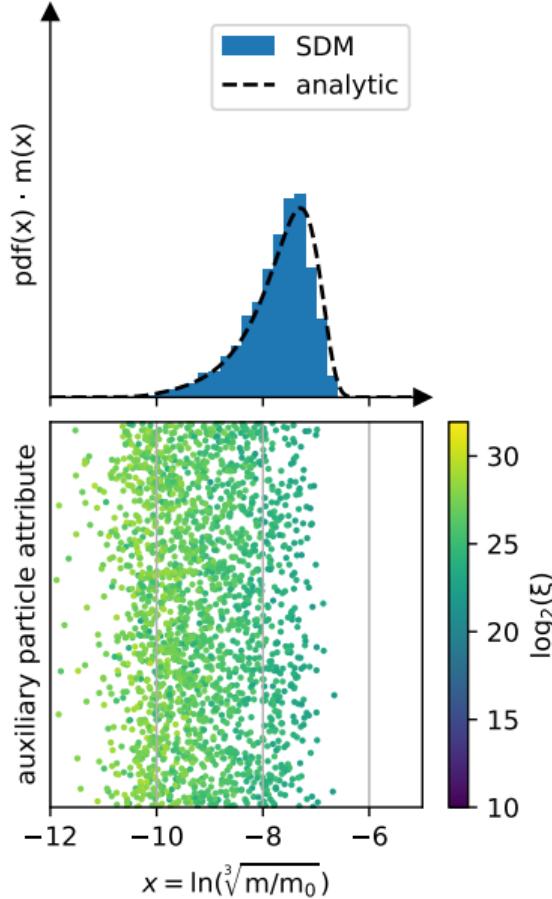
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 2300 s



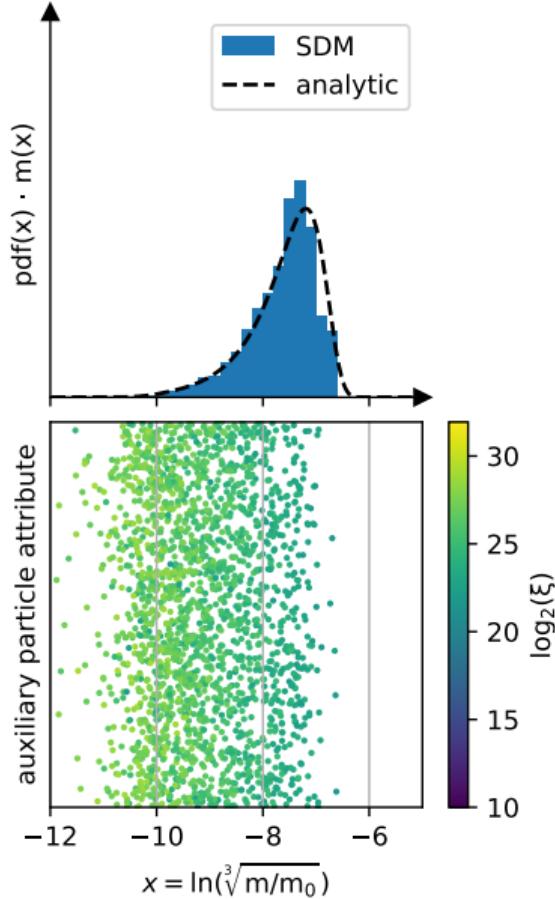
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 2400 s

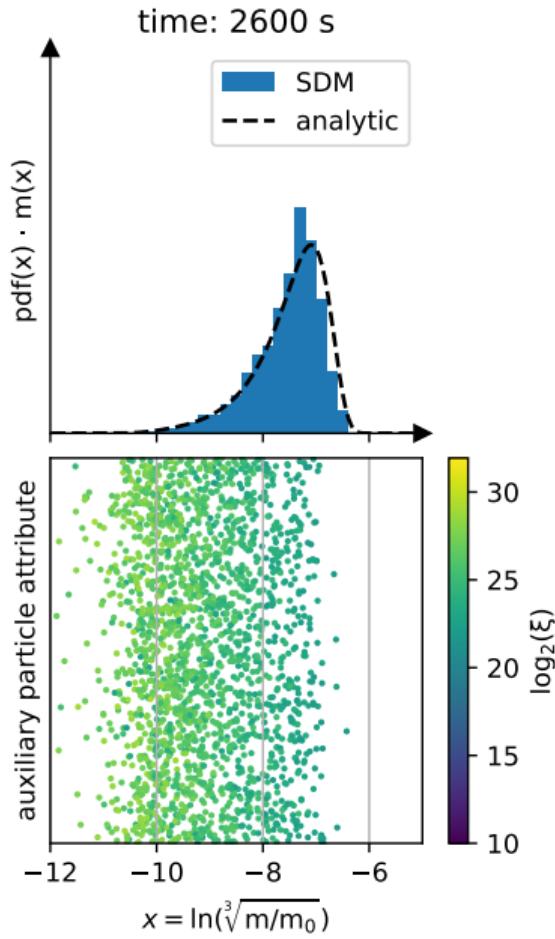


```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 2500 s



```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

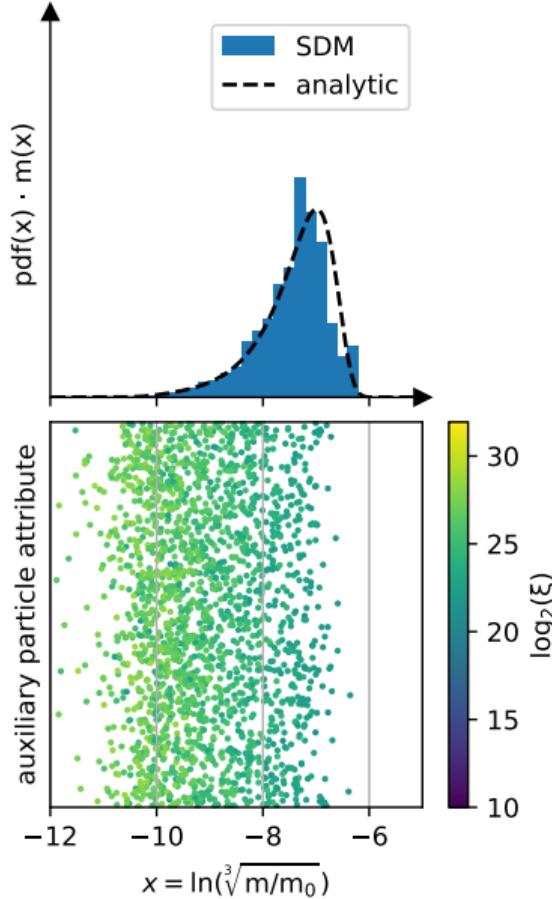


```

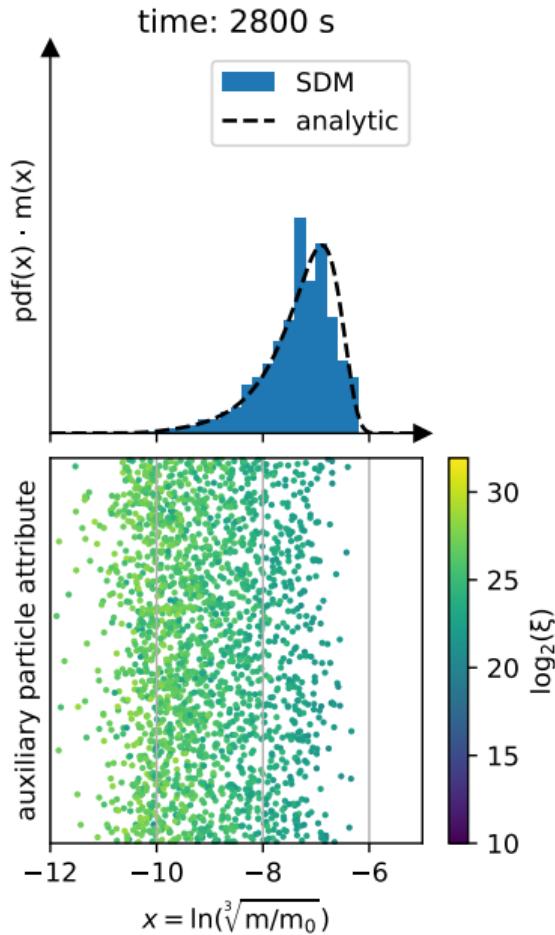
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

