# Numerical Analysis, Simulation and Control

# 2

M.Sc. Laboratory Advanced Control (WS 22/23)

Prof. Dr.–Ing. habil. Thomas Meurer, Chair of Automation and Control Chair

## Additional Tasks

### 2.3 Numerical Integration of Ordinary Differential Equations

In the previous exercise and the preparation tasks for this exercise, continuous dynamical systems[1] were considered, which are given by ordinary differential equations of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad t > t_0 \tag{2.1}$$

with the initial condition $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$. While this form is beneficial for the general understanding of the system and theoretical considerations, e.g., stability and observability, the controllers are often running in a discrete-time domain, e.g., on embedded systems, with a fixed time step $t_s$. The implementation of these controllers requires transforming the continuous-time dynamical system (2.1) in a discrete-time dynamical system of the form

$$\boldsymbol{x}[k+1] = \boldsymbol{f}_\mathrm{d}(\boldsymbol{x}[k], \boldsymbol{u}[k]) \tag{2.2}$$

with the initial condition $\boldsymbol{x}[0] = \boldsymbol{x}_0$ on the discretized time grid with $t_k = k\, t_s + t_0$. The discrete-time dynamical system (2.2) determines the state of the system at the time instance $t_{k+1} = t_k + t_s$ given the current system state $\boldsymbol{x}[k]$ and input $\boldsymbol{u}[k]$. It can be determined from the continuous-time dynamical system by solving the initial value problem (IVP) (2.1) such that

$$\boldsymbol{f}_\mathrm{d}(\boldsymbol{x}[k], \boldsymbol{u}[k]) = \boldsymbol{x}[k] + \int_{t_k}^{t_k+t_s} \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t))\,\mathrm{d}t \tag{2.3}$$

with $\boldsymbol{x}[k] = \boldsymbol{x}(t_k)$ and $\boldsymbol{u}[k] = \boldsymbol{u}(t_k)$. Usually, it is assumed that the system input $\boldsymbol{u}(t)$ remains constant during the time interval $t \in [t_k,\, t_{k+1})$.

In general, the IVP for a continuous-time ODEs cannot be solved directly on a computer and has to be approximated numerically. An exception to this are cases where an analytical solution exists that can be determined using symbolic computation with a computer algebra system (CAS), e.g., Mathematica, Maxima, or MATLAB Symbolic Toolbox. The majority of the other cases are solved by numerical integration of the ODE with a given initial state. There, the state values are determined at specific time points or instances in a given time interval starting from the initial time with the initial state. Therefore, a large number of algorithms exist, which are suited for different types of ODEs and requirements regarding the accuracy and stability of the solution. MATLAB provides the general IVP

[1] In the following, it is assumed that the system has no additional time dependency except the system state $\boldsymbol{x}(t)$ and the system input $\boldsymbol{u}(t)$.

solver `ode45` for non-stiff and `ode15s` for stiff ODEs, which use an adaptive step size control to achieve a given accuracy. These are also built into SIMULINK as variable-step solvers. If the interval between two time points is sufficiently small and constant for all sample times, also fixed-step solver can be used to solve the ODE. Due to their constant computation time, they are also suited for real-time applications in embedded systems. In the following, four algorithms for fixed-step ODE solvers are presented with different approximation orders. For their derivation and detailed information, please refer to [2]. Most ODE solvers expect an ODE in the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{g}(t, \boldsymbol{x}(t)) \tag{2.4}$$

with the state variable $\boldsymbol{x}(t)$. Therefore, the dynamical system (2.1) has to be reformulated to this representation.

The explicit Euler method is the simplest integration algorithm and provides a linear approximation. The iteration is calculated by

$$\boldsymbol{x}[k+1] = \boldsymbol{x}[k] + t_s \boldsymbol{g}(t_k, \boldsymbol{x}[k]), \tag{2.5}$$

where $\boldsymbol{g}(t_k, \boldsymbol{x}[k])$ is considered as the slope of the tangent at the position $t_k$ and the current state $\boldsymbol{x}[k]$. Following this tangent for the time step $t_s$ results in the next state $\boldsymbol{x}[k+1]$. It requires only one evaluation of the ODE function and has the lowest computational costs. Nevertheless, it has poor accuracy due to its first-order approximation. This can be improved by combining it with subgrid methods, where the time step is divided into smaller subintervalls that are integrated.

The explicit Heun's method is a second-order ODE integration algorithm based on the trapezoidal rule. Its iteration is defined as

$$\begin{aligned} \tilde{\boldsymbol{x}}[k+1] &= \boldsymbol{x}[k] + t_s \boldsymbol{g}(t_k, \boldsymbol{x}[k]) \\ \boldsymbol{x}[k+1] &= \boldsymbol{x}[k] + \frac{t_s}{2}\left(\boldsymbol{g}(t_k, \boldsymbol{x}[k]) + \boldsymbol{g}(t_k + t_s, \tilde{\boldsymbol{x}}[k+1])\right) \end{aligned} \tag{2.6}$$

using the mean of the derivatives at the initial state $\boldsymbol{x}[k]$ and the intermediate state $\tilde{\boldsymbol{x}}[k+1]$, which is given by the Euler method.

