

C++ code 6.3.3: Function `intpolyval()`

```
2 Eigen::VectorXd intpolyval(const Eigen::VectorXd& t, const Eigen::VectorXd& y,
3                           const Eigen::VectorXd& x) {
4     const unsigned int
5         n = t.size(), // no. of interpolation nodes = deg. of polynomial
6         N = x.size(); // no. of evaluation points
7
8     Eigen::VectorXd p = Eigen::VectorXd(N);
9
10    // Precompute the weights  $\lambda_i$  with effort  $O(n^2)$ 
11    Eigen::VectorXd lambda(n);
12    for (unsigned int k = 0; k < n; ++k) {
13        // little workaround: cannot subtract a vector from a scalar
14        // -> multiply scalar by vector of ones
15        lambda(k) =
16            1. /
17            ((t(k) * Eigen::VectorXd::Ones(k) - t.head(k)).prod() *
18             (t(k) * Eigen::VectorXd::Ones(n - k - 1) - t.tail(n - k - 1)).prod());
19    }
20    // Compute quotient of weighted sums of  $\frac{\lambda_i}{t-t_i}$ ,
21    // effort  $O(n)$ 
22    for (unsigned int i = 0; i < N; ++i) {
23        Eigen::VectorXd z = (x(i) * Eigen::VectorXd::Ones(n) - t);
24
25        // check if we want to evaluate at a node <-> avoid division by zero
26        double* ptr = std::find(z.data(), z.data() + n, 0.0);
27        if (ptr != z.data() + n) { // if ptr = z.data + n = z.end no zero was
28            found
29            p(i) = y(ptr - z.data()); // ptr - z.data gives the position of the
30            zero
31        } else {
32            Eigen::VectorXd mu = lambda.cwiseQuotient(z);
33            p(i) = (mu.cwiseProduct(y)).sum() / mu.sum();
34        }
35    }
36    return p;
37 }
```