

# PyMPDATA: an open-source, example-rich, just-in-time compiled implementation of MPDATA finite-difference scheme

---

**Sylwester Arabas**

AGH University of Krakow

June 12 2025 (KN Kernel seminar)



- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and usage examples
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)

# MPDATA key concepts: UPWIND discretisation

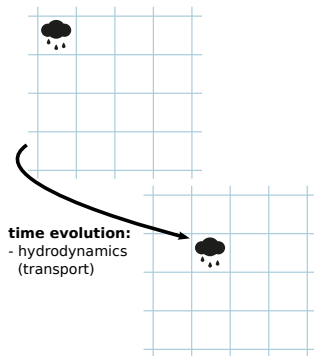
- advection equation / scalar conservation law:

$$\partial_t(G\psi) + \nabla \cdot (\mathbf{v}\psi) = GR$$

$\psi(\mathbf{x}, t)$ : advected scalar field (advectee),

$\mathbf{v} = \{u, \dots\} = G\dot{\mathbf{x}}$ : flow velocity vector field (advector),

$G(\mathbf{x})$ : fluid density, Jacobian of coordinate transformation, or their product



# MPDATA key concepts: UPWIND discretisation

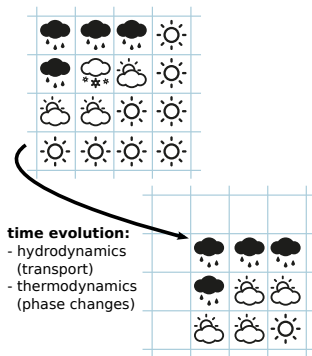
- advection equation / scalar conservation law:

$$\partial_t(G\psi) + \nabla \cdot (\mathbf{v}\psi) = GR$$

$\psi(\mathbf{x}, t)$ : advected scalar field (advectee),

$\mathbf{v} = \{u, \dots\} = G\dot{\mathbf{x}}$ : flow velocity vector field (advector),

$G(\mathbf{x})$ : fluid density, Jacobian of coordinate transformation, or their product



# MPDATA key concepts: UPWIND discretisation

- advection equation / scalar conservation law:

$$\partial_t(G\psi) + \nabla \cdot (\mathbf{v}\psi) = GR$$

$\psi(\mathbf{x}, t)$ : advected scalar field (advectee),

$\mathbf{v} = \{u, \dots\} = G\dot{\mathbf{x}}$ : flow velocity vector field (advector),

$G(\mathbf{x})$ : fluid density, Jacobian of coordinate transformation, or their product

- homogeneous problem in 1D and with  $G = 1$ :

$$\partial_t \psi + \partial_x (u\psi) = 0$$

# MPDATA key concepts: UPWIND discretisation

- advection equation / scalar conservation law:

$$\partial_t(G\psi) + \nabla \cdot (\mathbf{v}\psi) = GR$$

$\psi(\mathbf{x}, t)$ : advected scalar field (advectee),

$\mathbf{v} = \{u, \dots\} = G\dot{\mathbf{x}}$ : flow velocity vector field (advector),

$G(\mathbf{x})$ : fluid density, Jacobian of coordinate transformation, or their product

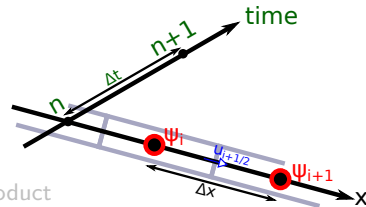
- homogeneous problem in 1D and with  $G = 1$ :

$$\partial_t \psi + \partial_x (u\psi) = 0$$

- UPWIND discretisation on a spatially staggered grid ( $n$  numbers time steps,  $i$  numbers grid steps):

$$\frac{\psi_i^{n+1} - \psi_i^n}{\Delta t} + \frac{\overbrace{f(\psi_i^n, \psi_{i+1}^n, u_{i+1/2}^n)}^{\text{right-hand wall flux}} - \overbrace{f(\psi_{i-1}^n, \psi_i^n, u_{i-1/2}^n)}^{\text{left-hand wall flux}}}{\Delta x} = 0$$

$$f(\psi_l, \psi_r, u) = \overbrace{\frac{u + |u|}{2}}^{\text{positive part}} \psi_l + \overbrace{\frac{u - |u|}{2}}^{\text{negative part}} \psi_r$$



# MPDATA key concepts: Courant number & UPWIND stability criterion

- introducing non-dimensional Courant number  $C = u \frac{\Delta t}{\Delta x}$ :

$$\psi_i^{n+1} = \psi_i^n - [f(\psi_i^n, \psi_{i+1}^n, C_{i+1/2}^n) - f(\psi_{i-1}^n, \psi_i^n, C_{i-1/2}^n)]$$

yields a conservative and sing-preserving “UPWIND” scheme which is stable for  $|C| \leq 1$ .

---

```
1 def f(psi_l, psi_r, C):
2     return .5 * (C + abs(C)) * psi_l + \
3         .5 * (C - abs(C)) * psi_r
4 def step(psi: np.ndarray, i: slice, C: np.ndarray):
5     psi[i] = psi[i] - (
6         f(psi[i], psi[i + one], C[i + hlf]) -
7         f(psi[i - one], psi[i], C[i - hlf])
8     )
9 def upwind(nt: int, C: np.ndarray, psi: np.ndarray):
10     i = slice(1, len(psi) - 1)
11     for _ in range(nt):
12         step(psi, i, C)
```

---

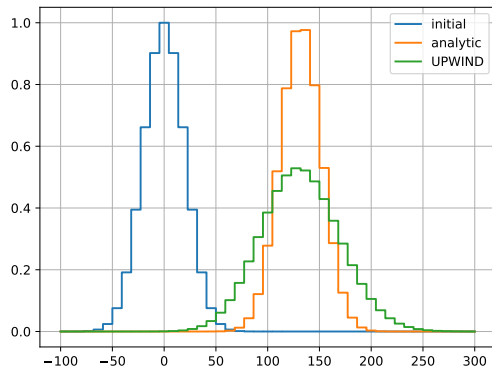
# MPDATA key concepts: Courant number & UPWIND stability criterion

- introducing non-dimensional Courant number  $C = u \frac{\Delta t}{\Delta x}$ :

$$\psi_i^{n+1} = \psi_i^n - [f(\psi_i^n, \psi_{i+1}^n, C_{i+1/2}^n) - f(\psi_{i-1}^n, \psi_i^n, C_{i-1/2}^n)]$$

yields a conservative and sing-preserving “UPWIND” scheme which is stable for  $|C| \leq 1$ .

```
1 def f(psi_l, psi_r, C):
2     return .5 * (C + abs(C)) * psi_l + \
3           .5 * (C - abs(C)) * psi_r
4 def step(psi: np.ndarray, i: slice, C: np.ndarray):
5     psi[i] = psi[i] - (
6         f(psi[i], psi[i + one], C[i + hlf]) -
7         f(psi[i - one], psi[i], C[i - hlf])
8     )
9 def upwind(nt: int, C: np.ndarray, psi: np.ndarray):
10     i = slice(1, len(psi) - 1)
11     for _ in range(nt):
12         step(psi, i, C)
```





