tasmania Documentation

Release 0.1.0

Stefano Ubbiali

May 16, 2018

CONTENTS:

CHAPTER

ONE

API DOCUMENTATION

1.1 Axis

class tasmania.grids.axis.**Axis**(*coords*, *dims*, *attrs=None*)

Class representing a one-dimensional axis. The class API is designed to be similar to that provided by `xarray.DataArray`.

Variables

- **coords** (*list*) – One-dimensional `numpy.ndarray` storing axis coordinates, wrapped within a list.
- **values** (*array_like*) – One-dimensional `numpy.ndarray` storing axis coordinates. This attrribute is semantically identical to `coords` and it is introduced only for the sake of compliancy with `xarray.DataArray`'s API.
- **dims** (*str*) – Axis dimension, i.e., label.
- **attrs** (*dict*) – Axis attributes, e.g., the units.

__getitem__(*i*)

Get direct access to the coordinate vector.

Parameters **i** (*int* or ‘array_like) – The index, or a sequence of indices.

Returns The coordinate(s).

Return type float

__init__(*coords*, *dims*, *attrs=None*)

Constructor.

Parameters

- **coords** (*array_like*) – One-dimensional `numpy.ndarray` representing the axis values.
- **dims** (*str*) – Axis label.
- **attrs** (*dict*, optional) – Axis attributes. This may be used to specify, e.g., the units, which, following the CF Conventions, may be either:
  - 'm' (meters) or multiples, for height-based coordinates;
  - 'Pa' (Pascal) or multiples, for pressure-based coordinates;
  - 'K' (Kelvin), for temperature-based coordinates;
  - 'degrees_east', for longitude;
  - 'degrees_north', for latitude.

_check_arguments(*coords*, *attr*)
Convert user-specified units to base units, e.g., km –> m, hPa –> Pa.

Parameters

- **coords** (*array_like*) – One-dimensional numpy.ndarray representing the axis values.
- **attrs** (*dict*) – Axis attributes. This may be used to specify, e.g., the units, which, following the CF Conventions, may be either:
  - 'm' (meters) or multiples, for height-based coordinates;
  - 'Pa' (Pascal) or multiples, for pressure-based coordinates;
  - 'K' (Kelvin), for temperature-based coordinates;
  - 'degrees_east', for longitude;
  - 'degrees_north', for latitude.

Returns The axis coordinates expressed in base units.

Return type array_like

1.2 Dynamics

1.2.1 Diagnostics

class tasmania.dycore.diagnostic_isentropic.**DiagnosticIsentropic**(*grid*, *moist_on*, *backend*)
Class implementing the diagnostic steps of the three-dimensional moist isentropic dynamical core using GT4Py stencils.

__init__(*grid*, *moist_on*, *backend*)
Constructor.

Parameters

- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – True for a moist dynamical core, False otherwise.
- **backend** (*obj*) – gridtools.mode specifying the backend for the GT4Py stencils.

_stencil_clipping_defs(*in_qv*, *in_qc*, *in_qr*)
GT4Py stencil clipping (i.e., setting to zero the negative values of) the mass fraction of each water constituent.

Parameters

- **in_qv** (*obj*) – gridtools.Equation representing the diagnosed mass fraction of water vapor.
- **in_qc** (*obj*) – gridtools.Equation representing the diagnosed mass fraction of cloud water.
- **in_qr** (*obj*) – gridtools.Equation representing the diagnosed mass fraction of precipitation water.

Returns

- **out_qv** (*obj*) – `gridtools.Equation` representing the clipped mass fraction of water vapor.
- **out_qc** (*obj*) – `gridtools.Equation` representing the clipped mass fraction of cloud water.
- **out_qr** (*obj*) – `gridtools.Equation` representing the clipped mass fraction of precipitation water.

_stencil_diagnosing_air_density_defs(*in_theta*, *in_s*, *in_h*)
GT4Py stencil diagnosing the density.

Parameters

- **in_theta** (*obj*) – `gridtools.Equation` representing the vertical half levels.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.
- **in_h** (*obj*) – `gridtools.Equation` representing the geometric height at the half-levels.

Returns `gridtools.Equation` representing the diagnosed density.

Return type obj

_stencil_diagnosing_air_density_initialize()
Initialize the GT4Py stencil in charge of diagnosing the density.

_stencil_diagnosing_air_density_set_inputs(*s*, *h*)
Update the private instance attributes which serve as inputs to the GT4Py stencil which diagnoses the density.

Parameters

- **s** (*array_like*) – `numpy.ndarray` with shape (`nx`, `ny`, `nz`) representing the isentropic density.
- **h** (*array_like*) – `numpy.ndarray` with shape (`nx`, `ny`, `nz+1`) representing the height of the half-levels.

_stencil_diagnosing_air_pressure_defs(*in_s*, *in_p*)
GT4Py stencil diagnosing the pressure.

Parameters

- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.
- **in_p** (*obj*) – `gridtools.Equation` representing the pressure.

Returns `gridtools.Equation` representing the diagnosed pressure.

Return type obj

_stencil_diagnosing_air_pressure_initialize()
Initialize the GT4Py stencil in charge of diagnosing the pressure.

_stencil_diagnosing_air_pressure_set_inputs(*s*)
Update the private instance attributes which serve as inputs to the GT4Py stencil which diagnoses the pressure.

Parameters **s** (*array_like*) – `numpy.ndarray` with shape (`nx`, `ny`, `nz`) representing the isentropic density.

_stencil_diagnosing_height_defs(*in_theta*, *in_exn*, *in_p*, *in_h*)
GT4Py stencil diagnosing the geometric height of the isentropes.

Parameters

- **in_theta** (*obj*) – gridtools.Equation representing the vertical half levels.
- **in_exn** (*obj*) – gridtools.Equation representing the Exner function.
- **in_p** (*obj*) – gridtools.Equation representing the pressure.
- **in_h** (*obj*) – gridtools.Equation representing the geometric height of the isentropes.

Returns gridtools.Equation representing the diagnosed geometric height of the isentropes.

Return type obj

_stencil_diagnosing_height_initialize()
Initialize the GT4Py stencil in charge of diagnosing the geometric height of the half-level isentropes.

_stencil_diagnosing_mass_fraction_of_water_constituents_in_air_defs(*in_s*, *in_Qv*, *in_Qc*, *in_Qr*)
GT4Py stencil diagnosing the mass fraction of each water constituent.

Parameters

- **in_s** (*obj*) – gridtools.Equation representing the isentropic density.
- **in_U** (*obj*) – gridtools.Equation representing the x-momentum.
- **in_V** (*obj*) – gridtools.Equation representing the y-momentum.
- **in_Qv** (*obj*) – gridtools.Equation representing the isentropic density of water vapor.
- **in_Qc** (*obj*) – gridtools.Equation representing the isentropic density of cloud water.
- **in_Qr** (*obj*) – gridtools.Equation representing the isentropic density of precipitation water.

Returns

- **out_qv** (*obj*) – gridtools.Equation representing the diagnosed mass fraction of water vapor.
- **out_qc** (*obj*) – gridtools.Equation representing the diagnosed mass fraction of cloud water.
- **out_qr** (*obj*) – gridtools.Equation representing the diagnosed mass fraction of precipitation water.

_stencil_diagnosing_mass_fraction_of_water_constituents_in_air_initialize()
Initialize the GT4Py stencil in charge of diagnosing the mass fraction of each water constituent.

_stencil_diagnosing_mass_fraction_of_water_constituents_in_air_set_inputs(*s*, *Qv*, *Qc*, *Qr*)
Update the private instance attributes which serve as inputs to the GT4Py stencil which diagnose the mass fraction of each water constituent.

Parameters

- **s** (*array_like*) – numpy.ndarray with shape (nx, ny, nz) representing the isentropic density.
- **Qv** (*obj*) – gridtools.Equation representing the isentropic density of water vapor.
- **Qc** (*obj*) – gridtools.Equation representing the isentropic density of cloud water.
- **Qr** (*obj*) – gridtools.Equation representing the isentropic density of precipitation water.

_stencil_diagnosing_montgomery_defs(*in_exn*, *in_mtg*)
GT4Py stencil diagnosing the Exner function.

Parameters

- **in_exn** (*obj*) – gridtools.Equation representing the Exner function.
- **in_mtg** (*obj*) – gridtools.Equation representing the Montgomery potential.

Returns gridtools.Equation representing the diagnosed Montgomery potential.

Return type obj

_stencil_diagnosing_montgomery_initialize()
Initialize the GT4Py stencil in charge of diagnosing the Montgomery potential.

_stencil_diagnosing_velocity_x_defs(*in_s*, *in_U*)
GT4Py stencil diagnosing the x-component of the velocity.

Parameters

- **in_s** (*obj*) – gridtools.Equation representing the isentropic density.
- **in_U** (*obj*) – gridtools.Equation representing the x-momentum.

Returns gridtools.Equation representing the diagnosed x-velocity.

Return type obj

_stencil_diagnosing_velocity_x_initialize()
Initialize the GT4Py stencil in charge of diagnosing the x-component of the velocity.

_stencil_diagnosing_velocity_y_defs(*in_s*, *in_V*)
GT4Py stencil diagnosing the y-component of the velocity.

Parameters

- **in_s** (*obj*) – gridtools.Equation representing the isentropic density.
- **in_V** (*obj*) – gridtools.Equation representing the y-momentum.

Returns gridtools.Equation representing the diagnosed y-velocity.

Return type obj

_stencil_diagnosing_velocity_y_initialize()
Initialize the GT4Py stencil in charge of diagnosing the y-component of the velocity.

_stencil_diagnosing_water_constituents_isentropic_density_defs(*in_s*, *in_qv*, *in_qc*, *in_qr*)
GT4Py stencil diagnosing the isentropic density of each water constituent, i.e., Q_v, Q_c and Q_r.

Parameters

- **in_s** (*obj*) – gridtools.Equation representing the isentropic density.

- **in_qv** (*obj*) – `gridtools.Equation` representing the mass fraction of water vapor.
- **in_qc** (*obj*) – `gridtools.Equation` representing the mass fraction of cloud water.
- **in_qr** (*obj*) – `gridtools.Equation` representing the mass fraction of precipitation water.

Returns

- **out_Qv** (*obj*) – `gridtools.Equation` representing the diagnosed Qv.
- **out_Qc** (*obj*) – `gridtools.Equation` representing the diagnosed Qc.
- **out_Qr** (*obj*) – `gridtools.Equation` representing the diagnosed Qr.

_stencil_diagnosing_water_constituents_isentropic_density_initialize()
Initialize the GT4Py stencil in charge of diagnosing the isentropic density of each water constituent.

_stencil_diagnosing_water_constituents_isentropic_density_set_inputs(*s*, *qv*, *qc*, *qr*)
Update the private instance attributes which serve as inputs to the GT4Py stencil which diagnoses the isentropic density of each water constituent.

Parameters

- **s** (*array_like*) – `numpy.ndarray` with shape (`nx`, `ny`, `nz`) representing the isentropic density.
- **qv** (*array_like*) – `numpy.ndarray` with shape (`nx`, `ny`, `nz`) representing the mass fraction of water vapor.
- **qc** (*array_like*) – `numpy.ndarray` with shape (`nx`, `ny`, `nz`) representing the mass fraction of cloud water.
- **qr** (*array_like*) – `numpy.ndarray` with shape (`nx`, `ny`, `nz`) representing the mass fraction of precipitation water.

_stencils_diagnosing_velocity_set_inputs(*s*, *U*, *V*)
Update the private instance attributes which serve as inputs to the GT4Py stencils which diagnose the velocity components.

Parameters

- **s** (*array_like*) – `numpy.ndarray` with shape (`nx`, `ny`, `nz`) representing the isentropic density.
- **U** (*array_like*) – `numpy.ndarray` with shape (`nx`, `ny`, `nz`) representing the x-velocity.
- **V** (*array_like*) – `numpy.ndarray` with shape (`nx`, `ny`, `nz`) representing the y-velocity.

diagnostic
Get the attribute implementing the diagnostic step of the three-dimensional moist isentropic dynamical core. If this is set to `None`, a `ValueError` is thrown.

Returns *DiagnosticIsentropic* carrying out the diagnostic step of the three-dimensional moist isentropic dynamical core.

Return type obj

get_air_density(*state*)
Diagnosis of the density.

Parameters **state** (*obj*) – *GridData* or one of its derived classes containing the following variables:

- air_isentropic_density (unstaggered);
- height or height_on_interface_levels (z-staggered).

Returns

GridData collecting the diagnosed variables, namely:

- air_density (unstaggered).

Return type obj

get_air_temperature(*state*)

Diagnosis of the temperature.

Parameters **state** (*obj*) – *GridData* or one of its derived classes containing the following variables:

- exner_function or exner_function_on_interface_levels (z-staggered).

Returns

GridData collecting the diagnosed variables, namely:

- air_temperature (unstaggered).

Return type obj

get_diagnostic_variables(*state*, *pt*)

Diagnosis of the pressure, the Exner function, the Montgomery potential, and the geometric height of the half-levels.

Parameters

- **state** (*obj*) – *GridData* or one of its derived classes containing the following variables:
  - air_isentropic_density (unstaggered).
- **pt** (*float*) – Pressure value at the top of the domain.

Returns

GridData collecting the diagnosed variables, namely:

- air_pressure_on_interface_levels (z-staggered);
- exner_function_on_interface_levels (z-staggered);
- montgomery_potential (unstaggered);
- height_on_interface_levels (z-staggered).

Return type obj

get_height(*state*, *pt*)

Diagnosis of the geometric height of the half-levels.

Parameters

- **state** (*obj*) – *GridData* or one of its derived classes containing the following variables:
  - air_isentropic_density (unstaggered).
- **pt** (*float*) – Pressure value at the top of the domain.

Returns

GridData collecting the diagnosed variables, namely:

- height_on_interface_levels (z-staggered).

Return type obj

get_mass_fraction_of_water_constituents_in_air(*state*)

Diagnosis of the mass fraction of each water constituents, i.e., q_v, q_c and q_r.

Parameters **state** (*obj*) – *GridData* or one of its derived classes containing the following variables:

- air_isentropic_density (unstaggered);
- water_vapor_isentropic_density (unstaggered);
- cloud_liquid_water_isentropic_density (unstaggered);
- precipitation_water_isentropic_density (unstaggered).

Returns

GridData collecting the diagnosed variables, namely:

- mass_fraction_of_water_vapor_in_air (unstaggered);
- mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
- mass_fraction_of_precipitation_water_in_air (unstaggered).

Return type obj

get_velocity_components(*state*, *state_old=None*)

Diagnosis of the velocity components u and v.

Parameters

- **state** (*obj*) – *GridData* or one of its derived classes containing the following variables:
  - air_isentropic_density (unstaggered);
  - x_momentum_isentropic (unstaggered);
  - y_momentum_isentropic (unstaggered).
- **state_old** (*obj*) – *GridData* or one of its derived classes containing the following variables, defined at the previous time level:
  - x_velocity (x-staggered);
  - y_velocity (y-staggered).

Returns

GridData collecting the diagnosed variables, namely:

- x_velocity (x-staggered);
- y_velocity (y-staggered).

Return type obj

Note: The first and last rows (respectively, columns) of the staggered x-velocity (resp., y-velocity) are set only if the state at the previous time level is provided.

---

get_water_constituents_isentropic_density(*state*)
Diagnosis of the isentropic density of each water constituent, i.e., Q_v, Q_c and Q_v.

Parameters **state** (*obj*) – *GridData* or one of its derived classes containing the following variables:

- air_isentropic_density (unstaggered);
- mass_fraction_of_water_vapor_in_air (unstaggered);
- mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
- mass_fraction_of_precipitation_water_in_air (unstaggered).

Returns

GridData collecting the diagnosed variables, namely:

- water_vapor_isentropic_density (unstaggered);
- cloud_liquid_water_isentropic_density (unstaggered);
- precipitation_water_isentropic_density (unstaggered).

Return type obj

1.2.2 Dynamical cores

class tasmania.dycore.dycore.**DynamicalCore**(*grid*, *moist_on*)
Abstract base class whose derived classes implement different dynamical cores. The class inherits sympl.TimeStepper.

__call__(*dt*, *state*, *tendencies=None*, *diagnostics=None*)
Call operator advancing the input state one step forward. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **dt** (*obj*) – datetime.timedelta object representing the time step.
- **state** (*obj*) – The current state, as an instance of *GridData* or one of its derived classes.
- **tendencies** (*obj*, optional) – *GridData* storing tendencies. Default is None.
- **diagnostics** (*obj*, optional) – *GridData* storing diagnostics. Default is None.

Returns

- **state_new** (*obj*) – The state at the next time level. This is of the same class of state.
- **diagnostics_out** (*obj*) – *GridData* storing output diagnostics.

__init__(*grid*, *moist_on*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **moist_on** (*bool*) – True for a moist dynamical core, False otherwise.

static factory(*model*, **args*, ***kwargs*)
Static method returning an instance of the derived class implementing the dynamical core specified by model.

Parameters

- **model** (*str*) – String specifying the dynamical core to implement. Either:
  - 'isentropic_conservative', for the isentropic dynamical core based on the conservative form of the governing equations;
  - 'isentropic_nonconservative', for the isentropic dynamical core based on the nonconservative form of the governing equations.
- ***args** – Positional arguments to forward to the derived class.
- ****kwargs** – Keyword arguments to forward to the derived class.

Returns Instance of the derived class implementing the specified model.

Return type obj

fast_tendency_parameterizations

Get the list of parameterizations calculating fast-varying tendencies. As this method is marked as abstract, its implementation is delegated to the derived classes.

Returns List containing instances of derived classes of *FastTendency* which are in charge of calculating fast-varying tendencies.

Return type list

get_initial_state(**args*)

Get the initial state. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters ***args** – The arguments depend on the specific dynamical core which the derived class implements.

Returns The initial state, as an instance of *GridData* or one of its derived classes.

Return type obj

microphysics

Get the attribute in charge of calculating the raindrop fall velocity. As this method is marked as abstract, its implementation is delegated to the derived classes.

Returns Instance of a derived class of either *SlowTendencyMicrophysics*, FastTendencyMicrophysics, or *AdjustmentMicrophysics* in charge of calculating the raindrop fall velocity.

Return type obj

time_levels

Get the number of time leves the dynamical core relies on. As this method is marked as abstract, its implementation is delegated to the derived classes.

Returns The number of time levels needed by the dynamical core.

Return type int

update_topography(*time*)

Update the underlying (time-dependent) topography.

Parameters **time** (*obj*) – datetime.timedelta representing the elapsed simulation time.

class tasmania.dycore.dycore_isentropic.**DynamicalCoreIsentropic**(*time_scheme*, *flux_scheme*, *horizontal_boundary_type*, *grid*, *moist_on*, *backend*, *damp_on=True*, *damp_type='rayleigh'*, *damp_depth=15*, *damp_max=0.0002*, *smooth_on=True*, *smooth_type='first_order'*, *smooth_damp_depth=10*, *smooth_coeff=0.03*, *smooth_coeff_max=0.24*, *smooth_moist_on=False*, *smooth_moist_type='first_order'*, *smooth_moist_damp_depth=10*, *smooth_moist_coeff=0.03*, *smooth_moist_coeff_max=0.24*, *physics_dynamics_coupling_on=False*, *sedimentation_on=False*, *sedimentation_flux_type='first_order_upwind'*, *sedimentation_substeps=2*)

This class inherits *DynamicalCore* to implement the three-dimensional (moist) isentropic dynamical core relying upon GT4Py stencils. The class offers different numerical schemes to carry out the prognostic step of the dynamical core, and supports different types of lateral boundary conditions. The conservative form of the governing equations is used.

__call__(*dt*, *state*, *tendencies=None*, *diagnostics=None*)

Call operator advancing the state variables one step forward.

Parameters

- **dt** (*obj*) – datetime.timedelta representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_velocity (y-staggered);
  - y_momentum_isentropic (unstaggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

- **tendencies** (*obj*, optional) – `GridData` storing tendencies, namely:
  - tendency_of_air_potential_temperature (unstaggered);
  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).

  Default is obj:*None*.
- **diagnostics** (*obj*, optional) – `GridData` storing diagnostics, namely:
  - accumulated_precipitation (unstaggered).

  Default is `None`.

Returns

- **state_new** (*obj*) – `StateIsentropic` representing the state at the next time level. It contains the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_velocity (y-staggered);
  - y_momentum_isentropic (unstaggered);
  - air_pressure_on_interface_levels (z-staggered);
  - exner_function_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
  - height_on_interface_levels (z-staggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - mass_fraction_of_precipitation_water_in_air (unstaggered);
  - air_density (unstaggered, only if cloud microphysics is switched on);
  - air_temperature (unstaggered, only if cloud microphysics is switched on).
- **diagnostics_out** (*obj*) – `GridData` storing output diagnostics, namely:
  - precipitation (unstaggered, only if rain sedimentation is switched on);
  - accumulated_precipitation (unstaggered, only if rain sedimentation is switched on);

__init__(*time_scheme*, *flux_scheme*, *horizontal_boundary_type*, *grid*, *moist_on*, *backend*, *damp_on=True*, *damp_type='rayleigh'*, *damp_depth=15*, *damp_max=0.0002*, *smooth_on=True*, *smooth_type='first_order'*, *smooth_damp_depth=10*, *smooth_coeff=0.03*, *smooth_coeff_max=0.24*, *smooth_moist_on=False*, *smooth_moist_type='first_order'*, *smooth_moist_damp_depth=10*, *smooth_moist_coeff=0.03*, *smooth_moist_coeff_max=0.24*, *physics_dynamics_coupling_on=False*, *sedimentation_on=False*, *sedimentation_flux_type='first_order_upwind'*, *sedimentation_substeps=2*)

Constructor.

Parameters

- **time_scheme** (*str*) – String specifying the time stepping method to implement. See *PrognosticIsentropic* for the available options.
- **flux_scheme** (*str*) – String specifying the numerical flux to use. See *FluxIsentropic* for the available options.
- **horizontal_boundary_type** (*str*) – String specifying the horizontal boundary conditions. See *HorizontalBoundary* for the available options.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils implementing the dynamical core.
- **damp_on** (*bool*, optional) – `True` if vertical damping is enabled, `False` otherwise. Default is `True`.
- **damp_type** (*str*, optional) – String specifying the type of vertical damping to apply. Default is 'rayleigh'. See *VerticalDamping* for the available options.
- **damp_depth** (*int*, optional) – Number of vertical layers in the damping region. Default is 15.
- **damp_max** (*float*, optional) – Maximum value for the damping coefficient. Default is 0.0002.
- **smooth_on** (*bool*, optional) – `True` if numerical smoothing is enabled, `False` otherwise. Default is `True`.
- **smooth_type** (*str*, optional) – String specifying the smoothing technique to implement. Default is 'first-order'. See *HorizontalSmoothing* for the available options.
- **smooth_damp_depth** (*int*, optional) – Number of vertical layers in the smoothing damping region. Default is 10.
- **smooth_coeff** (*float*, optional) – Smoothing coefficient. Default is 0.03.
- **smooth_coeff_max** (*float*, optional) – Maximum value for the smoothing coefficient. Default is 0.24. See *HorizontalSmoothing* for further details.
- **smooth_moist_on** (*bool*, optional) – `True` if numerical smoothing on water constituents is enabled, `False` otherwise. Default is `True`.
- **smooth_moist_type** (*str*, optional) – String specifying the smoothing technique to apply to the water constituents. Default is 'first-order'. See *HorizontalSmoothing* for the available options.
- **smooth_moist_damp_depth** (*int*, optional) – Number of vertical layers in the smoothing damping region for the water constituents. Default is 10.
- **smooth_moist_coeff** (*float*, optional) – Smoothing coefficient for the water constituents. Default is 0.03.
- **smooth_moist_coeff_max** (*float*, optional) – Maximum value for the smoothing coefficient for the water constituents. Default is 0.24. See *HorizontalSmoothing* for further details.
- **physics_dynamics_coupling_on** (*bool*, optional) – `True` to couple physics with dynamics, i.e., to account for the change over time in potential temperature, `False` otherwise. Default is `False`.

- **sedimentation_on** (*bool*, optional) – `True` to account for rain sedimentation, `False` otherwise. Default is `False`.
- **sedimentation_flux_type** (*str*) – String specifying the method used to compute the numerical sedimentation flux. Available options are:
  - 'first_order_upwind', for the first-order upwind scheme;
  - 'second_order_upwind', for the second-order upwind scheme.
- **sedimentation_substeps** (*int*) – Number of sub-timesteps to perform in order to integrate the sedimentation flux.

_step_dry(*dt*, *state*, *tendencies*, *diagnostics*)

Method advancing the dry isentropic state by a single time step.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_velocity (y-staggered);
  - y_momentum_isentropic (unstaggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
- **tendencies** (*obj*) – *GridData* storing tendencies. For the time being, this is not used.
- **diagnostics** (*obj*) – *GridData* storing diagnostics. For the time being, this is not used.

Returns

- **state_new** (*obj*) – *StateIsentropic* representing the state at the next time level. It contains the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_velocity (y-staggered);
  - y_momentum_isentropic (unstaggered);
  - air_pressure_on_interface_levels (z-staggered);
  - exner_function_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
  - height_on_interface_levels (z-staggered).
- **diagnostics_out** (*obj*) – Empty *GridData*, as no diagnostics are computed.

_step_moist(*dt*, *state*, *tendencies*, *diagnostics*)
Method advancing the moist isentropic state by a single time step.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state** (*obj*) – `StateIsentropic` representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_velocity (y-staggered);
  - y_momentum_isentropic (unstaggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **tendencies** (*obj*, optional) – `GridData` storing tendencies, namely:
  - tendency_of_air_potential_temperature (unstaggered);
  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).

  Default is obj:*None*.
- **diagnostics** (*obj*, optional) – `GridData` storing diagnostics, namely:
  - accumulated_precipitation (unstaggered).

  Default is `None`.

Returns

- **state_new** (*obj*) – `StateIsentropic` representing the state at the next time level. It contains the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_velocity (y-staggered);
  - y_momentum_isentropic (unstaggered);
  - air_pressure_on_interface_levels (z-staggered);
  - exner_function_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
  - height_on_interface_levels (z-staggered);

  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - mass_fraction_of_precipitation_water_in_air (unstaggered);
  - air_density (unstaggered, only if cloud microphysics is switched on);
  - air_temperature (unstaggered, only if cloud microphysics is switched on).
- **diagnostics_out** (*obj*) – `GridData` collecting output diagnostics, namely:
  - precipitation (unstaggered, only if rain sedimentation is switched on);
  - accumulated_precipitation (unstaggered, only if rain sedimentation is switched on).

fast_tendency_parameterizations
Get the list of parameterizations calculating fast-varying tendencies.

Returns List containing instances of derived classes of `FastTendency` which are in charge of calculating fast-varying tendencies.

Return type list

get_initial_state(*initial_time*, *initial_state_type*, ***kwargs*)
Get the initial state, based on the identifier `initial_state_type`. Particularly:

- if `initial_state_type == 0`:
  - $u(x, y, \theta, 0) = u_0$ and $v(x, y, \theta, 0) = v_0$;
  - the Exner function, the pressure, the Montgomery potential, the height of the isentropes, and the isentropic density are derived from the Brunt-Vaisala frequency N;
  - the mass fraction of water vapor is derived from the relative humidity, which is horizontally uniform and different from zero only in a band close to the surface;
  - the mass fraction of cloud water and precipitation water is zero;
- if `initial_state_type == 1`:
  - $u(x, y, \theta, 0) = u_0$ and $v(x, y, \theta, 0) = v_0$;
  - the Exner function, the pressure, the Montgomery potential, the height of the isentropes, and the isentropic density are derived from the Brunt-Vaisala frequency N;
  - the mass fraction of water vapor is derived from the relative humidity, which is sinusoidal in the x-direction and uniform in the y-direction, and different from zero only in a band close to the surface;
  - the mass fraction of cloud water and precipitation water is zero.
- if `initial_state_type == 2`:
  - $u(x, y, \theta, 0) = u_0$ and $v(x, y, \theta, 0) = v_0$;
  - $T(x, y, \theta, 0) = T_0$.

Parameters

- **initial_time** (*obj*) – `datetime.datetime` representing the initial simulation time.
- **case** (*int*) – Identifier.

Keyword Arguments

- **x_velocity_initial** (*float*) – The initial, uniform x-velocity u_0. Default is $10ms^{-1}$.
- **y_velocity_initial** (*float*) – The initial, uniform y-velocity v_0. Default is $0ms^{-1}$.
- **brunt_vaisala_initial** (*float*) – If `initial_state_type == 0`, the uniform Brunt-Vaisala frequence N. Default is 0.01.
- **temperature_initial** (*float*) – If `initial_state_type == 1`, the uniform initial temperature T_0. Default is $250K$.

Returns

StateIsentropic representing the initial state. It contains the following variables:

- air_density (unstaggered);
- air_isentropic_density (unstaggered);
- x_velocity (x-staggered);
- x_momentum_isentropic (unstaggered);
- y_velocity (y-staggered);
- y_momentum_isentropic (unstaggered);
- air_pressure_on_interface_levels (z-staggered);
- exner_function_on_interface_levels (z-staggered);
- montgomery_potential (unstaggered);
- height_on_interface_levels (z-staggered);
- air_temperature (unstaggered);
- mass_fraction_of_water_vapor_in_air (unstaggered, optional);
- mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
- mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

Return type obj

microphysics

Get the attribute in charge of calculating the raindrop fall velocity.

Returns Instance of a derived class of either *SlowTendencyMicrophysics*, `FastTendencyMicrophysics`, or *AdjustmentMicrophysics* in charge of calculating the raindrop fall velocity.

Return type obj

time_levels

Get the number of time leves the dynamical core relies on.

Returns The number of time levels needed by the dynamical core.

Return type int

class tasmania.dycore.dycore_isentropic_nonconservative.**DynamicalCoreIsentropicNonconservat**

This class inherits *DynamicalCore* to implement the three-dimensional (moist) isentropic dynamical core relying upon GT4Py stencils. The class offers different numerical schemes to carry out the prognostic step of the dynamical core, and supports different types of lateral boundary conditions. The nonconservative form of the governing equations is used.

__call__(*dt*, *state*, *diagnostics=None*, *tendencies=None*)

Call operator advancing the state variables one step forward.

Parameters

- **dt** (*obj*) – datetime.timedelta representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

- **diagnostics** (*obj*, optional) – `GridData` storing diagnostics, namely:
  - accumulated_precipitation (unstaggered).

  Default is `None`.
- **tendencies** (*obj*, optional) – `GridData` storing tendencies. Default is obj:*None*.

Returns

- **state_new** (*obj*) – `StateIsentropic` representing the state at the next time level. It contains the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - air_pressure_on_interface_levels (z-staggered);
  - exner_function_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
  - height_on_interface_levels (z-staggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - mass_fraction_of_precipitation_water_in_air (unstaggered);
  - air_density (unstaggered, only if cloud microphysics is switched on);
  - air_temperature (unstaggered, only if cloud microphysics is switched on).
- **diagnostics_out** (*obj*) – `GridData` storing output diagnostics, namely:
  - precipitation (unstaggered, only if rain sedimentation is switched on);
  - accumulated_precipitation (unstaggered, only if rain sedimentation is switched on);

__init__ (*time_scheme*, *flux_scheme*, *horizontal_boundary_type*, *grid*, *moist_on*, *backend*, *damp_on=True*, *damp_type='rayleigh'*, *damp_depth=15*, *damp_max=0.0002*, *smooth_on=True*, *smooth_type='first_order'*, *smooth_damp_depth=10*, *smooth_coeff=0.03*, *smooth_coeff_max=0.24*, *smooth_moist_on=False*, *smooth_moist_type='first_order'*, *smooth_moist_damp_depth=10*, *smooth_moist_coeff=0.03*, *smooth_moist_coeff_max=0.24*, *physics_dynamics_coupling_on=False*, *sedimentation_on=False*)

Constructor.

Parameters

- **time_scheme** (`str`) – String specifying the time stepping method to implement. See `PrognosticIsentropicNonconservative` for the available options.
- **flux_scheme** (`str`) – String specifying the numerical flux to use. See `FluxIsentropicNonconservative` for the available options.
- **horizontal_boundary_type** (`str`) – String specifying the horizontal boundary conditions. See `HorizontalBoundary` for the available options.
- **grid** (`obj`) – `GridXYZ` representing the underlying grid.
- **moist_on** (`bool`) – `True` for a moist dynamical core, `False` otherwise.

- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils implementing the dynamical core.
- **damp_on** (*bool*, optional) – `True` if vertical damping is enabled, `False` otherwise. Default is `True`.
- **damp_type** (*str*, optional) – String specifying the type of vertical damping to apply. Default is 'rayleigh'. See *VerticalDamping* for the available options.
- **damp_depth** (*int*, optional) – Number of vertical layers in the damping region. Default is 15.
- **damp_max** (*float*, optional) – Maximum value for the damping coefficient. Default is 0.0002.
- **smooth_on** (*bool*, optional) – `True` if numerical smoothing is enabled, `False` otherwise. Default is `True`.
- **smooth_type** (*str*, optional) – String specifying the smoothing technique to implement. Default is 'first-order'. See *HorizontalSmoothing* for the available options.
- **smooth_damp_depth** (*int*, optional) – Number of vertical layers in the smoothing damping region. Default is 10.
- **smooth_coeff** (*float*, optional) – Smoothing coefficient. Default is 0.03.
- **smooth_coeff_max** (*float*, optional) – Maximum value for the smoothing coefficient. Default is 0.24. See *HorizontalSmoothing* for further details.
- **smooth_moist_on** (*bool*, optional) – `True` if numerical smoothing on water constituents is enabled, `False` otherwise. Default is `True`.
- **smooth_moist_type** (*str*, optional) – String specifying the smoothing technique to apply to the water constituents. Default is 'first-order'. See *HorizontalSmoothing* for the available options.
- **smooth_moist_damp_depth** (*int*, optional) – Number of vertical layers in the smoothing damping region for the water constituents. Default is 10.
- **smooth_moist_coeff** (*float*, optional) – Smoothing coefficient for the water constituents. Default is 0.03.
- **smooth_moist_coeff_max** (*float*, optional) – Maximum value for the smoothing coefficient for the water constituents. Default is 0.24. See *HorizontalSmoothing* for further details.
- **physics_dynamics_coupling_on** (*bool*, optional) – `True` to couple physics with dynamics, i.e., to account for the change over time in potential temperature, `False` otherwise. Default is `False`.
- **sedimentation_on** (*bool*, optional) – `True` to account for rain sedimentation, `False` otherwise. Default is `False`.

_step_dry (*dt*, *state*, *diagnostics*, *tendencies*)

Method advancing the dry isentropic state by a single time step.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);

  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered).
- **diagnostics** (*obj*, optional) – `GridData` storing diagnostics. For the time being, this is not actually used.
- **tendencies** (*obj*, optional) – `GridData` storing tendencies. For the time being, this is not actually used.

Returns

- **state_new** (*obj*) – `StateIsentropic` representing the state at the next time level. It contains the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - air_pressure_on_interface_levels (z-staggered);
  - exner_function_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
  - height_on_interface_levels (z-staggered).
- **diagnostics_out** (*obj*) – Empty `GridData`, as no diagnostics are computed.

_step_moist (*dt*, *state*, *diagnostics*, *tendencies*)

Method advancing the moist isentropic state by a single time step.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state** (*obj*) – `StateIsentropic` representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - mass_fraction_of_precipitation_water_in_air (unstaggered).
- **diagnostics** (*obj*, optional) – `GridData` storing diagnostics, namely:
  - change_over_time_in_air_potential_temperature (unstaggered, required only if coupling between physics and dynamics is switched on);
  - accumulated_precipitation (unstaggered).

- **tendencies** (*obj*, optional) – `GridData` possibly storing tendencies. For the time being, this is not actually used.

Returns

- **state_new** (*obj*) – `StateIsentropic` representing the state at the next time level. It contains the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - air_pressure_on_interface_levels (z-staggered);
  - exner_function_on_interface_levels (z-staggered);
  - montgomery_potential (unstaggered);
  - height_on_interface_levels (z-staggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - mass_fraction_of_precipitation_water_in_air (unstaggered);
  - air_density (unstaggered, only if cloud microphysics is switched on);
  - air_temperature (unstaggered, only if cloud microphysics is switched on).
- **diagnostics_out** (*obj*) – `GridData` collecting output diagnostics, namely:
  - precipitation (unstaggered, only if rain sedimentation is switched on);
  - accumulated_precipitation (unstaggered, only if rain sedimentation is switched on).

get_initial_state(*initial_time*, *initial_state_type*, ***kwargs*)

Get the initial state, based on the identifier `initial_state_type`. Particularly:

- if `initial_state_type == 0`:
  - $u(x,\, y,\, \theta,\, 0) = u_0$ and $v(x,\, y,\, \theta,\, 0) = v_0$;
  - the Exner function, the pressure, the Montgomery potential, the height of the isentropes, and the isentropic density are derived from the Brunt-Vaisala frequency N;
  - the mass fraction of water vapor is derived from the relative humidity, which is horizontally uniform and different from zero only in a band close to the surface;
  - the mass fraction of cloud water and precipitation water is zero;
- if `initial_state_type == 1`:
  - $u(x,\, y,\, \theta,\, 0) = u_0$ and $v(x,\, y,\, \theta,\, 0) = v_0$;
  - the Exner function, the pressure, the Montgomery potential, the height of the isentropes, and the isentropic density are derived from the Brunt-Vaisala frequency N;
  - the mass fraction of water vapor is derived from the relative humidity, which is sinusoidal in the x-direction and uniform in the y-direction, and different from zero only in a band close to the surface;
  - the mass fraction of cloud water and precipitation water is zero.
- if `initial_state_type == 2`:
  - $u(x,\, y,\, \theta,\, 0) = u_0$ and $v(x,\, y,\, \theta,\, 0) = v_0$;

– $T(x, y, \theta, 0) = T_0$.

Parameters

- **initial_time** (*obj*) – `datetime.datetime` representing the initial simulation time.
- **case** (*int*) – Identifier.

Keyword Arguments

- **x_velocity_initial** (*float*) – The initial, uniform x-velocity u_0. Default is $10ms^{-1}$.
- **y_velocity_initial** (*float*) – The initial, uniform y-velocity v_0. Default is $0ms^{-1}$.
- **brunt_vaisala_initial** (*float*) – If `initial_state_type == 0`, the uniform Brunt-Vaisala frequence N. Default is 0.01.
- **temperature_initial** (*float*) – If `initial_state_type == 1`, the uniform initial temperature T_0. Default is $250K$.

Returns

StateIsentropic representing the initial state. It contains the following variables:

- air_density (unstaggered);
- air_isentropic_density (unstaggered);
- x_velocity (x-staggered);
- y_velocity (y-staggered);
- air_pressure_on_interface_levels (z-staggered);
- exner_function_on_interface_levels (z-staggered);
- montgomery_potential (unstaggered);
- height_on_interface_levels (z-staggered);
- air_temperature (unstaggered);
- mass_fraction_of_water_vapor_in_air (unstaggered, optional);
- mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
- mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

Return type obj

microphysics

Get the attribute taking care of the cloud microphysical dynamics.

Returns Instance of a derived class of either `TendencyMicrophysics` or *AdjustmentMicrophysics*.

Return type obj

time_levels

Get the number of time leves the dynamical core relies on.

Returns The number of time levels needed by the dynamical core.

Return type int

1.2.3 Lateral boundary conditions

class tasmania.dycore.horizontal_boundary.**HorizontalBoundary**(*grid*, *nb*)

Abstract base class whose derived classes implement different types of horizontal boundary conditions.

Variables

- **grid** (*obj*) – *GridXYZ* or one of its derived classes representing the underlying grid.
- ***nb*** (*int*) – Number of boundary layers.

