Pydro

Nils Deppe

Jun 07, 2020

OVERVIEW:

Pydro is a collection of reconstruction schemes and other components needed to solve systems of equations that can develop discontinuities, such as Burgers equation, and compressible Newtonian Euler. If Numba is available the code will be JITed for better performance.

CHAPTER

ONE

DERIVATIVES

Pydro provides both midpoint-to-node and midpoint-and-node-to-node [1] finite-difference schemes with up to tenth-order accuracy. Additionally, variable-order methods are provided where the order of the difference scheme used is adjusted locally at each grid point according to an input array of orders to use at each grid point.

class Derivative.**Scheme**

An enum of the various different differentiation routines that are supported.

MD = 1

Second-order finite-difference derivative [1]

$$\frac{\partial q_i}{\partial x} \approx \frac{q_{i+1/2} - q_{i-1/2}}{\Delta x}$$

MD10 = 5

Tenth-order finite-difference derivative using only face values [1]

$$\begin{aligned}\frac{\partial q_i}{\partial x} \approx \frac{1}{\Delta x}\Bigg[&\frac{19845}{16384}(q_{i+1/2} - q_{i-1/2}) - \frac{735}{8192}(q_{i+3/2} - q_{i-3/2}) \qquad (1.1)\\ &+\frac{567}{40960}(q_{i+5/2} - q_{i-5/2}) - \frac{405}{229376}(q_{i+7/2} - q_{i-7/2}) \qquad (1.2)\\ &+\frac{35}{294912}(q_{i+9/2} - q_{i-9/2})\Bigg] \qquad (1.3)\end{aligned}$$

MD4 = 2

Fourth-order finite-difference derivative using only face values [1]

$$\frac{\partial q_i}{\partial x} \approx \frac{1}{\Delta x}\left[\frac{9}{8}(q_{i+1/2} - q_{i-1/2}) - \frac{1}{24}(q_{i+3/2} - q_{i-3/2})\right]$$

MD6 = 3

Sixth-order finite-difference derivative using only face values [1]

$$\begin{aligned}\frac{\partial q_i}{\partial x} \approx \frac{1}{\Delta x}\Bigg[&\frac{75}{64}(q_{i+1/2} - q_{i-1/2}) - \frac{25}{384}(q_{i+3/2} - q_{i-3/2}) \qquad (1.4)\\ &+\frac{3}{640}(q_{i+5/2} - q_{i-5/2})\Bigg] \qquad (1.5)\end{aligned}$$

MD8 = 4

Eighth-order finite-difference derivative using only face values [1]

$$\begin{aligned}\frac{\partial q_i}{\partial x} \approx \frac{1}{\Delta x}\Bigg[&\frac{1225}{1024}(q_{i+1/2} - q_{i-1/2}) - \frac{245}{3072}(q_{i+3/2} - q_{i-3/2}) \qquad (1.6)\\ &+\frac{49}{5120}(q_{i+5/2} - q_{i-5/2}) - \frac{5}{7168}(q_{i+7/2} - q_{i-7/2})\Bigg] \qquad (1.7)\end{aligned}$$

MDV = 6

Variable-order finite-difference derivative using only face values.

The order is adjusted according to the *order_used* argument passed to `Derivative.differentiate_flux()`

MND10 = 10

Tenth-order finite-difference derivative using face and node values (MND) [1]

$$\frac{\partial q_i}{\partial x} \approx \frac{1}{\Delta x}\left[\frac{5}{3}(q_{i+1/2} - q_{i-1/2}) - \frac{10}{21}(q_{i+1} - q_{i-1}) + \frac{5}{42}(q_{i+3/2} - q_{i-3/2}) - \frac{5}{252}(q_{i+2} - q_{i-2}) + \frac{1}{630}(q_{i+5/2} - q_{i-5/2})\right]$$

MND4 = 7

Fourth-order finite-difference derivative using face and node values (MND) [1]

$$\frac{\partial q_i}{\partial x} \approx \frac{1}{\Delta x}\left[\frac{4}{3}(q_{i+1/2} - q_{i-1/2}) - \frac{1}{6}(q_{i+1} - q_{i-1})\right]$$

MND6 = 8

Sixth-order finite-difference derivative using face and node values (MND) [1]

$$\frac{\partial q_i}{\partial x} \approx \frac{1}{\Delta x}\left[\frac{3}{2}(q_{i+1/2} - q_{i-1/2}) - \frac{3}{10}(q_{i+1} - q_{i-1}) + \frac{1}{30}(q_{i+3/2} - q_{i-3/2})\right]$$

MND8 = 9

Eigth-order finite-difference derivative using face and node values (MND) [1]

$$\frac{\partial q_i}{\partial x} \approx \frac{1}{\Delta x}\left[\frac{8}{5}(q_{i+1/2} - q_{i-1/2}) - \frac{2}{5}(q_{i+1} - q_{i-1}) + \frac{8}{105}(q_{i+3/2} - q_{i-3/2}) - \frac{1}{140}(q_{i+2} - q_{i-2})\right]$$

MNDV = 11

Variable-order finite-difference derivative using face and node values (MND).

