Surface Layer Physics

June 29, 2025

1 Functionalities

The `SurfaceLayerPhysicsWindow` class provides the following functionalities:

- **Initialization**: Sets up a window for surface layer physics analysis, integrating with the model to access ocean and atmosphere temperatures and coupling parameters.
- **Simulation Control**: Allows users to start and stop a surface layer physics simulation, updating model parameters (drag coefficient, wind speed, sensible and latent heat coefficients) dynamically.
- **Visualization**: Displays a heatmap of surface currents and time series of mean heat flux and wind stress, updated in real-time via animation.
- **Parameter Input**: Enables users to input drag coefficient, wind speed, sensible heat coefficient, and latent heat coefficient, 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 `SurfaceLayerPhysicsWindow` centers around managing the surface layer physics analysis and visualization.

2.1 Initialization

- **Purpose**: Initializes the surface layer physics 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, wind speed, sensible heat coefficient, and latent heat coefficient, and buttons for starting/stopping the simulation.
  - Initializes a matplotlib figure with two subplots: one for a heatmap of surface currents and one for time series of heat flux and wind stress.
  - Sets up a simulation grid matching the model's ocean temperature dimensions (ny $\times$ nx) and initializes surface currents to zero.
  - 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 surface layer physics simulation.
- **Process (Start)**:
  - Validates inputs: drag coefficient (> 0), wind speed (≥ 0), sensible heat coefficient (≥ 0), latent heat coefficient (≥ 0).
  - Updates model parameters (`model.coupling.drag_coeff`, `model.coupling.wi`
  - Resets the simulation step, time steps, heat fluxes, wind stresses, and surface currents.
  - 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 surface currents, mean heat flux, and wind stress.
- **Process**:
  - Checks if the simulation should continue (current step $<$ total_time$/\Delta t$).
  - Validates and updates model parameters from user inputs.
  - Computes sensible and latent heat fluxes based on ocean and atmosphere temperatures and wind speed.
  - Computes wind stress based on drag coefficient and wind speed.
  - Updates surface currents using wind stress with a random perturbation scaled by the mixing coefficient.
  - Stores mean heat flux and wind stress for time series visualization.
  - Updates plots: a heatmap of surface currents and a time series of mean heat flux and wind stress.
  - Increments the simulation step and logs actions.

3 Physics and Mathematical Models

The `SurfaceLayerPhysicsWindow` class implements models for surface layer dynamics, focusing on heat fluxes and wind stress.

3.1 Sensible Heat Flux

- **Purpose**: Computes the sensible heat flux due to temperature differences between ocean and atmosphere.
- **Equation**:
$$Q_s = \rho_{\text{air}} C_p^{\text{air}} C_h U (T_o - T_a),$$
where $\rho_{\text{air}} = 1.225\,\text{kg/m}^3$, $C_p^{\text{air}} = 1005\,\text{J/kg/K}$, 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_s using user-input C_h and model temperature fields.

3.2 Latent Heat Flux

- **Purpose**: Computes the latent heat flux due to evaporation.

- **Equation**:
$$Q_l = \rho_{\text{air}} L_v C_e U q,$$
where $L_v = 2.5 \times 10^6$ J/kg, C_e is the latent heat coefficient (W/m^2), $q = 0.001$ is a placeholder specific humidity difference, and other variables are as above.

- **Implementation**: `update_plot` uses a constant q for simplicity, with user-input C_e.

3.3 Total Heat Flux

- **Purpose**: Combines sensible and latent heat fluxes.

- **Equation**:
$$Q_{\text{total}} = Q_s + Q_l,$$
where Q_{total} is the total heat flux (W/m^2).

- **Implementation**: `update_plot` computes the mean of Q_{total} for time series visualization.

3.4 Wind Stress

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

- **Equation**:
$$\tau = \rho_{\text{air}} C_d U^2,$$
where C_d is the drag coefficient, and other variables are as above.

- **Implementation**: `update_plot` uses user-input C_d and U.

3.5 Surface Currents

* **Purpose**: Updates surface currents driven by wind stress with turbulent mixing effects.

* **Equation**:
$$\mathbf{u}_{\text{surface}}(t + \Delta t) = \mathbf{u}_{\text{surface}}(t) + \Delta t \cdot \frac{\tau}{\rho_{\text{water}}} \cdot \mathbf{n},$$
$$\mathbf{n} \sim \mathcal{N}(0, k_m),$$
where $\rho_{\text{water}} = 1000\,\text{kg/m}^3$, k_m is the mixing coefficient, $\mathbf{n}$ is a random normal vector, and $\mathbf{u}_{\text{surface}}$ is clipped to $[-0.5, 0.5]$ m/s.

* **Implementation**: `update_plot` updates currents with a random perturbation scaled by k_m.

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 "Surface Layer Physics Analysis" and size to 900×600.
4. Create a control panel with inputs for drag coefficient (`model.coupling.d` wind speed (`model.coupling.wind_speed`), sensible heat coefficient (10.0 W/m^2/K), and latent heat coefficient (20.0 W/m^2).
5. Create start and stop buttons, with stop initially disabled.
6. Initialize a matplotlib figure with two subplots: heatmap and time series.
7. Set up simulation grid (ny $\times$ nx) from model's ocean temperatures and initialize surface currents to zero.
8. Set $\Delta t = 1800$ s.
9. Initialize empty lists for time steps, heat fluxes, and wind stresses.
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), wind speed (≥ 0), sensible heat coefficient (≥ 0), latent heat coefficient (≥ 0).
3. Update `model.coupling.drag_coeff` and `model.coupling.wind_spe`

4. Reset `current_step`, `time_steps`, `heat_fluxes`, `wind_stresses`, and `surface_currents`.

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$/\Delta t$ or model is invalid, call `stop_simulation`.

3. Validate and read inputs: drag coefficient, wind speed, sensible and latent heat coefficients.

4. Compute sensible heat flux: $Q_s = \rho_{\text{air}} C_p^{\text{air}} C_h U (T_o - T_a)$.

5. Compute latent heat flux: $Q_l = \rho_{\text{air}} L_v C_e U q$, with $q = 0.001$.

6. Compute total heat flux: $Q_{\text{total}} = Q_s + Q_l$.

7. Compute wind stress: $\tau = \rho_{\text{air}} C_d U^2$.

8. Update surface currents: $\mathbf{u}_{\text{surface}} += \Delta t \cdot (\tau / 1000) \cdot \mathbf{n}$, where $\mathbf{n} \sim \mathcal{N}(0, k_m)$, clipped to $[-0.5, 0.5]$ m/s.

9. Append mean Q_{total}, τ, and current time (step $\cdot \Delta t$) to respective lists.

10. Clear and update plots:

11. Top: Heatmap of $\mathbf{u}_{\text{surface}}$ with `coolwarm` colormap, $[-0.5, 0.5]$ m/s.

12. Bottom: Time series of mean Q_{total} and τ.

13. Increment `current_step`.

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