SOLUTION of (6-2.a):

There are many ways to implement the formulas of [Lecture → Def. 6.4.0.9]. Thanks to the restriction to explicit RK-SSM, all merely involve $\mathbf{f}$-evaluations.

C++11-code 6.2.1: Solution of (6-2.a): Constructor and private data ↪ GITHUB

```
class RKIntegrator {
 public:
  // Constructor for the RK method.
  RKIntegrator(const Eigen::MatrixXd &A, const Eigen::VectorXd &b)
      : A_(A), b_(b), s_(b.size()) {
    assert(A.cols() == A.rows() && "Matrix must be square.");
    assert(A.cols() == b.size() && "Incompatible matrix/vector size.");
  }

  // Explicit Runge-Kutta numerical integrator
  template <class Function>
  std::vector<Eigen::VectorXd> solve(Function &&f, double T,
                                     const Eigen::VectorXd &y0, int M) const;

 private:
  // Butcher data
  const Eigen::MatrixXd A_;
  const Eigen::VectorXd b_;
  int s_;  // size of Butcher tableau
};
```

C++11-code 6.2.2: Solution of (6-2.a) Solve method ↪ GITHUB

```
template <typename Function>
std::vector<Eigen::VectorXd> RKIntegrator::solve(Function &&f, double T,
                                                 const Eigen::VectorXd &y0,
                                                 int M) const {
  int dim = y0.size();  // dimension
  double h = T / M;     // step size
  std::vector<Eigen::VectorXd> sol;
  sol.reserve(M + 1);

  // Initial data
  sol.push_back(y0);

  // RK looping tools
  Eigen::VectorXd incr(dim);
  std::vector<Eigen::VectorXd> k;
  k.reserve(s_);  // Runge-Kutta Increments

  // Stepping
  Eigen::VectorXd step(dim);
  for (int iter = 0; iter < M; ++iter) {
    // clear looping variables
    step.setZero();
    k.clear();
    // explicit RK
    k.push_back(f(sol.at(iter)));
```

```
27      step = step + b_[0] * k[0];
28      for (int i = 1; i < s_; ++i) {
29        incr.setZero();
30        for (int j = 0; j < i; ++j) {
31          incr += A_(i, j) * k[j];
32        }
33        k.push_back(f(sol.at(iter) + h * incr));
34        step += b_[i] * k.back();
35      }
36
37      // step forward
38      sol.push_back(sol[iter] + h * step);
39    }
40    return sol;
41  }
```