```

time: 2700 s



```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

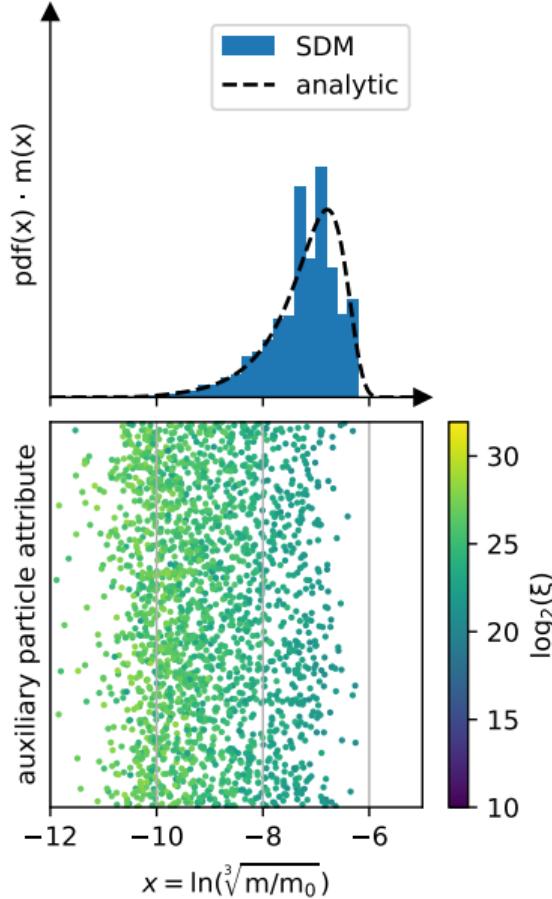


```

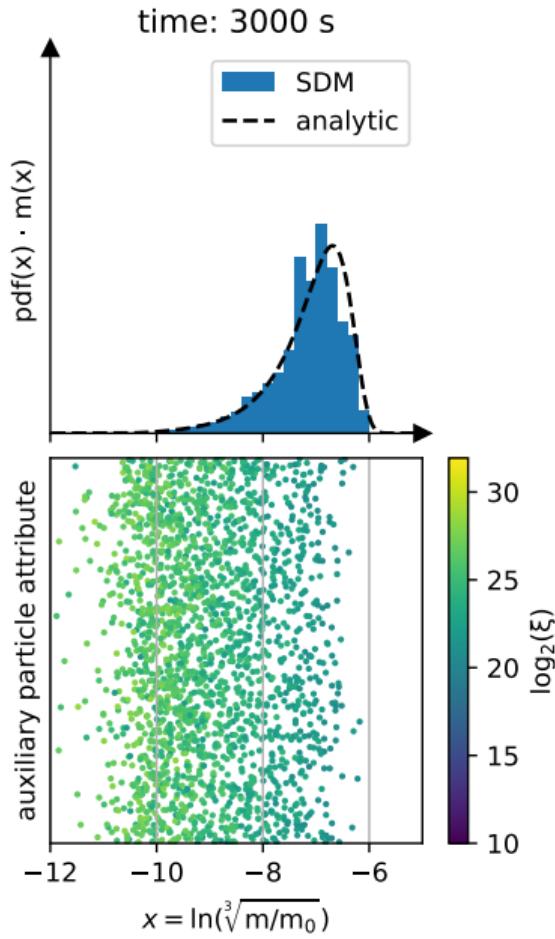
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

```

time: 2900 s



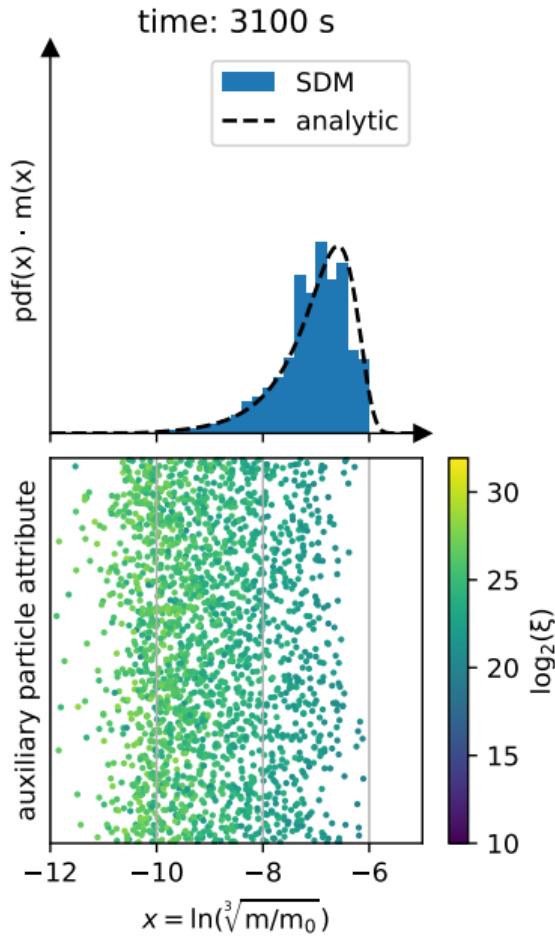
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

