Ocean-Atmosphere Coupling Model

June 29, 2025

1 Overview of `Model.py`

The `OceanAtmosphereModel` class in `Model.py` serves as the backbone of the simulator, managing the time evolution of ocean and atmosphere fields. It handles initialization, time stepping, and integration of physical processes such as heat, momentum, freshwater, and CO_2 fluxes, advection, diffusion, and turbulent mixing. It interfaces with `VariableResolution.py` for spatially variable grids, `NestedGrid.py` for nested grid updates, `AdaptiveMeshRefinement.py` for dynamic grid refinement, and `TwoWayCoupling.py` for flux and mixing calculations.

2 Functionalities

The `OceanAtmosphereModel` class provides the following functionalities:

- **Initialization of Fields**: Sets up a 2D grid ($N \times N$) with initial conditions for ocean temperature (T_o), atmosphere temperature (T_a), salinity (S), ocean velocities ($u_{\text{ocean}}, v_{\text{ocean}}$), moisture ($q$), and CO_2 concentrations ($\text{CO}_{2,\text{ocean}}, \text{CO}_{2,\text{atm}}).Configuresvariabler$ $Advancesthesimulationusingdistincttimestepsforocean(\Delta t \cdot$ ocean_time_scale) and atmosphere ($\Delta t \cdot$ atm_time_scale), updating fields with fluxes, advection, diffusion, and turbulent mixing, while applying numerical stability constraints.

- **Physical Process Integration**: Computes advection and diffusion for temperature, salinity, and CO_2 fields using finite difference methods, integrates fluxes from `TwoWayCoupling`, and applies Coriolis effects via momentum flux.

- **Grid Management**: Uses `VariableResolutionGrid` for spatially variable resolution, supports nested grids via `NestedGrid`, and employs `AdaptiveMeshRefinement` for dynamic grid refinement based on temperature gradients or vorticity.

- **Logging and Error Handling**: Logs initialization and step progress using the `logging` module, with exception handling to catch and log errors.

3 Simulation Logic

The simulation logic in `Model.py` is centered around the `OceanAtmosphereModel` class, which manages the time evolution of the coupled system.

3.1 Initialization

- **Purpose**: Sets up the initial state of ocean and atmosphere fields and configures numerical and physical parameters.
- **Process**:
  - Initializes 2D arrays for T_o (ocean temperature), T_a (atmosphere temperature), S (salinity), $u_{\text{ocean}}, v_{\text{ocean}}$ (ocean velocities), q (moisture), CO$_{2,\text{ocean}}, and$CO$_{2,\text{at}}$
    ocean_temp, $T_a =$ atm_temp, $S = 35.0$ psu, $q = 0.01$, CO$_{2,\text{atm}} = 400.0$ ppm$). Clips initial f$
    $[250, 350]$ K, $S \in [30, 40]$ psu, $q \in [0, 0.05]$, CO$_{2,\text{ocean}} \in [0, 10]$, CO$_{2,\text{atm}} \in$ $[200, 1000]$ ppm.
  - Creates a `VariableResolutionGrid` instance to define spatially variable Δx and Δy (finer near coasts).
  - Optionally initializes a `NestedGrid` for high-resolution subdomains.
  - Sets up an `AdaptiveMeshRefinement` instance with a specified threshold.
  - Configures a `TwoWayCoupling` instance with parameters for drag coefficient, wind speed, precipitation, evaporation, solar forcing, longwave coefficient, mixing coefficient, and CO$_2$ transfer coefficient.
  - Applies distinct time scales: $\Delta t_{\text{ocean}} = \Delta t \cdot$ ocean_time_scale (default 1.0), $\Delta t_{\text{atm}} = \Delta t \cdot$ atm_time_scale (default 0.1).