The explicit Runge-Kutta method of 4th order calculates four derivatives at the beginning, middle and end of the time interval. The next iterate

$$\begin{aligned} \boldsymbol{s}_1 &= \boldsymbol{g}(t_k, \boldsymbol{x}[k]) \\ \boldsymbol{s}_2 &= \boldsymbol{g}\left(t_k + \frac{t_s}{2}, \boldsymbol{x}[k] + \frac{t_s}{2}\boldsymbol{s}_1\right) \\ \boldsymbol{s}_3 &= \boldsymbol{g}\left(t_k + \frac{t_s}{2}, \boldsymbol{x}[k] + \frac{t_s}{2}\boldsymbol{s}_2\right) \\ \boldsymbol{s}_4 &= \boldsymbol{g}(t_k + t_s, \boldsymbol{x}[k] + t_s \boldsymbol{s}_3) \\ \boldsymbol{x}[k+1] &= \boldsymbol{x}[k] + \frac{t_s}{6}(\boldsymbol{s}_1 + 2\boldsymbol{s}_2 + 2\boldsymbol{s}_3 + \boldsymbol{s}_4) \end{aligned} \tag{2.7}$$

is calculated from the weighted sum of the four derivative values. Despite it requires four evaluations of the ODE, it provides a very accurate solution due to its high order. Therefore, it is the most common fixed-step method to solve non-stiff ODEs.

The implicit Euler method is similar to the explicit Euler method with the iteration rule

$$\boldsymbol{x}[k+1] = \boldsymbol{x}[k] + t_s \boldsymbol{g}(t_k + t_s, \boldsymbol{x}[k+1]) \tag{2.8}$$

where the predicted state $\boldsymbol{x}[k+1]$ shows up at both sides of the equation, leading to a system of nonlinear equations. It can be reformulated to a root-finding problem solved by, e.g., the Newton–Raphson method. This may also require calculating the Jacobian of the ODE $\boldsymbol{g}(t, \boldsymbol{x}(t))$, additionally to the multiple evaluations of the ODE. Implicit methods are better suited for stiff ODEs than explicit methods. This means for systems with very large differences in the magnitudes of their eigenvalues.

**Exercise 2.9.** *Implement separate* MATLAB *functions for the explicit Euler, Heun's, Runge-Kutta 4th order and implicit Euler methods. Create a* SIMULINK *model with* Simulink MATLAB Functions *to solve the second-order ODE system of the autonomous mass-spring-damper system*

$$m\ddot{p} + d\dot{p} + kp = 0, \quad t > 0 \tag{2.9}$$

*with the initial condition* $p(0) = 1.0$ *and* $\dot{p}(0) = 0.5$. *Set the mass, damping and stiffness parameters to* $m = 2.0$, $d = 1.0$ *and* $k = 100.0$. *Create a* SIMULINK *model with* Simulink MATLAB Functions *to simulate the system over* 10 s.

*Compare your simulation results to the exact analytical solution of the mass-spring-damper system*

$$p(t) = e^{-\delta t}\left(p(0)\cos(\omega t) + \frac{\dot{p}(0) + \delta\, p(0)}{\omega}\sin(\omega t)\right) \tag{2.10a}$$

$$\dot{p}(t) = e^{-\delta t}\left(-p(0)\omega\sin(\omega t) + (\dot{p}(0) + \delta\, p(0))\cos(\omega t)\right) - \delta\, p(t) \tag{2.10b}$$

*with the natural frequency* $\omega_0 = \sqrt{\frac{k}{m}}$, *damping ration* $\delta = \frac{d}{2m}$ *and the angular frequency of the damped harmonics* $\omega = \sqrt{{\omega_0}^2 - \delta^2}$.

## 2.4 The Discrete-Time Extended Kalman Filter

In contrast to the continuous Kalman filter that calculates the estimated state $\hat{\boldsymbol{x}}$ and the covariance matrix of the estimation error $P$ simultaneously, the discrete-time EKF performs a prediction and an update step calculating a priori and a posteriori estimates. In the prediction step

$$\hat{\boldsymbol{x}}^-[k] = \boldsymbol{f}_\mathrm{d}(\hat{\boldsymbol{x}}[k-1], \boldsymbol{u}[k-1]) \tag{2.11}$$

$$P^-[k] = F_\mathrm{d}[k-1]\, P[k-1]\, F_\mathrm{d}{}^\mathsf{T}[k-1] + t_s Q \tag{2.12}$$

the a priori state estimate $\hat{\boldsymbol{x}}^-[k]$ is calculated from the state of the discrete-time system given the last state $\hat{\boldsymbol{x}}[k-1]$ estimate and system input $\boldsymbol{u}[k-1]$. Since, in general, the integral in (2.3) cannot be evaluated directly, the approximation methods described above are used below to represent $\boldsymbol{f}_\mathrm{d}(\boldsymbol{x}[k], \boldsymbol{u}[k])$. The a priori covariance matrix of the estimation error $P^-[k]$ follows from the covariance matrix of the previous iterate $P[k-1]$ that is transformed by the Jacobian of the discrete-time system dynamics

$$F_\mathrm{d} = \left.\frac{\partial \boldsymbol{f}_\mathrm{d}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}, \boldsymbol{u}}. \tag{2.13}$$

