C++ code 5.2.35: Implementation of `coeff_sigma()` ⇝ GitLab

```
Eigen::VectorXd coeff_sigma(
    const Eigen::Matrix<double, 2, Eigen::Dynamic> &knots) {
  int M = knots.cols() - 1;
  Eigen::VectorXd sigma(M);
  // Definition of locations quadrature points (2-point Gauss rule)
  const double rh = .5 + .5 / std::sqrt(3.);
  const double lh = .5 - .5 / std::sqrt(3.);
  const double h = 1.0 / static_cast<double>(M);
  // Loop over all cells of the mesh
  for (int i = 0; i < M; ++i) {
    // Evaluate the y-component of uh on both quadrature points
    const double uh2l = (1 - lh) * knots(1, i) + lh * knots(1, i + 1);
    const double uh2r = (1 - rh) * knots(1, i) + rh * knots(1, i + 1);
    // Compute the derivative
    const Eigen::Vector2d duh = (1. / h) * (knots.col(i + 1) - knots.col(i));
    // Norm of the derivative vector
    const double duh_norm = duh.norm();
    // Compute sigma integrated over the segment normalized to unit
       length
    sigma[i] = 0.5 * (1. / (std::sqrt(-uh2r) * duh_norm) +
                      1. / (std::sqrt(-uh2l) * duh_norm));
  }
  return sigma;
}
```