

SOLUTION of (2-9.d):

It is important to communicate to EIGEN that **R** is upper triangular so that the library knows that it can skip the costly elimination step.

C++11-code 2.9.3: Block Gauss based solution

```
2 void solveelse(const Eigen::MatrixXd& R, const Eigen::VectorXd& v,  
3               const Eigen::VectorXd& u, const Eigen::VectorXd& bb,  
4               Eigen::VectorXd& x) {  
5     // size of R, which is size of u, v, and size of bb is n+1  
6     const unsigned int n = R.rows();  
7  
8     // TODO: (2-9.d)  
9     // i) Use assert() to check that R, v, u, bb all have the appropriate  
10    // sizes.  
11    // ii) Use (3-9.b) to solve the LSE.  
12    // Hint: Use R.triangularView<Eigen::Upper>() to make use of the  
13    // triangular  
14    // structure of R.  
15    // START  
16    assert(n == R.cols() && n == u.size() && n == v.size() &&  
17           n + 1 == bb.size() && "Size mismatch!");  
18  
19    const Eigen::TriangularView<const Eigen::MatrixXd, Eigen::Upper>& triR =  
20    R.triangularView<Eigen::Upper>();  
21  
22    // s is the Schur complement and, in this case, is a scalar  
23    // also  $b_s$  is a scalar  
24    //  $s_{nv} = s^{-1}$ ,  $b_s$  as in lecture notes  
25    //  $s_{invbs} = s^{-1} * b_s$   
26    const double sinv = -1. / u.dot(triR.solve(v));  
27    const double bs = (bb(n) - u.dot(triR.solve(bb.head(n))));  
28    const double sinvbs = sinv * bs;  
29  
30    // Stack the vector  $(z, \xi)^T =: x$   
31    x = Eigen::VectorXd::Zero(n + 1);  
32    x << triR.solve(bb.head(n) - v * sinvbs), sinvbs;  
33    // END  
34 }
```

Get it on 🐶 GitLab (blockdecomp.hpp).