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

Sylwester Arabas
AGH University of Krakow

June 2025

plan of the talk

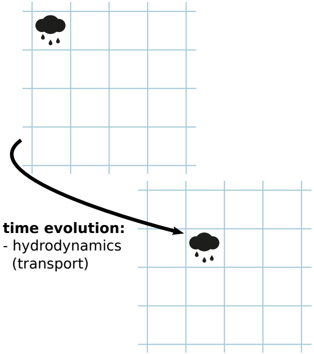
- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and usage examples
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)

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),
$\mathbf{v} = \{u, \ldots\} = G\dot{\mathbf{x}}$: flow velocity vector field (advector),
$G(\mathbf{x})$: fluid density, Jacobian of coordinate transformation, or their product

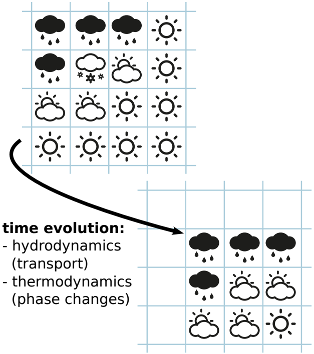


MPDATA key concepts: UPWIND discretisation

- advection equation / scalar conservation law:

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

$\psi(\mathbf{x}, t)$: advected scalar field (advectee),
$\mathbf{v} = \{u, \ldots\} = G\dot{\mathbf{x}}$: 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:

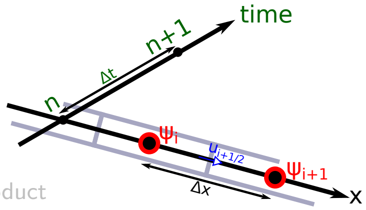
$$\partial_t(G\psi) + \nabla \cdot (\mathbf{v}\psi) = GR$$

$\psi(\mathbf{x}, t)$: advected scalar field (advectee),
$\mathbf{v} = \{u, \ldots\} = G\dot{\mathbf{x}}$: 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),
$\mathbf{v} = \{u, \ldots\} = G\dot{\mathbf{x}}$: 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 $C = u\frac{\Delta t}{\Delta x}$:

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

yields a conservative and sing-preserving "UPWIND" scheme which is stable for $|C| \leqslant 1$.

```python
def f(psi_l, psi_r, C):
    return .5 * (C + abs(C)) * psi_l + \
           .5 * (C - abs(C)) * psi_r
def upwind(psi: np.ndarray, i: slice, C: np.ndarray):
    psi[i] = psi[i] - (
        f(psi[i      ], psi[i + one], C[i + hlf]) -
        f(psi[i - one], psi[i      ], C[i - hlf])
    )
def solve_upwind(nt: int, C: np.ndarray, psi: np.ndarray):
    i = slice(1, len(psi) - 1)
    for _ in range(nt):
        upwind(psi, i, C)
```

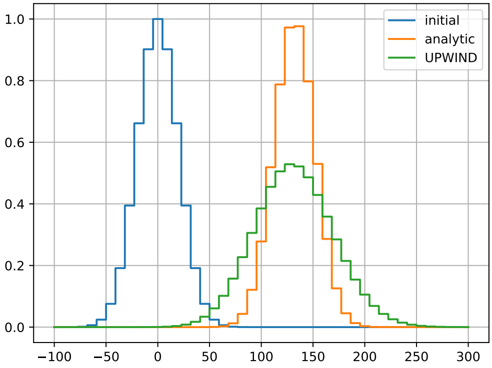
MPDATA key concepts: Courant number & UPWIND stability criterion

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

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

yields a conservative and sing-preserving "UPWIND" scheme which is stable for $|C| \leqslant 1$.

```python
def f(psi_l, psi_r, C):
    return .5 * (C + abs(C)) * psi_l + \
           .5 * (C - abs(C)) * psi_r
def upwind(psi: np.ndarray, i: slice, C: np.ndarray):
    psi[i] = psi[i] - (
        f(psi[i      ], psi[i + one], C[i + hlf]) -
        f(psi[i - one], psi[i      ], C[i - hlf])
    )
def solve_upwind(nt: int, C: np.ndarray, psi: np.ndarray)
    i = slice(1, len(psi) - 1)
    for _ in range(nt):
        upwind(psi, i, C)
```



