C++11 code 11.7.19: Code for `studyCvgMUSCLSolution()` ↬ GitLab

```
template <typename U0_FUNCTOR>
void studyCvgMUSCLSolution(U0_FUNCTOR &&u0, double T) {
  // For temporarily storing the number of cells and the associated
     error norms
  std::vector<std::tuple<std::size_t, double, double>> result{};
  constexpr int n_ref = 8192;
  // Compute reference solution
  std::cout << "Computing reference solution at T = " << T << " with " << n_ref
            << " cells" << std::endl;
  Eigen::VectorXd u_ref{solveClaw(std::forward<U0_FUNCTOR>(u0), T, n_ref)};
  // Compute solution on different meshes
  constexpr int lmin = 4;   // Coarsest mesh with 16 cells
  constexpr int lmax = 10;  // Finest mesh with 1024 cells
  for (int l = lmin; l <= lmax; ++l) {
    const std::size_t n = 1U << l;
    std::cout << "Computing with n = " << n << std::endl;
    // Compute solution
    Eigen::VectorXd u{solveClaw(std::forward<U0_FUNCTOR>(u0), T, n)};
    // Transfer solution to finest mesh
    Eigen::VectorXd u_prop = Eigen::VectorXd::Zero(u_ref.size());
    interpolate(u, u_prop);
    double linf_err = (u_prop - u_ref).lpNorm<Eigen::Infinity>();
    double l1_err = (u_prop - u_ref).lpNorm<1>() / n_ref;
    result.push_back({n, linf_err, l1_err});
  }
  std::cout << "n \t linf error \t l1 error" << std::endl;
  for (auto &data : result) {
    std::cout << std::get<0>(data) << " \t " << std::get<1>(data) << " \t "
              << std::get<2>(data) << std::endl;
  }
}
```