High-Order Time Stepping

June 29, 2025

1 Functionalities

The `HighOrderTimeSteppingWindow` class provides the following functionalities:

- **Initialization**: Sets up a window for high-order time stepping 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 (time step, drag coefficient, sensible heat coefficient, boundary layer depth, KPP mixing coefficient) and selecting between Bulk and KPP schemes and Euler or RK4 methods.
- **Visualization**: Displays a heatmap of heat flux or diffusivity and a time series of mean values, comparing Euler and RK4 methods, updated in real-time via animation.
- **Parameter Input**: Enables users to input simulation parameters and select schemes and methods, 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 `HighOrderTimeSteppingWindow` centers around managing high-order time stepping 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 time step, drag coefficient, sensible heat coefficient, boundary layer depth, KPP mixing coefficient, and selecting Bulk/KPP schemes and Euler/RK4 methods.
  - Initializes a matplotlib figure with two subplots: one for a heatmap of heat flux or diffusivity, and one for a time series of mean values.
  - Sets up a simulation grid (ny $\times$ nx) matching the model's ocean temperatures and initializes arrays for Euler and RK4 heat fluxes and diffusivities.
  - Sets default time step $\Delta t = 1800$ s.
  - 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 time stepping simulation.
- **Process (Start)**:
  - Validates inputs: time step (> 0), drag coefficient (> 0), sensible heat coefficient (≥ 0), boundary layer depth (> 0), KPP mixing coefficient (> 0).
  - Updates model parameters (`model.coupling.drag_coeff`) and sets `dt`.
  - Resets simulation step, time steps, Euler/RK4 values, and heat flux/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 mean values over time.
- **Process**:
  - Checks if the simulation should continue (current step $<$ total_time$/\Delta t$).
  - Validates and updates parameters from user inputs and selected scheme/method.
  - Computes heat flux (Bulk scheme) or diffusivity (KPP scheme) using Euler and RK4 methods.
  - Stores mean values for time series visualization.
  - Updates plots: a heatmap of heat flux or diffusivity with annotations, and a time series comparing Euler and RK4 mean values.
  - Increments the simulation step and logs actions.

3 Physics and Mathematical Models

The `HighOrderTimeSteppingWindow` class implements models for heat flux and diffusivity, comparing Euler and RK4 time stepping methods.

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 for the Euler method directly and as the initial step for RK4.

3.2 KPP Diffusivity (KPP Scheme)

- **Purpose**: Computes diffusivity for the K-Profile Parameterization (KPP) scheme.

- **Equation**:

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

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

- **Implementation**: `update_plot` computes $K(z)$ and Q_{KPP} for the Euler method and as the initial step for RK4.

3.3 Euler Method

- **Purpose**: Updates heat flux or diffusivity using a first-order explicit method.

- **Equation**:

$$Q(t + \Delta t) = Q(t),$$

where Q is Q_b (Bulk) or Q_{KPP} (KPP), computed directly from the current state.

- **Implementation**: `update_plot` applies the equations above for Euler updates.

3.4 RK4 Method

* **Purpose**: Updates heat flux or diffusivity using a fourth-order Runge-Kutta method.

* **Equations**:

$$
\begin{aligned}
k_1 &= f(T_o, T_a), \\
k_2 &= f\left(T_o + \frac{\Delta t}{2}\frac{k_1}{C_h}, T_a + \frac{\Delta t}{2}\frac{k_1}{C_h}\right) \quad \text{(Bulk)}, \\
k_2 &= f(T_o, T_a) \quad \text{(KPP)}, \\
k_3 &= f\left(T_o + \frac{\Delta t}{2}\frac{k_2}{C_h}, T_a + \frac{\Delta t}{2}\frac{k_2}{C_h}\right) \quad \text{(Bulk)}, \\
k_3 &= f(T_o, T_a) \quad \text{(KPP)}, \\
k_4 &= f\left(T_o + \Delta t\frac{k_3}{C_h}, T_a + \Delta t\frac{k_3}{C_h}\right) \quad \text{(Bulk)}, \\
k_4 &= f(T_o, T_a) \quad \text{(KPP)}, \\
Q(t + \Delta t) &= Q(t) + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4),
\end{aligned}
$$

where f is the flux derivative function (Q_b for Bulk, Q_{KPP} for KPP), and C_h is omitted for KPP as temperatures are not updated directly.

* **Implementation**: `update_plot` computes intermediate steps using `compute_flux_derivative` and updates Q with the RK4 formula.

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 ”High-Order Time Stepping Analysis” and size to 900×600.
4. Create a control panel with inputs for time step (1800 s), drag coefficient (`model.coupling.drag_coeff`), sensible heat coefficient (10.0 W/m^2/K), boundary layer depth (50.0 m), KPP mixing coefficient (0.01 m^2/s), and selectors for Bulk/KPP schemes and Euler/RK4 methods.
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 arrays for Euler and RK4 heat fluxes/diffusivities.
8. Set default $\Delta t = 1800$ s.
9. Initialize empty lists for time steps and Euler/RK4 values.
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: time step (> 0), drag coefficient (> 0), sensible heat coefficient (≥ 0), boundary layer depth (> 0), KPP mixing coefficient (> 0).

3. Update `model.coupling.drag_coeff` and set `dt`.

4. Reset `current_step`, `time_steps`, `euler_values`, `rk4_values`, 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$/\Delta t$ or model is invalid, call `stop_simulation`.

3. Validate and read inputs: time step, sensible heat coefficient, boundary layer depth, KPP mixing coefficient, scheme, and method.

4. For Euler method:

5. Bulk: Compute $Q_b = \rho_{\text{air}} C_p^{\text{air}} C_h U (T_o - T_a)$.
6. KPP: Compute $K(z) = k_m(1 - z/h)^2$, clipped to $[0, k_m]$, and $Q_{\text{KPP}} = -K \frac{\partial T_o}{\partial y}$.
7. For RK4 method:
8. Compute $k_1 = f(T_o, T_a)$.
9. Compute $k_2 = f(T_o + \frac{\Delta t}{2} \frac{k_1}{C_h}, T_a + \frac{\Delta t}{2} \frac{k_1}{C_h})$ (Bulk) or $f(T_o, T_a)$ (KPP).
10. Compute $k_3 = f(T_o + \frac{\Delta t}{2} \frac{k_2}{C_h}, T_a + \frac{\Delta t}{2} \frac{k_2}{C_h})$ (Bulk) or $f(T_o, T_a)$ (KPP).
11. Compute $k_4 = f(T_o + \Delta t \frac{k_3}{C_h}, T_a + \Delta t \frac{k_3}{C_h})$ (Bulk) or $f(T_o, T_a)$ (KPP).
12. Update: $Q = Q + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$.
13. Store mean values of heat flux (Bulk) or diffusivity (KPP) for Euler and RK4.
14. Update plots:
15. Left: Heatmap of Q_b or $K(z)$ with annotations every ny/$5 \times$ nx/5 points, using `coolwarm` (Bulk) or `viridis` (KPP).
16. Right: Time series of mean Euler and RK4 values.
17. Increment `current_step`.
18. Log completion or errors, displaying warnings or critical errors via `QMessageBox`.