Py4Incompact3D Documentation

Release 0.0.0

Yorgos Deskos
Paul Bartholomew

Nov 17, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Py4Incompact3D is a library for postprocessig data produced by Xcompact3D simulations. The aim of this project is to facillitate automated postprocessing of Xcompact3D simulations by providing, at first:

- Mesh class: this stores the domain data for the simulation
- Case class: this stores the information of the case: boundary conditions, fields etc.

With these building blocks, complex postprocessing tools may be built - for example, derivative calculateors to compute the vorticity and Q-criterion given the velocity field.

1.1 Installation

- Clone the git repository to a location on your `${PYTHONPATH}`
- Test module can be imported by python interpreter: `import Py4Incompact3D`
- [Optional] Install `h5py` using `pip3 install h5py` to work with HDF5 files generated by Xcompact3d - note this is only intended as a temporary solution until a proper integration with ADIOS2 is implemented.

1.2 Documentation

Documentation of functions can be found under *doc/build/latex/*.

To regenerate documentation, from the project root type `make -C doc/ latexpdf` (requires sphinx).

1.3 Contributing

It is hoped that users of Xcompact3D will find this library useful and contribute to its development, for instance by adding additional functionality.

CHAPTER

TWO

API

2.1 Postprocess

class Py4Incompact3D.postprocess.postprocess.**Postprocess**(**args*, ***kwargs*)

Postprocess is the highest level class of the Py4Incompact3D package. Import this class and instantiate it with a path to an input file to begin running Py4Incompact3D. Use the ``fields" attribute to access other objects within the model.

inputs: input_file: str - path to the nml input file

outputs: self: post - an instantiated post object

clear_data(*vars='all'*)

Clear stored data fields.

load(***kwargs*)

Load data.

write(***kwargs*)

Write data.

2.2 Mesh

class Py4Incompact3D.postprocess.mesh.**Mesh**(**arg*, ***kwargs*)

Mesh is a model object representing

compute_derivvars()

Compute variables required by derivative functions.

get_grid()

Return the x,y,z arrays that describe the mesh.

2.3 Derivatives

`Py4Incompact3D.deriv.deriv.`**`compute_deriv`**(*rhs*, *bc*, *npaire*)

Compute the derivative by calling to TDMA.

Parameters

- **rhs** (*numpy.ndarray*) – The rhs vector.
- **bc** (*int*) – The boundary condition for the axis.
- **npaire** (*bool*) – Does the field not 'point' in the same direction as the derivative?

Returns The derivative

Return type numpy.ndarray

`Py4Incompact3D.deriv.deriv.`**`compute_rhs`**(*postproc*, *field*, *axis*, *time*, *bc*)

Compute the rhs for the derivative.

Parameters

- **postproc** – The basic postprocessing object.
- **field** (*str*) – The name of the variable who's derivative we want.
- **axis** (*int*) – A number indicating direction in which to take derivative: 0=x; 1=y; 2=z.
- **time** (*int*) – The time to compute rhs for.
- **bc** (*int*) – The boundary condition: 0=periodic; 1=free-slip; 2=Dirichlet.

Returns rhs – the right-hand side vector.

Return type numpy.ndarray

`Py4Incompact3D.deriv.deriv.`**`compute_rhs_0`**(*mesh*, *field*, *axis*)

Compute the rhs for the derivative for periodic BCs.

Parameters

- **mesh** (`Py4Incompact3D.postprocess.mesh.Mesh`) – The mesh on which derivatives are taken.
- **field** – The field for the variable who's derivative we want.
- **axis** (*int*) – A number indicating direction in which to take derivative: 0=x; 1=y; 2=z.

Returns rhs – the right-hand side vector.

Return type numpy.ndarray

`Py4Incompact3D.deriv.deriv.`**`compute_rhs_1`**(*mesh*, *field*, *axis*, *field_direction*)

Compute the rhs for the derivative for free slip BCs.

Parameters

- **mesh** (`Py4Incompact3D.postprocess.mesh.Mesh`) – The mesh on which derivatives are taken.
- **field** (*np.ndarray*) – The field for the variable who's derivative we want.
- **axis** (*int*) – A number indicating direction in which to take derivative: 0=x; 1=y; 2=z.
- **field_direction** (*list of int*) – Indicates the direction of the field: -1=scalar; 0=x; 1=y; 2=z.

