SOLUTION of (6-1.b):

The explicit Euler method is given by

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + h\mathbf{f}(t_k, \mathbf{Y}_k).$$

For the given differential equation we therefore obtain

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + h\mathbf{A}\mathbf{Y}_k.$$

C++11-code 6.1.2: Explicit Euler method. ↪ GITHUB

```
2 Eigen::MatrixXd eeulstep(const Eigen::MatrixXd& A, const Eigen::MatrixXd& Y0,
3                          double h) {
4   Eigen::MatrixXd res = Eigen::MatrixXd::Zero(Y0.rows(), Y0.cols());
5   res = Y0 + h * A * Y0;
6   return res;
7 }
```

The implicit Euler method is given by

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + h\mathbf{f}(t_{k+1}, \mathbf{Y}_{k+1}).$$

For the given differential equation this yields

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + h\mathbf{A}\mathbf{Y}_{k+1} \quad \Longrightarrow \quad \mathbf{Y}_{k+1} = (\mathbf{I} - h\mathbf{A})^{-1}\mathbf{Y}_k$$

Hence, every step of the single-step method entails solving a linear system of equations, which can be done using EIGEN's direct elimination solvers introduced in .

C++11-code 6.1.3: Implicit Euler method.↪ GITHUB

```
2 Eigen::MatrixXd ieulstep(const Eigen::MatrixXd& A, const Eigen::MatrixXd& Y0,
3                          double h) {
4   Eigen::MatrixXd res = Eigen::MatrixXd::Zero(Y0.rows(), Y0.cols());
5   int n = A.rows();
6   res = (Eigen::MatrixXd::Identity(n, n) - h * A).partialPivLu().solve(Y0);
7   return res;
8 }
```

Finally, the implicit mid-point method is given by

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + h\mathbf{f}\left(\tfrac{1}{2}(t_k + t_{k+1}), \tfrac{1}{2}(\mathbf{Y}_k + \mathbf{Y}_{k+1})\right),$$

hence

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + \tfrac{h}{2}\mathbf{A}(\mathbf{Y}_k + \mathbf{Y}_{k+1}) \quad \Longrightarrow \quad \mathbf{Y}_{k+1} = \left(\mathbf{I} - \tfrac{h}{2}\mathbf{A}\right)^{-1}\left(\mathbf{Y}_k + \tfrac{h}{2}\mathbf{A}\mathbf{Y}_k\right).$$

C++11-code 6.1.4: Implicit midpoint method.↪ GITHUB

```
2 Eigen::MatrixXd impstep(const Eigen::MatrixXd& A, const Eigen::MatrixXd& Y0,
```

```
                        double h) {
  Eigen::MatrixXd res = Eigen::MatrixXd::Zero(Y0.rows(), Y0.cols());
  int n = A.rows();
  res = (Eigen::MatrixXd::Identity(n, n) - h * 0.5 * A)
            .partialPivLu()
            .solve(Y0 + h * 0.5 * A * Y0);
  return res;
}
```

!

The method `PartialPivLu()` can also be replaced with the generic call `lu()`. The last code well showcases “expression programming” in EIGEN.