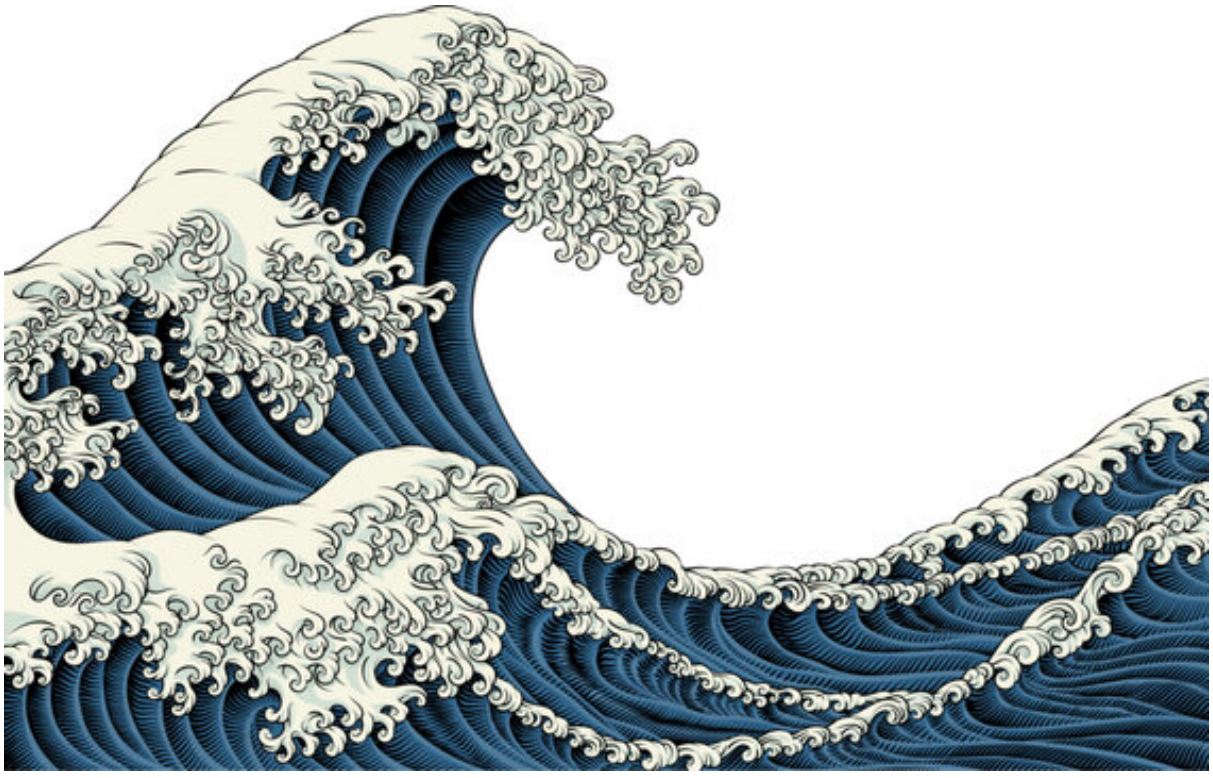
pyCoastal: a Python Tool for Coastal Engineering

Stefano Biondi
University of Florida



Contents

Preface

pyCoastal represents the culmination of my academic journey in numerical modeling, fluid dynamics, and hydraulics. Students often believe that numerical models are overwhelmingly complex tools; however, their structure follows recurrent patterns common to many modeling frameworks. When properly configured and validated, these models can accurately reproduce real-world phenomena, providing valuable insights and enabling the exploration of scenarios beyond direct observation. Today, computational modeling has become a fundamental tool across scientific disciplines, with coastal engineering relying almost entirely on solving partial differential equations through numerical techniques. Some researchers prefer to work in the analytical domain, which has its own advantages and limitations. I personally appreciate and admire those who can capture the complexity of natural processes through concise formulations that, even if simplified, reflect the essence of how the world operates. In this work, I aim to provide a complete manual to support the use of a Python based numerical model, structured around my experience of what is most useful for understanding how such a model works. I do not expect textttpyCoastal to be the tool behind groundbreaking discoveries, although I hope it may contribute to them or at least serve as a starting point for someone. - Stefano

1 Introduction

Coastal zones concentrate people, infrastructure, and ecosystems within environments that are energetic and constantly changing. Waves, tides, surges, and currents interact with complex shorelines to drive erosion, flooding, sediment transport, and water quality variability. Understanding and anticipating these processes is essential for safe navigation, coastal protection, habitat conservation, and climate adaptation. Numerical modeling skills are central to this effort because they allow engineers and scientists to translate governing equations and field observations into predictive tools that can be tested, refined, and applied to real world problems.

