Two-Way Coupling

June 29, 2025

1 Functionalities

The `TwoWayCoupling` class provides the following functionalities:

- **Initialization**: Configures physical parameters such as drag coefficient, wind speed, precipitation rate, evaporation rate, solar forcing, longwave coefficient, mixing coefficient, CO_2 transfer coefficient, and conservation coefficients for freshwater and CO_2.
- **Sea Surface Roughness**: Computes an adjusted drag coefficient based on wind speed and ocean currents, accounting for sea surface roughness.
- **Momentum Flux**: Calculates wind stress on the ocean surface using the adjusted drag coefficient.
- **Heat Flux**: Computes sensible and latent heat fluxes across the air-sea interface, driven by temperature differences and evaporation.
- **Freshwater Flux**: Models precipitation and evaporation to compute net freshwater flux and salinity change, ensuring mass conservation.
- **Moisture Advection**: Calculates moisture transport in the atmosphere using finite difference methods.
- **Turbulent Mixing**: Models mixing driven by temperature gradients and wind speed, applied to ocean and atmosphere fields.
- **CO_2 Flux**: Computes CO_2 exchange between ocean and atmosphere, ensuring mass conservation.
- **Radiative Flux**: Models solar and longwave radiation, incorporating greenhouse effects from CO_2 concentrations.
- **Logging and Error Handling**: Logs initialization and computation progress using the `logging` module, with exception handling to catch and log errors.

2 Simulation Logic

The simulation logic in `TwoWayCoupling.py` revolves around the `TwoWayCoupling` class, which provides methods to compute fluxes and mixing terms used in the time stepping of `Model.py`.

2.1 Initialization

- **Purpose**: Sets up physical parameters and constants for flux and mixing calculations.
- **Process**:
  - Initializes parameters: `drag_coeff`, `wind_speed`, `precip_rate`, `evap_rate`, `solar_forcing`, `longwave_coeff`, `mixing_coeff`, `co2_transfer_coeff`, `freshwater_conservation_coeff` (default 1.0), `co2_conservation_coeff` (default 1.0).
  - Sets physical constants: $\rho_{\text{air}} = 1.225\,\text{kg/m}^3$, $\rho_{\text{water}} = 1025\,\text{kg/m}^3$, $C_p^{\text{air}} = 1005\,\text{J/kg/K}$, $L_v = 2.5 \times 10^6\,\text{J/kg}$, $\alpha = 0.03$ (CO_2 solubility).
  - Logs initialization details and handles exceptions.

2.2 Flux and Mixing Computations

- **Purpose**: Computes fluxes and mixing terms for use in `Model.py`'s time stepping.
- **Process**: Each method (`compute_sea_surface_roughness`, `compute_momentum_flux` etc.) performs the following:
  1. Logs the start of computation.
  2. Applies the relevant physical equations (detailed in Section 4).
  3. Clips outputs to predefined ranges to ensure numerical stability.
  4. Returns the computed values for use in `Model.py`.
  5. Handles exceptions and logs errors if computations fail.

3 Physics and Mathematical Models

The `TwoWayCoupling` class implements physical models for ocean-atmosphere interactions, with equations formatted to fit within margins.

3.1 Sea Surface Roughness

- **Purpose**: Adjusts the drag coefficient based on wind and ocean currents to account for sea surface roughness.

- **Equations**:

$$
\begin{aligned}
u_* &= \sqrt{\frac{\rho_{\text{air}} C_d U^2}{\rho_{\text{water}}}}, \\
z_0 &= \alpha \frac{u_*^2 + 0.1(u_{\text{ocean}}^2 + v_{\text{ocean}}^2)}{g}, \\
C_d' &= C_d \left(1 + 0.1 \log_{10}(z_0)\right),
\end{aligned}
$$

  where u_* is friction velocity, C_d is the drag coefficient, U is wind speed, $\rho_{\text{air}} = 1.225$ kg/m^3, $\rho_{\text{water}} = 1025$ kg/m^3, $\alpha = 0.018$, $g = 9.81$ m/s^2, and z_0 is roughness length.