```

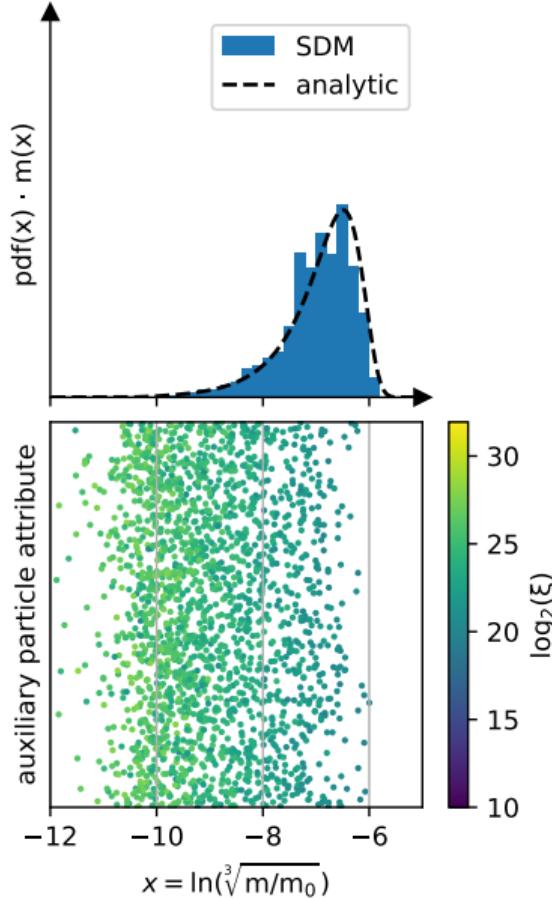


```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

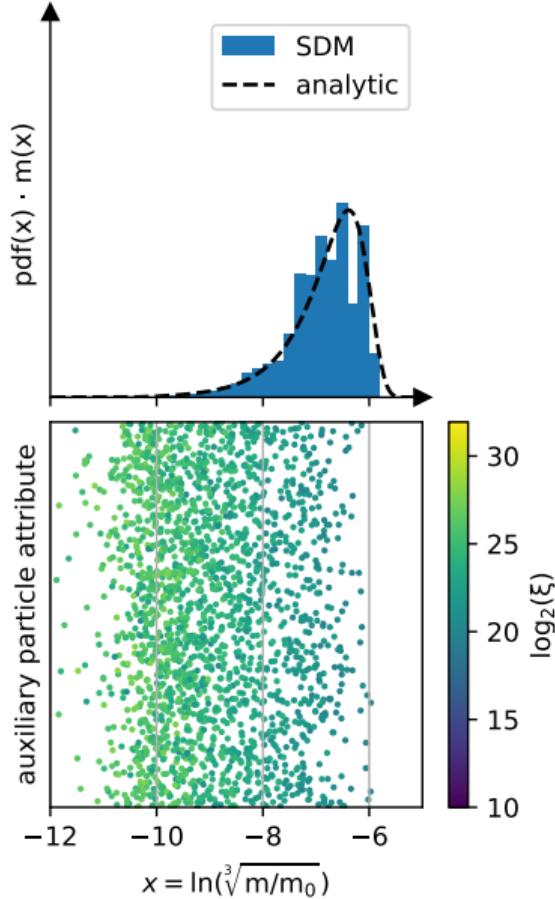
```

time: 3200 s

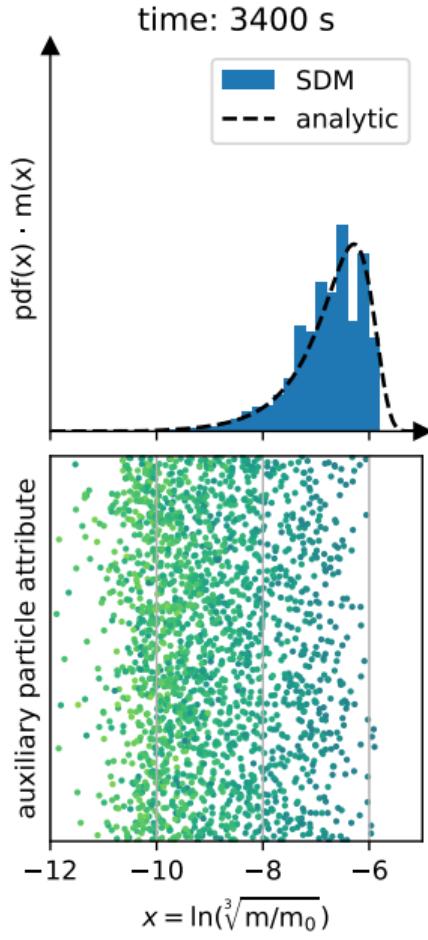


```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```

time: 3300 s



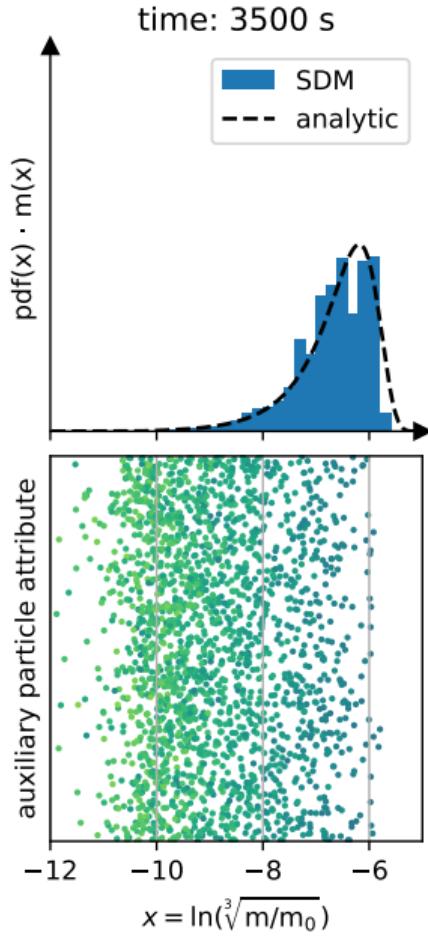
```
1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]
```



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

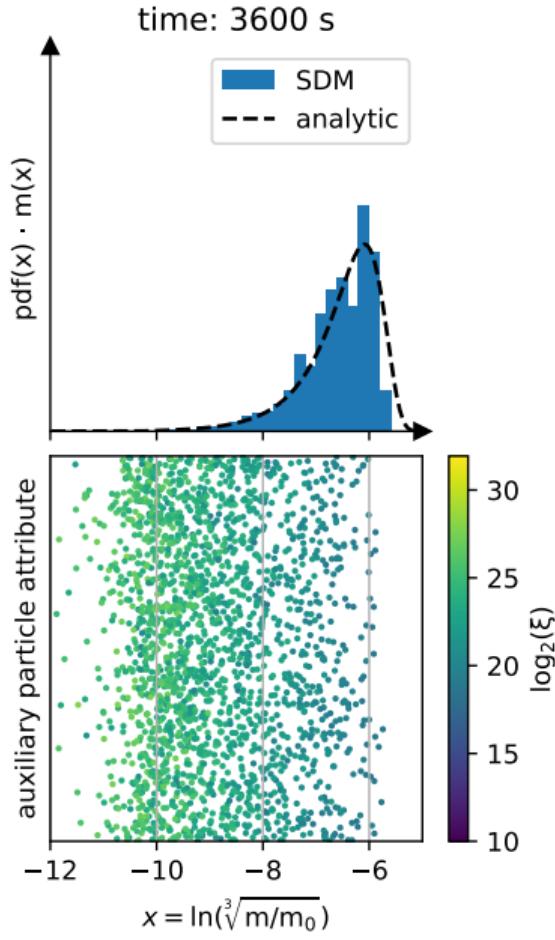
```



```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