The ability to set up, calibrate, and critically interpret numerical models of coastal processes provides a controlled way to explore conditions that cannot be reproduced easily in the laboratory or safely observed in the field. Skilled modelers can design appropriate grids, boundary conditions and parameterizations, evaluate model performance against data, and quantify the uncertainty of predictions. These capabilities support scenario analysis for storms, sea level rise, engineering interventions and nature based solutions, helping decision makers compare alternatives before committing to costly or irreversible actions.

Numerical models also act as a bridge between disciplines, linking hydrodynamics with sediment transport, morphodynamics and biogeochemistry. Proficiency with these tools enables the integration of diverse datasets, from remote sensing to in situ measurements, into coherent simulations of coastal evolution. In this way, numerical modeling skills are not only a technical asset but also a foundation for modern, evidence based management of coasts in a changing climate.

In general, computational coastal engineering connects governing equations with the numerical choices that approximate them. `pyCoastal` exposes the equations, shows exactly how space/time discretizations are assembled, and provides end-to-end examples that one can compare with theory or lab intuition. I deliberately favor clarity over ultimate efficiency so that students and practitioners can see and modify each numerical brick.

The package includes (i) core numerics—uniform grids, finite-difference operators, explicit time integrators, boundary handlers, and Poisson solvers—and (ii) physics modules for shallow water and incompressible-flow building blocks, plus wave, sediment, and basic structural tools common in practice. Every choice (stencil, integrator, boundary enforcement) is visible in the code and editable in YAML, allowing users to learn by changing one thing at a time.

1.1 Before the math

Before introducing the full mathematical formulation, it helps to clarify the main terms used throughout numerical models. A *field* is any variable defined over the domain, such as water depth, velocity, or pressure. A *grid* divides the domain into small cells where these variables are stored; in pyCoastal, the values live at cell centers with extra ghost cells around the boundary. A *stencil* is the small pattern of neighboring cells used to approximate derivatives through finite differences. These approximations form the spatial operators that replace continuous derivatives. Once the spatial terms are discretized, a *time integrator* updates the solution step by step, using the right-hand side that gathers fluxes, advection, diffusion, pressure gradients, and sources. Finally, *boundary conditions* specify what happens at the domain edges and determine how ghost cells are filled. Introducing these ideas up front helps readers understand how each equation is translated into simple algebraic updates that the code executes every time step.

2 Installation and Usage

2.1 Dependencies

To keep the toolchain lightweight and familiar:

- `python` ($\geq$3.9),
- `numpy` (array operations), `scipy` (linear algebra; sparse/iterative solvers),
- `matplotlib` (figures/animations),
- `pyyaml` (text-based configurations for reproducibility).

Optional: `scipy.sparse.linalg` for faster Poisson solves on larger grids.

2.2 Installation

We support both PyPI and editable installs for rapid iteration:

```
pip install pyCoastal
```

or

```
git clone https://github.com/stebiondi/pyCoastal.git
cd pyCoastal
python -m venv .venv && source .venv/bin/activate  # optional but recommended
pip install -e .
```

2.3 Configuration files (YAML)

You won't need to deep dive in the source code to make things work. A single YAML declares grid, time step, physics, boundaries, forcing, and I/O so that experiments can be repeated or swept via small text edits. Here are the variables used in the currently available examples:

- **grid:** `nx, ny, Lx, Ly` (cell counts, domain lengths); `dx, dy` optional (inferred if absent).
- **time:** `dt, t_end, output_every`.
- **physics:** flags (`use_swe`, `use_ns`), viscosity `nu`, turbulence options.
- **boundary:** per side `dirichlet/neumann/wall/sponge` and parameters (`eta_in(t)`, `sigma_max`, `L_sponge`).
- **forcing:** wave spectrum (PM/JONSWAP) or prescribed currents.
- **io:** output paths, figure size, animation toggles.
- **units:** `SI` (default; state explicitly for safety).

3 Numerical Framework

3.1 Grid

Most examples use a uniform Cartesian mesh because it is the simplest setting to illustrate discretization ideas. Two storage layouts are common in numerical modeling: *cell-centered* (unknowns at cell centers) and *node-centered* (unknowns at grid intersections). We adopt the cell-centered layout to keep stencils symmetric and boundary handling straightforward on rectangular domains (see discussions in [2, 3]).

Right outside the boundaries of the domain we can find a cool thing called ghost cells. Ghost cells are extra layers outside the physical domain that hold values implied by boundary

conditions. They allow interior stencils to be applied right up to the boundary, which keeps code simple and avoids ad-hoc one-sided operators. This "ghost cell" approach is standard in finite-volume/difference texts and software [4].

Indexing and memory layout. Interior cells are indexed (i, j) with $i = 1, \ldots, n_x$ and $j = 1, \ldots, n_y$. Ghost layers sit at $i = 0$ and $i = n_x + 1$ (and analogously in y). Arrays follow NumPy C-order; the i-loop (x) is contiguous and typically fastest. Grid spacings $\Delta x, \Delta y$ are constant unless specified.

