

C++ code 3.10.11: Sub-problem (3-10.e): The function `solveNormalEquations()`

```
2 Eigen::VectorXd solveNormalEquations(const Eigen::MatrixXd& D) {
3     Eigen::VectorXd x;
4     const unsigned int n = D.cols();
5     const unsigned int m = n * (n - 1) / 2;
6     // TODO: (3-10.e) Solve the normal equations by exploiting the
7     // Sherman-Morrison-Woodbury formula, i.e. by using your results from the
8     // previous subproblem
9     // START
10    // initialize right hand side
11    Eigen::VectorXd b = Eigen::VectorXd::Zero(m);
12    unsigned int curr = 0;
13    for (unsigned int i = 0; i < n - 1; ++i) {
14        for (unsigned int k = i + 1; k < n; ++k) {
15            b(curr) = D(i, k);
16            ++curr;
17        }
18    }
19    // Direct computation of A^T*b
20    Eigen::SparseMatrix<double> AT = initA(n).transpose();
21    Eigen::VectorXd ATb = AT * b;
22
23    // Alternative version
24
25    // Initialize the extended D
26    Eigen::MatrixXd D_full = Eigen::MatrixXd::Zero(n, n);
27    D_full.triangularView<Eigen::Upper>() =
28        D.triangularView<Eigen::Upper>(); // in case D is not just a upper
29                                           // triangular matrix
30    Eigen::MatrixXd D_fullT = D_full.transpose();
31    D_full = D_full - D_fullT;
32    // Calculate ATb in a more efficient way
33    Eigen::VectorXd ATb_alt =
34        D_full.transpose().block(0, 0, n - 1, n) * Eigen::VectorXd::Ones(n);
35
36    // We use ATb on forward but ATb == ATb_alt
37
38    // Use explicit expression of the inverse of A^T*A
39    // to compute the solution
40    x = (Eigen::MatrixXd::Identity(n - 1, n - 1) +
41        Eigen::MatrixXd::Ones(n - 1, n - 1)) *
42        ATb / n;
43    // END
44    return x;
45 }
```

Get it on 🐘 GitLab (distfitting.hpp).