

C++ code 5.12.9: Constructor of NotAKnotCubicSpline

```

2  NotAKnotCubicSpline::NotAKnotCubicSpline(Eigen::VectorXd t, Eigen::VectorXd y)
3      : t_(std::move(t)), y_(std::move(y)), c_(y_.size()) {
4      // TODO: (5-12.c) Modify the given constructor of a Natural Cubic
5      // Spline into
6      // an efficient constructor for Not-a-knot Cubic Spline
7      // START
8      const long n = t_.size() - 1;
9      assert(n > 2);
10     assert(n == y_.size() - 1);
11     // Sequence  $(h_i)_{i=1}^n$ , knots have to be sorted
12     auto h = (t_.tail(n) - t_.head(n)).array();
13     auto hs = h.pow(2); //  $h_i^2$ 
14     auto b = 1.0 / h; //  $b_i := h_i^{-1}$ 
15     auto a = 2.0 * (b.head(n - 1) + b.tail(n - 1)); //  $a_i$ 
16     // Initialize sparse coefficient matrix for the linear system of
17     // equations
18     // needed to compute the unknown slopes. Compared to [Lecture →
19     // (5.4.2.13)] only
20     // the first and last row has to be adjusted, see (??).
21     Eigen::SparseMatrix<double> M(n + 1, n + 1);
22     M.reserve(Eigen::RowVectorXi::Constant(n + 1, 3));
23     // Matrix row arising from the continuity of
24     M.insert(0, 0) = 6.0 / hs[0];
25     M.insert(0, 1) = (6.0 / hs[0] - 6.0 / hs[1]);
26     M.insert(0, 2) = -6.0 / hs[1];
27     for (int i = 1; i < n; ++i) {
28         M.insert(i, i - 1) = b[i - 1];
29         M.insert(i, i) = a[i - 1];
30         M.insert(i, i + 1) = b[i];
31     }
32     // Matrix due to continuity of  $s^{(3)}$  at  $t_{n-1}$ 
33     M.insert(n, n - 2) = 6.0 / hs[n - 2];
34     M.insert(n, n - 1) = 6.0 / hs[n - 2] - 6.0 / hs[n - 1];
35     M.insert(n, n) = -6.0 / hs[n - 1];
36     // Initialize right-hand-side vector, see (??)
37     Eigen::VectorXd rhs(n + 1);
38     rhs[0] = -12.0 / (hs[0] * h[0]) * (y_[0] - y_[1]) +
39             12.0 / (hs[1] * h[1]) * (y_[1] - y_[2]);
40     for (int i = 1; i < n; ++i) {
41         rhs[i] =
42             3.0 * ((y_[i] - y_[i - 1]) / hs[i - 1] + (y_[i + 1] - y_[i]) / hs[i]);
43     }
44     rhs[n] = -12.0 / (hs[n - 2] * h[n - 2]) * (y_[n - 2] - y_[n - 1]) +
45             12.0 / (hs[n - 1] * h[n - 1]) * (y_[n - 1] - y_[n]);
46     // Compute slopes by solving an (almost) tridiagonal linear system
47     // with asymptotic effort  $O(n)$  for  $n \rightarrow \infty$ .
48     Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
49     solver.compute(M);
50     if (solver.info() != Eigen::Success) {
51         throw std::runtime_error("LU factorization failed");
52     }
53     c_ = solver.solve(rhs);
54     if (solver.info() != Eigen::Success) {
55         throw std::runtime_error("Elimination failed");
56     }
57     if (checkC2()) {
58         std::cout << "C2 continuity satisfied" << std::endl;
59     }
60 }

```