MPDATA key concepts: numerical diffusion & modified-equation analysis

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

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

MPDATA key concepts: numerical diffusion & modified-equation analysis

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

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

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

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

MPDATA key concepts: numerical diffusion & modified-equation analysis

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

$$\psi_i^{n+1} = \psi_i^n + \partial_t\psi|_i^n\,(+\Delta t) + \frac{1}{2}\,\partial_t^2\psi\Big|_i^n\,(+\Delta t)^2 + O(\Delta t^3)$$
$$\psi_{i+1}^{n} = \psi_i^n + \partial_x\psi|_i^n\,(+\Delta x) + \frac{1}{2}\,\partial_x^2\psi\Big|_i^n\,(+\Delta x)^2 + O(\Delta x^3)$$
$$\psi_{i-1}^{n} = \psi_i^n + \partial_x\psi|_i^n\,(-\Delta x) + \frac{1}{2}\,\partial_x^2\psi\Big|_i^n\,(-\Delta x)^2 + O(\Delta x^3)$$

$$\rightsquigarrow \quad \psi_i^{n+1} = \psi_i^n - \left[\frac{C+|C|}{2}\left(\psi_i^n - \psi_{i-1}^n\right) + \frac{C-|C|}{2}\left(\psi_{i+1}^n - \psi_i^n\right)\right]$$

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

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

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:

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

(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)

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

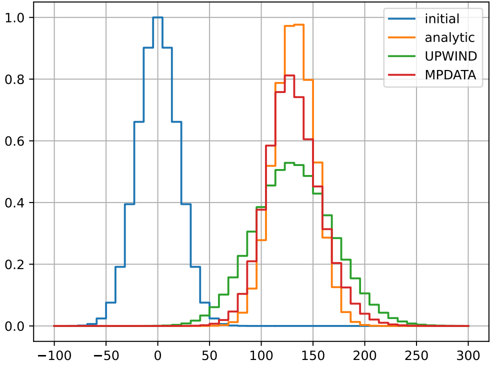
(Smolarkiewicz 1983 MWR, 1984 JCP: doi:10.1016/0021-9991(84)90121-9)

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

```python
def C_corr(C: np.ndarray, i: slice, psi: np.ndarray):
    return (abs(C[i - hlf]) - C[i - hlf] ** 2) * (
        psi[i] - psi[i - one]
    ) / (
        psi[i - one] + psi[i]
    )
def mpdata(nt: int, C: np.ndarray, psi: np.ndarray):
    i = slice(1, len(psi) - 1)
    i_ext = slice(1, len(psi))
    for _ in range(nt):
        upwind(psi, i, C)
        upwind(psi, i, C_corr(C, i_ext, psi))
```

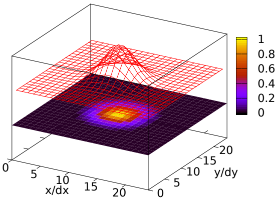
MPDATA hello-world (1D, single iteration) implementation

```python
def C_corr(C: np.ndarray, i: slice, psi: np.ndarray):
    return (abs(C[i - hlf]) - C[i - hlf] ** 2) * (
        psi[i] - psi[i - one]
    ) / (
        psi[i - one] + psi[i]
    )
def mpdata(nt: int, C: np.ndarray, psi: np.ndarray):
    i = slice(1, len(psi) - 1)
    i_ext = slice(1, len(psi))
    for _ in range(nt):
        upwind(psi, i, C)
        upwind(psi, i, C_corr(C, i_ext, psi))
```

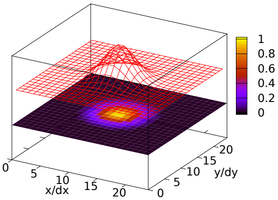


MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)