```

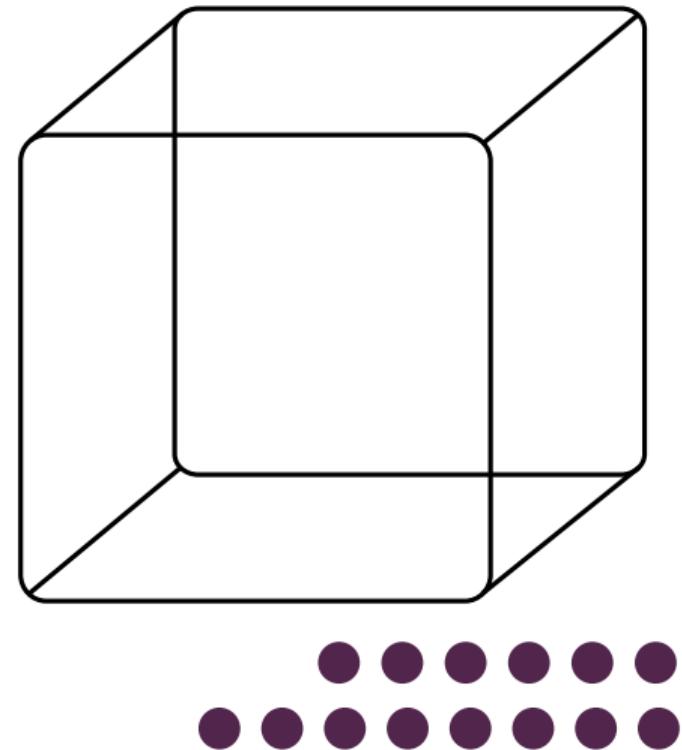


```

1 def sdm(*,
2     rng: np.random.Generator,
3     ξ: abc.MutableSequence[int],
4     m: abc.MutableSequence[float],
5     kern: abc.Callable[[float, float], float],
6     Δt: float,
7     Δv: float,
8     ):
9     """ SDM step assuming non-zero multiplicities """
10    n_s = len(ξ)
11    n_pair = n_s // 2
12    pairs = rng.permutation(n_s)[: 2 * n_pair]
13    φ = rng.uniform(0, 1, n_pair)
14    p_ratio = n_s * (n_s - 1) / 2 / n_pair
15    for α, (j, k) in enumerate(pairs.reshape(-1, 2)):
16        p_jk = kern(m[j], m[k]) * Δt / Δv
17        if ξ[j] < ξ[k]:
18            j, k = k, j
19        p_α = ξ[j] * p_ratio * p_jk
20        γ = p_α // 1 + ((p_α - p_α // 1) > φ[α])
21        if γ != 0:
22            γ = min(γ, (ξ[j] / ξ[k]) // 1)
23            if ξ[j] - γ * ξ[k] > 0:
24                ξ[j] -= γ * ξ[k]
25                m[k] += γ * m[j]
26            else:
27                ξ[j] = ξ[k] // 2
28                ξ[k] -= ξ[j]
29                m[k] += γ * m[j]
30                m[j] = m[k]

```

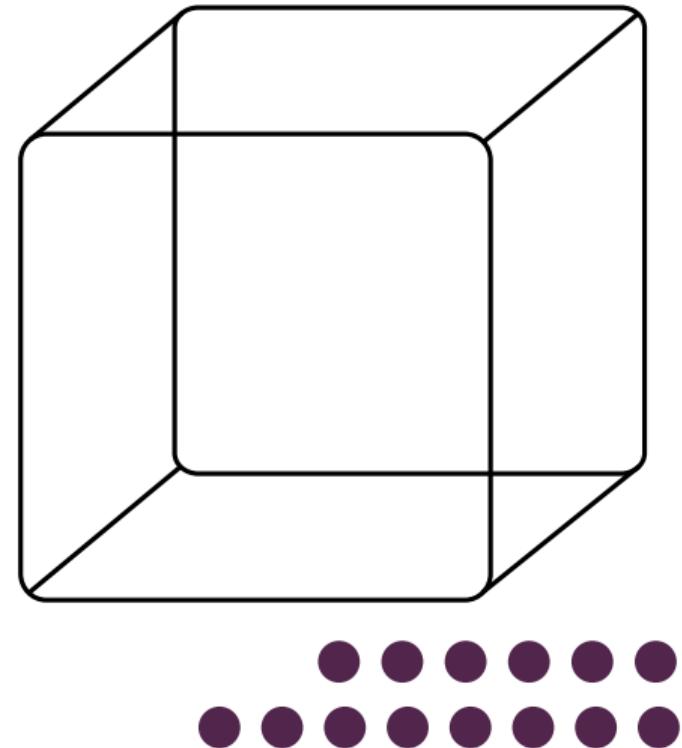
Domain randomly populated with " μ -physics information carriers"
(super particles / super droplets)

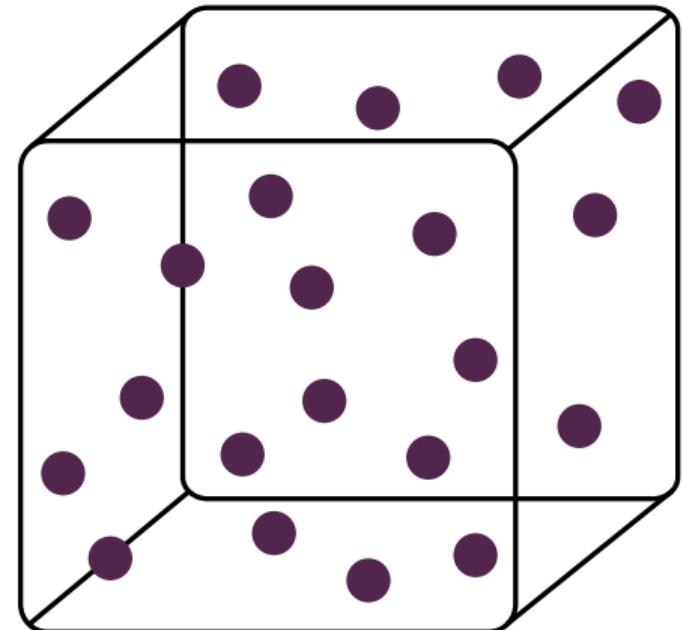


Domain randomly populated with " μ -physics information carriers"