__init__(*grid*, *nb*)

Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **nb** (*int*) – Number of boundary layers.

apply(*phi_new*, *phi_now*)

Apply the boundary conditions on the field phi_new, possibly relying upon the solution phi_now at the current time. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **phi_new** (*array_like*) – numpy.ndarray representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*) – numpy.ndarray representing the field at the current time.

static factory(*horizontal_boundary_type*, *grid*, *nb*)

Static method which returns an instance of the derived class which implements the boundary conditions specified by horizontal_boundary_type.

Parameters

- **horizontal_boundary_type** (*str*) – String specifying the type of boundary conditions to apply. Either:
  - 'periodic', for periodic boundary conditions;
  - 'relaxed', for relaxed boundary conditions;
  - 'relaxed_symmetric_xz', for relaxed boundary conditions for a xz-symmetric field.
  - 'relaxed_symmetric_yz', for relaxed boundary conditions for a yz-symmetric field.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **nb** (*int*) – Number of boundary layers.

Returns An instance of the derived class implementing the boundary conditions specified by horizontal_boundary_type.

Return type obj

from_computational_to_physical_domain(*phi_*, *out_dims*, *change_sign*)

Given a numpy.ndarray representing the computational domain of a stencil, return the associated physicalm field which may (or may not) satisfy the horizontal boundary conditions. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **phi** (*array_like*) – numpy.ndarray representing the computational domain of a stencil.
- **out_dims** (*tuple*) – Tuple of the output array dimensions.
- **change_sign** (*bool*) – True if the field should change sign through the symmetry plane (if any), False otherwise.

Returns numpy.ndarray representing the field defined over the physical domain.

Return type array_like

from_physical_to_computational_domain(*phi*)

Given a numpy.ndarray representing a physical field, return the associated stencils' computational domain, i.e., the numpy.ndarray (accomodating the boundary conditions) which will be input to the stencils. If the physical and computational fields coincide, a shallow copy of the physical domain is returned. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters **phi** (*array_like*) – numpy.ndarray representing the physical field.

Returns numpy.ndarray representing the stencils' computational domain.

Return type array_like

Note: The implementation should be designed to work with both staggered and unstaggared fields.

get_computational_grid()

Get the *computational* grid underlying the computational domain. As this method is marked as abstract, its implementation is delegated to the derived classes.

Returns Instance of the same class of tasmania.dycore.horizontal_boundary.HorizontalBoundary.grid representing the underlying computational grid.

Return type obj

set_outermost_layers_x(*phi_new*, *phi_now*)

Set the outermost layers of phi_new in the x-direction so to satisfy the lateral boundary conditions. For this, possibly rely upon the field phi_now at the current time. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **phi_new** (*array_like*) – numpy.ndarray representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*) – numpy.ndarray representing the field at the current time.

set_outermost_layers_y(*phi_new*, *phi_now*)

Set the outermost layers of phi_new in the y-direction so to satisfy the lateral boundary conditions. For this, possibly rely upon the field phi_now at the current time. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **phi_new** (*array_like*) – numpy.ndarray representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*) – numpy.ndarray representing the field at the current time.

class tasmania.dycore.horizontal_boundary_periodic.**Periodic**(*grid*, *nb*)

This class inherits *HorizontalBoundary* to implement horizontally periodic boundary conditions.

Variables

- **grid** (*obj*) – *GridXYZ* or one of its derived classes representing the underlying grid.
- ***nb*** (*int*) – Number of boundary layers.

__init__ (*grid*, *nb*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **nb** (*int*) – Number of boundary layers.

apply (*phi_new*, *phi_now=None*)
Apply horizontally periodic boundary conditions on phi_new.

Parameters

- **phi_new** (*array_like*) – numpy.ndarray representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*, optional) – numpy.ndarray representing the field at the current time.

Note: The argument phi_now is not required by the implementation, yet it is retained as optional argument for compliancy with the class hierarchy interface.

---

from_computational_to_physical_domain (*phi_*, *out_dims=None*, *change_sign=True*)
Shrink the field phi_ by removing the nb outermost layers.

Parameters

- **phi** (*array_like*) – The numpy.ndarray to shrink.
- **out_dims** (*tuple*) – Tuple of the output array dimensions.
- **change_sign** (*bool*) – True if the field should change sign through the symmetry plane (if any), False otherwise.

Returns The shrunk numpy.ndarray.

Return type array_like

Note: The arguments out_dims and change_sign are not required by the implementation, yet they are retained as optional arguments for compliancy with the class hierarchy interface.

---

from_physical_to_computational_domain (*phi*)
Periodically extend the field phi with nb extra layers.

Parameters **phi** (*array_like*) – The numpy.ndarray to extend.

Returns The extended numpy.ndarray.

Return type array_like

get_computational_grid ()
Get the *computational* grid underlying the computational domain.

Returns Instance of the same class of tasmania.dycore.horizontal_boundary.HorizontalBoundary.grid representing the underlying computational grid.

Return type obj

set_outermost_layers_x(*phi_new*, *phi_now=None*)
Set the outermost layers of `phi_new` in the x-direction so to satisfy the periodic boundary conditions.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*, optional) – `numpy.ndarray` representing the field at the current time.

Note: The argument `phi_now` is not required by the implementation, yet it is retained as optional argument for compliancy with the class hierarchy interface.

---

set_outermost_layers_y(*phi_new*, *phi_now=None*)
Set the outermost layers of `phi_new` in the y-direction so to satisfy the periodic boundary conditions.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*, optional) – `numpy.ndarray` representing the field at the current time.

Note: The argument `phi_now` is not required by the implementation, yet it is retained as optional argument for compliancy with the class hierarchy interface.

---

class tasmania.dycore.horizontal_boundary_periodic.**PeriodicXZ**(*grid*, *nb*)
This class inherits *HorizontalBoundary* to implement horizontally periodic boundary conditions for fields defined on a computational domain consisting of only one grid point in the y-direction.

Variables

- **grid** (*obj*) – *GridXYZ* or one of its derived classes representing the underlying grid.
- ***nb*** (*int*) – Number of boundary layers.

__init__(*grid*, *nb*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **nb** (*int*) – Number of boundary layers.

apply(*phi_new*, *phi_now=None*)
Apply horizontally periodic boundary conditions on `phi_new`.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*, optional) – `numpy.ndarray` representing the field at the current time.

Note: The argument `phi_now` is not required by the implementation, yet it is retained as optional argument for compliancy with the class hierarchy interface.

from_computational_to_physical_domain(*phi_*, *out_dims=None*, *change_sign=True*)
Return the central xz-slices of the input field `phi_`, removing the `nb` outermost layers in y-direction.

Parameters

- **phi** (*array_like*) – The `numpy.ndarray` to shrink.
- **out_dims** (*tuple*) – Tuple of the output array dimensions.
- **change_sign** (*bool*) – `True` if the field should change sign through the symmetry plane (if any), `False` otherwise.

Returns The shrunk `numpy.ndarray`.

Return type array_like

Note: The arguments `out_dims` and `change_sign` are not required by the implementation, yet they are retained as optional arguments for compliancy with the class hierarchy interface.

from_physical_to_computational_domain(*phi*)
Periodically extend the field `phi` with `nb` extra layers in the x-direction, then add `nb` ghost layers in the y-direction.

Parameters **phi** (*array_like*) – The `numpy.ndarray` to extend.

Returns The extended `numpy.ndarray`.

Return type array_like

get_computational_grid()
Get the *computational* grid underlying the computational domain.

Returns Instance of the same class of `tasmania.dycore.horizontal_boundary.HorizontalBoundary.grid` representing the underlying computational grid.

Return type obj

set_outermost_layers_x(*phi_new*, *phi_now=None*)
Set the outermost layers of `phi_new` in the x-direction so to satisfy the periodic boundary conditions.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*, optional) – `numpy.ndarray` representing the field at the current time.

Note: The argument `phi_now` is not required by the implementation, yet it is retained as optional argument for compliancy with the class hierarchy interface.

class tasmania.dycore.horizontal_boundary_periodic.**PeriodicYZ**(*grid*, *nb*)
This class inherits *HorizontalBoundary* to implement horizontally periodic boundary conditions for fields defined on a computational domain consisting of only one grid point in the x-direction.

Variables

- **grid** (*obj*) – *GridXYZ* or one of its derived classes representing the underlying grid.
- ***nb*** (*int*) – Number of boundary layers.

__init__ (*grid*, *nb*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **nb** (*int*) – Number of boundary layers.

apply (*phi_new*, *phi_now=None*)
Apply horizontally periodic boundary conditions on phi_new.

Parameters

- **phi_new** (*array_like*) – numpy.ndarray representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*, optional) – numpy.ndarray representing the field at the current time.

Note: The argument phi_now is not required by the implementation, yet it is retained as optional argument for compliancy with the class hierarchy interface.

---

from_computational_to_physical_domain (*phi_*, *out_dims=None*, *change_sign=True*)
Return the central yz-slices of the input field phi_, removing the nb outermost layers in x-direction.

Parameters

- **phi** (*array_like*) – The numpy.ndarray to shrink.
- **out_dims** (*tuple*) – Tuple of the output array dimensions.
- **change_sign** (*bool*) – True if the field should change sign through the symmetry plane (if any), False otherwise.

Returns The shrunk numpy.ndarray.

Return type array_like

Note: The arguments out_dims and change_sign are not required by the implementation, yet they are retained as optional arguments for compliancy with the class hierarchy interface.

---

from_physical_to_computational_domain (*phi*)
Periodically extend the field phi with nb extra layers in the y-direction, then add nb ghost layers in the x-direction.

Parameters **phi** (*array_like*) – The numpy.ndarray to extend.

Returns The extended numpy.ndarray.

Return type array_like

get_computational_grid ()
Get the *computational* grid underlying the computational domain.

Returns Instance of the same class of tasmania.dycore.horizontal_boundary.HorizontalBoundary.grid representing the underlying computational grid.

Return type obj

set_outermost_layers_y(*phi_new*, *phi_now=None*)
Set the outermost layers of `phi_new` in the y-direction so to satisfy the periodic boundary conditions.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*, optional) – `numpy.ndarray` representing the field at the current time.

Note: The argument `phi_now` is not required by the implementation, yet it is retained as optional argument for compliancy with the class hierarchy interface.

class tasmania.dycore.horizontal_boundary_relaxed.**Relaxed**(*grid*, *nb*)
This class inherits *HorizontalBoundary* to implement horizontally relaxed boundary conditions.

Variables

- **grid** (*obj*) – *GridXYZ* or one of its derived classes representing the underlying grid.
- ***nb*** (*int*) – Number of boundary layers.
- **nr** (*int*) – Number of layers which will be affected by relaxation.

__init__(*grid*, *nb*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **nb** (*int*) – Number of boundary layers.

apply(*phi_new*, *phi_now*)
Apply relaxed lateral boundary conditions.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*) – `numpy.ndarray` representing the field at the current time.

Note: The Dirichlet conditions at the boundaries are assumed to be time-independent, so that they can be inferred from the solution at current time.

from_computational_to_physical_domain(*phi_*, *out_dims=None*, *change_sign=True*)
As no extension is required to apply relaxed boundary conditions, return a shallow copy of the input field `phi_`.

Parameters

- **phi** (*array_like*) – A `numpy.ndarray`.
- **out_dims** (*tuple*, optional) – Tuple of the output array dimensions.
- **change_sign** (*bool*, optional) – `True` if the field should change sign through the symmetry plane, `False` otherwise.

Returns A shallow copy of `phi_`.

Return type array_like

Note: The arguments `out_dims` and `change_sign` are not required by the implementation, yet they are retained as optional arguments for compliancy with the class hierarchy interface.

---

from_physical_to_computational_domain(*phi*)

As no extension is required to apply relaxed boundary conditions, return a shallow copy of the input field `phi`.

Parameters **phi** (*array_like*) – A `numpy.ndarray`.

Returns A shallow copy of `phi`.

Return type array_like

get_computational_grid()

Get the *computational* grid underlying the computational domain.

Returns Instance of the same class of `tasmania.dycore.horizontal_boundary.HorizontalBoundary.grid` representing the underlying computational grid.

Return type obj

set_outermost_layers_x(*phi_new*, *phi_now*)

Set the outermost layers of `phi_new` in the x-direction equal to the corresponding layers of `phi_now`. In other words, apply Dirichlet conditions in x-direction.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*) – `numpy.ndarray` representing the field at the current time.

Note: The Dirichlet conditions at the boundaries are assumed to be time-independent, so that they can be inferred from the solution at current time.

---

set_outermost_layers_y(*phi_new*, *phi_now*)

Set the outermost layers of `phi_new` in the y-direction equal to the corresponding layers of `phi_now`. In other words, apply Dirichelt conditions in y-direction.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*) – `numpy.ndarray` representing the field at the current time.

Note: The Dirichlet conditions at the boundaries are assumed to be time-independent, so that they can be inferred from the solution at current time.

---

class tasmania.dycore.horizontal_boundary_relaxed.**RelaxedXZ**(*grid*, *nb*)

This class inherits *HorizontalBoundary* to implement horizontally relaxed boundary conditions for fields defined on a computational domain consisting of only one grid point in the y-direction.

Variables

- **grid** (*obj*) – *GridXYZ* or one of its derived classes representing the underlying grid.
- ***nb*** (*int*) – Number of boundary layers.
- **nr** (*int*) – Number of layers which will be affected by relaxation.

__init__ (*grid*, *nb*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **nb** (*int*) – Number of boundary layers.

apply (*phi_new*, *phi_now*)
Apply relaxed lateral boundary conditions along the x-axis.

Parameters

- **phi_new** (*array_like*) – numpy.ndarray representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*) – numpy.ndarray representing the field at the current time.

Note: The Dirichlet conditions at the boundaries are assumed to be time-independent, so that they can be inferred from the solution at current time.

---

from_computational_to_physical_domain (*phi_*, *out_dims=None*, *change_sign=True*)
Remove the nb outermost xz-slices from the input field phi_.

Parameters

- **phi** (*array_like*) – A numpy.ndarray.
- **out_dims** (*tuple*, optional) – Tuple of the output array dimensions.
- **change_sign** (*bool*, optional) – True if the field should change sign through the symmetry plane, False otherwise.

Returns The central xz-slice(s) of phi_.

Return type array_like

Note: The arguments out_dims and change_sign are not required by the implementation, yet they are retained as optional arguments for compliancy with the class hierarchy interface.

---

from_physical_to_computational_domain (*phi*)
While no extension is required to apply relaxed boundary conditions along the x-direction, nb ghost layers are appended in the y-direction.

Parameters **phi** (*array_like*) – A numpy.ndarray.

Returns A deep copy of phi, with nb ghost layers in the y-direction.

Return type array_like

get_computational_grid()
Get the *computational* grid underlying the computational domain.

Returns Instance of the same class of `tasmania.dycore.horizontal_boundary.HorizontalBoundary.grid` representing the underlying computational grid.

Return type obj

set_outermost_layers_x(*phi_new*, *phi_now*)
Set the outermost layers of `phi_new` in the x-direction equal to the corresponding layers of `phi_now`. In other words, apply Dirichlet conditions in x-direction.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*) – `numpy.ndarray` representing the field at the current time.

Note: The Dirichlet conditions at the boundaries are assumed to be time-independent, so that they can be inferred from the solution at current time.

---

class tasmania.dycore.horizontal_boundary_relaxed.**RelaxedYZ**(*grid*, *nb*)
This class inherits *HorizontalBoundary* to implement horizontally relaxed boundary conditions for fields defined on a computational domain consisting of only one grid point in the x-direction.

Variables

- **grid** (*obj*) – *GridXYZ* or one of its derived classes representing the underlying grid.
- ***nb*** (*int*) – Number of boundary layers.
- **nr** (*int*) – Number of layers which will be affected by relaxation.

__init__(*grid*, *nb*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **nb** (*int*) – Number of boundary layers.

apply(*phi_new*, *phi_now*)
Apply relaxed lateral boundary conditions along the x-axis.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*) – `numpy.ndarray` representing the field at the current time.

Note: The Dirichlet conditions at the boundaries are assumed to be time-independent, so that they can be inferred from the solution at current time.

---

from_computational_to_physical_domain(*phi_*, *out_dims=None*, *change_sign=True*)
Return a deep copy of the central yz-slices of the input field `phi_`.

Parameters

- **phi** (*array_like*) – A `numpy.ndarray`.
- **out_dims** (*tuple*, optional) – Tuple of the output array dimensions.

- **change_sign** (*bool*, optional) – `True` if the field should change sign through the symmetry plane, `False` otherwise.

Returns The central yz-slice(s) of `phi_`.

Return type array_like

Note: The arguments `out_dims` and `change_sign` are not required by the implementation, yet they are retained as optional arguments for compliancy with the class hierarchy interface.

---

from_physical_to_computational_domain(*phi*)

While no extension is required to apply relaxed boundary conditions along the y-direction, `nb` ghost layers are appended in the x-direction.

Parameters **phi** (*array_like*) – A `numpy.ndarray`.

Returns A deep copy of `phi`, with `nb` ghost layers along the y-layers.

Return type array_like

get_computational_grid()

Get the *computational* grid underlying the computational domain.

Returns Instance of the same class of `tasmania.dycore.horizontal_boundary.HorizontalBoundary.grid` representing the underlying computational grid.

Return type obj

set_outermost_layers_y(*phi_new*, *phi_now*)

Set the outermost layers of `phi_new` in the y-direction equal to the corresponding layers of `phi_now`. In other words, apply Dirichlet conditions in y-direction.

Parameters

- **phi_new** (*array_like*) – `numpy.ndarray` representing the field on which applying the boundary conditions.
- **phi_now** (*array_like*) – `numpy.ndarray` representing the field at the current time.

Note: The Dirichlet conditions at the boundaries are assumed to be time-independent, so that they can be inferred from the solution at current time.

---

class tasmania.dycore.horizontal_boundary_relaxed.**RelaxedSymmetricXZ**(*grid*, *nb*)

This class inherits *Relaxed* to implement horizontally relaxed boundary conditions for fields symmetric with respect to the xz-plane $y = y_c = 0.5(a_y + b_y)$, where a_y and b_y denote the extremes of the domain in the y-direction.

Variables

- **grid** (*obj*) – *GridXYZ* or one of its derived classes representing the underlying grid.
- ***nb*** (*int*) – Number of boundary layers.
- **nr** (*int*) – Number of layers which will be affected by relaxation.

__init__(*grid*, *nb*)

Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **nb** (*int*) – Number of boundary layers.

from_computational_to_physical_domain(*phi_*, *out_dims*, *change_sign=False*)
Mirror the computational domain with respect to the xz-plane $y = y_c$.

Parameters

- **phi** (*array_like*) – `numpy.ndarray` representing the computational domain of a stencil.
- **out_dims** (*tuple*) – Tuple of the output array dimensions.
- **change_sign** (*bool*, optional) – `True` if the field should change sign through the symmetry plane, `False` otherwise. Default is false.

Returns `numpy.ndarray` representing the field defined over the physical domain.

Return type array_like

from_physical_to_computational_domain(*phi*)
Return the y-lowermost half of the domain. To accomodate symmetric conditions, we retain (at least) `nb` additional layers in the positive direction of the y-axis.

Parameters **phi** (*array_like*) – `numpy.ndarray` representing the physical field.

Returns `numpy.ndarray` representing the stencils' computational domain.

Return type array_like

get_computational_grid()
Get the *computational* grid underlying the computational domain.

Returns Instance of the same class of `tasmania.dycore.horizontal_boundary.HorizontalBoundary.grid` representing the underlying computational grid.

Return type obj

class tasmania.dycore.horizontal_boundary_relaxed.**RelaxedSymmetricYZ**(*grid*, *nb*)
This class inherits *Relaxed* to implement horizontally relaxed boundary conditions for fields symmetric with respect to the yz-plane $x = x_c = 0.5(a_x + b_x)$, where a_x and b_x denote the extremes of the domain in the x-direction.

Variables

- **grid** (*obj*) – *GridXYZ* or one of its derived classes representing the underlying grid.
- ***nb*** (*int*) – Number of boundary layers.
- **nr** (*int*) – Number of layers which will be affected by relaxation.

__init__(*grid*, *nb*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **nb** (*int*) – Number of boundary layers.

from_computational_to_physical_domain(*phi_*, *out_dims*, *change_sign=False*)
Mirror the computational domain with respect to the yz-axis $x = x_c$.

Parameters

- **phi** (*array_like*) – numpy.ndarray representing the computational domain of a stencil.
- **out_dims** (*tuple*) – Tuple of the output array dimensions.
- **change_sign** (*bool*, optional) – True if the field should change sign through the symmetry plane, False otherwise. Default is false.

Returns numpy.ndarray representing the field defined over the physical domain.

Return type array_like

from_physical_to_computational_domain(*phi*)
Return the x-lowermost half of the domain. To accomodate symmetric conditions, we retain (at least) nb additional layers in the positive direction of the x-axis.

Parameters **phi** (*array_like*) – numpy.ndarray representing the physical field.

Returns numpy.ndarray representing the stencils' computational domain.

Return type array_like

get_computational_grid()
Get the *computational* grid underlying the computational domain.

Returns Instance of the same class of tasmania.dycore.horizontal_boundary.HorizontalBoundary.grid representing the underlying computational grid.

Return type obj

1.2.4 Horizontal smoothing

class tasmania.dycore.horizontal_smoothing.**HorizontalSmoothing**(*dims*, *grid*, *smooth_damp_depth*, *smooth_coeff*, *smooth_coeff_max*, *backend*)
Abstract base class whose derived classes apply horizontal numerical smoothing to a generic (prognostic) field by means of a GT4Py stencil.

__init__(*dims*, *grid*, *smooth_damp_depth*, *smooth_coeff*, *smooth_coeff_max*, *backend*)
Constructor.

Parameters

- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply numerical smoothing.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **smooth_damp_depth** (*int*) – Depth of the damping region, i.e., number of vertical layers in the damping region.
- **smooth_coeff** (*float*) – Value for the smoothing coefficient far from the top boundary.
- **smooth_coeff_max** (*float*) – Maximum value for the smoothing coefficient.
- **backend** (*obj*) – gridtools.mode specifying the backend for the GT4Py stencil implementing numerical smoothing.

apply(*phi*)
Apply horizontal smoothing to a prognostic field. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters **phi** (*array_like*) – numpy.ndarray representing the field to filter.

Returns numpy.ndarray representing the filtered field.

Return type array_like

static factory(*smooth_type*, *dims*, *grid*, *smooth_damp_depth*, *smooth_coeff*, *smooth_coeff_max*, *backend*)
Static method returning an instance of the derived class implementing the smoothing technique specified by smooth_type.

Parameters

- **smooth_type** (*string*) – String specifying the smoothing technique to implement. Either:
  - 'first_order', for first-order numerical smoothing;
  - 'second_order', for second-order numerical smoothing.
- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply numerical smoothing.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **smooth_damp_depth** (*int*) – Depth of the damping region, i.e., number of vertical layers in the damping region.
- **smooth_coeff** (*float*) – Value for the smoothing coefficient far from the top boundary.
- **smooth_coeff_max** (*float*) – Maximum value for the smoothing coefficient.
- **backend** (*obj*) – gridtools.mode specifying the backend for the GT4Py stencil implementing numerical smoothing. Default is gridtools.mode.NUMPY.

Returns Instance of the suitable derived class.

Return type obj

class tasmania.dycore.horizontal_smoothing.**HorizontalSmoothingFirstOrderXYZ**(*dims*, *grid*, *smooth_damp_depth=*, *smooth_coeff=0.03*, *smooth_coeff_max=0*, *backend=<Mode.NUMPY: 4>*)
This class inherits *HorizontalSmoothing* to apply first-order numerical smoothing to three-dimensional fields with at least three elements in each direction.

Note: An instance of this class should only be applied to fields whose dimensions match those specified at instantiation time. Hence, one should use (at least) one instance per field shape.

__init__(*dims*, *grid*, *smooth_damp_depth=10*, *smooth_coeff=0.03*, *smooth_coeff_max=0.24*, *backend=<Mode.NUMPY: 4>*)
Constructor.

Parameters

- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply numerical smoothing.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **smooth_damp_depth** (*int*, optional) – Depth of the damping region, i.e., number of vertical layers in the damping region. Default is 10.
- **smooth_coeff** (*float*, optional) – Value for the smoothing coefficient far from the top boundary. Default is 0.03.
- **smooth_coeff_max** (*float*, optional) – Maximum value for the smoothing coefficient. For the sake of numerical stability, it should not exceed 0.25. Default is 0.24.
- **backend** (*obj*, optional) – `gridtools.mode` specifying the backend for the GT4Py stencil implementing numerical smoothing. Default is `gridtools.mode.NUMPY`.

Note: To instantiate the class, please prefer the static method *factory()* of *HorizontalSmoothing*.

_stencil_defs(*in_phi*, *gamma*)

The GT4Py stencil applying first-order horizontal smoothing. A centered 5-points formula is used.

Parameters

- **in_phi** (*obj*) – `gridtools.Equation` representing the input field to filter.
- **gamma** (*obj*) – `gridtools.Equation` representing the smoothing coefficient.

Returns `gridtools.Equation` representing the filtered output field.

Return type obj

_stencil_initialize(*phi*)

Initialize the GT4Py stencil applying horizontal smoothing.

Parameters **phi** (*array_like*) – `numpy.ndarray` representing the field to filter.

apply(*phi*)

Apply first-order horizontal smoothing to a prognostic field.

Parameters **phi** (*array_like*) – `numpy.ndarray` representing the field to filter.

Returns `numpy.ndarray` representing the filtered field.

Return type array_like

class tasmania.dycore.horizontal_smoothing.**HorizontalSmoothingFirstOrderXZ**(*dims*, *grid*, *smooth_damp_depth=*, *smooth_coeff=0.03*, *smooth_coeff_max=0.4*, *backend=<Mode.NUMPY: 4>*)

This class inherits *HorizontalSmoothing* to apply first-order numerical smoothing to three-dimensional fields with only one element in the y-direction.

Note: An instance of this class should only be applied to fields whose dimensions match those specified at instantiation time. Hence, one should use (at least) one instance per field shape.

__init__(*dims*, *grid*, *smooth_damp_depth=10*, *smooth_coeff=0.03*, *smooth_coeff_max=0.49*, *backend=<Mode.NUMPY: 4>*)

Constructor.

Parameters

- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply numerical smoothing.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **smooth_damp_depth** (*int*, optional) – Depth of the damping region, i.e., number of vertical layers in the damping region. Default is 10.
- **smooth_coeff** (*float*, optional) – Value for the smoothing coefficient far from the top boundary. Default is 0.03.
- **smooth_coeff_max** (*float*, optional) – Maximum value for the smoothing coefficient. For the sake of numerical stability, it should not exceed 0.5. Default is 0.49.
- **backend** (*obj*, optional) – `gridtools.mode` specifying the backend for the GT4Py stencil implementing numerical smoothing. Default is `gridtools.mode.NUMPY`.

Note: To instantiate the class, please prefer the static method *factory()* of *HorizontalSmoothing*.

_stencil_defs(*in_phi*, *gamma*)

The GT4Py stencil applying horizontal smoothing. A standard centered 3-points formula is used.

Parameters

- **in_phi** (*obj*) – `gridtools.Equation` representing the input field to filter.
- **gamma** (*obj*) – `gridtools.Equation` representing the smoothing coefficient.

Returns `gridtools.Equation` representing the filtered output field.

Return type obj

_stencil_initialize(*phi*)

Initialize the GT4Py stencil applying horizontal smoothing.

Parameters **phi** (*array_like*) – `numpy.ndarray` representing the field to filter.

apply(*phi*)

Apply first-order horizontal smoothing to a prognostic field.

Parameters **phi** (*array_like*) – `numpy.ndarray` representing the field to filter.

Returns `numpy.ndarray` representing the filtered field.

Return type array_like

class tasmania.dycore.horizontal_smoothing.**HorizontalSmoothingFirstOrderYZ**(*dims*, *grid*, *smooth_damp_depth=1*, *smooth_coeff=0.03*, *smooth_coeff_max=0.4*, *backend=<Mode.NUMPY: 4>*)

This class inherits *HorizontalSmoothing* to apply first-order numerical smoothing to three-dimensional fields with only one element in the x-direction.

Note: An instance of this class should only be applied to fields whose dimensions match those specified at instantiation time. Hence, one should use (at least) one instance per field shape.

__init__(*dims*, *grid*, *smooth_damp_depth=10*, *smooth_coeff=0.03*, *smooth_coeff_max=0.49*, *backend=<Mode.NUMPY: 4>*)

Constructor.

Parameters

- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply numerical smoothing.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **smooth_damp_depth** (*int*) – Depth of the damping region, i.e., number of vertical layers in the damping region. Default is 10.
- **smooth_coeff** (*float*) – Value for the smoothing coefficient far from the top boundary. Default is 0.03.
- **smooth_coeff_max** (*float*) – Maximum value for the smoothing coefficient. For the sake of numerical stability, it should not exceed 0.5. Default is 0.49.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencil implementing numerical smoothing. Default is `gridtools.mode.NUMPY`.

Note: To instantiate the class, please prefer the static method *factory()* of *HorizontalSmoothing*.

_stencil_defs(*in_phi*, *gamma*)

The GT4Py stencil applying horizontal smoothing. A standard centered 3-points formula is used.

Parameters

- **in_phi** (*obj*) – `gridtools.Equation` representing the input field to filter.
- **gamma** (*obj*) – `gridtools.Equation` representing the smoothing coefficient.

Returns `gridtools.Equation` representing the filtered output field.

Return type obj

_stencil_initialize(*phi*)

Initialize the GT4Py stencil applying horizontal smoothing.

Parameters **phi** (*array_like*) – `numpy.ndarray` representing the field to filter.

apply(*phi*)
Apply first-order horizontal smoothing to a prognostic field.

Parameters **phi** (*array_like*) – numpy.ndarray representing the field to filter.

Returns numpy.ndarray representing the filtered field.

Return type array_like

class tasmania.dycore.horizontal_smoothing.**HorizontalSmoothingSecondOrderXYZ**(*dims*, *grid*, *smooth_damp_depth*, *smooth_coeff=0.03*, *smooth_coeff_max=*, *backend=<Mode.NUMPY: 4>*)

This class inherits *HorizontalSmoothing* to apply second-order numerical smoothing to three-dimensional fields.

Note: An instance of this class should only be applied to fields whose dimensions match those specified at instantiation time. Hence, one should use (at least) one instance per field shape.

__init__(*dims*, *grid*, *smooth_damp_depth=10*, *smooth_coeff=0.03*, *smooth_coeff_max=0.24*, *backend=<Mode.NUMPY: 4>*)
Constructor.

Parameters

- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply numerical smoothing.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **smooth_damp_depth** (*int*, optional) – Depth of the damping region, i.e., number of vertical layers in the damping region. Default is 10.
- **smooth_coeff** (*float*, optional) – Value for the smoothing coefficient far from the top boundary. Default is 0.03.
- **smooth_coeff_max** (*float*, optional) – Maximum value for the smoothing coefficient. For the sake of numerical stability, it should not exceed 0.25. Default is 0.24.
- **backend** (*obj*, optional) – gridtools.mode specifying the backend for the GT4Py stencil implementing numerical smoothing. Default is gridtools.mode.NUMPY.

Note: To instantiate the class, please prefer the static method *factory()* of *HorizontalSmoothing*.

_stencil_defs(*in_phi*, *gamma*)
The GT4Py stencil applying horizontal smoothing. A standard centered 5-points formula is used.

Parameters

- **in_phi** (*obj*) – gridtools.Equation representing the input field to filter.
- **gamma** (*obj*) – gridtools.Equation representing the smoothing coefficient.

Returns gridtools.Equation representing the filtered output field.

Return type obj

_stencil_initialize(*phi*)
Initialize the GT4Py stencil applying horizontal smoothing.

Parameters **phi** (*array_like*) – numpy.ndarray representing the field to filter.

apply(*phi*)
Apply second-order horizontal smoothing to a prognostic field.

Parameters **phi** (*array_like*) – numpy.ndarray representing the field to filter.

Returns numpy.ndarray representing the filtered field.

Return type array_like

class tasmania.dycore.horizontal_smoothing.**HorizontalSmoothingSecondOrderXZ**(*dims*, *grid*, *smooth_damp_depth=*, *smooth_coeff=0.03*, *smooth_coeff_max=0*, *backend=<Mode.NUMPY 4>*)

This class inherits *HorizontalSmoothing* to apply second-order numerical smoothing to three-dimensional fields with only one element in the y-direction.

Note: An instance of this class should only be applied to fields whose dimensions match those specified at instantiation time. Hence, one should use (at least) one instance per field shape.

__init__(*dims*, *grid*, *smooth_damp_depth=10*, *smooth_coeff=0.03*, *smooth_coeff_max=0.49*, *backend=<Mode.NUMPY: 4>*)
Constructor.

Parameters

- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply numerical smoothing.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **smooth_damp_depth** (*int*, optional) – Depth of the damping region, i.e., number of vertical layers in the damping region. Default is 10.
- **smooth_coeff** (*float*, optional) – Value for the smoothing coefficient far from the top boundary. Default is 0.03.
- **smooth_coeff_max** (*float*, optional) – Maximum value for the smoothing coefficient. For the sake of numerical stability, it should not exceed 0.5. Default is 0.49.
- **backend** (*obj*, optional) – gridtools.mode specifying the backend for the GT4Py stencil implementing numerical smoothing. Default is gridtools.mode.NUMPY.

Note: To instantiate the class, please prefer the static method *factory()* of *HorizontalSmoothing*.

_stencil_defs(*in_phi*, *gamma*)
The GT4Py stencil applying horizontal smoothing. A standard centered 5-points formula is used.

Parameters

- **in_phi** (*obj*) – gridtools.Equation representing the input field to filter.
- **gamma** (*obj*) – gridtools.Equation representing the smoothing coefficient.

Returns gridtools.Equation representing the filtered output field.

Return type obj

_stencil_initialize(*phi*)
Initialize the GT4Py stencil applying horizontal smoothing.

Parameters **phi** (*array_like*) – numpy.ndarray representing the field to filter.

apply(*phi*)
Apply second-order horizontal smoothing to a prognostic field.

Parameters **phi** (*array_like*) – numpy.ndarray representing the field to filter.

Returns numpy.ndarray representing the filtered field.

Return type array_like

class tasmania.dycore.horizontal_smoothing.**HorizontalSmoothingSecondOrderYZ**(*dims*, *grid*, *smooth_damp_depth=*, *smooth_coeff=0.03*, *smooth_coeff_max=0*, *backend=<Mode.NUMPY: 4>*)

This class inherits *HorizontalSmoothing* to apply second-order numerical smoothing to three-dimensional fields with only one element in the x-direction.

Note: An instance of this class should only be applied to fields whose dimensions match those specified at instantiation time. Hence, one should use (at least) one instance per field shape.

__init__(*dims*, *grid*, *smooth_damp_depth=10*, *smooth_coeff=0.03*, *smooth_coeff_max=0.49*, *backend=<Mode.NUMPY: 4>*)
Constructor.

Parameters

- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply numerical smoothing.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **smooth_damp_depth** (*int*, optional) – Depth of the damping region, i.e., number of vertical layers in the damping region. Default is 10.
- **smooth_coeff** (*float*, optional) – Value for the smoothing coefficient far from the top boundary. Default is 0.03.
- **smooth_coeff_max** (*float*, optional) – Maximum value for the smoothing coefficient. For the sake of numerical stability, it should not exceed 0.5. Default is 0.49.
- **backend** (*obj*, optional) – gridtools.mode specifying the backend for the GT4Py stencil implementing numerical smoothing. Default is gridtools.mode.NUMPY.

Note: To instantiate the class, please prefer the static method *factory()* of *HorizontalSmoothing*.

_stencil_defs(*in_phi*, *gamma*)
The GT4Py stencil applying horizontal smoothing. A standard centered 5-points formula is used.

Parameters

- **in_phi** (*obj*) – gridtools.Equation representing the input field to filter.
- **gamma** (*obj*) – gridtools.Equation representing the smoothing coefficient.

Returns gridtools.Equation representing the filtered output field.

Return type obj

_stencil_initialize(*phi*)
Initialize the GT4Py stencil applying horizontal smoothing.

Parameters **phi** (*array_like*) – numpy.ndarray representing the field to filter.

apply(*phi*)
Apply second-order horizontal smoothing to a prognostic field.

Parameters **phi** (*array_like*) – numpy.ndarray representing the field to filter.

Returns numpy.ndarray representing the filtered field.

Return type array_like

1.2.5 Numerical fluxes

class tasmania.dycore.flux_isentropic.**FluxIsentropic**(*grid*, *moist_on*)
Abstract base class whose derived classes implement different schemes to compute the numerical fluxes for the three-dimensional isentropic dynamical core. The conservative form of the governing equations is used.

__init__(*grid*, *moist_on*)
Constructor.

Parameters

- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – True for a moist dynamical core, False otherwise.

_compute_horizontal_fluxes(*i*, *j*, *k*, *dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_U*, *in_V*, *in_Qv*, *in_Qc*, *in_Qr*, *in_qv_tnd=None*, *in_qc_tnd=None*, *in_qr_tnd=None*)
Method computing the gridtools.Equations representing the x- and y-fluxes for all the conservative prognostic variables. The gridtools.Equations are then set as instance attributes. As this method is marked as abstract, the implementation is delegated to the derived classes.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **dt** (*obj*) – gridtools.Global representing the time step.
- **in_s** (*obj*) – gridtools.Equation representing the isentropic density.

- **in_u** (*obj*) – gridtools.Equation representing the x-velocity.
- **in_v** (*obj*) – gridtools.Equation representing the y-velocity.
- **in_mtg** (*obj*) – gridtools.Equation representing the Montgomery potential.
- **in_U** (*obj*) – gridtools.Equation representing the x-momentum.
- **in_V** (*obj*) – gridtools.Equation representing the y-momentum.
- **in_Qv** (*obj*) – gridtools.Equation representing the isentropic density of water vapor.
- **in_Qc** (*obj*) – gridtools.Equation representing the isentropic density of cloud liquid water.
- **in_Qr** (*obj*) – gridtools.Equation representing the isentropic density of precipitation water.
- **in_qv_tnd** (*obj*, optional) – gridtools.Equation representing the tendency of the mass fraction of water vapor.
- **in_qc_tnd** (*obj*, optional) – gridtools.Equation representing the tendency of the mass fraction of cloud liquid water.
- **in_qr_tnd** (*obj*, optional) – gridtools.Equation representing the tendency of the mass fraction of precipitation water.

_compute_vertical_fluxes(*i*, *j*, *k*, *dt*, *in_w*, *in_s*, *in_s_prv*, *in_U*, *in_U_prv*, *in_V*, *in_V_prv*, *in_Qv*, *in_Qv_prv*, *in_Qc*, *in_Qc_prv*, *in_Qr*, *in_Qr_prv*)

Method computing the gridtools.Equations representing the θ-flux for all the conservative model variables. The gridtools.Equations are then set as instance attributes. As this method is marked as abstract, the implementation is delegated to the derived classes.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **dt** (*obj*) – gridtools.Global representing the time step.
- **in_w** (*obj*) – gridtools.Equation representing the vertical velocity, i.e., the change over time in potential temperature.
- **in_s** (*obj*) – gridtools.Equation representing the current isentropic density.
- **in_s_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density, i.e., the isentropic density stepped disregarding the vertical advection.
- **in_U** (*obj*) – gridtools.Equation representing the current x-momentum.
- **in_U_prv** (*obj*) – gridtools.Equation representing the provisional x-momentum, i.e., the x-momentum stepped disregarding the vertical advection.
- **in_V** (*obj*) – gridtools.Equation representing the current y-momentum.
- **in_V_prv** (*obj*) – gridtools.Equation representing the provisional y-momentum, i.e., the y-momentum stepped disregarding the vertical advection.
- **in_Qv** (*obj*) – gridtools.Equation representing the current isentropic density of water vapor.

