C++ code 10.6.18: Implementation of `Eval()` member function ⇢ GitLab

```
template <class MESHFUNCTION_V>
typename SUAdvectionElemMatrixProvider<MESHFUNCTION_V>::ElemMat
SUAdvectionElemMatrixProvider<MESHFUNCTION_V>::Eval(
    const lf::mesh::Entity &cell) {
  LF_ASSERT_MSG(cell.RefEl() == lf::base::RefEl::kTria(),
                "Only implemented for triangles");
  // Obtain geometry information
  const lf::geometry::Geometry *geo_ptr = cell.Geometry();
  LF_ASSERT_MSG(geo_ptr->DimGlobal() == 2,
                "Only implemented for planar triangles");
  // Gram determinant at quadrature points
  const Eigen::VectorXd dets(geo_ptr->IntegrationElement(qr_.Points()));
  LF_ASSERT_MSG(dets.size() == qr_.NumPoints(),
                "Mismatch " << dets.size() << " <-> " << qr_.NumPoints());
  // Fetch the transformation matrices for the gradients
  const Eigen::MatrixXd JinvT(geo_ptr->JacobianInverseGramian(qr_.Points()));
  LF_ASSERT_MSG(JinvT.cols() == 2 * qr_.NumPoints(),
                "Mismatch " << JinvT.cols() << " <-> " << 2 * qr_.NumPoints());
  // Obtain values of velocity at the 6 quadrature points
  std::vector<Eigen::Vector2d> v_vals = v_(cell, qr_.Points());
  // 6 x 6 element matrix (There are six local shape functions.)
  ElemMat mat(6, 6);
  mat.setZero();
  // Compute the factor δ_K
  const Eigen::MatrixXd vertices{lf::geometry::Corners(*geo_ptr)};
  // Size of triangle
  const double hK = std::max({(vertices.col(1) - vertices.col(0)).norm(),
                              (vertices.col(2) - vertices.col(1)).norm(),
                              (vertices.col(0) - vertices.col(2)).norm()});
  // Maximal modulus of velocity in quadrature nodes
  const double vmax =
      std::max_element(v_vals.begin(), v_vals.end(),
                       [](Eigen::Vector2d &a, Eigen::Vector2d &b) -> bool {
                         return (a.norm() < b.norm());
                       })
          ->norm();
  const double delta_K = use_delta_ ? (hK / (2.0 * vmax)) : 1.0;
  // Outer summation loop over quadrature points
  for (unsigned int l = 0; l < 6; ++l) {
    // Metric factor |det DΦ_K(ζ̂_ℓ)| scaled with
    // quadrature weight ω_ℓ
    const double fac = qr_.Weights()[l] * dets[l];
    // Compute transformed gradients t_ℓ^i of all local shape
    // functions and collect them in the columns of a 2 x 6-matrix
    const Eigen::Matrix<double, 2, 6> TrfG =
        JinvT.block(0, 2 * l, 2, 2) *
        (grad_ref_lsf_.block(0, 2 * l, 6, 2).transpose());
    // Compute the inner products of the transformed gradients with the
    // velocity vector at the current quadrature point.
    const Eigen::Matrix<double, 1, 6> mvec = v_vals[l].transpose() * TrfG;
    // Loop over columns of element matrix
    for (unsigned int j = 0; j < 6; ++j) {
      // Loop over rows of element matrix
      for (unsigned int i = 0; i < 6; ++i) {
        mat(i, j) +=
            fac * (delta_K * mvec[i] * mvec[j] + mvec[j] * val_ref_lsf_(i, l));
```