```
import numpy as np
from dataclasses import (
    field,
)  # Not used here, but good practice if other fields were added
from typing import TYPE_CHECKING

from atmpy.infrastructure.utility import directional_indices
from atmpy.test_cases.base_test_case import BaseTestCase
from atmpy.configuration.simulation_configuration import SimulationConfig
from atmpy.infrastructure.enums import (
    BoundaryConditions as BdryType,
    BoundarySide,
    AdvectionRoutines,
    SlopeLimiters as LimiterType,
    VariableIndices as VI,
    HydrostateIndices as HI,
)
from atmpy.physics.thermodynamics import Thermodynamics

if TYPE_CHECKING:
    from atmpy.variables.variables import Variables
    from atmpy.variables.multiple_pressure_variables import MPV
    from atmpy.grid.kgrid import Grid  # Assuming Grid is in kgrid.py


class RisingBubble(BaseTestCase):
    """
    Rising bubble test case, translated from PyBella's example.
    A thermal perturbation in an otherwise isentropic, hydrostatic atmosphere.
    This version initializes on the full domain (including ghost cells)
    and then relies on boundary conditions to correct ghost cells.
    """

    def __init__(self, config_override: SimulationConfig = None):
        # Initialize with a default SimulationConfig, which will be modified in
setup
        _effective_config: SimulationConfig
        run_setup_method = False

        if config_override is not None:
            _effective_config = config_override
        else:
            # No override, create a default config. BaseTestCase will get this,
            # and then setup() will populate it.
            _effective_config = SimulationConfig()
            run_setup_method = True

        super().__init__(name="TravelingVortex", config=_effective_config)

        # Case-specific parameters
        self.del_theta_k: float = 2.0  # Initial potential temperature
perturbation [K]
        self.xc_bubble: float = 0.0  # Center of bubble x (non-dimensional)
        self.yc_bubble: float = 0.2  # Center of bubble y (non-dimensional)
        self.r0_bubble: float = 0.2  # Radius of bubble (non-dimensional)

        self.setup()

    def setup(self):
        """Configure the SimulationConfig for the Rising Bubble case."""
        print("Setting up Rising Bubble configuration...")

        # Grid Configuration
        grid_updates = {
```

```
            "ndim": 2,
            "nx": 120,
            "ny": 60,
            "nz": 0,
            "xmin": -1.0,
            "xmax": 1.0,
            "ymin": 0.0,
            "ymax": 1.0,
            "ngx": 2,
            "ngy": 2,
        }
        self.set_grid_configuration(grid_updates)

        # Boundary Conditions
        self.set_boundary_condition(
            BoundarySide.LEFT, BdryType.PERIODIC, mpv_type=BdryType.PERIODIC
        )
        self.set_boundary_condition(
            BoundarySide.RIGHT, BdryType.PERIODIC, mpv_type=BdryType.PERIODIC
        )
        # UPDATED Y-direction BCs
        self.set_boundary_condition(
            BoundarySide.BOTTOM,
            BdryType.REFLECTIVE_GRAVITY,
            mpv_type=BdryType.WALL,  # MPV often uses WALL for
REFLECTIVE_GRAVITY
        )
        self.set_boundary_condition(
            BoundarySide.TOP,
            BdryType.REFLECTIVE_GRAVITY,
            mpv_type=BdryType.WALL,  # MPV often uses WALL for
REFLECTIVE_GRAVITY
        )

        # Temporal Setting
        temporal_updates = {
            "CFL": 0.5,
            "dtfixed": 0.001,
            "dtfixed0": 0.1,
            "tout": np.arange(0.1, 0.71, 0.1),
            "stepmax": 10000,
            "use_acoustic_cfl": True
        }
        self.set_temporal(temporal_updates)

        # Global Constants
        constants_updates = {
            "grav": 10.0,
            "omega": 0.0,
            "R_gas": 287.4,
            "gamma": 1.4,
            "p_ref": 8.61e4,
            "T_ref": 300.0,
            "h_ref": 10000.0,
            "t_ref": 1000.0,
            "Nsq_ref": 1.3e-4,
        }
        self.set_global_constants(constants_updates)

        # Physics Settings
        physics_updates = {
            "wind_speed": [0.0, 0.0, 0.0],
            "gravity_strength": (0.0, 1.0, 0.0),  # Directional preference for
gravity
```

```
            "coriolis_strength": (0.0, 0.0, 0.0),
            "stratification": lambda y: 1.0,
        }
        self.set_physics(physics_updates)

        # Model Regimes