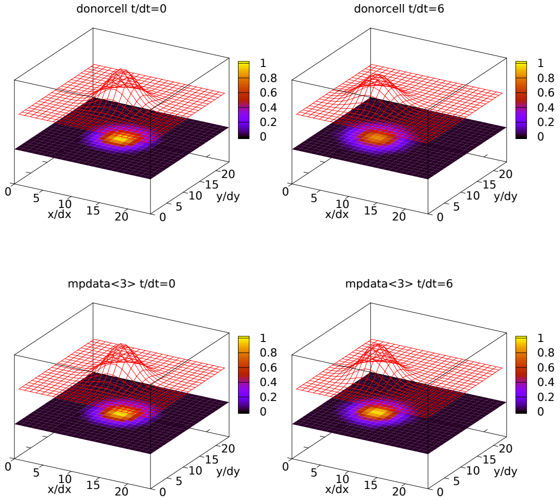
donorcell t/dt=0



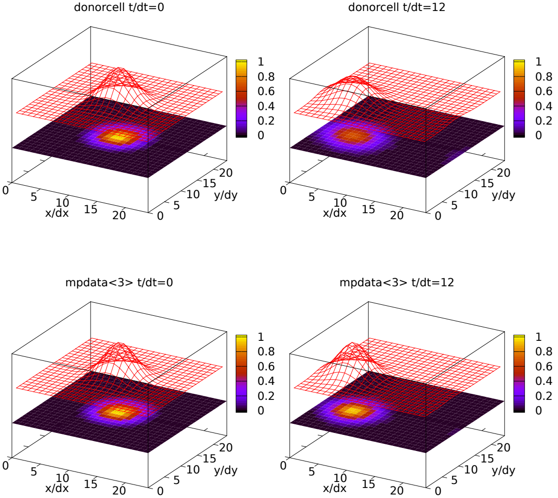
mpdata<3> t/dt=0



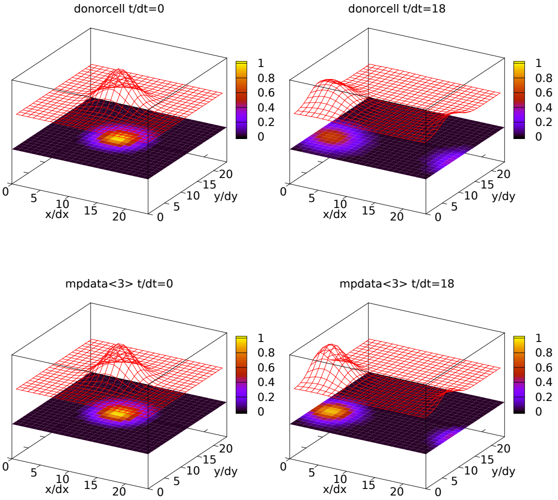
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



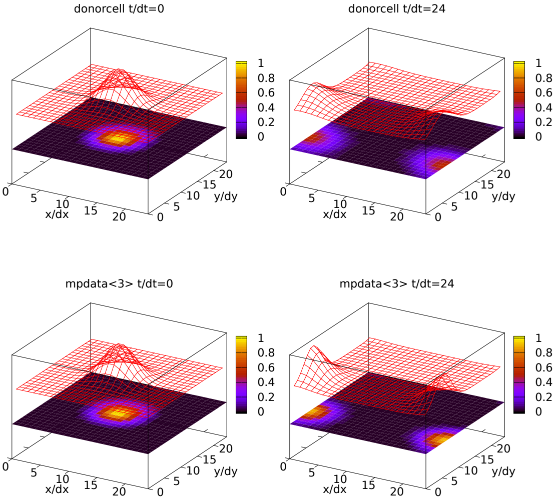
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



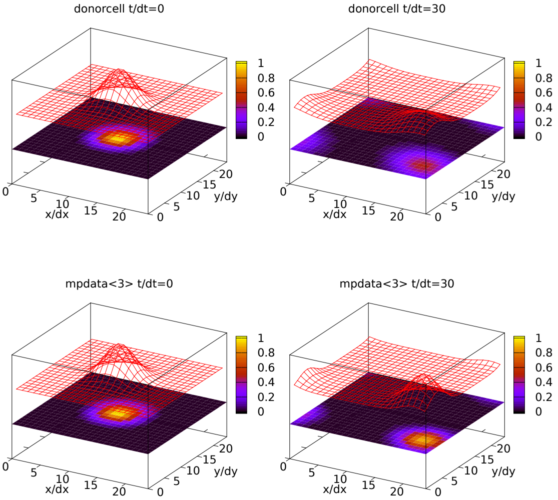
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