- **Implementation**: `compute_sea_surface_roughness` clips z_0 to $[10^{-6}, 10^{-2}]$ and C_d' to $[10^{-4}, 10^{-2}]$. Ocean speed is computed as $\sqrt{u_{\text{ocean}}^2 + v_{\text{ocean}}^2}$.

3.2 Momentum Flux

- **Purpose**: Computes wind stress on the ocean surface.

- **Equation**:

$$
\tau = \rho_{\text{air}} C_d' U^2,
$$

  where C_d' is the adjusted drag coefficient from `compute_sea_surface_roughness`.

- **Implementation**: `compute_momentum_flux` clips U^2 to $[0, 10^3]$ and τ to $[-10^5, 10^5]$ N/m^2.

3.3 Heat Flux

- **Purpose**: Computes sensible and latent heat fluxes across the air-sea interface.

- **Equations**:

$$
\begin{aligned}
k_{\text{sensible}} &= 0.01 \frac{C_d'}{C_d}, \\
Q_{\text{sensible}} &= \rho_{\text{air}} C_p^{\text{air}} k_{\text{sensible}} (T_a - T_o), \\
Q_{\text{latent}} &= \rho_{\text{air}} L_v E \,\text{sign}(T_a - T_o), \\
Q_{\text{total}} &= Q_{\text{sensible}} + Q_{\text{latent}},
\end{aligned}
$$

  where $C_p^{\text{air}} = 1005$ J/kg/K, $L_v = 2.5 \times 10^6$ J/kg, E is evaporation rate, T_a is atmosphere temperature, and T_o is ocean temperature.

- **Implementation**: `compute_heat_flux` clips $T_a - T_o$ to $[-100, 100]$ K and Q_{total} to $[-10^6, 10^6]$ W/m^2.

3.4 Freshwater Flux

- **Purpose**: Computes net freshwater flux and salinity change, ensuring mass conservation.
- **Equations**:

$$
\begin{aligned}
P &= P_0 \left(1 + 0.5 \frac{q}{0.01}\right), \\
E &= E_0 C_{\text{freshwater}} \left(1 + 0.1 \frac{S}{35}\right), \\
F &= P - E, \\
\frac{dS}{dt} &= -\frac{SF}{\rho_{\text{water}} H},
\end{aligned}
$$

  where P_0 is precipitation rate, E_0 is evaporation rate, $C_{\text{freshwater}}$ is the freshwater conservation coefficient, S is salinity, q is moisture, $\rho_{\text{water}} = 1025$ kg/m^3, and $H = 1000$ m is ocean depth.

- **Implementation**: `compute_freshwater_flux` clips $q/0.01$ to $[0, 2]$, $S/35$ to $[0.8, 1.2]$, and outputs $\frac{dS}{dt}$ and F to $[-10^{-3}, 10^{-3}]$.

3.5 Moisture Advection

- **Purpose**: Computes moisture transport in the atmosphere.
- **Equation**:

$$
\text{Advection} = -u_{\text{atm}} \frac{\partial q}{\partial x} - v_{\text{atm}} \frac{\partial q}{\partial y},
$$

  where:

$$
\begin{aligned}
\frac{\partial q}{\partial x} &\approx \frac{q_{i+1,j} - q_{i-1,j}}{2\Delta x_{i,j}}, \\
\frac{\partial q}{\partial y} &\approx \frac{q_{i,j+1} - q_{i,j-1}}{2\Delta y_{i,j}},
\end{aligned}
$$

  and $u_{\text{atm}}, v_{\text{atm}}$ are atmosphere velocities.

- **Implementation**: `compute_moisture_advection` uses central differences, handles scalar or array velocities, and clips output to $[-10^{-4}, 10^{-4}]$.