3.2 Finite Differences

Finite differences approximate derivatives by replacing them with algebraic expressions built from nearby grid values. On a uniform Cartesian grid, a first derivative at a cell center can be estimated using centered differences, which sample the function one point upstream and one downstream to maintain symmetry and second-order accuracy. Higher derivatives and one-sided formulas follow the same idea, using tailored stencils when boundaries or flow direction require it. This approach converts differential equations into systems of algebraic equations that can be solved efficiently, making finite differences a common choice for structured-grid problems in fluid mechanics and wave modeling. They are widely used in CFD/coastal settings for their transparency and ease of change, even if production models often prefer finite volumes on complex grids [2, 1, 3].

Centered operators Centered operators estimate derivatives using information taken symmetrically around the point of interest. Instead of looking only upstream or downstream, they combine values from both sides, which keeps the approximation balanced and avoids directional bias. This symmetry leads to higher accuracy for smooth solutions and makes centered operators a natural choice when the underlying physics is not dominated by advection in a single direction. They appear in many discretizations of wave, diffusion, and elliptic equations because they preserve the structure of the continuous operators while remaining simple to implement on uniform grids. Centered operator reads for derivatives (1) and laplacian (2):

$$\left.\frac{\partial f}{\partial x}\right|_{i,j} \approx \frac{f_{i+1,j} - f_{i-1,j}}{2\Delta x}, \qquad \left.\frac{\partial f}{\partial y}\right|_{i,j} \approx \frac{f_{i,j+1} - f_{i,j-1}}{2\Delta y}, \tag{1}$$

$$\left.\nabla^2 f\right|_{i,j} \approx \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{\Delta x^2} + \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{\Delta y^2}. \tag{2}$$

Upwind for advection Upwind operators estimate derivatives using information taken from the direction of the incoming flow. For an advection term with velocity u, the discretization selects values from upstream, because the solution at a point is determined by what arrives from that direction. This produces a one-sided stencil, which introduces numerical diffusion but ensures stability for hyperbolic problems and prevents nonphysical oscillations. Upwind operators are therefore preferred when modeling transport dominated by a clear flow direction, such as wave propagation, sediment advection, or scalar transport in channels. The

$$\left.\frac{\partial f}{\partial x}\right|_{i} \approx \begin{cases} \dfrac{f_i - f_{i-1}}{\Delta x}, & u > 0, \\ \dfrac{f_{i+1} - f_i}{\Delta x}, & u < 0. \end{cases} \tag{3}$$

3.3 Time Integration

Once space has been discretized, the remaining task is to evolve the solution forward in time. The semi-discrete variable $u(t)$changes according to a right–hand side that gathers all fluxes and source terms. Explicit schemes compute the new state using only information from the current time level, which makes them simple to apply and easy to check for errors. Understanding time discretization is essential because it controls how numerical solutions follow the physical evolution of the system. It determines stability, accuracy, and how fast information can move through the grid, so choosing an appropriate scheme is as important as the spatial discretization.

Schemes

- **Forward Euler** (1st order; simplest for prototypes): The first order Euler method updates the solution by taking the current value and adding a time step multiplied by its time rate of change. It treats that rate as constant over each step, which makes the method very simple to apply. Because it uses only present information, it is easy to implement but accurate only for small time steps.

$$u^{n+1} = u^n + \Delta t\,\mathrm{RHS}(u^n).$$

- **RK3–SSP** (3rd order; robust default for advection): The RK3 SSP method advances the solution through three short stages, each using the result of the previous one. At every stage it blends the new prediction with the earlier state to keep the solution stable when shocks or sharp gradients are present. By the end of the three stages, the method achieves higher accuracy in time than Euler while still keeping the monotonic behavior needed for advection-dominated problems.

$$u^{(1)} = u^n + \Delta t\,\mathrm{RHS}(u^n), \tag{4}$$

$$u^{(2)} = \tfrac{3}{4}u^n + \tfrac{1}{4}\Big[u^{(1)} + \Delta t\,\mathrm{RHS}(u^{(1)})\Big], \tag{5}$$

$$u^{n+1} = \tfrac{1}{3}u^n + \tfrac{2}{3}\Big[u^{(2)} + \Delta t\,\mathrm{RHS}(u^{(2)})\Big]. \tag{6}$$

  I prefer RK3–SSP as a safe all-rounder for wave/transport demos because they retain monotonicity properties under CFL limits [9]..