# MPDATA key concepts: numerical diffusion & modified-equation analysis

- UPWIND incurs **numerical diffusion**, quantifiable using Taylor expansion (const.  $C$  for simplicity):

$$\begin{aligned}\psi_i^{n+1} &= \psi_i^n + \partial_t \psi|_i^n (+\Delta t) + \frac{1}{2} \partial_t^2 \psi|_i^n (+\Delta t)^2 + O(\Delta t^3) \\ \psi_{i+1}^n &= \psi_i^n + \partial_x \psi|_i^n (+\Delta x) + \frac{1}{2} \partial_x^2 \psi|_i^n (+\Delta x)^2 + O(\Delta x^3) \\ \psi_{i-1}^n &= \psi_i^n + \partial_x \psi|_i^n (-\Delta x) + \frac{1}{2} \partial_x^2 \psi|_i^n (-\Delta x)^2 + O(\Delta x^3)\end{aligned} \quad \rightsquigarrow \quad \begin{aligned}\psi_i^{n+1} &= \psi_i^n - \left[ \frac{C + |C|}{2} (\psi_i^n - \psi_{i-1}^n) \right. \\ &\quad \left. + \frac{C - |C|}{2} (\psi_{i+1}^n - \psi_i^n) \right]\end{aligned}$$

# MPDATA key concepts: numerical diffusion & modified-equation analysis

- UPWIND incurs **numerical diffusion**, quantifiable using Taylor expansion (const.  $C$  for simplicity):

$$\begin{aligned}
 \psi_i^{n+1} &= \psi_i^n + \partial_t \psi|_i^n (+\Delta t) + \frac{1}{2} \partial_t^2 \psi|_i^n (+\Delta t)^2 + O(\Delta t^3) \\
 \psi_{i+1}^n &= \psi_i^n + \partial_x \psi|_i^n (+\Delta x) + \frac{1}{2} \partial_x^2 \psi|_i^n (+\Delta x)^2 + O(\Delta x^3) \\
 \psi_{i-1}^n &= \psi_i^n + \partial_x \psi|_i^n (-\Delta x) + \frac{1}{2} \partial_x^2 \psi|_i^n (-\Delta x)^2 + O(\Delta x^3)
 \end{aligned}
 \quad \rightsquigarrow \quad
 \begin{aligned}
 \psi_i^{n+1} &= \psi_i^n - \left[ \frac{C + |C|}{2} (\psi_i^n - \psi_{i-1}^n) \right. \\
 &\quad \left. + \frac{C - |C|}{2} (\psi_{i+1}^n - \psi_i^n) \right]
 \end{aligned}$$

- which substituted to the UPWIND formulæ yields (up to second-order terms):

$$\partial_t \psi|_i^n \Delta t + \underbrace{\partial_t^2 \psi|_i^n}_{u^2 \partial_x^2 \psi} \frac{\Delta t^2}{2} = -C \Delta x \partial_x \psi|_i^n + \frac{|C|}{2} \Delta x^2 \partial_x^2 \psi|_i^n$$

# MPDATA key concepts: numerical diffusion & modified-equation analysis

- UPWIND incurs **numerical diffusion**, quantifiable using Taylor expansion (const.  $C$  for simplicity):

$$\begin{aligned}
 \psi_i^{n+1} &= \psi_i^n + \partial_t \psi|_i^n (+\Delta t) + \frac{1}{2} \partial_t^2 \psi|_i^n (+\Delta t)^2 + O(\Delta t^3) \\
 \psi_{i+1}^n &= \psi_i^n + \partial_x \psi|_i^n (+\Delta x) + \frac{1}{2} \partial_x^2 \psi|_i^n (+\Delta x)^2 + O(\Delta x^3) \\
 \psi_{i-1}^n &= \psi_i^n + \partial_x \psi|_i^n (-\Delta x) + \frac{1}{2} \partial_x^2 \psi|_i^n (-\Delta x)^2 + O(\Delta x^3)
 \end{aligned}
 \quad \rightsquigarrow \quad
 \begin{aligned}
 \psi_i^{n+1} &= \psi_i^n - \left[ \frac{C + |C|}{2} (\psi_i^n - \psi_{i-1}^n) \right. \\
 &\quad \left. + \frac{C - |C|}{2} (\psi_{i+1}^n - \psi_i^n) \right]
 \end{aligned}$$

- which substituted to the UPWIND formulæ yields (up to second-order terms):

$$\partial_t \psi|_i^n \Delta t + \underbrace{\partial_t^2 \psi|_i^n}_{u^2 \partial_x^2 \psi} \frac{\Delta t^2}{2} = -C \Delta x \partial_x \psi|_i^n + \frac{|C|}{2} \Delta x^2 \partial_x^2 \psi|_i^n$$

where  $\partial_t^2 \psi$  can be replaced with spatial derivative using a time derivative of the advection eq.:

$$\partial_t \psi|_i^n + u \partial_x \psi|_i^n = \underbrace{\left( |u| \frac{\Delta x}{2} - u^2 \frac{\Delta t}{2} \right)}_{k - \text{numerical diffusion}} \partial_x^2 \psi|_i^n$$

(e.g., Roberts & Weiss 1966, doi:10.2307/2003507)

# MPDATA key concepts: antidiffusive pseudo-velocities

- diffusion can be cast as advection with a pseudo-velocity:

$$\partial_t \psi + k \partial_x^2 \psi = \dots \quad \rightsquigarrow \quad \partial_t \psi + \underbrace{\partial_x \left( k \frac{\partial_x \psi}{\psi} \right)}_{\text{pseudo-velocity}} \psi = \dots$$

(e.g., Lange 1973, doi:10.2172/4308175)

# MPDATA key concepts: antidiffusive pseudo-velocities

- diffusion can be cast as advection with a pseudo-velocity:

$$\partial_t \psi + k \partial_x^2 \psi = \dots \quad \rightsquigarrow \quad \partial_t \psi + \underbrace{\partial_x \left( k \frac{\partial_x \psi}{\psi} \psi \right)}_{\text{pseudo-velocity}} = \dots$$

(e.g., Lange 1973, doi:10.2172/4308175)

- “Smolarkiewicz algorithm” (MPDATA): upwind-integrate backwards-in-time, with an anti-diffusive pseudo velocity to reverse the effects of numerical diffusion, iteratively ( $m$  numbers iteration)

$$C_{i-1/2}^{m+1} = \frac{\Delta t}{\Delta x} k_{i-1/2}^m \left. \frac{\partial_x \psi}{\psi} \right|_{i-1/2}^m \approx \begin{cases} 0 & \text{if } \psi_i^m + \psi_{i-1}^m = 0 \\ \left[ |C_{i-1/2}^m| - (C_{i-1/2}^m)^2 \right] \frac{\psi_i^m - \psi_{i-1}^m}{\psi_i^m + \psi_{i-1}^m} & \text{otherwise} \end{cases}$$

