Turbulent Mixing

June 29, 2025

1 Functionalities

The `TurbulentMixingWindow` class provides the following functionalities:

- **Initialization**: Sets up a window for turbulent mixing analysis, integrating with the model to access ocean temperatures, salinity, and coupling parameters.
- **Simulation Control**: Allows users to start and stop a turbulent mixing simulation, updating model parameters (mixing coefficient, wind speed) dynamically.
- **Visualization**: Displays a thermal front with mixing gradients and an Ekman spiral with vertical temperature profiles, updated in real-time via animation.
- **Parameter Input**: Enables users to input mixing coefficient and wind speed, 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 `TurbulentMixing.py` centers around the `TurbulentMixingWindow` class, which manages the turbulent mixing analysis and visualization.

2.1 Initialization

- **Purpose**: Initializes the turbulent mixing analysis window with model integration and visualization setup.
- **Process**:
  - Stores the model instance for access to ocean temperatures, salinity,

and coupling parameters.

  – Sets up a window with a control panel for inputting mixing coefficient and wind speed, and buttons for starting/stopping the simulation.

  – Initializes a matplotlib figure with two subplots: one for the thermal front and mixing gradients, and one for the Ekman spiral and vertical temperature profile.

  – Configures a vertical grid (z) from 0 to -100 m with 50 points ($\Delta z = -2$ m) and an initial linear temperature profile from 300 K to 290 K.

  – Sets up a timer for animation updates 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 turbulent mixing simulation.
- **Process (Start)**:
  – Validates inputs: mixing coefficient (> 0) and wind speed (≥ 0).

  – Updates model parameters (`model.coupling.mixing_coeff`, `model.coupling.w`

  – Resets the simulation step and temperature profile.

  – 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 the thermal front, mixing gradients, Ekman spiral, and vertical temperature profile.
- **Process**:
  – Checks if the simulation should continue (current step $<$ total_time$/\Delta t$).

- Validates and updates model parameters from user inputs.
- Generates a synthetic thermal front based on ocean temperatures and a sinusoidal perturbation driven by wind speed.
- Computes turbulent mixing using `model.coupling.compute_turbulent_mixing`
- Calculates mixing gradients for visualization.
- Computes Ekman spiral velocities (u, v) and updates the vertical temperature profile using a diffusion equation.
- Updates the plots: a contour plot with quiver arrows for the thermal front and mixing gradients, and a line plot for the Ekman spiral and temperature profile.
- Increments the simulation step and logs actions.

3 Physics and Mathematical Models

The `TurbulentMixingWindow` class implements models for visualizing turbulent mixing and Ekman dynamics, using numerical approximations for gradients and diffusion.

3.1 Thermal Front

- **Purpose**: Generates a synthetic thermal front to visualize turbulent mixing effects.
- **Equation**:

$$T_{\text{front}}(x, y) = T_{\text{cold}} + \frac{T_{\text{hot}} - T_{\text{cold}}}{2} \left[1 + \tanh\left(\frac{x - x_0}{L}\right)\right],$$

  where $T_{\text{cold}} = 290\,\text{K}$, $T_{\text{hot}} = 300\,\text{K}$, $L = 5.0$, and $x_0 = \frac{\text{nx}}{2} + 10 \sin(0.01 \cdot \text{step} \cdot U)$, with U as wind speed.
- **Implementation**: `update_plot` blends the front with ocean temperatures: $T_{\text{ocean}} = 0.8 \cdot T_{\text{ocean}} + 0.2 \cdot T_{\text{front}}$.

3.2 Mixing Gradients

- **Purpose**: Visualizes the spatial variation of turbulent mixing.

- **Equation**:

$$
\begin{aligned}
\nabla M &= \left(\frac{\partial M}{\partial x}, \frac{\partial M}{\partial y} \right), \\
\frac{\partial M}{\partial x} &\approx \frac{M_{i+1,j} - M_{i-1,j}}{2\Delta x}, \\
\frac{\partial M}{\partial y} &\approx \frac{M_{i,j+1} - M_{i,j-1}}{2\Delta y},
\end{aligned}
$$

  where M is the mixing field from `compute_turbulent_mixing`.

- **Implementation**: `update_plot` uses `np.gradient` to compute gradients, visualized with quiver arrows on a subsampled grid (every 5th point).

3.3 Ekman Spiral Velocities

- **Purpose**: Models the velocity profile in the ocean surface layer due to wind stress and Coriolis effects.