        regime_updates = {
            "is_ArakawaKonor": 0,
            "is_nongeostrophic": 1,
            "is_nonhydrostatic": 1,
            "is_compressible": 1,  # Effectively sets the d(rhoY)/dt term in
pressure solver to zero
            # Set to 1 if you want the fully compressible solver pressure term
        }
        self.set_model_regimes(regime_updates)

        # Numerics
        numerics_updates = {
            "limiter_scalars": LimiterType.VAN_LEER,  # UPDATED Limiter
            "advection_routine": AdvectionRoutines.STRANG_SPLIT,
            "tol": 1.0e-8,
            "max_iterations": 6000,
            "initial_projection": False,
        }
        self.set_numerics(numerics_updates)

        # Outputs
        output_updates = {
            "output_type": "test",
            "output_folder": "rising_bubble",
            "output_base_name": "_rising_bubble",
            "output_timesteps": True,
        }
        self.set_outputs(output_updates)

        # Diagnostics
        diag_updates = {
            "diag": True,
            "diag_current_run": "atmpy_rising_bubble",
        }
        self.set_diagnostics(diag_updates)

        self.config.update_all_derived_fields()
        self._update_output_suffix()

        print(
            f"Rising Bubble configuration complete. Msq =
{self.config.model_regimes.Msq:.4e}"
        )
        print(f"Output files: {self.config.outputs.output_filename}")

    def initialize_solution(self, variables: "Variables", mpv: "MPV"):
        """Initialize density, momentum, potential temperature, and pressure
fields
        on the full domain (including ghost cells)."""
        print("Initializing solution for Rising Bubble (full domain)...")

        grid_obj: "Grid" = (
            self.config.spatial_grid.grid
        )  # Access the actual Grid object
        thermo = Thermodynamics()
        thermo.update(self.config.global_constants.gamma)
        Msq = self.config.model_regimes.Msq
        THETA_0 = 300
```

```
        self.config.update_all_derived_fields()

        # Calculate Hydrostatic Base State using scaled gravity from physics
config
        # mpv.hydrostate variables (1D) are defined on the full 1D grid
(including ghosts)
        mpv.state(self.config.physics.gravity_strength, Msq)

        # Get Cell-Centered Coordinates (FULL Domain, including ghosts)
        if grid_obj.ndim == 2:
            # These are 1D arrays including ghost cell coordinates
            X_cell_full = grid_obj.x_cells
            Y_cell_full = grid_obj.y_cells
            # meshgrid with 'ij' indexing gives XC(nx_total,ny_total),
YC(nx_total,ny_total)
            XC_full, YC_full = np.meshgrid(X_cell_full, Y_cell_full,
indexing="ij")
        else:
            raise NotImplementedError(
                "Rising bubble initialization only implemented for 2D"
            )

        # Calculate Radial Distance for Perturbation (on full grid)
        r = (
            np.sqrt((XC_full - self.xc_bubble) ** 2 + (YC_full - self.yc_bubble)
** 2)
            / self.r0_bubble
        )

        p0 = mpv.hydrostate.cell_vars[..., HI.P0]
        rhoY0 = mpv.hydrostate.cell_vars[..., HI.RHOY0]

        icshape = self.config.spatial_grid.grid.icshape
        slices = [slice(None)] * 2
        x_inner_slice = slices
        x_inner_slice[0] = slice(2, -2)
        x_inner_slice = tuple(x_inner_slice)
        y_inner_slice = slices
        y_inner_slice[1] = slice(2, -2)
        y_inner_slice = tuple(y_inner_slice)
        p = np.repeat(p0.reshape(1, -1), icshape[0], axis=0)
        rhoY = np.repeat(rhoY0.reshape(1, -1), icshape[0], axis=0)

        perturbation = (self.del_theta_k/THETA_0)*(np.cos(0.5 * np.pi * r)**2)
        perturbation[np.where(r > 1.0)] = 0.0
        rho = rhoY / (self.config.physics.stratification(Y_cell_full)+
perturbation[x_inner_slice])

        inner_slice = self.config.spatial_grid.grid.get_inner_slice()

        variables.cell_vars[x_inner_slice + (VI.RHO,)] = rho
        variables.cell_vars[..., VI.RHOU] = 0
        variables.cell_vars[..., VI.RHOV] = 0
        variables.cell_vars[..., VI.RHOW] = 0
        variables.cell_vars[x_inner_slice + (VI.RHOY,)] = rhoY

        mpv.p2_nodes[...] = mpv.hydrostate.node_vars[..., HI.P2_0]
        # mpv.p2_nodes[...] = 0
        print("Full domain solution initialization complete for Rising Bubble.")
        print("Ghost cells will be overwritten by BoundaryManager.")
```