- **Adams–Bashforth 2** (2nd order; cheap multi-step): ses the time derivative from the current step and the previous step to predict the next state. It forms a weighted combination of these two derivatives to achieve higher accuracy than a single step Euler update. Since it relies on past information, it needs one initial step from another method before it can start. This approach gives a simple multistep scheme that works well when the solution is smooth in time.

$$u^{n+1} = u^n + \tfrac{\Delta t}{2}\Big(3\,\mathrm{RHS}(u^n) - \mathrm{RHS}(u^{n-1})\Big).$$

Stability The CFL condition states that information carried by advection cannot move farther than one cell in a single time step. If it does, the numerical solution becomes unstable. For advective speed c, require:

$$\mathrm{CFL} = \max\left(\frac{c\Delta t}{\Delta x}, \frac{c\Delta t}{\Delta y}\right) < 1.$$

The diffusion limit reflects how fast diffusive processes spread. Because diffusion acts over shorter spatial scales, its stability restriction is often more severe. In practice, the time step must satisfy both conditions so that neither advection nor diffusion destabilizes the computation.

$$\Delta t \leq \frac{1}{2\nu}\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}\right)^{-1}.$$

In practice, the time step must satisfy both conditions so that neither advection nor diffusion destabilizes the computation.

3.4 Boundary Conditions

Boundary conditions specify how the solution behaves at the edges of the computational domain. They provide the information needed to close the system and ensure that the numerical solution remains well posed. Depending on the physics, boundaries may fix the value of a variable, prescribe its gradient, allow waves or flow to enter or leave, or enforce symmetry. In a discretized model, these conditions are encoded through ghost cells or specialized stencils that shape how interior points interact with the boundary. Clear and consistent boundary conditions are essential because they control how signals reflect, how energy enters or exits, and how the interior solution evolves over time. Implementing them via ghost cells lets one reuse interior stencils up to the boundary and keeps discrete balances neat [4].

Common types (with ghost-cell formulas).

- **Dirichlet (value)**: Dirichlet conditions fix the value of a variable at the boundary, such as prescribing a known water level or velocity. $q|_\Gamma = q_D(\mathbf{x},t)$. At $x=0$ (1-D) set $q_0 = 2q_b - q_1$ with $q_b = q_D$.
- **Neumann (gradient)**: Neumann conditions fix the gradient instead, which is used when the flux across the boundary is known. $\partial q/\partial n|_\Gamma = g_N(\mathbf{x},t)$. At $x=0$, $(q_1-q_0)/\Delta x = g_N \Rightarrow q_0 = q_1 - \Delta x\, g_N$.
- **Solid wall**: a wall condition enforces zero normal velocity at a solid boundary and often sets tangential derivatives to represent a no-slip or free-slip surface. $u_n = 0$; mirror the normal component antisymmetrically. For tangential velocity, free-slip is symmetric; no-slip antisymmetric.
- **Sponge layer** (wave absorption): a sponge layer is a damping zone placed near the boundary of a computational domain to absorb outgoing waves or disturbances. Inside this region, the equations include extra terms that gradually relax the solution toward a reference state, such as still water. By smoothly reducing wave energy before it reaches the boundary, the sponge layer prevents artificial reflections that would otherwise contaminate the interior solution. This makes it a practical tool in wave and coastal models where clean open boundaries are important. It relaxes to a reference state,

$$\frac{\partial q}{\partial t} = -\sigma(\mathbf{x})\,[q - q_{\text{ref}}],$$

  with ramp $\sigma(s) = \sigma_{\max}\sin^2(\frac{\pi s}{2})$, $s = (x-x_0)/L_{\text{sp}} \in [0,1]$.

Tuning. Choose $L_{\text{sp}} \approx$ 1–3 wavelengths and $\sigma_{\max} \sim \kappa\, c/L_{\text{sp}}$ ($\kappa \approx$ 3–6). Monitor reflections with in-domain gauges.

4 Governing Equations

4.1 Shallow Water Equations (SWE)

Now we move on to the equations that need to be solved. The shallow water equations describe the motion of a fluid layer whose horizontal scales are much larger than its depth. They track the free surface elevation and the depth-averaged velocity, assuming the vertical structure is simple enough to be represented by a single layer. The equations express conservation of mass and momentum, with pressure given by the hydrostatic approximation. They capture key processes such as wave propagation, flooding and draining, and large-scale currents.

The continuity equation (7) governs how the water height changes due to horizontal transport. The momentum equations (8) and (9) describe how velocities evolve under the effects of gravity, pressure gradients, bed friction, and external forcing. Together, these form a coupled system that models the dynamics of shallow flows in coastal and hydraulic applications. [14, 2].

$$\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} = 0, \tag{7}$$

$$\frac{\partial (hu)}{\partial t} + \frac{\partial}{\partial x}\Big(hu^2 + \tfrac{1}{2}gh^2\Big) + \frac{\partial}{\partial y}(huv) = S_x, \tag{8}$$

$$\frac{\partial (hv)}{\partial t} + \frac{\partial}{\partial x}(huv) + \frac{\partial}{\partial y}\Big(hv^2 + \tfrac{1}{2}gh^2\Big) = S_y. \tag{9}$$

Here h is total depth [m]; u, v are depth-averaged velocities [$\mathrm{m\,s^{-1}}$]; g is gravity; source terms S_x, S_y collect bed slope, bottom friction, and Coriolis. Preserve "lake-at-rest" balance when adding sources.