(Smolarkiewicz 1983 MWR, 1984 JCP: doi:10.1016/0021-9991(84)90121-9)

# MPDATA hello-world (1D, single iteration) implementation

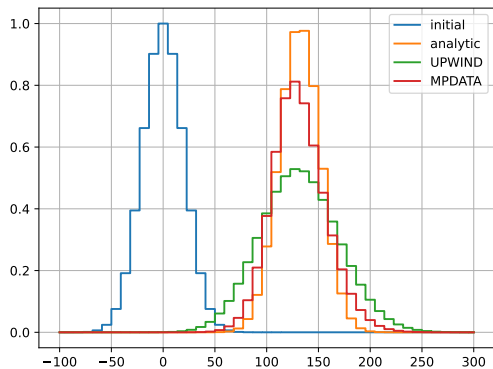
---

```
1 def C_corr(C: np.ndarray, i: slice, psi: np.ndarray):
2     return (abs(C[i - hlf]) - C[i - hlf] ** 2) * (
3         psi[i] - psi[i - one]
4     ) / (
5         psi[i - one] + psi[i]
6     )
7 def mpdata(nt: int, C: np.ndarray, psi: np.ndarray):
8     i = slice(1, len(psi) - 1)
9     i_ext = slice(1, len(psi))
10    for _ in range(nt):
11        upwind(psi, i, C)
12        upwind(psi, i, C_corr(C, i_ext, psi))
```

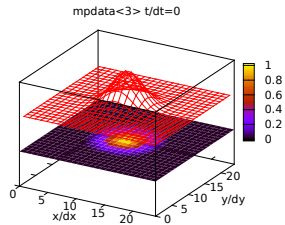
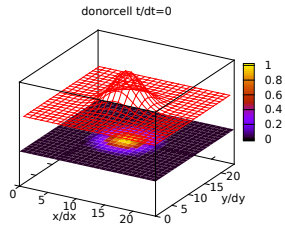
---

# MPDATA hello-world (1D, single iteration) implementation

```
1 def C_corr(C: np.ndarray, i: slice, psi: np.ndarray):
2     return (abs(C[i - hlf]) - C[i - hlf] ** 2) * (
3         psi[i] - psi[i - one]
4     ) / (
5         psi[i - one] + psi[i]
6     )
7 def mpdata(nt: int, C: np.ndarray, psi: np.ndarray):
8     i = slice(1, len(psi) - 1)
9     i_ext = slice(1, len(psi))
10    for _ in range(nt):
11        upwind(psi, i, C)
12        upwind(psi, i, C_corr(C, i_ext, psi))
```

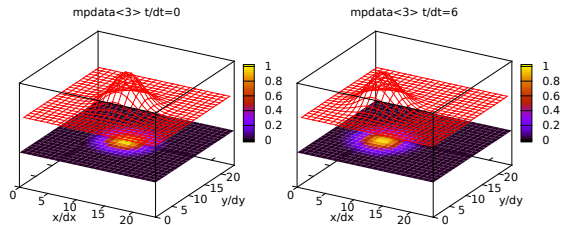
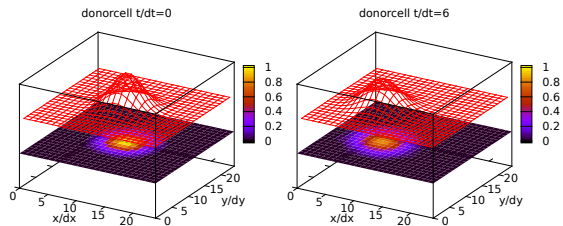


# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)

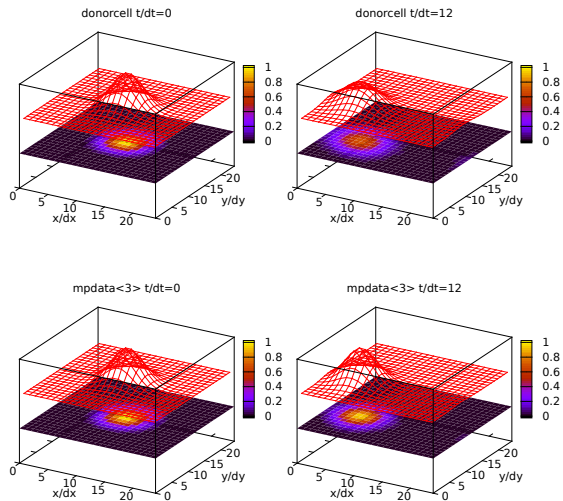




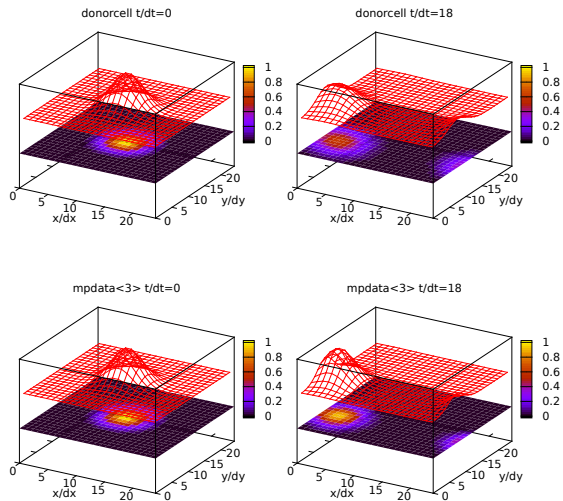
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



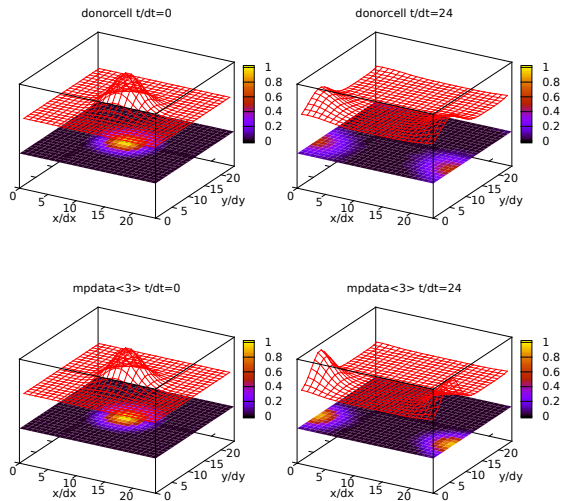
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



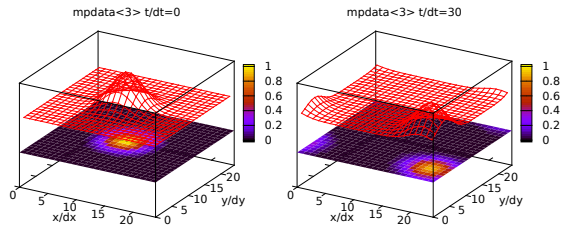
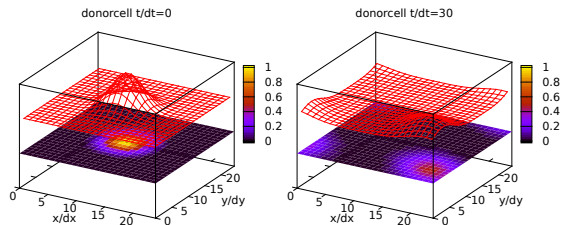
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



