

C++ code 11.9.8: Sub-problem (11-9.b): The function `computeIsolePointsDQ()`

```
2 template <typename FFuncor>
3 Eigen::Matrix<double, 2, Eigen::Dynamic> computeIsolePointsDQ(
4     FFuncor &&F, Eigen::Vector2d y0, double T) {
5     // States will need to be reshaped to 2*(N+1).
6     Eigen::Matrix<double, 2, Eigen::Dynamic> States =
7         Eigen::Matrix2Xd::Zero(2, 1);
8     // TODO: (11-9.d) Solve the IVP as given in the task description by using the
9     // symmetric difference quotient for the gradient of F. You may copy code from
10    // above.
11    // START
12    // Get gradient of F by symmetric difference quotients with a
13    // span given by the root of the machine precision in order to curb
14    // cancellation as much as possible.
15    const double epsrt = std::sqrt(std::numeric_limits<double>::epsilon());
16    auto gradF = [epsrt, F](const Eigen::Vector2d &x) {
17        Eigen::Vector2d grad, dx, dy;
18        dx << epsrt, 0.;
19        dy << 0., epsrt;
20        grad(0) = (F(x + dx) - F(x - dx)) / (2 * epsrt);
21        grad(1) = (F(x + dy) - F(x - dy)) / (2 * epsrt);
22        return grad;
23    };
24    // The remainder as in computeIsolePoints()
25    auto rhs = [gradF](const Eigen::Vector2d &x) {
26        Eigen::Vector2d gradFx = gradF(x);
27        Eigen::Vector2d out(-gradFx(1), gradFx(0)); // rotate
28        out /= gradFx.norm(); // scale
29        return out;
30    };
31
32    Ode45<Eigen::Vector2d> integrator(rhs);
33    integrator.options().rtol = 0.00001;
34    integrator.options().atol = 1e-9;
35
36    std::vector<std::pair<Eigen::Vector2d, double>> sol = integrator.solve(y0, T);
37    const unsigned int N = sol.size() - 1;
38    States = Eigen::MatrixXd::Zero(2, N + 1);
39    for (unsigned int i = 0; i <= N; i++) {
40        States.col(i) = sol[i].first;
41    }
42    // END
43    return States;
44 }
```

Get it on 🐱 GitLab (contour.hpp).