Let $t = t_k$ (some discrete time point) and let the initial time $t_0 = t_{k-1}$ (the previous discrete time point). Assume that $A(\tau)$, $B(\tau)$, and $u(\tau)$ are approximately constant in the interval of integration. We then obtain

$$x(t_k) = e^{A(t_k - t_{k-1})} x(t_{k-1}) + \int_{t_{k-1}}^{t_k} e^{A(t_k - \tau)}\, d\tau\, B u(t_{k-1}) \tag{1.101}$$

Now define $\Delta t = t_k - t_{k-1}$, define $\alpha = \tau - t_{k-1}$, and substitute for τ in the above equation to obtain

$$\begin{aligned} x(t_k) &= e^{A\Delta t} x(t_{k-1}) + \int_0^{\Delta t} e^{A(\Delta t - \alpha)}\, d\alpha\, B u(t_{k-1}) \\ &= e^{A\Delta t} x(t_{k-1}) + e^{A\Delta t} \int_0^{\Delta t} e^{-A\alpha}\, d\alpha\, B u(t_{k-1}) \\ x_k &= F_{k-1} x_{k-1} + G_{k-1} u_{k-1} \end{aligned} \tag{1.102}$$

where x_k, F_k, G_k, and u_k are defined by the above equation. This is a linear discrete-time approximation to the continuous-time dynamics given in Equation (1.67). Note that this discrete-time system defines x_k only at the discrete time points $\{t_k\}$; it does not say anything about what happens to the continuous-time signal $x(t)$ in between the discrete time points.

The difficulty with the above discrete-time system is the computation of the integral of the matrix exponential, which is necessary in order to compute the G matrix. This computation can be simplified if A is invertible:

$$\begin{aligned} \int_0^{\Delta t} e^{-A\tau} d\tau &= \int_0^{\Delta t} \sum_{j=0}^{\infty} \frac{(-A\tau)^j}{j!}\, d\tau \\ &= \int_0^{\Delta t} \left[I - A\tau + A^2\tau^2/2! - \cdots \right] d\tau \\ &= \left[I\tau - A\tau^2/2! + A^2\tau^3/3! - \cdots \right]_0^{\Delta t} \\ &= \left[I\Delta t - A(\Delta t)^2/2! + A^2(\Delta t)^3/3! - \cdots \right] \\ &= \left[A\Delta t - (A\Delta t)^2/2! + (A\Delta t)^3/3! - \cdots \right] A^{-1} \\ &= \left[I - e^{-A\Delta t} \right] A^{-1} \end{aligned} \tag{1.103}$$

The conversion from continuous-time system matrices A and B to discrete-time system matrices F and G can be summarized as follows:

$$\begin{aligned} F &= e^{A\Delta t} \\ G &= F \int_0^{\Delta t} e^{-A\tau}\, d\tau\, B \\ &= F \left[I - e^{-A\Delta t} \right] A^{-1} B \end{aligned} \tag{1.104}$$

where Δt is the discretization step size.

1.5 SIMULATION

In this section, we discuss how to simulate continuous-time systems (either linear or nonlinear) on a digital computer. We consider the following form of the general

system equation from Equation (1.83):

$$\dot{x} = f(x, u, t) \tag{1.105}$$

where $u(t)$ is a known control input. In order to simulate this system on a computer, we need to program a computer to solve for $x(t_f)$ at some user-specified value of t_f. In other words, we want to compute

$$x(t_f) = x(t_0) + \int_{t_0}^{t_f} f[x(t), u(t), t]\, dt \tag{1.106}$$

Often, the initial time is taken as $t_0 = 0$, in which case we have the slightly simpler looking equation

$$x(t_f) = x(0) + \int_{0}^{t_f} f[x(t), u(t), t]\, dt \tag{1.107}$$

We see that in order to find the solution $x(t_f)$ to the differential equation $\dot{x} = f(x, u, t)$, we need to compute an integral. The problem of finding the solution $x(t_f)$ is therefore commonly referred to as an integration problem.

Now suppose that we divide the time interval $[0, t_f]$ into L equally spaced intervals so that $t_k = kT$ for $k = 0, \cdots, L$, and the time interval $T = t_f/L$. From this we note that $t_f = t_L$. With this division of the time interval, we can write the solution of Equation (1.107) as

$$\begin{aligned} x(t_f) &= x(t_L) \\ &= x(0) + \sum_{k=0}^{L} \int_{t_k}^{t_{k+1}} f[x(t), u(t), t]\, dt \end{aligned} \tag{1.108}$$

More generally, for some $n \in [0, L-1]$, we can write $x(t_n)$ and $x(t_{n+1})$ as

$$\begin{aligned} x(t_n) &= x(0) + \sum_{k=0}^{n} \int_{t_k}^{t_{k+1}} f[x(t), u(t), t]\, dt \\ x(t_{n+1}) &= x(0) + \sum_{k=0}^{n+1} \int_{t_k}^{t_{k+1}} f[x(t), u(t), t]\, dt \end{aligned} \tag{1.109}$$

which means that

$$x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} f[x(t), u(t), t]\, dt \tag{1.110}$$

If we can find a way to approximate the integral on the right side of the above equation, we can repeatedly propagate our $x(t)$ approximation from time t_n to time t_{n+1}, thus obtaining an approximation for $x(t)$ at any desired time t. The algorithm could look something like the following.

