C++ code 3.9.11: Implementation of `Eval()` method of VectorProjectionMatrixProvider
→ GITHUB

```cpp
Eigen::MatrixXd VectorProjectionMatrixProvider::Eval(
    const lf::mesh::Entity &entity) {
  Eigen::MatrixXd elMat_vec;  // element matrix to be returned
  // Throw error in case cell is not Tria nor Quad
  LF_VERIFY_MSG(entity.RefEl() == lf::base::RefEl::kTria() ||
                    entity.RefEl() == lf::base::RefEl::kQuad(),
                "Unsupported cell type " << entity.RefEl());

  if (entity.RefEl() == lf::base::RefEl::kTria()) {
    elMat_vec = Eigen::MatrixXd::Zero(6, 6);
    // For TRIANGULAR CELLS
    // Compute the area of the triangle cell
    const double area = lf::geometry::Volume(*(entity.Geometry()));
    // Assemble the mass element matrix over the cell
    // clang-format off
      elMat_vec << 2.0, 0.0, 1.0, 0.0, 1.0, 0.0,
                   0.0, 2.0, 0.0, 1.0, 0.0, 1.0,
                1.0, 0.0, 2.0, 0.0, 1.0, 0.0,
                0.0, 1.0, 0.0, 2.0, 0.0, 1.0,
                1.0, 0.0, 1.0, 0.0, 2.0, 0.0,
                0.0, 1.0, 0.0, 1.0, 0.0, 2.0;
    // clang-format on
    elMat_vec *= area / 12.0;
  } else {
    // for QUADRILATERAL CELLS
    elMat_vec = Eigen::MatrixXd::Zero(8, 8);
    Eigen::MatrixXd elMat_scal =
        Eigen::MatrixXd::Zero(4, 4);  // element matrix for scalar FEM
    // Tensor product Gauss-Legendre quadrature rule of order 4
    const lf::quad::QuadRule qr{
        lf::quad::make_QuadRule(lf::base::RefEl::kQuad(), 3)};
    // Reference quadrature points
    const Eigen::MatrixXd zeta_ref{qr.Points()};
    // Quadrature weights
    const Eigen::VectorXd w_ref{qr.Weights()};
    // Number of quadrature points
    const lf::base::size_type P = qr.NumPoints();

    // Reference tensor product basis functions on quadrilateral ref entity
    std::vector<std::function<double(coord_t)>> ref_basis_vec;
    auto b0_ref = [](coord_t x) -> double { return (1 - x(0)) * (1 - x(1)); };
    auto b1_ref = [](coord_t x) -> double { return x(0) * (1 - x(1)); };
    auto b2_ref = [](coord_t x) -> double { return x(0) * x(1); };
    auto b3_ref = [](coord_t x) -> double { return (1 - x(0)) * x(1); };
    ref_basis_vec.push_back(b0_ref);
    ref_basis_vec.push_back(b1_ref);
    ref_basis_vec.push_back(b2_ref);
    ref_basis_vec.push_back(b3_ref);

    const lf::geometry::Geometry &geo{*(entity.Geometry())};
    const Eigen::VectorXd gram_dets{geo.IntegrationElement(zeta_ref)};
    for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
        for (int l = 0; l < P; l++) {
          elMat_scal(i, j) += w_ref[l] * ref_basis_vec[i](zeta_ref.col(l)) *
```