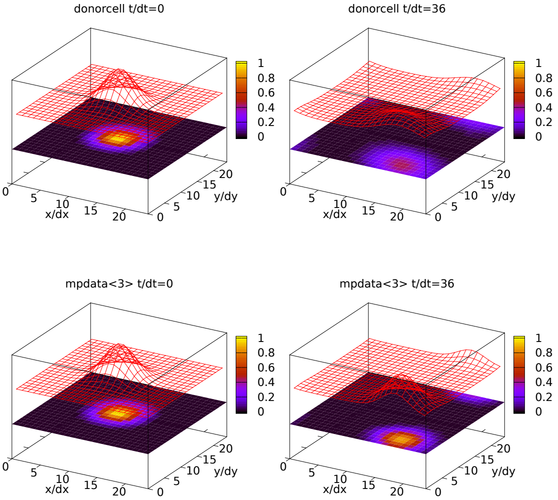
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



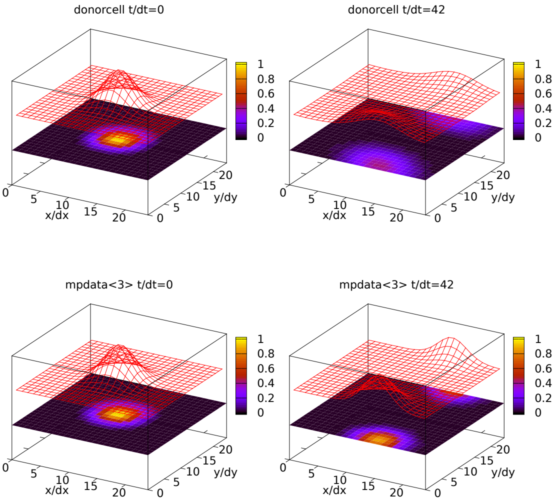
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



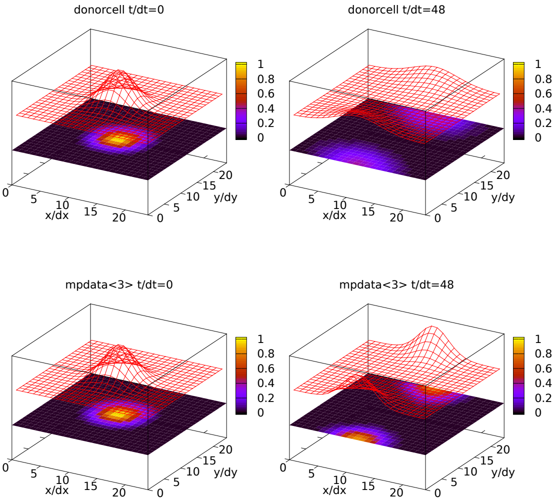
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



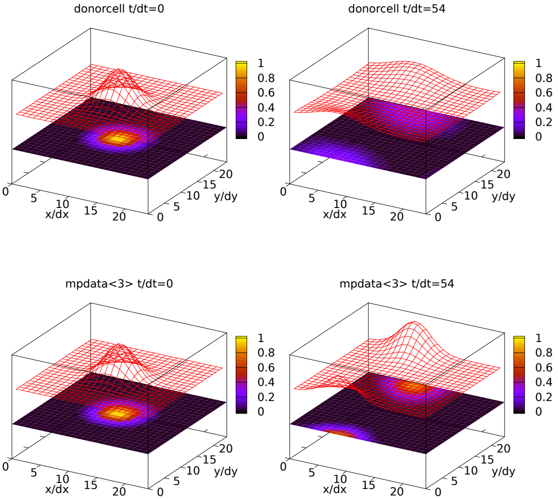
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