The order is adjusted according to the *order_used* argument passed to `Derivative.differentiate_flux()`

`Derivative.`**`differentiate_flux`**(*scheme*, *dx*, *numerical_fluxes*, *center_flux=None*, *order_used=None*)

Compute the derivatives using the *scheme* and spacing *dx*.

Applies the finite-difference scheme given by the *scheme* argument to all variables in the *numerical_fluxes*.

Parameters

- **scheme** (`Derivative.Scheme`) – The finite-difference scheme to use.
- **dx** (*double*) – The grid spacing
- **numerical_fluxes** (*list*) – The numerical fluxes at the cell faces.
- **center_flux** (*list*) – The flux at the cell centers. Only needed for MND schemes.
- **order_used** (*list*) – A list of *int* at each cell indicating the finite-difference order to use at the cell. Normally this order is determined by the reconstruction scheme.

CHAPTER

TWO

PLOTTING 1D SIMULATION

Pydro provides plotting routines for plotting snapshots at a single time, as well as for plotting spacetime diagrams of quantities.

`Plotting.`**`generate_plot_with_reference`**(*x*, *x_ref*, *func*, *reference_solution*, *quantity_name*, *file_name*, *ref_label*, *every_n=0*, *set_log_y=False*)

Generate a plot of a snapshot from a 1d simulation and write it to disk.

Parameters

- **`x`** (*`list`*) – The x-coordinates on which *func* is defined.
- **`x_ref`** (*`list`*) – The x-coordinates on which *reference_solution* is defined.
- **`func`** (*`list`*) – The function/variable to plot.
- **`reference_solution`** (*`list`*) – The reference solution to plot.
- **`quantity_name`** (*`str`*) – The label for the function/variable being plotted.
- **`file_name`** (*`str`*) – The name of the file to write.
- **`ref_label`** (*`str`*) – The reference label. Typically either 'Exact' or 'Reference' depending on whether the reference solution is an exact analytic solution or obtained from a high-resolution simulation.
- **`every_n`** (*`int`*) – If non-zero only plot every nth grid point in *func*.
- **`set_log_y`** (*`bool`*) – If *True* then the y-axis is plotted on a log scale.

`Plotting.`**`generate_spacetime_plot`**(*file_name*, *var*, *var_name*, *x*, *times*, *smoothen*, *set_log_y*, *vmin=None*, *vmax=None*, *time_max_elements=100*, *x_max_elements=200*)

Generate a spacetime plot from a 1d simulation and write it to disk.

Parameters

- **`file_name`** (*`str`*) – The name of the file to write.
- **`var`** (*`list`*) – A list of the variable/function to plot at each time. The data must be in the same order as the *times* argument.
- **`var_name`** (*`str`*) – The name of the variable being plotted.
- **`x`** (*`list`*) – The x-coordinates on which *var* is defined. Assumed to be time-independent.
- **`times`** (*`list`*) – A list of the times at which data was recorded during the evolution.
- **`smoothen`** (*`bool`*) – If *True* applies a *gouraud* smoothening to the data during rendering.
- **`set_log_y`** (*`bool`*) – If *True* then the log of *var* is plotted.
- **`vmin`** (*`double`*) – If specified sets the lower limit of the range for plotting *var*.

- **vmax** (*double*) – If specified sets the upper limit of the range for plotting *var*.
- **time_max_elements** (*int*) – The maximum number of cells to plot in time. Increasing this increases the temporal resolution but makes the generated images larger and more expensive to render.
- **x_max_elements** (*int*) – The maximum number of cells to plot in space. Increasing this increases the temporal resolution but makes the generated images larger and more expensive to render.

CHAPTER

THREE

RECONSTRUCTION SCHEMES

Pydro provides two variants of reconstruction schemes. The first are the standard reconstruction schemes including TVD and WENO-type schemes, while the second are positivity-preserving adaptive-order schemes. The two scheme classes have different interfaces and so must be used slightly differently. The interface for each is documented in the sections below.

3.1 Standard reconstruction

The standard reconstruction schemes such as total variation diminishing (TVD) schemes and different flavors of higher-order essentially non-oscillatory or weighted compact schemes are available through a common interface. The `Reconstruction.reconstruct()` allows reconstructing a series of variables using one of the available schemes (see `Reconstruction.Scheme` for a list).

class `Reconstruction.`**`Scheme`**

An enum of the various different reconstruction schemes that are supported.

`Minmod = 1`

Minmod reconstruction.

