C++11 code 3.3.15: Sub-problem (3-3.m): Implementation of `normsSolutionPointLoadDirichletBVP()` ➙ GitLab

```cpp
  // Assemble matrix A
  lf::assemble::COOMatrix<double> A(N_dofs, N_dofs);
  lf::uscalfe::LinearFELaplaceElementMatrix loc_mat_laplace{};
  lf::assemble::AssembleMatrixLocally(0, dofh, dofh, loc_mat_laplace, A);
  // Build rhs vector using dedicated ENTITY_VECTOR_PROVIDER
  Eigen::VectorXd rhs(N_dofs);
  rhs.setZero();
  PointEvaluationRhs::DeltaLocalVectorAssembler myvec_pro(source_point);
  lf::assemble::AssembleVectorLocally(0, dofh, myvec_pro, rhs);

  // Enforce Dirichlet boundary conditions
  const double boundary_val = 0;  // zero Dirichlet boundary conditions
  auto bd_flags{lf::mesh::utils::flagEntitiesOnBoundary(dofh.Mesh(), 2)};
  auto my_selector = [&dofh, &bd_flags, &boundary_val](unsigned int dof_idx) {
    if (bd_flags(dofh.Entity(dof_idx))) {
      return (std::pair<bool, double>(true, boundary_val));
    }  // interior node: the value we return here does not matter
    return (std::pair<bool, double>(false, 42.0));
  };
  lf::assemble::FixFlaggedSolutionComponents<double>(my_selector, A, rhs);

  // Solve linear system of equations A*x = rhs
  const Eigen::SparseMatrix<double> A_crs(A.makeSparse());
  Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
  solver.compute(A_crs);
  if (solver.info() == Eigen::Success) {
    sol_vec = solver.solve(rhs);
  } else {
    LF_ASSERT_MSG(false, "Eigen Factorization failed");
  }
```