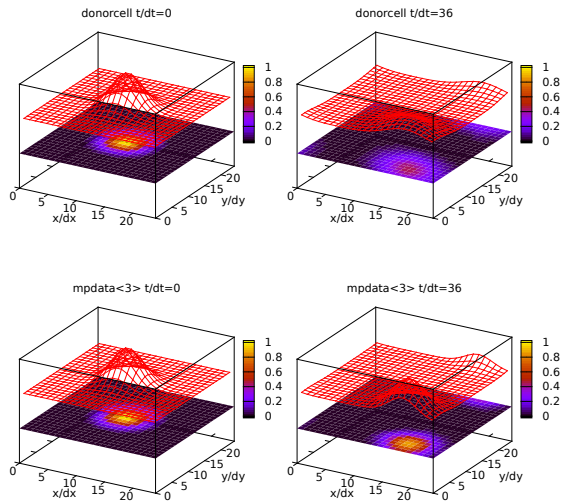
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



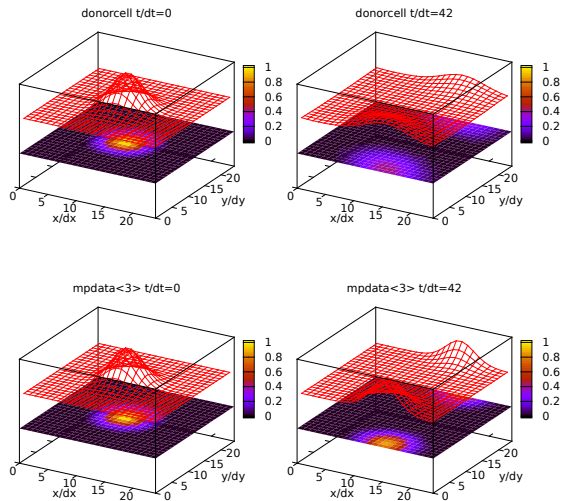
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



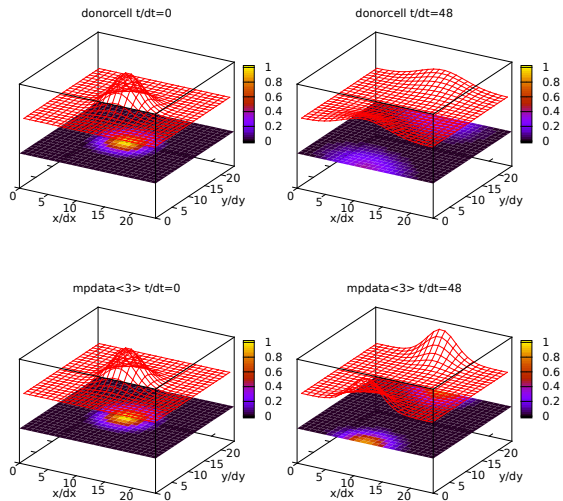
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)

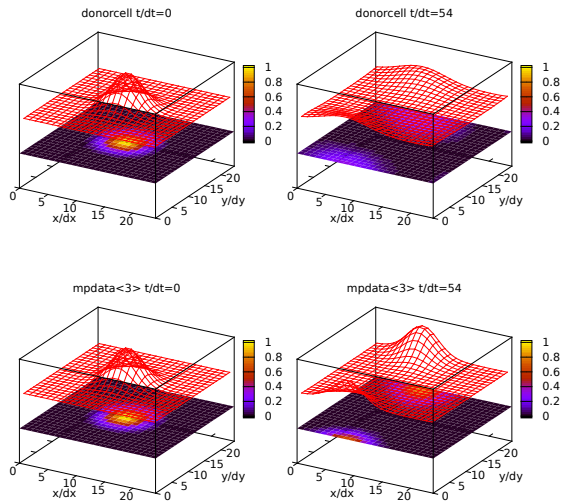


# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)

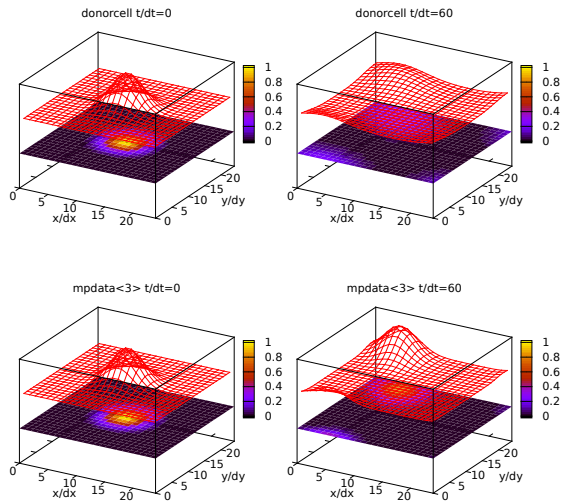




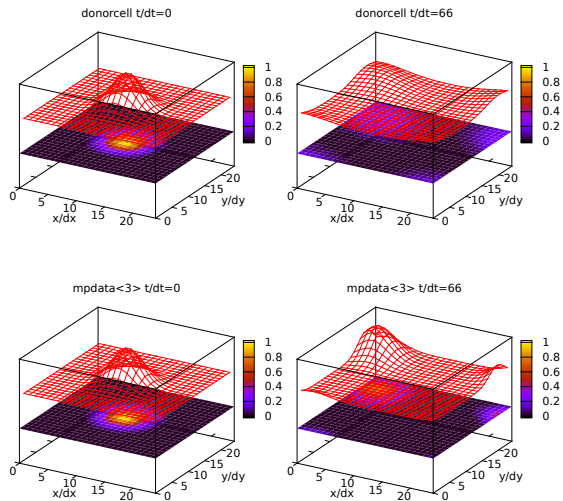
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



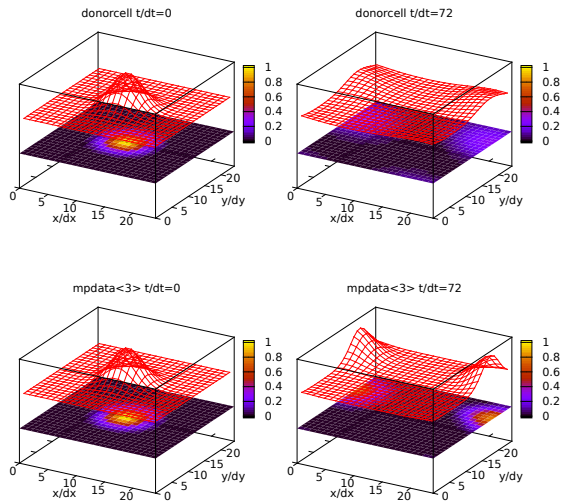
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