Minmod reconstruction is performed as

$$\sigma_j = \operatorname{minmod}\left(\frac{q_i - q_{i-1}}{\Delta\xi}, \frac{q_{i+1} - q_i}{\Delta\xi}\right). \tag{3.1}$$

where $\Delta\xi$ is the grid spacing and $\operatorname{minmod}(a, b)$ is defined as

$$\operatorname{minmod}(a, b) = \begin{cases} \operatorname{sgn}(a)\min(|a|, |b|) & \text{if } \operatorname{sgn}(a) = \operatorname{sgn}(b) \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

The reconstructed solution at the faces is given by

$$\hat{q}_{i+1/2} = q_i + \frac{\Delta\xi}{2}\sigma_i$$

See, e.g. section 9.3.1 of [2] for a discussion.

`Wcns3 = 2`

Third order weighted compact nonlinear scheme reconstruction [3].

Third order WCNS3 reconstruction is done by first defining oscillation indicators β_0 and β_1 as

$$\beta_0 = (q_i - q_{i-1})^2 \tag{3.3}$$

$$\beta_1 = (q_{i+1} - q_i)^2 \tag{3.4}$$

Then coefficients α_k are defined as

$$\alpha_k = \frac{c_k}{(\beta_k + \epsilon_k)^2}$$

where ϵ_k is a factor used to avoid division by zero and is set to

$$\epsilon_0 = 10^{-17}\left(1 + |q_i| + |q_{i-1}|\right) \tag{3.5}$$
$$\epsilon_1 = 10^{-17}\left(1 + |q_i| + |q_{i+1}|\right) \tag{3.6}$$

and the linear weights are $c_0 = 1/4$ and $c_1 = 3/4$. Finally, we define the nonlinear weights:

$$\omega_k = \frac{\alpha_k}{\sum_{k=0}^{1} \alpha_k}$$

The reconstruction stencils are given by:

$$q^0_{i+1/2} = \frac{3}{2}q_i - \frac{1}{2}q_{i-1} \tag{3.7}$$
$$q^1_{i+1/2} = \frac{1}{2}q_i + \frac{1}{2}q_{i+1} \tag{3.8}$$

The final reconstructed solution is given by

$$\hat{q}_{i+1/2} = \sum_{k=0}^{1} \omega_k q^k_{i+1/2}$$

`Wcns5 = 4`

Fifth order weighted compact nonlinear scheme reconstruction [1].

The oscillation indicators are given by

$$\beta_0 = \frac{1}{4}\left(q_{i-2} - 4q_{i-1} + 3q_i\right)^2 + \left(q_{i-2} - 2q_{i-1} + q_i\right)^2 \tag{3.9}$$
$$\beta_1 = \frac{1}{4}\left(q_{i-1} - q_{i+1}\right)^2 + \left(q_{i-1} - 2q_i + q_{i+1}\right)^2 \tag{3.10}$$
$$\beta_2 = \frac{1}{4}\left(3q_i - 4q_{i+1} + q_{i+2}\right)^2 + \left(q_i - 2q_{i+1} + q_{i+2}\right)^2 \tag{3.11}$$

Then coefficients α_k are defined as

$$\alpha_k = \frac{c_k}{(\beta_k + \epsilon_k)^2}$$

where ϵ_k is a factor used to avoid division by zero and is set to

$$\epsilon_0 = 2 \times 10^{-16}\left(1 + |q_i| + |q_{i-1}| + |q_{i-2}|\right) \tag{3.12}$$
$$\epsilon_1 = 2 \times 10^{-16}\left(1 + |q_i| + |q_{i+1}| + |q_{i-1}|\right) \tag{3.13}$$
$$\epsilon_2 = 2 \times 10^{-16}\left(1 + |q_i| + |q_{i+1}| + |q_{i+2}|\right) \tag{3.14}$$

and the linear weights are $c_0 = 1/16, c_1 = 10/16$, and $c_2 = 5/16$. Finally, we define the nonlinear weights:

$$\omega_k = \frac{\alpha_k}{\sum_{k=0}^{2} \alpha_k}$$

The reconstruction stencils are given by:

$$q^0_{i+1/2} = \frac{3}{8}q_{i-2} - \frac{5}{4}q_{i-1} + \frac{15}{8}q_i, \tag{3.15}$$
$$q^1_{i+1/2} = -\frac{1}{8}q_{i-1} + \frac{3}{4}q_i + \frac{3}{8}q_{i+1}, \tag{3.16}$$
$$q^2_{i+1/2} = \frac{3}{8}q_i + \frac{3}{4}q_{i+1} - \frac{1}{8}q_{i+2} \tag{3.17}$$

The final reconstructed solution is given by

$$\hat{q}_{i+1/2} = \sum_{k=0}^{2} \omega_k q^k_{i+1/2}$$

`Wcns5Weno = 6`

Fifth order weighted compact nonlinear scheme reconstruction with the Jiang and Shu [4] weights.

