C++11 code 2.13.4: Sub-problem (2-13.b): Implementation of `Eval()` member function of AnisotropicDiffusionElementMatrixProvider *for quadrilaterals* ↬ GitLab

```
      // I. OBTAIN COORDINATES OF THE MIDPOINTS
      // I.i Hard-code the midpoints of the edges on the reference quadrilateral
      Eigen::Matrix<double, 2, 4> midpoints_ref(2, 4);
      midpoints_ref << 0.5, 1, 0.5, 0, 0, 0.5, 1, 0.5;
      // I.ii Obtain the midpoints of the parametrized quadrilateral
      auto midpoints_param = cell_geometry->Global(midpoints_ref);

      // II. COMPUTE LOCAL INTEGRATION DATA
      // II.i Compute the inverse of the tranposed Jacobian of the
      //      (J^T)^{-1} = J*(J^T*J)^{-1}
      // parametrization map at the midpoints of the reference quadrilateral
      const Eigen::MatrixXd JinvT(
          cell_geometry->JacobianInverseGramian(midpoints_ref));
      // II.ii Compute the integration element
      //      integration_element(x) = sqrt(det(J^T*J))
      // where J is the Jacobian of the parametrization map
      const Eigen::VectorXd integration_element(
          cell_geometry->IntegrationElement(midpoints_ref));
      // II.iii Hard-code a matrix-valued function that returns the gradients of
      // the reference basis functions evaluated at coordinates
      auto gradients_ref =
          [](Eigen::Vector2d x) -> Eigen::Matrix<double, 2, 4> {
        Eigen::Matrix<double, 2, 4> element_matrix;
        element_matrix(0, 0) = x(1) - 1;
        element_matrix(1, 0) = x(0) - 1;
        element_matrix(0, 1) = 1 - x(1);
        element_matrix(1, 1) = -x(0);
        element_matrix(0, 2) = x(1);
        element_matrix(1, 2) = x(0);
        element_matrix(0, 3) = -x(1);
        element_matrix(1, 3) = 1 - x(0);
        return element_matrix;
      };

      // III. PERFORM NUMERICAL QUADRATURE
      element_matrix = Eigen::MatrixXd::Zero(4, 4);
      for (int i = 0; i < 4; i++) {  // for each local degree of freedom
        // III.i Evaluate the diffusion tensor
        Eigen::Vector2d anisotropy_vec =
            anisotropy_vec_field_(midpoints_param.col(i));
        Eigen::Matrix2d diffusion_tensor =
            Eigen::Matrix2d::Identity(2, 2) +
            anisotropy_vec * anisotropy_vec.transpose();
        // III.ii Compute gradients of the global basis functions
        Eigen::Matrix<double, 2, 4> gradients_param{
            JinvT.block(0, 2 * i, 2, 2) * gradients_ref(midpoints_ref.col(i))};
        // III.iii Compute local contribution to the element matrix
        for (int j = 0; j < 4; j++) {
          for (int k = 0; k < 4; k++) {
            double integrand = (gradients_param.col(j).transpose() *
                                diffusion_tensor * gradients_param.col(k));
            element_matrix(j, k) += integration_element(i) * integrand;
          }
        }
      }
```