3.6 Turbulent Mixing

- **Purpose**: Models mixing driven by temperature gradients and wind speed.

- **Equations**:

$$\begin{aligned}
u_* &= \sqrt{\frac{\rho_{\text{air}} C_d U^2}{\rho_{\text{water}}}}, \\
M &= k_m u_* \left(\frac{\partial T}{\partial x} + \frac{\partial T}{\partial y} \right), \\
\frac{\partial T}{\partial x} &\approx \frac{T_{i+1,j} - T_{i-1,j}}{2\Delta x_{i,j}}, \\
\frac{\partial T}{\partial y} &\approx \frac{T_{i,j+1} - T_{i,j-1}}{2\Delta y_{i,j}},
\end{aligned}$$

where k_m is the mixing coefficient, and C_d is the drag coefficient.

- **Implementation**: `compute_turbulent_mixing` clips gradients to $[-10^3, 10^3]$ and output to $[-10^3, 10^3]$.

3.7 CO_2 Flux

- **Purpose**: Computes CO_2 exchange between ocean and atmosphere, ensuring mass conservation.

- **Equations**:

$$\begin{aligned}
p\text{CO}_{2,\text{ocean}} &= \frac{\text{CO}_{2,\text{ocean}}}{\alpha}, \\
p\text{CO}_{2,\text{atm}} &= \text{CO}_{2,\text{atm}}, \\
F_{\text{CO}_2} &= k_{\text{CO}_2} C_{\text{CO}_2} (p\text{CO}_{2,\text{ocean}} - p\text{CO}_{2,\text{atm}}), \\
F_{\text{CO}_2,\text{ocean}} &= \frac{F_{\text{CO}_2}}{H}, \\
F_{\text{CO}_2,\text{atm}} &= -\frac{F_{\text{CO}_2}}{H_{\text{atm}}},
\end{aligned}$$

where $\alpha = 0.03$ is CO_2 solubility, k_{CO_2} is the transfer coefficient, C_{CO_2} is the CO_2 conservation coefficient, $H = 1000$ m is ocean depth, and $H_{\text{atm}} = 10000$ m is atmosphere height.

- **Implementation**: `compute_co2_flux` clips outputs to $[-10^{-3}, 10^{-3}]$.

3.8 Radiative Flux

- **Purpose**: Models solar and longwave radiation with greenhouse effects.

- **Equation**:

$$\begin{aligned}
Q_{\text{rad}} = Q_{\text{solar}} - \varepsilon\sigma \left(\frac{T}{300} \right)^4 \\
\times \left(1 + 0.1 \log \left(\frac{\text{CO}_{2,\text{atm}}}{400} \right) \right),
\end{aligned}$$

where $\sigma = 5.67 \times 10^{-8}$ W/m^2/K^4, ε is the longwave coefficient, and Q_{solar} is solar forcing.

- **Implementation**: `compute_radiative_flux` clips $T/300$ to $[0.8, 1.2]$, $\log(\text{CO}_{2,\text{atm}}/400)$ to $[-1, 1]$, and output to $[-10^6, 10^6]$ W/m^2.

4 Algorithms

4.1 Initialization Algorithm

- **Input**: Parameters (`drag_coeff`, `wind_speed`, `precip_rate`, `evap_rate`, `solar_forcing`, `longwave_coeff`, `mixing_coeff`, `co2_transfer_coeff`, `freshwater_conservation_coeff`, `co2_conservation_coeff`).
- **Steps**:
  1. Log initialization parameters.
  2. Store input parameters as instance variables.
  3. Set physical constants: $\rho_{\text{air}} = 1.225$ kg/m^3, $\rho_{\text{water}} = 1025$ kg/m^3, $C_p^{\text{air}} = 1005$ J/kg/K, $L_v = 2.5 \times 10^6$ J/kg, $\alpha = 0.03$.
  4. Log completion.
  5. Handle exceptions and log errors if initialization fails.