Follows the procedure of *`Wcns5()`* except using the oscillation indicators given by

$$\beta_0 = \frac{1}{4}\left(q_{i-2} - 4q_{i-1} + 3q_i\right)^2 + \frac{13}{12}\left(q_{i-2} - 2q_{i-1} + q_i\right)^2 \tag{3.18}$$

$$\beta_1 = \frac{1}{4}\left(q_{i-1} - q_{i+1}\right)^2 + \frac{13}{12}\left(q_{i-1} - 2q_i + q_{i+1}\right)^2 \tag{3.19}$$

$$\beta_2 = \frac{1}{4}\left(-3q_i + 4q_{i+1} - q_{i+2}\right)^2 + \frac{13}{12}\left(q_i - 2q_{i+1} + q_{i+2}\right)^2 \tag{3.20}$$

`Wcns5z = 5`

Fifth order weighted compact nonlinear scheme reconstruction with the Z oscillation indicator.

Follows the procedure of *`Wcns5()`* except using the oscillation indicators given by

$$\beta^Z_k = \frac{\beta_k + \epsilon_k}{\beta_k + \tau_5 + \epsilon_k}$$

where

$$\tau_5 = |\beta_2 - \beta_0|$$

and the oscillation indicators are the ones from Jiang and Shu [4], as described in *`Wcns5Weno()`*.

`Weno3 = 3`

Third order weighted essentially non-oscillarity reconstruction.

The same as the *`Wcns3()`* reconstruction except with $c_0 = 1/3$ and $c_1 = 2/3$.

`Reconstruction.`**`reconstruct`**(*vars_to_reconstruct*, *scheme*, *order_used*)

Reconstructs all variables using the requested scheme.

Parameters

- **`vars_to_reconstruct`** (*`list of list of double`*) – The variables at the cell centers.
- **`scheme`** (`Reconstruction.Scheme`) – The reconstruction scheme to use.
- **`order_used`** (*`list of int`*) – Filled by the function and is used to return the order of the reconstruction used.

Returns (*list of list of double*) The face reconstructed variables. Each variable is of length *2 * number_of_cells*

3.2 Positivity-preserving reconstruction

The PPAO schemes need to be wrapped in a function specific for each evolution system so that the appropriate variables can have their positivity preserved.

`ReconstructionPpao.`**`adaptive_order_1`**(*q*, *i*, *j*, *recons*)

First-order reconstruction.

First-order reconstruction is given by

$$\hat{q}_{i+1/2} = q_i$$

`ReconstructionPpao.`**`adaptive_order_3`**(*q*, *i*, *j*, *recons*, *keep_positive*, *alpha=3.0*, *eps=1e-36*)

Uses a third-order centered stencil for reconstruction

$$\hat{q}_{i+1/2} = \frac{1}{8}q_{i-1} + \frac{3}{4}q_i + \frac{3}{8}q_{i+1}$$

How oscillatory the resulting polynomial is can be determined by comparing

$$s_N^j = \frac{1}{2}\log_{10}\left(\frac{\bar{\kappa}_N}{\kappa_N + \epsilon}\right),$$

where

$$\bar{\kappa}_3 = \frac{2}{5}\left(\frac{3}{4}q_{i+1} - \frac{3}{2}q_i + \frac{3}{4}q_{i-1}\right)^2, \tag{3.21}$$

$$\kappa_3 = \left(\frac{3}{8}q_{i+1} + \frac{1}{4}q_i + \frac{3}{8}q_{i-1}\right)\left(\frac{31}{20}q_{i+1} - \frac{1}{10}q_i + \frac{11}{20}q_{i-1}\right) \tag{3.22}$$

to

$$-\alpha\log_{10}(4)$$

Typically $\alpha \sim 4$ so that the coefficients decay as $1/3^4$.

Parameters

- **`q`** (*`list of double`*) – The variable values at the cell centers.
- **`i`** (*`int`*) – The index into the reconstructed array
- **`j`** (*`int`*) – The index of the cell whose faces are being reconstructed in *q*
- **`recons`** (*`list of double`*) – The array of the reconstructed variable.
- **`keep_positive`** (*`bool`*) – If *True* then returns *False* if the reconstructed solution is not positive.
- **`alpha`** (*`double`*) – The expected decay of increasing coefficients in the method.
- **`eps`** (*`double`*) – The *epsilon* parameter to ignore small values and impose an absolute tolerance.

Returns (*bool*) *True* if the reconstruction was successful, otherwise *False*

`ReconstructionPpao.`**`adaptive_order_5`**(*q*, *i*, *j*, *recons*, *keep_positive*, *alpha=5.0*, *eps=1e-36*)

Uses a fifth-order centered stencil for reconstruction

$$\hat{q}_{i+1/2} = \frac{3}{128}q_{i-2} - \frac{5}{32}q_{i-1} + \frac{45}{64}q_i + \frac{15}{32}q_{i+1} - \frac{5}{128}q_{i+2}$$

Parameters

- **q** (*list of double*) – The variable values at the cell centers.
- **i** (*int*) – The index into the reconstructed array
- **j** (*int*) – The index of the cell whose faces are being reconstructed in *q*
- **recons** (*list of double*) – The array of the reconstructed variable.
- **keep_positive** (*bool*) – If *True* then returns *False* if the reconstructed solution is not positive.
- **alpha** (*double*) – The expected decay of increasing coefficients in the method.
- **eps** (*double*) – The *epsilon* parameter to ignore small values and impose an absolute tolerance.