4.2 Incompressible Navier–Stokes (projection)

For incompressible viscous flow, a popular explicit strategy is the Chorin projection (fractional-step) method: predict velocity with advection+diffusion, solve a Poisson equation for pressure to enforce zero divergence, then correct [6, 5].

$$\tilde{\mathbf{u}} = \mathbf{u}^n + \Delta t(\mathcal{N}(\mathbf{u}^n) + \nu \nabla^2 \mathbf{u}^n), \tag{10}$$

$$\nabla^2 p^{n+1} = \Delta t^{-1} \nabla \cdot \tilde{\mathbf{u}}, \tag{11}$$

$$\mathbf{u}^{n+1} = \tilde{\mathbf{u}} - \Delta t\, \nabla p^{n+1}. \tag{12}$$

$\mathbf{u} = (u, v)$ [$\mathrm{m\,s^{-1}}$], p kinematic pressure [$\mathrm{m^2\,s^{-2}}$]; ν viscosity.

4.3 Poisson Solver

A Poisson solver computes a field whose Laplacian matches a given source term. This type of equation appears in pressure projection, potential flow, and many elliptic problems. After discretizing the Laplacian on the grid, the solver must determine the values that satisfy the resulting linear system while honoring the boundary conditions. This step is central because it links information across the entire domain and ensures the modeled fields remain physically consistent. Poisson problems appear in pressure projection, potential flow, and diffusion steady states. On a uniform grid, the five-point Laplacian leads to a sparse linear system. Iterative methods (Jacobi, Gauss–Seidel/SOR, Conjugate Gradient) are standard workhorses; FFT-based solvers are efficient on rectangles with simple BCs [7, 8].

Continuous problem (typical BC mix).

$$\nabla^2\phi = b \text{ in } \Omega, \qquad \phi|_{\Gamma_D} = \phi_D, \quad \frac{\partial\phi}{\partial n}\Big|_{\Gamma_N} = g_N,$$

with compatibility $\int_\Omega b\, d\Omega + \int_{\Gamma_N} g_N\, d\Gamma = 0$ for pure Neumann.

Discrete operator (uniform grid).

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = b_{i,j}.$$

Jacobi iteration (illustrative, $h = \Delta x = \Delta y$).

$$\phi_{i,j}^{(k+1)} = \tfrac{1}{4}\Big(\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k)} + \phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k)} - h^2\, b_{i,j}\Big)\,,$$

and stop when $\|\mathbf{b} - A\boldsymbol{\phi}^{(k)}\|_2 / \|\mathbf{b}\|_2 < \varepsilon$.

5 Turbulence Models

Here is cool stuff. Turbulence models provide a way to represent the effects of unresolved turbulent motions without resolving every eddy in the flow. They modify the governing equations by adding terms that mimic the mixing, momentum transfer, and energy dissipation produced by small-scale turbulence. Different models make different assumptions about how these unresolved motions behave. Simple eddy-viscosity models relate turbulent stresses to mean velocity gradients, while more advanced two-equation models predict how turbulent kinetic energy and its dissipation evolve. Large eddy approaches resolve the largest motions and model only the smallest ones. In all cases, the goal is to recover the influence of turbulence on the mean flow at an affordable computational cost. When grid/step sizes cannot resolve all scales, eddy-viscosity closures emulate subgrid transport. It's included standard variants; production RANS/LES demand careful calibration and near-wall treatment [10, 11].

5.1 Smagorinsky Eddy Viscosity (LES)

The Smagorinsky model is a simple large eddy simulation approach that represents the effect of small turbulent motions by adding an eddy viscosity proportional to the local strain rate. It assumes that unresolved eddies act like an enhanced viscosity that increases where velocity gradients are strong. The eddy viscosity is computed from the grid size and a constant coefficient, so the model automatically adjusts to the flow structure without solving extra transport equations. This makes it easy to implement and suitable for capturing the dominant turbulent mixing at scales just below the grid resolution. For resolved unsteady flows, Smagorinsky relates eddy viscosity to strain and grid scale:

$$\nu_t = (C_s\Delta)^2\,|S|, \qquad |S| = \sqrt{2S_{ij}S_{ij}}, \qquad S_{ij} = \tfrac{1}{2}(\partial u_i/\partial x_j + \partial u_j/\partial x_i)\,,$$

with $C_s \approx 0.1$–0.2, $\Delta = \sqrt{\Delta x\,\Delta y}$.