Differential equation solution

Assume that $x(0)$ is given
for t $= 0 : T : t_f - T$
  Find an approximation $I(t) \approx \int_t^{t+T} f[x(t), u(t), t]\, dt$
  $x(t+T) = x(t) + TI(t)$
end

In the following sections, we present three different ways to approximate this integral. The approximations, in order of increasing computational effort and increasing accuracy, are rectangular integration, trapezoidal integration, and fourth-order Runge–Kutta integration.

1.5.1 Rectangular integration

If the time interval $(t_{n+1} - t_n)$ is small, then $f[x(t), u(t), t]$ is approximately constant in this interval. Equation (1.110) can therefore be approximated as

$$\begin{aligned} x(t_{n+1}) &\approx x(t_n) + \int_{t_n}^{t_{n+1}} f[x(t_n), u(t_n), t_n]\, dt \\ &\approx x(t_n) + f[x(t_n), u(t_n), t_n]T \end{aligned} \tag{1.111}$$

Equation (1.109) can therefore be approximated as

$$\begin{aligned} x(t_n) &\approx x(0) + \sum_{k=0}^{n} \int_{t_k}^{t_{k+1}} f[x(t_k), u(t_k), t_k]\, dt \\ &= x(0) + \sum_{k=0}^{n} f[x(t_k), u(t_k), t_k]T \end{aligned} \tag{1.112}$$

This is called Euler integration, or rectangular integration, and is illustrated in Figure 1.2. As long as T is sufficiently small, this gives a good approximation for $x(t_n)$.

This gives the following algorithm for integrating continuous-time dynamics using rectangular integration. The time loop in the algorithm is executed for $t = 0, T, 2T, \cdots, t_f - T$.

Rectangular integration

Assume that $x(0)$ is given
for t $= 0 : T : t_f - T$
  Compute $f[x(t), u(t), t]$
  $x(t+T) = x(t) + f[x(t), u(t), t]T$
end

1.5.2 Trapezoidal integration

An inspection of Figure 1.2 suggests an idea for improving the approximation for $x(t)$. Instead of approximating each area as a rectangle, what if we approximate each area as a trapezoid? Figure 1.3 shows how an improved integration algorithm can be implemented. This is called modified Euler integration, or trapezoidal integration. A comparison of Figures 1.2 and 1.3 shows that trapezoidal integration

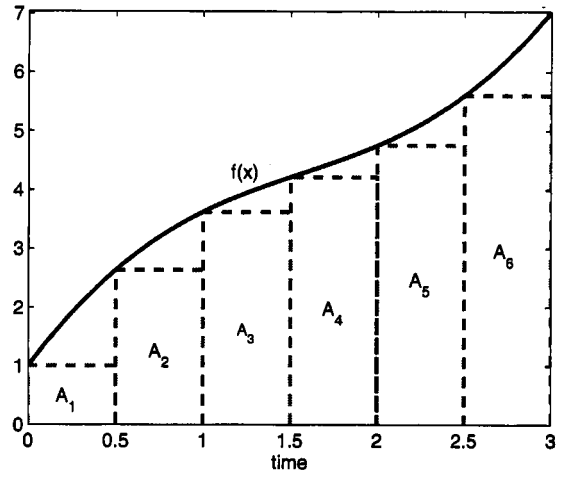

Figure 1.2 An illustration of rectangular integration. We have $\dot{x} = f(x)$, so $x(t)$ is the area under the $f(x)$ curve. This area can be approximated as the sum of the rectangular areas A_i. That is, $x(0.5) \approx A_1$, $x(1) \approx A_1 + A_2, \cdots$.

