

C++ code 3.13.7: Implementation of `qrStep()`

```
2  std::pair<Eigen::VectorXd, Eigen::VectorXd> qrStep(const Eigen::VectorXd &dT,
3                                                    const Eigen::VectorXd &uT) {
4      const long n = dT.size();
5      assert(uT.size() == n - 1);
6      // Defining vectors for  $T'$  (Written as  $T_p$ )
7      Eigen::VectorXd dT_p = Eigen::VectorXd::Zero(n);
8      Eigen::VectorXd uT_p = Eigen::VectorXd::Zero(n - 1);
9
10     // TODO: (3-13.c) Compute the defining vectors  $d(T')$  and  $u(T')$  of  $T' :=$ 
11     //  $RQ$  with
12     //  $O(n)$  asymptotic complexity, given the defining vectors  $d(T)$  and
13     //  $u(T)$  of  $T =$ 
14     //  $QR$ 
15     // START
16     // Auxiliary  $4 \times n$  matrix  $U \in \mathbb{R}^{4,n}$ 
17     Eigen::MatrixXd U{Eigen::MatrixXd::Zero(4, n)};
18     U.block(1, 1, 1, n - 1) = uT.transpose();
19     U.block(2, 0, 1, n) = dT.transpose();
20     U.block(3, 0, 1, n - 1) = uT.transpose();
21     // Stage I: QR decomposition by  $n-1$  Givens rotations
22     std::vector<Eigen::Matrix2d> givensrots; // Store rotations!
23     for (long i = 0; i < n - 1; ++i) {
24         const Eigen::Vector2d g = givens(Eigen::Vector2d(U(2, i), U(3, i)));
25         const Eigen::Matrix2d G{
26             (Eigen::Matrix2d(2, 2) << g[0], -g[1], g[1], g[0]).finished()};
27         U.block(2, i, 2, 1).applyOnTheLeft(G);
28         U.block(1, i + 1, 2, 1).applyOnTheLeft(G);
29         if (i < n - 2) {
30             U.block(0, i + 2, 2, 1).applyOnTheLeft(G);
31         }
32         givensrots.push_back(G);
33     }
34     // At this point  $U$  encodes the  $R$ -factor of the  $QR$ -decomposition of
35     //  $T$ . In particular, the bottom row of  $U$  should contain
36     // only entries that are "numerically zero".
37
38     // Stage II: Apply transposed rotations from right to  $R$  according to
39     // formulas (3.13.6)
40     for (long i = 1; i < n; ++i) {
41         const Eigen::Matrix2d &G{givensrots[i - 1]}; // An alias
42         Eigen::Vector2d tmp;
43         tmp = G * Eigen::Vector2d(U(3, i - 1), U(2, i));
44         U(3, i - 1) = tmp[0];
45         U(2, i) = tmp[1];
46         tmp = G * Eigen::Vector2d(U(2, i - 1), U(1, i));
47         U(2, i - 1) = tmp[0];
48         U(1, i) = tmp[1];
49         tmp = G * Eigen::Vector2d(U(1, i - 1), U(0, i));
50         U(1, i - 1) = tmp[0];
51         U(0, i) = tmp[1];
52     }
53     // Extract diagonal and sub-/super-diagonal of  $T'$ .
54     dT_p = U.block(2, 0, 1, n).transpose();
55     uT_p = U.block(1, 1, 1, n - 1).transpose();
56     // END
57
58     return {dT_p, uT_p};
59 }
```