5.2 k–ε Model (RANS)

The k epsilon model introduces two extra transport equations to describe how turbulence evolves. One equation predicts the turbulent kinetic energy k, which measures the intensity of velocity fluctuations. The other predicts its dissipation rate ε, which controls how rapidly those fluctuations lose energy. The model combines these two quantities to compute an eddy viscosity that influences the momentum equations. By tracking both production and dissipation of turbulent energy, the k epsilon formulation captures a broader range of turbulent behavior than simple algebraic models while remaining computationally efficient.Two equations transport TKE k and dissipation ε; the eddy viscosity is $\nu_t = C_\mu k^2/\varepsilon$:

$$\partial_t k + U_j \partial_{x_j} k = P - \varepsilon + \partial_{x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_k}\right)\partial_{x_j} k\right], \tag{13}$$

$$\partial_t \varepsilon + U_j \partial_{x_j} \varepsilon = C_{\varepsilon 1}\frac{\varepsilon}{k}P - C_{\varepsilon 2}\frac{\varepsilon^2}{k} + \partial_{x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_\varepsilon}\right)\partial_{x_j}\varepsilon\right], \tag{14}$$

with $P = 2\nu_t S_{ij} S_{ij}$ [16].

5.3 k–ω Model (RANS)

The k ω model also uses two transport equations, but it pairs the turbulent kinetic energy kkk with the specific dissipation rate . The variableω represents how quickly turbulent kinetic energy is dissipated per unit energy and naturally adapts to changes near walls and in regions with strong shear. From k and ω, the model computes an eddy viscosity that feeds back into the momentum equations. Because ω is more sensitive to near-wall effects than ε the $k\,\omega$ formulation often performs better in boundary layers and separated flows, giving it an advantage in many engineering applications. Here $\nu_t = k/\omega$ and the specific dissipation $\omega = \varepsilon/k$ is transported; ω behaves well near walls, easing low-y^+ use [11]:

$$\partial_t k + U_j \partial_{x_j} k = P - \beta^* k\,\omega + \partial_{x_j}\Big[(\nu + \sigma_k^*\,\nu_t)\,\partial_{x_j} k\Big], \tag{15}$$

$$\partial_t \omega + U_j \partial_{x_j} \omega = \alpha\,\frac{\omega}{k}\,P - \beta\,\omega^2 + \partial_{x_j}\Big[(\nu + \sigma_\omega\,\nu_t)\,\partial_{x_j}\omega\Big]. \tag{16}$$

6 Wave Tools

6.1 Linear dispersion and derived numbers

In here, we talk waves. Linear dispersion describes how waves of different wavelengths travel at different speeds in a fluid. In the linear theory, each wavelength has its own phase speed determined by gravity and water depth. Long waves feel the full depth and move faster, while short waves are more influenced by the surface and move slower. This wavelength dependence spreads an initial disturbance into distinct components as it travels. Linear dispersion is central for understanding how wave groups form, how wave energy moves, and how numerical models must represent wave propagation accurately. Under linear (small-amplitude) wave theory, frequency, wavenumber, and depth are linked via the dispersion relation. This sets phase/group speeds and underpins many coastal scalings [12, 13, 14]:

$$\omega^2 = gk\tanh(kh), \qquad c = \omega/k, \qquad c_g = \tfrac{1}{2}c\left(1 + \frac{2kh}{\sinh(2kh)}\right).$$

A Newton iteration solves $F(k) = \omega^2 - gk\tanh(kh) = 0$ efficiently. Useful nondimensional numbers include the Iribarren (surf similarity) $\xi_0 = \tan\beta/\sqrt{H_0/L_0}$ and Ursell $U = HL^2/h^3$, with deepwater wavelength $L_0 = gT^2/(2\pi)$.

6.2 Spectral synthesis (PM/JONSWAP)

A wave spectrum describes how the energy of a sea state is distributed across different wave frequencies. Instead of representing the ocean surface as a single wave, the spectrum treats it as the sum of many components, each with its own frequency, amplitude, and direction. The spectrum shows which frequencies carry most of the energy and how broad or narrow the sea state is. Common spectral shapes, such as Pierson–Moskowitz or JONSWAP, capture typical conditions in wind-driven seas. Wave spectra are essential because they provide a statistical way to represent irregular waves and serve as inputs for numerical models, coastal engineering design, and offshore structure analysis. To drive a numerical wave flume with irregular seas, synthesize a surface elevation time series by random-phase superposition consistent with a target spectrum (PM fully developed; JONSWAP fetch-limited) [17, 18, 13, 14]:

$$\eta(t) = \sum_{i=1}^{N} \sqrt{2S(f_i)\Delta f}\,\cos(2\pi f_i t + \phi_i), \qquad \phi_i \sim U[0, 2\pi].$$

Moments $m_n = \int_0^\infty f^n S(f)\,df$ give bulk estimates $H_s \approx 4\sqrt{m_0}$ and $T_z \approx \sqrt{m_0/m_2}$.

