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⇝precipitation

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

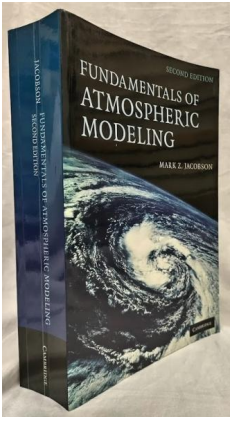
(photo: Yevgen Timashov / National Geographic)

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

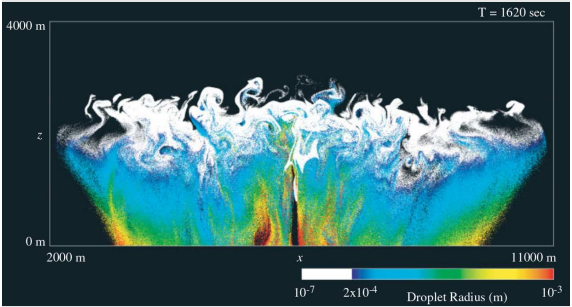
Eulerian vs. Lagrangian microphysics: a (probabilistic) breakthrough

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

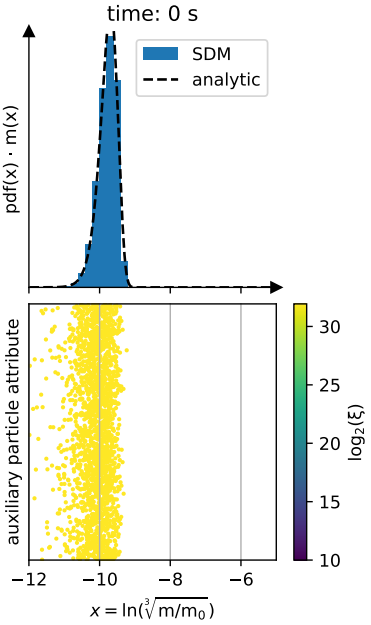


Super-droplet simulation of a shallow convective cloud

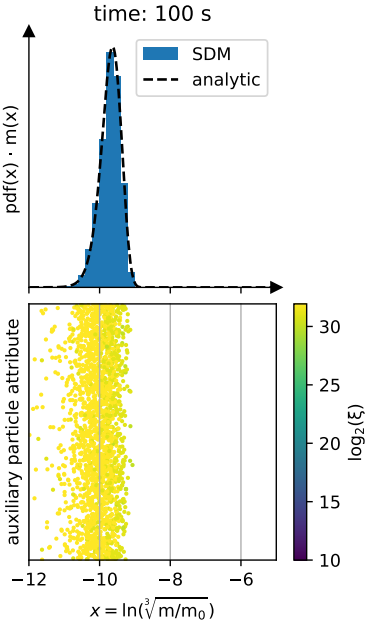
(figure: Shima et al. 2009, QJRMS)

[a] Jacobson 2005: Fundamentals of Atmospheric Modeling

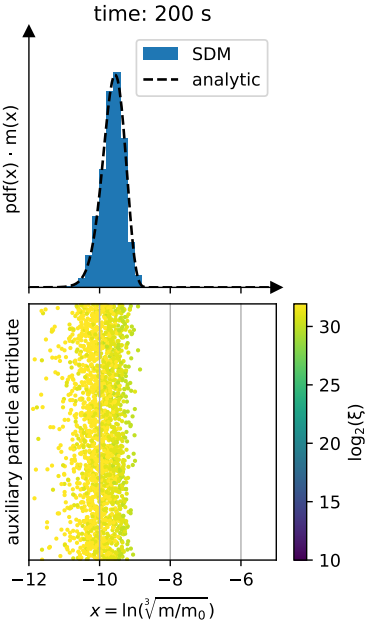
[b] Simmel & Wurzler 2006: Condensation and activation in sectional cloud microphysical models