- **in_Qv_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density of water vapor, i.e., the isentropic density of water vapor stepped disregarding the vertical advection.
- **in_Qc** (*obj*) – `gridtools.Equation` representing the current isentropic density of cloud water.
- **in_Qc_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density of cloud water, i.e., the isentropic density of cloud water stepped disregarding the vertical advection.
- **in_Qr** (*obj*) – `gridtools.Equation` representing the current isentropic density of precipitation water.
- **in_Qr_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density of precipitation water, i.e., the isentropic density of precipitation water stepped disregarding the vertical advection.

static factory(*scheme*, *grid*, *moist_on*)

Static method which returns an instance of the derived class implementing the numerical scheme specified by `scheme`.

Parameters

- **scheme** (*str*) – String specifying the numerical scheme to implement. Either:
  - 'upwind', for the upwind scheme;
  - 'centered', for a second-order centered scheme;
  - 'maccormack', for the MacCormack scheme.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.

Returns Instance of the derived class implementing the scheme specified by `scheme`.

Return type obj

get_horizontal_fluxes(*i*, *j*, *k*, *dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_U*, *in_V*, *in_Qv=None*, *in_Qc=None*, *in_Qr=None*, *in_qv_tnd=None*, *in_qc_tnd=None*, *in_qr_tnd=None*)

Method returning the `gridtools.Equations` representing the x- and y-fluxes for all the conservative model variables.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity.
- **in_mtg** (*obj*) – `gridtools.Equation` representing the Montgomery potential.
- **in_U** (*obj*) – `gridtools.Equation` representing the x-momentum.
- **in_V** (*obj*) – `gridtools.Equation` representing the y-momentum.

- **in_Qv** (*obj*, optional) – `gridtools.Equation` representing the isentropic density of water vapor.
- **in_Qc** (*obj*, optional) – `gridtools.Equation` representing the isentropic density of cloud water.
- **in_Qr** (*obj*, optional) – `gridtools.Equation` representing the isentropic density of precipitation water.
- **in_qv_tnd** (*obj*, optional) – `gridtools.Equation` representing the tendency of the mass fraction of water vapor.
- **in_qc_tnd** (*obj*, optional) – `gridtools.Equation` representing the tendency of the mass fraction of cloud liquid water.
- **in_qr_tnd** (*obj*, optional) – `gridtools.Equation` representing the tendency of the mass fraction of precipitation water.

Returns

- **flux_s_x** (*obj*) – `gridtools.Equation` representing the x-flux for the isentropic density.
- **flux_s_y** (*obj*) – `gridtools.Equation` representing the y-flux for the isentropic density.
- **flux_U_x** (*obj*) – `gridtools.Equation` representing the x-flux for the x-momentum.
- **flux_U_y** (*obj*) – `gridtools.Equation` representing the y-flux for the x-momentum.
- **flux_V_x** (*obj*) – `gridtools.Equation` representing the x-flux for the y-momentum.
- **flux_V_y** (*obj*) – `gridtools.Equation` representing the y-flux for the y-momentum.
- **flux_Qv_x** (*obj*, optional) – `gridtools.Equation` representing the x-flux for the isentropic density of water vapor.
- **flux_Qv_y** (*obj*, optional) – `gridtools.Equation` representing the y-flux for the isentropic density of water vapor.
- **flux_Qc_x** (*obj*, optional) – `gridtools.Equation` representing the x-flux for the isentropic density of cloud liquid water.
- **flux_Qc_y** (*obj*, optional) – `gridtools.Equation` representing the y-flux for the isentropic density of cloud liquid water.
- **flux_Qr_x** (*obj*, optional) – `gridtools.Equation` representing the x-flux for the isentropic density of precipitation water.
- **flux_Qr_y** (*obj*, optional) – `gridtools.Equation` representing the y-flux for the isentropic density of precipitation water.

get_vertical_fluxes(*i*, *j*, *k*, *dt*, *in_w*, *in_s*, *in_s_prv*, *in_U*, *in_U_prv*, *in_V*, *in_V_prv*, *in_Qv=None*, *in_Qv_prv=None*, *in_Qc=None*, *in_Qc_prv=None*, *in_Qr=None*, *in_Qr_prv=None*)

Method returning the `gridtools.Equations` representing the θ-fluxes for all the conservative model variables.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.

- **dt** (*obj*) – gridtools.Global representing the time step.
- **in_w** (*obj*) – gridtools.Equation representing the vertical velocity, i.e., the change over time in potential temperature.
- **in_s** (*obj*) – gridtools.Equation representing the current isentropic density.
- **in_s_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density, i.e., the isentropic density stepped disregarding the vertical advection.
- **in_U** (*obj*) – gridtools.Equation representing the current x-momentum.
- **in_U_prv** (*obj*) – gridtools.Equation representing the provisional x-momentum, i.e., the x-momentum stepped disregarding the vertical advection.
- **in_V** (*obj*) – gridtools.Equation representing the current y-momentum.
- **in_V_prv** (*obj*) – gridtools.Equation representing the provisional y-momentum, i.e., the y-momentum stepped disregarding the vertical advection.
- **in_Qv** (*obj*, optional) – gridtools.Equation representing the current isentropic density of water vapor.
- **in_Qv_prv** (*obj*, optional) – gridtools.Equation representing the provisional isentropic density of water vapor, i.e., the isentropic density of water vapor stepped disregarding the vertical advection.
- **in_Qc** (*obj*, optional) – gridtools.Equation representing the current isentropic density of cloud water.
- **in_Qc_prv** (*obj*, optional) – gridtools.Equation representing the provisional isentropic density of cloud water, i.e., the isentropic density of cloud water stepped disregarding the vertical advection.
- **in_Qr** (*obj*, optional) – gridtools.Equation representing the current isentropic density of precipitation water.
- **in_Qr_prv** (*obj*, optional) – gridtools.Equation representing the provisional isentropic density of precipitation water, i.e., the isentropic density of precipitation water stepped disregarding the vertical advection.

Returns

- **flux_s_z** (*obj*) – gridtools.Equation representing the θ-flux for the isentropic density.
- **flux_U_z** (*obj*) – gridtools.Equation representing the θ-flux for the x-momentum.
- **flux_V_z** (*obj*) – gridtools.Equation representing the θ-flux for the y-momentum.
- **flux_Qv_z** (*obj*, optional) – gridtools.Equation representing the θ-flux for the isentropic density of water vapor.
- **flux_Qc_z** (*obj*, optional) – gridtools.Equation representing the θ-flux for the isentropic density of cloud water.
- **flux_Qr_z** (*obj*, optional) – gridtools.Equation representing the θ-flux for the isentropic density of precipitation water.

class tasmania.dycore.flux_isentropic_upwind.**FluxIsentropicUpwind**(*grid*, *moist_on*)

Class which inherits *FluxIsentropic* to implement the upwind scheme to compute the numerical fluxes for the governing equations expressed in conservative form using isentropic coordinates.

Variables

- **nb** (*int*) – Number of boundary layers.
- **order** (*int*) – Order of accuracy.

__init__(*grid*, *moist_on*)
Constructor.

Parameters

- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.

_compute_horizontal_fluxes(*i*, *j*, *k*, *dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_U*, *in_V*, *in_Qv*, *in_Qc*, *in_Qr*, *in_qv_tnd=None*, *in_qc_tnd=None*, *in_qr_tnd=None*)
Method computing the `gridtools.Equation`s representing the upwind x- and y-fluxes for all the conservative prognostic variables. The `gridtools.Equation`s are then set as instance attributes.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity.
- **in_mtg** (*obj*) – `gridtools.Equation` representing the Montgomery potential.
- **in_U** (*obj*) – `gridtools.Equation` representing the x-momentum.
- **in_V** (*obj*) – `gridtools.Equation` representing the y-momentum.
- **in_Qv** (*obj*) – `gridtools.Equation` representing the isentropic density of water vapor.
- **in_Qc** (*obj*) – `gridtools.Equation` representing the isentropic density of cloud water.
- **in_Qr** (*obj*) – `gridtools.Equation` representing the isentropic density of precipitation water.
- **in_qv_tnd** (*obj*, optional) – `gridtools.Equation` representing the tendency of the mass fraction of water vapor.
- **in_qc_tnd** (*obj*, optional) – `gridtools.Equation` representing the tendency of the mass fraction of cloud liquid water.
- **in_qr_tnd** (*obj*, optional) – `gridtools.Equation` representing the tendency of the mass fraction of precipitation water.

Note: `in_qv_tnd`, `in_qc_tnd`, and `in_qr_tnd` are not actually used, yet they appear as default arguments for compliancy with the class hierarchy interface.

_compute_vertical_fluxes(*i*, *j*, *k*, *dt*, *in_w*, *in_s*, *in_s_prv*, *in_U*, *in_U_prv*, *in_V*, *in_V_prv*, *in_Qv*, *in_Qv_prv*, *in_Qc*, *in_Qc_prv*, *in_Qr*, *in_Qr_prv*)
Method computing the `gridtools.Equation`s representing the upwind θ-flux for all the conservative model variables. The `gridtools.Equation`s are then set as instance attributes.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_w** (*obj*) – `gridtools.Equation` representing the vertical velocity, i.e., the change over time of potential temperature.
- **in_s** (*obj*) – `gridtools.Equation` representing the current isentropic density.
- **in_s_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density, i.e., the isentropic density stepped disregarding the vertical advection.
- **in_U** (*obj*) – `gridtools.Equation` representing the current x-momentum.
- **in_U_prv** (*obj*) – `gridtools.Equation` representing the provisional x-momentum, i.e., the x-momentum stepped disregarding the vertical advection.
- **in_V** (*obj*) – `gridtools.Equation` representing the current y-momentum.
- **in_V_prv** (*obj*) – `gridtools.Equation` representing the provisional y-momentum, i.e., the y-momentum stepped disregarding the vertical advection.
- **in_Qv** (*obj*) – `gridtools.Equation` representing the current isentropic density of water vapor.
- **in_Qv_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density of water vapor, i.e., the isentropic density of water vapor stepped disregarding the vertical advection.
- **in_Qc** (*obj*) – `gridtools.Equation` representing the current isentropic density of cloud water.
- **in_Qc_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density of cloud water, i.e., the isentropic density of cloud water stepped disregarding the vertical advection.
- **in_Qr** (*obj*) – `gridtools.Equation` representing the current isentropic density of precipitation water.
- **in_Qr_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density of precipitation water, i.e., the isentropic density of precipitation water stepped disregarding the vertical advection.

_get_upwind_flux_x(*i*, *j*, *k*, *in_u*, *in_phi*)

Get the `gridtools.Equation` representing the upwind flux in x-direction for a generic prognostic variable ϕ.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity.
- **in_phi** (*obj*) – `gridtools.Equation` representing the field ϕ.

Returns `gridtools.Equation` representing the upwind flux in x-direction for ϕ.

Return type obj

_get_upwind_flux_y(*i*, *j*, *k*, *in_v*, *in_phi*)
Get the `gridtools.Equation` representing the upwind flux in y-direction for a generic prognostic variable ϕ.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity.
- **in_phi** (*obj*) – `gridtools.Equation` representing the field ϕ.

Returns `gridtools.Equation` representing the upwind flux in y-direction for ϕ.

Return type obj

_get_upwind_flux_z(*i*, *j*, *k*, *tmp_w_mid*, *in_phi*)
Get the `gridtools.Equation` representing the upwind flux in θ-direction for a generic prognostic variable ϕ.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **tmp_w_mid** (*obj*) – `gridtools.Equation` representing the vertical velocity, i.e., the change over time in potential temperature, at the model half levels.
- **in_phi** (*obj*) – `gridtools.Equation` representing the field ϕ.

Returns `gridtools.Equation` representing the upwind flux in θ-direction for ϕ.

Return type obj

class tasmania.dycore.flux_isentropic_centered.**FluxIsentropicCentered**(*grid*, *moist_on*)
Class which inherits `FluxIsentropicNonconservative` to implement a centered scheme to compute the numerical fluxes for the prognostic model variables. The conservative form of the governing equations, expressed using isentropic coordinates, is used.

Variables

- ***nb*** (*int*) – Number of boundary layers.
- **order** (*int*) – Order of accuracy.

__init__(*grid*, *moist_on*)
Constructor.

Parameters

- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.

_compute_horizontal_fluxes(*i*, *j*, *k*, *dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_U*, *in_V*, *in_Qv*, *in_Qc*, *in_Qr*, *in_qv_tnd=None*, *in_qc_tnd=None*, *in_qr_tnd=None*)
Method computing the `gridtools.Equation`s representing the centered x- and y-fluxes for all the conservative prognostic variables. The `gridtools.Equation`s are then set as instance attributes.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity.
- **in_mtg** (*obj*) – `gridtools.Equation` representing the Montgomery potential.
- **in_U** (*obj*) – `gridtools.Equation` representing the x-momentum.
- **in_V** (*obj*) – `gridtools.Equation` representing the y-momentum.
- **in_Qv** (*obj*) – `gridtools.Equation` representing the isentropic density of water vapor.
- **in_Qc** (*obj*) – `gridtools.Equation` representing the isentropic density of cloud liquid water.
- **in_Qr** (*obj*) – `gridtools.Equation` representing the isentropic density of precipitation water.
- **in_qv_tnd** (*obj*, optional) – `gridtools.Equation` representing the tendency of the mass fraction of water vapor.
- **in_qc_tnd** (*obj*, optional) – `gridtools.Equation` representing the tendency of the mass fraction of cloud liquid water.
- **in_qr_tnd** (*obj*, optional) – `gridtools.Equation` representing the tendency of the mass fraction of precipitation water.

Note: `in_qv_tnd`, `in_qc_tnd`, and `in_qr_tnd` are not actually used, yet they appear as default arguments for compliancy with the class hierarchy interface.

_compute_vertical_fluxes(*i*, *j*, *k*, *dt*, *in_w*, *in_s*, *in_s_prv*, *in_U*, *in_U_prv*, *in_V*, *in_V_prv*, *in_Qv*, *in_Qv_prv*, *in_Qc*, *in_Qc_prv*, *in_Qr*, *in_Qr_prv*)

Method computing the `gridtools.Equation`s representing the centered θ-flux for all the conservative model variables. The `gridtools.Equation`s are then set as instance attributes.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_w** (*obj*) – `gridtools.Equation` representing the vertical velocity, i.e., the change over time of potential temperature.
- **in_s** (*obj*) – `gridtools.Equation` representing the current isentropic density.
- **in_s_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density, i.e., the isentropic density stepped disregarding the vertical advection.

- **in_U** (*obj*) – gridtools.Equation representing the current x-momentum.
- **in_U_prv** (*obj*) – gridtools.Equation representing the provisional x-momentum, i.e., the x-momentum stepped disregarding the vertical advection.
- **in_V** (*obj*) – gridtools.Equation representing the current y-momentum.
- **in_V_prv** (*obj*) – gridtools.Equation representing the provisional y-momentum, i.e., the y-momentum stepped disregarding the vertical advection.
- **in_Qv** (*obj*) – gridtools.Equation representing the current isentropic density of water vapor.
- **in_Qv_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density of water vapor, i.e., the isentropic density of water vapor stepped disregarding the vertical advection.
- **in_Qc** (*obj*) – gridtools.Equation representing the current isentropic density of cloud water.
- **in_Qc_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density of cloud water, i.e., the isentropic density of cloud water stepped disregarding the vertical advection.
- **in_Qr** (*obj*) – gridtools.Equation representing the current isentropic density of precipitation water.
- **in_Qr_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density of precipitation water, i.e., the isentropic density of precipitation water stepped disregarding the vertical advection.

_get_centered_flux_x(*i*, *j*, *k*, *in_u*, *in_phi*)

Get the gridtools.Equation representing the centered flux in x-direction for a generic prognostic variable ϕ.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **in_u** (*obj*) – gridtools.Equation representing the x-velocity.
- **in_phi** (*obj*) – gridtools.Equation representing the field ϕ.

Returns gridtools.Equation representing the centered flux in x-direction for ϕ.

Return type obj

_get_centered_flux_y(*i*, *j*, *k*, *v*, *in_phi*)

Get the gridtools.Equation representing the centered flux in y-direction for a generic prognostic variable ϕ.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **v** (*obj*) – gridtools.Equation representing the y-velocity.
- **in_phi** (*obj*) – gridtools.Equation representing the field ϕ.

Returns gridtools.Equation representing the centered flux in y-direction for ϕ.

Return type obj

_get_centered_flux_z(*i*, *j*, *k*, *tmp_w_mid*, *in_phi*)
Get the gridtools.Equation representing the centered flux in θ-direction for a generic prognostic variable ϕ.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **tmp_w_mid** (*obj*) – gridtools.Equation representing the vertical velocity, i.e., the change over time in potential temperature, at the model half levels.
- **in_phi** (*obj*) – gridtools.Equation representing the field ϕ.

Returns gridtools.Equation representing the centered flux in θ-direction for ϕ.

Return type obj

class tasmania.dycore.flux_isentropic_maccormack.**FluxIsentropicMacCormack**(*grid*, *moist_on*)
Class which inherits *FluxIsentropic* to implement the MacCormack scheme to compute the numerical fluxes for the governing equations expressed in conservative form using isentropic coordinates.

Variables

- ***nb*** (*int*) – Number of boundary layers.
- **order** (*int*) – Order of accuracy.

__init__(*grid*, *moist_on*)
Constructor.

Parameters

- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – True for a moist dynamical core, False otherwise.

_compute_horizontal_fluxes(*i*, *j*, *k*, *dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_U*, *in_V*, *in_Qv*, *in_Qc*, *in_Qr*, *in_qv_tnd=None*, *in_qc_tnd=None*, *in_qr_tnd=None*)
Method computing the gridtools.Equations representing the MacCormack x- and y-fluxes for all the conservative prognostic variables. The gridtools.Equations are then set as instance attributes.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **dt** (*obj*) – gridtools.Global representing the time step.
- **in_s** (*obj*) – gridtools.Equation representing the isentropic density.
- **in_u** (*obj*) – gridtools.Equation representing the x-velocity.
- **in_v** (*obj*) – gridtools.Equation representing the y-velocity.
- **in_mtg** (*obj*) – gridtools.Equation representing the Montgomery potential.
- **in_U** (*obj*) – gridtools.Equation representing the x-momentum.

- **in_V** (*obj*) – gridtools.Equation representing the y-momentum.
- **in_Qv** (*obj*) – gridtools.Equation representing the isentropic density of water vapour.
- **in_Qc** (*obj*) – gridtools.Equation representing the isentropic density of cloud water.
- **in_Qr** (*obj*) – gridtools.Equation representing the isentropic density of precipitation water.
- **in_qv_tnd** (*obj*, optional) – gridtools.Equation representing the tendency of the mass fraction of water vapor.
- **in_qc_tnd** (*obj*, optional) – gridtools.Equation representing the tendency of the mass fraction of cloud liquid water.
- **in_qr_tnd** (*obj*, optional) – gridtools.Equation representing the tendency of the mass fraction of precipitation water.

_compute_vertical_fluxes(*i*, *j*, *k*, *dt*, *in_w*, *in_s*, *in_s_prv*, *in_U*, *in_U_prv*, *in_V*, *in_V_prv*, *in_Qv*, *in_Qv_prv*, *in_Qc*, *in_Qc_prv*, *in_Qr*, *in_Qr_prv*)

Method computing the gridtools.Equations representing the MacCormack θ-flux for all the conservative model variables. The gridtools.Equations are then set as instance attributes.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **dt** (*obj*) – gridtools.Global representing the time step.
- **in_w** (*obj*) – gridtools.Equation representing the vertical velocity, i.e., the change over time of potential temperature.
- **in_s** (*obj*) – gridtools.Equation representing the current isentropic density.
- **in_s_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density, i.e., the isentropic density stepped disregarding the vertical advection.
- **in_U** (*obj*) – gridtools.Equation representing the current x-momentum.
- **in_U_prv** (*obj*) – gridtools.Equation representing the provisional x-momentum, i.e., the x-momentum stepped disregarding the vertical advection.
- **in_V** (*obj*) – gridtools.Equation representing the current y-momentum.
- **in_V_prv** (*obj*) – gridtools.Equation representing the provisional y-momentum, i.e., the y-momentum stepped disregarding the vertical advection.
- **in_Qv** (*obj*) – gridtools.Equation representing the current isentropic density of water vapor.
- **in_Qv_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density of water vapor, i.e., the isentropic density of water vapor stepped disregarding the vertical advection.
- **in_Qc** (*obj*) – gridtools.Equation representing the current isentropic density of cloud water.
- **in_Qc_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density of cloud water, i.e., the isentropic density of cloud water stepped disregarding the vertical advection.

- **in_Qr** (*obj*) – gridtools.Equation representing the current isentropic density of precipitation water.
- **in_Qr_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density of precipitation water, i.e., the isentropic density of precipitation water stepped disregarding the vertical advection.

_get_maccormack_flux_x(*i*, *j*, *k*, *tmp_u_unstg*, *in_phi*, *tmp_u_prd_unstg*, *tmp_phi_prd*)

Get the gridtools.Equation representing the MacCormack flux in x-direction for a generic prognostic variable ϕ.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **tmp_u_unstg** (*obj*) – gridtools.Equation representing the unstaggered x-velocity at the current time.
- **in_phi** (*obj*) – gridtools.Equation representing the field ϕ at the current time.
- **tmp_u_prd_unstg** (*obj*) – gridtools.Equation representing the predicted value for the unstaggered x-velocity.
- **tmp_phi_prd** (*obj*) – gridtools.Equation representing the predicted value for the field ϕ.

Returns gridtools.Equation representing the MacCormack flux in x-direction for ϕ.

Return type obj

_get_maccormack_flux_x_s(*i*, *j*, *k*, *in_U*, *tmp_U_prd*)

Get the gridtools.Equation representing the MacCormack flux in x-direction for the isentropic density.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **in_U** (*obj*) – gridtools.Equation representing the x-momentum at the current time.
- **tmp_U_prd** (*obj*) – gridtools.Equation representing the predicted value for the x-momentum.

Returns gridtools.Equation representing the MacCormack flux in x-direction for the isentropic density.

Return type obj

_get_maccormack_flux_y(*i*, *j*, *k*, *tmp_v_unstg*, *in_phi*, *tmp_v_prd_unstg*, *tmp_phi_prd*)

Get the gridtools.Equation representing the MacCormack flux in y-direction for a generic prognostic variable ϕ.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.

- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **tmp_v_unstg** (*obj*) – gridtools.Equation representing the unstaggered y-velocity at the current time.
- **in_phi** (*obj*) – gridtools.Equation representing the field ϕ at the current time.
- **tmp_v_prd_unstg** (*obj*) – gridtools.Equation representing the predicted value for the unstaggered y-velocity.
- **tmp_phi_prd** (*obj*) – gridtools.Equation representing the predicted value for the field ϕ.

Returns gridtools.Equation representing the MacCormack flux in y-direction for ϕ.

Return type obj

_get_maccormack_flux_y_s(*i*, *j*, *k*, *in_V*, *tmp_V_prd*)

Get the gridtools.Equation representing the MacCormack flux in y-direction for the isentropic density.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **in_V** (*obj*) – gridtools.Equation representing the y-momentum at the current time.
- **tmp_V_prd** (*obj*) – gridtools.Equation representing the predicted value for the y-momentum.

Returns gridtools.Equation representing the MacCormack flux in y-direction for the isentropic density.

Return type obj

_get_maccormack_flux_z(*i*, *j*, *k*, *in_w*, *in_phi*, *in_phi_prv*, *tmp_phi_prd*)

Get the gridtools.Equation representing the MacCormack flux in θ-direction for a generic prognostic variable ϕ.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **in_w** (*obj*) – gridtools.Equation representing the vertical velocity, i.e., the change over time in potential temperature.
- **in_phi** (*obj*) – gridtools.Equation representing the field ϕ at current time.
- **in_phi_prv** (*obj*) – gridtools.Equation representing the provisional value for ϕ, i.e., ϕ stepped disregarding the vertical advection.
- **tmp_phi_prd** (*obj*) – gridtools.Equation representing the predicted value for the field ϕ.

Returns gridtools.Equation representing the MacCormack flux in θ-direction for ϕ.

Return type obj

_get_maccormack_horizontal_predicted_value_Q(*i*, *j*, *k*, *dt*, *in_s*, *tmp_u_unstg*, *tmp_v_unstg*, *in_Q*, *in_q_tnd*)

Get the `gridtools.Equation` representing the predicted value for the isentropic density of a generic water constituent, computed without taking the vertical advection into account.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the air isentropic density.
- **tmp_u_unstg** (*obj*) – `gridtools.Equation` representing the unstaggered x-velocity.
- **tmp_v_unstg** (*obj*) – `gridtools.Equation` representing the unstaggered y-velocity.
- **in_Q** (*obj*) – `gridtools.Equation` representing the isentropic density of a generic water constituent.
- **in_q_tnd** (*obj*) – `gridtools.Equation` representing the tendency of the mass fraction of the water constituent.

Returns `gridtools.Equation` representing the predicted value for the water constituent.

Return type obj

_get_maccormack_horizontal_predicted_value_U(*i*, *j*, *k*, *dt*, *in_s*, *tmp_u_unstg*, *tmp_v_unstg*, *in_mtg*, *in_U*)

Get the `gridtools.Equation` representing the predicted value for the x-momentum, computed without taking the vertical advection into account.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.
- **tmp_u_unstg** (*obj*) – `gridtools.Equation` representing the unstaggered x-velocity.
- **tmp_v_unstg** (*obj*) – `gridtools.Equation` representing the unstaggered y-velocity.
- **in_mtg** (*obj*) – `gridtools.Equation` representing the Montgomery potential.
- **in_U** (*obj*) – `gridtools.Equation` representing the x-momentum.

Returns `gridtools.Equation` representing the predicted value for the x-momentum.

Return type obj

_get_maccormack_horizontal_predicted_value_V(*i*, *j*, *k*, *dt*, *in_s*, *tmp_u_unstg*, *tmp_v_unstg*, *in_mtg*, *in_V*)

Get the `gridtools.Equation` representing the predicted value for the y-momentum, computed without taking the vertical advection into account.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.
- **tmp_u_unstg** (*obj*) – `gridtools.Equation` representing the unstaggered x-velocity.
- **tmp_v_unstg** (*obj*) – `gridtools.Equation` representing the unstaggered y-velocity.
- **in_mtg** (*obj*) – `gridtools.Equation` representing the Montgomery potential.
- **in_V** (*obj*) – `gridtools.Equation` representing the y-momentum.

Returns `gridtools.Equation` representing the predicted value for the y-momentum.

Return type obj

_get_maccormack_horizontal_predicted_value_s(*i*, *j*, *k*, *dt*, *in_s*, *in_U*, *in_V*)

Get the `gridtools.Equation` representing the predicted value for the isentropic density, computed without taking the vertical advection into account.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.
- **in_U** (*obj*) – `gridtools.Equation` representing the x-momentum.
- **in_V** (*obj*) – `gridtools.Equation` representing the y-momentum.

Returns `gridtools.Equation` representing the predicted value for the isentropic density.

Return type obj

_get_maccormack_vertical_predicted_value(*i*, *j*, *k*, *dt*, *in_w*, *in_phi*, *in_phi_prv*)

Get the `gridtools.Equation` representing the predicted value for a generic conservative prognostic variable ϕ, computed taking only the vertical advection into account.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.

- **in_w** (*obj*) – gridtools.Equation representing the vertical velocity, i.e., the change over time in potential temperature.
- **in_phi** (*obj*) – gridtools.Equation representing the field ϕ at current time.
- **in_phi_prv** (*obj*) – gridtools.Equation representing the provisional value for ϕ, i.e., ϕ stepped disregarding the vertical advection.

Returns gridtools.Equation representing the predicted value for ϕ.

Return type obj

_get_velocity(*i*, *j*, *k*, *s*, *mnt*)
Get the gridtools.Equation representing an unstaggered velocity component.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **s** (*obj*) – gridtools.Equation representing the isentropic density.
- **mnt** (*obj*) – gridtools.Equation representing either the x- or the y-momentum.

Returns gridtools.Equation representing the diagnosed unstaggered velocity component.

Return type obj

class tasmania.dycore.flux_isentropic_nonconservative.**FluxIsentropicNonconservative**(*grid*, *moist_on*)
Abstract base class whose derived classes implement different schemes to compute the numerical fluxes for the three-dimensional isentropic dynamical core. The nonconservative form of the governing equations is used.

__init__(*grid*, *moist_on*)
Constructor.

Parameters

- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – True for a moist dynamical core, False otherwise.

_compute_horizontal_fluxes(*i*, *j*, *k*, *dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_qv*, *in_qc*, *in_qr*)
Method computing the gridtools.Equations representing the x- and y-fluxes for all the prognostic variables. The gridtools.Equations are then set as instance attributes. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **dt** (*obj*) – gridtools.Global representing the time step.
- **in_s** (*obj*) – gridtools.Equation representing the isentropic density.
- **in_u** (*obj*) – gridtools.Equation representing the x-velocity.
- **in_v** (*obj*) – gridtools.Equation representing the y-velocity.
- **in_mtg** (*obj*) – gridtools.Equation representing the Montgomery potential.

- **in_qv** (*obj*) – gridtools.Equation representing the mass fraction of water vapour.
- **in_qc** (*obj*) – gridtools.Equation representing the mass fraction of cloud liquid water.
- **in_qr** (*obj*) – gridtools.Equation representing the mass fraction of precipitation water.

_compute_vertical_fluxes(*i*, *j*, *k*, *dt*, *in_w*, *in_s*, *in_s_prv*, *in_U*, *in_U_prv*, *in_V*, *in_V_prv*, *in_Qv*, *in_Qv_prv*, *in_Qc*, *in_Qc_prv*, *in_Qr*, *in_Qr_prv*)

Method computing the gridtools.Equations representing the θ-fluxes for all the prognostic model variables. The gridtools.Equations are then set as instance attributes. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **dt** (*obj*) – gridtools.Global representing the time step.
- **in_w** (*obj*) – gridtools.Equation representing the vertical velocity, i.e., the change over time of potential temperature.
- **in_s** (*obj*) – gridtools.Equation representing the current isentropic density.
- **in_s_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density, i.e., the isentropic density stepped disregarding the vertical advection.
- **in_u** (*obj*) – gridtools.Equation representing the current x-velocity.
- **in_u_prv** (*obj*) – gridtools.Equation representing the provisional x-velocity, i.e., the x-velocity stepped disregarding the vertical advection.
- **in_v** (*obj*) – gridtools.Equation representing the current y-velocity.
- **in_v_prv** (*obj*) – gridtools.Equation representing the provisional y-velocity, i.e., the y-velocity stepped disregarding the vertical advection.
- **in_qv** (*obj*) – gridtools.Equation representing the current mass fraction of water vapor.
- **in_qv_prv** (*obj*) – gridtools.Equation representing the provisional mass fraction of water vapor, i.e., the mass fraction of water vapor stepped disregarding the vertical advection.
- **in_qc** (*obj*) – gridtools.Equation representing the current mass fraction of cloud liquid water.
- **in_qc_prv** (*obj*) – gridtools.Equation representing the provisional mass fraction of cloud liquid water, i.e., the mass fraction of cloud liquid water stepped disregarding the vertical advection.
- **in_qr** (*obj*) – gridtools.Equation representing the current mass fraction of precipitation water.
- **in_qr_prv** (*obj*) – gridtools.Equation representing the provisional mass fraction of precipitation water, i.e., the mass fraction of precipitation water stepped disregarding the vertical advection.

static factory(*scheme*, *grid*, *moist_on*)

Static method which returns an instance of the derived class implementing the numerical scheme specified by `scheme`.

Parameters

- **scheme** (*str*) – String specifying the numerical scheme to implement. Either:
  - 'centered', for a second-order centered scheme.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.

Returns Instance of the derived class implementing the scheme specified by `scheme`.

Return type obj

get_horizontal_fluxes(*i*, *j*, *k*, *dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_qv=None*, *in_qc=None*, *in_qr=None*)

Method returning the `gridtools.Equation`s representing the x- and y-fluxes for all the prognostic model variables.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity.
- **in_mtg** (*obj*) – `gridtools.Equation` representing the Montgomery potential.
- **in_qv** (*obj*, optional) – `gridtools.Equation` representing the mass fraction of water vapour.
- **in_qc** (*obj*, optional) – `gridtools.Equation` representing the mass fraction of cloud liquid water.
- **in_qr** (*obj*, optional) – `gridtools.Equation` representing the mass fraction of precipitation water.

Returns

- **flux_s_x** (*obj*) – `gridtools.Equation` representing the x-flux for the isentropic density.
- **flux_s_y** (*obj*) – `gridtools.Equation` representing the y-flux for the isentropic density.
- **flux_u_x** (*obj*) – `gridtools.Equation` representing the x-flux for the x-velocity.
- **flux_u_y** (*obj*) – `gridtools.Equation` representing the y-flux for the x-velocity.
- **flux_v_x** (*obj*) – `gridtools.Equation` representing the x-flux for the y-velocity.
- **flux_v_y** (*obj*) – `gridtools.Equation` representing the y-flux for the y-velocity.
- **flux_qv_x** (*obj*, optional) – `gridtools.Equation` representing the x-flux for the mass fraction of water vapor.

- **flux_qv_y** (*obj*, optional) – `gridtools.Equation` representing the y-flux for the mass fraction of water vapor.
- **flux_qc_x** (*obj*, optional) – `gridtools.Equation` representing the x-flux for the mass fraction of cloud liquid water.
- **flux_qc_y** (*obj*, optional) – `gridtools.Equation` representing the y-flux for the mass fraction of cloud liquid water.
- **flux_qr_x** (*obj*, optional) – `gridtools.Equation` representing the x-flux for the mass fraction of precipitation water.
- **flux_qr_y** (*obj*, optional) – `gridtools.Equation` representing the y-flux for the mass fraction of precipitation water.

get_vertical_fluxes(*i*, *j*, *k*, *dt*, *in_w*, *in_s*, *in_s_prv*, *in_u*, *in_u_prv*, *in_v*, *in_v_prv*, *in_qv=None*, *in_qv_prv=None*, *in_qc=None*, *in_qc_prv=None*, *in_qr=None*, *in_qr_prv=None*)

Method returning the `gridtools.Equation`s representing the θ-flux for all the prognostic model variables.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_w** (*obj*) – `gridtools.Equation` representing the vertical velocity, i.e., the change over time in potential temperature.
- **in_s** (*obj*) – `gridtools.Equation` representing the current isentropic density.
- **in_s_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density, i.e., the isentropic density stepped disregarding the vertical advection.
- **in_u** (*obj*) – `gridtools.Equation` representing the current x-velocity.
- **in_u_prv** (*obj*) – `gridtools.Equation` representing the provisional x-velocity, i.e., the x-velocity stepped disregarding the vertical advection.
- **in_v** (*obj*) – `gridtools.Equation` representing the current y-velocity.
- **in_v_prv** (*obj*) – `gridtools.Equation` representing the provisional y-velocity, i.e., the y-velocity stepped disregarding the vertical advection.
- **in_qv** (*obj*, optional) – `gridtools.Equation` representing the current mass fraction of water vapor.
- **in_qv_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional mass fraction of water vapor, i.e., the mass fraction of water vapor stepped disregarding the vertical advection.
- **in_qc** (*obj*, optional) – `gridtools.Equation` representing the current mass fraction of cloud liquid water.
- **in_qc_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional mass fraction of cloud liquid water, i.e., the mass fraction of cloud liquid water stepped disregarding the vertical advection.
- **in_qr** (*obj*, optional) – `gridtools.Equation` representing the current mass fraction of precipitation water.

- **in_qr_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional mass fraction of precipitation water, i.e., the mass fraction of precipitation water stepped disregarding the vertical advection.

Returns

- **flux_s_z** (*obj*) – `gridtools.Equation` representing the θ-flux for the isentropic density.
- **flux_u_z** (*obj*) – `gridtools.Equation` representing the θ-flux for the x-velocity.
- **flux_v_z** (*obj*) – `gridtools.Equation` representing the θ-flux for the y-velocity.
- **flux_qv_z** (*obj*, optional) – `gridtools.Equation` representing the θ-flux for the mass fraction of water vapor.
- **flux_qc_z** (*obj*, optional) – `gridtools.Equation` representing the θ-flux for the mass fraction of cloud liquid water.
- **flux_qr_z** (*obj*, optional) – `gridtools.Equation` representing the θ-flux for the mass fraction of precipitation water.

class tasmania.dycore.flux_isentropic_nonconservative_centered.**FluxIsentropicNonconservativ**

Class which inherits `FluxIsentropicNonconservative` to implement a centered scheme to compute the numerical fluxes for the prognostic model variables. The nonconservative form of the governing equations, expressed using isentropic coordinates, is used.

Variables

- ***nb*** (*int*) – Number of boundary layers.
- **order** (*int*) – Order of accuracy.

__init__(*grid*, *moist_on*)
Constructor.

Parameters

- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.

_compute_horizontal_fluxes(*i*, *j*, *k*, *dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_qv*, *in_qc*, *in_qr*)
Method computing the `gridtools.Equation`s representing the x- and y-fluxes for all the prognostic variables. The `gridtools.Equation`s are then set as instance attributes.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity.
- **in_mtg** (*obj*) – `gridtools.Equation` representing the Montgomery potential.
- **in_qv** (*obj*) – `gridtools.Equation` representing the mass fraction of water vapour.

- **in_qc** (*obj*) – gridtools.Equation representing the mass fraction of cloud liquid water.
- **in_qr** (*obj*) – gridtools.Equation representing the mass fraction of precipitation water.

_compute_vertical_fluxes(*i*, *j*, *k*, *dt*, *in_w*, *in_s*, *in_s_prv*, *in_U*, *in_U_prv*, *in_V*, *in_V_prv*, *in_Qv*, *in_Qv_prv*, *in_Qc*, *in_Qc_prv*, *in_Qr*, *in_Qr_prv*)

Method computing the gridtools.Equations representing the θ-fluxes for all the prognostic model variables. The gridtools.Equations are then set as instance attributes.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **dt** (*obj*) – gridtools.Global representing the time step.
- **in_w** (*obj*) – gridtools.Equation representing the vertical velocity, i.e., the change over time of potential temperature.
- **in_s** (*obj*) – gridtools.Equation representing the current isentropic density.
- **in_s_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density, i.e., the isentropic density stepped disregarding the vertical advection.
- **in_u** (*obj*) – gridtools.Equation representing the current x-velocity.
- **in_u_prv** (*obj*) – gridtools.Equation representing the provisional x-velocity, i.e., the x-velocity stepped disregarding the vertical advection.
- **in_v** (*obj*) – gridtools.Equation representing the current y-velocity.
- **in_v_prv** (*obj*) – gridtools.Equation representing the provisional y-velocity, i.e., the y-velocity stepped disregarding the vertical advection.
- **in_qv** (*obj*) – gridtools.Equation representing the current mass fraction of water vapor.
- **in_qv_prv** (*obj*) – gridtools.Equation representing the provisional mass fraction of water vapor, i.e., the mass fraction of water vapor stepped disregarding the vertical advection.
- **in_qc** (*obj*) – gridtools.Equation representing the current mass fraction of cloud liquid water.
- **in_qc_prv** (*obj*) – gridtools.Equation representing the provisional mass fraction of cloud liquid water, i.e., the mass fraction of cloud liquid water stepped disregarding the vertical advection.
- **in_qr** (*obj*) – gridtools.Equation representing the current mass fraction of precipitation water.
- **in_qr_prv** (*obj*) – gridtools.Equation representing the provisional mass fraction of precipitation water, i.e., the mass fraction of precipitation water stepped disregarding the vertical advection.