7 Sediment and Structural Tools

7.1 Sediment transport (first checks)

A sediment transport formulation provides a mathematical description of how sediment moves under the action of flowing water or waves. It links hydrodynamics to the motion of particles by relating transport rates to shear stress, flow velocity, and sediment properties such as grain size and density. Most formulations distinguish between bed load, where particles roll or hop along the bed, and suspended load, where turbulence keeps particles in the water column. The transport rate is often expressed as an empirical or semi-empirical function of excess shear stress, capturing the idea that motion begins only when the flow is strong enough to overcome the particle's weight and friction. These formulations allow models to predict erosion, deposition, and bed evolution in rivers, estuaries, and coastal settings. In pyCoastal, the Shields parameter gages bed mobility under shear:

$$\theta = \frac{\tau_b}{(\rho_s - \rho)gd}, \qquad \tau_b = \rho u_*^2,$$

with critical $\theta_c \approx 0.03$–0.06 for sand (site-specific). Empirical formulations (e.g., Van Rijn) convert mobility into transport rates [19].

7.2 Wave Runup

Wave runup is the maximum vertical rise of water on a beach or coastal structure as waves rush up after breaking. It combines two processes: the uprush of individual waves and the longer oscillations of the shoreline driven by groups of waves. Runup depends on wave height, period, beach slope, and roughness, and it plays a key role in coastal flooding because it determines

how far waves can reach inland. Understanding runup is important for designing dunes, revetments, and seawalls so that they withstand extreme conditions without overtopping. First-pass estimators for runup on beaches/armor use slope and incident wave bulk parameters; a common parametric form is

$$R_{2\%} = a\,\beta_f\,H_s\,\xi_0 + b,$$

with coefficients tuned by data [20]. Use as screening tools before detailed modeling.

8 Examples (from YAML to figures)

Each example is fully configured by a YAML in `examples/configs/`. The goal is to connect one governing equation, one set of numerics, and one diagnostic to a single learning objective.

8.1 Water Drop (2D wave equation)

Objective. Verify Laplacian accuracy and RK stability on a simple hyperbolic system.

$$\frac{\partial^2\eta}{\partial t^2} = c^2\nabla^2\eta.$$

Initialize a Gaussian bump; check circular wavefront speed $\approx c$ and convergence under grid/time refinement.

8.2 Irregular Wave Forcing (numerical flume)

Objective. Generate target spectra and manage reflections. Use the synthesized $\eta(t)$ as Dirichlet forcing at inflow and a sponge at outflow; place gauges to compare input and interior spectra (H_s, T_p). For wave generation/absorption strategies also see [21].

8.3 Current and Pollutant (advection–diffusion)

Objective. Observe upwind smearing and diffusive spreading rate.

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} + v\frac{\partial C}{\partial y} = \nu\nabla^2 C.$$

For pure diffusion, variance grows as $2\nu t$. Compare measured growth with theory.

8.4 Viscous Fluid (velocity decay)

Objective. Check diffusive time scale and operator accuracy with a mode that has a known decay.

$$\frac{\partial u}{\partial t} = \nu\nabla^2 u, \qquad u(x,y,0) = U_0\sin(k_x x)\sin(k_y y),$$

which decays as $u(x,y,t) = U_0\sin(k_x x)\sin(k_y y)\exp[-\nu(k_x^2+k_y^2)t]$.

9 Diagnostics, Verification, and Validation

Numerical experiments should be accompanied by simple, repeatable checks:

- **CFL monitor:** compute/report CFL each step; adapt Δt if needed.

- **Divergence (NS):** after projection, track $\|\nabla\cdot\mathbf{u}\|_2$; it should sit near Poisson-solver tolerance.
- **Energy/variance:** monitor $\int \eta^2$ or kinetic energy to spot drift or reflections.
- **Convergence:** refine $\Delta x, \Delta y, \Delta t$ and verify expected order.
- **Spectral targets:** compute H_s, T_p against input; estimate reflection from incident/reflected separation.

10 I/O and Visualization

Save arrays as NumPy files and figures with `matplotlib`. For animations, export `.mp4` or `.gif`. Store run metadata (git hash, YAML, seeds) alongside outputs so plots can be regenerated byte-for-byte.

11 Running Examples

```
python examples/wave2D_irregular.py
python examples/numerics/water_drop.py
python examples/current.py
python examples/pollutant.py
```

Each script reads its YAML from `examples/configs/`; adjust parameters there and rerun to explore sensitivities.

12 Extending `pyCoastal`

To add physics or numerics:

1. Implement a right-hand side RHS(u, t) and register it with the integrator.
2. Provide boundary handlers (ghost-cell formulas) consistent with your stencils.
3. Add YAML keys and sensible defaults.
4. Include a small example and docstring describing equations, variables, numerics, and stability limits.

13 Limitations

- Uniform Cartesian grids only (no curvilinear/unstructured meshes).
- Explicit time stepping can be restrictive for stiff sources (e.g., ε or ω transport).
- Turbulence closures are educational scaffolds, not production RANS/LES.
- No wetting/drying or moving shorelines in the present setup.