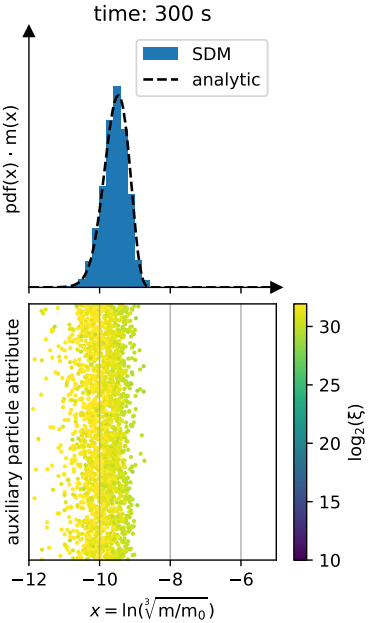

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



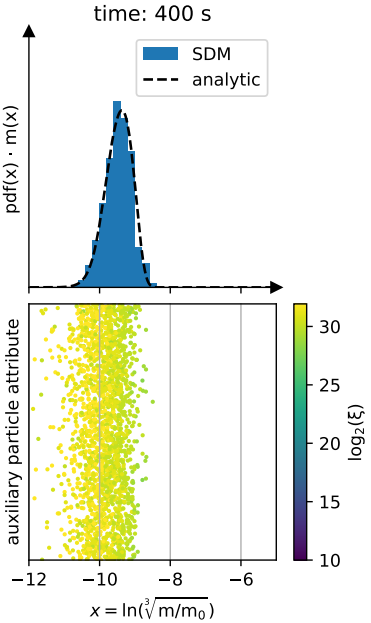

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



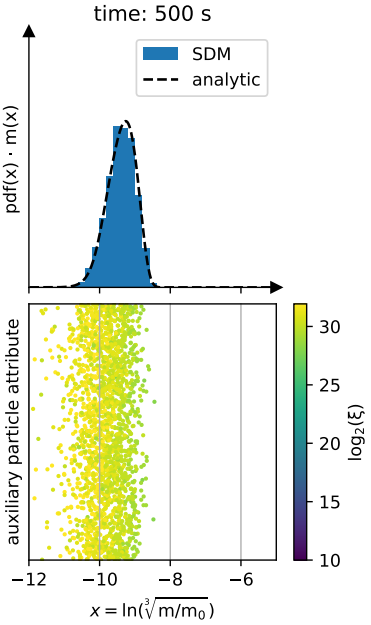

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



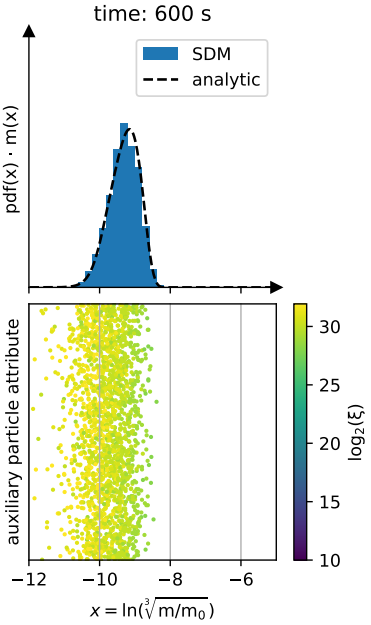

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



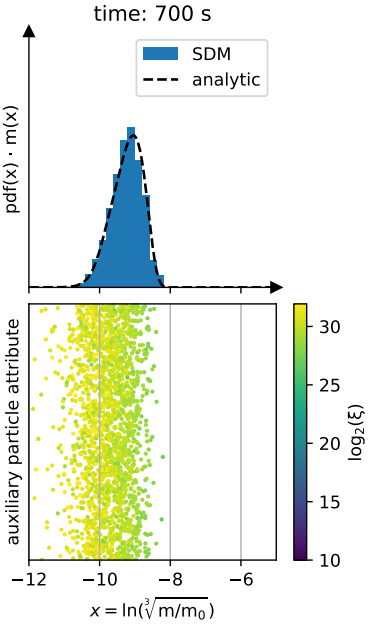

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



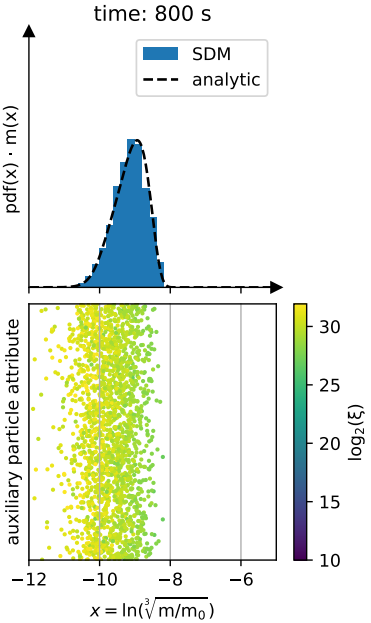

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



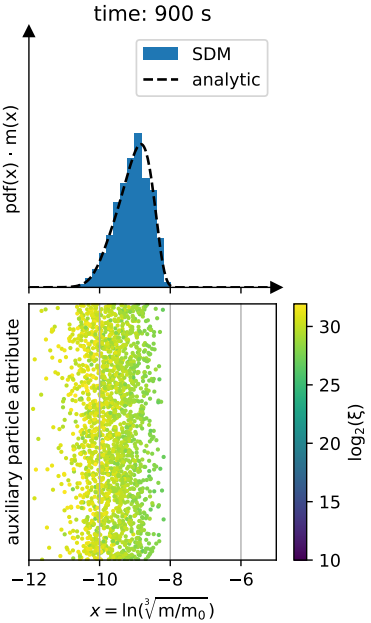

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



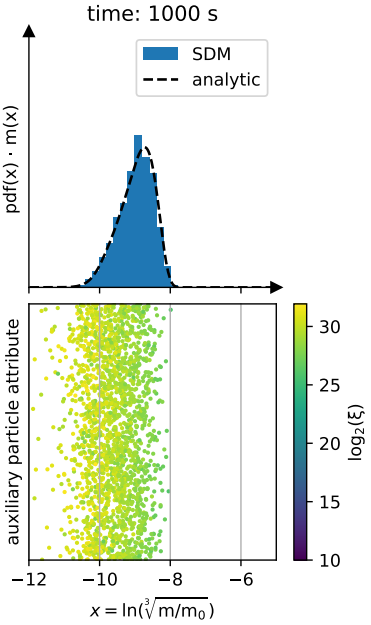

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



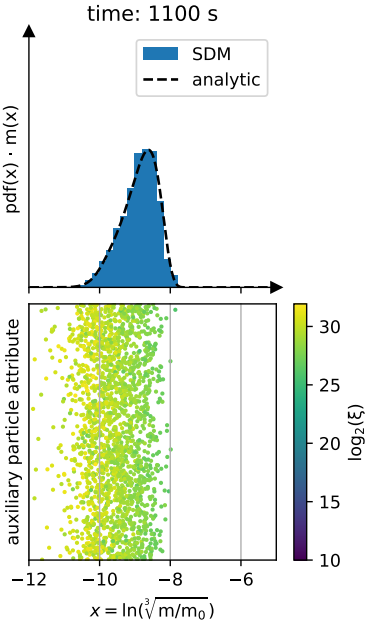

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