(super particles / super droplets)

carrier attributes:

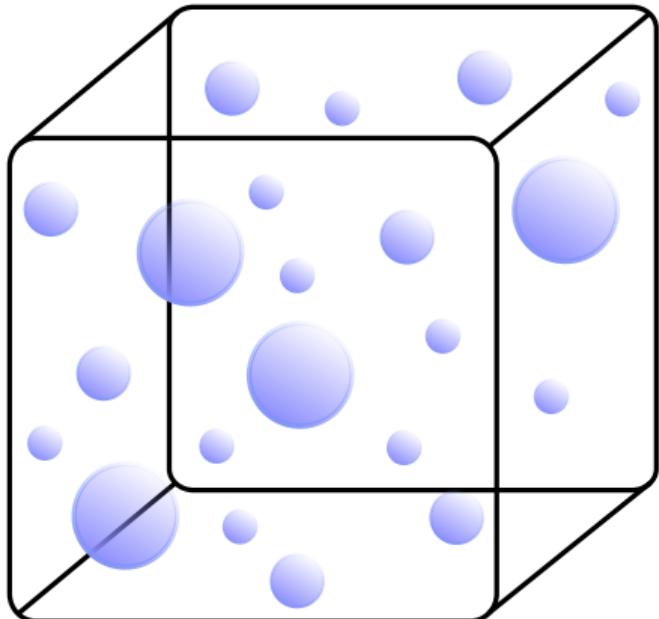




Domain randomly populated with " μ -physics information carriers"
(super particles / super droplets)

carrier attributes:

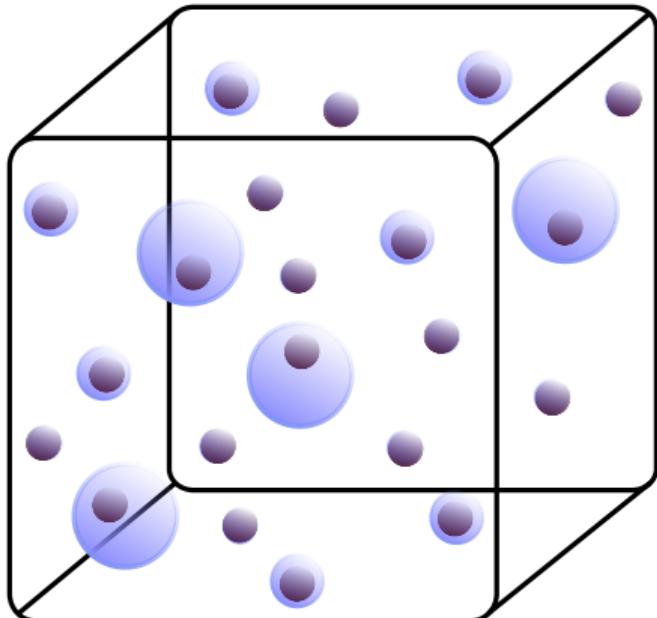
- location



Domain randomly populated with " μ -physics information carriers"
(super particles / super droplets)

carrier attributes:

- location
- wet radius



Domain randomly populated with " μ -physics information carriers"
(super particles / super droplets)

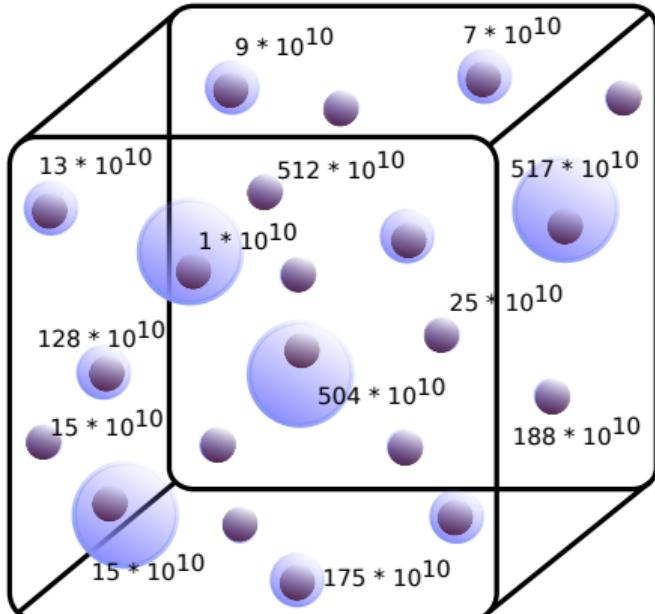
carrier attributes:

- location
- wet radius
- CCN/INP characteristics

Domain randomly populated with " μ -physics information carriers"
(super particles / super droplets)

carrier attributes:

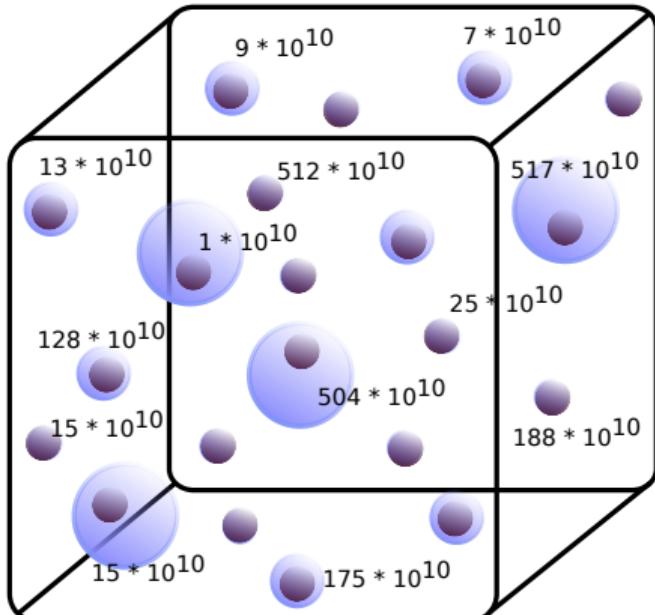
- location
- wet radius
- CCN/INP characteristics
- multiplicity



Domain randomly populated with " μ -physics information carriers"
(super particles / super droplets)

carrier attributes:

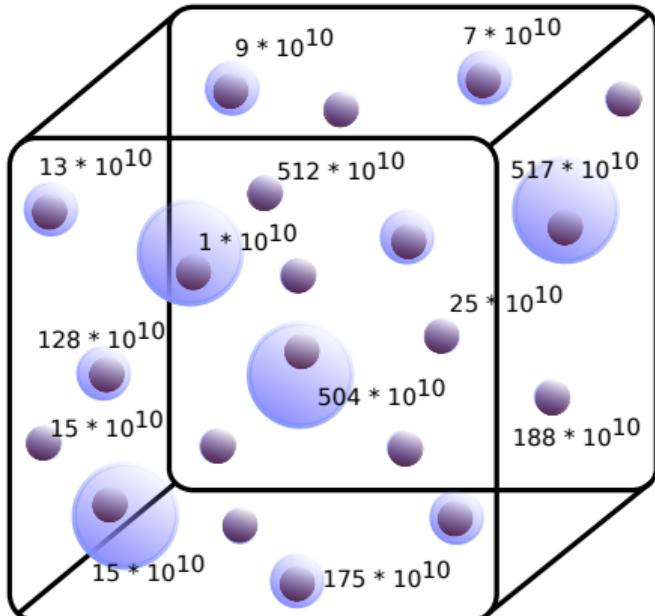
- location
- wet radius
- CCN/INP characteristics
- multiplicity
- ...



Domain randomly populated with " μ -physics information carriers"
(super particles / super droplets)

carrier attributes:

- location
- wet radius
- CCN/INP characteristics
- multiplicity
- ...
- freezing temperature?
- insoluble nucleus size?



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

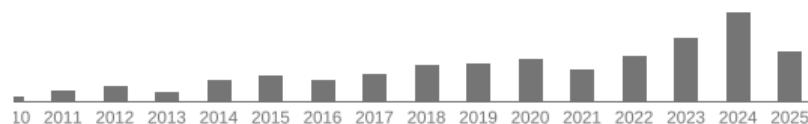
Journal Quarterly Journal of the Royal Meteorological Society

Volume 135

Issue 642

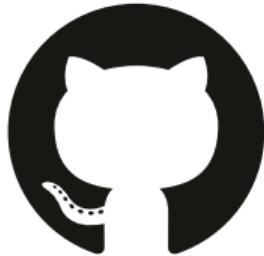
Pages 1307-1320

Total citations [Cited by 309](#)



**JIT-compiled SDM usable on Colab
for active learning and research
RSE goal: maintain reproducibility**

100%  python™ open-source code:



/ OPEN

ATMOS



/ **PySDM**

PySDM: example gallery

Literature Reference	Cond/Evap	Collisions	Isotopes	Breakup	Transport	Chemistry	Freezing	Comments/keywords
no-environment								
Merlivat & Nief 1967			x					
Gedzelman & Arnold 1994	x		x					
Pierchala et al. 2022			x					theoretical curves for a lab experiment
OD box environment								
Shima et al. 2009		x						Golovin kernel example
Berry 1967		x						Several different kernels
Bieli et al. 2022		x		x				
Alpert & Knopf 2016							x	
OD parcel environment								
Kreidenweis et al. 2003	x					x		"Hoppel" gap
Jaruga & Pawlowska 2018	x					x		"Hoppel" gap
Lowe et al. 2019	x							surfactants
Yang et al. 2018	x							ripening (depending on the definition)
Graf et al. 2019	x		x					
Arabas and Shima 2017	x							monodisperse, activation/deactivation cycle
Abdul-Razzak & Ghan 2000	x							parcel vs. activation parameterisation
1D single-column kinematic env.								
Shipway & Hill 2012	x	x			x			KiD 1D
deJong et al. 2023 (figures 6-8)	x	x		x	x			
2D prescribed-flow environment								
Arabas et al. 2015	x	x			x			includes GUI
Arabas et al. 2025	x	x			x		x	Paraview automation example

PySDM: smoke tests



open-atmos / PySDM

Type ⌘ to search



Code

Issues 193

Pull requests 38

Discussions

Actions

Wiki

Security

Insights

Settings



main ▾

PySDM / tests / smoke_tests / parcel_c / abdul_razzak_ghan_2000 / test_ARG_example.py



slayoo split unit tests, parcel smoke tests and multi-process/condensation e...

c38c81f · 2 years ago History



main ▾

PySDM / tests / smoke_tests / parcel_c / abdul_razzak_ghan_2000 / test_ARG_example.py

↑ Top

Code

Blame



Raw



```
3      import pytest
4      from PySDM_examples.Abdul_Razzak_Ghan_2000 import data_from_ARG2000_paper as ARG_paper
5      from PySDM_examples.Abdul_Razzak_Ghan_2000.run_ARG_parcel import run_parcel
6
7      from PySDM.physics import si
8
9
10     class TestARGExample: # pylint: disable=too-few-public-methods
11         @staticmethod
12         @pytest.mark.parametrize("N2i", np.linspace(100, 5000, 5) / si.cm**3)
13         def test_ARG_fig1(N2i):
14             w = 0.5 * si.m / si.s
15             sol2 = 1.0 # 100% ammonium sulfate
16             rad2 = 50.0 * si.nm
17
18             n_sd_per_mode = 10
19
20             idx = np.argmin(np.abs(ARG_paper.Fig1_N2_param - N2i * si.cm**3))
21             output = run_parcel(w, sol2, N2i, rad2, n_sd_per_mode)
22
23             assert np.isclose(
24                 output.activated_fraction_S[0],
25                 ARG_paper.Fig1_AF_param[idx],
26                 atol=output.error[0] * 2,
27             )
```



b06464d

PySDM / tests / smoke_tests / parcel_d / jensen_and_nugent_2017 / test_fig_6.py

↑ Top

Code

Blame

80 lines (65 loc) · 2.58 KB



```
24     @pytest.fixture(scope="session", name="variables")
25     def variables_fixture():
26         return notebook_vars(
27             file=Path(Jensen_and_Nugent_2017.__file__).parent / "Fig_6.ipynb", plot=PLOT
28         )
29
30
31     class TestFig6:
32
33
34         @staticmethod
35         @pytest.mark.parametrize("drop_id", range(int(0.77 * N_SD), N_SD))
36         def test_radii(variables, drop_id):
37             """checks that the largest aerosol activate and still grow upon descent"""
38             # arrange
39             cb_idx = find_cloud_base_index(variables["output"]["products"])
40             ma_idx = find_max_alt_index(variables["output"]["products"])
41
42             radii = variables["output"]["attributes"]["radius"][drop_id]
43             r1 = radii[0]
44             r2 = radii[cb_idx]
45             r3 = radii[ma_idx]
46
47             assert r1 < r2 < r3
```