_get_centered_flux_x_s(*i*, *j*, *k*, *in_u*, *in_s*)

Get the x-flux for the isentropic density.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.

- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.

Returns `gridtools.Equation` representing the x-flux for the isentropic density.

Return type obj

_get_centered_flux_x_u(*i*, *j*, *k*, *in_u*)
Get the x-flux for the x-velocity.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity.

Returns `gridtools.Equation` representing the x-flux for the x-velocity.

Return type obj

_get_centered_flux_x_unstg(*i*, *j*, *k*, *in_phi*)
Get the x-flux for a generic x-unstaggered field.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **in_phi** (*obj*) – `gridtools.Equation` representing the advected field.

Returns `gridtools.Equation` representing the x-flux for the advected field.

Return type obj

_get_centered_flux_y_s(*i*, *j*, *k*, *in_v*, *in_s*)
Get the y-flux for the isentropic density.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density.

Returns `gridtools.Equation` representing the y-flux for the isentropic density.

Return type obj

_get_centered_flux_y_unstg(*i*, *j*, *k*, *in_phi*)
Get the y-flux for a generic y-unstaggered field.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **in_phi** (*obj*) – `gridtools.Equation` representing the advected field.

Returns `gridtools.Equation` representing the y-flux for the advected field.

Return type obj

_get_centered_flux_y_v(*i*, *j*, *k*, *in_v*)
Get the y-flux for the y-velocity.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity.

Returns `gridtools.Equation` representing the y-flux for the y-velocity.

Return type obj

1.2.6 Prognostics

class tasmania.dycore.prognostic_isentropic.**PrognosticIsentropic**(*flux_scheme*, *grid*, *moist_on*, *backend*, *physics_dynamics_coupling_on*, *sedimentation_on*, *sedimentation_flux_type*, *sedimentation_substeps*)
Abstract base class whose derived classes implement different schemes to carry out the prognostic steps of the three-dimensional moist isentropic dynamical core. The conservative form of the governing equations is used.

Variables ***fast_tendency_parameterizations*** (*list*) – List containing instances of derived classes of *FastTendency* which are in charge of calculating fast-varying tendencies.

__init__(*flux_scheme*, *grid*, *moist_on*, *backend*, *physics_dynamics_coupling_on*, *sedimentation_on*, *sedimentation_flux_type*, *sedimentation_substeps*)
Constructor.

Parameters

- **flux_scheme** (*str*) – String specifying the flux scheme to use. Either:
  - 'upwind', for the upwind flux;
  - 'centered', for a second-order centered flux;
  - 'maccormack', for the MacCormack flux.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.

- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.
- **physics_dynamics_coupling_on** (*bool*) – `True` to couple physics with dynamics, i.e., to account for the change over time in potential temperature, `False` otherwise.
- **sedimentation_on** (*bool*) – `True` to account for rain sedimentation, `False` otherwise.
- **sedimentation_flux_type** (*str*) – String specifying the method used to compute the numerical sedimentation flux. Available options are:
  - 'first_order_upwind', for the first-order upwind scheme;
  - 'second_order_upwind', for the second-order upwind scheme.
- **sedimentation_substeps** (*int*) – Number of sub-timesteps to perform in order to integrate the sedimentation flux.

_stencil_stepping_by_coupling_physics_with_dynamics_allocate_inputs()
Allocate the attributes which serve as inputs to the GT4Py stencil which step the solution by coupling physics with dynamics, i.e., accounting for the change over time in potential temperature.

_stencil_stepping_by_coupling_physics_with_dynamics_allocate_outputs()
Allocate the Numpy arrays which will store the solution updated by coupling physics with dynamics.

_stencil_stepping_by_coupling_physics_with_dynamics_defs(*dt*, *in_w*, *in_s_now*, *in_s_prv*, *in_U_now*, *in_U_prv*, *in_V_now*, *in_V_prv*, *Qv_now=None*, *Qv_prv=None*, *Qc_now=None*, *Qc_prv=None*, *Qr_now=None*, *Qr_prv=None*)
GT4Py stencil stepping the solution by coupling physics with dynamics, i.e., by accounting for the change over time in potential temperature. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_w** (*array_like*) – `numpy.ndarray` representing the vertical velocity, i.e., the change over time in potential temperature.
- **in_s_now** (*obj*) – `gridtools.Equation` representing the current isentropic density.
- **in_s_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density.
- **in_U_now** (*obj*) – `gridtools.Equation` representing the current x-momentum.
- **in_U_prv** (*obj*) – `gridtools.Equation` representing the provisional x-momentum.
- **in_V_now** (*obj*) – `gridtools.Equation` representing the current y-momentum.

- **in_V_prv** (*obj*) – `gridtools.Equation` representing the provisional y-momentum.
- **in_Qv_now** (*obj*, optional) – `gridtools.Equation` representing the current isentropic density of water vapor.
- **in_Qv_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional isentropic density of water vapor.
- **in_Qc_now** (*obj*, optional) – `gridtools.Equation` representing the current isentropic density of cloud liquid water.
- **in_Qc_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional isentropic density of cloud liquid water.
- **in_Qr_now** (*obj*, optional) – `gridtools.Equation` representing the current isentropic density of precipitation water.
- **in_Qr_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional isentropic density of precipitation water.

Returns

- **out_s** (*obj*) – `gridtools.Equation` representing the updated isentropic density.
- **out_U** (*obj*) – `gridtools.Equation` representing the updated x-momentum.
- **out_V** (*obj*) – `gridtools.Equation` representing the updated y-momentum.
- **out_Qv** (*obj*, optional) – `gridtools.Equation` representing the updated isentropic density of water vapor.
- **out_Qc** (*obj*, optional) – `gridtools.Equation` representing the updated isentropic density of cloud liquid water.
- **out_Qr** (*obj*, optional) – `gridtools.Equation` representing the updated isentropic density of precipitation water.

_stencil_stepping_by_coupling_physics_with_dynamics_initialize(*state_now*)

Initialize the GT4Py stencil in charge of stepping the solution by coupling physics with dynamics, i.e., by accounting for the change over time in potential temperature.

Parameters **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:

- air_isentropic_density (unstaggered).

_stencil_stepping_by_coupling_physics_with_dynamics_set_inputs(*dt*, *state_now*, *state_prv*, *tendencies*)

Update the attributes which serve as inputs to the GT4Py stencil which steps the solution by integrating the vertical advection, i.e., by accounting for the change over time in potential temperature.

Parameters

- **dt** (*obj*) – A `datetime.timedelta` representing the time step.
- **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_momentum_isentropic (unstaggered);

- y_momentum_isentropic (unstaggered);
- water_vapor_isentropic_density (unstaggered, optional);
- cloud_liquid_water_isentropic_density (unstaggered, optional);
- precipitation_water_isentropic_density (unstaggered, optional).

- **state_prv** (*obj*) – *StateIsentropic* representing the provisional state, i.e., the state stepped taking only the horizontal derivatives into account. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_momentum_isentropic (unstaggered);
  - y_momentum_isentropic (unstaggered);
  - water_vapor_isentropic_density (unstaggered, optional);
  - cloud_liquid_water_isentropic_density (unstaggered, optional);
  - precipitation_water_isentropic_density (unstaggered, optional).

  This may be the output of *step_neglecting_vertical_advection()*.
- **tendencies** (*obj*) – *GridData* collecting the following tendencies:
  - tendency_of_air_potential_temperature (unstaggered).

_stencils_stepping_by_neglecting_vertical_advection_allocate_inputs(*mi*, *mj*, *tendencies*)

Allocate the attributes which serve as inputs to the GT4Py stencils which step the solution disregarding the vertical advection.

Parameters

- **mi** (*int*) – x-extent of an input array representing an x-unstaggered field.
- **mj** (*int*) – y-extent of an input array representing a y-unstaggered field.
- **tendencies** (*obj*) – *GridData* storing the following tendencies:
  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).

_stencils_stepping_by_neglecting_vertical_advection_allocate_outputs(*mi*, *mj*)

Allocate the Numpy arrays which will store the solution updated by neglecting the vertical advection.

Parameters

- **mi** (*int*) – x-extent of an output array representing an x-unstaggered field.
- **mj** (*int*) – y-extent of an output array representing a y-unstaggered field.

_stencils_stepping_by_neglecting_vertical_advection_set_inputs(*dt*, *state*, *tendencies*)

Update the attributes which serve as inputs to the GT4Py stencils which step the solution disregarding the vertical advection.

Parameters

- **dt** (*obj*) – A `datetime.timedelta` representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_momentum_isentropic (unstaggered);
  - montgomery_potential (isentropic);
  - water_vapor_isentropic_density (unstaggered, optional);
  - cloud_liquid_water_isentropic_density (unstaggered, optional);
  - precipitation_water_isentropic_density (unstaggered, optional).
- **tendencies** (*obj*) – *GridData* storing the following tendencies:
  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).

boundary

Get the attribute implementing the horizontal boundary conditions. If this is set to `None`, a `ValueError` is thrown.

Returns Instance of the derived class of *HorizontalBoundary* implementing the horizontal boundary conditions.

Return type obj

diagnostic

Get the attribute implementing the diagnostic step of the three-dimensional moist isentropic dynamical core. If this is set to `None`, a `ValueError` is thrown.

Returns *DiagnosticIsentropic* carrying out the diagnostic step of the three-dimensional moist isentropic dynamical core.

Return type obj

static factory(*time_scheme*, *flux_scheme*, *grid*, *moist_on*, *backend*, *physics_dynamics_coupling_on*, *sedimentation_on*, *sedimentation_flux_type*, *sedimentation_substeps*)

Static method returning an instace of the derived class implementing the time stepping scheme specified by `time_scheme`, using the flux scheme specified by `flux_scheme`.

Parameters

- **time_scheme** (*str*) – String specifying the time stepping method to implement. Either:
  - 'forward_euler', for the forward Euler scheme;
  - 'centered', for a centered scheme.
- **flux_scheme** (*str*) – String specifying the scheme to use. Either:
  - 'upwind', for the upwind flux;

  - 'centered', for a second-order centered flux;
  - 'maccormack', for the MacCormack flux.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.
- **backend** (*obj*) – `gridtools.Mode` specifying the backend for the GT4Py stencils.
- **physics_dynamics_coupling_on** (*bool*) – `True` to couple physics with dynamics, i.e., to account for the change over time in potential temperature, `False` otherwise.
- **sedimentation_on** (*bool*) – `True` to account for rain sedimentation, `False` otherwise.
- **sedimentation_flux_type** (*str*) – String specifying the method used to compute the numerical sedimentation flux. Available options are:
  - 'first_order_upwind', for the first-order upwind scheme;
  - 'second_order_upwind', for the second-order upwind scheme.
- **sedimentation_substeps** (*int*) – Number of sub-timesteps to perform in order to integrate the sedimentation flux.

Returns An instace of the derived class implementing the scheme specified by `scheme`.

Return type obj

microphysics

Get the attribute in charge of calculating the raindrop fall velocity. If this is set to `None`, a `ValueError` is thrown.

Returns Instance of a derived class of either `TendencyMicrophysics` or *AdjustmentMicrophysics* which provides the raindrop fall velocity.

Return type obj

nb

Get the number of lateral boundary layers.

Returns The number of lateral boundary layers.

Return type int

step_coupling_physics_with_dynamics(*dt*, *state_now*, *state_prv*, *tendencies*)

Method advancing the conservative, prognostic model variables one time step forward by coupling physics with dynamics, i.e., by accounting for the change over time in potential temperature.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_momentum_isentropic (unstaggered);
  - y_momentum_isentropic (unstaggered);
  - water_vapor_isentropic_density (unstaggered, optional);
  - cloud_liquid_water_isentropic_density (unstaggered, optional);
  - precipitation_water_isentropic_density (unstaggered, optional).

- **state_prv** (*obj*) – *StateIsentropic* representing the provisional state, i.e., the state stepped taking only the horizontal derivatives into account. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_momentum_isentropic (unstaggered);
  - y_momentum_isentropic (unstaggered);
  - water_vapor_isentropic_density (unstaggered, optional);
  - cloud_liquid_water_isentropic_density (unstaggered, optional);
  - precipitation_water_isentropic_density (unstaggered, optional).

  This may be the output of *step_neglecting_vertical_advection()*.
- **tendencies** (*obj*) – *GridData* collecting the following tendencies:
  - tendency_of_air_potential_temperature (unstaggered).

Returns

StateIsentropic containing the updated prognostic variables, i.e.,

- air_isentropic_density (unstaggered);
- x_momentum_isentropic (unstaggered);
- y_momentum_isentropic (unstaggered);
- water_vapor_isentropic_density (unstaggered, optional);
- cloud_liquid_water_isentropic_density (unstaggered, optional);
- precipitation_water_isentropic_density (unstaggered, optional).

Return type obj

step_integrating_sedimentation_flux(*dt*, *state_now*, *state_prv*, *diagnostics=None*)

Method advancing the mass fraction of precipitation water by taking the sedimentation into account. For the sake of numerical stability, a time-splitting strategy is pursued, i.e., sedimentation is resolved using a timestep which may be smaller than that specified by the user. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - height or height_on_interface_levels (z-staggered);
  - mass_fraction_of_precipitation_water_in air (unstaggered).
- **state_prv** (*obj*) – *StateIsentropic* representing the provisional state, i.e., the state stepped without taking the sedimentation flux into account. It should contain the following variables:
  - mass_fraction_of_precipitation_water_in_air (unstaggered).

  This may be the output of either *step_neglecting_vertical_advection()* or *step_coupling_physics_with_dynamics()*.

- **diagnostics** (*obj*, optional) – *GridData* collecting the following diagnostics:
  - accumulated_precipitation (unstaggered, two-dimensional);
  - precipitation (unstaggered, two-dimensional).

Returns

- **state_new** (*obj*) – *StateIsentropic* containing the following updated variables:
  - mass_fraction_of_precipitation_water_in air (unstaggered).
- **diagnostics_out** (*obj*) – *GridData* collecting the output diagnostics, i.e.:
  - accumulated_precipitation (unstaggered, two-dimensional);
  - precipitation (unstaggered, two-dimensional).

step_neglecting_vertical_advection(*dt*, *state*, *state_old=None*, *tendencies=None*)

Method advancing the conservative, prognostic model variables one time step forward. Only horizontal derivates are considered; possible vertical derivatives are disregarded. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_momentum_isentropic (unstaggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (isentropic);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **state_old** (*obj*, optional) – *StateIsentropic* representing the old state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_momentum_isentropic (unstaggered);
  - y_momentum_isentropic (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **tendencies** (*obj*, optional) – *GridData* storing the following tendencies:
  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);

– tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).

Default is None.

Returns

StateIsentropic containing the updated prognostic variables, i.e.,

- air_isentropic_density (unstaggered);
- x_momentum_isentropic (unstaggered);
- y_momentum_isentropic (unstaggered);
- water_vapor_isentropic_density (unstaggered, optional);
- cloud_liquid_water_isentropic_density (unstaggered, optional);
- precipitation_water_isentropic_density (unstaggered, optional).

Return type obj

class tasmania.dycore.prognostic_isentropic_centered.**PrognosticIsentropicCentered**(*flux_scheme*, *grid*, *moist_on*, *backend*, *coupling_physics_dynamics_on*, *sedimentation_on*, *sedimentation_flux_type*, *sedimentation_substeps*)

This class inherits *PrognosticIsentropic* to implement a centered time-integration scheme to carry out the prognostic step of the three-dimensional moist isentropic dynamical core.

Variables

- ***time_levels*** (*int*) – Number of time levels the scheme relies on.
- **steps** (*int*) – Number of steps the scheme entails.

__init__(*flux_scheme*, *grid*, *moist_on*, *backend*, *coupling_physics_dynamics_on*, *sedimentation_on*, *sedimentation_flux_type*, *sedimentation_substeps*)

Constructor.

Parameters

- **flux_scheme** (*str*) – String specifying the flux scheme to use. Either:
  - 'upwind', for the upwind flux;
  - 'centered', for a second-order centered flux;

  - 'maccormack', for the MacCormack flux.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.
- **physics_dynamics_coupling_on** (*bool*) – `True` to couple physics with dynamics, i.e., to account for the change over time in potential temperature, `False` otherwise.
- **sedimentation_on** (*bool*) – `True` to account for rain sedimentation, `False` otherwise.
- **sedimentation_flux_type** (*str*) – String specifying the method used to compute the numerical sedimentation flux. Available options are:
  - 'first_order_upwind', for the first-order upwind scheme;
  - 'second_order_upwind', for the second-order upwind scheme.
- **sedimentation_substeps** (*int*) – Number of sub-timesteps to perform in order to integrate the sedimentation flux.

Note: To instantiate an object of this class, one should prefer the static method *factory()* of *PrognosticIsentropic*.

_stencil_clipping_mass_fraction_and_diagnosing_isentropic_density_of_precipitation_wat

GT4Py stencil clipping the negative values for the mass fraction of precipitation water, and diagnosing the isentropic density of precipitation water.

Parameters

- **in_s** (*obj*) – `gridtools.Equation` representing the air isentropic density.
- **in_qr** (*obj*) – `gridtools.Equation` representing the mass fraction of precipitation water in air.

Returns

- **out_qr** (*obj*) – `gridtools.Equation` representing the clipped mass fraction of precipitation water.
- **out_Qr** (*obj*) – `gridtools.Equation` representing the isentropic density of precipitation water.

_stencil_computing_slow_tendencies_defs (*dt*, *in_s_old*, *in_s_prv*, *in_qr_old*, *in_qr_prv*)

GT4Py stencil computing the slow tendencies required to resolve rain sedimentation.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the large timestep.
- **in_s_old** (*obj*) – `gridtools.Equation` representing the old isentropic density.
- **in_s_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density.
- **in_qr_old** (*obj*) – `gridtools.Equation` representing the old mass fraction of precipitation water.
- **in_qr_prv** (*obj*) – `gridtools.Equation` representing the provisional mass fraction of precipitation water.

Returns

- **out_s_tnd** (*obj :*) – `gridtools.Equation` representing the slow tendency for the isentropic density.
- **out_qr_tnd** (*obj :*) – `gridtools.Equation` representing the slow tendency for the mass fraction of precipitation water.

_stencil_ensuring_vertical_cfl_is_obeyed_defs(*dts*, *in_h*, *in_vt*)

GT4Py stencil ensuring that the vertical CFL condition is fulfilled. This is achieved by clipping the raindrop fall velocity field: if a cell does not satisfy the CFL constraint, the vertical velocity at that cell is reduced so that the local CFL number equals 0.95.

Parameters

- **dts** (*obj*) – `gridtools.Global` representing the large timestep.
- **in_h** (*obj*) – `gridtools.Equation` representing the geometric height.
- **in_vt** (*obj*) – `gridtools.Equation` representing the raindrop fall velocity.

Returns `gridtools.Equation` representing the clipped raindrop fall velocity.

Return type obj

_stencil_stepping_by_coupling_physics_with_dynamics_defs(*dt*, *in_w*, *in_s*, *in_s_prv*, *in_U*, *in_U_prv*, *in_V*, *in_V_prv*, *in_Qv=None*, *in_Qv_prv=None*, *in_Qc=None*, *in_Qc_prv=None*, *in_Qr=None*, *in_Qr_prv=None*)

GT4Py stencil stepping the solution by coupling physics with dynamics, i.e., by accounting for the change over time in potential temperature.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_w** (*obj*) – `gridtools.Equation` representing the vertical velocity, i.e., the change over time in potential temperature.
- **in_s** (*obj*) – `gridtools.Equation` representing the current isentropic density.
- **in_s_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density.
- **in_U** (*obj*) – `gridtools.Equation` representing the current x-momentum.
- **in_U_prv** (*obj*) – `gridtools.Equation` representing the provisional x-momentum.
- **in_V** (*obj*) – `gridtools.Equation` representing the current y-momentum.
- **in_V_prv** (*obj*) – `gridtools.Equation` representing the provisional y-momentum.
- **in_Qv** (*obj*, optional) – `gridtools.Equation` representing the current isentropic density of water vapor.
- **in_Qc** (*obj*, optional) – `gridtools.Equation` representing the current isentropic density of cloud liquid water.

- **in_Qr** (*obj*, optional) – `gridtools.Equation` representing the current isentropic density of precipitation water.
- **in_Qv_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional isentropic density of water vapor.
- **in_Qc_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional isentropic density of cloud liquid water.
- **in_Qr_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional isentropic density of precipitation water.

Returns

- **out_s** (*obj*) – `gridtools.Equation` representing the updated isentropic density.
- **out_U** (*obj*) – `gridtools.Equation` representing the updated x-momentum.
- **out_V** (*obj*) – `gridtools.Equation` representing the updated y-momentum.
- **out_Qv** (*obj*, optional) – `gridtools.Equation` representing the updated isentropic density of water vapor.
- **out_Qc** (*obj*, optional) – `gridtools.Equation` representing the updated isentropic density of cloud liquid water.
- **out_Qr** (*obj*, optional) – `gridtools.Equation` representing the updated isentropic density of precipitation water.

_stencil_stepping_by_integrating_sedimentation_flux_defs(*dts*, *in_rho*, *in_s*, *in_h*, *in_qr*, *in_vt*, *in_s_tnd*, *in_qr_tnd*)

GT4Py stencil stepping the isentropic density and the mass fraction of precipitation water by integrating the precipitation flux.

Parameters

- **dts** (*obj*) – `gridtools.Global` representing the small timestep.
- **in_rho** (*obj*) – `gridtools.Equation` representing the air density.
- **in_s** (*obj*) – `gridtools.Equation` representing the air isentropic density.
- **in_h** (*obj*) – `gridtools.Equation` representing the geometric height of the model half-levels.
- **in_qr** (*obj*) – `gridtools.Equation` representing the input mass fraction of precipitation water.
- **in_vt** (*obj*) – `gridtools.Equation` representing the raindrop fall velocity.
- **in_s_tnd** (*obj*) – `gridtools.Equation` representing the contribution from the slow tendencies for the isentropic density.
- **in_qr_tnd** (*obj*) – `gridtools.Equation` representing the contribution from the slow tendencies for the mass fraction of precipitation water.

Returns

- **out_s** (*obj*) – `gridtools.Equation` representing the output isentropic density.
- **out_qr** (*obj*) – `gridtools.Equation` representing the output mass fraction of precipitation water.

_stencil_stepping_by_neglecting_vertical_advection_defs(*dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_U*, *in_V*, *in_s_old*, *in_U_old*, *in_V_old*, *in_Qv=None*, *in_Qc=None*, *in_Qr=None*, *in_Qv_old=None*, *in_Qc_old=None*, *in_Qr_old=None*, *in_qv_tnd=None*, *in_qc_tnd=None*, *in_qr_tnd=None*)

GT4Py stencil implementing the centered time-integration scheme.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density at the current time.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity at the current time.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity at the current time.
- **in_mtg** (*obj*) – `gridtools.Equation` representing the Montgomery potential at the current time.
- **in_U** (*obj*) – `gridtools.Equation` representing the x-momentum at the current time.
- **in_V** (*obj*) – `gridtools.Equation` representing the y-momentum at the current time.
- **in_s_old** (*obj*) – `gridtools.Equation` representing the isentropic density at the previous time level.
- **in_U_old** (*obj*) – `gridtools.Equation` representing the x-momentum at the previous time level.
- **in_V_old** (*obj*) – `gridtools.Equation` representing the y-momentum at the previous time level.
- **in_Qv** (*obj*, optional) – `gridtools.Equation` representing the isentropic density of water vapour at the current time.
- **in_Qc** (*obj*, optional) – `gridtools.Equation` representing the isentropic density of cloud water at the current time.
- **in_Qr** (*obj*, optional) – `gridtools.Equation` representing the isentropic density of precipitation water at the current time.
- **in_Qv_old** (*obj*, optional) – `gridtools.Equation` representing the isentropic density of water vapour at the previous time level.
- **in_Qc_old** (*obj*, optional) – `gridtools.Equation` representing the isentropic density of cloud water at the previous time level.
- **in_Qr_old** (*obj*, optional) – `gridtools.Equation` representing the isentropic density of precipitation water at the previous time level.

- **in_qv_tnd** (*obj*, optional) – `gridtools.Equation` representing the parameterized tendency of the mass fraction of water vapor.
- **in_qc_tnd** (*obj*, optional) – `gridtools.Equation` representing the parameterized tendency of the mass fraction of cloud liquid water.
- **in_qr_tnd** (*obj*, optional) – `gridtools.Equation` representing the parameterized tendency of the mass fraction of precipitation water.

Returns

- **out_s** (*obj*) – `gridtools.Equation` representing the stepped isentropic density.
- **out_U** (*obj*) – `gridtools.Equation` representing the stepped x-momentum.
- **out_V** (*obj*) – `gridtools.Equation` representing the stepped y-momentum.
- **out_Qv** (*obj*, optional) – `gridtools.Equation` representing the stepped mass of water vapour.
- **out_Qc** (*obj*, optional) – `gridtools.Equation` representing the stepped mass of cloud water.
- **out_Qr** (*obj*, optional) – `gridtools.Equation` representing the stepped mass of precipitation water.

_stencil_stepping_by_neglecting_vertical_advection_initialize(*state*, *tendencies*)

Initialize the GT4Py stencil implementing a time-integration centered scheme to step the solution by neglecting vertical advection.

Parameters

- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered).
- **tendencies** (*obj*) – *GridData* storing the following tendencies:
  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).

_stencils_stepping_by_integrating_sedimentation_flux_initialize()

Initialize the GT4Py stencils in charge of stepping the mass fraction of precipitation water by integrating the sedimentation flux.

_stencils_stepping_by_neglecting_vertical_advection_allocate_inputs(*mi*, *mj*, *tendencies*)

Allocate the attributes which will serve as inputs to the GT4Py stencil stepping the solution by neglecting vertical advection.

Parameters

- **mi** (*int*) – x-extent of an input array representing an x-unstaggered field.
- **mj** (*int*) – y-extent of an input array representing a y-unstaggered field.
- **tendencies** (*obj*) – *GridData* storing the following tendencies:

- tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered);
- tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
- tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).

_stencils_stepping_by_neglecting_vertical_advection_set_inputs(*dt*, *state*, *state_old=None*, *tendencies=None*)

Update the attributes which serve as inputs to the GT4Py stencil stepping the solution by neglecting vertical advection.

Parameters

- **dt** (*obj*) – A `datetime.timedelta` representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_velocity (y-staggered);
  - y_momentum_isentropic (unstaggered);
  - montgomery_potential (isentropic);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **state_old** (*obj*, optional) – *StateIsentropic* representing the old state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_momentum_isentropic (unstaggered);
  - y_momentum_isentropic (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **tendencies** (*obj*, optional) – *GridData* storing the following tendencies:
  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).

step_integrating_sedimentation_flux(*dt*, *state_now*, *state_prv*, *diagnostics=None*)

Method advancing the mass fraction of precipitation water by taking the sedimentation into account. For the sake of numerical stability, a time-splitting strategy is pursued, i.e., sedimentation is resolved using a timestep which may be smaller than that specified by the user.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.

- **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_density (unstaggered);
  - air_isentropic_density (unstaggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - height or height_on_interface_levels (z-staggered);
  - mass_fraction_of_precipitation_water_in air (unstaggered).
- **state_prv** (*obj*) – *StateIsentropic* representing the provisional state, i.e., the state stepped without taking the sedimentation flux into account. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - mass_fraction_of_precipitation_water_in air (unstaggered).

  This may be the output of either *step_neglecting_vertical_advection()* or *step_coupling_physics_with_dynamics()*.
- **diagnostics** (*obj*, optional) – *GridData* collecting the following diagnostics:
  - accumulated_precipitation (unstaggered, two-dimensional).

Returns

- **state_new** (*obj*) – *StateIsentropic* containing the following updated variables:
  - mass_fraction_of_precipitation_water_in air (unstaggered);
  - precipitation_water_isentropic_density (unstaggered).
- **diagnostics_out** (*obj*) – *GridData* collecting the output diagnostics, i.e.:
  - accumulated_precipitation (unstaggered, two-dimensional);
  - precipitation (unstaggered, two-dimensional).

step_neglecting_vertical_advection(*dt*, *state*, *state_old=None*, *tendencies=None*)

Method advancing the conservative, prognostic model variables one time step forward via a centered time-integration scheme. Only horizontal derivates are considered; possible vertical derivatives are disregarded.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_velocity (y-staggered);
  - y_momentum_isentropic (unstaggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (isentropic);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);

  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **state_old** (*obj*, optional) – *StateIsentropic* representing the old state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_momentum_isentropic (unstaggered);
  - y_momentum_isentropic (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **tendencies** (*obj*, optional) – *GridData* storing the following tendencies:
  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).

  Default is `None`.

Returns

StateIsentropic containing the updated prognostic variables, i.e.,

- air_isentropic_density (unstaggered);
- x_momentum_isentropic (unstaggered);
- y_momentum_isentropic (unstaggered);
- water_vapor_isentropic_density (unstaggered, optional);
- cloud_liquid_water_isentropic_density (unstaggered, optional);
- precipitation_water_isentropic_density (unstaggered, optional).

Return type obj

class tasmania.dycore.prognostic_isentropic_forward_euler.**PrognosticIsentropicForwardEuler**

This class inherits *PrognosticIsentropic* to implement the forward Euler scheme carrying out the prognostic step of the three-dimensional moist isentropic dynamical core.

Variables

- ***time_levels*** (*int*) – Number of time levels the scheme relies on.
- **steps** (*int*) – Number of steps the scheme entails.

__init__ (*flux_scheme*, *grid*, *moist_on*, *backend*, *coupling_physics_dynamics_on*, *sedimentation_on*, *sedimentation_flux_type*, *sedimentation_substeps*)

Constructor.

Parameters

- **flux_scheme** (*str*) – String specifying the flux scheme to use. Either:
  - 'upwind', for the upwind flux;
  - 'centered', for a second-order centered flux;
  - 'maccormack', for the MacCormack flux.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.
- **physics_dynamics_coupling_on** (*bool*) – `True` to couple physics with dynamics, i.e., to account for the change over time in potential temperature, `False` otherwise.
- **sedimentation_on** (*bool*) – `True` to account for rain sedimentation, `False` otherwise.
- **sedimentation_flux_type** (*str*) – String specifying the method used to compute the numerical sedimentation flux. Available options are:
  - 'first_order_upwind', for the first-order upwind scheme;
  - 'second_order_upwind', for the second-order upwind scheme.

- **sedimentation_substeps** (*int*) – Number of sub-timesteps to perform in order to integrate the sedimentation flux.

Note: To instantiate an object of this class, one should prefer the static method *factory()* of *PrognosticIsentropic*.

_stencil_clipping_mass_fraction_and_diagnosing_isentropic_density_of_precipitation_wat

GT4Py stencil clipping the negative values for the mass fraction of precipitation water, and diagnosing the isentropic density of precipitation water.

Parameters

- **in_s** (*obj*) – gridtools.Equation representing the air isentropic density.
- **in_qr** (*obj*) – gridtools.Equation representing the mass fraction of precipitation water in air.

Returns

- **out_qr** (*obj*) – gridtools.Equation representing the clipped mass fraction of precipitation water.
- **out_Qr** (*obj*) – gridtools.Equation representing the isentropic density of precipitation water.

_stencil_computing_slow_tendencies_defs (*dt*, *in_s*, *in_s_prv*, *in_qr*, *in_qr_prv*)

GT4Py stencil computing the slow tendencies required to resolve rain sedimentation.

Parameters

- **dt** (*obj*) – gridtools.Global representing the large timestep.
- **in_s** (*obj*) – gridtools.Equation representing the current isentropic density.
- **in_s_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density.
- **in_qr** (*obj*) – gridtools.Equation representing the current mass fraction of precipitation water.
- **in_qr_prv** (*obj*) – gridtools.Equation representing the provisional mass fraction of precipitation water.

Returns

- **out_s_tnd** (*obj :*) – gridtools.Equation representing the slow tendency for the isentropic density.
- **out_qr_tnd** (*obj :*) – gridtools.Equation representing the slow tendency for the mass fraction of precipitation water.

_stencil_ensuring_vertical_cfl_is_obeyed_defs (*dts*, *in_h*, *in_vt*)

GT4Py stencil ensuring that the vertical CFL condition is fulfilled. This is achieved by clipping the raindrop fall velocity field: if a cell does not satisfy the CFL constraint, the vertical velocity at that cell is reduced so that the local CFL number equals 0.95.

Parameters

- **dts** (*obj*) – gridtools.Global representing the large timestep.
- **in_h** (*obj*) – gridtools.Equation representing the geometric height.
- **in_vt** (*obj*) – gridtools.Equation representing the raindrop fall velocity.

Returns `gridtools.Equation` representing the clipped raindrop fall velocity.

Return type obj

_stencil_stepping_by_coupling_physics_with_dynamics_defs(*dt*, *in_w*, *in_s*, *in_s_prv*, *in_U*, *in_U_prv*, *in_V*, *in_V_prv*, *in_Qv=None*, *in_Qv_prv=None*, *in_Qc=None*, *in_Qc_prv=None*, *in_Qr=None*, *in_Qr_prv=None*)

GT4Py stencil stepping the solution by coupling physics with dynamics, i.e., by accounting for the change over time in potential temperature.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_w** (*obj*) – `gridtools.Equation` representing the vertical velocity, i.e., the change over time in potential temperature.
- **in_s** (*obj*) – `gridtools.Equation` representing the current isentropic density.
- **in_s_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density.
- **in_U** (*obj*) – `gridtools.Equation` representing the current x-momentum.
- **in_U_prv** (*obj*) – `gridtools.Equation` representing the provisional x-momentum.
- **in_V** (*obj*) – `gridtools.Equation` representing the current y-momentum.
- **in_V_prv** (*obj*) – `gridtools.Equation` representing the provisional y-momentum.
- **in_Qv** (*obj*, optional) – `gridtools.Equation` representing the current isentropic density of water vapor.
- **in_Qv_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional isentropic density of water vapor.
- **in_Qc** (*obj*, optional) – `gridtools.Equation` representing the current isentropic density of cloud liquid water.
- **in_Qc_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional isentropic density of cloud liquid water.
- **in_Qr** (*obj*, optional) – `gridtools.Equation` representing the current isentropic density of precipitation water.
- **in_Qr_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional isentropic density of precipitation water.

Returns

- **out_s** (*obj*) – `gridtools.Equation` representing the updated isentropic density.
- **out_U** (*obj*) – `gridtools.Equation` representing the updated x-momentum.
- **out_V** (*obj*) – `gridtools.Equation` representing the updated y-momentum.

- **out_Qv** (*obj*, optional) – `gridtools.Equation` representing the updated isentropic density of water vapor.
- **out_Qc** (*obj*, optional) – `gridtools.Equation` representing the updated isentropic density of cloud liquid water.
- **out_Qr** (*obj*, optional) – `gridtools.Equation` representing the updated isentropic density of precipitation water.

_stencil_stepping_by_integrating_sedimentation_flux_defs(*dts*, *in_rho*, *in_s*, *in_h*, *in_qr*, *in_vt*, *in_s_tnd*, *in_qr_tnd*)

GT4Py stencil stepping the isentropic density and the mass fraction of precipitation water by integrating the precipitation flux.

Parameters

- **dts** (*obj*) – `gridtools.Global` representing the small timestep.
- **in_rho** (*obj*) – `gridtools.Equation` representing the air density.
- **in_s** (*obj*) – `gridtools.Equation` representing the air isentropic density.
- **in_h** (*obj*) – `gridtools.Equation` representing the geometric height of the model half-levels.
- **in_qr** (*obj*) – `gridtools.Equation` representing the input mass fraction of precipitation water.
- **in_vt** (*obj*) – `gridtools.Equation` representing the raindrop fall velocity.
- **in_s_tnd** (*obj*) – `gridtools.Equation` representing the contribution from the slow tendencies for the isentropic density.
- **in_qr_tnd** (*obj*) – `gridtools.Equation` representing the contribution from the slow tendencies for the mass fraction of precipitation water.

Returns

- **out_s** (*obj*) – `gridtools.Equation` representing the output isentropic density.
- **out_qr** (*obj*) – `gridtools.Equation` representing the output mass fraction of precipitation water.

_stencil_stepping_by_neglecting_vertical_advection_first_defs(*dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_U*, *in_V*, *in_Qv=None*, *in_Qc=None*, *in_Qr=None*, *in_qv_tnd=None*, *in_qc_tnd=None*, *in_qr_tnd=None*)

GT4Py stencil stepping the isentropic density and the water constituents via the forward Euler scheme. Further, it computes provisional values for the momentums, i.e., it updates the momentums disregarding the forcing terms involving the Montgomery potential.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the time step.

- **in_s** (*obj*) – gridtools.Equation representing the isentropic density at the current time.
- **in_u** (*obj*) – gridtools.Equation representing the x-velocity at the current time.
- **in_v** (*obj*) – gridtools.Equation representing the y-velocity at the current time.
- **in_mtg** (*obj*) – gridtools.Equation representing the Montgomery potential at the current time.
- **in_U** (*obj*) – gridtools.Equation representing the x-momentum at the current time.
- **in_V** (*obj*) – gridtools.Equation representing the y-momentum at the current time.
- **in_Qv** (*obj*, optional) – gridtools.Equation representing the mass of water vapour at the current time.
- **in_Qc** (*obj*, optional) – gridtools.Equation representing the mass of cloud water at the current time.
- **in_Qr** (*obj*, optional) – gridtools.Equation representing the mass of precipitation water at the current time.
- **in_qv_tnd** (*obj*, optional) – gridtools.Equation representing the parameterized tendency of the mass fraction of water vapor.
- **in_qc_tnd** (*obj*, optional) – gridtools.Equation representing the parameterized tendency of the mass fraction of cloud liquid water.
- **in_qr_tnd** (*obj*, optional) – gridtools.Equation representing the parameterized tendency of the mass fraction of precipitation water.

Returns

- **out_s** (*obj*) – gridtools.Equation representing the stepped isentropic density.
- **out_U** (*obj*) – gridtools.Equation representing the provisional x-momentum.
- **out_V** (*obj*) – gridtools.Equation representing the provisional y-momentum.
- **out_Qv** (*obj*, optional) – gridtools.Equation representing the stepped mass of water vapour.
- **out_Qc** (*obj*, optional) – gridtools.Equation representing the stepped mass of cloud water.
- **out_Qr** (*obj*, optional) – gridtools.Equation representing the stepped mass of precipitation water.

_stencil_stepping_by_neglecting_vertical_advection_second_defs(*dt*, *in_s*, *in_mtg*, *in_U*, *in_V*)

GT4Py stencil stepping the momentums via a one-time-level scheme.

Parameters

- **dt** (*obj*) – gridtools.Global representing the time step.
- **in_s** (*obj*) – gridtools.Equation representing the stepped isentropic density.
- **in_mtg** (*obj*) – gridtools.Equation representing the Montgomery potential diagnosed from the stepped isentropic density.

- **in_U** (*obj*) – `gridtools.Equation` representing the provisional x-momentum.
- **in_V** (*obj*) – `gridtools.Equation` representing the provisional y-momentum.

Returns

- **out_U** (*obj*) – `gridtools.Equation` representing the stepped x-momentum.
- **out_V** (*obj*) – `gridtools.Equation` representing the stepped y-momentum.

_stencils_stepping_by_integrating_sedimentation_flux_initialize()
Initialize the GT4Py stencils in charge of stepping the mass fraction of precipitation water by integrating the sedimentation flux.

_stencils_stepping_by_neglecting_vertical_advection_allocate_temporaries(*mi*, *mj*)
Allocate the Numpy arrays which will store temporary fields to be shared between the stencils stepping the solution by neglecting vertical advection.

Parameters

- **mi** (*int*) – x-extent of an input array representing an x-unstaggered field.
- **mj** (*int*) – y-extent of an input array representing a y-unstaggered field.