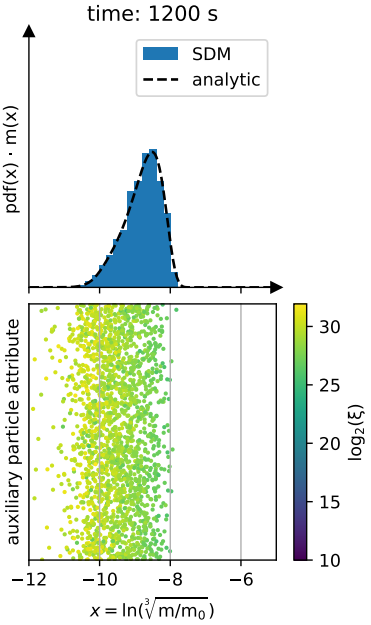

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



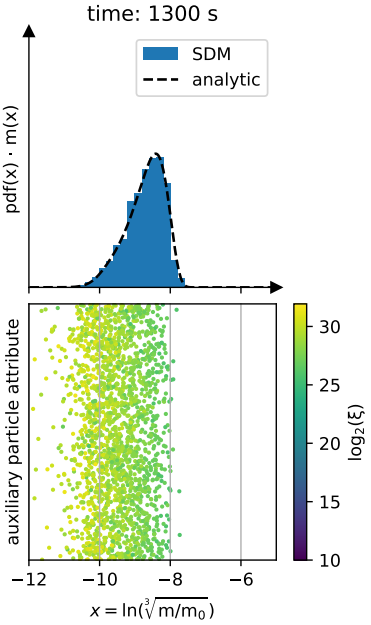

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



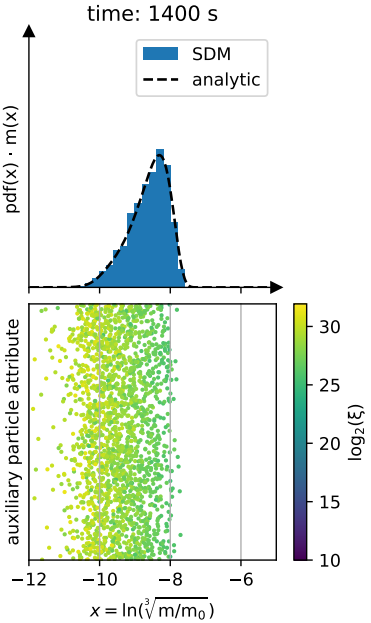

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



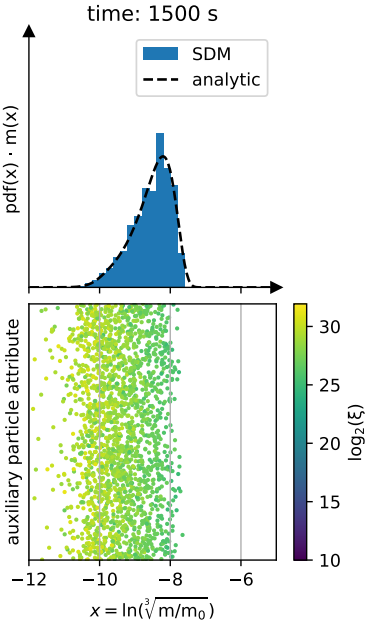

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



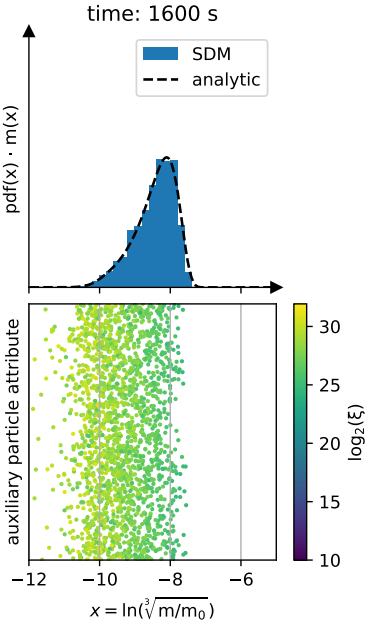

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



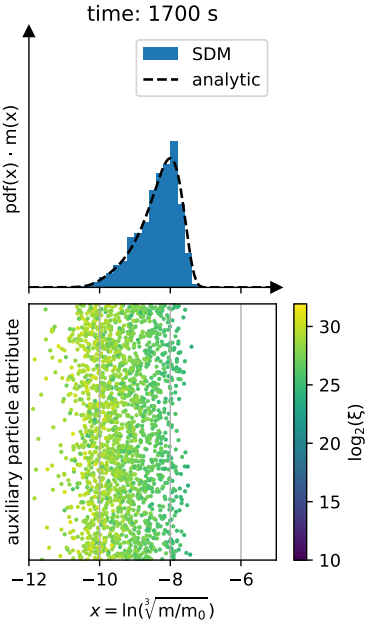

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



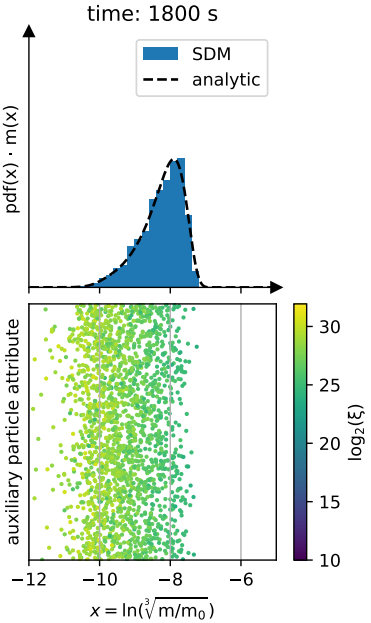

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



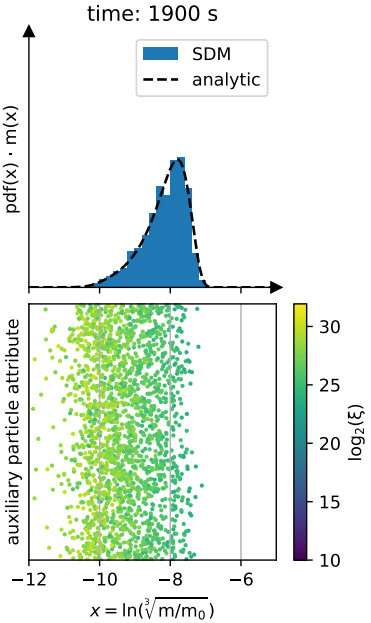

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



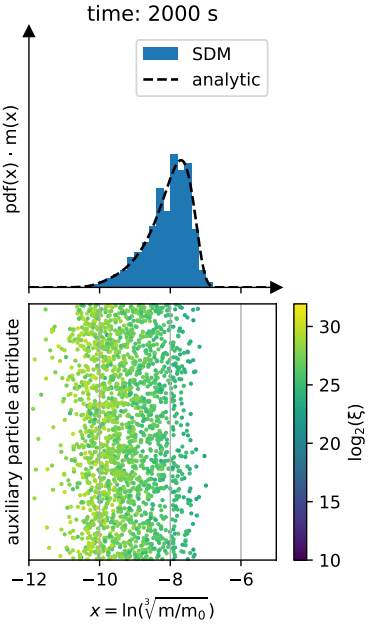