Returns rhs – the right-hand side vector.

Return type numpy.ndarray

`Py4Incompact3D.deriv.deriv.`**`compute_rhs_2`**(*mesh*, *field*, *axis*)

Compute the rhs for the derivative for Dirichlet BCs.

Parameters

- **`mesh`** (`Py4Incompact3D.postprocess.mesh.Mesh`) – The mesh on which derivatives are taken.
- **`field`** – The field for the variable who's derivative we want.
- **`axis`** (*`int`*) – A number indicating direction in which to take derivative: 0=x; 1=y; 2=z.

Returns rhs – the right-hand side vector.

Return type numpy.ndarray

`Py4Incompact3D.deriv.deriv.`**`deriv`**(*postproc*, *phi*, *axis*, *time*)

Take the derivative of field 'phi' along axis.

Parameters

- **`postproc`** (`Py4Incompact3D.postprocess.postprocess.Postprocess`) – The basic Postprocess object.
- **`phi`** (*`str`*) – The name of the variable who's derivative we want.
- **`axis`** (*`int`*) – A number indicating direction in which to take derivative: 0=x; 1=y; 2=z.
- **`time`** (*`int`*) – The time stamp to compute derivatives for.

Returns dphidx – the derivative

Return type numpy.ndarray

`Py4Incompact3D.deriv.deriv.`**`tdma`**(*a*, *b*, *c*, *rhs*, *overwrite=True*)

The Tri-Diagonal Matrix Algorithm.

Solves tri-diagonal matrices using TDMA where the matrices are of the form [b0 c0

a1 b1 c1 a2 b2 c2

an-2 bn-2 cn-1 an-1 bn-1]

Parameters

- **`a`** (*`numpy.ndarray`*) – The `left' coefficients.
- **`b`** (*`numpy.ndarray`*) – The diagonal coefficients. (All ones?)
- **`c`** (*`numpy.ndarray`*) – The 'right' coefficients.
- **`rhs`** (*`numpy.ndarray`*) – The right-hand side vector.
- **`overwrite`** (*`bool`*) – Should the rhs and diagonal coefficient (b) arrays be overwritten?

Returns rhs – the rhs vector overwritten with derivatives.

Return type numpy.ndarray

`Py4Incompact3D.deriv.deriv.`**`tdma_periodic`**(*a*, *b*, *c*, *rhs*)

Periodic form of Tri-Diagonal Matrix Algorithm.

Solves periodic tri-diagonal matrices using TDMA where the matrices are of the form [b0 c0 c1

a1 b1 c1 a2 b2 c2

an-2 bn-2 cn-2

cn-1 an-1 bn-1]

Parameters

- **a** (*numpy.ndarray*) – The `left' coefficients.
- **b** (*numpy.ndarray*) – The diagonal coefficients. (All ones?)
- **c** (*numpy.ndarray*) – The 'right' coefficients.
- **rhs** (*numpy.ndarray*) – The right-hand side vector.

Returns rhs – the rhs vector overwritten with derivatives.

Return type numpy.ndarray

2.4 Tools

General postprocessing tools go here

`Py4Incompact3D.tools.gradu.`**`calc_gradu`**(*postprocess*, *time=- 1*)

Computes the gradient of the velocity field, assumes ux uy uz have all been loaded.

Parameters

- **postprocess** (`Py4Incompact3D.postprocess.postprocess.Postprocess`) – The postprocessing object.
- **time** (*int or list of int*) – The time to compute vorticity at, -1 means all times.

`Py4Incompact3D.tools.gradu.`**`get_gradu_name`**(*i*, *j*)

Determine the name for a component of the velocity gradient tensor.

Parameters

- **i** (*int*) – The velocity component.
- **j** (*int*) – The gradient component.

Returns The name of the specified component of the velocity gradient tensor.

Return type str

`Py4Incompact3D.tools.gradu.`**`get_gradu_tensor`**(*postprocess*, *time=- 1*)

Construct the gradient tensor from the individual components.