_stencils_stepping_by_neglecting_vertical_advection_initialize(*state*, *tendencies*)
Initialize the GT4Py stencils implementing the forward Euler scheme to step the solution by neglecting vertical advection.

Parameters **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:

- air_isentropic_density (unstaggered).

step_integrating_sedimentation_flux(*dt*, *state_now*, *state_prv*, *diagnostics=None*)
Method advancing the mass fraction of precipitation water by taking the sedimentation into account. For the sake of numerical stability, a time-splitting strategy is pursued, i.e., sedimentation is resolved using a timestep which may be smaller than that specified by the user.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_density (unstaggered);
  - air_isentropic_density (unstaggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - height or height_on_interface_levels (z-staggered);
  - mass_fraction_of_precipitation_water_in air (unstaggered).
- **state_prv** (*obj*) – *StateIsentropic* representing the provisional state, i.e., the state stepped without taking the sedimentation flux into account. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - mass_fraction_of_precipitation_water_in air (unstaggered).

  This may be the output of either *step_neglecting_vertical_advection()* or *step_coupling_physics_with_dynamics()*.

- **diagnostics** (*obj*, optional) – *GridData* collecting the following diagnostics:
  - accumulated_precipitation (unstaggered, two-dimensional).

Returns

- **state_new** (*obj*) – *StateIsentropic* containing the following updated variables:
  - mass_fraction_of_precipitation_water_in air (unstaggered);
  - precipitation_water_isentropic_density (unstaggered).
- **diagnostics_out** (*obj*) – *GridData* collecting the output diagnostics, i.e.:
  - accumulated_precipitation (unstaggered, two-dimensional);
  - precipitation (unstaggered, two-dimensional).

step_neglecting_vertical_advection(*dt*, *state*, *state_old=None*, *tendencies=None*)

Method advancing the conservative, prognostic model variables one time step forward via the forward Euler method. Only horizontal derivates are considered; possible vertical derivatives are disregarded.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - x_momentum_isentropic (unstaggered);
  - y_momentum_isentropic (unstaggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (isentropic);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **state_old** (*obj*, optional) – *StateIsentropic* representing the old state. This is not actually used, yet it appears as default argument for compliancy with the class hierarchy interface.
- **tendencies** (*obj*, optional) – *GridData* storing the following tendencies:
  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).

  Default is `None`.

Returns

StateIsentropic containing the updated prognostic variables, i.e.,

- air_isentropic_density (unstaggered);
- x_momentum_isentropic (unstaggered);

- y_momentum_isentropic (unstaggered);
- water_vapor_isentropic_density (unstaggered, optional);
- cloud_liquid_water_isentropic_density (unstaggered, optional);
- precipitation_water_isentropic_density (unstaggered, optional).

Return type obj

class tasmania.dycore.prognostic_isentropic_nonconservative.**PrognosticIsentropicNonconserva**

Abstract base class whose derived classes implement different schemes to carry out the prognostic steps of the three-dimensional moist isentropic dynamical core. The nonconservative form of the governing equations is used.

__init__(*flux_scheme*, *grid*, *moist_on*, *backend*, *physics_dynamics_coupling_on*, *sedimentation_on*)
Constructor.

Parameters

- **flux_scheme** (*str*) – String specifying the flux scheme to use. Either:
  - 'centered', for a second-order centered flux.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.
- **physics_dynamics_coupling_on** (*bool*) – `True` to couple physics with dynamics, i.e., to account for the change over time in potential temperature, `False` otherwise.
- **sedimentation_on** (*bool*) – `True` to account for rain sedimentation, `False` otherwise.

_stencil_stepping_by_coupling_physics_with_dynamics_allocate_inputs()
Allocate the attributes which serve as inputs to the GT4Py stencil which step the solution by coupling physics with dynamics, i.e., accounting for the change over time in potential temperature.

_stencil_stepping_by_coupling_physics_with_dynamics_allocate_outputs()
Allocate the Numpy arrays which will store the solution updated by coupling physics with dynamics.

_stencil_stepping_by_coupling_physics_with_dynamics_defs(*dt*, *in_w*, *in_s_now*, *in_s_prv*, *in_u_now*, *in_u_prv*, *in_v_now*, *in_v_prv*, *qv_now=None*, *qv_prv=None*, *qc_now=None*, *qc_prv=None*, *qr_now=None*, *qr_prv=None*)

GT4Py stencil stepping the solution by coupling physics with dynamics, i.e., by accounting for the change over time in potential temperature. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_w** (*array_like*) – `numpy.ndarray` representing the vertical velocity, i.e., the change over time in potential temperature.
- **in_s_now** (*obj*) – `gridtools.Equation` representing the current isentropic density.
- **in_s_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density.
- **in_u_now** (*obj*) – `gridtools.Equation` representing the current x-velocity.
- **in_u_prv** (*obj*) – `gridtools.Equation` representing the provisional x-velocity.
- **in_v_now** (*obj*) – `gridtools.Equation` representing the current y-velocity.
- **in_v_prv** (*obj*) – `gridtools.Equation` representing the provisional y-velocity.
- **in_qv_now** (*obj*, optional) – `gridtools.Equation` representing the current mass fraction of water vapor.
- **in_qv_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional mass fraction of water vapor.
- **in_qc_now** (*obj*, optional) – `gridtools.Equation` representing the current mass fraction of cloud liquid water.
- **in_qc_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional mass fraction of cloud liquid water.
- **in_qr_now** (*obj*, optional) – `gridtools.Equation` representing the current mass fraction of precipitation water.
- **in_qr_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional mass fraction of precipitation water.

Returns

- **out_s** (*obj*) – `gridtools.Equation` representing the updated isentropic density.
- **out_u** (*obj*) – `gridtools.Equation` representing the updated x-velocity.
- **out_v** (*obj*) – `gridtools.Equation` representing the updated y-velocity.

- **out_qv** (*obj*, optional) – `gridtools.Equation` representing the updated mass fraction of water vapor.
- **out_qc** (*obj*, optional) – `gridtools.Equation` representing the updated mass fraction of cloud liquid water.
- **out_qr** (*obj*, optional) – `gridtools.Equation` representing the updated mass fraction of precipitation water.

_stencil_stepping_by_coupling_physics_with_dynamics_initialize(*state_now*)

Initialize the GT4Py stencil in charge of stepping the solution by coupling physics with dynamics, i.e., by accounting for the change over time in potential temperature.

Parameters **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:

- air_isentropic_density (unstaggered).

_stencil_stepping_by_coupling_physics_with_dynamics_set_inputs(*dt*, *state_now*, *state_prv*, *diagnostics*)

Update the attributes which serve as inputs to the GT4Py stencil which steps the solution by integrating the vertical advection, i.e., by accounting for the change over time in potential temperature.

Parameters

- **dt** (*obj*) – A `datetime.timedelta` representing the time step.
- **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **state_prv** (*obj*) – *StateIsentropic* representing the provisional state, i.e., the state stepped taking only the horizontal derivatives into account. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

  This may be the output of *step_neglecting_vertical_advection()*.
- **diagnostics** (*obj*) – *GridData* collecting the following variables:
  - change_over_time_in_air_potential_temperature (unstaggered).

_stencils_stepping_by_neglecting_vertical_advection_allocate_inputs(*mi*, *mj*)

Allocate the attributes which serve as inputs to the GT4Py stencils which step the solution disregarding the vertical advection.

Parameters

- **mi** (*int*) – x-extent of an input array representing an x-unstaggered field.
- **mj** (*int*) – y-extent of an input array representing a y-unstaggered field.

_stencils_stepping_by_neglecting_vertical_advection_allocate_outputs(*mi*, *mj*)

Allocate the Numpy arrays which will store the solution updated by neglecting the vertical advection.

Parameters

- **mi** (*int*) – x-extent of an output array representing an x-unstaggered field.
- **mj** (*int*) – y-extent of an output array representing a y-unstaggered field.

_stencils_stepping_by_neglecting_vertical_advection_set_inputs(*dt*, *state*)

Update the attributes which serve as inputs to the GT4Py stencils which step the solution disregarding the vertical advection.

Parameters

- **dt** (*obj*) – A `datetime.timedelta` representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - montgomery_potential (isentropic);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

boundary

Get the attribute implementing the horizontal boundary conditions. If this is set to `None`, a `ValueError` is thrown.

Returns Instance of the derived class of *HorizontalBoundary* implementing the horizontal boundary conditions.

Return type obj

diagnostic

Get the attribute implementing the diagnostic steps of the three-dimensional moist isentropic dynamical core. If this is set to `None`, a `ValueError` is thrown.

Returns *DiagnosticIsentropic* carrying out the diagnostic step of the three-dimensional moist isentropic dynamical core.

Return type obj

static factory(*time_scheme*, *flux_scheme*, *grid*, *moist_on*, *backend*, *physics_dynamics_coupling_on*, *sedimentation_on*)

Static method returning an instace of the derived class implementing the time stepping scheme specified by `time_scheme`, using the flux scheme specified by `flux_scheme`.

Parameters

- **time_scheme** (*str*) – String specifying the time stepping method to implement. Either:
  - 'centered', for a centered scheme.
- **flux_scheme** (*str*) – String specifying the scheme to use. Either:
  - 'centered', for a second-order centered flux.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.
- **backend** (*obj*) – `gridtools.Mode` specifying the backend for the GT4Py stencils.
- **physics_dynamics_coupling_on** (*bool*) – `True` to couple physics with dynamics, i.e., to account for the change over time in potential temperature, `False` otherwise.
- **sedimentation_on** (*bool*) – `True` to account for rain sedimentation, `False` otherwise.

Returns An instace of the derived class implementing the scheme specified by `scheme`.

Return type obj

microphysics

Get the attribute taking care of microphysics. If this is set to `None`, a `ValueError` is thrown.

Returns Instance of a derived class of either `TendencyMicrophysics` or *AdjustmentMicrophysics*.

Return type obj

nb

Get the number of lateral boundary layers.

Returns The number of lateral boundary layers.

Return type int

step_coupling_physics_with_dynamics(*dt*, *state_now*, *state_prv*, *diagnostics*)

Method advancing the prognostic model variables one time step forward by coupling physics with dynamics, i.e., by accounting for the change over time in potential temperature.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

- **state_prv** (*obj*) – *StateIsentropic* representing the provisional state, i.e., the state stepped taking only the horizontal derivatives into account. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

  This may be the output of *step_neglecting_vertical_advection()*.
- **diagnostics** (*obj*) – *GridData* collecting the following variables:
  - change_over_time_in_air_potential_temperature (unstaggered).

Returns

StateIsentropic containing the updated prognostic variables, i.e.,

- air_isentropic_density (unstaggered);
- x_velocity (x-staggered);
- y_velocity (y-staggered);
- mass_fraction_of_water_vapor_in_air (unstaggered, optional);
- mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
- mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

Return type obj

step_integrating_sedimentation_flux(*dt*, *state_now*, *state_prv*, *diagnostics=None*)

Method advancing the mass fraction of precipitation water by taking the sedimentation into account. For the sake of numerical stability, a time-splitting strategy is pursued, i.e., the sedimentation is integrated using a timestep which may be smaller than that specified by the user. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - height or height_on_interface_levels (z-staggered);
  - mass_fraction_of_precipitation_water_in air (unstaggered).
- **state_prv** (*obj*) – *StateIsentropic* representing the provisional state, i.e., the state stepped without taking the sedimentation flux into account. It should contain the following variables:
  - mass_fraction_of_precipitation_water_in_air (unstaggered).

  This may be the output of either *step_neglecting_vertical_advection()* or *step_coupling_physics_with_dynamics()*.

- **diagnostics** (*obj*, optional) – *GridData* collecting the following diagnostics:
  - accumulated_precipitation (unstaggered, two-dimensional);
  - precipitation (unstaggered, two-dimensional).

Returns

- **state_new** (*obj*) – *StateIsentropic* containing the following updated variables:
  - mass_fraction_of_precipitation_water_in air (unstaggered).
- **diagnostics_out** (*obj*) – *GridData* collecting the output diagnostics, i.e.:
  - accumulated_precipitation (unstaggered, two-dimensional);
  - precipitation (unstaggered, two-dimensional).

step_neglecting_vertical_advection (*dt*, *state*, *state_old=None*, *diagnostics=None*, *tendencies=None*)

Method advancing the prognostic model variables one time step forward. Only horizontal derivates are considered; possible vertical derivatives are disregarded. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (isentropic);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **state_old** (*obj*, optional) – *StateIsentropic* representing the old state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **diagnostics** (*obj*, optional) – *GridData* possibly storing diagnostics. For the time being, this is not actually used.
- **tendencies** (*obj*, optional) – *GridData* possibly storing tendencies. For the time being, this is not actually used.

Returns

StateIsentropic containing the updated prognostic variables, i.e.,

- air_isentropic_density (unstaggered);
- x_velocity (x-staggered);
- y_velocity (y-staggered);
- mass_fraction_of_water_vapor_in_air (unstaggered, optional);
- mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
- mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

Return type obj

class tasmania.dycore.prognostic_isentropic_nonconservative_centered.**PrognosticIsentropicNo**

This class inherits *PrognosticIsentropicNonconservative* to implement a centered time-integration scheme to carry out the prognostic step of the three-dimensional moist isentropic dynamical core. The nonconservative form of the governing equations, expressed using isentropic coordinates, is used.

Variables

- ***time_levels*** (*int*) – Number of time levels the scheme relies on.
- **steps** (*int*) – Number of steps the scheme entails.

__init__(*flux_scheme*, *grid*, *moist_on*, *backend*, *coupling_physics_dynamics_on*, *sedimentation_on*)
Constructor.

Parameters

- **flux_scheme** (*str*) – String specifying the flux scheme to use. Either:
  - 'upwind', for the upwind flux;
  - 'centered', for a second-order centered flux;
  - 'maccormack', for the MacCormack flux.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.
- **moist_on** (*bool*) – `True` for a moist dynamical core, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.
- **physics_dynamics_coupling_on** (*bool*) – `True` to couple physics with dynamics, i.e., to account for the change over time in potential temperature, `False` otherwise.
- **sedimentation_on** (*bool*) – `True` to account for rain sedimentation, `False` otherwise.

Note: To instantiate an object of this class, one should prefer the static method *factory()* of *PrognosticIsentropicNonconservative*.

_stencil_clipping_mass_fraction_and_diagnosing_isentropic_density_of_precipitation_wat

GT4Py stencil clipping the negative values for the mass fraction of precipitation water, and diagnosing the isentropic density of precipitation water.

Parameters

- **in_s** (*obj*) – gridtools.Equation representing the air isentropic density.
- **in_qr** (*obj*) – gridtools.Equation representing the mass fraction of precipitation water in air.

Returns

- **out_qr** (*obj*) – gridtools.Equation representing the clipped mass fraction of precipitation water.
- **out_Qr** (*obj*) – gridtools.Equation representing the isentropic density of precipitation water.

_stencil_computing_slow_tendencies_defs(*dt*, *in_s_old*, *in_s_prv*, *in_qr_old*, *in_qr_prv*)

GT4Py stencil computing the slow tendencies required to resolve rain sedimentation.

Parameters

- **dt** (*obj*) – gridtools.Global representing the large timestep.
- **in_s_old** (*obj*) – gridtools.Equation representing the old isentropic density.
- **in_s_prv** (*obj*) – gridtools.Equation representing the provisional isentropic density.
- **in_qr_old** (*obj*) – gridtools.Equation representing the old mass fraction of precipitation water.
- **in_qr_prv** (*obj*) – gridtools.Equation representing the provisional mass fraction of precipitation water.

Returns

- **out_s_tnd** (*obj :*) – gridtools.Equation representing the slow tendency for the isentropic density.
- **out_qr_tnd** (*obj :*) – gridtools.Equation representing the slow tendency for the mass fraction of precipitation water.

_stencil_stepping_by_coupling_physics_with_dynamics_defs(*dt*, *in_w*, *in_s_now*, *in_s_prv*, *in_u_now*, *in_u_prv*, *in_v_now*, *in_v_prv*, *qv_now=None*, *qv_prv=None*, *qc_now=None*, *qc_prv=None*, *qr_now=None*, *qr_prv=None*)

GT4Py stencil stepping the solution by coupling physics with dynamics, i.e., by accounting for the change over time in potential temperature. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_w** (*array_like*) – `numpy.ndarray` representing the vertical velocity, i.e., the change over time in potential temperature.
- **in_s_now** (*obj*) – `gridtools.Equation` representing the current isentropic density.
- **in_s_prv** (*obj*) – `gridtools.Equation` representing the provisional isentropic density.
- **in_u_now** (*obj*) – `gridtools.Equation` representing the current x-velocity.
- **in_u_prv** (*obj*) – `gridtools.Equation` representing the provisional x-velocity.
- **in_v_now** (*obj*) – `gridtools.Equation` representing the current y-velocity.
- **in_v_prv** (*obj*) – `gridtools.Equation` representing the provisional y-velocity.
- **in_qv_now** (*obj*, optional) – `gridtools.Equation` representing the current mass fraction of water vapor.
- **in_qv_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional mass fraction of water vapor.
- **in_qc_now** (*obj*, optional) – `gridtools.Equation` representing the current mass fraction of cloud liquid water.
- **in_qc_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional mass fraction of cloud liquid water.
- **in_qr_now** (*obj*, optional) – `gridtools.Equation` representing the current mass fraction of precipitation water.
- **in_qr_prv** (*obj*, optional) – `gridtools.Equation` representing the provisional mass fraction of precipitation water.

Returns

- **out_s** (*obj*) – `gridtools.Equation` representing the updated isentropic density.
- **out_u** (*obj*) – `gridtools.Equation` representing the updated x-velocity.
- **out_v** (*obj*) – `gridtools.Equation` representing the updated y-velocity.
- **out_qv** (*obj*, optional) – `gridtools.Equation` representing the updated mass fraction of water vapor.
- **out_qc** (*obj*, optional) – `gridtools.Equation` representing the updated mass fraction of cloud liquid water.
- **out_qr** (*obj*, optional) – `gridtools.Equation` representing the updated mass fraction of precipitation water.

_stencil_stepping_by_integrating_sedimentation_flux_defs(*dts*, *in_rho*, *in_s*, *in_h*, *in_qr*, *in_vt*, *in_s_tnd*, *in_qr_tnd*)

GT4Py stencil stepping the isentropic density and the mass fraction of precipitation water by integrating the precipitation flux.

Parameters

- **dts** (*obj*) – gridtools.Global representing the small timestep.
- **in_rho** (*obj*) – gridtools.Equation representing the air density.
- **in_s** (*obj*) – gridtools.Equation representing the air isentropic density.
- **in_h** (*obj*) – gridtools.Equation representing the geometric height of the model half-levels.
- **in_qr** (*obj*) – gridtools.Equation representing the input mass fraction of precipitation water.
- **in_vt** (*obj*) – gridtools.Equation representing the raindrop fall velocity.
- **in_s_tnd** (*obj*) – gridtools.Equation representing the contribution from the slow tendencies for the isentropic density.
- **in_qr_tnd** (*obj*) – gridtools.Equation representing the contribution from the slow tendencies for the mass fraction of precipitation water.

Returns

- **out_s** (*obj*) – gridtools.Equation representing the output isentropic density.
- **out_qr** (*obj*) – gridtools.Equation representing the output mass fraction of precipitation water.

_stencil_stepping_by_neglecting_vertical_advection_unstaggered_defs(*dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_s_old*, *in_qv=None*, *in_qc=None*, *in_qr=None*, *in_qv_old=None*, *in_qc_old=None*, *in_qr_old=None*)

GT4Py stencil advancing the isentropic density and, possibly, the water constituents via a centered time-integration scheme.

Parameters

- **dt** (*obj*) – gridtools.Global representing the time step.
- **in_s** (*obj*) – gridtools.Equation representing the isentropic density at the current time level.
- **in_u** (*obj*) – gridtools.Equation representing the x-velocity at the current time level.
- **in_v** (*obj*) – gridtools.Equation representing the y-velocity at the current time level.
- **in_s** – gridtools.Equation representing the Montgomery potential at the current time level.
- **in_s_old** (*obj*) – gridtools.Equation representing the isentropic density at the previous time level.

- **in_qv** (*obj*, optional) – `gridtools.Equation` representing the mass fraction of water vapor at the current time level.
- **in_qc** (*obj*, optional) – `gridtools.Equation` representing the mass fraction of cloud liquid water at the current time level.
- **in_qr** (*obj*, optional) – `gridtools.Equation` representing the mass fraction of precipitation water at the current time level.
- **in_qv_old** (*obj*, optional) – `gridtools.Equation` representing the mass fraction of water vapor at the previous time level.
- **in_qc_old** (*obj*, optional) – `gridtools.Equation` representing the mass fraction of cloud liquid water at the previous time level.
- **in_qr_old** (*obj*, optional) – `gridtools.Equation` representing the mass fraction of precipitation water at the previous time level.

Returns

- **out_s** (*obj*) – `gridtools.Equation` representing the stepped isentropic density.
- **out_qv** (*obj*, optional) – `gridtools.Equation` representing the stepped mass fraction of water vapour.
- **out_qc** (*obj*, optional) – `gridtools.Equation` representing the stepped mass fraction of cloud liquid water.
- **out_qr** (*obj*, optional) – `gridtools.Equation` representing the stepped mass fraction of precipitation water.

_stencil_stepping_by_neglecting_vertical_advection_velocity_x_defs(*dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_u_old*)

GT4Py stencil advancing the x-velocity via a centered time-integration scheme.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density at the current time level.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity at the current time level.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity at the current time level.
- **in_mtg** (*obj*) – `gridtools.Equation` representing the Montgomery potential at the current time level.
- **in_u_old** (*obj*) – `gridtools.Equation` representing the x-velocity at the previous time level.

Returns `gridtools.Equation` representing the stepped x-velocity.

Return type obj

_stencil_stepping_by_neglecting_vertical_advection_velocity_y_defs(*dt*, *in_s*, *in_u*, *in_v*, *in_mtg*, *in_v_old*)

GT4Py stencil advancing the y-velocity via a centered time-integration scheme.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **in_s** (*obj*) – `gridtools.Equation` representing the isentropic density at the current time level.
- **in_u** (*obj*) – `gridtools.Equation` representing the x-velocity at the current time level.
- **in_v** (*obj*) – `gridtools.Equation` representing the y-velocity at the current time level.
- **in_mtg** (*obj*) – `gridtools.Equation` representing the Montgomery potential at the current time level.
- **in_v_old** (*obj*) – `gridtools.Equation` representing the y-velocity at the previous time level.

Returns `gridtools.Equation` representing the stepped y-velocity.

Return type obj

_stencils_stepping_by_integrating_sedimentation_flux_initialize()

Initialize the GT4Py stencils in charge of stepping the mass fraction of precipitation water by integrating the sedimentation flux.

_stencils_stepping_by_neglecting_vertical_advection_allocate_inputs(*mi*, *mj*)

Allocate the attributes which will serve as inputs to the GT4Py stencil stepping the solution by neglecting vertical advection.

Parameters

- **mi** (*int*) – x-extent of an input array representing an x-unstaggered field.
- **mj** (*int*) – y-extent of an input array representing a y-unstaggered field.

_stencils_stepping_by_neglecting_vertical_advection_initialize(*state*)

Initialize the GT4Py stencils implementing a time-integration centered scheme to step the solution by neglecting vertical advection.

Parameters **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:

- air_isentropic_density (unstaggered).

_stencils_stepping_by_neglecting_vertical_advection_set_inputs(*dt*, *state*, *state_old=None*)

Update the attributes which serve as inputs to the GT4Py stencils stepping the solution by neglecting vertical advection.

Parameters

- **dt** (*obj*) – A `datetime.timedelta` representing the time step.

- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - montgomery_potential (isentropic);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **state_old** (*obj*, optional) – *StateIsentropic* representing the old state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

step_integrating_sedimentation_flux(*dt*, *state_now*, *state_prv*, *diagnostics=None*)

Method advancing the mass fraction of precipitation water by taking the sedimentation into account. For the sake of numerical stability, a time-splitting strategy is pursued, i.e., the sedimentation flux is integrated using a timestep which may be smaller than that specified by the user.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state_now** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_density (unstaggered);
  - air_isentropic_density (unstaggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - height or height_on_interface_levels (z-staggered);
  - mass_fraction_of_precipitation_water_in air (unstaggered).
- **state_prv** (*obj*) – *StateIsentropic* representing the provisional state, i.e., the state stepped without taking the sedimentation flux into account. It should contain the following variables:
  - air_density (unstaggered);
  - air_isentropic_density (unstaggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - mass_fraction_of_precipitation_water_in air (unstaggered).

  This may be the output of either *step_neglecting_vertical_advection()* or *step_coupling_physics_with_dynamics()*.

- **diagnostics** (*obj*, optional) – *GridData* collecting the following diagnostics:
  - accumulated_precipitation (unstaggered, two-dimensional).

Returns

- **state_new** (*obj*) – *StateIsentropic* containing the following updated variables:
  - mass_fraction_of_precipitation_water_in air (unstaggered);
  - mass_fraction_of_precipitation_water_in_air (unstaggered).
- **diagnostics_out** (*obj*) – *GridData* collecting the output diagnostics, i.e.:
  - accumulated_precipitation (unstaggered, two-dimensional);
  - precipitation (unstaggered, two-dimensional).

step_neglecting_vertical_advection (*dt*, *state*, *state_old=None*, *diagnostics=None*, *tendencies=None*)

Method advancing the prognostic model variables one time step forward via a centered time-integration scheme. Only horizontal derivates are considered; possible vertical derivatives are disregarded.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **state** (*obj*) – *StateIsentropic* representing the current state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - air_pressure or air_pressure_on_interface_levels (z-staggered);
  - montgomery_potential (isentropic);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **state_old** (*obj*, optional) – *StateIsentropic* representing the old state. It should contain the following variables:
  - air_isentropic_density (unstaggered);
  - x_velocity (x-staggered);
  - y_velocity (y-staggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered, optional);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
  - mass_fraction_of_precipitation_water_in_air (unstaggered, optional).
- **diagnostics** (*obj*, optional) – *GridData* possibly storing diagnostics. For the time being, this is not actually used.
- **tendencies** (*obj*, optional) – *GridData* possibly storing tendencies. For the time being, this is not actually used.

Returns

StateIsentropic containing the updated prognostic variables, i.e.,

- air_isentropic_density (unstaggered);
- x_velocity (x-staggered);
- y_velocity (y-staggered);
- mass_fraction_of_water_vapor_in_air (unstaggered, optional);
- mass_fraction_of_cloud_liquid_water_in_air (unstaggered, optional);
- mass_fraction_of_precipitation_water_in_air (unstaggered, optional).

Return type obj

1.2.7 Sedimentation flux

class tasmania.dycore.flux_sedimentation.**FluxSedimentation**

Abstract base class whose derived classes discretize the vertical derivative of the sedimentation flux with different orders of accuracy.

static factory(*sedimentation_flux_type*)

Static method returning an instance of the derived class which discretizes the vertical derivative of the sedimentation flux with the desired level of accuracy.

Parameters **sedimentation_flux_type** (*str*) – String specifying the method used to compute the numerical sedimentation flux. Available options are:

- 'first_order_upwind', for the first-order upwind scheme;
- 'second_order_upwind', for the second-order upwind scheme.

Returns

Return type Instance of the derived class implementing the desired method.

get_vertical_derivative_of_sedimentation_flux(*i*, *j*, *k*, *in_rho*, *in_h*, *in_qr*, *in_vt*)

Get the vertical derivative of the sedimentation flux. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **i** (*obj*) – gridtools.Index representing the index running along the x-axis.
- **j** (*obj*) – gridtools.Index representing the index running along the y-axis.
- **k** (*obj*) – gridtools.Index representing the index running along the θ-axis.
- **in_rho** (*obj*) – gridtools.Equation representing the air density.
- **in_h** (*obj*) – gridtools.Equation representing the geometric height of the model half-levels.
- **in_qr** (*obj*) – gridtools.Equation representing the mass fraction of precipitation water in air.
- **in_vt** (*obj*) – gridtools.Equation representing the raindrop fall velocity.

Returns gridtools.Equation representing the vertical derivative of the sedimentation flux.

Return type obj

class tasmania.dycore.flux_sedimentation.**FluxSedimentationUpwindFirstOrder**

This class inherits SedimentationFlux to implement a standard, first-order accurate upwind method to discretize the vertical derivative of the sedimentation flux.

Variables ***nb*** (*int*) – Extent of the stencil in the upward vertical direction.

__init__()
Constructor.

Note: To instantiate an object of this class, one should prefer the static method `factory()` of `SedimentationFlux`.

---

get_vertical_derivative_of_sedimentation_flux(*i*, *j*, *k*, *in_rho*, *in_h*, *in_qr*, *in_vt*)
Get the vertical derivative of the sedimentation flux.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **in_rho** (*obj*) – `gridtools.Equation` representing the air density.
- **in_h** (*obj*) – `gridtools.Equation` representing the geometric height of the model half-levels.
- **in_qr** (*obj*) – `gridtools.Equation` representing the mass fraction of precipitation water in air.
- **in_vt** (*obj*) – `gridtools.Equation` representing the raindrop fall velocity.

Returns `gridtools.Equation` representing the vertical derivative of the sedimentation flux.

Return type obj

class tasmania.dycore.flux_sedimentation.**FluxSedimentationUpwindSecondOrder**
This class inherits `SedimentationFlux` to implement a second-order accurate upwind method to discretize the vertical derivative of the sedimentation flux.

Variables ***nb*** (*int*) – Extent of the stencil in the upward vertical direction.

__init__()
Constructor.

Note: To instantiate an object of this class, one should prefer the static method `factory()` of `SedimentationFlux`.

---

get_vertical_derivative_of_sedimentation_flux(*i*, *j*, *k*, *in_rho*, *in_h*, *in_qr*, *in_vt*)
Get the vertical derivative of the sedimentation flux.

Parameters

- **i** (*obj*) – `gridtools.Index` representing the index running along the x-axis.
- **j** (*obj*) – `gridtools.Index` representing the index running along the y-axis.
- **k** (*obj*) – `gridtools.Index` representing the index running along the θ-axis.
- **in_rho** (*obj*) – `gridtools.Equation` representing the air density.
- **in_h** (*obj*) – `gridtools.Equation` representing the geometric height of the model half-levels.

- **in_qr** (*obj*) – gridtools.Equation representing the mass fraction of precipitation water in air.
- **in_vt** (*obj*) – gridtools.Equation representing the raindrop fall velocity.

Returns gridtools.Equation representing the vertical derivative of the sedimentation flux.

Return type obj

1.2.8 Wave absorber

class tasmania.dycore.vertical_damping.**VerticalDamping**(*dims*, *grid*, *damp_depth*, *damp_max*, *backend*)

Abstract base class whose derived classes implement different vertical damping, i.e., wave absorbing, techniques.

__init__(*dims*, *grid*, *damp_depth*, *damp_max*, *backend*)

Constructor.

Parameters

- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply vertical damping.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **damp_depth** (*int*) – Number of vertical layers in the damping region.
- **damp_max** (*float*) – Maximum value for the damping coefficient.
- **backend** (*obj*) – gridtools.mode specifying the backend for the GT4Py stencils implementing the dynamical core.

apply(*dt*, *phi_now*, *phi_new*, *phi_ref*)

Apply vertical damping to a generic field ϕ. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **dt** (*obj*) – datetime.timedelta representing the time step.
- **phi_now** (*array_like*) – numpy.ndarray representing the field ϕ at the current time level.
- **phi_new** (*array_like*) – numpy.ndarray representing the field ϕ at the next time level, on which the absorber will be applied.
- **phi_ref** (*array_like*) – numpy.ndarray representing a reference value for ϕ.

Returns numpy.ndarray representing the damped field ϕ.

Return type array_like

static factory(*damp_type*, *dims*, *grid*, *damp_depth*, *damp_max*, *backend*)

Static method which returns an instance of the derived class implementing the damping method specified by damp_type.

Parameters

- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply vertical damping.

- **damp_type** (*str*) – String specifying the damper to implement. Either:
  - 'rayleigh', for a Rayleigh damper.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **damp_depth** (*int*) – Number of vertical layers in the damping region. Default is 15.
- **damp_max** (*float*) – Maximum value for the damping coefficient. Default is 0.0002.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils implementing the dynamical core.

Returns An instance of the derived class implementing the damping method specified by `damp_type`.

Return type obj

class tasmania.dycore.vertical_damping.**VerticalDampingRayleigh**(*dims*, *grid*, *damp_depth*, *damp_max*, *backend*)

This class inherits *VerticalDamping* to implement a Rayleigh absorber.

__init__(*dims*, *grid*, *damp_depth*, *damp_max*, *backend*)

Constructor.

Parameters

- **dims** (*tuple*) – Tuple of the dimension of the (three-dimensional) arrays on which to apply vertical damping.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **damp_depth** (*int*) – Number of vertical layers in the damping region.
- **damp_max** (*float*) – Maximum value for the damping coefficient.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils implementing the dynamical core.

_stencil_defs(*dt*, *phi_now*, *phi_new*, *phi_ref*, *R*)

The GT4Py stencil applying Rayleigh vertical damping.

Parameters

- **dt** (*obj*) – `gridtools.Global` representing the time step.
- **phi_now** (*ibj*) – `gridtools.Equation` representing the field ϕ at the current time level.
- **phi_new** (*obj*) – `gridtools.Equation` representing the field ϕ at the next time level, on which the absorber will be applied.
- **phi_ref** (*obj*) – `gridtools.Equation` representing a reference value for ϕ.
- **R** (*obj*) – `gridtools.Equation` representing the damping coefficient.

Returns `gridtools.Equation` representing the damped field ϕ.

Return type obj

_stencil_initialize()

Initialize the GT4Py stencil applying Rayleigh vertical damping.

Parameters **phi_now** (*array_like*) – numpy.ndarray representing the field ϕ at the current time level.

_stencil_set_inputs(*dt*, *phi_now*, *phi_new*, *phi_ref*)
Update the attributes which stores the stencil's input fields.

Parameters

- **dt** (*obj*) – datetime.timedelta representing the time step.
- **phi_now** (*array_like*) – numpy.ndarray representing the field ϕ at the current time level.
- **phi_new** (*array_like*) – numpy.ndarray representing the field ϕ at the next time level, on which the absorber will be applied.
- **phi_ref** (*array_like*) – numpy.ndarray representing a reference value for ϕ.

apply(*dt*, *phi_now*, *phi_new*, *phi_ref*)
Apply vertical damping to a generic field ϕ.

Parameters

- **dt** (*obj*) – datetime.timedelta representing the time step.
- **phi_now** (*array_like*) – numpy.ndarray representing the field ϕ at the current time level.
- **phi_new** (*array_like*) – numpy.ndarray representing the field ϕ at the next time level, on which the absorber will be applied.
- **phi_ref** (*array_like*) – numpy.ndarray representing a reference value for ϕ.

Returns numpy.ndarray representing the damped field ϕ.

Return type array_like

1.3 Grids

1.3.1 Two-dimensional grids

class tasmania.grids.grid_xy.**GridXY**(*domain_x*, *nx*, *domain_y*, *ny*, *units_x='degrees_east'*, *dims_x='longitude'*, *units_y='degrees_north'*, *dims_y='latitude'*)
Rectangular and regular two-dimensional grid embedded in a reference system whose coordinates are, in the order, x and y. No assumption is made on the nature of the coordinates. For instance, x may be the longitude, in which case $x \equiv \lambda$, and y may be the latitude, in which case $y \equiv \phi$.

Variables

- ***x*** (*obj*) – *Axis* storing the x-coordinates of the mass points.
- ***x_at_u_locations*** (*obj*) – *Axis* storing the x-coordinates at the u-locations.
- ***nx*** (*int*) – Number of grid points along x.
- ***dx*** (*float*) – The x-spacing.
- ***y*** (*obj*) – *Axis* storing the y-coordinates of the mass points.
- ***y_at_v_locations*** (*obj*) – *Axis* storing the y-coordinates at the v-locations.
- ***ny*** (*int*) – Number of grid points along y.

- **dy** (*float*) – The y-spacing.

__init__(*domain_x*, *nx*, *domain_y*, *ny*, *units_x='degrees_east'*, *dims_x='longitude'*, *units_y='degrees_north'*, *dims_y='latitude'*)

Constructor.

Parameters

- **domain_x** (*tuple*) – Tuple in the form $(x_{start},\ x_{stop})$.
- **nx** (*int*) – Number of grid points along x.
- **domain_y** (*tuple*) – Tuple in the form $(y_{start},\ y_{stop})$.
- **ny** (*int*) – Number of grid points along y.
- **units_x** (*str*, optional) – Units for the x-coordinate.
- **dims_x** (*str*, optional) – Label for the x-coordinate.
- **units_y** (*str*, optional) – Units for the y-coordinate.
- **dims_y** (*str*, optional) – Label for the y-coordinate.

Note: Axes labels should use the CF Conventions.

class tasmania.grids.grid_xz.**GridXZ**(*domain_x*, *nx*, *domain_z*, *nz*, *units_x='m'*, *dims_x='x'*, *units_z='m'*, *dims_z='z'*, *z_interface=None*, *topo_type='terrain_flat'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

Rectangular and regular two-dimensional grid embedded in a reference system whose coordinates are

- the horizontal coordinate x;
- the vertical (terrain-following) coordinate z.

The vertical coordinate z may be formulated to define a hybrid terrain-following coordinate system with terrain-following coordinate lines between the surface terrain-height and $z = z_F$, where z-coordinate lines change back to flat horizontal lines. However, no assumption is made on the actual nature of z which may be either pressure-based or height-based.

Variables

- ***x*** (*obj*) – *Axis* representing the x-axis.
- ***nx*** (*int*) – Number of grid points along x.
- ***dx*** (*float*) – The x-spacing.
- **z** (*obj*) – *Axis* representing the z-main levels.
- **z_on_interface_levels** (*obj*) – *Axis* representing the z-half levels.
- **nz** (*int*) – Number of vertical main levels.
- **dz** (*float*) – The z-spacing.
- **z_interface** (*float*) – The interface coordinate z_F.

Note: For the sake of compliancy with the COSMO model, the vertical grid points are ordered from the top of the domain to the surface.

__init__(*domain_x*, *nx*, *domain_z*, *nz*, *units_x='m'*, *dims_x='x'*, *units_z='m'*, *dims_z='z'*, *z_interface=None*, *topo_type='terrain_flat'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

Constructor.

Parameters

- **domain_x** (*tuple*) – Tuple in the form $(x_{left},\ x_{right})$.
- **nx** (*int*) – Number of grid points in the x-direction.
- **domain_z** (*tuple*) – Tuple in the form $(z_{top},\ z_{surface})$.
- **nz** (*int*) – Number of vertical main levels.
- **units_x** (*str*, optional) – Units for the x-coordinate. Must be compliant with the CF Conventions (see also *__init__()*).
- **dims_x** (*str*, optional) – Label for the x-coordinate.
- **units_z** (*str*, optional) – Units for the z-coordinate. Must be compliant with the CF Conventions (see also *__init__()*).
- **dims_z** (*str*, optional) – Label for the z-coordinate.
- **z_interface** (*float*, optional) – Interface value z_F. If not specified, it is assumed that $z_F = z_T$, with z_T the value of z at the top of the domain. In other words, a fully terrain-following coordinate system is supposed.
- **topo_type** (*str*, optional) – Topography type. See *topography* for further details.
- **topo_time** (*obj*, optional) – `datetime.timedelta` representing the simulation time after which the topography should stop increasing. Default is 0, corresponding to a time-invariant terrain surface-height. See *topography* for further details.

Keyword Arguments **kwargs** – Keyword arguments to be forwarded to the constructor of *Topography1d*.

topography_height

Get the topography (i.e., terrain-surface) height.