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



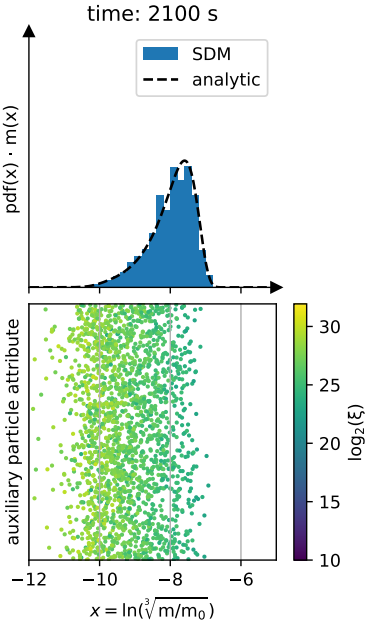

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



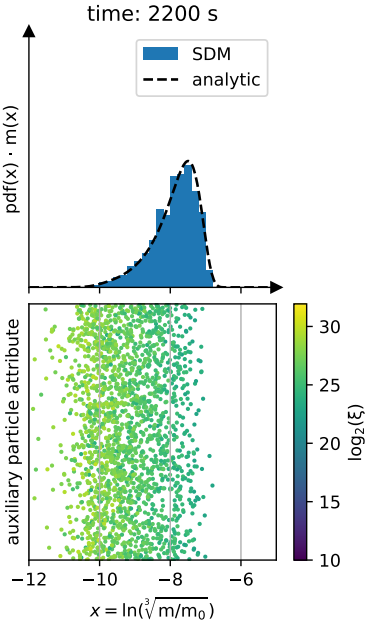

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



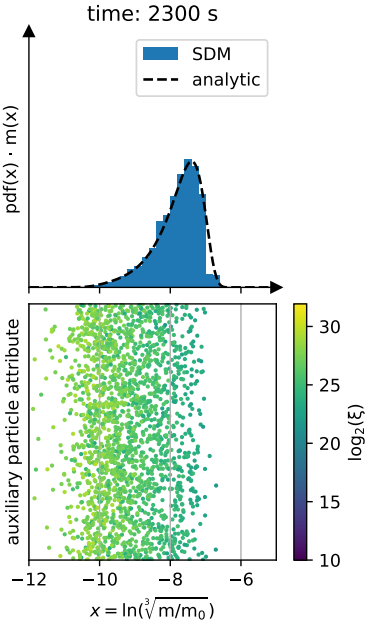

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



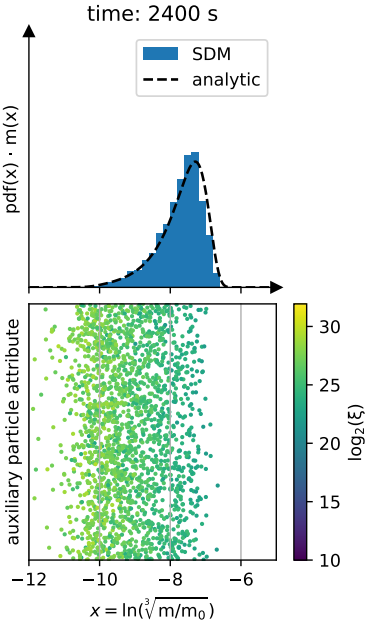

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



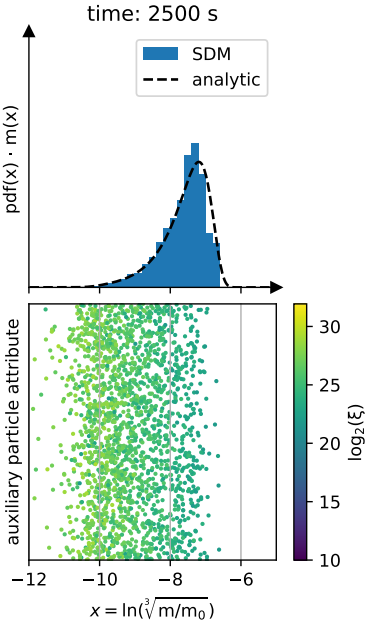

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



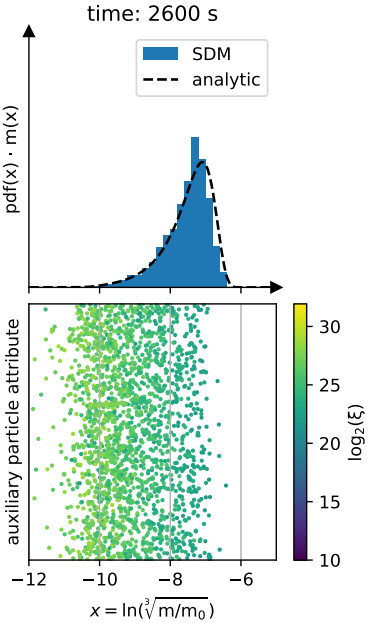

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



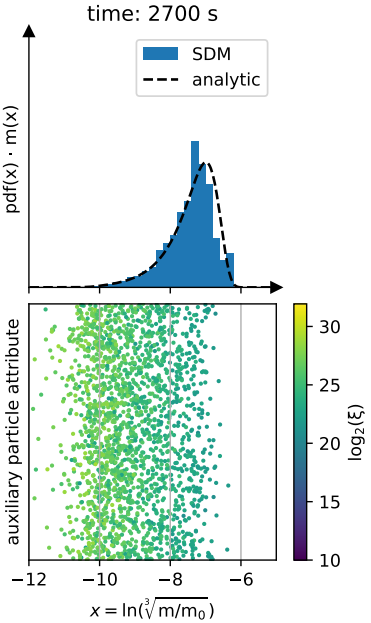

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



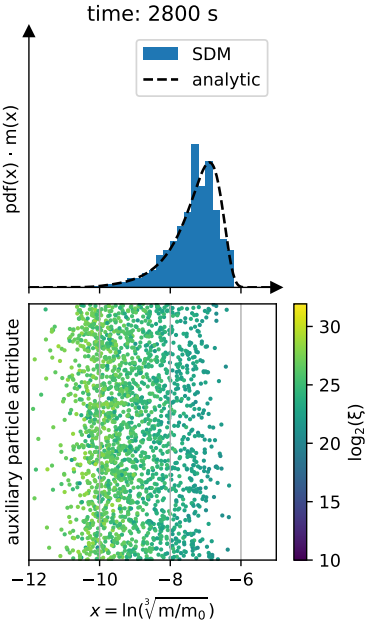

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



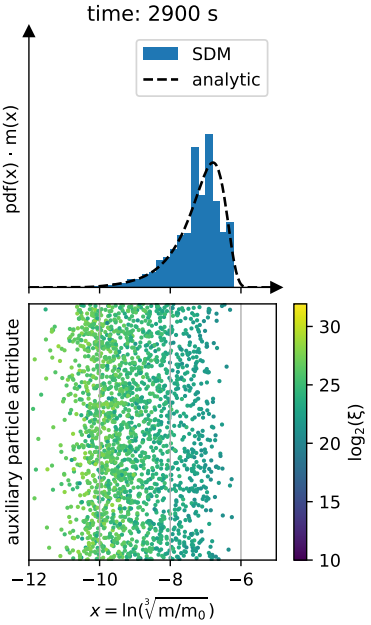