PySDM: units and unit tests

```
$ cloc tests/unit_tests
```

Language	files	blank	comment	code
Python	119	1862	1214	10912
SUM:	119	1862	1214	10912

```
$ cloc tests/smoke_tests
```

Language	files	blank	comment	code
Python	87	942	478	5228
SUM:	87	942	478	5228

```
$ cloc PySDM
```

Language	files	blank	comment	code
Python	424	3468	3604	17624
SUM:	424	3468	3604	17624

Files ↑	Tracked lines	Covered	Partial	Missed	Coverage %
📁 attributes	551	502	0	49	<div style="width: 91.11%;"><div style="width: 91.11%;"></div></div> 91.11%
📁 backends	3619	3235	0	384	<div style="width: 89.39%;"><div style="width: 89.39%;"></div></div> 89.39%
📁 dynamics	1229	1074	0	155	<div style="width: 87.39%;"><div style="width: 87.39%;"></div></div> 87.39%
📁 environments	273	223	0	50	<div style="width: 81.68%;"><div style="width: 81.68%;"></div></div> 81.68%
📁 exporters	286	75	0	211	<div style="width: 26.22%;"><div style="width: 26.22%; background-color: red;"></div></div> 26.22%
📁 impl	204	199	0	5	<div style="width: 97.55%;"><div style="width: 97.55%;"></div></div> 97.55%
📁 initialisation	375	355	0	20	<div style="width: 94.67%;"><div style="width: 94.67%;"></div></div> 94.67%
📁 physics	1985	1701	0	284	<div style="width: 85.69%;"><div style="width: 85.69%;"></div></div> 85.69%
📁 products	1279	1062	0	217	<div style="width: 83.03%;"><div style="width: 83.03%;"></div></div> 83.03%
📄 __init__.py	11	9	0	2	<div style="width: 81.82%;"><div style="width: 81.82%;"></div></div> 81.82%
📄 builder.py	89	80	0	9	<div style="width: 89.89%;"><div style="width: 89.89%;"></div></div> 89.89%
📄 formulae.py	206	203	0	3	<div style="width: 98.54%;"><div style="width: 98.54%;"></div></div> 98.54%
📄 particulator.py	159	147	0	12	<div style="width: 92.45%;"><div style="width: 92.45%;"></div></div> 92.45%
Subtotal	10266	8865	0	1401	

PySDM: IoC for physical formulae

```
based on Table 1 from B.Bolin 1958  
"On the use of tritium as a tracer for water in nature"  
(https://digilibRARY.un.org/record/3892725)
```

```
import os, sys  
os.environ['NUMBA_THREADS_LAYER'] = 'workqueue'  
if 'google.colab' in sys.modules:  
    !pip --quiet install open-atmos-jupyter-utils  
    from open_atmos_jupyter_utils import pip_install_on_colab  
    pip_install_on_colab('PySDM-examples', 'PySDM')  
  
import numpy as np  
from PySDM import Formulae  
  
formulae = Formulae(  
    terminal_velocity = "RogersYau",  
    drop_growth = "Mason1971",  
    diffusion_thermics = "Neglect",  
    saturation_vapour_pressure = "AugustRocheMagnus",  
    ventilation = "Froessling1938",  
    particle_shape_and_density = "LiquidSpheres",  
    air_dynamic_viscosity = "ZografosEtAll1987",  
    isotope_equilibrium_fractionation_factors = "VanHook1968",  
    isotope_diffusivity_ratios = "GrahamsLaw",  
    constants = {"BOLIN_ISOTYPE_TIMESCALE_COEFF_C1": 1.63},  
    isotope_relaxation_timescale = "Bolin1958",  
)
```

PySDM companion packages:
PyMPDATA & Numba-MPI

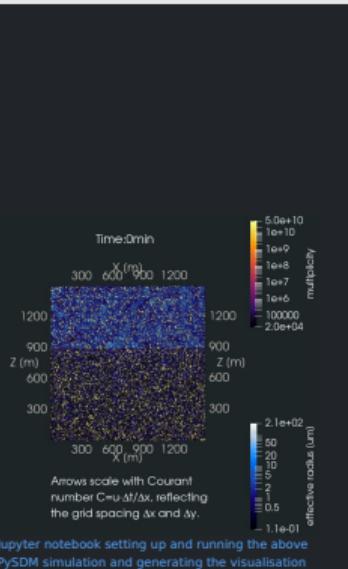
docstring-based docs with automatic graphics updates & code-linked bibliography

<https://open-atmos.github.io/PySDM>



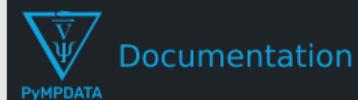
What is PySDM?

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-Dropot 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).



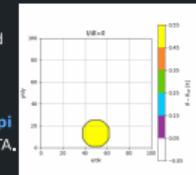
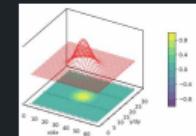
Jupyter notebook setting up and running the above PySDM simulation and generating the visualisation using Paraview

<https://open-atmos.github.io/PyMPDATA>



What is PyMPDATA?

PyMPDATA is a **Numba-accelerated** multi-threaded Pythonic implementation of the **MPDATA algorithm** of Smolarkiewicz et al., used in geophysical fluid dynamics and beyond for **numerically solving generalised convection-diffusion PDEs**. PyMPDATA supports integration in 1D, 2D and 3D structured meshes with optional coordinate transformations. The first animation shown depicts a "hello-world" 2D advection-only simulation with dotted lines indicating **domain decomposition** across three threads. The second animation depicts an MPDATA solution to coupled mass and momentum conservation equations for a buoyancy-driven flow in Boussinesq approximation (see Jaruga et al. 2015 example).



A separate project called **PyMPDATA-MPI** depicts how **numba-mpi** can be used to enable **distributed memory parallelism** in PyMPDATA.