Returns One-dimensional `numpy.ndarray` representing the topography height.

Return type array_like

update_topography(*time*)

Update the (time-dependent) topography.

Parameters **time** (*obj*) – `datetime.timedelta` representing the elapsed simulation time.

class tasmania.grids.sigma.**Sigma2d**(*domain_x*, *nx*, *domain_z*, *nz*, *units_x='m'*, *dims_x='x'*, *z_interface=None*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

This class inherits *GridXZ* to represent a rectangular and regular two-dimensional grid embedded in a reference system whose coordinates are

- the horizontal coordinate x;
- the pressure-based terrain-following coordinate $\sigma = p/p_{SL}$, where p is the pressure and p_{SL} the pressure at the sea level.

The vertical coordinate σ may be formulated to define a hybrid terrain-following coordinate system with terrain-following coordinate lines between the surface terrain-height and $\sigma = \sigma_F$, where σ-coordinate lines change back to flat horizontal lines.

Variables

- **x** (*obj*) – *Axis* representing the x-axis.
- **nx** (*int*) – Number of grid points along x.
- **dx** (*float*) – The x-spacing.
- **z** (*obj*) – *Axis* representing the σ-main levels.
- **z_on_interface_levels** (*obj*) – *Axis* representing the σ-half levels.
- **nz** (*int*) – Number of vertical main levels.
- **dz** (*float*) – The σ-spacing.
- **z_interface** (*float*) – The interface coordinate σ_F.
- **height** (*obj*) – xarray.DataArray representing the geometric height of the main levels.
- **height_half_levels** (*obj*) – xarray.DataArray representing the geometric height of the half levels.
- **height_interface** (*float*) – Geometric height corresponding to $\sigma = \sigma_F$.
- **reference_pressure** (*obj*) – xarray.DataArray representing the reference pressure at the main levels.
- **reference_pressure_half_levels** (*obj*) – xarray.DataArray representing the reference pressure at the half levels.

__init__(*domain_x*, *nx*, *domain_z*, *nz*, *units_x='m'*, *dims_x='x'*, *z_interface=None*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

Constructor.

Parameters

- **domain_x** (*tuple*) – Tuple in the form $(x_{left},\ x_{right})$.
- **nx** (*int*) – Number of grid points in the x-direction.
- **domain_z** (*tuple*) – Tuple in the form $(\sigma_{top},\ \sigma_{surface})$.
- **nz** (*int*) – Number of vertical main levels.
- **units_x** (*str*, optional) – Units for the x-coordinate. Must be compliant with the CF Conventions (see also *__init__()*).
- **dims_x** (*str*, optional) – Label for the x-coordinate.
- **z_interface** (*float*, optional) – Interface value σ_F. If not specified, it is assumed that $\sigma_F = \sigma_T$, with σ_T the value of σ at the top of the domain. In other words, a fully terrain-following coordinate system is supposed.
- **topo_type** (*str*, optional) – Topography type. Default is 'flat_terrain'. See *topography* for further details.
- **topo_time** (*obj*, optional) – datetime.timedelta representing the simulation time after which the topography should stop increasing. Default is 0, corresponding to a time-invariant terrain surface-height. See *topography* for further details.

Keyword Arguments ****kwargs** – Keyword arguments to be forwarded to the constructor of *Topography1d*.

_update_metric_terms()

Update the class by computing the metric terms, i.e., the geometric height and the reference pressure, at both half and main levels. In doing this, a logarithmic vertical profile of reference pressure is assumed. This method should be called every time the topography is updated or changed.

plot(***kwargs*)
Plot the grid half levels using `matplotlib.pyplot`'s utilities.

Keyword Arguments ****kwargs** – Keyword arguments to be forwarded to `matplotlib.pyplot.subplots()`.

Note: For the sake of compliancy with the notation employed by COSMO, the vertical geometric height is denoted by z.

---

update_topography(*time*)
Update the (time-dependent) topography. In turn, the metric terms are re-computed.

Parameters **time** (*obj*) – `datetime.timedelta` representing the elapsed simulation time.

class tasmania.grids.gal_chen.**GalChen2d**(*domain_x*, *nx*, *domain_z*, *nz*, *units_x='m'*, *dims_x='x'*, *z_interface=None*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)
This class inherits `GridXZ` to represent a rectangular and regular two-dimensional grid embedded in a reference system whose coordinates are

- the horizontal coordinate x;
- the height-based Gal-Chen terrain-following coordinate μ.

The vertical coordinate μ may be formulated to define a hybrid terrain-following coordinate system with terrain-following coordinate lines between the surface terrain-height and $\mu = \mu_F$, where μ-coordinate lines change back to flat horizontal lines.

Variables

- ***x*** (*obj*) – *Axis* object representing the x-axis.
- ***nx*** (*int*) – Number of grid points along x.
- ***dx*** (*float*) – The x-spacing.
- **z** (*obj*) – *Axis* representing the μ-main levels.
- **z_on_interface_levels** (*obj*) – *Axis* representing the μ-half levels.
- **nz** (*int*) – Number of vertical main levels.
- **dz** (*float*) – The μ-spacing.
- **z_interface** (*float*) – The interface coordinate μ_F.
- **height** (*obj*) – `xarray.DataArray` representing the geometric height of the main levels.
- **height_half_levels** (*obj*) – `xarray.DataArray` representing the geometric height of the half levels.
- **height_interface** (*float*) – Geometric height corresponding to $\mu = \mu_F$.
- **reference_pressure** (*obj*) – `xarray.DataArray` representing the reference pressure at the main levels.
- **reference_pressure_half_levels** (*obj*) – `xarray.DataArray` representing the reference pressure at the half levels.

__init__(*domain_x*, *nx*, *domain_z*, *nz*, *units_x='m'*, *dims_x='x'*, *z_interface=None*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)
Constructor.

Parameters

- **domain_x** (*tuple*) – Tuple in the form $(x_{left},\ x_{right})$.
- **nx** (*int*) – Number of grid points in the x-direction.
- **domain_z** (*tuple*) – Tuple in the form $(\mu_{top},\ \mu_{surface})$.
- **nz** (*int*) – Number of vertical main levels.
- **units_x** (*str*, optional) – Units for the x-coordinate. Must be compliant with the CF Conventions (see also *tasmania.grids.axis.Axis.__init__()*).
- **dims_x** (*str*, optional) – Label for the x-coordinate.
- **z_interface** (*float*, optional) – Interface value μ_F. If not specified, it is assumed that $\mu_F = \mu_T$, with μ_T the value of μ at the top of the domain. In other words, a fully terrain-following coordinate system is supposed.
- **topo_type** (*str*, optional) – Topography type. Default is 'flat_terrain'. See *topography* for further details.
- **topo_time** (*obj*, optional) – `datetime.timedelta` representing the simulation time after which the topography should stop increasing. Default is 0, corresponding to a time-invariant terrain surface-height. See *topography* for further details.

Keyword Arguments ****kwargs** – Keyword arguments to be forwarded to the constructor of *Topography1d*.

_update_metric_terms()

Update the class by computing the metric terms, i.e., the geometric height and the reference pressure, at both half and main levels. In doing this, a logarithmic vertical profile of reference pressure is assumed. This method should be called every time the topography is updated or changed.

plot(***kwargs*)

Plot the grid half levels using `matplotlib.pyplot`'s utilities.

Keyword Arguments ****kwargs** – Keyword arguments to be forwarded to `matplotlib.pyplot.subplots()`.

Note: For the sake of compliancy with the notation employed by COSMO, the vertical geometric height is denoted by z.

---

update_topography(*time*)

Update the (time-dependent) topography. In turn, the metric terms are re-computed.

Parameters **time** (*obj*) – `datetime.timedelta` representing the elapsed simulation time.

class tasmania.grids.sleve.**SLEVE2d**(*domain_x*, *nx*, *domain_z*, *nz*, *units_x='m'*, *dims_x='x'*, *z_interface=None*, *N=100*, *s1=8000.0*, *s2=5000.0*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

This class inherits *GridXZ* to represent a rectangular and regular two-dimensional grid embedded in a reference system whose coordinates are

- the horizontal coordinate x;
- the height-based SLEVE terrain-following coordinate μ.

The vertical coordinate μ may be formulated to define a hybrid terrain-following coordinate system with terrain-following coordinate lines between the surface terrain-height and $\mu = \mu_F$, where μ-coordinate lines change back to flat horizontal lines.

Variables

- **x** (*obj*) – *Axis* representing the x-axis.
- **nx** (*int*) – Number of grid points along x.
- **dx** (*float*) – The x-spacing.
- **z** (*obj*) – *Axis* representing the μ-main levels.
- **z_on_interface_levels** (*obj*) – *Axis* representing the μ-half levels.
- **nz** (*int*) – Number of vertical main levels.
- **dz** (*float*) – The μ-spacing.
- **z_interface** (*float*) – The interface coordinate μ_F.
- **height** (*obj*) – xarray.DataArray representing the geometric height of the main levels.
- **height_half_levels** (*obj*) – xarray.DataArray representing the geometric height of the half levels.
- **height_interface** (*float*) – Geometric height corresponding to $\mu = \mu_F$.
- **reference_pressure** (*obj*) – xarray.DataArray representing the reference pressure at the main levels.
- **reference_pressure_half_levels** (*obj*) – xarray.DataArray representing the reference pressure at the half levels.

__init__(*domain_x*, *nx*, *domain_z*, *nz*, *units_x='m'*, *dims_x='x'*, *z_interface=None*, *N=100*, *s1=8000.0*, *s2=5000.0*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

Constructor.

Parameters

- **domain_x** (*tuple*) – Tuple in the form $(x_{left},\ x_{right})$.
- **nx** (*int*) – Number of grid points in the x-direction.
- **domain_z** (*tuple*) – Tuple in the form $(\mu_{top},\ \mu_{surface})$.
- **nz** (*int*) – Number of vertical main levels.
- **units_x** (*str*, optional) – Units for the x-coordinate. Must be compliant with the CF Conventions (see also *__init__()*).
- **dims_x** (*str*, optional) – Label for the x-coordinate.
- **z_interface** (*float*, optional) – Interface value μ_F. If not specified, it is assumed that $\mu_F = \mu_T$, with μ_T the value of μ at the top of the domain. In other words, a fully terrain-following coordinate system is supposed.
- **N** (*int*, optional) – Number of filter iterations performed to extract the large-scale component of the surface terrain-height. Defaults to 100.
- **s1** (*float*, optional) – Large-scale decay constant. Defaults to $8000\ m$.
- **s2** (*float*, optional) – Small-scale decay constant. Defaults to $5000\ m$.
- **topo_type** (*str*, optional) – Topography type. Defaults to 'flat_terrain'. See *topography* for further details.

- **topo_time** (*obj*, optional) – `datetime.timedelta` representing the simulation time after which the topography should stop increasing. Default is 0, corresponding to a time-invariant terrain surface-height. See *topography* for further details.

Keyword Arguments ****kwargs** – Keyword arguments to be forwarded to the constructor of *Topography1d*.

_update_metric_terms()
Update the class by computing the metric terms, i.e., the geometric height and the reference pressure, at both half and main levels. In doing this, a logarithmic vertical profile of reference pressure is assumed. This method should be called every time the topography is updated or changed.

plot(***kwargs*)
Plot the grid half levels using `matplotlib.pyplot`'s utilities.

Keyword Arguments ****kwargs** – Keyword arguments to be forwarded to `matplotlib.pyplot.subplots()`.

Note: For the sake of compliancy with the notation employed by COSMO, the vertical geometric height is denoted by z.

---

update_topography(*time*)
Update the (time-dependent) topography. In turn, the metric terms are re-computed.

Parameters **time** (*obj*) – `datetime.timedelta` representing the elasped simulation time.

1.3.2 Three-dimensional grids

class tasmania.grids.grid_xyz.**GridXYZ**(*domain_x*, *nx*, *domain_y*, *ny*, *domain_z*, *nz*, *units_x='degrees_east'*, *dims_x='longitude'*, *units_y='degrees_north'*, *dims_y='latitude'*, *units_z='m'*, *dims_z='z'*, *z_interface=None*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)
Rectangular and regular three-dimensional grid embedded in a reference system whose coordinates are

- the first horizontal coordinate x;
- the second horizontal coordinate y;
- the vertical (terrain-following) coordinate z.

The vertical coordinate z may be formulated to define a hybrid terrain-following coordinate system with terrain-following coordinate lines between the surface terrain-height and $z = z_F$, where z-coordinate lines change back to flat horizontal lines. However, no assumption is made on the actual nature of z which may be either pressure-based or height-based.

Variables

- **xy_grid** (*obj*) – *GridXY* representing the horizontal grid..
- **z** (*obj*) – *Axis* representing the z-main levels.
- **z_on_interface_levels** (*obj*) – *Axis* representing the z-half levels.
- **nz** (*int*) – Number of vertical main levels.
- **dz** (*float*) – The z-spacing.
- **z_interface** (*float*) – The interface coordinate z_F.

- ***topography*** (*obj*) – *Topography2d* representing the underlying topography.

Note: For the sake of compliancy with the COSMO model, the vertical grid points are ordered from the top of the domain to the surface.

__init__(*domain_x*, *nx*, *domain_y*, *ny*, *domain_z*, *nz*, *units_x='degrees_east'*, *dims_x='longitude'*, *units_y='degrees_north'*, *dims_y='latitude'*, *units_z='m'*, *dims_z='z'*, *z_interface=None*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

Constructor.

Parameters

- **domain_x** (*tuple*) – Tuple in the form $(x_{start},\ x_{stop})$.
- **nx** (*int*) – Number of grid points in the x-direction.
- **domain_y** (*tuple*) – Tuple in the form $(y_{start},\ y_{stop})$.
- **ny** (*int*) – Number of grid points in the y-direction.
- **domain_z** (*tuple*) – Tuple in the form $(z_{top},\ z_{surface})$.
- **nz** (*int*) – Number of vertical main levels.
- **units_x** (*str*, optional) – Units for the x-coordinate. Must be compliant with the CF Conventions.
- **dims_x** (*str*, optional) – Label for the x-coordinate.
- **units_y** (*str*, optional) – Units for the y-coordinate. Must be compliant with the CF Conventions.
- **dims_y** (*str*, optional) – Label for the y-coordinate.
- **units_z** (*str*, optional) – Units for the z-coordinate. Must be compliant with the CF Conventions.
- **dims_z** (*str*, optional) – Label for the z-coordinate.
- **z_interface** (*float*, optional) – Interface value z_F. If not specified, it is assumed that $z_F = z_T$, with z_T the value of z at the top of the domain. In other words, a fully terrain-following coordinate system is supposed.
- **topo_type** (*str*, optional) – Topography type. Default is 'flat_terrain'. See *topography* for further details.
- **topo_time** (*obj*, optional) – `datetime.timedelta` representing the simulation time after which the topography should stop increasing. Default is 0, corresponding to a time-invariant terrain surface-height. See *topography* for further details.

Keyword Arguments **kwargs** – Keyword arguments to be forwarded to the constructor of *Topography2d*.

dx

Get the x-spacing.

Returns The x-spacing.

Return type float

dy

Get the y-spacing.

Returns The y-spacing.

Return type float

nx
Get the number of grid points in the x-direction.

Returns Number of grid points in the x-direction.

Return type int

ny
Get the number of grid points in the y-direction.

Returns Number of grid points in the y-direction.

Return type int

topography_height
Get the topography (i.e., terrain-surface) height.

Returns Two-dimensional `numpy.ndarray` representing the topography height.

Return type array_like

update_topography(*time*)
Update the (time-dependent) topography.

Parameters **time** (*obj*) – `datetime.timedelta` representing the elapsed simulation time.

x
Get the x-coordinates of the mass points.

Returns `Axis` storing the x-coordinates of the mass points.

Return type obj

x_at_u_locations
Get the x-coordinates at the u-locations.

Returns `Axis` storing the x-coordinates at the u-locations.

Return type obj

y
Get the y-coordinates of the mass points.

Returns `Axis` storing the y-coordinates of the mass points.

Return type obj

y_at_v_locations
Get the y-coordinates at the v-locations.

Returns `Axis` storing the y-coordinates at the v-locations.

Return type obj

class tasmania.grids.sigma.**Sigma3d**(*domain_x*, *nx*, *domain_y*, *ny*, *domain_z*, *nz*, *units_x='degrees_east'*, *dims_x='longitude'*, *units_y='degrees_north'*, *dims_y='latitude'*, *z_interface=None*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)
This class inherits `GridXYZ` to represent a rectangular and regular computational grid embedded in a three-dimensional terrain-following reference system, whose coordinates are:

- first horizontal coordinate x, e.g., the longitude;
- second horizontal coordinate y, e.g., the latitude;

- the pressure-based terrain-following coordinate $\sigma = p/p_{SL}$, where p is the pressure and p_{SL} the pressure at the sea level.

The vertical coordinate σ may be formulated to define a hybrid terrain-following coordinate system with terrain-following coordinate lines between the surface terrain-height and $\sigma = \sigma_F$, where σ-coordinate lines change back to flat horizontal lines.

Variables

- **xy_grid** (*obj*) – *GridXY* representing the horizontal grid.
- **z** (*obj*) – *Axis* representing the σ-main levels.
- **z_on_interface_levels** (*obj*) – *Axis* representing the σ-half levels.
- **nz** (*int*) – Number of vertical main levels.
- **dz** (*float*) – The σ-spacing.
- **z_interface** (*float*) – The interface coordinate σ_F.
- ***topography*** (*obj*) – *Topography2d* representing the underlying topography.
- **height** (*obj*) – xarray.DataArray representing the geometric height of the main levels.
- **height_half_levels** (*obj*) – xarray.DataArray representing the geometric height of the half levels.
- **height_interface** (*float*) – Geometric height corresponding to $\sigma = \sigma_F$.
- **reference_pressure** (*obj*) – xarray.DataArray storing the reference pressure at the main levels.
- **reference_pressure_half_levels** (*obj*) – xarray.DataArray storing the reference pressure at the half levels.

__init__(*domain_x*, *nx*, *domain_y*, *ny*, *domain_z*, *nz*, *units_x='degrees_east'*, *dims_x='longitude'*, *units_y='degrees_north'*, *dims_y='latitude'*, *z_interface=None*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

Constructor.

Parameters

- **domain_x** (*tuple*) – Tuple in the form $(x_{left},\ x_{right})$.
- **nx** (*int*) – Number of grid points in the x-direction.
- **domain_y** (*tuple*) – Tuple in the form $(y_{left},\ y_{right})$.
- **ny** (*int*) – Number of grid points in the y-direction.
- **domain_z** (*tuple*) – Tuple in the form $(\sigma_{top},\ \sigma_{surface})$.
- **nz** (*int*) – Number of vertical main levels.
- **units_x** (*str*, optional) – Units for the x-coordinate. Must be compliant with the CF Conventions (see also *__init__()*).
- **dims_x** (*str*, optional) – Label for the x-coordinate.
- **units_y** (*str*, optional) – Units for the y-coordinate. Must be compliant with the CF Conventions (see also *__init__()*).
- **dims_y** (*str*, optional) – Label for the y-coordinate.

- **z_interface** (*float*, optional) – Interface value σ_F. If not specified, it is assumed that $\sigma_F = \sigma_T$, with σ_T the value of σ at the top of the domain. In other words, a fully terrain-following coordinate system is supposed.
- **topo_type** (*str*, optional) – Topography type. Default is 'flat_terrain'. See *topography* for further details.
- **topo_time** (*obj*, optional) – `datetime.timedelta` representing the simulation time after which the topography should stop increasing. Default is 0, corresponding to a time-invariant terrain surface-height. See *topography* for further details.

Keyword Arguments ****kwargs** – Keyword arguments to be forwarded to the constructor of *Topography2d*.

_update_metric_terms()

Update the class by computing the metric terms, i.e., the geometric height and the reference pressure, at both half and main levels. In doing this, a logarithmic vertical profile of reference pressure is assumed. This method should be called every time the topography is updated or changed.

update_topography(*time*)

Update the (time-dependent) topography. In turn, the metric terms are re-computed.

Parameters **time** (*obj*) – `datetime.timedelta` representing the elapsed simulation time.

class tasmania.grids.gal_chen.**GalChen3d**(*domain_x*, *nx*, *domain_y*, *ny*, *domain_z*, *nz*, *units_x='degrees_east'*, *dims_x='longitude'*, *units_y='degrees_north'*, *dims_y='latitude'*, *z_interface=None*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

This class inherits *GridXYZ* to represent a rectangular and regular computational grid embedded in a three-dimensional terrain-following reference system, whose coordinates are:

- first horizontal coordinate x, e.g., the longitude;
- second horizontal coordinate y, e.g., the latitude;
- the Gal-Chen terrain-following coordinate μ.

The vertical coordinate μ may be formulated to define a hybrid terrain-following coordinate system with terrain-following coordinate lines between the surface terrain-height and $\mu = \mu_F$, where μ-coordinate lines change back to flat horizontal lines.

Variables

- **xy_grid** (*obj*) – *GridXY* representing the horizontal grid.
- **z** (*obj*) – *Axis* representing the z-main levels.
- **z_on_interface_levels** (*obj*) – *Axis* representing the z-half levels.
- **nz** (*int*) – Number of vertical main levels.
- **dz** (*float*) – The z-spacing.
- **z_interface** (*float*) – The interface coordinate z_F.
- ***topography*** (*obj*) – *Topography2d* representing the underlying topography.
- **height** (*obj*) – `xarray.DataArray` representing the geometric height of the main levels.
- **height_half_levels** (*obj*) – `xarray.DataArray` representing the geometric height of the half levels.
- **height_interface** (*float*) – Geometric height corresponding to $\mu = \mu_F$.

- **reference_pressure** (*obj*) – xarray.DataArray representing the reference pressure at the main levels.
- **reference_pressure_half_levels** (*obj*) – xarray.DataArray representing the reference pressure at the half levels.

__init__(*domain_x*, *nx*, *domain_y*, *ny*, *domain_z*, *nz*, *units_x='degrees_east'*, *dims_x='longitude'*, *units_y='degrees_north'*, *dims_y='latitude'*, *z_interface=None*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

Constructor.

Parameters

- **domain_x** (*tuple*) – Tuple in the form $(x_{left},\ x_{right})$.
- **nx** (*int*) – Number of grid points in the x-direction.
- **domain_y** (*tuple*) – Tuple in the form $(y_{left},\ y_{right})$.
- **ny** (*int*) – Number of grid points in the y-direction.
- **domain_z** (*tuple*) – Tuple in the form $(\mu_{top},\ \mu_{surface})$.
- **nz** (*int*) – Number of vertical main levels.
- **units_x** (*str*, optional) – Units for the x-coordinate. Must be compliant with the CF Conventions (see also *tasmania.grids.axis.Axis.__init__()*).
- **dims_x** (*str*, optional) – Label for the x-coordinate.
- **str, optional** (*units_y*) – Units for the y-coordinate. Must be compliant with the CF Conventions (see also *tasmania.grids.axis.Axis.__init__()*).
- **dims_y** (*str*, optional) – Label for the y-coordinate.
- **z_interface** (*float*, optional) – Interface value μ_F. If not specified, it is assumed that $\mu_F = \mu_T$, with μ_T the value of μ at the top of the domain. In other words, a fully terrain-following coordinate nsystem is supposed.
- **topo_type** (*str*, optional) – Topography type. Default is 'flat_terrain'. See *topography* for further details.
- **topo_time** (*obj*, optional) – datetime.timedelta representing the simulation time after which the topography should stop increasing. Default is 0, corresponding to a time-invariant terrain surface-height. See *topography*] for further details.

Keyword Arguments ****kwargs** – Keyword arguments to be forwarded to the constructor of *Topography2d*.

_update_metric_terms()

Update the class by computing the metric terms, i.e., the geometric height and the reference pressure, at both half and main levels. In doing this, a logarithmic vertical profile of reference pressure is assumed. This method should be called every time the topography is updated or changed.

update_topography(*time*)

Update the (time-dependent) topography. In turn, the metric terms are re-computed.

Parameters **time** (*obj*) – datetime.timedelta representing the elapsed simulation time.

class tasmania.grids.sleve.**SLEVE3d**(*domain_x*, *nx*, *domain_y*, *ny*, *domain_z*, *nz*, *units_x='degrees_east'*, *dims_x='longitude'*, *units_y='degrees_north'*, *dims_y='latitude'*, *z_interface=None*, *N=100*, *s1=8000.0*, *s2=5000.0*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

This class inherits *GridXYZ* to represent a rectangular and regular computational grid embedded in a three-dimensional terrain-following reference system, whose coordinates are:

- first horizontal coordinate x, e.g., the longitude;
- second horizontal coordinate y, e.g., the latitude;
- the SLEVE terrain-following coordinate μ.

The vertical coordinate μ may be formulated to define a hybrid terrain-following coordinate system with terrain-following coordinate lines between the surface terrain-height and $\mu = \mu_F$, where μ-coordinate lines change back to flat horizontal lines.

Variables

- **xy_grid** (*obj*) – *GridXY* representing the horizontal grid.
- **z** (*obj*) – *Axis* representing the z-main levels.
- **z_on_interface_levels** (*obj*) – *Axis* representing the z-half levels.
- **nz** (*int*) – Number of vertical main levels.
- **dz** (*float*) – The z-spacing.
- **z_interface** (*float*) – The interface coordinate z_F.
- ***topography*** (*obj*) – *Topography2d* representing the underlying topography.
- **height** (*obj*) – xarray.DataArray representing the geometric height of the main levels.
- **height_half_levels** (*obj*) – xarray.DataArray representing the geometric height of the half levels.
- **height_interface** (*float*) – Geometric height corresponding to $\mu = \mu_F$.
- **reference_pressure** (*obj*) – xarray.DataArray representing the reference pressure at the main levels.
- **reference_pressure_half_levels** (*obj*) – xarray.DataArray representing the reference pressure at the half levels.

__init__(*domain_x*, *nx*, *domain_y*, *ny*, *domain_z*, *nz*, *units_x='degrees_east'*, *dims_x='longitude'*, *units_y='degrees_north'*, *dims_y='latitude'*, *z_interface=None*, *N=100*, *s1=8000.0*, *s2=5000.0*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

Constructor.

Parameters

- **domain_x** (*tuple*) – Tuple in the form $(x_{left},\ x_{right})$.
- **nx** (*int*) – Number of grid points in the x-direction.
- **domain_y** (*tuple*) – Tuple in the form $(y_{left},\ y_{right})$.
- **ny** (*int*) – Number of grid points in the y-direction.
- **domain_z** (*tuple*) – Tuple in the form $(\mu_{top},\ \mu_{surface})$.
- **nz** (*int*) – Number of vertical main levels.
- **units_x** (*str*, optional) – Units for the x-coordinate. Must be compliant with the CF Conventions (see also *__init__()*).
- **dims_x** (*str*, optional) – Label for the x-coordinate.

- **units_y** (*str*, optional) – Units for the y-coordinate. Must be compliant with the CF Conventions (see also *__init__()*).
- **dims_y** (*str*, optional) – Label for the y-coordinate.
- **z_interface** (*float*, optional) – Interface value μ_F. If not specified, it is assumed that $\mu_F = \mu_T$, with μ_T the value of μ at the top of the domain. In other words, a fully terrain-following coordinate nsystem is supposed.
- **N** (*int*, optional) – Number of filter iterations performed to determine the large-scale component of the surface terrain-height. Defaults to 100.
- **s1** (*float*, optional) – Large-scale decay constant. Defaults to $8000\ m$.
- **s2** (*float*, optional) – Small-scale decay constant. Defaults to $5000\ m$.
- **topo_type** (*str*, optional) – Topography type. Defaults to 'flat_terrain'. See *topography* for further details.
- **topo_time** (*obj*, optional) – `datetime.timedelta` representing the simulation time after which the topography should stop increasing. Default is 0, corresponding to a time-invariant terrain surface-height. See *topography* for further details.

Keyword Arguments ****kwargs** – Keyword arguments to be forwarded to the constructor of *Topography2d*.

_update_metric_terms()
Update the class by computing the metric terms, i.e., the geometric height and the reference pressure, at both half and main levels. In doing this, a logarithmic vertical profile of reference pressure is assumed. This method should be called every time the topography is updated or changed.

update_topography(*time*)
Update the (time-dependent) topography. In turn, the metric terms are re-computed.

Parameters **time** (*obj*) – `datetime.timedelta` representing the elapsed simulation time.

1.4 Model

class tasmania.model.**Model**
This class is intended to represent and run a generic climate or meteorological numerical model. A model is made up of:

- a dynamical core (mandatory);
- a set of parameterizations providing *slow-varying tendencies*, i.e., physical parameterization schemes which, within a timestep, are evaluated *before* the dynamical core; they are intended to supply the dynamical core with physical tendencies;
- a set of parameterizations providing *fast-varying tendencies*, i.e., physical parameterization schemes which, within a timestep, are evaluated *within* the dynamical core; they are intended to supply the dynamical core with physical tendencies;
- a set of *adjustment-performing* parameterizations, i.e., physical parametrization schemes which, within a timestep, are performed *after* the dynamical core; they are intended to perform physical adjustements on the state variables and possibly provide some diagnostics.

__call__(*dt*, *simulation_time*, *state*, *save_iterations=[]*)
Call operator integrating the model forward in time.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the time step.
- **simulation_time** (*obj*) – `datetime.timedelta` representing the simulation time.
- **state** (*obj*) – The initial state, as an instance of *GridData* or one of its derived classes.
- **save_freq** (*tuple*, optional) – The iterations at which the state should be saved. Default is empty, meaning that only the initial and final states are saved.

Returns

- **state_out** (*obj*) – The final state, of the same class of `state`.
- **state_save** (*obj*) – The sequence of saved states, of the same class of `state`.
- **diagnostics_save** (*obj*)

__init__()
Default constructor.

add_adjustment_parameterization(*adjustment*)
Add an *adjustment-performing* parameterization to the model.

Parameters **adjustment** (*obj*) – Instance of a derived class of *Adjustment* representing an adjustment-performing parameterization.

Note: In a simulation, adjustment-performing parameterizations will be executed in the same order they have been added to the model.

---

add_fast_tendency_parameterization(*tendency*)
Add to the model a parameterization providing fast-varying tendencies.

Parameters **tendency** (*obj*) – Instance of a derived class of *FastTendency* representing a parameterization providing fast-varying tendencies.

Note: In a simulation, parameterizations calculating fast-varying tendencies will be executed in the same order they have been added to the model.

---

add_slow_tendency_parameterization(*tendency*)
Add to the model a parameterization providing slow-varying tendencies.

Parameters **tendency** (*obj*) – Instance of a derived class of *SlowTendency* representing a parameterization providing slow-varying tendencies.

Note: In a simulation, parameterizations calculating slow-varying tendencies will be executed in the same order they have been added to the model.

---

set_dynamical_core(*dycore*)
Set the dynamical core.

Parameters **dycore** (*obj*) – Instance of a derived class of *DynamicalCore* representing the dynamical core.

1.5 Namelist

Configuration and global variables used throughout the package.

Physical constants:

- `p_ref`: Reference pressure ([Pa]).
- `p_sl`: Reference pressure at sea level ([Pa]).
- `T_sl`: Reference temperature at sea level ([K]).
- `beta`: Rate of increase in reference temperature with the logarithm of reference pressure ([$K\ Pa^{-1}$]).
- `Rd`: Gas constant for dry air ([$J\ K^{-1}\ Kg^{-1}$]).
- `Rv`: Gas constant for water vapor ([$J\ K^{-1}\ Kg^{-1}$]).
- `cp`: Specific heat of dry air at constant pressure ([$J\ K^{-1}\ Kg^{-1}$]).
- `g`: Mean gravitational acceleration ([$m\ s^{-2}$]).
- `L`: Specific latent heat of condensation of water ([$J\ kg^{-1}$]).
- `rho_water`: Water density ([$kg\ m^{-3}$]).

Grid settings:

- `domain_x`: Tuple storing the boundaries of the domain in the x-direction in the form (x_{west}, x_{east}).
- `nx`: Number of grid points in the x-direction.
- `domain_y`: Tuple storing the boundaries of the domain in the y-direction in the form (y_{south}, y_{north}).
- `ny`: Number of grid points in the y-direction.
- `domain_z`: Tuple storing the boundaries of the domain in the z-direction in the form (z_{top}, z_{bottom}).
- `nz`: Number of grid points in the z-direction.
- `z_interface`: For a hybrid coordinate system, interface level at which terrain-following z-coordinate lines get back to horizontal lines.
- `topo_type`: Topography type. Available options are:
  - 'flat_terrain';
  - 'gaussian';
  - 'schaer';
  - 'user_defined'.
- `topo_time`: `datetime.timedelta` object representing the elapsed simulation time after which the topography should stop increasing.
- `topo_max_height`: When `topo_type` is 'gaussian', maximum mountain height ([m]).
- `topo_width_x`: When `topo_type` is 'gaussian', mountain half-width in x-direction ([m]).
- `topo_width_y`: When `topo_type` is 'gaussian', mountain half-width in y-direction ([m]).
- `topo_str`: When `topo_type` is 'user_defined', terrain profile expression in the independent variables x and y. Must be fully C++-compliant.
- `topo_smooth`: `True` to smooth the topography out, `False` otherwise.
- `topo_kwargs`: Dictionary storing `topo_max_height`, `topo_width_x`, `topo_width_y`, `topo_str`, and `topo_smooth`.

Model settings:

- `model_name`: Name of the model to implement. Available options are:
  - 'isentropic_conservative', for the isentropic model based on the conservative form of the governing equations;
  - 'isentropic_nonconservative', for the isentropic model based on the nonconservative form of the governing equations.
- `moist_on`: `True` if water constituents should be taken into account, `False` otherwise.
- `horizontal_boundary_type`: Horizontal boundary conditions. Available options are:
  - 'periodic', for periodic boundary conditions;
  - 'relaxed', for relaxed boundary conditions.

Numerical settings:

- `time_scheme`: Time integration scheme to implement. Available options are:
  - 'forward_euler', for the forward Euler scheme ('isentropic_conservative');
  - 'centered', for a centered time-integration scheme ('isentropic_conservative', 'isentropic_nonconservative').
- `flux_scheme`: Numerical flux to use. Available options are:
  - 'upwind', for the upwind scheme ('isentropic_conservative');
  - 'centered', for a second-order centered scheme ('isentropic_conservative', 'isentropic_nonconservative');
  - 'maccormack', for the MacCormack scheme ('isentropic_conservative').
- **`damp_on`: `True` if (explicit) vertical damping should be applied,** `False` otherwise. Note that when vertical damping is switched off, the numerical diffusion coefficient is monotonically increased towards the top of the model, so to act as a diffusive wave absorber.
- `damp_type`: Type of vertical damping to apply. Available options are:
  - 'rayleigh', for Rayleigh vertical damping.
- `damp_depth`: Number of levels (either main levels or half levels) in the absorbing region.
- `damp_max`: Maximum value which should be assumed by the damping coefficient.
- `smooth_on`: `True` to enable numerical horizontal smoothing, `False` otherwise.
- `smooth_type`: Type of smoothing technique to implement. Available options are:
  - 'first_order', for first-order smoothing;
  - 'second_order', for second-order smoothing.
- `smooth_depth`: Number of levels (either main levels or half levels) in the smoothing absorbing region.
- `smooth_coeff`: The smoothing coefficient.
- `smooth_coeff_max`: Maximum value for the smoothing coefficient when smoothing vertical damping is enabled.
- `smooth_moist_on`: `True` to enable numerical horizontal smoothing on the moisture constituents, `False` otherwise.
- `smooth_moist_type`: Type of smoothing technique to apply on the moisture constituents. Available options are:

  - 'first_order', for first-order smoothing;
  - 'second_order', for second-order smoothing.

- `smooth_moist_depth`: Number of levels (either main levels or half levels) in the smoothing absorbing region for the moisture constituents.
- `smooth_coeff_moist`: The smoothing coefficient for the moisture components.
- `smooth_coeff_moist_max`: Maximum value for the smoothing coefficient for the moisture components when smoothing vertical damping is enabled.

Microphysics settings:

- `physics_dynamics_coupling_on`: `True` to couple physics with dynamics, i.e., to take the change over time in potential temperature into account, `False` otherwise.
- `sedimentation_on`: `True` to account for rain sedimentation, `False` otherwise.
- `sedimentation_flux_type`: String specifying the method used to compute the numerical sedimentation flux. Available options are:
  - 'first_order_upwind', for the first-order upwind scheme;
  - 'second_order_upwind', for the second-order upwind scheme.
- `sedimentation_substeps`: If rain sedimentation is switched on, number of sub-timesteps to perform in order to integrate the sedimentation flux.
- `rain_evaporation_on`: `True` to account for rain evaporation, `False` otherwise.
- `slow_tendency_microphysics_on`: `True` to include a parameterization scheme providing slow-varying cloud microphysical tendencies, `False` otherwise.
- `slow_tendency_microphysics_type`: The name of the parameterization scheme in charge of providing slow-varying cloud microphysical tendencies. Available options are:
  - 'kessler_wrf', for the WRF version of the Kessler scheme.
- `slow_tendency_microphysics_kwargs`: Keyword arguments for the parameterization scheme in charge of providing slow-varying cloud microphysical tendencies. Please see `slow_tendencies` for many more details.
- `fast_tendency_microphysics_on`: `True` to include a parameterization scheme providing fast-varying cloud microphysical tendencies, `False` otherwise.
- `fast_tendency_microphysics_type`: The name of the parameterization scheme in charge of providing fast-varying cloud microphysical tendencies. Available options are:
  - 'kessler_wrf', for the WRF version of the Kessler scheme.
- `fast_tendency_microphysics_kwargs`: Keyword arguments for the parameterization scheme in charge of providing fast-varying cloud microphysical tendencies. Please see `fast_tendencies` for many more details.
- `adjustment_microphysics_on`: `True` to include a parameterization scheme performing cloud microphysical adjustments, `False` otherwise.
- `adjustment_microphysics_type`: The name of the parameterization scheme in charge of performing cloud microphysical adjustments. Available options are:
  - 'kessler_wrf', for the WRF version of the Kessler scheme;
  - 'kessler_wrf_saturation', for the WRF version of the Kessler scheme, carrying out only the saturation adjustment.

- `adjustment_microphysics_kwargs`: Keyword arguments for the parameterization scheme in charge of performing cloud microphysical adjustments. Please see `adjustments` for many more details.

Simulation settings:

- `dt`: `datetime.timedelta` object representing the timestep.
- `initial_time`: `datetime.datetime` representing the initial simulation time.
- `simulation_time`: `datetime.timedelta` object representing the simulation time.
- `initial_state_type`: Integer identifying the initial state. See the documentation for the method `get_initial_state()` of `DycoreIsentropic`.
- `x_velocity_initial`: The initial, uniform x-velocity ($[ms^{-1}]$).
- `y_velocity_initial`: The initial, uniform y-velocity ($[ms^{-1}]$).
- `brunt_vaisala_initial`: The initial, uniform Brunt-Vaisala frequency.
- `temperature_initial`: The initial, uniform temperature ($[K]$).
- `initial_state_kwargs`: Dictionary storing `x_velocity_initial`, `y_velocity_initial`, `brunt_vaisala_initial`, and `temperature_initial`.
- `backend`: GT4Py backend to use. Available options are:
  - `gridtools.mode.NUMPY`: Numpy (i.e., vectorized) backend.
- `save_iterations`: List of the iterations at which the state should be saved.
- `save_dest`: Path to the location where results should be saved.
- `tol`: Tolerance used to compare floats (see `utils`).
- `datatype`: Datatype for `numpy.ndarray`. Either `np.float32` or `np.float64`.

1.6 Parameterizations

class tasmania.parameterizations.adjustments.**Adjustment**(*grid*)

Abstract base class whose derived classes implement different physical adjustment schemes. The hierarchy lays on top of `sympl.Implicit`.

Note: All the derived classes should be model-agnostic.