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



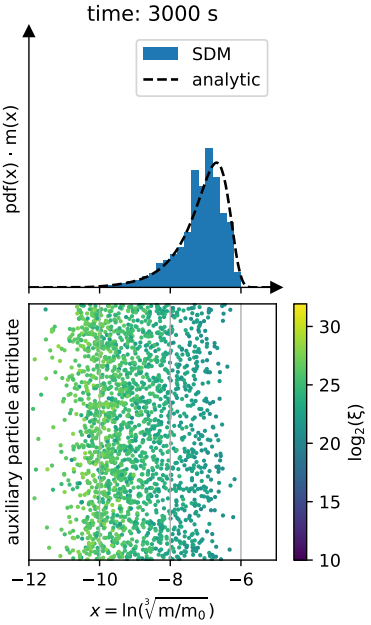

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



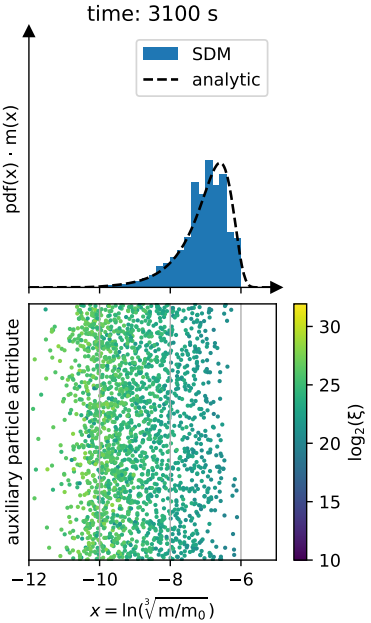

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



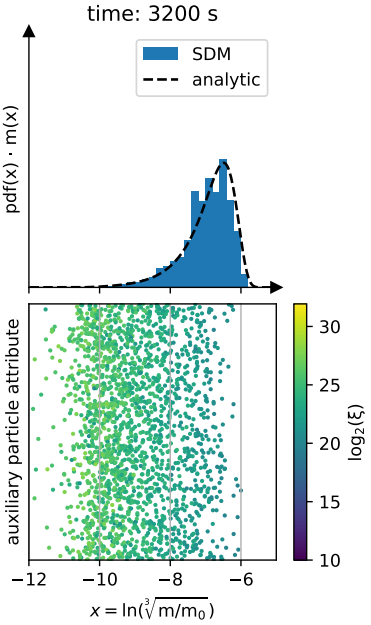

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



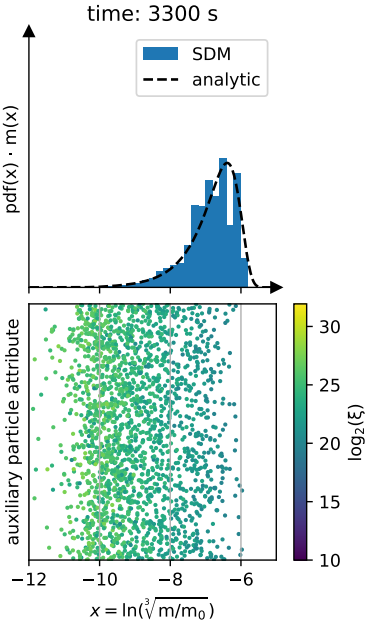

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



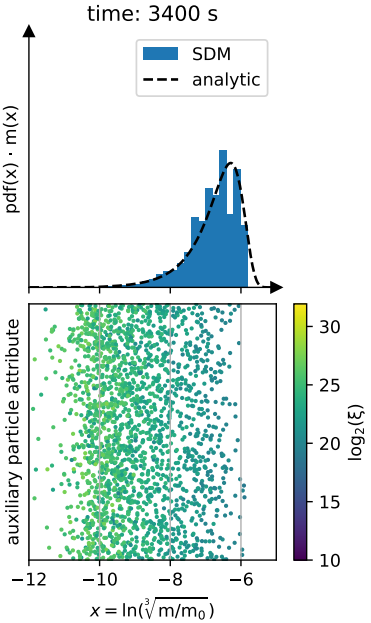

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



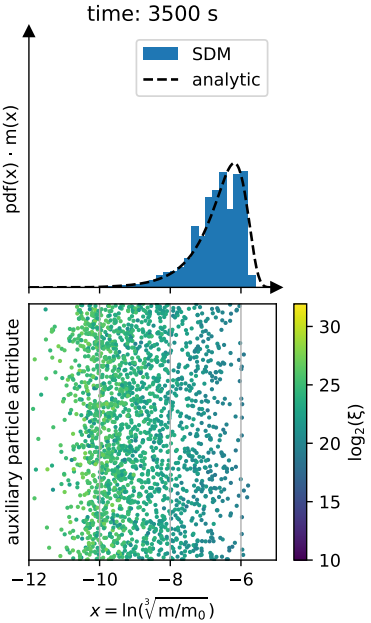

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



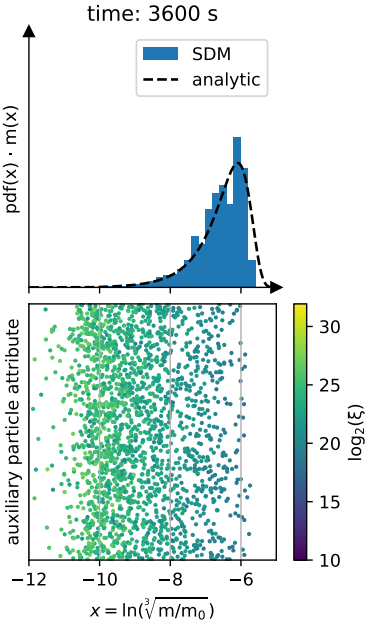

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



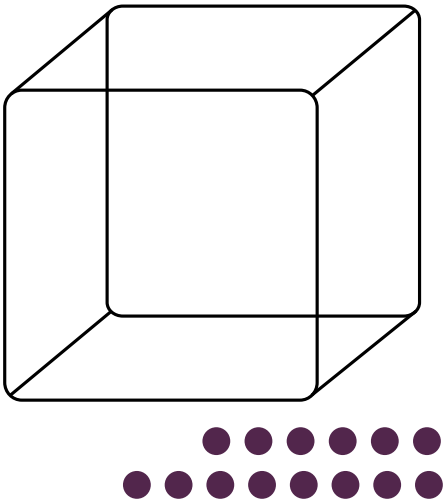

```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```