4.2 Sea Surface Roughness Algorithm (`compute_sea_surface_roughnes`

- **Input**: `wind_speed`, u_{ocean}, v_{ocean}.
- **Steps**:
  1. Log start of computation.
  2. Set $g = 9.81$ m/s^2, $\alpha = 0.018$.
  3. Compute friction velocity: $u_* = \sqrt{\frac{\rho_{\text{air}} C_d U^2}{\rho_{\text{water}}}}$.
  4. Compute ocean speed: $\sqrt{u_{\text{ocean}}^2 + v_{\text{ocean}}^2}$.
  5. Compute roughness length: $z_0 = \alpha \frac{u_*^2 + 0.1(u_{\text{ocean}}^2 + v_{\text{ocean}}^2)}{g}$, clipped to $[10^{-6}, 10^{-2}]$.
  6. Compute adjusted drag coefficient: $C_d' = C_d(1 + 0.1 \log_{10}(z_0))$, clipped to $[10^{-4}, 10^{-2}]$.
  7. Return C_d'.

8. Log errors if computation fails.

4.3 Momentum Flux Algorithm (`compute_momentum_flux`)

- **Input**: `wind_speed`, u_{ocean}, v_{ocean}.
- **Steps**:
  1. Log start of computation.
  2. Compute C_d' using `compute_sea_surface_roughness`.
  3. Compute wind stress: $\tau = \rho_{\text{air}} C_d' U^2$, with U^2 clipped to $[0, 10^3]$.
  4. Clip τ to $[-10^5, 10^5]$ N/m^2.
  5. Return τ.
  6. Log errors if computation fails.

4.4 Heat Flux Algorithm (`compute_heat_flux`)

- **Input**: T_a, T_o, u_{ocean}, v_{ocean}.
- **Steps**:
  1. Log start of computation.
  2. Compute C_d' using `compute_sea_surface_roughness`.
  3. Compute sensible heat coefficient: $k_{\text{sensible}} = 0.01 \frac{C_d'}{C_d}$.
  4. Compute sensible heat flux: $Q_{\text{sensible}} = \rho_{\text{air}} C_p^{\text{air}} k_{\text{sensible}} (T_a - T_o)$, with $T_a - T_o$ clipped to $[-100, 100]$ K.
  5. Compute latent heat flux: $Q_{\text{latent}} = \rho_{\text{air}} L_v E \text{sign}(T_a - T_o)$.
  6. Compute total heat flux: $Q_{\text{total}} = Q_{\text{sensible}} + Q_{\text{latent}}$, clipped to $[-10^6, 10^6]$ W/m^2.
  7. Return Q_{total}.
  8. Log errors if computation fails.

4.5 Freshwater Flux Algorithm (`compute_freshwater_flux`)

- **Input**: S, q, `ocean_depth` (default 1000 m).
- **Steps**:
  1. Log start of computation.

2. Compute precipitation: $P = P_0(1 + 0.5\frac{q}{0.01})$, with $q/0.01$ clipped to $[0, 2]$.
3. Compute evaporation: $E = E_0 C_{\text{freshwater}}(1 + 0.1\frac{S}{35})$, with $S/35$ clipped to $[0.8, 1.2]$.
4. Compute net freshwater flux: $F = P - E$, clipped to $[-10^{-3}, 10^{-3}]$.
5. Compute salinity change: $\frac{dS}{dt} = -\frac{SF}{\rho_{\text{water}} H}$, clipped to $[-10^{-3}, 10^{-3}]$.
6. Return $\frac{dS}{dt}$, F.
7. Log errors if computation fails.