Returns (*bool*) *True* if the reconstruction was successful, otherwise *False*

`ReconstructionPpao.`**`adaptive_order_7`**(*q*, *i*, *j*, *recons*, *alpha=5.0*, *eps=1e-36*)

Uses a seventh-order centered stencil for reconstruction

$$\hat{q}_{i+1/2} = -\frac{5}{1024}q_{i-3} + \frac{21}{512}q_{i-2} - \frac{175}{1024}q_{i-1} + \frac{175}{256}q_i \tag{3.23}$$

$$+\frac{525}{1024}q_{i+1} - \frac{35}{512}q_{i+2} + \frac{7}{1024}q_{i+3} \tag{3.24}$$

Parameters

- **q** (*list of double*) – The variable values at the cell centers.
- **i** (*int*) – The index into the reconstructed array
- **j** (*int*) – The index of the cell whose faces are being reconstructed in *q*
- **recons** (*list of double*) – The array of the reconstructed variable.
- **keep_positive** (*bool*) – If *True* then returns *False* if the reconstructed solution is not positive.
- **alpha** (*double*) – The expected decay of increasing coefficients in the method.
- **eps** (*double*) – The *epsilon* parameter to ignore small values and impose an absolute tolerance.

Returns (*bool*) *True* if the reconstruction was successful, otherwise *False*

`ReconstructionPpao.`**`adaptive_order_9`**(*q*, *i*, *j*, *recons*, *alpha=5.0*, *eps=1e-36*)

Uses a ninth-order centered stencil for reconstruction

$$\hat{q}_{i+1/2} = \frac{35}{32768}q_{i-4} - \frac{45}{4096}q_{i-3} + \frac{441}{8291}q_{i-2} - \frac{735}{4096}q_{i-1} \tag{3.25}$$

$$+\frac{11025}{16384}q_i + \frac{2205}{4096}q_{i+1} - \frac{735}{8192}q_{i+2} + \frac{63}{4096}q_{i+3} \tag{3.26}$$

$$-\frac{45}{32768}q_{i+4} \tag{3.27}$$

Parameters

- **q** (*list of double*) – The variable values at the cell centers.
- **i** (*int*) – The index into the reconstructed array
- **j** (*int*) – The index of the cell whose faces are being reconstructed in *q*
- **recons** (*list of double*) – The array of the reconstructed variable.

- **keep_positive** (*bool*) – If *True* then returns *False* if the reconstructed solution is not positive.
- **alpha** (*double*) – The expected decay of increasing coefficients in the method.
- **eps** (*double*) – The *epsilon* parameter to ignore small values and impose an absolute tolerance.

Returns (*bool*) *True* if the reconstruction was successful, otherwise *False*

`ReconstructionPpao.`**`adaptive_order_wcns3`**(*q*, *i*, *j*, *recons*, *keep_positive*, *eps=1e-17*, *c0=0.25*, *c1=0.75*, *liu_indicators=True*, *exponent=2*)

A general third order weight compact nonlinear scheme.

Third order WCNS3 reconstruction is done by first defining oscillation indicators β_0 and β_1 as

$$\beta_0 = (q_i - q_{i-1})^2 \tag{3.28}$$

$$\beta_1 = (q_{i+1} - q_i)^2 \tag{3.29}$$

We refer to these as the standard oscillation indicators, but also provide the improved oscillation indicators of Liu [5]:

$$\beta_0 = \frac{1}{4}\left(|q_{i+1} - q_{i-1}| - |4q_i - 3q_{i-1} - q_{i+1}|\right)^2, \tag{3.30}$$

$$\beta_1 = \frac{1}{4}\left(|q_{i+1} - q_{i-1}| - |3q_{i+1} + q_{i-1} - 4q_i|\right)^2 \tag{3.31}$$

Then coefficients α_k are defined as

$$\alpha_k = \frac{c_k}{(\beta_k + \epsilon_k)^p}$$

where ϵ_k is a factor used to avoid division by zero and is set to

$$\epsilon_0 = \epsilon\left(1 + |q_i| + |q_{i-1}|\right) \tag{3.32}$$

$$\epsilon_1 = \epsilon\left(1 + |q_i| + |q_{i+1}|\right) \tag{3.33}$$

and the linear weights are $c_0 = 1/4$ and $c_1 = 3/4$. Finally, we define the nonlinear weights:

$$\omega_k = \frac{\alpha_k}{\sum_{k=0}^{1} \alpha_k}$$

The reconstruction stencils are given by:

$$q^0_{i+1/2} = \frac{3}{2}q_i - \frac{1}{2}q_{i-1}, \tag{3.34}$$

$$q^1_{i+1/2} = \frac{1}{2}q_i + \frac{1}{2}q_{i+1}, \tag{3.35}$$

The final reconstructed solution is given by

$$\hat{q}_{i+1/2} = \sum_{k=0}^{1} \omega_k q^k_{i+1/2}$$

Parameters

- **q** (*list of double*) – The variable values at the cell centers.
- **i** (*int*) – The index into the reconstructed array
- **j** (*int*) – The index of the cell whose faces are being reconstructed in *q*

