C++ code 3.13.14: lf::fe::ScalarReferenceFiniteElement-based implementation of `stabFlux()` ⇝ GitLab

```
template <typename SIGMAFUNCTION, typename PSIGRAD>
double stabFluxTRF(
    std::shared_ptr<const lf::uscalfe::FeSpaceLagrangeO1<double>> fe_space,
    const Eigen::VectorXd &sol_vec, SIGMAFUNCTION &&sigma, PSIGRAD &&gradpsi) {
  // Underlying FE mesh
  const lf::mesh::Mesh &mesh{*(fe_space->Mesh())};
  // Local-to-Global map for local/global shape function indices
  const lf::assemble::DofHandler &dofh{fe_space->LocGlobMap()};
  // Reference coordinates of "midpoint" of a triangle (center of gravity)
  const Eigen::MatrixXd zeta_ref{
      (Eigen::Matrix<double, 2, 1>() << 1.0 / 3.0, 1.0 / 3.0).finished()};
  // Obtain gradients of reference shape functions at center of gravity
  const lf::fe::ScalarReferenceFiniteElement<double> &ref_lsf{
      *fe_space->ShapeFunctionLayout(lf::base::RefEl::kTria())};
  LF_ASSERT_MSG(
      ref_lsf.NumRefShapeFunctions() == 3,
      "There must be 3 reference shape functions associated with vertices");
  const Eigen::MatrixXd ref_lsf_grads{
      ref_lsf.GradientsReferenceShapeFunctions(zeta_ref)};
  LF_ASSERT_MSG(ref_lsf_grads.rows() == 3, "Three gradients required!");
  LF_ASSERT_MSG(ref_lsf_grads.cols() == 2, "Two components expected!");
  // Summation variable
  double s = 0.0;
  // Loop over all cells
  for (const lf::mesh::Entity *cell : mesh.Entities(0)) {
    LF_ASSERT_MSG(cell->RefEl() == lf::base::RefEl::kTria(),
                  "Not implemented for " << *cell);
    // Obtain geometry information for entity
    const lf::geometry::Geometry &geo{*cell->Geometry()};
    // Fetch the transformation matrix for gradients
    const Eigen::MatrixXd JinvT{geo.JacobianInverseGramian(zeta_ref)};
    LF_ASSERT_MSG(
        (JinvT.rows() == 2) && (JinvT.cols() == 2),
        "JinvT is " << JinvT.rows() << " x " << JinvT.cols() << "-matrix!");
    // Compute the gradients of the local shape functions by transformation
    const auto grad_lsf{JinvT * ref_lsf_grads.transpose()};
    // Compute the area of the triangle
    const double area = lf::geometry::Volume(geo);
    // DofHandler must provide three degrees of freedom per cell for piecewise
    // linear Lagrangian finite elements on triangles
    LF_ASSERT_MSG(dofh.NumLocalDofs(*cell) == 3,
                  "contactFlux: 3 dofs per triangle mandatory!");
    // Fetch indices of global shape functions associated with triangle
    const auto glob_dof_idx{dofh.GlobalDofIndices(*cell)};
    // Compute (constant) local gradient of the finite element solution
    const Eigen::Vector2d local_gradient{
        grad_lsf * (Eigen::Vector3d() << sol_vec[glob_dof_idx[0]],
                    sol_vec[glob_dof_idx[1]], sol_vec[glob_dof_idx[2]])
                       .finished()};
    // Find physical/world coordinates of "midpoint" of triangle
    const Eigen::MatrixXd zeta{geo.Global(zeta_ref)};
    // Add contribution of triangle
    const auto quadnode{zeta.col(0)};
    s += area * (gradpsi(quadnode)).dot(sigma(quadnode) * local_gradient);
  }
```