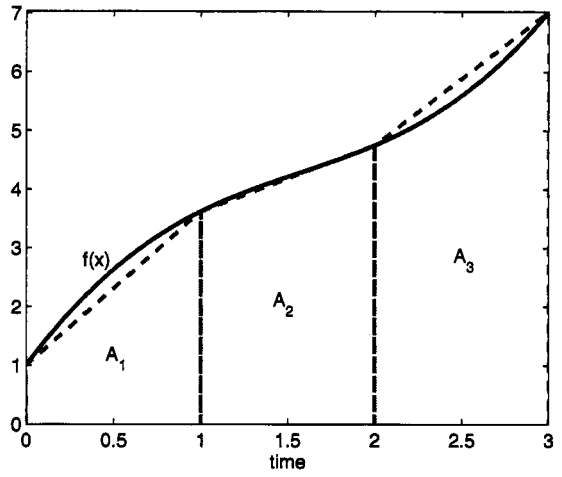

Figure 1.3 An illustration of trapezoidal integration. We have $\dot{x} = f(x)$, so $x(t)$ is the area under the $f(x)$ curve. This area can be approximated as the sum of trapezoidal areas A_i. That is, $x(1) \approx A_1$, $x(2) \approx A_1 + A_2$, and $x(3) \approx A_1 + A_2 + A_3$.

appears to give a better approximation than rectangular integration, even though the time axis is only divided into half as many intervals in trapezoidal integration.

With rectangular integration we approximated $f[x(t), u(t), t]$ as a constant in the interval $t \in [t_n, t_{n+1}]$. With trapezoidal integration, we instead approximate $f[x(t), u(t), t]$ as a linear function in the interval $t \in [t_n, t_{n+1}]$. That is,

$$f[x(t)] \approx f[x(t_n), u(t_n), t_n] + \left(\frac{f[x(t_{n+1}), u(t_{n+1}), t_{n+1}] - f[x(t_n), u(t_n), t_n]}{T}\right)(t - t_n)$$
$$\text{for } t \in [t_n, t_{n+1}] \tag{1.113}$$

Equation (1.110) can therefore be approximated as

$$\begin{aligned} x(t_{n+1}) &\approx x(t_n) + \int_{t_n}^{t_{n+1}} \left\{ f[x(t_n), u(t_n), t_n] + \left(\frac{f[x(t_{n+1}), u(t_{n+1}), t_{n+1}] - f[x(t_n), u(t_n), t_n]}{T}\right)(t - t_n) \right\} dt \\ &= x(t_n) + \left(\frac{f[x(t_n), u(t_n), t_n] + f[x(t_{n+1}), u(t_{n+1}), t_{n+1}]}{2}\right) T \\ &= x(t_n) + \frac{1}{2}(f[x(t_n), u(t_n), t_n]T + f[x(t_{n+1}), u(t_{n+1}), t_{n+1}]T) \end{aligned} \tag{1.114}$$

This equation to approximate $x(t_{n+1})$, however, has $x(t_{n+1})$ on the right side of the equation. How can we plug $x(t_{n+1})$ into the right side of the equation if we do not yet know $x(t_{n+1})$? The answer is that we can use the rectangular integration approximation from the previous section for $x(t_{n+1})$ on the right side of the equation. The above equation can therefore be written as

$$\begin{aligned} \Delta x_1 &= f[x(t_n), u(t_n), t_n]T \\ \Delta x_2 &= f[x(t_{n+1}), u(t_{n+1}), t_{n+1}]T \\ &\approx f[x(t_n) + \Delta x_1, u(t_{n+1}), t_{n+1}]T \\ x(t_{n+1}) &\approx x(t_n) + \frac{1}{2}(\Delta x_1 + \Delta x_2) \end{aligned} \tag{1.115}$$

This gives the following algorithm for integrating continuous-time dynamics using trapezoidal integration. The time loop in the algorithm is executed for $t = 0, T, 2T, \cdots, t_f - T$.

Trapezoidal integration
Assume that $x(0)$ is given
for t = $0 : T : t_f - T$
$\Delta x_1 = f[x(t), u(t), t]T$
$\Delta x_2 = f[x(t) + \Delta x_1, u(t + T), t + T]T$
$x(t + T) = x(t) + (\Delta x_1 + \Delta x_2)/2$
end

1.5.3 Runge–Kutta integration

From the previous sections, we see that rectangular integration involves the calculation of one function value at each time step, and trapezoidal integration involves the calculation of two function values at each time step. In order to further improve the integral approximation, we can perform additional function calculations at each time step. nth-order Runge–Kutta integration is the approximation of an integral

by performing n function calculations at each time step. Rectangular integration is therefore equivalent to first-order Runge–Kutta integration, and trapezoidal integration is equivalent to second-order Runge–Kutta integration.