- **recons** (*list of double*) – The array of the reconstructed variable.
- **keep_positive** (*bool*) – If *True* then returns *False* if the reconstructed solution is not positive.
- **eps** (*double*) – The *epsilon* parameter to avoid division by zero.
- **c0** (*double*) – The optimal linear weight c_0. For 3rd order use $1/4$. For 2nd order but increased robustness use $1/2$.
- **c1** (*double*) – The optimal linear weight c_1. For 3rd order use $3/4$. For 2nd order but increased robustness use $1/2$.
- **liu_indicators** (*bool*) – If *True* use the oscillation indicators of [5]
- **exponent** (*int*) – The exponent p in denominator of the α_k

Returns (*bool*) *True* if the reconstruction was successful, otherwise *False*

`ReconstructionPpao.`**`adaptive_order_wcns3z`** (*q*, *i*, *j*, *recons*, *keep_positive*, *eps=1e-17*, *c0=0.25*, *c1=0.75*, *liu_indicators=True*)

A general third order weight compact nonlinear scheme using the Z weights.

The same as `adaptive_order_wcns3()` except that the Z weights are used. First we define

$$\tau_3 = |\beta_1 - \beta_0|$$

Then the new α_k are given by

$$\alpha_k = c_k \left(1 + \frac{\tau_3}{\beta_k + \epsilon_k}\right),$$

Parameters

- **q** (*list of double*) – The variable values at the cell centers.
- **i** (*int*) – The index into the reconstructed array
- **j** (*int*) – The index of the cell whose faces are being reconstructed in q
- **recons** (*list of double*) – The array of the reconstructed variable.
- **keep_positive** (*bool*) – If *True* then returns *False* if the reconstructed solution is not positive.
- **eps** (*double*) – The *epsilon* parameter to avoid division by zero.
- **c0** (*double*) – The optimal linear weight c_0. For 3rd order use $1/4$. For 2nd order but increased robustness use $1/2$.
- **c1** (*double*) – The optimal linear weight c_1. For 3rd order use $3/4$. For 2nd order but increased robustness use $1/2$.
- **liu_indicators** (*bool*) – If *True* use the oscillation indicators of [5]

Returns (*bool*) *True* if the reconstruction was successful, otherwise *False*

`ReconstructionPpao.`**`adaptive_order_weno3_robust`** (*q*, *i*, *j*, *recons*, *keep_positive*, *eps=1e-17*, *c1=1.0*, *c2=1000.0*, *c3=1.0*, *exponent=4*, *wenoz=False*)

A robust WENO3 reconstruction using 5 points.

The individual polynomials stencils for the reconstruction are written as

$$u(\xi) = u_0 + u_\xi P_1(\xi) + u_{\xi\xi} P_2(\xi) \tag{3.36}$$

The left-, central-, and right-biased stencils for the one-dimensional coefficients are:

$$u_\xi^{(L)} = \frac{1}{2}u_{-2} - 2u_{-1} + \frac{3}{2}u_0 \tag{3.37}$$

$$u_{\xi\xi}^{(L)} = \frac{u_{-2} - 2u_{-1} + u_0}{2} \tag{3.38}$$

$$u_\xi^{(C)} = \frac{1}{2}(u_1 - u_{-1}) \tag{3.39}$$

$$u_{\xi\xi}^{(C)} = \frac{u_{-1} - 2u_0 + u_1}{2} \tag{3.40}$$

$$u_\xi^{(R)} = -\frac{3}{2}u_0 + 2u_1 - \frac{1}{2}u_2 \tag{3.41}$$

$$u_{\xi\xi}^{(R)} = \frac{u_0 - 2u_1 + u_2}{2} \tag{3.42}$$

The oscillation indicators are given by

$$\beta_{(i)} = \left(u_\xi^{(i)}\right)^2 + \frac{13}{3}\left(u_{\xi\xi}^{(i)}\right)^2,$$

where $i \in \{L, C, R\}$. The nonlinear weights are:

$$\omega_k = \frac{\alpha_k}{\sum_{l=0}^{2}\alpha_l} \tag{3.43}$$

$$\alpha_k = \frac{\lambda_k}{(\beta_k + \epsilon_k)^p} \tag{3.44}$$

where p is usually chosen to be 4 or 8, and $\lambda_0 = 1$, $\lambda_1 = 10^5$, and $\lambda_2 = 1$.