3.2 Time Stepping (`step` Method)

- **Purpose**: Advances the simulation by one time step, updating all fields.
- **Process**:
  1. Copies current fields ($T_o, T_a, S, u_{\text{ocean}}, v_{\text{ocean}}, q,$ CO$_{2,\text{ocean}}$, CO$_{2,\text{atm}}$) to avoid in-place modifications.
  2. Computes a refinement mask using `AdaptiveMeshRefinement.compute_refiner` based on temperature gradients or vorticity.
  3. Updates fields on the nested grid (if enabled) using `NestedGrid.update`.

4. Defines time-varying atmosphere velocities:

$$u_{\text{atm}} = \text{adv\_velocity} \cdot \cos\left(\frac{2\pi \cdot \text{step} \cdot \Delta t}{\text{total\_time}}\right),$$
$$v_{\text{atm}} = \text{adv\_velocity} \cdot \sin\left(\frac{2\pi \cdot \text{step} \cdot \Delta t}{\text{total\_time}}\right).$$

5. Computes fluxes and mixing using `TwoWayCoupling`:

   – Heat flux (Q) using $T_a, T_o, u_{\text{ocean}}, v_{\text{ocean}}$.

   – Radiative flux ($R_{\text{ocean}}, R_{\text{atm}}$) using T_o, T_a, and $\text{CO}_{2,\text{atm}}.Freshwaterflux($d S$\frac{}{dt, F_{\text{freshwater}}}$) using S and q.

   – Momentum flux (τ) using wind speed, $u_{\text{ocean}}, v_{\text{ocean}}$.

   – Moisture advection (M_{adv}) using $q, \Delta x, \Delta y, u_{\text{atm}}, v_{\text{atm}}$.

   – Turbulent mixing ($\text{mix}_{\text{ocean}}, \text{mix}_{\text{atm}}$) using $T_o, T_a, S, \Delta x, \Delta y$, wind speed.

– CO_2 flux ($F_{\text{CO}_2,\text{ocean}}, F_{\text{CO}_2,\text{atm}}$) using $\text{CO}_{2,\text{ocean}}, \text{CO}_{2,\text{atm}}$.

6. Updates ocean velocities:

$$u_{\text{ocean}}^{n+1} = u_{\text{ocean}}^{n} + \Delta t_{\text{ocean}} \cdot \frac{\tau}{\rho_{\text{water}}},$$
$$v_{\text{ocean}}^{n+1} = v_{\text{ocean}}^{n} + \Delta t_{\text{ocean}} \cdot \frac{\tau}{\rho_{\text{water}}}.$$

7. Computes advection for $T_o, T_a, S, \text{CO}_{2,\text{ocean}}, \text{CO}_{2,\text{atm}}$ using `compute_advection`.

8. Computes diffusion for T_o, T_a, S using `compute_diffusion`.

9. Updates fields using a semi-implicit scheme ($\alpha = 0.5$):

   – Ocean temperature, atmosphere temperature, salinity, moisture, and CO_2 concentrations (detailed in Section 5).

10. Clips updated fields to physical ranges.

11. Applies AMR to refine T_o, T_a, S where the refinement mask is active.

12. Returns current time, updated T_o, T_a, and refinement mask.

4 Physics and Mathematical Models

The `OceanAtmosphereModel` integrates physical processes through numerical updates, relying on `TwoWayCoupling` for flux calculations. Below are the key models and equations implemented in `Model.py`, formatted to fit within margins.

4.1 Ocean Temperature Update

- **Purpose**: Updates ocean temperature (T_o) based on heat flux, radiative flux, advection, diffusion, and turbulent mixing.
- **Equation**:

$$T_o^{n+1} = T_o^n + \Delta t_{\text{ocean}} \cdot \left[\alpha \cdot \left(\frac{Q}{C_o} + \frac{R_{\text{ocean}}}{C_o} - \text{adv}_{\text{ocean}} \right. \right.$$
$$\left. \left. + \text{diff}_{\text{ocean}} + \frac{\text{mix}_{\text{ocean}}}{C_o} \right) + (1-\alpha) \cdot \left(\frac{Q}{C_o} + \frac{R_{\text{ocean}}}{C_o} \right) \right],$$

  where:

  - Q is heat flux from `compute_heat_flux` ($\mathrm{W/m^2}$).
  - R_{ocean} is radiative flux from `compute_radiative_flux` ($\mathrm{W/m^2}$).
  - $\text{adv}_{\text{ocean}}$ is advection from `compute_advection` ($\mathrm{K/s}$).
  - $\text{diff}_{\text{ocean}}$ is diffusion from `compute_diffusion` ($\mathrm{K/s}$).
  - $\text{mix}_{\text{ocean}}$ is turbulent mixing from `compute_turbulent_mixing` ($\mathrm{W/m^2}$).
  - C_o is ocean heat capacity ($\mathrm{J/kg/K}$).
  - $\alpha = 0.5$ is the semi-implicit factor.
  - $\Delta t_{\text{ocean}} = \Delta t \cdot$ ocean_time_scale (s).
- **Implementation**: Clips terms to $[-10^3, 10^3]$ and T_o to $[250, 350]$ K.

4.2 Atmosphere Temperature Update

- **Purpose**: Updates atmosphere temperature (T_a) based on heat flux, radiative flux, advection, diffusion, and turbulent mixing.
- **Equation**:

$$T_a^{n+1} = T_a^n + \Delta t_{\text{atm}} \cdot \left[\alpha \cdot \left(-\frac{Q}{C_a} + \frac{R_{\text{atm}}}{C_a} - \text{adv}_{\text{atm}} \right. \right.$$
$$\left. \left. + \text{diff}_{\text{atm}} + \frac{\text{mix}_{\text{atm}}}{C_a} \right) + (1-\alpha) \cdot \left(-\frac{Q}{C_a} + \frac{R_{\text{atm}}}{C_a} \right) \right],$$

  where:

  - Q is heat flux (negative for atmosphere due to coupling).
  - R_{atm} is radiative flux.

– adv_{atm} is advection.

– diff_{atm} is diffusion.

– mix_{atm} is turbulent mixing.

– C_a is atmosphere heat capacity ($\mathrm{J/kg/K}$).

– $\Delta t_{\text{atm}} = \Delta t \cdot \text{atm\_time\_scale}$ (s).

- **Implementation**: Clips terms to $[-10^3, 10^3]$ and T_a to $[250, 350]$ K.

4.3 Salinity Update

- **Purpose**: Updates salinity (S) based on freshwater flux, diffusion, and turbulent mixing.

- **Equation**:
$$S^{n+1} = S^n + \Delta t \cdot \left(\frac{dS}{dt} + \text{diff}_{\text{salinity}} + \text{mix}_{\text{ocean}} \right),$$
where:

  – $\frac{dS}{dt}$ is salinity change from `compute_freshwater_flux`.

  – $\text{diff}_{\text{salinity}}$ is diffusion.

  – $\text{mix}_{\text{ocean}}$ is turbulent mixing.

- **Implementation**: Clips terms to $[-10^{-2}, 10^{-2}]$ and S to $[30, 40]$ psu.

4.4 Moisture Update

- **Purpose**: Updates atmospheric moisture (q) based on advection and freshwater flux.

- **Equation**:
$$q^{n+1} = q^n + \Delta t \cdot (M_{\text{adv}} - F_{\text{freshwater}}),$$
where:

  – M_{adv} is moisture advection from `compute_moisture_advection`.

  – $F_{\text{freshwater}}$ is freshwater flux from `compute_freshwater_flux`.

- **Implementation**: Clips terms to $[-10^{-4}, 10^{-4}]$ and q to $[0, 0.05]$.

4.5 CO_2 Concentration Updates

- **Purpose**: Updates ocean and atmosphere CO_2 concentrations ($CO_{2,\text{ocean}}$, $CO_{2,\text{atm}}$)$basedonC$ flux and advection.

