**C++11 code 11.7.16: Code for `solveClaw()` ↝ GitLab**

```
template <typename U0_FUNCTOR>
Eigen::VectorXd solveClaw(U0_FUNCTOR &&u0, double T, unsigned int n) {
  // Set up spacial mesh and inital data.
  double a = 0.0;
  double b = 1.0;
  double h = (b - a) / n;
  Eigen::VectorXd x = Eigen::VectorXd::LinSpaced(n, a + 0.5 * h, b - 0.5 * h);

  // Approximate dual cell averages at t=0
  Eigen::VectorXd mu = x.unaryExpr(u0);

  double alpha = mu.minCoeff();  // lower bound for initial data
  double beta = mu.maxCoeff();   // upper bound for initial data
  assert(alpha > 0.0 && beta > 0.0);

  // Set timestep tau according to the CFL-condition
  double tau =
      h / std::max(std::abs(std::log(alpha)), std::abs(std::log(beta)));
  // Semi-discretization (discretization in space)
  double h_inv = 1.0 / h;
  // Define right-hand-side for the SSP ODE solver
  auto semi_discrete_rhs =
      [h_inv](const Eigen::VectorXd &mu) -> Eigen::VectorXd {
    return -h_inv * slopelimfluxdiffper(mu, &logGodunovFlux, &limiterMC);
  };
  // Timestepping:  Solve the semi-discrete ODE
  int N = (int)(T / tau + 0.5);
  for (int i = 0; i < N; ++i) mu = sspEvolop(semi_discrete_rhs, mu, tau);

  return mu;
}
```