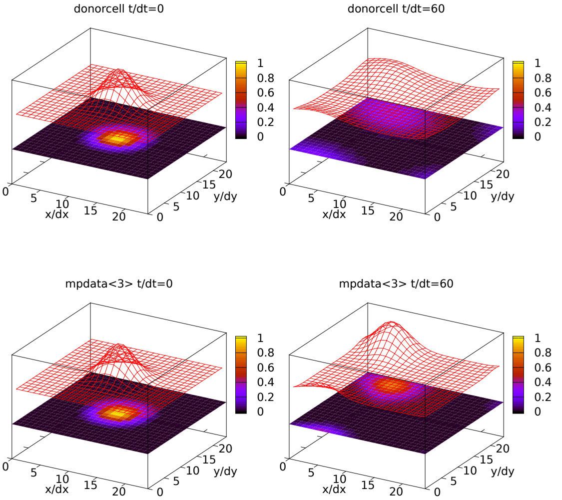
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



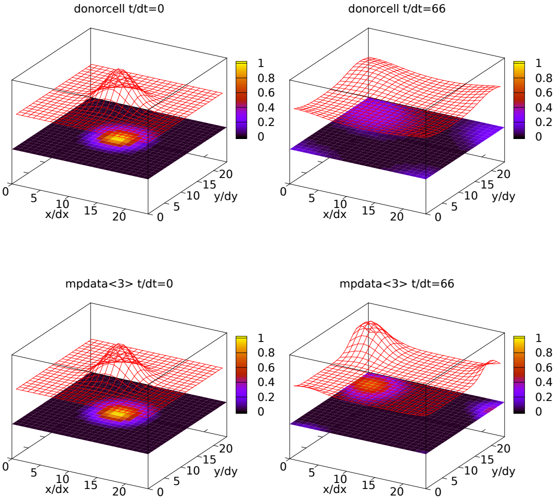
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



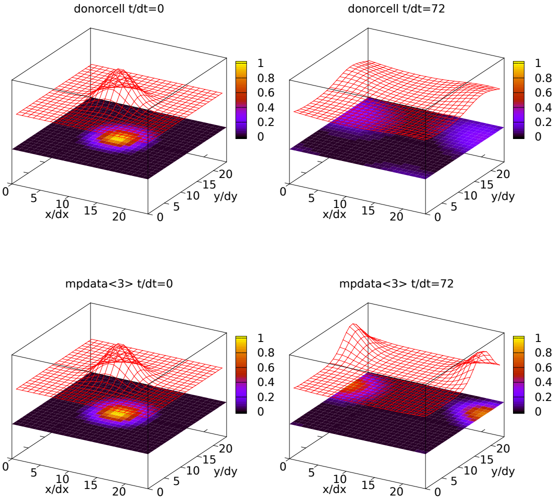
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



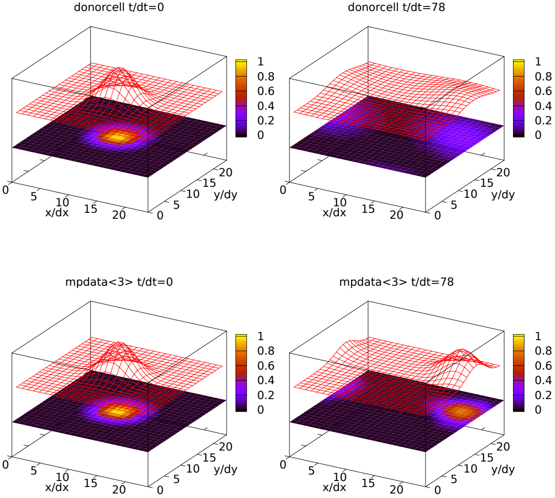
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



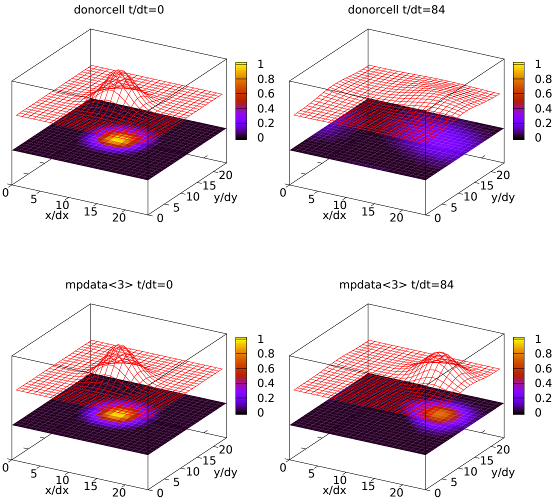
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



