

C++ code 5.15.21: Constructor of ClosedQuadraticSplineCurve

```
2  ClosedQuadraticSplineCurve::ClosedQuadraticSplineCurve(  
3      const Eigen::Matrix2Xd &p)  
4      : n_(p.cols()), p_(2, p.cols() + 1), t_(p.cols() + 1), x_(2, p.cols()) {  
5      assert((n_ > 2) && "At least three points have to be supplied");  
6      assert((n_ % 2 == 1) &&  
7          "Number of points must be odd!"); //  
8      // Save point coordinates, duplicate endpoints  
9      p_.col(0) = p.col(n_ - 1);  
10     p_.rightCols(n_) = p;  
11     // TODO: (5-15.c) Write the constructor.  
12     // START  
13     // Compute knot set by accumulating length of polygon segments  
14     Eigen::VectorXd lensum{n_};  
15     lensum[0] = (p_.col(1) - p_.col(0)).norm();  
16     for (unsigned int l = 1; l < n_; ++l) {  
17         lensum[l] = lensum[l - 1] + (p_.col(l + 1) - p_.col(l)).norm();  
18     }  
19     t_[0] = 0.0;  
20     for (unsigned int l = 1; l <= n_; ++l) {  
21         t_[l] = lensum[l - 1] / lensum[n_ - 1];  
22     }  
23     // Lengths of knot intervals  
24     const Eigen::VectorXd h = t_.tail(n_) - t_.head(n_);  
25     // Compute vector coefficients  $x^j$   
26     #ifndef SMART_LSE_SOLUTION  
27         // Initialize sparse matrix efficiently, see  
28         Eigen::SparseMatrix<double> A(n_, n_);  
29         A.reserve(Eigen::RowVectorXi::Constant(n_, 2));  
30         A.insert(0, 0) = 1.0;  
31         A.insert(0, n_ - 1) = 1.0;  
32         for (unsigned int k = 1; k < n_; ++k) {  
33             A.insert(k, k) = 1.0;  
34             A.insert(k, k - 1) = 1.0;  
35         }  
36         A.makeCompressed();  
37         // Compute LU-decomposition  
38         Eigen::SparseLU<Eigen::SparseMatrix<double>> solver(A);  
39         if (solver.info() != Eigen::Success) {  
40             throw std::runtime_error("Matrix factorization failed!");  
41         }  
42     #endif  
43     // Right-hand side vector for linear system of equations  
44     Eigen::VectorXd b(n_);  
45     // Treat first and second component  
46     for (unsigned int d = 0; d <= 1; ++d) {  
47         // Compile right-hand side vector  
48         const auto c{p_.row(d)};  
49         b[0] = (c[n_] - c[n_ - 1]) / h[n_ - 1] - (c[1] - c[0]) / h[0];  
50         for (unsigned int k = 1; k < n_; ++k) {  
51             b[k] = (c[k] - c[k - 1]) / h[k - 1] - (c[k + 1] - c[k]) / h[k];  
52         }  
53     #ifndef SMART_LSE_SOLUTION  
54         // Forward elimination in-situ in b  
55         for (unsigned int j = 1; j < n_; ++j) {  
56             b[j] -= b[j - 1];  
57         }  
58     #endif
```