- **Equations**:

$$
\begin{aligned}
\text{CO}_{2,\text{ocean}}^{n+1} &= \text{CO}_{2,\text{ocean}}^{n} + \Delta t \cdot \left(F_{\text{CO}_2,\text{ocean}} + \text{adv}_{\text{co2\_o}}\right), \\
\text{CO}_{2,\text{atm}}^{n+1} &= \text{CO}_{2,\text{atm}}^{n} + \Delta t \cdot \left(F_{\text{CO}_2,\text{atm}} + \text{adv}_{\text{co2\_a}}\right),
\end{aligned}
$$

  where:

  – $F_{\text{CO}_2,\text{ocean}}, F_{\text{CO}_2,\text{atm}}$ are CO_2 fluxes from `compute_co2_flux`.

  – $\text{adv}_{\text{co2\_o}}, \text{adv}_{\text{co2\_a}}$ are advection terms from `compute_advection`.

- **Implementation**: Clips terms to $[-10^{-2}, 10^{-2}]$, $\text{CO}_{2,\text{ocean}} to$[0, 10]$, and \text{CO}_{2,\text{atm}} to$[200, 1000] ppm.

4.6 Ocean Velocity Update

- **Purpose**: Updates ocean velocities ($u_{\text{ocean}}, v_{\text{ocean}}$) based on momentum flux.
- **Equations**:

$$
\begin{aligned}
u_{\text{ocean}}^{n+1} &= u_{\text{ocean}}^{n} + \Delta t_{\text{ocean}} \cdot \frac{\tau}{\rho_{\text{water}}}, \\
v_{\text{ocean}}^{n+1} &= v_{\text{ocean}}^{n} + \Delta t_{\text{ocean}} \cdot \frac{\tau}{\rho_{\text{water}}},
\end{aligned}
$$

  where:

  – τ is momentum flux from `compute_momentum_flux` ($\mathrm{N/m^2}$).

  – $\rho_{\text{water}} = 1025\,\text{kg/m}^3$ (from `TwoWayCoupling`).

- **Implementation**: Clips velocities to $[-10, 10]$ m/s.

4.7 Advection

- **Purpose**: Computes advection for any field (T, S, CO_2).
- **Equations**:

$$
\begin{aligned}
\frac{\partial T}{\partial t} &= -u \cdot \frac{\partial T}{\partial x} - v \cdot \frac{\partial T}{\partial y}, \\
\frac{\partial T}{\partial x} &\approx \frac{T_{i+1,j} - T_{i-1,j}}{2\Delta x_{i,j}}, \\
\frac{\partial T}{\partial y} &\approx \frac{T_{i,j+1} - T_{i,j-1}}{2\Delta y_{i,j}}.
\end{aligned}
$$

- **Implementation**: Uses central differences, supports array-based or scalar velocities, and clips output to $[-10^5, 10^5]$.

4.8 Diffusion

- **Purpose**: Computes diffusion for temperature or salinity fields.
- **Equations**:

$$\frac{\partial T}{\partial t} = D \cdot \nabla^2 T,$$
$$\nabla^2 T \approx \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x_{i,j}^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y_{i,j}^2},$$

  where $D = 10^{-6}\,\mathrm{m}^2/\mathrm{s}$.

- **Implementation**: Uses central differences and clips output to $[-10^5, 10^5]$.

5 Algorithms

5.1 Initialization Algorithm

- **Steps**:
  1. Log initialization parameters.
  2. Store input parameters as instance variables.
  3. Compute $\Delta t_{\text{ocean}} = \Delta t \cdot$ocean_time_scale and $\Delta t_{\text{atm}} = \Delta t \cdot$atm_time_scale.
  4. Initialize `TwoWayCoupling` with flux parameters.
  5. Create `VariableResolutionGrid` and get $\Delta x, \Delta y$.
  6. If `use_nested_grid`, initialize `NestedGrid`.
  7. Initialize `AdaptiveMeshRefinement` with `amr_threshold`.
  8. Set up 2D arrays:
     - `ocean_temps` = `ocean_temp`, clipped to $[250, 350]$ K.
     - `atm_temps` = `atm_temp`, clipped to $[250, 350]$ K.
     - `salinity` = 35.0 psu.
     - `u_ocean`, `v_ocean` = 0.
     - `moisture` = 0.01.
     - `co2_ocean` = 2.0, `co2_atm` = 400.0 ppm.
  9. Log completion.