```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]
```

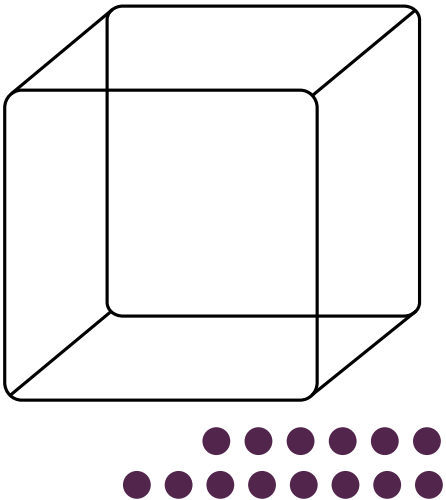



```python
def sdm(*,
    rng: np.random.Generator,
    ξ: abc.MutableSequence[int],
    m: abc.MutableSequence[float],
    kern: abc.Callable[[float, float], float],
    Δt: float,
    Δv: float,
):
    """ SDM step assuming non-zero multiplicities """
    n_s = len(ξ)
    n_pair = n_s // 2
    pairs = 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]
```



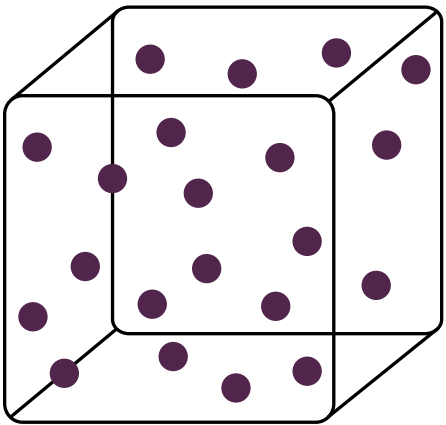
Domain randomly populated with "μ-physics information carriers"

(super particles / super droplets)



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

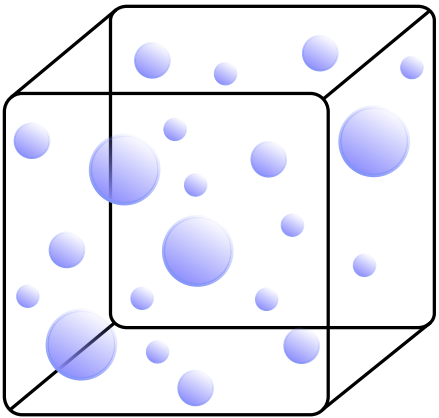
carrier attributes:



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

carrier attributes:

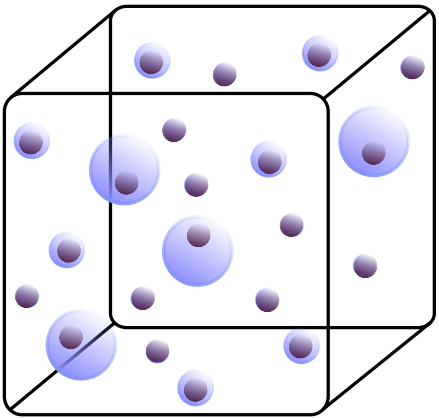
- location



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

carrier attributes:

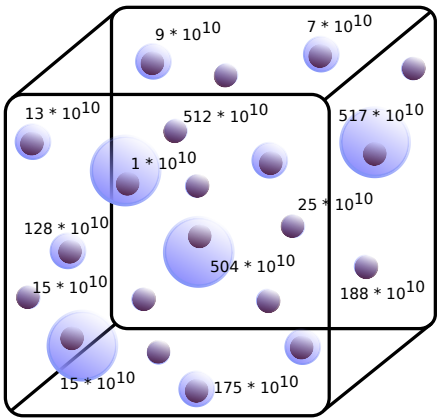
- location
- wet radius



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

carrier attributes:

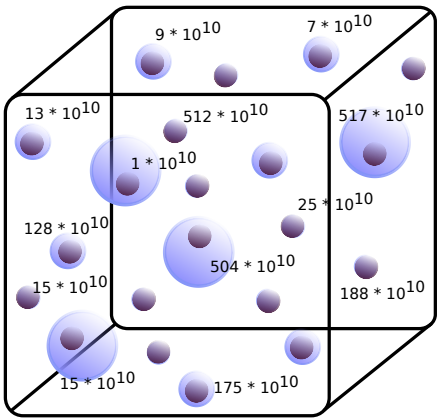
- location
- wet radius
- CCN/INP characteristics



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

carrier attributes:

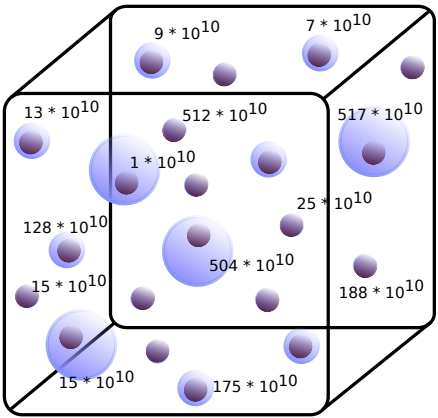
- location
- wet radius
- CCN/INP characteristics
- multiplicity



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?

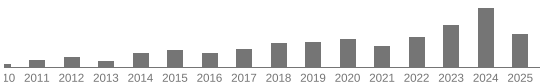
Shima et al. 2009 algorithm (BTW, Shima's PhD advisor was Yoshiki Kuramoto)

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

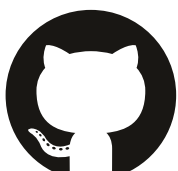
[PDF] from arxiv.org
Get this item at AGH

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

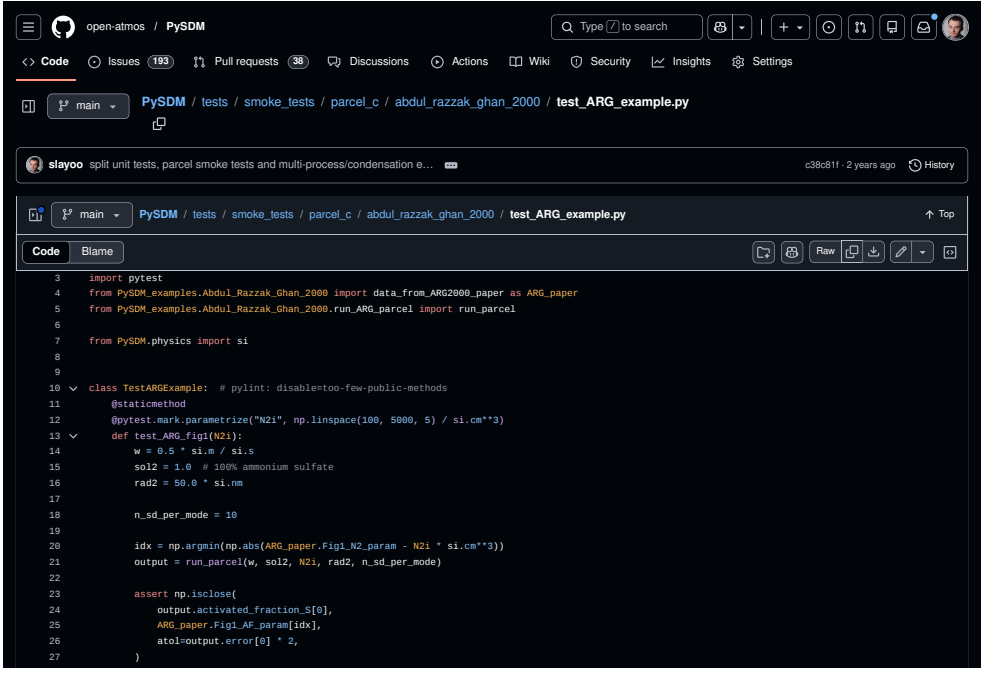
100%  python™ open-source code:

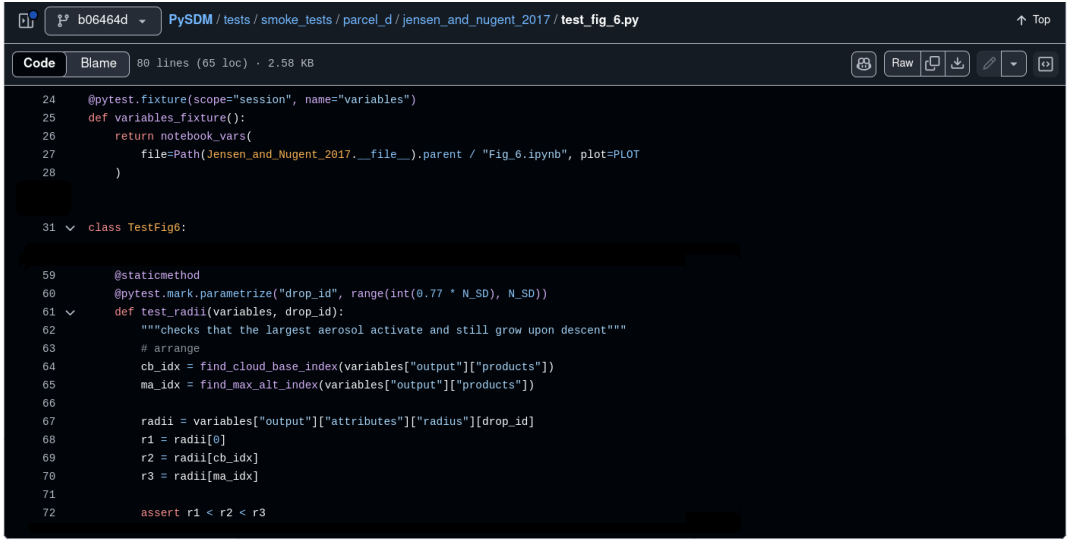
 / OPENATMOS / PySDM

PySDM: example gallery

Literature Reference	Cond/Evap	Collisions	Isotopes	Breakup	Transport	Chemistry	Freezing	Comments/keywords
no-environment								
Merlivat & Nief 1967			×					
Gedzelman & Arnold 1994	×		×					
Pierchala et al. 2022			×					theoretical curves for a lab experiment