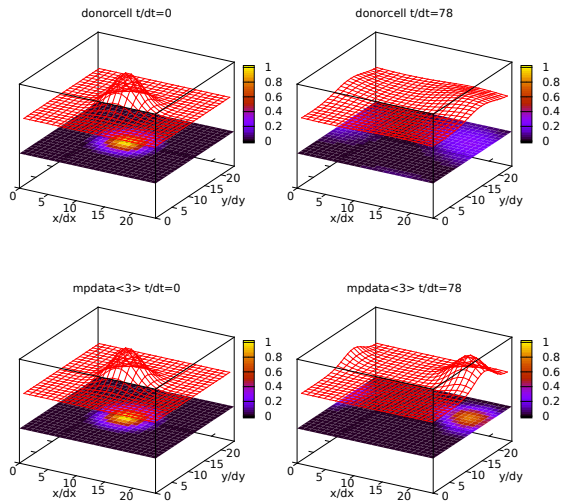
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



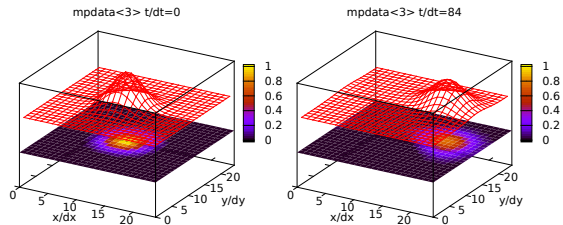
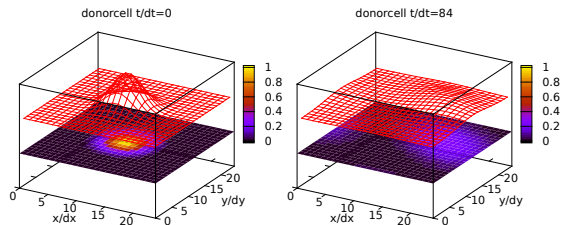
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



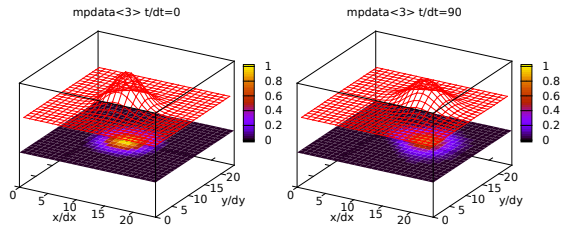
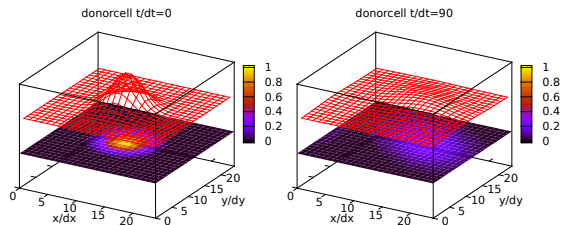
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



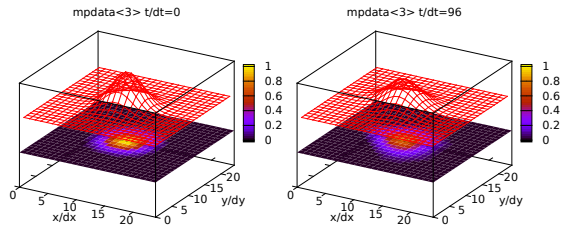
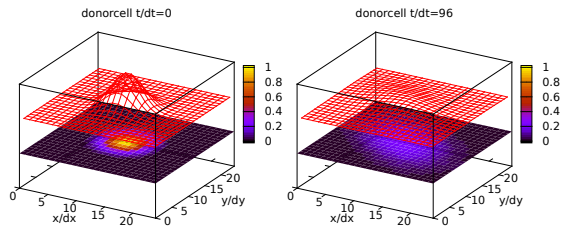
# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)

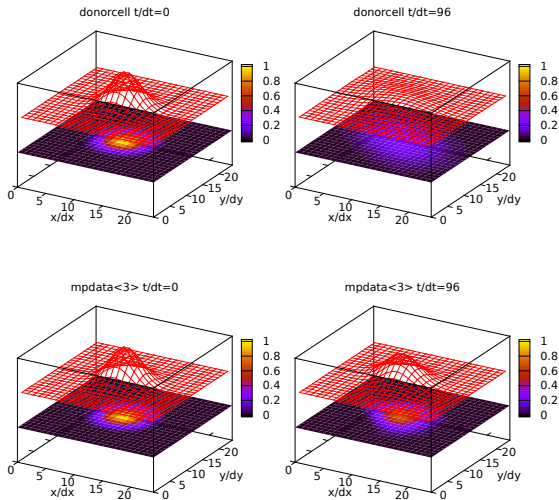


# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)





# MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



$$\sum_{d=0}^1 \psi_{[i,j]+\pi_{1,0}^d} \equiv \psi_{[i+1,j]} + \psi_{[i,j+1]}$$

$$C'^{[d]}_{[i,j]+\pi_{1/2,0}^d} = \left| C^{[d]}_{[i,j]+\pi_{1/2,0}^d} \right| \cdot \left[ 1 - \left| C^{[d]}_{[i,j]+\pi_{1/2,0}^d} \right| \right] \cdot A^{[d]}_{[i,j]}(\psi)$$

$$- \sum_{q=0, q \neq d}^N C^{[d]}_{[i,j]+\pi_{1/2,0}^d} \cdot \overline{C}^{[q]}_{[i,j]+\pi_{1/2,0}^d} \cdot B^{[d]}_{[i,j]}(\psi)$$

$$\overline{C}^{[q]}_{[i,j]+\pi_{1/2,0}^d} = \frac{1}{4} \cdot \left( C^{[q]}_{[i,j]+\pi_{1,1/2}^d} + C^{[q]}_{[i,j]+\pi_{0,1/2}^d} + \right.$$

$$\left. C^{[q]}_{[i,j]+\pi_{1,-1/2}^d} + C^{[q]}_{[i,j]+\pi_{0,-1/2}^d} \right)$$

$$A^{[d]}_{[i,j]} = \frac{\psi_{[i,j]+\pi_{1,0}^d} - \psi_{[i,j]}}{\psi_{[i,j]+\pi_{1,0}^d} + \psi_{[i,j]}}$$

$$B^{[d]}_{[i,j]} = \frac{\frac{1}{2} \psi_{[i,j]+\pi_{1,1}^d} + \psi_{[i,j]+\pi_{0,1}^d} - \psi_{[i,j]+\pi_{1,-1}^d} - \psi_{[i,j]+\pi_{0,-1}^d}}{\psi_{[i,j]+\pi_{1,1}^d} + \psi_{[i,j]+\pi_{0,1}^d} + \psi_{[i,j]+\pi_{1,-1}^d} + \psi_{[i,j]+\pi_{0,-1}^d}}$$

# MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

**integrated into CFD packages:**

# MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

**integrated into CFD packages:**

AtmosFOAM	C++ & Python/Numba	3D	 /AtmosFOAM	U. Reading
-----------	--------------------	----	------------------------------------------------------------------------------------------------	------------



# MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

**integrated into CFD packages:**

AtmosFOAM	C++ & Python/Numba	3D	 /AtmosFOAM	U. Reading
ROMS	FORTTRAN	3D	 /myroms	UCLA (?)




# MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

**integrated into CFD packages:**

AtmosFOAM	C++ & Python/Numba	3D	 /AtmosFOAM	U. Reading
ROMS	FORTTRAN	3D	 /myroms	UCLA (?)
PISM	C++	2D	 /pism	U. Alaska Fairbanks





# MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

**integrated into CFD packages:**

AtmosFOAM	C++ & Python/Numba	3D	 /AtmosFOAM	U. Reading
ROMS	FORTRAN	3D	 /myroms	UCLA (?)
PISM	C++	2D	 /pism	U. Alaska Fairbanks
babyEULAG	FORTRAN	2D	 /igfuw/bE_SDs	NCAR






# MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

**integrated into CFD packages:**

AtmosFOAM	C++ & Python/Numba	3D	 /AtmosFOAM	U. Reading
ROMS	FORTRAN	3D	 /myroms	UCLA (?)
PISM	C++	2D	 /pism	U. Alaska Fairbanks
babyEULAG	FORTRAN	2D	 /igfuw/bE_SDs	NCAR
apc-llc/mpdata	C/CUDA	3D	 /apc-llc	RAS (?)






# MPDATA implementations

known closed-source (Numerical Weather Prediction):


- COSMO
- ECMWF IFS

open-source:

**integrated into CFD packages:**

AtmosFOAM	C++ & Python/Numba	3D	 /AtmosFOAM	U. Reading
ROMS	FORTRAN	3D	 /myroms	UCLA (?)
PISM	C++	2D	 /pism	U. Alaska Fairbanks
babyEULAG	FORTRAN	2D	 /igfuw/bE_SDs	NCAR
apc-llc/mpdata	C/CUDA	3D	 /apc-llc	RAS (?)

**reusable:**

libmpdata++	C++/Blitz++	1,2,3D	 /igfuw	IGF FUW
-------------	-------------	--------	--------------------------------------------------------------------------------------------	---------








# MPDATA implementations

## known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

## open-source:

### integrated into CFD packages:

AtmosFOAM	C++ & Python/Numba	3D	 /AtmosFOAM	U. Reading
ROMS	FORTRAN	3D	 /myroms	UCLA (?)
PISM	C++	2D	 /pism	U. Alaska Fairbanks
babyEULAG	FORTRAN	2D	 /igfuw/bE_SDs	NCAR
apc-llc/mpdata	C/CUDA	3D	 /apc-llc	RAS (?)

### reusable:

libmpdata++	C++/Blitz++	1,2,3D	 /igfuw	IGF FUW
PyMPDATA	Python/Numba	1,2,3D	 /open-atmos	UJ, AGH

- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and usage examples
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)

## PyMPDATA: design goals, tech stack, features

- Numba JIT  $\rightsquigarrow$  pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)



PyMPDATA

## PyMPDATA: design goals, tech stack, features



- Numba JIT  $\rightsquigarrow$  pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- single-click/command installation on Linux, macOS & Windows (PyPI) (including all dependencies, no compilation)

## PyMPDATA: design goals, tech stack, features



**PyMPDATA**

- Numba JIT  $\rightsquigarrow$  pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- single-click/command installation on Linux, macOS & Windows (PyPI) (including all dependencies, no compilation)
- 92% unit-test coverage (codecov) and growing...

## PyMPDATA: design goals, tech stack, features



**PyMPDATA**

- Numba JIT  $\rightsquigarrow$  pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- single-click/command installation on Linux, macOS & Windows (PyPI) (including all dependencies, no compilation)
- 92% unit-test coverage (codecov) and growing...
- 5-class & single "advance" method API

## PyMPDATA: design goals, tech stack, features



- Numba JIT  $\rightsquigarrow$  pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- single-click/command installation on Linux, macOS & Windows (PyPI) (including all dependencies, no compilation)
- 92% unit-test coverage (codecov) and growing...
- 5-class & single "advance" method API
- docs (and CI) covers usage from Python, Julia, Matlab and Rust

## PyMPDATA: design goals, tech stack, features



- Numba JIT  $\rightsquigarrow$  pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- single-click/command installation on Linux, macOS & Windows (PyPI) (including all dependencies, no compilation)
- 92% unit-test coverage (codecov) and growing...
- 5-class & single "advance" method API
- docs (and CI) covers usage from Python, Julia, Matlab and Rust
- suite of 20+ Jupyter notebook examples maintained with the project all with badges enabling single-click execution on Colab or mybinder.org



# PyMPDATA: design goals, tech stack, features



- Numba JIT  $\rightsquigarrow$  pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- single-click/command installation on Linux, macOS & Windows (PyPI) (including all dependencies, no compilation)
- 92% unit-test coverage (codecov) and growing...
- 5-class & single "advance" method API
- docs (and CI) covers usage from Python, Julia, Matlab and Rust
- suite of 20+ Jupyter notebook examples maintained with the project all with badges enabling single-click execution on Colab or mybinder.org
- examples in 1D, 2D & 3D: advection-diffusion, bin cloud  $\mu$ -physics, spherical coordinates, shallow-water, Black-Scholes, Burgers, Boussinesq

- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- **PyMPDATA documentation and usage examples**
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)



## Documentation

### What is PyMPDATA?

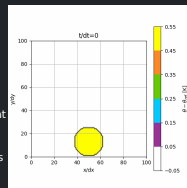
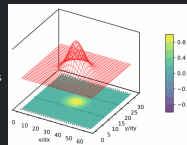
PyMPDATA is a **Numba-accelerated** **multi-threaded** Pythonic implementation of the **MPDATA algorithm** of Smolarkiewicz et al. used in geophysical fluid dynamics and beyond for **numerically solving generalised convection-diffusion PDEs**. PyMPDATA supports integration in 1D, 2D and 3D structured 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.

### What is the difference between PyMPDATA and PyMPDATA-examples?

PyMPDATA is a Python package that provides the MPDATA algorithm implementation. It is a library that can be used in your own projects.

PyMPDATA-examples is a Python package that provides examples of how to use PyMPDATA. It includes common Python modules used in PyMPDATA smoke tests and in example Jupyter notebooks (but the package wheels do not include the notebooks, only .py files imported from the notebooks and PyMPDATA tests).



## Bibliography with code cross-references

The list below summarises all literature references included in PyMPDATA codebase and includes links to both the referenced papers, as well as to the referring PyMPDATA source files.