__call__(*state*, *dt*)

Entry-point method applying the parameterization scheme. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **state** (*obj*) – `GridData` or one of its derived classes representing the current state.
- **dt** (*obj*) – `datetime.timedelta` representing the timestep.

Returns

- **diagnostics** (*obj*) – `GridData` storing possible output diagnostics.
- **state_new** (*obj*) – `GridData` storing the output, adjusted state.

__init__(*grid*)
Constructor.

Parameters **grid**(*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.

time_levels
Get the attribute representing the number of time levels the dynamical core relies on.

Returns The number of time levels the dynamical core relies on.

Return type int

class tasmania.parameterizations.slow_tendencies.**SlowTendency**(*grid*)
Abstract base class whose derived classes implement different parameterization schemes providing slow-varying tendencies. Here, *slow-varying* refers to those tendencies which should be calculated on the largest model timestep. Hence, these classes should be ideally called *outside* the dynamical core.

Note: All the derived classes should be model-agnostic.

__call__(*state*, *dt*)
Entry-point method applying the parameterization scheme. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **state** (*obj*) – *GridData* or one of its derived classes representing the current state.
- **dt** (*obj*) – datetime.timedelta representing the timestep.

Returns

- **tendencies** (*obj*) – *GridData* storing the calculated tendencies.
- **diagnostics** (*obj*) – *GridData* storing possible output diagnostics.

__init__(*grid*)
Constructor.

Parameters **grid**(*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.

time_levels
Get the attribute representing the number of time levels the dynamical core relies on.

Returns The number of time levels the dynamical core relies on.

Return type int

class tasmania.parameterizations.fast_tendencies.**FastTendency**(*grid*)
Abstract base class whose derived classes implement different parameterization schemes providing fast-varying tendencies. Here, *fast-varying* refers to those tendencies which should be calculated on the smallest model timestep. Hence, these classes should be ideally called *within* the dynamical core.

Note: All the derived classes should be model-agnostic.

__call__(*dt*, *state*)
Entry-point method applying the parameterization scheme. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the timestep.
- **state** (*obj*) – *GridData* or one of its derived classes representing the current state.

Returns

- **tendencies** (*obj*) – *GridData* storing the output tendencies.
- **diagnostics** (*obj*) – *GridData* storing possible output diagnostics.

__init__(*grid*)
Constructor.

Parameters **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.

time_levels
Get the attribute representing the number of time levels the dynamical core relies on.

Returns The number of time levels the dynamical core relies on.

Return type int

1.6.1 Microphysics

class tasmania.parameterizations.adjustments.**AdjustmentMicrophysics**(*grid*, *rain_evaporation_on*, *backend*)

Abstract base class whose derived classes implement different parameterization schemes carrying out microphysical adjustments. The model variables which get adjusted are:

- the mass fraction of water vapor;
- the mass fraction of cloud liquid water;
- the mass fraction of precipitation water.

The derived classes also compute the following diagnostics:

- the raindrop fall speed ($[m\,s^{-1}]$).

__init__(*grid*, *rain_evaporation_on*, *backend*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **rain_evaporation_on** (*bool*) – `True` if the evaporation of raindrops should be taken into account, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.

static factory(*micro_scheme*, *grid*, *rain_evaporation_on*, *backend*, ***kwargs*)
Static method returning an instance of the derived class implementing the microphysics scheme specified by `micro_scheme`.

Parameters

- **micro_scheme** (*str*) – String specifying the microphysics parameterization scheme to implement. Either:
  - 'kessler_wrf', for the WRF version of the Kessler scheme;

- 'kessler_wrf_saturation', for the WRF version of the Kessler scheme, performing only the saturation adjustment.

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **rain_evaporation_on** (*bool*) – `True` if the evaporation of raindrops should be taken into account, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.
- ****kwargs** – Keyword arguments to be forwarded to the derived class.

get_raindrop_fall_velocity(*state*)

Get the raindrop fall velocity. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters **state** (*obj*) – *GridData* or one of its derived classes representing the current state. It should contain the following variables:

- air_density (unstaggered);
- mass_fraction_of_precipitation_water_in_air (unstaggered).

Returns `numpy.ndarray` representing the raindrop fall velocity.

Return type array_like

class tasmania.parameterizations.adjustment_microphysics_kessler_wrf.**AdjustmentMicrophysics**

This class inherits *AdjustmentMicrophysics* to implement the WRF version of the Kessler scheme.

Variables

- **inputs** (*tuple of str*) – The variables required in the input state when the object is called.
- **diagnostics** (*tuple of str*) – The diagnostic variables output by the object.
- **outputs** (*tuple of str*) – The variables which gets adjusted by the object.
- **input_properties** (*dict*) – A dictionary whose keys are variables required in the state when the object is called, and values are dictionaries which indicate 'dims' and 'units'.
- **diagnostic_properties** (*dict*) – A dictionary whose keys are diagnostics output by the object, and values are dictionaries which indicate 'dims' and 'units'.
- **output_properties** (*dict*) – A dictionary whose keys are variables adjusted when the object is called, and values are dictionaries which indicate 'dims' and 'units'.
- **a** (*float*) – Autoconversion threshold, in units of $[g\ kg^{-1}]$.
- **k1** (*float*) – Rate of autoconversion, in units of $[s^{-1}]$.
- **k2** (*float*) – Rate of autoaccretion, in units of $[s^{-1}]$.

__call__(*state*, *dt*)

Entry-point method performing microphysics adjustments.

Parameters

- **state** (*obj*) – *GridData* or one of its derived classes representing the current state. It should contain the following variables:
  - air_density (unstaggered);
  - air_pressure (unstaggered) or air_pressure_on_interface_levels (z-staggered);
  - exner_function (unstaggered) or exner_function_on_interface_levels (z-staggered);
  - air_temperature (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - mass_fraction_of_precipitation_water_in_air (unstaggered).
- **dt** (*obj*) – `datetime.timedelta` representing the timestep.

Returns

- **diagnostics** (*obj*) – Empty *GridData*, as no diagnostics are computed.
- **state_new** (*obj*) – *GridData* or one of its derived classes representing the adjusted state. It contains the following updated variables:
  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - mass_fraction_of_precipitation_water_in_air (unstaggered).

__init__ (*grid*, *rain_evaporation_on*, *backend*, *a=0.5*, *k1=0.001*, *k2=2.2*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **rain_evaporation_on** (*bool*) – `True` if the evaporation of raindrops should be taken into account, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.

Keyword Arguments

- **a** (*float*) – Autoconversion threshold, in units of [$g\ kg^{-1}$]. Default is $0.5\ g\ kg^{-1}$.
- **k1** (*float*) – Rate of autoconversion, in units of [s^{-1}]. Default is $0.001\ s^{-1}$.
- **k2** (*float*) – Rate of autoaccretion, in units of [s^{-1}]. Default is $2.2\ s^{-1}$.

Note: To instantiate this class, one should prefer the static method *factory()* of *AdjustmentMicrophysics*.

_stencil_adjustment_defs (*dt*, *in_rho*, *in_p*, *in_ps*, *in_exn*, *in_T*, *in_qv*, *in_qc*, *in_qr*)
GT4Py stencil carrying out the microphysical adjustments.

Parameters

- **in_rho** (*obj*) – `gridtools.Equation` representing the air density.
- **in_p** (*obj*) – `gridtools.Equation` representing the air pressure.
- **in_ps** (*obj*) – `gridtools.Equation` representing the saturation vapor pressure.

- **in_exn** (*obj*) – `gridtools.Equation` representing the Exner function.
- **in_T** (*obj*) – `gridtools.Equation` representing the air temperature.
- **in_qv** (*obj*) – `gridtools.Equation` representing the mass fraction of water vapor.
- **in_qc** (*obj*) – `gridtools.Equation` representing the mass fraction of cloud liquid water.
- **in_qr** (*obj*) – `gridtools.Equation` representing the mass fraction of precipitation water.

Returns

- **out_qv** (*obj*) – `gridtools.Equation` representing the adjusted mass fraction of water vapor.
- **out_qc** (*obj*) – `gridtools.Equation` representing the adjusted mass fraction of cloud liquid water.
- **out_qr** (*obj*) – `gridtools.Equation` representing the adjusted mass fraction of precipitation water.

References

Doms, G., et al. (2015). *A description of the nonhydrostatic regional COSMO-model. Part II: Physical parameterization.* Retrieved from COSMO. Mielikainen, J., B. Huang, J. Wang, H. L. A. Huang, and M. D. Goldberg. (2013). *Compute Unified Device Architecture (CUDA)-based parallelization of WRF Kessler cloud microphysics scheme*. Computer & Geosciences, 52:292-299.

_stencil_adjustment_initialize()
Initialize the GT4Py stencil in charge of carrying out the cloud microphysical adjustments.

_stencil_raindrop_fall_velocity_defs(*in_rho*, *in_rho_s*, *in_qr*)
GT4Py stencil calculating the raindrop velocity.

Parameters

- **in_rho** (*obj*) – `gridtools.Equation` representing the density.
- **in_rho_s** (*obj*) – `gridtools.Equation` representing the surface density.
- **in_qr** (*obj*) – `gridtools.Equation` representing the mass fraction of precipitation water.

Returns `gridtools.Equation` representing the raindrop fall velocity.

Return type obj

_stencil_raindrop_fall_velocity_initialize()
Initialize the GT4Py stencil calculating the raindrop velocity.

get_raindrop_fall_velocity(*state*)
Get the raindrop fall velocity.

Parameters **state** (*obj*) – *GridData* or one of its derived classes representing the current state. It should contain the following variables:

- air_density (unstaggered);
- mass_fraction_of_precipitation_water_in_air (unstaggered).

Returns `numpy.ndarray` representing the raindrop fall velocity.

Return type array_like

class tasmania.parameterizations.adjustment_microphysics_kessler_wrf_saturation.**AdjustmentM**

This class inherits *AdjustmentMicrophysics* to implement the saturation adjustment as predicted by the WRF version of the Kessler scheme.

Variables

- **inputs** (*tuple of str*) – The variables required in the input state when the object is called.
- **diagnostics** (*tuple of str*) – The diagnostic variables output by the object.
- **outputs** (*tuple of str*) – The variables which gets adjusted by the object.
- **input_properties** (*dict*) – A dictionary whose keys are variables required in the state when the object is called, and values are dictionaries which indicate 'dims' and 'units'.
- **diagnostic_properties** (*dict*) – A dictionary whose keys are diagnostics output by the object, and values are dictionaries which indicate 'dims' and 'units'.
- **output_properties** (*dict*) – A dictionary whose keys are variables adjusted when the object is called, and values are dictionaries which indicate 'dims' and 'units'.

__call__(*state*, *dt*)

Entry-point method performing the saturation adjustment.

Parameters

- **state** (*obj*) – *GridData* or one of its derived classes representing the current state. It should contain the following variables:
  - air_density (unstaggered);
  - air_pressure (unstaggered) or air_pressure_on_interface_levels (z-staggered);
  - exner_function (unstaggered) or exner_function_on_interface_levels (z-staggered);
  - air_temperature (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
- **dt** (*obj*) – `datetime.timedelta` representing the timestep.

Returns

- **diagnostics** (*obj*) – Empty *GridData*, as no diagnostics are computed.
- **state_new** (*obj*) – *GridData* or one of its derived classes representing the adjusted state. It contains the following updated variables:
  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);

__init__(*grid*, *rain_evaporation_on*, *backend*, ***kwargs*)

Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **rain_evaporation_on** (*bool*) – `True` if the evaporation of raindrops should be taken into account, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.

Note: To instantiate this class, one should prefer the static method *factory()* of *AdjustmentMicrophysics*.

_stencil_adjustment_defs(*dt*, *in_p*, *in_ps*, *in_exn*, *in_T*, *in_qv*, *in_qc*)
GT4Py stencil carrying out the saturation adjustment.

Parameters

- **in_p** (*obj*) – `gridtools.Equation` representing the air pressure.
- **in_ps** (*obj*) – `gridtools.Equation` representing the saturation vapor pressure.
- **in_exn** (*obj*) – `gridtools.Equation` representing the Exner function.
- **in_T** (*obj*) – `gridtools.Equation` representing the air temperature.
- **in_qv** (*obj*) – `gridtools.Equation` representing the mass fraction of water vapor.
- **in_qc** (*obj*) – `gridtools.Equation` representing the mass fraction of cloud liquid water.

Returns

- **out_qv** (*obj*) – `gridtools.Equation` representing the adjusted mass fraction of water vapor.
- **out_qc** (*obj*) – `gridtools.Equation` representing the adjusted mass fraction of cloud liquid water.

References

Doms, G., et al. (2015). *A description of the nonhydrostatic regional COSMO-model. Part II: Physical parameterization.* Retrieved from COSMO. Mielikainen, J., B. Huang, J. Wang, H. L. A. Huang, and M. D. Goldberg. (2013). *Compute Unified Device Architecture (CUDA)-based parallelization of WRF Kessler cloud microphysics scheme*. Computer & Geosciences, 52:292-299.

_stencil_adjustment_initialize()
Initialize the GT4Py stencil in charge of carrying out the saturation adjustment.

_stencil_raindrop_fall_velocity_defs(*in_rho*, *in_rho_s*, *in_qr*)
GT4Py stencil calculating the raindrop velocity.

Parameters

- **in_rho** (*obj*) – `gridtools.Equation` representing the density.
- **in_rho_s** (*obj*) – `gridtools.Equation` representing the surface density.
- **in_qr** (*obj*) – `gridtools.Equation` representing the mass fraction of precipitation water.

Returns `gridtools.Equation` representing the raindrop fall velocity.

Return type obj

_stencil_raindrop_fall_velocity_initialize()
Initialize the GT4Py stencil calculating the raindrop velocity.

get_raindrop_fall_velocity(*state*)
Get the raindrop fall velocity.

Parameters **state** (*obj*) – *GridData* or one of its derived classes representing the current state. It should contain the following variables:

- air_density (unstaggered);
- mass_fraction_of_precipitation_water_in_air (unstaggered).

Returns numpy.ndarray representing the raindrop fall velocity.

Return type array_like

class tasmania.parameterizations.slow_tendencies.**SlowTendencyMicrophysics**(*grid*, *rain_evaporation_on*, *backend*)
Abstract base class whose derived classes implement different parameterization schemes providing slow-varying microphysical tendencies. The derived classes also compute the following diagnostics:

- the raindrop fall speed ([$m\,s^{-1}$]).

__init__(*grid*, *rain_evaporation_on*, *backend*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **rain_evaporation_on** (*bool*) – True if the evaporation of raindrops should be taken into account, False otherwise.
- **backend** (*obj*) – gridtools.mode specifying the backend for the GT4Py stencils.

static factory(*micro_scheme*, *grid*, *rain_evaporation_on*, *backend*, ***kwargs*)
Static method returning an instance of the derived class implementing the microphysics scheme specified by micro_scheme.

Parameters

- **micro_scheme** (*str*) – String specifying the microphysics parameterization scheme to implement. Either:
  - 'kessler_wrf', for the WRF version of the Kessler scheme, with the calculated tendencies which do not include the source terms deriving from the saturation adjustment;
  - 'kessler_wrf'_saturation, for the WRF version of the Kessler scheme, with the calculated tendencies which do include the source terms deriving from the saturation adjustment.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **rain_evaporation_on** (*bool*) – True if the evaporation of raindrops should be taken into account, False otherwise.
- **backend** (*obj*) – gridtools.mode specifying the backend for the GT4Py stencils.
- ****kwargs** – Keyword arguments to be forwarded to the derived class.

get_raindrop_fall_velocity(*state*)
Get the raindrop fall velocity. As this method is marked as abstract, its implementation is delegated to the derived classes.

Parameters **state** (*obj*) – *GridData* or one of its derived classes representing the current state. It should contain the following variables:

- air_density (unstaggered);
- mass_fraction_of_precipitation_water_in_air (unstaggered).

Returns numpy.ndarray representing the raindrop fall velocity.

Return type array_like

class tasmania.parameterizations.slow_tendency_microphysics_kessler_wrf.**SlowTendencyMicroph**

This class inherits *SlowTendencyMicrophysics* to implement the WRF version of the Kessler scheme.

Note: The calculated tendencies do not include the source terms deriving from the saturation adjustment.

Variables

- **a** (*float*) – Autoconversion threshold, in units of $[g\ kg^{-1}]$.
- **k1** (*float*) – Rate of autoconversion, in units of $[s^{-1}]$.
- **k2** (*float*) – Rate of autoaccretion, in units of $[s^{-1}]$.

__call__(*dt*, *state*)
Entry-point method computing slow-varying cloud microphysics tendencies.

Parameters

- **dt** (*obj*) – datetime.timedelta representing the timestep.
- **state** (*obj*) – *GridData* or one of its derived classes representing the current state. It should contain the following variables:
  - air_density (unstaggered);
  - air_pressure (unstaggered) or air_pressure_on_interface_levels (z-staggered);
  - exner_function (unstaggered) or exner_function_on_interface_levels (z-staggered);
  - air_temperature (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - mass_fraction_of_precipitation_water_in_air (unstaggered).

Returns

- **tendencies** (*obj*) – *GridData* storing the following tendencies:
  - tendency_of_air_potential_temperature (unstaggered; only if rain evaporation is switched on);

  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered; only if rain evaporation is switched on);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).
- **diagnostics** (*obj*) – An empty `GridData`, since no diagnostic is computed.

__init__(*grid*, *rain_evaporation_on*, *backend*, *a=0.5*, *k1=0.001*, *k2=2.2*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of `GridXYZ` or one of its derived classes.
- **rain_evaporation_on** (*bool*) – `True` if the evaporation of raindrops should be taken into account, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.

Keyword Arguments

- **a** (*float*) – Autoconversion threshold, in units of $[g\ kg^{-1}]$. Default is $0.5\ g\ kg^{-1}$.
- **k1** (*float*) – Rate of autoconversion, in units of $[s^{-1}]$. Default is $0.001\ s^{-1}$.
- **k2** (*float*) – Rate of autoaccretion, in units of $[s^{-1}]$. Default is $2.2\ s^{-1}$.

Note: To instantiate this class, one should prefer the static method `factory()` of `SlowTendencyMicrophysics`.

_stencil_raindrop_fall_velocity_defs(*in_rho*, *in_rho_s*, *in_qr*)
GT4Py stencil calculating the raindrop velocity.

Parameters

- **in_rho** (*obj*) – `gridtools.Equation` representing the density.
- **in_rho_s** (*obj*) – `gridtools.Equation` representing the surface density.
- **in_qr** (*obj*) – `gridtools.Equation` representing the mass fraction of precipitation water.

Returns `gridtools.Equation` representing the raindrop fall velocity.

Return type obj

_stencil_raindrop_fall_velocity_initialize()
Initialize the GT4Py stencil calculating the raindrop velocity.

_stencil_tendency_defs(*dt*, *in_rho*, *in_p*, *in_ps*, *in_exn*, *in_T*, *in_qv*, *in_qc*, *in_qr*)
GT4Py stencil calculating the cloud microphysical tendencies.

Parameters

- **in_rho** (*obj*) – `gridtools.Equation` representing the air density.
- **in_p** (*obj*) – `gridtools.Equation` representing the air pressure.
- **in_ps** (*obj*) – `gridtools.Equation` representing the saturation vapor pressure.
- **in_exn** (*obj*) – `gridtools.Equation` representing the Exner function.
- **in_T** (*obj*) – `gridtools.Equation` representing the air temperature.

- **in_qv** (*obj*) – `gridtools.Equation` representing the mass fraction of water vapor.
- **in_qc** (*obj*) – `gridtools.Equation` representing the mass fraction of cloud liquid water.
- **in_qr** (*obj*) – `gridtools.Equation` representing the mass fraction of precipitation water.

Returns

- **out_qc_tnd** (*obj*) – `gridtools.Equation` representing the tendency of the mass fraction of cloud liquid water.
- **out_qr_tnd** (*obj*) – `gridtools.Equation` representing the tendency of the mass fraction of precipitation water.
- **out_w** (*obj*, optional) – `gridtools.Equation` representing the change over time in potential temperature.
- **out_qv_tnd** (*obj*, optional) – `gridtools.Equation` representing the tendency of the mass fraction of water vapor.

References

Doms, G., et al. (2015). *A description of the nonhydrostatic regional COSMO-model. Part II: Physical parameterization.* Retrieved from COSMO. Mielikainen, J., B. Huang, J. Wang, H. L. A. Huang, and M. D. Goldberg. (2013). *Compute Unified Device Architecture (CUDA)-based parallelization of WRF Kessler cloud microphysics scheme*. Computer & Geosciences, 52:292-299.

_stencil_tendency_initialize()
Initialize the GT4Py stencil in charge of calculating the cloud microphysical tendencies.

get_raindrop_fall_velocity(*state*)
Get the raindrop fall velocity.

Parameters **state** (*obj*) – *GridData* or one of its derived classes representing the current state. It should contain the following variables:

- air_density (unstaggered);
- mass_fraction_of_precipitation_water_in_air (unstaggered).

Returns `numpy.ndarray` representing the raindrop fall velocity.

Return type array_like

class tasmania.parameterizations.slow_tendency_microphysics_kessler_wrf_saturation.**SlowTend**

This class inherits *SlowTendencyMicrophysics* to implement the WRF version of the Kessler scheme.

Note: The calculated tendencies do include the source terms deriving from the saturation adjustment.

Variables

- **a** (*float*) – Autoconversion threshold, in units of $[g\ kg^{-1}]$.

- **k1** (*float*) – Rate of autoconversion, in units of $[s^{-1}]$.
- **k2** (*float*) – Rate of autoaccretion, in units of $[s^{-1}]$.

__call__ (*dt*, *state*)
Entry-point method computing slow-varying cloud microphysics tendencies.

Parameters

- **dt** (*obj*) – `datetime.timedelta` representing the timestep.
- **state** (*obj*) – *GridData* or one of its derived classes representing the current state. It should contain the following variables:
  - air_density (unstaggered);
  - air_pressure (unstaggered) or air_pressure_on_interface_levels (z-staggered);
  - exner_function (unstaggered) or exner_function_on_interface_levels (z-staggered);
  - air_temperature (unstaggered);
  - mass_fraction_of_water_vapor_in_air (unstaggered);
  - mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - mass_fraction_of_precipitation_water_in_air (unstaggered).

Returns

- **tendencies** (*obj*) – *GridData* storing the following tendencies:
  - tendency_of_air_potential_temperature (unstaggered; only if rain evaporation is switched on);
  - tendency_of_mass_fraction_of_water_vapor_in_air (unstaggered; only if rain evaporation is switched on);
  - tendency_of_mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
  - tendency_of_mass_fraction_of_precipitation_water_in_air (unstaggered).
- **diagnostics** (*obj*) – An empty *GridData*, since no diagnostic is computed.

__init__ (*grid*, *rain_evaporation_on*, *backend*, *a=0.5*, *k1=0.001*, *k2=2.2*)
Constructor.

Parameters

- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- **rain_evaporation_on** (*bool*) – `True` if the evaporation of raindrops should be taken into account, `False` otherwise.
- **backend** (*obj*) – `gridtools.mode` specifying the backend for the GT4Py stencils.

Keyword Arguments

- **a** (*float*) – Autoconversion threshold, in units of $[g\ kg^{-1}]$. Default is $0.5\ g\ kg^{-1}$.
- **k1** (*float*) – Rate of autoconversion, in units of $[s^{-1}]$. Default is $0.001\ s^{-1}$.
- **k2** (*float*) – Rate of autoaccretion, in units of $[s^{-1}]$. Default is $2.2\ s^{-1}$.

Note: To instantiate this class, one should prefer the static method *factory()* of *SlowTendencyMicrophysics*.

_stencil_raindrop_fall_velocity_defs(*in_rho*, *in_rho_s*, *in_qr*)
GT4Py stencil calculating the raindrop velocity.

Parameters

- **in_rho** (*obj*) – gridtools.Equation representing the density.
- **in_rho_s** (*obj*) – gridtools.Equation representing the surface density.
- **in_qr** (*obj*) – gridtools.Equation representing the mass fraction of precipitation water.

Returns gridtools.Equation representing the raindrop fall velocity.

Return type obj

_stencil_raindrop_fall_velocity_initialize()
Initialize the GT4Py stencil calculating the raindrop velocity.

_stencil_tendency_defs(*dt*, *in_rho*, *in_p*, *in_ps*, *in_exn*, *in_T*, *in_qv*, *in_qc*, *in_qr*)
GT4Py stencil calculating the cloud microphysical tendencies.

Parameters

- **in_rho** (*obj*) – gridtools.Equation representing the air density.
- **in_p** (*obj*) – gridtools.Equation representing the air pressure.
- **in_ps** (*obj*) – gridtools.Equation representing the saturation vapor pressure.
- **in_exn** (*obj*) – gridtools.Equation representing the Exner function.
- **in_T** (*obj*) – gridtools.Equation representing the air temperature.
- **in_qv** (*obj*) – gridtools.Equation representing the mass fraction of water vapor.
- **in_qc** (*obj*) – gridtools.Equation representing the mass fraction of cloud liquid water.
- **in_qr** (*obj*) – gridtools.Equation representing the mass fraction of precipitation water.

Returns

- **out_qv_tnd** (*obj*) – gridtools.Equation representing the tendency of the mass fraction of water vapor.
- **out_qc_tnd** (*obj*) – gridtools.Equation representing the tendency of the mass fraction of cloud liquid water.
- **out_qr_tnd** (*obj*) – gridtools.Equation representing the tendency of the mass fraction of precipitation water.
- **out_w** (*obj*) – gridtools.Equation representing the change over time in potential temperature.

References

Doms, G., et al. (2015). *A description of the nonhydrostatic regional COSMO-model. Part II: Physical parameterization.* Retrieved from COSMO. Mielikainen, J., B. Huang, J. Wang, H. L. A. Huang, and M. D. Goldberg. (2013). *Compute Unified Device Architecture (CUDA)-based parallelization of WRF Kessler cloud microphysics scheme*. Computer & Geosciences, 52:292-299.

_stencil_tendency_initialize()
Initialize the GT4Py stencil in charge of calculating the cloud microphysical tendencies.

get_raindrop_fall_velocity(*state*)
Get the raindrop fall velocity.

Parameters **state** (*obj*) – *GridData* or one of its derived classes representing the current state. It should contain the following variables:

- air_density (unstaggered);
- mass_fraction_of_precipitation_water_in_air (unstaggered).

Returns numpy.ndarray representing the raindrop fall velocity.

Return type array_like

1.7 Storages

class tasmania.storages.grid_data.**GridData**(*time*, *grid*, ***kwargs*)
Class storing and handling time-dependent variables defined on a grid. Ideally, this class should be used to represent the state, or a sequence of states at different time levels, of a *generic* climate or meteorological model. The model variables, in the shape of numpy.ndarrays, are passed to the constructor as keyword arguments. After conversion to xarray.DataArrays, the variables are packed in a dictionary whose keys are the input keywords. The class attribute units lists, for any admissible keyword, the units in which the associated field should be expressed. Any variable can be accessed via the accessor operator by specifying the corresponding keyword. Other methods are provided to update the state, or to create a sequence of states (useful for animation purposes). This class is designed to be as general as possible. Hence, it is not endowed with any method whose implementation depends on the variables actually stored by the class. This kind of methods might be provided by some derived classes, each one representing the state of a *specific* model.

Variables **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.

__getitem__(*key*)
Get a shallow copy of a gridded variable.

Parameters **key** (*str*) – The name of the variable to return.

Returns Shallow copy of the xarray.DataArray representing the variable, or None if the variable is not found.

Return type obj

__iadd__(*other*)
In-place addition operator. In detail:

- if the incoming object contains a variable which is not contained in the current object: the variable is deep-copied in the current object;
- if the incoming object contains a variable which is contained in the current object, but defined at a different time level: the internal variable is set equal to the incoming one;

- if the incoming object contains a variable which is contained in the current object and both are defined at the same time level: the internal variable is augmented by the incoming one.

Parameters **other** (*obj*) – Another *GridData*, or one of its derived classes.

Returns Updated version of the current object.

Return type obj

__init__ (*time*, *grid*, ***kwargs*)
Constructor.

Parameters

- **time** (*obj*) – datetime.datetime representing the time instant at which the variables are defined.
- **grid** (*obj*) – The underlying grid, as an instance of *GridXYZ* or one of its derived classes.
- ****kwargs** (*array_like*) – numpy.ndarray representing a gridded variable.

add_variables (*time*, ***kwargs*)
Add a list of variables, passed as keyword arguments.

Parameters

- **time** (*obj*) – datetime.datetime representing the time instant at which the variables are defined.
- ****kwargs** (*array_like*) – numpy.ndarray representing a gridded variable.

animation_profile_x (*field_to_plot*, *y_level*, *z_level*, *destination*, ***kwargs*)
Generate an animation showing a field along a section line orthogonal to the yz-plane.

Parameters

- **field_to_plot** (*str*) – The name of the field to plot.
- **y_level** (*int*) – y-index identifying the section line.
- **z_level** (*int*) – z-index identifying the section line.
- **destination** (*str*) – String specifying the path to the location where the movie will be saved. Note that the string should include the extension as well.
- ****kwargs** – Keyword arguments to specify different plotting settings. See *animation_profile_x()* for the complete list.

append (*other*)
Append a new state to the sequence of states.

Parameters **other** (*obj*) – Another *GridData* (or a derived class), whose xarray.DataArrays will be concatenated along the temporal axis to the corresponding ones in the current object.

Note: other is supposed to contain exactly the same variables stored by the current object.

---

extend (*other*)
Extend the current object by adding the variables stored by another object.

Notes

- The variables are deep copied.
- The incoming variables do not need to be defined at the same time level.

Parameters **other** (*obj*) – Another *GridData* (or a derived class) with which the current object will be synced.

extend_and_update(*other*)
Sync the current object with another *GridData* (or a derived class). This implies that, for each variable stored in the incoming object:

- if the current object contains a variable with the same name, that variable is updated;
- if the current object does not contain any variable with the same name, that variable is deep copied inside the current object.

Note: After the update, the stored variables might not be all defined at the same time level.

Parameters **other** (*obj*) – Another *GridData* (or a derived class) with which the current object will be synced.

get_max(*key*)
Get the maximum value of a variable.

Parameters **key** (*str*) – The name of the variable.

Returns The maximum value of the variable of interest.

Return type float

get_min(*key*)
Get the minimum value of a variable.

Parameters **key** (*str*) – The name of the variable.

Returns The minimum value of the variable of interest.

Return type float

pop(*key*)
Get a shallow copy of a gridded variable, then remove it from the dictionary.

Parameters **key** (*str*) – The name of the variable to return and remove.

Returns Shallow copy of the `xarray.DataArray` representing the variable, or `None` if the variable is not found.

Return type obj

time
Shortcut to the time instant at which the variables are defined.

Returns `datetime.timedelta` representing the time instant at which the variables are defined.

Return type obj

> **Warning:** Within an instance of this class, variables are not forced to be defined at the same time level, so the behaviour of this method might be undefined.

update(*other*)
Sync the current object with another *GridData* (or a derived class).

Notes

- It is assumed that *all* the variables stored by the input object are present also in the current object.
- After the update, the stored variables might not be all defined at the same time level.

Parameters **other** (*obj*) – Another *GridData* (or a derived class) with which the current object will be synced.

variable_names
Get the names of the stored variables.

Returns List of the names of the stored variables.

Return type list

class tasmania.storages.state_isentropic.**StateIsentropic**(*time*, *grid*, ***kwargs*)
This class inherits *GridData* to represent the state of the three-dimensional (moist) isentropic model. The stored variables might be:

- air_density (unstaggered);
- air_isentropic_density (unstaggered);
- x_velocity (x-staggered);
- y_velocity (y-staggered);
- x_momentum_isentropic (unstaggered);
- y_momentum_isentropic (unstaggered);
- air_pressure or air_pressure_on_interface_levels (z-staggered);
- exner_function or exner_function_on_interface_levels (z-staggered);
- montgomery_potential (unstaggered);
- height or height_on_interface_levels (z-staggered);
- air_temperature (unstaggered);
- mass_fraction_water_vapor_in_air (unstaggered);
- water_vapor_isentropic_density (unstaggered);
- mass_fraction_of_cloud_liquid_water_in_air (unstaggered);
- cloud_liquid_water_isentropic_density (unstaggered);
- mass_fraction_of_precipitation_water_in_air (unstaggered);
- precipitation_water_isentropic_density (unstaggered).

Note: At any point, an instance of this class may or may not contain all the listed model variables. Indeed, variables might be added gradually, according to user's needs.

Variables **grid** (*obj*) – *GridXYZ* representing the underlying grid.

__init__ (*time*, *grid*, ***kwargs*)
Constructor.

Parameters

- **time** (*obj*) – `datetime.timedelta` representing the time instant at which the state is defined.
- **grid** (*obj*) – *GridXYZ* representing the underlying grid.

Keyword Arguments

- **air_density** (*array_like*) – `numpy.ndarray` representing the density.
- **air_isentropic_density** (*array_like*) – `numpy.ndarray` representing the isentropic density.
- **x_velocity** (*array_like*) – `numpy.ndarray` representing the x-velocity.
- **y_velocity** (*array_like*) – `numpy.ndarray` representing the y-velocity.
- **x_momentum_isentropic** (*array_like*) – `numpy.ndarray` representing the (isentropic) x-momentum.
- **y_momentum_isentropic** (*array_like*) – `numpy.ndarray` representing the (isentropic) y-momentum.
- **or air_pressure_on_interface_levels** (*air_pressure*) – `numpy.ndarray` representing the pressure.
- **or exner_function_on_interface_levels** (*exner_function*) – `numpy.ndarray` representing the Exner function.
- **montgomery_potential** (*array_like*) – `numpy.ndarray` representing the Montgomery potential.
- **or height_on_interface_levels** (*height*) – `numpy.ndarray` representing the geometrical height of the half-levels.
- **air_temperature** (*array_like*) – `numpy.ndarray` representing the temperature.
- **mass_fraction_of_water_vapor_in_air** (*array_like*) – `numpy.ndarray` representing the mass fraction of water vapor.
- **water_vapor_isentropic_density** (*array_like*) – `numpy.ndarray` representing the isentropic density of water vapor.
- **mass_fraction_of_cloud_liquid_water_in_air** (*array_like*) – `numpy.ndarray` representing the mass fraction of cloud water.
- **cloud_liquid_water_isentropic_density** (*array_like*) – `numpy.ndarray` representing the isentropic density of cloud water.
- **mass_fraction_of_precipitation_water_in_air** (*array_like*) – `numpy.ndarray` representing the mass fraction of precipitation water.

- **precipitation_water_isentropic_density** (*array_like*) – numpy.ndarray representing the isentropic density of precipitation water.

animation_contourf_xz (*field_to_plot*, *y_level*, *destination*, ***kwargs*)

Generate an animation showing the time evolution of the contourfs of a field at a cross-section parallel to the xz-plane.

Parameters

- **field_to_plot** (*str*) – String specifying the field to plot. This might be:
  - the name of a variable stored in the current object.
- **y_level** (*int*) – y-index identifying the cross-section.
- **destination** (*str*) – String specifying the path to the location where the movie will be saved. Note that the string should include the extension as well.
- ****kwargs** – Keyword arguments to specify different plotting settings. See *animation_contourf_xz()* for the complete list.

contour_xz (*field_to_plot*, *y_level*, *time_level*, ***kwargs*)

Generate the contour plot of a field at a cross-section parallel to the xz-plane.

Parameters

- **field_to_plot** (*str*) – String specifying the field to plot. This might be:
  - the name of a variable stored in the current object;
  - 'x_velocity_unstaggered_perturbation', for the discrepancy of the x-velocity with respect to the initial condition; the current object must contain the following variables:
    - air_isentropic_density;
    - x_momentum_isentropic;
  - 'vertical_velocity', for the vertical velocity; only for two-dimensional steady-state flows; the current object must contain the following variables:
    - air_isentropic_density;
    - x_momentum_isentropic;
    - height or height_on_interface_levels.
- **y_level** (*int*) – y-index identifying the cross-section.
- **time_level** (*int*) – The time level.
- ****kwargs** – Keyword arguments to specify different plotting settings. See *contour_xz()* for the complete list.

contourf_xy (*field_to_plot*, *z_level*, *time_level*, ***kwargs*)

Generate the contourf plot of a field at a cross-section parallel to the xy-plane.

Parameters

- **field_to_plot** (*str*) – String specifying the field to plot. This might be:
  - the name of a variable stored in the current object;
  - 'horizontal_velocity', for the horizontal velocity; the current object must contain the following variables:
    - air_isentropic_density;
    - x_momentum_isentropic;

        * y_momentum_isentropic.
    - **z_level** (*int*) – z-index identifying the cross-section.
    - **time_level** (*int*) – The time level.
    - ****kwargs** – Keyword arguments to specify different plotting settings. See *contourf_xy()* for the complete list.

contourf_xz (*field_to_plot*, *y_level*, *time_level*, ***kwargs*)
Generate the contourf plot of a field at a cross-section parallel to the xz-plane.

Parameters

- **field_to_plot** (*str*) – String specifying the field to plot. This might be:
  - the name of a variable stored in the current object;
  - 'x_velocity_unstaggered_perturbation', for the discrepancy of the x-velocity with respect to the initial condition; the current object must contain the following variables:
    * air_isentropic_density;
    * x_momentum_isentropic;
  - 'vertical_velocity', for the vertical velocity; only for two-dimensional steady-state flows; the current object must contain the following variables:
    * air_isentropic_density;
    * x_momentum_isentropic;
    * height or height_on_interface_levels.
- **y_level** (*int*) – y-index identifying the cross-section.
- **time_level** (*int*) – The time level.
- ****kwargs** – Keyword arguments to specify different plotting settings. See *contourf_xz()* for the complete list.

get_cfl (*dt*)
Compute the CFL number.

Parameters **dt** (*obj*) – `datetime.timedelta` representing the time step.

Returns The CFL number.

Return type float

Note: If the CFL number exceeds the unity, i.e., if the CFL condition is violated, the method throws a warning.

---

quiver_xy (*field_to_plot*, *z_level*, *time_level*, ***kwargs*)
Generate the quiver plot of a vector field at a cross section parallel to the xy-plane.

Parameters

- **field_to_plot** (*str*) – String specifying the field to plot. This might be:
  - 'horizontal_velocity', for the horizontal velocity; the current object must contain the following variables:
    * air_isentropic_density;

        * x_momentum_isentropic;
        * y_momentum_isentropic.
    - **z_level** (*int*) – z-level identifying the cross-section.
    - **time_level** (*int*) – The time level.
    - ****kwargs** – Keyword arguments to specify different plotting settings. See *quiver_xy()* for the complete list.

quiver_xz (*field_to_plot*, *y_level*, *time_level*, ***kwargs*)
Generate the quiver plot of a vector field at a cross section parallel to the xz-plane.

Parameters

- **field_to_plot** (*str*) – String specifying the field to plot. This might be:
  - 'velocity', for the velocity field; the current object must contain the following variables:
    * air_isentropic_density;
    * x_momentum_isentropic;
    * height or height_on_interface_levels.
- **y_level** (*int*) – y-level identifying the cross-section.
- **time_level** (*int*) – The time level.
- ****kwargs** – Keyword arguments to specify different plotting settings. See *quiver_xz()* for the complete list.

streamplot_xz (*y_level*, *time_level*, ***kwargs*)
Generate the streamplot of a two-dimensional velocity field.

Note: The current object should contain the following variables:

- air_isentropic_density;
- x_momentum_isentropic;
- height or height_on_interface_levels.

Parameters

- **y_level** (*int*) – y-index identifying the cross-section.
- **time_level** (*int*) – The time level.
- ****kwargs** – Keyword arguments to specify different plotting settings. See *streamplot_xz()* for the complete list.

