

C++ code 5.10.5: Sub-problem (5-10.b): Hermite interpolation of a closed curve

```
2  std::vector<Eigen::Vector2d> closedHermiteInterpolant(  
3      std::vector<Eigen::Vector2d> &Sigma, const Eigen::VectorXd &x) {  
4      // Note: we assume that x is sorted  
5      assert(x.minCoeff() >= 0 && x.maxCoeff() <= 1 && "x out of range");  
6      std::vector<Eigen::Vector2d> res(x.size());  
7  
8      unsigned int n = Sigma.size();  
9      // extend Sigma with periodic condition  
10     Sigma.push_back(Sigma[0]);  
11  
12     // define variables delta_1,...,delta_n and lambda_0,...,lambda_n  
13     std::vector<Eigen::Vector2d> d(n);  
14     std::vector<double> delta(n);  
15     std::vector<double> lambda(n + 1);  
16  
17     lambda[0] = 0.0;  
18     for (unsigned int i = 0; i < n; ++i) {  
19         d[i] = Sigma[i + 1] - Sigma[i];  
20         delta[i] = d[i].norm();  
21         lambda[i + 1] = lambda[i] + delta[i];  
22     }  
23  
24     // define the slopes as in the definition of the interpolating  
25     // closed cubic Hermite curve  
26     std::vector<Eigen::Vector2d> slopes(n + 1);  
27     // TODO: (5-10.b)  
28     // START  
29     Eigen::Vector2d s =  
30         (delta[n - 1] / delta[0] * d[0] + delta[0] / delta[n - 1] * d[n - 1]) /  
31         (delta[n - 1] + delta[0]);  
32     slopes[0] = s / s.norm();  
33     for (unsigned int i = 1; i < n; ++i) {  
34         s = (delta[i] / delta[i - 1] * d[i - 1] + delta[i - 1] / delta[i] * d[i]) /  
35             (delta[i] + delta[i - 1]);  
36         slopes[i] = s / s.norm();  
37     }  
38     slopes[n] = slopes[0];  
39  
40     Eigen::VectorXd x_scl = x * lambda[n];  
41     unsigned int j = 0;  
42     // loop over sides of polygon  
43     for (unsigned int i = 1; i <= n; ++i) {  
44         // loop over evaluation points x  
45         while (j < x.size() && x_scl(j) <= lambda[i]) {  
46             // Local cubic hermite interpolation  
47             double coordx =  
48                 hermloceval(x_scl(j), lambda[i - 1], lambda[i], Sigma[i - 1](0),  
49                             Sigma[i](0), slopes[i - 1](0), slopes[i](0));  
50             double coordy =  
51                 hermloceval(x_scl(j), lambda[i - 1], lambda[i], Sigma[i - 1](1),  
52                             Sigma[i](1), slopes[i - 1](1), slopes[i](1));  
53             // store the next point  
54             res[j] = {coordx, coordy};  
55             ++j;  
56         }  
57         if (j == x.size()) break;
```