The approximation of the discrete-time system dynamics Jacobian based on the Jacobian of the continuous-time system is described below. Additionally, the process noise covariance matrix of the continuous-time system $Q$ is added with a scaling factor depending on the time step.

In the update step, first, the Kalman gain matrix is computed as

$$K[k] = P^-[k]\, H^\mathsf{T}[k] \left( H[k]\, P^-[k]\, H^\mathsf{T}[k] + \frac{1}{1} R \right)^{-1} \tag{2.14}$$

from the a priori state covariance $P^-[k]$, the measurement noise covariance matrix $R$ and the Jacobian of the measurement function

$$H = \left. \frac{\partial \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}} \right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}, \boldsymbol{u}}, \tag{2.15}$$

identical to (2.12). The a priori state estimate is corrected in the update step

$$\hat{\boldsymbol{x}}[k] = \hat{\boldsymbol{x}}^-[k] + \left( y[k] - h(\hat{\boldsymbol{x}}^-[k]) \right) \tag{2.16}$$

$$P[k] = (I - K[k]\, H[k])\, P^-[k] \tag{2.17}$$

to the a posteriori state estimate $\hat{\boldsymbol{x}}[k]$ with the error between the measured system output $y[k]$ and the estimated system output $\hat{y}[k] = h(\hat{\boldsymbol{x}}^-[k])$ transformed by the Kalman gain matrix $K[k]$. The a posteriori covariance matrix of the estimation error $P[k]$ updates the a priori covariance matrix $P^-[k]$ using the Kalman gain matrix and measurement Jacobian.

The Jacobian of the discrete-time system dynamics depends on the discretization method if it is given from a continuous-time model. For an ODE in the form (2.4) it is assumed that its Jacobian

$$G(t, \boldsymbol{x}(t)) = \left. \frac{\partial \boldsymbol{g}(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \right|_{\boldsymbol{x}=\boldsymbol{x}(t)} \tag{2.18}$$

is provided in an analytical form, e.g., from a CAS. The discrete-time Jacobian $G_\mathrm{d}$ for an ODE that was discretized using the Runge-Kutta of 4th order then follows from successive deriving as

$$\begin{aligned}
\boldsymbol{s}_1[k] &= \boldsymbol{g}\left(t_k, \boldsymbol{y}[k]\right) \\
\boldsymbol{s}_2[k] &= \boldsymbol{g}\left(t_k + \frac{t_s}{2}, \boldsymbol{y}[k] + \frac{t_s}{2}\, \boldsymbol{s}_1[k]\right) \\
\boldsymbol{s}_3[k] &= \boldsymbol{g}\left(t_k + \frac{t_s}{2}, \boldsymbol{y}[k] + \frac{t_s}{2}\, \boldsymbol{s}_2[k]\right) \\
\boldsymbol{s}_4[k] &= \boldsymbol{g}\left(t_k + t_s, \boldsymbol{y}[k] + t_s\, \boldsymbol{s}_3[k]\right) \\
S_1[k] &= G\left(t_k, \boldsymbol{y}[k]\right) \\
S_2[k] &= G\left(t_k + \frac{t_s}{2}, \boldsymbol{y}[k] + \frac{t_s}{2}\, \boldsymbol{s}_1[k]\right)\left(I + \frac{t_s}{2} S_1[k]\right) \\
S_3[k] &= G\left(t_k + \frac{t_s}{2}, \boldsymbol{y}[k] + \frac{t_s}{2}\, \boldsymbol{s}_2[k]\right)\left(I + \frac{t_s}{2} S_2[k]\right) \\
S_4[k] &= G\left(t_k + t_s, \boldsymbol{y}[k] + t_s\, \boldsymbol{s}_3[k]\right)(I + t_s S_3[k]) \\
G_\mathrm{d}[k] &= I + \frac{t_s}{6}\left(S_1[k] + 2\, S_2[k] + 2\, S_3[k] + S_4[k]\right) .
\end{aligned} \tag{2.19}$$

**Exercise 2.10.** *Implement a discrete-time EKF for state estimation of the double pendulum system Exercise 1.7. Use the Runge-Kutta method with order 4 as the fixed-step integration method and a sampling time of* 10 ms.

*Test the system with the same parameters as in Exercise 1.8 of the preparation exercise and compare the estimated states to the states of the original system.*