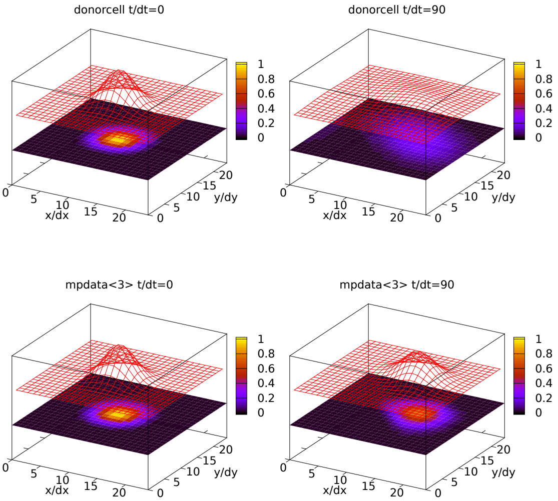
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



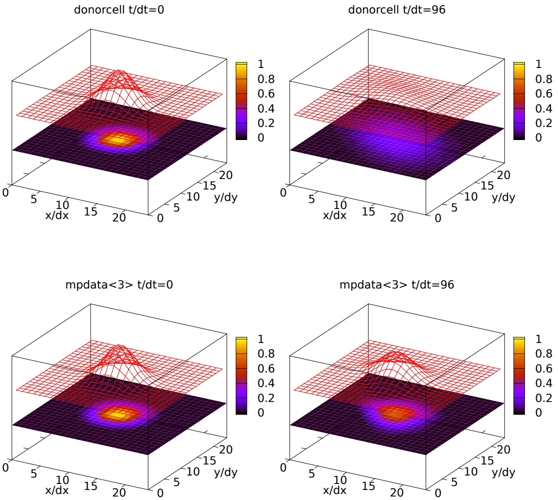
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



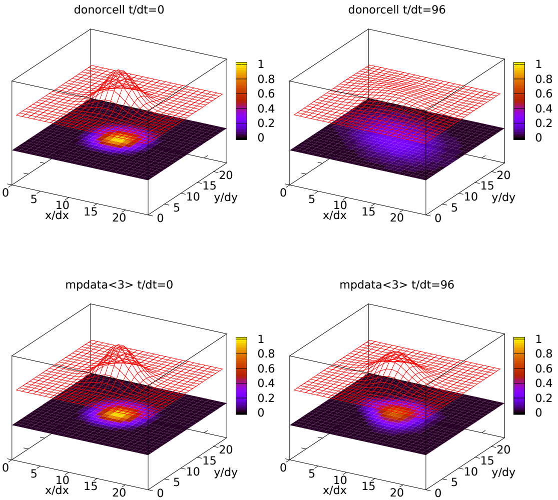
MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



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)



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

$$\begin{aligned}
C'^{[d]}_{[i,j]+\pi^d_{1/2,0}} = {} & \left|C^{[d]}_{[i,j]+\pi^d_{1/2,0}}\right| \cdot \left[1 - \left|C^{[d]}_{[i,j]+\pi^d_{1/2,0}}\right|\right] \cdot A^{[d]}_{[i,j]}(\psi) \\
& - \sum_{q=0,q\neq d}^{N} C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \cdot \overline{C}^{[q]}_{[i,j]+\pi^d_{1/2,0}} \cdot B^{[d]}_{[i,j]}(\psi)
\end{aligned}$$

$$\begin{aligned}
\overline{C}^{[q]}_{[i,j]+\pi^d_{1/2,0}} = \frac{1}{4} \cdot \Bigg(& C^{[q]}_{[i,j]+\pi^d_{1,1/2}} + C^{[q]}_{[i,j]+\pi^d_{0,1/2}} + \\
& C^{[q]}_{[i,j]+\pi^d_{1,-1/2}} + C^{[q]}_{[i,j]+\pi^d_{0,-1/2}} \Bigg)
\end{aligned}$$

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

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

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	FORTRAN	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	FORTRAN	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

plan of the talk

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

- 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

- 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

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

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

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
- 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