1.8 Topography

Classes representing one- and two-dimensional topographies, possibly time-dependent. Indeed, although clearly not physical, a terrain surface (slowly) growing in the early stages of a simulation may help to retrieve numerical stability, as it prevents steep gradients in the first few iterations.

Letting $h_s = h_s(x)$ be a one-dimensional topography, with $x \in [a, b]$, the user may choose among:

- a flat terrain, i.e., $h_s(x) \equiv 0$;
- a Gaussian-shaped mountain, i.e.,

$$h_s(x) = h_{max} \exp\left[-\left(\frac{x-c}{\sigma_x}\right)^2\right],$$

  where $c = 0.5(a+b)$.

For the two-dimensional case, letting $h_s = h_s(x,y)$ be the topography, with $x \in [a_x, b_x]$ and $y \in [a_y, b_y]$, the following profiles are provided:

- flat terrain, i.e., $h_s(x,y) \equiv 0$;
- Gaussian shaped-mountain, i.e.

$$h_s(x,y) = h_{max} \exp\left[-\left(\frac{x-c_x}{\sigma_x}\right)^2 - \left(\frac{y-c_y}{\sigma_y}\right)^2\right];$$

- modified Gaussian-shaped mountain proposed by Schaer and Durran (1997),

$$h_s(x,y) = \frac{h_{max}}{\left[1 + \left(\frac{x-c_x}{\sigma_x}\right)^2 + \left(\frac{y-c_y}{\sigma_y}\right)^2\right]^{3/2}}.$$

Yet, user-defined profiles are supported as well, provided that they admit an analytical expression. This is passed to the class as a string, which is then parsed in C++ via Cython (see `parser_1d` and `parser_2d`). Hence, the string itself must be fully C++-compliant.

References

Schaer, C., and Durran, D. R. (1997). *Vortex formation and vortex shedding in continuosly stratified flows past isolated topography*. Journal of Atmospheric Sciences, 54:534-554.

class tasmania.grids.topography.**Topography1d**(*x*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)

Class representing a one-dimensional topography.

Variables

- **topo** (*array_like*) – `xarray.DataArray` representing the topography (in meters).
- **topo_type** (*str*) – Topography type. Either:
  - 'flat_terrain';
  - 'gaussian';
  - 'user_defined'.
- **topo_time** (*obj*) – `datetime.timedelta` object representing the elapsed simulation time after which the topography should stop increasing.

- **topo_fact** (*float*) – Topography factor. It runs in between 0 (at the beginning of the simulation) and 1 (once the simulation has been run for `topo_time`).

__init__(*x*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)
Constructor.

Parameters

- **x** (*obj*) – *Axis* representing the underlying horizontal axis.
- **topo_type** (*str*, optional) – Topography type. Either:
  - 'flat_terrain' (default);
  - 'gaussian';
  - 'user_defined'.
- **topo_time** (*obj*) – class:*datetime.timedelta* representing the elapsed simulation time after which the topography should stop increasing. Default is 0, corresponding to a time-invariant terrain surface-height.

Keyword Arguments

- **topo_max_height** (*float*) – When `topo_type` is 'gaussian', maximum mountain height (in meters). Default is 500.
- **topo_width_x** (*float*) – When `topo_type` is 'gaussian', mountain half-width (in meters). Default is 10000.
- **topo_str** (*str*) – When `topo_type` is 'user_defined', terrain profile expression in the independent variable x. Must be fully C++-compliant.
- **topo_smooth** (*bool*) – `True` to smooth the topography out, `False` otherwise. Default is `False`.

update(*time*)
Update topography at current simulation time.

Parameters **time** (*obj*) – `datetime.timedelta` representing the elapsed simulation time.

class tasmania.grids.topography.**Topography2d**(*grid*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)
Class representing a two-dimensional topography.

Variables

- **topo** (*array_like*) – `xarray.DataArray` representing the topography (in meters).
- **topo_type** (*str*) – Topography type. Either:
  - 'flat_terrain';
  - 'gaussian';
  - 'schaer';
  - 'user_defined'.
- **topo_time** (*obj*) – `datetime.timedelta` representing the elapsed simulation time after which the topography should stop increasing.
- **topo_fact** (*float*) – Topography factor. It runs in between 0 (at the beginning of the simulation) and 1 (once the simulation has been run for `topo_time`).

- **topo_kwargs** (*dict*) – Dictionary storing all the topography settings which could be passed to the constructor as keyword arguments.

__init__(*grid*, *topo_type='flat_terrain'*, *topo_time=datetime.timedelta(0)*, ***kwargs*)
Constructor.

Parameters

- **grid** (*obj*) – *GridXY* representing the underlying grid.
- **topo_type** (*str*, optional) – Topography type. Either:
  - 'flat_terrain' (default);
  - 'gaussian';
  - 'schaer';
  - 'user_defined'.
- **topo_time** (*obj*) – `datetime.timedelta` representing the elapsed simulation time after which the topography should stop increasing. Default is 0, corresponding to a time-invariant terrain surface-height.

Keyword Arguments

- **topo_max_height** (*float*) – When `topo_type` is either 'gaussian' or 'schaer', maximum mountain height (in meters). Default is 500.
- **topo_center_x** (*float*) – When `topo_type` is either 'gaussian' or 'schaer', x-coordinate of the mountain center (in meters). By default, the mountain center is placed in the center of the domain.
- **topo_center_y** (*float*) – When `topo_type` is either 'gaussian' or 'schaer', y-coordinate of the mountain center (in meters). By default, the mountain center is placed in the center of the domain.
- **topo_width_x** (*float*) – When `topo_type` is either 'gaussian' or 'schaer', mountain half-width in x-direction (in meters). Default is 10000.
- **topo_width_y** (*float*) – When `topo_type` is either 'gaussian' or 'schaer', mountain half-width in y-direction (in meters). Default is 10000.
- **topo_str** (*str*) – When `topo_type` is 'user_defined', terrain profile expression in the independent variables x and y. Must be fully C++-compliant.
- **topo_smooth** (*bool*) – `True` to smooth the topography out, `False` otherwise. Default is `False`.

plot(*grid*, ***kwargs*)
Plot the topography using the mplot3d toolkit.

Parameters **grid** (*obj*) – *GridXY* representing the underlying grid.

Keyword Arguments ****kwargs** – Keyword arguments to be forwarded to `matplotlib.pyplot.figure()`.

update(*time*)
Update topography at current simulation time.

Parameters **time** (*obj*) – `datetime.timedelta` representing the elapsed simulation time.

1.8.1 Parsers

class tasmania.grids.parser.parser_1d.**Parser1d**
Cython wrapper for the C++ class parser_1d_cpp.

Variables **parser** (*obj*) – Pointer to a parser_1d_cpp object.

evaluate()
Evaluate the expression.

Returns numpy.ndarray of the evaluations.

class tasmania.grids.parser.parser_2d.**Parser2d**
Cython wrapper for the C++ class parser_2d_cpp.

Variables **parser** (*obj*) – Pointer to a parser_2d_cpp object.

evaluate()
Evaluate the expression.

Returns Two-dimensional numpy.ndarray of the evaluations.

1.9 Utilities

1.9.1 General-purpose utilities

Some useful utilities.

tasmania.utils.utils.**_get_prefix**(*units*)
Extract the prefix from the units name.

Parameters **units** (*str*) – The units.

Returns The prefix.

Return type str

tasmania.utils.utils.**convert_datetime64_to_datetime**(*time*)
Convert numpy.datetime64 to datetime.datetime.

Parameters **time** (*obj*) – The numpy.datetime64 object to convert.

Returns The converted datetime.datetime object.

Return type obj

References

https://stackoverflow.com/questions/13703720/converting-between-datetime-timestamp-and-datetime64.

tasmania.utils.utils.**equal_to**(*a*, *b*, *tol=None*)
Compare floating point numbers (or arrays of floating point numbers), properly accounting for round-off errors.

Parameters

- **a** (*float* or *array_like*) – Left-hand side.
- **b** (*float* or *array_like*) – Right-hand side.
- **tol** (*float*, optional) – Tolerance.

Returns `True` if `a` is equal to `b` up to `tol`, `False` otherwise.

Return type bool

`tasmania.utils.utils.`**`get_factor`**(*units*)
Convert units prefix to the corresponding factor. For the conversion, the CF Conventions are used.

Parameters **`units`** (*str*) – The units.

Returns The factor.

Return type float

`tasmania.utils.utils.`**`greater_or_equal_than`**(*a*, *b*, *tol=None*)
Compare floating point numbers (or arrays of floating point numbers), properly accounting for round-off errors.

Parameters

- **`a`** (*float* or *array_like*) – Left-hand side.
- **`b`** (*float* or *array_like*) – Right-hand side.
- **`tol`** (*float*, optional) – Tolerance.

Returns `True` if `a` is greater than or equal to `b` up to `tol`, `False` otherwise.

Return type bool

`tasmania.utils.utils.`**`greater_than`**(*a*, *b*, *tol=None*)
Compare floating point numbers (or arrays of floating point numbers), properly accounting for round-off errors.

Parameters

- **`a`** (*float* or *array_like*) – Left-hand side.
- **`b`** (*float* or *array_like*) – Right-hand side.
- **`tol`** (*float*, optional) – Tolerance.

Returns `True` if `a` is greater than `b` up to `tol`, `False` otherwise.

Return type bool

`tasmania.utils.utils.`**`smaller_or_equal_than`**(*a*, *b*, *tol=None*)
Compare floating point numbers (or arrays of floating point numbers), properly accounting for round-off errors.

Parameters

- **`a`** (*float* or *array_like*) – Left-hand side.
- **`b`** (*float* or *array_like*) – Right-hand side.
- **`tol`** (*float*, optional) – Tolerance.

Returns `True` if `a` is smaller than or equal to `b` up to `tol`, `False` otherwise.

Return type bool

`tasmania.utils.utils.`**`smaller_than`**(*a*, *b*, *tol=None*)
Compare floating point numbers (or arrays of floating point numbers), properly accounting for round-off errors.

Parameters

- **`a`** (*float* or *array_like*) – Left-hand side.
- **`b`** (*float* or *array_like*) – Right-hand side.
- **`tol`** (*float*, optional) – Tolerance.

Returns `True` if `a` is smaller than `b` up to `tol`, `False` otherwise.

Return type bool

tasmania.set_namelist.**set_namelist**(*user_namelist=None*)

Place the user-defined namelist module in the Python search path. This is achieved by physically copying the content of the user-provided module into TASMANIA_ROOT/namelist.py.

Parameters **user_namelist** (*str*) – Path to the user-defined namelist. If not specified, the default namelist TASMANIA_ROOT/_namelist.py is used.

1.9.2 Meteo utilities

Meteo-oriented utilities.

tasmania.utils.utils_meteo.**apply_goff_gratch_formula**(*T*)

Compute the saturation vapor pressure over water at a given temperature, relying upon the Goff-Gratch formula.

Parameters **T** (*array_like*) – numpy.ndarray representing the temperature ([K]).

Returns numpy.ndarray representing the saturation water vapor pressure ([Pa]).

Return type array_like

References

Goff, J. A., and S. Gratch. (1946). *Low-pressure properties of water from -160 to 212 F*. Transactions of the American Society of Heating and Ventilating Engineers, 95-122.

tasmania.utils.utils_meteo.**apply_teten_formula**(*T*)

Compute the saturation vapor pressure over water at a given temperature, relying upon the Teten's formula.

Parameters **T** (*array_like*) – numpy.ndarray representing the temperature ([K]).

Returns numpy.ndarray representing the saturation water vapor pressure ([Pa]).

Return type array_like

tasmania.utils.utils_meteo.**convert_relative_humidity_to_water_vapor**(*method*, *p*, *T*, *rh*)

Convert relative humidity to water vapor mixing ratio.

Parameters

- **method** (*str*) – String specifying the formula to be used to compute the saturation water vapor pressure. Either:
  - 'teten', for the Teten's formula;
  - 'goff_gratch', for the Goff-Gratch formula.
- **p** (*array_like*) – numpy.ndarray representing the pressure ([Pa]).
- **T** (*array_like*) – numpy.ndarray representing the temperature ([K]).
- **rh** (*array_like*) – numpy.ndarray representing the relative humidity ([$-$]).

Returns numpy.ndarray representing the fraction of water vapor ([$g\,g^{-1}$]).

Return type array_like

References

Vaisala, O. (2013). *Humidity conversion formulas: Calculation formulas for humidity*. Retrieved from https://www.vaisala.com.

tasmania.utils.utils_meteo.**get_isentropic_isothermal_analytical_solution**(*grid*, *x_velocity_initial*, *temperature*, *mountain_height*, *mountain_width*, *x_staggered=True*, *z_staggered=False*)

Get the analytical expression of a two-dimensional, hydrostatic, isentropic and isothermal flow over an isolated *Switch of Agnesi* mountain.

Parameters

- **grid** (*obj*) – *GridXYZ* representing the underlying grid. It must consist of only one points in y-direction.
- **x_velocity_initial** (*float*) – The initial x-velocity, in units of $[m\,s^{-1}]$.
- **temperature** (*float*) – The temperature, in units of $[K]$.
- **mountain_height** (*float*) – The maximum mountain height, in units of $[m]$.
- **mountain_width** (*float*) – The mountain half-width at half-height, in units of $[m]$.
- **x_staggered** (*bool*, optional) – `True` if the solution should be staggered in the x-direction, `False` otherwise. Default is `True`.
- **z_staggered** (*bool*, optional) – `True` if the solution should be staggered in the vertical direction, `False` otherwise. Default is `False`.

Returns

- **u** (*array_like*) – `numpy.ndarray` representing the x-velocity.
- **w** (*array_like*) – `numpy.ndarray` representing the vertical velocity.

References

Durran, D. R. (1981). *The effects of moisture on mountain lee waves*. Doctoral dissertation, Massachussets Institute of Technology.

1.9.3 Plotting utilities

Plotting utilities.

tasmania.utils.utils_plot.**_reverse_colormap**(*cmap*, *name=None*)

Reverse a Matplotlib colormap.

Parameters

- **cmap** (*obj*) – The matplotlib.colors.LinearSegmentedColormap to invert.
- **name** (*str*, optional) – The name of the reversed colormap. By default, this is obtained by appending '_r' to the name of the input colormap.

Returns The reversed matplotlib.colors.LinearSegmentedColormap.

Return type obj

References

https://stackoverflow.com/questions/3279560/invert-colormap-in-matplotlib.

tasmania.utils.utils_plot.**animation_contourf_xz**(*destination*, *time*, *x*, *z*, *field*, *topography*, ***kwargs*)

Generate an animation showing the time evolution of the contourfs of a field at a cross-section parallel to the xz-plane.

Parameters

- **destination** (*str*) – String specifying the path to the location where the movie will be saved. Note that the string should include the extension as well.
- **time** (*array_like*) – Array of datetime.datetimes representing the time instants of the frames.
- **x** (*array_like*) – Two-dimensional numpy.ndarray representing the underlying x-grid. This is assumed to be time-independent.
- **z** (*array_like*) – Three-dimensional numpy.ndarray representing the underlying z-grid. It is assumed that:
  - the first array axis represents x;
  - the second array axis represents the vertical coordinate;
  - the third array axis represents the time.
- **field** (*array_like*) – Three-dimensional numpy.ndarray representing the field to plot. It is assumed that:
  - the first array axis represents x;
  - the second array axis represents the vertical coordinate;
  - the third array axis represents the time.
- **topography** (*array_like*, optional) – Two-dimensional numpy.ndarray representing the underlying topography. It is assumed that:
  - the first array axis represents x;
  - the second array axis represents the time.

Keyword Arguments

- **fontsize** (*int*) – The fontsize to be used. Default is 12.
- **figsize** (*sequence*) – Sequence representing the figure size. Default is [8,8].
- **title** (*str*) – The figure title. Default is an empty string.
- **x_label** (*str*) – Label for the x-axis. Default is 'x'.
- **x_factor** (*float*) – Scaling factor for the x-axis. Default is 1.

- **x_lim** (*sequence*) – Sequence representing the interval of the x-axis to visualize. By default, the entire domain is shown.
- **z_label** (*str*) – Label for the z-axis. Default is 'z'.
- **z_factor** (*float*) – Scaling factor for the z-axis. Default is 1.
- **z_lim** (*sequence*) – Sequence representing the interval of the z-axis to visualize. By default, the entire domain is shown.
- **field_factor** (*float*) – Scaling factor for the field. Default is 1.
- **draw_z_isolines** (*bool*) – `True` to draw the z-isolines, `False` otherwise. Default is `True`.
- **cmap_name** (*str*) – Name of the Matplotlib's color map to be used. All the color maps provided by Matplotlib, as well as the corresponding inverted versions, are available.
- **cbar_levels** (*int*) – Number of levels for the color bar. Default is 14.
- **cbar_ticks_step** (*int*) – Distance between two consecutive labelled ticks of the color bar. Default is 1, i.e., all ticks are displayed with the corresponding label.
- **cbar_center** (*float*) – Center of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field over time.
- **cbar_half_width** (*float*) – Half-width of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field over time.
- **cbar_x_label** (*str*) – Label for the horizontal axis of the color bar. Default is an empty string.
- **cbar_y_label** (*str*) – Label for the vertical axis of the color bar. Default is an empty string.
- **cbar_orientation** (*str*) – Orientation of the color bar. Either 'vertical' (default) or 'horizontal'.
- **fps** (*int*) – Frames per second. Default is 15.
- **text** (*str*) – Text to be added to the figure as anchored text. By default, no extra text is shown.
- **text_loc** (*str*) – String specifying the location where the text box should be placed. Default is 'upper right'; please see `matplotlib.offsetbox.AnchoredText` for all the available options.

tasmania.utils.utils_plot.**animation_profile_x**(*time*, *x*, *field*, *destination*, ***kwargs*)

Generate an animation showing the time evolution of a field along a cross line orthogonal to the yz-plane.

Parameters

- **time** (*array_like*) – Array of :class:'datetime.datetime'~s representing the time instants of the frames.
- **x** (*array_like*) – One-dimensional `numpy.ndarray` representing the underlying x-grid.
- **field** (*array_like*) – Two-dimensional `numpy.ndarray` representing the field to plot. It is assumed that:
  - the first array axis represents x;

  - the second array axis represents the time.

- **destination** (*str*) – String specifying the path to the location where the movie will be saved. Note that the string should include the extension as well.

Keyword Arguments

- **fontsize** (*int*) – The fontsize to be used. Default is 12.
- **figsize** (*sequence*) – Sequence representing the figure size. Default is [8,8].
- **title** (*str*) – The figure title. Default is an empty string.
- **x_label** (*str*) – Label for the x-axis. Default is 'x'.
- **x_factor** (*float*) – Scaling factor for the x-axis. Default is 1.
- **x_lim** (*sequence*) – Sequence representing the interval of the x-axis to visualize. By default, the entire domain is shown.
- **y_label** (*str*) – Label for the y-axis. Default is 'y'.
- **y_factor** (*float*) – Scaling factor for the field. Default is 1.
- **y_lim** (*sequence*) – Sequence representing the interval of the y-axis to visualize. By default, the entire domain is shown.
- **color** (*str*) – String specifying the color line. Default is 'blue'.
- **linewidth** (*float*) – The linewidth. Default is 1.
- **grid_on** (*bool*) – `True` to draw the grid, `False` otherwise. Default is `True`.
- **fps** (*int*) – Frames per second. Default is 15.
- **text** (*str*) – Text to be added to the figure as anchored text. By default, no extra text is shown.
- **text_loc** (*str*) – String specifying the location where the text box should be placed. Default is 'upper right'; please see `matplotlib.offsetbox.AnchoredText` for all the available options.

`tasmania.utils.utils_plot.`**animation_profile_x_comparison**(*time*, *x*, *field*, *destination*, ***kwargs*)

Generate an animation showing the time evolution of one or more fields along a cross line orthogonal to the yz-plane.

Parameters

- **time** (*array_like*) – Array of :class:'datetime.datetime'~s representing the time instants of the frames.
- **x** (*list*) – Two-dimensional `numpy.ndarray` storing the x-grids underlying each field. It is assumed that:
  - the fields are concatenated along the first array axis;
  - the second array axis represents x.
- **field** (*array_like*) – Three-dimensional `numpy.ndarray` storing the fields to plot. It is assumed that:
  - the fields are concatenated along the first array axis;
  - the second array axis represents x;
  - the third array axis represents the time.

- **destination** (*str*) – String specifying the path to the location where the movie will be saved. Note that the string should include the extension as well.

Keyword Arguments

- **fontsize** (*int*) – The fontsize to be used. Default is 12.
- **figsize** (*sequence*) – Sequence representing the figure size. Default is [8,8].
- **title** (*str*) – The figure title. Default is an empty string.
- **x_label** (*str*) – Label for the x-axis. Default is 'x'.
- **x_factor** (*float*) – Scaling factor for the x-axis. Default is 1.
- **x_lim** (*sequence*) – Sequence representing the interval of the x-axis to visualize. By default, the entire domain is shown.
- **y_label** (*str*) – Label for the y-axis. Default is 'y'.
- **y_lim** (*sequence*) – Sequence representing the interval of the y-axis to visualize. By default, the entire domain is shown.
- **field_factor** (*list*) – List storing the scaling factor for each field. By default, all scaling factors are assumed to be 1.
- **color** (*list*) – List of strings specifying the line color for each field. The default sequence of colors is: 'blue', 'red', 'green', 'black'.
- **linestyle** (*list*) – List of strings specifying the line style for each field. The default line style is '-'.
- **linewidth** (*list*) – List of floats representing the line width for each field. The default line width is 1.
- **grid_on** (*bool*) – `True` to draw the grid, `False` otherwise. Default is `True`.
- **fps** (*int*) – Frames per second. Default is 15.
- **legend** (*list*) – List gathering the legend entries for each field. Default is 'field1', 'field2', etc.
- **legend_loc** (*str*) – String specifying the location where the legend box should be placed. Default is 'best'; please see `matplotlib.pyplot.legend()` for all the available options.

`tasmania.utils.utils_plot.`**contour_xz**(*x*, *z*, *field*, *topography*, ***kwargs*)

Generate the contour plot of a gridded field at a cross-section parallel to the xz-plane.

Parameters

- **x** (*array_like*) – Two-dimensional `numpy.ndarray` representing the underlying x-grid.
- **z** (*array_like*) – Two-dimensional `numpy.ndarray` representing the underlying z-grid.
- **field** (*array_like*) – Two-dimensional `numpy.ndarray` representing the field to plot.
- **topography** (*array_like*) – One-dimensional `numpy.ndarray` representing the underlying topography.

Keyword Arguments

- **show** (*bool*) – `True` if the plot should be showed, `False` otherwise. Default is `True`.

- **destination** (*str*) – String specifying the path to the location where the plot will be saved. Default is `None`, meaning that the plot will not be saved. Note that the plot may be saved only if `show` is set to `False`.
- **fontsize** (*int*) – The fontsize to be used. Default is 12.
- **figsize** (*sequence*) – Sequence representing the figure size. Default is [8,8].
- **title** (*str*) – The figure title. Default is an empty string.
- **x_label** (*str*) – Label for the x-axis. Default is 'x'.
- **x_factor** (*float*) – Scaling factor for the x-axis. Default is 1.
- **x_lim** (*sequence*) – Sequence representing the interval of the x-axis to visualize. By default, the entire domain is shown.
- **z_label** (*str*) – Label for the z-axis. Default is 'z'.
- **z_factor** (*float*) – Scaling factor for the z-axis. Default is 1.
- **z_lim** (*sequence*) – Sequence representing the interval of the z-axis to visualize. By default, the entire domain is shown.
- **field_factor** (*float*) – Scaling factor for the field. Default is 1.
- **draw_z_isolines** (*bool*) – `True` to draw the z-isolines, `False` otherwise. Default is `True`.
- **text** (*str*) – Text to be added to the figure as anchored text. By default, no extra text is shown.
- **text_loc** (*str*) – String specifying the location where the text box should be placed. Default is 'upper right'; please see `matplotlib.offsetbox.AnchoredText` for all the available options.

`tasmania.utils.utils_plot.`**contourf_xy**(*x*, *y*, *topography*, *field*, ***kwargs*)

Generate the contourf plot of a field at a cross-section parallel to the xy-plane.

Parameters

- **x** (*array_like*) – Two-dimensional `numpy.ndarray` representing the underlying x-grid.
- **y** (*array_like*) – Two-dimensional `numpy.ndarray` representing the underlying y-grid.
- **topography** (*array_like*) – Two-dimensional `numpy.ndarray` representing the underlying topography height.
- **field** (*array_like*) – Two-dimensional `numpy.ndarray` representing the field to plot.

Keyword Arguments

- **show** (*bool*) – `True` if the plot should be showed, `False` otherwise. Default is `True`.
- **destination** (*str*) – String specify the path to the location where the plot will be saved. Default is `None`, meaning that the plot will not be saved. Note that the plot may be saved only if `show` is set to `False`.
- **fontsize** (*int*) – The fontsize to be used. Default is 12.
- **figsize** (*sequence*) – Sequence representing the figure size. Default is [8,8].
- **title** (*str*) – The figure title. Default is an empty string.

- **x_label** (*str*) – Label for the x-axis. Default is 'x'.
- **x_factor** (*float*) – Scaling factor for the x-axis. Default is 1.
- **x_lim** (*sequence*) – Sequence representing the interval of the x-axis to visualize. By default, the entire domain is shown.
- **y_label** (*str*) – Label for the y-axis. Default is 'y'.
- **y_factor** (*float*) – Scaling factor for the y-axis. Default is 1.
- **y_lim** (*sequence*) – Sequence representing the interval of the y-axis to visualize. By default, the entire domain is shown.
- **field_factor** (*float*) – Scaling factor for the field. Default is 1.
- **cmap_name** (*str*) – Name of the Matplotlib's color map to be used. All the color maps provided by Matplotlib, as well as the corresponding inverted versions, are available.
- **cbar_levels** (*int*) – Number of levels for the color bar. Default is 14.
- **cbar_ticks_step** (*int*) – Distance between two consecutive labelled ticks of the color bar. Default is 1, i.e., all ticks are displayed with the corresponding label.
- **cbar_center** (*float*) – Center of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field.
- **cbar_half_width** (*float*) – Half-width of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field.
- **cbar_x_label** (*str*) – Label for the horizontal axis of the color bar. Default is an empty string.
- **cbar_y_label** (*str*) – Label for the vertical axis of the color bar. Default is an empty string.
- **cbar_orientation** (*str*) – Orientation of the color bar. Either 'vertical' (default) or 'horizontal'.
- **text** (*str*) – Text to be added to the figure as anchored text. By default, no extra text is shown.
- **text_loc** (*str*) – String specifying the location where the text box should be placed. Default is 'upper right'; please see `matplotlib.offsetbox.AnchoredText` for all the available options.

tasmania.utils.utils_plot.**contourf_xz**(*x*, *z*, *field*, *topography*, ***kwargs*)

Generate the contourf plot of a gridded field at a cross-section parallel to the xz-plane.

Parameters

- **x** (*array_like*) – Two-dimensional `numpy.ndarray` representing the underlying x-grid.
- **z** (*array_like*) – Two-dimensional `numpy.ndarray` representing the underlying z-grid.
- **field** (*array_like*) – Two-dimensional `numpy.ndarray` representing the field to plot.
- **topography** (*array_like*) – One-dimensional `numpy.ndarray` representing the underlying topography.

Keyword Arguments

- **show** (*bool*) – `True` if the plot should be showed, `False` otherwise. Default is `True`.
- **destination** (*str*) – String specifying the path to the location where the plot will be saved. Default is `None`, meaning that the plot will not be saved. Note that the plot may be saved only if `show` is set to `False`.
- **fontsize** (*int*) – The fontsize to be used. Default is 12.
- **figsize** (*sequence*) – Sequence representing the figure size. Default is [8,8].
- **title** (*str*) – The figure title. Default is an empty string.
- **x_label** (*str*) – Label for the x-axis. Default is 'x'.
- **x_factor** (*float*) – Scaling factor for the x-axis. Default is 1.
- **x_lim** (*sequence*) – Sequence representing the interval of the x-axis to visualize. By default, the entire domain is shown.
- **z_label** (*str*) – Label for the z-axis. Default is 'z'.
- **z_factor** (*float*) – Scaling factor for the z-axis. Default is 1.
- **z_lim** (*sequence*) – Sequence representing the interval of the z-axis to visualize. By default, the entire domain is shown.
- **field_factor** (*float*) – Scaling factor for the field. Default is 1.
- **draw_z_isolines** (*bool*) – `True` to draw the z-isolines, `False` otherwise. Default is `True`.
- **cmap_name** (*str*) – Name of the Matplotlib's color map to be used. All the color maps provided by Matplotlib, as well as the corresponding inverted versions, are available.
- **cbar_levels** (*int*) – Number of levels for the color bar. Default is 14.
- **cbar_ticks_step** (*int*) – Distance between two consecutive labelled ticks of the color bar. Default is 1, i.e., all ticks are displayed with the corresponding label.
- **cbar_center** (*float*) – Center of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field.
- **cbar_half_width** (*float*) – Half-width of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field.
- **cbar_x_label** (*str*) – Label for the horizontal axis of the color bar. Default is an empty string.
- **cbar_y_label** (*str*) – Label for the vertical axis of the color bar. Default is an empty string.
- **cbar_orientation** (*str*) – Orientation of the color bar. Either 'vertical' (default) or 'horizontal'.
- **text** (*str*) – Text to be added to the figure as anchored text. By default, no extra text is shown.
- **text_loc** (*str*) – String specifying the location where the text box should be placed. Default is 'upper right'; please see `matplotlib.offsetbox.AnchoredText` for all the available options.

tasmania.utils.utils_plot.**quiver_xy**(*x*, *y*, *topography*, *vx*, *vy*, *scalar=None*, ***kwargs*)

Generate the quiver plot of a gridded vectorial field at a cross-section parallel to the xy-plane.

Parameters

- **x** (*array_like*) – Two-dimensional numpy.ndarray representing the underlying x-grid.
- **y** (*array_like*) – Two-dimensional numpy.ndarray representing the underlying y-grid.
- **topography** (*array_like*) – Two-dimensional numpy.ndarray representing the underlying topography height.
- **vx** (*array_like*) – numpy.ndarray representing the x-component of the field to plot.
- **vy** (*array_like*) – numpy.ndarray representing the y-component of the field to plot.
- **scalar** (*array_like*, optional) – numpy.ndarray representing a scalar field associated with the vectorial field. The arrows will be colored based on the associated scalar value. If not specified, the arrows will be colored based on their magnitude.

Keyword Arguments

- **show** (*bool*) – True if the plot should be showed, False otherwise. Default is True.
- **destination** (*str*) – String specify the path to the location where the plot will be saved. Default is None, meaning that the plot will not be saved. Note that the plot may be saved only if show is set to False.
- **fontsize** (*int*) – The fontsize to be used. Default is 12.
- **figsize** (*sequence*) – Sequence representing the figure size. Default is [8,8].
- **title** (*str*) – The figure title. Default is an empty string.
- **x_label** (*str*) – Label for the x-axis. Default is 'x'.
- **x_factor** (*float*) – Scaling factor for the x-axis. Default is 1.
- **x_lim** (*sequence*) – Sequence representing the interval of the x-axis to visualize. By default, the entire domain is shown.
- **x_step** (*int*) – Maximum distance between the x-index of a drawn point, and the x-index of any of its neighbours. Default is 2, i.e., only half of the points will be drawn.
- **y_label** (*str*) – Label for the y-axis. Default is 'y'.
- **y_factor** (*float*) – Scaling factor for the y-axis. Default is 1.
- **y_lim** (*sequence*) – Sequence representing the interval of the y-axis to visualize. By default, the entire domain is shown.
- **y_step** (*int*) – Maximum distance between the y-index of a drawn point, and the :math:'y-index of any of its neighbours. Default is 2, i.e., only half of the points will be drawn.
- **field_factor** (*float*) – Scaling factor for the field. Default is 1.
- **cmap_name** (*str*) – Name of the Matplotlib's color map to be used. All the color maps provided by Matplotlib, as well as the corresponding inverted versions, are available. If not specified, no color map will be used, and the arrows will draw black.
- **cbar_levels** (*int*) – Number of levels for the color bar. Default is 14.

- **cbar_ticks_step** (*int*) – Distance between two consecutive labelled ticks of the color bar. Default is 1, i.e., all ticks are displayed with the corresponding label.
- **cbar_center** (*float*) – Center of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field.
- **cbar_half_width** (*float*) – Half-width of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field.
- **cbar_x_label** (*str*) – Label for the horizontal axis of the color bar. Default is an empty string.
- **cbar_y_label** (*str*) – Label for the vertical axis of the color bar. Default is an empty string.
- **cbar_orientation** (*str*) – Orientation of the color bar. Either 'vertical' (default) or 'horizontal'.
- **text** (*str*) – Text to be added to the figure as anchored text. By default, no extra text is shown.
- **text_loc** (*str*) – String specifying the location where the text box should be placed. Default is 'upper right'; please see `matplotlib.offsetbox.AnchoredText` for all the available options.

`tasmania.utils.utils_plot.`**quiver_xz**(*x*, *z*, *topography*, *vx*, *vz*, *scalar=None*, ***kwargs*)

Generate the quiver plot of a gridded vectorial field at a cross-section parallel to the xz-plane.

Parameters

- **x** (*array_like*) – Two-dimensional `numpy.ndarray` representing the underlying x-grid.
- **z** (*array_like*) – Two-dimensional `numpy.ndarray` representing the underlying z-grid.
- **topography** (*array_like*) – One-dimensional `numpy.ndarray` representing the underlying topography height.
- **vx** (*array_like*) – `numpy.ndarray` representing the x-component of the field to plot.
- **vz** (*array_like*) – `numpy.ndarray` representing the z-component of the field to plot.
- **scalar** (*array_like*, optional) – `numpy.ndarray` representing a scalar field associated with the vectorial field. The arrows will be colored based on the associated scalar value. If not specified, the arrows will be colored based on their magnitude.

Keyword Arguments

- **show** (*bool*) – `True` if the plot should be showed, `False` otherwise. Default is `True`.
- **destination** (*str*) – String specify the path to the location where the plot will be saved. Default is `None`, meaning that the plot will not be saved. Note that the plot may be saved only if `show` is set to `False`.
- **fontsize** (*int*) – The fontsize to be used. Default is 12.
- **figsize** (*sequence*) – Sequence representing the figure size. Default is [8,8].
- **title** (*str*) – The figure title. Default is an empty string.

- **x_label** (*str*) – Label for the x-axis. Default is 'x'.
- **x_factor** (*float*) – Scaling factor for the x-axis. Default is 1.
- **x_lim** (*sequence*) – Sequence representing the interval of the x-axis to visualize. By default, the entire domain is shown.
- **x_step** (*int*) – Maximum distance between the x-index of a drawn point, and the x-index of any of its neighbours. Default is 2, i.e., only half of the points will be drawn.
- **z_label** (*str*) – Label for the z-axis. Default is 'z'.
- **z_factor** (*float*) – Scaling factor for the z-axis. Default is 1.
- **z_lim** (*sequence*) – Sequence representing the interval of the z-axis to visualize. By default, the entire domain is shown.
- **z_step** (*int*) – Maximum distance between the z-index of a drawn point, and the :math:'z-index of any of its neighbours. Default is 2, i.e., only half of the points will be drawn.
- **field_factor** (*float*) – Scaling factor for the field. Default is 1.
- **cmap_name** (*str*) – Name of the Matplotlib's color map to be used. All the color maps provided by Matplotlib, as well as the corresponding inverted versions, are available. If not specified, no color map will be used, and the arrows will draw black.
- **cbar_levels** (*int*) – Number of levels for the color bar. Default is 14.
- **cbar_ticks_step** (*int*) – Distance between two consecutive labelled ticks of the color bar. Default is 1, i.e., all ticks are displayed with the corresponding label.
- **cbar_center** (*float*) – Center of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field.
- **cbar_half_width** (*float*) – Half-width of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field.
- **cbar_x_label** (*str*) – Label for the horizontal axis of the color bar. Default is an empty string.
- **cbar_y_label** (*str*) – Label for the vertical axis of the color bar. Default is an empty string.
- **cbar_orientation** (*str*) – Orientation of the color bar. Either 'vertical' (default) or 'horizontal'.
- **text** (*str*) – Text to be added to the figure as anchored text. By default, no extra text is shown.
- **text_loc** (*str*) – String specifying the location where the text box should be placed. Default is 'upper right'; please see `matplotlib.offsetbox.AnchoredText` for all the available options.

tasmania.utils.utils_plot.**streamplot_xz**(*x*, *z*, *u*, *w*, *color*, *topography*, ***kwargs*)

Generate the streamplot of a gridded vector field at a cross-section parallel to the xz-plane.

Parameters

- **x** (*array_like*) – Two-dimensional `numpy.ndarray` representing the underlying x-grid.

- **z** (*array_like*) – Two-dimensional numpy.ndarray representing the underlying z-grid.
- **u** (*array_like*) – Two-dimensional numpy.ndarray representing the x-velocity.
- **w** (*array_like*) – Two-dimensional numpy.ndarray representing the z-velocity.
- **color** (*array_like*) – Two-dimensional numpy.ndarray representing the streamlines color.
- **topography** (*array_like*) – One-dimensional numpy.ndarray representing the underlying topography.

Keyword Arguments

- **show** (*bool*) – True if the plot should be showed, False otherwise. Default is True.
- **destination** (*str*) – String specifying the path to the location where the plot will be saved. Default is None, meaning that the plot will not be saved. Note that the plot may be saved only if show is set to False.
- **fontsize** (*int*) – The fontsize to be used. Default is 12.
- **figsize** (*sequence*) – Sequence representing the figure size. Default is [8,8].
- **title** (*str*) – The figure title. Default is an empty string.
- **x_label** (*str*) – Label for the x-axis. Default is 'x'.
- **x_factor** (*float*) – Scaling factor for the x-axis. Default is 1.
- **x_lim** (*sequence*) – Sequence representing the interval of the x-axis to visualize. By default, the entire domain is shown.
- **z_label** (*str*) – Label for the z-axis. Default is 'z'.
- **z_factor** (*float*) – Scaling factor for the z-axis. Default is 1.
- **z_lim** (*sequence*) – Sequence representing the interval of the z-axis to visualize. By default, the entire domain is shown.
- **u_factor** (*float*) – Scaling factor for the x-velocity. Default is 1.
- **w_factor** (*float*) – Scaling factor for the z-velocity. Default is 1.
- **color_factor** (*float*) – Scaling factor for the color field. Default is 1.
- **draw_z_isolines** (*bool*) – True to draw the z-isolines, False otherwise. Default is False.
- **cmap_name** (*str*) – Name of the Matplotlib's color map to be used. All the color maps provided by Matplotlib, as well as the corresponding inverted versions, are available.
- **cbar_levels** (*int*) – Number of levels for the color bar. Default is 14.
- **cbar_ticks_step** (*int*) – Distance between two consecutive labelled ticks of the color bar. Default is 1, i.e., all ticks are displayed with the corresponding label.
- **cbar_center** (*float*) – Center of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field.
- **cbar_half_width** (*float*) – Half-width of the range covered by the color bar. By default, the color bar covers the spectrum ranging from the minimum to the maximum assumed by the field.

- **cbar_x_label** (*str*) – Label for the horizontal axis of the color bar. Default is an empty string.
- **cbar_y_label** (*str*) – Label for the vertical axis of the color bar. Default is an empty string.
- **cbar_orientation** (*str*) – Orientation of the color bar. Either 'vertical' (default) or 'horizontal'.
- **text** (*str*) – Text to be added to the figure as anchored text. By default, no extra text is shown.
- **text_loc** (*str*) – String specifying the location where the text box should be placed. Default is 'upper right'; please see `matplotlib.offsetbox.AnchoredText` for all the available options.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t