- **Equations**:

$$
\begin{aligned}
V_0 &= \frac{\tau}{\rho\sqrt{2\nu f}}, \\
d &= \sqrt{\frac{2\nu}{f}}, \\
u(z) &= V_0 e^{z/d} \cos\left(\frac{\pi}{4} + \frac{z}{d} \right), \\
v(z) &= V_0 e^{z/d} \sin\left(\frac{\pi}{4} + \frac{z}{d} \right),
\end{aligned}
$$

  where $\tau = 0.1U^2$ is wind stress, $\rho = 1000\,\mathrm{kg/m^3}$, ν is the mixing coefficient, $f = 10^{-4}\,\mathrm{s^{-1}}$ is the Coriolis parameter, and $z \in [0, -100]\,\mathrm{m}$.

- **Implementation**: `update_plot` computes u and v along the vertical grid for visualization.

3.4 Vertical Temperature Diffusion

- **Purpose**: Updates the vertical temperature profile due to turbulent mixing.

- **Equations**:

$$
\begin{aligned}
\frac{dT}{dt} &= K_v(z) \frac{\partial^2 T}{\partial z^2}, \\
K_v(z) &= \nu e^{z/50}, \\
\frac{\partial^2 T}{\partial z^2} &\approx \frac{T_{k+1} - 2T_k + T_{k-1}}{\Delta z^2},
\end{aligned}
$$

  where ν is the mixing coefficient, $\Delta z = -2\,\mathrm{m}$, and boundary conditions set $\frac{dT}{dt} = 0$ at the surface and $\frac{dT}{dt}_{k=-1} = \frac{dT}{dt}_{k=-2}$ at the bottom.

- **Implementation**: `update_plot` applies the diffusion equation with $\Delta t = 0.1$ s, updating the temperature profile and setting the surface temperature to the ocean temperature at the grid center.

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 "Turbulent Mixing Analysis" and size to 900×600.
4. Create a control panel with inputs for mixing coefficient (`model.coupling.mix` and wind speed (`model.coupling.wind_speed`).
5. Create start and stop buttons, with stop initially disabled.
6. Set up a matplotlib figure with two subplots.
7. Initialize a vertical grid: $z = [0, -100]$ m, nz $= 50$, $\Delta z = -2$ m.
8. Initialize a linear temperature profile: $T = [300, 290]$ K.
9. Set $\Delta t = 0.1$ s and initialize a timer for 100 ms updates.
10. Call `update_plot(0)` to initialize the visualization.
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: mixing coefficient (> 0), wind speed (≥ 0).
3. Update `model.coupling.mixing_coeff` and `model.coupling.wind_speed`.

4. Reset `current_step` to 0 and temperature profile to linear $[300, 290]$ K.
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 update mixing coefficient (> 0) and wind speed (≥ 0).
4. Generate thermal front:
   * Compute $x_0 = \text{nx}/2 + 10\sin(0.01 \cdot \text{step} \cdot U)$.
   * Compute $T_{\text{front}} = 290 + 5(1 + \tanh((x - x_0)/5))$.
   * Blend: $T_{\text{ocean}} = 0.8 \cdot T_{\text{ocean}} + 0.2 \cdot T_{\text{front}}$.
5. Compute mixing field using `model.coupling.compute_turbulent_mixing`.
6. Compute gradients: $\nabla M = (\frac{\partial M}{\partial x}, \frac{\partial M}{\partial y})$ using `np.gradient`.
7. Subsample gradients every 5th point for quiver plot.

8. Compute Ekman spiral:
   * $\tau = 0.1U^2$, $f = 10^{-4}\,\text{s}^{-1}$, $\rho = 1000\,\text{kg/m}^3$.
   * $V_0 = \tau/(\rho\sqrt{2\nu f})$, $d = \sqrt{2\nu/f}$.
   * $u(z) = V_0 e^{z/d}\cos(\pi/4 + z/d)$, $v(z) = V_0 e^{z/d}\sin(\pi/4 + z/d)$.
9. Update temperature profile:
   * Set surface temperature to $T_{\text{ocean}}[\text{ny}/2, \text{nx}/2]$.
   * Compute $K_v(z) = \nu e^{z/50}$.
   * Compute $\frac{\partial^2 T}{\partial z^2} = \frac{T_{k+1} - 2T_k + T_{k-1}}{\Delta z^2}$.
   * Update: $T += \Delta t \cdot K_v \cdot \frac{\partial^2 T}{\partial z^2}$, with boundary conditions.
10. Clear and update plots:
    * Left: Contour plot of T_{ocean} with quiver arrows for ∇M.
    * Right: Line plots of $u(z)$, $v(z)$, and $T(z)$.
11. Increment `current_step`.
12. Log completion or errors, displaying warnings or critical errors via `QMessageBox`.