C++11 code 2.13.13: Sub-problem (2-13.g): Unit test for FESourceElemVecProvider ➙ GitLab

```
TEST(ParametricElementMatrices, TestLoad) {
  // use test mesh (with only affine equivalent cells!) to set up fe space
  auto mesh = lf::mesh::test_utils::GenerateHybrid2DTestMesh(5, 1);
  auto fe_space =
      std::make_shared<lf::uscalfe::FeSpaceLagrangeO1<double>>(mesh);
  const lf::assemble::DofHandler &dofh{fe_space->LocGlobMap()};
  const lf::base::size_type N_dofs(dofh.NumDofs());
  // An affine linear function that can be represented exactly
  // in the space of p.w. linear Lagrangian finite element functions
  auto w_func = [](Eigen::Vector2d x) -> double { return (3.0 * x[0] - x[1]); };
  lf::mesh::utils::MeshFunctionGlobal mf_w_func{w_func};
  // function f(x) = 1/(1 + w(x)^2)
  auto f = [&w_func](Eigen::Vector2d x) -> double {
    return 1. / (1. + std::pow(w_func(x), 2));
  };
  lf::mesh::utils::MeshFunctionGlobal mf_f{f};

  // interpolation of w on fe space: reproduces function
  auto w = lf::fe::NodalProjection<double>(*fe_space, mf_w_func);

  // use own method to assemble load vector
  auto elvec_builder =
      ParametricElementMatrices::FESourceElemVecProvider(fe_space, w);
  Eigen::Matrix<double, Eigen::Dynamic, 1> phi(N_dofs);
  phi.setZero();
  AssembleVectorLocally(0, dofh, elvec_builder, phi);
  // set up midpoint quadrature rules for all cell types
  std::map<lf::base::RefEl, lf::quad::QuadRule> quad_rules{
      {lf::base::RefEl::kTria(), lf::quad::make_TriaQR_EdgeMidpointRule()},
      {lf::base::RefEl::kQuad(), lf::quad::make_QuadQR_EdgeMidpointRule()}};

  // compute library solution
  Eigen::Matrix<double, Eigen::Dynamic, 1> phi_lib(N_dofs);
  phi_lib.setZero();
  lf::uscalfe::ScalarLoadElementVectorProvider<double, decltype(mf_f)>
      elvec_builder_(fe_space, mf_f, quad_rules);
  AssembleVectorLocally(0, dofh, elvec_builder_, phi_lib);

  // compare the results
  for (int i = 0; i < N_dofs; i++) {
    ASSERT_NEAR(phi(i), phi_lib(i), 1E-9);
  }
}
```