4.6 Moisture Advection Algorithm (`compute_moisture_advection`)

- **Input**: q, Δx, Δy, `step`, u_{atm}, v_{atm}.
- **Steps**:
  1. Log start of computation.
  2. Initialize advection array (zeros, same shape as q).
  3. For each grid point (i, j):
     - Compute u_{ij}, v_{ij} (from $u_{\text{atm}}, v_{\text{atm}}$, handling scalar or array inputs).
     - $\text{adv}_x = -u_{ij}\frac{q_{i+1,j} - q_{i-1,j}}{2\Delta x_{i,j}}$.
     - $\text{adv}_y = -v_{ij}\frac{q_{i,j+1} - q_{i,j-1}}{2\Delta y_{i,j}}$.
     - $\text{advection}[i, j] = \text{adv}_x + \text{adv}_y$, clipped to $[-10^{-4}, 10^{-4}]$.
  4. Return advection array.
  5. Log errors if computation fails.

4.7 Turbulent Mixing Algorithm (`compute_turbulent_mixing`)

- **Input**: T, S, Δx, Δy, `wind_speed`.
- **Steps**:
  1. Log start of computation.
  2. Compute friction velocity: $u_* = \sqrt{\frac{\rho_{\text{air}} C_d U^2}{\rho_{\text{water}}}}$.
  3. Initialize mixing array (zeros, same shape as T).

4. For each grid point (i, j):
   - $\text{grad}_x = \frac{T_{i+1,j} - T_{i-1,j}}{2\Delta x_{i,j}}$.
   - $\text{grad}_y = \frac{T_{i,j+1} - T_{i,j-1}}{2\Delta y_{i,j}}$.
   - $\text{mixing}[i, j] = k_m u_* (\text{grad}_x + \text{grad}_y)$, with gradients clipped to $[-10^3, 10^3]$.
5. Clip mixing to $[-10^3, 10^3]$.
6. Return mixing array.
7. Log errors if computation fails.

4.8 CO_2 Flux Algorithm (`compute_co2_flux`)

- **Input**: $CO_{2,\text{ocean}}$, $CO_{2,\text{atm}}$, `ocean_depth`$(default 1000m)$, `atm_height`$(default 10000m)$.**Steps**
-
  - Log start of computation.
  - Compute partial pressures: $pCO_{2,\text{ocean}} = \frac{CO_{2,\text{ocean}}}{\alpha}$, $pCO_{2,\text{atm}} = CO_{2,\text{atm}}$.
  - Compute flux: $F_{CO_2} = k_{CO_2} C_{CO_2} (pCO_{2,\text{ocean}} - pCO_{2,\text{atm}})$.
  - Normalize fluxes: $F_{CO_2,\text{ocean}} = \frac{F_{CO_2}}{H}$, $F_{CO_2,\text{atm}} = -\frac{F_{CO_2}}{H_{\text{atm}}}$, clipped to $[-10^{-3}, 10^{-3}]$.
  - Return $F_{CO_2,\text{ocean}}$, $F_{CO_2,\text{atm}}$.
  - Log errors if computation fails.

4.9 Radiative Flux Algorithm (`compute_radiative_flux`)

- **Input**: T, $CO_{2,\text{atm}}$.**Steps** :
-
  - Log start of computation.
  - Set $\sigma = 5.67 \times 10^{-8}$ W/m^2/K^4.
  - Compute normalized temperature: $T_{\text{norm}} = \frac{T}{300}$, clipped to $[0.8, 1.2]$.
  - Compute longwave radiation: $\text{longwave} = \varepsilon\sigma(T_{\text{norm}} \cdot 300)^4$.
  - Compute greenhouse effect: $\text{greenhouse} = 0.1 \log\left(\frac{CO_{2,\text{atm}}}{400}\right)$, clipped to $[-1, 1]$.
  - Compute radiative flux: $Q_{\text{rad}} = Q_{\text{solar}} - \text{longwave}(1 + \text{greenhouse})$, clipped to $[-10^6, 10^6]$ W/m^2.

- Return Q_{rad}.
- Log errors if computation fails.