

C++ code 12.8.5: Sub-problem (12-8.e): Auxiliary function `solveGenStageEquation()`

```
2 template <typename Functor, typename Jacobian>
3 Eigen::VectorXd solveGenStageEquation(Functor &&f, Jacobian &&J,
4                                     const Eigen::VectorXd &y,
5                                     const Eigen::VectorXd &b, double h,
6                                     double rtol = 1E-6, double atol = 1E-8) {
7     Eigen::VectorXd g = Eigen::VectorXd::Zero(
8         y.size()); // initial guess g=0 for Newton iteration.
9
10    // TODO: (12-8.e; optional) Solve one stage equation using the Newton
11    // iteration.
12    // START
13
14    // Need to solve the equation lhs(g) = g - h*f(y+g)/4 - b = 0.
15    auto lhs = [f, y, b, h](const Eigen::VectorXd &g) {
16        // lhs(g) = g - h*f(y+g)/4 - b
17        Eigen::VectorXd val = g - 0.25 * h * f(y + g) - b;
18        return val;
19    };
20    auto Jlhs = [J, y, h](const Eigen::VectorXd &g) {
21        // lhs(g) = g - h*f(y+g)/4 - b, so the Jacobian is
22        // Jlhs(g) = Id - h*Jf(y+g)/4
23        int dim = y.size();
24        Eigen::MatrixXd Jval =
25            Eigen::MatrixXd::Identity(dim, dim) - 0.25 * h * J(y + g);
26        return Jval;
27    };
28
29    // Perform Newton iterations:
30    Eigen::VectorXd delta =
31        -Jlhs(g).lu().solve(lhs(g)); // Newton correction term.
32    int iter = 0,
33        maxiter = 100; // If correction based termination does not work.
34    while (delta.norm() > atol && delta.norm() > rtol * g.norm() &&
35           iter < maxiter) {
36        g = g + delta;
37        delta = -Jlhs(g).lu().solve(lhs(g));
38        iter++;
39    }
40    g = g + delta; // Perform the final step
41    // END
42
43    return g;
44 }
```

Get it on 🐱 GitLab ([gradientflow.hpp](#)).