

C++ code 12.9.14: Implementation of `odeROW()`

```
2 template <typename STATETYPE, typename RHSTYPE, typename JACTYPE>
3 std::vector<STATETYPE> odeROW(RHSTYPE &&rhs, JACTYPE &&Jacobian,
4                               const STATETYPE &y0, double T, unsigned int M,
5                               const ROW &row) {
6     using MATTYPE = decltype(Jacobian(y0)); // Type for Jacobi matrix
7     unsigned int N = y0.size(); // Dimension of state space
8     std::vector<STATETYPE> y(M + 1); // Vector of computed states
9     // TODO: (12-9.b) implement ROW method for autonomous IVPs
10    // START
11    y[0] = y0;
12    std::vector<STATETYPE> k(row.s_); // Increments  $k_i$ 
13    std::vector<STATETYPE> z(row.s_); // Auxiliary states  $z_i$ 
14    const double h = T / M; // timestep size
15    // Main timestepping loop, y[m] contains the current iterate
16    for (unsigned int m = 0; m < M; ++m) {
17        const MATTYPE J = Jacobian(y[m]);
18        const MATTYPE IgJ = MATTYPE::Identity(N, N) - h * row.d_gamma_ * J;
19        auto IgJ_lu = IgJ.lu(); // Precompute LU decomposition
20        z[0] = y[m];
21        k[0] = IgJ_lu.solve(rhs(z[0])); // Just fwd./bkw. substitution
22        y[m + 1] = y[m] + h * row.b_[0] * k[0]; // Next state
23        for (unsigned int i = 1; i < row.s_; ++i) {
24            // Compute auxiliary states
25            z[i] = y[m];
26            for (unsigned int j = 0; j < i; ++j) {
27                z[i] += h * row.alpha_(i, j) * k[j];
28            }
29            // Compute increments
30            STATETYPE tmp = row.gamma_(i, 0) * k[0];
31            for (unsigned int j = 1; j < i; ++j) {
32                tmp += row.gamma_(i, j) * k[j];
33            }
34            k[i] =
35                IgJ_lu.solve(rhs(z[i]) + h * J * tmp); // Just fwd./bkw.
36                // substitution
37                // Update next state
38            y[m + 1] += h * row.b_[i] * k[i];
39        }
40    }
41    // END
42    return y;
43 }
```