List of Symbols

x, y	Coordinates [m]; t time [s].
$\Delta x, \Delta y$	Grid spacings [m]; Δt time step [s].
u, v	Velocity components [m s^{-1}]; $\mathbf{u}$ velocity vector.
h	Total water depth [m]; η free-surface elevation [m].
g	Gravitational acceleration [m s^{-2}].

S_x, S_y	SWE momentum sources (bed slope, friction, Coriolis).
ν, ν_t	Molecular and eddy viscosities [m^2 s^{-1}].
S_{ij}	Mean strain-rate tensor [s^{-1}]; $P = 2\nu_t S_{ij} S_{ij}$.
c	Characteristic speed for CFL [m s^{-1}]; CFL Courant number.
k_T, ε, ω	TKE [m^2 s^{-2}], dissipation [m^2 s^{-3}], specific dissipation [s^{-1}].
$C_\mu, C_{\varepsilon 1}, C_{\varepsilon 2}, \sigma_k, \sigma_\varepsilon$	k–ε constants.
$\alpha, \beta, \beta^*, \sigma_k^*, \sigma_\omega$	k–ω constants.
C_s	Smagorinsky constant [–]; Δ filter width [m].
k_w, ω_w	Wavenumber [m^{-1}] and angular frequency [s^{-1}] in wave theory.
c, c_g	Phase and group speeds [m s^{-1}]; T period [s]; f frequency [Hz].
L, L_0	Wavelength; deepwater wavelength [m]; H, H_s, H_0 heights [m].
β, β_f	Beach/foreshore slopes [–]; ξ_0, U Iribarren/Ursell [–].
$S(f)$	Spectrum [m^2/Hz]; m_n spectral moment [m^2 Hz^n].
C	Passive scalar concentration; θ Shields parameter [–]; τ_b bed shear [Pa]; u_* shear velocity [m s^{-1}].
$\sigma(\mathbf{x})$	Sponge absorption [s^{-1}]; L_{sp} thickness [m].
$\tilde{\mathbf{u}}$	Predictor velocity; p kinematic pressure [m^2 s^{-2}].

14 References

References

[1] E. F. Toro (1999). *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer.

[2] C. Hirsch (2007). *Numerical Computation of Internal and External Flows*. Butterworth–Heinemann.

[3] H. K. Versteeg and W. Malalasekera (2007). *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson.

[4] R. J. LeVeque (2002). *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press.

[5] J. H. Ferziger and M. Perić (2002). *Computational Methods for Fluid Dynamics*. Springer.

[6] A. J. Chorin (1968). Numerical solution of the Navier–Stokes equations. *Math. Comp.* 22(104):745–762.

[7] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst (1994). *Templates for the Solution of Linear Systems*. SIAM.

[8] Y. Saad (2003). *Iterative Methods for Sparse Linear Systems*. SIAM.

[9] S. Gottlieb, C.-W. Shu, and E. Tadmor (2001). Strong stability-preserving high-order time discretization methods. *SIAM Review* 43(1):89–112.

[10] S. B. Pope (2000). *Turbulent Flows*. Cambridge University Press.

[11] D. C. Wilcox (1998). *Turbulence Modeling for CFD*. DCW Industries.

[12] R. G. Dean and R. A. Dalrymple (1991). *Water Wave Mechanics for Engineers and Scientists*. Dover.

[13] L. H. Holthuijsen (2007). *Waves in Oceanic and Coastal Waters*. Cambridge University Press.

[14] USACE (2003, 2006 update). *Coastal Engineering Manual (EM 1110-2-1100)*. U.S. Army Corps of Engineers.

[15] J. Smagorinsky (1963). General circulation experiments with the primitive equations. *Monthly Weather Review*, 91, 99–164.

[16] B. E. Launder and D. B. Spalding (1974). The numerical computation of turbulent flows. *Comp. Methods Appl. Mech. Eng.*, 3(2), 269–289.

[17] W. J. Pierson and L. Moskowitz (1964). A proposed spectral form for fully developed wind seas. *J. Geophys. Res.*, 69(24), 5181–5190.

[18] K. Hasselmann et al. (1973). Measurements of wind-wave growth and swell decay during JONSWAP. *Deutsche Hydrographische Zeitschrift*, 12, 1–95.

[19] L. C. van Rijn (1984). Sediment transport, part I: bed load transport. *J. Hydraulic Engineering*, 110(10), 1431–1456.

[20] H. F. Stockdon, R. A. Holman, P. A. Howd, and A. H. Sallenger (2006). Empirical parameterization of setup, swash, and runup. *Coastal Engineering*, 53(7), 573–588.

[21] N. G. Jacobsen, D. R. Fuhrman, and J. Fredsøe (2012). A wave generation toolbox for the open-source CFD library OpenFOAM. *Int. J. Numer. Methods Fluids*, 70(9), 1073–1088.