The most commonly used integration scheme of this type is fourth-order Runge–Kutta integration. We present the fourth-order Runge–Kutta integration algorithm (without derivation) as follows:

$$\begin{aligned} \Delta x_1 &= f[x(t_k), u(t_k), t_k]T \\ \Delta x_2 &= f[x(t_k) + \Delta x_1/2, u(t_{k+1/2}), t_{k+1/2}]T \\ \Delta x_3 &= f[x(t_k) + \Delta x_2/2, u(t_{k+1/2}), t_{k+1/2}]T \\ \Delta x_4 &= f[x(t_k) + \Delta x_3, u(t_{k+1}), t_{k+1}]T \\ x(t_{k+1}) &\approx x(t_k) + (\Delta x_1 + 2\Delta x_2 + 2\Delta x_3 + \Delta x_4)/6 \end{aligned} \tag{1.116}$$

where $t_{k+1/2} = t_k + T/2$. Fourth-order Runge–Kutta integration is more computationally demanding than rectangular or trapezoidal integration, but it also provides far greater accuracy. This gives the following algorithm for integrating continuous-time dynamics using fourth-order Runge–Kutta integration. The time loop in the algorithm is executed for $t = 0, T, 2T, \cdots, t_f - T$.

Fourth-order Runge–Kutta integration

Assume that $x(0)$ is given
for t = $0 : T : t_f - T$
  $t_1 = t + T/2$
  $\Delta x_1 = f[x(t), u(t), t]T$
  $\Delta x_2 = f[x(t) + \Delta x_1/2, u(t_1), t_1]T$
  $\Delta x_3 = f[x(t) + \Delta x_2/2, u(t_1), t_1]T$
  $\Delta x_4 = f[x(t) + \Delta x_3, u(t+T), t+T]T$
  $x(t+T) = x(t) + (\Delta x_1 + 2\Delta x_2 + 2\Delta x_3 + \Delta x_4)/6$
end

Runge–Kutta integration was invented by Carl Runge, a German mathematician and physicist, in 1895. It was independently invented and generalized by Wilhelm Kutta, a German mathematician and aerodynamicist, in 1901. More accurate integration algorithms have also been derived and are sometimes used, but fourth-order Runge–Kutta integration is generally considered a good trade-off between accuracy and computational effort. Further information and derivations of numerical integration algorithms can be found in many numerical analysis texts, including [Atk89].

■ **EXAMPLE 1.5**

Suppose we want to numerically compute $x(t)$ at $t = 1$ based on the differential equation

$$\dot{x} = \cos t \tag{1.117}$$

with the initial condition $x(0) = 0$. We can analytically integrate the equation to find out that $x(1) = \sin 1 \approx 0.8415$. If we use a numerical integration scheme, we have to choose the step size T. Table 1.1 shows the error of the rectangular, trapezoidal, and fourth-order Runge–Kutta integration methods for this example for various values of T. As expected, Runge–Kutta is more accurate than trapezoidal, and trapezoidal is more accurate than rectangular.

Also as expected, the error for given method decreases as T decreases. However, perhaps the most noteworthy feature of Table 1.1 is *how* the integration error decreases with T. We can see that with rectangular integration, when T is halved, the integration error is also halved. With trapezoidal integration, when T is halved, the integration error decreases by a factor of four. With Runge–Kutta integration, when T is halved, the integration error decreases by a factor of 16. We conclude that (in general) the error of rectangular integration is proportional to T, the error of trapezoidal integration is proportional to T^2, and the error of Runge–Kutta integration is proportional to T^4.

Table 1.1 Example 1.5 results. Percent errors when numerically integrating $\dot{x} = \cos t$ from $t = 0$ to $t = 1$, for various integration algorithms, and for various time step sizes T.

	$T = 0.1$	$T = 0.05$	$T = 0.025$
Rectangular	2.6	1.3	0.68
Trapezoidal	0.083	0.021	0.0052
Fourth-order Runge–Kutta	3.5×10^{-6}	2.2×10^{-7}	1.4×10^{-8}

$\nabla\nabla\nabla$

1.6 STABILITY

In this section, we review the concept of stability for linear time-invariant systems. We first deal with continuous-time systems in Section 1.6.1, and then discrete-time systems in Section 1.6.2. We state the important results here without proof. The interested reader can refer to standard books on linear systems for more details and additional results [Kai80, Bay99, Che99].

1.6.1 Continuous-time systems

Consider the zero-input, linear, continuous-time system

$$\begin{aligned} \dot{x} &= Ax \\ y &= Cx \end{aligned} \tag{1.118}$$

The definitions of marginal stability and asymptotic stability are as follows.

Definition 1 *A linear continuous-time, time-invariant system is marginally stable if the state $x(t)$ is bounded for all t and for all bounded initial states $x(0)$.*

Marginal stability is also called Lyapunov stability.

Definition 2 *A linear continuous-time, time-invariant system is asymptotically stable if, for all bounded initial states $x(0)$,*

$$\lim_{t \to \infty} x(t) = 0 \tag{1.119}$$