🔍

[Help](#) [Docs](#) [Sponsors](#) [Log in](#) [Register](#)

pympdata-mpi 0.1.1

✓ [Latest version](#)

`pip install pympdata-mpi` ⬇️

Released: Apr 4, 2025

PyMPDATA + numba-mpi coupler sandbox

Navigation

☰ Project description

⌚ Release history

⬇️ Download files

Verified details ✓

These details have been [verified by PyPI](#)

Maintainers



Sfonxu

Project description

PyMPDATA-MPI

Python 3 LLVM Numba Linux macOS Maintained? yes

PL Funding by NCN License GPL v3 Copyright Jagiellonian University DOI 10.5281/zenodo.10866521

pull requests 7 open pull requests 131 closed

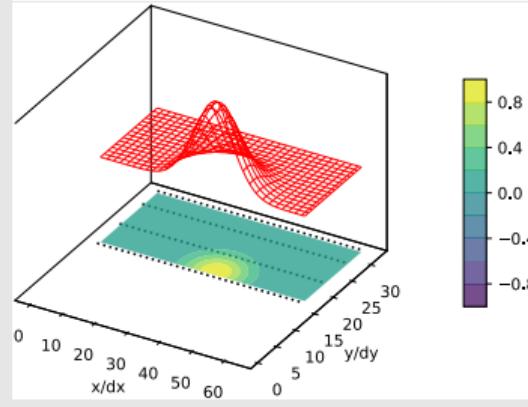
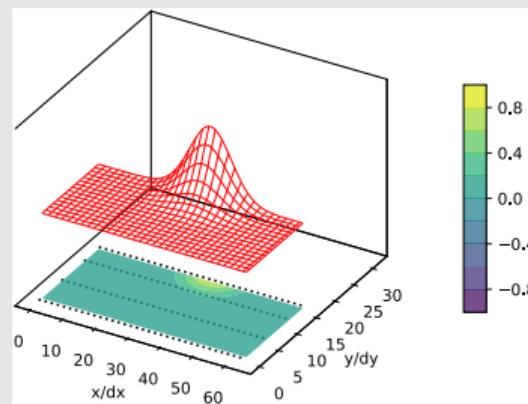
issues 14 open issues 36 closed

tests+pypi no status pypi package 0.1.1 docs pdoc.dev codecov 72%

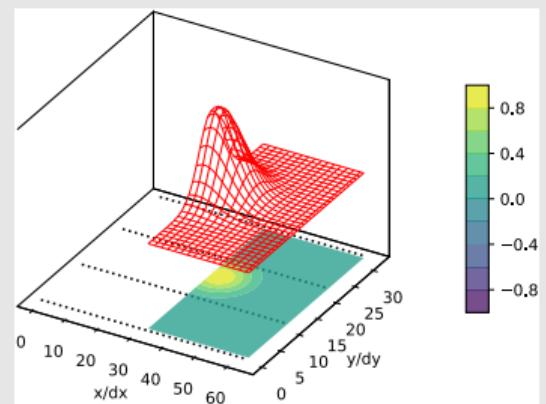
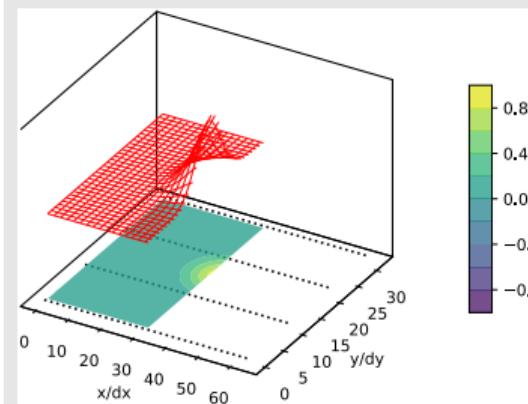
PyMPDATA-MPI constitutes a [PyMPDATA](#) + [numba-mpi](#) coupler enabling numerical solutions of transport equations with the MPDATA numerical scheme in a hybrid parallelisation model with both multi-threading and MPI distributed memory communication. PyMPDATA-MPI adapts to API of PyMPDATA offering domain decomposition logic.

PyMPDATA-MPI: customisable hybrid threading + MPI parallelisation

threading dim = MPI dim



threading dimension \neq MPI dimension



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

Under a Creative Commons license 

● Open access

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.

kudos team & contributors!

students in Kraków:

Graduate Students (@agh.edu.pl)



Agnieszka Zaba

Ongoing PhD project co-advised with Miroslaw 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 Strząbala

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 Bhogade

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 Butenok

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

**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>
© Author(s) 2023. This work is distributed under the Creative Commons Attribution 4.0 License.



Geoscientific
Model Development
Open Access


Development and technical paper

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

Emily de Jong¹, John Ben Mackay^{2,a}, Oleksii Bulenok³, Anna Jaruga⁴, and Sylwester Arabas^{5,b,c}

¹Department of Mechanical and Civil Engineering, California Institute of Technology, Pasadena, CA, USA

²Scripps Institution of Oceanography, San Diego, CA, USA

³Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

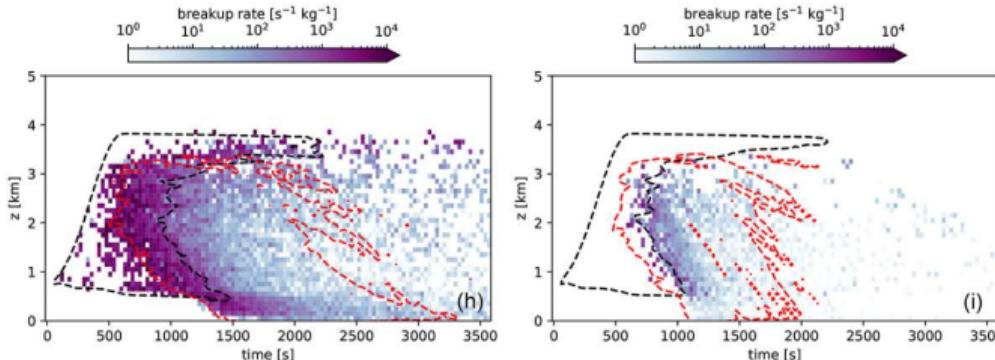
⁴Department of Environmental Science and Engineering, California Institute of Technology, Pasadena, CA, USA

⁵Faculty of Physics and Applied Computer Science, AGH University of Krakow, Kraków, Poland

^aformerly at: Department of Environmental Science and Engineering, California Institute of Technology, Pasadena, CA, USA

^bformerly at: Department of Atmospheric Sciences, University of Illinois Urbana-Champaign, Urbana, IL, USA

^cformerly at: Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland



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

Emily de Jong et al.



Code and data availability

Implementation of this breakup algorithm in the SDM is available at <https://doi.org/10.5281/zenodo.7851352> (Arabas et al., 2023a). The simulations presented in this work (and all necessary input information) are available in the folder "deJong_Mackay_2022" at <https://doi.org/10.5281/zenodo.7851288> (Arabas et al., 2023b). The notebooks in this folder reproduce all results and figures presented in this study, with no external datasets required. The scripts run the relevant model configuration in a matter of minutes and plot the resulting output. All results presented in this paper can be reproduced by one of two means: (1) downloading and installing "PySDM" and "PySDM-examples" (e.g., using "pip install") and running the notebooks locally or (2) accessing the PySDM-examples repository online and running the examples notebooks in the folder "deJong_Mackay_2022" on Google Colab. These codes, PySDM and PySDM-examples, are continuously under development at <https://github.com/atmos-cloud-sim-uj/PySDM> and <https://github.com/atmos-cloud-sim-uj/PySDM-examples> (last access: 21 April 2023) and are further documented in a software publication (de Jong et al., 2023).