

SOLUTION of (11-1.b):

The **explicit Euler** method is given by

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

For the given differential equation we therefore obtain

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

C++11-code 11.1.2: Explicit Euler method.

```
2 Eigen::MatrixXd eeulstep(const Eigen::MatrixXd &A, const Eigen::MatrixXd &Y0,  
3 double h) {  
4     Eigen::MatrixXd Y1 = Y0;  
5     // TODO: (11-1.b) Implement ONE step of explicit euler applied to Y0,  
6     // for the ODE Y' = A*Y  
7     // START  
8     Y1 = Y0 + h * A * Y0;  
9     // END  
10    return Y1;  
11 }
```

Get it on  GitLab (matode.hpp).

The **implicit Euler** method is given by

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

For the given differential equation this yields

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

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 [Lecture → § 2.5.0.8].

C++11-code 11.1.3: Implicit Euler method.

```
2 Eigen::MatrixXd ieulstep(const Eigen::MatrixXd &A, const Eigen::MatrixXd &Y0,  
3 double h) {  
4     const unsigned int n = A.rows();  
5     Eigen::MatrixXd Y1 = Y0;  
6     // TODO: (11-1.b) Implement ONE step of implicit euler applied to Y0,  
7     // for the ODE Y' = A*Y  
8     // START  
9     Y1 = (Eigen::MatrixXd::Identity(n, n) - h * A).partialPivLu().solve(Y0);  
10    // END  
11    return Y1;  
12 }
```

Get it on  GitLab (matode.hpp).

Finally, the **implicit mid-point method** is given by

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

hence

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

C++11-code 11.1.4: Implicit midpoint method.

```
2 Eigen::MatrixXd impstep(const Eigen::MatrixXd &A, const Eigen::MatrixXd &Y0,  
3                        double h) {  
4     const unsigned int n = A.rows();  
5     Eigen::MatrixXd Y1 = Y0;  
6     // TODO: (11-1.b) Implement ONE step of implicit midpoint rule applied to Y0,  
7     // for the ODE  $Y' = A*Y$   
8     // START  
9     Y1 = (Eigen::MatrixXd::Identity(n, n) - h * 0.5 * A)  
10          .partialPivLu()  
11          .solve(Y0 + h * 0.5 * A * Y0);  
12     // END  
13     return Y1;  
14 }
```

Get it on  GitLab (matode.hpp).

!

The method `PartialPivLu()` can also be replaced with the generic call `lu()`.