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
- 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 include 1D, 2D & 3D setups: advection-diffusion, bin cloud μ-physics, spherical coordinates, shallow-water, Black-Scholes, Burgers, Boussinesq

PyMPDATA v1.0: Bartman et al. 2022 (JOSS)

The Journal of Open Source Software

DOI: 10.21105/joss.03896

Software

- Review
- Repository
- Archive

Editor: Arfon Smith

Reviewers:

- @Chiil
- @wdeconinck

Submitted: 25 October 2021
Published: 05 September 2022

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

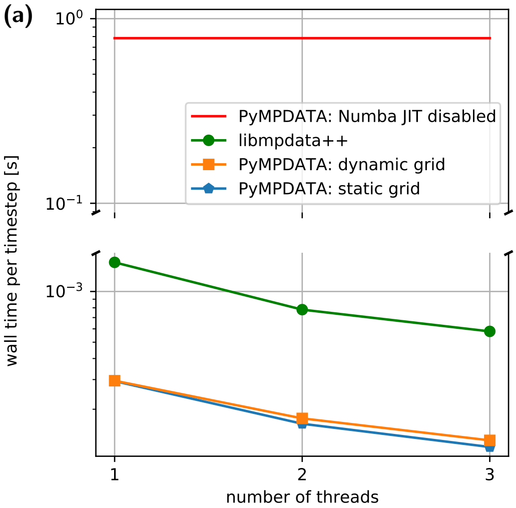
Piotr Bartman[1], **Jakub Banaśkiewicz**[1], **Szymon Drenda**[1], **Maciej Manna**[1], **Michael A. Olesik**[1], **Paweł Rozwoda**[1], **Michał Sadowski**[1], **and Sylwester Arabas**[1,2]

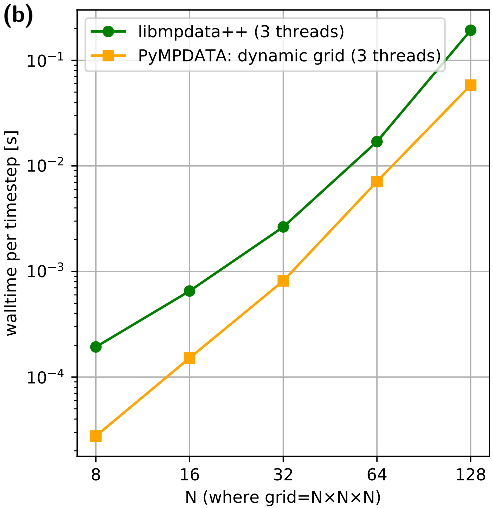
1 Jagiellonian University, Kraków, Poland **2** University of Illinois at Urbana-Champaign, IL, USA

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





plan of the talk

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

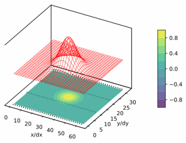
PyMPDATA & PyMPDATA-examples docs: open-atmos.github.io/PyMPDATA

PyMPDATA

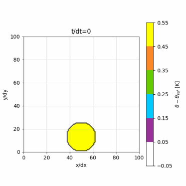
Documentation

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).

PyMPDATA & PyMPDATA-examples docs: open-atmos..io/PyMPDATA

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

PyMPDATA & PyMPDATA-examples docs: open-atmos.github.io/PyMPDATA

Module Index

Contents

Submodules

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 & PyMPDATA-examples docs: open-atmos.github.io/PyMPDATA

PyMPDATA_examples

Search...

Submodules

- Bjerksund_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 pdoc

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

PyMPDATA & PyMPDATA-examples docs: open-atmos.github.io/PyMPDATA

PyMPDATA

API Documentation

- class PostStepNull
  - call()
  - class_type
- class PostIterNull
  - call()
  - class_type
- class Solver
  - Solver()
  - advectee
  - advector
  - g_factor
  - advance()

built with pdoc

```
def advance(
    self,
    n_steps: int,
    mu_coeff: Optional[tuple] = None,
    post_step=None,
    post_iter=None
):
```