1. Anderson & Fattahi 1974: "A Comparison of Numerical Solutions of the Advective Equation"
  - [examples/PyMPDATA\\_examples/Molenkamp\\_test\\_as\\_in\\_Jaruga\\_et\\_al\\_2015\\_Fig\\_12/demo.ipynb](#)
  - [examples/PyMPDATA\\_examples/wikipedia\\_example/demo.ipynb](#)
2. Arabas & Farhat 2020 (J. Comput. Appl. Math. 373): "Derivative pricing as a transport problem: MPDATA solutions to Black-Scholes-type equations"
  - [examples/PyMPDATA\\_examples/Arabas\\_and\\_Farhat\\_2020/\\_init\\_.py](#)
  - [examples/PyMPDATA\\_examples/Arabas\\_and\\_Farhat\\_2020/fig\\_1.ipynb](#)
  - [examples/PyMPDATA\\_examples/Arabas\\_and\\_Farhat\\_2020/fig\\_2.ipynb](#)
  - [examples/PyMPDATA\\_examples/Arabas\\_and\\_Farhat\\_2020/fig\\_3.ipynb](#)
  - [examples/PyMPDATA\\_examples/Arabas\\_and\\_Farhat\\_2020/tab\\_1.ipynb](#)
3. Arabas et al. 2014 (Sci. Prog. 22): "Formula Translation in Blitz++, NumPy and Modern Fortran: A Case Study of the Language Choice Tradeoffs"
  - [docs/markdown/pympdata\\_landing.md](#)
4. Barraquand & Pudet 1996 (Math. Financ. 6): "Pricing of American path-dependent contingent claims"
  - [examples/PyMPDATA\\_examples/Magnuszewski\\_et\\_al\\_2025/barraquand\\_data.py](#)
5. Bartman et al. 2022 (J. Open Source Soft. 7): "PyMPDATA v1: Numba-accelerated implementation of MPDATA with examples in Python, Julia and Matlab"
  - [examples/PyMPDATA\\_examples/Bartman\\_et\\_al\\_2022/\\_init\\_.py](#)
  - [examples/PyMPDATA\\_examples/Bartman\\_et\\_al\\_2022/fig\\_X.ipynb](#)
6. Beason & Margolin 1988 (Nuclear explosives code developer's conference, Boulder, CO, USA): "DPDC (double-pass donor cell): A second-order monotone scheme for advection"
  - [PyMPDATA/options.py](#)
  - [examples/docs/pympdata\\_examples\\_landing.md](#)
7. Capiński and Zastawniak 2012 (Cambridge University Press): "Numerical Methods in Finance with C++"
  - [examples/PyMPDATA\\_examples/Magnuszewski\\_et\\_al\\_2025/monte\\_carlo.py](#)
8. Jarecka et al. 2015 (J. Comp. Phys. 289): "A spreading drop of shallow water"
  - [examples/PyMPDATA\\_examples/jarecka\\_et\\_al\\_2015/\\_init\\_.py](#)
  - [examples/PyMPDATA\\_examples/jarecka\\_et\\_al\\_2015/fig\\_6.ipynb](#)
9. Jaruga et al. 2015 (Geosci. Model Dev. 8): "libmpdata++ 1.0: a library of parallel MPDATA solvers for systems of generalised transport equations"
  - [docs/markdown/pympdata\\_landing.md](#)
  - [examples/PyMPDATA\\_examples/jaruga\\_et\\_al\\_2015/\\_init\\_.py](#)
  - [examples/PyMPDATA\\_examples/jaruga\\_et\\_al\\_2015/fig19.ipynb](#)

Module Index

Contents

Introduction

Tutorial (in Python, Julia, Rust and Matlab)

Options class

Arakawa-C grid layer and boundary conditions

Stepper

Solver

Debugging

Contributing, reporting issues, seeking support

Other open-source MPDATA implementations:

Other Python packages for solving hyperbolic transport equations

Submodules

boundary\_conditions

impl

options

scalar\_field

As an example, the code below shows how to instantiate a scalar and a vector field given a 2D constant-velocity problem, using a grid of 24x24 points, Courant numbers of -0.5 and -0.25 in "x" and "y" directions, respectively, with periodic boundary conditions and with an initial Gaussian signal in the scalar field (settings as in Fig. 5 in [Arabas et al. 2014](#)):

► Julia code (click to expand)

▼ Matlab code (click to expand)

```
ScalarField = py.importlib.import_module('PyMPDATA').ScalarField;
VectorField = py.importlib.import_module('PyMPDATA').VectorField;
Periodic = py.importlib.import_module('PyMPDATA.boundary_conditions').Periodic;

nx = int32(24);
ny = int32(24);

Cx = -.5;
Cy = -.25;

[xi, yi] = meshgrid(double(0:1:nx-1), double(0:1:ny-1));

halo = options.n_halo;
advectee = ScalarField(pyargs(...
    'data', py.numpy.array(exp( ...
        -(xi+.5-double(nx)/2).^2 / (2*(double(nx)/10)^2) ...
        -(yi+.5-double(ny)/2).^2 / (2*(double(ny)/10)^2) ...
    )), ...
    'halo', halo, ...
    'boundary_conditions', py.tuple({Periodic(), Periodic()}) ...
));
advector = VectorField(pyargs(...
    'data', py.tuple({ ...
        Cx * py.numpy.ones(int32([nx+1 ny])), ...
        Cy * py.numpy.ones(int32([nx ny+1])) ...
    }), ...
    'halo', halo, ...
    'boundary_conditions', py.tuple({Periodic(), Periodic()}) ...
));
```

► Rust code (click to expand)

▼ Python code (click to expand)

PyMPDATA\_examples

Search...

Submodules

[Bjerk Sund and Stensland 1993](#)

[Black Scholes 1973](#)

[analysis\\_figures\\_2\\_and\\_3](#)

[analysis\\_table\\_1](#)


[colors](#)

[options](#)

[setup1\\_european\\_corridor](#)

[setup2\\_american\\_put](#)

[simulation](#)

built with 

## PyMPDATA\_examples

### .Arabas\_and\_Farhat\_2020

This example implements simulations presented in the [Arabas and Farhat 2020](#) study on pricing of European and American options using MPDATA.

Each notebook in this directory corresponds to a figure or a table in the paper.

fig\_1.ipynb:

[render on](#) [GitHub](#) [launch](#) [binder](#) [Open in Colab](#)

fig\_2.ipynb:

[render on](#) [GitHub](#) [launch](#) [binder](#) [Open in Colab](#)

fig\_3.ipynb:

[render on](#) [GitHub](#) [launch](#) [binder](#) [Open in Colab](#)

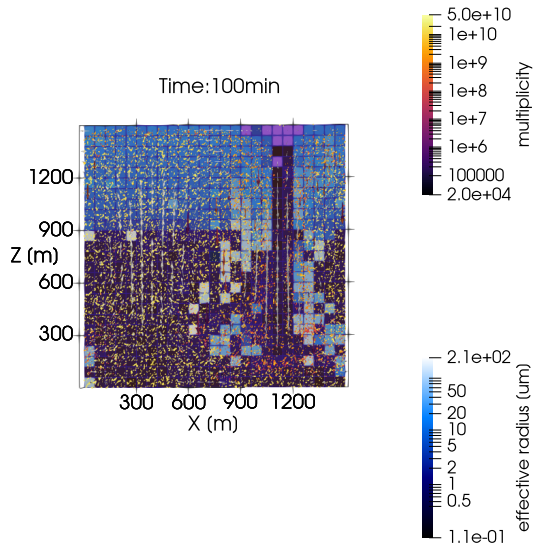
tab\_1.ipynb:

[render on](#) [GitHub](#) [launch](#) [binder](#) [Open in Colab](#)

[View Source](#)

# Eulerian transport for PySDM examples (the original reason for PyMPDATA dev)

<https://pypi.org/p/PySDM>



- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and usage examples
- **PyMPDATA performance vs. C++**
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)





DOI: [10.21105/joss.03896](https://doi.org/10.21105/joss.03896)

Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

---

Editor: [Arfon Smith](#) ↗

Reviewers:

- [@Chil](#)
- [@wdeconinck](#)

Submitted: 25 October 2021

Published: 05 September 2022

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## PyMPDATA v1: Numba-accelerated implementation of MPDATA with examples in Python, Julia and Matlab

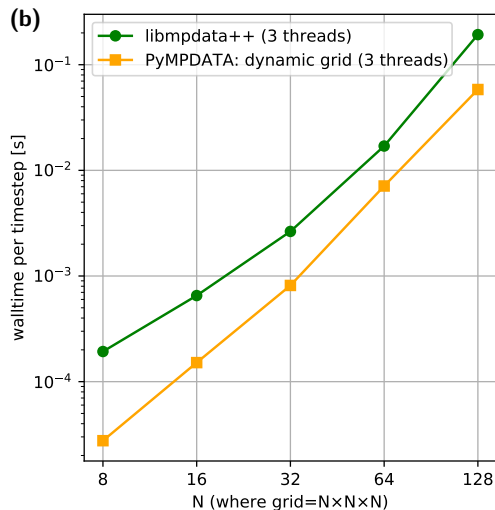
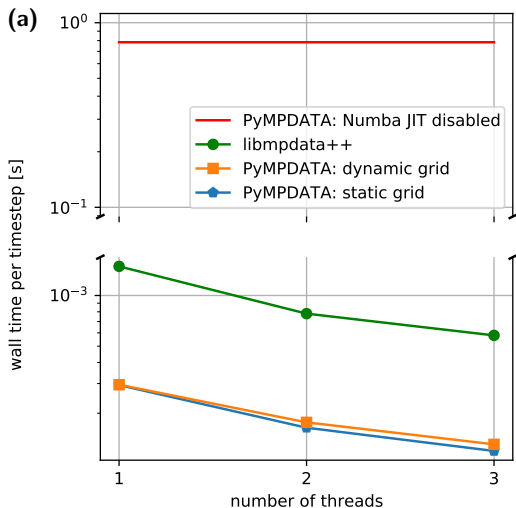
Piotr Bartman<sup>1</sup>, Jakub Banaśkiewicz<sup>1</sup>, Szymon Drenda<sup>1</sup>, Maciej Manna<sup>1</sup>, Michael A. Olesik<sup>1</sup>, Paweł Rozwoda<sup>1</sup>, Michał Sadowski<sup>1</sup>, and Sylwester Arabas<sup>1,2</sup>

<sup>1</sup> Jagiellonian University, Kraków, Poland <sup>2</sup> University of Illinois at Urbana-Champaign, IL, USA

### Statement of need

Convection-diffusion problems arise across a wide range of pure and applied research, in particular in geosciences, aerospace engineering, and financial modelling (for an overview of applications, see, e.g., section 1.1 in Morton (1996)). One of the key challenges in numerical solutions of problems involving advective transport is sign preservation of the advected field (for an overview of this and other aspects of numerical solutions to advection problems, see, e.g., Røed (2019)). The Multidimensional Positive Definite Advection Transport Algorithm (MPDATA) is a robust, explicit-in-time, and sign-preserving solver introduced in Smolarkiewicz (1983) and Smolarkiewicz (1984). MPDATA has been subsequently developed into a family of numerical schemes with numerous variants and solution procedures addressing a diverse set of problems in geophysical fluid dynamics and beyond. For reviews of MPDATA applications and variants, see, e.g., Smolarkiewicz & Margolin (1998) and Smolarkiewicz (2006).

# Numba JIT & multithreading: PyMPDATA vs. libmpdata++ performance



- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and usage examples
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)



Original software publication

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

Kacper Derlatka <sup>a 1</sup>, Maciej Manna <sup>a 2</sup>, Oleksii Bulenok <sup>a 3</sup>, David Zwicker <sup>b</sup>, Sylwester Arabas <sup>c</sup>  

<sup>a</sup> Faculty of Mathematics and Computer Science, Jagiellonian University in Kraków, Poland

<sup>b</sup> Max Planck Institute for Dynamics and Self-Organization, Göttingen, Germany

<sup>c</sup> 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.



Search projects

Help Docs Sponsors Log in Register

## pympdata-mpi 0.1.1

`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.

- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and usage examples
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)

## **code contributors (CS, math & physics students):**

Jakub Banaśkiewicz (UJ), **Piotr Bartman** (UJ), Kacper Derlatka (UJ, Pega), Szymon Drenda (UJ),  
Adrian Jaśkowiec (AGH), Piotr Karaś (AGH), Norbert Klockiewicz (AGH), Michał Kowalczyk (AGH),  
Kacper Majchrzak (AGH), Paweł Magnuszewski (AGH), Maciej Manna (UJ, Autodesk),  
Wojciech Neuman (AGH), Michael Olesik (UJ), Arkadiusz Paterak (AGH), Paulina Pojda (AGH),  
Wiktor Prosowicz (AGH), Weronika Romaniec (AGH), Paweł Rozwoda (UJ), Michał Sadowski (UJ),  
Jan Stryszewski (AGH), Michał Szczygiał (AGH), Michał Wroński (AGH), Joanna Wójcicka (AGH),  
Antoni Zięciak (AGH), Agnieszka Żaba (AGH), **YOU?!**

## **code contributors (CS, math & physics students):**

Jakub Banaśkiewicz (UJ), **Piotr Bartman** (UJ), Kacper Derlatka (UJ, Pega), Szymon Drenda (UJ), Adrian Jaśkowiec (AGH), Piotr Karaś (AGH), Norbert Klockiewicz (AGH), Michał Kowalczyk (AGH), Kacper Majchrzak (AGH), Paweł Magnuszewski (AGH), Maciej Manna (UJ, Autodesk), Wojciech Neuman (AGH), Michael Olesik (UJ), Arkadiusz Paterak (AGH), Paulina Pojda (AGH), Wiktor Prosowicz (AGH), Weronika Romaniec (AGH), Paweł Rozwoda (UJ), Michał Sadowski (UJ), Jan Stryszewski (AGH), Michał Szczygiał (AGH), Michał Wroński (AGH), Joanna Wójcicka (AGH), Antoni Zięciak (AGH), Agnieszka Żaba (AGH), **YOU?!**

## **funding:**

Foundation for Polish Science, Poland's National Science Centre

**Thank you for your attention!**

[sylwester.arabas@agh.edu.pl](mailto:sylwester.arabas@agh.edu.pl)