Boundary Layer Schemes

June 29, 2025

1 Functionalities

The `BoundaryLayerSchemesWindow` class provides the following functionalities:

- **Initialization**: Sets up a window for boundary layer schemes analysis, integrating with the model to access ocean and atmosphere temperatures and coupling parameters.
- **Simulation Control**: Allows users to start and stop a simulation, updating parameters (drag coefficient, sensible heat coefficient, boundary layer depth, KPP mixing coefficient) and selecting between Bulk and KPP schemes.
- **Visualization**: Displays a heatmap of heat flux (Bulk scheme) or diffusivity (KPP scheme) with annotations and a scatter plot of flux/diffusivity versus temperature gradient, updated in real-time via animation.
- **Parameter Input**: Enables users to input simulation parameters and select schemes, with validation to ensure physical constraints.
- **Logging and Error Handling**: Logs initialization, simulation control, and plot updates using the `logging` module, with exception handling to display errors via message boxes.

2 Simulation Logic

The simulation logic in `BoundaryLayerSchemesWindow` centers around managing boundary layer schemes analysis and visualization.

2.1 Initialization

- **Purpose**: Initializes the analysis window with model integration and visualization setup.
- **Process**:

- Stores the model instance for access to ocean and atmosphere temperatures and coupling parameters.
- Sets up a window with a control panel for inputting drag coefficient, sensible heat coefficient, boundary layer depth, KPP mixing coefficient, and selecting Bulk/KPP schemes.
- Initializes a matplotlib figure with two subplots: one for a heatmap of heat flux or diffusivity with annotations, and one for a scatter plot of flux/diffusivity versus temperature gradient.
- Sets up a simulation grid (ny $\times$ nx) matching the model's ocean temperatures and initializes arrays for heat flux and diffusivity.
- Sets the time step $\Delta t = 1800$ s to match the main simulation.
- Sets up a timer for animation updates every 100 ms and initializes the plot.
- Logs initialization details and handles exceptions, displaying errors via `QMessageBox`.

2.2 Simulation Control

- **Purpose**: Manages the start and stop of the boundary layer schemes simulation.
- **Process (Start)**:
  - Validates inputs: drag coefficient (> 0), sensible heat coefficient (≥ 0), boundary layer depth (> 0), KPP mixing coefficient (> 0).
  - Updates model parameters (`model.coupling.drag_coeff`).
  - Resets simulation step, time steps, flux arrays, and diffusivity arrays.
  - Starts a timer to trigger animation updates every 100 ms.
  - Disables the start button and enables the stop button.
- **Process (Stop)**:
  - Stops the timer and animation.
  - Enables the start button and disables the stop button.
  - Logs actions and handles exceptions, displaying errors via `QMessageBox`.

2.3 Plot Update

- **Purpose**: Updates the visualization of heat flux or diffusivity and their relationship with temperature gradients.
- **Process**:
  - Checks if the simulation should continue (current step $<$ total_time$/\Delta t$).
  - Validates and updates parameters from user inputs and selected scheme.
  - Computes heat flux (Bulk scheme) or diffusivity and heat flux (KPP scheme) based on ocean and atmosphere temperatures and wind speed.
  - Stores flux and temperature gradient data for visualization.
  - Updates plots: a heatmap of heat flux or diffusivity with sparse annotations, and a scatter plot of flux/diffusivity versus temperature gradient colored by time.
  - Increments the simulation step and logs actions.

3 Physics and Mathematical Models

The `BoundaryLayerSchemesWindow` class implements models for boundary layer dynamics, comparing Bulk and KPP schemes.

3.1 Bulk Heat Flux (Bulk Scheme)

- **Purpose**: Computes sensible heat flux between ocean and atmosphere.
- **Equation**:
$$Q_b = \rho_{\text{air}} C_p^{\text{air}} C_h U (T_o - T_a),$$
  where $\rho_{\text{air}} = 1.225$ kg/m^3, $C_p^{\text{air}} = 1005$ J/kg/K, C_h is the sensible heat coefficient (W/m^2/K), U is wind speed (m/s), T_o is ocean temperature (K), and T_a is atmosphere temperature (K).
- **Implementation**: `update_plot` computes Q_b using user-input C_h and model temperature fields.

3.2 KPP Scheme

- **Purpose**: Computes turbulent diffusivity and heat flux using the K-Profile Parameterization (KPP) scheme.