OD box environment								
Shima et al. 2009		×						Golovin kernel example
Berry 1967		×						Several different kernels
Bieli et al. 2022		×		×				
Alpert & Knopf 2016							×	
OD parcel environment								
Kreidenweis et al. 2003	×					×		"Hoppel" gap
Jaruga & Pawlowska 2018	×					×		"Hoppel" gap
Lowe et al. 2019	×							surfactants
Yang et al. 2018	×							ripening (depending on the definition)
Graf et al. 2019	×		×					
Arabas and Shima 2017	×							monodisperse, activation/deactivation cycle
Abdul-Razzak & Ghan 2000	×							parcel vs. activation parameterisation
1D single-column kinematic env.								
Shipway & Hill 2012	×	×			×			KiD 1D
deJong et al. 2023 (figures 6-8)	×	×		×	×			
2D prescribed-flow environment								
Arabas et al. 2015	×	×			×			includes GUI
Arabas et al. 2025	×	×			×		×	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
Raw
@pytest.fixture(scope="session", name="variables")
def variables_fixture():
return notebook_vars(
file=Path(Jensen_and_Nugent_2017.__file__).parent / "Fig_6.ipynb", plot=PLOT
)
class TestFig6:
@staticmethod
@pytest.mark.parametrize("drop_id", range(int(0.77 * N_SD), N_SD))
def test_radii(variables, drop_id):
"""checks that the largest aerosol activate and still grow upon descent"""
# arrange
cb_idx = find_cloud_base_index(variables["output"]["products"])
ma_idx = find_max_alt_index(variables["output"]["products"])
radii = variables["output"]["attributes"]["radius"][drop_id]
r1 = radii[0]
r2 = radii[cb_idx]
r3 = radii[ma_idx]
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	91.11%
backends	3619	3235	0	384	89.39%
dynamics	1229	1074	0	155	87.39%
environments	273	223	0	50	81.68%
exporters	286	75	0	211	26.22%
impl	204	199	0	5	97.55%
initialisation	375	355	0	20	94.67%
physics	1985	1701	0	284	85.69%
products	1279	1062	0	217	83.03%
__init__.py	11	9	0	2	81.82%
builder.py	89	80	0	9	89.89%
formulae.py	206	203	0	3	98.54%
particulator.py	159	147	0	12	92.45%
Subtotal	10266	8865	0	1401	

PySDM: IoC for physical formulae

jupyter table_1 Last Checkpoint: 48 minutes ago

File Edit View Run Kernel Settings Help

Code

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

