

C++ code 8.14.17: Sub-problem (8-14.g): The function `symmRankOneApprox()`

```
2 Eigen::VectorXd symmRankOneApprox(const Eigen::MatrixXd &M,
3                                 const double rtol = 1e-6,
4                                 const double atol = 1e-8) {
5     // ensure M is square
6     assert(M.cols() == M.rows() && "Matrix must be square");
7
8     const unsigned int n = M.cols();
9     // Initial guess: Column of symmetric part of M with largest norm
10    auto Msym = 0.5 * (M + M.transpose());
11    Eigen::Index j, dummy;
12    Msym.colwise().norm().maxCoeff(&dummy, &j);
13    // Set initial guess. This vector is also used to return the result
14    Eigen::VectorXd z{Msym.col(j)};
15    // TODO: (8-14.g)
16    // solve equ. (8.14.1) using Gauss-Newton iteration with tolerances rtol and
17    // atol
18    // START
19    // Gauss-Newton correction vector
20    Eigen::VectorXd s(n);
21    // Direct access to the array storing the entries of M
22    const double *dataM = M.data();
23
24    std::cout << " Correction s to current newton step:\n";
25    // Main Gauss-Newton loop with correction-based termination
26    do {
27        // Compute b = vec(M) - kron(z,z)
28        const Eigen::MatrixXd Z = z * z.transpose();
29        const double *dataz = Z.data();
30        const Eigen::VectorXd b = Eigen::MatrixXd::Map(dataM, n * n, 1) -
31                                Eigen::MatrixXd::Map(dataz, n * n, 1);
32        // Solve normal equations
33        const double normalize = 2 * z.squaredNorm();
34        Eigen::VectorXd kronMult = computeKronProdVecMult(z, b);
35        s = (kronMult - z.dot(kronMult) / normalize * z) / normalize;
36        // Apply Gauss-Newton update
37        z = z + s;
38        std::cout << "|s| = " << s.norm() << std::endl;
39    } while (s.norm() > std::max(atol, rtol * z.norm()));
40    std::cout << "\n";
41    // END
42    return z;
43 }
```

Get it on 🐘 [GitLab](#) (`symrank1.hpp`).