- **Equations**:

$$K(z) = k_m \left(1 - \frac{z}{h}\right)^2,$$
$$Q_{\text{KPP}} = -K(z)\frac{\partial T_o}{\partial y},$$

where k_m is the KPP mixing coefficient (m^2/s), h is the boundary layer depth (m), $z \in [0, h]$, and $\frac{\partial T_o}{\partial y} \approx \frac{T_{o,i+1,j} - T_{o,i-1,j}}{2\Delta y}$. $K(z)$ is clipped to $[0, k_m]$.

- **Implementation**: `update_plot` computes $K(z)$ and Q_{KPP} using user-input k_m and h.

3.3 Temperature Gradient

- **Purpose**: Computes the temperature difference driving the heat flux.

- **Equation**:

$$\Delta T = T_o - T_a,$$

where ΔT is the temperature gradient (K).

- **Implementation**: `update_plot` computes ΔT for scatter plot visualization.

4 Algorithms

4.1 Initialization Algorithm

- **Input**: Model instance (`model`).

- **Steps**:

1. Log initialization.

2. Store `model` as an instance variable.

3. Set window title to "Boundary Layer Schemes Analysis" and size to 900×600.

4. Create a control panel with inputs for drag coefficient (`model.coupling.drag_coe` sensible heat coefficient ($10.0\,\text{W/m}^2/\text{K}$), boundary layer depth ($50.0\,\text{m}$), KPP mixing coefficient ($0.01\,\text{m}^2/\text{s}$), and a selector for Bulk/KPP schemes.

5. Create start and stop buttons, with stop initially disabled.

6. Initialize a matplotlib figure with two subplots: heatmap and scatter plot.

7. Set up simulation grid (ny $\times$ nx) from model's ocean temperatures and initialize arrays for heat flux and diffusivity.

8. Set $\Delta t = 1800$ s.

9. Initialize empty lists for time steps, bulk fluxes, KPP fluxes, and temperature gradients.

10. Set up a timer for 100 ms updates and call `update_plot(0)`.

11. Log completion or errors, displaying critical errors via `QMessageBox`.

4.2 Start Simulation Algorithm (`start_simulation`)

- **Input**: None (uses input fields).
- **Steps**:

  1. Log start of simulation.

  2. Read and validate inputs: drag coefficient (> 0), sensible heat coefficient (≥ 0), boundary layer depth (> 0), KPP mixing coefficient (> 0).

  3. Update `model.coupling.drag_coeff`.

  4. Reset `current_step`, `time_steps`, `bulk_fluxes`, `kpp_fluxes`, `temp_gradients`, and heat flux/diffusivity arrays.

  5. Start timer (100 ms interval).

  6. Disable start button and enable stop button.

  7. Log completion or errors, displaying warnings or critical errors via `QMessageBox`.

4.3 Stop Simulation Algorithm (`stop_simulation`)

- **Input**: None.
- **Steps**:

  1. Log stop of simulation.

  2. Stop timer and animation (`anim.event_source.stop()`).

  3. Set `anim` to None.

  4. Enable start button and disable stop button.

  5. Log completion or errors, displaying critical errors via `QMessageBox`.

4.4 Plot Update Algorithm (update_plot)

- **Input**: Frame number (unused directly, tracks via current_step).
- **Steps**:
  1. Log start of update at current_step.
  2. If current_step $\geq$ total_time/Δt or model is invalid, call stop_simulation.
  3. Validate and read inputs: drag coefficient, sensible heat coefficient, boundary layer depth, KPP mixing coefficient, and scheme.
  4. Compute Bulk heat flux: $Q_b = \rho_{\text{air}} C_p^{\text{air}} C_h U (T_o - T_a)$.
  5. Compute KPP diffusivity and flux:
     - $K(z) = k_m \left(1 - \frac{z}{h}\right)^2$, clipped to $[0, k_m]$.
     - $Q_{\text{KPP}} = -K \frac{\partial T_o}{\partial y}$, with $\frac{\partial T_o}{\partial y} \approx \frac{T_{o,i+1,j} - T_{o,i-1,j}}{2\Delta y}$.
  6. Update fields: Q_b for Bulk scheme, $K(z)$ for KPP scheme.
  7. Store flattened Q_b, Q_{KPP}, and $\Delta T = T_o - T_a$ for scatter plot.
  8. Update plots:
     - Left: Heatmap of Q_b (Bulk, coolwarm) or $K(z)$ (KPP, viridis) with annotations every ny$/5 \times$ nx$/5$.
     - Right: Scatter plot of Q_b or $K(z)$ vs. ΔT, colored by time (plasma).
  9. Increment current_step.
  10. Log completion or errors, displaying warnings or critical errors via QMessageBox.