5.2 Time Stepping Algorithm (step)

- **Input**: Step number (`step`).
- **Steps**:
  1. Copy current fields to avoid in-place modification.
  2. Compute `refinement_mask` using `AdaptiveMeshRefinement.compute_refinem`
  3. If `nested_grid` exists, update T_o, T_a using `NestedGrid.update`.
  4. Compute atmosphere velocities:

$$u_{\text{atm}} = \text{adv\_velocity} \cdot \cos\left(\frac{2\pi \cdot \text{step} \cdot \Delta t}{\text{total\_time}}\right),$$
$$v_{\text{atm}} = \text{adv\_velocity} \cdot \sin\left(\frac{2\pi \cdot \text{step} \cdot \Delta t}{\text{total\_time}}\right).$$

  5. Compute fluxes and mixing using `TwoWayCoupling` methods.
  6. Update velocities: $u_{\text{new}}, v_{\text{new}}$ using momentum flux.
  7. Compute advection for $T_o, T_a, S, \text{CO}_{2,\text{ocean}}, \text{CO}_{2,\text{atm}}$.
  8. Compute diffusion for T_o, T_a, S.
  9. Update fields with semi-implicit scheme ($\alpha = 0.5$).
  10. Clip updated fields to physical ranges.
  11. Apply AMR refinement where `refinement_mask` is True.
  12. Log step completion and return time, `ocean_temps`, `atm_temps`, `refinement_mask`
  13. Handle exceptions and log errors if they occur.

5.3 Advection Algorithm (compute_advection)

- **Input**: Field T, Δx, Δy, `step`, optional velocities u, v.
- **Steps**:
  1. Initialize advection array (zeros, same shape as T).
  2. If u, v not provided, use:

$$u_{\text{vel}} = \text{adv\_velocity} \cdot \cos\left(\frac{2\pi \cdot \text{step} \cdot \Delta t}{\text{total\_time}}\right),$$
$$v_{\text{vel}} = \text{adv\_velocity} \cdot \sin\left(\frac{2\pi \cdot \text{step} \cdot \Delta t}{\text{total\_time}}\right).$$

3. For each grid point (i, j):
   - Compute u_{ij}, v_{ij} (from $u_{\text{vel}}, v_{\text{vel}}$, handling scalar or array inputs).
   - $\text{adv}_x = -u_{ij} \cdot \frac{T_{i+1,j} - T_{i-1,j}}{2\Delta x_{i,j}}$.
   - $\text{adv}_y = -v_{ij} \cdot \frac{T_{i,j+1} - T_{i,j-1}}{2\Delta y_{i,j}}$.
   - $\text{advection}[i, j] = \text{adv}_x + \text{adv}_y$, clipped to $[-10^5, 10^5]$.
4. Return advection array.
5. Log errors if computation fails.

5.4 Diffusion Algorithm (`compute_diffusion`)

- **Input**: Field T, Δx, Δy.
- **Steps**:
  1. Initialize diffusion array (zeros, same shape as T).
  2. Set $D = 10^{-6}\,\text{m}^2/\text{s}$.
  3. For each grid point (i, j):
     - $\text{diff}_x = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x_{i,j}^2}$.
     - $\text{diff}_y = \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y_{i,j}^2}$.
     - $\text{diffusion}[i, j] = D \cdot (\text{diff}_x + \text{diff}_y)$, clipped to $[-10^5, 10^5]$.
  4. Return diffusion array.
  5. Log errors if computation fails.