

C++ code 2.19.12: Solving Sylvester equation (2.19.4).

```
2 Eigen::MatrixXd solveSpecialSylvesterEq(const Eigen::SparseMatrix<double> &A) {
3     const int n = A.rows();
4     assert(n == A.cols() && "Matrix A must be square.");
5     Eigen::MatrixXd X(n, n);
6
7     // TODO: (2-19.j) Solve the equation  $X \cdot A^{-1} + A \cdot X = I$  .
8
9     // START
10    // The equation is equivalent to  $X + A \cdot X \cdot A = A$ ,
11    // which has the system matrix  $C = \text{kron}(I, I) + \text{kron}(A, A)$  .
12    // Define left hand side of  $C \cdot \text{Vec}(X) = b$ .
13    Eigen::SparseMatrix<double> C(n * n, n * n);
14    C.setIdentity();
15    // May involve expensive copying of data?
16    C += sparseKron(A);
17
18    // Define right hand side.
19    // Do not use a "sparse vector"!
20    Eigen::VectorXd b(n * n);
21    b.setZero();
22    const int *JA = A.outerIndexPtr();
23    const int *IA = A.innerIndexPtr();
24    const double *valA = A.valuePtr();
25    // Form b by stacking the columns of A.
26    // Iterate over columns of A:
27    for (int j = 0; j < n; j++) {
28        // Loop over the non-zero entries in column j:
29        for (int s = JA[j]; s < JA[j + 1]; s++) {
30            const int i = IA[s]; // Row index of non-zero entry #s.
31            b[j * n + i] = valA[s]; //  $b(j \cdot n + i) = A_{ij}$ 
32        }
33    }
34    // Call sparse direct solver.
35    Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
36    // Since C is s.p.d., we could also use
37    // SimplicialLLT<Eigen::SparseMatrix<double>> solver;
38    solver.compute(C);
39    //  $x = \text{Vec}(X)$ 
40    Eigen::VectorXd x;
41    x = solver.solve(b);
42    X = Eigen::MatrixXd::Map(x.data(), n, n);
43    // END
44
45    return X;
46 }
```

Get it on 🐱 GitLab (Sylvester.hpp).