C++ code 9.8.33: Solver for nonlinear Schrödinger equation ⇝ GitLab

```cpp
  // Load mesh and initalize FE space and DOF handler
  auto mesh_factory = std::make_unique<lf::mesh::hybrid2d::MeshFactory>(2);
  const lf::io::GmshReader reader(std::move(mesh_factory),
                                  "meshes/square_64.msh");
  auto mesh_p = reader.mesh();
  auto fe_space =
      std::make_shared<lf::uscalfe::FeSpaceLagrangeO1<double>>(mesh_p);
  const lf::assemble::DofHandler &dofh{fe_space->LocGlobMap()};
  const lf::uscalfe::size_type N_dofs(dofh.NumDofs());

  // Mass matrix
  lf::assemble::COOMatrix<double> D_COO(N_dofs, N_dofs);
  NonLinSchroedingerEquation::MassElementMatrixProvider mass_emp;
  lf::assemble::AssembleMatrixLocally(0, dofh, dofh, mass_emp, D_COO);
  Eigen::SparseMatrix<double> D = D_COO.makeSparse();
  Eigen::SparseMatrix<std::complex<double>> M = std::complex<double>(0, 1) * D;

  // Stiffness matrix
  lf::assemble::COOMatrix<double> A_COO(N_dofs, N_dofs);
  lf::uscalfe::LinearFELaplaceElementMatrix stiffness_emp;
  lf::assemble::AssembleMatrixLocally(0, dofh, dofh, stiffness_emp, A_COO);
  Eigen::SparseMatrix<double> A = A_COO.makeSparse();

  // Prepare timestepping
  int timesteps = 100;
  double T = 1.0;
  double tau = T / timesteps;

// Prepare inital data
  const double PI = 3.14159265358979323846;
  auto u0 = [PI](Eigen::Vector2d x) -> double {
    return 4.0 * std::cos(PI * x(0)) * std::cos(PI * x(1));
  };
  lf::mesh::utils::MeshFunctionGlobal mf_u0{u0};
  Eigen::VectorXcd mu = lf::fe::NodalProjection(*fe_space, mf_u0);

  // Prepare split-step propagator for full step τ
  NonLinSchroedingerEquation::SplitStepPropagator splitStepPropagator(A, M,
                                                                      tau);

  // Arrays for storing "energies" contributing to the Hamiltonian
  Eigen::VectorXd norm(timesteps + 1);
  Eigen::VectorXd E_kin(timesteps + 1);
  Eigen::VectorXd E_int(timesteps + 1);
  // Timestepping
  for (int j = 0; j < timesteps; ++j) {
    // Compute norm and energy along the solution
    norm(j) = NonLinSchroedingerEquation::Norm(mu, D);
    E_kin(j) = NonLinSchroedingerEquation::KineticEnergy(mu, A);
    E_int(j) = NonLinSchroedingerEquation::InteractionEnergy(mu, D);
    // Timestep tau according to Strang splitting
    mu = splitStepPropagator(mu);
  }
  norm(timesteps) = NonLinSchroedingerEquation::Norm(mu, D);
  E_kin(timesteps) = NonLinSchroedingerEquation::KineticEnergy(mu, A);
  E_int(timesteps) = NonLinSchroedingerEquation::InteractionEnergy(mu, D);
```