

C++ code 3.15.6: EIGEN-based implementation of `eco_svd_concat()`

```
2  std::tuple<Eigen::MatrixXd, Eigen::VectorXd, Eigen::MatrixXd> eco_svd_concat(
3      const Eigen::MatrixXd &A1, const Eigen::MatrixXd &B1,
4      const Eigen::MatrixXd &A2, const Eigen::MatrixXd &B2) {
5      // Matrix dimensions and consistency checks
6      const unsigned int m = A1.rows();
7      assert(A2.rows() == m);
8      const unsigned int n = B1.rows();
9      assert(B2.rows() == n);
10     const unsigned int k = A1.cols();
11     assert((k == A2.cols()) && (k == B1.cols()) && (k == B2.cols()));
12     // Matrices/vector to be returned by the function
13     Eigen::MatrixXd U = Eigen::MatrixXd::Zero(m, 2 * k);
14     Eigen::VectorXd s = Eigen::VectorXd::Zero(2 * k);
15     Eigen::MatrixXd V = Eigen::MatrixXd::Zero(2 * n, 2 * k);
16
17     // TODO: (3-15.b) Compute the economical SVD with optimal complexity
18     // and store
19     // the results in U, s and V.
20     // START
21
22     // Computation of economical QR-decompositions of B-factors
23     // incurs a computational effort of
24     //  $O(nk^2)$ 
25     Eigen::HouseholderQR<Eigen::MatrixXd> qrB1(B1); //
26     const Eigen::MatrixXd Q1{qrB1.householderQ() *
27                             Eigen::MatrixXd::Identity(n, k)}; //
28     Eigen::HouseholderQR<Eigen::MatrixXd> qrB2(B2); //
29     const Eigen::MatrixXd Q2{qrB2.householderQ() *
30                             Eigen::MatrixXd::Identity(n, k)}; //
31     // Computation of the matrix Z as defined in (3.15.5)
32     // Uses EIGEN's matrix concatenation operator
33
34     Eigen::MatrixXd Z(m, 2 * k);
35     Z << (A1 * (qrB1.matrixQR()
36              .block(0, 0, k, k)
37              .template triangularView<Eigen::Upper>())
38          .transpose()),
39          (A2 * (qrB2.matrixQR()
40              .block(0, 0, k, k)
41              .template triangularView<Eigen::Upper>())
42          .transpose());
43
44     // Economical SVD of Z
45     Eigen::JacobiSVD<Eigen::MatrixXd> svdZ(
46         Z, Eigen::ComputeThinU | Eigen::ComputeThinV); //
47     // Extract desired factors of final economical SVD of X
48     U = svdZ.matrixU(); // left factor of eco-SVD of X
49     s = svdZ.singularValues(); // singular values in a vector
50     // Build right factor of final economical SVD
51     Eigen::MatrixXd VZ{svdZ.matrixV()};
52     V.block(0, 0, n, 2 * k) = Q1 * VZ.block(0, 0, k, 2 * k); //
53     V.block(n, 0, n, 2 * k) = Q2 * VZ.block(k, 0, k, 2 * k); //
54     // END
55
56     return {U, s, V};
57 }
```