To obtain the WENOZ weights use $p = 1$ and with the new oscillation indicators

$$\beta_k^Z = \frac{\beta_k}{\beta_k + \tau_5 + \epsilon_k}$$

where

$$\tau_5 = |\beta_3 - \beta_1|.$$

Parameters

- **q** (*list of double*) – The variable values at the cell centers.
- **i** (*int*) – The index into the reconstructed array
- **j** (*int*) – The index of the cell whose faces are being reconstructed in *q*
- **recons** (*list of double*) – The array of the reconstructed variable.
- **keep_positive** (*bool*) – If *True* then returns *False* if the reconstructed solution is not positive.
- **eps** (*double*) – The *epsilon* parameter to avoid division by zero.
- **c0** (*double*) – The linear weight λ_0.
- **c1** (*double*) – The linear weight λ_1.
- **c2** (*double*) – The linear weight λ_2.
- **exponent** (*double*) – The exponent p in denominator of the α_k.
- **wenoz** (*bool*) – If *True* then use the WENOZ weights.

Returns (*bool*) *True* if the reconstruction was successful, otherwise *False*

CHAPTER FOUR

TIME STEPPERS

Pydro offers a few different time steppers, including many strong-stability-preserving (SSP) time steppers. The linear Runge-Kutta (RK) time steppers, while simple, should be avoided since they can only achieve the promised order of accuracy for linear systems. The SSP RK3 and SSP RK4 are both robust choices for nonlinear hydrodynamics simulations. The SSP RK4 has a maximum step size roughly 50% larger than an Euler step and the SSP RK3, making it a more efficient method than the SSP RK3 There are also (mostly complete) Adams-Bashforth time steppers up to fourth order, which still require a self-starting procedure in order to truly reach the high-order accuracy.

class `TimeStepper.`**`LinearRk4Ssp`**(*time_deriv*, *initial_state*, *initial_time*)

A fourth-order SSP linear Runge-Kutta method from [6]

Warning: This time stepper is a linear method and should not be used on nonlinear systems.

Warning: This time stepper passes *None* as the time.

Parameters

- **`time_deriv`** (*`funcion`*) – The time derivative function which must be invokable with two arguments, the evolved variables and the time.
- **`initial_state`** (*`list`*) – A list of the values of the variables at the initial time.
- **`time`** (*`double`*) – Initial time.

`get_cfl_coefficient`()

Returns the CFL factor required for stability.

`get_evolved_vars`()

Returns an list of the evolved variables.

`get_time`()

Returns the current time.

`take_step`(*dt*)

Take a time step.

Parameters **`dt`** (*`double`*) – The time step size to use already including the CFL coefficient for the time stepper.

class `TimeStepper.`**`LinearRk6Ssp`**(*time_deriv*, *initial_state*, *initial_time*)

A sixth-order SSP linear Runge-Kutta method from [6]

Warning: This time stepper is a linear method and should not be used on nonlinear systems.

Warning: This time stepper passes *None* as the time.

Parameters

- **`time_deriv`** (*`funcion`*) – The time derivative function which must be invokable with two arguments, the evolved variables and the time.
- **`initial_state`** (*`list`*) – A list of the values of the variables at the initial time.
- **`time`** (*`double`*) – Initial time.

`get_cfl_coefficient``()`
Returns the CFL factor required for stability.

`get_evolved_vars``()`
Returns an list of the evolved variables.

`get_time``()`
Returns the current time.

`take_step``(`*dt*`)`
Take a time step.

Parameters **`dt`** (*`double`*) – The time step size to use already including the CFL coefficient for the time stepper.

`class` `TimeStepper.`**`LinearRk8Ssp`**`(`*time_deriv*, *initial_state*, *initial_time*`)`
A eighth-order SSP linear Runge-Kutta method from [6]

Warning: This time stepper is a linear method and should not be used on nonlinear systems.

Warning: This time stepper passes *None* as the time.

Parameters

- **`time_deriv`** (*`funcion`*) – The time derivative function which must be invokable with two arguments, the evolved variables and the time.
- **`initial_state`** (*`list`*) – A list of the values of the variables at the initial time.
- **`time`** (*`double`*) – Initial time.

`get_cfl_coefficient``()`
Returns the CFL factor required for stability.

`get_evolved_vars``()`
Returns an list of the evolved variables.

`get_time``()`
Returns the current time.

take_step(*dt*)
Take a time step.

