

C++ code 3.18.7: Efficient implementation of `lowRankLsqSol()`

```
2 Eigen::VectorXd lowRankLsqSol(  
3     const LowRankMatrix &M, const Eigen::VectorXd &b,  
4     double tol = std::numeric_limits<double>::epsilon()) {  
5     assert(M.B_.rows() == b.size());  
6     // Maximal rank of matrix M  
7     unsigned int p = M.A_.cols();  
8     // Economical (!) QR decompositions of matrix factors  
9     Eigen::HouseholderQR<Eigen::MatrixXd> qrA(M.A_); //  
10    Eigen::HouseholderQR<Eigen::MatrixXd> qrB(M.B_); //  
11    // Extract R factors and form their product  
12    const Eigen::MatrixXd RA = //  
13        qrA.matrixQR().block(0, 0, p, p).template triangularView<Eigen::Upper>();  
14    const Eigen::MatrixXd RB = //  
15        qrB.matrixQR().block(0, 0, p, p).template triangularView<Eigen::Upper>();  
16    // Compute SVD of  $R_A R_B^T$   
17    Eigen::JacobiSVD<Eigen::MatrixXd> svd_object(  
18        RA * (RB.transpose()),  
19        Eigen::ComputeThinU | Eigen::ComputeThinV); //  
20    // Determine numerical rank  
21    const Eigen::VectorXd sv = svd_object.singularValues();  
22    const unsigned int n = sv.size();  
23    unsigned int r = 0;  
24    while ((r < n) && (sv(r) >= sv(0) * M.A_.rows() * M.B_.rows() * tol)) r++;  
25    // Compute SVD factors  
26    const Eigen::MatrixXd U =  
27        qrA.householderQ() *  
28        (Eigen::MatrixXd::Identity(M.A_.rows(), p) * svd_object.matrixU()); //  
29  
30    const Eigen::MatrixXd V =  
31        qrB.householderQ() *  
32        (Eigen::MatrixXd::Identity(M.B_.rows(), p) * svd_object.matrixV()); //  
33    return V.leftCols(r) *  
34        (sv.head(r).cwiseInverse().asDiagonal() * //  
35         (U.leftCols(r).transpose() * b));  
36 }
```