C++11 code 3.3.9: Sub-problem (3-3.f), `GlobalInverseQuad()`: inverting parameterization ⇝ GitLab

```
  // Use corners to calculate matrix A, the translation t
  // and the coefficient vector d
  Eigen::Matrix2d A(2, 2);
  A.col(0) = corner1 - corner0;
  A.col(1) = corner3 - corner0;
  Eigen::Vector2d t = corner0;
  Eigen::Vector2d d = corner2 + corner0 - corner1 - corner3;

  // Calculate the coefficients in the non-linear system of equations
  Eigen::Vector2d y = A.partialPivLu().solve(x - t);
  Eigen::Vector2d q = A.partialPivLu().solve(d);
  Eigen::Vector2d q_orth(q(1), -q(0));

  // Treat exceptional cases
  // If q vanishes
  const double ref_size = y.norm();
  if (q.norm() < kEPS * ref_size) {
    x_hat = y;
  } else if (std::abs(q(0)) < kEPS * ref_size) {
    x_hat(0) = y(0);
    x_hat(1) = y(1) / (1 + q(1) * x_hat(0));
  } else if (std::abs(q(1)) < kEPS * ref_size) {
    x_hat(1) = y(1);
    x_hat(0) = y(0) / (1 + q(0) * x_hat(1));
  } else {
    // Generic case; solve quadratic equation
    double a = q(0);
    double b = 1 + y.dot(q_orth);
    double c = -y(1);

    auto [x_op1, x_op2] = solveQuadraticEquation(a, b, c);
    // No solution
    if (std::isnan(x_op1) || std::isnan(x_op2)) {
      x_hat(0) = NAN;
      x_hat(1) = NAN;
    } else {
      // There is a solution
      // Find root in the unit interval -> x_op1
      if ((x_op1 < 0.) || (x_op1 > 1.)) {
        if ((x_op2 >= 0.) && (x_op2 <= 1.)) {
          x_op1 = x_op2;
        }
      }
      x_hat(1) = x_op1;
      x_hat(0) = (y.dot(q_orth) + q(0) * x_hat(1)) / q(1);
    }
  }
```