Parameters **dt** (*double*) – The time step size to use already including the CFL coefficient for the time stepper.

class TimeStepper.**Rk3Ssp**(*time_deriv*, *initial_state*, *initial_time*)
A third-order strong-stability-preserving nonlinear Runge-Kutta time stepper [7]

Denoting the time derivative operator by $\mathcal{L}$, the stepper is given by

$$v^{(1)} = q^n + \Delta t \mathcal{L}(u^n, t^n) \tag{4.1}$$

$$v^{(2)} = \frac{1}{4}\left[3q^n + v^{(1)} + \Delta t \mathcal{L}\left(v^{(1)}, t^n + \Delta t\right)\right] \tag{4.2}$$

$$q^{n+1} = \frac{1}{3}\left[q^n + 2v^{(2)} + 2\Delta t \mathcal{L}\left(v^{(2)}, t^n + \frac{1}{2}\Delta t\right)\right] \tag{4.3}$$

Parameters

- **time_deriv** (*funcion*) – The time derivative function which must be invokable with two arguments, the evolved variables and the time.
- **initial_state** (*list*) – A list of the values of the variables at the initial time.
- **time** (*double*) – Initial time.

get_cfl_coefficient()
Returns the CFL factor required for stability.

get_evolved_vars()
Returns an list of the evolved variables.

get_time()
Returns the current time.

take_step(*dt*)
Take a time step.

Parameters **dt** (*double*) – The time step size to use already including the CFL coefficient for the time stepper.

class TimeStepper.**Rk4Ssp**(*time_deriv*, *initial_state*, *initial_time*)
A fourth-order strong-stability-preserving nonlinear Runge-Kutta time stepper [7]

Denoting the time derivative operator by $\mathcal{L}$, the stepper is given by

$$v^{(1)} = q^n + 0.39175222700392\Delta t\mathcal{L}(u^n, t^n) \tag{4.4}$$

$$v^{(2)} = 0.44437049406734q^n + 0.55562950593266v^{(1)} \tag{4.5}$$

$$+ 0.36841059262959\Delta t\mathcal{L}\left(v^{(1)}, t^n + 0.39175222700392\Delta t\right) \tag{4.6}$$

$$v^{(3)} = 0.6201018513854q^n + 0.37989814861464v^{(2)} \tag{4.7}$$

$$+ 0.25189177424738\Delta t\mathcal{L}\left(v^{(2)}, t^n + 0.5860796889678\Delta t\right) \tag{4.8}$$

$$v^{(4)} = 0.17807995410773q^n + 0.82192004589227v^{(3)} \tag{4.9}$$

$$+ 0.54497475021237\Delta t\mathcal{L}\left(v^{(3)}, t^n + 0.47454236302687\Delta t\right) \tag{4.10}$$

$$q^{n+1} = 0.00683325884039q^n + 0.51723167208978v^{(2)} \tag{4.11}$$

$$+ 0.12759831133288v^{(3)} + 0.34833675773694v^{(4)} \tag{4.12}$$

$$+ 0.08460416338212\Delta t\mathcal{L}\left(v^{(3)}, t^n + 0.47454236302687\Delta t\right) \tag{4.13}$$

$$+ 0.22600748319395\Delta t\mathcal{L}\left(v^{(4)}, t^n + 0.93501063100924\Delta t\right) \tag{4.14}$$

(4.15)

Parameters

- **`time_deriv`** (*`funcion`*) – The time derivative function which must be invokable with two arguments, the evolved variables and the time.
- **`initial_state`** (*`list`*) – A list of the values of the variables at the initial time.
- **`time`** (*`double`*) – Initial time.

`get_cfl_coefficient``()`
Returns the CFL factor required for stability.

`get_evolved_vars``()`
Returns an list of the evolved variables.

`get_time``()`
Returns the current time.

`take_step``(`*dt*`)`
Take a time step.

Parameters **`dt`** (*`double`*) – The time step size to use already including the CFL coefficient for the time stepper.

CHAPTER

FIVE

REFERENCES

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Taku Nonomura and Kozo Fujii. Robust explicit formulation of weighted compact nonlinear scheme. *Computers & Fluids*, 85:8 – 18, 2013. International Workshop on Future of CFD and Aerospace Sciences. doi:10.1016/j.compfluid.2012.09.001.

[2] Rezzolla, L. and Zanotti, O. *Relativistic Hydrodynamics*. Oxford University Press, sep 2013. URL: {http://adsabs.harvard.edu/abs/2013rehy.book.....R}.

[3] Xiaogang Deng and Hanxin Zhang. Developing high-order weighted compact nonlinear schemes. *Journal of Computational Physics*, 165(1):22 – 44, 2000. doi:10.1006/jcph.2000.6594.

[4] Guang-Shan Jiang and Chi-Wang Shu. Efficient implementation of weighted eno schemes. *Journal of Computational Physics*, 126(1):202 – 228, 1996. doi:10.1006/jcph.1996.0130.

[5] Shengping Liu, Yiqing Shen, Bei Chen, and Fangjun Zeng. Novel local smoothness indicators for improving the third-order weno scheme. *International Journal for Numerical Methods in Fluids*, 87(2):51–69, 2018. doi:10.1002/fld.4480.

[6] Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Review*, 43:, 05 2001. doi:10.1137/S003614450036757X.

[7] Jan S Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

PYTHON MODULE INDEX

INDEX