Returns the gradient tensor in the form

$$\frac{\partial u^i}{\partial x^j}$$

where i is the first index and j the second index, i.e.

$$grad\left(\boldsymbol{u}\right)\left[1\right]\left[2\right] = \frac{\partial v}{\partial z}$$

Parameters

- **postprocess** (`Py4Incompact3D.postprocess.postprocess.Postprocess`) – The post processing object
- **time** (*int or list of int*) – The time(s) to get the gradient tensor for.

Returns :math`boldsymbol{nabla}boldsymbol{u}` a time-keyed dictionary of the gradient tensor.

Return type dict

`Py4Incompact3D.tools.vort.`**`calc_vort`**(*postprocess*, *time=- 1*)

Computes the vorticity of the velocity field, assumes ux uy and uz have all been loaded.

Parameters

- **`postprocess`** (`Py4Incompact3D.postprocess.postprocess.Postprocess`) – The postprocessing object.
- **`time`** (*`int or list of int`*) – The time to compute vorticity at, -1 means all times.

`Py4Incompact3D.tools.vort.`**`get_vort_name`**(*i*, *j*)

Get the name for the specified component of the vorticity tensor.

Parameters

- **`i`** (*`int`*) – The first component.
- **`j`** (*`int`*) – The second component.

Returns The name of the specified component of the vorticity tensor.

Return type str

`Py4Incompact3D.tools.vort.`**`get_vort_tensor`**(*postprocess*, *time=- 1*)

Construct the vorticity tensor from the individual components.

Returns the vorticity tensor in the form

$$\frac{1}{2}\left(\frac{\partial u^i}{\partial x^j} - \frac{\partial u^j}{\partial x^i}\right)$$

where i is the first index and j the second index, i.e.

$$\Omega\left[1\right]left[2] = \frac{1}{2}\left(\frac{\partial v}{\partial z} - \frac{\partial w}{\partial z}\right)$$

Parameters

- **`postprocess`** (`Py4Incompact3D.postprocess.postprocess.Postprocess`) – The post processing object.
- **`time`** (*`int or list of int`*) – The time(s) to get the vorticity tensor for.

Returns :math`boldsymbol{Omega}` a time-keyed dictionary of the vorticity tensor.

Return type dict

`Py4Incompact3D.tools.qcrit.`**`calc_qcrit`**(*postprocess*, *time=- 1*)

Computes the q-criterion of the velocit field, assumes ux uy uz vortx vorty vortz have all been loaded/computed.

Parameters

- **`postprocess`** (`Py4Incompact3D.postprocess.postprocess.Postprocess`) – The postprocessing object.
- **`time`** (*`int or list of int`*) – The time to compute vorticity at, -1 means all times.

`Py4Incompact3D.tools.lockexch.`**`calc_h`**(*postprocess*, *field='rho'*, *gamma=0.998*, *time=- 1*)

Calculates the "height" of the gravity-current, assumes name field (default ρ) is available.

This is based on the technique proposed in Birman2005 where the height of the gravity current is defined as:

$$h\left(x\right) = \frac{1}{L_y}\left(\frac{1}{1-\gamma}\int_0^{L_y}\overline{\rho}\left(x,y\right)dy - \frac{\gamma}{1-\gamma}\right)$$

where $\overline{\rho}$ is ρ averaged over the z axis.

Parameters

- **postprocess** (Py4Incompact3D.postprocess.postprocess.Postprocess) – The postprocessing object.
- **field** (*str*) – The name of the field to calculate the height by
- **gamma** (*float*) – The density ratio, defined as $\gamma = \frac{\rho_1}{\rho_2}, \ 0 \leq \gamma < 1$
- **time** (*int or list of int*) – The time(s) to compute h for, -1 means all times.

Returns h – a time-keyed dictionary of $h\left(x\right)$

Return type dict

Note: In the Boussinesq limit, the appropriate field is a concentration field $0 \leq c \leq 1$ for which, set $\gamma = 0$.

`Py4Incompact3D.tools.lockexch.`**`get_frontidx_birman`**(*h*)

Determines the array indidices for front locations according to Birman2005.

Parameters **h** (*dict*) – Time-keyed dictionary of gravity-current height.

Returns idxr, idxw, idxf: time-keyed dictionaries containing the indices of the front locations.

Return type dict, dict, dict

Note: In the case the front cannot be found, the index will have value None.

PYTHON MODULE INDEX

INDEX

Q

T

V

W