```
import os, sys
os.environ['NUMBA_THREADING_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')
```

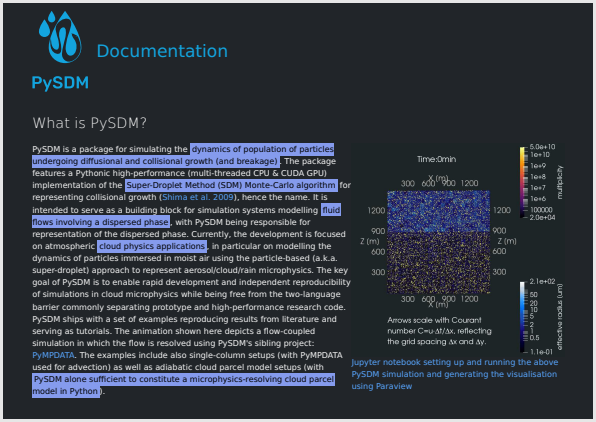
```
import numpy as np
from PySDM import Formulae
```

```
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 = "ZografosEtAl1987",
    isotope_equilibrium_fractionation_factors = "VanHook1968",
    isotope_diffusivity_ratios = "GrahamsLaw",
    constants = {"BOLIN_ISOTOPE_TIMESCALE_COEFF_C1": 1.63},
    isotope_relaxation_timescale = "Bolin1958",
})
```

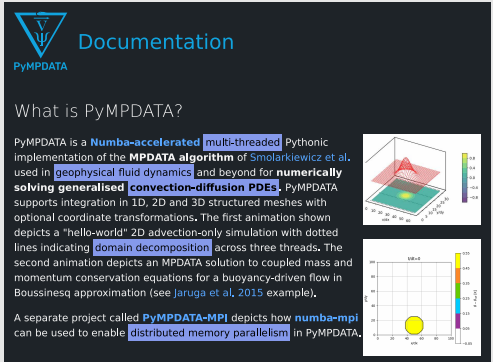
PySDM companion packages: PyMPDATA & Numba-MPI

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

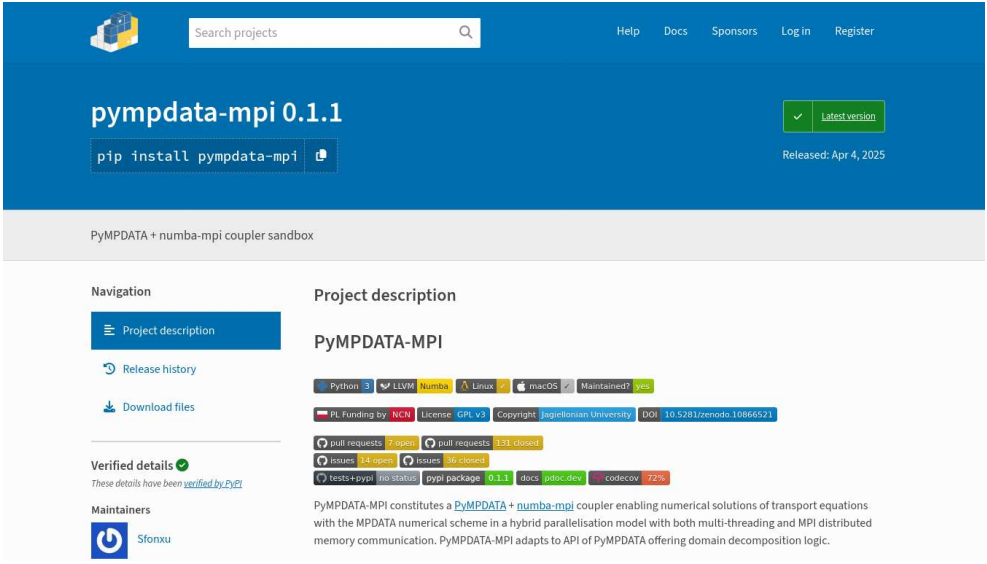
https://open-atmos.github.io/PySDM



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

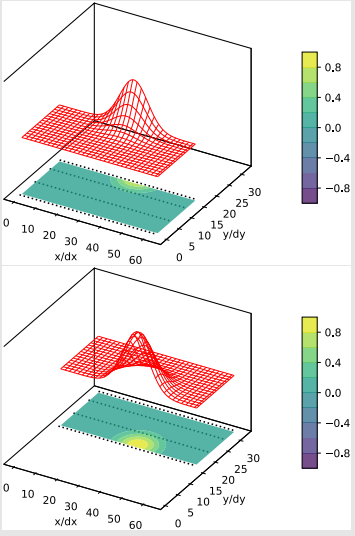


PyMPDATA-MPI

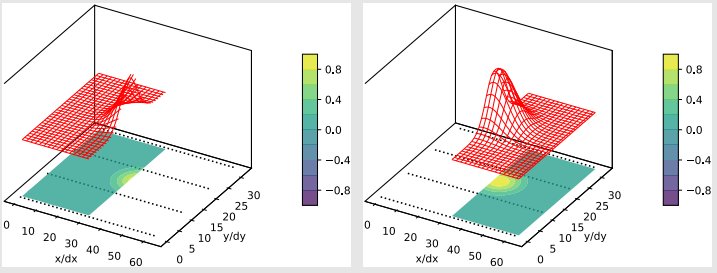


PyMPDATA-MPI: customisable hybrid threading + MPI parallelisation

threading dim = MPI dim



threading dimension ≠ MPI dimension



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

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.

kudos team & contributors!

students in Kraków:

Graduate Students (@agh.edu.pl)

Agnieszka Zaba
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

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

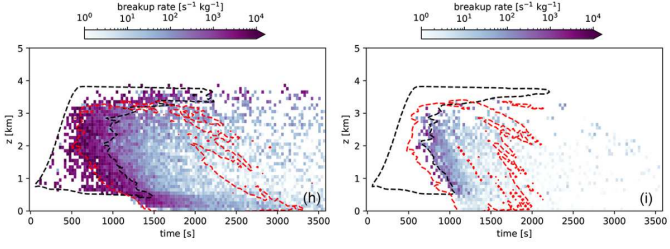
Geoscientific Model Development
Open Access
EGU

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



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).