▾View Source

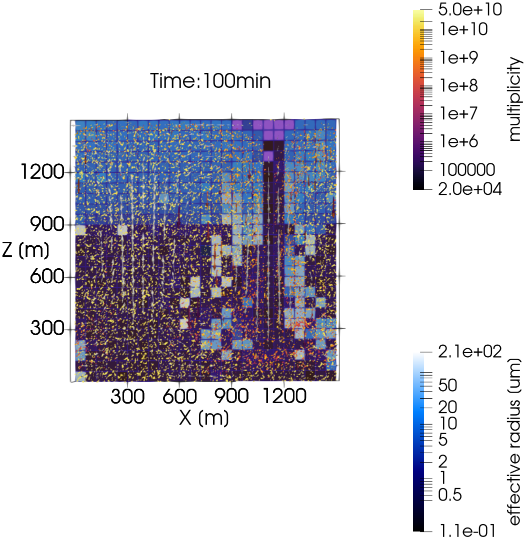
```
    def advance(
        self,
        n_steps: int,
        mu_coeff: Union[tuple, None] = None,
        post_step=None,
        post_iter=None,
    ):
        """advances solution by `n_steps` steps, optionally accepts: a tuple of dif
        coefficients (one value per dimension) as well as `post_iter` and `post_ste
        callbacks expected to be `numba.jitclass`es with a `call` method, for
        signature see `PostStepNull` and `PostIterNull`;
        returns CPU time per timestep (units as returned by `clock.clock()`)"""
        if mu_coeff is not None:
            assert self.__stepper.options.non_zero_mu_coeff
        else:
            mu_coeff = (0.0, 0.0, 0.0)
        if (
            self.__stepper.options.non_zero_mu_coeff
            and not self.__fields["g_factor"].meta[META_IS_NULL]
        ):
            raise NotImplementedError()

        post_step = post_step or PostStepNull()
        post_iter = post_iter or PostIterNull()

        return self.__stepper(
            n_steps=n_steps,
            mu_coeff=mu_coeff,
            post_step=post_step,
            post_iter=post_iter,
            fields=self.__fields,
        )
```

advances solution by `n_steps` steps, optionally accepts: a tuple of diffusion coefficients (one value per dimension) as well as `post_iter` and `post_step` callbacks expected to be `numba.jitclass`es with a `call` method, for signature see `PostStepNull` and `PostIterNull`; returns CPU time per timestep (units as returned by `clock.clock()`)

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

https://pypi.org/p/PySDM

PySDM



plan of the talk

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

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

ScienceDirect

SoftwareX

Volume 28, December 2024, 101897

Original software publication

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

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

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

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

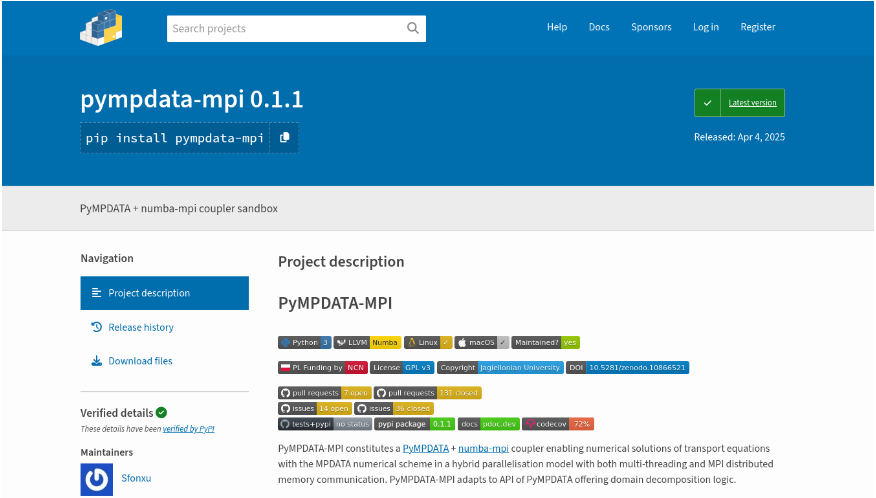
[c] Faculty of Physics and Applied Computer Science, AGH University of Krakow, Poland

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

Abstract

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

PyMPDATA-MPI



plan of the talk

- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and usage examples
- 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 Szczygieł (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 Szczygieł (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